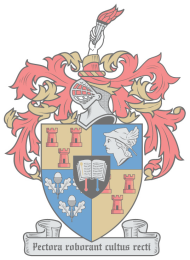
Ensemble methods in multi-label classification

Annegret Muller



UNIVERSITEIT
iYUNIVESITHI
STELLENBOSCH
UNIVERSITY

Thesis presented in partial fulfilment of the requirements for the degree of
MCom (Mathematical Statistics) in the Faculty of Economic and Management Sciences at
Stellenbosch University

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

December 2018

Supervisor: Professor SJ Steel

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Initials and surname	**Date**
A Muller	December 2018

Abstract

There are many scenarios where several labels may be associated simultaneously with each data case in a dataset. Therefore, a large number of multi-label datasets are found in a variety of domains, including image annotation, text annotation, bioacoustics, music research and medical diagnostics. In this thesis, we focus on such multi-label datasets with the particular goal of performing multi-label classification. Multi-label classification is an extension of binary- and multi-class classification to scenarios where each instance in a dataset can have multiple or none of K labels.

Methods to perform multi-label classification can be divided into three categories: problem transformation methods, algorithm adaptation methods and multi-label ensemble methods. Multi-label ensemble methods receive particular attention in this research. We discuss previously proposed multi-label ensemble methods and also propose a new multi-label ensemble method, named label dependent splitting (LDsplit) with trees.

LDsplit with trees constructs an ensemble of tree-structures by considering different permutations of the labels in a multi-label dataset. The method differs from other multi-label ensemble methods since each tree-structure splits the multi-label data in a label-dependent way, whilst incorporating label correlation. Furthermore, each split of a node in a tree-structure is performed by considering a binary classification problem. By performing an empirical study on benchmark datasets, the predictive performance of LDsplit with trees is compared to that of other multi-label learning methods. LDsplit with trees produces very promising results, allowing us to believe that with further modifications the procedure may become a highly competitive multi-label learning method.

Furthermore, we also explore aspects of analysing text data. We perform an extensive analysis on a practical multi-label text dataset. The practical dataset consists of online comments, where each comment is labeled to identify if any or multiple so-called “toxicity” are present in the comment. Our model may therefore be used to identify different types of toxicity present in online comments and help reduce online abuse and harassment. A particular challenge faced in the practical data analysis is the sparsity of the labels.

Keywords:

Classification, multi-label, ensemble methods, text data

Opsomming

Baie scenarios bestaan waar verskeie etikette gelyktydig met elke datageval in ‘n datastel geassosieer word. Daarom vind ons ‘n groot hoeveelheid multi-etiket datastelle in ‘n verskeidenheid van domeine, insluitend beeld-annotasie, teks-annotasie, bio-akoestieks, musieknavorsing en mediese diagnose. In hierdie tesis fokus ons op sulke multi-etiket datastelle met die spesifieke doel om multi-etiket klassifikasie uit te voer. Multi-etiket klassifikasie is ‘n uitbreiding van binêre- en multiklas klassifikasie na scenarios waar elke geval in ‘n datastel verskeie of geen van K etikette het.

Metodes wat gebruik word om multi-etiket klassifikasie uit te voer, kan in drie kategorieë verdeel word: probleem transformasie metodes, algoritme aanpassings metodes en multi-etiket ensemble metodes. Multi-etiket ensemble metodes kry spesifieke aandag in hierdie navorsing. Ons bespreek voorheen voorgestelde multi-etiket ensemble metodes en stel ook ‘n nuwe multi-etiket ensemble metode voor, genaamd etiket afhanklike splitting (LDsplit) met bome.

LDsplit met bome bou ‘n ensemble van boom-strukture deur verskillende permutasies van die etikette in die multi-etiket datastel te beskou. Die metode verskil van ander multi-etiket ensemble metodes aangesien elke boom-struktuur die multi-etiket data op ‘n etiket-afhanklike basis opdeel, terwyl etiket-korrelasie terselfdetyd geïnkorporeer word. Daarbenewens word elke splitsing van ‘n node in ‘n boom-struktuur uitgevoer deur ‘n binêre klassifikasie probleem te beskou. Met behulp van ‘n empiriese studie op standaard datastelle word die voorspellings-prestasie van LDsplit met bome vergelyk met dié van ander multi-etiket leermetodes. LDsplit met bome lewer baie belowende resultate, wat ons dus toelaat om te glo dat, met verdere aanpassings, mag die prosedure ‘n hoogs kompeterende multi-etiket leermetode word.

Verder verken ons ook aspekte van teksdata. Ons voer ‘n uitgebreide analise op ‘n praktiese multi-etiket teksdatastel uit. Die praktiese-datastel bestaan uit aanlyn kommentaar, waar elke individu se kommentaar gemerk word om vas te stel of enige of verskeie tipes toksisiteit teenwoordig is. Ons model kan dus gebruik word om verskillende tipes toksisiteit in aanlyn kommentaar te identifiseer en help om aanlyn misbruik en teistering te verminder. ‘n Besonderse uitdaging in die praktiese-data analise is die ylheid van die etikette.

Sleutelwoorde:

Klassifikasie, multi-etiket, ensemble metodes, teksdata

Acknowledgements

I hereby wish to acknowledge the Department of Statistics and Actuarial Science of Stellenbosch University for providing me with the necessary support to complete this thesis. I am also thankful for the South African Statistical Association National Research Foundation (SASA-NRF) grant. Finally, I wish to thank Professor S.J. Steel for supervising my research.

Table of contents

List of figures

List of tables

List of appendices

List of abbreviations and/or acronyms

DTM	Document-term-matrix
FN	False negative
FP	False positive
GUI	Graphical user interface
HOMER	Hierarchy of multilabel classifiers
LDsplit	Label dependent splitting
MODT	Multi-objective decision tree
NLP	Natural language processing
PCorpus	Permanent corpus
PCT	Predictive clustering tree
RAkEL	Random $k-$labelsets
ROC	Receiver operating characteristic
SMOTE	Synthetic minority over-sampling technique
TDM	Term-document-matrix
TN	True negative
TP	True positive
VCorpus	Volatile corpus

CHAPTER 1: INTRODUCTION

1.1 CLASSIFICATION

This thesis falls within the scope of supervised statistical learning, particularly, multi-label classification. In supervised statistical learning, a set of input variables, also known as predictors, $X_1, X_2, ..., X_p$, are present along with output variables $Y_1, Y_2, ..., Y_K$. The output variables are also referred to as response variables. In supervised learning the goal is to fit a function, $f(X_1, X_2, ..., X_p) = f(\mathbf{X})$, relating the input variables to the response variables. The training data can be denoted by: $\{(\mathbf{x}_i, \mathbf{y}_i), i = 1, 2, ..., N\}$. Here $\mathbf{x}_i$, $i = 1, 2, ..., N$, denote N p- component observations of input variables $X_1, X_2, ..., X_p$. The K-component response vectors corresponding to the N observations are denoted by $\mathbf{y}_i$, $i = 1, 2, ..., N$. The nature of the response variables, $Y_1, Y_2, ..., Y_K$, leads to the distinction between regression problems and classification problems. If the response variables are numeric, *i.e.* $Y_1, Y_2, ..., Y_K$ are quantitative, we refer to the prediction process of the response as a regression problem. If the response variables are specified in terms of classes, *i.e.* $Y_1, Y_2, ..., Y_K$ are qualitative, we refer to the prediction of the class(es) to which an input vector belongs as a classification problem.

A common type of classification problem is binary classification. In the case of binary classification, $K = 1$ and $Y \in \{0, 1\}$. In other words, there exists two disjoint classes, one class denoted by $Y = 1$ and the other class denoted by $Y = 0$. If observation i is included in the first class this is denoted by letting $y_i = 1$ and if observation i is included in the second class this is denoted by letting $y_i = 0$. A different way of viewing this is to let K denote the number of so called "labels". Then, in the case of binary classification, where $K = 1$, we have one label for the data. Then each observation either has this label present (denoted by $y_i = 1$) or does not have this label present (denoted by $y_i = 0$). Many classification problems resemble this form. For example, a dataset may consist of credit card transactions where each transaction is classified to be fraudulent or not. In this case, fraudulent transactions can be denoted by letting $y_i = 1$, and nonfraudulent transactions can be denoted by letting $y_i = 0$. Viewed differently, there is one label in this dataset, the "fraudulent"-label. Each observation either has this label present, *i.e.* the

transaction is fraudulent, denoted by $y_i = 1$, or does not have this label present, *i.e.* the transaction is not fraudulent, denoted by $y_i = 0$.

In multi-class classification we have $K = 1$ and $Y \in \{1, 2, ..., G\}$. Here, each observation belongs to one of G classes and the G classes are disjoint. If the i th observation belongs to the g th class, this can be denoted by letting $y_i = g$ where $i = 1, ..., N$ and $g = 1, ..., G$. A multi-class classification problem can for example be the classification of handwritten digits. In this scenario, digits zero to nine denote the $G = 10$ classes and each handwritten observation belongs to one of the ten classes.

In multi-label classification we have $K > 1$ and $Y_k \in \{1, 0\}$ where $k = 1, ..., K$. Here each observation can have multiple or none of K labels. The labels are not disjoint. The k th entry of $\mathbf{y}_i$ equals 1 if the i th observation has the k th label present, where $k = 1, ..., K$ and $i = 1, ..., N$. Multiple entries of $\mathbf{y}_i$ can therefore equal 1 if observation i has multiple of the K labels present. It may also be that observation i has none of the labels present, in which case all the entries of $\mathbf{y}_i$ equal 0.

Finally, in multi-class multi-label classification, $K > 1$ and $Y_k \in \{1, ..., G\}$ where $k = 1, ..., K$. Now the K labels each consists of G disjoint label classes. This is arguably the most complex type of classification and is also sometimes referred to as multidimensional classification. In this thesis we focus on the slightly simpler classification type, namely multi-label classification in which case we have $K > 1$ and $Y_k \in \{1, 0\}$ for $k = 1, ..., K$.

1.2 EXAMPLES OF MULTI-LABEL DATASETS

In the modern world where vast amounts of digital data have become the norm, many multi-label datasets can be found in a variety of fields. Some particular examples of multi-label dataset domains include image annotation, text annotation, bioacoustics, music and medical diagnostics. In this section we list a few of the large number of multi-label datasets found in a variety of domains. It is clear that multi-label data analysis has become an active area of research.

1.2.1 Image annotation

In image annotation, the data scientist can for example aim to supply tags to images on the internet. In this case, the K labels of the multi-label dataset correspond to K possible image-tags. Tags can for example include: sunset, mountain, beach, field, lighthouse, tower, etc. The training data of such an analysis consists of many internet images along with the appropriate tags that are present for each image. It may be useful for image search engines like Google Images or Pinterest to use the results of such a multi-label image analysis in order to improve search results for their users. Many multi-label classification image datasets can be found in practice. Another example is found on http://www.kaggle.com, an online data-sharing platform, hereafter referred to as Kaggle. Here, users are provided with a dataset containing a collection of movie posters along with the genres each movie falls into. The aim of the analysis is to build a classifier that can identify all the appropriate genres a movie falls into by only considering the movie poster. Each movie falls into one or multiple genres. Genres include: horror, romance, animation, comedy, etc. Here the data scientist might take note of the colours used on the posters, the brightness of the colours or the expressions of the actors on the posters, in order to find hints of the specific genres the movies fall into (Kaggle: Movie genre from its poster, 2018).

Multi-label image datasets are not only limited to still images. Observations in the form of video clips are also common. For example, a dataset found on Kaggle contains over 7 million YouTube videos along with 4716 video labels. A very large dataset indeed. The aim is to build a powerful classifier that can provide relevant video labels to new videos and by doing so improve the search and organization of video archives (Kaggle: Google Cloud & YouTube-8M video understanding challenge, 2017).

1.2.2 Text annotation

In text annotation it might be that the aim is to attach categories to text documents. Then the K labels of the multi-label dataset correspond to K possible text categories. Categories may include: biography, review, sport, food, blog or even “fake news”. A multi-label text dataset is also found on Kaggle: Greek media monitoring multi-label classification. The data consists of a number of Greek print media from May 2013 to September 2013, categorized to one or more topics by human annotators. Topics include specific persons, products or companies as well as more general topics such as economy and environment. The goal is to build a classifier that can identify the appropriate topics present for a new article. Some of the topics, such as specific persons, products or companies, might be easier to classify since they are based on keywords. Other

topics, such as environment or economy, might be more difficult to detect in an article since they are more general concepts (Kaggle: Greek media monitoring multi-label classification, 2014).

1.2.3 Bioacoustics

In the bioacoustics domain, the data may for example consist of sound fragments and the aim may be to identify all the different bird species in each sound fragment. In this case, the K labels of the multi-label dataset correspond to K different bird species and the observations of this multi-label classification dataset are in the form of audio sound fragments.

1.2.4 Music

Another example of multi-label classification, where the observations may be in the form of audio fragments, is instrument recognition. Different types of musical instruments playing simultaneously can for example be identified in music pieces. The training data of such an analysis consist of many musical pieces along with the corresponding instruments present in each musical piece. Therefore the K labels in this multi-label dataset correspond to K musical instruments.

1.2.5 Medical diagnostics

In medical diagnostics, multiple medical conditions can be identified for a patient. For example, a dataset available on Kaggle contains 5606 chest X-ray images of patients and a total of 14 disease-labels. Diseases found in each of the X-ray images are labelled as such. Some X-ray images have no disease present, whereas others have one or multiple of the 14 disease-labels present. Additional information, such as the patient's age and gender, are also available for each of the 5606 observations. Building a powerful classifier for such a multi-label dataset can improve diagnoses of these diseases for future patients (Kaggle: Random sample of NIH chest X-ray dataset, 2017).

1.3 NOTATION

In this thesis the multi-label classification data, $\{(\mathbf{x}_i, \mathbf{y}_i),\ i = 1, 2, ..., N\}$, are summarized as a $N \times (p + K)$ matrix $\left[\underset{N\times p}{\mathbf{X}} \quad \underset{N\times K}{\mathbf{Y}}\right]$. A matrix like this can resemble Figure 1.1.

	Input variables				Label indicator variables			
Data observations	X_1	X_2	...	X_p	Y_1	Y_2	...	Y_K
1	$x_{1,1}$	$x_{1,2}$	...	$x_{1,p}$	1	0	...	1
2	$x_{2,1}$	$x_{2,2}$	...	$x_{2,p}$	1	1	...	1
3	$x_{3,1}$	$x_{3,2}$	...	$x_{3,p}$	0	1	...	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
N	$x_{N,1}$	$x_{N,2}$	...	$x_{N,p}$	0	1	...	0

Figure 1.1: Matrix representation of multi-label classification dataset

Figure 1.1 consists of N p- component observations, $\mathbf{x}_i, i=1,...,N,$ of the input variables $X_1, X_2, ..., X_p$. There is a total of K labels denoted by $Y_1, Y_2, ..., Y_K$. Therefore, each observation has a K- component label indicator resulting in a total of N K-component label indicators, $\mathbf{y}_i, i=1,...,N,$ for the dataset. The first observation, for example, has both labels Y_1 and Y_K present, the second observation has labels Y_1, Y_2 and Y_K present and so on for all N observations. We denote the collection of labels, $Y_1, Y_2, ..., Y_K,$ by L.

In some cases, the labels of a training dataset may be in the form of a hierarchical structure. This means that the labels are organized into general-to-specific levels. The top level of the hierarchy consists of a few general classes. Each of these classes in turn subdivide into slightly more specific classes, and these classes are in turn subdivided, and so on. The classification process of new observations in such a hierarchical structure is referred to as hierarchical classification. If observations are labelled with more than one node in the hierarchical structure, the classification process of a new observation is referred to as hierarchical multi-label classification (Tsoumakas and Katakis, 2007:3). In this thesis, the focus will however be on non-hierarchical multi-label classification. Therefore, when multi-label classification is mentioned, we are in fact referring to non-hierarchical multi-label classification.

Our goal in the multi-label analysis is multi-label classification. We thus aim to fit a function relating the input variables to the labels, with the goal of using this fitted function to classify an unseen observation to relevant labels as accurately as possible. Inference may also be a goal of a multi-label analysis. Understanding the relationship between different labels and different variables are

often of interest. In medical diagnostics, for example, it can be very advantageous if an analysis reveals which predictors are correlated with which medical conditions. Another multi-label analysis goal worth mentioning is multi-label ranking. In the case of multi-label ranking, the aim is to fit a function relating the input variables to the labels, with the goal of using this fitted function to provide an unseen observation with a preference list, *i.e.* a ranking of the labels from the set of possible labels (Madjarov *et al*., 2012). Multi-label ranking is also an important topic in multi-label data analysis. An image search engine, for example, does not only provide users with relevant images when a search is carried out, but also provides users with a ranking of the images. Here the intention is that the first image presented to the user corresponds better to the user's preference than those images that appear after it.

1.4 OVERVIEW

This thesis consists of six chapters. The first is an introductory chapter where we distinguish multi-label classification from other classification types. We also refer to some examples of multi-label dataset domains, state our multi-label analysis goal and introduce the notation used for multi-label data in this thesis.

After multi-label data are introduced in the introductory chapter, Chapter 2 refers to unique characteristics of multi-label data compared to other data structures. Performance measures for multi-label data are also given.

Multi-label learning methods are discussed in Chapter 3. We outline the three categories of multi-label learning and state that multi-label ensemble methods receive particular attention in this thesis. Three multi-label ensemble methods are discussed in this chapter.

In Chapter 4 we explain a new multi-label ensemble method. The predictive performance of this method is compared to that of previously proposed multi-label learning methods by performing a benchmark dataset analysis. We also highlight the key differences between the new approach and previously proposed multi-label ensemble methods discussed in this thesis.

In Chapter 5 our focus shifts to multi-label text data. We briefly discuss aspects of analysing text data and refer to useful R packages for text and multi-label analyses. The chapter concludes with a practical multi-label text data analysis. We describe the cleaning and pre-processing of the data, the feature selection process and multi-label characteristics of the data. Lastly, we present and discuss the results obtained when performing multi-label classification.

Finally, the last chapter provides concluding remarks as well as opportunities for future research.

CHAPTER 2: MULTI-LABEL DATA

2.1 ASPECTS OF MULTI-LABEL DATA

Multi-label datasets have some unique characteristics compared to other data structures. Some of these aspects might influence the performance of different multi-label learning methods and are thus important to take note of. Other aspects might also make the analysis of a multi-label dataset more difficult. In this section we therefore briefly discuss some of these unique aspects, including: the value of K, label imbalance, labelsets, label correlation, dimension reduction, subsampling, benchmark datasets and synthetic generation of multi-label data.

2.1.1 Number of labels, K

Some multi-label datasets have a very large number of labels, K. The high dimensionality of datasets of this form can significantly influence the performance of multi-label methods. Some multi-label methods have a significant increase in computational cost if the number of labels is very large and for this reason are simply an inappropriate choice for datasets with large K. Examples of domains with large numbers of labels include text categorization, protein function classification and semantic annotation of multimedia (Tsoumakas *et al*., 2008). The hierarchy of multilabel classifiers (HOMER) algorithm, described in Tsoumakas *et al*. (2008), is one example of a procedure developed to deal with this aspect of multi-label data.

2.1.2 Label imbalance

Another characteristic is the number of observations annotated with each label in the dataset. Sometimes the labels in a multi-label dataset can be imbalanced. Some labels may occur multiple times, *i.e.* the labels are present for many observations, whereas other labels may occur rarely, *i.e.* the labels are present for very few of the observations. Learning methods trained on such imbalanced labels might struggle to identify new observations which have the less dominant labels present, since observations of this kind are so sparse in the training data. On the other hand, if K is very large, each observation might only have a few of the large number of labels present. In other words, all the labels are sparse. The average number of labels of the observations in the multi-label dataset is defined as label cardinality and is given by $cardinality = \frac{1}{N}\sum_{i=1}^{N}\left|\mathbf{y}_i\right|$. Here, $\left|\mathbf{y}_i\right|$ denotes the number of entries in $\mathbf{y}_i$ that are equal to 1, *i.e.* the total number of labels observation i has present. The label density of a multi-label dataset is the

average number of labels of the observations divided by K, thus $density = \frac{1}{N}\sum_{i=1}^{N}\frac{|\mathbf{y}_i|}{K}$. Label density therefore also takes the number of labels in the dataset into consideration. Two datasets may have the same cardinality but can differ significantly with regard to density. This can cause the two datasets to behave differently when the same multi-label learning method is applied (Tsoumakas and Katakis, 2007:8). The interested reader may also consult Charte and Charte (2018) for a definition of the so-called "imbalance ratio", IRLbl.

2.1.3 Labelsets

The number of distinct label combinations per observation found in the dataset, *i.e.* the number of labelsets, may also influence the performance of multi-label methods. Some multi-label learning methods make specific use of the different labelsets in the learning process so that a large number of labelsets can significantly increase computational cost.

2.1.4 Label correlation

In multi-label analyses we might also consider label correlation. For example, we might ask which labels in the multi-label dataset are interdependent? Aspects of label correlation are useful for both inference and classification of unseen observations. Some labels in a multi-label dataset may have strong interdependencies. For a dataset like this, learning methods that try to exploit label dependencies may be a better choice than learning methods that do not try to exploit label dependencies. However, learning methods that incorporate label dependencies may be more computationally intensive, especially for large datasets. A simpler method might be faster, but has the disadvantage of possibly ignoring important information contained in the dataset. We discuss both types of learning methods in later chapters.

2.1.5 Dimension reduction

Some multi-label datasets have a large number of predictors. A wide variety of studies are dedicated to the topic of dimension reduction for single-label classification, where $K = 1$. In the case of multi-label classification, where $K > 1$, various ad hoc approaches for variable selection have been proposed. Approaches for multi-label dimension reduction still pose an avenue for further research. The challenge is that X_3 can, for example, assist in predicting Y_1, but also make predicting Y_2 more difficult. In other words, what determines the importance of a predictor in the multi-label classification scenario? A predictor has global importance when the predictor is globally correlated with all the labels, but some predictors are locally important in which case the

predictor is correlated only with a subset of the labels. Some approaches to multi-label variable selection include the filter approach, wrapper approach and the embedded approach, as outlined in Spolaôr *et al*. (2013).

2.1.6 Subsampling

When multi-label data are subsampled, as in the case of $k-$fold cross-validation for example, the division is slightly more difficult compared to single-label data. In the multi-label case, we would like the proportion of observations per label in each subset to be approximately equal to that of the complete dataset. Randomly splitting the multi-label data into k folds will probably not allow this to be true. This is not ideal. For one or more of the folds it can for example happen that none of the observations are annotated with a rare label, or some labels may be overrepresented in some folds and completely underrepresented in others. Ideally the k folds should act as smaller versions of the complete dataset, having similar characteristics. Sechidis *et al*. (2011) outline stratification in the context of multi-label data. One approach focusses on distinct labelsets and the other considers each label independently of the rest.

2.1.7 Benchmark datasets

When research is carried out in the field of multi-label classification, researchers often make use of common multi-label benchmark datasets. The benchmark datasets are openly available to researchers. Analyses from different research papers can therefore be compared when the same benchmark datasets are used. The datasets belong to different domains, including music, images, text, etc. Table 2.1 summarizes some benchmark datasets used in multi-label research.

Table 2.1: Benchmark datasets

Dataset	Domain	N	p	K	Label cardinality	Label density
Emotions	Music	593	72	6	1.87	0.311
Flags	Image	194	19	7	3.39	0.485
Scene	Image	2407	294	6	1.07	0.179
Tmc2007	Text	28596	49060	22	2.16	0.098
Yeast	Biology	2417	103	14	4.24	0.303

Source: Charte and Charte, 2017.

The *Emotions* dataset consists of 593 music-piece observations. Each music piece is labelled with a selection of six possible emotions namely: sad-lonely, angry-aggressive, amazed-surprised, relaxing-calm, quiet-still and happy-pleased. The *Flags* dataset consists of 194 flags and 19 flag features. Each flag is annotated with up to seven colours. The *Scene* dataset contains 2407 images annotated with up to six concepts such as beach, mountain and field. The *Tmc2007* dataset gives aviation safety reports that document problems that occurred during certain flights as observations. The labels of the *Tmc2007* dataset are the problems being described by the reports. The *Yeast* dataset consists of 2417 genes and each gene can be associated with 14 biological functions, so that the biological functions form the 14 labels of the *Yeast* dataset.

2.1.8 Synthetic data generation

When new multi-label learning methods are introduced, the benchmark datasets can be useful to investigate the performance of the newly introduced method against other learning methods. The benchmark datasets are also useful when properties of multi-label datasets are investigated. These datasets are useful, but they are not entirely sufficient. The "truth" of the benchmark datasets are unknown to researchers. For example, it is unknown which labels are correlated, which predictors are correlated or what the relationship between the labels and the predictors in the benchmark datasets are. Researchers generate synthetic data in order to have control and precise knowledge of such properties in investigations. In single-label classification, it is simple to systematically generate synthetic single-label classification data. It is however much more difficult to simulate synthetic multi-label data. In the case of multi-label analysis, we would like to incorporate multi-label characteristics, like those mentioned in this section, into the data. For example, the generated data should consist of K correlated binary labels as well as p input vectors of a specified multi-variate distribution. We would like the input vectors to depend on the labels in some way. We would also for example like some predictors to be good at predicting a certain label while simultaneously being bad at predicting another label. These issues make systematic generation of multi-label data more difficult. Few proposals exist for systematically generating synthetic multi-label data and this may still be a field that requires further research. Tomás *et al*. (2014) has contributed by proposing Mldatagen, a multi-label dataset generator framework. Other contributors include Sandrock and Steel (2017) proposing an algorithm that offers users the option to specify many aspects regarding the data to be generated.

2.2 PERFORMANCE MEASURES

Some of the above-mentioned aspects may influence the performance of different multi-label methods, but how can the performance of different multi-label methods be measured? The multi-label data structure requires different performance measures from those used in traditional single-label classification. However, a large number of evaluation measures have been proposed for multi-label data. Measures exist for the evaluation of both multi-label classification as well as multi-label ranking. The following discussion is based on Tsoumakas *et al.* (2010).

For multi-label classification, evaluation measures can be grouped into two main categories: example-based and label-based measures. As the name suggests, example-based measures consider each observation individually and computes the metric for each. The final performance value is obtained by averaging the performance values for the individual observations. Label-based metrics on the other hand consider each label individually and computes the metric for each label, instead of each observation. Figure 2.1 outlines the categorization of performance measures for multi-label learning.

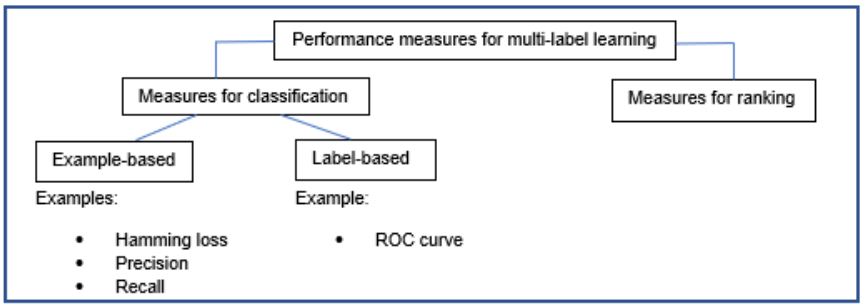


Figure 2.1: Categorization of performance measures for multi-label learning

Consider a test dataset, $\{(\mathbf{x}_i, \mathbf{y}_i), i = 1, 2, ..., M\}$, and suppose our multi-label classifier gives classifications $f(\mathbf{x}_i) = \mathbf{z}_i$, $i = 1, ..., M$, where $\mathbf{z}_i \in \{0,1\}^K$. Some example-based measures include the following. For a single test case, Hamming loss is the proportion of incorrectly predicted labels. Therefore, for the test data, Hamming loss is defined as:

$$\frac{1}{KM} \sum_{m=1}^{M} \sum_{k=1}^{K} \left| y_{mk} - z_{mk} \right|.$$

Precision on the other hand is given by:

$$precision = \frac{1}{M}\sum_{m=1}^{M}\left\{\frac{\sum_{k=1}^{K} y_{mk}\, z_{mk}}{\sum_{k=1}^{K} z_{mk}}\right\}.$$

Due to the way precision is defined, it produces an average proportion of those labels predicted to be present $(Z=1)$ and which are correctly predicted, *i.e.* the label is in fact present $(Y=1)$, provided the denominator is not zero.

Recall is defined as:

$$recall = \frac{1}{M}\sum_{m=1}^{M}\left\{\frac{\sum_{k=1}^{K} y_{mk}\, z_{mk}}{\sum_{k=1}^{K} y_{mk}}\right\}.$$

Recall is the average proportion of true labels, *i.e.* labels that are present $(Y=1)$, which are predicted to be present $(Z=1)$, provided the denominator is not zero.

Precision and recall can also be combined into a single measure which is referred to as the F-score. The F-score is defined as:

$$F = \frac{2 \times precision \times recall}{precision + recall}.$$

Precision and recall are related to the concepts of sensitivity and specificity. Consider the confusion matrix given in Figure 2.2. Two types of incorrect classifications can be made by a multi-label classifier. If a label is present, *i.e.* $Y=1$, the classifier can predict the label not to be present, *i.e.* $Z=0$. This type of error is referred to as a false negative (FN) classification. On the other hand, if a label is not present, *i.e.* $Y=0$, the classifier can predict the label to be present, *i.e.* $Z=1$. This type of error is referred to as a false positive (FP) classification. When a label is present, *i.e.* $Y=1$, and it is predicted to be present, *i.e.* $Z=1$, we refer to the classification as a true positive (TP) classification. When a label is not present, *i.e.* $Y=0$, and it is predicted not to be present, *i.e.* $Z=0$, we refer to the classification as a true negative (TN) classification.

Considering the total number of all these different classifications for a dataset, sensitivity is defined as $\frac{TP}{TP+FN}$ and specificity is defined as $\frac{TN}{FP+TN}$. Precision is also given by $\frac{TP}{TP+FP}$ and recall is defined in the same way as sensitivity, *i.e.* $\frac{TP}{TP+FN}$. A trade-off exists between precision and recall, with equality when $FP = FN$.

		True label	
Predicted label		$Y=1$	$Y=0$
	$Z=1$	True Positive (TP)	False Positive (FP)
	$Z=0$	False Negative (FN)	True Negative (TN)

Figure 2.2: Confusion Matrix

A label-based measure, on the other hand, can be any known evaluation measure for binary classification. The TP and FP rates for a single label can for example be used to construct a curve referred to as the receiver operating characteristic (ROC) curve. With TP rate given on the y-axis and FP rate given on the x-axis, the curve is traced out by varying the threshold value of the classifier. The overall performance of the classifier is given by the area under the ROC curve. The larger the area, the better the performance of the classifier, so that an ideal ROC curve hugs the top left corner of the graph. R packages that allow users easy comparison of the areas under the ROC curves and plotting of these curves include ROCR and pROC (Robin *et al.*, 2011).

Denote by TP_k, TN_k, FP_k and FN_k the number of true positive, true negative, false positive and false negative classifications after binary evaluation of label k. Then, if a binary evaluation measure, calculated based on the number of TP, TN, FP and FN classifications, is denoted by $B(TP, TN, FP, FN)$, the so-called macro-averaged and micro-averaged versions of B are given by:

$$B_{macro} = \frac{1}{K}\sum_{k=1}^{K} B(TP_k, TN_k, FP_k, FN_k)$$

and

$$B_{micro} = B\left(\sum_{k=1}^{K} TP_k, \sum_{k=1}^{K} TN_k, \sum_{k=1}^{K} FP_k, \sum_{k=1}^{K} FN_k\right).$$

A multi-label learning method usually performs differently for different performance measures (Wu and Zhou, 2016). Therefore, multiple multi-label performance measures are used in a multi-label analysis.

This chapter highlights different challenges faced when analysing multi-label datasets. It is clear that single-label datasets and multi-label datasets cannot be approached in the same way. The multi-label data structure has some unique characteristics and requires specific tools in the analysis of the dataset, including the evaluation measures used when analysing predictive performance. In the next chapter multi-label learning methods are discussed. These learning methods are also particularly developed to handle the multi-label data structure.

CHAPTER 3: MULTI-LABEL LEARNING METHODS

3.1 CATEGORIES

In this chapter specific learning methods for multi-label classification are discussed. Several multi-label learning methods exist. Initially, Tsoumakas and Katakis (2007) grouped these learning methods into two main categories: algorithm adaptation methods and problem transformation methods. In more recent years, multi-label learning methods have also been extended to a third category, namely multi-label ensemble methods.

Algorithm adaptation methods start with a previously defined single-label learning algorithm and aim to transform the algorithm to handle the multi-label data directly. Some examples of such algorithms that have been extended to the multi-label structure are decision trees, boosting, neural networks, support vector machines and k-nearest neighbours. The interested reader may consult Madjarov *et al*. (2012) for detailed descriptions of the adaptations.

Problem transformation methods are methods that use a single-label classification method (*i.e.* multi-class or binary) for the multi-label data analysis. Usually the multi-label data are transformed in a specified way so that a single-label classification method can be applied. The multi-label learning problem is therefore split into one or more single-label classification problems. One single-label classification method is normally applied to the transformed data and the output is combined in a particular way so that the results have a multi-label structure. Many problem transformation methods exist. We highlight three problem transformation methods, namely binary relevance, label powerset and classifier chains. These three methods are sometimes used as base classifiers in other multi-label learning methods, we therefore highlight them in particular.

In this thesis, the main focus will be on the third group of learning methods for multi-label data, namely ensemble methods. Ensemble schemes use several function estimates or predictions and combine them in a specified way in order to build one powerful, in our case, classifier. The base classifiers of these ensemble methods in our context belong to either of the two previously mentioned categories of multi-label methods, *i.e.* algorithm adaptation and problem transformation methods. Examples of ensemble methods for multi-label classification are ensembles of classifier chains in Read *et al*. (2011), random forests of predictive clustering trees in Kocev *et al*. (2007) and random $k-$labelsets (Tsoumakas *et al*., 2011).

3.2 BINARY RELEVANCE

One of the simplest problem transformation methods is known as binary relevance. The following discussion is based on Tsoumakas and Katakis (2007). For illustration, consider the multi-label dataset given in Table 3.1 that contains $N=5$ observations, $\mathbf{x}_i$, $i=1,...,5$, and $K=4$ labels, denoted by Y_k, $k=1,...,4$.

Table 3.1: Example of multi-label dataset

Input variables	Label indicator variables			
	Y_1	Y_2	Y_3	Y_4
$\mathbf{x}_1$	1	0	0	1
$\mathbf{x}_2$	0	0	1	1
$\mathbf{x}_3$	1	0	0	0
$\mathbf{x}_4$	0	1	1	0
$\mathbf{x}_5$	1	1	1	0

A problem transformation method transforms the multi-label data, such as that given in Table 3.1, so that a single-label classification method can be applied. In the case of binary relevance, the multi-label data are transformed by considering the data as K separate binary classification problems, one for each label. First an appropriate base classifier is chosen that can handle standard binary classification problems. Examples of such classifiers include linear discriminant analysis, logistic regression and support vector machines. We then fit K binary classification models, one for each label, using the base classifier. This results in f_k, $k=1,...,K$, where $f_k:\mathbf{x}\to Y_k(\mathbf{x})\in\{0;1\}$, $k=1,...,K$. For the data in Table 3.1, this would mean that the multi-label data are divided into $K=4$ separate binary classification problems, similar to Table 3.2. The base classifier is applied to each of the $K=4$ binary classification versions of the data and f_k, $k=1,...,4$, are found.

Table 3.2: Splitting multi-label data into binary classification data

	Y_1
$\mathbf{x}_1$	1
$\mathbf{x}_2$	0
$\mathbf{x}_3$	1
$\mathbf{x}_4$	0
$\mathbf{x}_5$	1

	Y_2
$\mathbf{x}_1$	0
$\mathbf{x}_2$	0
$\mathbf{x}_3$	0
$\mathbf{x}_4$	1
$\mathbf{x}_5$	1

	Y_3
$\mathbf{x}_1$	0
$\mathbf{x}_2$	1
$\mathbf{x}_3$	0
$\mathbf{x}_4$	1
$\mathbf{x}_5$	1

	Y_4
$\mathbf{x}_1$	1
$\mathbf{x}_2$	1
$\mathbf{x}_3$	0
$\mathbf{x}_4$	0
$\mathbf{x}_5$	0

Suppose we aim to classify an unseen observation, $\mathbf{x}$. In binary relevance, the K functions, f_k , $k = 1,...,K,$ are applied to the unseen observation, resulting in a separate classification for each of the K labels. For the data in Table 3.1, this would mean that the unseen observation, $\mathbf{x}$, is classified by $\left[f_1(\mathbf{x}) \quad f_2(\mathbf{x}) \quad f_3(\mathbf{x}) \quad f_4(\mathbf{x}) \right]$.

The main advantages of binary relevance are that it is a simple and fast procedure. Unfortunately, binary relevance does not take label correlation into consideration. Each label prediction is made separately from the other labels and no label interactions are used in the binary relevance procedure.

3.3 LABEL POWERSET

This section is based on Tsoumakas *et al*. (2011). The label powerset method is a problem transformation method that attempts to take label correlation into consideration. The label powerset approach recognizes each unique set of labels that exists in the multi-label dataset as a class. The multi-label data are therefore transformed to a multi-class structure. Applying this idea to the data in Table 3.1 causes the data to resemble Table 3.3. A standard multi-class classifier can then be applied to the multi-class version of the data. This multi-class classifier can then be used to classify an unseen observation by classifying the observation to the class with the highest posterior probability.

Table 3.3: Transformation of data in Table 3.1 to multi-class structure

	Y_1	Y_1 and Y_4	Y_1, Y_2 and Y_3	Y_2 and Y_3	Y_3 and Y_4
$\mathbf{x}_1$	0	1	0	0	0
$\mathbf{x}_2$	0	0	0	0	1
$\mathbf{x}_3$	1	0	0	0	0
$\mathbf{x}_4$	0	0	0	1	0
$\mathbf{x}_5$	0	0	1	0	0

Unfortunately, the label powerset method can only classify an unseen observation to a labelset present in the training data. This limits the procedure since no new label combination can be formed. New label combinations, not found in the training dataset, can easily occur in test datasets. This property of the label powerset method may cause an increase in its test error.

The label powerset procedure also has the disadvantage that the transformation of the multi-label data to a multi-class structure may lead to a dataset with a large number of classes and few observations per class. This is especially true for multi-label datasets that have a large number of labels. The total number of possible labelsets for a multi-label dataset, and therefore also the total number of possible classes for the multi-class version of the data, is upper bounded by $\min\left(N, 2^K\right)$. In practice, the number is normally much smaller than the upper-bound, however it can still be large enough to make the learning process difficult. Depending on the multi-class classification algorithm used in the label powerset procedure, a large number of labelsets may significantly increase computational cost. Furthermore, with few observations per class in the multi-class version of the data, the learning process becomes more difficult so that the fitted model may not perform so well on test data.

3.4 CLASSIFIER CHAINS

The classifier chains approach is a problem transformation method based on the binary relevance method. The method aims to improve on some of the shortcomings of the binary relevance method, while retaining the advantages of being simple and fast. The following discussion is based on Read *et al*. (2011).

The main disadvantage of binary relevance that the classifier chains method aims to improve, is the fact that binary relevance does not take label correlation into consideration. In binary relevance, K binary classification models, f_k , $k=1,...,K,$ are fit separately, one for each label. Classifier chains also applies K binary transformations that result in K classifiers, f_k , $k=1,...,K.$ However, when fitting classifier chains, the input space of each binary model is extended with the $0/1$ label relevance of all previous classifiers. The procedure is therefore appropriately named, since a chain of classifiers is indeed formed.

Suppose a multi-label classification dataset, $\{(\mathbf{x}_i, \mathbf{y}_i), i=1,2,...,N\},$ is given consisting of N p-component observations of input variables $X_1,...,X_p$ and a total of K labels, $Y_1, Y_2,...,Y_K$. After an appropriate base classifier that can handle standard binary classification problems is chosen, the classifier chains method is implemented as follows.

The first binary classifier, f_1, is found by fitting the base classifier to the binary classification data formed by considering $\mathbf{x}_1,...,\mathbf{x}_N,$ along with the N response entries of the first label. Considering the data in Table 3.1 as an example, we therefore fit the base classifier to the binary classification data given in Table 3.4, in order to find f_1.

Table 3.4: Binary classification data of Table 3.1 used to find f_1

Input	Y_1
$\mathbf{x}_1$	1
$\mathbf{x}_2$	0
$\mathbf{x}_3$	1
$\mathbf{x}_4$	0
$\mathbf{x}_5$	1

In order to obtain the second binary classifier, $f_2,$ the input variables become $\{X_1,...,X_p\} \cup \{Y_1\}$. Therefore f_2 is found by fitting the base classifier to the binary classification data formed by

considering the N observations of the input variables $\{X_1,...,X_p\} \cup \{Y_1\}$, along with the N response entries of the second label. Considering the data in Table 3.1, we therefore fit the base classifier to the binary classification data given in Table 3.5 to find f_2.

Table 3.5: Binary classification data of Table 3.1 used to find f_2

Input		Y_2
$\mathbf{x}_1$	$y_{1,1}=1$	0
$\mathbf{x}_2$	$y_{2,1}=0$	0
$\mathbf{x}_3$	$y_{3,1}=1$	0
$\mathbf{x}_4$	$y_{4,1}=0$	1
$\mathbf{x}_5$	$y_{5,1}=1$	1

In general, f_k is trained using input variables $\{X_1,...,X_p\} \cup \{Y_1,...,Y_{k-1}\}$ and the response entries of the k^{th} label. Here $k=1,...,K$ so that a chain of K classifiers is formed, $\mathbf{f}=(f_1,...,f_K)$. Again consider the data in Table 3.1 as an example. In this case we find a chain of $K=4$ classifiers by fitting the base classifier to each of the four binary classification datasets given in Table 3.6.

Table 3.6: Classifier chains applied to data in Table 3.1

Input	Y_1
$\mathbf{x}_1$	1
$\mathbf{x}_2$	0
$\mathbf{x}_3$	1
$\mathbf{x}_4$	0
$\mathbf{x}_5$	1

f_1

Input		Y_2
$\mathbf{x}_1$	1	0
$\mathbf{x}_2$	0	0
$\mathbf{x}_3$	1	0
$\mathbf{x}_4$	0	1
$\mathbf{x}_5$	1	1

f_2

Input			Y_3
$\mathbf{x}_1$	1	0	0
$\mathbf{x}_2$	0	0	1
$\mathbf{x}_3$	1	0	0
$\mathbf{x}_4$	0	1	1
$\mathbf{x}_5$	1	1	1

f_3

Input				Y_4
$\mathbf{x}_1$	1	0	0	1
$\mathbf{x}_2$	0	0	1	1
$\mathbf{x}_3$	1	0	0	0
$\mathbf{x}_4$	0	1	1	0
$\mathbf{x}_5$	1	1	1	0

f_4

Once the chain, $\mathbf{f}$, is complete, containing K classifiers, the classification of an unseen observation, $\mathbf{x}$, can be made. Starting at f_1 we find the $0/1$ classification, $\hat{f}_1(\mathbf{x})$. Then the second binary classifier, f_2, predicts the relevance of the second label, given the input space augmented by the $0/1$ value, $\hat{f}_1(\mathbf{x})$. In general, the k^{th} binary classifier predicts the relevance of the k^{th} label, given the input space augmented by the $0/1$ classifications of all the previous binary classifiers in the chain. This is illustrated in Figure 3.1.

Input							$0/1$ classification	
x_1	x_2	…	x_p				→	$\hat{f}_1(\mathbf{x})$
x_1	x_2	…	x_p	$\hat{f}_1(\mathbf{x})$			→	$\hat{f}_2(\mathbf{x})$
x_1	x_2	…	x_p	$\hat{f}_1(\mathbf{x})$	$\hat{f}_2(\mathbf{x})$		→	$\hat{f}_3(\mathbf{x})$
⋮	⋮	⋮	⋮	⋮	⋮			⋮
x_1	x_2	…	x_p	$\hat{f}_1(\mathbf{x})$	$\hat{f}_2(\mathbf{x})$	… $\hat{f}_{K-1}(\mathbf{x})$	→	$\hat{f}_K(\mathbf{x})$

Final classification	Y_1	Y_2	…	Y_K
$\hat{\mathbf{f}}(\mathbf{x}) =$	$\hat{f}_1(\mathbf{x})$	$\hat{f}_2(\mathbf{x})$	…	$\hat{f}_K(\mathbf{x})$

Figure 3.1: Classification procedure of classifier chains for observation $\mathbf{x}$

Since the classifier chains method passes label information between classifiers, it incorporates label correlation and overcomes the disadvantage of binary relevance of not taking label correlation into consideration. Read *et al*. (2011) state that although the additional attributes make up a small part of the total attribute space, if strong correlations exist, these attributes give any base classifier relatively more predictive power.

Note that only one ordering of the K labels is used in the classifier chains method. This is a possible disadvantage of classifier chains. Predictive performance of the fitted classifier chains

model is very dependent on the order of the chain, *i.e.* the order of the labels $Y_1,...,Y_K$, it was trained on. It may happen that one or more of the first classifiers in the chain classify poorly, which means that the classifiers later in the chain use these poor classifications to find a classification for the labels that occur later in the chain. This concern may be addressed by considering an ensemble of classifier chains, which is described in the next section.

3.5 ENSEMBLES OF CLASSIFIER CHAINS

Instead of considering only one ordering of the K labels, as is the case for the classifier chains method, the ensemble of classifier chains method considers multiple orderings of the K labels and fits an ensemble of classifier chains. This multi-label ensemble method, given in Read *et al*. (2011), is described in this section.

Again consider a multi-label classification dataset, $\{(\mathbf{x}_i, \mathbf{y}_i), i = 1, 2, ..., N\}$. It is straightforward to fit an ensemble of classifier chains. Instead of considering only one order of the labels to form one chain, $\mathbf{f}$, we consider m random orderings of the K labels in order to fit m classifier chains, $\mathbf{f}_1,...,\mathbf{f}_m$. Each classifier is trained on a bootstrap sample formed by sampling N times with replacement from the observations in the multi-label training dataset. Therefore, along with the m random orderings of the K labels, m bootstrap samples of the multi-label classification training dataset are required to fit m classifier chains, $\mathbf{f}_1,...,\mathbf{f}_m$. It was initially suggested, in Read *et al.* (2009), to use a subset of the observations in the training dataset to fit each classifier. Thus, sampling without replacement in this case. However, it was discovered that using bootstrap samples can achieve higher predictive performance with only a small increase in computational cost. We therefore focus on the bootstrap implementation of ensembles of classifier chains in our description of the method.

To form the first classifier, $\mathbf{f}_1$, the N $p-$component observations of the first bootstrap sample are considered along with their corresponding response entries of the K labels. Using these N observations and the first random label ordering, a classifier chain is fit as described in Section 3.4. This produces $\mathbf{f}_1$. In order to find the second classifier, we consider the second bootstrap sample observations along with the second random label ordering and fit a classifier chain. This produces $\mathbf{f}_2$. We continue in this way until m multi-label classifiers, $\mathbf{f}_1,...,\mathbf{f}_m$, have been found.

Classifying an unseen observation, $\mathbf{x}$, proceeds as follows. Each of the m multi-label classifiers, $\mathbf{f}_1,...,\mathbf{f}_m$, produces a multi-label classification for observation $\mathbf{x}$ by applying the classifier chain classification procedure described in Section 3.4. Thus m multi-label classifications are found for observation $\mathbf{x}$, given by $\hat{\mathbf{f}}_1(\mathbf{x}),\hat{\mathbf{f}}_2(\mathbf{x}),...,\hat{\mathbf{f}}_m(\mathbf{x})$. Now consider the binary predictions produced by each of the classifiers, $\mathbf{f}_1,...,\mathbf{f}_m$, for each of the K labels. For each label, the mean of the binary predictions over the m classifications is found. This produces a vector of confidence outputs, $\hat{\mathbf{w}} = [\hat{w}_1, \hat{w}_2, ..., \hat{w}_K]$, where $\hat{w}_k = \frac{1}{m}\sum_{j=1}^{m} \hat{f}_{j,k}(\mathbf{x})$. A threshold function can be applied to $\hat{\mathbf{w}}$ to obtain a final classification for observation $\mathbf{x}$. For example, we may conclude that label Y_k is present if $\hat{w}_k \geq 0.5$, otherwise it is not present.

If $K = 4$ and $m = 5$, the classification procedure for an unseen observation, $\mathbf{x}$, when applying a threshold of 0.5 for each label, resembles Figure 3.2.

	Y_1	Y_2	Y_3	Y_4
$\hat{\mathbf{f}}_1(\mathbf{x}) =$	$\hat{f}_{1,1}(\mathbf{x}) = 0$	$\hat{f}_{1,2}(\mathbf{x}) = 1$	$\hat{f}_{1,3}(\mathbf{x}) = 1$	$\hat{f}_{1,4}(\mathbf{x}) = 1$
$\hat{\mathbf{f}}_2(\mathbf{x}) =$	$\hat{f}_{2,1}(\mathbf{x}) = 1$	$\hat{f}_{2,2}(\mathbf{x}) = 0$	$\hat{f}_{2,3}(\mathbf{x}) = 0$	$\hat{f}_{2,4}(\mathbf{x}) = 1$
$\hat{\mathbf{f}}_3(\mathbf{x}) =$	$\hat{f}_{3,1}(\mathbf{x}) = 0$	$\hat{f}_{3,2}(\mathbf{x}) = 0$	$\hat{f}_{3,3}(\mathbf{x}) = 1$	$\hat{f}_{3,4}(\mathbf{x}) = 1$
$\hat{\mathbf{f}}_4(\mathbf{x}) =$	$\hat{f}_{4,1}(\mathbf{x}) = 0$	$\hat{f}_{4,2}(\mathbf{x}) = 0$	$\hat{f}_{4,3}(\mathbf{x}) = 1$	$\hat{f}_{4,4}(\mathbf{x}) = 1$
$\hat{\mathbf{f}}_5(\mathbf{x}) =$	$\hat{f}_{5,1}(\mathbf{x}) = 0$	$\hat{f}_{5,2}(\mathbf{x}) = 1$	$\hat{f}_{5,3}(\mathbf{x}) = 0$	$\hat{f}_{5,4}(\mathbf{x}) = 1$
$\hat{\mathbf{w}} =$	$\hat{w}_1 = \frac{1}{5}$	$\hat{w}_2 = \frac{2}{5}$	$\hat{w}_3 = \frac{3}{5}$	$\hat{w}_4 = 1$
Final classification	0	0	1	1

Figure 3.2: Classification procedure for ensembles of classifier chains

In Figure 3.2 the multi-label classification of observation $\mathbf{x}$ for each of the $m = 5$ classifier chains is given. For each of the $K = 4$ labels, the mean of the binary predictions over the $m = 5$

classifications is found. A final classification of "1" is assigned if this mean exceeds a threshold value of 0.5, else a final classification of "0" is assigned.

Since the classifications of m classifier chains are combined in order to produce a final classification, this is a multi-label ensemble method with base learner the classifier chains method. Fitting an ensemble of classifier chains reduces the effect of a poor label ordering on the overall classification accuracy of the model. Furthermore, the procedure still remains quite fast compared to the classifier chains method. The time cost is only affected linearly by the number of orderings used in the ensemble. Results in Read *et al*. (2011) show that using an ensemble of classifier chains has a significant positive effect on predictive performance compared to classifier chains. Setting $m \geq 10$ allows ensembles of classifier chains to perform better than classifier chains in almost all cases presented in the experimental evaluation given in Read *et al*. (2011). On the other hand, in the Madjarov *et al*. (2012) review, classifier chains fared better than ensembles of classifier chains for some datasets.

As an ensemble of binary transformations, it may be that fitting an ensemble of classifier chains can result in a very large number of observations to be processed. Each label gives rise to $m \times N$ observations in the fitting process. Furthermore, this calls for considerable redundancy in the learning space. Read *et al*. (2011) propose a simple strategy for reducing redundancy in the learning space and therefore also causing a reduction in time and memory requirements. This comes at only a slight loss in predictive performance. For each iteration, along with each random label ordering, they propose taking random subsets of both the observations and the predictors when fitting the classifier chains model in each case. Their study investigates the loss in predictive performance compared to the reduction in running time when the percentages of observations and predictors used to fit each ensemble member, are decreased. It appears that comparable results in predictive performance can be obtained for significantly less computational requirements. For example, in the experimental evaluation given in Read *et al.* (2011), with $m = 10$, using subsets that contain 75% of training observations and 50% of predictors, accuracy is negligibly less compared to using 100% of both.

3.6 RANDOM FORESTS OF PREDICTIVE CLUSTERING TREES

Bagging and random forests are two ensemble methods often used in the context of decision trees. We can also use bagging and random forests for multi-label classification, as outlined in Kocev *et al*. (2007).

For a single-label classification dataset, bagging is implemented by first forming B bootstrap samples of the training data. If a decision tree is the chosen base classifier, a decision tree is fit to each of the B bootstrap samples. Suppose each tree produces a classifier for a $K-$class response. For an unseen observation, $\mathbf{x}$, the bagged estimate, $\hat{f}_{bag}(\mathbf{x})$, is a $K-$vector, $\left[p_1(\mathbf{x}), p_2(\mathbf{x}), \ldots, p_K(\mathbf{x})\right]$. Here, $p_k(\mathbf{x})$ is the proportion of trees predicting class k for $\mathbf{x}$. The bagged classifier selects the class having the most votes, *i.e.* $\arg\max_k \hat{f}_{bag}(\mathbf{x})$. Posterior probabilities of each class may also be used instead of a majority class voting procedure (Hastie *et al.*, 2009: 283). With bagging we aim to average many noisy but approximately unbiased models in order to reduce the variance. Decision trees are therefore often the chosen base classifier.

Random forests aim to improve on the variance reduction of bagging. This is achieved by reducing the correlation between the B trees without increasing the variance too much. When fitting random forests, we also construct B bootstrap samples of the single-label data; however, an additional constraint is added when fitting the B decision trees: at each split, when constructing a tree, only a random subset of the complete set of predictors available, is considered (Hastie *et al.*, 2009: 587).

Instead of using a single-label classification decision tree as base learner, we may extend bagging and random forests to the multi-label scenario by using a decision tree that produces multi-label classifications. Multi-objective decision trees (MODTs), which are an instantiation of predictive clustering trees (PCTs), can be used to produce multi-label classifications.

A PCT is viewed as a hierarchy of clusters. The topnode of the PCT contains all the training data. This topnode is partitioned into smaller clusters, and these resulting clusters are partitioned into smaller clusters, and so on as we move down the tree. The clusters are formed by maximizing the variance reduction achieved when partitioning the training observations, thereby forming clusters with maximum cluster homogeneity. In order to construct a PCT we require a so-called variance function and prototype function. The homogeneous clusters are formed by considering the variance function as a measure of cluster impurity. The prototype function is used to define the classification given by a cluster.

MODTs extend the variance and prototype functions to multi-label learning. In this case, cluster impurities are measured using the multi-label structure of observations, while the prototype function produces a multi-label classification. The variance function can for example be defined

as the sum of entropies per label, $\sum_{k=1}^{K} Entropy(\mathbf{X}, Y_k)$. The prototype function, on the other hand, may return a $K-$component vector defining the classification for a cluster. Here the k^{th} entry may contain the majority vote or posterior probability of Y_k within the cluster.

Suppose we use such a MODT as base learner when implementing bagging or random forests. Classification of an unseen observation, $\mathbf{x}$, produces a $K-$vector estimate, $\left[p_1(\mathbf{x}), p_2(\mathbf{x}), \ldots, p_K(\mathbf{x})\right]$. In this case, $p_k(\mathbf{x})$ is the majority vote or posterior probability of label Y_k produced when considering the classification of observation $\mathbf{x}$ across all the fitted MODTs. If majority votes are used, $\left[p_1(\mathbf{x}), p_2(\mathbf{x}), \ldots, p_K(\mathbf{x})\right]$ is the final multi-label classification. If posterior probabilities are used, a threshold function is applied to $\left[p_1(\mathbf{x}), p_2(\mathbf{x}), \ldots, p_K(\mathbf{x})\right]$ in order to produce a final multi-label classification.

3.7 RANDOM k-LABELSETS

This section is based on Tsoumakas *et al*. (2011). Random $k-$labelsets (RAkEL) is an ensemble method that attempts to improve on the limitations of the label powerset method. Two different versions of RAkEL exist; the first is a disjoint version of the method, RAkEL$_d$, and the second an overlapping version, RAkEL$_o$.

3.7.1 Description

If the collection of K labels, $Y_1, \ldots, Y_K$, is denoted by L, the **disjoint** version of the random $k-$labelset method, RAkEL$_d$, is implemented as follows. We aim to partition L randomly into m disjoint labelsets, R_j, $j = 1, \ldots, m$, where $\bigcap_{j=1}^{m} R_j = \phi$. If the labelset-size is chosen as k, we have $m = \left\lceil \frac{K}{k} \right\rceil$, *i.e.* the ceiling function of $\frac{K}{k}$. If $\frac{K}{k}$ is an integer, all m labelsets are of size k and are therefore all $k-$labelsets. If, however, $\frac{K}{k}$ is not an integer, labelsets R_j, $j = 1, \ldots, m-1$, are of size k and are referred to as $k-$labelsets. In this case, R_m contains the remaining $K \bmod k$ labels. For example, suppose $K = 10$ and $k = 3$, then $L = \{Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7, Y_8, Y_9, Y_{10}\}$. An

example of $m = 4$ disjoint labelsets can then be: $R_1 = \{Y_1, Y_2, Y_4\}$, $R_2 = \{Y_3, Y_5, Y_7\}$, $R_3 = \{Y_6, Y_8, Y_{10}\}$ and $R_4 = \{Y_9\}$.

After the m disjoint labelsets have been identified, RAkEL$_d$ applies the label powerset procedure separately to each of the m labelsets. Thus, m multi-label classifiers are found, h_j , $j = 1, ..., m.$

In other words, the multi-label data are broken up into m smaller multi-label datasets and the label powerset method is applied to each. If we again consider the above example, the data can resemble Figure 3.3.

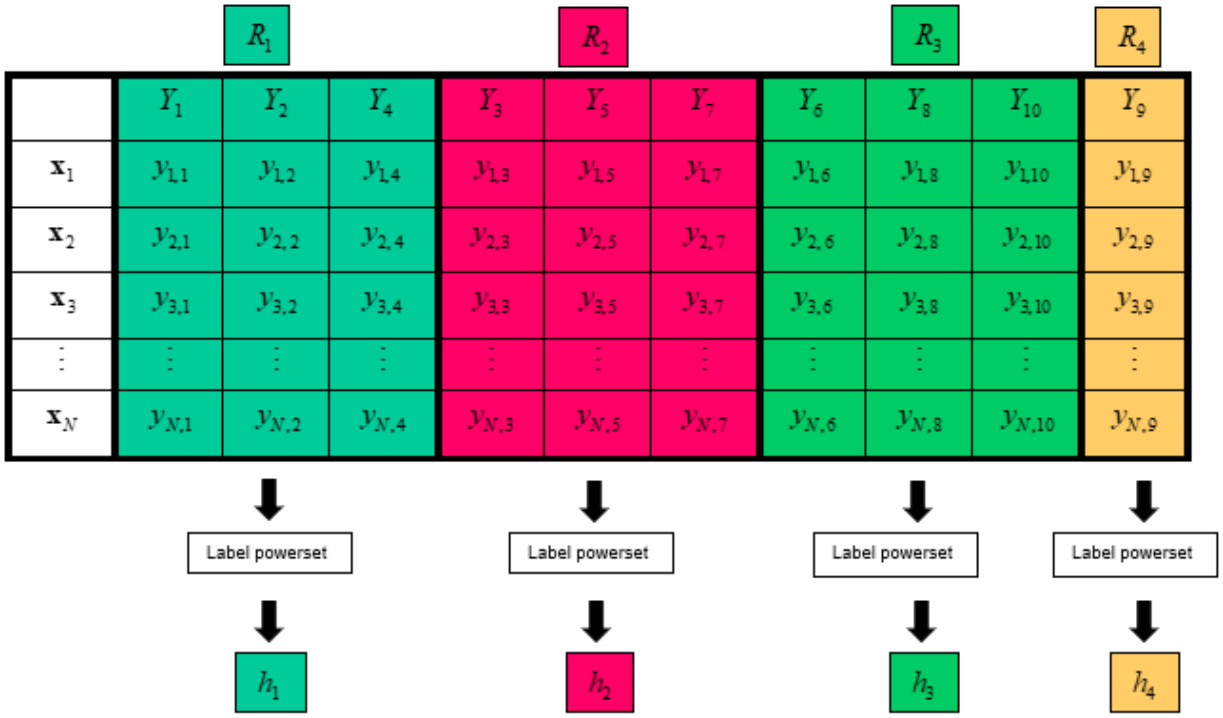


Figure 3.3: RAkEL$_d$

In Figure 3.3, the multi-label data, consisting of N observations and $K = 10,$ labels are divided into $m = 4$ labelsets, R_j , $j = 1, ..., 4.$ The training sets consist of all N observations annotated

with their corresponding labels in R_j , $j=1,...,4$. Denote these $m=4$ multi-label datasets by D_j , $j=1,...,4$ (also given by the shaded areas in Figure 3.3). The label powerset method is applied to each D_j , $j=1,...,4$. Thus, each unique set of labels that exists in each of D_j , $j=1,...,4$, is regarded as a class and therefore each of D_j , $j=1,...,4$, is transformed to a multi-class dataset. A standard multi-class classifier is applied to the multi-class version of each D_j , $j=1,...,4$, and h_j , $j=1,...,4$, are found.

Note that since the multi-label data are divided in this way, this may lead to the empty set appearing as an annotation for an observation. For example, in Figure 3.3, consider R_1 and suppose, for observation $\mathbf{x}_3$ we have $\left[y_{3,1}=0 \text{ , } y_{3,2}=0 \text{ , } y_{3,4}=0 \right]$. This is not a problem. The empty set is simply another class of the multi-class version of D_1. Observation $\mathbf{x}_3$ is therefore not excluded when finding h_1. The empty set is an acceptable annotation for an observation, as was also stated in our definition of multi-label data.

Suppose an unseen observation, $\mathbf{x}$, has to be classified. The functions h_j , $j=1,...,m$, are applied to the unseen observation. Each classifier, h_j , $j=1,...,m$, classifies the unseen observation to the class with the highest posterior probability in the individual multi-class problems. Since the labelsets, R_j , $j=1,...,m$, are disjoint, the m classifications are simply gathered together to produce the final multi-label classification. Once again considering the data in Figure 3.3 as an example, we may have a classification for $\mathbf{x}$ resembling Figure 3.4.

	Class with highest posterior probability
h_1	Y_1 and Y_4 present
h_2	Empty set
h_3	Y_6, Y_8 and Y_{10} present
h_4	Y_9 present

	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9	Y_{10}
h_1	1	0		1						
h_2			0		0		0			
h_3						1		1		1
h_4									1	
Final prediction	1	0	0	1	0	1	0	1	1	1

Figure 3.4: Prediction using RAkEL$_d$

In Figure 3.4, for each of the $m=4$ classifiers, h_j, $j=1,...,4$, the class that resulted in the highest posterior probability is given. Since each classification only involves those labels that are present in each of R_j, $j=1,...,4$, the classifications have to be combined in order to produce the overall multi-label classification. Because different multi-label classifiers are combined in order to produce a final classifier, it is clear that RAkEL$_d$ is an ensemble method with base learner the label powerset method.

The **overlapping** version of random $k-$labelsets, RAkEL$_o$, also aims to partition L randomly into m labelsets, R_j, $j=1,...,m$. However, the labelsets no longer have to be disjoint, as was the case with RAkEL$_d$. Denote by L^k the set of all distinct $k-$labelsets of L. The size of L^k is given by $\binom{K}{k}$. If m labelsets of size k are desired, RAkEL$_o$ selects m $k-$labelsets, R_j, $j=1,...,m$, by sampling without replacement from L^k. By doing this, some of the labelsets

may overlap, *i.e.* some of the K labels may be found in more than one of R_j, $j=1,...,m$. If $mk > K$, the overlap definitely occurs. The training set once again consists of all N observations annotated with their corresponding labels in R_j, $j=1,...,m$, so that, similar to RAkEL$_d$, the multi-label data are divided into m smaller multi-label datasets, D_j, $j=1,...,m$. Once again the label powerset procedure is applied to the m multi-label datasets so that m multi-label classifiers are found, h_j, $j=1,...,m$.

As an example, suppose $K=10$, $k=3$, $m=6$ and $L=\{Y_1,Y_2,Y_3,Y_4,Y_5,Y_6,Y_7,Y_8,Y_9,Y_{10}\}$. The $m=6$ labelsets can then for example be: $R_1=\{Y_1,Y_2,Y_3\}$, $R_2=\{Y_3,Y_7,Y_{10}\}$, $R_3=\{Y_4,Y_6,Y_9\}$, $R_4=\{Y_5,Y_8,Y_9\}$, $R_5=\{Y_1,Y_6,Y_7\}$ and $R_6=\{Y_1,Y_8,Y_9\}$. For this example the multi-label data resemble Figure 3.5.

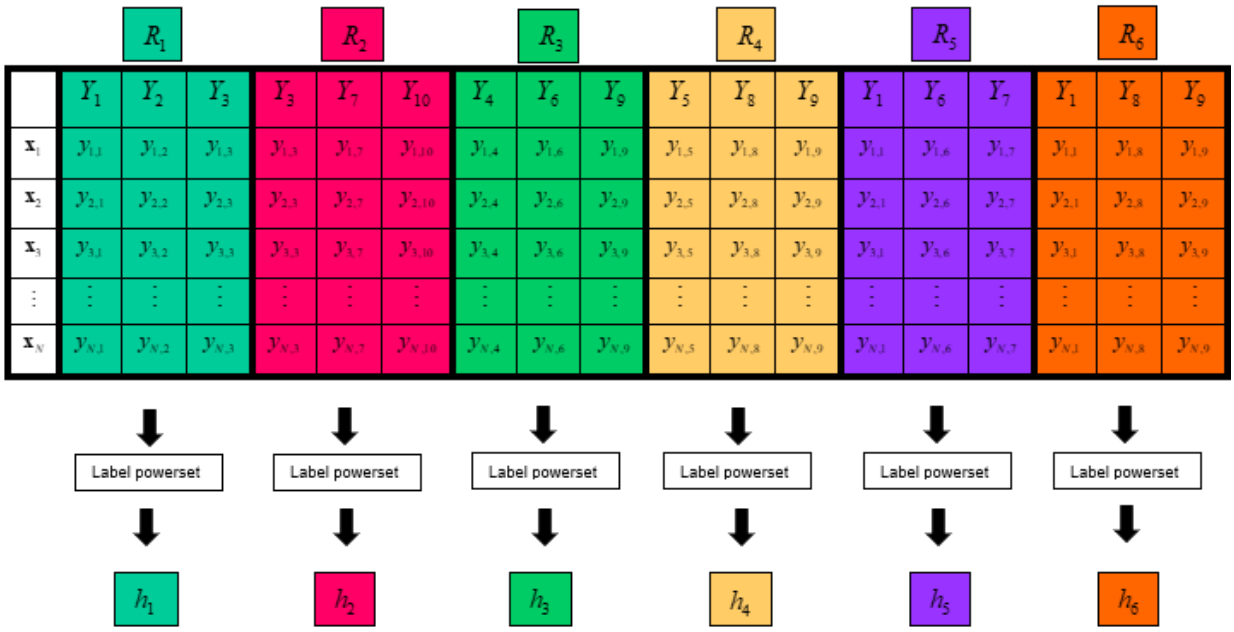


Figure 3.5: RAkEL$_o$

In Figure 3.5 we see that the training set consists of the N observations annotated with their corresponding labels in R_j, $j=1,...,6$, of our example. Therefore the multi-label data are divided

into $m=6$ smaller multi-label datasets, D_j , $j=1,...,6$. The label powerset method is applied to each D_j , $j=1,...,6$. Once again this means that each unique set of labels that exists in each of D_j is regarded as a class and therefore each D_j is transformed to a multi-class dataset. A standard multi-class classifier is applied to the multi-class version of each D_j , $j=1,...,6$ and h_j , $j=1,...,6$ is found.

Suppose an unseen observation, $\mathbf{x}$, has to be classified by RAkEL$_o$. The functions h_j , $j=1,...,m$, are applied to the unseen observation. Each of these classifiers classifies the unseen observation to the class with the highest posterior probability in the individual multi-class problems. In other words, we in fact obtain a binary prediction for each label in the corresponding $k-$labelset of each R_j , $j=1,...,m$. Since the labelsets may overlap, the m classifications cannot simply be gathered together, as in the case of RAkEL$_d$, to find the final multi-label classification. Instead, RAkEL$_o$ considers all the binary predictions produced by the classifiers, h_j , $j=1,...,m$, for each label. For each label, the mean of these binary predictions is calculated. When this mean exceeds a threshold of 0.5 for a label, the final decision is that the label is in fact present for observation $\mathbf{x}$, otherwise it is not present.

Consider the data in Figure 3.5 as an example. Suppose we aim to classify an unseen observation $\mathbf{x}$. We may have a classification for $\mathbf{x}$ resembling Figure 3.6.

	Class with highest posterior probability
h_1	Empty set
h_2	Y_{10} present
h_3	Y_4 and Y_6 present
h_4	Y_5, Y_8 and Y_9 present
h_5	Y_1 and Y_6 present
h_6	Y_1, Y_8 and Y_9 present

	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9	Y_{10}
h_1	0	0	0							
h_2			0				0			1
h_3				1		1			0	
h_4					1			1	1	
h_5	1					1	0			
h_6	1							1	1	
Final prediction	1	0	0	1	1	1	0	1	1	1

Figure 3.6: Prediction using RAkEL$_o$

In Figure 3.6, for each of the $m=6$ classifiers, h_j, $j=1,...,6,$ the class that resulted in the highest posterior probability is given. This results in binary predictions for each of the $k=3$ labels in each of the $m=6$ labelsets. Since the labelsets overlap, the mean of the binary predictions is calculated for each label. The labels for which the mean values exceed a threshold of 0.5 are marked present with "1", else the label is not present, denoted with "0". Again we see that different multi-label classifiers are combined in order to produce a final classifier, and therefore it is also clear that RAkEL$_o$ is an ensemble method with base learner the label powerset method.

It should be noted that since we sample m $k-$labelsets, R_j, $j=1,...,m,$ from L^k, it can happen that some of the labels in L are not present in any of the labelsets, R_j, $j=1,...,m.$ In this case

the algorithm will not make any classifications for such a label. Setting m appropriately large can prevent this from happening.

3.7.2 Performance

The empirical study of Tsoumakas *et al*. (2011) shows evidence that both RAkEL$_d$ and RAkEL$_o$ perform better than the label powerset procedure. This is particularly true when the number of labels of the relevant multi-label dataset is large. This may be because RAkEL$_d$ and RAkEL$_o$ possess the following advantages over the label powerset method.

Since the final predictions of RAkEL$_d$ and RAkEL$_o$ are based on combining predictions that occur for m different labelsets, it is possible for both procedures to predict a labelset that does not appear in the training data. Recall that this is not the case for the label powerset method. The fact that m label powerset classifiers are trained on different output spaces in RAkEL, offers a diverse view for the classification of an unseen observation. This gives RAkEL an advantage.

The label powerset method has the disadvantage that it may result in a multi-class dataset with a large number of classes and few observations per class. This increases computational cost and makes the learning process difficult. This is especially true for multi-label datasets with a large number of labels. In RAkEL, the division of the multi-label data into m smaller multi-label datasets, D_j, $j=1,...,m,$ can hopefully prevent this from happening.

Each of the m smaller multi-label datasets have $k<K$ labels. For RAkEL$_d$ the m^{th} dataset contains $K \bmod k$ labels, which is also less than K when $k<K$. The m multi-class datasets, obtained by transforming the m multi-label datasets, D_j, $j=1,...,m,$ are characterized by a much more balanced distribution of observations per class. Therefore, a standard multi-class classifier is applied to m simpler multi-class datasets.

Recall that the total number of possible labelsets for a multi-label dataset with K labels is upper bounded by $\min\left(N, 2^K\right)$. Therefore, for the m smaller multi-label datasets, D_j, $j=1,...,m,$ this upper-bound is given by $\min\left(N, 2^k\right)$, where $k<K$. Suppose the complexity of the multi-class classification algorithm used in RAkEL is given by $O\left(g\left(\ G\ ,\ N\ ,\ A\ \right)\right)$, where G denotes the number of classes, N denotes the number of observations and A the predictive attributes. The

complexity of RAkEL is then given by $O\left(m\ g\left(\ \min\left(N, 2^k\right),\ N\ ,\ A\ \right)\right)$. This complexity is linear with respect to the number of label powerset classifiers, m, and grows exponentially with respect to k. Depending on the multi-class classification algorithm used, the m classification tasks can potentially be less computationally intensive than applying the label powerset procedure to the full multi-label dataset, containing K labels.

RAkEL$_o$ normally achieves higher predictive performance than RAkEL$_d$. This might have been expected. The overlapping labels of RAkEL$_o$ is advantageous. The m classifiers are trained on different output spaces giving a diverse view for the classification of an unseen observation. Now, because some labels overlap, a voting procedure is required to produce a final classification. The voting procedure increases the possibility of an incorrect classification being outvoted.

RAkEL$_o$ also fairs well when compared to other multi-label learning methods, not only the label powerset method. As always, no method outperforms all the others for every dataset. The empirical study of Tsoumakas *et al*. (2011) shows RAkEL$_d$ outperforming RAkEL$_o$ for a special dataset containing a small number of distinct label combinations, despite its large number of labels. The label powerset method also outperforms RAkEL$_d$ for a benchmark dataset with a small number of labels.

3.7.3 Parameters

We have two parameters for RAkEL$_o$, namely the number of classifiers, m, and the size of the labelsets, k. For RAkEL$_d$, m is given by $\lceil \frac{K}{k} \rceil$, and we therefore have one parameter, k, in this case.

One of the possible reasons that RAkEL has better predictive performance than the label powerset method is that RAkEL transforms the multi-label data into smaller multi-label datasets, containing only k labels each. We might therefore prefer the value of k to be appropriately small when RAkEL is implemented. The empirical study of Tsoumakas *et al*. (2011) shows that RAkEL$_d$ generally performs well for small values of k. Unfortunately, a smaller value of k means that the corresponding labelsets are smaller so that we potentially take less correlations into consideration. This could be why a small value of k does not always guarantee better results. In general, however, a value of k close to the value of K, performs worse. It is recommended to

set k to a small value, especially in scenarios where the relevant multi-label dataset has a large number of labels.

As mentioned, the overlapping labels of RAkEL$_o$ is advantageous, because it increases the possibility that the voting procedure will eliminate some misclassifications. The expected number of predictions per label is given by $\frac{km}{K}$. Since K is fixed, this value is influenced by parameter-values m and k. The empirical study of Tsoumakas *et al*. (2011) compares the percentage of improvement RAkEL$_o$ has over the label powerset method for increasing values of k. With a fixed value of m, varying k over a series of small values (2 to 10), increases the percentage improvement. For fixed K and m, a larger value of k results in a larger value of $\frac{km}{K}$. We therefore have more votes for each label and more votes can lead to better classifications. We may therefore conclude that a larger expected number of predictions per label, $\frac{km}{K}$, causes better predictive accuracy. If we require a large value for $\frac{km}{K}$, increasing the values of m or k can achieve this. However, since the complexity of RAkEL grows exponentially with respect to k, but only linearly with respect to m, we choose to increase the value of m instead of the value of k. Now m should be chosen appropriately large to compensate for a small value of k. We would furthermore like to set m to a large value in order to prevent any labels from being unrepresented in R_j, $j=1,...,m$. For all datasets, however, the value of m can be increased up to a certain value, after which an additional increase causes little improvement in performance. In most cases K is a good approximation of this number. Tsoumakas *et al*. (2011) suggests using a small value for k such as $k=3$ and a value that is between K and $2K$ for m.

In the next chapter a new ensemble method of multi-label classification is introduced. We also aim to compare the newly proposed method to the existing learning methods given in this chapter. Comparisons are drawn in terms of the fitting and classification procedures as well as predictive performance on benchmark datasets.

CHAPTER 4: NEW APPROACH

4.1 INTRODUCTION

In this chapter a new ensemble method of multi-label classification is introduced. The method aims to split the training data in a label-dependent way whilst incorporating label correlation. A decision tree is used as splitting tool. For this reason, we will refer to this method as label dependent splitting (LDsplit) with trees.

4.2 LDSPLIT WITH TREES

Suppose a multi-label classification training dataset, $\{(\mathbf{x}_i, \mathbf{y}_i), i = 1, 2, ..., N\}$, is given, containing N p - component observations and a total of K labels. The collection of K labels, $Y_1, Y_2, ..., Y_K$, is denoted by L. Denote by L^P the set of all permutations of the K labels, $Y_1, Y_2, ..., Y_K$. The size of L^P is thus $K!$. LDsplit with trees is implemented as follows.

We start by sampling without replacement M unique permutations from L^P. Denote each unique permutation by P_j, where $j = 1, 2, ..., M$. If $M = K!$, all the permutations in L^P are selected. Each permutation, P_j, $j = 1, ..., M$, has K entries, denoted by $P_{j,s}$, $s = 1, ..., K$, and each of these entries correspond to one of the K labels in the training data. For example, say $M = 3$ and $K = 4$, then we could have $P_1 = [Y_3\ Y_2\ Y_4\ Y_1]$, $P_2 = [Y_4\ Y_3\ Y_1\ Y_2]$ and $P_3 = [Y_1\ Y_4\ Y_2\ Y_3]$. In this case $P_{1,1} = Y_3$, $P_{1,2} = Y_2$, $P_{1,3} = Y_4$ and $P_{1,4} = Y_1$. For each of the M permutations a tree-structure, T_j, $j = 1, ..., M$, is constructed as follows.

4.2.1 Fitting a tree-structure

In order to construct one tree-structure, T_j, for a given permutation, P_j, we proceed as follows. Consider all N training observations, $\{(\mathbf{x}_i), i = 1, 2, ..., N\}$, along with the label indicator variable which corresponds to the first entry of P_j, *i.e.* the N response entries of label $P_{j,1}$. Since this data, $\{(\mathbf{x}_i, y_i), i = 1, 2, ..., N\}$, are now in the form of a binary classification problem, a simple binary classifier, such as a tree, can easily be fit to the data. We fit a stump to this data, *i.e.* a tree containing only one split point. This means that an optimal splitting variable and its corresponding optimal splitting value are identified. The identified variable and split point minimize some

measure of node impurity, for example misclassification error. Observations for which the corresponding variable value is less than the splitting value move down the first (left) branch of the tree and observations for which the corresponding variable value is larger than or equal to the splitting value move down the second (right) branch of the tree. The N observations are thus split into two disjoint groups, called nodes. Suppose the root node of the tree, where all the observations are present, is numbered as the first node of the tree and therefore referred to as Node 1. We number the node containing the observations for which the splitting variable values are smaller than the splitting value as the second node, and refer to it as Node 2. Similarly, the node containing the observations for which the splitting variable values are larger than or equal to the splitting value is referred to as Node 3. Figure 4.1 demonstrates how fitting a stump to this binary classification data splits the data into two new nodes.

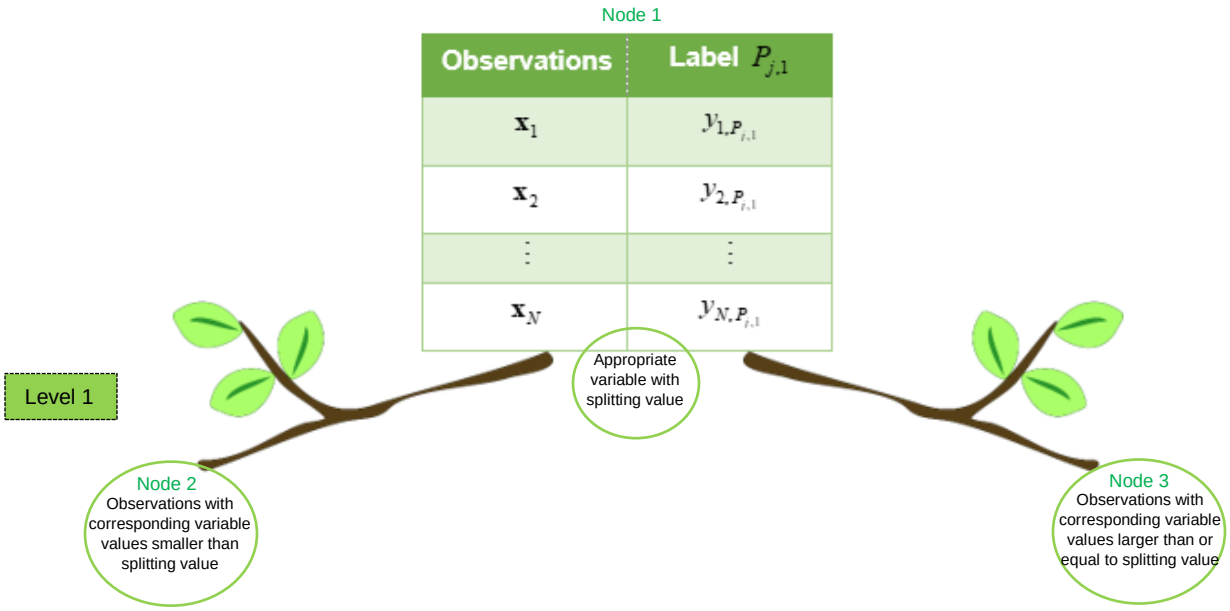


Figure 4.1: Representation of a stump fit to binary classification data

In Figure 4.1 we consider all N observations along with label $P_{j,1}$. A stump is fit to the binary classification data so that the N observations are split into two new nodes. We will refer to this initial splitting of the observations that results in the formation of Node 2 and Node 3 as "Level 1".

Now consider all the observations in Node 2 along with the label indicator variable which corresponds to the second entry of P_j, *i.e.* label $P_{j,2}$. A new binary classification dataset is formed

containing all the observations in Node 2 and their corresponding response entries for label $P_{j,2}$. Since this is once again a simple binary classification problem, a stump can easily be fit to this binary classification dataset. After fitting a stump, the observations in Node 2 are split into two new nodes. The node containing the observations for which the splitting variable values are smaller than the splitting value is named “Node 4” and the node containing the observations for which the splitting variable values are larger than or equal to the splitting value is named “Node 5”.

We can follow a similar process for the observations in Node 3. Consider all the observations in Node 3 and construct a binary classification dataset containing these observations and their corresponding response entries for label $P_{j,2}$. Fit a stump to this binary classification dataset to form two new nodes. The node containing the observations for which the splitting variable values are smaller than the splitting value is named “Node 6” and the node containing the observations for which the splitting variable values are larger than or equal to the splitting value is named “Node 7”. Figure 4.2 illustrates how Node 2 and Node 3 are each split into two new nodes by fitting stumps to the corresponding binary classification datasets.

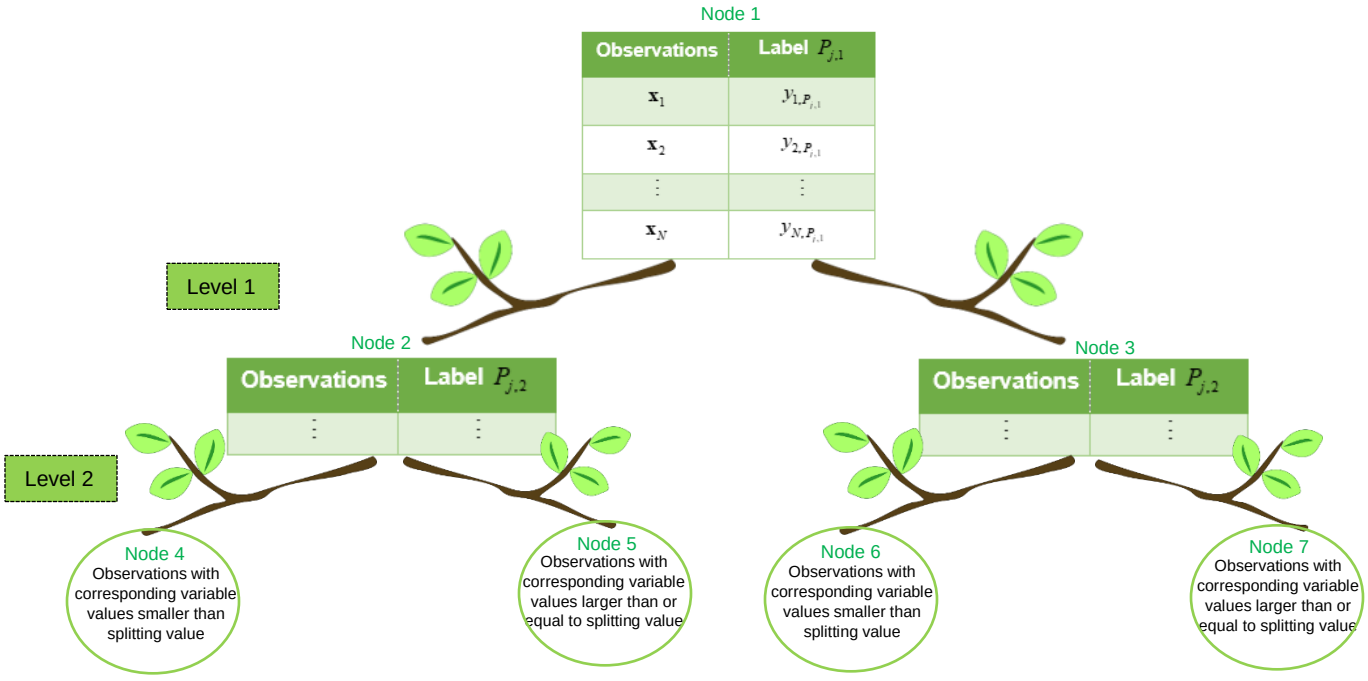


Figure 4.2: Representation of tree-structure with two levels

In Figure 4.2, after Level 1 has been formed, we consider all the observations in Node 2 along with label $P_{j,2}$. A stump is fit to the binary classification data so that the observations are split into two new nodes, Node 4 and Node 5. Similarly, we consider all the observations in Node 3 along with label $P_{j,2}$ and once again fit a stump to the binary classification data, forming Node 6 and Node 7. In other words, all the nodes that are formed in Level 1 are each considered individually along with the label $P_{j,2}$ so that each of these nodes is split into two new nodes by means of a stump. We will refer to this second series of splitting that results in the formation of Node 4, Node 5, Node 6 and Node 7 as "Level 2".

Continuing in this way means that all the nodes that are formed in Level 2 are each considered individually along with the label $P_{j,3}$ so that each of these nodes is again split into two new nodes by means of a stump. This third series of splitting is referred to as "Level 3" and will result in the formation of Node 8 to Node 15. Splitting continues until Level K is reached. In other words, the process will continue until all the nodes that are formed in Level $K-1$ is each considered individually along with the label $P_{j,K}$ so that each of these nodes is again split into two new nodes by means of a stump. Once level K has been reached, we have constructed the corresponding tree-structure, T_j, for a given permutation, P_j.

4.2.2 Issues regarding the tree-structure

Since a permutation contains K entries, a complete tree-structure consists of a total of K levels. Each level has a corresponding set of nodes and each of these nodes have a specific number. Table 4.1 summarizes the total number of nodes at each level as well as the specific numbers of the nodes that are present at each level.

Table 4.1: Summary of tree-structure levels

Level	Nodes present	Total number of nodes
Root / Level 0	1, i.e. root node	$1 = 2^0$
Level 1	2 , 3	2^1
Level 2	4 , 5 , 6 , 7	2^2
Level 3	8 , 9 , 10 , ... , 15	2^3
$\vdots$	$\vdots$	$\vdots$
Level k	2^k , 2^k+1 , ... , $2^{k+1}-1$	2^k
$\vdots$	$\vdots$	$\vdots$
Level K	2^K , 2^K+1 , ... , $2^{K+1}-1$	2^K

In general, at Level k, the nodes numbered 2^k , 2^k+1 , ... , $2^{k+1}-1$ are present, resulting in a total of 2^k nodes at this level. A complete tree-structure, consisting of K levels, therefore has a total of $2^{K+1}-1$ nodes. The nodes at Level K form terminal nodes since no further splitting occurs at this level. This means that in general a total of 2^K-1 splits are made for a complete tree-structure containing K levels.

A complete $K-$level tree-structure has 2^K-1 splits; however, we may frequently have the following situation in which the total number of splits will be less than 2^K-1. While constructing a tree-structure for a given permutation, P_j , it may happen that some splits results in a very small number of observations to be present in one or both of the resulting nodes. If this happens, it may be inappropriate to fit a stump to a node such as this, which contains only a few or even no observations. If the splitting variable and splitting value are determined by using only a few observations, the split may not be very informative or appropriate. For this reason, a minimum node-size, say 5, should be identified before the M tree-structures are fit. When constructing a tree-structure, before a stump is fit to a node at any of the levels, it should first be determined how many observations are present in the relevant node. Only when the number of observations in the node exceeds the pre-determined minimum node-size, will a stump be fit to the node. If a stump is not fit to the observations in the node, this node becomes a terminal node. To avoid confusion, even if a node becomes a terminal node at one of the levels preceding Level K, the

numbering of the nodes will not change. Nodes that would have sprouted from a node that has now become a terminal node, keep their numbers and are simply regarded as being empty. However, the total number of splits for the tree-structure will be less than $2^K - 1$ in this case.

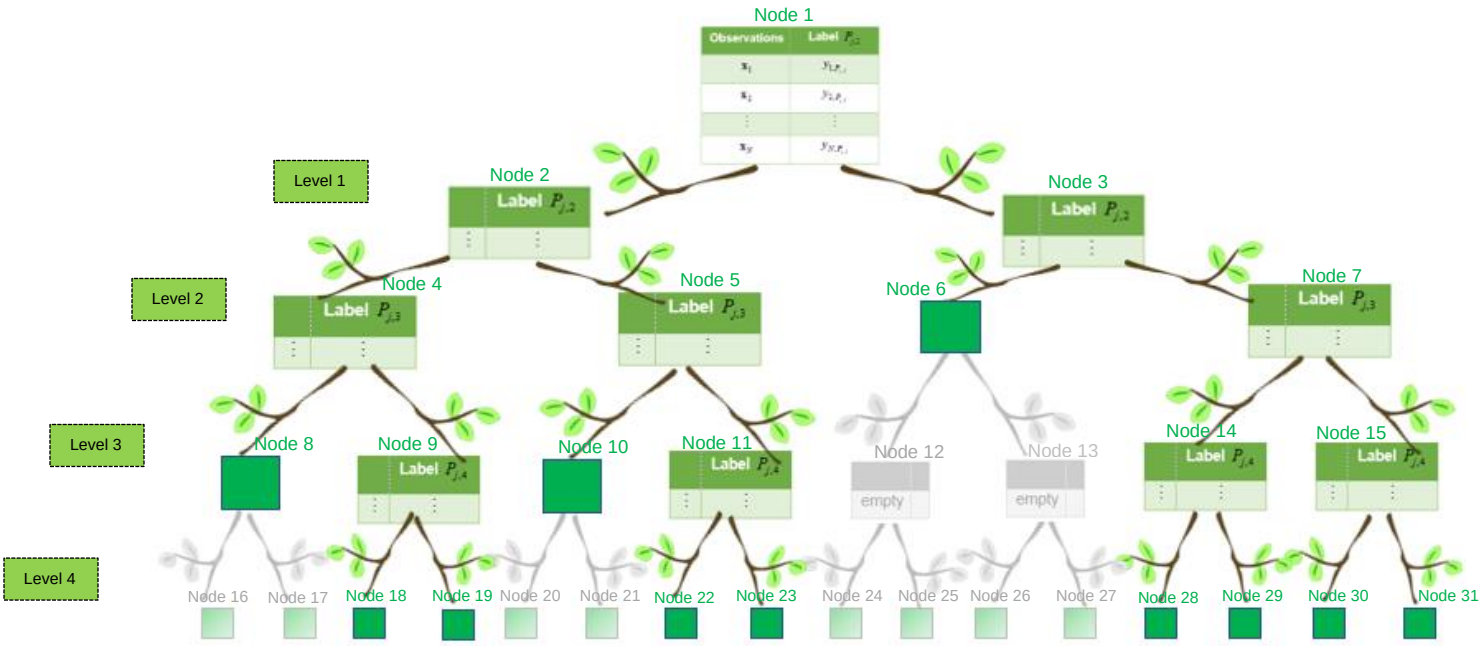


Figure 4.3: Tree-structure with four levels

For example, say the multi-label classification data consists of $K = 4$ labels. A four-level tree-structure for a given permutation P_j may resemble Figure 4.3. A four-level tree-structure in general has $2^{4+1} - 1 = 31$ nodes, assuming that splitting continuous along every branch. However, in the case of Figure 4.3, a stump was not fit to the observations in Node 6, Node 8 or Node 10. In other words, the number of observations in these nodes did not exceed the minimum node-size. Node 6, Node 8 and Node 10 therefore form terminal nodes along with Node 18, Node 19, Node 22, Node 23, Node 28, Node 29, Node 30 and Node 31. Considering Level 3, if Node 6 had not been a terminal node, Node 12 and Node 13 would have sprouted from Node 6. To prevent confusion, we do not name the two nodes that sprout from Node 7 as Node 12 and Node 13. Instead, we allow Node 12 and Node 13 to keep their numbering and consider them as empty or non-existent. As before, it is Node 14 and Node 15 that sprout from Node 7.

The following algorithm summarizes fitting M tree-structures T_j, $j=1,...,M$, using LDsplit with trees.

LDsplit with trees algorithm

1) Sample M times without replacement from L^P, producing P_j, $j=1,...,M$.
2) For $j=1,...,M$:
   a) Fit a stump to all observations $\mathbf{x}_i$, $i=1,...,N$, and label $P_{j,1}$ to produce Level 1 consisting of Node 2 and Node 3.
   b) For $k=2,...,K$:

      Producing level k: Consider each node, n, where $n=$node number 2^{k-1}, ... , node number 2^k-1.

      - When the number of observations in $n>$ minimum node size $\Rightarrow$ fit stump to observations in node n along with their corresponding response entries for label $P_{j,k}$.
      - When the number of observations in $n\leq$ minimum node size $\Rightarrow$ no stump is fit to node n and node n does not split. This node then becomes a terminal node.

4.3 CLASSIFICATION

Suppose a set of unseen observations, $\{(\mathbf{x}_i), i=1,2,...,N_{new}\}$, is available and we aim to use the M tree-structures, T_j, $j=1,...,M$, to find a classification for each of the K labels. Recall that each level of each tree-structure, T_j, is formed by fitting several simple binary classification stumps. Therefore, it should come as no surprise that the classification procedure of a tree-structure, T_j, resembles the classification procedure of a binary classification tree.

In the case of a binary classification tree, the training observations in each terminal node of the classification tree are considered in order to identify the most commonly occurring class in the node. The majority class of each terminal node is assigned to the node. If a set of new observations is dropped into the root node of the tree, all the observations filter down the tree until each reaches a terminal node. Each observation is classified to the majority class that was

identified for the node in which it landed. This works well because in binary classification there are two disjoint classes. For the multi-label classification case, this principle will be adapted so that for a given tree-structure, T_j, the classification of the set of unseen observations, $\{(\mathbf{x}_i), i = 1, 2, ..., N_{new}\}$, is made as follows.

4.3.1 Classification by means of a tree-structure

The first level of T_j consists of Node 2 and Node 3. Consider the training observations present in each of these nodes along with their response entries for label $P_{j,1}$. Determine whether the majority of the training observations in Node 2 have label $P_{j,1}$ present, *i.e.* have a response of 1 for label $P_{j,1}$. If this is true, Node 2 is assigned a 1 for label $P_{j,1}$, else Node 2 is assigned a 0 for label $P_{j,1}$. If the majority of the observations in Node 3 have label $P_{j,1}$ present, Node 3 is assigned a 1 for label $P_{j,1}$. If this is not the case, Node 3 is assigned a 0 for label $P_{j,1}$. The same is done for the second level of T_j. Training observations in Nodes 4, 5, 6 and 7 are considered along with their response entries for label $P_{j,2}$. For each node a 1 or a 0 is assigned depending on whether the majority of training observations for that node have label $P_{j,2}$ present or not. In general, for Level k, where $k = 1, ..., K$, each existing node on Level k is assigned a 1 or a 0 for label $P_{j,k}$ in this manner. A 1 or a 0 can therefore be assigned to every existing/non-empty node of the tree-structure, T_j.

Assume the appropriate assignments have been made to each existing/non-empty node in T_j. The set of unseen observations is now dropped into the root node of T_j and the observations filter down the tree-structure by following the splitting rules of T_j until each observation reaches some terminal node. Since each level of T_j is constructed using a different label, considering only the terminal nodes of T_j in order to make a classification for an unseen observation would be inappropriate. In the case of T_j, a terminal node provides a classification for only one of the K labels. In order to provide a classification regarding each of the K labels, it should be noted which node an unseen observation moves through at each of the levels of T_j before it reaches the relevant terminal node. The node that the observation moves through at Level k provides the

classification of that observation for label $P_{j,k}$, where $k = 1,...,K$. In general, if the unseen observation moves through a node at Level k that was assigned a 1, the observation is classified to have label $P_{j,k}$ present. If the unseen observation moves through a node at Level k that was assigned a 0, we conclude that the observation does not have label $P_{j,k}$ present.

When the terminal node in which the unseen observation ends up in, is at Level K, the relevant observation moves through each of the K levels of the tree-structure, T_j. By doing so, the observation collects a corresponding classification for each of the K labels, $P_{j,k}$, $k = 1,...,K$. It may however happen that the terminal node in which the unseen observation ends up in is not on Level K, but on Level t where $t < K$. In this case the relevant observation only collects a corresponding classification for labels $P_{j,1}, P_{j,2}, ..., P_{j,t}$. We still require a classification regarding labels $P_{j,t+1}, ..., P_{j,K}$, for such an observation. Therefore, if one or more terminal nodes occur at any of the levels preceding Level K, the following additional step is required in order for a tree-structure, T_j, to classify a set of unseen observations.

When a terminal node occurs on a level preceding Level K, say Level t, we consider all the training observations in this terminal node, along with their response entries for the labels $P_{j,t+1}, ..., P_{j,K}$. Take label $P_{j,t+1}$ for example. If the majority of the training observations in the terminal node has label $P_{j,t+1}$ present, unseen observations that end up in this terminal node are assigned a 1 for label $P_{j,t+1}$, else they are assigned a 0 for label $P_{j,t+1}$. The same procedure is used for labels $P_{j,t+2}, ..., P_{j,K}$. We therefore determine the majority class of the training observations in the terminal node for each of the labels $P_{j,t+1}, ..., P_{j,K}$. Unseen observations that end up in this terminal node at Level t collect classifications for labels $P_{j,1}, P_{j,2}, ..., P_{j,t}$ by moving through nodes at Level 1 to Level t. The classification for the remaining labels, $P_{j,t+1}, ..., P_{j,K}$, is found by finding the majority class of labels $P_{j,t+1}, ..., P_{j,K}$ for the training observations in the terminal node.

4.3.2 Obtaining a final classification

For each tree-structure, T_j, $j=1,...,M$, we obtain a classification for each unseen observation in the set $\{(\mathbf{x}_i), i=1,2,...,N_{new}\}$, for each of the K labels. The results can be summarized in M $N_{new} \times K$ matrices. The rows of each matrix correspond to the newly classified observations and the columns correspond to the K labels, $Y_1,...,Y_K$. Since tree-structure T_j is constructed using permutation P_j and label $P_{j,1}$ may not equal label Y_1, we should take care that the classifications are placed in the matrices in such a way that the entry of the s^{th} row and the t^{th} column of the j^{th} matrix corresponds to the classification of observation $\mathbf{x}_s$ for label Y_t using the j^{th} tree-structure.

Each of the observations in $\{(\mathbf{x}_i), i=1,2,...,N_{new}\}$, now have M classifications for each of the K labels. In order to produce a final classification for a given observation and label, a majority vote is taken over the M classifications. Let $\hat{T}_j(\mathbf{x}_i, k)$ be the class prediction of observation $\mathbf{x}_i$ for label Y_k given by the j^{th} tree-structure. Then the final classification for observation $\mathbf{x}_i$ for label Y_k is given by $majority\,vote\left\{\hat{T}_j(\mathbf{x}_i, k)\right\}_{j=1}^{M}$.

4.3.3 Using posterior probabilities

Instead of using class predictions 1 and 0 when finding the classifications of a tree-structure T_j, we might prefer to use posterior probabilities. In this case, for tree-structure T_j, each existing node at Level k is assigned the posterior probability of label $P_{j,k}$, where $k=1,...,K$, by considering the training observations in that node. When a terminal node occurs at a level preceding level K, say Level t, the posterior probabilities of the training observations in the terminal node are found for the labels $P_{j,t+1},...,P_{j,K}$. Unseen observations that end up in a terminal node at Level k collect posterior probabilities for labels $P_{j,1}, P_{j,2},..., P_{j,k}$ by moving through nodes at Level 1 to Level k. If $k<K$, the posterior probabilities of the remaining labels, $P_{j,k+1},...,P_{j,K}$, are given by the posterior probabilities of the training observations in the terminal node. Doing this for all the tree-structures, T_j, $j=1,...,M$, means that an unseen observation now has M posterior probabilities for each of the K labels. In this case let $\hat{T}_j(\mathbf{x}_i, k)$ be the posterior

probability of label Y_k for observation $\mathbf{x}_i$ given by the j^{th} tree-structure. Then in order to find the classification of observation $\mathbf{x}_i$ for label Y_k, we calculate $\frac{1}{M}\sum_{j=1}^{M}\hat{T}_j(\mathbf{x}_i,k)$ and conclude that the label is present for observation $\mathbf{x}_i$ if this quantity exceeds some predefined threshold. This classification procedure that makes use of posterior probabilities is a softer way of establishing classifications of unseen observations, as opposed to the $1/0$ approach.

4.4 ADAPTATION

When the multi-label classification dataset consists of a total of K labels, each tree-structure, T_j, $j=1,...,M$, consists of a total of K levels. Some multi-label classification datasets, however, consist of a large number of labels. This means that each tree-structure would consist of this equally large number of levels. Recall that a stump is fit to a node only if the number of observations in the node exceeds the minimum node-size. If the number of labels of the multi-label dataset is large and the total number of observations in the dataset is not large enough, we might find that the last levels of the tree-structures constructed from the dataset, are all empty. In this case, all the terminal nodes of the tree-structures occur at levels preceding Level K. It may even be that the terminal nodes all occur at levels that are significantly far from Level K, so that many levels of the tree-structures are empty. Simply using the training observations in the terminal nodes to make majority votes for the large number of labels that correspond to the empty levels, seems inappropriate. Instead we may use the following strategy.

4.4.1 LDsplit with trees while $m \leq K$

In order to fit M tree-structures, T_j, $j=1,...,M$, we no longer sample M times without replacement from L^P, and use the M permutations, P_j, $j=1,...,M$, to find the corresponding $K-$level tree-structures. Instead, we introduce a new parameter, m, where $m \leq K$ and we write L^m for the set of all $m-$permutations of the elements in L. In order to fit M tree-structures, T_j, $j=1,...,M$, we sample M times without replacement from L^m, producing M permutations, P_j, $j=1,...,M$. Now these M permutations are used to fit the corresponding M $m-$level tree-structures as before.

When implementing this strategy, the following algorithm may be used to fit M tree-structures, T_j, $j=1,...,M$. Note that if $m=K$ we fit LDsplit with trees as introduced in Section 4.2.

Algorithm for LDsplit with trees when $m \leq K$

1) Specify $m \leq K$.
2) Sample M times without replacement from L^m, producing P_j, $j=1,...,M$.
3) For $j=1,...,M$:
   a) Fit a stump to all observations $\mathbf{x}_i$, $i=1,...,N$, and label $P_{j,1}$ to produce Level 1 consisting of Node 2 and Node 3.
   b) For $k=2,...,m$:

      Producing level k: Consider each node, n, where $n=$node number 2^{k-1}, ... , node number 2^k-1.

      - When the number of observations in $n >$ minimum node size $\Rightarrow$ fit stump to observations in node n along with their corresponding response entries for label $P_{j,k}$.
      - When the number of observations in $n \leq$ minimum node size $\Rightarrow$ no stump is fit to node n and node n does not split. This node then becomes a terminal node.

By specifying $m<K$, each tree-structure consists of a smaller number of levels than the maximum, K. In this case, the data are split into a maximum of $2^{m+1}-1$ nodes per tree-structure. Choosing m small enough would mean that the resulting terminal nodes of tree-structures occur near or at Level m. Since tree-structures are smaller and we no longer use all K labels when constructing each tree-structure, intuitively we might prefer M larger than in the case where $m=K$. The empirical study, given in Section 4.5, investigates this and other issues regarding the values of m and M, as well as the relationship between the two values.

Note however that when $m<K$, M can be chosen too small if K is large. For example, say $m=3$ and $K=28$. In this case we cannot for example set $M=7$, because this would mean that we use a maximum of 21 different labels when constructing the $M=7$ tree-structures, even

though we have a total of $K = 28$ different labels. With M too small, some labels are excluded entirely in the fitting process. In this case we would be unable to provide unseen observations with a classification regarding these excluded labels. However, even if M is specified large enough so that all the labels are used in the fitting process, we might furthermore prefer a label to be used to construct several tree-structures, not simply one. Intuitively it might be sensible to set M significantly larger than its minimum value.

4.4.2 Classification

Once M tree-structures are formed applying the adapted algorithm, these tree-structures can be used to classify a set of unseen observations, $\{(\mathbf{x}_i),\ i = 1, 2, ..., N_{new}\}$. We proceed as in Section 4.3. For each tree-structure, T_j, each existing node at Level r is assigned the majority class or posterior probability of label $P_{j,r}$, where $r = 1, ..., m$, considering the training observations in the node. Unseen observations that end up in a terminal node at Level r, collect classifications/posterior probabilities for labels $P_{j,1}, P_{j,2}, ..., P_{j,r}$ by moving through nodes at Level 1 to Level r. If $r < m$, the classifications/posterior probabilities of the remaining labels, $P_{j,r+1}, ..., P_{j,m}$, are given by the majority votes/posterior probabilities of the training observations in the terminal node. However, if this is done for tree-structures T_j, $j = 1, ..., M$, each unseen observation no longer necessarily has M classifications/posterior probabilities for each of the K labels, since each tree-structure does not necessarily contain a classification for all K labels. Therefore, when using posterior probabilities, denote by $R_{k,i}$ the collection of tree-structures that produce a posterior probability for label Y_k for observation $\mathbf{x}_i$. Also let $\hat{T}_j(\mathbf{x}_i, k)$ be the posterior probability of label Y_k for observation $\mathbf{x}_i$ given by the j^{th} tree-structure. Then in order to find the classification of observation $\mathbf{x}_i$ for label Y_k, we calculate $\frac{1}{|R_{k,i}|} \sum_{j \in R_{k,i}} \hat{T}_j(\mathbf{x}_i, k)$ and conclude that the label is present for observation $\mathbf{x}_i$ if this quantity exceeds some predefined threshold. Here $|R_{k,i}|$ denotes the number of tree-structures that produce a posterior probability for label Y_k for observation $\mathbf{x}_i$.

4.5 EXPERIMENTAL EVALUATION OF LDSPLIT WITH TREES

In this section LDsplit with trees with $m \leq K$ is fit to three of the benchmark datasets given in Table 2.1, namely *Emotions*, *Flags* and *Yeast*. From Table 2.1 we see that these datasets vary in size in terms of numbers of observations, predictors and labels, as well as label cardinality. The datasets come from three domains: music, image and biology. The benchmark datasets are obtained using the R package mldr.datasets. This package is discussed in Chapter 5. The datasets are divided into train and test parts, with the training sets comprising approximately two thirds of the complete datasets and the remaining thirds acting as test datasets. Hamming-loss, precision, recall and the F-score are used as evaluation measures of predictive performance. The performance of LDsplit with trees on these benchmark datasets is compared to the performance of six existing multi-label learning methods, namely binary relevance, label powerset, classifier chains, ensembles of classifier chains, RAkEL$_d$ and RAkEL$_o$.

In this empirical study we are particularly interested in investigating the level of performance of LDsplit with trees compared to that of existing methods. Does LDsplit with trees produce promising results, possibly allowing it to be a useful and competitive learning method in future analyses? We are furthermore interested in examining the role of the parameters m and M, while the minimum node size is fixed throughout the empirical study as 5. When m is small, each tree-structure consists of a few levels only. Should M be chosen particularly large in cases like this to compensate for the smaller tree-structure size? Everything else kept fixed, computational cost increases when the value of M increases. Does an increase in M produce a significant increase in performance? Can M be chosen too large, allowing us to overfit? For larger values of m, more labels are used to construct each tree-structure, arguably fitting larger tree-structures taking more label correlation into account. However, more levels per tree-structure may lead to several terminal nodes occurring on levels preceding Level m, as a result of the specified minimum node size. We would therefore also like to investigate whether overfitting is a concern for large values of m.

Our empirical study is carried out in R. Functions in the R package utiml are used to fit the six existing multi-label learning methods and to classify test observations using these fitted functions. The utiml package is discussed in more detail in Chapter 5. The base classifier used for all six of the existing multi-label learning methods is decision trees. Package rpart is loaded in R along with package utiml in order to do so. A decision tree is the chosen base classifier to allow for a fair comparison to LDsplit with trees. Since LDsplit with trees is a new multi-label ensemble method,

no R functions exist to fit LDsplit with trees or to classify unseen observations in the specified way. Specific R functions are written for this purpose. Figure 4.4 provides a summary of all the functions written to fit LDsplit with trees and to classify unseen observations. The full R functions are provided in Appendix A.1. Furthermore, posterior probabilities are used in the classification process of LDsplit with trees. When the posterior probability of a label is larger than or equal to 0.5, the label is regarded as being present. This threshold is applied to all the learning methods and all the labels for each of the three datasets. In all cases the evaluation measures are calculated using the utiml package function *multilabel_evaluate()*.

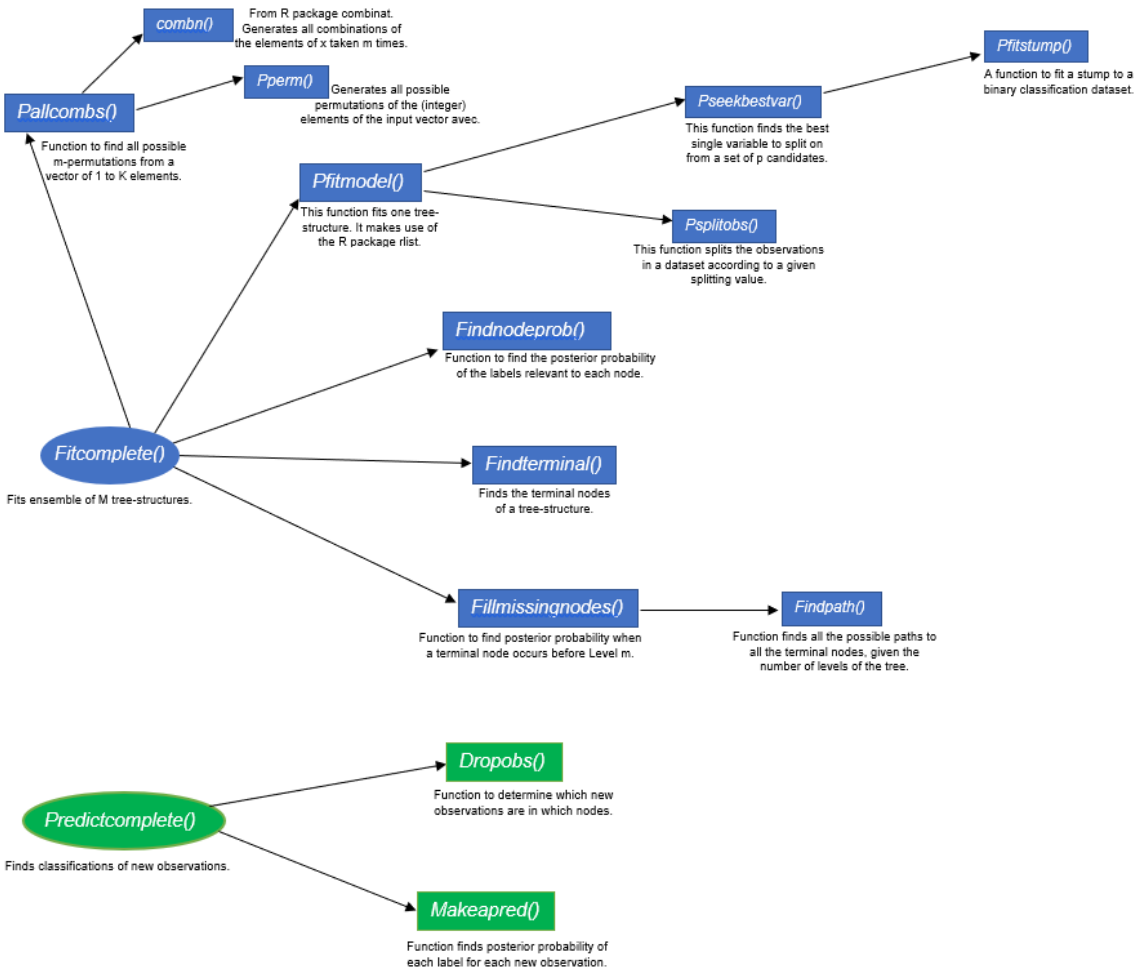


Figure 4.4: Summary of functions used for LDsplit with trees

4.5.1 Experimental design

The *Emotions* dataset consists of $N = 593$ observations, $p = 72$ predictors and $K = 6$ labels. We use 400 observations as training data and 193 observations as test data. The *Flags* dataset consists of $N = 194$ observations, $p = 19$ predictors and $K = 7$ labels. The *Flags* dataset is the smallest dataset used in our empirical study. We use 130 observations as training data and 94 observations as test data. Finally, the *Yeast* dataset consists of $N = 2417$ observations, $p = 103$ predictors and $K = 14$ labels. Consequently, the *Yeast* dataset is the largest dataset used in our empirical study. We use 1611 observations as training data and 806 observations as test data.

The six existing multi-label learning methods are fit to the respective training datasets and the resulting functions are used to classify the test data observations.

When fitting classifier chains on the *Emotions* data the order of the labels is "amazed-surprised", followed by "happy-pleased", "relaxing-calm", "quiet-still", "sad-lonely" and lastly "angry-aggressive". For the *Flags* data, the order of the labels is "red", followed by "green", "blue", "yellow", "white", "black" and lastly "orange". We use the 1-14 ordering of the classes, as given, for the *Yeast* data.

For all three benchmark datasets the ensemble of classifier chains model is based on 10 random orderings of the labels and the percentage of training observations used for each classifier is 75%. The percentage of predictors used for each classifier is 100% to allow for a fair comparison to LDsplit with trees.

For the *Emotions* data, RAkEL$_d$ is fit with $k = 3$ and RAkEL$_o$ with $k = 3$ and $m = 2K$. For both the *Flags* data and the *Yeast* data we fit RAkEL$_d$ with $k = 4$ and RAkEL$_o$ with $k = 4$ and $m = 2K$. These parameters are chosen based on the guidelines given in Tsoumakas *et al*. (2011).

In order to examine the parameters m and M of LDsplit with trees, we proceed as follows. For the *Emotions* data we vary m between $m = 2$ and $m = 6$. By doing so we fit tree-structures having a few levels only and we are also able to examine the consequences of allowing tree-structures to reach their maximum number of levels, in this case, $m = K = 6$. For the *Flags* data we vary m between $m = 2$ and $m = 7$. We therefore again fit tree-structures with a few levels and allow tree-structures to reach their maximum number of levels, in this case, $m = K = 7$, as well. We investigate what the effect of a large $M-$value is for each of the specified $m-$values, with the

results summarised in Table 4.2 for the *Emotions* data and in Table 4.5 for the *Flags* data. To see if a large value of M in fact leads to a significant increase in performance, we also allow the value of M to be smaller, varying M between K and $2K$. This is summarised in Table 4.3 for the *Emotions* data and in Table 4.6 for the *Flags* data.

For the *Yeast* data we vary m between $m=2$ and $m=7$ while $K \leq M \leq 2K$. These results are summarised in Table 4.8. Since the *Yeast* data is the largest dataset used in our empirical study and consists of the largest number of labels, we do not set M larger than $2K=28$. We therefore summarize the results of all the LDsplit models of the *Yeast* data in one table.

For each of the three benchmark datasets, we also construct an additional table that allows for easy comparison between the best performing LDsplit models and the existing multi-label learning methods. For each benchmark dataset we identify the best performing LDsplit models for each of the $m-$values regarding the varying $M-$values. In the respective tables the results of these models are summarised along with those for the existing multi-label learning methods. For each table we highlight, in yellow, the lowest Hamming loss and highest F-score produced by an LDsplit model. We also highlight, in green, the lowest Hamming loss and highest F-score produced when only comparing the six existing multi-label learning methods. For the *Emotions* data this is given in Table 4.4, for the *Flags* data this is given in Table 4.7 and for the *Yeast* data this is given in Table 4.9.

The following three sections discuss the results regarding each of the three benchmark datasets.

4.5.2 Emotions

The *Emotions* data produce the following results.

Table 4.2: Emotions data, LDsplit models with large M

Method	Hamming loss	Precision	Recall	F-score
LDsplit with trees $m=2$, $M=14$	0.2556131	0.6027634	0.4887737	0.539816553
LDsplit with trees $m=2$, $M=20$	0.2590674	0.5863558	0.5008636	0.540248411
LDsplit with trees $m=2$, $M=30$	0.2564767	0.5958549	0.4991364	0.543224169
LDsplit with trees $m=3$, $M=20$	0.2461140	0.6234888	0.5164076	0.564918627
LDsplit with trees $m=3$, $M=40$	0.2512953	0.5975820	0.4974093	0.542913618
LDsplit with trees $m=3$, $M=80$	0.2573402	0.5872193	0.4905009	0.534520175
LDsplit with trees $m=3$, $M=120$	0.2564767	0.5924007	0.4827288	0.53197104
LDsplit with trees $m=4$, $M=60$	0.2607945	0.5898100	0.4611399	0.517598269
LDsplit with trees $m=4$, $M=120$	0.2538860	0.5949914	0.4844560	0.534064288
LDsplit with trees $m=4$, $M=240$	0.2538860	0.6001727	0.4879102	0.538250132
LDsplit with trees $m=4$, $M=360$	0.2530225	0.6027634	0.4905009	0.540868279
LDsplit with trees $m=5$, $M=120$	0.2607945	0.5872193	0.4568221	0.513877618
LDsplit with trees $m=5$, $M=240$	0.2573402	0.5949914	0.4637306	0.521224115
LDsplit with trees $m=5$, $M=360$	0.2530225	0.6079447	0.4706390	0.530552215
LDsplit with trees $m=5$, $M=400$	0.2556131	0.5975820	0.4671848	0.524398821
LDsplit with trees $m=5$, $M=700$	0.2530225	0.6027634	0.4715026	0.529113851
LDsplit with trees $m=6$, $M=120$	0.2633851	0.6001727	0.4335060	0.503402975
LDsplit with trees $m=6$, $M=200$	0.2625216	0.5924007	0.4369603	0.50294423
LDsplit with trees $m=6$, $M=360$	0.2677029	0.5872193	0.4291883	0.495918474

Table 4.3: Emotions data, LDsplit models with $K \leq M \leq 2K$

Method	Hamming loss	Precision	Recall	F-score
LDsplit with trees $m=2$, $M=6$	0.2512953	0.5975820	0.5086356	0.549532893
LDsplit with trees $m=2$, $M=9$	0.2616580	0.5777202	0.4965458	0.534066123
LDsplit with trees $m=2$, $M=12$	0.2573402	0.5854922	0.4930915	0.535333933
LDsplit with trees $m=3$, $M=6$	0.2651123	0.5751295	0.4879102	0.527941806
LDsplit with trees $m=3$, $M=9$	0.2625216	0.5811744	0.4913644	0.532509239
LDsplit with trees $m=3$, $M=12$	0.2625216	0.5690846	0.4887737	0.525880613
LDsplit with trees $m=4$, $M=6$	0.2711572	0.5742660	0.4585492	0.509925135
LDsplit with trees $m=4$, $M=9$	0.2521589	0.5932642	0.5077720	0.547198992
LDsplit with trees $m=4$, $M=12$	0.2530225	0.5993092	0.4775475	0.531544467
LDsplit with trees $m=5$, $M=6$	0.2564767	0.6018998	0.4861831	0.537888263
LDsplit with trees $m=5$, $M=9$	0.2556131	0.6044905	0.4853195	0.53838931
LDsplit with trees $m=5$, $M=12$	0.2659758	0.5958549	0.4386874	0.505332719
LDsplit with trees $m=6$, $M=6$	0.2659758	0.6001727	0.4801382	0.533486869
LDsplit with trees $m=6$, $M=9$	0.29965458	0.55872193	0.36096718	0.438583598
LDsplit with trees $m=6$, $M=12$	0.2633851	0.5820380	0.4481865	0.506416949

Examining Table 4.2 and Table 4.3 leads to the following conclusions. For $m=3$, larger $M-$values, such as those given in Table 4.2, produce lower Hamming loss values and larger F-scores than when $m=3$ in Table 4.3. The difference in F-scores may not be significant, but Hamming loss values in Table 4.2 are lower than those in Table 4.3 by a value of approximately 0.1. For $m=4$, however, allowing M to become larger than 6 does not lead to much improvement in performance. In Table 4.3, for $m=4$, $M-$values that are larger than 6 produce very similar results to those given in Table 4.2, even though the $M-$values are much larger in Table 4.2. For $m=4$, it therefore appears that performance stabilizes after $M>6$, with no significant improvement in performance for larger M. A similar conclusion can be made for $m=5$ and $m=6$. Furthermore, it appears that allowing m to be its maximum value, $m=6$, produces weak results compared to those cases where $m<K$. In fact, $m=6$ produces the weakest results of all

$m-$values. In both Table 4.2 and Table 4.3 the Hamming loss values are the largest and the F-scores are the lowest for $m=6$.

Table 4.4: Emotions data model summary

Method	Hamming loss	Precision	Recall	F-score
LDsplit with trees $m=2$, $M=6$	0.2512953	0.5975820	0.5086356	0.549532893
LDsplit with trees $m=3$, $M=20$	0.2461140	0.6234888	0.5164076	0.564918627
LDsplit with trees $m=4$, $M=9$	0.2521589	0.5932642	0.5077720	0.547198992
LDsplit with trees $m=5$, $M=9$	0.2556131	0.6044905	0.4853195	0.53838931
LDsplit with trees $m=6$, $M=200$	0.2625216	0.5924007	0.4369603	0.50294423
Binary relevance	0.2720207	0.5660622	0.5587219	0.562368099
Label powerset	0.3031088	0.4689119	0.3963731	0.429601954
Classifier chains	0.2849741	0.5457686	0.5656304	0.555522025
Ensembles of classifier chains (10)	0.2236615	0.6278066	0.6571675	0.64215161
RAkEL$_d$ $k=3$	0.2538860	0.5639033	0.5932642	0.578211262
RAkEL$_o$ $k=3$, $m=12$	0.2305699	0.6373057	0.6295337	0.63339586

Table 4.4 reveals that amongst the LDsplit with trees models fit in this analysis, the best performing model is the model having $m=3$ with $M=20$. It appears that models having lower $M-$values perform better in general, since $M=20$ is also not a very large $M-$value. Considering all LDsplit with trees models having $m=6$, the model with $M=200$ is regarded as the best performing model. However, this model does not significantly outperform other models with $m=6$ and, compared to models having $m<K$, also has the weakest performance.

The best performing LDsplit with trees model, having parameters $m=3$ and $M=20$, seems to outperform all the problem transformation methods fit in the analysis, namely binary relevance, label powerset and classifier chains. The model has a lower Hamming loss and higher F-score than all three of the problem transformation models. Compared to other ensemble methods, the best performing LDsplit with trees model has a lower Hamming loss than RAkEL$_d$; however RAkEL$_d$ has a higher F-score. RAkEL$_o$ and the ensemble of classifier chains model outperform

the best performing LDsplit with trees model. In this analysis it appears that the ensemble of classifier chains model performs best overall for the *Emotions* data.

4.5.3 Flags

The *Flags* data produce the following results.

Table 4.5: Flags data, LDsplit models with large M

Method	Hamming loss	Precision	Recall	F-score
LDsplit with trees $m=2$, $M=20$	0.2254464	0.7666667	0.7568824	0.761743132
LDsplit with trees $m=2$, $M=28$	0.2343750	0.7572917	0.7568824	0.757086995
LDsplit with trees $m=2$, $M=42$	0.2254464	0.7666667	0.7568824	0.761743132
LDsplit with trees $m=3$, $M=70$	0.2343750	0.7572917	0.7568824	0.757086995
LDsplit with trees $m=3$, $M=140$	0.2299107	0.7598958	0.7672991	0.763579506
LDsplit with trees $m=3$, $M=210$	0.2321429	0.7598958	0.7646949	0.762287797
LDsplit with trees $m=4$, $M=180$	0.2700893	0.7033854	0.8119420	0.753775255
LDsplit with trees $m=4$, $M=280$	0.2656250	0.7085938	0.7859003	0.745247613
LDsplit with trees $m=4$, $M=560$	0.2477679	0.7130208	0.8068824	0.757053389
LDsplit with trees $m=4$, $M=840$	0.2544643	0.7177083	0.7911086	0.752623076
LDsplit with trees $m=5$, $M=180$	0.2879464	0.6729167	0.8455357	0.74941446
LDsplit with trees $m=5$, $M=280$	0.3325893	0.6244792	0.8845982	0.732120402
LDsplit with trees $m=5$, $M=560$	0.3013393	0.6523438	0.8507440	0.738449974
LDsplit with trees $m=5$, $M=840$	0.2946429	0.6627604	0.8455357	0.743073696
LDsplit with trees $m=6$, $M=180$	0.3437500	0.6166667	0.8845982	0.726723515
LDsplit with trees $m=6$, $M=280$	0.3325893	0.6255208	0.8845982	0.732835722
LDsplit with trees $m=6$, $M=560$	0.3325893	0.6244792	0.8845982	0.732120402
LDsplit with trees $m=7$, $M=180$	0.3415179	0.6192708	0.8845982	0.728528662
LDsplit with trees $m=7$, $M=280$	0.3415179	0.6192708	0.8845982	0.728528662
LDsplit with trees $m=7$, $M=560$	0.3415179	0.6192708	0.8845982	0.728528662

Table 4.6: Flags data, LDsplit models with $K \leq M \leq 2K$

Method	Hamming loss	Precision	Recall	F-score
LDsplit with trees $m=2$, $M=7$	0.2321429	0.7598958	0.7646949	0.762287797
LDsplit with trees $m=2$, $M=10$	0.2366071	0.7653646	0.7442336	0.754651207
LDsplit with trees $m=2$, $M=14$	0.2388393	0.7653646	0.7416295	0.753310137
LDsplit with trees $m=3$, $M=7$	0.2410714	0.7549479	0.7360491	0.745378726
LDsplit with trees $m=3$, $M=10$	0.2299107	0.7653646	0.7516741	0.758457575
LDsplit with trees $m=3$, $M=14$	0.2276786	0.7661458	0.7542783	0.760165735
LDsplit with trees $m=4$, $M=7$	0.2611607	0.7015625	0.8273065	0.759263503
LDsplit with trees $m=4$, $M=10$	0.2812500	0.6710938	0.8355283	0.744337763
LDsplit with trees $m=4$, $M=14$	0.2633929	0.6979167	0.8113095	0.750353325
LDsplit with trees $m=5$, $M=7$	0.3325893	0.6171875	0.8972470	0.731321999
LDsplit with trees $m=5$, $M=10$	0.2991071	0.6697917	0.8205357	0.737539955
LDsplit with trees $m=5$, $M=14$	0.3080357	0.6455729	0.8507440	0.734092185
LDsplit with trees $m=6$, $M=7$	0.2991071	0.6497396	0.8481399	0.735800282
LDsplit with trees $m=6$, $M=10$	0.2946429	0.6476563	0.6476563	0.6476563
LDsplit with trees $m=6$, $M=14$	0.3013393	0.6471354	0.8533482	0.736071795
LDsplit with trees $m=7$, $M=7$	0.3370536	0.6281250	0.8775670	0.732183968
LDsplit with trees $m=7$, $M=10$	0.3325893	0.6171875	0.8972470	0.731321999
LDsplit with trees $m=7$, $M=14$	0.3258929	0.6270833	0.8819940	0.733009115

For $m=2$ the results summarized in Table 4.5 and Table 4.6 do not differ much, but it appears as if larger $M-$values lead to slightly better performance. However, performance seems to stabilize once $M \geq 20$, so that increasing M beyond $M=20$ seems unnecessary when $m=2$. For $m=3$, performance seems to stabilize once $M>7$. If $m \geq 4$, in both Table 4.5 and Table 4.6, Hamming loss appears to increase quite rapidly and the F-score values seem to decline. It appears that setting $m \leq 3$ leads to the best performing LDsplit models in this analysis.

Table 4.7: Flags data model summary

Method	Hamming loss	Precision	Recall	F-score
LDsplit with trees $m=2$, $M=7$	0.2321429	0.7598958	0.7646949	0.762287797
LDsplit with trees $m=2$, $M=20$	0.2254464	0.7666667	0.7568824	0.761743132
LDsplit with trees $m=3$, $M=14$	0.2276786	0.7661458	0.7542783	0.760165735
LDsplit with trees $m=3$, $M=140$	0.2299107	0.7598958	0.7672991	0.763579506
LDsplit with trees $m=4$, $M=7$	0.2611607	0.7015625	0.8273065	0.759263503
LDsplit with trees $m=4$, $M=560$	0.2477679	0.7130208	0.8068824	0.757053389
LDsplit with trees $m=5$, $M=10$	0.2991071	0.6697917	0.8205357	0.737539955
LDsplit with trees $m=5$, $M=180$	0.2879464	0.6729167	0.8455357	0.74941446
LDsplit with trees $m=6$, $M=7$	0.2991071	0.6497396	0.8481399	0.735800282
LDsplit with trees $m=6$, $M=280$	0.3325893	0.6255208	0.8845982	0.732835722
LDsplit with trees $m=7$, $M=10$	0.3325893	0.6171875	0.8972470	0.731321999
LDsplit with trees $m=7$, $M=180$	0.3415179	0.6192708	0.8845982	0.728528662
Binary relevance	0.2656250	0.7132812	0.7335938	0.723294916
Label powerset	0.3616071	0.7734375	0.3409970	0.473316049
Classifier chains (1-7 ordering)	0.2633929	0.7151786	0.7495908	0.731980473
Ensembles of classifier chains (10)	0.2656250	0.6967262	0.7761533	0.734298141
RAkEL$_d$ $k=4$	0.3482143	0.7317708	0.4524182	0.559144576
RAkEL$_o$ $k=4$, $m=14$	0.2611607	0.7111979	0.7409970	0.725791711

From Table 4.7 we see that the LDsplit with trees model that results in the lowest Hamming loss is the model having $m=2$ and $M=20$. The highest F-score is given by the LDsplit model having $m=3$ and $M=140$. This F-score is not much larger than that found for the model with $m=2$ and $M=20$. Since the LDsplit with trees model having $m=2$ and $M=20$ requires less computation, it is regarded as the overall best performing LDsplit model of this analysis. Comparing this model to the six existing multi-label learning methods, it is clear that LDsplit with trees outperforms all six, resulting in a lower Hamming loss and higher F-score than all six of the other existing multi-label learning methods.

4.5.4 Yeast

The *Yeast* data results are discussed in this section.

Table 4.8: Yeast data, LDsplit models

Method	Hamming loss	Precision	Recall	F-score
LDsplit with trees $m=2$, $M=14$	0.22057781	0.71360924	0.46190490	0.560809255
LDsplit with trees $m=2$, $M=17$	0.22172988	0.71257533	0.46060217	0.559529557
LDsplit with trees $m=2$, $M=21$	0.22102091	0.71534326	0.45793467	0.558402185
LDsplit with trees $m=2$, $M=28$	0.22190713	0.71125192	0.45841766	0.557508627
LDsplit with trees $m=3$, $M=14$	0.22235023	0.71203474	0.46233717	0.560640328
LDsplit with trees $m=3$, $M=17$	0.21969160	0.72985053	0.44765409	0.554937228
LDsplit with trees $m=3$, $M=21$	0.22048919	0.71710091	0.45351494	0.555632279
LDsplit with trees $m=3$, $M=28$	0.22243885	0.71107172	0.45801542	0.557155752
LDsplit with trees $m=4$, $M=14$	0.260900390	0.686455749	0.283763235	0.40154008
LDsplit with trees $m=4$, $M=17$	0.22332506	0.71993383	0.43786939	0.544543289
LDsplit with trees $m=4$, $M=21$	0.22332506	0.71497991	0.44882443	0.551468042
LDsplit with trees $m=4$, $M=28$	0.24645516	0.61021801	0.48287067	0.539126211
LDsplit with trees $m=7$, $M=14$	0.22642680	0.73792390	0.37570709	0.497908631
LDsplit with trees $m=7$, $M=17$	0.25203828	0.72950786	0.26533986	0.389139985
LDsplit with trees $m=7$, $M=21$	0.25168380	0.62719189	0.42651551	0.507744501
LDsplit with trees $m=7$, $M=28$	0.22952854	0.73004549	0.38583997	0.504855991

From Table 4.8, it appears that once $m \geq 4$, Hamming loss values increase and the F-score values decrease. Setting $m \leq 3$ leads to the best performing LDsplit models in this analysis. For each $m-$value, increasing M does not allow for significant differences in performance and the best performing $M-$value is often one of the smaller $M-$values. It should be noted that when m is small, suppose $m=2$, we cannot for example set $M=6$ for the *Yeast* data, because this would mean that we consider a maximum of 12 different labels when constructing the $M=6$ tree-structures, which is insufficient given that we have 14 different labels in the dataset.

Table 4.9: Yeast data model summary

Method	Hamming loss	Precision	Recall	F-score
LDsplit with trees $m=2$, $M=14$	0.22057781	0.71360924	0.46190490	0.560809255
LDsplit with trees $m=3$, $M=17$	0.21969160	0.72985053	0.44765409	0.554937228
LDsplit with trees $m=4$, $M=21$	0.22332506	0.71497991	0.44882443	0.551468042
LDsplit with trees $m=7$, $M=14$	0.22642680	0.73792390	0.37570709	0.497908631
Binary relevance	0.2242113	0.6609536	0.5542033	0.602889497
Label powerset	0.31407302	0.35421836	0.14246032	0.203198015
Classifier chains (1-14 ordering)	0.2266927	0.6413949	0.5542186	0.594628588
Ensembles of classifier chains (10)	0.2052464	0.6848591	0.6087062	0.644541068
RAkEL$_d$ $k=4$	0.27437079	0.60413565	0.21940756	0.321906434
RAkEL$_o$ $k=4$, $m=28$	0.2126019	0.6921216	0.5513305	0.613755444

Examining Table 4.9, we identify the LDsplit with trees model having $m=2$ and $M=14$ as the best LDsplit with trees model in this analysis. This model has low computational cost and produces the highest F-score and second lowest Hamming loss when only comparing LDsplit models. Comparing this model to the six existing multi-label learning methods, we see that LDsplit with trees outperforms the label powerset method and RAkEL$_d$. Both the binary relevance model and classifier chains model produce a higher F-score than the best performing LDsplit with trees model. However, Table 4.9 also shows how unbalanced recall and precision are for the LDsplit models. Here recall is generally quite small. We may be able to address this by adjusting the threshold value used for classification, possibly making the threshold value smaller for some labels. RAkEL$_o$ and the ensemble of classifier chains model outperform the LDsplit with trees model, both producing a lower Hamming loss and a higher F-score. We conclude that the ensemble of classifier chains model is the overall best performing model in this analysis.

4.5.5 Conclusions

With regard to the parameters m and M of LDsplit with trees, our empirical study reveals the following. It appears that smaller values of m often perform better than larger $m-$values. The results from this study suggest setting $m \leq 3$. It appears that overfitting becomes a concern once $m \geq 4$. Thus setting $m=K$ (maximum number of levels) can lead to overfitting. However, we should also note that the benchmark datasets used in this empirical study do not contain many

observations, so that a different conclusion may be reached for datasets with large N. Furthermore, the results of this study suggest that beyond a certain value of M, performance starts to stabilize so that increasing the value of M still further does not allow for a significant increase in performance. Results of the study indicate that setting $M \leq 20$ is sufficient. However, overfitting does not seem to be a concern for large values of M. In other words, if computational cost is not a concern, M can be made larger.

LDsplit with trees performed promisingly compared to the other multi-label learning methods fit in this empirical study, even outperforming all the other methods for one of the datasets. It is difficult to conclude exactly why LDsplit outperforms all the other ensemble methods on the *Flags* dataset only. A possible influence may be the fact that the *Flags* dataset has the smallest number of predictors, compared to the other two datasets. If some of the predictors are very informative, it might be that the LDsplit classifier is able to exploit this characteristic of the dataset. We can investigate this in future research. Thus, with further adaptations to the learning method, LDsplit with trees may well become a competitive multi-label learning method.

4.6 CONCLUSION AND COMPARISON TO EXISTING METHODS

LDsplit with trees uses a different label when constructing each level of a tree-structure. For tree-structure T_j, the nodes formed at Level k are however not only dependent on label $P_{j,k}$, but they are in fact also dependent on all the previous splits using labels $P_{j,1},...,P_{j,k-1}$. Since each level of a tree-structure is dependent on the results of the previous levels, possible label correlations are implicitly incorporated into the model. However, a tree-structure is very dependent on the order of the labels used to construct it. We overcome this concern by using M different orderings of the labels and construct M corresponding tree-structures, thus fitting an ensemble of tree-structures. Since each tree-structure produces a multi-label classification, LDsplit with trees is a multi-label ensemble method.

Comparing the LDsplit with trees model to other existing multi-label ensemble methods studied in this thesis, produces the following conclusions.

4.6.1 Ensembles of classifier chains

When fitting an ensemble of classifier chains, we consider several random orderings of the K labels, $Y_1,...,Y_K$, and fit a classifier chains model to each, using bootstrap samples of the training data. In other words, for a given bootstrap sample and label ordering, we first consider all the

bootstrap observations along with the first label and fit the base classifier to the binary classification data. This produces a binary classifier. The input space is now extended with the first label. We then again consider all the bootstrap observations in this extended input space along with the second label and fit the base classifier to the binary classification data. This is different from LDsplit with trees. In LDsplit with trees we also consider different label orderings to fit an ensemble of multi-label classifiers. However, for a given label ordering, we consider the first label and optimally split the training data into two separate nodes. Each of these nodes are considered separately along with the second label in order to optimally split each of them in turn. Therefore, in both ensembles of classifier chains and LDsplit with trees, label ordering helps incorporate correlation into the models. However, LDsplit with trees aims to optimally split the training data using each label in turn, whereas a classifier chains model uses all the observations when fitting each binary classifier while the input space is extended.

4.6.2 Random forests of predictive clustering trees

Both random forests of predictive clustering trees and LDsplit with trees have a base learner that partitions the data into hierarchical clusters. For random forests of predictive clustering trees, this base learner is a MODT with variance and prototype functions defined to handle multi-label data, as discussed in Section 3.6. For LDsplit with trees, the base learner is the tree-structure described in this chapter. The two approaches clearly differ.

The base learner of LDsplit with trees, splits the data by considering one label per level, in effect forming appropriate binary classification datasets to perform the splitting. A MODT, on the other hand, does not form binary classification datasets or consider the labels in turn in order to perform the clustering of the data. Instead it considers the full set of labels when finding homogenous clusters.

Due to the way the base learner of LDsplit with trees is defined, we form an ensemble of models by considering different permutations of the labels and by fitting a tree-structure for each label ordering. We do not use permutations of the labels when fitting a random forest of predictive clustering trees; instead, the ensemble of models is formed by applying the random forest ensemble strategy that uses bootstrap samples of the data.

4.6.3 Random k-labelsets

RAkEL partitions L randomly into l labelsets, R_j, $j = 1,...,l$, with the labelsets being disjoint or not, depending on the version of RAkEL we wish to apply. The training-set consists of all N

observations annotated with their corresponding labels in R_j, $j = 1,...,l$, so that the multi-label data are effectively divided into l smaller multi-label datasets. An appropriate multi-label learning method can then be applied to the l smaller multi-label datasets, thereby becoming the base learner. This produces l multi-label classifiers. In this way RAkEL creates an ensemble of multi-label classifiers by manipulating the label space using randomization. Even though RAkEL is defined by using the label powerset method as base learner, it is possible to extend the concept of RAkEL by using a different base learner and therefore making the procedure more general. The selected base learner of this general version of RAkEL should, however, strongly depend on the specific set of labels used to annotate each example. This is exactly the case for the label powerset procedure. It would for example be of no benefit to use binary relevance as a base learner.

An LDsplit with trees model with $m < K$ resembles RAkEL$_o$ when we specify the base classifier of RAkEL$_o$ as follows. Consider M labelsets, R_j, $j = 1,...,M$, each of size $k = m$. Fit a tree-structure, T_j, as described in this chapter, to each labelset. The order of the labels used to construct a tree-structure is simply given by the order used when defining the relevant labelset. This produces a model that is similar to that described in this chapter. However, in RAkEL$_o$, in order to produce M labelsets, we sample M times without replacement from L^k, where L^k is defined as the set of all distinct $k-$labelsets of L. Therefore for a labelset of $k = m$ labels, we are only considering one ordering of the labels in that set. In LDsplit with trees, the same set of m labels may be used to construct two or more tree-structures by using different orderings of the labels in the set.

LDsplit with trees, as introduced in this chapter, is a multi-label ensemble method that splits multi-label training data using trees as a splitting tool. One of its most important properties compared to existing multi-label ensemble methods, is that it aims to do the splitting of the data in a label-dependent way, whilst incorporating label correlation. Several permutations of the labels are used to accomplish this and a multi-label classification model is formed by combining the collection of simple base models. The method shows potential when compared to other multi-label learning methods, as was seen in our experimental evaluation. In Chapter 6 we refer to future research ideas regarding the LDsplit framework given in this chapter. The following chapter, however, focuses on analysis of multi-label text data. Reference to useful R packages for such analyses are also made. The chapter concludes with a practical data analysis.

CHAPTER 5: PRACTICAL DATA ANALYSIS

5.1 ASPECTS OF ANALYSING TEXT DATA

Natural language processing (NLP) is an area of research that aims to interpret text or speech using computer-based methods. This presents many challenges. In text documents we can for example have a mixture of lower- and upper-case letters, as well as shorter and longer sentences. Some words have synonyms, alternative spellings, different tenses or more than one definition. A new word, only recently added to dictionaries, might be encountered in a text document. Other words might be field specific so that they are not included in dictionaries. A word might be used in a figurative manner, where in other cases it might be meant literally. Sentences can also be ambiguous or sarcastic. These are only a few issues that make it difficult for computer software to interpret text documents. Kaggle presents examples of analyses requiring NLP. One dataset challenges data scientists to build a model that can generate the answers to questions, given the Wikipedia article text the questions were originally generated from (Kaggle: Question-Answer dataset, 2017). Building a model that automatically answers questions is no easy task in NLP.

Taking it one step further, an analysis might not only require interpretation of sentences, but the aim might also be to understand what the implicit meanings of the sentences are. What did the writer feel? A movie review dataset found on Kaggle, for example, consists of phrases found on a movie review website along with corresponding sentiment classes. Classes include: negative, somewhat negative, neutral, somewhat positive and positive. The goal is to build a classifier that can interpret a phrase and determine the sentiment class that the phrase belongs to. Results from such an analysis can be used to determine the general feeling towards a movie. An analysis of this type is referred to as a sentiment analysis (Kaggle: Sentiment analysis on movie reviews, 2015).

As mentioned, multi-label datasets, where the observations are in the form of text, are also common. A data scientist might for example aim to attach text categories to text documents.

No matter what the goal of the text analysis is, in most cases the pre-processing and cleaning of the data are especially important. Text data can be very noisy. It is important to organize the data so that it is easier to work with and so that it becomes possible to extract the important information from the data. When cleaning, pre-processing and organization of the data are not a high priority of the analysis, even the best learning methods will not produce good predictive performance.

5.1.1 Some approaches to organizing text data

Suppose the text data are represented in terms of several text documents. One approach to organizing the text data is **semantic parsing**. Here the word “semantic” refers to meaning in language, whereas “parsing” refers to breaking up a sentence into its component parts and describing their syntactic roles. The process of semantic parsing thus aims to map text into a formal representation of its meaning. Here, word type and order of the text may be important.

Different types of semantic parsing exist. A shallow approach may be to continuously break up text forming a tree-like structure. The top level of a tree may be one full sentence which is broken up into a noun-phrase and a verb-phrase. The verb-phrase may then for example be broken up as a verb, article, adverb, and so on, forming a tree-like structure. In this case a word has multiple tags corresponding to the different levels of the tree. A single word may for example be part of a sentence, be part of a verb-phrase and lastly be tagged as a verb. A semantic parsing approach like this can therefore result in many features (DataCamp: Text mining, 2018). Semantic parsing may be a useful way to organize the Kaggle-dataset that challenges data scientists to generate the answers to Wikipedia article questions. For our study, we will however focus on another, slightly simpler, approach to organizing text data. This approach is referred to as the bag-of-words approach.

In the **bag-of-words** approach we do not consider word type or order; instead word frequencies are used as features. In order to organize the text data, we first identify all the unique terms found throughout all the text documents. These unique terms form our “bag of words”. In this case, we do not consider where the words occur in the documents or in what order they occur. We simply form a vocabulary of unique terms. Now, for each text document, it is noted how many times each word in the “bag of words” occurs in the text document. In other words, each word in a text document is treated as a single token so that we can add up the total number of tokens each unique term has in each text document.

Using the bag-of-words approach, the data can be represented either in terms of a document-term-matrix (DTM) or a term-document-matrix (TDM). A DTM is a matrix with the respective text documents represented as rows, while the columns of the matrix are all the unique terms found throughout all the documents. A DTM is illustrated in Figure 5.1.

	Term 1	Term 2	⋯	Term p
Document 1	$n_{1,1}$	$n_{1,2}$	⋯	$n_{1,p}$
Document 2	$n_{2,1}$	$n_{2,2}$	⋯	$n_{2,p}$
⋮	⋮	⋮	⋯	⋮
Document N	$n_{N,1}$	$n_{N,2}$	⋯	$n_{N,p}$

Figure 5.1: Representation of DTM

Figure 5.1 shows that there are N documents and p unique terms in the text data. The DTM, therefore, consists of N rows and p columns (represented by the shaded area in Figure 5.1). Each entry of the matrix corresponds to the number of times each term occurs in each document. For example, the row marked "Document 1" represents the first document in the text data. For this document, the number of times each unique term occurs in the document is represented by the respective column entries, so that "Term 1" occurs $n_{1,1}$ times in the first document, "Term 2" occurs $n_{1,2}$ times in the first document, and so on.

A TDM, on the other hand, is a matrix with the documents represented as columns, while the rows of the matrix are all the unique terms found throughout all the documents. In general, a TDM is thus the transpose of a DTM. Figure 5.2 provides an illustration.

	Document 1	Document 2	⋯	Document N
Term 1	$n_{1,1}$	$n_{2,1}$	⋯	$n_{N,1}$
Term 2	$n_{1,2}$	$n_{2,2}$	⋯	$n_{N,2}$
⋮	⋮	⋮	⋯	⋮
Term p	$n_{1,p}$	$n_{2,p}$	⋯	$n_{N,p}$

Figure 5.2: Representation of TDM

Using the bag-of-words approach to organize text data, an appropriate model can be fit to the $N \times p$ DTM. In this case, the rows of the DTM will act as the observations and each observation

will be a p-component vector, where p corresponds to the total number of unique terms in the DTM. The p input variables of such an analysis will thus be the frequencies of the p terms for each of the documents.

5.1.2 R packages

When analysing text data in R, packages tm, wordcloud and SnowballC can be very useful. The tm package in particular helps users to easily import, clean and pre-process text data. Functions are also included for inspection of text data, as well as quick construction of a DTM or TDM (Feinerer and Hornik, 2018). Feinerer (2018) gives a helpful introduction to the tm package. Packages wordcloud (Fellows, 2014) and SnowballC (Bouchet-Valat, 2014), on the other hand, provide effective ways of visualizing text data.

- **Importing data**

In the tm package, the main approach used to organize text documents is via a so-called “corpus”. A corpus in R consists of a collection of documents that R recognizes as a data type. There are two main types of corpus data, namely a permanent corpus (PCorpus) and a volatile corpus (VCorpus). R stores the two types differently, which is the main difference between them. A VCorpus stores documents fully in R, whereas a PCorpus stores the documents outside of R. We prefer working with a VCorpus since it appears to be the most straightforward.

The function *VCorpus()*, available in the tm package, creates a VCorpus. The function has two arguments: x and readerControl. Here argument “x” should be in the form of a so-called “source”-object. Predefined source-functions that create source-objects are found in the tm package. These functions include *DirSource()*, *VectorSource()* and *DataframeSource()*. We prefer using *VectorSource()*. This function takes a vector as input and interprets each element in the vector as a document. Argument “readerControl” in the *VCorpus()* function is a list with components “reader” and “language”. Each source-object has a default “reader”. A source-object created with *VectorSource()* as the x-input for *VCorpus()*, has its own default value for the “reader” component. The language component of “readerControl” sets the text language. The default language is English.

When importing text data into R, one of the easiest methods is to import the data as a vector with elements the respective documents. A vector like this can then be converted to a source-object using the function *VectorSource()*, after which *VCorpus()* can be used to convert this source-object into a VCorpus.

A VCorpus object is a nested list. To review a single document, we subset with double square brackets, indicating the number of the document we wish to access. To review the text of the document itself, we have to subset twice. To access the text of a particular document, we subset the first argument of the document. As an example, suppose we wish to find the text of the fifth document of a VCorpus object named “texttrain”. The following R code gives the desired result:

```
> texttrain[[5]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 67

> texttrain[[5]][1]
$content
[1] "You, sir, are my hero. Any chance you remember what page that's on?"
```

- **Cleaning**

Once the data are imported into R, the VCorpus has to be cleaned. Before organizing the text data by using for example the bag-of-words approach, it is important to first clean the data as far as possible. We may for example wish to remove all numbers from the text documents or we may decide that it is not necessary to distinguish between lower- and upper-case letters. All text datasets are not cleaned in the exact same way. The nature of the text data and the goal of the analysis would affect what the cleaning process would involve. If the analysis focusses on numeric aspects, it might for example be inappropriate to remove the numbers from the text documents.

Using the R function *tolower()*, a function in base R, translates all characters to lower-case characters. This is useful if there is no need to distinguish between words spelled exactly the same with the only difference the use of lower- and upper-case characters. The tm package provides many other pre-processing functions that can be applied to a corpus. Some of these functions include the following.

The function *stripWhitespace()* removes tabs and extra spaces from a text document so that multiple whitespace characters are collapsed to a single blank. The function *removeWords()* removes specific words from a text document. This is useful if the data scientist would like to remove words that are believed to be uninformative from the documents. For example, it might be appropriate to remove words like “I”, “me”,”are”, etc. A list of 174 common English words, normally appropriate to exclude, is provided in the tm package. This list is given in Appendix B.1. The *removeWords()* function also allows specification of additional words to remove from a

corpus. The function *removePunctuation()* removes punctuation marks, such as periods and exclamation points, from a text document. Similarly, the function *removeNumbers()* removes numbers from a text document.

These transformation functions can be applied to a corpus using the *tm_map()* function found in the tm package. The individual functions work on single text documents, so the *tm_map()* function is used to apply the relevant function to all the documents in the corpus. The *tm_map()* function takes a corpus as input as well as one of the pre-processing functions. If we would like to use a pre-processing function not contained in the tm package, such a function can still be applied to the corpus by wrapping it in the *content_transformer()* function of the tm package. In this case the wrapped function is used as the second argument in the *tm_map()* function. The R package qdap offers more text cleaning functions (Goodrich *et al*., 2018).

When analysing text data, it might be unnecessary to distinguish between different tenses or plurals of a word. If the bag-of-words approach is for example used, distinguishing between different tenses or plurals may result in an unnecessarily large number of features. The extra features might not contain a valuable amount of extra information. Therefore, another pre-processing step may involve “word stemming”. For example, we may decide it is unnecessary to distinguish between the words “girl” and “girls” in documents. We might prefer stemming both words to their base: “girl”. Words can be stemmed in a corpus by hand, or a function like *stemDocument()*, found in the tm package, can be used.

- **DTM and TDM**

Once a corpus is cleaned, we may construct a DTM or TDM of the cleaned corpus. A DTM can be constructed using the function *DocumentTermMatrix()*, available in the tm package. The function takes a corpus as input to construct the corresponding DTM. Similarly, a TDM can be constructed using the function *TermDocumentMatrix()*, also available in the tm package. The function is also applied to a corpus in order to construct its corresponding TDM.

The tm package provides some useful functions that can be applied to a DTM or a TDM for quick inspections. For example, the function *findFreqTerms()* finds frequent terms in a DTM or TDM. The function has three arguments. The first argument is a DTM or TDM for which we wish to find frequent terms. The second argument is a lower frequency bound and the third an upper frequency bound. In other words, if we wish to see which terms occur more than 10 times in a DTM, for example, we would apply the function to the relevant DTM and specify the lower frequency bound as 10. In order to simply view all the terms of a DTM or TDM, the function

Terms() may be used. This function takes a DTM or TDM as input and provides all the terms in the matrix as output.

The function *removeSparseTerms()* takes a DTM or TDM as input, as well as a numeric value between zero and one. The output is a smaller DTM or TDM. The numeric value denotes the maximum proportion of sparsity of the terms allowed in the DTM or TDM. Consider for example a DTM. For a specific term, corresponding to a column of the DTM, the proportion of sparsity of this term refers to the proportion of time the column entries are zero. The function therefore removes those terms in the DTM having a higher proportion of sparsity than the specified numeric value.

The tm package also provides many other functions that uses a DTM or TDM as input. Feinerer and Hornik (2018) may be referred to for more of these functions.

- **Analysis via word clouds**

In a text analysis, visual representations of the data may be very helpful. One of these representations is a so-called “word cloud”. A word cloud is a cloud of words, where terms that correspond to a larger digit are represented by a larger font than terms corresponding to a smaller digit. For example, these digits may be the number of times each term occurs throughout all the text documents. At a quick glance, the reader can see which terms occur often and which terms occur less often, by considering the relative font sizes of the relevant terms in the word cloud. Packages wordcloud and SnowballC have useful functions, for example *wordcloud()*, that can easily plot word clouds (STHDA: Text mining and word cloud fundamentals in R, 2018). An example of a word cloud is given in Figure 5.3.

Data:

1	multi-label	200
2	classification	120
3	ensemble	80
4	NLP	20
5	text	120
6	classes	47
7	label	189
8	Emotions	12
9	loss	25
10	labelsets	37
11	Benchmark	51
12	ROC	23
13	recall	68
14	powerset	70
15	DTM	157
16	corpus	105
17	cardinality	72
18	data	167
19	algorithm	12
20	trees	41
21	multi-class	7
22	observations	33
23	precision	27
24	packages	19
25	accuracy	22
26	Hamming	8
27	Scene	15
28	performance	23
29	correlation	31
30	density	9

Corresponding word cloud:

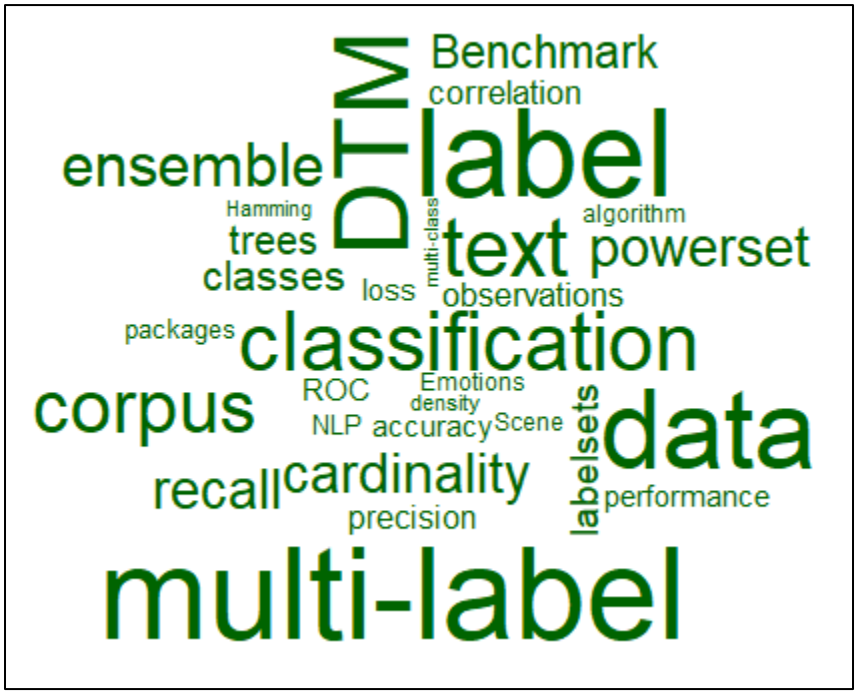


Figure 5.3: Word cloud example

5.2 MULTI-LABEL ANALYSIS IN R

R packages, particularly designed to handle multi-label datasets, can be very helpful in a multi-label analysis. Considerable time can be saved if a package provides functions that can quickly import multi-label data and calculate multi-label data characteristics, such as the number of labelsets or the total number of instances in each labelset, for example. When it comes to multi-label classification, it is also useful if a R package can apply standard multi-label learning methods, allow users the opportunity to easily classify unseen observations and evaluate the results obtained.

5.2.1 Package mldr

A useful R package designed to handle multi-label datasets is package mldr (Charte and Charte, 2018). The package helps users load multi-label datasets into R easily, calculate certain attributes of the datasets and produce interesting plots. Charte and Charte (2018) prove to be a helpful guide in mastering this package.

The function *mldr()* found in the mldr package is used to construct an mldr-object of a multi-label dataset. An mldr-object is the format used by the mldr package to express multi-label datasets. The object contains the full multi-label dataset as well as additional measurements. Some of these additional measurements include the number of observations, the number of input variables, the number of labels, the number of observations per label, the number of distinct labelsets and the number of instances in each labelset.

We use the *mldr_from_dataframe()* function, also available in the mldr package, to create an mldr-object. This function creates an mldr-object from a dataframe of the relevant multi-label dataset and a vector indicating the label indices. Using the *summary()* function on an mldr-object gives a quick summary of the mldr-object. Among others characteristics, the summary also includes the cardinality and density of the multi-label data.

The mldr package also has its own *plot()* function. This function has two arguments. The first argument is the relevant mldr-object and the second argument is the type of plot to be drawn. There are several types of plots available in the mldr package, each highlighting specific characteristics of the relevant mldr-object. For example, type “LB” is a label barplot, displaying the number of instances for each label. On the other hand, type “LSB” is a labelset barplot, displaying the number of instances for each of the respective labelsets. A cardinality histogram is given by “CH”. This plot illustrates the number of labels observations have in general. A pie chart, that illustrates attribute types, is given by “AT”. Lastly, a concurrence plot is given by “LC”. This

plot gives a visual representation of interactions among labels. It is circular and its circumference is partitioned into disjoint arcs representing the labels of the dataset. Each arc, corresponding to a specific label, has length proportional to the number of instances where this label is present, along with one or more of the other labels in the dataset. An arc may be connected to other arcs by means of colour coded bands that join two arcs at a time. The width of each band connecting two arcs, is proportional to the number of instances in which the corresponding two labels appear together. In this way the plot shows the relation between labels. To produce a concurrence plot of a random subset of the K labels, argument “labelCount” can be used. The argument should be an integer indicating the number of labels to choose at random. If users would like to specify which labels the concurrence plot should consist of, argument “labelIndices” can be used.

The mldr package also provides users with a web-based graphical user interface (GUI). This works on top of the R package shiny. A GUI like this is very useful if users would like to find different measurements easily and save high quality plots and other results for a relevant mldr-object. Even those users with little experience in R are thereby given the opportunity to use some of the useful mldr package tools.

The GUI is launched from the R console using the following command once the mldr package has been loaded:

```
> mldrGUI()
```

The command opens the default browser in which the GUI is displayed. From here the relevant mldr-object can be selected for analysis.

5.2.2 Package mldr.datasets

An R package that allows easy access to a large number of multi-label benchmark datasets, is package mldr.datasets (Charte and Charte, 2017). The package provides a variety of benchmark datasets from different domains such as text, sound, music, image and biology. The datasets also vary in size in terms of the numbers of observations, input variables, labels and label cardinality. Function *available.mldrs()* gives an up-to-date list of additional datasets not included in the mldr.datasets package, but which can be downloaded and saved. The datasets in the package are mainly stored as mldr-objects.

5.2.3 Package utiml

When it comes to classification, an R package useful for multi-label classification is package utiml (Rivolli, 2018). The package includes certain multi-label classification methods, sampling methods, threshold functions and evaluation metrics. Most of the functions in the utiml package require multi-label datasets to be expressed as mldr-objects. Rivolli (2016) is a useful guide to the utiml package, showing implementations of the package functions.

Binary relevance, classifier chains, label powerset and ensembles of classifier chains can be fit to an mldr-object using the following utiml package functions: *br()*, *cc()*, *lp()* and *ecc()*. The utiml package also has the function *rakel()* that fits RAkEL. Both RAkEL$_d$ and RAkEL$_o$ can be fit using this function, simply setting the "overlapping"-argument equal to "FALSE" to fit RAkEL$_d$. The default value of the overlapping-argument is "TRUE", so that RAkEL$_o$ is the default approach.

In order to use different base learners for each of the learning methods provided in the utiml package, additional R packages should be loaded which support the base learners. Base learners do not come pre-installed with the utiml package. Table 5.1 highlights possible base learners with their corresponding R packages.

Table 5.1: R packages for base learners supported by the utiml package

Base learner	**R Package**
Classification tree	rpart
$k-$nearest neighbours	kknn
Random forest	randomForest
Support vector machine	e1071

Source: Rivolli, 2016.

If it is for example necessary to create a k-fold partitioning of an mldr-object, we may use *create_kfold_partition()*, found in the utiml package. Using the "method"-argument, we specify the method used when constructing the k folds. For example, setting the method-argument equal to "random", randomly splits the data into k folds. On the other hand, if the method-argument is

equal to “iterative”, the folds are constructed considering the label proportions individually, whereas “stratified” constructs the folds considering the labelset proportions. It is also possible for users to create their own partitioning methods.

Some of the threshold functions provided in the utiml package are *fixed_threshold()*, *lcard_threshold()* and *scut_threshold()*. The simplest of these is *fixed_threshold()*. With this function users manually specify the threshold used to transform a matrix of posterior probabilities to $1/0$ classifications. A global threshold can be specified for all the labels or a threshold can be specified for each label separately.

For evaluation metrics, the utiml function *multilabel_evaluate()* can be used. Several evaluation metrics can be calculated including, among others, accuracy, precision, recall and Hamming loss. A confusion matrix can also be constructed using the function *multilabel_confusion_matrix()*.

The utiml package provides many other functions; in this discussion we simply highlight some of those functions that are considered to be the most useful for our analysis. Rivolli (2018) can be investigated for further helpful utiml-package functions and descriptions of their use.

Table 5.2 summarizes the R packages and useful functions discussed in Section 5.1.2, Section 5.2.1, Section 5.2.2 and Section 5.2.3.

Table 5.2: Summary of R packages

Package tm (Section 5.1.2)	Allows users to easily import, clean, pre-process and inspect text data.
Functions:	**Usages:**
VCorpus()	Creates a VCorpus.
DirSource(), *VectorSource()* and *DataframeSource()*	Predefined source-functions that create source-objects.
stripWhitespace()	Removes tabs and extra spaces from a text document so that multiple whitespace characters are collapsed to a single blank.
removeWords()	Removes specific words from a text document.
removePunctuation()	Removes punctuation marks from a text document.
removeNumbers()	Removes numbers from a text document.
tm_map()	Used to apply a transformation function to a corpus.
content_transformer()	Used to apply a pre-processing function, not contained in the tm package, to a corpus.
DocumentTermMatrix()	Constructs a DTM of a corpus.
TermDocumentMatrix()	Constructs a TDM of a corpus.
findFreqTerms()	Finds frequent terms in a DTM or TDM.
Terms()	Used to view all the terms a DTM or DTM.
removeSparseTerms()	Removes sparse terms from a DTM or TDM.
Package wordcloud and SnowballC (Section 5.1.2)	Useful when visualizing text data.
Functions:	**Usages:**
wordcloud()	Constructs a word cloud.
Package mldr (Section 5.2.1)	Useful when loading multi-label datasets into R, calculating attributes of multi-label data or to produce interesting plots.
Functions:	**Usages:**
mldr()	Constructs an mldr-object of a multi-label dataset found in a file.
mldr_from_dataframe()	Creates an mldr-object from a dataframe of multi-label data.
summary()	Used on an mldr-object to produce a summary of the object.
plot()	type=”LB” gives a label barplot. type=”LSB” gives a labelset barplot. type=”CH” gives a cardinality histogram. type=”AT” gives a pie chart of attribute types. type= “LC” gives a concurrence plot.
mldrGUI()	Opens default browser in which GUI is displayed.

Package mldr.datasets (Section 5.2.2)	Allows easy access to a large number of multi-label benchmark datasets.
Package utiml (Section 5.2.3)	Useful for multi-label classification.
Functions:	**Usages:**
br()	Fits binary relevance model.
cc()	Fits classifier chains model.
lp()	Fits label powerset model.
ecc()	Fits ensemble of classifier chains model.
rakel()	Fits RAkEL$_d$ or RAkEL$_o$.
create_kfold_partition()	Creates a k-fold partitioning of an mldr-object.
fixed_threshold()	Transforms a matrix of posterior probabilities to 1/0 classifications applying the user-specified threshold.
multilabel_evaluate()	Calculates several multi-label evaluation measures.
multilabel_confusion_matrix()	Constructs confusion matrices.

5.3 PRACTICAL DATA ANALYSIS

In this section a multi-label classification text dataset, available on Kaggle, is used to perform a practical data analysis. The dataset consists of 159571 online comments from Wikipedia talk page edits and six labels, namely: Toxic, Severe Toxic, Obscene, Threat, Insult and Identity hate. Each comment is individually labeled by human raters to identify if any or multiple so-called "toxicity" are present in the comment. Here "toxicity" refers to comments that may be interpreted as rude or disrespectful. The aim is to identify different types of toxicity present in online comments. The overall goal is to reduce abuse and harassment online and help online discussions become more productive and respectful (Kaggle: Toxic comment classification challenge, 2018)

The data are initially randomly split into training and test data. Since the dataset is quite large, approximately two thirds of the data are used as training observations (106381 comments) and the remaining third of the data are used as test observations.

Since the observations in the dataset are in the form of text, the R package tm is used. The "bag-of-words" approach is used to organize the data.

The training data are imported into R as a vector with elements the respective comments. This training-vector is converted using the function *VectorSource()* in order to interpret each element of the training-vector, in other words each comment, as a document. The function *VCorpus()* is used to convert this source-object into a VCorpus. We convert the data to a VCorpus so that R recognizes the collection of comments as a data type to which the tm pre-processing functions can easily be applied.

5.3.1 Initial cleaning and pre-processing

Merely using the training corpus as given above to apply the bag-of-words approach, would result in noise in the data that could have been avoided. Considering the nature of our text data and the goal of our analysis, the following approach is used in order to clean and pre-process the data.

Functions *tolower()*, *stripWhitespace()*, *removeWords()*, *removePunctuation()* and *removeNumbers()* are used to convert all text to lower-case, remove extra white spaces, remove the tm package list of 174 common English words, remove punctuation and remove numbers from the text documents. The function arguments applied to the training corpus are provided in Appendix B.2.2.

After the training corpus has been cleaned in the above way, an initial inspection of the data is executed as follows. A DTM is constructed using the function *DocumentTermMatrix()* in the tm package. The function uses the cleaned training corpus as input to construct the corresponding DTM. In our case the DTM contains 106381 documents (comments) and 171082 unique terms. The DTM is illustrated in Figure 5.4.

	Term 1	Term 2	$\cdots$	Term 171082
Comment 1	$n_{1,1}$	$n_{1,2}$	$\cdots$	$n_{1,171082}$
Comment 2	$n_{2,1}$	$n_{2,2}$	$\cdots$	$n_{2,171082}$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
Comment 106381	$n_{106381,1}$	$n_{106381,2}$	$\cdots$	$n_{106381,171082}$

Figure 5.4: Representation of DTM

5.3.2 Rude terms

Due to the nature of the dataset, some of the text may be considered profane, vulgar or offensive. However, since the goal is to detect toxic comments, intuitively, the presence of a rude term in a comment would be a strong indication of the presence of toxicity in that comment. To avoid continuously referring to these rude terms in the analysis, a function was written to replace commonly occurring rude terms with alternative codenames. Note that this function uses a corpus as input, therefore the replacement takes place in the training corpus. The function furthermore requires a list of rude terms and their corresponding codenames as input. A list of commonly occurring rude terms is formed by continuously identifying rude terms as they appear in the analysis described in this section. Even though the analysis is described below, and the list of rude terms is only complete after the analysis has been completed, we replace all the rude terms with their alternative codenames in the training corpus and repeat our analysis with the codenames in place. This is done to prevent the reader from having continuous exposure to these rude terms. Therefore, the analysis described below contains the codenames, not the original rude terms.

There are 104 rude terms identified, with the codenames being “aanstoot1”, “aanstoot2” up until “aanstoot104”. It is important to make sure that the chosen codenames are terms which do not occur in the text data beforehand, otherwise the replacement process creates extra noise in the data. If a codename is already included in the data before the replacement process takes place, the frequency of this term is falsely increased after the replacement. The term may now also appear in a context it did not appear in previously. The codenames should thus be chosen with care.

Since the data consist of informal comments, there are many alternative spellings and synonyms of the 104 rude terms. The synonyms and alternative spellings of the 104 terms are therefore also identified and replaced with one of the corresponding 104 codenames. A DTM is constructed using the training corpus that contains the codenames.

5.3.3 Top occurring terms

The DTM, containing the 104 codenames, is now used to inspect the overall top occurring terms. This is done by using the function *removeSparseTerms()*. Recall that this function takes a DTM and a numeric value between zero and one as input. The numeric value denotes the maximum allowed proportion of sparsity of the terms. Applying the function to the newly constructed DTM and using an initial numeric value of 0.9988, the top occurring terms are found. To view these terms, the sparse DTM is used as input for the function *Terms()*. These 3036 terms can now be carefully inspected.

Some of the top terms include synonyms or different spellings of a term, for example: encyclopaedia, encyclopedia. Some of these top terms also include the different tenses of a word, for example: delete, deleted, deleting. We also see that some words are listed along with their plural form, for example: animal, animals. As stated previously, it might be unnecessary to distinguish between synonyms, different tenses or plurals of a word for a text data analysis. We can for example combine “encyclopaedia” and “encyclopedia” into one of the two terms. We can also for example combine “delete”, “deleted” and “deleting” into one term: delete. Similarly, we can combine “animal” and “animals” into one term: animal. Some terms, however, can be defined as both a verb or a noun, for example: attack. The list of top terms include "attack", "attacked", "attacking" and "attacks". Combining all these terms into one term, “attack”, may be detrimental to the analysis, since their usages are not quite interchangeable.

The top terms are examined in order to decide which terms to combine and which terms to leave separate. This is a very subjective process. The main consideration used to assist in making this decision is to determine if the relevant term has both a noun and verb definition. If the term is only regarded as a verb, and the different tenses of this word is found in the list of top terms, these terms are all combined into the present tense form of the term. If the term is only regarded as a noun, and the plural of the term is also found in the list of top occurring terms, the plural form is replaced by the singular form. In the case where the term has both a noun and verb definition, such as in the case of the term “attack”, not all the forms are combined. Instead the words “attacked” and “attacking” are combined and replaced with the word “attacked”. The forms “attack” and “attacks” are left untouched. Some top terms include adverbs, for example “badly”. Both the words “bad” and “badly” are included in the list of top terms. Adverbs and adjectives are not removed by combining them with other terms. The function used to execute the replacements is the same function that was used to replace the rude terms with codenames. Even with the above criteria in mind, the process of deciding whether it would be advantageous to combine words, remains subjective. Those terms for which the decision was not clear are rather not combined. A list of those terms combined is found in Appendix B.2.7.

After these specific terms are combined in the training corpus, we create a new DTM of this corpus and use the *removeSparseTerms()* function with numeric value 0.9988 to create a new list of overall top occurring terms. This new list of overall top occurring terms is found in Appendix B.2.9.

5.3.4 Label inspection

Up until this point the training corpus has been cleaned as follows. All upper-case characters were replaced with lower-case characters. Extra white spaces were removed. We removed the tm package list of 174 common English words. We removed punctuation from the training corpus, as well as all numbers. We replaced rude words and their variations with 104 codenames. We also combined some words in the training corpus that seemed to have interchangeable usages. This cleaned training corpus is now used in order to inspect the labels more closely.

Before the label inspection process is described, some notation for the data is introduced. As mentioned before, there are $K = 6$ labels in this multi-label classification dataset, namely Toxic, Severe Toxic, Obscene, Threat, Insult and Identity hate. These labels are denoted by $\{Y_1, Y_2, ..., Y_6\}$ respectively. Each one of the 106381 comments in the training corpus has a corresponding six-component vector containing the binary label variables, $\mathbf{y}_i$, $i = 1, ..., 106381$.

The labels are sparse, *i.e.* it is rare for $\mathbf{y}_i$, $i = 1,...,106381,$ to have one or more entries equal to 1.

The Toxic label is the least sparse of the six labels. Approximately 9.613% of the training comments have $Y_1 = 1$. The Severe Toxic label has $Y_2 = 1$ for approximately 0.989% of the training comments. The Obscene label has $Y_3 = 1$ for approximately 5.301% of the training comments. The Threat label is the sparsest, having $Y_4 = 1$ for only approximately 0.311% of the training comments. The Insult label has $Y_5 = 1$ for approximately 4.954% of the training comments. Lastly, the Identity hate label has $Y_6 = 1$ for approximately 0.856% of the training comments.

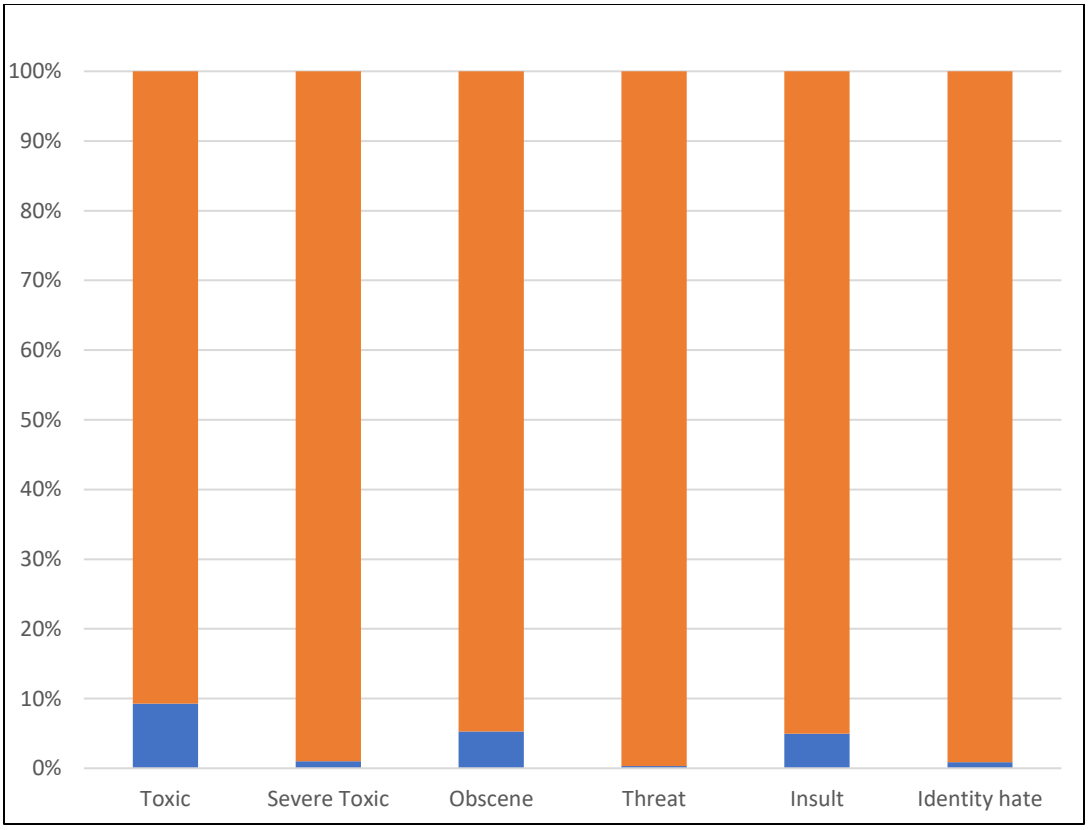


Figure 5.5: Stacked column chart illustrating sparseness of labels

Figure 5.5 illustrates the sparsity of the labels in the dataset. Each column represents one of the six labels, $\{Y_1, Y_2, ..., Y_6\}$. The percentage of comments having $Y_k = 1$, where $k = 1, 2, ..., 6$, is represented in blue, whereas the percentage of comments having $Y_k = 0$, where $k = 1, 2, ..., 6$, is represented in orange. From Figure 5.5 it is clear that the labels are very sparse.

It might be advantageous to specifically investigate the rare comments that are equal to 1 for one or more of the labels. We would like to investigate each label separately and determine what are the top occurring terms when considering only those comments for which this label is present. In order to do this, we proceed as follows.

For illustration, consider Y_1. We first split the cleaned training corpus into two corpuses. The first corpus contains all those comments for which $Y_1 = 1$ and the second corpus contains all those comments for which $Y_1 = 0$. Now a TDM is constructed for the first corpus by using the tm package function *TermDocumentMatrix()*. This TDM is used to find the frequency of each of the respective terms given as rows in the TDM. Thus, the row-sums of the TDM are calculated. Using these term-frequencies, the fifty top occurring terms are determined and represented in a barplot. This process is repeated for labels $Y_2, Y_3, ..., Y_6$. The results for all six labels are found in Figure 5.6.

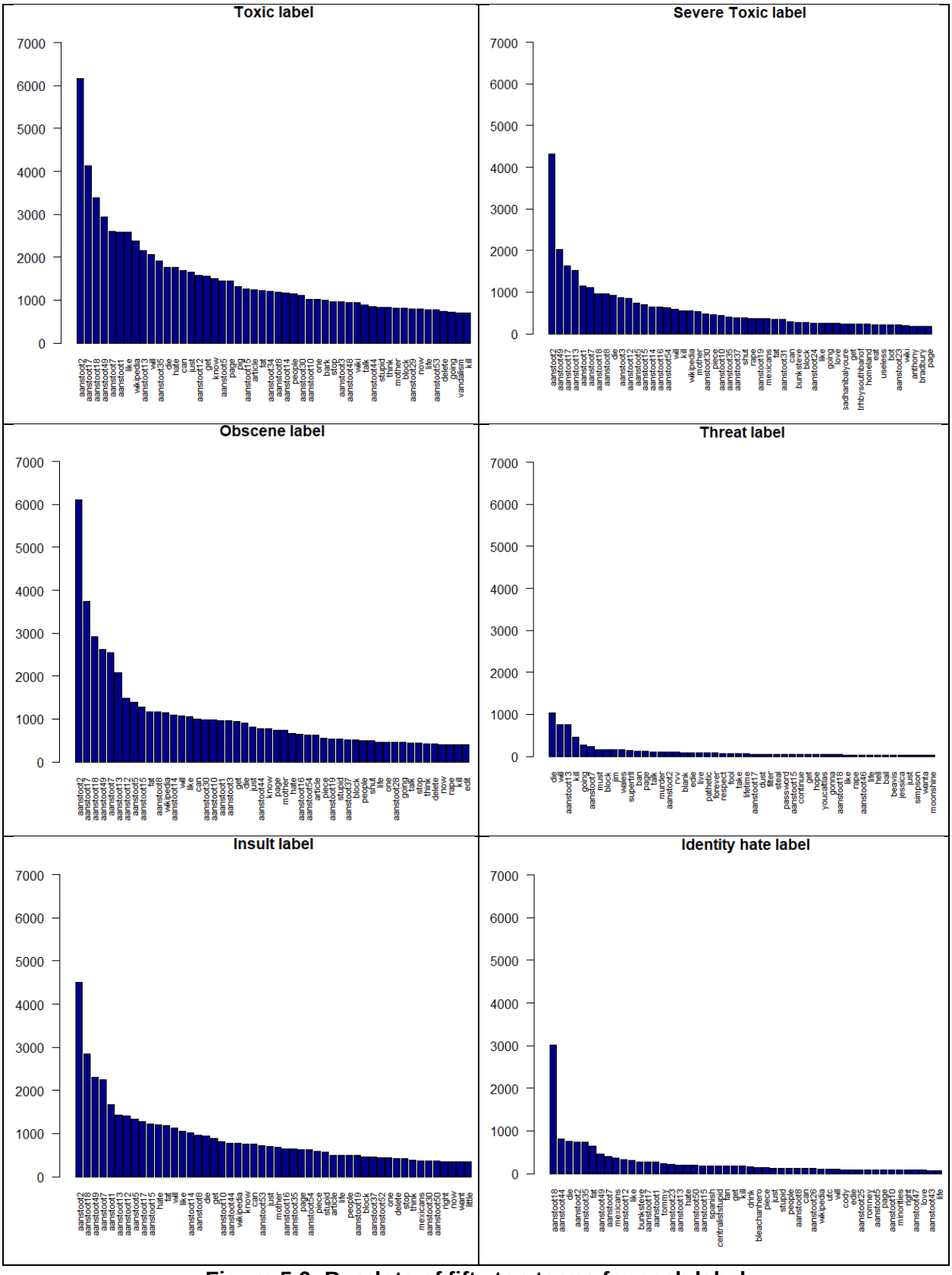


Figure 5.6: Barplots of fifty top terms for each label

Inspecting the barplots in Figure 5.6, it appears that for each label there is one term with a significantly higher frequency than the other terms. For all the labels, except the Threat and Identity hate label, this term is “aanstoot2”. The term that occurs significantly more than the other terms for the Threat label is “die”. In the case of the Identity hate label, this term is “aanstoot18”. Inspecting Figure 5.6 more closely, it also appears that following the term with the highest frequency are four or five terms that occur at a moderate frequency, before the frequencies drop significantly and only small differences in frequencies are seen for the remaining terms.

It is also noted that some of the terms found in Figure 5.6, that thus form part of the fifty top terms for a particular label, are not included in the list of overall top occurring terms for the entire text dataset. This might be due to the fact that the labels are so sparse. An example of such a term is “bark”.

There are also some unexpected terms in Figure 5.6. For example, for the Threat label, one of the terms is a proper noun: jim. Why would this term have such a high frequency? Upon further inspection, it turns out that one of the comments repeats the words “jim wales must die” 157 times. In fact, there appears to be several comments that consist of one sentence that is repeated multiple times. This behaviour also explains other unfamiliar terms in Figure 5.6: “jessica”, “bunksteve”, “beavis”, etc. These terms have high frequencies because they appear a significant number of times in a few comments. However, we might be more interested in terms that have a high frequency because they appear a few times per comment, but appear in many of the toxic comments. Terms with frequencies more spread out like this might be more informative to the analysis. In order to reduce the effect that a few individuals’ inappropriate behaviour has on the term frequencies, we proceed as follows.

For each label, we no longer merely use the row-sums of the TDM as the term-frequencies. Instead, for each separate comment, it is noted if a term is found in that particular comment or not, regardless of how many times it occurs in the comment. This can easily be found for a particular label by replacing all the entries in the label’s TDM that are larger than or equal to one with a “1”. The TDM then only consists of “1”s and “0”s. An entry is equal to 1 if the comment (corresponding to the column of that entry) contains the term (corresponding to the particular row of that entry), else the entry is 0. We execute this replacement process for each of the six TDMs that correspond to the labels. Considering each label separately, we can calculate the percentage of comments containing each term. This is possible since it is known how many training comments have $Y_k = 1$ for $k = 1, 2, ..., 6$. Using these percentages, we construct a barplot of the fifty terms

with the highest percentages of occurrence for each label. These six barplots are given in Figure 5.7.

The newly constructed barplots differ from the barplots in Figure 5.6. The term “aanstoot2” is the term with the highest percentage of occurrence in the Toxic, Severe Toxic, Obscene, Insult and Identity hate label. When only considering frequencies, “aanstoot18” is the term with the highest frequency for the Identity hate label. However, when considering the percentage of occurrence, as in Figure 5.7, “aanstoot18” is in the fourth position for the Identity hate label. When only considering the frequencies, as in Figure 5.6, the term with the highest frequency for the Threat label is “die”. Now, when considering the percentage of occurrence, the term with the highest percentage of occurrence for the Threat label is “will” and “die” has moved to the third position.

In Figure 5.6, for each label, there appears to be one term with a significantly higher frequency compared to the other terms. However, in Figure 5.7, for each label, the difference between the first and second terms do not appear to be as significant. The labels with the largest difference in percentage between the first and second terms are Severe Toxic, Obscene and Threat. These differences do not seem to be much larger than 10%.

For each label, we also construct a word cloud of the percentages of occurrence using packages wordcloud and SnowballC. The particular function arguments are given in Appendix B.2.11. The results for all six labels are shown in Figure 5.8. Now, with a quick glance, we can see which terms occur often and which terms occur less often by considering the relative font sizes.

The slightly unusual or unexpected terms found in Figure 5.6, like “jim” and “beavis”, are not present in Figure 5.7 or Figure 5.8. This is due to the fact that these terms only occur in a few comments, and thus the percentages of occurrence of these terms are low. We do however see that there are still terms in Figure 5.7 and Figure 5.8 that are not contained in the list of overall top occurring terms. An example of such a term is “burn”, seen in the barplot and word cloud of the Threat label. As seen in the word cloud of the Threat label, the term “burn” appears to be quite an important term for this label since the font size is not very small. In fact, the term has a bigger font than some other terms in the word cloud which are in fact contained in the list of overall top occurring terms.

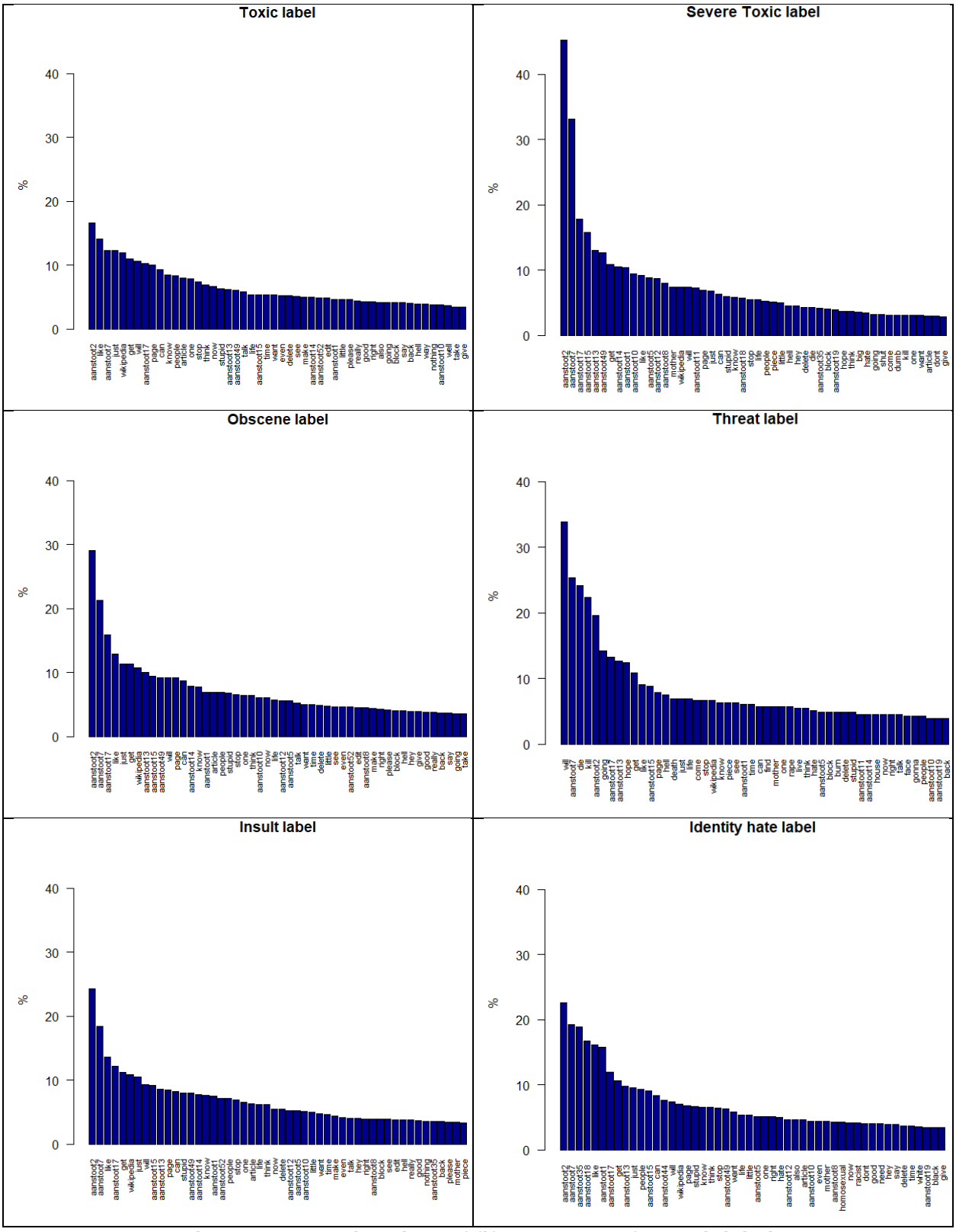


Figure 5.7: New barplots of fifty top terms for each label

Figure 5.8: Word cloud for each label

5.3.5 Feature selection

In later stages of the analysis, we would like to fit a model to a $N \times p$ DTM of our cleaned text data. At that stage the rows of the DTM will act as the observations. Each observation, corresponding to one of the cleaned text comments, will be a p-component vector, where p denotes the total number of unique terms in the DTM. The p input variables will thus be the frequencies of the p terms for each of the comments. The response variables will be the six-component vectors containing the binary label variables. Intuitively, in order to obtain better results, we might prefer that the value of p is not too large. In order to be able to execute this process, we need to perform feature selection, *i.e.* identify the terms to appear in the final DTM.

One simple option is to use p of the overall top occurring terms when viewing all 106381 comments. However, if we merely use these terms, we will exclude a term like “burn”, as identified above. This term is one of the terms with the highest percentage of occurrence in comments when considering only those comments that contain toxicity. Since the labels of our data are sparse, we would like to prevent comments not containing toxicity to overpower the analysis. Therefore, it might be inappropriate to merely use p of the overall top occurring terms. It may be advantageous to include a term like “burn”, for example, since our goal is to detect toxicity in comments.

In order to identify p appropriate terms that will act as input variables for our analysis, we will first identify t, $t > p$, terms from the text data. Denote the collection of t terms by T. In a later stage we will select p of the t terms in T to be the input variables. This selection process is described later. We first describe how to select t terms to form the collection T.

- **Forming collection T**

Since the p input variables (terms) are selected from T, this collection consists of all possible candidates for the final input variables. It is an option to include the overall top occurring terms in T. However, for reasons explained above, we may also wish to include a term such as “burn”, which is not included in the list of overall top occurring terms. We would like to identify more terms such as “burn” to include in T. In Figure 5.7 we constructed a barplot for each label using the fifty terms with the highest percentage of occurrence in comments containing toxicity. In order to identify more terms such as “burn”, we will not only consider the first fifty terms with the highest

percentages of occurrence in comments containing toxicity. Instead, we will view the first m_k terms for Y_k where $m_k > 50$ for $k = 1,...,6$.

It should be noted that for Y_k all m_k terms are unique, however some terms may be identified for more than one of the labels. Thus, viewing all $\sum_{k=1}^{6} m_k$ terms, we first remove duplicated terms and then determine which of these remaining unique terms do not occur in the list of overall top occurring terms. In this way we identify more terms such as “burn” which do not occur in the list of overall top occurring terms, but have high percentages of occurrence when considering only those comments that contain toxicity. The list of overall top occurring terms are therefore augmented by these newly identified terms.

Choosing the values of m_k, $k = 1,2,...,6,$ is subjective. For each label we order the percentages of occurrence for all the terms, from the term with the highest percentage to the term with the lowest. We would like to identify a percentage so that from this percentage onward, there are little differences in term percentages. If this is the case for Y_k, for example, that percentage can be an appropriate cut-off point for Y_k and the number of terms exceeding this percentage for Y_k is equal to m_k. Unfortunately, such a percentage does not exist for any of the labels, because there are no clear cut-off points. Since this is the case, the values of m_k, $k = 1,2,...,6,$ are simply chosen.

It might be best if the values m_k, $k = 1,2,...,6,$ are not too small. In this case, more terms are identified and eventually added to T. Ideally, the selection process would then ensure that terms that are indeed less important are excluded from the p chosen terms. If we however make the $m_k -$values too small, we may exclude some important terms from T and therefore these terms are not allowed the opportunity to be chosen as one of the input variables. On the other hand, making the $m_k -$values too large may include very strange terms that occur in a few comments only. We do not want to allow such terms the opportunity to be identified as one of the p input variables. Considering this, we decide to consider all the terms exceeding approximately 0.2% occurrence for the Toxic, Severe Toxic, Obscene and Insult label. This results in $m_1 = 1301$, $m_2 = 776$, $m_3 = 1147$ and $m_5 = 1128$. Since the Threat and Identity hate labels are sparser, we

consider all terms exceeding approximately 0.9% occurrence for the Threat label and 0.4% for the Identity hate label. This results in $m_4 = 357$ and $m_6 = 721$.

Using the values chosen for m_k, $k = 1,2,...,6$, removing duplicated terms as well as terms that are included in the list of overall top occurring terms, we are left with 272 unique terms. However, as was the case when the overall top occurring terms were identified, the process of identifying these 272 terms have allowed us to spot more words that might be considered as synonyms. Some of these newly found 272 terms include synonyms or different spellings of a term, for example: “gonna” and “gunna”, informal versions of “going to”. Some of the 272 terms are listed along with their plural form, for example: “pig”, “pigs”. Also, some of the 272 terms are the incorrect spelling of a word. For example, one of the 272 terms is “freinds”, which we assume was meant to be “friends”. However, the word “friends” is actually included in the list of overall top occurring terms. The term “bullying” is included in the list of overall top occurring terms, however the term “bullied” is one of the newly found 272 terms. Since these two words are simply different tenses, we might consider combining them into one term: “bullied”. All of these issues can be dealt with by constructing an additional list of synonyms and using the function we used previously to do the appropriate replacements in our current training corpus.

Some of the 272 terms are, however, words like “youre”, “didnt” and “doesnt”. Recall that we initially removed 174 commonly occurring English terms from our training corpus. The terms “you’re”, “didn’t” and “doesn’t” were among the 174 commonly occurring English terms. It appears that some comments, however, include terms like “youre”, “didnt” and “doesnt”, where the author of the comment failed to include the appropriate apostrophe. Fortunately, the tm package function, *removeWords()*, allows us the ability to add additional words we wish to remove from a corpus. Since we removed the correctly spelt versions of the words “youre”, “didnt” and “doesnt”, we should also remove these incorrect versions from our training corpus.

Therefore, before continuing with our analysis, we first clean our current training corpus further by combining some terms of which the full list of combinations is found in Appendix B.2.12. We then apply the *removeWords()* function to this corpus in order to remove additional words from the corpus. The additional terms that are removed are given in Appendix B.2.13. Throughout further analysis this updated training corpus will be used.

Since we have an updated training corpus, we need to update the list of overall top occurring terms as well. This is done by creating a new DTM from the updated training corpus and using the function *removeSparseTerms()* with numeric value 0.9988.

The process of identifying the t terms forming the collection T is now repeated using the updated list of overall top occurring terms and the updated training corpus. Thus, we again view the m_k, $k = 1, 2, ..., 6$, terms with the highest percentages of occurrence in comments containing toxicity. Since some terms are combined and removed in the updated training corpus, these m_k – values will now be slightly smaller, because fewer terms will exceed our specified percentages of occurrence. If we again view terms exceeding approximately 0.2% occurrence for the Toxic, Severe Toxic, Obscene and Insult label, this results in $m_1 = 1290$, $m_2 = 757$, $m_3 = 1135$ and $m_5 = 1114$. Viewing terms exceeding approximately 0.9% and 0.4% occurrence for the Threat and Identity hate labels result in $m_4 = 353$ and $m_6 = 707$.

Using these values for m_k, $k = 1, 2, ..., 6$, removing duplicated terms as well as terms that are included in the updated list of overall top occurring terms, we are now left with 244 unique terms. These terms are added to T along with the overall top occurring terms. Since the codenames of the rude terms are numbered from 1 to 104, it is easy to see which rude terms are not included in T at this point. Since the p input variables are selected from T, we decide to include all 104 rude terms in T so that all the identified rude terms are candidates for the final input variables. Therefore, those rude terms that are not present in T at this point, are also added to the collection. The collection, T, is now complete, containing $t = 3042$ terms.

- **Identifying p terms from T**

We now aim to select p terms from the t terms in T to be the final input variables of the analysis.

One option may be to use a χ^2 feature ranking method separately for each label. This is a so-called filter approach. We obtain a ranking of the terms for each label, using χ^2 tests for each of the labels separately. For each term, we determine the maximum rank obtained considering all six of the label rankings. We now select the top 600 terms based on their maximum rank over all six labels. As stated in Tsoumakas *et al*. (2011), a similar approach was found to have high performance in experimental work on textual data.

We also implement another strategy to select p input variables from T. We specifically develop this method to deal with the sparsity of the labels in the data. The aim of the selection process is to identify p terms that appear to show the largest difference between comments containing toxicity and comments not containing toxicity. The strategy is implemented as follows.

First we construct a DTM of our training corpus; however, we specify the terms of the DTM to be the t terms in T. This is easily done using an additional argument when using the function, *DocumentTermMatrix()*. The DTM therefore consists of 106381 rows, each corresponding to a comment, and t columns, each corresponding to one of the terms in T. Each entry indicates the number of times a term is found in a comment. However, we would now like to transform this DTM so that for each separate comment, it is noted whether a term is found in that particular comment or not, regardless of how many times it occurs in the comment. To execute this transformation, all positive entries in the DTM are replaced with a "1", similar to the strategy used before. Denote the respective columns of this transformed matrix by $X_1,...,X_t$.

Now for each of the six labels separately, the transformed matrix is split row-wise into two sub-matrices. The first sub-matrix contains all the comments for which $Y_k = 1$ and the second matrix contains all the comments for which $Y_k = 0$. For illustration, consider $k = 1$. Say the first matrix is denoted by mat_1 and contains N_1 comments, *i.e.* N_1 of the 106381 comments have $Y_1 = 1$. This matrix is therefore of size $N_1 \times t$. Similarly, say the second matrix is denoted by mat_2 and contains N_2 comments, *i.e.* N_2 of the 106381 comments have $Y_1 = 0$ and mat_2 is of size $N_2 \times t$. We thus have, $N_1 = \sum_{i=1}^{N=106381} Ind\left(Y_1 = 1\right)$, $N_2 = \sum_{i=1}^{N=106381} Ind\left(Y_1 = 0\right)$ and $N_1 + N_2 = 106381$.

We continue by calculating the column sums of each of the two matrices, mat_1 and mat_2. For mat_1, the j^{th} column sum is denoted by N_{1j}. Therefore, of the N_1 comments that have $Y_1 = 1$, N_{1j} of them have the j^{th} term present, where $j = 1,...,t$. Similarly, for mat_2, the j^{th} column sum is denoted by N_{2j}. Therefore, of the N_2 comments that have $Y_1 = 0$, N_{2j} of them have the j^{th} term present, where $j = 1,...,t$. We estimate $P\left(X_j = 1 | Y_1 = 1\right)$ by $\hat{P}\left(X_j = 1 | Y_1 = 1\right) = \frac{N_{1j}}{N_1}$,

$P\left(X_j = 1 | Y_1 = 0\right)$ by $\hat{P}\left(X_j = 1 | Y_1 = 0\right) = \frac{N_{2j}}{N_2}$ and $P\left(X_j = 1\right)$ by $\hat{P}\left(X_j = 1\right) = \frac{N_{1j} + N_{2j}}{N}$. Now for $j = 1, \ldots, t$ we find:

$$score_j = \frac{\left|\hat{P}\left(X_j = 1 | Y_1 = 1\right) - \hat{P}\left(X_j = 1 | Y_1 = 0\right)\right|}{\hat{P}\left(X_j = 1\right)} = \frac{\left|\frac{N_{1j}}{N_1} - \frac{N_{2j}}{N_2}\right|}{\frac{N_{1j} + N_{2j}}{N}}.$$

We repeat this process for all the labels in order to find $K = 6$ $score-$values for each of the t terms, one $score-$value corresponding to each label. For each label we order its t corresponding $score-$values decreasingly. The results are summarized in Figure 5.9.

In Figure 5.9, we see that a clear cut-off $score-$value exists for each of the six labels. Consequently, we identify no more than 500 important terms per label. Since a term may be identified for more than one of the labels, we form a collection of unique terms when considering all the important terms identified for all six labels. Doing this we are left with a collection of 606 terms that we believe show the largest difference between comments containing toxicity and comments not containing toxicity.

The collection of 600 terms identified using the χ^2 feature ranking method and the collection of 606 terms identified using the method described above are very similar. Many terms are found in both collections. Since the developed method selects similar terms to those selected when using the χ^2 feature ranking method, we decide to use the 606 terms found using our developed method as the p input variables of the analysis. We are hopeful that these input variables will help identify the sparse comments that contain toxicity.

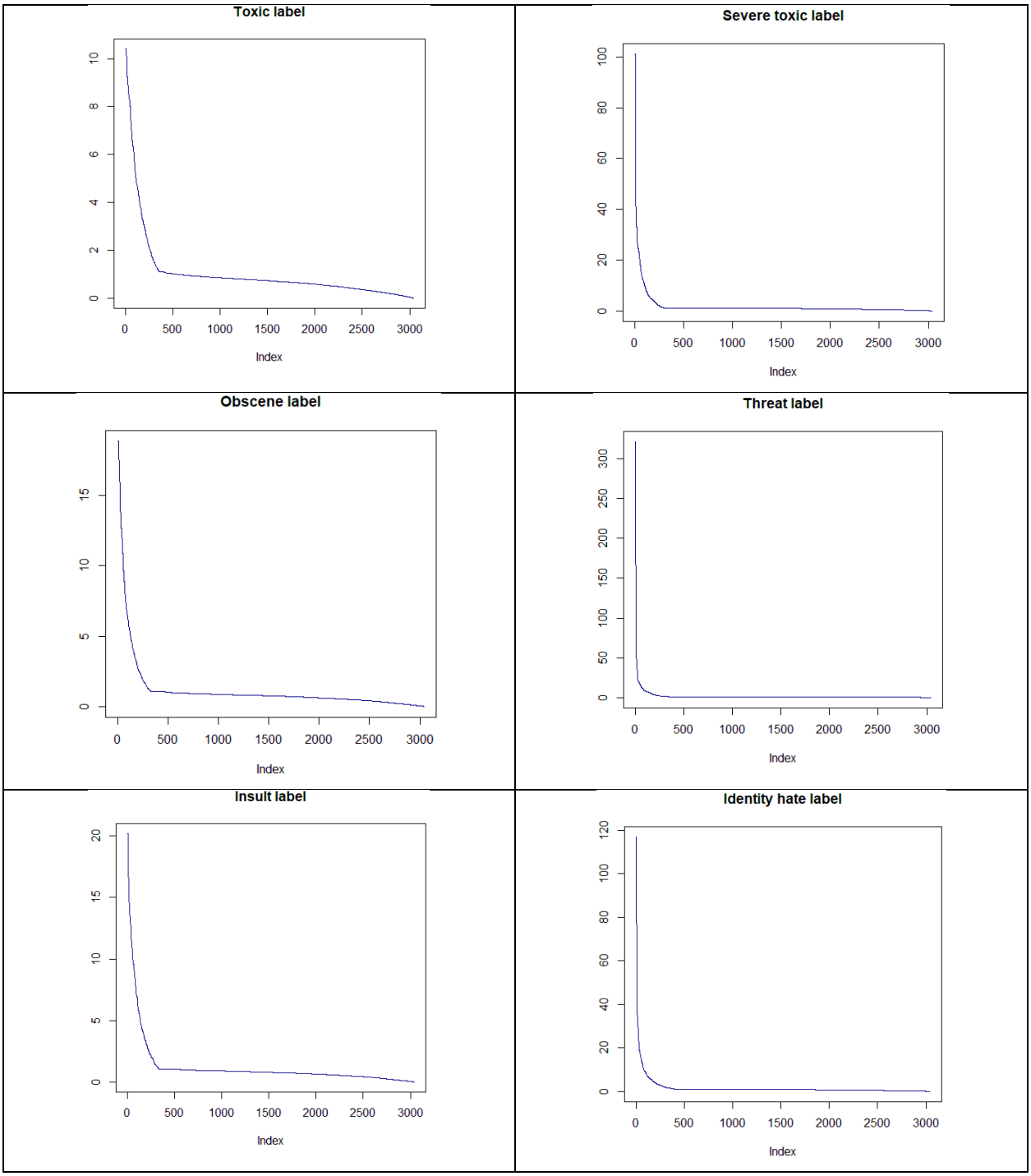


Figure 5.9: Summary of $score-$ values per label

5.3.6 Multi-label inspection

An $N \times p$ DTM of our cleaned text data can now be constructed. The $N = 106381$ observations correspond to the cleaned text comments. Each observation is a $p = 606-$component vector, where each component in the vector is the frequency of each of the p terms in the comment. This DTM forms the input space of our cleaned multi-label classification dataset. The response space is the six-component vectors containing the binary labels for each of the comments. Therefore, our multi-label classification dataset can now be summarized as a 106381×612 matrix, $\left[\underset{N \times p}{\mathbf{X}} \quad \underset{N \times K}{\mathbf{Y}} \right]$, resembling Figure 5.10.

	Terms				Label indicator variables			
Comment	X_1	X_2	...	X_{606}	Y_1	Y_2	...	Y_6
1	$x_{1,1}$	$x_{1,2}$	...	$x_{1,606}$	$y_{1,1}$	$y_{1,2}$	...	$y_{1,6}$
2	$x_{2,1}$	$x_{2,2}$	...	$x_{2,606}$	$y_{2,1}$	$y_{2,2}$	...	$y_{2,6}$
3	$x_{3,1}$	$x_{3,2}$	...	$x_{3,606}$	$y_{3,1}$	$y_{3,2}$	...	$y_{3,6}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
106381	$x_{106381,1}$	$x_{106381,2}$	...	$x_{106381,606}$	$y_{106381,1}$	$y_{106381,2}$	...	$y_{106381,6}$

Figure 5.10: Matrix representation of multi-label classification dataset

The matrix given in Figure 5.10 can be used to construct an mldr-object of the multi-label data. It is useful to construct an mldr-object of our multi-label classification training data, because for both R packages, mldr and utiml, mldr-objects are the format used to express multi-label datasets. We first convert the matrix to a dataframe and then use the function *mldr_from_dataframe()*, available in the mldr package, to form the mldr-object. A summary of this mldr-object is given in Figure 5.11. In Figure 5.11, the specified table summary, containing several characteristics of the mldr-object, is obtained by using the *summary()* function. Furthermore, using the mldr package function, *plot()*, we obtain a cardinality histogram and labelset barplot of the mldr-object.

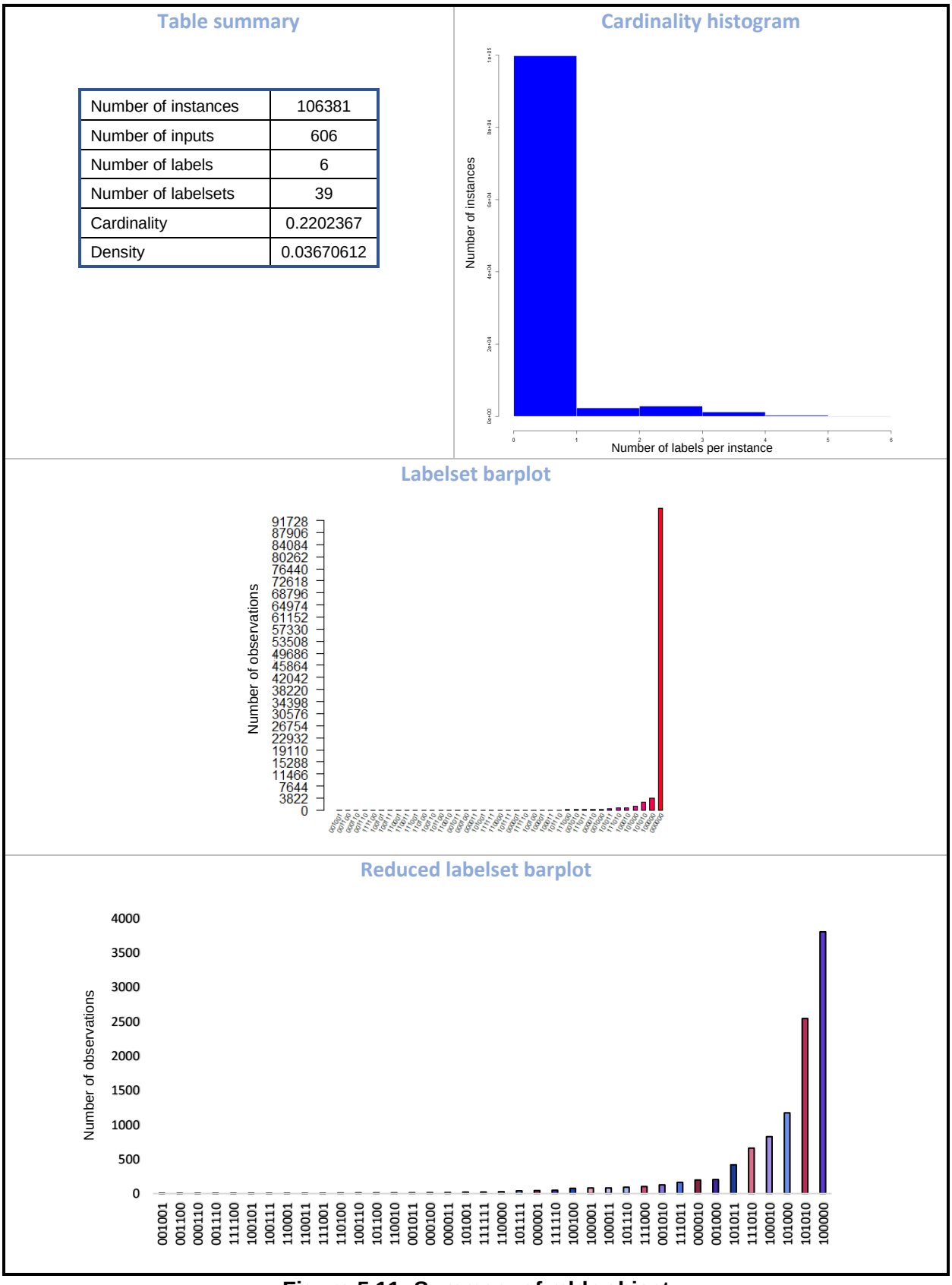


Figure 5.11: Summary of mldr-object

As seen in Figure 5.11, the cardinality of our mldr-object is very low, approximately 0.22. This is expected since many observations have no labels present. Furthermore, since the cardinality histogram shows the extreme accumulation of the data on the left, it is also clear that many observations do not have a notable number of labels present. This is an important characteristic of our multi-label dataset and will probably make classification of unseen comments more difficult. Since many comments do not have any labels present, we can achieve deceivingly good results if we simply classify all test comments to have no labels present. The small number of comments that, in truth, have labels present, will then be the only comments misclassified. However, the goal of the analysis is to detect toxicity in online comments. Simply classifying all comments to be free of toxicity would be of no use. It is important to detect those comments that have forms of toxicity present. This is an important problem in all unbalanced datasets.

Figure 5.11 also indicates that there are 39 labelsets in the mldr-object. The labelset barplot shows the number of observations in each labelset. It is clear that the labelset corresponding to the case where none of the labels are present, completely overpowers the other labelsets. By using the GUI of the mldr package, it is found that 95540 observations have no labels present. In other words, approximately 89.81% of the training comments have no labels.

In order to better inspect those labelsets that have at least one label present, we construct a reduced barplot containing all the labelsets, except the labelset corresponding to the case where none of the labels are present. The frequencies of each labelset is obtained from the GUI. This reduced labelset barplot is also given in Figure 5.11. The most common labelset in this reduced setting is the labelset corresponding to the case where the Toxic label is the only label present. This is followed by the labelset that only has the Toxic, Obscene and Insult labels present. These three labels are also the top three most common labels. The third most common labelset is the labelset that only has the Toxic and Obscene labels present. Furthermore, in this reduced setting, we see that the different labelsets often include the Toxic label. For the top 21 labelsets in the reduced setting, the Toxic label is present in 17 of them.

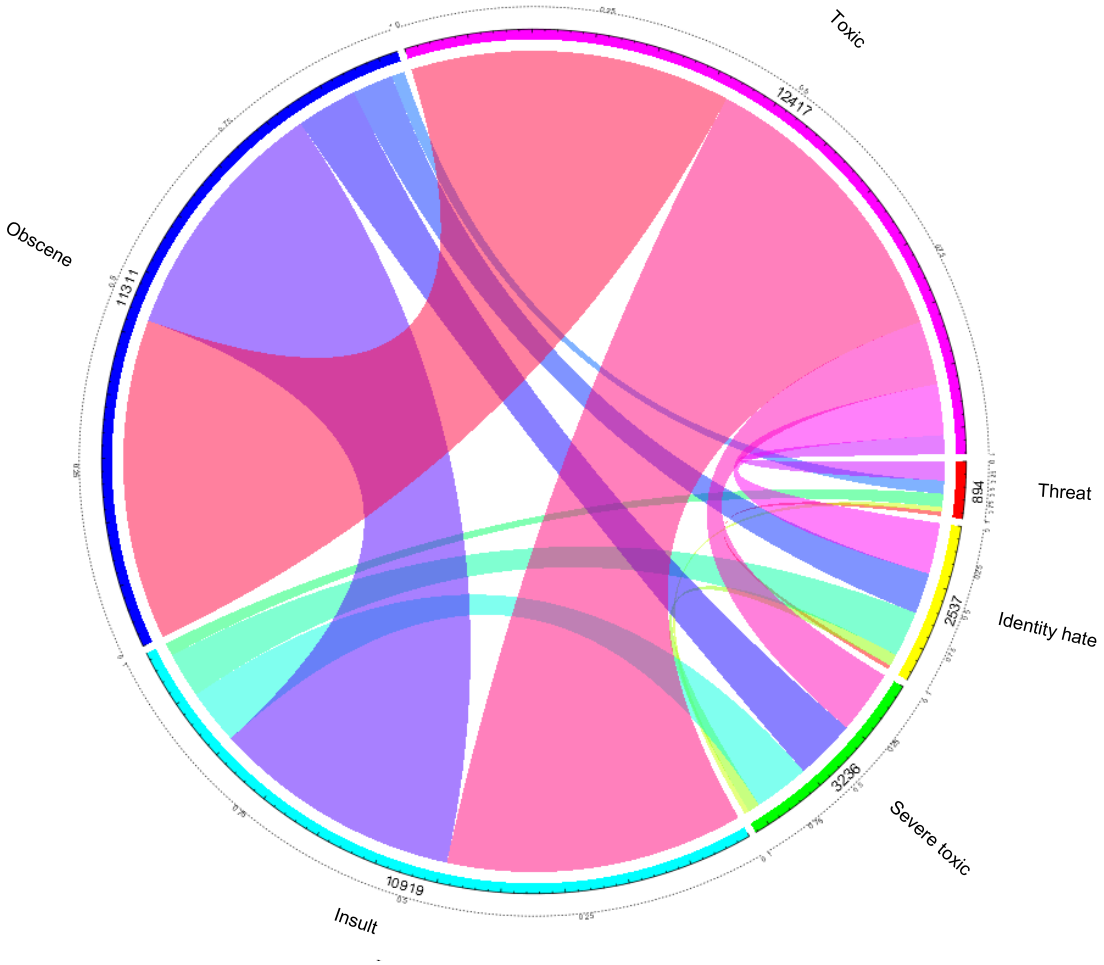


Figure 5.12: Concurrence plot of mldr-object

We also construct a concurrence plot of our mldr-object. This concurrence plot is given in Figure 5.12. Since the Toxic, Obscene and Insult labels are present for the most comments, compared to the other three labels, these three labels have the longest arcs in Figure 5.12. The bands that connect these three labels are also very wide. Recall that the width of a band connecting two arcs is proportional to the number of instances in which the corresponding two labels appear together. Therefore, since one of the most common labelsets in the dataset is the labelset that has the Toxic, Obscene and Insult label present, it is no surprise that the bands connecting these three labels are wide. From Figure 5.12 it is also clear that the labels are not balanced. The Severe Toxic, Threat and Identity hate labels have relatively much shorter arcs than the three most common labels. We expect that it will be especially difficult to detect comments that have these three least common labels present.

5.3.7 Test data

Using the utiml package, our mldr-object will be used to fit several multi-label learning methods. We will use the fitted functions to classify our test observations, in order to evaluate the predictive performance of the models. Since the utiml package requires multi-label datasets to be expressed as mldr-objects, we also need to express our test data as an mldr-object.

At this stage our test data consists of 53190 raw comments. We need to organize the test data so that its structure resembles the training data structure. In other words, we need to clean the test comments as well, so that we can express the test data as a 53190×606 matrix. The rows of this matrix will correspond to the 53190 test comments and the columns will correspond to the $p = 606$ terms we identified. Therefore, each test observation will be a $p = 606$ component vector, where each component in the vector is the frequency of one of the p terms in the relevant test comment. Converting this matrix to an mldr-object will produce the required test mldr-object. In order to find this test mldr-object, we proceed as follows.

The 53190 raw test comments are imported into R as a vector with elements the respective comments. This test-vector is converted into a source-object using the function *VectorSource()*. Then the function *VCorpus()* is used to convert this source-object into a VCorpus. We proceed by cleaning the test corpus in a similar manner as we did the training corpus. Thus, functions *tolower()*, *stripWhitespace()*, *removePunctuation()* and *removeNumbers()* are used to convert all text to lower-case, remove extra white spaces, remove punctuation and remove numbers from the text documents. We replace rude words and their variations with the 104 codenames identified from the training data. We also use the two lists of synonyms formed when we inspected the training data in order to replace synonyms in the test corpus as well. We use *DocumentTermMatrix()* to form a DTM of the test corpus, specifying the terms of the DTM to be the 606 terms we identified using the training data. Finally, we convert the matrix to a dataframe after which the function *mldr_from_dataframe()* is used to form the test mldr-object.

5.3.8 Large data

Another considerable challenge of this multi-label data analysis, in addition to the sparsity of the labels, is the fact that the dataset is very large. Since the data analysis is executed in R on a non-scientific computer with CPU at 2.4 GHz and 8 GB ram, running time and memory is a big challenge. A strategy used to overcome this is to save different R workspaces for different stages of the analysis and ensure that each R workspace is as clean as possible, containing only the

objects needed for that stage of the analysis. This allows for more memory when saving DTMs, TDMs, mldr-objects and matrices.

In some cases, even when implementing the above strategy, a shortage of memory still occurs. A typical error message may resemble:

```
Error: cannot allocate vector of size 135.1 Gb
```

In cases like this it may be useful to use the R function *memory.limit()*. By using this function, we can manually increase the amount of memory used in R. The size is given in MB.

5.3.9 Results

Our training mldr-object consists of 95540 observations that have no labels present and 10841 observations that have at least one label present. Since running time is a concern and the labels in the dataset are so sparse, we form an mldr-object that consists of less comments for which none of the labels are present. This allows for a reduction in running time when models are fit to the mldr-object. Furthermore, we are also hopeful that training models on data not completely dominated by observations with no labels, might allow the models to detect infrequent comments that contain toxicity, more easily.

We randomly select 14159 of the 95540 comments with no labels and use all 10841 comments that have at least one label present, to form a smaller mldr-object containing 25000 observations. We fit several multi-label learning methods to the smaller mldr-object using utiml package functions. The models are used to classify all the observations in the test mldr-object. Due to the concern of memory and running time, LDsplit with trees is not fit to the data.

The predictive performance is evaluated using confusion matrices, constructed using the utiml package function, *multilabel_confusion_matrix()*. Confusion matrices are a useful tool to evaluate predictive performance in our analysis, since we are able to easily compare TP, FP, FN and TN rates for all the fitted models. These rates can also be expressed per label, so that we can identify which models struggle to classify which labels.

We report the results found when fitting a label powerset model with a decision tree as base learner, as well as two ensemble of classifier chains models. The first ensemble of classifier chains model uses a decision tree as base learner and the second uses random forests. The random forest grows 6 trees and randomly samples 24 predictors as candidates for each split. Both the ensemble of classifier chains models are based on 10 random orderings of the labels

and the percentage of training observations used for each classifier is 90%, while 100% of the predictors are used. A decision tree is an effective base learner since the learning method has no trouble handling the sparsity of the training data.

No R utilities are available to fit a random forest of predictive clustering trees. Also, since Tsoumakas *et al.* (2011) suggests that RAkEL achieves better results for datasets with a large number of labels, we do not report the results when fitting a RAkEL model.

For each of the three fitted functions, we apply two different threshold strategies in order to obtain multi-label classifications of the test observations. The first is a global threshold of 0.5. In other words, for all the labels, when the posterior probability of the label is larger than or equal to 0.5, the label is classified as present, else it is not. Using this strategy, it appears that the fitted functions classify the Toxic label present while all the other labels are classified as absent, for most of the observations. Therefore, our second threshold strategy is label-based and we investigate the effect of an extreme threshold strategy. In this case, if the posterior probability of the Toxic label is smaller than 1, the Toxic label is classified as absent. Only when the posterior probability of the Toxic label is exactly equal to 1, is the Toxic label classified as present. For all five of the other labels, Severe Toxic, Obscene, Threat, Insult and Identity hate, if the posterior probability of the label is greater than 0, the label is classified as present, else it is absent.

For each learning method the confusion matrices found when applying both threshold strategies are given in Figure 5.13. The label-based confusion matrices are also constructed for each of the three learning methods. These matrices are given in Figure 5.14, Figure 5.16 and Figure 5.18, when applying the global threshold, and in Figure 5.15, Figure 5.17 and Figure 5.19, when applying the label-based threshold.

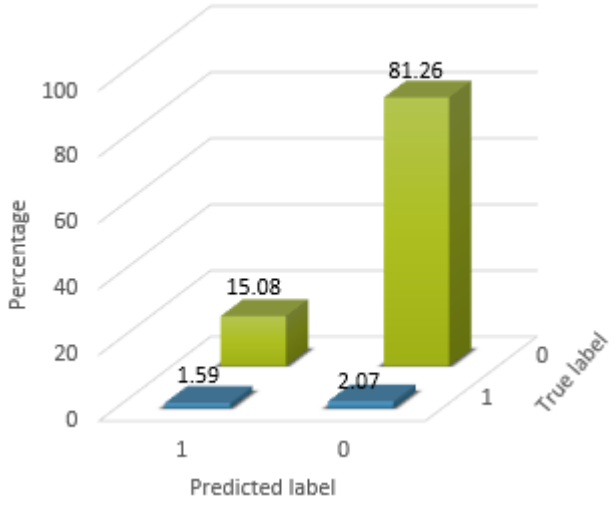
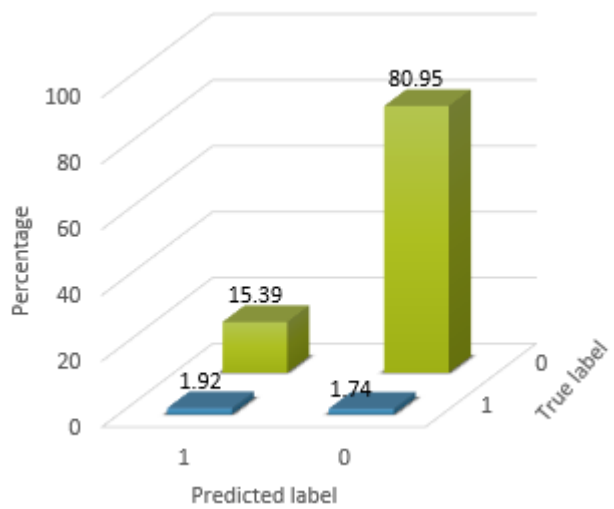
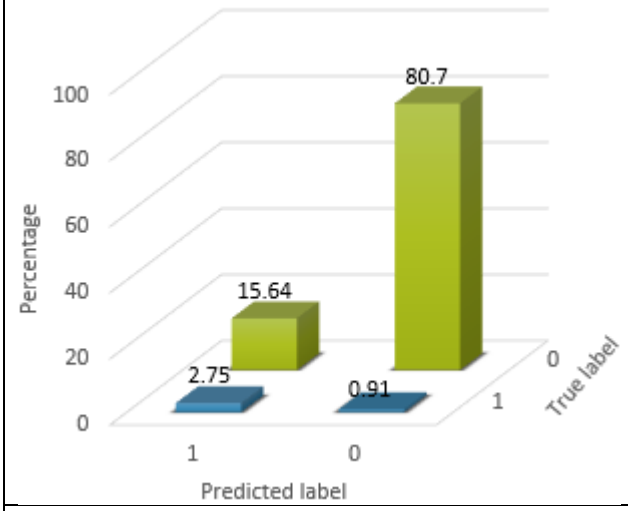
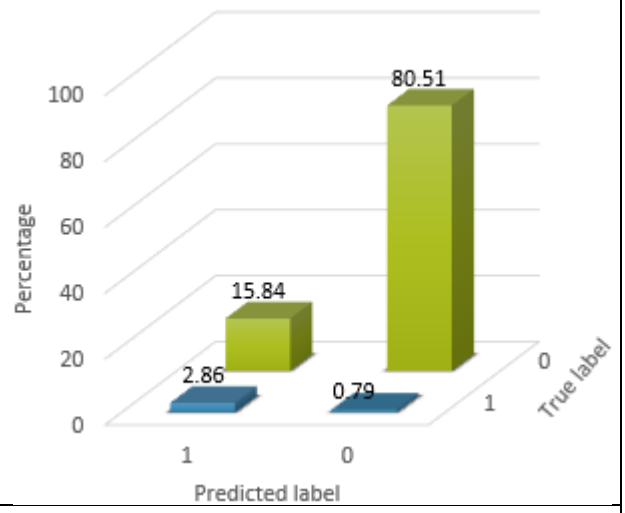
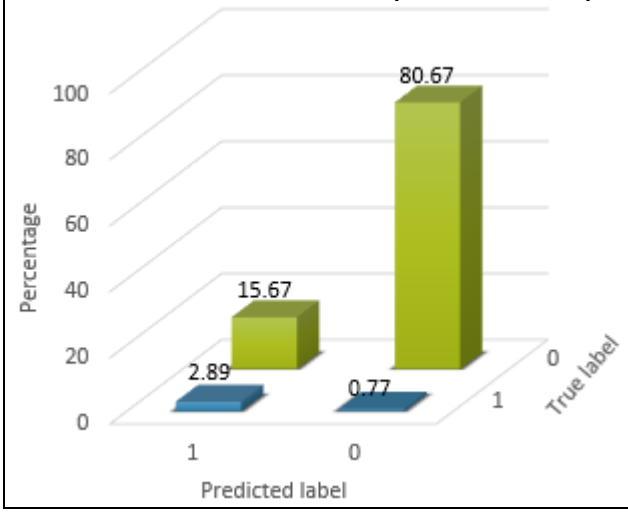
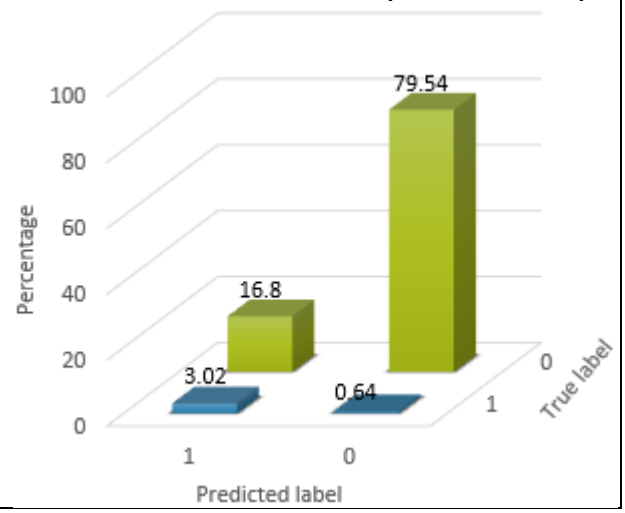
Global threshold	*Label-based threshold*
Label powerset (Tree)	**Label powerset (Tree)**
	
Ensemble of classifier chains (Tree)	**Ensemble of classifier chains (Tree)**
	
Ensemble of classifier chains (Random Forest)	**Ensemble of classifier chains (Random Forest)**
	

Figure 5.13: Confusion matrices per learning method

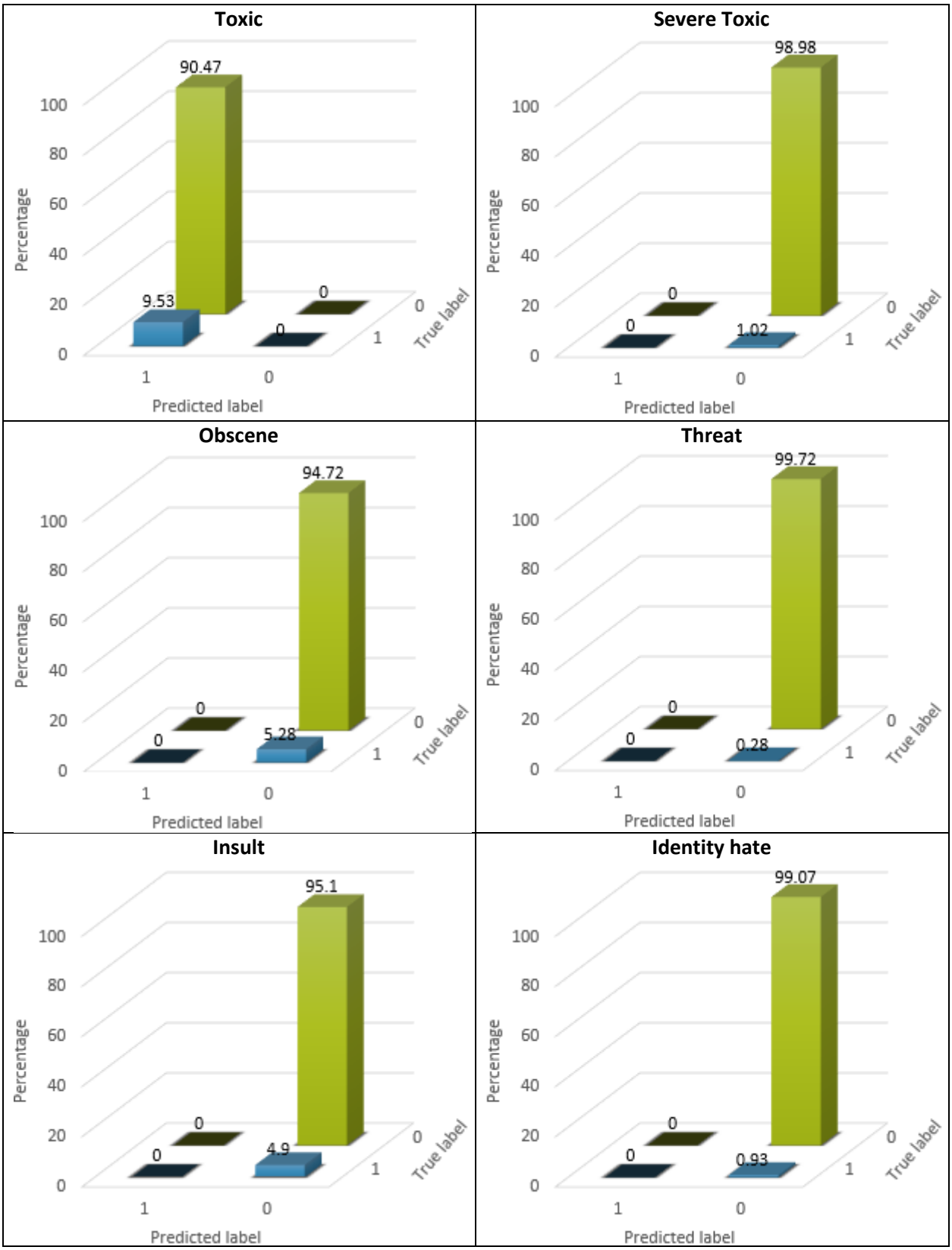


Figure 5.14: Label powerset with global threshold

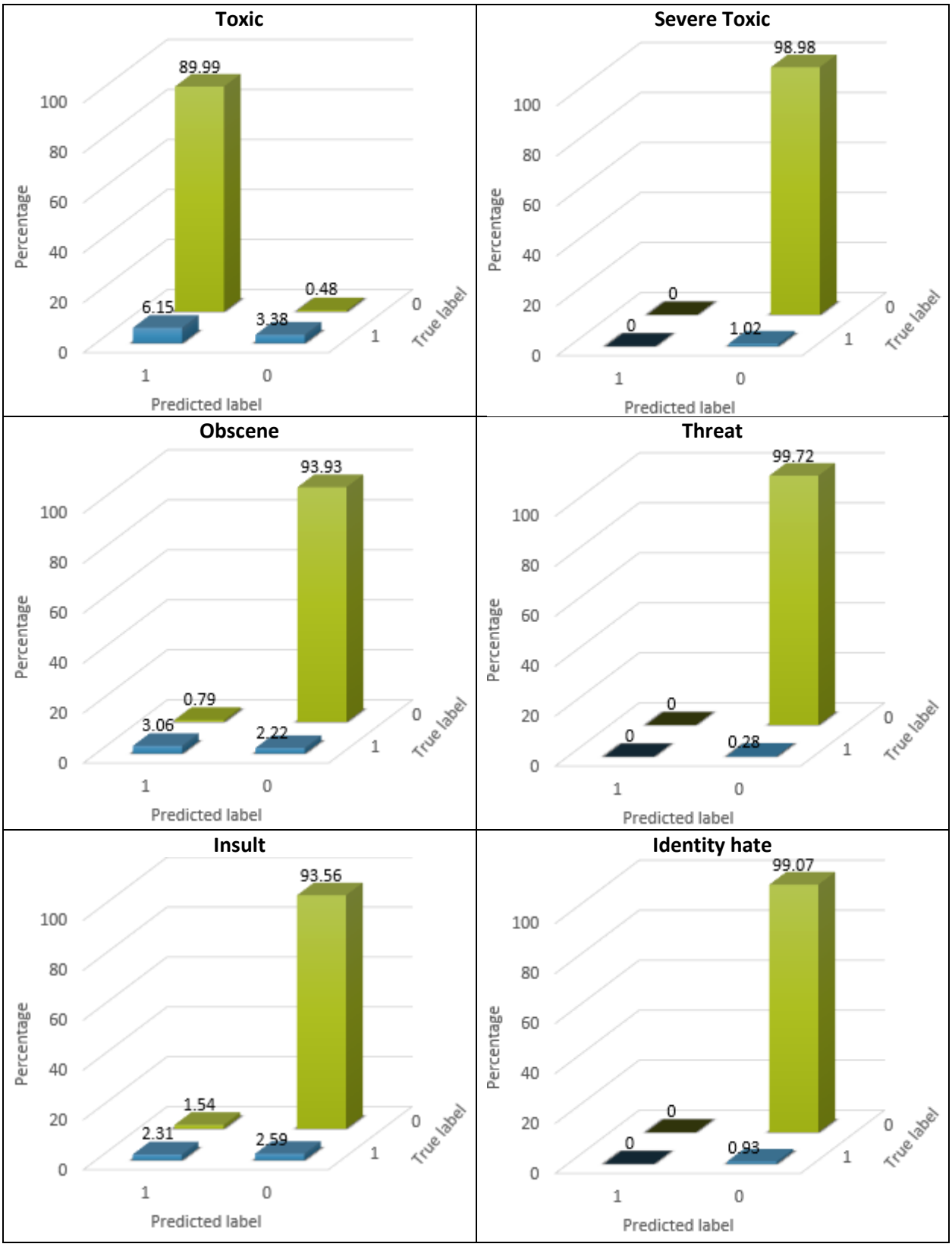


Figure 5.15: Label powerset with label-based threshold

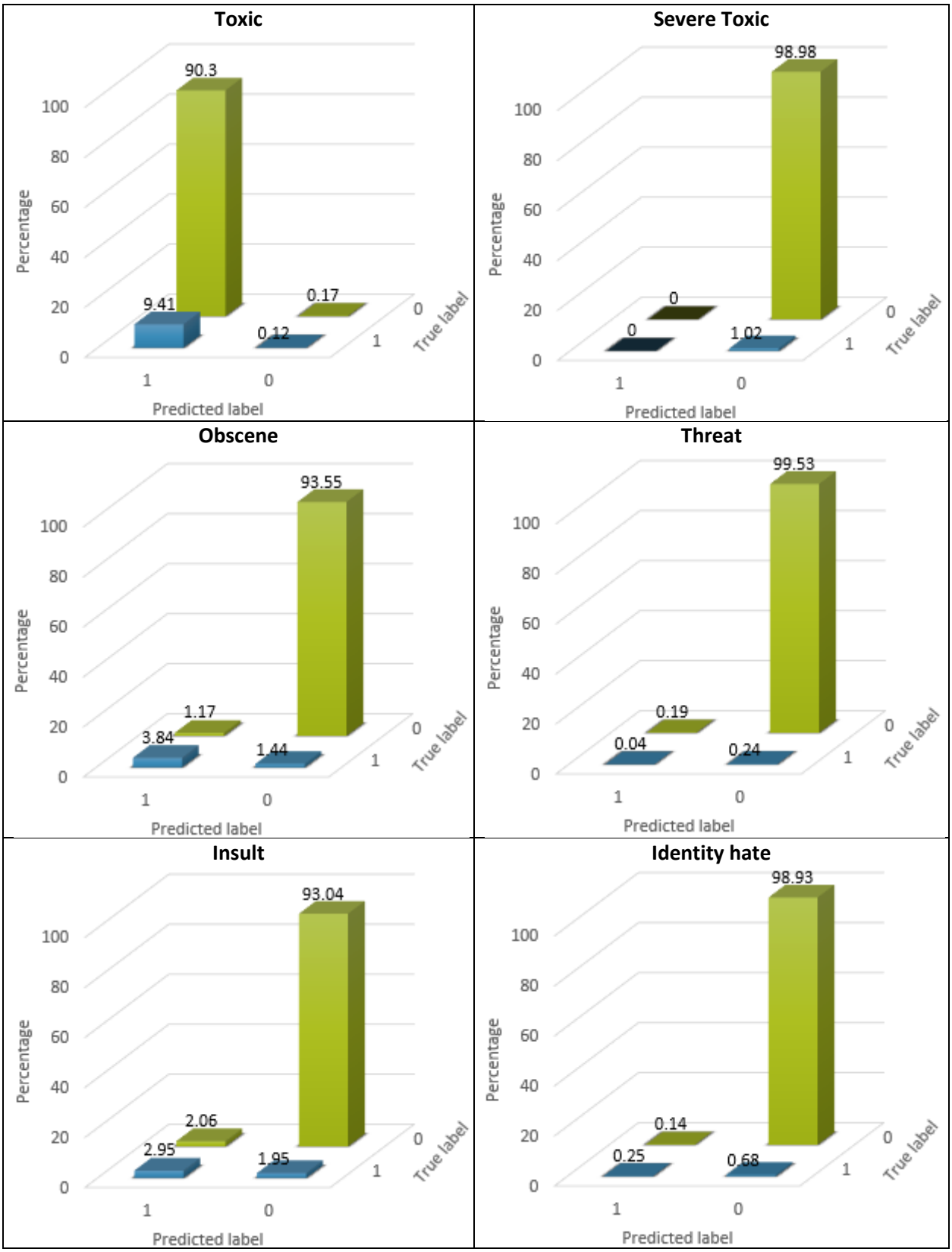


Figure 5.16: Ensemble of classifier chains (Tree) with global threshold

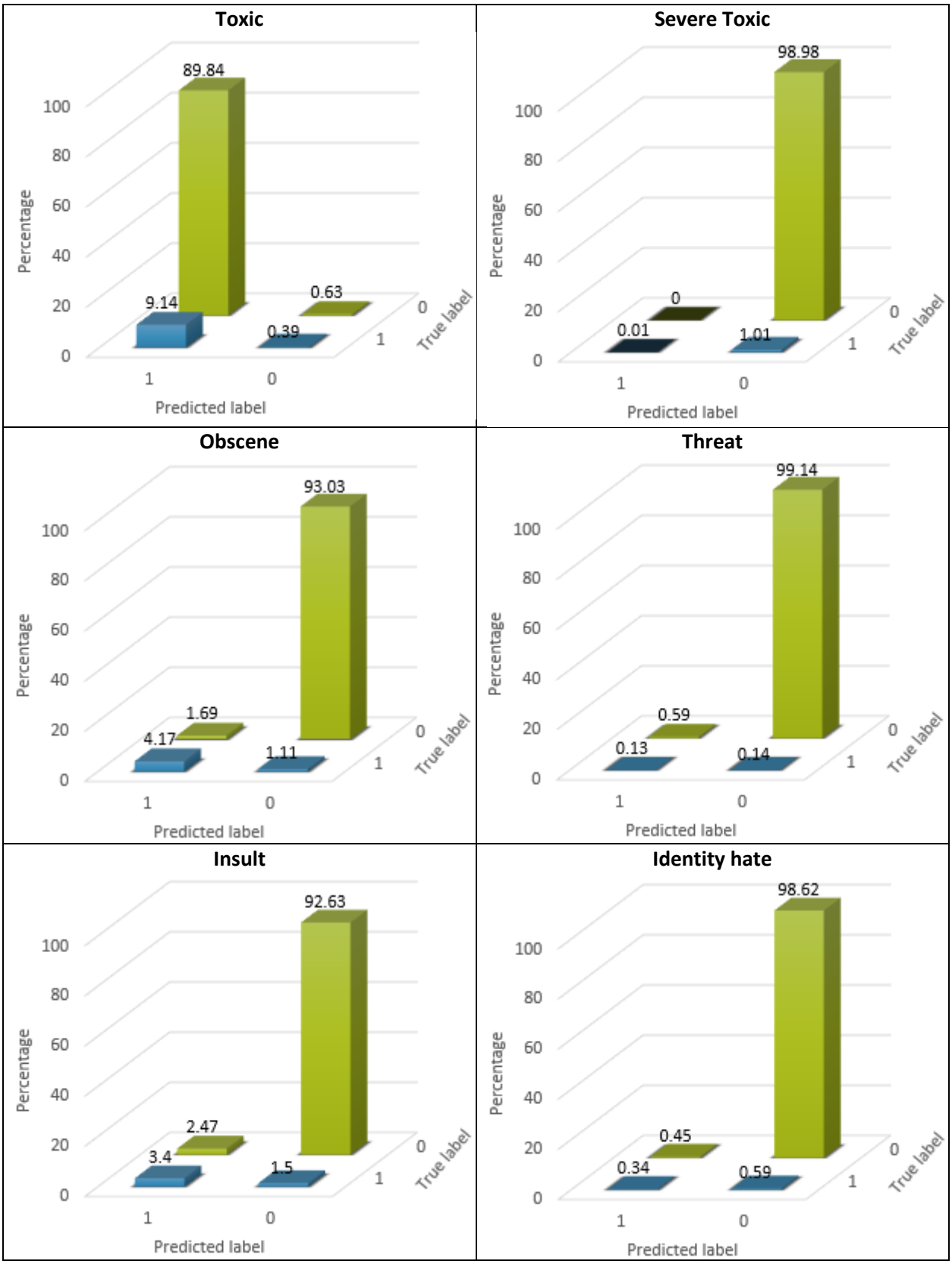


Figure 5.17: Ensemble of classifier chains (Tree) with label-based threshold

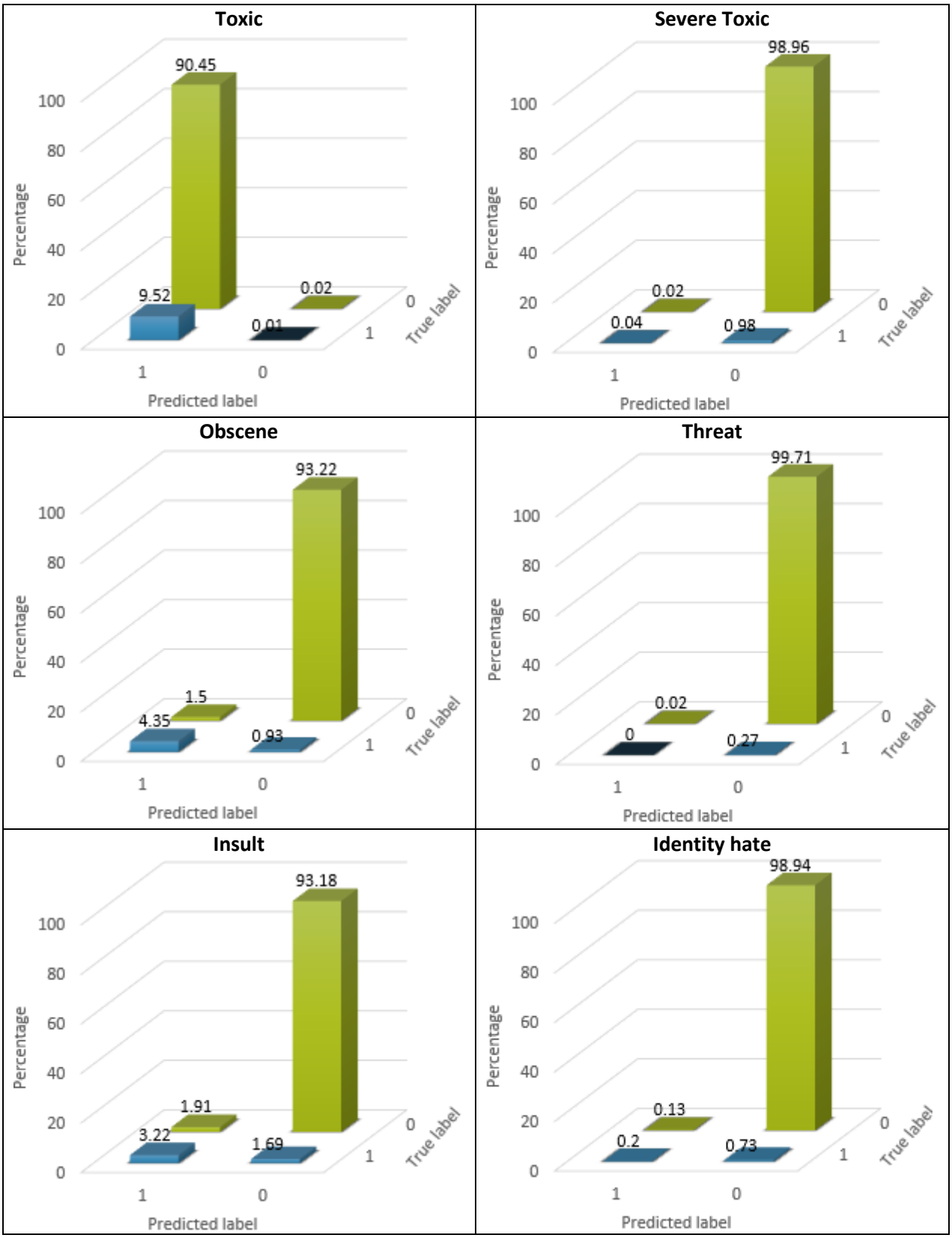


Figure 5.18: Ensemble of classifier chains (Random Forest) with global threshold

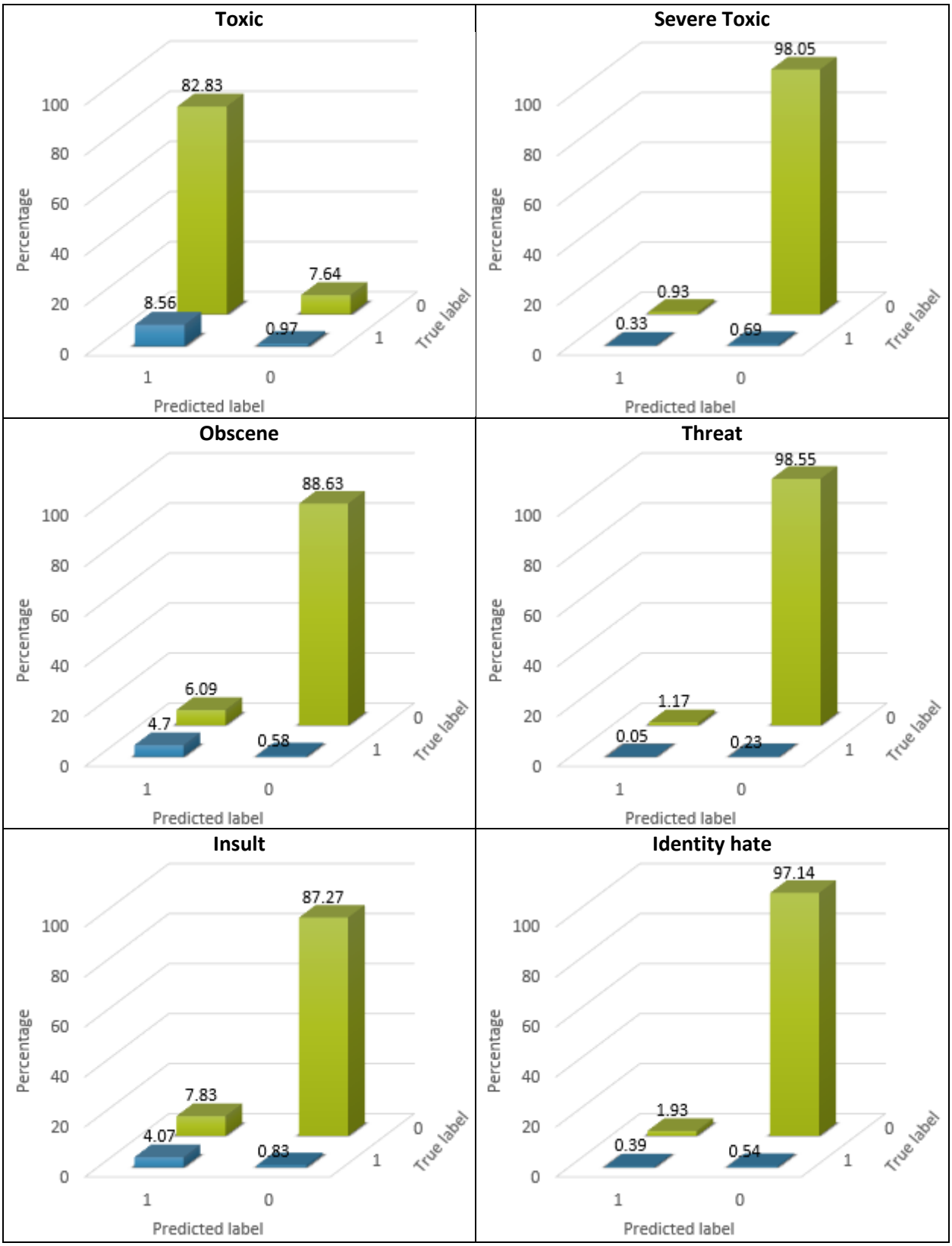


Figure 5.19: Ensemble of classifier chains (Random Forest) with label-based threshold

From Figure 5.14 we see that applying the global threshold to the label powerset method causes the model to classify the Toxic label present for all the test observations, while all the other labels are classified as absent. Therefore, in Figure 5.13, the TP rate of the corresponding model will be equal to the percentage of observations in the test data that in truth have the Toxic label present. Since the model simply classifies all observations to one labelset, regardless, it does not aid us in detecting toxicity in comments. Using the label powerset method with the label-based threshold improves performance slightly, but not significantly. Figure 5.15 shows that the model continues to classify the three sparsest labels, namely Severe Toxic, Threat and Identity hate, absent for all the test observations. The model does classify the Obscene and Insult labels to be present for some observations and classifies the Toxic label to be absent for some observations. The results however remain undesirable.

In general, when examining all the label-based confusion matrices, it appears that all the models seldomly classify the Toxic label absent. For all the label-based confusion matrices reported, the FP rates are high. This may be a consequence of the fact that the Toxic label is the most common label in the training data. It does however also raise the question whether this label is well defined. Since the comments are individually labeled by human raters, the question arises whether it is clearly defined when exactly a comment should be labelled as Toxic. Our focus therefore shifts to the performance of the models specifically when classifying the other five labels.

The performance of the ensemble of classifier chains model with a random forest base learner seems promising. Figure 5.18 shows that when using the global threshold, the model is able to correctly identify the Obscene and Insult labels present for some of the test observations. Only 5.28% of the test observations in truth have the Obscene label present. The TP rate of the model is reported as 4.35 in Figure 5.18 for the Obscene label. In other words, approximately 82% of the observations that in truth have the Obscene label present, are correctly identified by the model. The Insult label is in truth present for approximately 4.9% of the test observations. The TP rate of the model is reported as 3.22 in Figure 5.18 for the Insult label. Therefore, approximately 66% of the test observations that have the Insult label present, are correctly identified by the model. The three sparsest labels of the dataset, however, remain difficult to classify. Figure 5.18 indicates that even though the model classifies the three sparsest labels to be present for some observations, these classifications are seldom correct. The reported TP rates are low.

By using the label-based threshold for this ensemble of classifier chains model, we manage to increase the TP rates of the three sparsest labels slightly, as shown in Figure 5.19. The TP rates also increase for the Obscene and Insult labels. However, along with these increases in TP rates comes increases in FP rates as well. Since our goal is to detect toxicity in online comments, the cost of having a slight increase in FP rates in order to increase TP rates, may be a cost we can bare. A false identification of toxicity, in which case the comment does not contain toxicity in truth, but we classify it as containing toxicity, might be more favourable than the case where a comment in fact contains toxicity, but slips through unidentified. We may prefer viewing the classification of comments as a flagging process. In this case, if our model classifies a comment to contain toxicity, this in fact flags the comment. Flagged comments can then be reviewed by website owners in order to determine whether the comments are acceptable or not.

In Figure 5.13, examining the results of the ensemble of classifier chains model using a decision tree as base learner, we see that the behaviour of this model is similar to that of the ensemble of classifier chains model that uses a random forest as base learner, with only slight differences. Comparing the label-based confusion matrices for the two different base learners, we observe the following.

With the decision tree base learner and using the global threshold, Figure 5.16 shows that the model never classifies the Severe Toxic label present for any observations. Even when using the label-based threshold, in Figure 5.17, the Severe Toxic label is classified absent for almost all the test observations. This is not true for the random forest base learner. Using both threshold strategies, the random forest base learner classifies the Severe Toxic label present for some observations.

With the decision tree base learner, using both threshold strategies, the Obscene label is classified present for a smaller number of observations than when using a random forest base learner. The TP rates are also lower when using the decision tree base learner, compared to the TP rates of the random forest base learner. However, for the random forest base learner, the FP rates are higher. A similar conclusion may be made for the Insult label. As mentioned, a slight increase in FP rates may be a cost we can bare in order to obtain an increase in TP rates. Furthermore, it appears that the two base learners perform similarly when classifying the Identity hate label. Both base learners struggle to identify those observations for which this sparse label is present.

Interestingly, for the Threat label the decision tree base learner seems to produce better classifications than the random forest base learner. Using the global threshold, the random forest base learner classifies the Threat label absent for almost all observations. This is not true for the decision tree base learner. Using the label-based threshold, the decision tree base learner produces the highest TP rate, higher than any other TP rate produced for the Threat label. Not only is the TP rate higher when using the decision tree base learner, but this model also produces a lower FP rate than the random forest base learner using the label-based threshold.

Recall that the Threat label is the sparsest label in the training dataset. Furthermore, only 0.17% of the test observations in truth have the Threat label present. The decision tree base learner with the label-based threshold is able to correctly identify almost half of these infrequent observations. It is difficult to explain why this is the case. It may be that the presence of two or three key words in a comment, immediately causes that comment to be considered a threat. Examples include words such as “will”, “kill” and “die”, as seen in the word cloud of the Threat label given in Figure 5.8. It might be that a simple decision tree base learner is more sensitive to such a characteristic than a random forest base learner.

In conclusion, considering the performance on all six labels, it appears that the overall most promising model is the ensemble of classifier chains model using a random forest as base learner. If we view the classification of comments as a flagging process, where a slight increase in FP rates are acceptable, we suggest using the label-based threshold. When classifying the Threat label, however, we suggest using the ensemble of classifier chains model with a decision tree as base learner and the label-based threshold.

This chapter highlights the importance of a pre-processing step in a multi-label text data analysis. For text data analyses, it is not unusual for the pre-processing and cleaning of the data to be one of the most time-consuming steps in the analysis. If it is however not done thoroughly, predictive performance may suffer. The next chapter provides concluding remarks as well as opportunities for future research. Possible aspects which can be improved, regarding the practical analysis given in this chapter, are also presented.

CHAPTER 6: CONCLUSION AND OPPORTUNITIES FOR FUTURE RESEARCH

There are many scenarios where several labels may be associated simultaneously with each data case in a dataset. Therefore, a large number of multi-label datasets are found in a variety of domains, as also seen in this thesis where a text domain dataset was a focus. In this thesis the multi-label analysis goal was multi-label classification. Therefore, the work presented entailed a detailed study of the literature on multi-label classification, with a particular focus on multi-label ensemble methods. We also proposed a new multi-label ensemble method that can be used to perform multi-label classification. In this final chapter we give an overview of the work presented in the previous chapters and discuss opportunities for future research.

The first three chapters of this thesis provided a theoretical background of multi-label classification. We highlighted the differences between multi-label classification and binary or multi-class classification, also discussing unique characteristics of the multi-label structure. Since the multi-label structure is unique, it requires specific tools for analysis, including specific evaluation measures of predictive performance. Considering the three categories of multi-label learning, we also presented some multi-label learning methods as proposed in the literature.

In Chapter 4 we presented a new multi-label ensemble method, named LDsplit with trees. We compared the predictive performance of LDsplit with trees to that of previously defined multi-label learning methods, by performing an empirical study on benchmark datasets. LDsplit with trees produced promising results allowing us to believe that with further modifications the procedure may become a competitive multi-label learning method. We also compared LDsplit with trees to other multi-label ensemble methods, in particular. LDsplit with trees is different from ensembles of classifier chains, RAkEL and random forests of predictive clustering trees. The approach differs, since it constructs an ensemble of tree-structures by considering different permutations of the labels in a multi-label dataset. Each tree-structure is constructed for a given permutation in a manner that incorporates label dependencies. Furthermore, each split of a node in a tree-structure is performed by considering a binary classification problem.

In LDsplit with trees we use a stump as binary classifier in order to split the data of a node. A stump is useful if we aim to perform slow learning. The principle of slow learning lies in the idea that if a node is split into more than two branches, important information may go by unnoticed. Small modifications should be made to the data with each split in order to “learn slowly”. In future

studies, however, we may also consider using other binary classifiers, instead of stumps, in order to split the data of a node.

Applying LDsplit with trees, using a stump as the binary classifier, an optimal splitting variable and split point are identified in order to split a node in a tree-structure. In future work we may explore the possibility of using these splitting variables to obtain information regarding variable importances. Those variables used for multiple splitting rules in multiple tree-structures may for example be regarded as more important than variables that are seldomly or never used in splitting rules. Can we somehow develop a variable importance measure for LDsplit with trees?

Furthermore, for a tree-structure, T_j, the nodes that make up Level k are found by fitting stumps to the binary classification problems formed by considering the data in the respective nodes at Level $k-1$, along with label $P_{j,k}$. Therefore, the splitting variables used to split the nodes of Level $k-1$ may be correlated with label $P_{j,k}$. Thus, it may be possible to explore variable and label correlation in future research. If the same variables are often associated with splits corresponding to a certain label, we might conclude that these variables are highly correlated with the specific label. The research idea raised in the previous paragraph therefore considers aspects of global importance of predictors, whereas we consider aspects of local importance in this paragraph.

LDsplit with trees is a multi-label ensemble method since it combines the output obtained from several multi-label tree-structures in order to obtain a final multi-label classification. As described in Chapter 4, in order to obtain a final multi-label classification, we use a majority vote or average the posterior probabilities of the labels produced by the respective tree-structures. An opportunity for future research may be to combine the output of the respective tree-structures in a different way, for example by using an approach similar to boosting.

In the case of boosting, a base procedure is refitted multiple times to different forms of reweighted data, forming multiple function estimates, $\hat{g}^{[1]}(.)$, $\hat{g}^{[2]}(.)$, ..., $\hat{g}^{[M]}(.)$. An ensemble of the function estimates is obtained in the form $\sum_{j=1}^{M} \alpha_j \hat{g}^{[j]}(.)$ (Bühlmann and Hothorn, 2007:478). Each function estimate therefore has a corresponding weight when contributing to the final prediction. Take the AdaBoost algorithm, developed by Robert Schapire and Yoav Freund for binary classification, for example. The AdaBoost algorithm fits a base classifier to reweighted versions of the observed training data. After fitting the base classifier, the predicted classes and true

classes can be compared in order to find a weighted misclassification error, err_j. Then α_j is computed as $\alpha_j = \log\left[\frac{1-err_j}{err_j}\right]$. In the final classification the weights give a larger contribution to those classifiers that result in lower training misclassification errors. In future work we may be able to define such a contributing weight, α_j, for the LDsplit with trees model. In this case all tree-structures will no longer make the same contribution to the final multi-label classification. Instead we would like to allow tree-structures that result in lower training misclassifications to contribute more to the final multi-label classification.

In Chapter 5, a practical multi-label text data analysis was presented. We highlighted important aspects when such an analysis is carried out in R. We also emphasised the importance of cleaning and pre-processing of text data. Considerable challenges faced in the multi-label data analysis were the sparsity of the labels as well as the size of the dataset. We compared the confusion matrices obtained when fitting three different multi-label learning methods, applying both a global and label-based threshold. The sparsest labels proved to be the most difficult to classify. Our chosen model fits an ensemble of classifier chains with a random forest base-learner. The label-based threshold is used if the classification of comments is viewed as a flagging process. We did however also suggest using an ensemble of classifier chains model with a decision tree as base classifier when classifying the sparsest label.

In future work we may wish to improve the results of the practical analysis presented in Chapter 5. One possible way this may be achieved is to improve the cleaning and pre-processing of the text data. The text data consist of informal Wikipedia comments. Therefore, the text was particularly challenging since we were confronted with rude terms, misspelt words as well as informal or slang words not found in dictionaries. We tried to address these issues as far as possible, as we described in Chapter 5. In a future analysis, we can however also address some additional issues. Table 6.1 outlines examples of such issues.

Table 6.1: Possible issues to address

Issue	Example in text
Words for which one or multiple letters are unnecessarily repeated	"foreverrrrrrrrrrrrrrrrrrr"
Using the spacebar between the letters of a word	"u g l y"
Using the number "0" as a "o"	"c00l"
Using symbols "@,#,&" within a word	"f@t"
Use of symbols to express emotion	"☺"

In Chapter 5 we also described how our test data were organized to resemble the training data structure. We did not take into consideration that there will most likely appear new terms in the test cases not found in any of the training cases. For example, there may be additional rude terms in the test data apart from the 104 rude terms identified when considering the training data. It may be that the presence of such a new rude term would be a strong indication of the presence of toxicity. Since we simply disregard terms not found in the training data, this may be an issue that can be addressed in future studies.

The bag-of-words approach was used to organize the text data. It may also be advantageous to use a more complex approach, such as semantic parsing, in a future analysis. This would take word type and order into consideration as well.

If the text data analysis is carried out on a scientific computer, we would be able to alleviate the concern of memory and running time. In this case the multi-label learning methods can for example be fit to the full set of training observations. We would also be able to specify the number of cores to parallelize the training by using the "cores"-argument when fitting a utiml package function. Furthermore, LDsplit with trees can also be fit to the text data if memory and running time is no longer a concern.

Finally, many of the comments in the practical dataset do not contain toxicity, so that the labels of the dataset are very sparse. One of the strategies used in Chapter 5 in order to increase the ability of the multi-label classifiers to identify the infrequent comments containing toxicity, was to fit the multi-label learning methods to a training mldr-object containing less comments that are free of toxicity. In future studies we can also explore other strategies to address the sparsity of the labels. For example, we may use the Synthetic Minority Over-sampling Technique (SMOTE). In general, the SMOTE approach under-samples the majority class of a dataset while

simultaneously performing a special over-sampling technique of the minority class (Chawla *et al.*, 2002). Using such an approach in our multi-label data setting could be beneficial.

Overall, this thesis illustrates how predictive performance of a multi-label base classifier can be improved using an ensemble of multi-label classifiers. Modern multi-label datasets, such as the toxic comment classification dataset, are becoming increasingly available. Multi-label ensemble methods that can be used in such analyses are therefore of substantial use.

REFERENCES

Bouchet-Valat, M. (2014). SnowballC: Snowball Stemmers based on the C libstemmer UTF-8 library. R package version 0.5.1. Available: https://cran.r-project.org/web/packages/SnowballC/index.html.

Bühlmann, P. and Hothorn, T. 2007. Boosting Algorithms: Regularization, Prediction and Model Fitting. *Statistical Science*, 22(4):477-505.

Charte, D. and Charte, F. (2017). mldr.datasets: R ultimate multilabel dataset repository. R package version 0.4.0. Available: https://cran.r-project.org/web/packages/mldr.datasets/index.html.

Charte, D. and Charte, F. (2018). mldr: Exploratory data analysis and manipulation of multi-label data sets. R package version 0.4.0. Available: https://cran.r-project.org/web/packages/mldr/index.html.

Charte, D and Charte F. D. (2018). Working with multilabel datasets in R: the mldr package [Online]. Available: https://cran.r-project.org/web/packages/mldr/vignettes/mldr.pdf.

Chawla, N.V., Bowyer, K.W., Hall, L.O. and Kegelmeyer, W.P. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321-357.

DataCamp: Text mining: Bag of words [Online]. 2018, February 21. Available: https://www.datacamp.com/courses/intro-to-text-mining-bag-of-words. [2018, March 3].

Feinerer, I. (2018). Introduction to the tm Package – Text Mining in R [Online]. Available: https://cran.r-project.org/web/packages/tm/vignettes/tm.pdf.

Feinerer, I. and Hornik K. (2018). tm: Text Mining Package. R package version 0.7-4. Available: https://cran.r-project.org/web/packages/tm/index.html.

Fellows, I. (2014). wordcloud: Word Clouds. R package version 2.5. Available: https://cran.r-project.org/web/packages/wordcloud/index.html.

Goodrich, B., Kurkiewicz, D. and Rinker, T. (2018). qdap: Bridging the gap between qualitative data and quantitative analysis. R package version 2.3.0. Available: https://cran.r-project.org/web/packages/qdap/index.html.

Hastie, T., Tibshirani, R. and Friedman, J. 2009. *The Elements of Statistical Learning: Data Mining, Inference and Prediction (Second edition)*. Springer.

Kaggle: Google Cloud & YouTube-8M video understanding challenge [Online]. 2017, June 3. Available: https://www.kaggle.com/c/youtube8m. [2018, June 14].

Kaggle: Greek media monitoring multi-label classification (WISE 2014) [Online]. 2014, July 16. Available: https://www.kaggle.com/c/wise-2014. [2018, June 13].

Kaggle: Movie genre from its poster [Online]. 2018, May 15. Available: https://www.kaggle.com/neha1703/movie-genre-from-its-poster. [2018, June 13].

Kaggle: Question-Answer dataset [Online]. 2017, September 28. Available: https://www.kaggle.com/rtatman/questionanswer-dataset/home. [2018, June 26].

Kaggle: Random sample of NIH chest X-ray dataset [Online]. 2017, November 23. Available: https://www.kaggle.com/nih-chest-xrays/sample. [2018, June 13].

Kaggle: Sentiment analysis on movie reviews [Online]. 2015, March 1. Available: https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews. [2018, June 26].

Kaggle: Toxic comment classification challenge [Online]. 2018, March 21. Available: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge. [2018, February 20].

Kocev, D., Vens, C., Struyf, J. and Džeroski, S. 2007. Ensembles of multi-objective decision trees. *Proceedings of the 18th European Conference on Machine Learning*, 624-631.

Madjarov, G., Kocev, D., Gjorgjevikj, D. and Džeroski, S. 2012. An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition*, 45:3084-3104.

Read, J., Pfahringer, B., Holmes, G. and Frank, E. 2009. Classifier chains for multi-label classification. *Joint European Conference on Machine Learning and Knowledge Discory in Databases*. Springer: 254-269.

Read, J., Pfahringer, B., Holmes, G. and Frank, E. 2011. Classifier chains for multi-label classification. *Machine Learning*, 85:333-359.

Rivolli, A. (2016). utiml: Utilities for multi-label learning [Online]. Available: https://cran.r-project.org/web/packages/utiml/vignettes/utiml-overview.pdf.

Rivolli, A. (2018). utiml: Utilities for multi-label learning. R package version 0.1.4 Available: https://cran.r-project.org/web/packages/utiml/index.html.

Robin, X., Turck, N., Hainard, A., Tiberti, N., Lisacek, F., Sanchez, J.C. and Müller, M. 2011. pROC: an open-source package for R and S+ to analyse and compare ROC curves. *BMC bioinformatics*, 12(1):77.

Sandrock, T. and Steel, S.J. 2017. An algorithm for generating multi-label classification data. Technical report, Stellenbosch University.

Sechidis, K., Tsoumakas, G. and Vlahavas, I. 2011. On the stratification of multi-label data. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer: 53-59.

Spolaôr, N., Cherman, E.A., Monard, M.C. and Lee, H.D. 2013. A comparison of multi-label feature selection methods using the problem transformation approach. *Electronic Notes in Theoretical Computer Science*, 292:135-151.

STHDA: Text mining and word cloud fundamentals in R: 5 simple steps you should know [Online]. 2018, February 21. Available: http://www.sthda.com/english/wiki/text-mining-and-word-cloud-fundamentals-in-r-5-simple-steps-you-should-know. [2018, March 3].

Tomás, J.T., Spolaôr, N., Cherman, E.A. and Monard, M.C. 2014. A framework to generate synthetic multi-label datasets. *Electronic Notes in Theoretical Computer Science*, 302:155-176.

Tsoumakas, G. and Katakis, I. 2007. Multi-label classification: an overview. *International Journal of Data Warehousing and Mining*, 3(3):1-13.

Tsoumakas, G., Katakis, I. and Vlahavas, I. 2008. Effective and efficient multilabel classification in domains with large number of labels. *Proceedings of ECML/PKDD 2008 Workshop on Mining Multidimensional Data*, 21:53-59.

Tsoumakas, G., Katakis, I. and Vlahavas, I. 2010. Mining multi-label data. *Data Mining and Knowledge Discovery Handbook*. Springer: 667-685.

Tsoumakas, G., Katakis, I. and Vlahavas, I. 2011. Random k-labelsets for multilabel classification. *IEE Transactions on Knowledge and Data Engineering*, 23(7):1079:1089.

Wu, X.Z. and Zhou, Z.H. 2016. A unified view of multi-label performance measures. *arXiv preprint arXiv: 1609.00288.*

APPENDIX A: Chapter 4

A.1 FUNCTIONS FOR LDSPLIT WITH TREES

Pfitstump()

```
function (x,y)
{
#########################################################################
# This function fits a stump to a binary classification dataset.
# The inputs are:
#       x = the vector of input values
#       y = a vector of 0's and 1's indicating group membership
# The output from the function is:
#       splitpmin = the optimal value on which to split
#       critmin = the minimum value of the error measure
#       yhat = the vector of predicted group memberships for the training data
#########################################################################

 n = length(x)
 splitp = rep(0,(n-1))
 crit = rep(0,(n-1))
 yhat = rep(0,n)
 yhatmat = matrix(0,nrow=n-1,ncol=n)
 xsort = sort(x, decreasing=FALSE, index.return=TRUE)
 indeks = xsort$ix
 ysort = y[indeks]

#############################################################
# The next loop cycles through the possible split points
#############################################################
 for (i in 1:(n-1)) {
    splitp[i] = mean(c(xsort$x[i],xsort$x[i+1]))
    ynul = sum(ysort[1:i]==0)
    yeen = sum(ysort[1:i]==1)
    if(ynul>yeen) {
       yhat[indeks[1:i]] = 0
       yhat[indeks[(i+1):n]] = 1  }
    if(ynul<yeen) {
       yhat[indeks[1:i]] = 1
       yhat[indeks[(i+1):n]] = 0  }
    if(ynul==yeen) {
       u = runif(1,0,1)
       if (u<0.5) {
          yhat[indeks[1:i]] = 0
          yhat[indeks[(i+1):n]] = 1  }
       if (u>=0.5) {
```

```
            yhat[indeks[1:i]] = 1
            yhat[indeks[(i+1):n]] = 0  }
                        }
    yhatmat[i,] = yhat

#####################################################
# Misclassification loss is used as splitting criterion
#####################################################

    crit[i] = sum(y!=yhat)/n  }
 critmin = min(crit)
 optsplit = which.min(crit)
 splitpmin = splitp[optsplit]
 yhat = yhatmat[optsplit,]

 uit = list(splitpmin,critmin,yhat)
 return(uit)
}
```

Pseekbestvar()

```
function(xmat,yvec)
{
#############################################################################
#
# This function finds the best single variable to split on from a set of p candidates.
# The inputs are:
#    xmat = the matrix of input values
#    yvec = a vector of 0's and 1's indicating group membership
# The output from the function is:
#    bestvar = the number of the best variable (the one minimising misclassification loss)
#    bestsplit = the value of bestvar on which we should split
#
#############################################################################

 n = length(yvec)
 p = ncol(xmat)
 critvec = rep(0,p)
 splitp = rep(0,p)
 for (j in 1:p) {
    x = xmat[,j]
    y = yvec
    uit = Pfitstump(x,y)
    critvec[j] = uit[[2]]
    splitp[j] = uit[[1]]
                }
```

```
 bestvar = which.min(critvec)
 bestsplit = splitp[bestvar]
 uit = list(bestvar,bestsplit)
 return(uit)
}
```

Psplitobs()

```
function(xvec,splitp)
{
#############################################################################
# This function splits the observations in a dataset according to a given splitting value.
# The inputs are:
#       xvec = the vector of input values
#       splitp = the x-value that should be used to split the data cases
# The output from the function is:
#       ygroup1 = the set of indices of the cases for which x < splitp
#       ygroup2 = the set of indices of the cases for which x >= splitp
#############################################################################

 n = length(xvec)
 ygroup1 = NULL
 ygroup2 = NULL
 tel1 = 0
 tel2 = 0
 for (i in 1:n) {
    if (xvec[i]<splitp) {
       tel1 = tel1+1
       ygroup1[tel1] = i }
    if (xvec[i]>=splitp) {
       tel2 = tel2+1
       ygroup2[tel2] = i }
                }
 uit = list(ygroup1,ygroup2)
 return(uit)
}
```

Pfitmodel()

```
function (Xmat, Ymat, yperm, minsize,m)
{
################################################################################
# This function fits the intended ML classification model to a given data set.
# The inputs are:
#       Xmat = the matrix of input values
#       Ymat = the matrix of 0-1 label vectors
#       yperm = the vector giving the order in which the labels should be fitted
#       minsize = the minimum number of cases for splitting to be allowed
#      *m: number of levels the tree should have
# The output from the function is:
#       splitvariables = the set of variables on which the splits were made
#       splitvalues = the corresponding set of splitting values for these variables
#       obsinnodes = a list containing the indices of the cases in all of the nodes
#
################################################################################
 n = nrow(Xmat)
 p = ncol(Xmat)
 K = ncol(Ymat)
 splitvariables = rep(0,(2^m)-1)
 splitvalues = rep(0,(2^m)-1)
 Ymatperm = matrix(0,nrow=n,ncol=m)
 for (k in 1:m) Ymatperm[,k] = Ymat[,yperm[k]]


xmat = Xmat
yvec = Ymatperm[,1]
uit1 = Pseekbestvar(xmat,yvec)
splitvariables[1] = uit1[[1]]
splitvalues[[1]] = uit1[[2]]
xvec = xmat[,uit1[[1]]]
uit2 = Psplitobs(xvec,uit1[[2]])
obsinnodes = list(1:n, uit2[[1]], uit2[[2]])
```

```
for (k in 1:(m-1)) {
   yvec = Ymatperm[,k+1]
   beginn = 2^k
   eindig = 2^(k+1) -1
   for (t in beginn:eindig) {
      indekse = obsinnodes[[t]]
      if (length(indekse)>minsize) {
         uit1 = Pseekbestvar(xmat[indekse,],yvec[indekse])
         splitvariables[t] = uit1[[1]]
         splitvalues[t] = uit1[[2]]
         xvec = xmat[indekse,uit1[[1]]]
         uit2 = Psplitobs(xvec,uit1[[2]])
         obsinnodes = list.append(obsinnodes,obsinnodes[[t]][uit2[[1]]],obsinnodes[[t]][uit2[[2]]])
                                                  }

       if (length(indekse)<=minsize) {
          splitvariables[t] = -1
          splitvalues[t] = -1
          obsinnodes = list.append(obsinnodes,-1,-1)
                                                        }
                                              }
                                    }
 uit = list(splitvariables,splitvalues,obsinnodes)
 return(uit)
}
```

Pperm()

```
function(avec)
{
#########################################################################
# This function generates all possible permutations of the (integer)
# elements of the input vector avec
#########################################################################

 n = length(avec)
 Amat = matrix(0,nrow=factorial(n),ncol=n)
 count = 1
 cvec = rep(0,n)
 Amat[count,] = avec


i = 0
while (i<n) {
   if (cvec[i+1]<i) {
```

```
        if (2*(i%/%2)==i) {
           temp = avec[1]
           avec[1] = avec[i+1]
           avec[i+1] = temp }

        if (2*(i%/%2)!=i) {
           temp = avec[cvec[i+1]+1]
           avec[cvec[i+1]+1] = avec[i+1]
           avec[i+1] = temp }

        count = count+1
        Amat[count,] = avec
        cvec[i+1] = cvec[i+1]+1
        i = 0   }

    if (cvec[i+1]>=i) {
        cvec[i+1] = 0
        i = i+1 }
    }

 return(Amat)
}
```

Pallcombs()

```
function (K,m)
{
#############################################################
# Function to find all possible m-permutations from a vector
# of 1 to K elements.
# Uses package combinat
# Inputs:
#   K: total number of labels
#   m: number of levels of each tree
# Output:
#   (K*(K-1)*...*(K-m+1)) * m matrix of all possibilities
#############################################################

labels<-c(1:K)
allcombn<-t(combn(labels,m))
Ctotal<-nrow(allcombn)
Ttotal<-prod(c((K-m+1):K))
percomb<-prod(c(1:m))        #m!
```

```
allpossible<-matrix(0,nrow=Ttotal,ncol=m)
allpossible[1:percomb,]<-Pperm(allcombn[1,])
if(Ctotal>1){
for(k in 1:(Ctotal-1)){
allpossible[((k*percomb)+1):(percomb*(k+1)),]<-Pperm(allcombn[(k+1),])}}

return(allpossible)
}
```

Findnodeprob()

```
function (Ymat,yperm,obslist,m)
{
##############################################
# Function to find the posterior probability
# of the labels relevant to each node.
#Inputs:
#   Ymat:the matrix of 0-1 label vectors
#   yperm:the vector giving the order in which the labels were fitted
#   obslist:a list containing the indices of the cases in all of the nodes
#   *m: number of levels the tree should have
#Outputs:
#    probs:vector of posterior probability of 1 of corresponding
#          label for each node, excluding the first node.
#
#####################################################


n<-nrow(Ymat)
K<-ncol(Ymat)

Ymatperm<-matrix(0,nrow=n,ncol=m)
for(k in 1:m){Ymatperm[,k]<-Ymat[,yperm[k]]}

probs<-rep(0,(2^(m+1))-1)

for(k in 1:m){
beginn<-2^k
eindig<-2^(k+1)-1

for(t in beginn:eindig){
indekse<-obslist[[t]]
```

```
if(sum(indekse)>0.5){
need<-Ymatperm[indekse,k]
ene<-sum(need)
aantal<-length(indekse)
probs[t]<-ene/aantal}

if(sum(indekse)<0.5){
probs[t]<--1}
}}

probs<-probs[-1]
return(probs)
}
```

Findterminal()

```
function (splitvar,m)
{
#############################################################################
## Function to find terminal nodes of a tree-structure
## Inputs:
##      splitvar: vector of variables split on (output of Pfitmod)
##      m: number of levels the tree has
## Output:
##      terminals:vector where each entry corresponds to the terminal node
##                for that particular path
##      pathindices: number of the terminal node entry in path
#############################################################################
paths<-Findpath(m)

terminals<-rep(0,2^m)
pathindices<-rep(0,2^m)

for(j in 1:(2^m)){

if( sum(which(splitvar[paths[j,]]<0)) >0.5){
terminals[j]<- paths[j,min(which(splitvar[paths[j,]]<0))]
pathindices[j]<-min(which(splitvar[paths[j,]]<0))}

if(sum(which(splitvar[paths[j,]]<0)) <0.5){
terminals[j]<- paths[j,m]
pathindices[j]<-m}

}
return(list(terminals=terminals,pathindices=pathindices))
}
```

Findpath()

```
function (m)
{
########################################
# Function to find all the possible paths
# Input: number of levels of a tree
########################################
paths<-matrix(0,nrow=2^m,ncol=m)

for(k in 1:m){
beginn<-2^k
eindig<-2^(k+1)-1
lengte<-2^k
use<-c(beginn:eindig)
veelvoud<-(2^m)/lengte

paths[1:veelvoud,k]<-rep(use[1],2^(m-k))
for(j in 2:lengte){
paths[((veelvoud*(j-1))+1):(veelvoud*(j)),k]<-rep(use[j],2^(m-k))}
}
return(paths)
}
```

Fillmissingnodes()

```
function (pathind,terminal,Ymat,yperm,m,obslist,probs)
{
####################################
## Function to find posterior probability of 1 when
## a terminal node occurs before Level m
## Inputs:
##      pathind: number of the terminal node entry in path
##      terminal: vector of nodes that are terminal
##      Ymat:the matrix of 0-1 label vectors
##      yperm:the vector giving the order in which the labels were fitted
##      m: number of levels
##      obslist:a list containing the indices of the cases in all of the nodes
##      probs: output from Findnodeprob() which to extend
## Outputs:
##     complete vector of posterior probability of 1 of corresponding
##         label for each node, excluding the first node.
#########################################################################
```

```
n<-nrow(Ymat)
Ymatperm<-matrix(0,nrow=n,ncol=m)
for(k in 1:m){Ymatperm[,k]<-Ymat[,yperm[k]]}

probsextend<-c(-1,probs)

missingreplace<-rep(-1,(2^(m+1))-1)
paths<-Findpath(m)
rownumbers<-which(pathind<m)

if(sum(rownumbers)>0.5){
colnumbers<-pathind[rownumbers]
nodenumbers<-terminal[rownumbers]
howmany<-length(rownumbers)


for(k in 1:howmany){
colnow<-colnumbers[k]      #path
rownow<-rownumbers[k]       #path
nodenow<-nodenumbers[k]
indekse<-obslist[[nodenow]]

begin<-colnow+1
eindig<-m
for(j in begin:eindig){
need<-Ymatperm[indekse,j]
ene<-sum(need)
aantal<-length(indekse)
missingreplace[paths[rownow,j]]<-ene/aantal}}

nonentries<-which(missingreplace>-1)

probsextend[nonentries]<-missingreplace[nonentries]}
probsextend<-probsextend[-1]
return(probsextend)
}
```

Fitcomplete()

```
function (Xmat,Ymat,K,M,m,minsize)
{
#############################################################################
## Function used to fit ensemble of M tree-structures
## Inputs
##     Xmat: the matrix of input values
##     Ymat: the matrix of 0-1 label vectors
##     K: total number of labels
##     M: number of permutations to use
##     m: total number of levels for each tree
##     minsize: the minimum number of cases for splitting to be allowed
## Outputs:
##   keepvariables: matrix where each row is vector
##                  of splitvariables (output from Pfitmodel) for a permutation
##   keepsplitvalues: matrix where each row is vector
##                   of splitvalues (output from Pfitmodel) for a permutation
##   keepnodeprobs: matrix where each row is node probabilities of a permutation
##   theperms: the matrix of permutations used
#############################################################################

#Sample M permutations using Pallcombs
theperms<-matrix(0,nrow=M,ncol=m)
allperms<-Pallcombs(K,m)
numberofperms<-nrow(allperms)
permsused<-sample(c(1:numberofperms),M)
for(k in 1:M){
theperms[k,]<-allperms[permsused[k],]}

#Keep
nsplits<-(2^m)-1
nnodes<-(2^(m+1))-2
keepvariables<-matrix(0,nrow=M,ncol=nsplits)
keepsplitvalues<-matrix(0,nrow=M,ncol=nsplits)
keepnodeprobs<-matrix(0,nrow=M,ncol=nnodes)


for(j in 1:M){

#Use Pfitmodel for each permutation
out<-Pfitmodel(Xmat,Ymat,theperms[j,],minsize,m)

#Find node probabilities for a permutation
nodeprobsout<-Findnodeprob(Ymat,theperms[j,],out[[3]],m)
```

```
#Find terminal nodes of a permutation
pathindout<-Findterminal(out[[1]],m)$pathindices
terminalout<-Findterminal(out[[1]],m)$terminal
fillednodeprobsout<-Fillmissingnodes(pathindout,terminalout,Ymat,
theperms[j,],m,out[[3]],nodeprobsout)

keepvariables[j,]<-out[[1]]
keepsplitvalues[j,]<-out[[2]]
keepnodeprobs[j,]<-fillednodeprobsout
}
return(list(keepvariables,keepsplitvalues,keepnodeprobs,theperms))
}
```

Dropobs()

```
function (Xmatnew,splitvar,splitval,m)
{
#######################################################
# Function used to fit model to new observations,
# here first determining which
# observations are in which nodes
#Inputs:
#   Xmatnew:the matrix of new input values
#   splitvar:variables on which to split
#   splitval:corresponding values on which to split
#   m: number of levels of tree
#Outputs:
#    obsinnodes:a list containing the indices of the
#               cases in all of the nodes
#######################################################

n<-nrow(Xmatnew)
xvec<-Xmatnew[,splitvar[1]]
uit<-Psplitobs(xvec,splitval[1])
obsinnodes<-list(1:n,uit[[1]],uit[[2]])

for(k in 1:(m-1)){
beginn<-2^k
eindig<-2^(k+1)-1

for(t in beginn:eindig){
indekse<-obsinnodes[[t]]
```

```
if((splitvar[t]>0.5) & (sum(indekse)>0.5)) {
xvec<-Xmatnew[indekse,splitvar[t]]
uit<-Psplitobs(xvec,splitval[t])
obsinnodes<-list.append(obsinnodes,
obsinnodes[[t]][uit[[1]]],obsinnodes[[t]][uit[[2]]])}

if((splitvar[t]>0.5) & (sum(indekse)<0.5)) {
obsinnodes<-list.append(obsinnodes,-1,-1)}

if(splitvar[t]<0.5){
obsinnodes<-list.append(obsinnodes,-1,-1)}
}}
return(obsinnodes)
}
```

Makeapred()

```
function (obslist,probs,K,Nnew,yperm,m)
{
#############################################################################
# Function to find posterior probabilities of 1 for each
# corresponding label for each new observation
# Inputs:
#    obslist: list of new observations in each node
#    probs: vector of posterior probabilities of 1 for each node excluding first
#    K:number of labels
#    Nnew:number of new observations
#    yperm:the vector giving the order in which the labels should be fitted
#    m: number of levels per tree
#  Output:
#   Nnew*K matrix of posterior probabilities
#############################################################################

postprobmat<-matrix(-1,nrow=Nnew,ncol=m)
probs<-c(-1,probs)

for(k in 1:m){
beginn<-2^k
eindig<-2^(k+1)-1

for(t in beginn:eindig){
indekse<-obslist[[t]]

if(sum(indekse)>0.5){
postprobmat[indekse,k]<-probs[t]}
```

```
if(sum(indekse)<0.5){
postprobmat[indekse,k]<-probs[t]}

}
}

postprobmatorder<-matrix(-1,nrow=Nnew,ncol=K)
for(k in 1:m){
postprobmatorder[,yperm[k]]<-postprobmat[,k]}

return(postprobmatorder)
}
```

Predictcomplete()

```
function (Xmatnew,variables,splitvalues,nodeprobs,perms,M,K,m)
{
##############################################################################
## Function used to find final multi-label classification for observations
## Inputs:
##   Xmatnew:the matrix of new input values
##   variables: matrix where each row is vector
##              of splitvariables (output from Pfitmodel) for a permutation
##   splitvalues: matrix where each row is vector
##               of splitvalues (output from Pfitmodel)for a permutation
##   nodeprobs: matrix where each row is node probabilities of a permutation
##   perms: matrix where each row gives permutation used
##   M: number of permutations
##   K: total number of labels
##   m: total number of levels of a tree
## Output:
##    Nnew*K matrix of posterior probabilities
##############################################################################


Nnew<-nrow(Xmatnew)

#Keep predictions
allpredictions<-array(-1,dim=c(Nnew,K,M))

for(j in 1:M){

# Drop observations in each tree-structure
obslist<-Dropobs(Xmatnew,variables[j,],splitvalues[j,],m)
```

```
#Find predictions of permutations
thepredictions<-Makeapred(obslist,nodeprobs[j,],K,Nnew,perms[j,],m)
allpredictions[,,j]<-thepredictions
}

finalresults<-matrix(0,nrow=Nnew,ncol=K)

for(n in 1:Nnew){
for(k in 1:K){
tel<-0
for(j in 1:M){
if(allpredictions[n,k,j]>=0){
tel<-tel+1
finalresults[n,k]<-finalresults[n,k]+allpredictions[n,k,j]}
}
finalresults[n,k]<-(finalresults[n,k])/tel
}}

finalresults[finalresults>=0.5]<-1
finalresults[finalresults<0.5]<-0
return(finalresults)
}
```

A.2 EMPIRICAL STUDY

Emotions

```
##########################################
## Creating train and test mldr-objects ##
##########################################

> Emotionsmat<-as.matrix((emotions$dataset)[,1:78])
> Emotionsx<-Emotionsmat[,1:72]
> Emotionsy<-Emotionsmat[,73:78]
> Emotionstrainx<-Emotionsx[1:400,]
> Emotionstrainy<-Emotionsy[1:400,]
> Emotionstestx<-Emotionsx[401:593,]
> Emotionstesty<-Emotionsy[401:593,]
> Emotionsmldr<-mldr_from_dataframe(as.data.frame(cbind(Emotionstrainx,
Emotionstrainy)),labelIndices=c(73:78),name="trainEmotions")
> Emotionstestmldr<-mldr_from_dataframe(as.data.frame(cbind(Emotionstestx,
Emotionstesty)),labelIndices=c(73:78),name="testEmotions")
```

```
########################################
## Fitting LDsplit with trees models ##
########################################

## m=2 M=6
> emotionsresults26<-Fitcomplete(Emotionstrainx,Emotionstrainy,6,6,2,5)
> emotionspredict26<-Predictcomplete(Emotionstestx,emotionsresults26[[1]],
emotionsresults26[[2]],emotionsresults26[[3]],emotionsresults26[[4]],6,6,2)
> emotions26confmat<-multilabel_confusion_matrix(Emotionstestmldr,
emotionspredict26)
> multilabel_evaluate(emotions26confmat)

# Fit other LDsplit models in similar manner, varying m and M values
```

```
##########################
## Fitting other models ##
##########################

# Binary relevance
> library(rpart)
> emotionsbrresults<-br(Emotionsmldr,"CART")
> emotionsbrpredict<-predict(emotionsbrresults,
as.data.frame(Emotionstestx))
> matemotionsbrpredict<-as.matrix(emotionsbrpredict)
> matemotionsbrpredict[emotionsbrpredict>=0.5]<-1
> matemotionsbrpredict[emotionsbrpredict<0.5]<-0
> emotionsbrconfmat<-multilabel_confusion_matrix(Emotionstestmldr,
matemotionsbrpredict)
> multilabel_evaluate(emotionsbrconfmat)

# Label powerset
> emotionslpresults<-lp(Emotionsmldr,"CART")
> emotionslppredict<-predict(emotionslpresults,
as.data.frame(Emotionstestx))
> matemotionslppredict<-as.matrix(emotionslppredict)
> matemotionslppredict[emotionslppredict>=0.5]<-1
> matemotionslppredict[emotionslppredict<0.5]<-0
> emotionslpconfmat<-multilabel_confusion_matrix(Emotionstestmldr,
matemotionslppredict)
> multilabel_evaluate(emotionslpconfmat)

# Classifier chains
> emotionsccresults<-cc(Emotionsmldr,"CART")
> emotionsccpredict<-predict(emotionsccresults,
as.data.frame(Emotionstestx))
> matemotionsccpredict<-as.matrix(emotionsccpredict)
> matemotionsccpredict[emotionsccpredict>=0.5]<-1
> matemotionsccpredict[emotionsccpredict<0.5]<-0
> emotionsccconfmat<-multilabel_confusion_matrix(Emotionstestmldr,
matemotionsccpredict)
> multilabel_evaluate(emotionsccconfmat)
```

```
# Ensemble of classifier chains
> emotionseccresults10<-ecc(Emotionsmldr,"CART",10,0.75,1)
> emotionseccpredict10<-predict(emotionseccresults10,
as.data.frame(Emotionstestx))
> matemotionseccpredict10<-as.matrix(emotionseccpredict10)
> matemotionseccpredict10[emotionseccpredict10>=0.5]<-1
> matemotionseccpredict10[emotionseccpredict10<0.5]<-0
> emotionseccconfmat10<-multilabel_confusion_matrix(Emotionstestmldr,
matemotionseccpredict10)
> multilabel_evaluate(emotionseccconfmat10)

# RAkEL disjoint
> emotionsrakeldresults<-rakel(Emotionsmldr,"CART",3,overlapping=FALSE)
> emotionsrakeldpredict<-predict(emotionsrakeldresults,
as.data.frame(Emotionstestx))
> matemotionsrakeldpredict<-as.matrix(emotionsrakeldpredict)
> matemotionsrakeldpredict[emotionsrakeldpredict>=0.5]<-1
> matemotionsrakeldpredict[emotionsrakeldpredict<0.5]<-0
> emotionsrakeldconfmat<-multilabel_confusion_matrix(Emotionstestmldr,
matemotionsrakeldpredict)
> multilabel_evaluate(emotionsrakeldconfmat)

# RAkEL overlapping
> emotionsrakeloresults<-rakel(Emotionsmldr,"CART",3)
> emotionsrakelopredict<-predict(emotionsrakeloresults,
as.data.frame(Emotionstestx))
> matemotionsrakelopredict<-as.matrix(emotionsrakelopredict)
> matemotionsrakelopredict[emotionsrakelopredict>=0.5]<-1
> matemotionsrakelopredict[emotionsrakelopredict<0.5]<-0
> emotionsrakeloconfmat<-multilabel_confusion_matrix(Emotionstestmldr,
matemotionsrakelopredict)
> multilabel_evaluate(emotionsrakeloconfmat)
```

Flags

```
##########################################
## Creating train and test mldr-objects ##
##########################################

> flagsmat<-flags$dataset[,1:26]
> flagsmat<-(as.matrix(as.data.frame(lapply(flagsmat, as.numeric))))
> Flagsmat<-flagsmat
> Flagsx<-Flagsmat[,1:19]
> Flagsy<-Flagsmat[,20:26]
> Flagstrainx<-Flagsx[1:130,]
> Flagstrainy<-Flagsy[1:130,]
> Flagstestx<-Flagsx[131:194,]
> Flagstesty<-Flagsy[131:194,]
> Flagsmldr<-mldr_from_dataframe(as.data.frame(
cbind(Flagstrainx,Flagstrainy)),labelIndices=c(20:26),name="trainFLAGS")
> Flagstestmldr<-mldr_from_dataframe(as.data.frame(
cbind(Flagstestx,Flagstesty)),labelIndices=c(20:26),name="testFLAGS")
```

```
########################################
## Fitting LDsplit with trees models ##
########################################


## m=2 M=7
> flagsresults27<-Fitcomplete(Flagstrainx,Flagstrainy,7,7,2,5)
> flagspredict27<-Predictcomplete(Flagstestx,flagsresults27[[1]],
flagsresults27[[2]],flagsresults27[[3]],flagsresults27[[4]],7,7,2)
> flags27confmat<-multilabel_confusion_matrix(Flagstestmldr,
flagspredict27)
> multilabel_evaluate(flags27confmat)



# Fit other LDsplit models in similar manner, varying m and M values
```

```
##########################
## Fitting other models ##
##########################

# Binary relevance
> flagsbrresults<-br(Flagsmldr,"CART")
> flagsbrpredict<-predict(flagsbrresults,
as.data.frame(Flagstestx))
> matflagsbrpredict<-as.matrix(flagsbrpredict)
> matflagsbrpredict[flagsbrpredict>=0.5]<-1
> matflagsbrpredict[flagsbrpredict<0.5]<-0
> flagsbrconfmat<-multilabel_confusion_matrix(Flagstestmldr,
matflagsbrpredict)
> multilabel_evaluate(flagsbrconfmat)

# Label powerset
> flagslpresults<-lp(Flagsmldr,"CART")
> flagslppredict<-predict(flagslpresults,
as.data.frame(Flagstestx))
> matflagslppredict<-as.matrix(flagslppredict)
> matflagslppredict[flagslppredict>=0.5]<-1
> matflagslppredict[flagslppredict<0.5]<-0
> flagslpconfmat<-multilabel_confusion_matrix(Flagstestmldr,
matflagslppredict)
> multilabel_evaluate(flagslpconfmat)

# Classifier chains
> flagsccresults<-cc(Flagsmldr,"CART")
> flagsccpredict<-predict(flagsccresults,
as.data.frame(Flagstestx))
> matflagsccpredict<-as.matrix(flagsccpredict)
> matflagsccpredict[flagsccpredict>=0.5]<-1
> matflagsccpredict[flagsccpredict<0.5]<-0
> flagsccconfmat<-multilabel_confusion_matrix(Flagstestmldr,
matflagsccpredict)
> multilabel_evaluate(flagsccconfmat)
```

```
# Ensemble of classifier chains
> flagseccresults<-ecc(Flagsmldr,"CART",10,0.75,1)
> flagseccpredict<-predict(flagseccresults,
as.data.frame(Flagstestx))
> matflagseccpredict<-as.matrix(flagseccpredict)
> matflagseccpredict[flagseccpredict>=0.5]<-1
> matflagseccpredict[flagseccpredict<0.5]<-0
> flagseccconfmat<-multilabel_confusion_matrix(Flagstestmldr,
matflagseccpredict)
> multilabel_evaluate(flagseccconfmat)

# RAkEL disjoint
> flagsrakeldresults<-rakel(Flagsmldr,"CART",4,overlapping=FALSE)
> flagsrakeldpredict<-predict(flagsrakeldresults,
as.data.frame(Flagstestx))
> matflagsrakeldpredict<-as.matrix(flagsrakeldpredict)
> matflagsrakeldpredict[flagsrakeldpredict>=0.5]<-1
> matflagsrakeldpredict[flagsrakeldpredict<0.5]<-0
> flagsrakeldconfmat<-multilabel_confusion_matrix(Flagstestmldr,
matflagsrakeldpredict)
> multilabel_evaluate(flagsrakeldconfmat)

# RAkEL overlapping
> flagsrakeloresults<-rakel(Flagsmldr,"CART",4)
> flagsrakelopredict<-predict(flagsrakeloresults,
as.data.frame(Flagstestx))
> matflagsrakelopredict<-as.matrix(flagsrakelopredict)
> matflagsrakelopredict[flagsrakelopredict>=0.5]<-1
> matflagsrakelopredict[flagsrakelopredict<0.5]<-0
> flagsrakeloconfmat<-multilabel_confusion_matrix(Flagstestmldr,
matflagsrakelopredict)
> multilabel_evaluate(flagsrakeloconfmat)
```

Yeast

```
##########################################
## Creating train and test mldr-objects ##
##########################################

> yeast<-yeast()
> toBibtex(yeast)
> yeastmat<-yeast$dataset[,1:117]
> yeastmat<-(as.matrix(as.data.frame(lapply(yeastmat, as.numeric))))
> Yeastmat<-yeastmat
> Yeastx<-Yeastmat[,1:103]
> Yeasty<-Yeastmat[,104:117]
> Yeasttrainx<-Yeastx[1:1611,]
> Yeasttrainy<-Yeasty[1:1611,]
> Yeasttestx<-Yeastx[1612:2417,]
> Yeasttesty<-Yeasty[1612:2417,]
> Yeastmldr<-mldr_from_dataframe(as.data.frame(
cbind(Yeasttrainx,Yeasttrainy)),labelIndices=c(104:117),name="trainYEAST")
> Yeasttestmldr<-mldr_from_dataframe(as.data.frame(
cbind(Yeasttestx,Yeasttesty)),labelIndices=c(104:117),name="testYEAST")
```

```
########################################
## Fitting LDsplit with trees models ##
########################################

## m=2 M=14
> yeastresults214<-Fitcomplete(Yeasttrainx,Yeasttrainy,14,14,2,5)
> yeastpredict214<-Predictcomplete(Yeasttestx,yeastresults214[[1]],
yeastresults214[[2]],yeastresults214[[3]],yeastresults214[[4]],14,14,2)
> yeast214confmat<-multilabel_confusion_matrix(Yeasttestmldr,
yeastpredict214)
> multilabel_evaluate(yeast214confmat)



# Fit other LDsplit models in similar manner, varying m and M values
```

```
##########################
## Fitting other models ##
##########################


# Binary relevance
> yeastbrresults<-br(Yeastmldr,"CART")
> yeastbrpredict<-predict(yeastbrresults,
as.data.frame(Yeasttestx))
> matyeastbrpredict<-as.matrix(yeastbrpredict)
> matyeastbrpredict[yeastbrpredict>=0.5]<-1
> matyeastbrpredict[yeastbrpredict<0.5]<-0
> yeastbrconfmat<-multilabel_confusion_matrix(Yeasttestmldr,
matyeastbrpredict)
> multilabel_evaluate(yeastbrconfmat)


# Label powerset
> yeastlpresults<-lp(Yeastmldr,"CART")
> yeastlppredict<-predict(yeastlpresults,
as.data.frame(Yeasttestx))
> matyeastlppredict<-as.matrix(yeastlppredict)
> matyeastlppredict[yeastlppredict>=0.5]<-1
> matyeastlppredict[yeastlppredict<0.5]<-0
> yeastlpconfmat<-multilabel_confusion_matrix(Yeasttestmldr,
matyeastlppredict)
> multilabel_evaluate(yeastlpconfmat)

# Classifier chains
> yeastccresults<-cc(Yeastmldr,"CART")
> yeastccpredict<-predict(yeastccresults,
as.data.frame(Yeasttestx))
> matyeastccpredict<-as.matrix(yeastccpredict)
> matyeastccpredict[yeastccpredict>=0.5]<-1
> matyeastccpredict[yeastccpredict<0.5]<-0
> yeastccconfmat<-multilabel_confusion_matrix(Yeasttestmldr,
matyeastccpredict)
> multilabel_evaluate(yeastccconfmat)
```

```
# Ensemble of classifier chains
> yeasteccresults10<-ecc(Yeastmldr,"CART",10,0.75,1)
> yeasteccpredict10<-predict(yeasteccresults10,
as.data.frame(Yeasttestx))
> matyeasteccpredict10<-as.matrix(yeasteccpredict10)
> matyeasteccpredict10[yeasteccpredict10>=0.5]<-1
> matyeasteccpredict10[yeasteccpredict10<0.5]<-0
> yeastecconfmat10<-multilabel_confusion_matrix(Yeasttestmldr,
matyeasteccpredict10)
> multilabel_evaluate(yeastecconfmat10)

# RAkEL disjoint
> yeastrakeldresults<-rakel(Yeastmldr,"CART",4,overlapping=FALSE)
> yeastrakeldpredict<-predict(yeastrakeldresults,
as.data.frame(Yeasttestx))
> matyeastrakeldpredict<-as.matrix(yeastrakeldpredict)
> matyeastrakeldpredict[yeastrakeldpredict>=0.5]<-1
> matyeastrakeldpredict[yeastrakeldpredict<0.5]<-0
> yeastrakeldconfmat<-multilabel_confusion_matrix(Yeasttestmldr,
matyeastrakeldpredict)
> multilabel_evaluate(yeastrakeldconfmat)

# RAkEL overlapping
> yeastrakeloresults<-rakel(Yeastmldr,"CART",4)
> yeastrakelopredict<-predict(yeastrakeloresults,
as.data.frame(Yeasttestx))
> matyeastrakelopredict<-as.matrix(yeastrakelopredict)
> matyeastrakelopredict[yeastrakelopredict>=0.5]<-1
> matyeastrakelopredict[yeastrakelopredict<0.5]<-0
> yeastrakeloconfmat<-multilabel_confusion_matrix(Yeasttestmldr,
matyeastrakelopredict)
> multilabel_evaluate(yeastrakeloconfmat)
```

APPENDIX B: Chapter 5

B.1 LIST OF 174 STOPWORDS IN TM PACKAGE

```
> stopwords(kind="en")
  [1] "i"          "me"         "my"         "myself"     "we"
  [6] "our"        "ours"       "ourselves"  "you"        "your"
 [11] "yours"      "yourself"   "yourselves" "he"         "him"
 [16] "his"        "himself"    "she"        "her"        "hers"
 [21] "herself"    "it"         "its"        "itself"     "they"
 [26] "them"       "their"      "theirs"     "themselves" "what"
 [31] "which"      "who"        "whom"       "this"       "that"
 [36] "these"      "those"      "am"         "is"         "are"
 [41] "was"        "were"       "be"         "been"       "being"
 [46] "have"       "has"        "had"        "having"     "do"
 [51] "does"       "did"        "doing"      "would"      "should"
 [56] "could"      "ought"      "i'm"        "you're"     "he's"
 [61] "she's"      "it's"       "we're"      "they're"    "i've"
 [66] "you've"     "we've"      "they've"    "i'd"        "you'd"
 [71] "he'd"       "she'd"      "we'd"       "they'd"     "i'll"
 [76] "you'll"     "he'll"      "she'll"     "we'll"      "they'll"
 [81] "isn't"      "aren't"     "wasn't"     "weren't"    "hasn't"
 [86] "haven't"    "hadn't"     "doesn't"    "don't"      "didn't"
 [91] "won't"      "wouldn't"   "shan't"     "shouldn't"  "can't"
 [96] "cannot"     "couldn't"   "mustn't"    "let's"      "that's"
[101] "who's"      "what's"     "here's"     "there's"    "when's"
[106] "where's"    "why's"      "how's"      "a"          "an"
[111] "the"        "and"        "but"        "if"         "or"
[116] "because"    "as"         "until"      "while"      "of"
[121] "at"         "by"         "for"        "with"       "about"
[126] "against"    "between"    "into"       "through"    "during"
[131] "before"     "after"      "above"      "below"      "to"
[136] "from"       "up"         "down"       "in"         "out"
[141] "on"         "off"        "over"       "under"      "again"
[146] "further"    "then"       "once"       "here"       "there"
[151] "when"       "where"      "why"        "how"        "all"
[156] "any"        "both"       "each"       "few"        "more"
[161] "most"       "other"      "some"       "such"       "no"
[166] "nor"        "not"        "only"       "own"        "same"
[171] "so"         "than"       "too"        "very"
```

B.2 PRACTICAL DATA ANALYSIS

B.2.1 Forming initial training corpus

```
> library(NLP)
> library(tm)
> textdata<-read.csv(file=" ",header=TRUE,sep=",",stringsAsFactors=FALSE)
> textcolumn<-textdata[,2]
> texttrain<-textcolumn[1:106381]
> texttrain<-as.vector(texttrain)
> texttest<-textcolumn[106382:159571]
> texttest<-as.vector(texttest)
> labelscolumn<-textdata[,3:8]
> labelstrain<-labelscolumn[1:106381,]
> labelstest<-labelscolumn[106382:159571,]
> texttrain<-VectorSource(texttrain)
> texttrain<-VCorpus(texttrain)
```

B.2.2 Converting text to lower-case, removing extra white spaces, removing particular words, removing punctuation and removing numbers from the text documents

```
> fix(nocaps)
function (x)
{
nocaps<-x
for(i in 1:106381){
nocaps[[i]][1]$content<-tolower(x[[i]][1]$content)}
return(nocaps)
}

> fix(my.removewords)
function (x)
{
nowords<-x
for(i in 1:106381){
nowords[[i]][1]$content<-removeWords(x[[i]][1]$content,
stopwords(kind="en"))}
return(nowords)
}
```

```
> fix(my.removewhitespace)
function (x)
{
nowhitespace<-x
for(i in 1:106381){
nowhitespace[[i]][1]$content<-stripWhitespace(x[[i]][1]$content)}
return(nowhitespace)
}


> fix(cleandata)
function (data)
{
#Caps
capsremoved<-nocaps(data)

#white space
whitespaceremoved<-my.removewhitespace(capsremoved)

#remove words
wordsremoved<-my.removewords(whitespaceremoved)

#white space
againremoved<-my.removewhitespace(wordsremoved)

#punctuation
puncremoved<-tm_map(againremoved,removePunctuation)

#numbers
everythingremoved<-tm_map(puncremoved,removeNumbers)

return(everythingremoved)
}

> texttrainclean<-cleandata(texttrain)
```

B.2.3 Constructing initial DTM

```
> DTMtexttrainclean<-DocumentTermMatrix(texttrainclean)
> DTMtexttrainclean
<<DocumentTermMatrix (documents: 106381, terms: 171082)>>
Non-/sparse entries: 2848452/18197025790
Sparsity           : 100%
Maximal term length: 4928
Weighting          : term frequency (tf)
```

B.2.4 Function used to replace rude terms in training corpus with specified codenames

```
replaceSynonyms<-content_transformer(function(x,syn=NULL){
Reduce(function(a,b){
gsub(paste0("\\b(",paste(b$syns,collapse="|"),")\\b"),
b$word,a)},syn,x)})
```

B.2.5 Constructing DTM of training corpus that contains codenames

```
> cleansynonymslelik<-tm_map(texttrainclean,replaceSynonyms,synonymslelik)
> DTMtexttrainclean1<-DocumentTermMatrix(cleansynonymslelik)
> DTMtexttrainclean1
<<DocumentTermMatrix (documents: 106381, terms: 170780)>>
Non-/sparse entries: 2847999/18164899181
Sparsity           : 100%
Maximal term length: 4928
Weighting          : term frequency (tf)
```

B.2.6 Finding initial list of overall top occurring terms

```
> DTM1topterms<-removeSparseTerms(DTMtexttrainclean1, 0.9988)
> DTM1topterms
<<DocumentTermMatrix (documents: 106381, terms: 3036)>>
Non-/sparse entries: 2130992/320841724
Sparsity           : 99%
Maximal term length: 21
Weighting          : term frequency (tf)
```

B.2.7 Terms combined

```
synonymsander<-list(list(word="apologise",syns=c("apologize"))
,list(word="account",syns=c("accounts"))
,list(word="accusation",syns=c("accusations"))
,list(word="acknowledge",syns=c("acknowledged"))
,list(word="actor",syns=c("actors"))
,list(word="administrator",syns=c("administrators"))
,list(word="album",syns=c("albums"))
,list(word="allow",syns=c("allowed","allowing","allows"))
,list(word="alter",syns=c("altered"))
,list(word="among",syns=c("amongst"))
,list(word="animal",syns=c("animals"))
,list(word="anyone",syns=c("anybody"))
,list(word="apology",syns=c("apologies"))
,list(word="appear",syns=c("appeared"))
,list(word="appreciate",syns=c("appreciated"))
,list(word="area",syns=c("areas"))
```

```
,list(word="article",syns=c("articles"))
,list(word="artist",syns=c("artists"))
,list(word="ask",syns=c("asked","asking"))
,list(word="assume",syns=c("assumed","assuming"))
,list(word="assumptions",syns=c("assumptions"))
,list(word="author",syns=c("authors"))
,list(word="award",syns=c("awards"))
,list(word="behaviour",syns=c("behavior"))
,list(word="belief",syns=c("beliefs"))
,list(word="believe",syns=c("believed","believes"))
,list(word="biography",syns=c("biographies"))
,list(word="blatant",syns=c("blatantly"))
,list(word="blog",syns=c("blogs"))
,list(word="boy",syns=c("boys"))
,list(word="bring",syns=c("bringing","brings","brought"))
,list(word="broke",syns=c("broken"))
,list(word="brother",syns=c("brothers"))
,list(word="called",syns=c("calling"))
,list(word="candidate",syns=c("candidates"))
,list(word="category",syns=c("categories"))
,list(word="caused",syns=c("causing"))
,list(word="centory",syns=c("centuries"))
,list(word="changed",syns=c("changing"))
,list(word="child",syns=c("children"))
,list(word="choose",syns=c("chose","chosen"))
,list(word="citation",syns=c("citations"))
,list(word="city",syns=c("cities"))
,list(word="claimed",syns=c("claiming"))
,list(word="colour",syns=c("colours","color","colors"))
,list(word="commented",syns=c("commenting"))
,list(word="complain",syns=c("complaining","complained"))
,list(word="complaint",syns=c("complaints"))
,list(word="concept",syns=c("concepts"))
,list(word="confirm",syns=c("confirmed","confirmed"))
,list(word="confused",syns=c("confusing"))
,list(word="consider",syns=c("considered","considering"))
,list(word="constitute",syns=c("constitutes"))

,list(word="contain",syns=c("contained","contains"))
,list(word="contribute",syns=c("contributed","contributing"))
,list(word="contribution",syns=c("contributions"))
,list(word="contributor",syns=c("contributors"))
,list(word="convention",syns=c("conventions"))
,list(word="convince",syns=c("convinced"))
,list(word="corrected",syns=c("correcting"))
,list(word="created",syns=c("creating"))
,list(word="crime",syns=c("crimes"))
```

```
,list(word="criticism",syns=c("criticisms"))
,list(word="day",syns=c("days"))
,list(word="decide",syns=c("decided"))
,list(word="decision",syns=c("decisions"))
,list(word="define",syns=c("defined"))
,list(word="definition",syns=c("definitions"))
,list(word="delete",syns=c("deleted","deleting"))
,list(word="deletion",syns=c("deletions"))
,list(word="demonstrate",syns=c("demonstrated"))
,list(word="deny",syns=c("denied"))
,list(word="describe",syns=c("described","describes","describing"))
,list(word="deserve",syns=c("deserves"))
,list(word="destroy",syns=c("destroyed"))
,list(word="develop",syns=c("developed"))
,list(word="disagreement",syns=c("disagreements"))
,list(word="discuss",syns=c("discussed","discussing"))
,list(word="discussion",syns=c("discussions"))
,list(word="dog",syns=c("dogs"))
,list(word="eat",syns=c("eats"))
,list(word="edited",syns=c("editing"))
,list(word="election",syns=c("elections"))
,list(word="encourage",syns=c("encouraged"))
,list(word="encyclopedia",syns=c("encyclopaedia"))
,list(word="episode",syns=c("episodes"))
,list(word="error",syns=c("errors"))
,list(word="event",syns=c("events"))
,list(word="everyone",syns=c("everybody"))
,list(word="example",syns=c("examples"))
,list(word="exist",syns=c("existed","existing","exists"))
,list(word="expand",syns=c("expanded","expanding"))
,list(word="expect",syns=c("expected"))
,list(word="explain",syns=c("explained","explaining","explains"))
,list(word="explanation",syns=c("explanations"))
,list(word="eye",syns=c("eyes"))
,list(word="fact",syns=c("facts"))
,list(word="favour",syns=c("favor"))
,list(word="february",syns=c("feb"))

,list(word="field",syns=c("fields"))
,list(word="fixed",syns=c("fixing"))
,list(word="forget",syns=c("forgot"))
,list(word="forum",syns=c("forums"))
,list(word="game",syns=c("games"))
,list(word="girl",syns=c("girls"))
,list(word="government",syns=c("governments"))
,list(word="grow",syns=c("growing"))
,list(word="happen",syns=c("happened","happening","happens"))
```

```
,list(word="hear",syns=c("heard","hearing"))
,list(word="helped",syns=c("helping"))
,list(word="historian",syns=c("historians"))
,list(word="hour",syns=c("hours"))
,list(word="hundred",syns=c("hundreds"))
,list(word="idea",syns=c("ideas"))
,list(word="ignore",syns=c("ignored","ignoring"))
,list(word="imply",syns=c("implying","implies"))
,list(word="improve",syns=c("improved","improving"))
,list(word="improvement",syns=c("improvements"))
,list(word="include",syns=c("included","includes","including"))
,list(word="indicate",syns=c("indicated","indicates"))
,list(word="insert",syns=c("inserted","inserting"))
,list(word="intend",syns=c("intended"))
,list(word="intention",syns=c("intentions"))
,list(word="introduce",syns=c("introduced"))
,list(word="involve",syns=c("involved","involving"))
,list(word="january",syns=c("jan"))
,list(word="joined",syns=c("joining"))
,list(word="june",syns=c("jun"))
,list(word="justify",syns=c("justified"))
,list(word="language",syns=c("languages"))
,list(word="learn",syns=c("learned","learning"))
,list(word="letter",syns=c("letters"))
,list(word="love",syns=c("loves"))
,list(word="member",syns=c("members"))
,list(word="mentioned",syns=c("mentioning"))
,list(word="merge",syns=c("merged","merging"))
,list(word="message",syns=c("messages"))
,list(word="method",syns=c("methods"))
,list(word="minute",syns=c("minutes"))
,list(word="month",syns=c("months"))
,list(word="mother",syns=c("mom","mothjer","mum"))
,list(word="movie",syns=c("movies"))
,list(word="newspaper",syns=c("newspapers"))
,list(word="nominate",syns=c("nominated"))
,list(word="november",syns=c("nov"))

,list(word="objection",syns=c("objections"))
,list(word="occur",syns=c("occured"))
,list(word="october",syns=c("oct"))
,list(word="opinion",syns=c("opinions"))
,list(word="oppose",syns=c("opposed","opposing"))
,list(word="organisation",syns=c("organization","organizations","organisations"))
,list(word="paragraph",syns=c("paragraphs"))
,list(word="photo",syns=c("photos"))
,list(word="pointed",syns=c("pointing"))
```

```
,list(word="prefer",syns=c("preferred"))
,list(word="presented",syns=c("presenting"))
,list(word="problem",syns=c("problems"))
,list(word="prove",syns=c("proved","proven","proves"))
,list(word="provide",syns=c("provided","provides","providing"))
,list(word="publication",syns=c("publications"))
,list(word="publish",syns=c("published"))
,list(word="reader",syns=c("readers"))
,list(word="realize",syns=c("realized"))
,list(word="receive",syns=c("received","receiving"))
,list(word="recognize",syns=c("recognized"))
,list(word="refuse",syns=c("refused"))
,list(word="regarded",syns=c("regarding"))
,list(word="renamed",syns=c("renaming"))
,list(word="repeated",syns=c("repeating"))
,list(word="replace",syns=c("replaced","replacing"))
,list(word="reported",syns=c("reporting"))
,list(word="requested",syns=c("requesting"))
,list(word="require",syns=c("required"))
,list(word="requirement",syns=c("requirements"))
,list(word="respond",syns=c("responded","responding"))
,list(word="response",syns=c("responses"))
,list(word="restore",syns=c("restored","restoring"))
,list(word="reverted",syns=c("reverting"))
,list(word="reviewed",syns=c("reviewing"))
,list(word="rewrite",syns=c("rewritten"))
,list(word="ruined",syns=c("ruining"))
,list(word="scholar",syns=c("scholars"))
,list(word="school",syns=c("schools"))
,list(word="scientist",syns=c("scientists"))
,list(word="send",syns=c("sending","sent"))
,list(word="sentence",syns=c("sentences"))
,list(word="separate",syns=c("seperate"))
,list(word="showed",syns=c("showing","shown"))
,list(word="sock",syns=c("socks"))
,list(word="sockpuppet",syns=c("sockpuppets"))
,list(word="somebody",syns=c("someone"))

,list(word="song",syns=c("songs"))
,list(word="sourced",syns=c("sourcing"))
,list(word="started",syns=c("starting"))
,list(word="story",syns=c("stories"))
,list(word="student",syns=c("students"))
,list(word="suggest",syns=c("suggested","suggesting","suggests"))
,list(word="suggestion",syns=c("suggestions"))
,list(word="supported",syns=c("supporting"))
,list(word="system",syns=c("systems"))
```

```
,list(word="table",syns=c("tables"))
,list(word="talked",syns=c("talking"))
,list(word="template",syns=c("templates"))
,list(word="thread",syns=c("threads"))
,list(word="topic",syns=c("topics"))
,list(word="toward",syns=c("towards"))
,list(word="translate",syns=c("translated"))
,list(word="turned",syns=c("turning"))
,list(word="unblock",syns=c("unblocked"))
,list(word="updated",syns=c("updating"))
,list(word="uploaded",syns=c("uploading"))
,list(word="vandalize",syns=c("vandalise","vandalized",
"vandalised","vandalizing","vandalising"))
,list(word="verify",syns=c("verified"))
,list(word="victim",syns=c("victims"))
,list(word="video",syns=c("videos"))
,list(word="violate",syns=c("violated","violates","violating"))
,list(word="violation",syns=c("violations"))
,list(word="voted",syns=c("voting"))
,list(word="website",syns=c("websites"))
,list(word="week",syns=c("weeks"))
,list(word="writer",syns=c("writers"))
,list(word="year",syns=c("years"))
)
```

B.2.8 Creating list of overall top occurring terms

```
> cleansynonymsander<-tm_map(cleansynonymslelik,replaceSynonyms,synonymsander)
> DTMtexttrainclean2<-DocumentTermMatrix(cleansynonymsander)
> DTM2topterms<-removeSparseTerms(DTMtexttrainclean2, 0.9988)
> DTM2topterms
<<DocumentTermMatrix (documents: 106381, terms: 2775)>>
Non-/sparse entries: 2115705/293091570
Sparsity           : 99%
Maximal term length: 21
Weighting          : term frequency (tf)
```

B.2.9 List of overall top occurring terms

```
> Terms(DTM2topterms)
   [1] "â€“"                "â€”"                "â€”â"
   [4] "â€”preceding"       "â€¢"                "â€œ"
   [7] "âtʼ"              "aanstoot1"          "aanstoot10"
  [10] "aanstoot11"        "aanstoot12"         "aanstoot13"
  [13] "aanstoot14"        "aanstoot15"         "aanstoot17"
  [16] "aanstoot18"        "aanstoot19"         "aanstoot2"
  [19] "aanstoot20"        "aanstoot21"         "aanstoot22"
  [22] "aanstoot30"        "aanstoot34"         "aanstoot35"
  [25] "aanstoot37"        "aanstoot44"         "aanstoot48"
  [28] "aanstoot49"        "aanstoot5"          "aanstoot51"
  [31] "aanstoot52"        "aanstoot53"         "aanstoot55"
  [34] "aanstoot66"        "aanstoot7"          "aanstoot78"
  [37] "aanstoot8"         "aanstoot9"          "ability"
  [40] "able"              "absolute"           "absolutely"
  [43] "absurd"            "abuse"              "abusing"
  [46] "abusive"           "academic"           "accept"
  [49] "acceptable"        "accepted"           "access"
  [52] "accident"          "accordance"         "according"
  [55] "accordingly"       "account"            "accuracy"
  [58] "accurate"          "accurately"         "accusation"
  [61] "accuse"            "accused"            "accusing"
  [64] "achieve"           "acknowledge"        "across"
  [67] "act"               "acting"             "action"
  [70] "actions"           "active"             "activities"
  [73] "activity"          "actor"              "acts"
  [76] "actual"            "actually"           "add"
  [79] "added"             "adding"             "addition"
  [82] "additional"        "additionally"       "additions"
  [85] "address"           "addressed"          "addresses"
  [88] "addressing"        "adds"               "admin"
  [91] "administration"    "administrative"     "administrator"
  [94] "admins"            "adminship"          "admit"
  [97] "admitted"          "adopted"            "adult"
 [100] "advance"           "advertising"        "advice"
 [103] "advise"            "afd"                "afraid"
 [106] "africa"            "african"            "age"
 [109] "agenda"            "aggressive"         "ago"
 [112] "agree"             "agreed"             "agreement"
 [115] "agrees"            "ahead"              "aim"
 [118] "aint"              "air"                "aircraft"
 [121] "aka"               "album"              "alive"
 [124] "allegations"       "alleged"            "allow"
 [127] "almost"            "alone"              "along"
 [130] "alot"              "already"            "alright"
 [133] "also"              "alter"              "alternative"
 [136] "although"          "altogether"         "always"
 [139] "amazing"           "america"            "american"
 [142] "americans"         "among"              "amount"
 [145] "amounts"           "analysis"           "ancient"
 [148] "andrew"            "angry"              "ani"
 [151] "animal"            "announced"          "annoying"
 [154] "anon"              "anonymous"          "another"
 [157] "answer"            "answered"           "answers"
 [160] "anymore"           "anyone"             "anything"
 [163] "anyway"            "anyways"            "anywhere"
 [166] "apart"             "apologise"          "apology"
 [169] "apparent"          "apparently"         "appeal"
 [172] "appear"            "appearance"         "appears"
 [175] "applicable"        "application"        "applied"
 [178] "applies"           "apply"              "appreciate"
 [181] "approach"          "appropriate"        "approved"
 [184] "apr"               "april"              "arbcom"
 [187] "arbitration"       "archive"            "archived"
 [190] "archives"          "area"               "argue"
 [193] "argued"            "arguing"            "argument"
 [196] "arguments"         "arms"               "army"
 [199] "around"            "arrogant"           "art"
 [202] "article"           "artist"             "asia"
 [205] "asian"             "aside"              "ask"
 [208] "aspect"            "aspects"            "assert"
 [211] "assertion"         "assertions"         "assessment"
 [214] "assist"            "assistance"         "associated"
 [217] "association"       "assume"             "assumption"
 [220] "assure"            "attached"           "attack"
 [223] "attacked"          "attacking"          "attacks"
 [226] "attempt"           "attempted"          "attempting"
 [229] "attempts"          "attention"          "attitude"
 [232] "attributed"        "audio"              "august"
 [235] "australia"         "australian"         "author"
 [238] "authorities"       "authority"          "automatically"
 [241] "available"         "average"            "avoid"
 [244] "award"             "aware"              "away"
 [247] "awesome"           "baby"               "back"
 [250] "backed"            "background"         "bad"
 [253] "badly"             "balance"            "balanced"
 [256] "ban"               "band"               "bands"
 [259] "bank"              "banned"             "banning"
 [262] "bar"               "barely"             "barnstar"
 [265] "base"              "based"              "basic"
 [268] "basically"         "basis"              "battle"
 [271] "bbc"               "bear"               "beat"
 [274] "beautiful"         "became"             "become"
 [277] "becomes"           "becoming"           "began"
 [280] "begin"             "beginning"          "behalf"
 [283] "behaviour"         "behind"             "belief"
 [286] "believe"           "belong"             "belongs"
 [289] "benefit"           "benefits"           "besides"
 [292] "best"              "bet"                "better"
 [295] "beyond"            "bias"               "biased"
 [298] "bible"             "big"                "bigger"
 [301] "biggest"           "bill"               "bio"
 [304] "biographical"      "biography"          "birth"
 [307] "bit"               "bits"               "black"
 [310] "blame"             "blank"              "blanking"
 [313] "blatant"           "blind"              "block"
 [316] "blocked"           "blocking"           "blocks"
 [319] "blog"              "blood"              "bloody"
 [322] "blp"               "blue"               "board"
 [325] "bob"               "body"               "bold"
 [328] "book"              "books"              "border"
 [331] "borderpx"          "born"               "bot"
 [334] "bother"            "bothered"           "bottom"
 [337] "box"               "boy"                "brain"
 [340] "brand"             "break"              "breaking"
 [343] "brief"             "briefly"            "bring"
 [346] "britain"           "british"            "broad"
 [349] "broke"             "brother"            "brown"
 [352] "btw"               "buddy"              "build"
 [355] "building"          "built"              "bully"
 [358] "bullying"          "bunch"              "bush"
 [361] "business"          "busy"               "button"
 [364] "buy"               "bye"                "california"
 [367] "call"              "called"             "calls"
 [370] "calm"              "came"               "campaign"
 [373] "can"               "canada"             "canadian"
 [376] "candidate"         "cant"               "capable"
 [379] "capacity"          "capital"            "caption"
 [382] "car"               "care"               "career"
 [385] "careful"           "carefully"          "cares"
 [388] "carried"           "carry"              "case"
 [391] "cases"             "cast"               "cat"
 [394] "catch"             "category"           "catholic"
 [397] "caught"            "cause"              "caused"
 [400] "causes"            "cease"              "cellpadding"
 [403] "censor"            "censorship"         "census"
 [406] "center"            "centory"            "central"
 [409] "century"           "certain"            "certainly"
 [412] "challenge"         "challenged"         "chance"
 [415] "change"            "changed"            "changes"
```

[418] "channel"	"chapter"	"character"
[421] "characters"	"charge"	"charges"
[424] "charles"	"chart"	"chat"
[427] "check"	"checked"	"checking"
[430] "checkuser"	"cheers"	"chief"
[433] "child"	"childish"	"china"
[436] "chinese"	"choice"	"choose"
[439] "chris"	"christ"	"christian"
[442] "christianity"	"christians"	"christmas"
[445] "church"	"circumstances"	"citation"
[448] "cite"	"cited"	"cites"
[451] "citing"	"citizens"	"city"
[454] "civil"	"civility"	"claim"
[457] "claimed"	"claims"	"clarification"
[460] "clarify"	"class"	"classic"
[463] "clean"	"cleaning"	"cleanup"
[466] "clear"	"clearer"	"clearly"
[469] "click"	"clicking"	"close"
[472] "closed"	"closely"	"closer"
[475] "closing"	"club"	"clue"
[478] "coast"	"code"	"coi"
[481] "cold"	"collaboration"	"collection"
[484] "college"	"colour"	"column"
[487] "come"	"comes"	"coming"
[490] "comment"	"commentary"	"commented"
[493] "comments"	"commercial"	"committed"
[496] "committee"	"common"	"commonly"
[499] "commons"	"communicate"	"communication"
[502] "communist"	"community"	"companies"
[505] "company"	"compare"	"compared"
[508] "comparison"	"complain"	"complaint"
[511] "complete"	"completed"	"completely"
[514] "complex"	"complicated"	"comply"
[517] "comprehensive"	"compromise"	"computer"
[520] "concept"	"concern"	"concerned"
[523] "concerning"	"concerns"	"conclusion"
[526] "conclusions"	"condition"	"conditions"

[628] "csd"	"cultural"	"culture"
[631] "cup"	"curious"	"current"
[634] "currently"	"cut"	"daily"
[637] "damage"	"dangerous"	"daniel"
[640] "dare"	"dark"	"data"
[643] "date"	"dated"	"dates"
[646] "daughter"	"david"	"day"
[649] "dead"	"deal"	"dealing"
[652] "dealt"	"dear"	"death"
[655] "debate"	"dec"	"decades"
[658] "december"	"decent"	"decide"
[661] "decision"	"declared"	"decline"
[664] "declined"	"dedicated"	"deep"
[667] "deeply"	"defend"	"defending"
[670] "defense"	"define"	"definitely"
[673] "definition"	"degree"	"delay"
[676] "delete"	"deletion"	"deliberately"
[679] "demand"	"democracy"	"democratic"
[682] "demonstrate"	"denial"	"deny"
[685] "department"	"depending"	"depth"
[688] "derived"	"describe"	"description"
[691] "deserve"	"design"	"designed"
[694] "desire"	"desk"	"despite"
[697] "destroy"	"detail"	"detailed"
[700] "details"	"determine"	"determined"
[703] "develop"	"development"	"dictionary"
[706] "didnt"	"die"	"died"
[709] "diff"	"difference"	"differences"
[712] "different"	"differently"	"difficult"
[715] "diffs"	"direct"	"directed"
[718] "direction"	"directly"	"director"
[721] "directory"	"dirty"	"disagree"
[724] "disagreement"	"disambiguation"	"discovered"
[727] "discuss"	"discussion"	"disease"
[730] "disgusting"	"dislike"	"display"
[733] "displayed"	"dispute"	"disputed"
[736] "disputes"	"disruption"	"disruptive"

[529] "conduct"	"conference"	"confirm"
[532] "conflict"	"conflicts"	"conform"
[535] "conformance"	"confused"	"confusion"
[538] "congratulations"	"congress"	"connected"
[541] "connection"	"consensus"	"conservative"
[544] "consider"	"consideration"	"consistent"
[547] "consistently"	"conspiracy"	"constant"
[550] "constantly"	"constitute"	"constitution"
[553] "construction"	"constructive"	"contact"
[556] "contain"	"contemporary"	"content"
[559] "contentious"	"contents"	"contest"
[562] "contested"	"context"	"continue"
[565] "continued"	"continues"	"continuing"
[568] "contrary"	"contrast"	"contribs"
[571] "contribute"	"contribution"	"contributor"
[574] "control"	"controlled"	"controversial"
[577] "controversy"	"convention"	"conversation"
[580] "convince"	"cool"	"cooperation"
[583] "copied"	"copy"	"copyright"
[586] "copyrighted"	"copyrights"	"core"
[589] "correct"	"corrected"	"correction"
[592] "corrections"	"correctly"	"cost"
[595] "council"	"count"	"counter"
[598] "countries"	"country"	"counts"
[601] "county"	"couple"	"coupled"
[604] "course"	"court"	"cover"
[607] "coverage"	"covered"	"covers"
[610] "crazy"	"create"	"created"
[613] "creation"	"creative"	"creator"
[616] "credibility"	"credible"	"credit"
[619] "crime"	"criminal"	"criteria"
[622] "criterion"	"critical"	"criticism"
[625] "critics"	"cross"	"cry"

[739] "distinct"	"distinction"	"district"
[742] "division"	"doc"	"doctor"
[745] "document"	"documentation"	"documented"
[748] "documents"	"doesnt"	"dog"
[751] "domain"	"donâ€™t"	"done"
[754] "dont"	"double"	"doubt"
[757] "dozen"	"draft"	"drama"
[760] "draw"	"drive"	"drop"
[763] "dropdown"	"dropped"	"dubious"
[766] "dude"	"due"	"dumb"
[769] "dutch"	"earlier"	"early"
[772] "earth"	"easier"	"easily"
[775] "east"	"eastern"	"easy"
[778] "eat"	"economic"	"edit"
[781] "edited"	"edition"	"editor"
[784] "editorial"	"editors"	"edits"
[787] "editwarring"	"education"	"educational"
[790] "effect"	"effective"	"effectively"
[793] "effects"	"effort"	"efforts"
[796] "either"	"election"	"element"
[799] "elements"	"else"	"elses"
[802] "elsewhere"	"email"	"emphasis"
[805] "empire"	"encourage"	"encyclopedia"
[808] "encyclopedic"	"end"	"ended"
[811] "ending"	"ends"	"energy"
[814] "engage"	"engaged"	"engaging"
[817] "engine"	"england"	"english"
[820] "enjoy"	"enough"	"ensure"
[823] "enter"	"entire"	"entirely"
[826] "entitled"	"entity"	"entries"
[829] "entry"	"environment"	"episode"
[832] "equal"	"equally"	"equivalent"
[835] "era"	"error"	"especially"

[838] "essay"	"essentially"	"establish"
[841] "established"	"etc"	"ethnic"
[844] "ethnicity"	"europe"	"european"
[847] "even"	"event"	"eventually"
[850] "ever"	"every"	"everyone"
[853] "everything"	"everywhere"	"evidence"
[856] "evil"	"evolution"	"exact"
[859] "exactly"	"example"	"excellent"
[862] "except"	"exception"	"exchange"
[865] "exclusively"	"excuse"	"exist"
[868] "existence"	"expand"	"expansion"
[871] "expect"	"experience"	"experienced"
[874] "experiment"	"experimenting"	"expert"
[877] "expertise"	"experts"	"explain"
[880] "explanation"	"explicit"	"explicitly"
[883] "express"	"expressed"	"expression"
[886] "extended"	"extensive"	"extent"
[889] "external"	"extra"	"extreme"
[892] "extremely"	"eye"	"fac"
[895] "face"	"facebook"	"facilitate"
[898] "fact"	"factor"	"factual"
[901] "factually"	"fail"	"failed"
[904] "fails"	"failure"	"fair"
[907] "fairly"	"faith"	"fake"
[910] "fall"	"falls"	"false"
[913] "familiar"	"family"	"famous"
[916] "fan"	"fans"	"far"
[919] "fashion"	"fast"	"fat"
[922] "father"	"fault"	"favorite"
[925] "favour"	"fear"	"feature"
[928] "featured"	"features"	"february"
[931] "federal"	"feedback"	"feel"
[934] "feeling"	"feelings"	"feels"
[937] "fellow"	"felt"	"female"
[940] "fiction"	"fictional"	"field"
[943] "fight"	"fighting"	"figure"
[946] "figured"	"figures"	"file"

[1048] "getting"	"gfdl"	"girl"
[1051] "give"	"given"	"gives"
[1054] "giving"	"glad"	"global"
[1057] "goal"	"god"	"gods"
[1060] "goes"	"going"	"gold"
[1063] "gone"	"gonna"	"good"
[1066] "google"	"got"	"gotten"
[1069] "government"	"grammar"	"grand"
[1072] "grant"	"granted"	"great"
[1075] "greater"	"greatest"	"greatly"
[1078] "greece"	"greek"	"green"
[1081] "greetings"	"ground"	"grounds"
[1084] "group"	"groups"	"grow"
[1087] "guess"	"guidance"	"guide"
[1090] "guideline"	"guidelines"	"guilty"
[1093] "gun"	"guy"	"guys"
[1096] "haha"	"hair"	"half"
[1099] "hand"	"handle"	"hands"
[1102] "happen"	"happy"	"harassing"
[1105] "harassment"	"hard"	"hardly"
[1108] "harm"	"hate"	"hatred"
[1111] "head"	"heading"	"heads"
[1114] "health"	"hear"	"heart"
[1117] "heavily"	"heavy"	"heck"
[1120] "held"	"hell"	"hello"
[1123] "help"	"helped"	"helpful"
[1126] "helpme"	"helps"	"hence"
[1129] "heritage"	"hesitate"	"hey"
[1132] "hidden"	"hide"	"hiding"
[1135] "high"	"higher"	"highest"
[1138] "highly"	"hindu"	"historian"
[1141] "historic"	"historical"	"historically"
[1144] "history"	"hit"	"hitler"
[1147] "hits"	"hmm"	"hoax"
[1150] "hold"	"holder"	"holding"
[1153] "holds"	"hole"	"holocaust"
[1156] "holy"	"home"	"homosexual"

[949] "filed"	"files"	"fill"
[952] "filled"	"film"	"films"
[955] "final"	"finally"	"financial"
[958] "find"	"finding"	"fine"
[961] "finish"	"finished"	"fire"
[964] "first"	"firstly"	"fit"
[967] "five"	"fix"	"fixed"
[970] "flag"	"flow"	"focus"
[973] "focused"	"folks"	"follow"
[976] "followed"	"following"	"follows"
[979] "food"	"fool"	"football"
[982] "footnote"	"force"	"forced"
[985] "forces"	"foreign"	"forever"
[988] "forget"	"forgive"	"form"
[991] "formal"	"format"	"formatting"
[994] "formed"	"former"	"forms"
[997] "forth"	"forum"	"forward"
[1000] "fought"	"found"	"foundation"
[1003] "founded"	"founder"	"four"
[1006] "fourth"	"fox"	"france"
[1009] "frank"	"frankly"	"free"
[1012] "freedom"	"freely"	"french"
[1015] "frequently"	"friend"	"friendly"
[1018] "friends"	"fringe"	"front"
[1021] "full"	"fully"	"fun"
[1024] "function"	"fundamental"	"funny"
[1027] "furthermore"	"future"	"fyi"
[1030] "gain"	"game"	"gang"
[1033] "garbage"	"gas"	"gave"
[1036] "general"	"generally"	"generation"
[1039] "genocide"	"genre"	"genuine"
[1042] "george"	"german"	"germans"
[1045] "germany"	"get"	"gets"

[1159] "honest"	"honestly"	"hope"
[1162] "hopefully"	"hoping"	"horrible"
[1165] "horse"	"hot"	"hour"
[1168] "house"	"however"	"huge"
[1171] "huh"	"human"	"hundred"
[1174] "hurt"	"idea"	"identical"
[1177] "identified"	"identify"	"identity"
[1180] "ignorance"	"ignorant"	"ignore"
[1183] "ill"	"illegal"	"image"
[1186] "images"	"imagine"	"immediate"
[1189] "immediately"	"imo"	"impact"
[1192] "imply"	"importance"	"important"
[1195] "importantly"	"impossible"	"impression"
[1198] "improve"	"improvement"	"inaccurate"
[1201] "inappropriate"	"incident"	"include"
[1204] "inclusion"	"incorrect"	"increase"
[1207] "indeed"	"indefinitely"	"independence"
[1210] "independent"	"india"	"indian"
[1213] "indicate"	"indication"	"individual"
[1216] "individuals"	"industry"	"influence"
[1219] "influenced"	"info"	"infobox"
[1222] "inform"	"information"	"informative"
[1225] "informed"	"initial"	"initially"
[1228] "inline"	"innocent"	"input"
[1231] "insert"	"inside"	"insist"
[1234] "instance"	"instances"	"instead"
[1237] "institute"	"instructions"	"insult"
[1240] "insulting"	"insults"	"integrity"
[1243] "intellectual"	"intelligence"	"intelligent"
[1246] "intend"	"intent"	"intention"
[1249] "interest"	"interested"	"interesting"
[1252] "interests"	"internal"	"international"
[1255] "internet"	"interpretation"	"intervention"

[1258] "interview"	"intro"	"introduce"
[1261] "introduction"	"invented"	"investigation"
[1264] "invite"	"invited"	"involve"
[1267] "involvement"	"ips"	"iran"
[1270] "iranian"	"iraq"	"ireland"
[1273] "irish"	"irrelevant"	"isbn"
[1276] "islam"	"islamic"	"island"
[1279] "isnt"	"israel"	"israeli"
[1282] "issue"	"issued"	"issues"
[1285] "italian"	"italy"	"item"
[1288] "items"	"ive"	"jack"
[1291] "james"	"january"	"japan"
[1294] "japanese"	"jesus"	"jewish"
[1297] "jim"	"jimbo"	"jimmy"
[1300] "job"	"joe"	"john"
[1303] "join"	"joined"	"joke"
[1306] "joseph"	"journal"	"judge"
[1309] "judgement"	"july"	"jump"
[1312] "june"	"just"	"justice"
[1315] "justification"	"justify"	"keep"
[1318] "keeping"	"keeps"	"kept"
[1321] "key"	"kid"	"kids"
[1324] "kill"	"killed"	"killing"
[1327] "kind"	"kinda"	"kindly"
[1330] "kinds"	"king"	"kingdom"
[1333] "knew"	"know"	"knowing"
[1336] "knowledge"	"known"	"knows"
[1339] "label"	"lack"	"lacking"
[1342] "lady"	"land"	"language"
[1345] "large"	"largely"	"larger"
[1348] "largest"	"last"	"late"
[1351] "later"	"latest"	"latin"
[1354] "latter"	"laugh"	"law"
[1357] "laws"	"lazy"	"lead"
[1360] "leader"	"leaders"	"leading"
[1363] "leads"	"league"	"learn"
[1366] "least"	"leave"	"leaves"

[1468] "map"	"maps"	"mar"
[1471] "march"	"mark"	"marked"
[1474] "market"	"marriage"	"married"
[1477] "martin"	"mass"	"massacre"
[1480] "massive"	"master"	"match"
[1483] "matches"	"mate"	"material"
[1486] "materials"	"matter"	"matters"
[1489] "may"	"maybe"	"mean"
[1492] "meaning"	"meaningful"	"meaningless"
[1495] "means"	"meant"	"meantime"
[1498] "meanwhile"	"measure"	"media"
[1501] "mediation"	"medical"	"medicine"
[1504] "meet"	"meeting"	"meets"
[1507] "member"	"memory"	"men"
[1510] "mental"	"mention"	"mentioned"
[1513] "mentions"	"mere"	"merely"
[1516] "merge"	"merit"	"merits"
[1519] "mess"	"message"	"met"
[1522] "metal"	"method"	"michael"
[1525] "middle"	"might"	"mike"
[1528] "military"	"million"	"millions"
[1531] "mind"	"minds"	"mine"
[1534] "minimum"	"minister"	"minor"
[1537] "minority"	"minute"	"misleading"
[1540] "miss"	"missed"	"missing"
[1543] "mission"	"mistake"	"mistaken"
[1546] "mistakes"	"misunderstanding"	"mixed"
[1549] "model"	"modern"	"modified"
[1552] "moment"	"money"	"month"
[1555] "moral"	"moreover"	"morning"
[1558] "mos"	"mostly"	"mother"
[1561] "mouth"	"move"	"moved"
[1564] "movement"	"moves"	"movie"
[1567] "moving"	"much"	"multiple"

[1369] "leaving"	"led"	"lede"
[1372] "left"	"legal"	"legitimate"
[1375] "length"	"less"	"let"
[1378] "lets"	"letter"	"letting"
[1381] "level"	"levels"	"liar"
[1384] "liberal"	"library"	"license"
[1387] "licensed"	"licensing"	"lie"
[1390] "lies"	"life"	"light"
[1393] "like"	"liked"	"likely"
[1396] "likes"	"likewise"	"limit"
[1399] "limited"	"limits"	"line"
[1402] "lines"	"link"	"linked"
[1405] "linking"	"links"	"list"
[1408] "listed"	"listen"	"listing"
[1411] "lists"	"literally"	"literature"
[1414] "little"	"live"	"lived"
[1417] "lives"	"living"	"local"
[1420] "located"	"location"	"locked"
[1423] "log"	"logged"	"logic"
[1426] "logical"	"logo"	"lol"
[1429] "london"	"long"	"longer"
[1432] "look"	"looked"	"looking"
[1435] "looks"	"lord"	"lose"
[1438] "loss"	"lost"	"lot"
[1441] "lots"	"love"	"low"
[1444] "lower"	"luck"	"lying"
[1447] "machine"	"mad"	"made"
[1450] "magazine"	"mail"	"main"
[1453] "mainly"	"mainstream"	"maintain"
[1456] "major"	"majority"	"make"
[1459] "makes"	"making"	"male"
[1462] "man"	"managed"	"management"
[1465] "manner"	"manual"	"many"

[1570] "murder"	"music"	"musical"
[1573] "muslim"	"muslims"	"must"
[1576] "name"	"named"	"names"
[1579] "naming"	"nasty"	"nation"
[1582] "national"	"nationalist"	"nationality"
[1585] "nations"	"native"	"natural"
[1588] "naturally"	"nature"	"nazi"
[1591] "nazis"	"near"	"nearly"
[1594] "necessarily"	"necessary"	"need"
[1597] "needed"	"needs"	"negative"
[1600] "neither"	"net"	"network"
[1603] "neutral"	"neutrality"	"never"
[1606] "nevertheless"	"new"	"newcomers"
[1609] "news"	"newspaper"	"next"
[1612] "nice"	"night"	"nobody"
[1615] "nominate"	"nomination"	"non"
[1618] "none"	"nonfree"	"nonsense"
[1621] "normal"	"normally"	"north"
[1624] "northern"	"notability"	"notable"
[1627] "note"	"noted"	"notes"
[1630] "nothing"	"notice"	"noticeboard"
[1633] "noticed"	"notices"	"notification"
[1636] "notion"	"novel"	"november"
[1639] "now"	"nowhere"	"npov"
[1642] "number"	"numbers"	"numerous"
[1645] "obama"	"object"	"objection"
[1648] "objective"	"obscure"	"obtained"
[1651] "obvious"	"obviously"	"occupation"
[1654] "occur"	"occurred"	"october"
[1657] "odd"	"offended"	"offense"
[1660] "offensive"	"offer"	"offered"
[1663] "office"	"official"	"officially"
[1666] "often"	"oil"	"okay"

[1669] "old"	"older"	"one"
[1672] "ones"	"ongoing"	"online"
[1675] "onto"	"open"	"opened"
[1678] "opening"	"operation"	"opinion"
[1681] "opportunity"	"oppose"	"opposite"
[1684] "opposition"	"option"	"options"
[1687] "order"	"organisation"	"origin"
[1690] "original"	"originally"	"origins"
[1693] "others"	"otherwise"	"outcome"
[1696] "outright"	"outside"	"overall"
[1699] "overview"	"owned"	"owner"
[1702] "ownership"	"padding"	"page"
[1705] "pages"	"paid"	"pain"
[1708] "pakistan"	"paper"	"papers"
[1711] "paragraph"	"parents"	"park"
[1714] "part"	"participants"	"participate"
[1717] "participation"	"particular"	"particularly"
[1720] "parties"	"parts"	"party"
[1723] "pass"	"passage"	"passed"
[1726] "passing"	"past"	"paste"
[1729] "pathetic"	"patience"	"pattern"
[1732] "paul"	"pay"	"peace"
[1735] "peer"	"people"	"peoples"
[1738] "per"	"perfect"	"perfectly"
[1741] "perform"	"performance"	"perhaps"
[1744] "period"	"permanent"	"permanently"
[1747] "permission"	"permitted"	"persian"
[1750] "persistent"	"person"	"personal"
[1753] "personality"	"personally"	"persons"
[1756] "perspective"	"peter"	"petty"
[1759] "philosophy"	"phone"	"photo"
[1762] "phrase"	"phrases"	"physical"
[1765] "physics"	"pic"	"pick"
[1768] "picked"	"picture"	"pictures"
[1771] "piece"	"pieces"	"pillars"
[1774] "place"	"placed"	"places"
[1777] "placing"	"plain"	"plan"

[1879] "promote"	"promotes"	"promoting"
[1882] "promotion"	"promotional"	"proof"
[1885] "propaganda"	"proper"	"properly"
[1888] "property"	"proposal"	"propose"
[1891] "proposed"	"proposing"	"prose"
[1894] "protect"	"protected"	"protecting"
[1897] "protection"	"proud"	"prove"
[1900] "provide"	"province"	"proxy"
[1903] "public"	"publication"	"publish"
[1906] "publisher"	"publishing"	"pull"
[1909] "pump"	"punk"	"puppet"
[1912] "puppetry"	"pure"	"purely"
[1915] "purpose"	"purposes"	"push"
[1918] "pushing"	"put"	"puts"
[1921] "putting"	"qualified"	"qualify"
[1924] "quality"	"queen"	"question"
[1927] "questionable"	"questions"	"quick"
[1930] "quickly"	"quit"	"quite"
[1933] "quotation"	"quote"	"quoted"
[1936] "quotes"	"quoting"	"race"
[1939] "racial"	"racism"	"racist"
[1942] "radio"	"raise"	"raised"
[1945] "ran"	"random"	"range"
[1948] "rape"	"rare"	"rarely"
[1951] "rate"	"rather"	"rational"
[1954] "rationale"	"reach"	"reached"
[1957] "reaction"	"read"	"reader"
[1960] "reading"	"reads"	"ready"
[1963] "real"	"realise"	"reality"
[1966] "realize"	"really"	"reason"
[1969] "reasonable"	"reasonably"	"reasoning"
[1972] "reasons"	"recall"	"receive"
[1975] "recent"	"recently"	"recognition"
[1978] "recognize"	"recommend"	"recommended"
[1981] "record"	"recorded"	"records"
[1984] "recreate"	"red"	"redirect"
[1987] "redirected"	"redirects"	"reduce"

[1780] "planet"	"planning"	"plans"
[1783] "play"	"played"	"player"
[1786] "players"	"playing"	"please"
[1789] "plenty"	"plot"	"plus"
[1792] "point"	"pointed"	"pointless"
[1795] "points"	"poland"	"police"
[1798] "policies"	"policy"	"polish"
[1801] "polite"	"political"	"politically"
[1804] "politics"	"poll"	"poor"
[1807] "poorly"	"pop"	"popular"
[1810] "population"	"portal"	"portion"
[1813] "position"	"positions"	"positive"
[1816] "possibility"	"possible"	"possibly"
[1819] "post"	"posted"	"posting"
[1822] "posts"	"potential"	"potentially"
[1825] "pov"	"power"	"powerful"
[1828] "powers"	"practice"	"practices"
[1831] "preceding"	"precise"	"precisely"
[1834] "prefer"	"prepared"	"presence"
[1837] "present"	"presented"	"president"
[1840] "press"	"presumably"	"pretend"
[1843] "pretty"	"prevent"	"previous"
[1846] "previously"	"primarily"	"primary"
[1849] "prime"	"principle"	"principles"
[1852] "print"	"printed"	"prior"
[1855] "private"	"pro"	"probably"
[1858] "problem"	"problematic"	"procedure"
[1861] "proceed"	"process"	"prod"
[1864] "produce"	"produced"	"product"
[1867] "production"	"productive"	"products"
[1870] "professional"	"professor"	"profile"
[1873] "program"	"progress"	"project"
[1876] "projects"	"prominent"	"promise"

[1990] "redundant"	"ref"	"refer"
[1993] "reference"	"referenced"	"references"
[1996] "referencing"	"referred"	"referring"
[1999] "refers"	"reflect"	"refrain"
[2002] "refs"	"refuse"	"regard"
[2005] "regarded"	"regardless"	"regards"
[2008] "region"	"regional"	"regions"
[2011] "register"	"registered"	"regular"
[2014] "regularly"	"rejected"	"related"
[2017] "relating"	"relation"	"relations"
[2020] "relationship"	"relative"	"relatively"
[2023] "release"	"released"	"relevance"
[2026] "relevant"	"reliability"	"reliable"
[2029] "religion"	"religious"	"remain"
[2032] "remaining"	"remains"	"remark"
[2035] "remarks"	"remember"	"remind"
[2038] "removal"	"remove"	"removed"
[2041] "removing"	"renamed"	"render"
[2044] "repeat"	"repeated"	"repeatedly"
[2047] "replace"	"replacement"	"replied"
[2050] "reply"	"report"	"reported"
[2053] "reports"	"represent"	"representation"
[2056] "representative"	"represented"	"represents"
[2059] "republic"	"republican"	"reputable"
[2062] "reputation"	"request"	"requested"
[2065] "requests"	"require"	"requirement"
[2068] "requires"	"research"	"resolution"
[2071] "resolve"	"resolved"	"resource"
[2074] "resources"	"respect"	"respected"
[2077] "respond"	"response"	"responsibility"
[2080] "responsible"	"rest"	"restore"
[2083] "result"	"results"	"retarded"
[2086] "return"	"returned"	"reversion"
[2089] "revert"	"reverted"	"reverts"
[2092] "review"	"reviewed"	"reviewer"
[2095] "reviews"	"revision"	"rewrite"
[2098] "rfa"	"rfc"	"richard"

[2101] "rid"	"ridiculous"	"right"
[2104] "rights"	"rise"	"risk"
[2107] "river"	"road"	"robert"
[2110] "rock"	"role"	"roman"
[2113] "room"	"round"	"route"
[2116] "royal"	"rubbish"	"rude"
[2119] "rule"	"rules"	"run"
[2122] "running"	"runs"	"russia"
[2125] "russian"	"sad"	"sadly"
[2128] "safe"	"said"	"sake"
[2131] "sales"	"san"	"sandbox"
[2134] "satisfy"	"save"	"saved"
[2137] "saw"	"say"	"saying"
[2140] "says"	"scale"	"scene"
[2143] "scheme"	"scholar"	"scholarly"
[2146] "school"	"science"	"scientific"
[2149] "scientist"	"scope"	"screen"
[2152] "script"	"sea"	"search"
[2155] "searching"	"season"	"second"
[2158] "secondary"	"secondly"	"seconds"
[2161] "secret"	"section"	"sections"
[2164] "security"	"see"	"seeing"
[2167] "seek"	"seeking"	"seem"
[2170] "seemed"	"seems"	"seen"
[2173] "sees"	"selected"	"selecting"
[2176] "self"	"sell"	"semiprotected"
[2179] "send"	"sense"	"sentence"
[2182] "separate"	"september"	"serbian"
[2185] "series"	"serious"	"seriously"
[2188] "serve"	"served"	"serves"
[2191] "service"	"services"	"set"
[2194] "setting"	"settled"	"seven"
[2197] "several"	"sexual"	"shall"
[2200] "shame"	"shape"	"share"
[2203] "shared"	"short"	"shortly"
[2206] "shot"	"show"	"showed"
[2209] "shows"	"shut"	"sick"

[2323] "start"	"started"	"starts"
[2326] "state"	"stated"	"statement"
[2329] "statements"	"states"	"stating"
[2332] "station"	"statistics"	"status"
[2335] "stay"	"step"	"steps"
[2338] "steve"	"stick"	"still"
[2341] "stone"	"stop"	"stopped"
[2344] "story"	"straight"	"strange"
[2347] "street"	"strict"	"strictly"
[2350] "strike"	"strong"	"strongly"
[2353] "structure"	"stub"	"stuck"
[2356] "student"	"studies"	"study"
[2359] "stuff"	"stupid"	"style"
[2362] "styleverticalaligntop"	"stylewidth"	"subject"
[2365] "subjective"	"subjects"	"submission"
[2368] "submit"	"submitted"	"subsection"
[2371] "subsequent"	"substantial"	"substantially"
[2374] "success"	"successful"	"suddenly"
[2377] "sufficient"	"suggest"	"suggestion"
[2380] "suit"	"suitable"	"summaries"
[2383] "summary"	"summer"	"sun"
[2386] "super"	"supply"	"support"
[2389] "supported"	"supporters"	"supports"
[2392] "suppose"	"supposed"	"supposedly"
[2395] "supreme"	"sure"	"surely"
[2398] "surprise"	"surprised"	"survey"
[2401] "suspect"	"suspected"	"synthesis"
[2404] "system"	"tab"	"table"
[2407] "tactics"	"tag"	"tagged"
[2410] "tagging"	"tags"	"take"
[2413] "taken"	"takes"	"taking"
[2416] "talk"	"talkâ"	"talkcontribs"
[2419] "talked"	"talkpage"	"talks"
[2422] "target"	"task"	"taught"
[2425] "teach"	"teacher"	"team"
[2428] "teams"	"technical"	"technically"
[2431] "technology"	"television"	"tell"

[2212] "side"	"sides"	"sign"
[2215] "signature"	"signed"	"significance"
[2218] "significant"	"significantly"	"signing"
[2221] "silly"	"similar"	"similarly"
[2224] "simple"	"simply"	"since"
[2227] "sincerely"	"single"	"sir"
[2230] "sister"	"sit"	"site"
[2233] "sites"	"sitting"	"situation"
[2236] "six"	"size"	"skills"
[2239] "sleep"	"slightly"	"slow"
[2242] "small"	"smaller"	"smart"
[2245] "smith"	"social"	"society"
[2248] "sock"	"sockpuppet"	"sockpuppetry"
[2251] "software"	"sold"	"sole"
[2254] "solely"	"solid"	"solution"
[2257] "solve"	"somebody"	"somehow"
[2260] "someones"	"something"	"sometimes"
[2263] "somewhat"	"somewhere"	"son"
[2266] "song"	"soon"	"sorry"
[2269] "sort"	"sorts"	"sound"
[2272] "sounds"	"source"	"sourced"
[2275] "sources"	"south"	"southern"
[2278] "soviet"	"space"	"spam"
[2281] "spamming"	"spanish"	"speak"
[2284] "speakers"	"speaking"	"speaks"
[2287] "special"	"species"	"specific"
[2290] "specifically"	"specified"	"specifies"
[2293] "specify"	"speculation"	"speech"
[2296] "speed"	"speedily"	"speedy"
[2299] "spell"	"spelled"	"spelling"
[2302] "spend"	"spending"	"spent"
[2305] "spirit"	"split"	"spoke"
[2308] "spoken"	"sports"	"spot"
[2311] "spread"	"stable"	"staff"
[2314] "stage"	"stalking"	"stance"
[2317] "stand"	"standard"	"standards"
[2320] "standing"	"stands"	"star"

[2434] "telling"	"tells"	"template"
[2437] "ten"	"tend"	"term"
[2440] "terminology"	"terms"	"terrible"
[2443] "territory"	"terrorism"	"terrorist"
[2446] "test"	"testing"	"tests"
[2449] "texas"	"text"	"texts"
[2452] "thank"	"thanks"	"thats"
[2455] "theories"	"theory"	"therefore"
[2458] "thing"	"things"	"think"
[2461] "thinking"	"thinks"	"third"
[2464] "thomas"	"thoroughly"	"though"
[2467] "thought"	"thoughts"	"thousand"
[2470] "thousands"	"thread"	"threat"
[2473] "threaten"	"threatened"	"threatening"
[2476] "threats"	"three"	"threerevert"
[2479] "throughout"	"throw"	"thus"
[2482] "tildes"	"till"	"time"
[2485] "times"	"tiny"	"tips"
[2488] "tired"	"title"	"titled"
[2491] "titles"	"today"	"together"
[2494] "told"	"tom"	"tomorrow"
[2497] "tone"	"tonight"	"tony"
[2500] "took"	"tool"	"tools"
[2503] "top"	"topic"	"total"
[2506] "totally"	"touch"	"tough"
[2509] "toward"	"town"	"track"
[2512] "trade"	"tradition"	"traditional"
[2515] "training"	"translate"	"translation"
[2518] "treat"	"treated"	"treatment"
[2521] "tree"	"trial"	"tried"
[2524] "tries"	"trip"	"trivia"
[2527] "trivial"	"troll"	"trolling"
[2530] "trolls"	"trouble"	"true"
[2533] "truly"	"trust"	"truth"
[2536] "try"	"trying"	"turkey"
[2539] "turkish"	"turks"	"turn"
[2542] "turned"	"turns"	"tutorial"

```
[2545] "twice"                "two"                "type"
[2548] "types"                "typical"            "typically"
[2551] "typing"               "ugly"               "ultimately"
[2554] "unable"               "unacceptable"       "unbiased"
[2557] "unblock"              "uncivil"            "unclear"
[2560] "unconstructive"       "understand"         "understanding"
[2563] "understood"           "undo"               "undoing"
[2566] "undue"                "unfair"             "unfortunately"
[2569] "union"                "unique"             "unit"
[2572] "united"               "universal"          "universe"
[2575] "university"           "unknown"            "unless"
[2578] "unlike"               "unlikely"           "unnecessary"
[2581] "unreferenced"         "unrelated"          "unreliable"
[2584] "unsigned"             "unsourced"          "untrue"
[2587] "update"               "updated"            "upload"
[2590] "uploaded"             "upon"               "upset"
[2593] "url"                  "usa"                "usage"
[2596] "use"                  "used"               "useful"
[2599] "useless"              "user"               "username"
[2602] "userpage"             "users"              "uses"
[2605] "using"                "usual"              "usually"
[2608] "utc"                  "utterly"            "vague"
[2611] "valid"                "validity"           "valuable"
[2614] "value"                "van"                "vandal"
[2617] "vandalism"            "vandalize"          "vandals"
[2620] "variety"              "various"            "vast"
[2623] "vehicle"              "verifiability"      "verifiable"
[2626] "verification"         "verify"             "version"
[2629] "versions"             "verticalaligntop"   "via"
[2632] "victim"               "video"              "view"
[2635] "viewed"               "viewpoint"          "views"
[2638] "village"              "violate"            "violation"
[2641] "violence"             "virtually"          "visible"
[2644] "visit"                "voice"              "volume"
[2647] "vote"                 "voted"              "votes"
[2650] "wait"                 "waiting"            "wales"
[2653] "walk"                 "wall"               "wanna"
```

```
[2656] "want"                 "wanted"             "wanting"
[2659] "wants"                "war"                "warn"
[2662] "warned"               "warning"            "warnings"
[2665] "warrant"              "warring"            "wars"
[2668] "washington"           "waste"              "wasting"
[2671] "watch"                "watching"           "watchlist"
[2674] "water"                "way"                "ways"
[2677] "weak"                 "web"                "website"
[2680] "week"                 "weekend"            "weight"
[2683] "weird"                "welcome"            "welcomeâ"
[2686] "well"                 "wellknown"          "went"
[2689] "west"                 "western"            "whatever"
[2692] "whats"                "whatsoever"         "whenever"
[2695] "whereas"              "whether"            "whilst"
[2698] "white"                "whoever"            "whole"
[2701] "whose"                "wide"               "widely"
[2704] "width"                "wife"               "wiki"
[2707] "wikimedia"            "wikipedia"          "wikipediaarticles"
[2710] "wikipediaimage"       "wikipedian"         "wikipedians"
[2713] "wikipediaquestions"   "wikipedias"         "wikipediawikiproject"
[2716] "wikiproject"          "wikis"              "wild"
[2719] "will"                 "william"            "willing"
[2722] "win"                  "window"             "wing"
[2725] "wise"                 "wish"               "wishes"
[2728] "within"               "without"            "wizard"
[2731] "woman"                "women"              "won"
[2734] "wonder"               "wonderful"          "wondering"
[2737] "wont"                 "word"               "wording"
[2740] "words"                "work"               "worked"
[2743] "working"              "works"              "world"
[2746] "worlds"               "worldwide"          "worry"
[2749] "worse"                "worst"              "worth"
[2752] "worthless"            "worthy"             "wow"
[2755] "wpblp"                "wpnpov"             "wprs"
[2758] "wpv"                  "write"              "writer"
```

```
[2761] "writes"               "writing"            "writings"
[2764] "written"              "wrong"              "wrote"
[2767] "yeah"                 "year"               "yes"
[2770] "yesterday"            "yet"                "york"
[2773] "young"                "youtube"            "zero"
```

B.2.10 Inspecting labels

```
> labelscolumn<-textdata[,3:8]
> labelstrain<-labelscolumn[1:106381,]
> labelstest<-labelscolumn[106382:159570,]
> vec1tot106381<-c(1:106381)
> labeltoxic<-vec1tot106381[ifelse(labelstrain[,1]==1,TRUE,FALSE)]
> labelseveretoxic<-vec1tot106381[ifelse(labelstrain[,2]==1,TRUE,FALSE)]
> labelobscene<-vec1tot106381[ifelse(labelstrain[,3]==1,TRUE,FALSE)]
> labelthreat<-vec1tot106381[ifelse(labelstrain[,4]==1,TRUE,FALSE)]
> labelinsult<-vec1tot106381[ifelse(labelstrain[,5]==1,TRUE,FALSE)]
> labelidentityhate<-vec1tot106381[ifelse(labelstrain[,6]==1,TRUE,FALSE)]

> fix(onlylabel)
function (x,ids)
{
n<-length(ids)
clearvec<-rep("",n)
for(j in 1:n){
clearvec[j]<-x[[ids[j]]][1]$content}
clearvec<-VectorSource(clearvec)
clearvec<-VCorpus(clearvec)
return(clearvec)
}
```

```
# Toxic
> onlytoxiclabelcorpus<-onlylabel(x=cleansynonymsander,ids=labeltoxic)
> onlytoxiclabelTDM<-TermDocumentMatrix(onlytoxiclabelcorpus)
> onlytoxiclabelfreqs<-sort((rowSums(ifelse((as.matrix(onlytoxiclabelTDM))
>=1,1,0)))/10226*100,decreasing=TRUE)
# Severe Toxic
> onlyseveretoxiclabelcorpus<-onlylabel(x=cleansynonymsander,ids=labelseveretoxic)
> onlyseveretoxiclabelTDM<-TermDocumentMatrix(onlyseveretoxiclabelcorpus)
> onlyseveretoxiclabelfreqs<-sort((rowSums(ifelse((as.matrix(onlyseveretoxiclabelTDM))
>=1,1,0)))/1052*100,decreasing=TRUE)
# Obscene
> onlyobscenelabelcorpus<-onlylabel(x=cleansynonymsander,ids=labelobscene)
> onlyobscenelabelTDM<-TermDocumentMatrix(onlyobscenelabelcorpus)
> onlyobscenelabelfreqs<-sort((rowSums(ifelse((as.matrix(onlyobscenelabelTDM))
>=1,1,0)))/5639*100,decreasing=TRUE)
# Threat
> onlythreatlabelcorpus<-onlylabel(x=cleansynonymsander,ids=labelthreat)
> onlythreatlabelTDM<-TermDocumentMatrix(onlythreatlabelcorpus)
> onlythreatlabelfreqs<-sort((rowSums(ifelse((as.matrix(onlythreatlabelTDM))
>=1,1,0)))/331*100,decreasing=TRUE)
# Insult
> onlyinsultlabelcorpus<-onlylabel(x=cleansynonymsander,ids=labelinsult)
> onlyinsultlabelTDM<-TermDocumentMatrix(onlyinsultlabelcorpus)
> onlyinsultlabelfreqs<-sort((rowSums(ifelse((as.matrix(onlyinsultlabelTDM))
>=1,1,0)))/5270*100,decreasing=TRUE)
# Identity hate
> onlyidentityhatelabelcorpus<-onlylabel(x=cleansynonymsander,ids=labelidentityhate)
> onlyidentityhatelabelTDM<-TermDocumentMatrix(onlyidentityhatelabelcorpus)
> onlyidentityhatelabelfreqs<-sort((rowSums(ifelse((as.matrix(onlyidentityhatelabelTDM))
>=1,1,0)))/911*100,decreasing=TRUE)
```

B.2.11 Constructing word clouds

```
> library(wordcloud)
> library(SnowballC)
> wordcloud(names(onlytoxiclabelfreqs),onlytoxiclabelfreqs,max.words=80,
min.freq=1,random.order=FALSE,rot.per=0.35,colors=brewer.pal(8,"Dark2"))
> wordcloud(names(onlyseveretoxiclabelfreqs),onlyseveretoxiclabelfreqs,max.words=80,
min.freq=1,random.order=FALSE,rot.per=0.35,colors=brewer.pal(8,"Dark2"))
> wordcloud(names(onlyobscenelabelfreqs),onlyobscenelabelfreqs,max.words=80,
min.freq=1,random.order=FALSE,rot.per=0.35,colors=brewer.pal(8,"Dark2"))
> wordcloud(names(onlythreatlabelfreqs),onlythreatlabelfreqs,max.words=80,
min.freq=1,random.order=FALSE,rot.per=0.35,colors=brewer.pal(8,"Dark2"))
> wordcloud(names(onlyinsultlabelfreqs),onlyinsultlabelfreqs,max.words=80,
min.freq=1,random.order=FALSE,rot.per=0.35,colors=brewer.pal(8,"Dark2"))
> wordcloud(names(onlyidentityhatelabelfreqs),onlyidentityhatelabelfreqs,max.words=80,
min.freq=1,random.order=FALSE,rot.per=0.35,colors=brewer.pal(8,"Dark2"))
```

B.2.12 Additional terms combined

```
synonymsnogander<-list(list(word="awww",
syns=c("awwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww",
"awwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww"))
,list(word="brain",syns=c("brains"))
,list(word="bullied",syns=c("bullying"))
,list(word="computer",syns=c("computers","computa"))
,list(word="coward",syns=c("cowards"))
,list(word="delete",syns=c("deletes","delte","deletin"))
,list(word="father",syns=c("farthers","dad","dads","daddy","daddys"))
,list(word="fascist",syns=c("fascists"))
,list(word="friend",syns=c("fiend"))
,list(word="friends",syns=c("freinds"))
,list(word="gonna",syns=c("gona","gunna"))
,list(word="gun",syns=c("guns"))
,list(word="haha",syns=c("hahaha","hahahahahaha"))
,list(word="hang",syns=c("hanging"))
,list(word="hates",syns=c("hated"))
,list(word="little",syns=c("lil"))
,list(word="looking",syns=c("looken"))
,list(word="making",syns=c("makin"))
,list(word="monkey",syns=c("monkeys"))
,list(word="mother",syns=c("moms","mothers",
"mommmmmmmmmmmmm","mums","mutha"))
,list(word="nerd",syns=c("nerds"))
,list(word="people",syns=c("peple"))
,list(word="piece",syns=c("peice"))
,list(word="pig",syns=c("pigs"))
,list(word="proxy",syns=c("proxies"))
,list(word="rape",syns=c("raped","raping"))
,list(word="retard",syns=c("retards"))
,list(word="sooo",syns=c("soooo"))
,list(word="sticking",syns=c("stikin"))
,list(word="well",syns=c("weeeelllll"))
,list(word="yeah",syns=c("yea","yeh","yeahhhhh"))
)
```

B.2.13 Additional terms removed

```
> all_stops<-c("arent","didnt","cant","doesnt","dont","ive","shes",
"becaus","urself","wasnt","wont","youre")
```

B.2.14 Final input variables

```
  [1] "aanstoot1"     "aanstoot10"    "aanstoot100"   "aanstoot101"
  [5] "aanstoot102"   "aanstoot103"   "aanstoot104"   "aanstoot11"
  [9] "aanstoot12"    "aanstoot13"    "aanstoot14"    "aanstoot15"
 [13] "aanstoot16"    "aanstoot17"    "aanstoot18"    "aanstoot19"
 [17] "aanstoot2"     "aanstoot20"    "aanstoot22"    "aanstoot23"
 [21] "aanstoot24"    "aanstoot25"    "aanstoot26"    "aanstoot27"
 [25] "aanstoot28"    "aanstoot29"    "aanstoot3"     "aanstoot30"
 [29] "aanstoot31"    "aanstoot32"    "aanstoot33"    "aanstoot34"
 [33] "aanstoot35"    "aanstoot36"    "aanstoot37"    "aanstoot38"
 [37] "aanstoot39"    "aanstoot4"     "aanstoot40"    "aanstoot41"
 [41] "aanstoot42"    "aanstoot43"    "aanstoot44"    "aanstoot45"
 [45] "aanstoot46"    "aanstoot47"    "aanstoot48"    "aanstoot49"
 [49] "aanstoot5"     "aanstoot50"    "aanstoot51"    "aanstoot52"
 [53] "aanstoot53"    "aanstoot54"    "aanstoot55"    "aanstoot56"
 [57] "aanstoot57"    "aanstoot58"    "aanstoot59"    "aanstoot6"
 [61] "aanstoot60"    "aanstoot61"    "aanstoot62"    "aanstoot63"
 [65] "aanstoot64"    "aanstoot65"    "aanstoot66"    "aanstoot67"
 [69] "aanstoot68"    "aanstoot69"    "aanstoot7"     "aanstoot70"
 [73] "aanstoot71"    "aanstoot72"    "aanstoot73"    "aanstoot74"
 [77] "aanstoot75"    "aanstoot76"    "aanstoot77"    "aanstoot78"
 [81] "aanstoot79"    "aanstoot8"     "aanstoot80"    "aanstoot81"
 [85] "aanstoot82"    "aanstoot83"    "aanstoot84"    "aanstoot85"
 [89] "aanstoot86"    "aanstoot87"    "aanstoot88"    "aanstoot89"
 [93] "aanstoot9"     "aanstoot90"    "aanstoot91"    "aanstoot92"
 [97] "aanstoot93"    "aanstoot94"    "aanstoot95"    "aanstoot96"
[101] "aanstoot97"    "aanstoot98"    "aanstoot99"    "adult"
[105] "ahead"         "aids"          "aint"          "alive"
[109] "alot"          "america"       "americans"     "andrew"
[113] "angry"         "animal"        "annoying"      "anthony"
[117] "antisemite"    "antisemitic"   "arrest"        "arrogant"
[121] "ashamed"       "asian"         "awww"          "babies"
[125] "baby"          "bag"           "ban"           "banning"
[129] "basement"      "beat"          "bed"           "bet"
[133] "bible"         "big"           "bigger"        "biggest"
[137] "billcj"        "billcjs"       "bite"          "black"
[141] "blacks"        "blah"          "blank"         "blocking"
[145] "blood"         "bloody"        "body"          "bored"
[149] "boss"          "bout"          "boy"           "boyfriend"
[153] "bradbury"      "brain"         "britain"       "bro"
[157] "buddy"         "bull"          "bully"         "bunch"
[161] "burn"          "burned"        "bush"          "butt"
[165] "bye"           "cancer"        "car"           "cares"
[169] "catholic"      "cease"         "censor"        "censoring"
[173] "censorship"    "century"       "chaaaaa"       "chair"
[177] "chicken"       "child"         "childish"      "chinese"
[181] "choke"         "christ"        "christian"     "christians"
[185] "cleaning"      "clown"         "come"          "commie"
[189] "commit"        "computer"      "continues"     "cool"
[193] "corrupt"       "coward"        "cowardly"      "crack"
[197] "crazy"         "criminal"      "cry"           "crying"
[201] "cult"          "cultural"      "cut"           "cuz"
[205] "dare"          "dat"           "daughter"      "dead"
[209] "death"         "deep"          "defending"     "deserve"
[213] "destroy"       "die"           "diego"         "dies"
[217] "dildo"         "dirty"         "dis"           "disease"
[221] "disgrace"      "disgusting"    "dog"           "donkey"
[225] "dragon"        "dreadstar"     "drink"         "drown"
[229] "duck"          "dude"          "dumb"          "dutch"
[233] "ear"           "earth"         "eat"           "eating"
[237] "educational"   "ego"           "err"           "europeans"
[241] "everyday"      "everywhere"    "evil"          "express"
[245] "extended"      "face"          "facebook"      "fake"
[249] "falls"         "family"        "fascist"       "fat"
[253] "father"        "favour"        "female"        "filled"
[257] "filter"        "filthy"        "fing"          "finger"
[261] "finished"      "fire"          "flaming"       "flying"
[265] "fool"          "foolish"       "fools"         "forces"
[269] "forever"       "fought"        "founder"       "freak"
[273] "freaking"      "friends"       "front"         "garbage"
[277] "gas"           "geek"          "geeky"         "genocide"
[281] "germans"       "germany"       "giant"         "girl"
[285] "girlfriend"    "god"           "going"         "gonna"
[289] "goodbye"       "grave"         "greatest"      "gross"
[293] "ground"        "grow"          "gun"           "guy"
[297] "hack"          "haha"          "hair"          "hairy"
[301] "hands"         "hang"          "harass"        "harassing"
[305] "harry"         "hate"          "hates"         "hatred"
[309] "head"          "heart"         "heck"          "hell"
[313] "heritage"      "hes"           "hey"           "hiding"
[317] "hire"          "hit"           "hitler"        "hole"
[321] "holocaust"     "homie"         "homosexual"    "homosexuality"
[325] "homosexuals"   "hope"          "hoping"        "horrible"
[329] "horse"         "hot"           "house"         "huge"
[333] "huh"           "hunt"          "hurt"          "hypocrite"
[337] "idiotic"       "ignorant"      "ill"           "ima"
[341] "imbecile"      "immature"      "indian"        "indians"
[345] "insane"        "insult"        "insulted"      "ips"
[349] "iran"          "iranian"       "iraq"          "irish"
[353] "islam"         "islamic"       "jack"          "jesus"
[357] "jewish"        "jim"           "jimmy"         "joke"
[361] "jump"          "jus"           "kick"          "kicked"
[365] "kid"           "kidding"       "kids"          "kill"
[369] "killed"        "killing"       "kinda"         "kiss"
[373] "knife"         "knock"         "lady"          "lame"
[377] "laugh"         "lazy"          "leaves"        "leftist"
[381] "liar"          "liberal"       "liberals"      "lies"
[385] "life"          "likes"         "listen"        "live"
[389] "location"      "lol"           "love"          "lover"
[393] "loving"        "lucky"         "mad"           "mah"
[397] "man"           "massive"       "master"        "matches"
[401] "mate"          "meaningless"   "meat"          "medicine"
[405] "men"           "mental"        "messing"       "mexican"
[409] "mexicans"      "miserable"     "moist"         "monkey"
[413] "mother"        "mouth"         "murder"        "muslim"
[417] "muslims"       "nasty"         "nation"        "nationalist"
[421] "nazi"          "nazis"         "neck"          "neonazi"
[425] "nerd"          "nerdy"         "net"           "normal"
[429] "nose"          "notices"       "nuts"          "oil"
[433] "omg"           "onto"          "pain"          "painful"
[437] "painfully"     "pakistani"     "palestine"     "pants"
[441] "parents"       "passing"       "pathetic"      "pedophile"
[445] "perform"       "piece"         "pieces"        "pig"
[449] "pile"          "planet"        "politically"   "poor"
[453] "pray"          "prepared"      "priest"        "prison"
[457] "proceed"       "profile"       "protecting"    "proud"
[461] "province"      "proxy"         "pull"          "punch"
[465] "punk"          "puppet"        "quit"          "race"
[469] "racial"        "racism"        "racist"        "rape"
[473] "rapist"        "redneck"       "religion"      "republican"
[477] "retard"        "retarded"      "reverts"       "rot"
[481] "round"         "rubbish"       "ruin"          "ruined"
[485] "russia"        "russian"       "rvv"           "sad"
[489] "sand"          "scared"        "self"          "serve"
[493] "sexuality"     "shall"         "shame"         "shoot"
[497] "shot"          "shove"         "sht"           "shut"
[501] "sick"          "silence"       "sin"           "sincerely"
[505] "sir"           "sister"        "sit"           "sitting"
[509] "skin"          "slave"         "sleep"         "slit"
[513] "slow"          "slowly"        "smart"         "smell"
[517] "smelly"        "soldiers"      "son"           "sons"
[521] "sooo"          "speech"        "spell"         "spends"
[525] "spent"         "splatter"      "spreading"     "stalker"
[529] "steal"         "steve"         "stick"         "sticking"
[533] "stinks"        "stinky"        "street"        "stupid"
[537] "stupidity"     "sue"           "summer"        "sup"
[541] "super"         "superior"      "supertrll"     "supreme"
[545] "swear"         "sweet"         "taste"         "taught"
[549] "tea"           "teacher"       "telly"         "terrorist"
[553] "terrorists"    "thats"         "threat"        "threaten"
[557] "threatening"   "throat"        "throw"         "tight"
[561] "tiny"          "tom"           "tonight"       "tough"
[565] "track"         "trash"         "tree"          "troll"
[569] "trolls"        "tuesday"       "turkish"       "turks"
[573] "typical"       "ugly"          "undoing"       "useless"
[577] "utter"         "vile"          "vista"         "vomit"
[581] "wait"          "waiting"       "wales"         "walk"
[585] "wanna"         "wannabe"       "warn"          "wasting"
[589] "wat"           "watch"         "whats"         "white"
[593] "whoever"       "wife"          "will"          "window"
[597] "wing"          "wipe"          "wit"           "woman"
[601] "women"         "worship"       "worthless"     "wow"
[605] "yall"          "yeah"
```

B.2.15 Forming train- and test-mldr objects

```
> myptermeDTM<-DocumentTermMatrix(cleansynonymsnogander,
control=list(dictionary=mypterme))
> myptermetrainmat<-as.matrix(myptermeDTM)
> myptermetrainmatandlabels<-cbind(myptermetrainmat,labelstrain)
> myptermetraindtf<-as.data.frame(myptermetrainmatandlabels)
> myptermemldr<-mldr_from_dataframe(myptermetraindtf,
labelIndices=c(607,608,609,610,611,612),name="trainptermeMLDR")
> texttest<-as.vector(texttest)
> texttest<-VectorSource(texttest)
> texttest<-VCorpus(texttest)
> texttestclean<-cleandata(texttest)
> cleansynonymsleliktest<-tm_map(texttestclean,replaceSynonyms,
synonymslelik)
> cleansynonymsandertest<-tm_map(cleansynonymsleliktest,
replaceSynonyms,synonymsander)
> cleansynonymsnogandertest<-tm_map(cleansynonymsandertest,
replaceSynonyms,synonymsnogander)
> myptermetestDTM<-DocumentTermMatrix(cleansynonymsnogandertest,
control=list(dictionary=mypterme))
> myptermetestmat<-as.matrix(myptermetestDTM)
> myptermetestmatandlabels<-cbind(myptermetestmat,labelstest)
> myptermetestdtf<-as.data.frame(myptermetestmatandlabels)
> myptermetestmldr<-mldr_from_dataframe(myptermetestdtf,
labelIndices=c(607,608,609,610,611,612),name="testptermeMLDR")
```