Resampling Algorithms for Multi-Label Classification

Ulrich Kotze

Thesis presented in fulfilment of the requirements for the degree of Master of Commerce (Statistics) in the Faculty of Economic and Management Science at Stellenbosch University.

April 2022

Supervisor: Dr. T Sandrock

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
U. Kotze	April 2022

Acknowledgments

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. I would like to thank Dr. T Sandrock for her supervision and support of my research.

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.

Abstract

Multi-label classification is a member of the supervised learning family and represents a scenario where we wish to classify an observation into many of many classes. Therefore, in the classification paradigm an observation can belong to more than one class simultaneously.

Imbalanced data is a common problem in the multi-label paradigm of learning. This project investigated resampling algorithms as a pre-processing mechanism to address the manifestation of imbalance in multi-label data to improve multi-label classification performance.

Imbalance can manifest itself through a sparse data matrix at small global densities. Imbalance can also manifest itself through a disparity in local label density at larger global densities. The effect of resampling algorithms on multi-label performance is studied for both of these forms of imbalance. We specifically study the effect of these resampling algorithms on multi-label performance at changing levels of global density.

The thesis made use of simulated data, five common multi-label classification techniques and seven of the most popular resampling algorithms. Three example-based, label-based and ranking-based evaluation metrics were used to assess the effect of the resampling algorithms on multi-label classification performance.

Keywords:
Classification, multi-label, resampling algorithms, simulated data

Opsomming

Multi-etiket klassifikasie is 'n voorbeeld van onder toesig leer en verteenwoordig 'n scenario waarin ons 'n waarneming in baie van baie klasse wil klassifiseer. Daarom kan 'n waarneming in 'n klassifikasieparadigma gelyktydig aan meer as een klas behoort.

Ongebalanseerde data is 'n algemene probleem in die multi-etiket paradigma van leer. Hierdie tesis het hersteekproefnemingalgoritmes ondersoek as 'n voorverwerkingsmeganisme om die manifestasie van wanbalans in multi-etiket data aan te spreek om multi-etiket klassifikasieprestasie te verbeter.

Wanbalans kan manifesteer deur 'n yl data matriks by klein globale digthede of deur 'n verskil in plaaslike etiketdigtheid by groter globale digthede. Die effek van hersteekproefnemingalgoritmes op multi-etiket prestasie word bestudeer vir beide hierdie vorme van wanbalans. Ons bestudeer spesifiek die effek van hierdie hersteekproefnemingalgoritmes op multi-etiket prestasie by veranderende vlakke van globale digtheid.

Die studie het gebruik gemaak van gesimuleerde data, vyf algemene multi-etiket klassifikasietegnieke en sewe van die gewildste hersteekproefnemingalgoritmes. Drie voorbeeld-gebaseerde, etiket-gebaseerde en ranglys-gebaseerde evalueringsmetings is gebruik om die effek van die hersteekproefnemingalgoritmes op multi-etiket klassifikasieprestasie te bepaal.

Sleutelwoorde:
Klassifikasie, multi-etiket, hersteekproefnemingalgoritmes, gesimuleerde data

Table of contents

List of figures

List of tables

List of abbreviations:

MLC	Multi-label Classification
LC	Label Concurrence
MLR	Multi-label ranking
kNN	k-Nearest Neighbours
RF	Random Forest
SVM	Support Vector Machines
XGB	XGBoost
CART	Classification And Regression Tree
NB	Naive Bayes
MLkNN	Multi-label k-Nearest Neighbours
BR	Binary Relevance
BRplus	BR+
DBR	Dependant Binary Relevance
CC	Classifier chains
CLR	Calibrated Label Ranking
EBR	Ensemble of Binary Relevance
ECC	Ensemble of Classifier Chains
LP-ROS	Label-Powerset Random Oversampling
LP-RUS	Label-Powerset Random Undersampling
ML-ROS	Multi-Label Random Oversampling
ML-RUS	Multi-Label Random Undersampling
MLTL	Multi-Label Tomek-Links
REMEDIAL	Resampling multi-label datasets by decoupling highly imbalanced
MLSMOTE	Multi-Label Synthetic Minority Oversampling Technique
MLSOL	Synthetic Oversampling of Multi-Label Data based on Local Label Distribution
RHwRSMT	REMEDIAL-Hybridisation with synthetic instance generation
Pdf	Probability Density Function
LP	Label-Powerset
HD	Hamming Distance
AHD	Adjusted Hamming Distance

Chapter 1: Introduction

1.1 Multi-label Classification

Multi-label classification (MLC) is a lesser-known member of the supervised learning family. Although MLC has been extensively studied in recent years, it remains less common than many other traditional classification tasks such as binary and multi-class classification. In recent years there has been a considerable increase in the research performed on MLC. The research is being spearheaded by among others: Grigorios Tsoumakas[1], Francisco Charte[2], Antonio J. Rivera[3], Maria J. del Jesus[4], Andre C. P. L. F. de Carvalho[5], and Francisco Herrera[6]. The rise of complex data structures like text, video, and genomics that need to be annotated with more than one tag facilitated the need for advancement in MLC research.

The dominant areas of application are (Tsoumakas et al., 2009):

- Text annotation (Tsoumakas et al., 2011)
- Semantic annotation of images and video (Yang et al., 2007)
- Functional genomics (Barutcuoglu et al., 2006)
- Music categorisation into emotions (Trohidis et al., 2011)
- Directed marketing (Yi Zhang et al., 2007)

An application of MLC that has recently become popular is online text editing, like the website and application Grammarly. Grammarly has a tone detector feature, which tells a user the different tones of voice being used in a piece of text. The passage of writing could, for example, be labelled as “formal, assertive and confident”. Therefore, it is a form of text annotation. This website has an extensive list of tones and after text processing a label is assigned based on this list, where a piece of text can be assigned multiple tones from the list.

Another prominent application often encountered is the emotional annotation of tweets from Twitter (Yang et al., 2014). Each tweet is processed and annotated with the emotions expressed in the tweet. The emotional annotation can form a powerful combination with sentiment analysis. A tweet can, for example, be labelled as just “Angry” or “Angry, Frustrated and Impatient”, therefore assigning multiple labels to a tweet from a list of possible labels.

In the field of bioinformatics, we often wish to diagnose patients with certain illnesses. Suppose we are given a set of predictor variables such as symptoms or RNA-seq data. The patient could potentially have more than one illness—multi-label classification facilitates this

1 Grigorios Tsoumakas. Retrieved August 20, 2021, from https://scholar.google.com/citations?user=PlGKUhwAAAAJ&hl=en

2 Francisco Charte. Retrieved August 20, 2021, from https://scholar.google.com/citations?user=i8l_80EAAAAJ&hl=en

3 Antonio J. Rivera. Retrieved August 20, 2021, from https://scholar.google.com/citations?user=VW2FhqgAAAAJ&hl=en

4 Maria J Del Jesus. Retrieved August 20, 2021, from https://scholar.google.com/citations?user=1n84M0kAAAAJ&hl=en

5 Andre Carlos Ponce De Leon Ferreira De Carvalho. Retrieved August 20, 2021, from https://scholar.google.com/citations?user=B3L9jQMAAAAJ&hl=en

6 Francisco Herrera. . Retrieved August 20, 2021, from https://scholar.google.com/citations?user=HULIk-QAAAAJ&hl=en

by allowing observations to belong to more than one class. Therefore, given a set of symptoms, a patient can be related to several possible illnesses (Keren et al., 2011).

1.2 The classification paradigm

This thesis falls within the supervised learning paradigm. The classification paradigm of supervised learning contains binary classification, multi-class classification and multi-label classification. The complexity of solving the classification task increases as we move from binary to multi-class and from multi-class to multi-label. In this section we clearly distinguish the differences between the three paradigms of the supervised learning classification problem.

The classification paradigm of supervised learning is concerned with using a set of predictor variables $X_1, X_2, \dots, X_p$ to predict a set of qualitative response variables $Y_1, Y_2, \dots, Y_K$. The training data consists of $(\boldsymbol{x}_i, \boldsymbol{y}_i)\ \forall\ i \in 1,2, \dots, N$. A function $f(X_1, X_2, \dots, X_p)$ is used to map the input variables to the response variables $Y_1, Y_2, \dots, Y_K$. Therefore, the function f uses the predictor variables $X_1, X_2, \dots, X_p$ to predict the qualitative response variables $Y_1, Y_2, \dots, Y_K$.

The simplest example of the classification problem is binary classification, where the aim is to classify an observation into one of two classes, for example, "Class 1 or Class 2", "True or False", "spam or not spam". The two classes are mutually exclusive (cannot coincide). The binary classifier needs to classify the observation into one of these two classes. Therefore, the set of training observations $(\boldsymbol{x}_1, \boldsymbol{y}_1) \dots (\boldsymbol{x}_n, \boldsymbol{y}_n)$ is used to build a binary classifier, with the purpose of predicting the binary (two-level) qualitative response variable Y. Therefore, $K = 1$ and $Y \in [0,1]$. If an observation $y_i = 0$, it belongs to class 1 and if $y_i = 1$, the observation belongs to class 2. For supervised learning algorithms, this task is trivial at the best of times and very difficult on some occasions.

The natural way of increasing the complexity of binary classification is to add more classes to the problem, therefore moving from having just two classes to having many classes. The resultant effect is a multi-class classification problem since the aim is now to classify an observation into one of many classes. An observation can be one of many things, for example "Iris-virginica or Iris-versicolor or Iris-setosa", "blue or green or red", "one star or two stars or three stars or four stars or five stars". These classes are all mutually exclusive. The training observations $(\boldsymbol{x}_1, \boldsymbol{y}_1) \dots (\boldsymbol{x}_n, \boldsymbol{y}_n)$ are used to build a multi-class classifier, with the purpose of predicting the G-level qualitative response variable Y. Therefore, $K = 1$ and $Y \in \{1,2, \dots, \mathrm{G}\}$. The multi-class classifier needs to classify each of the observations into one of the G classes. $y_i = g$ if the i^{th} observation belongs to the g^{th} class, where $i \in \{1,2, \dots, \mathrm{n}\}$ and $g \in \{1,2, \dots, \mathrm{G}\}$. Multi-class classification has been one of the focus areas of supervised learning research for many years and forms an essential part of many supervised learning applications in practice.

Multi-label classification increases the complexity of the multi-class problem by allowing observations to belong to more than one class simultaneously, changing the task from "one of many" to "many of many". The classes are not mutually exclusive. It is reasonable to assume that this increases the complexity of the multi-class classification task further. The

number of possible class combinations for each observation will grow exponentially for the multi-label task, whereas it will grow linearly for the multi-class task as the number of possible classes grows. An observation can now be many of many things. Observations could belong to one class, two classes, or even all the classes simultaneously. The training observations $(\boldsymbol{x}_1, \boldsymbol{y}_1) \dots (\boldsymbol{x}_n, \boldsymbol{y}_n)$ are used to build a multi-label classifier, with the purpose of predicting the K binary response variables. Therefore $K > 1$ and $Y_k \in \{0,1\}$. Each response variable is binary and represents one of the labels in the multi-label problem. If the i^{th} observation has the k^{th} label present $y_i = 1$ at the k^{th} entry, where $i \in \{1,2,\dots,\mathrm{n}\}$ and $k \in \{1,2,\dots,\mathrm{K}\}$. Standard machine learning algorithms need to be adapted and changed or the multi-label data needs to be transformed to solve the multi-label problem. Most of the successful MLC techniques in the literature propose simple, pragmatic approaches to solving the multi-label problem.

Multi-label proposals contain many contrasting ways of approaching the multi-label classification task, which can broadly be categorised as either problem transformation or algorithm adaption approaches. Problem transformation techniques attempt to transform the multi-label data into a different problem that can be solved using existing binary and multi-class classification techniques. In comparison, algorithm adaption techniques attempt to alter existing machine learning techniques to solve the multi-label problem directly.

A typical multi-label dataset D consists of $i = 1,2,\dots,n$ observations with a matrix of predictor variables $X \in \mathbb{R}^{N \times P}$ and a matrix of response variables $Y \in \{0,1\}^{N \times K}$ that contain the labels. Therefore, each observation will have a vector of predictor variables $X = [X_1, X_2, \dots, X_p]$ and a binary vector of responses $Y = [Y_1, Y_2, \dots, Y_K]$ called the label-set of the i^{th} observation. Therefore, we want to use the information available in $X = [X_1, X_2, \dots, X_p]$ to predict $Y = [Y_1, Y_2, \dots, Y_K]$. The more information X contains about Y, the better our MLC techniques can predict the response Y. MLC falls under the supervised learning paradigm of machine learning techniques since our data has both predictor variables and a response variable that we are predicting.

The validation and training process of building multi-label models remain the same as in the supervised learning paradigm. The basic principle of training a model on training data and validating the model on “unseen” test data is still the foundation of the model fitting process. Some significant challenges in dealing with MLC come within the MLC models themselves: evaluation, visualisation, and describing the dataset’s attributes. The following sections explore how multi-label datasets are described and visualised.

1.3 Describing a multi-label dataset

The first step in any classification task, binary or multi-class, would be to look at the class distributions of the data. The class distributions show the proportion of the dataset that belongs to each class and is straightforward for binary and multi-class data since the classes are all mutually exclusive. In the multi-label case, the classes overlap and are not mutually exclusive since observations can belong to more than one class simultaneously. Therefore, we need to find additional ways of describing how the classes are distributed.

Figure 1 below shows an uncomplicated visualisation of the class distributions for the binary and multi-class classification tasks. It is easy to spot those classes containing more observations or fewer observations and make conclusions on the general distribution of the data among classes. Determining imbalance in the data can be done visually or by looking at the proportion of the data found in each class.

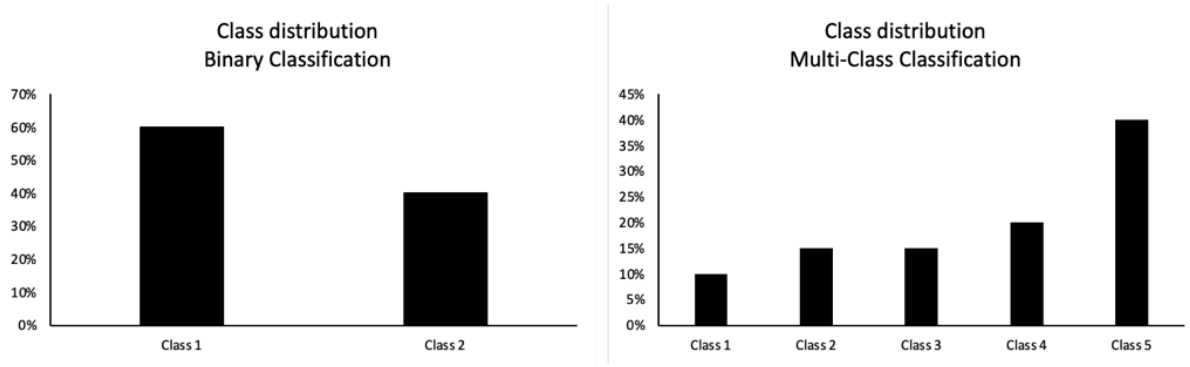


Figure 1:Class distributions

MLC addresses the problem of class imbalance by using different measures in addition to the traditional class proportions. In addition to looking at the proportion of labels that falls in each class, MLC also looks at the local label density, global density and cardinality of the dataset. These measures represent different ways of counting and describing the frequency with which labels are present in the dataset. The descriptive measures employed in multi-label data are:

1.3.1 Label Cardinality
1.3.2 Global Density
1.3.3 Local Label Density

1.3.1 Label cardinality

Label cardinality is the average number of labels that are present per observation. An overall label cardinality of 2.1 tells us that the average number of labels per observation is 2.1. Some observations will have more than two labels, and other observations will have less than two labels, but on average, an observation will have 2.1 labels.

$Label\ Cardinality = \frac{1}{n}\sum_{i=1}^{n}|Y_i|$, where n = Number of observations and $|Y_i|$ =Number of labels present for observation i.

The advantage of label cardinality is that it is intuitive since it is on the same scale as the labels. If the dataset has ten labels and the cardinality is 4, we know that there is, on average, four of the ten labels present per observation. However, the cardinality cannot be compared from one dataset to another since most datasets do not have the same number of labels. Label cardinality can only be compared from one dataset to another if the datasets have the same number of labels. (Tsoumakas et al., 2009)

1.3.2 Global Density

The overall or global label density, not to be confused with the local label density (1.3.3), describes how often labels are present relative to the total number of labels in the dataset. It can be interpreted as the relative percentage of labels that are present for the average observation. The global density is closely related to cardinality but is different since it gives the proportion of present labels rather than the number. If the overall label density is high, it shows that an observation will have more labels present on average, whereas a smaller global density shows that, on average, an observation will have fewer labels present. A label density of 0.1 shows that the average observation in the dataset will have 10% of its labels present.

$Global\ density = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i|}{K}$, where K = Number of labels.

The advantage of using the global label density is that it is a relative measure that can be compared from dataset to dataset. If one dataset has a higher label density than another dataset, we know that its labels are present more often. The same is not valid for a measure such as cardinality. (Tsoumakas et al., 2009b)

1.3.3 Local Label Density

The label density can also be calculated for every label individually (local label density). Local label density is the closest measure to the class distributions seen for binary and multi-class classification. It is calculated by summing the total number of observations present for the specific label and dividing it by the total number of observations in the dataset.

$Local\ density\ for\ label\ Y_k = \frac{\sum_{i=1}^{n} I_{Y_{i,k}}}{n}$, where $I_{Y_{i,k}} = 1$, if label k is present for the i^{th} observation and 0 otherwise.

The local label density becomes useful when we want to visualise the multi-label dataset and compare the presence of different labels in the dataset. If the local label density for one label is higher than for another, we know that label is present more often. The global label density is also equal to the average of the K local label densities.

These descriptive measures do not necessarily describe the class distribution as explicitly as it is for the binary and multi-class cases, but it does give us a fair idea of how the data is

distributed across labels. Multi-label data also poses unique challenges, which requires different solutions to the standard way of thinking.

1.4 Imbalance in multi-label data

1.4.1 Reasons for imbalance in multi-label data

A common problem that arises in most MLC tasks is imbalanced data. There are two mechanisms through which imbalance manifests itself in multi-label data:

Although each observation can belong to many classes, they seldom do and usually tend to have very few labels or a small global density and cardinality. This gives rise to an imbalance in our data since the low cardinality and global density in general causes a sparse response data matrix Y. A sparse response data matrix will have many zeros among the responses and very few ones, indicating the lack of present labels. Supervised learning algorithms thrive on an abundance of data. Therefore, a sparse data matrix provides a challenging environment within which the supervised learning algorithms must function. (Charte et al., 2019a)

Multi-label data often has some labels with very high local label densities and other labels with very small local label densities. This disparity in the local label density of the labels causes a problem for machine learning algorithms since they tend to be naive in these scenarios. Datasets that have some labels with very small local label densities that are accompanied by other labels with larger local label densities are common in multi-label data, for example the cal500 (Turnbull et al., 2008) dataset seen in Section 1.7. These datasets consist of a few dominant labels that are often present and have a large local label density, accompanied by other labels with very small local label densities that are seldom present.

1.4.2 Measuring imbalance

Given how common imbalanced datasets are in MLC, it is an essential part of any MLC problem to quantify the imbalance present in the dataset. The following metrics can be calculated to describe the imbalance in a multi-label dataset:

1.4.2.1 Imbalance Ratio Per Label (IRLbl)

1.4.2.2 Mean Imbalance Ratio (MeanIR)

1.4.2.3 Score of ConcUrrence among iMBalanced LabEls (SCUMBLE)

1.4.2.1 IRLbl

The Imbalance Ratio Per Label (IRLbl) (Charte et al, 2015b) is a way of measuring the local imbalance of individual labels. The higher the IRLbl of a label is, the larger the imbalance in that label is. The smaller the IRLbl of a label is, the smaller the imbalance in that label is. The IRLbl allows us to know which labels are majority classes and which labels are minority classes. The majority labels will have smaller IRLbl values, whereas minority labels will have larger IRLbl values. Imbalance refers to a small proportion of 1's, and not to a small proportion of 0's as well.

The IRLbl can be calculated with the following formula:

$IRLbl(y_k) = \frac{max_{y'_k \in Y}\left(\sum_{i=1}^{n}\left[\left[y'_k \in Y_i\right]\right]\right)}{\sum_{i=1}^{n}\left[\left[y_k \in Y_i\right]\right]}$, where y_k is the k^{th} label being analysed and y'_k are all of the labels, excluding the k^{th} label.

The double square brackets [[ ]] represent the indicator function that return one if the expression in the brackets is true and 0 otherwise and y_k is the label that is being analysed. The most commonly occurring label is used as the reference. (Charte et al., 2019)

1.4.2.2 MeanIR

The Mean Imbalance Ratio (MeanIR) is a way of measuring the global imbalance of a multi-label dataset, where the MeanIR is the average of the IRLbl for all the labels. The smaller MeanIR is, the less imbalance there is in a dataset, and the larger the MeanIR is, the more imbalance there is in the dataset. High MeanIR can occur for different reasons. There could be high MeanIR because IRLbl is large for many labels or if there is extreme imbalance in only some of the labels. The MeanIR can be calculated with the following formula:

$MeanIR = \frac{\sum_{k=1}^{K} IRLbl(y_k)}{K}$, where K is the number of labels in the dataset.

Therefore, IRLbl calculates the local imbalance for specific labels, and MeanIR calculates the global imbalance for the entire dataset. The IRLbl and MeanIR are extremely important to MLC and form an integral part of resampling algorithms for MLC. Those labels that have $IRLbl > MeanIR$ are seen as minority labels, and the labels that have $IRLbl \leq MeanIR$ are seen as majority labels. Resampling algorithms will be discussed in detail in Chapter 3. (Charte et al., 2019)

1.4.2.3 SCUMBLE

A characteristic that emerges in multi-label learning is that some labels are correlated with each other. The correlation between labels can be expected since it is reasonable to assume that one characteristic (class) could often be paired with another characteristic (class). These label interdependencies could add useful information to an MLC problem. Some MLC techniques try to take advantage of these correlations to better predict the labels. MLC

models such as classifier chains try to exploit the label correlations by ordering the labels in the chain in a specific order to exploit the correlations among labels.

The correlations can also create problems for MLC and resampling algorithms. It is troublesome for resampling algorithms when majority labels are correlated to minority labels. This situation is referred to as concurrence. Concurrence is another barrier in the way of resampling algorithms for MLC. Concurrence can be measured by calculating a Score of ConcUrrence among iMBalanced LabEls (SCUMBLE) (Charte et al., 2014) score as follows:

If the label k is present for observation i, then $IRLbl_{ik} = IRLbl(k)$ otherwise $IRLbl_{ik} = 0$. $\overline{IRLbl_i} = \frac{\sum_{k=1}^{K} IRLbl_{ik}}{K}$, the average of $IRLbl_{ik}$ for all of the labels appearing in observation i. The $IRLbl_i$ scores are used to calculate the concurrence per instance called $SCUMBLE_{ins}$.

$$SCUMBLE_{ins} = 1 - \frac{1}{\overline{IRLbl_i}} \left(\prod_{k=1}^{K} IRLbl_{ik} \right)$$

The $SCUMBLE_{ins}$ values are averaged for all the observations in the dataset to obtain the final SCUMBLE score.

$$SCUMBLE(D) = \frac{1}{n} \sum_{i=1}^{n} SCUMBLE_{ins}(i)$$

It is essential to mention that datasets can have observations that dominate the $SCUMBLE_{ins}$ that can be seen as outliers. These outliers can artificially inflate the SCUMBLE score. To measure differences in concurrence among observations, we can calculate SCUMBLE.CV as follows:

$$SCUMBLE_{\sigma} = \sqrt{\sum_{i=1}^{n} \frac{(SCUMBLE_{ins}(i) - SCUMBLE)^2}{n-1}}$$

$$SCUMBLE.CV = \frac{SCUMBLE_{\sigma}}{SCUMBLE}$$

SCUMBLE.CV allows us to determine if the large SCUMBLE is due to a few observations that dominate the SCUMBLE score or whether it is a general problem in the dataset. Although SCUMBLE does not directly reflect imbalance, it plays a significant role in dealing with imbalance. The resampling algorithms discussed in Chapter 3 are specifically adversely affected by this phenomenon. The SCUMBLE measure is an indication of how hard it is to deal with a certain multi-label dataset for resampling algorithms. Therefore, when SCUMBLE is large, resampling would struggle to improve MLC performance and when SCUMBLE is relatively smaller, resampling algorithms would find it easier to improve MLC performance (Charte et al., 2019).

1.5 Visualising multi-label data

Visualising a multi-label dataset is almost as important as describing it using metrics like density and MeanIR. Visualising a multi-label dataset aims to achieve a high-level understanding of what the dataset looks like and what characteristics the dataset might have. The descriptive statistics (1.4) and the visualisations of the multi-label data (1.5) should be used in conjunction with each other. Neither can form a complete picture of the multi-label dataset by itself and should therefore be used together. We use the "flags" (Goncalves et al., 2013) dataset for demonstration purposes as it is one of the benchmark datasets and is commonly found in MLC research. This dataset contains details of some countries and their flags, and the goal is to predict the colours contained in the flags, where one flag could contain multiple colours.

The local label densities can be visualised by creating a bar plot. Figure 2 shows an example of this plot. The plot of local label density can be used in conjunction with the global density, to compare the local label densities to the global density. In figure 2, the heights of the bars represent the local label density of the individual labels and the red horizontal line represents the global density, which is the average of the local label densities. Therefore, we can identify those labels that have a below average local label density and those labels that have an above average local label density.

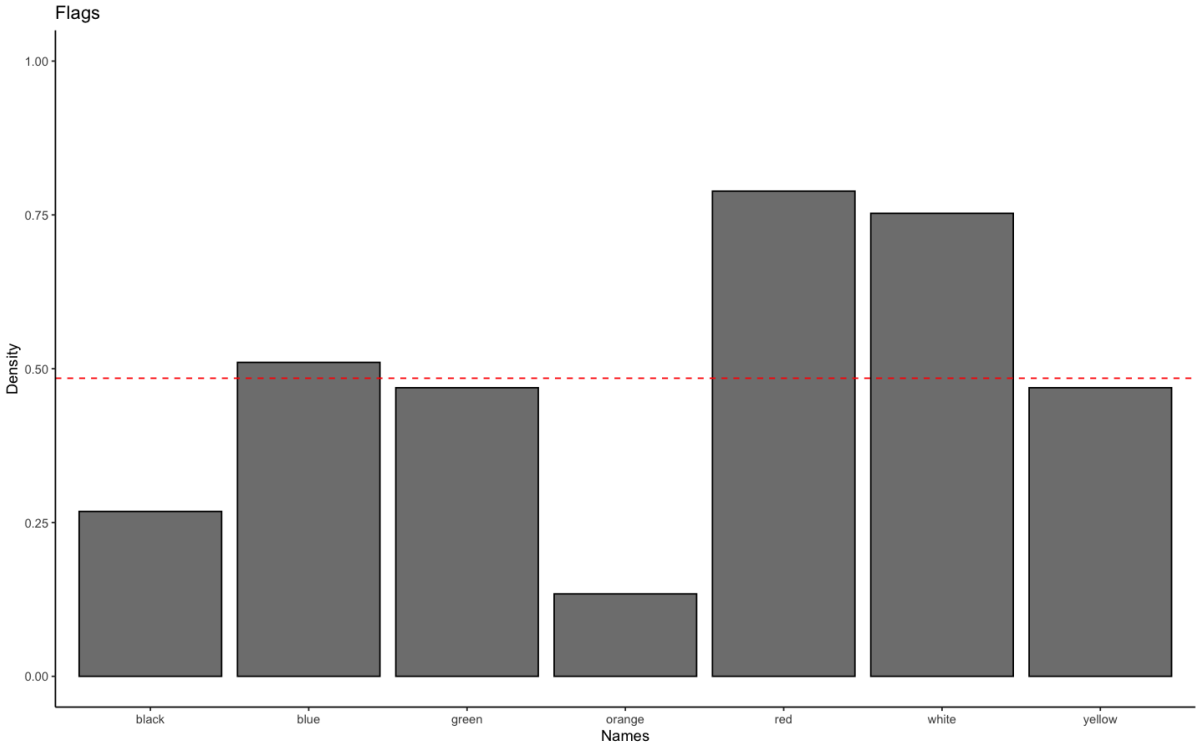


Figure 2: Local label density (flags)

A histogram is used to plot the number of labels per observation. The histogram will show us the distribution of labels per observation. It can be used in conjunction with cardinality. The

cardinality is the average number of labels per observation. Therefore, the cardinality will be the central tendency of the distribution of labels per observation.

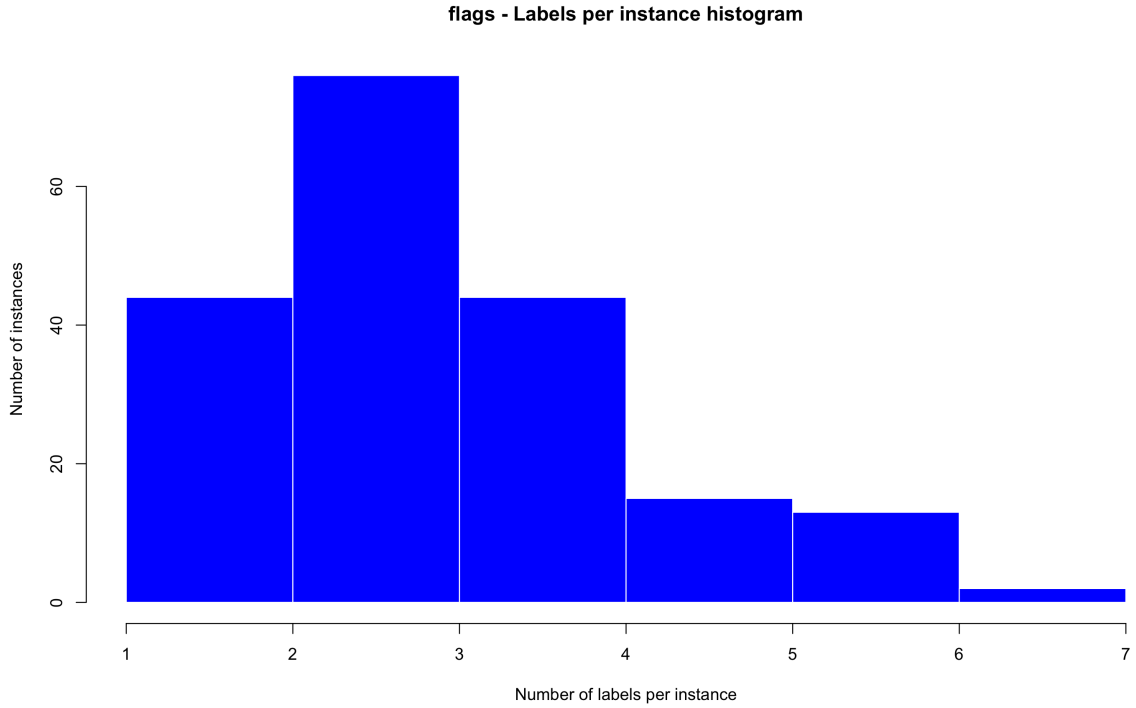


Figure 3: Labels per instance histogram flags

Extending from the discussion of imbalance before, the local and global imbalance levels given by IRLbl and MeanIR can be visualised together in an imbalance plot. This visualisation is not found in the literature but is very useful. The visualisation plots a bar plot of the label specific IRLbl's, with a red horizontal line representing the MeanIR. The imbalance plot allows us to see which labels have IRLbl larger than the MeanIR (minority labels) and which labels have IRLbl smaller or equal to MeanIR (majority labels). Therefore, this plot gives a global perspective for the prevalence of imbalance in the dataset.

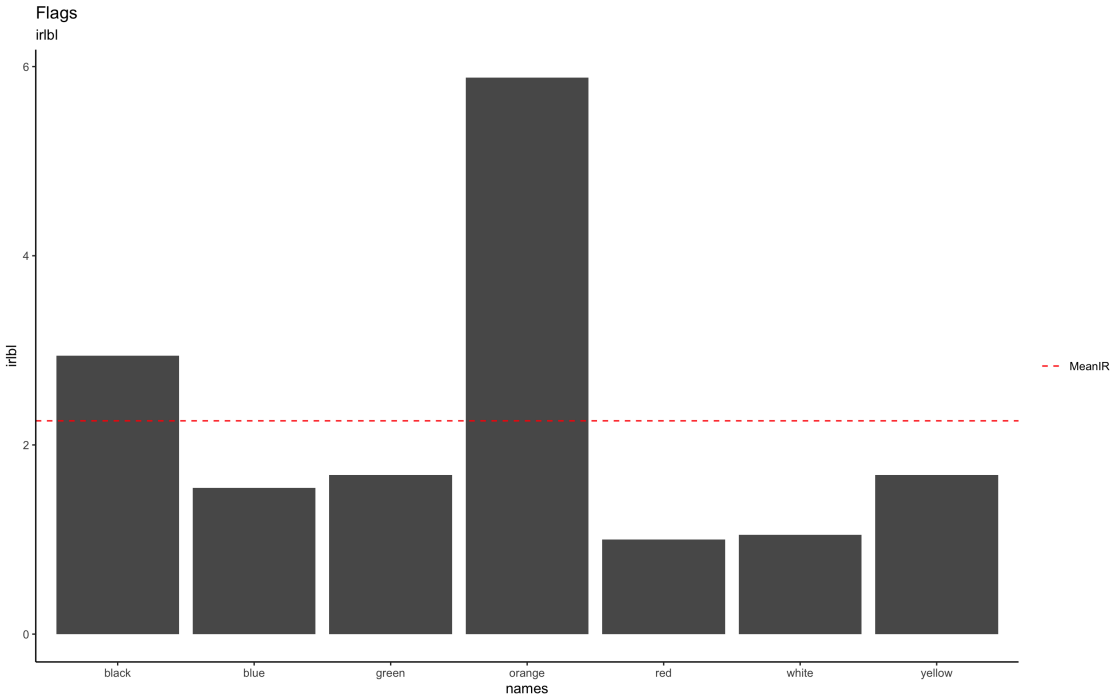


Figure 4: Imbalance plot (flags)

Figure 4 represents something close to the inverse of Figure 2 seen before. These plots show the close relationship that exists between the local label densities and the IRLbl of the individual labels. This relationship is explored further in Chapter 3.

The prevalence of concurrence can be visualised by using the label concurrence (LC) plot observed in Figure 5 below (Charte et al., 2019), which allows us to visualise the local label density of each label and the labels that appear with it most often. We observe the coloured filled arcs on the outside of Figure 5, representing the local densities of the individual labels. The larger the arc is, the larger the local label density of that label is. Filled lines also join these coloured arcs. These lines connect labels that often occur together. Therefore, those labels that tend to co-occur will be joined by filled lines. Observing labels with small local densities connected to labels with large local densities may indicate SCUMBLE in a dataset. The LC plots become ineffective when we have too many labels present since the graph becomes unreadable and takes a substantial amount of time to plot.

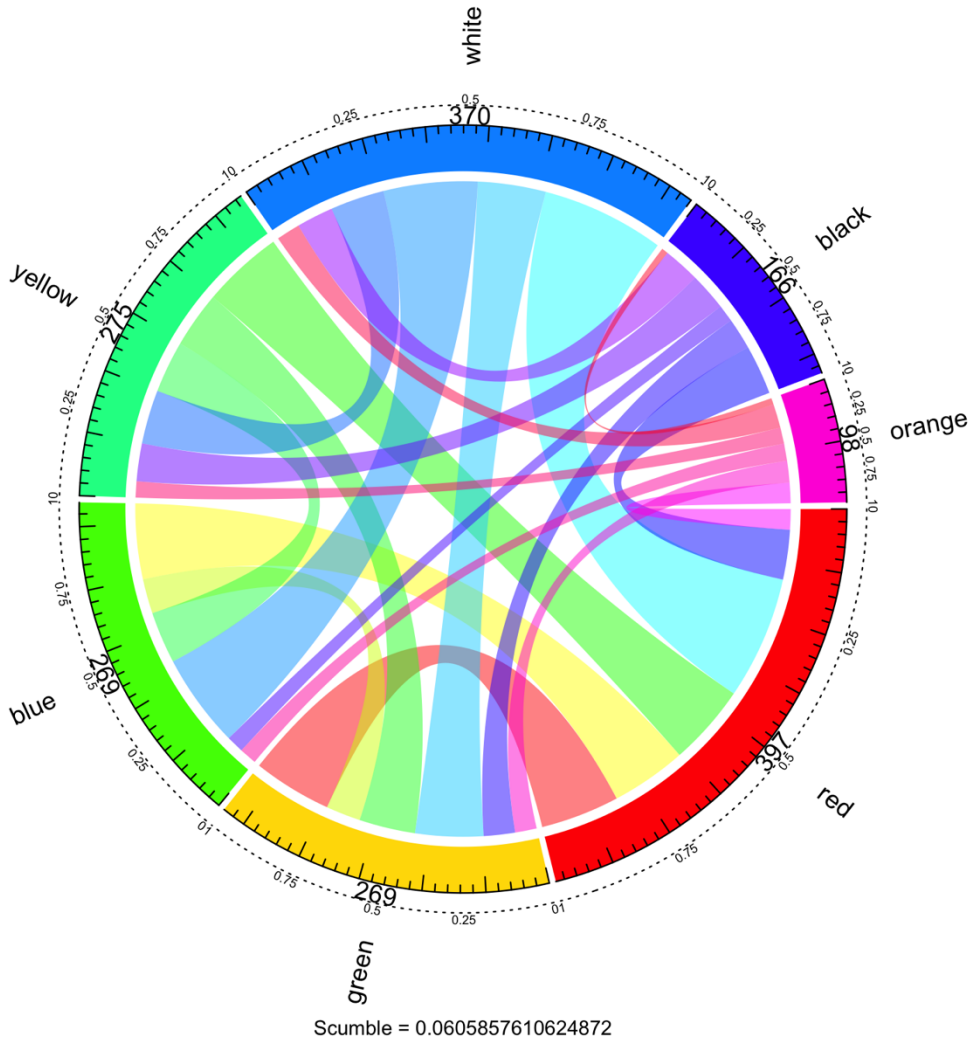


Figure 5: Label concurrence plot (flags)

1.6 Resampling algorithms

Resampling algorithms will be briefly introduced in this section since it is vital to understand the role that these resampling algorithms play in MLC before discussing the purpose of this thesis. Chapter 3 will provide an in-depth discussion of the resampling algorithms used in this thesis. The primary purpose of resampling algorithms is to improve MLC performance by reducing the prevalence of imbalance in a multi-label dataset as a preprocessing tool for MLC. Therefore, resampling algorithms are designed to either reduce MeanIR or SCUMBLE. By reducing the imbalance or SCUMBLE, it is reasonable to assume that the performance of MLC will improve.

Traditionally in binary classification there are two broad heuristic approaches for reducing imbalance in a dataset, oversampling the minority class (1.6.1) and undersampling the majority class (1.6.2)

1.6.1 Oversampling the minority class

Oversampling selects random observations with replacement from the minority class and adds them to the dataset, therefore artificially inflating the minority class so that it will have more observations and balance the class distributions. The advantage of oversampling the minority class is that we do not lose any information because no observations are being deleted from the dataset. However, it is debatable whether adding copies of the same observation in the dataset will help classification performance. Nevertheless, it will balance the class distributions.

1.6.2 Undersampling the majority class

Undersampling deletes random observations in the majority class from the dataset, therefore artificially deflating the majority class by removing observations to balance the class distributions. The advantage of this approach is that the observations in the minority class are not changed. Therefore, we are not artificially adding any information to the dataset that could skew the analysis. The drawback of this approach is the necessity of a large dataset since observations are being removed from the dataset. Removing observations from a dataset will lead to a loss of information.

For large datasets, undersampling the majority class will be preferred. However, when the dataset is not large enough, oversampling of the minority class needs to be performed. In general, oversampling is preferred.

Undersampling and oversampling are more complicated in the multi-label paradigm than in the single-label paradigm. In the multi-label paradigm, observations can belong to more than one class simultaneously. Therefore, it becomes difficult to distinguish minority observations from majority observations since it is possible for an observation to belong to both minority and majority labels at the same time. This leads to problems for normal heuristic oversampling and undersampling methods since oversampling the minority classes might implicitly lead to oversampling of the majority classes and vice versa for undersampling. This

specifically becomes a problem when SCUMBLE is high in the dataset. A high SCUMBLE score is an indication that minority and majority labels are correlated with each other. Therefore, in these scenarios normal undersampling and oversampling might be redundant and lead to a degradation in data quality since the inflation of the minority labels could also lead to an inflation of the majority labels. A reduction in the majority labels could also lead to a reduction in the minority labels, which is an undesirable outcome for the resampling algorithms.

On top of the basic heuristic methods, many other approaches use more complex algorithms to create new artificial minority observations or delete the most ambiguous majority examples. These algorithms will be covered in depth for the multi-label scenario in Chapter 3.

1.7 Benchmark datasets

Research performed in the paradigm of multi-label classification often makes use of benchmark datasets. These datasets are open source and freely available and serve as a point of comparison for MLC techniques. Although this thesis discusses the drawbacks of the benchmark datasets and instead makes use of simulated data, these datasets are very useful and form an integral part of the vast majority of multi-label research. Charte & Charte, (2015) provides an overview of the benchmark datasets in the R package "mldr.datasets". The R package contains a list of 49 benchmark datasets. A summary of a few of the commonly used benchmark datasets is given in the table below:

Table 1: Benchmark datasets

Name	Observations	Variables	Labels	Domain	Density	Cardinality
Yeast (Elisseeff & Weston, 2002)	2417	133	14	Biology	0.303	4.237
Cal500 (Turnbull et al., 2008)	502	242	174	Music	0.149	26.044
Emotions (Wieczorkowska et al., 2006)	593	72	6	Music	0.311	1.868
LangLog (Read et al., 2012)	1460	1004	75	Text	0.016	1.180
Scene (Boutell et al., 2004)	2407	294	6	Image	0.179	1.074

The list above is not complete and only contains a few of the benchmark datasets. The Yeast dataset is a multi-label dataset from the Biology domain, it contains protein profiles and their categories. Cal500 and Emotions are multi-label datasets from the music domain, it contains features extracted from music tracks and the emotions they produce. LangLog is a multi-label dataset from the text domain containing language forum discussion. Scene is a multi-label dataset from the image domain and contains images with different natural scenes.

In Section 1.5, we observe that the global density of a multi-label dataset refers to the relative percentage of labels that are present per observation. A list of the publicly available multi-label datasets is provided by Moyano (2021). In this list, we observe that most of the datasets have a global density smaller than 0.1.

In Table 1, we also observe that the cardinality is close to one for three of the five datasets. A cardinality close to one indicates that the observations on average only have one label present. This indicates that these datasets are almost multi-class datasets and not multi-label since the labels are present so seldomly. This is a common theme in multi-label data and one of the main reasons we make use of simulated data, instead of the benchmark datasets.

1.8 Research goals

In this thesis, there are two research goals we would like to address:

1.8.1 Are resampling algorithms effective at improving MLC performance at changing levels of global density?

The efficacy of resampling algorithms as a preprocessing mechanism for multi-label classification at disparate levels of global density will be investigated. Therefore, the research aims to use the most popular resampling algorithms to preprocess the data and then fit various MLC models on this data. We can then investigate what effect the resampling algorithms had on the performance of the MLC models and how the performance of resampled datasets compares to the performance of data that has not been resampled. These experiments will be performed with multi-label data at disparate levels of global density. Therefore, the resampling algorithms will be tested on imbalanced data, due to a sparse data matrix and on imbalanced data due to a disparity in local label density at larger levels of global density. The results for this question are discussed in Chapter 7.

1.8.2 Is there one form of resampling preferred to all others?

Secondly, we would like to determine if one or more resampling algorithms should be preferred to all others. We know that most of the selected resampling algorithms are effective under certain conditions. We want to determine if any of these resampling algorithms emerge as a dominant one across the diverse set of conditions created in the experiments. We also wish to make recommendations relating to the use of the resampling algorithms, relating to MLC performance, computational efficiency and degree of resampling. These results are discussed in Chapter 8.

It would be possible to perform our research on the list of publicly available datasets since there are datasets available at all levels of global density. However, a significant hurdle to our research would be that all these datasets come from different domains. It is reasonable to assume that the domain that the dataset comes from will significantly impact the performance of an MLC model (domain effect). Therefore, we will use artificially simulated data to create datasets with disparate levels of global density whilst still having the same

strength of dependence between the predictor variables and the response variables, mitigating the domain effect on the MLC performance. The use of simulated data has an added benefit in that it mitigates confirmation bias. We do not have the option of only selecting the datasets that support our narrative but are dependent on our simulations' data. A new way of simulating multi-label data is proposed by Sandrock & Steel (2017). This method of simulating multi-label data provides control over the local label densities, which forms a fundamental part of this thesis.

The purpose of this thesis is not the comparison of MLC techniques, feature selection nor evaluation measures. Although MLC techniques and evaluation metrics need to be chosen, the goal is not to find the best MLC technique, compare MLC techniques with each other or claim that one evaluation metric is better than another. The purpose of this thesis is to investigate the efficacy of resampling algorithms as a preproccesing tool for MLC at various levels of global density, using a wide variety of evaluation metrics.

1.9 Moving forward

The structure of the thesis is as follows. In Chapter 2 MLC techniques are discussed, where we provide an overview of the different approaches to solving the MLC problem and an in-depth look at the MLC models used in this thesis. Chapter 3 will discuss the various resampling algorithms. Chapter 4 discusses the mechanism used to simulate artificial multi-label data. Chapter 5 introduces the evaluation metrics used to assess MLC performance. Chapter 6 contains the experimental design and provides an overview of the experiments performed in the analysis. Chapters 7 and 8 address the two research goals posed in Section 1.8. Lastly Chapter 9 is a conclusion of the entire thesis and discusses all the main findings in the thesis and makes recommendations for future research.

Chapter 2: Multi-label Learning

2.1 Multi-label learning tasks

Multi-label learning falls under the paradigm of supervised learning. The data consists of a matrix of predictor variables $X_{N,P}$ and a matrix of response variables $Y_{N,K}$. Each observation can belong to more than one class simultaneously, where these classes are referred to as labels. It is the job of the multi-label classifier, which can be referred to as some function $f(X)$, to predict the labels belonging to each observation. The goal of multi-label learning is to choose the function $f(X)$ such that the multi-label performance is optimised in terms of the chosen performance metric. An MLC model can form a label-set prediction in the following two ways:

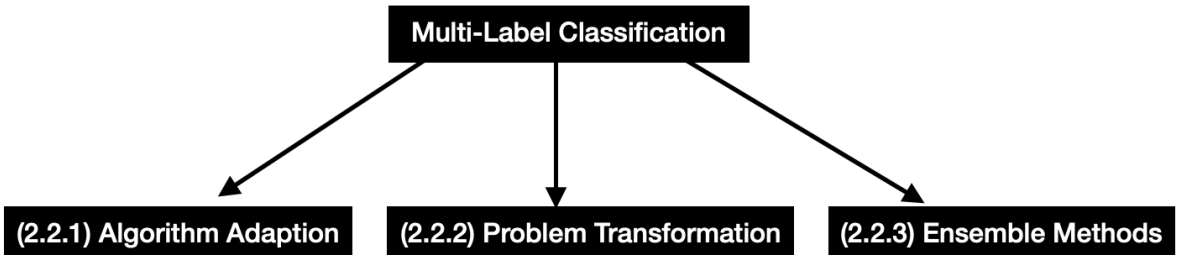
Multi-label classification refers to a predictive model $f(X)$ that attempts to directly predict the label-set belonging to each observation. A prediction of $y_i = 1$ for the k^{th} label indicates the presence of label k and $y_i = 0$ indicates the absence of the k^{th} label, where $k \in \{1,2, \dots, K\}$. Therefore, MLC directly makes a classification of the labels.

In contrast to MLC, multi-label ranking (MLR) does not directly predict the labels. Instead, MLR generates a "preference" for each of the labels, where this preference is given in terms of a label ranking $0 \leq r_i \leq 1$. For each observation, we observe a vector of label rankings $r_i = [r_{i1}, r_{i2}, \dots, r_{iK}] \; \forall \; i \in \{1,2, \dots, N\} \; and \; k \in \{1,2, \dots, K\}$. The higher the label ranking is, the more likely it is that this label will be present. The lower the label ranking is, the less likely it is that this label is present. Therefore, the labels can be ranked according to their relevance for the given observation. When MLR is employed, a classification can be found using some threshold t to choose the present labels. Therefore, when the label ranking is above some threshold t, the label is present, and when the ranking is below the threshold t, the label is not present. Formally $y_{ik} = 0 \; if \; r_{ik} \leq t$ and $y_{ik} = 1 \; if \; r_{ik} > t$, where $i = 1,2,, \dots, N \; and \; k = 1,2, \dots, K$.

This chapter first provides an overview of the general approaches to solving the multi-label learning problem, with references to the most popular approaches. After this, we will give an in-depth discussion of the models we have chosen to use in this thesis. We are not necessarily looking to optimise MLC performance but would like to choose a diverse group of models that cover most of the approaches for multi-label learning, are computationally efficient and can function under a diverse set of conditions. The analysis aims to investigate the effect that the resampling algorithms have on the performance of MLC in general and not to compare the performance of one MLC model with another.

2.2 Multi-label classification methods

Multi-label classification methods can broadly be split into three main categories, described below:



2.2.1 Algorithm Adaption

Algorithm adaption methods are multi-label learning techniques that modify, expand, and alter existing machine learning algorithms to handle the multi-label problem directly. There is an abundance of machine learning algorithms for binary and multi-class problems that produce outstanding results. A natural way to solve the multi-label problem would be to adapt these algorithms to also solve the multi-label problem directly. Some of the algorithms that have been adapted are:

- Boosting (Schapire et al., 2000)
- kNN (Zhou et al., 2006)
- C4.5 (Clare & King, 2001)
- Neural Networks (Min-Ling Zhang & Zhi-Hua Zhou, 2006)
- Support Vector Machines (Elisseeff & Weston, 2002)

All the algorithms mentioned above were not specifically developed for the multi-label paradigm. These algorithms had to be changed and adapted to be able to solve the multi-label classification problem. The struggle with algorithm adaption techniques is what makes multi-label learning so enjoyable. In multi-label learning we face a different landscape where these well-known algorithms do not necessarily maintain the same success they achieved in the multi-class classification paradigm (Tsoumakas, 2019a).

2.2.2 Problem Transformation

Problem transformation methods introduce pragmatic approaches to solving the multi-label learning problem. The problem transformation algorithms transform the multi-label problem into one or many binary and multi-class problems. These binary and multi-class problems can then be solved using standard machine learning algorithms. Therefore, the problem transformation approach uses the vast number of approaches available for binary and multi-class classification to solve the multi-label problem. The following are some popular approaches to transforming the multi-label problem:

- Binary Relevance (BR) (Tsoumakas & Katakis, 2007)
- Classifier Chains (CC) (Read et al., 2011)
- Label Powerset (LP) (Tsoumakas & Katakis, 2007)
- Pruned Problem Transformation (PPT) (Read et al., 2009)
- Ranking by Pairwise Comparison (RPC) (Hüllermeier et al., 2008)
- Calibrated Label Ranking (CLR) (Brinker et al., 2006)

Problem transformation algorithms range from simple pragmatic solutions like Binary Relevance to advanced techniques such as Calibrated Label Ranking. However, all these techniques have the same goal and that is to break up the multi-label classification problem into a simpler problem that can be solved by techniques that are known to be successful in the binary and multi-class classification domains. (Tsoumakas, 2009a)

An example of a problem transformation technique is the label-powerset (LP) technique. LP attempts to change the multi-label learning problem into a multi-class problem by considering every distinct label-set as a class in a multi-class problem. Therefore, the multi-class classifier will classify an observation to the most likely label-set. The labels in the label-set then form the prediction for that observation. LP leverages the suggestions that are already available on multi-class classification. LP, therefore, uses standard machine learning techniques, mitigating the need for an elaborate multi-label model. A possible disadvantage of this approach is when there is low cardinality present for certain labels, the binary classification that needs to be performed will also be an imbalanced classification problem. This could become a problem if there is low cardinality for many labels.

2.2.3 Ensemble methods

In the general case, an ensemble method combines multiple models to generate better predictive performance. The ensemble should generate better performance than any of its members could by itself. An ensemble's basic idea is to train a group of models with low bias and high variance (highly flexible models). When we average this group of models, we are left with a model that still has a low bias but reduces variance caused by the averaging. For an ensemble to be effective, it is paramount that the member models of the ensemble are not too similar since this will mitigate the effect of the averaging. A popular way this is achieved is by fitting the member models (which are flexible models with low bias) of the ensemble on bootstrap datasets (bagging) since these datasets should look like the original dataset but slightly different. The predictions from each of these highly flexible models should then look different for the bootstrap datasets, averaging them leading to a better model.

The idea of ensemble models in multi-label learning is built on a similar principle. The models used within the ensemble are either algorithm adaption models or problem transformation models. The basic proposition remains the same, namely training multiple diverse multi-label models and then averaging these models to come to a final model that has better predictive performance than any of its members by itself. In the multi-label paradigm, there are different ways in which we can achieve diversity in models—bootstrapping the dataset still being the most obvious solution to the problem. Another solution could be to consider only a subset of the labels for each model in the ensemble. An obvious drawback of these models is the computational complexity of having to fit many multi-label models. Fitting a multi-label model by itself is already expensive computationally. Fitting many of them could lead to long runtimes. Nevertheless, ensembles have the potential to vastly improve classification performance. Some of the most popular ensemble-based approaches to solving the multi-label problem are:

- Ensemble of Binary Relevance (Read et al., 2009)
- Ensemble of Classifier Chains (Read et al., 2009)
- RAkEL (Tsoumakas et al., 2011)
- Ensemble of Pruned Sets (Read et al., 2008)
- Recursive Dependent Binary Relevance (Rauber et al., 2014)

2.3 Base classifiers

We observe that both the problem transformation and the ensemble methods that make use of problem transformation models, make use of binary and multi-class classifiers within the algorithms. The binary or multi-class classifier that the algorithm uses is called the base classifier. The "utiml" package (Rivolli & Carvalho, 2019) is a framework that allows the user to apply multi-label classification in the R programming language. A framework similar to MULAN (Tsoumakas et al., 2011) in the Waikato Environment for Knowledge Analysis (Weka) allows the user to easily apply multi-label classification, sampling methods, transformation strategies, threshold functions, pre-processing techniques and evaluation metrics. The following base classifiers are available in the R package "utiml":

- Random Forest (RF)
- Support Vector Machines (SVM)
- XGBoost (XGB)
- C5.0
- K-Nearest Neighbours (KNN)
- Classification And Regression Trees (CART)
- Naive Bayes (NB)

For this thesis, a base classifier that is computationally efficient and produces consistent results across multiple models is required since the goal of this thesis is not to find the best multi-label classification model. One base classifier will therefore be used, to avoid adding unnecessary complexity to the experimental conditions. Madjarov et al. (2012) compare several MLC techniques in a comparative study and find that SVM's are the most appropriate base classifier for BR, CC, CLR, EBR and ECC. Therefore, SVM's will be used as the base classifier for all the multi-label models in this thesis, except for MlkNN which is an algorithm adaptation technique and therefore does not require a base classifier.

2.4 Support Vector Machines

Support Vector Machines (SVMs) fall under supervised learning algorithms and were initially intended for binary classification but can also be used for multi-class classification and regression. A brief, high-level discussion of the Support Vector Machine algorithm follows from the formulation of James et al. (2013). The formulation for the SVM algorithm is very extensive, but this discussion will be kept short and to the point since the SVM is a well-known algorithm and does not necessarily fall within the scope of our research. For a more comprehensive discussion, refer to Hastie et al. (2013). We will start with a hyperplane and then move to classification using a hyperplane. The next step thereafter is to look at the maximal margin classifier before arriving at the SVM algorithm.

2.4.1 Hyperplane

In a p dimensional space, a hyperplane is a $(p-1)$ dimensional flat affine surface. Therefore, in two dimensions a hyperplane will be a straight line. In three dimensions, a hyperplane will be a flat surface that cuts the three dimensions. In more than three dimensions, it becomes

difficult to visualise a hyperplane. The black line in Figure 6 represents a hyperplane in two dimensions.

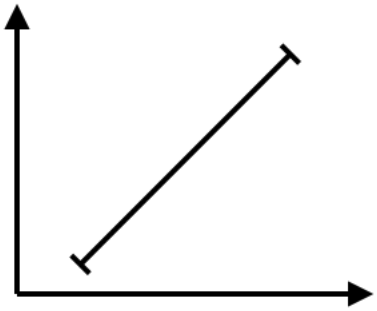


Figure 6: Hyperplane

2.4.2 Classification using a separating hyperplane

Suppose we consider a data matrix $X_{n,p}$ with n observations in a p dimensional space, where observations can belong to one of two classes $y_i \in \{-1,1\} \forall\, i = 1,2,\dots,N$. In this case the classes are assumed to be linearly separable (note that this is not always the case). A separating hyperplane is a hyperplane that intersects these two classes. In the figure 7, we observe a separating hyperplane in two dimensions.

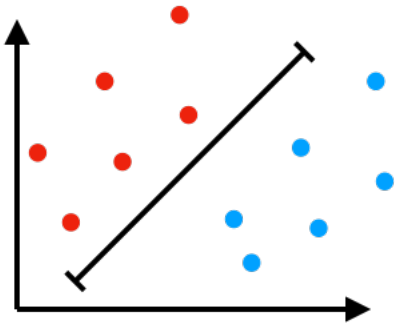


Figure 7: Separating hyperplane

If we were to think about the hyperplane in terms of a classification rule, we could classify new observations that fall to the one side of the hyperplane as $y_i = -1$ and observations that fall on the other side of the hyperplane as $y_i = 1$, where $i = 1,2,\dots,n$. Any line or surface in $(p-1)$ dimensions that perfectly separates these two classes can be used as a separating hyperplane. Therefore, technically speaking, when the classes are perfectly separable (no overlap between classes), an infinite number of separating hyperplanes could separate the classes. Changing the slope of the separating hyperplane by a tiny fraction would lead to a new separating hyperplane. The further an observation is away from the hyperplane, the more certain we can be about the classification since it is further away from the class boundary and the closer an observation is to the hyperplane the less certainty we have about the classification.

2.4.3 Maximal Margin Classifier

Addressing the problem of having an infinite number of separating hyperplanes, the maximal margin classifier tries to find the separating hyperplane that is farthest from the training observations. If we consider the margin to be the distance between the hyperplane and the closest training observation, we will find the hyperplane that maximises the margin, therefore trying to find the hyperplane that optimally separates the two classes. The observations that lie on the margin, those observations "supporting the margin", are called support vectors, the origin of the name support vector machines. All the observations lying on the margin can be seen as support vectors to the separating hyperplane. The support vectors are the observations that are the hardest to classify since they lie on the margin.

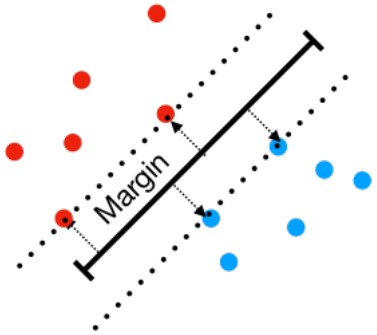


Figure 8: Maximal margin classifier

The maximal margin classifier tries to find the hyperplane that optimally separates the two classes by finding the hyperplane that maximises the margin. The support vectors are the observations that would change the position of the separating hyperplane since moving these observations would have a direct impact on the margin. The major stumbling block to this technique, as mentioned previously, is that classes are rarely perfectly separable, like we see in this example. Therefore, the technique was extended to allow observations to be on the wrong side of the hyperplane and this generalisation is referred to as the support vector classifier.

2.4.4 The Support Vector Classifier

The support vector classifier extends the maximal margin classifier by allowing the training observations to be on the wrong side of the margin or hyperplane. Therefore, we are willing to accept a hyperplane that does not perfectly separate the two classes. Allowing the observations to be on the wrong side of the margin relaxes the need for the hyperplane to be a perfect separating hyperplane. Each observation receives a slack variable $\epsilon_i \forall i \in \{1,2,\dots,n\}$. These slack variables ϵ_i measure how far each observation is on the wrong side of the margin. A tuning parameter C is introduced through a new constraint $\sum_{i=1}^{N} \epsilon_i \leq C$. Therefore, the user can specify C and, by doing so, chooses how much violation of the margin will be tolerated. As C increases we allow more observations to violate the margin, leading to a wider margin and vice versa. In practice C is often chosen through cross-validation since C is the parameter that controls the bias-variance trade-off for the SVM. A small C will lead to a model that has low bias, but high variance and a large C will lead to a model with high bias and low variance.

The goal of the classifier remains to maximise the margin M, but subject to the new constraint introduced through the C parameter.

The algorithm above is for a linear classifier and can easily be extended to the non-linear case by considering quadratic, cubic or higher-order polynomial functions of the predictors instead of a linear function, and in this case the classifier is referred to as the Support Vector Machine.

2.4.5 Support Vector Machine

The support vector machine extends the support vector classifier by enlarging the feature space to introduce a non-linear decision boundary. The feature space is enlarged by using quadratic, cubic and polynomial functions of the predictors, which allows us to address the possibility of non-linear decision boundaries. However, enlarging the feature space in this manner can quickly lead to a large number of features and an optimisation problem that is hard to solve efficiently. SVMs addresses this efficiency problem by using kernels in a specific way to enlarge the feature space.

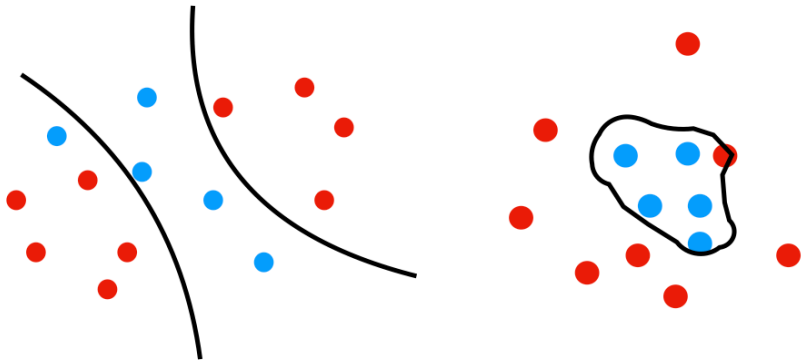
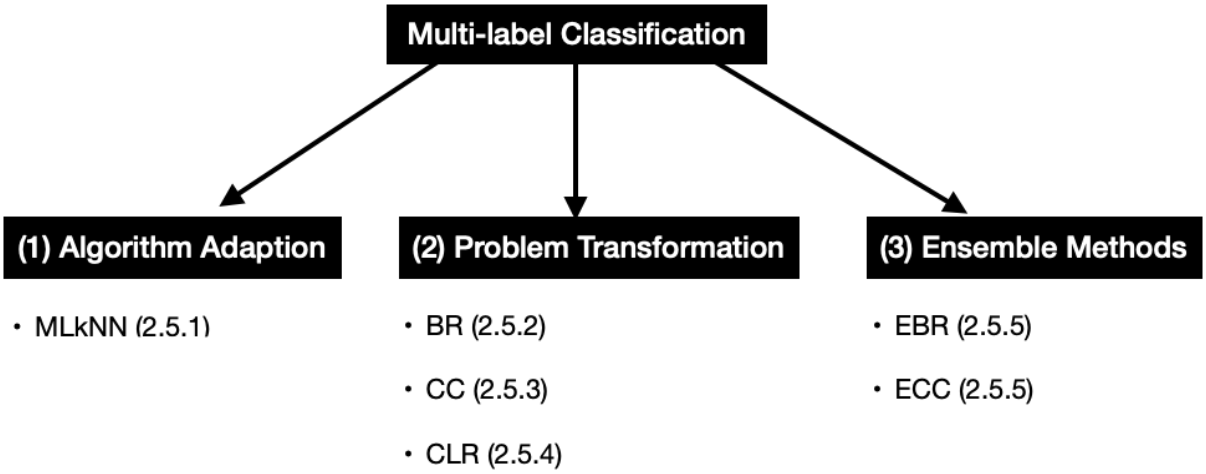


Figure 9: Decision boundaries (SVM)

An advantage of SVMs is that it makes use of cross-validation to select the tuning parameter that controls the bias-variance trade-off. This provides stability to the algorithm and ensures that it does not overfit the data. SVMs can also be used as both binary and multi-class classifiers, which makes it ideal to be used as a base classifier. Figure 9 shows two examples of what non-linear decision boundaries could look like when kernels are used in SVMs. The practical application of SVMs is not too cumbersome. In R, the package “e1071” (Meyer et al., 2021) is used to fit SVMs. Throughout this thesis, SVMs are used as the base classifiers in all the multi-label models fitted.

2.5 Multi-label classification models

We wish not to limit ourselves to one group of models or approaches. The purpose of this thesis is to investigate MLC performance at changing global density in general. For this reason, we need to include a broad scope of models in our analysis and not just focus on choosing the models that would optimise performance in a specific context. The models used in this thesis are limited by the software packages used. For consistency in results, only the packages "mldr" (Charte & Charte, 2015) and "utiml" (Rivolli & Carvalho, 2019) were used in R to fit MLC models. Refer to Rivolli & Carvalho, (2019) for a complete list of MLC models that can be used in the "utiml" package. Only one algorithm adaption technique is chosen since MLkNN is the only algorithm adaption technique available in the "utiml" package. The tree diagram below describes the models that were chosen to be used in this analysis; these are also the models discussed in this section:



2.5.1 Multi-label k-nearest neighbours

k-Nearest Neighbours (kNN) is not an unfamiliar model in data science circles and is often the starting point for non-parametric classification techniques. kNN is built on the premise that similar observations will lie close to each other in distance (usually Euclidean distance). If observations from class $y = 1$ mostly surround a test observation, then the test observation is also most likely from the class $y = 1$. This is a reasonable assumption since observations from the same class tend to be similar and should therefore lie close to each other in a Euclidean space, and observations from different classes tend to be dissimilar and should therefore lie further from each other in a Euclidean space. kNN is a lazy learning algorithm. Lazy learning implies that there is no explicit training and test step in the model, in the sense that an explicit decision boundary is not derived. When a new test observation is queried, we find the k closest neighbours to the test observation. The test observation is then classified to the majority class among the k-nearest neighbours.

Multi-label k-Nearest Neighbours (MLkNN) proposed by Zhang & Zhou, (2007) is built on the same premise as kNN, MLkNN is therefore an algorithm adaption of kNN. The assumption is made that observations that have similar label-sets should lie close to each other in the Euclidean space, and observations that have dissimilar label-sets should lie further from each

other in a Euclidean space. Multi-label classification poses a significant challenge to the kNN algorithm since each observation can now belong to more than one class.

MLkNN (Herrera, 2016) starts by training a model on two pieces of information, the prior probability and the conditional probability of each label individually. These probabilities are calculated for each label separately as a binary class. The prior probability for each label is the number of times the label appears, divided by the total number of observations in the dataset. Therefore, the prior probability is related to the local label density of the individual labels. The prior probability is multiplied by a smoothing factor, to avoid multiplication by zero. The conditional probability of each label is the proportion of the observations with this label whose k-nearest neighbours also have the same label. The correlations between labels are completely disregarded by the MLkNN algorithm since each label is considered separately as a binary class. Therefore, the calculation of the prior and conditional probabilities can be seen as a training step for the MLkNN model, before new test observations can be classified.

To classify a new test observation, the MLkNN algorithm follows the steps:

1. Calculate the k-nearest neighbours to the test observation, using the Euclidean distance.
2. The presence of each label in the neighbourhood of the k-nearest neighbours is used to calculate the Maximum A Posteriori (MAP) probabilities from the prior and conditional probabilities calculated at the start for each label.
3. The label-set for the new test observation is generated from the MAP probabilities, where the probabilities serve as a confidence level for each label. This also allows the labels to be ranked.

Although kNN is a lazy learning algorithm, MLkNN is not a lazy learning algorithm in the same sense. MLkNN does implicitly contain a training step for the model, where the calculation of the prior and conditional probabilities at the start of the algorithm can be seen as the training step. Once the training step has taken place, new observations can be labelled by using the MAP probabilities calculated in step 2 above.

2.5.2 Binary Relevance

Binary Relevance (BR) (Tsoumakas & Katakis, 2007) is the first problem transformation method considered. BR implements a pragmatic approach to solving the multi-label problem by leveraging the vast amount of binary classifiers at our disposal. BR can work in conjunction with any binary classifier and offers a simple and easy approach to solving the multi-label problem.

One classifier is trained for each label. Therefore, we fit K binary classifiers, $f_k \; \forall \; k \in \{1,2,\dots,K\}$ where $Y_k \in \{0,1\}$. BR starts by transforming the multi-label dataset D into K smaller datasets, where there is one dataset for each of the K labels—illustrated by the diagram below. Each observation can then be a $y = 0$ or a $y = 1$, where a 0 indicates that the label is not present and a 1 indicates that the label is present. Therefore, we now have K binary datasets. A separate binary classifier is now trained on each of these K binary datasets.

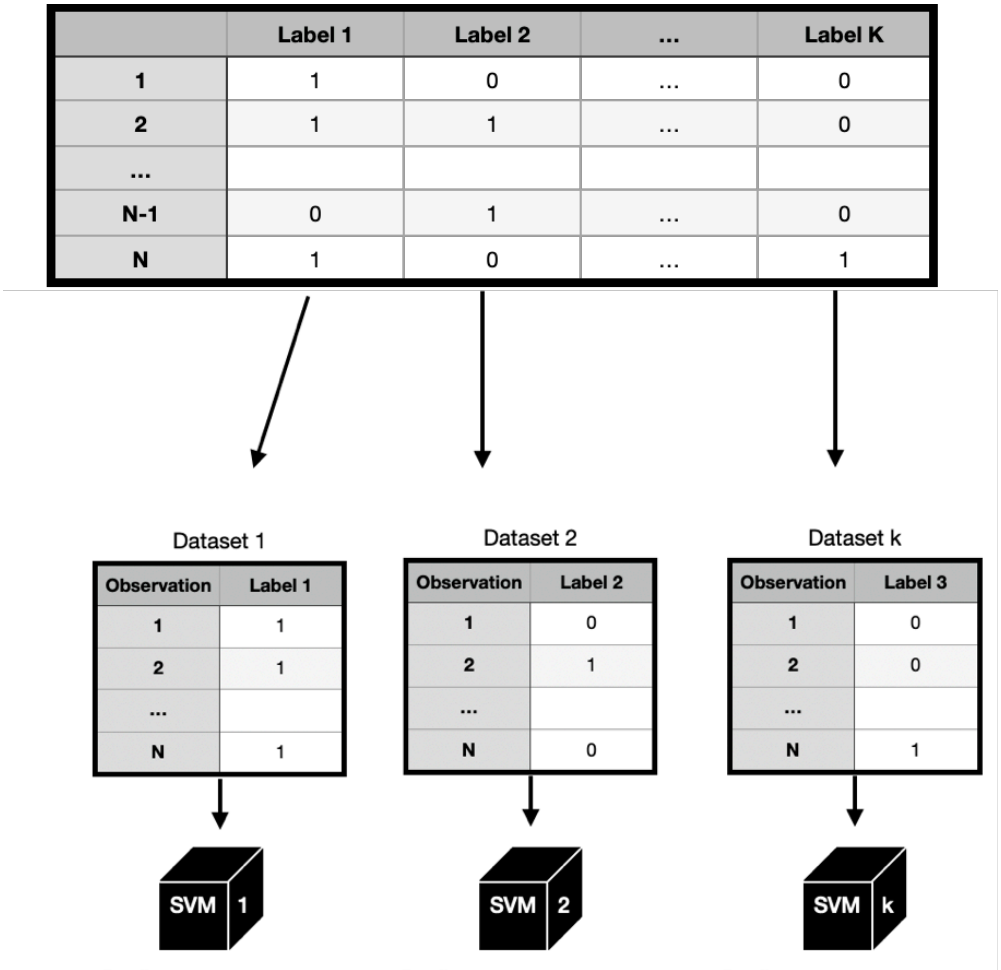


Figure 10: Binary Relevance

A new test observation is given a label-set by allowing each of these K classifiers f_k to predict $y = 1$ or $y = 0$ for the k^{th} label. The label-set of the test observation is then the union of the predictions made by the K classifiers. Therefore, the test observation receives the labels for which the individual classifiers returned positive results.

BR does not take label correlations into account, which could have a detrimental effect on performance in datasets that have label dependencies. Sometimes the weaknesses of a model can also be its strengths. Not taking label correlations into account makes BR resistant to overfitting label combinations. It also allows BR to add and remove labels from the analysis easily. Another weakness of BR is the imbalanced classes that often occur when the data is aggregated. Multi-label data often has low cardinality. Therefore, we will likely encounter an abundance of zeros and very few positive examples when considering a label by itself. The imbalance poses a problem to the base classifiers fit on each label individually, as these classifiers may be biased towards the majority class. The computational cost could also become a problem in certain paradigms where we have enormous datasets.

There are other variations of Binary Relevance that exist. These variations try to address some of the shortcomings found for BR. Two of the common variations come across in the literature is BR+ (BRplus) proposed by Alvares-Cherman et al., (2012) and dependant binary relevance (DBR) proposed by Montañes et al. (2014). BRplus is an extension of BR that allows the BR model to take label correlations into account. Dependant Binary Relevance combines properties from both Classifier Chains and the Stacking approach proposed by Godbole & Sarawagi, (2004) to better model the dependencies between labels.

2.5.3 Classifier Chains

Classifier Chains (CC) (Tsoumakas & Katakis, 2007) is another problem transformation technique, which can be seen as a natural extension of the BR technique, discussed previously. The dataset D is transformed similarly, where a binary dataset is created for each label. A binary classifier is fit on each of these K binary datasets. Up to this point, CC is equivalent to BR.

The property that separates CC from BR is the inclusion of previous label classifications into the training dataset. The binary classifiers are not all fit at the same time. The ordering of the chain is random or can be chosen by the user. The classifiers are then fit to the binary datasets one by one, sequentially. The chain structure allows the prediction made by the previous classifiers to be inserted as predictor variables into the next classifier, allowing CC to take label correlations into account.

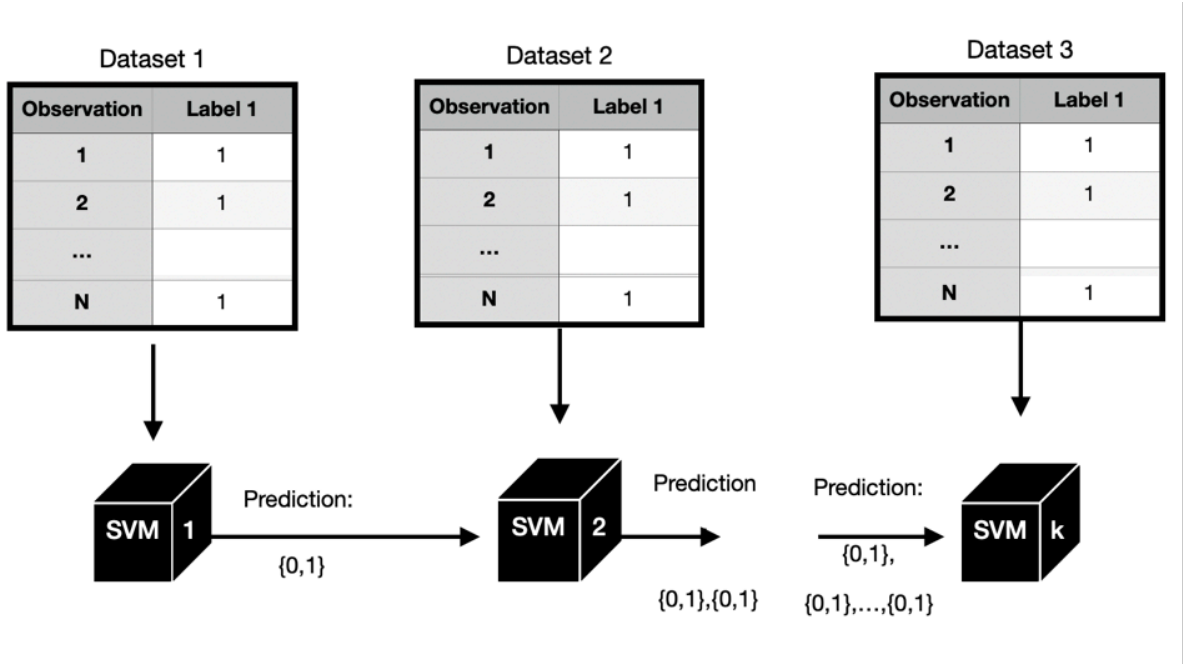


Figure 11: Classifier Chains

In the diagram above, we observe the idea of a classifier chain, where a binary classifier such as an SVM is fit on the binary dataset emerging from each label. The predictions made by the previous classifiers in the chain are carried forward as predictor variables to the following classifiers in the chain. It is reasonable to assume that the presence or absence of specific labels will influence the presence of other labels, referred to as label correlations. This is the idea behind carrying the predictions made by the previous classifiers forward. Therefore, decisions made by classifiers early in the chain could affect decisions made by classifiers later in the chain. It is implicitly taking label correlations into account.

The classifier chain does not have to be ordered like it is in the diagram. The SVMs can be fit in any order. This ordering does however affect the model's performance. There are several heuristics for the ordering of the chain. Often a random ordering is used. If there is specific domain knowledge of correlations between certain labels, the chain could be ordered accordingly. For example, if label 10 turns out to be a good predictor of label 1 and label 3. It

would be desirable to have label 10 in the chain before label 1 and label 3, allowing us to take the correlation of these labels into account when predicting label 1 and label 3.

The criticisms of CC are similar to that of BR since the two approaches are so similar, except for the ability of CC to take label correlations into account. However, CC comes with an added risk of error propagation. The classes in a CC are ordered linearly. Therefore, an error made early in the chain will impact the rest of the chain and cause more errors down the chain. Nevertheless, CC performs well in comparative studies. Read et al., (2021) found classifier chains to have state of the art performance across many datasets and evaluation metrics.

2.5.4 Calibrated Label Ranking

Multi-label ranking is concerned with learning a model $f(X)$ that produces both a ranking of the complete label-set and a partition of this label-set into relevant and irrelevant labels. The set of labels $Y_k \; \forall \; k \in \{1,2,\dots,K\}$ is to be split into a set of relevant labels P_x and a set of irrelevant labels N_x.

To convert the label rankings into a multi-label prediction, Calibrated Label Ranking (CLR) (Brinker et al., 2006) needs to determine a point at which the label rankings are split into sets of relevant and irrelevant labels.

The fundamental idea behind CLR is to introduce a virtual label Y_0 as a split point between the relevant and irrelevant labels. The virtual label can be seen as an artificial calibration label. The set of relevant labels are all preferred to the virtual label. The virtual label is preferred to all labels in the set of irrelevant labels. An implicit assumption of the model is that the partition made by the model is, in general, not independent of the ranking of the model.

A calibrated label ranking $Y_{i1} \succ Y_{i2} \succ \cdots \succ Y_{ij} \succ Y_0 \succ Y_{ij+1} \succ Y_{iK}$ induces a ranking among the labels and a dichotomous partition $P_x = Y_{i1}, \dots, Y_{ij}$ and $N_x = Y_{ij+1}, \dots, Y_{iK}$. All of the labels that are preferred to the virtual label Y_0 are part of the set of relevant labels P_x and those labels that are not preferred to the virtual label Y_0 form the set of irrelevant labels N_x. Therefore, those labels that form part of the set of relevant labels P_x will have $y_i = 1$ and the labels that form part of the set of irrelevant labels N_x will have $y_i = 0$.

The CLR model can be learned by solving the conventional ranking problem in the augmented CLR space (majority voting). This can now be seen as a ranking problem with $K + 1$ alternatives—the K labels, with the addition of the virtual label. (Brinker et al., 2006)

2.5.5 Multi-label ensemble schemes

Multi-label ensemble schemes (Read et al., 2011) follow the bagging framework found in the supervised learning literature. Bagging fits B models on B bootstrap samples of the data; these B models are then averaged to produce a final model. Combining these B models will have better performance than any of the individual models within the framework. By fitting models with small bias and large variance on the individual bootstrap samples, averaging

these B models will lead to a model with small bias, but a reduction in variance caused by the averaging of the models. In the supervised framework, the fundamental principle behind Bagging is to reduce the model's variance without increasing the model's bias.

2.5.5.1 Bootstrap sampling

Many variations of the bootstrap exist. Our focus falls on the standard non-parametric bootstrap. Suppose we consider a dataset D, with n observations. One bootstrap sample is a sample of size n, sampled with replacement from D, usually in the context of Bagging we would be interested in taking B bootstrap samples. Therefore, a bootstrap sample will be the same size as the original dataset but will consist of a similar but slightly different set of observations, where duplicate observations can occur.

2.5.5.2 Bagging

Bagging (Bootstrap aggregation) fits a model on each of the B bootstrap samples. The predictions made by the B models are then averaged to obtain the final predictions by the model. The idea of Bagging can be seen in the diagram below. In the context of Bagging, we can make B as large as we want, without any risk of overfitting the model. The limitation on B is computational since we need to sample and fit a model on each of these datasets.

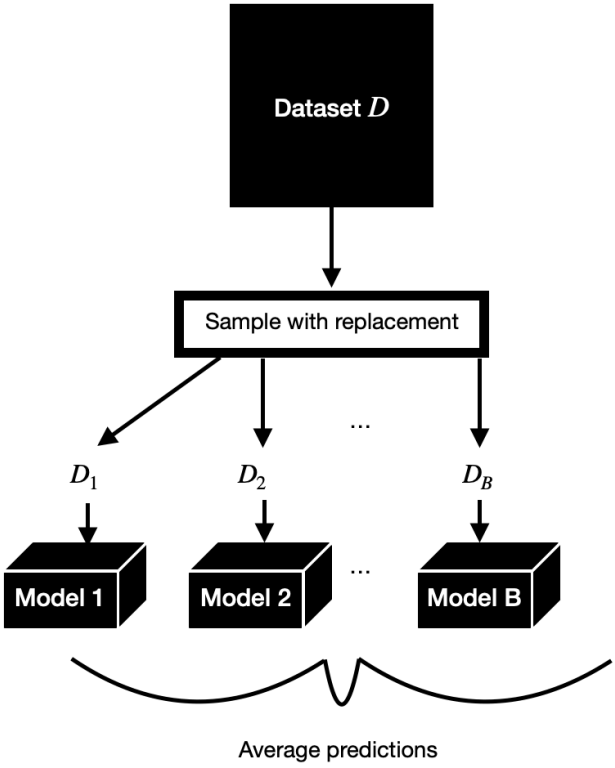


Figure 12: Bagging

2.5.5.3 Multi-label context

In the multi-label paradigm (Read et al., 2011), the same procedure as explained above is followed; the individual models used are BR and CC. Therefore, BR and CC models will be fit on bootstrap samples of the data, and their predictions averaged according to some threshold to obtain the final label-set prediction.

More formally, fit a BR/CC model on each bootstrap sample $D_i \forall i \in \{1,2,\dots,B\}$, where D_i are bootstrap samples from the original dataset. Produce a vector of confidence outputs, $\widehat{w} = [\widehat{w_1}, \widehat{w_2}, \dots, \widehat{w_K}]$ where $\widehat{w_k}$ represents the confidence for the k^{th} label $\forall\, k \in \{1,2,\dots K\}$. In other words, $\widehat{w_k}$ is the average number of times that label k was predicted as present by the B models.

$$\widehat{w_k} = \frac{1}{B}\sum_{b=1}^{B} y_{k,b}$$

To create a bipartition of relevant and irrelevant labels, we can apply a threshold function to $\widehat{w}$. For a new observation:

$$y_k = \begin{matrix} 1 \;\; if \;\; \hat{w}_k \geq t \\ 0 \;\; if \;\; \hat{w}_k < t \end{matrix}$$

where t could be set to any value. For the best performance Read et al., (2011) propose using:

$$t = argmin_t \left\| LCARD(D) - \left(\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{L} 1_{\widehat{w_j} \geq t} \right) \right\|$$

The threshold above tries to achieve cardinality in the new predictions as close as possible to the cardinality in the training data, which is a reasonable approach since we would expect that new test observations should have similar cardinality to the training dataset on average. The approach outlined above should achieve better results than a threshold set at some arbitrary value like 0.5. The approach outlined above will be used with BR and CC. The resulting models are an ensemble of binary relevance (EBR) and an ensemble of classifier chains (ECC).

Chapter 3: Resampling Algorithms

3.1 The Resampling task

In Chapter 1, a short introduction to imbalance was given. Chapter 3 is dedicated to discussing resampling algorithms that combat the prevalence of imbalance and attempt to improve MLC performance in multi-label datasets. In the multi-label paradigm, two mechanisms can introduce imbalance:

3.1.1 Low cardinality and global density in general

The first mechanism for imbalance is introduced through a sparse data matrix, which can be seen through a lack of label presence in general throughout the dataset. This form of imbalance is associated with large MeanIR values and small global densities. A sparse data matrix will have an abundance of zeros and very few ones, a common occurrence in the multi-label paradigm, where some datasets have such low global densities that it can almost be seen as a multi-class problem. A dataset that has a very small global density and hence a large MeanIR can be seen as a dataset with imbalance due to a sparse data matrix. This form of imbalance will be referred to as the "special case" and is explained further in Section 4.4.1.

3.1.2 The polarity of local density between labels

The second mechanism through which imbalance can manifest itself is a polarity between the local label density of individual labels. Therefore, some labels are much more common than other labels, which is more in line with the traditional idea of imbalance seen in other forms of classification. Therefore, the data will consist of minority labels with very small local label density (high IRLbl) and majority labels with large local label density (small IRLbl). The imbalance is due to the polarity in local label density of these minority and majority labels. Machine learning algorithms may be biased towards those labels with a higher local label density since they will favour the labels that occur more often. Section 4.4.2 explores the simulation of data where imbalance manifests itself as a polarity between the local label density of individual labels. This form of imbalance is important to this thesis since it describes the situation where imbalance can manifest itself at larger global densities.

The traditional way of measuring imbalance is through the IRLbl (locally per label) and MeanIR (globally):

$$IRLbl(y_k) = \frac{max_{y'_k \in L}\left(\sum_{i=1}^{n} [[y'_k \in Y_i]] \right)}{\sum_{i=1}^{n} [[y_k \in Y_i]]}$$

$$MeanIR = \frac{\sum_{k=1}^{K} IRLbl(y_k)}{K}$$

If we look a little deeper into these measures, we find that IRLbl almost ends up being a proxy for the local label density, and the MeanIR ends up being a proxy for the global label density. Generally, the labels with small local label densities tend to have large IRLbl scores and those labels that have relatively larger local label densities tend to have smaller IRLbl scores. Datasets that have smaller global densities tend to have larger MeanIR scores and datasets that have larger global densities tend to have smaller MeanIR scores. The effect this has is that when we have datasets with higher global densities, the MeanIR is lower. A small MeanIR indicates that these datasets are not imbalanced; however, these datasets could still be imbalanced due to a polarity in local label density. Even at higher global density levels, there could still be a polarity in local label density, a manifestation of imbalance in the dataset. However, the MeanIR measures imbalance through the IRLbl; this might provide false security that there is no imbalance in the dataset.

Figure 13 illustrates this idea. In the first figure, we observe the local label densities. There is some imbalance present in this dataset, caused by the polarity of the label densities. If we compare the local densities in the top graph with the IRLbl in the second graph, we observe that the IRLbl of the individual labels represents something similar to the inverse of the local label densities. If the local label density is low, the IRLbl will be high, and if the local density is high, the IRLbl will be low.

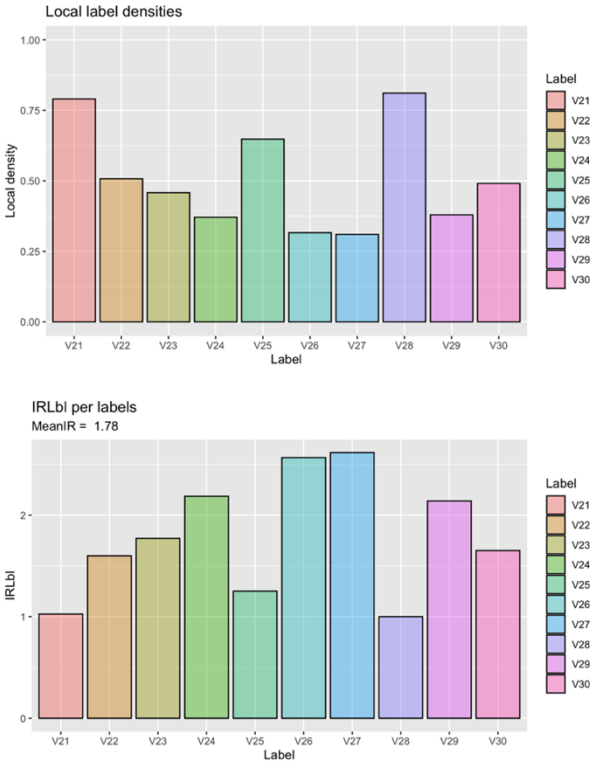


Figure 13: IRLbl vs Local label densities

The general purpose of resampling algorithms would be to balance the class distributions. Therefore, we want the labels with comparatively small local label density to be better represented in the dataset if oversampling is used or the labels with large local label density to be reduced if undersampling is used. The goal of the resampling algorithms is to reduce

the IRLbl of labels with high IRLbl scores. We are indirectly increasing the local density of labels with small local densities.

In Figure 14, we observe the effect of a resampling algorithm on the IRLbl of a dataset. The figure on the left gives the IRLbl of the dataset before resampling, and on the right-hand side, we observe the IRLbl of the datasets after resampling. The resampling algorithm ML-ROS was used for illustration purposes. The IRLbl has been reduced for all labels with large IRLbl on the left-hand side. A large IRLbl is indicative of a minority label. The MeanIR has also been reduced from 6.9 to 1.75.

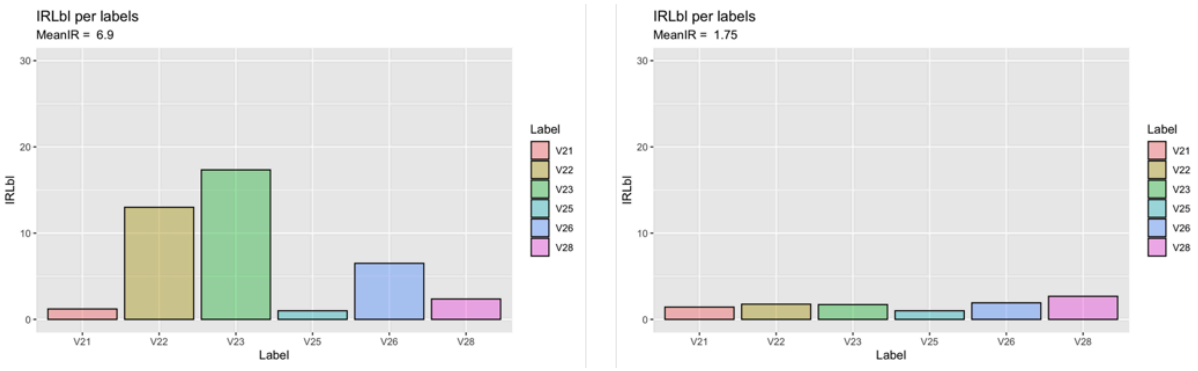


Figure 14: Change in IRLbl

In Figure 15, we observe the local label densities corresponding to the IRLbl scores seen above. We observe that the reduction in IRLbl seen above coincides with an increase in local label density seen below. Labels V22, V23 and V26 have experienced an increase in local label density because of the resampling algorithm. Therefore, labels with a high IRLbl score above have seen an increase in local label density because of the resampling algorithm.

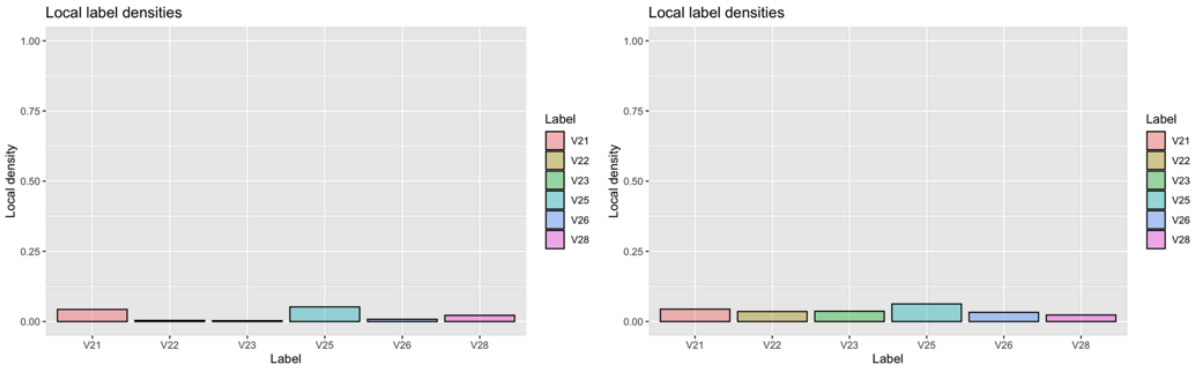


Figure 15: Change in local label density

Resampling algorithms work with a mechanism to identify minority and majority labels (minority labels being those labels with large IRLbl and small local label density and majority labels those with small IRLbl and large local label density). To address the imbalance in the dataset, we need to perform undersampling of the majority classes or oversampling of the minority classes. For undersampling or oversampling to be implemented, we need to identify minority and majority labels first to perform the resampling.

The most common mechanism seen in resampling algorithms for identifying minority and majority labels is categorising those labels with $IRLbl \leq MeanIR$ as majority labels and those labels with $IRLbl > MeanIR$ as minority labels. Figure 16 below illustrates this principle. On the left-hand side, we observe the IRLbl per label. The horizontal dashed red line represents the MeanIR. Therefore, all labels with IRLbl above the red line (MeanIR) are minority labels, and all labels with IRLbl smaller than the red line (MeanIR) are majority labels. In the figure on the right, we observe the distribution of the IRLbl scores for all labels. The exponential distribution is the general shape found for the distribution of IRLbl in most multi-label datasets. Labels that find their IRLbl to the right of the red line (MeanIR) are seen as minority labels, and those labels that find their IRLbl to the left of the red line are majority labels. An interesting thought experiment would be to use the medianIR instead of the MeanIR. The distribution of the IRLbl seen below is the shape often found in multi-label datasets and is skewed to the right. The medianIR would shift the red line leftwards and cause the algorithm to select more minority classes and fewer majority classes.

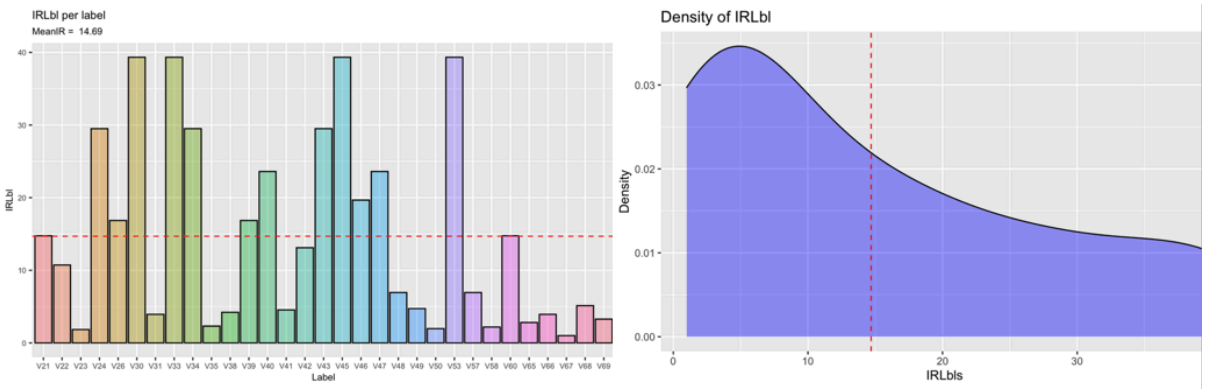


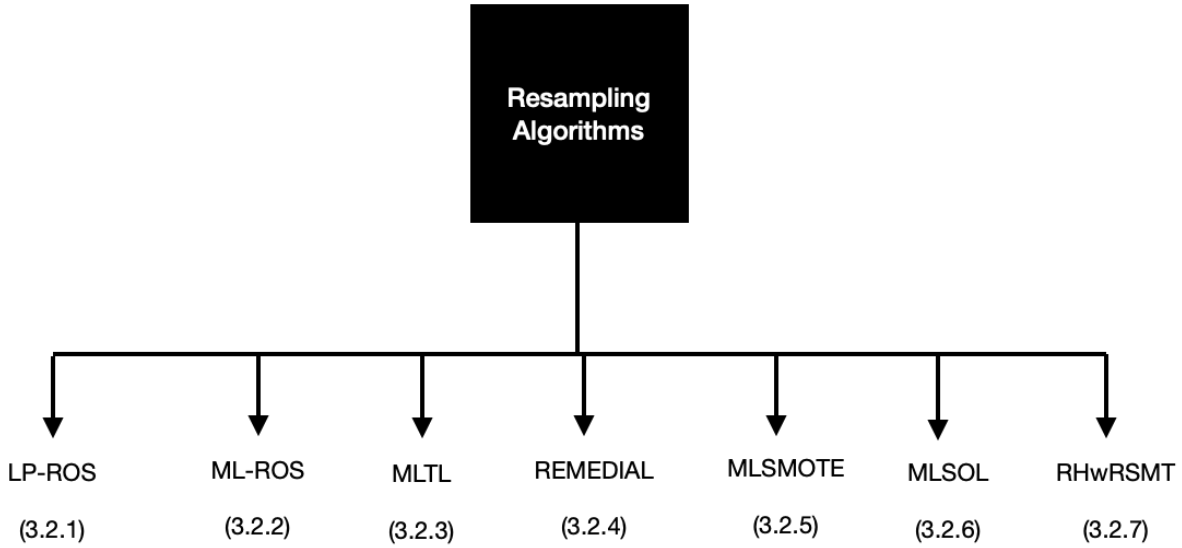
Figure 16: Distribution of IRLbl

Therefore, the traditional application of resampling algorithms in the context of multi-label classification is to reduce the MeanIR in a dataset by balancing the class distributions through some form of undersampling of majority classes or oversampling of minority classes, with the implicit assumption that this will improve classification performance. This investigation into resampling algorithms examines the usefulness of these algorithms in scenarios where the MeanIR is not necessarily high or indicative of imbalance, but where the polarity of label density is present and causes imbalance. Therefore, we wish to apply the resampling algorithms at divergent levels of global density to investigate whether the resampling algorithms effectively improve multi-label performance even when the MeanIR is not necessarily large.

The resampling algorithms are applied as a pre-processing step for multi-label classification. Therefore, if the multi-label data is split up into a test and training dataset, only the training data gets pre-processed using one of the resampling algorithms. It is important that the test dataset remains untouched and does not get pre-processed by the resampling algorithm. Therefore, the MLC model is fit on the training data that has been pre-processed and then evaluated on the test data that has remained untouched.

All the resampling algorithms used in this thesis are shown in the figure below. The following section provides an in-depth discussion of each of these algorithms. Of particular interest for each algorithm is

- The mechanism used to find minority and majority classes.
- The use of undersampling or oversampling.
- Presence of synthetic instance generation.
- Pseudocode of the algorithm.



The only resampling algorithm for multi-label data that is available in R or Python packages is REMEDIAL. Therefore, all the resampling algorithms were coded in R from first principles using the pseudocode and explanations of the algorithms in the literature.

3.2 Resampling algorithms

3.2.1 LP-ROS

Label-Powerset-Random Oversampling (LP-ROS) proposed in Charte et al. (2015a) is a resampling algorithm based on the Label Powerset approach and uses random oversampling of label-sets as its mechanism to generate duplicate observations. LP-ROS and LP-RUS (Undersampling version of LP-ROS) are analogous since they use the same mechanism to undersample and oversample the label-sets. LP-ROS is based on oversampling of label-sets, and LP-RUS is based on undersampling of label-sets. LP-ROS does not use synthetic instance generation since the random sampling of minority label-sets add new observations to the dataset. These new observations are not synthetically generated since they are clones of observations that are already in the dataset.

The foundational principle of the LP-ROS algorithm is to oversample the minority label-sets. Minority label-sets are those label-sets that occur a below-average amount of time. Therefore, the algorithm needs to identify the minority label-sets and randomly oversample these label-sets in an attempt to balance the class distributions of the dataset.

The LP-ROS algorithm needs to determine the number of samples to generate, which can be done by calculating a $P\%$ (user defined) size increase in the dataset. The dataset is transformed to LP form and split up so that all the observations from unique label-sets are put into their own bags. Therefore, each label-set has a bag of observations with that label-set. The average number of observations per bag is calculated. Minority bags are then identified as those bags with fewer observations than the average number of observations per label-set bag. These minority bags are created to facilitate oversampling. A mean increment is calculated as the number of samples to generate divided by the number of minority bags. The minority bags are processed from largest to smallest, random observations from the minority bag are sampled and added to the dataset. The sampling is repeated until the minority bag reaches the mean size (used to select the minority bags at the start) or until the mean increment number of samples has been added. The algorithm is described by the flow diagram in Figure 18.

The pseudocode for the LP-RUS (undersampling) algorithm is observed in Figure 17 below. A few small changes are made to accommodate LP-ROS (oversampling) from this pseudocode. A collection of minority groups $minBag_i$ with $(|labelsetBag_i| < meanSize)$ is obtained, a $meanInc = sampleGenerate/minBag$ is calculated. Processing the minority groups from the largest to the smallest, an individual increment for each $minBag_i$ is determined. If a $minBag_i$ reaches $meanSize$ samples before $incrementBag_i$ observations have been added, the excess is distributed among the other bags (Charte et al., 2015a).

Algorithm 1. LP-RUS algorithm's pseudo-code.

Inputs: ⟨Dataset⟩ D, ⟨Percentage⟩ P
Outputs: Preprocessed dataset
$samplesToDelete \leftarrow |D|/100 * P$ ▷ P% size reduction
▷ Group samples according to their labelsets
$MeanIR \leftarrow$ calculateMeanIR(D, L)
for each *label* **in** L **do** ▷ Bags of minority labels samples
  $IRLbl_{label} \leftarrow$ calculateIRperLabel$(D, label)$
  if $IRLbl_{label} > MeanIR$ **then**
    $minBag_{i++} \leftarrow Bag_{label}$
  end if
end for
while $samplesToClone > 0$ **do** ▷ Instances cloning loop
  ▷ Clone a random sample from each minority bag
  for each $minBag_i$ **in** $minBag$ **do**
    $x \leftarrow$ random$(1, |minBag_i|)$
    cloneSample$(minBag_i, x)$
    if $IRLbl_{minBag_i} <= MeanIR$ **then**
      $minBag \rightarrow minBag_i$ ▷ Exclude from cloning
    end if
    $--samplesToClone$
  end for
end while

Figure 17: LP-RUS pseudo code (Charte et al., 2015a)

LP-ROS is typically preferred over LP-RUS because of its use of oversampling instead of undersampling. Even in the best of cases, multi-label data leads to a sparse data matrix. Undersampling the majority classes will lead to an even further increases in the sparseness of the data matrix—an undesirable outcome for multi-label performance. Therefore, the oversampling version of the LP-resampling algorithms is preferred (Charte et al., 2015a).

Potential issues occur with the LP-RUS and LP-ROS algorithms when there are many distinct label-sets. Many label-sets distort the notion of which label-sets are majorities and minorities since most of the label-sets will have very few observations. A dataset such as CAL500 (Turnbull et al., 2008) presents a unique case where each observation in the dataset has a unique label-set. Therefore, each label-set bag will contain only one observation, which will prove problematic for the LP-RUS and LP-ROS algorithms since all label-set bags will be minorities and majorities at the same time.

Figure 18 describes the LP-ROS algorithm in a flow-diagram.

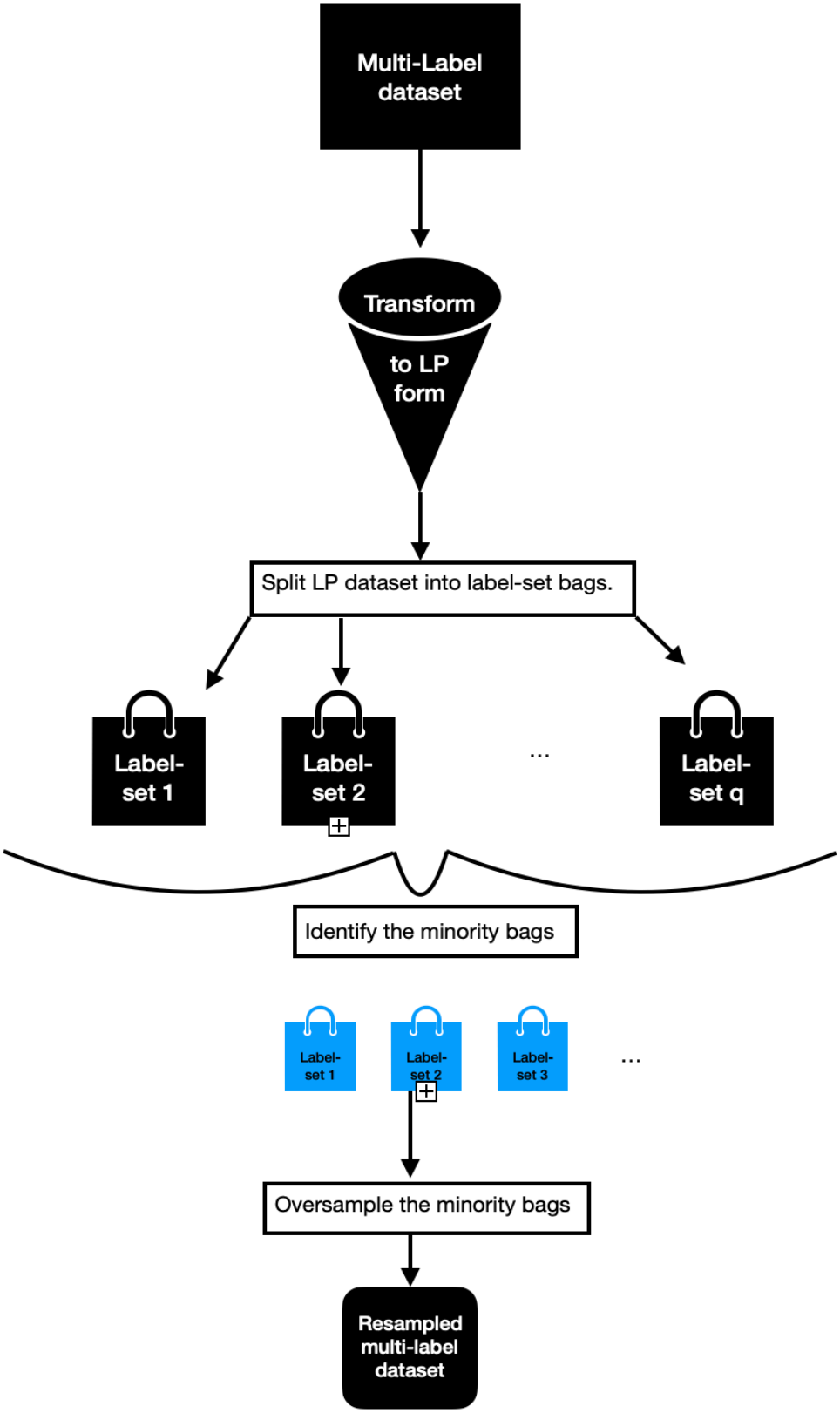


Figure 18: LP-ROS flow-diagram

3.2.2 ML-ROS

Multi-label-Random Oversampling (ML-ROS) proposed in Charte et al. (2015a) is a label based random oversampling algorithm. ML-ROS differs from the LP-based methods seen in Section 3.2.1 since it uses individual labels to identify the observations to oversample, rather than using label-sets to identify minority observations. Minority labels are identified as labels with $IRLbl > MeanIR$. No synthetic instance generation is used since observations are randomly oversampled from bags of minority observations.

The number of samples to generate is given by a $P\%$ (user defined) increase in the overall size of the dataset. The first step is to identify all the minority labels, therefore those labels with $IRLbl > MeanIR$. Observations that contain minority labels are placed into minority bags. Each minority label has its bag containing all the observations that have that label. The next step is to oversample these minority bags. The number of samples to clone/oversample is determined by the $P\%$ increase in the overall size of the dataset. While the samples to clone is greater than zero, we iterate over the bags of labels and oversample one random observation from each bag—every time an observation is oversampled, the number of samples to clone decreases by one. When the IRLbl of a label is no longer greater than the MeanIR, that bag is removed from the bags to oversample from since it is no longer a minority label.

Some of the criticisms towards the ML-ROS resampling algorithm comes from the presence of concurrence in some datasets. Concurrence is when there are some minority labels that are correlated to majority labels. Therefore, there will be observations in the dataset that belong to both minority labels and majority labels. When we oversample these observations, we are inflating the minority labels as well as the majority labels—mitigating the usefulness of the resampling. For this reason, ML-ROS is less effective on datasets with high SCUMBLE (Charte et al., 2015a).

ML-ROS is preferred over ML-RUS (undersampling version of ML-ROS) due to its use of oversampling rather than undersampling. In multi-label data we often deal with a sparse data matrix, therefore undersampling will lead to a loss of information in the dataset that is not desirable, especially at smaller global densities. (Charte el al., 2015a)

The pseudocode and flow diagram for the ML-ROS algorithm is observed below in Figure 19 and 20:

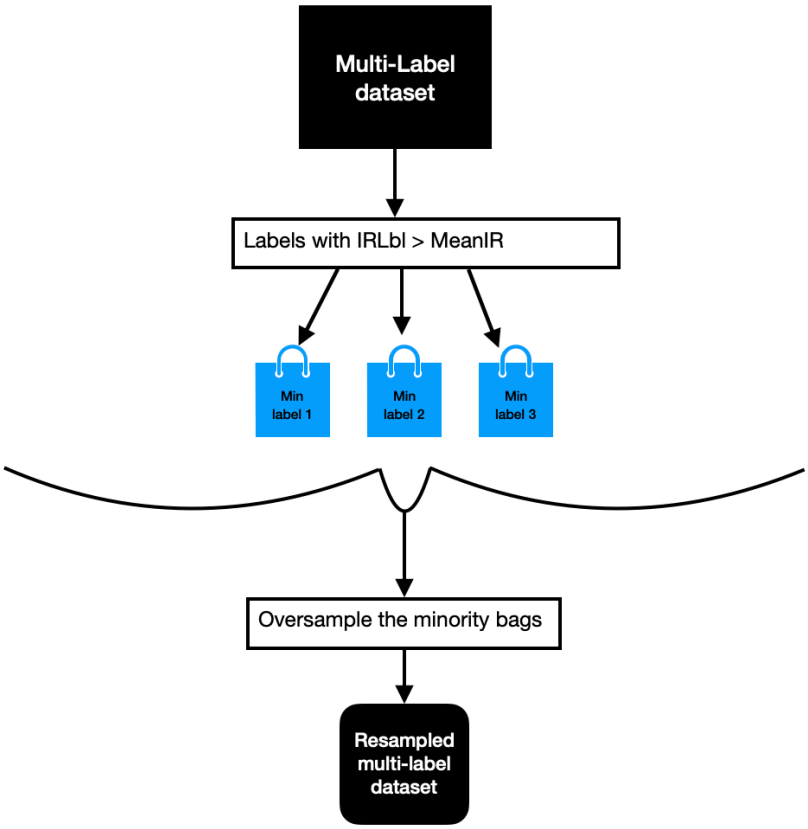


Figure 20: ML-ROS flow-diagram

Algorithm 2. ML-ROS algorithm's pseudo-code.

Inputs: ⟨Dataset⟩ D, ⟨Percentage⟩ P
Outputs: Preprocessed dataset
$samplesToClone \leftarrow |D|/100 * P$ ▷ P% size increment
$L \leftarrow$ labelsInDataset(D) ▷ Obtain the full set of labels
$MeanIR \leftarrow$ calculateMeanIR(D, L)
for each *label* **in** L **do** ▷ Bags of minority labels samples
  $IRLbl_{label} \leftarrow$ calculateIRperLabel($D, label$)
  if $IRLbl_{label} > MeanIR$ **then**
    $minBag_{i++} \leftarrow Bag_{label}$
  end if
end for
while $samplesToClone > 0$ **do** ▷ Instances cloning loop
  ▷ Clone a random sample from each minority bag
  for each $minBag_i$ **in** $minBag$ **do**
    $x \leftarrow$ random$(1, |minBag_i|)$
    cloneSample($minBag_i, x$)
    if $IRLbl_{minBag_i} <= MeanIR$ **then**
      $minBag \rightarrow minBag_i$ ▷ Exclude from cloning
    end if
    $--samplesToClone$
  end for
end while

Figure 19: ML-ROS pseudo code (Charte et al., 2015a)

3.2.3 MLTL

3.2.3.1 Tomek-Links:

The traditional use of Tomek-Links (Kubat, 2017) is as a preprocessing algorithm for binary and multi-class classification data. The goal of the algorithm is to remove ambiguous observations from the dataset. Ambiguous observations are those observations that lie on the boundary between two classes.

Three conditions need to be true for two observations X1 and X2, to be considered a Tomek-Link:

I. X1 needs to be the nearest neighbour of X2
II. X2 needs to be the nearest neighbour of X1
III. X1 and X2 need to belong to different classes.

The diagram below explains the idea of what a Tomek-Link is. Those pairs of observations that have been circled are seen as Tomek-Links. These observations are each other's nearest neighbours and belong to opposite classes (blue and red). The topmost red observation is not part of a Tomek-Link since the blue observation is its nearest neighbour, but the red observation is not the blue observations nearest neighbour.

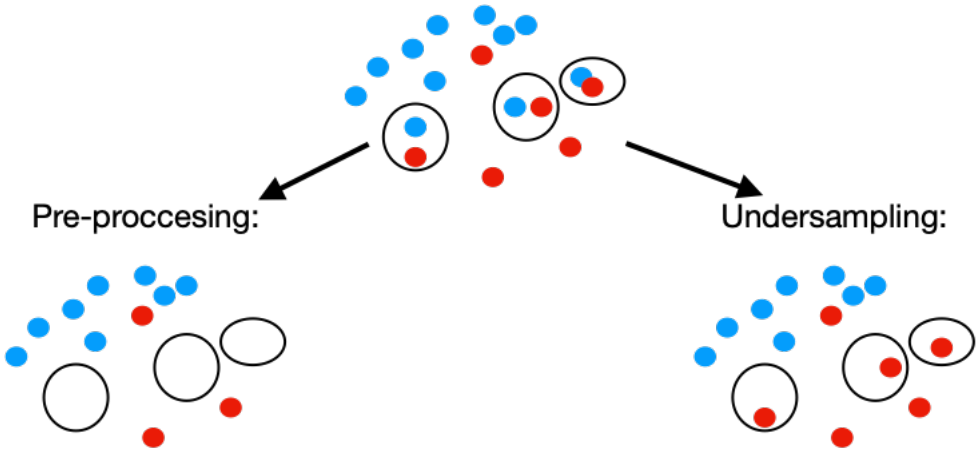


Figure 21: Tomek-Links

The standard procedure is to remove Tomek-Link pairs from a dataset, which should lead to an improvement in the data quality through the removal of ambiguous observations. One needs to be careful when datasets are small since removing more observations from a small dataset could be detrimental to the analysis. Using Tomek-Links for an imbalanced dataset could also be troublesome. Removing all the Tomek-Links could potentially lead to the removal of many of the minority class observations. The solution to the imbalanced class problem for Tomek-Links leads us to the undersampling proposition by the Tomek-Link algorithm. Instead of removing the pair of observations found to be a Tomek-Link, only the majority class observation is removed from the dataset, therefore leaving the minority class intact while still removing some of the ambiguity from the dataset.

3.2.3.2 Multi-label Tomek-Links

Multi-label Tomek-Links (MLTL) proposed by Pereira et al. (2020) is based on a similar idea. There are however some challenges that the multi-label paradigm poses to the algorithm above:

- Observations can belong to more than one majority class.
- The algorithm needs to determine which classes are majority classes.
- The algorithm needs to determine whether observations belong to different classes, even though they can belong to more than label.

To determine the majority classes, MLTL uses the classical mechanism that takes all labels with $IRLbl \leq MeanIR$ as majority labels. Finding the nearest neighbours of an observation is not difficult for multi-label data, as this is dependent on the predictor variables X and not the labels. However, to determine if two observations are Tomek-Links, we need to determine if they have opposite classes. The MLTL algorithm proposes the use of the Hamming distance to determine the similarity between the label-sets of two observations.

The Hamming Distance (HD) calculates how different the label-sets of two observations are. Instead of using the normal Hamming distance, the adjusted Hamming distance is proposed. The adjusted Hamming Distance (AHD) considers the active labels between the two considered observations instead of all the labels. The active labels can be seen as the union of the two label-sets. Using the Adjusted Hamming Distance, instead of the normal Hamming distance, will allow us to avoid very low scores when there are many labels in the dataset.

Formally the process is as follows. Given two vectors of binary numbers a and b of length N, consider $\sum_i a_i$ and $\sum_i b_i$. Let $XOR(a, b)$, be the "exclusive or" between the vectors a and b. The HD and AHD can be calculated using the following formulae:

$$HD_{\%}(a,b) = \frac{\sum_i^N \big(XOR(a,b)\big)_i}{N}$$

$$AHD_{\%}(a,b) = \frac{\sum_i^N \big(XOR(a,b)\big)_i}{\sum_i^N \big(OR(a,b)\big)_i}, 0 \leq AHD_{\%}(a,b) \leq 1$$

The larger the AHD is, the more significant the difference between the vectors a and b. A large AHD between a and b indicates that they are dissimilar. Therefore, an indication that they are Tomek-Links and should be removed. However, the question arises what should be considered as a large AHD.

A threshold is needed to determine whether the AHD is large enough to warrant that the two observations are indeed Tomek-Links. This threshold (TH) is based on the level of imbalance in the dataset. The threshold is determined using the following formulae:

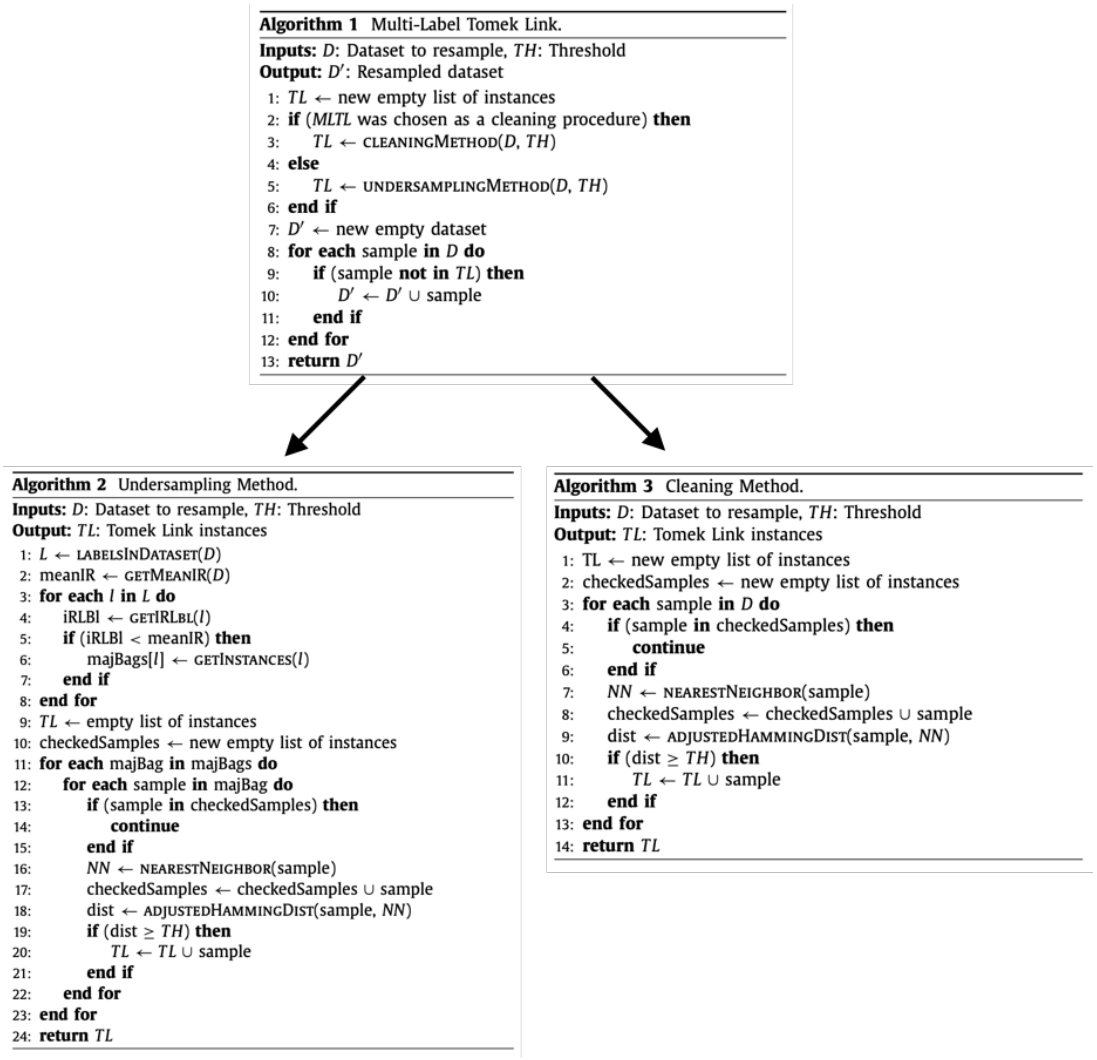
$$I = \frac{1}{\sqrt{MeanIR}}$$

$$TH = 0.5, if\ I \geq 0.5$$

$$TH = 0.3, if\ 0.5 > I \geq 0.3$$

$$TH = 0.15, if\ I < 0.3$$

Therefore, if the AHD of two observations exceeds TH, the observations are seen as Tomek-Links. The Multi-label Tomek Link (MLTL) approach can also be split up into an undersampling or preprocessing/cleaning method. The pseudocode of the MLTL approach is observed below.



Algorithm 1 Multi-Label Tomek Link.

Inputs: D: Dataset to resample, TH: Threshold
Output: D': Resampled dataset

```
TL ← new empty list of instances
if (MLTL was chosen as a cleaning procedure) then
    TL ← CLEANINGMETHOD(D, TH)
else
    TL ← UNDERSAMPLINGMETHOD(D, TH)
end if
D' ← new empty dataset
for each sample in D do
    if (sample not in TL) then
        D' ← D' ∪ sample
    end if
end for
return D'
```

Algorithm 2 Undersampling Method.

Inputs: D: Dataset to resample, TH: Threshold
Output: TL: Tomek Link instances

```
L ← LABELSINDATASET(D)
meanIR ← GETMEANIR(D)
for each l in L do
    iRLBl ← GETIRLBL(l)
    if (iRLBl < meanIR) then
        majBags[l] ← GETINSTANCES(l)
    end if
end for
TL ← empty list of instances
checkedSamples ← new empty list of instances
for each majBag in majBags do
    for each sample in majBag do
        if (sample in checkedSamples) then
            continue
        end if
        NN ← NEARESTNEIGHBOR(sample)
        checkedSamples ← checkedSamples ∪ sample
        dist ← ADJUSTEDHAMMINGDIST(sample, NN)
        if (dist ≥ TH) then
            TL ← TL ∪ sample
        end if
    end for
end for
return TL
```

Algorithm 3 Cleaning Method.

Inputs: D: Dataset to resample, TH: Threshold
Output: TL: Tomek Link instances

```
TL ← new empty list of instances
checkedSamples ← new empty list of instances
for each sample in D do
    if (sample in checkedSamples) then
        continue
    end if
    NN ← NEARESTNEIGHBOR(sample)
    checkedSamples ← checkedSamples ∪ sample
    dist ← ADJUSTEDHAMMINGDIST(sample, NN)
    if (dist ≥ TH) then
        TL ← TL ∪ sample
    end if
end for
return TL
```

Figure 22: MLTL pseudo code (Pereira et al., 2020)

3.2.3.2.1 Undersampling

For the undersampling method, the labels are split up into majority bags. Therefore, all observations that contain a majority class label are put into the majority bag of the corresponding label. The AHD from every observation in the majority bags to its nearest neighbour is calculated. If the AHD is larger than TH, the observation is added to the array of Tomek-Links,. Only the majority class observation is added to the array and not the nearest neighbour. An observation is not checked twice. Therefore, if an observation belongs to more than one majority label, it does not get rechecked. All of the majority class observations in the array of Tomek-Links are removed from the dataset.

3.2.3.2.2 Cleaning

For the cleaning method, a similar approach is followed. However, instead of just the observations belonging to the majority classes being checked, all of the observations are checked. Therefore, we calculate the AHD from all observations to their nearest neighbour and check if it is greater than TH. If the AHD > TH, the observation and its nearest neighbour are added to an array of Tomek-Links marked for removing.

For this thesis, only the undersampling algorithm will be studied. Therefore, MLTL is an undersampling technique that does not make use of synthetic instance generation.

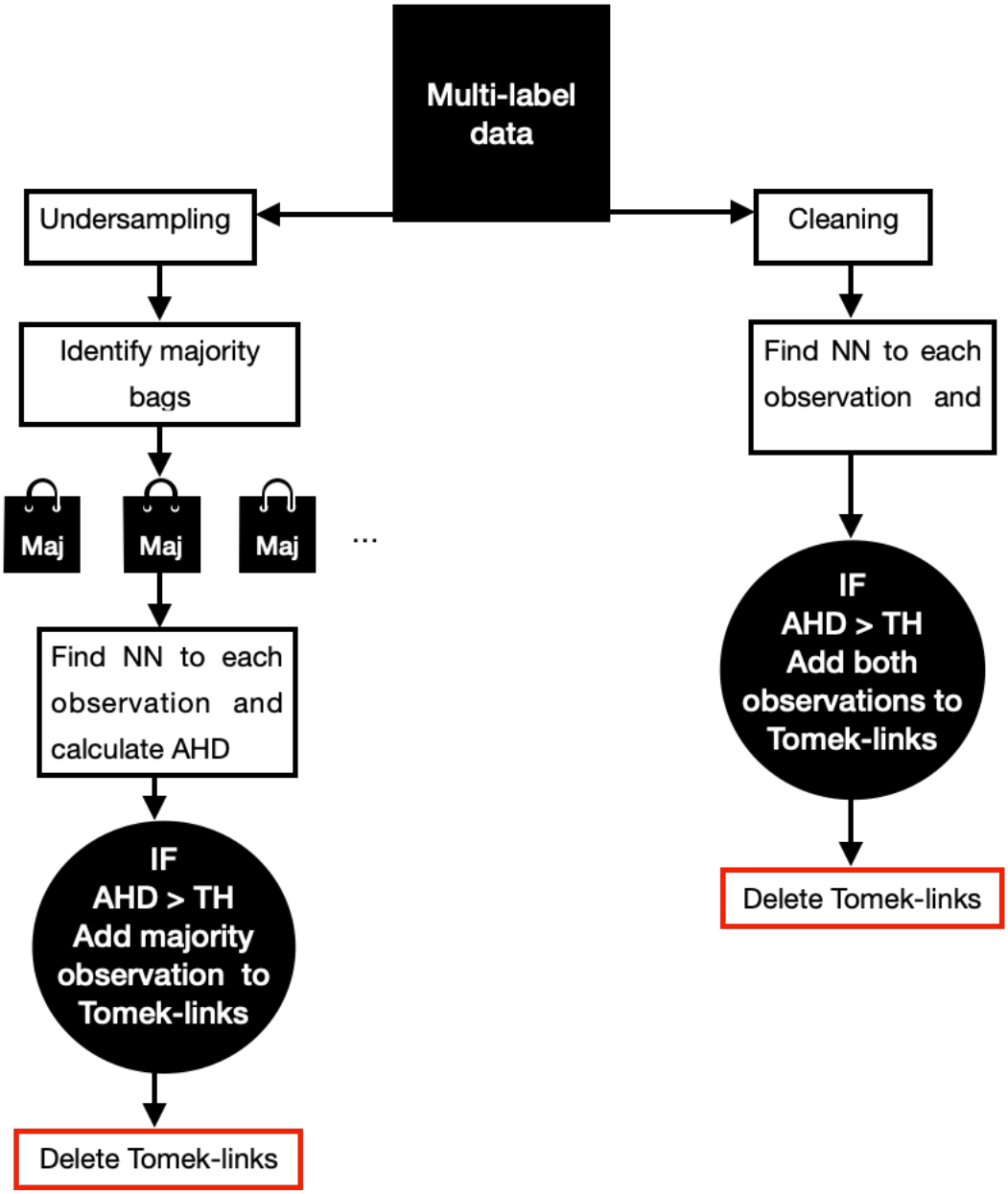


Figure 23: MLTL flow-diagram

3.2.4 REMEDIAL

Resampling multi-label datasets by decoupling highly imbalanced labels (REMEDIAL) proposed by Charte et al., (2019a) is an oversampling algorithm. It has specifically been developed for multi-label datasets with concurrence between imbalanced labels, therefore datasets with high SCUMBLE. REMEDIAL attempts to decouple observations that belong to both minority and majority classes. REMEDIAL splits existing observations into two observations, one with the minority labels and the other with the majority labels. We do not necessarily expect REMEDIAL to be competitive with any of the other resampling algorithms in all instances since it was developed for datasets with large SCUMBLE. However, it would be worthwhile to investigate the effect that increasing global density has on the label decoupling technique.

The following formulae were defined in Chapter 1 and are used to decide which observations need to be decoupled into majority and minority labels. $SCUMBLE_{ins}$ refers to the $SCUMBLE$ per individual observation and $SCUMBLE(D)$ refers to the overall $SCUMBLE$ in the dataset.

$$SCUMBLE_{ins} = 1 - \frac{1}{\overline{IRLbl_i}}\left(\prod_{k=1}^{K} IRLbl_{ik}\right)$$

$$SCUMBLE(D) = \frac{1}{N}\sum_{i=1}^{N} SCUMBLE_{ins}(i)$$

The observations with $SCUMBLE_{ins} > SCUMBLE(D)$ will be decoupled. Therefore, all the observations that have an above-average amount of $SCUMBLE$ will be decoupled. By decoupling, we mean that the observations will be split into two observations. One will contain all the majority labels, and the other will contain all the minority labels. The majority labels are those with $IRLbl \leq MeanIR$, and the minority labels are those with $IRLbl > MeanIR$. The pseudocode for the algorithm is found below:

```
1: function REMEDIAL(MLD D, Labels L)
2:     ▷ Calculate imbalance levels
3:     IRLbl_l ← calculateIRLbl(l in L)
4:     IRMean ← mean(IRLbl)
5:     ▷ Calculate SCUMBLE
6:     SCUMBLEIns_i ← calculateSCUMBLE(D_i in D)
7:     SCUMBLE ← mean(SCUMBLEIns)
8:     for each instance i in D do
9:         if SCUMBLEIns_i > SCUMBLE then
10:            D'_i ← D_i          ▷ Clone the affected instance
11:            ▷ Maintain minority labels
12:            D_i[labels_{IRLbl<=IRMean}] ← 0
13:            ▷ Maintain majority labels
14:            D'_i[labels_{IRLbl>IRMean}] ← 0
15:            D ← D + D'_i
16:        end if
17:    end for
18: end function
```

Figure 24: REMEDIAL pseudo code (Charte et al., 2019a)

REMEDIAL is technically an oversampling algorithm since it produces new observations. The decoupling process produces new observations but also modifies existing observations. One unique feature of REMEDIAL that is not seen in other resampling algorithms is that it does not change the label frequencies. The number of observations belonging to the majority and minority labels does not change, even though the composition of the dataset has been changed. REMEDIAL will not affect the $IRLbl$ of individual labels or the MeanIR. It is, however, effective at reducing the overall SCUMBLE in the dataset. The decoupling of labels in observations that contain both minority and majority labels will cause the dataset to have fewer observations that contain both minority and majority labels simultaneously. Therefore, the correlation between majority and minority labels will be reduced, causing a reduction in SCUMBLE.

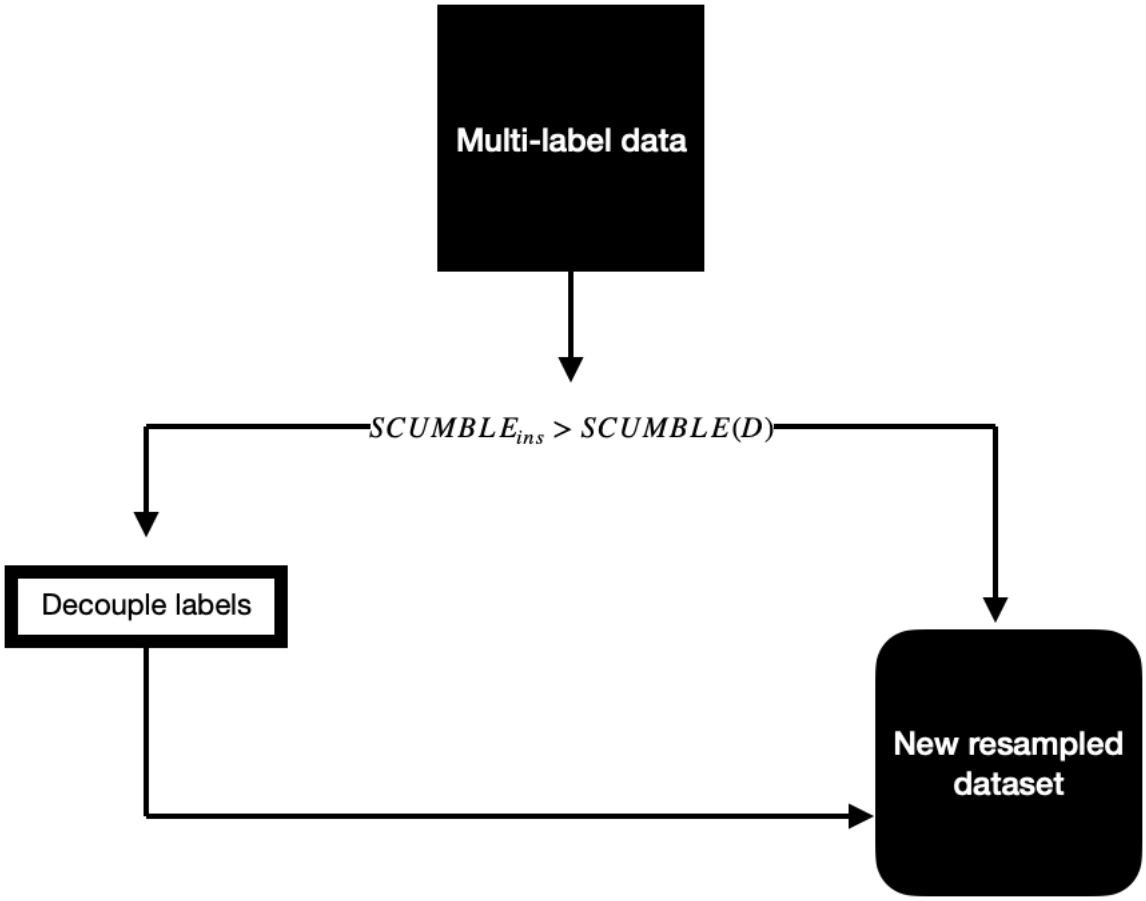


Figure 25: REMEDIAL flow-diagram

3.2.5 MLSMOTE

3.2.5.1 SMOTE (binary classification)

Synthetic Minority Oversampling Technique (SMOTE) (Chawla et al., 2002), is one of the most popular techniques to oversample a dataset using synthetic instance generation. Synthetic instance generation refers to the production of new minority samples that get added to the dataset. Therefore, SMOTE does not oversample existing minority observations, but rather produces new synthetic observations.

The general procedure for SMOTE in the case of binary classification is as follows. For all observations in the minority class, pick a random instance among the k nearest neighbours of that instance (minority class neighbours). A new instance is generated by interpolating the features between the observation and the random neighbour. This new synthetic observation also belongs to the minority class. This procedure will produce new artificial minority class observations that will be similar to the existing ones. It is reasonable to assume that this procedure should lead to a more balanced dataset and improve classification performance.

The diagram below gives a visualisation of the SMOTE procedure. For all the minority observations, pick a random neighbour amongst the k nearest neighbours. Produce a new minority class observation by interpolating the features from the minority observation and the random neighbour. The synthetic observation need not lie precisely between the minority observation and the random neighbour but could be anywhere on the dashed line.

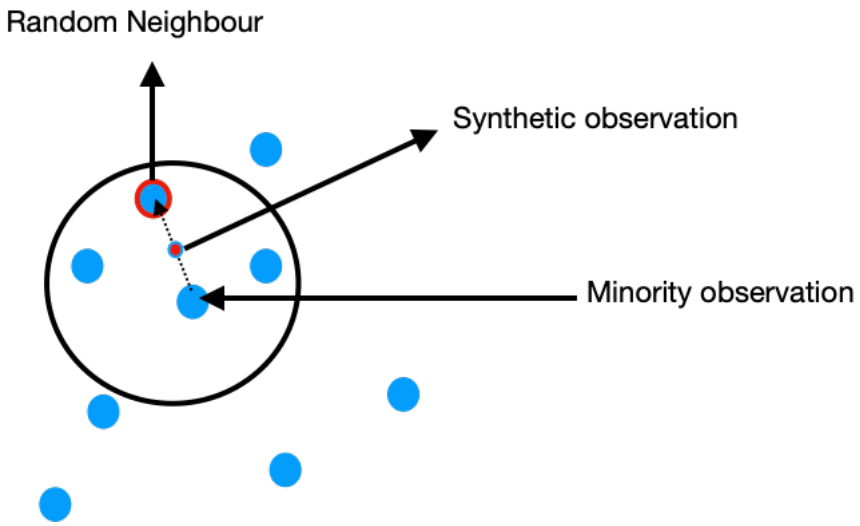


Figure 26: SMOTE

Chawla et al. (2002) found that SMOTE could improve the accuracy of classification for minority classes and performed better than plain undersampling. Out of a total of 48 experiments, SMOTE was only outperformed on 4 occasions.

3.2.5.2 MLSMOTE

Multi-label Synthetic Oversampling Technique (MLSMOTE) proposed by Charte et al. (2015), attempts to extend SMOTE to the multi-label paradigm. As we saw with MLTL, extending a single-label paradigm to the multi-label paradigm is not that straightforward. Many changes need to be accounted for, for instance how the nearest neighbours will be determined, which observations will be regarded as minorities and what label-set should be assigned to the new synthetic instance.

The MLSMOTE proposal is as follows. Iterating over the labels, check if the label is a minority label. Minority labels are defined to have $IRLbl > MeanIR$. If the label is a minority label, place all the observations belonging to that label in a bag. For each observation in the minority bag, find the k nearest neighbours. Select a random neighbour from the k neighbours. Generate a new synthetic observation using algorithm 2, with the minority observation and the random neighbour as parameters. Interpolate the features between the minority bag observation and the neighbour according to lines 23-33 in algorithm 2 and assign these features to the synthetic observation. Generate the label-set for the new synthetic instance according to lines 35-38 of algorithm 2 and assign the label-set to the new observation. Add the new synthetic observation to the dataset. Repeat for all labels. If the IRLbl of a label reaches the MeanIR, this label is no longer oversampled.

Algorithm 1. MLSMOTE algorithm's pseudo-code.

```
Inputs:
    D   ▷ Dataset to oversample
    k   ▷ Number of nearest neighbors
 1: L ← labelsInDataset(D)   ▷ Full set of labels
 2: MeanIR ← calculateMeanIR(D, L)
 3: for each label in L do
 4:     IRLbl_label ← calculateIRperLabel(D, label)
 5:     if IRLbl_label > MeanIR then
 6:         ▷ Bags of minority labels samples
 7:         minBag ← getAllInstancesOfLabel(label)
 8:         for each sample in minBag do
 9:             distances ← calcDistance(sample, minBag)
10:             sortSmallerToLargest(distances)
11:             ▷ Neighbor set selection
12:             neighbors ← getHeadItems(distances, k)
13:             refNeigh ← getRandNeighbor(neighbors)
14:             ▷ Feature set and labelset generation
15:             synthSmpl ← newSample(sample,
16:                         refNeigh, neighbors)
17:             D = D + synthSmpl
18:         end for
19:     end if
20: end for
```

Algorithm 2. Function: Generation of new synthetic instances.

```
21: function NEWSAMPLE(sample, refNeigh, neighbors)
22:     synthSmpl ← new Sample   ▷ New empty instance
23:     ▷ Feature set assignment
24:     for each feat in synthSmpl do
25:         if typeOf(feat) is numeric then
26:             diff ← refNeigh.feat − sample.feat
27:             offset ← diff ∗ randInInterval(0,1)
28:             value ← sample.feat + offset
29:         else
30:             value ← mostFreqVal(neighbors,feat)
31:         end if
32:         synthSmpl.feat ← value
33:     end for
34:     ▷ Label set assignment
35:     lblCounts ← counts(sample.labels)
36:     lblCounts + ← counts(neighbors.labels)
37:     labels ← lblCounts > (k + 1)/2
38:     synthSmpl.labels ← labels
39:     return synthSmpl
40: end function
```

Figure 27: MLSMOTE pseudo code (Charte et al., 2015)

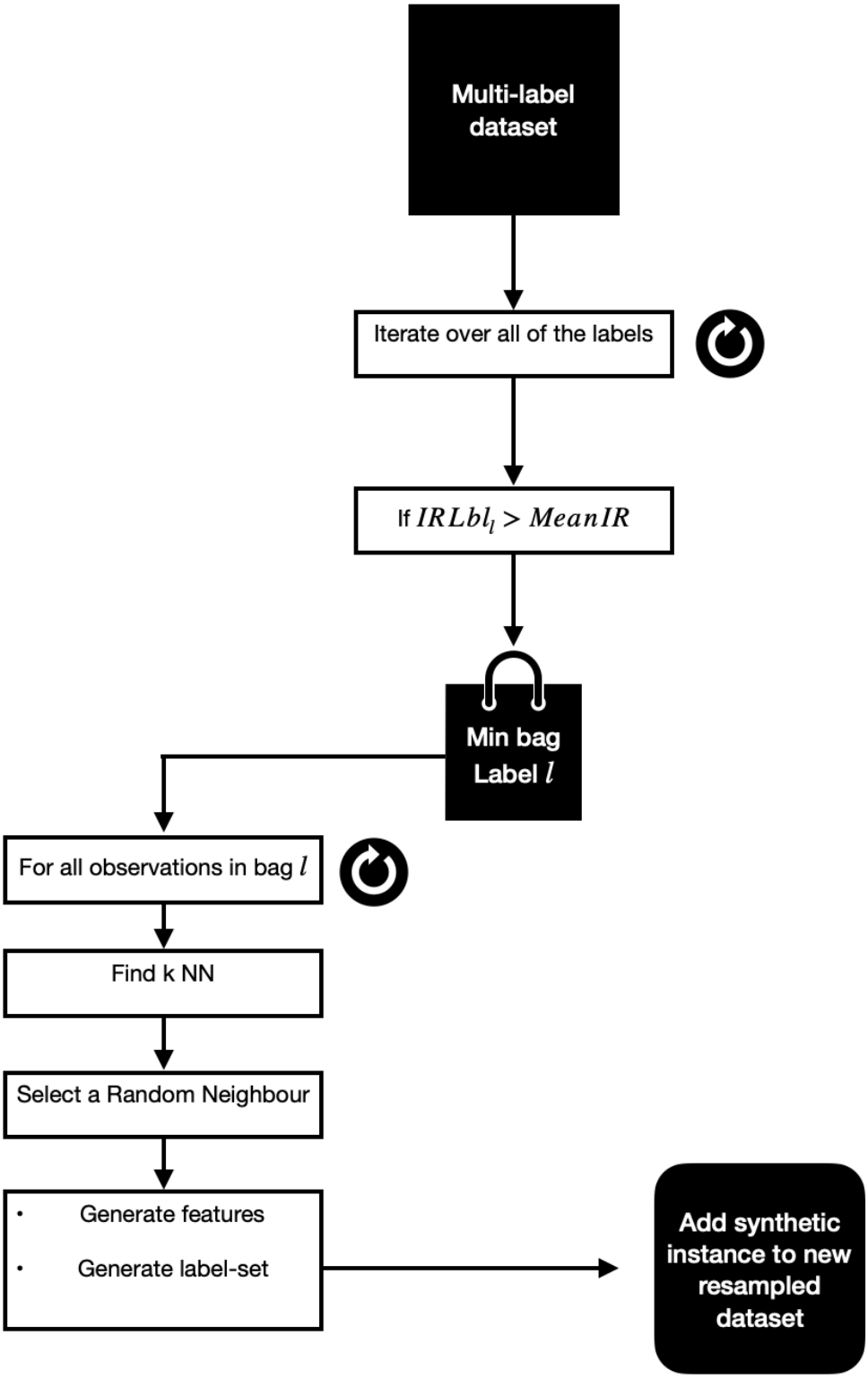


Figure 28: MLSMOTE flow-diagram

3.2.6 MLSOL

Synthetic Oversampling of Multi-Label Data based on Local Label Distribution (MLSOL) proposed by Liu & Tsoumakas (2020a) is the most elaborate algorithm we will consider. MLSOL is an oversampling algorithm that makes use of synthetic instance generation based on local label distributions. Making use of the local label distributions allows MLSOL to generate more diverse and better-labelled observations. MLSOL can be seen as a direct opponent to MLSMOTE. Both algorithms propose the synthetic generation of observations based on the local neighbourhood of observations.

MLSOL is a complicated resampling algorithm and makes use of a number of mechanisms to facilitate the generation of synthetic observations in order to balance the class distributions. Steps 1,2 and 3 work together as a training step before synthetic observations are generated. These three steps have the purpose of calculating a weight (w_i) that is assigned to each observation that represents how difficult it is to classify the minority labels for the observation. The weights calculated in step 3 are used to select the most appropriate seed observations used in steps 4 and 5 to generate the synthetic observations. The larger the weight of an observation is, the larger is the probability that it will be selected as a seed observation. Therefore, the observations whose minority labels are the hardest to classify are the most likely to be selected as seed observations in the algorithm since these observations will have the largest weights. Once a seed observation has been selected, a random neighbour is selected as a reference instance and a new observation is generated according to the algorithm on page 71, using the seed instance and the randomly selected neighbour as a reference instance. The generation of new synthetic observations is repeated in steps 4 and 5, until enough samples have been generated.

A summary of the algorithm is given below. Each section within the summary is discussed separately, referring to the pseudocode.

Step 1: Find bNN to each observation:
Using the Euclidean distance, the b nearest neighbours to every observation in the dataset is calculated. The result is a n by b matrix with the distance to the b closest observations for each observation $i \in \{1,2,\dots,n\}$ in the dataset. Another n by b matrix is created with the index of the nearest neighbours in the original dataset. These neighbours are referenced at a later stage.

Step 2: Calculate C
The MLSOL algorithm samples seed observations with replacement. The probability of selecting an observation is proportional to the number of minority class observations in its neighbourhood. Therefore, the probability of selecting an observation is weighted by the proportion of majority class observations in its b nearest neighbours. Calculating C is an intermediate step in generating a sampling weight for each observation based on its local neighbourhood.

$$C_{ik} = \frac{1}{b} \sum_{x_m \in bNN(x_i)} [[y_{mk} \neq y_{ik}]]\text{, where } C_{i,k} \in \{0,1\}$$

A value of $C_{i,k}$ close to zero is indicative of a safe environment, where the observation is surrounded mainly by similarly labelled observations. A value of $C_{i,k}$ close to one is indicative of a hostile environment, where the observation is surrounded mainly by oppositely labelled observations. It is reasonable to assume that the difficulty of classifying an observation is proportional to the number of opposite class values in its neighbourhood.

Step 3: Calculate w

To arrive at a single sampling weight w_i per observation, $C_{i,k}$ needs to be aggregated per observation. Therefore, each observation x_i will receive a corresponding weight w_i, $\forall\, i \in \{1,2,\dots,n\}$. w_i represents the difficulty of correctly predicting the minority class. Larger values of w_i relate to observations that are more likely to be selected in the random sampling of observations and vice versa for smaller values of w_i.

$$w_i = \sum_{k=1}^{K} \frac{C_{i,k}\left[\left[y_{i,k}=1\right]\right]\left[\left[C_{i,k}<1\right]\right]}{\sum_{i=1}^{n} C_{i,k}\left[\left[y_{i,k}=1\right]\right]\left[\left[C_{i,k}<1\right]\right]}$$

Probability Proportionate to Size (PPS) sampling is used to select random seed observations based on w_i. A uniformly distributed random number between 0 and 1 is generated and is multiplied with the sum of all the weights ($\sum_{i=1}^{n} w_i$). The index corresponding to the interval within which the random number falls is the randomly selected observation.

In Table 2 below, we observe an example of one seed instance being selected. The uniformly distributed random number between 0 and 1 is generated and multiplied with $\sum_{i=1}^{n} w_i$ and equals 0.25. We observe that 0.25 falls in the interval $(0.1, 0.3]$. This interval relates to $i = 2$. Therefore, observation two is selected as a seed instance.

Table 2: Selecting seed observations for MLSOL

Index	Weight (w_i)	Cumulative weight	Interval	Weighted Random Number
1	0.1	0.1	[0,0.1)	
2	0.2	0.3	(0.1,0.3]	0.25
…	…	…	…	
N	0.05	$\sum_{i=1}^{N} w_i$	$\left(\dots, \sum_{i=1}^{N} w_i\right]$	

The probability of selecting observation i as a seed instance is equal to $\frac{w_i}{\sum_{i=1}^{n} w_i}$. Therefore, the larger w_i is, the larger the probability is of observation i being selected.

Step 4: Find observation types
When the new synthetic observations are generated, the type of observation we are dealing with is essential for label assignment. The minority observations will be split into four different types: Safe(SF), Borderline(BD), Rare(RR), and Outlier(OT).

- Safe (SF): $0 \leq C_{i,k} < 0.3$ A safe observation is in a region dominated by minority examples.
- Borderline (BD): $0.3 \leq C_{i,k} < 0.7$ Located close to the decision boundary between majority and minority classes.
- Rare (RR): $0.7 \leq C_{i,k} < 1$ Located in the majority region and is far from the decision boundary.
- Outlier (OT): $C_{i,k} = 1$ Surrounded by majority examples.

The split is done according to the proportion of neighbours from the same minority class. The pseudocode is outlined in algorithm 2.

Step 5: Synthetic observation generation
Choose a random seed observation, using w_i as set out in step 3. Randomly select one of the b nearest neighbours to the seed observation as a reference observation. According to the pseudocode in algorithm 3, generate a new observation using the seed observation and reference observation. Add the new synthetic observation to the dataset. Repeat this process of generating new observations until enough samples have been generated.

Algorithm 1: MLSOL

input : multi-label data set: D, percentage of instances to generated: P, number of nearest neighbour: k
output: new data set D'

$GenNum \leftarrow |D| * P$; `/* number of instances to generate */`
$D' \leftarrow D$;
Find the kNN of each instance ;
Calculate $\boldsymbol{C}$ according to Eq.(1) ;
Compute $\boldsymbol{w}$ according to Eq.(3) ;
$\boldsymbol{T} \leftarrow$ `InitTypes`$(\boldsymbol{C}, k)$; `/* Initialize the type of instances */`
while $GenNum > 0$ **do**
    Select a seed instance $(\boldsymbol{x}_s, \boldsymbol{y}_s)$ from D based on the $\boldsymbol{w}$;
    Randomly choose a reference instance $(\boldsymbol{x}_r, \boldsymbol{y}_r)$ from $kNN(\boldsymbol{x}_s)$;
    $(\boldsymbol{x}_c, \boldsymbol{y}_c) \leftarrow$ `GenerateInstance` $((\boldsymbol{x}_s, \boldsymbol{y}_s), T_s, (\boldsymbol{x}_r, \boldsymbol{y}_r), T_r)$;
    $D' \leftarrow D' \cup (\boldsymbol{x}_c, \boldsymbol{y}_c)$;
    $GenNum \leftarrow GenNum - 1$;
return D' ;

Figure 29: MLSOL (main algorithm) pseudo code (Liu & Tsoumakas, 2020a)

Algorithm 3: GenerateInstance

input : seed instance: $(\boldsymbol{x}_s, \boldsymbol{y}_s)$, types of seed instance: T_s, reference instance: $(\boldsymbol{x}_r, \boldsymbol{y}_r)$, types of reference instance: T_r
output: synthetic instance: $(\boldsymbol{x}_c, \boldsymbol{y}_c)$

for $j \leftarrow 1$ **to** d **do**
  $x_{cj} \leftarrow x_{sj} + \texttt{Random}(0,1) * (x_{rj} - x_{sj})$; /* Random(0,1) generates a random value between 0 and 1 */
$d_s \leftarrow dist(x_c, x_s)$, $d_r \leftarrow dist(x_c, x_r)$; /* $dist$ return the distance between 2 instances */
$cd \leftarrow d_s/(d_s + d_r)$;
for $j \leftarrow 1$ **to** q **do**
  if $y_{sj} = y_{rj}$ **then**
    $y_{cj} \leftarrow y_{sj}$;
  else
    if $T_{sj} = MJ$ **then** /* ensure y_{sj} being minority class */
      $s \longleftrightarrow r$; /* swap indices of seed and reference instance */
      $cd \leftarrow 1 - cd$;
    switch T_{sj} **do**
      case SF **do** $\theta \leftarrow 0.5$; break ;
      case BD **do** $\theta \leftarrow 0.75$; break ;
      case RR **do** $\theta \leftarrow 1 + 1e - 5$; break ;
      case OL **do** $\theta \leftarrow 0 - 1e - 5$; break ;
    if $cd \leqslant \theta$ **then**
      $y_{cj} \leftarrow y_{sj}$;
    else
      $y_{cj} \leftarrow y_{rj}$;
return $(\boldsymbol{x}_t, \boldsymbol{y}_t)$;

Figure 30: MLSOL (GenerateInstance) pseudo code (Liu & Tsoumakas, 2020a)

Algorithm 2: InitTypes

input : The matrix storing proportion of kNNs with opposite class for each instance and each label: $\boldsymbol{C}$, number of nearest neighbour: k
output: types of instances $\boldsymbol{T}$

for $i \leftarrow 1$ **to** n **do** /* n is the number of instances */
  for $j \leftarrow 1$ **to** q **do** /* q is the number of labels */
    if $y_{ij} =$ *majority class* **then**
      $T_{ij} \leftarrow MJ$;
    else /* y_{ij} is the minority class */
      if $C_{ij} < 0.3$ **then** $T_{ij} \leftarrow SA$;
      else if $C_{ij} < 0.7$ **then** $T_{ij} \leftarrow BD$;
      else if $C_{ij} < 1$ **then** $T_{ij} \leftarrow RR$;
      else $T_{ij} \leftarrow OT$;
repeat/* re-examine RR type */
  for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** q **do**
      if $T_{ij} = RR$ **then**
        foreach $\boldsymbol{x}_m$ *in* $kNN(\boldsymbol{x}_i)$ **do**
          if $T_{mj} = SF$ *or* $T_{mj} = BD$ **then**
            $T_{ij} \leftarrow BD$;
            break ;
until *no change in* $\boldsymbol{T}$;
return $\boldsymbol{T}$;

Figure 31: MLSOL (InitTypes) pseudo code (Liu & Tsoumakas, 2020a)

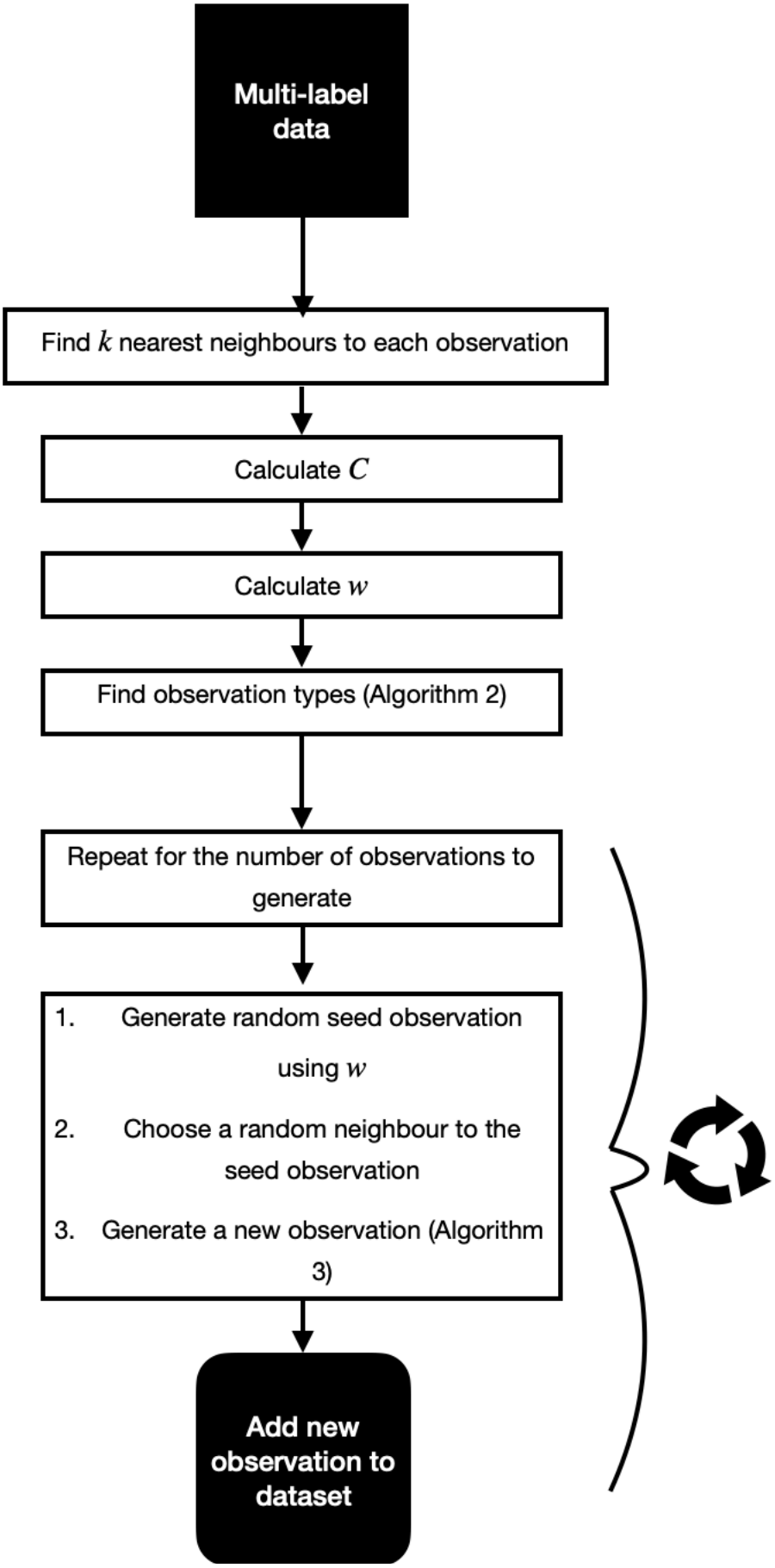


Figure 32: MLSOL flow-diagram

3.2.7 RHwRSMT

REMEDIAL-Hybridisation with synthetic instance generation (RHwRSMT) proposed by Charte et al. (2019) is an oversampling algorithm that combines two resampling algorithms we have already seen. REMEDIAL and MLSMOTE are used together. Firstly, REMEDIAL is used to decouple the majority and minority labels and reduce the presence of SCUMBLE in the dataset. Once REMEDIAL has been used to decouple the data, MLSMOTE is applied to the decoupled data.

As we have seen previously in Section 3.2.5, the first step for MLSMOTE is to place all observations that belong to minority labels in a bag. The decoupling procedure of REMEDIAL will make it easier for MLSMOTE to differentiate between minority and majority observations. Therefore, those observations placed in the minority bag will have label-sets better representing minority observations since the minority label-sets have been decoupled from the majority label-sets. Therefore, the minority observations selected by MLSMOTE and their neighbours could potentially have different label-sets that could better represent minority observations. This stands to benefit the MLSMOTE algorithm since the algorithm makes use of the minority observations to generate new synthetic observations. If the minority observations and their neighbours have label-sets better representing minority observations, the MLSMOTE algorithm could potentially be more effective at balancing the class distributions.

MLSMOTE produces synthetic label-sets for new observations. The set of labels is produced from the nearest neighbours to the observation being processed. The decoupling procedure performed by REMEDIAL before the minority observations are placed in a bag will influence the chosen observations and their neighbours. This could lead to a more balanced label distribution for the new synthetically generated observations.

A drawback of the algorithm highlighted in Charte et al., (2019) is that although REMEDIAL changes the label-sets of the decoupled observations, it does not change the location of the observations in terms of the feature set. A potential remedy to this problem would be a relocation of the decoupled observations once they have been split into minority and majority observations.

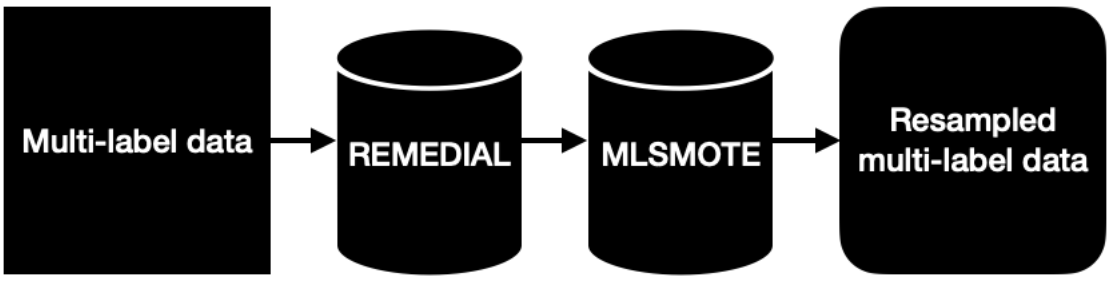


Figure 33: RHwRSMT flow-diagram

Chapter 4: Simulating multi-label data

4.1 Introduction

Many studies performed in multi-label literature use the benchmark datasets discussed in Chapter 1 to evaluate the performance of different multi-label techniques. Although the benchmark datasets provide a practical way of comparing the performance of multi-label techniques, it is not ideal for this thesis. The analysis of multi-label performance is limited to the specific attributes that these datasets have. Two datasets are rarely the same, which is especially true for multi-label data. Each multi-label dataset is unique and provides different challenges and characteristics to the algorithms.

It is reasonable to assume that the benchmark datasets do not fully encapsulate the scope of characteristics that multi-label data could have. Most of the benchmark datasets have very low cardinality and density, almost to the point where they are multi-class and not multi-label. In this thesis we look at addressing imbalance at changing levels of global density. Therefore, we require datasets at all levels of global density.

Domain bias is another stumbling block when making use of the benchmark datasets. The multi-label performance for two similar datasets from different domains could be completely different. A dataset from the text domain might yield completely different multi-label performance than a dataset from the biology domain, even though their characteristics might be similar. Therefore, comparing performance of resampling algorithms on these datasets would be naïve since the changes in performance might be due to the differences in domains, rather than the differences in the resampling algorithms.

An ethical problem with using the benchmark datasets is “cherry-picking” the datasets for which the technique works the best and leaving out those datasets on which the technique struggles. Cherry-picking becomes a problem since we need to know under which circumstances techniques perform well and under which circumstances they do not perform well. If we only select the datasets that support our arguments, this will defeat the purpose of the analysis.

This thesis proposes an impartial approach in studying the effectiveness of resampling algorithms for multi-label classification using simulated data. Simulating data allows control over specific characteristics of the experimental conditions. The simulation mechanism allows us to specify the local label densities directly and consequently indirectly control which labels are majorities and minorities. A diverse set of experimental conditions is created to put the array of resampling algorithms to the test. Since we are not proposing our own algorithm, our thesis is purely observational. We want to identify the scenarios in which the resampling algorithms excel, as well as the scenarios in which the resampling algorithms should be avoided.

4.2 MLDatagen

Researchers simulate data to avoid some of the pitfalls mentioned in Section 4.1. In the single-label paradigm it is much easier to generate simulated data than in the multi-label paradigm. In the multi-label paradigm we need to incorporate many characteristics into the data. Some of the characteristics that need to be taken into consideration are the number of observations, number of variables, number of labels, the correlation between labels and the correlation between variables.

MLDatagen was proposed by Tomás et al. (2014) and is one of the very few proposals for the simulation of multi-label data. MLDatagen provides an online multi-label data simulation framework that allows the user to control certain characteristics of the multi-label data being simulated. Two strategies are implemented in MLDatagen: hypercubes and hyperspheres. These randomly generated geometric shapes populate the observations and their associated labels. MLDatagen gives the user control over the following characteristics of the data:

- Strategy: A choice between hypercubes and hyperspheres
- Number of relevant features
- Number of irrelevant features
- Number of redundant features
- Number of labels
- Number of observations
- Noise parameter (0 to 1)
- Maximum and minimum radius of the hypersphere/hypercube.

MLDatagen is an extremely useful tool in the context of multi-label research and has been used successfully in many multi-label studies. However, for the purpose of this thesis, MLDatagen lacks the ability to directly control the global density and the local label densities, which is an aspect of data simulation that is critical to this thesis since we wish to study the efficacy of resampling algorithms at increasing levels of global density.

A comprehensive discussion of the MLDatagen simulation mechanism falls outside the scope of this thesis, for a more thorough discussion of the simulation mechanism refer to Tomás et al., (2014).

4.3 Simulating data

The simulation mechanism used in this thesis is from the proposal of Sandrock & Steel (2017). This proposal gives specific control over local label densities. Having control over local label densities also indirectly gives control over the global density of the dataset since the global density of a multi-label dataset is the average of all the local label densities. Control over the local label densities is critical to the outcome of our research since we are interested in investigating the efficacy of the resampling algorithms at divergent levels of global density. Having control over the local label densities also gives the ability to create imbalance through diversity in local label densities.

A high-level discussion of the data simulation is explored below. A more in-depth discussion of the simulation mechanism can be found in Sandrock & Steel (2017).

The following parameters are at our disposal to generate simulated data:

- N → The number of observations.
- K →The number of labels.
- p → The number of relevant predictor variables.
- $pnoise$ → The number of irrelevant predictor variables.
- $\underline{A}$ → A matrix that controls the global and local relevancies of variables for labels. A variable is locally relevant for a label if it aids in predicting whether that label is present or not, irrespective of its relevance for any other labels. If a variable is globally relevant, it is relevant for several or all of the labels.
- ρ → Controls the correlation between labels.
- $Signal$ → Strength of dependence between the predictor variables and the labels. It represents how much information is available in the input variables to predict the labels.
- $pvec$ → Vector of label densities.
- $Sigmax$ → Allows control over the correlation between predictor variables.

The experimental design (Chapter 6) outlines the parameters that were chosen to simulate multi-label data for the empirical analysis. Section 4.4 provides additional information and examples relating to how the multi-label data is simulated and what a typical simulated multi-label dataset would look like using the simulation mechanism mentioned in Section 4.3.

In addition to the parameters specified above, the desired global density ξ can also be specified. Due to the nature of the simulations, the global density will not exactly be equal to ξ, but will be very close. The desired level of global density is achieved through the $pvec$ defined above. The local label densities assigned to the $pvec$ are drawn from distributions defined in Section 4.4 and ξ is equal to the average of the local label densities.

4.4 Examples

4.4.1 Special case, ξ close to 0

The “special case” is a scenario where we replicate a dataset with a very sparse data matrix. The global density in this case will be very small. The purpose of the special case is to create a very extreme scenario to test the resampling algorithms. The extreme case will create a large polarity between the local label densities of the minority labels and the majority labels. The minority labels will have extremely small local label densities and the majority labels will have relatively larger local label densities. The global density of a multi-label dataset is the average of the local label densities, therefore the global density for the special case will be very small.

The probability density functions (pdf) from which the minority and majority label densities are randomly drawn are observed below in Figure 34. The local densities for both the minority and majority labels will be very small to create a sparse data matrix. The minority classes will have a density, a random observation drawn from the pdf shown on the left-hand side in the figure below. This will create an extreme case with very small local label densities. The majority classes will have a local label density, a random observation drawn from the pdf on the right. The majority classes will, in a relative sense, have much larger densities than the minority classes. However, in a real sense, these label densities are still very small.

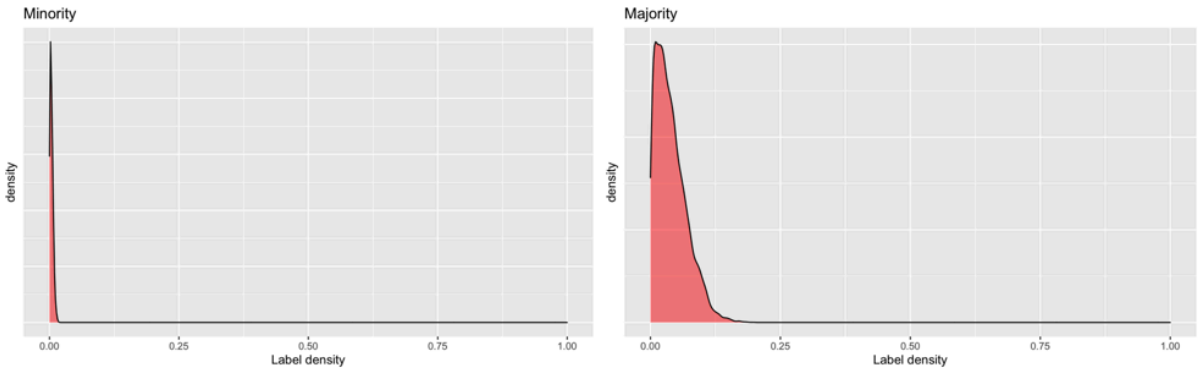


Figure 34: Minority and Majority pdf's for "special case"

Data will be simulated to illustrate the functionality of the simulations. Two datasets with 20 labels were simulated for the special case. The local label densities from the two simulations are observed in Figure 35. We observe that there are many labels with very small local label densities, relating to the labels simulated for the minority classes. There are also labels with comparatively larger local label densities, relating to the labels simulated from the majority classes. Comparing the two simulations allows us to observe how the simulated data might change from one iteration to the next. The simulated datasets will be homogeneous yet different for each iteration, as observed below.

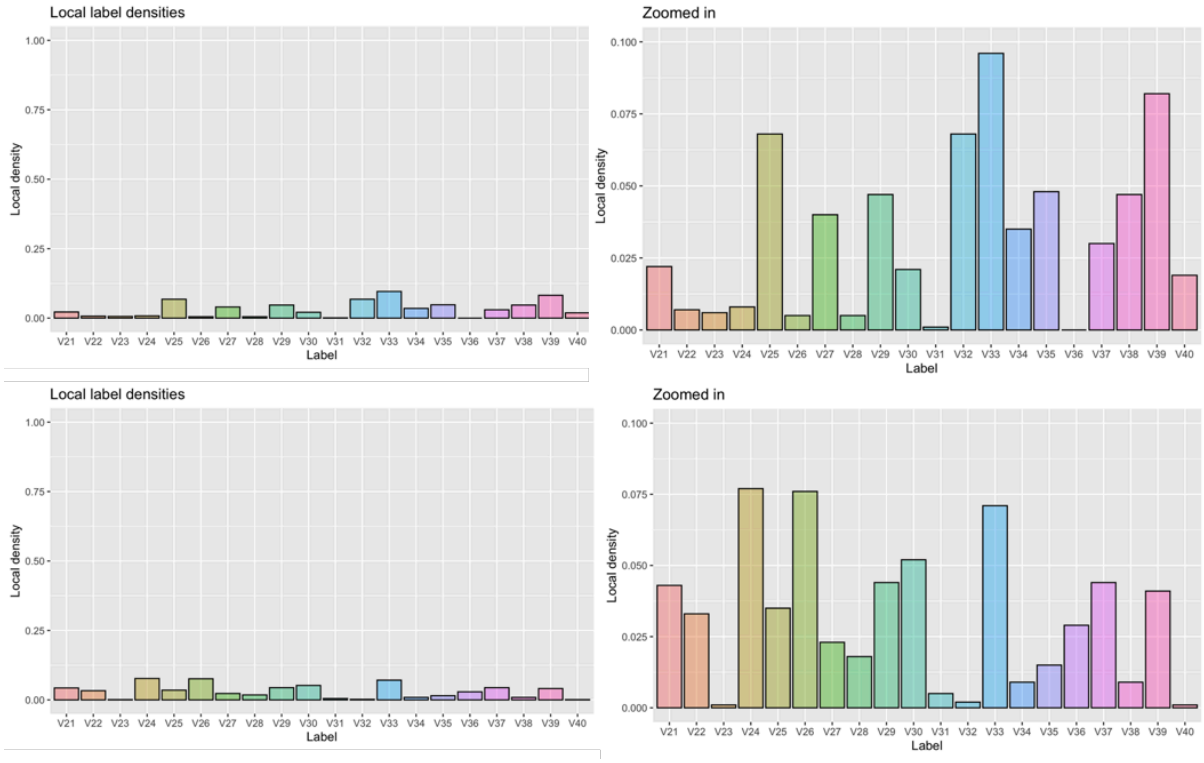


Figure 35: Local label densities for "special case"

4.4.2 $\xi \in \{0.1, 0.2, \ldots, 0.5\}$

In the repository for multi-label data found at Moyano (2021), there are no datasets with global density larger than 0.5, therefore we simulate data for values of ξ ranging from 0.1 to 0.5. Local label density vectors are created in the same way as above. The probability density functions from which the local label densities are randomly drawn will be explored below. In this case, we will create imbalance through the disparity of the minority and majority local label densities. Although, the local label densities are now larger, the data will be simulated in such a way that the local label densities of the minority labels are much smaller than the local label densities of the majority labels. As ξ increases, the global density of the datasets also increases, where $\xi \approx global\ density$. The minority and majority labels will be drawn from the pdfs illustrated below to create a disparity in local label density between the minority labels and the majority labels.

In Figure 36 below, we observe the pdf's relating to the local label densities of the minority and majority labels, where each colour corresponds to a larger level of global density (ξ). We observe the pdfs from which the minority class local label densities will be drawn on the left-hand side. On the right-hand side, we observe the pdf's from which the majority class local label densities will be drawn. A disparity in local label density between the minority and majority classes is created by randomly sampling the label densities from the different pdfs. The disparity in label density between the minority and majority local label density is observed in Figure 36.

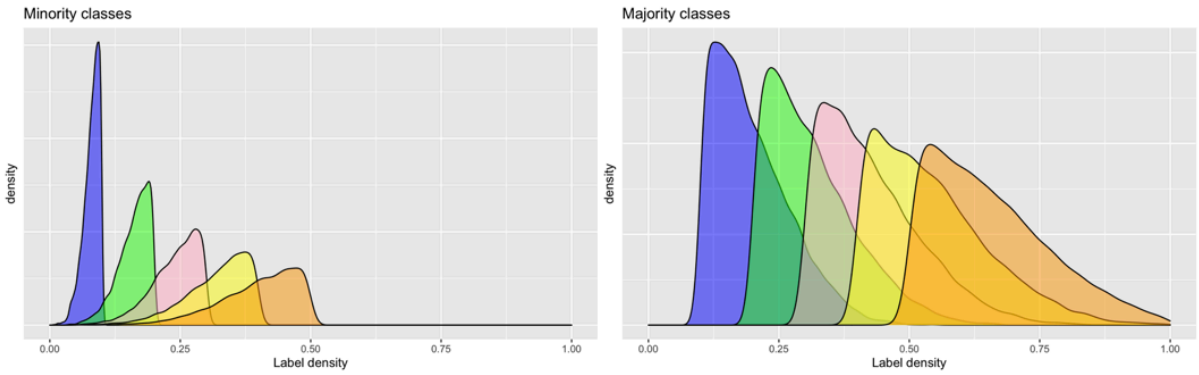


Figure 36: Minority and Majority pdf's

In Figure 37 below, we observe the local label densities for datasets simulated with $\xi = 0.1, 0.2, 0.3, 0.4, 0.5$, respectively, with $K = 20$ labels. These simulated datasets are random and will change from one simulation to the next. However, as mentioned previously: The signal, the number of relevant and irrelevant predictors, ρ and A matrices all remain constant to ensure that even though the datasets may look different in terms of local label densities, the domain effect does not play a role. The height of the bars represent the densities and the x-axis represents simulated labels as V21, V22, …, V40.

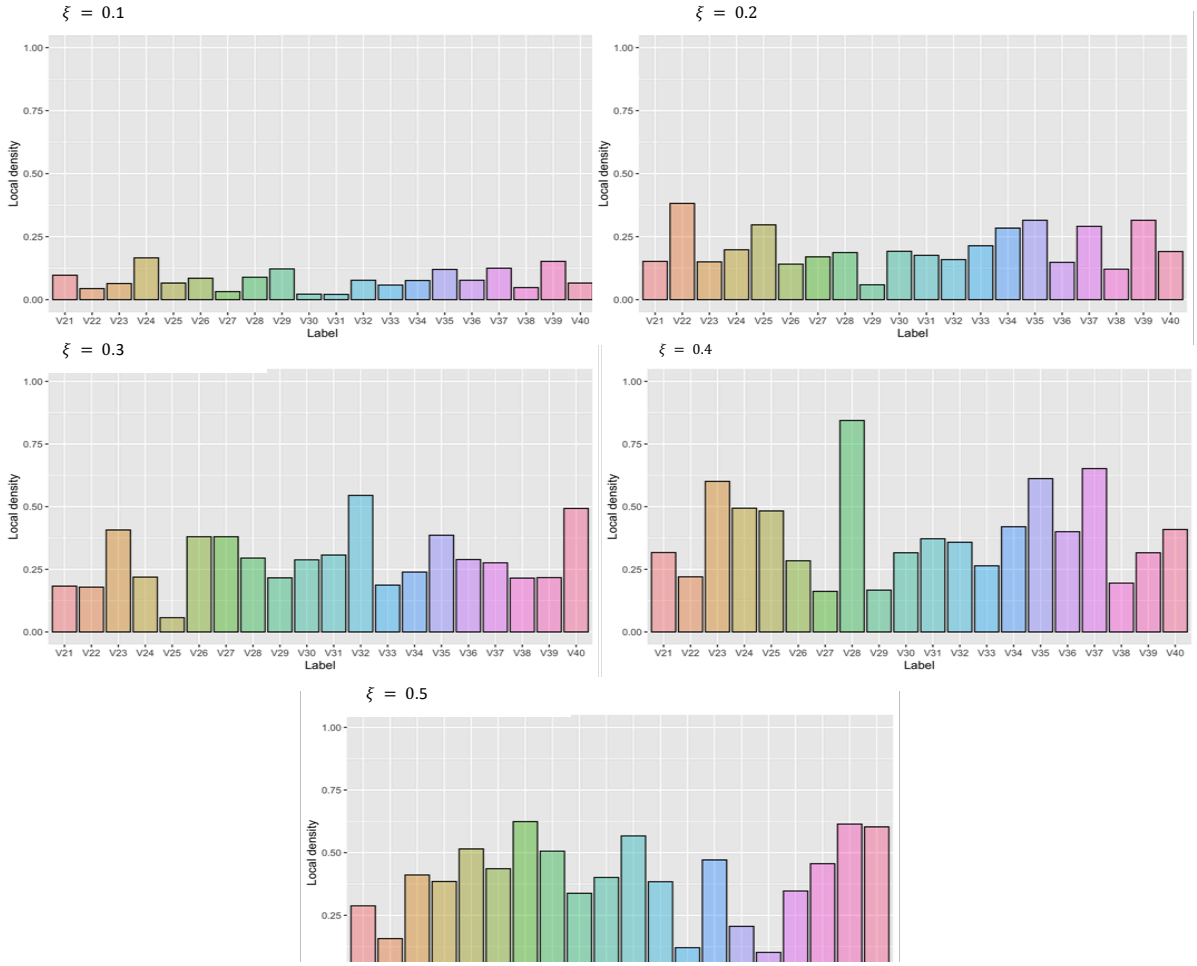


Figure 37: Local label densities

The simulated data displayed above represents how data used in the experimental design will look. The datasets change from one iteration/simulation to the next. The differences in local label density between labels is due to the local label densities being randomly sampled from the majority and minority pdf's mentioned above. The purpose is to have many datasets representing a similar situation in terms of global density and imbalance.

In Figure 37, we observe that although there is a difference in label density between the majority classes and the minority classes, the level of imbalance is not extreme at larger densities. Creating extreme imbalance at larger levels of global density is difficult since the global density is the average of the local label densities, therefore even though some of the labels will have very small local label densities, the average of the local label densities still needs to be large enough to obtain a larger global density. The scenario we have created represents imbalanced data, however the imbalance at larger global densities is not as extreme as it is observed for the "special case". An avenue of future research might be to simulate datasets that have extreme imbalance at large global densities.

From one data simulation to the next, the level of imbalance measured by MeanIR remains consistent. The largest deviations in MeanIR from one simulation to the next is observed for the "special case". As an example, for ten simulated datasets with $\xi = 0$, the MeanIR was $(10.20,\ 8.40\ 6.93, 15.71,\ 8.01, 10.01,\ 7.37,\ 8.91,\ 4.95\ 10.51)$. Therefore, for the special case we observed large levels of MeanIR, due to the small local label densities of both the minority and majority classes. At larger levels of global density, we observe that the MeanIR of the datasets are smaller. For ten simulated datasets with $\xi = 0.1$, the following MeanIR's were observed: $(3.14, 3.01, 2.16, 4.51, 2.74, 2.28, 2.87, 2.87, 2.36, 2.21)$. Therefore, the MeanIR's are now more consistent from one simulation to the next, but are not as large as we observed for the "special case". For a global density of $\xi = 0.2$, the following MeanIR's were observed: (2.19, 3.07, 1.98, 2.24, 3.03, 1.75, 2.06, 3.13, 3.48, 2.82). A similar trend is observed for datasets with global density larger than $\xi = 0.2$, where the MeanIR tends to be between 2 and 3 and remains consistent from one simulation to the next.

The simulation mechanism of Sandrock and Steel (2017) does provide control over the correlation between labels in general, but not explicit control over the correlation between specific labels. Without explicit control over the dependence between labels, creating SCUMBLE in the datasets is not feasible. Therefore, SCUMBLE in the simulated datasets is not large.

4.4.3 Example dataset

As an example, a dataset will be simulated with ten labels and a global density of $\xi = 0.2$. The figures below visualise this dataset in terms of its label concurrence plots, local label density and labels per observation. The global density for the simulated dataset is the average of the local label densities. The global density will not be precisely equal to ξ in each simulation but should be approximately equal to ξ. For this example, the local densities were: 0.168, 0.143, 0.174, 0.106, 0.301, 0.201, 0.179, 0.146, 0.388, 0.197. The average of these densities leads to $\xi = 0.2003 \approx 0.2$. The label concurrence plot shows that most labels are somewhat correlated and look reasonable for multi-label data. The histogram with the number of labels per observation has an exponential shape.

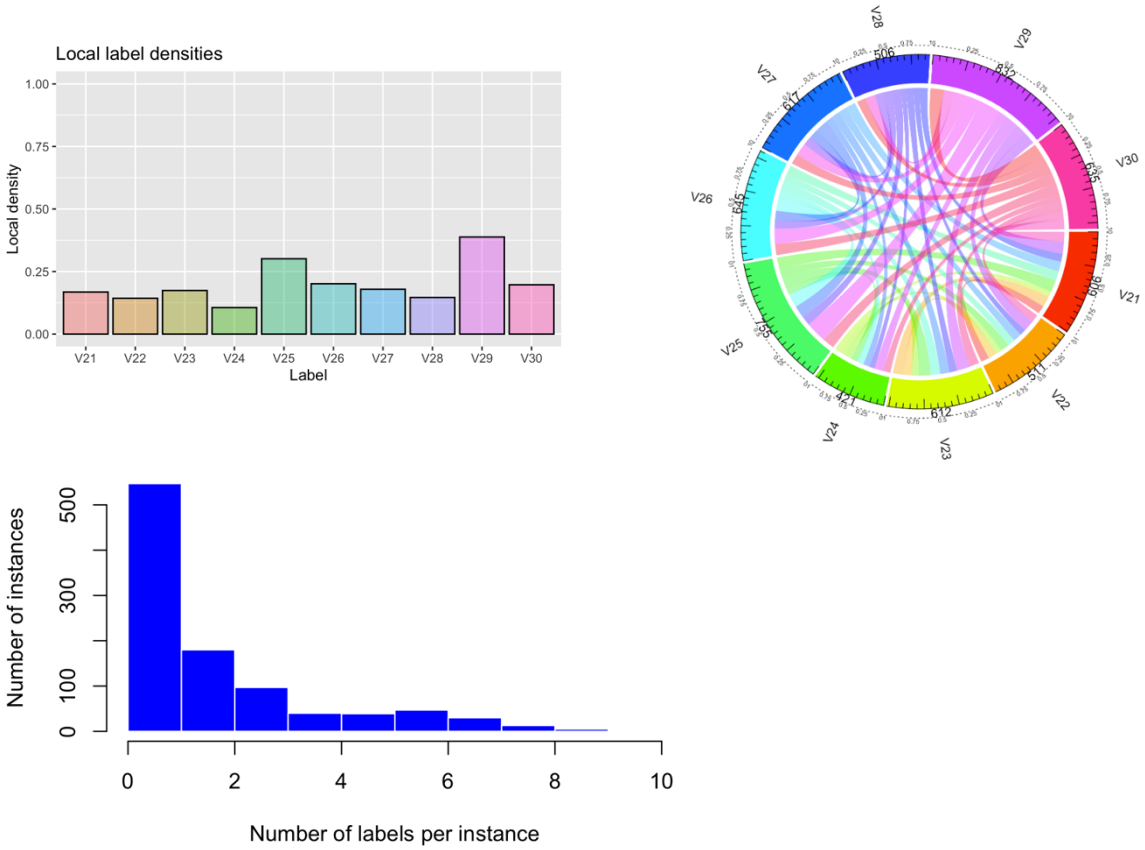


Figure 38: Example simulated dataset

Chapter 5: Multi-label performance measures

5.1 Introduction

Evaluation metrics are used to quantify the performance of supervised learning algorithms. In binary and multi-class classification we can rely on a few key metrics that allow us to quantify classification performance. Metrics such as Accuracy, F-measure and Area Under Curve (AUC) are often used to evaluate the performance of classification algorithms. However, multi-label performance is more complicated since each observation can now be associated with more than one class. The multi-label paradigm therefore poses a significant challenge to the status quo and requires alternative approaches.

The multi-label nature of the data leads to some questions that need to be answered: Firstly, is it better to predict a label as present that is not present, or is it better not to predict a label that is present? Secondly, is wrongly predicting three labels in one observation the same as wrongly predicting one label in three observations? Thirdly, how do we differentiate the performance between minority and majority labels? Lastly, what impact does global density and cardinality have on multi-label performance measures? These questions should be addressed when multi-label evaluation metrics are designed.

For comparative studies like this one, evaluation metrics focusing on various aspects of multi-label performance are chosen. It becomes essential to include different and contrasting evaluation measures, ensuring that we fully encapsulate the performance of the models. If too few evaluation metrics are chosen, it might add bias to the analysis since we might come to false conclusions if some of the models only perform well on the chosen metrics and not on some of the other metrics not included in the study. Therefore, we should make use of a diverse group of evaluation metrics.

A metric such as Subset Accuracy is commonly used in binary and multi-class classification, but is not as common in the multi-label paradigm. Subset Accuracy is an extremely strict measure and is rarely a good way of measuring multi-label performance. Subset Accuracy requires the predicted label-set to be an exact match to the true label-set. It is difficult for MLC to predict all labels in the label-set correctly. Therefore, MLC techniques could do poorly on the Subset Accuracy and lead us to wrongly conclude that our model is doing poorly when in reality it could only be misclassifying a small number of the labels. A measure such as the Hamming loss is more common in the multi-label paradigm and can be seen as a relaxed version of the Subset Accuracy since it measures the proportion of wrongly predicted labels.

Evaluation metrics can be split into two groups—those metrics that are bipartition-based and those that are ranking-based - see Tsoumakas & Katakis (2007).

5.1.1 Bipartition-based

Bipartition-based metrics compare the predicted present labels with the actual present labels. Therefore, we are trying to determine how the labels we predict match the actual observed labels. Bipartition-based metrics can further be split into example-based and label-based metrics:

Example-based: These metrics average the difference between the actual and predicted label-sets over all the observations in the dataset.

Label-based: Label-based metrics can further be split up into micro and macro averaged metrics.

- Macro-averaged metrics calculate any binary evaluation metric on each label separately and then averages these K metrics to come to the final macro-averaged metric, therefore treating each of the K labels equally.
- Micro-averaged metrics aggregate the contribution of all the labels and calculates the average metric of these aggregated values. Therefore, the classes are not treated equally.

5.1.2 Ranking-based

Ranking-based metrics are calculated on the label rankings and not from the binary predictions. These metrics compare the ranking of the labels with the true label-sets.

5.2 Notation

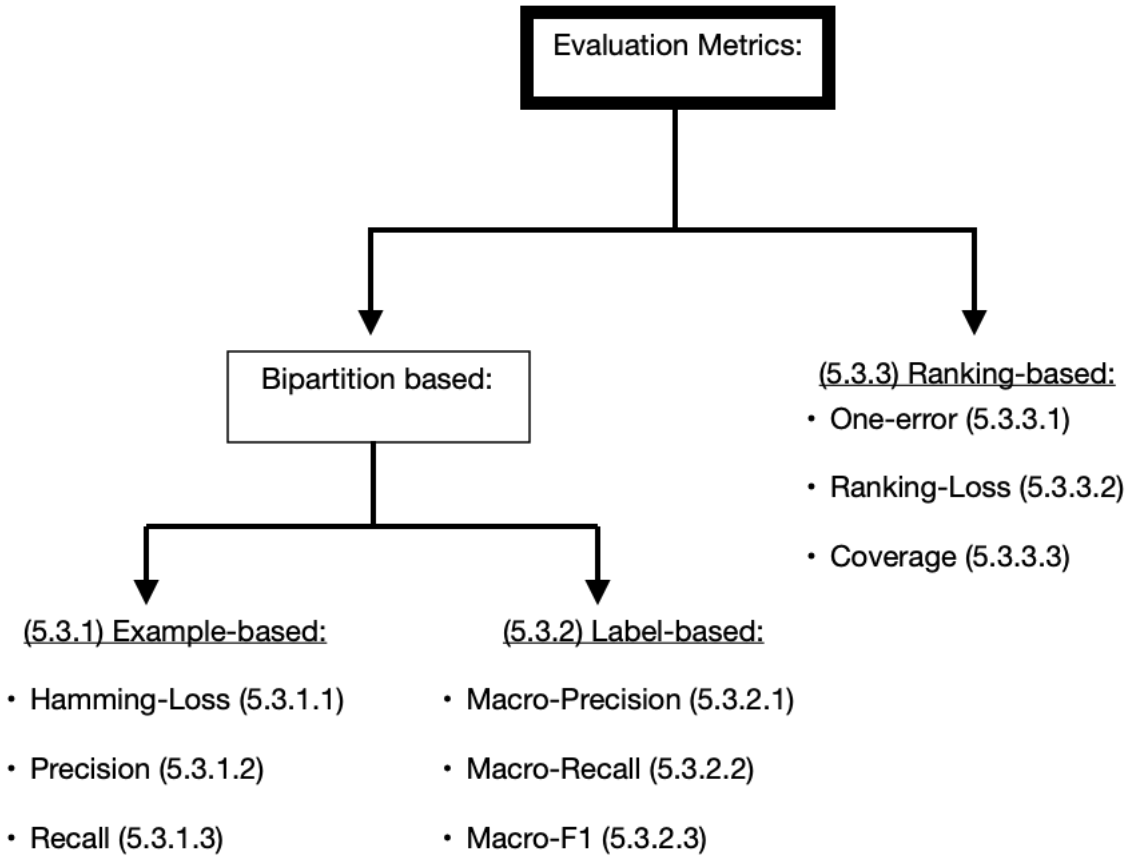
Classifiers can make different types of predictions and mistakes. These can be categorised into different groups:

- True Positives (TP): A label that is present is predicted as present,
- True Negative (TN): A label that is not present is predicted as not being present,
- False Positive (FP): A label that is not present is incorrectly predicted as present,
- False Negative (FN): A label that is present is predicted as not present,

where TP, TN, FP and FN represent the totals for the entire dataset. These can also be calculated for individual labels, where $TP_k, TN_k, FP_k, FN_k \; \forall \; k \in \{1,2, \dots, K\}$ represent the totals for label k.

Let the dataset D have N observations and K labels $Y_k \; \forall \; k \in \{1,2, \dots, K\}$. The matrix of predictor variables is $X \in \mathbb{R}^{N \times P}$ and the label matrix is $Y \in \{0,1\}^{N \times K}$. $f(x_i)$ denotes the prediction of y_i, a vector of label predictions for the $i'th$ observation, where the true prediction is $\mathcal{Y}_i$. $f_k(x_i)$ denotes the prediction for the $k'th$ label of the $i'th$ observation y_{ik}, where the true prediction is $\mathcal{Y}_{ik}$. Let $r_i \; \forall \; i \in \{1,2, \dots, N\}$ be the vector of label rankings for observation i.

5.3 Evaluation metrics



5.3.1 Example-based metrics

As mentioned in Section 5.2.1, the example-based metrics calculate the difference between the actual and predicted label-sets and average across all the observations in the dataset to calculate the final metrics. (Madjarov et al., 2012)

5.3.1.1 Hamming loss

Hamming loss represents the fraction of misclassified labels. A misclassified label can be a label not belonging to the observation being predicted or a label belonging to the observation not being predicted. Formally, Hamming loss is defined by

$$Hamming - Loss = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{K}|f(x_i) \triangle \mathcal{Y}_i|$$

where $f(x_i) \triangle \mathcal{Y}_i$ is the symmetric difference between $f(x_i)$, the prediction and $\mathcal{Y}_i$, the ground truth. Performance increases as Hamming loss decreases. Therefore, we would want the Hamming loss to be as small as possible, where Hamming loss is measured by a score between 0 and 1.

5.3.1.2 Precision

Precision is the proportion of predicted present labels that are in fact present. It is the proportion of labels that we predict as positive, that were positive. Therefore, it is the proportion of positive predictions that were true positives (TP). Formally, Precision is defined by

$$Precision = \frac{1}{N}\sum_{i=1}^{N}\frac{|f(x_i) \cap \mathcal{Y}_i|}{|\mathcal{Y}_i|} = \frac{TP}{FP + TP}$$

Performance increases as Precision increases. Therefore, we would want the Precision to be as large as possible, where Precision is measured by a score between 0 and 1.

5.3.1.3 Recall

Recall is the proportion of true positives among all true examples. It is the proportion of observations belonging to a class that has been predicted as belonging to that class. Therefore, it is the proportion of relevant labels that have been selected. Formally, Recall is defined by

$$Recall = \frac{1}{N}\sum_{i=1}^{N}\frac{|f(x_i) \cap \mathcal{Y}_i|}{|f(x_i)|} = \frac{TP}{FN + TP} \in [0,1]$$

Performance increases as Recall increases. Therefore, we would want the Recall to be as large as possible, where Recall is measured by a score between 0 and 1.

5.3.2 Label-based metrics

The label-based bipartition evaluation metrics that follow were all chosen to be macro-based. Macro-based means that the metric is calculated for each label separately as a binary class. The metrics are then averaged per label to obtain the final evaluation metric. Each label receives the same weight in the final metric, regardless of how many positive examples are in the class. These macro-based metrics will penalise models that perform poorly on the minority classes and only perform well on the majority classes, making these metrics robust against imbalanced class distributions. (Madjarov, 2012a)

5.3.2.1 Macro-Precision

Macro-Precision is the average Precision per label. The Precision is calculated for each label as a binary class. These scores are then averaged to obtain the Macro-Precision. Formally, Macro-Precision is defined by

$$\Rightarrow Macro - Precision = \frac{1}{K}\sum_{k=1}^{K}\frac{TP_k}{FP_k + TP_k} \in [0,1]$$

Performance increases as Macro-Precision increases. Therefore, we would want Macro-Precision to be as large as possible, where Macro-Precision is measured by a score between 0 and 1. (Madjarov, 2012a)

5.3.2.2 Macro-Recall

Macro-Recall is the average Recall per label. The Recall is calculated for each label as a binary class. These scores are then averaged to obtain the final Macro-Recall. Formally, Macro-Recall is defined by

$$Macro - Recall = \frac{1}{K}\sum_{k=1}^{K}\frac{TP_k}{TP_k + FN_k}$$

Performance increases as Macro-Recall increases. Therefore, we would want Macro-Recall to be as large as possible, where Macro-Recall is measured by a score between 0 and 1. (Madjarov, 2012a)

5.3.2.3 Macro-F1

Macro-F1 is the average F1 score per label. The F1-score is calculated for each label as a binary class. These scores are then averaged to obtain the final Macro-F1 score. The F1-score is the harmonic mean of the Recall and Precision, thus taking both metrics into account. The F1-score will be optimal when a classifier finds a balance between Precision and Recall and will be penalised if it only does well on one of the two metrics. The Macro-F1 score is the standard

way to evaluate the efficacy of resampling algorithms in the literature. If the resampling algorithms can improve the Macro-F1 score, it is reasonable to assume that they have successfully addressed imbalance and led to better classification of minority observations. Formally, Macro-F1 is defined by:

$$Macro - F1 = \frac{1}{K}\sum_{k=1}^{K}\frac{2.P_k \,.\, R_k}{P_k + R_k}$$

where P_k is the Precision for the $k'th$ label and R_k is the Recall for the $k'th$ label.

Performance increases as Macro-F1 increases. Therefore, we would want Macro-F1 to be as large as possible, where Macro-F1 is measured by a score between 0 and 1 (Madjarov, 2012a).

5.3.3 Ranking-based metrics

As mentioned in Section 5.2.1 the ranking-based metrics are calculated on the label rankings $r_i \; \forall \; i \in \{1,2,\dots,N\} \; \& \; k \in \{1,2,\dots,K\}$. Therefore, we compare the predicted label rankings with the ground truth label-sets. These (and other) ranking-based metrics are discussed in detail in (Tsoumakas et al., 2009).

5.3.3.1 One-Error

One-Error measures how often the top-ranked label is not included in the set of relevant labels. Formally, One-Error is defined by:

$$One - Error = \frac{1}{N}\sum_{i=1}^{N}\delta\left(argmin_{r_i}(Y)\right)$$

where $\delta(Y) = 1 \; if \; Y \notin \mathcal{Y}_i \; and \; 0 \; otherwise.$

Performance increases as One-Error decreases. Therefore, we would want One-Error to be as small as possible, where One-Error is measured by a score between 0 and 1. (Tsoumakus et al., 2009)

5.3.3.2 Ranking-Loss

Ranking-Loss measures the average percentage of label pairs that are reversely ordered for the example. Formally, Ranking-Loss is defined by:

$$Ranking - Loss = \frac{1}{N}\sum_{i=1}^{N}\frac{|D_i|}{|\mathcal{Y}_i||\bar{\mathcal{Y}}_i|}$$

$D_i = \{(Y_a, Y_b) : r_i(Y_a) > r_i(Y_b), (Y_a, Y_b) \in \mathcal{Y}_i \,.\, \bar{\mathcal{Y}}_i\}$ and $\bar{\mathcal{Y}}$ is the complementary set of $\mathcal{Y}$.

Performance increases as Ranking-Loss decreases. Therefore, we would want to minimise the Ranking-Loss, where Ranking-Loss is measured by a score between 0 and 1 (Tsoumakas et al., 2009).

5.3.3.3 Coverage

Coverage measures how far we need to go down the list of ranked labels to cover all the relevant labels of the observation. The smallest value that the Coverage can be is the cardinality of the dataset. Formally, Coverage is defined by:

$$Coverage = \frac{1}{N}\sum_{i=1}^{N} max_{r_i}(Y) - 1$$

where $Y \in Y_i$.

Performance increases as the Coverage decreases. Therefore, we would want to minimise the Coverage. (Tsoumakus et al., 2009). Coverage is not measured with a score between 0 and 1 like all of the other metrics, but is rather given in terms of the cardinality of the dataset and will therefore be visualised separately from the other ranking-based metrics in Chapter 7.

Chapter 6: Experimental design

6.1 Introduction

1. Are resampling algorithms effective at improving MLC performance at changing levels of global density?
2. Is there one form of resampling preferred to all others?

So far, we have explored MLC models, resampling algorithms, simulated data and evaluation metrics. This section intends to explain how all these elements were integrated to address the research goals above.

When designing the experiments, there were some stumbling blocks and considerations that needed to be made.

6.1.1 Resampling algorithms

The first major stumbling block encountered in this thesis was that only one of the chosen resampling algorithms is available in R or Python. Therefore, all of the resampling algorithms explored in Chapter 3, with the exception of REMEDIAL had to be programmed from scratch. These algorithms are large and elaborate and took a considerable amount of time and effort to programme correctly. The process of coding these algorithms formed a large portion of the work performed in this thesis. Resampling algorithms for multi-label data are mostly available on the Java platform through the Mulan library in the Weka environment (Tsoumakas et al., 2011). Developing these algorithms for widespread use in R and Python could lead to an increase in the popularity of the resampling algorithms.

6.1.2 Computational efficiency

In our experiments we found that computational efficiency was an important factor to be considered. Our experiments involved fitting many models on a large number of datasets along with running a large number of resampling algorithms. Given the nature of our experiments, this became a significant consideration in all decisions made. To ensure that experiments did not run for weeks at a time, we selected BR, CC, MLkNN, ECC, EBR and ECC as our MLC models since these models are efficient and would work under a diverse set of conditions. Section 8.4 provides a breakdown of the computational efficiency of the various resampling algorithms used.

6.1.3 Stability of models

Models were being fitted in large loops. Therefore, we needed the models to be stable and work under a diverse set of conditions created by the simulated data. If a model did not work for whatever reason, the experiments had to be stopped and restarted from the beginning. The simulated data ranged from very sparse data to data with high global density. We required the MLC models to be stable under all these conditions. The same applied to

resampling algorithms. Therefore, the experiments were designed to be as stable and efficient as possible by selecting simple MLC models and adding a cleaning step for the data to avoid anomalies in the data. The experiments can be split up into six steps and are explained in detail in the following section.

6.2 Experiments

It is important to note that for each combination of Y and ξ, ten datasets were simulated and the results across the ten datasets were averaged to obtain the final results. The evaluation metrics calculated at the end of each iteration of the experiments are averaged. A flow diagram can be found in the following section, visually outlining this process.

6.2.1 Simulate data

The starting point for the experiments was to simulate the data using the simulation mechanism of Sandrock & Steel (2017) explained in Chapter 4. Data was simulated for $K = 5, 10$ and 20 labels and for global density $\xi = 0, 0.1, \ldots, 0.5$. The simulated datasets had $N = 1000$ observations, had 10 relevant features, 10 redundant features, the signal was 0.5 and correlation between labels $\rho = 0.25$. Each experiment consisted of one combination of K and ξ. Therefore, the first experiment had $K = 5$ and $\xi = 0$ for all ten iterations. This was done for all possible combinations of K and ξ. The results were aggregated accordingly, allowing us to compare the performance at different number of labels and changing global density.

The table below illustrates the different combinations of K and ξ that were used to simulate datasets.

Table 3: Combinations of global density and number of labels

Labels	$\xi = 0$	$\xi = 0.1$	$\xi = 0.2$	$\xi = 0.3$	$\xi = 0.4$	$\xi = 0.5$
K = 5	$K = 5$ & $\xi = 0$	$K = 5$ & $\xi = 0.1$	$K = 5$ & $\xi = 0.2$	$K = 5$ & $\xi = 0.3$	$K = 5$ & $\xi = 0.4$	$K = 5$ & $\xi = 0.5$
K = 10	$K = 10$ & $\xi = 0$	$K = 10$ & $\xi = 0.1$	$K = 10$ & $\xi = 0.2$	$K\ 10$ & $\xi = 0.3$	$K = 10$ & $\xi = 0.4$	$K = 10$ & $\xi = 0.5$
K = 20	$K = 20$ & $\xi = 0$	$K = 20$ & $\xi = 0.1$	$K = 20$ & $\xi = 0.2$	$K = 20$ & $\xi = 0.3$	$K = 20$ & $\xi = 0.4$	$K = 20$ & $\xi = 0.5$

6.2.2 Cleaning

A cleaning step was included to ensure the stability of the algorithms. All observations that contained no labels were removed from the analysis. On rare occasions, the simulation mechanism led to some of the labels having no observations belonging to it. These labels were also removed as a pre-processing step.

6.2.3 Test and training split

A random test and training split was applied. We preferred to keep this part of the experiment as simple as possible. Cross-validation was considered. However, this would have added even more complexity to an already complex set of experimental conditions. A 70/30 test and training split was used. Therefore, a random 70% of the observations were chosen as the training dataset. The remaining 30% of the data formed the test dataset.

It is important to note how the resampling algorithms apply to the data when a test and training split occurs. The resampling algorithms are only applied to the training data. The test data is not resampled and must remain untouched.

6.2.4 Resampling algorithms

The seven resampling algorithms as explored in Chapter 3 were applied to only the training data, creating seven new datasets. Each dataset represented the training data that was resampled using one of the resampling algorithms. Alongside the resampled datasets, we also kept the original training dataset, which was not resampled and allowed us to compare the performance on the resampled datasets to the performance on the initial training dataset. Therefore, for each simulated dataset, 7 new training datasets were created, one for each of the resampling techniques.

6.2.5 Fit MLC models

After step 4, we were left with eight training datasets—the seven datasets from the resampling algorithms and the original training dataset. The MLC models outlined in Chapter 2 (MLkNN, BR, CC, CLR, ECC and EBR) were then fit on these eight datasets. Therefore, all six MLC models were fit on each of the eight datasets.

6.2.6 Calculate evaluation metrics

This is the first time that the test dataset was used. The models trained in step 5 were used to make predictions for the unseen test dataset. The evaluation metrics outlined in Chapter 4 were then calculated for the predictions made from all the models. Evaluation metrics were calculated by comparing the predictions made with the ground truth in the unseen test data.

Since we are interested in the effect of the resampling algorithms on performance, the evaluation metrics for the five MLC models for each resampling algorithm is averaged to obtain evaluation metrics for each form of resampling and the original training data. These evaluation metrics are the final product for each iteration of the experiments. Therefore, these evaluation metrics are averaged over the ten iterations to obtain the results for datasets with the characteristics K and ξ.

The result for one experiment was an 8 by 11 matrix. The matrix contained the evaluation metrics as well as the average time taken by each resampling algorithm and the average change in the number of observations resulting from the resampling. Therefore, each

combination of K and ξ will have a resulting matrix (18 total matrices). Consider the table below, which contains the results for $K = 5$ and $\xi = 0$:

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.2581	0.0758	0.0705	0.0436	0.3880	0.0730	0.9240	0.6414	0.0146
LP_ROS	0.9487	71.4000	0.2577	0.0768	0.0716	0.0417	0.3961	0.0729	0.9233	0.6224	0.0144
ML_ROS	0.5320	40.0000	0.2580	0.0765	0.0713	0.0436	0.3867	0.0749	0.9237	0.6283	0.0146
MLSMOTE	0.0947	4.5000	0.2597	0.0732	0.0680	0.0338	0.3562	0.0596	0.9270	0.7003	0.0165
MLSOL	4.7485	210.0000	0.2573	0.0782	0.0735	0.0460	0.4509	0.0798	0.9210	0.5413	0.0138
RHwRSMT	0.1764	13.2000	0.2581	0.0757	0.0701	0.0739	0.3787	0.0730	0.9243	0.6555	0.0149
REMEDIAL	0.0857	8.7000	0.2588	0.0750	0.0696	0.0457	0.3587	0.0747	0.9250	0.6578	0.0151
MLTL	0.1764	4.4000	0.2575	0.0772	0.0718	0.0462	0.4077	0.0760	0.9230	0.5742	0.0143

Figure 39: Example of results matrix

Therefore, for each simulated dataset, a test and training split was performed. The seven resampling algorithms were applied to the training data, and five MLC models were trained on the resulting resampled training datasets, also the original training dataset. The five MLC models trained on the various resampled datasets were used to predict the test dataset, and evaluation metrics were then calculated for all the MLC models. These evaluation metrics were averaged over the five models for each form of resampling. Averaging the performance over the five MLC models is reasonable since they had very similar performance for the simulated data. Therefore, we were left with seven forms of resampling and the original dataset, which have associated evaluation metrics. This procedure is performed ten times. Thus, the evaluation metrics for the resampled datasets are averaged over the ten iterations to obtain the final matrix of evaluation metrics for each form of resampling.

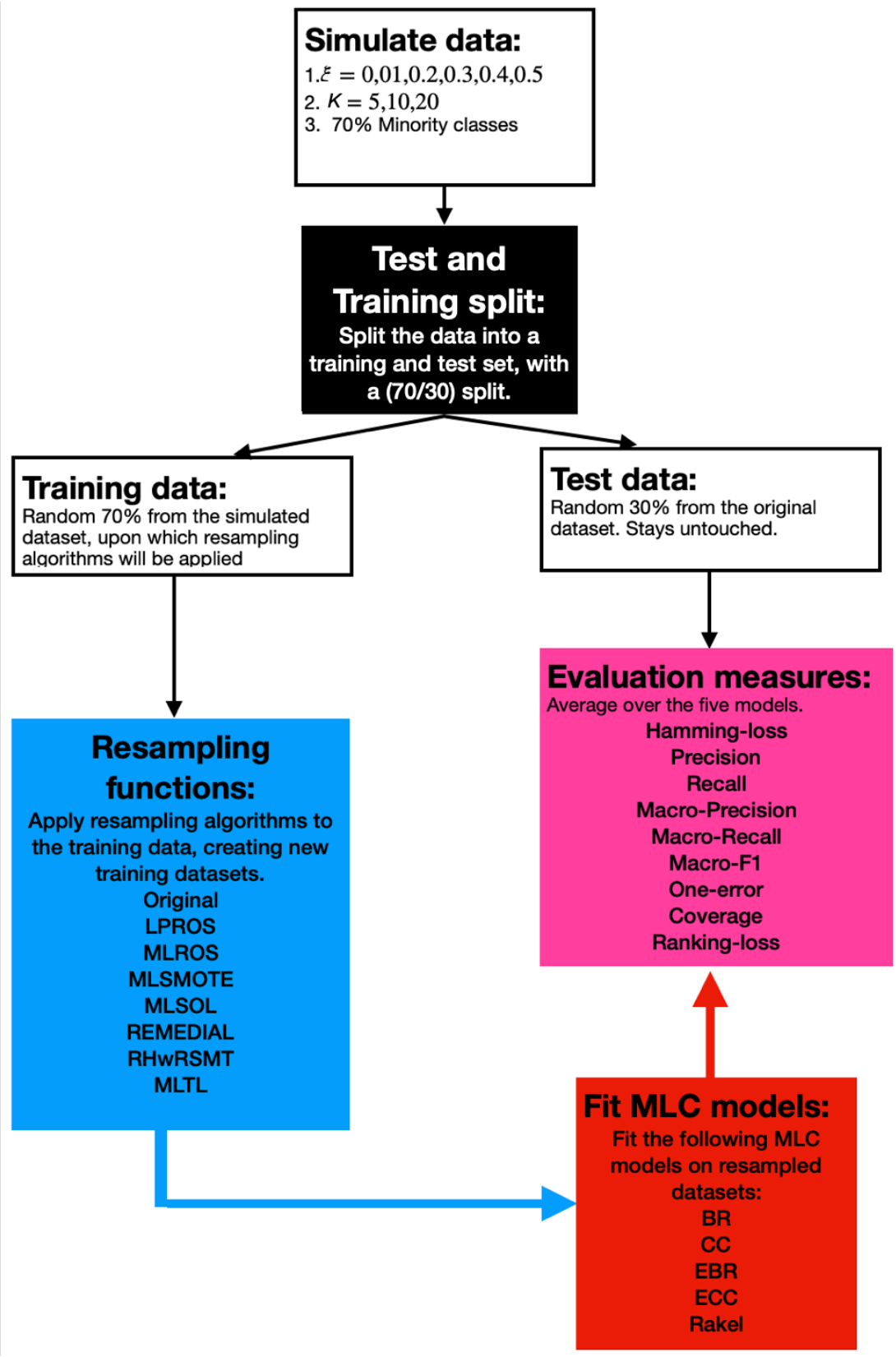


Figure 40: Experimental design

Chapter 7: Performance of resampling algorithms

7.1 Introduction

The first research goal was to determine whether resampling algorithms are effective at improving MLC performance at changing levels of global density. The seven resampling algorithms discussed in Chapter 3 were chosen because they were designed to improve MLC performance in highly imbalanced datasets. We want to evaluate this notion at differing levels of global density. When the global density increases, imbalance no longer manifests itself as high MeanIR but instead occurs due to a disparity in local label densities. It is reasonable to assume that some of the resampling algorithms could effectively improve MLC performance by balancing the class distributions at larger global densities. Therefore, in Chapter 7, we wish to determine if the resampling algorithms effectively improve MLC performance in general.

Chapter 8 extends from Chapter 7 and investigates whether there are resampling algorithms preferred in general and whether there are specific resampling algorithms that perform well under certain conditions and other algorithms that need to be avoided under certain conditions. Chapter 8 also provides recommendations on when to use and avoid the resampling algorithms, the computational efficiency of the resampling algorithms and the degree of the resampling imposed at different global densities.

To investigate the first research goal we used the experimental design found in Chapter 6. Data is simulated for $K = 5$, $K = 10$ and $K = 20$ labels at rising levels of global densities, where imbalance manifests itself in these datasets as a disparity in local label densities as seen in Chapter 4. For each combination of K and $\xi = 0, 0.1, 0.2, 0.3, 0.4, 0.5$, 10 datasets were simulated. The 7 resampling algorithms were applied to the resulting training datasets. 5 MLC models were fit on each of these resampled training datasets, as well as the original training dataset. The 5 MLC models were used to make predictions on the respective test datasets and the evaluation metrics were calculated accordingly. For each combination of K and ξ, the performance metrics were averaged over the 10 dataset and the 5 MLC models.

A reasonable assumption is that a resampling algorithm would be effective as a preprocessing tool if it improves MLC performance over the original dataset, therefore if the MLC models achieve better MLC performance when it uses the resampled datasets than when just the original data is used. Therefore, we wish to compare the performance of the MLC models that were fit using preprocessed datasets to MLC models that were fit using the original data. Throughout Chapter 7, the performance of the models on the original dataset is used as a reference to gauge if the preprocessing algorithms are a viable solution in the experimental conditions. The following sections are split by category of evaluation metrics, therefore a section for example-based, label-based and ranking-based metrics respectively.

7.2 Example-based

7.2.1 $K = 5$

In Figure 41 below, we observe line graphs representing the average performance of the MLC models for $K = 5$ labels across the three example-based evaluation metrics, where each figure represents a different level of global density. The resampling algorithms are observed on the horizontal-axis, and the evaluation metrics are observed on the vertical-axis. The dashed lines in each graph represent the performance of the models on the original data. Therefore, it serves as a point of reference for the other forms of resampling.

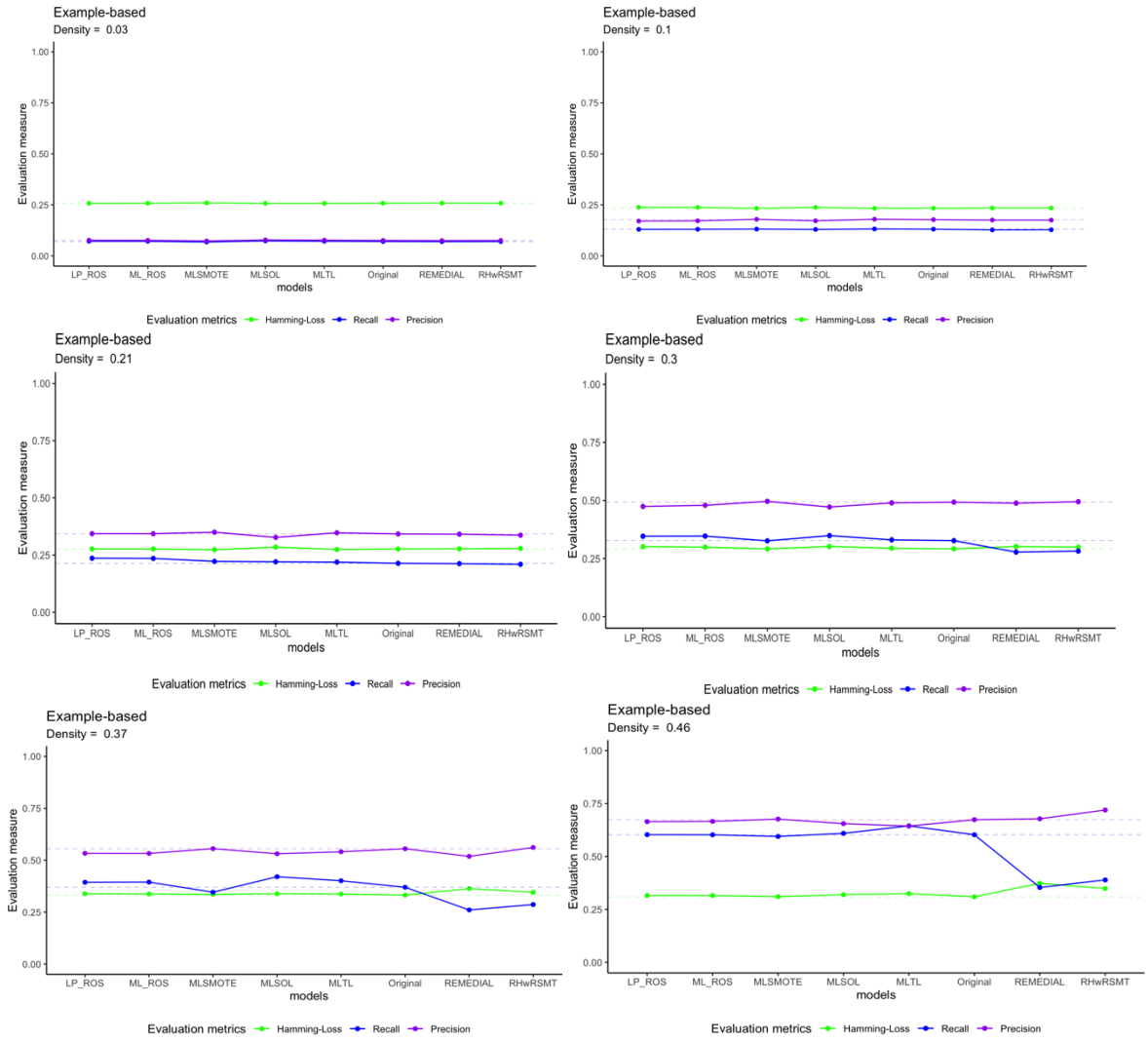


Figure 41: Example-based for $K = 5$

The first observation we make above is that the deviation in performance from the original data increases as the global density increases. In the top-left figure, we observe that all three lines are essentially flat, barely showing any deviation in performance from the dashed lines (original dataset). Only upon closer inspection, we observe how the resampling algorithms deviate in performance from the original dataset. However, as we move to the figures at

larger global densities, the deviations in performance away from the dashed lines become larger. The top left figure has a global density of 0.03, representing the "special case" discussed in Chapter 5. We expect the resampling algorithms to be effective in this scenario.

The visualisation below in Figure 42 allows us to zoom in on the "special case" mentioned above. The height of the bars represents the change in performance from the resampling algorithms relative to the original dataset. Therefore, the height of the bars represents the difference in performance between the original dataset and the resampled datasets. We observe that LP-ROS, ML-ROS, MLSOL and MLTL effectively improve MLC performance in terms of all three metrics. Therefore, these algorithms effectively reduce the Hamming loss and increase the Precision and Recall relative to what the MLC models were able to achieve on the original data. These changes in performance are extremely small, only leading to marginal improvements in performance. MLSMOTE performed poorly and led to a degradation in performance. Upon closer inspection we found that MLSMOTE only added a small number of observations, compared to the other algorithms.

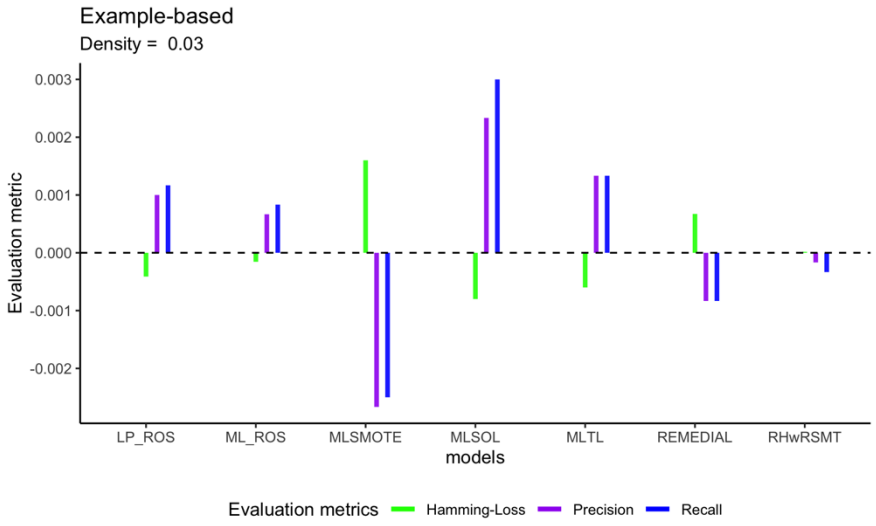


Figure 42: Example based change in performance $K = 5$, density = 0.03

When we move away from the "special case" to larger global densities, we observe a change in the resampling algorithms that are effective at improving performance. Figure 44 below shows the same visualisation as above, but for a global density of 0.1 and 0.21. We observe that MLSMOTE and MLTL effectively improve performance at a global density of 0.1 and 0.21. LP-ROS and ML-ROS are also effective at improving performance at a global density of 0.21. The changes in performance at these levels of global density are still small, as seen in the line graphs above, but larger than for the special case.

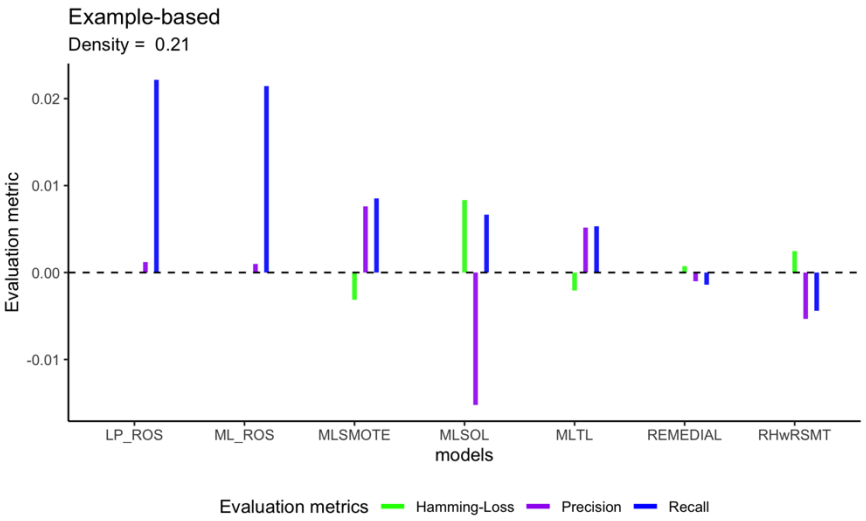


Figure 43: Example-based change in performance for $K = 5$, density = 0.21

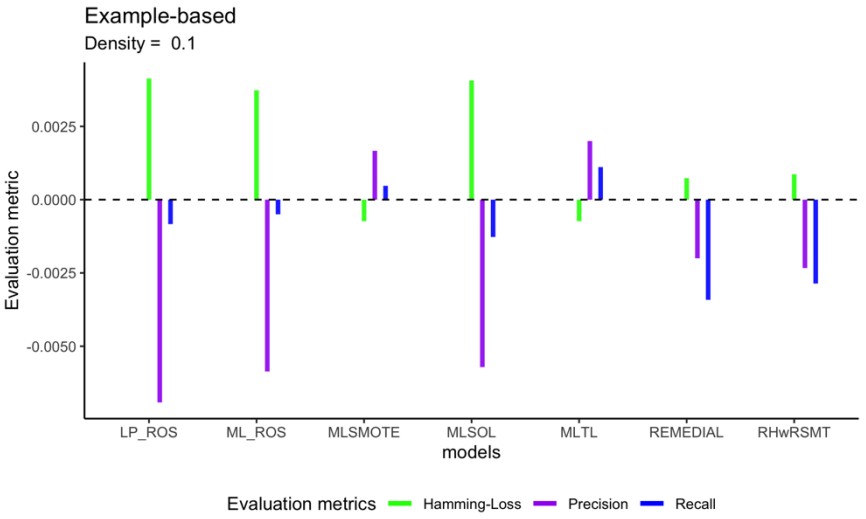


Figure 44: Example-based change in performance for $K = 5$, density = 0.1

Referring to Figure 41, when the global density becomes larger than 0.3, we observe that the changes in performance are much larger. However, most of the algorithms are only able to improve Recall and often lead to a reduction in performance in terms of Hamming loss and Precision. REMEDIAL and RHWrSMT particularly start to struggle when the global density becomes larger.

7.2.2 $K = 10$

In Figure 45, we observe line graphs representing the average performance of the MLC models for $K = 10$ labels across the three example-based evaluation metrics, where each plot represents a different level of global density.

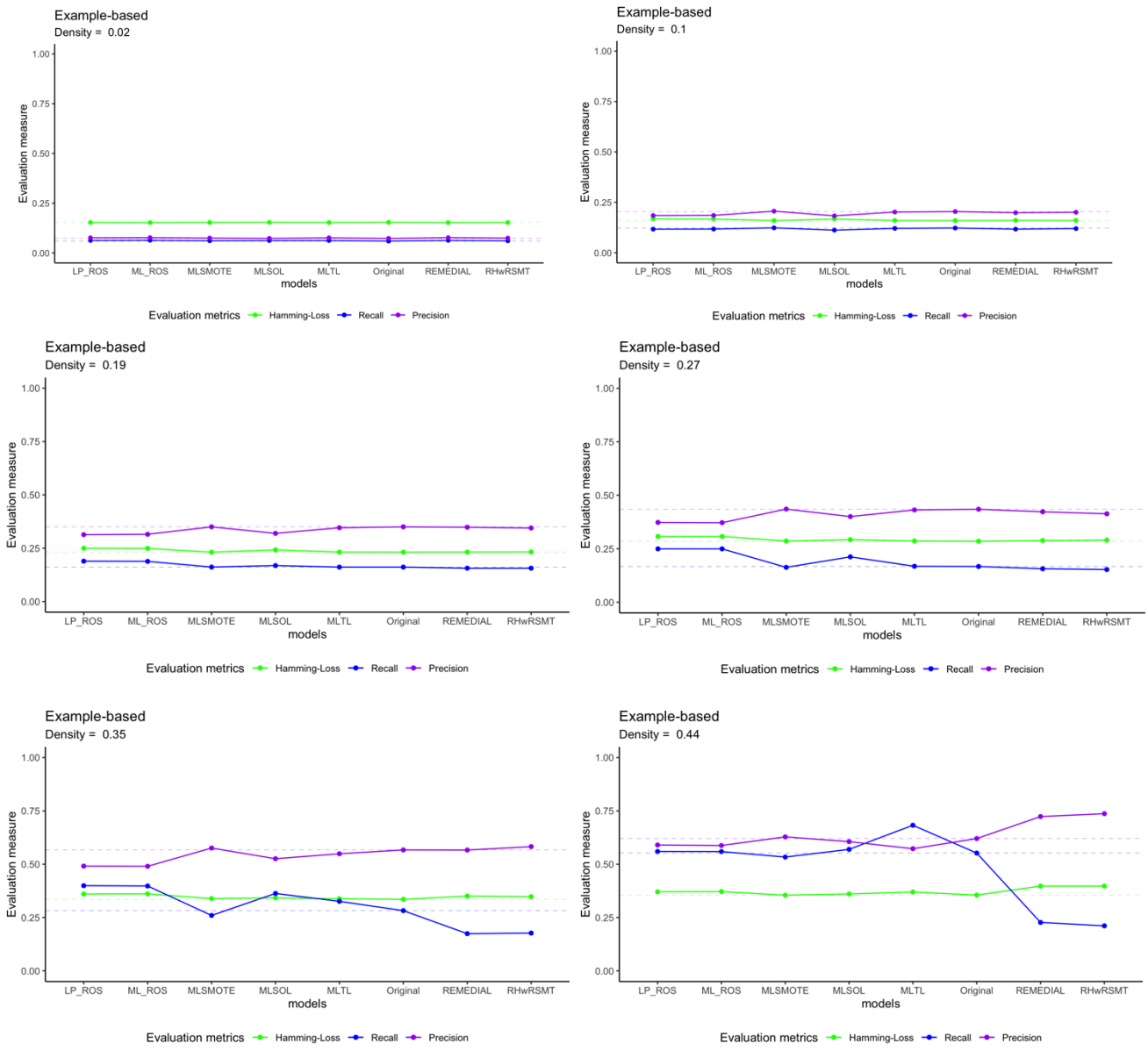


Figure 45: Example-based performance for $K = 10$

In Figure 45 we observe the same phenomenon as we did for $K = 5$, where the changes in performance due to the resampling algorithms become more exaggerated as the global density becomes larger. In the top-left figure we observe the performance for the "special case" with very small global density. The deviations in performance are very small and is cumbersome to see on the visualisation above. Figure 46 below allows us to zoom in closer, to inspect the deviations in performance from the original data.

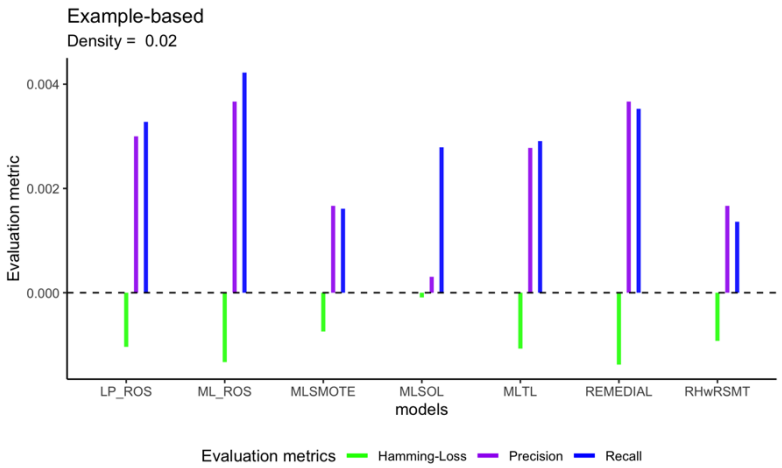


Figure 46: Change in example-based performance for K = 10, density = 0.02

In Figure 46 above, we observe that all the resampling algorithms improve MLC performance in terms of all three metrics. The performance improvement comes through an increase in both Recall and Precision and a decrease in Hamming loss. The resampling algorithms with the most significant impact are LP-ROS, ML-ROS and REMEDIAL. However, the changes in performance remain small.

Referring to Figure 45, as we move toward larger global densities, we observe that MLSMOTE and MLTL are the most successful algorithms. At global densities of 0.1 and 0.21, both algorithms could generate minor improvements in MLC performance, while most other resampling algorithms experienced relatively larger reductions in performance. An inherent characteristic that seems to emerge is that all the algorithms can successfully improve Recall, but this is consistently paired with a reduction in Precision and an increase in Hamming loss. The only exception to this rule is MLSMOTE, REMEDIAL and RHwRSMT, for which the opposite is true. However, for REMEDIAL and RHwRSMT, the increase in Precision is small relative to the reduction in Recall. In the cases where MLSMOTE did not improve the performance in terms of all three metrics, it improved the Precision but not the Recall, which is also true for the $K = 5$ case. It is just less pronounced than for $K = 10$.

7.2.3 $K = 20$

In Figure 47, we observe line graphs representing the performance of the MLC models for $K = 20$ labels across the three example-based evaluation metrics, where each plot represents a different level of global density.

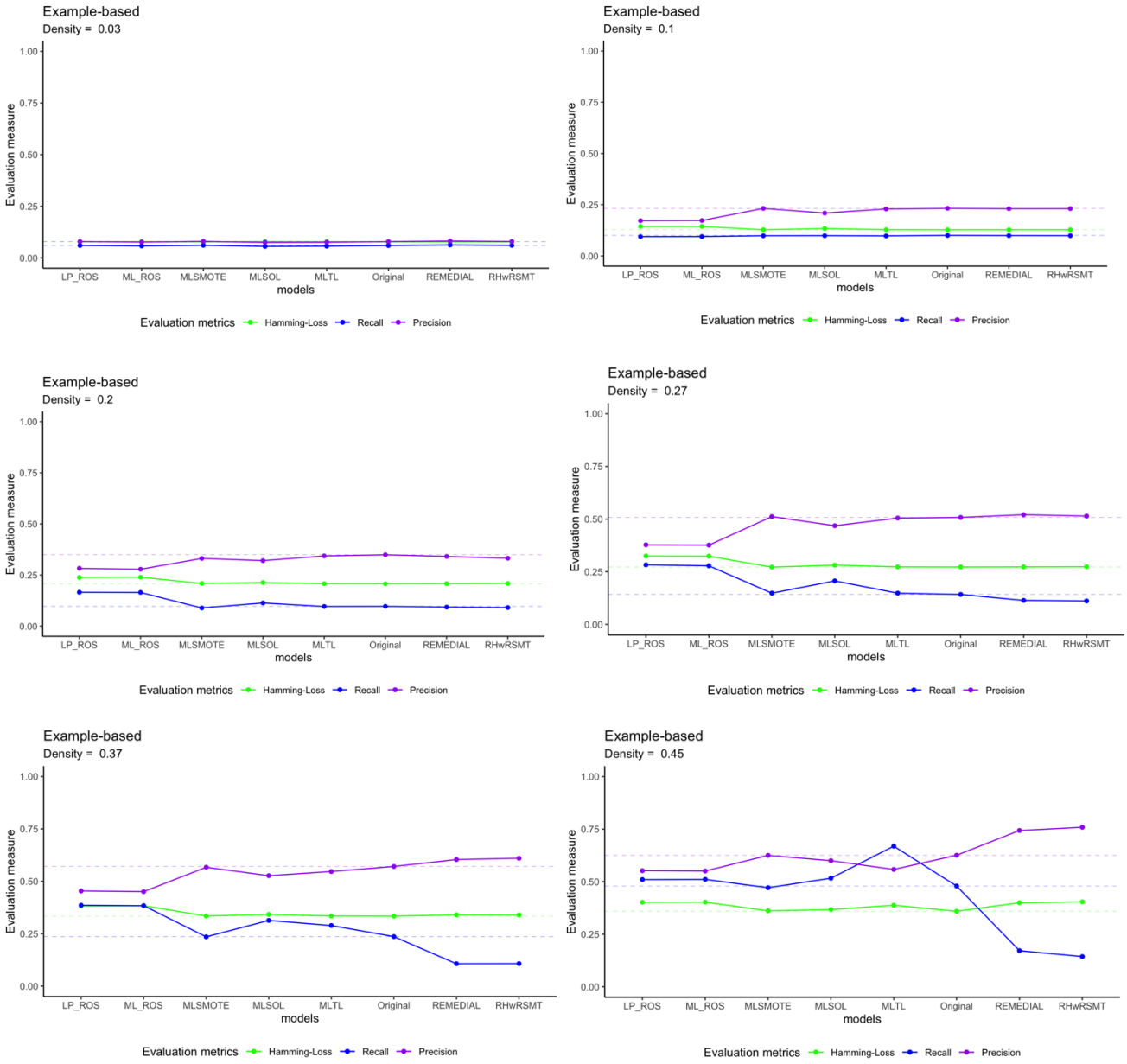


Figure 47: Example based performance for $K = 20$

In the top-left plot above, we observe the performance of all the resampling algorithms for the "special case" with very small global density. Once again, the deviations in performance from the original data are very small when the global density is small and becomes more exaggerated as the global density increases. Figure 48 allows us to zoom in on the deviations in performance for the top-left plot above.

We observe that the most successful resampling algorithms were MLSMOTE, REMEDIAL and RHwRSMT. These models are able to improve performance in terms of all three metrics. Therefore, Hamming loss is reduced and Precision and Recall are increased, relative to the use of the original data. The improvements are very small, but this was the general trend when the global density is very small.

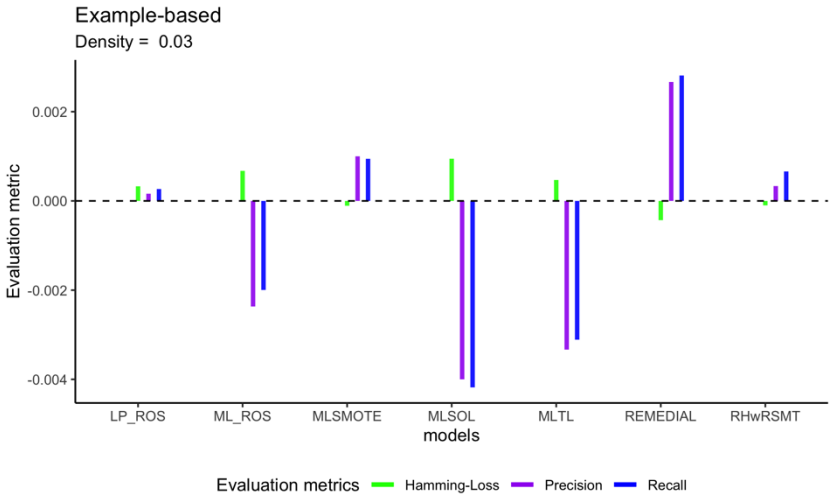


Figure 48: Change in example-based performance for $K = 20$, density = 0.03

As the global density increases, we observe that MLSMOTE tends to be the most successful resampling algorithm since it does not lead to a degradation in performance in terms of any of the three metrics. Generally, MLSMOTE either improved performance or retained the same performance as the original dataset. For $K = 20$, we observe that the resampling algorithms struggle to improve performance in terms of more than one of the evaluation metrics. Similarly to $K = 10$, many of the algorithms can significantly improve Recall, but this is paired with a reduction in performance in terms of Hamming loss and Precision. MLSMOTE, REMEDIAL and RHwRSMT are the algorithms that are able to improve Precision. However, this improvement in Precision is offset with a more significant reduction in Recall.

7.2.4 Conclusions

We found that the resampling algorithms struggled to improve all three of the example-based evaluation metrics simultaneously. The resampling algorithms tend to significantly improve performance in terms of one or two of the evaluation metrics at the cost of worse performance on the other metrics. Most of the algorithms were able to improve Recall. However, this comes at the cost of a reduction in Precision. The only resampling algorithms that were able to improve Precision consistently were MLSMOTE, REMEDIAL and RHwRSMT. Very few of the algorithms were able to improve Hamming loss, and when there were improvements, it tended to be minor improvements.

A trade-off exists between the improvement of Precision and Recall. Including the F1-score, which is the harmonic mean between the Precision and Recall would have been helpful to determine which effect is the largest. The F1-score rewards an algorithm for finding a balance between the Precision and Recall.

The only situations where Hamming loss was improved was at the special cases with very small global density or when an undersampling algorithm like MLTL was used. Therefore, we conclude that the Hamming loss generally does not stand to gain from oversampling at larger global densities. Undersampling algorithms might be a future avenue of research to explore whether improvements in Hamming loss can be found.

Deviations in performance consistently became larger as the global density became larger. Although the resampling algorithms were more successful at improving performance in terms of all three example-based metrics at small global densities, the positive deviations in performance tended to be small. As the global density increased, the deviations in performance became larger. However, generally, the algorithms were only successful at improving one or two of the metrics simultaneously, usually paired with a reduction in performance for the other metrics.

MLSMOTE was the most consistent resampling algorithm throughout the experiments. Except for the "special case" with $K = 5$ labels, MLSMOTE could improve performance in terms of all three example-based metrics or retain the same performance as the original data. MLSMOTE very rarely led to a reduction in performance, and when this was the case, the reduction was minimal. LP-ROS, ML-ROS, MLSOL and MLTL were very successful at improving Recall. However, this was often paired with a reduction in performance in other metrics. REMEDIAL, MLMSOTE and RHwRSMT were able to improve Precision, especially at larger global densities, but this was usually paired with an even bigger reduction in Recall.

Therefore, in terms of the example-based metrics, resampling is not a tool that can lead to a general improvement in MLC performance. Resampling algorithms are able to improve some of the metrics, but not all of them. Therefore, if a researcher is looking to improve a specific metric like Precision or Recall, resampling might be a viable option to improve the specific metric. If a researcher is looking to improve the performance of Hamming loss, undersampling algorithms could be a future avenue for research since generally the oversampling algorithms led to a reduction in performance for the Hamming loss.

7.3 Label-based

The three label-based metrics are Macro-Precision, Macro-Recall and Macro-F1. From the three categories of evaluation metrics, it is reasonable to assume that the resampling algorithms will have the largest positive impact on the label-based metrics. The label-based metrics are sensitive to imbalance in the dataset, therefore MLC models will do well on the label-based metrics if they can classify the minority labels well. Since most of the resampling algorithms try to inflate the minority classes and reduce imbalance, we expect the performance on resampled datasets to be better than the performance on the original dataset. For all three metrics an increase would indicate an improvement in MLC performance, resulting from the resampling.

7.3.1 $K = 5$

In Figure 49 below, we observe line graphs representing the average performance of the MLC models for $K = 5$ labels across the three label-based evaluation metrics, where each plot represents a different level of global density.

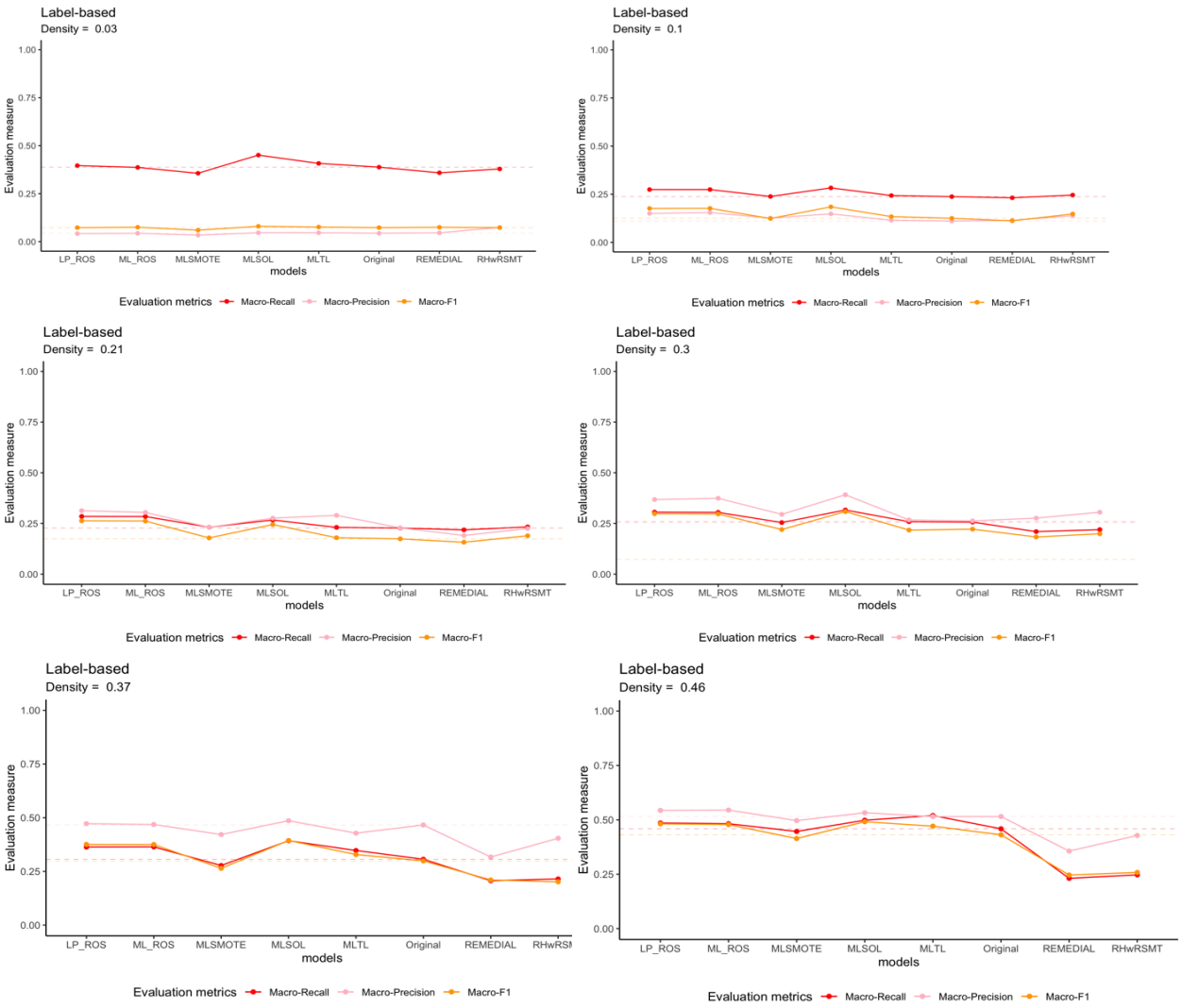


Figure 49: Label-based performance for $K = 5$

From Figure 49 above, we observe that the deviations in performance from the resampling algorithms to the original dataset are much larger than we observed for the example-based metrics. These deviations are also large at smaller global densities, which was not the case for the example-based metrics previously. It is clear that many of the resampling algorithms effectively improve MLC performance. For the $K = 5$ case, the algorithms struggle to improve MLC performance for the "special case" with a very small global density. However, they are very effective at all other levels of global density.

In Figure 50 below, we observe the deviation in performance for the resampling algorithms from the original dataset, when the global density is very small ("special case").

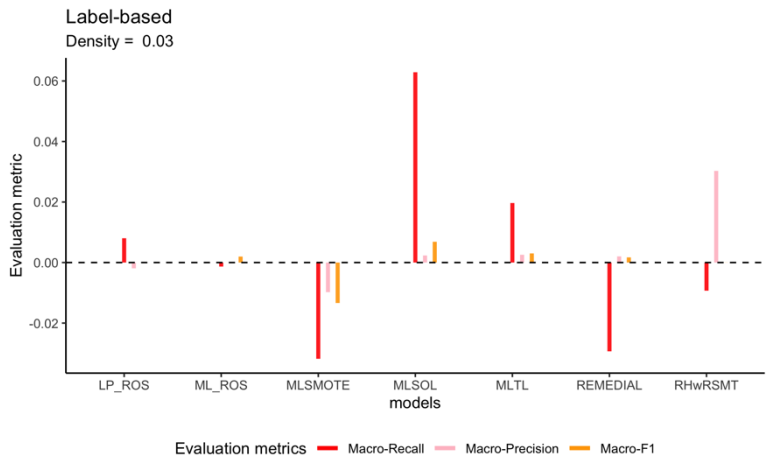


Figure 50: Change in label-based performance for $K = 5$, density = 0.03

From Figure 50, we observe that most resampling algorithms are unsuccessful at improving MLC performance. The resampling algorithms that successfully improve MLC performance in terms of all three metrics are MLSOL and MLTL. MLSMOTE, one of the most successful algorithms in terms of example-based metrics, causes a reduction in performance for all three metrics, which was also the case for the example-based metrics with K = 5 and the "special case".

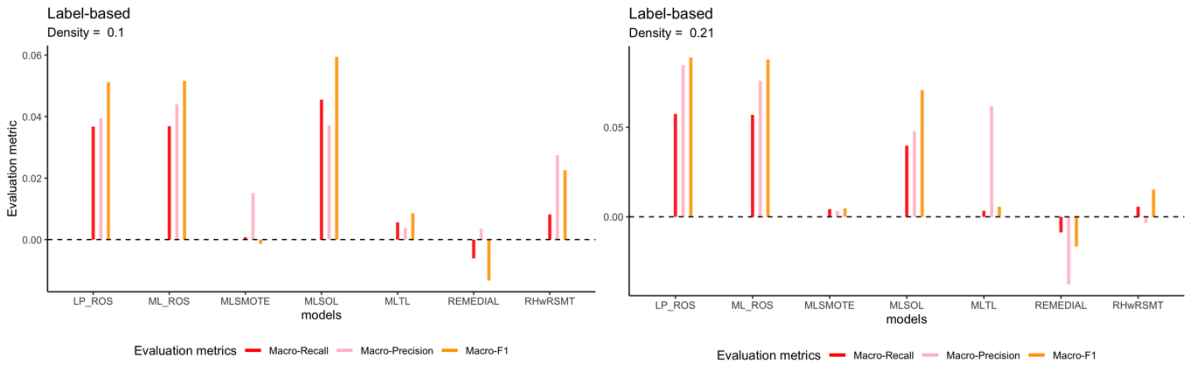


Figure 51: Change in label-based performance for $K = 5$, density = 0.1, 0.21

Figure 51 shows the same visualisation as before, but for global densities of 0.1 and 0.21. We observe that many of the algorithms are now successful at improving MLC performance for all three metrics. LP-ROS, ML-ROS, MLSOL and MLTL are all successful at improving MLC performance for all three metrics. The algorithms are especially effective at improving Macro-F1.

The deviations in performance from the original dataset for global densities of 0.3, 0.37 and 0.46 are observed in Figure 52. As the global density becomes larger than 0.3, we observe that the dominant resampling algorithms are LP-ROS, ML-ROS and MLSOL. These resampling algorithms are effective at improving MLC performance for all three metrics. We also observe that REMEDIAL and RHwRSMT lead to reductions in performance for all three metrics as the global density becomes larger.

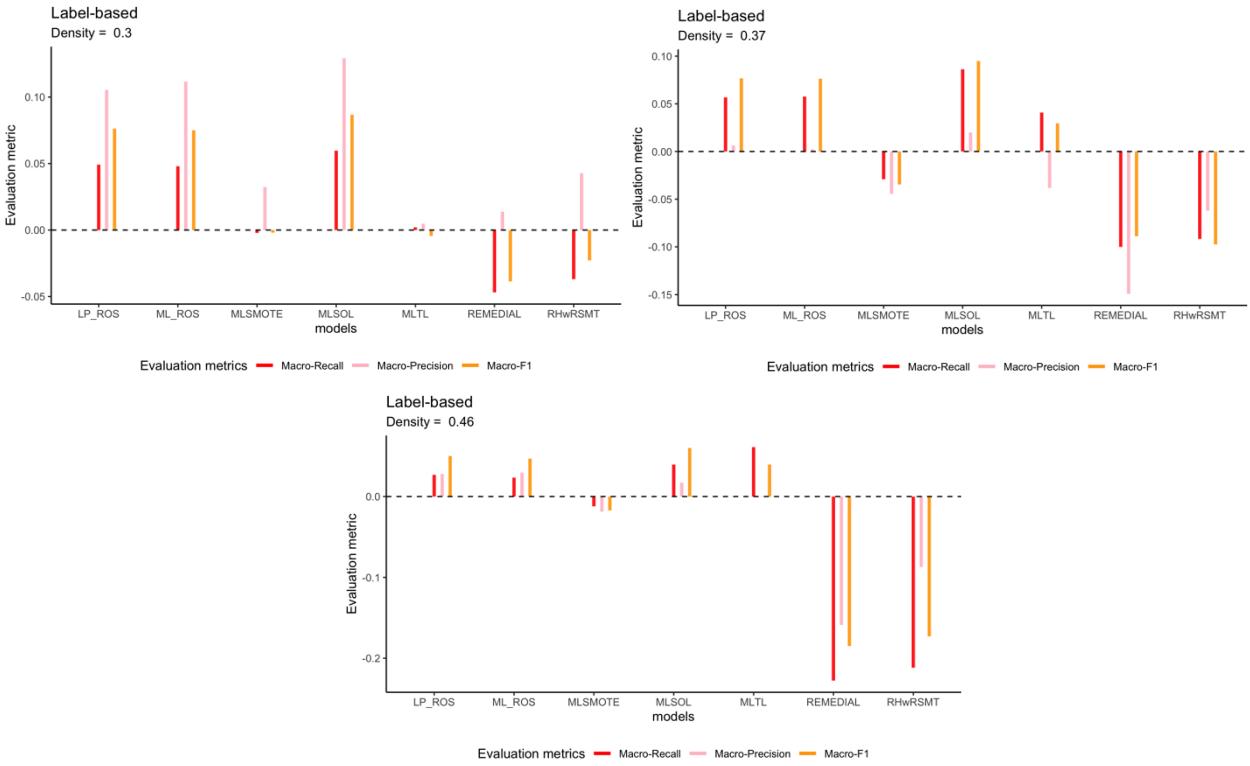


Figure 52: Change in label-based performance $\bar{K}$ = 5, density = 0.3, 0.37, 0.46

7.3.2 $K = 10$

In Figure 53, we observe line graphs representing the average performance of the MLC models for $K = 10$ labels across the three label-based evaluation metrics, where each plot represents a different level of global density.

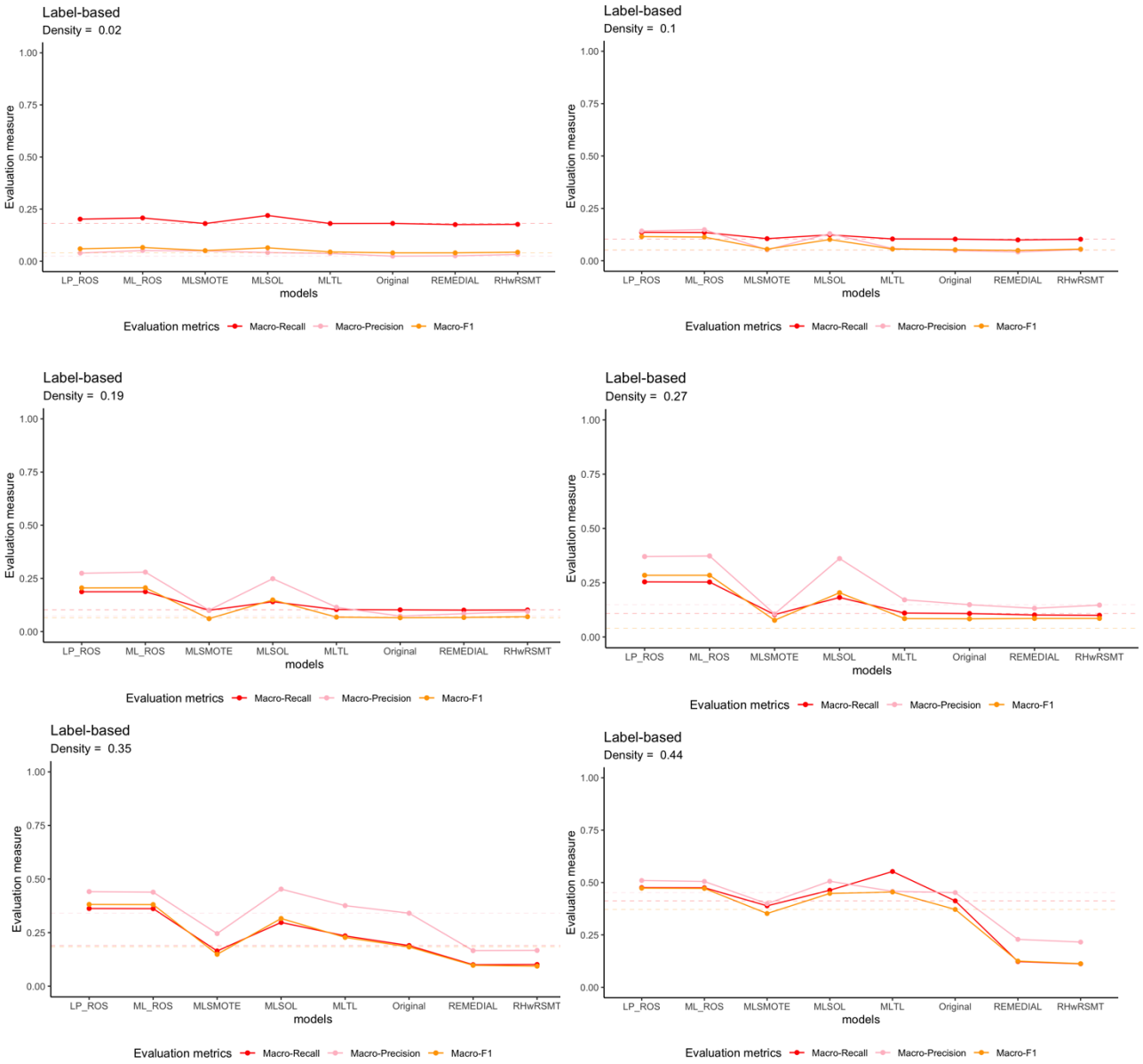


Figure 53: Label-based performance for $K = 10$

From Figure 53, we observe that the deviations in performance from the original data are already large at small densities and these deviations continue to become larger as the global densities increases. For K = 10, we observe a few resampling algorithms that are consistently the most effective at improving the label-based metrics across different global densities. The most successful algorithms are LP-ROS, ML-ROS and MLSOL. These algorithms consistently produced large improvements in performance over the original data.

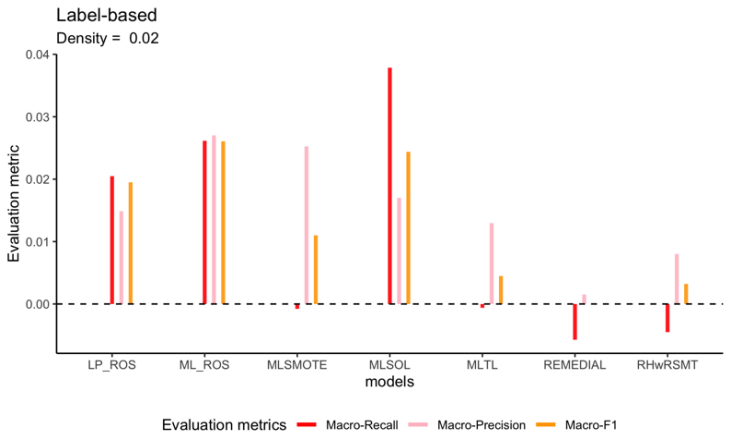


Figure 54: Change in label-based performance for K = 10, density = 0.02

In Figure 54, we observe the deviations in performance from the original data for the "special case", with a global density of 0.02. Unlike for $K = 5$, there are several algorithms that led to small improvements in terms of the label-based metrics for the "special case". LP-ROS, ML-ROS and MLSOL improve the label-based performance in terms of all three metrics.

Referring to Figure 53, as the global density increases, we observe that LP-ROS, ML-ROS and MLSOL remain the most successful resampling algorithms and improve performance in terms of all three metrics at all levels of global density. For a global density of 0.35 and 0.44, we observe that MLTL is also succesful at improving performance for all three label-based metrics, along with the other three algorithms.

Similarly to $K = 5$, we observe that REMEDIAL and RHwRSMT experienced a large reduction in performance as the global density became larger than 0.27. This reduction increases as the global density increases. We also observe that the deviations in performance relative to the original data are very large. Therefore, those algorithms that improve MLC performance lead to large improvements. MLSMOTE, one of the most successful algorithms in terms of the example-based metrics experienced a consistent decline in performance as the global density increased.

7.3.3 $K = 20$

In Figure 55, we observe line graphs representing the average performance of the MLC models for $K = 20$ labels across the three label-based evaluation metrics, where each plot represents a different level of global density.

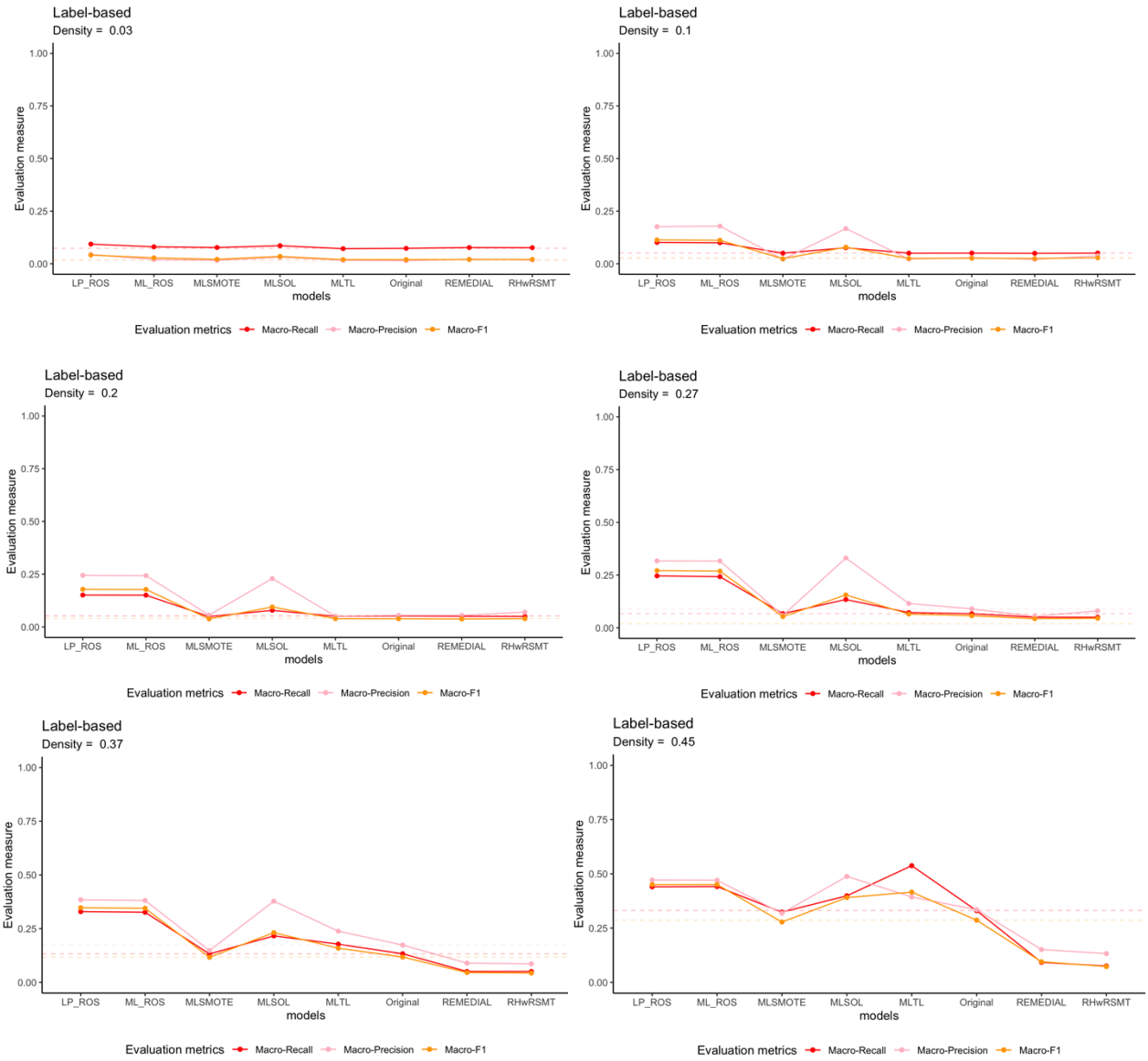


Figure 55: Label-based performance for $K = 20$

In Figure 55, we observe similar patterns to what has been observed for $K = 5$ and $K = 10$. We observe that the most successful resampling algorithms at all levels of global density are ML-ROS, LP-ROS and MLSOL. These algorithms consistently improve MLC performance in terms of all three label-based metrics.

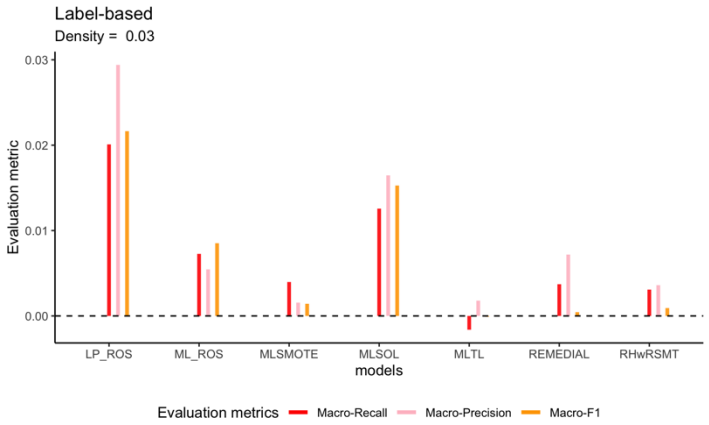


Figure 56: Change in label-based performance for K = 20 , density = 0.03

In Figure 56, we observe the deviations in performance from the orignal data for each of the resampling algorithms, when we obsevrve the "special case" with a global density of 0.03. We observe that LP-ROS has the largest positive impact on MLC performance, followed by MLSOL and ML-ROS. MLSMOTE, REMEDIAL and RHwRSMT also have a positive impact on performance in terms of all three metrics.

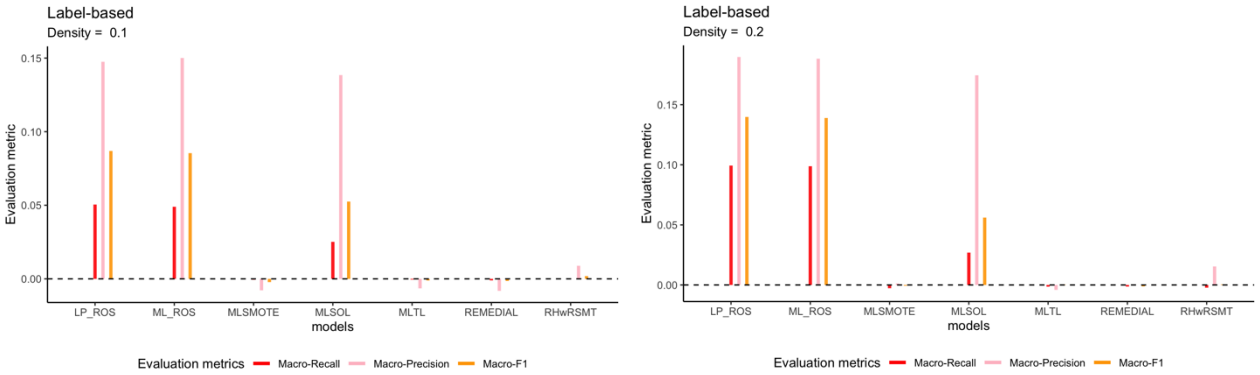


Figure 57: Change in label-based performance for K = 20, density = 0.1,0.2

When the global density increases to 0.1 and 0.2, we observe that only LP-ROS, ML-ROS and MLSOL are able to successfully improve MLC perfromance in terms of all three label-based metrics. MLSMOTE, REMEDIAL and RHwRSMT are no longer able to improve MLC performance.

In Figure 58, we observe the deviations in performance for global density of 0.27, 0.37 and 0.45. We observe that LP-ROS, ML-ROS and MLSOL consistently improve MLC performance across all levels of global density. MLTL increases in performance as the global density becomes larger and is on par with the other algorithms for a global density of 0.45. Once again, we observe that REMEDIAL and RHwRSMT experience a considerable decline in performance as the global density increases.

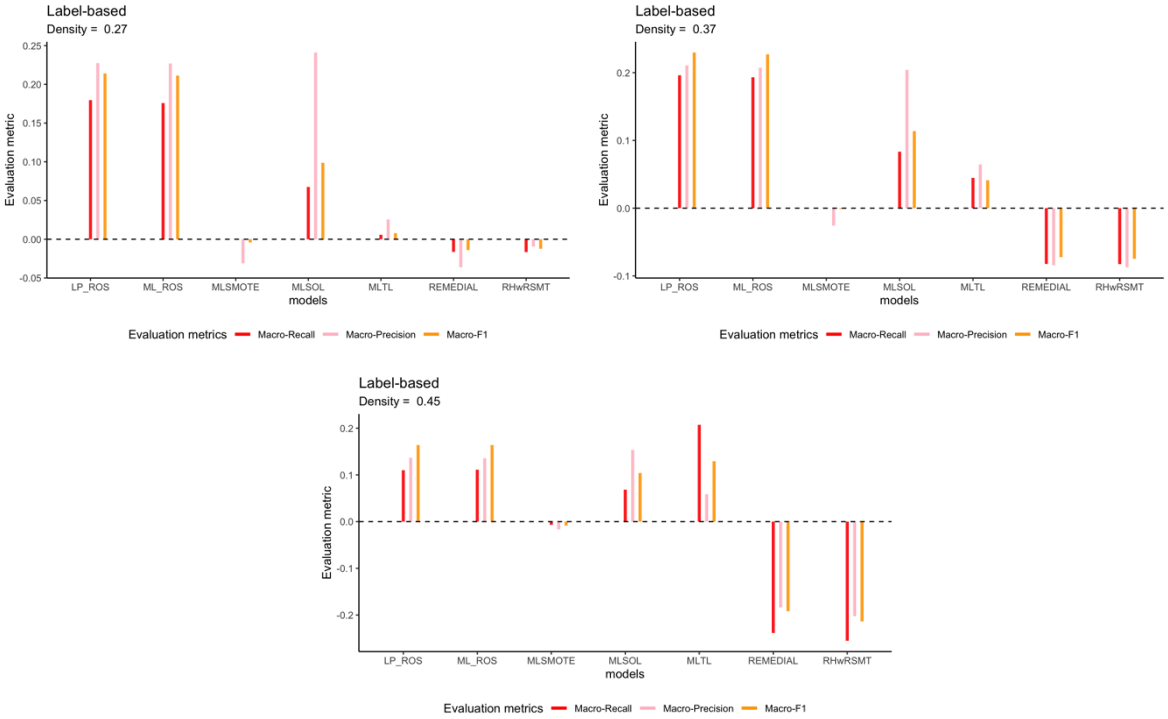


Figure 58: Change in label-based performance for $\bar{K}$ = 20, density = 0.27, 0.37, 0.45

7.3.4 Conclusions

The results from the label-based metrics produce contrasting results to the example-based metrics seen previously. We expected the resampling algorithms to be very successful at improving the MLC performance on the label-based metrics and this was the case. The resampling algorithms were able to produce large improvements in MLC performance over the original data and these deviations in performance were large at small global densities and got even bigger at larger global densities.

The label-based metrics were specifically chosen as imbalance aware evaluation metrics that are sensitive to predictions on the minority classes. The Macro-F1 score is often chosen as the evaluation metric to rank the efficacy of resampling algorithms in multi-label literature. An improvement in the label-based metrics shows that the resampling algorithms were able to make better predictions on the minority classes, compared to when the original data was used. Therefore, it is reasonable to assume that an improvement in the label-based metrics indicates that the resampling algorithms were able to successfully address imbalance and hence improve the classification of the minority labels.

Three resampling algorithms emerged as the most successful for the label-based metrics. LP-ROS, ML-ROS and MLSOL were able to improve performance in terms of all three metrics for every combination of labels and global density. These algorithms were also very successful at larger global densities and were able to generate large improvements in performance at these global densities. MLTL was also successful at improving performance in terms of all three metrics on occasions. Therefore, if the goal is to improve the classification performance on the minority labels of an imbalanced multi-label classification problem, LP-ROS, ML-ROS and MLSOL would be recommended.

An interesting observation was how REMEDIAL and RHwRSMT experienced a consistent decline in performance as the global density increased. This decline was amplified at a global density of 0.4 and 0.5. The common denominator among the two algorithms are that they both make use of label decoupling. Label decoupling will have a negative impact and cause a reduction in label-based performance as the global density increases. Therefore, we would recommend avoiding the use of algorithms that make use of label decoupling when the global density becomes larger in multi-label classification problems.

MLSMOTE, the most successful algorithm for the example-based metrics, did not replicate its impressive performance for the label-based metrics. MLSMOTE either experienced a small reduction in performance or retained the same performance as the original data.

7.4 Ranking-based metrics

The three ranking-based metrics are Coverage, Ranking-Loss and One-Error. These metrics are calculated on the label rankings rather than the binary predictions. Therefore, it is an entirely different way of assessing an MLC model from the two sections seen before. From Chapter 5, we observe that Coverage is measured on a different scale to all the other evaluation metrics and will therefore be discussed separately from Ranking-Loss and One-Error in this section. For all the metrics a decrease would indicate an improvement in MLC performance, resulting from the resampling.

7.4.1 $K = 5$

In Figure 59, we observe line graphs representing the average performance for One-Error and Ranking-Loss on the MLC models for $K = 5$ labels, where each plot represents a different level of global density.

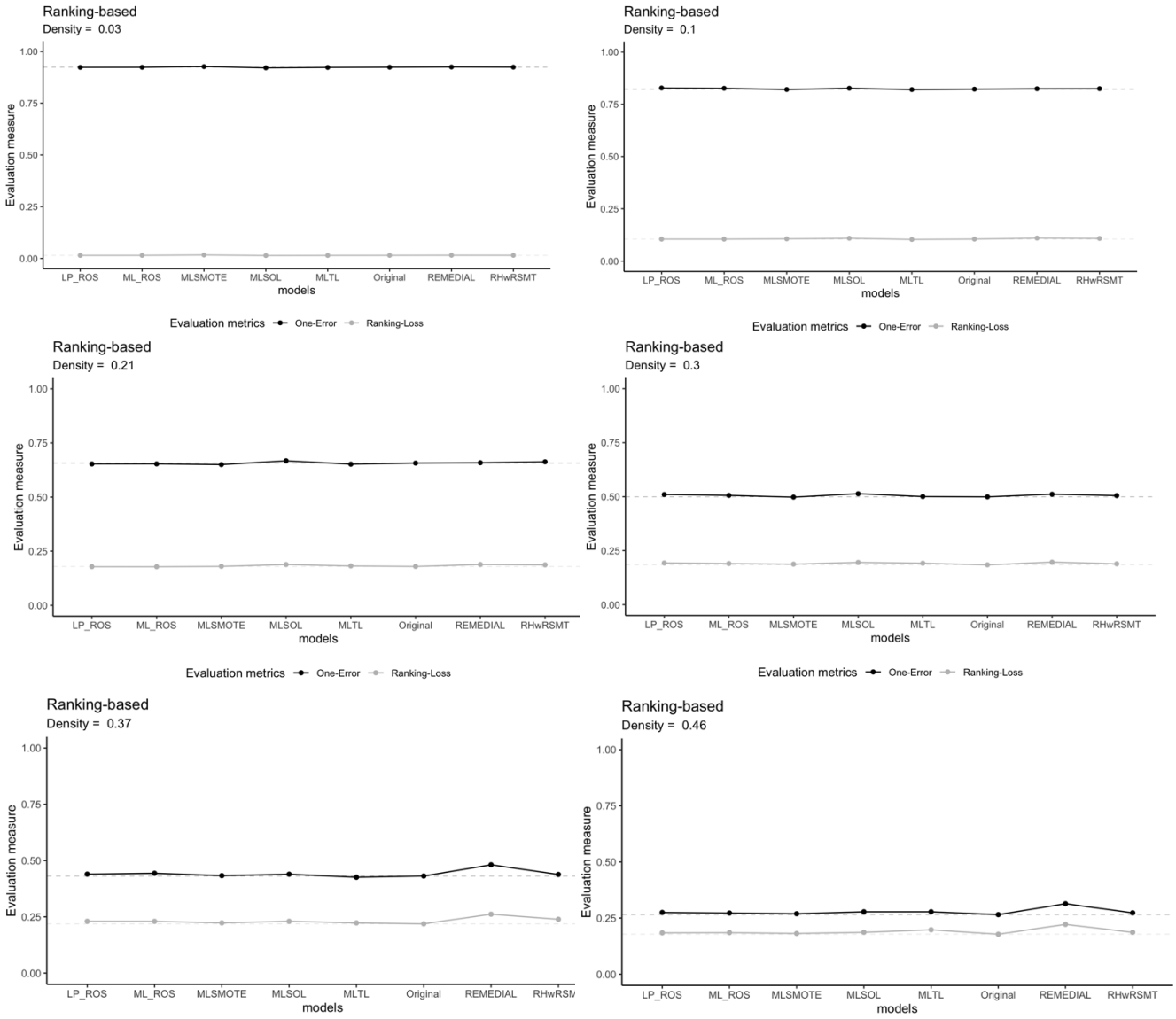


Figure 59: Ranking-based for $K = 5$

From Figure 59, we observe that the deviations in performance for One-Error and Ranking-Loss are not large. The deviations in performance do become larger as the global density increases, however it is still small in comparison to the example-based and label-based metrics seen previously. We observe that none of the models can improve performance in any meaningful way at any level of global density. REMEDIAL and MLSOL are the two worst-performing resampling algorithms and lead to the most significant reduction in performance. The decrease in performance from REMEDIAL becomes larger as the global density increases.

In Figure 60, we observe line graphs representing the performance for Coverage on the MLC models for $K = 5$ labels, where each plot represents a different level of global density.

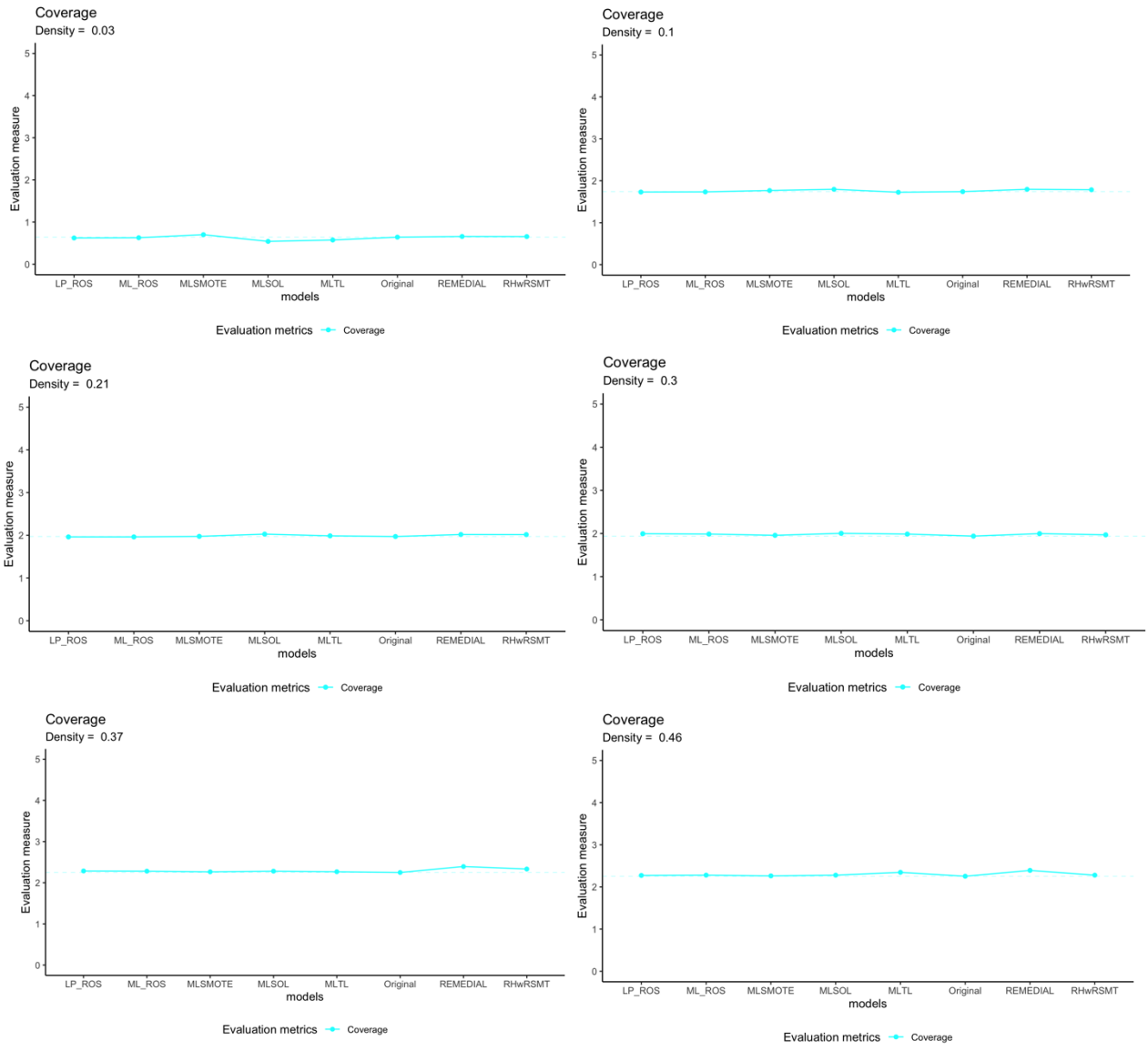


Figure 60: Coverage for $K = 5$

From Figure 60, we observe that there are only very small deviations in Coverage at all levels of global densities. For the "special case" when the global density is very small, MLSOL, MLTL,

LP-ROS, and ML-ROS can improve Coverage. At a global density of 0.1, LP-ROS, ML-ROS and MLTL are effective at producing a slight increase in performance, and at a global density of 0.2, only LP-ROS and ML-ROS were able to improve Coverage. At global densities larger than 0.2, none of the algorithms could effectively improve Coverage. REMEDIAL was the worst performing algorithm in terms of Coverage and led to increasing reductions in performance as the global density increased.

7.4.2 $K = 10$

In Figure 61, we observe line graphs representing the performance for One-Error and Ranking-Loss on the MLC models for $K = 10$ labels, where each plot represents a different level of global density.

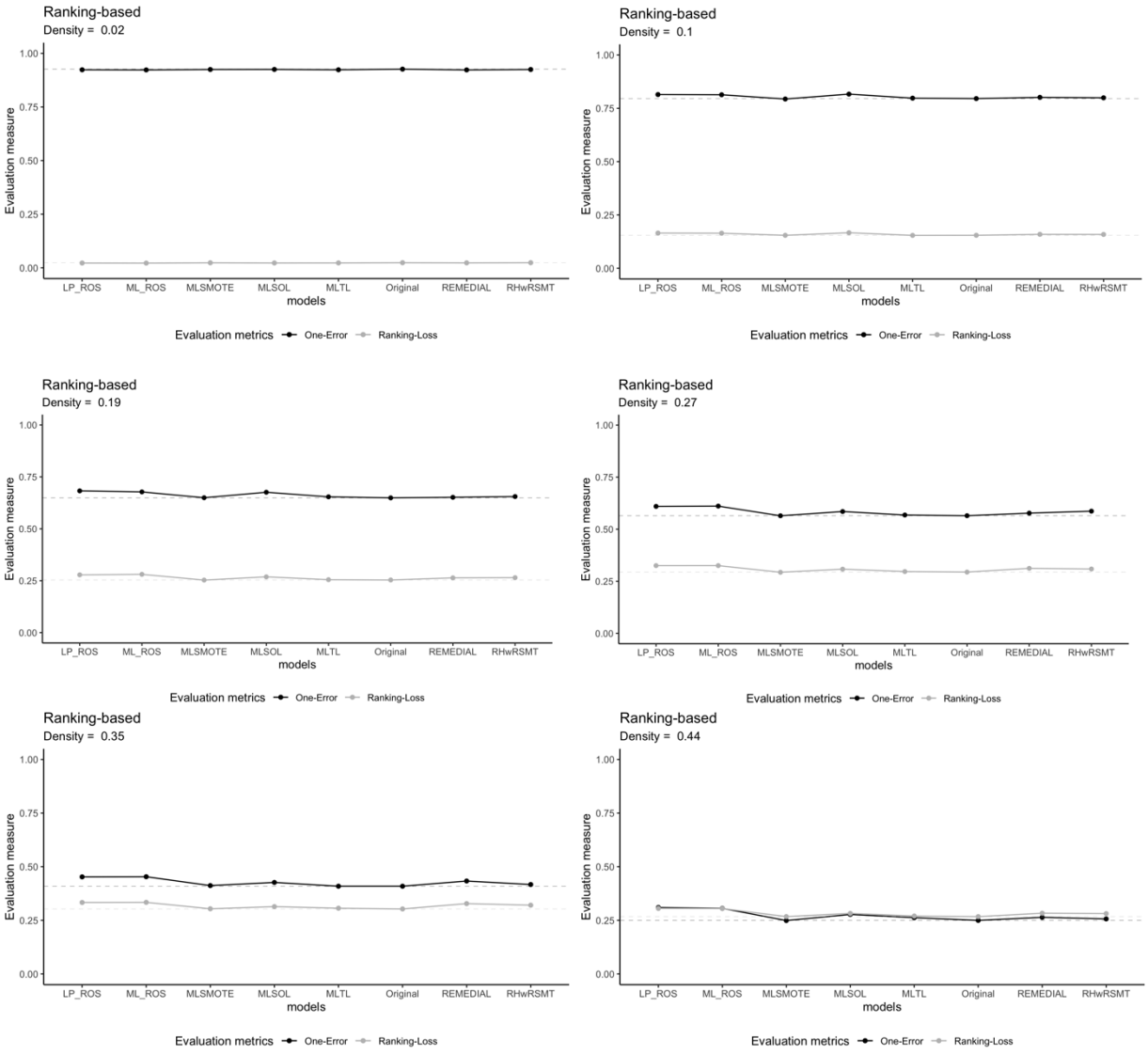


Figure 61: Ranking-based for $K = 10$

For $K = 10$, we also observe that the deviations in performance are minimal. The deviations do increase as the global density increases but are still small compared to the deviations

observed for the label-based metrics. For the "special case" with a very small global density, we observe that all the resampling algorithms can make minor improvements in terms of both metrics. However, as the global density increases, none of the algorithms can consistently improve performance. MLSMOTE and MLTL are the two most successful algorithms since they improved performance on some occasions and never led to a significant reduction in performance. As the global density increases, the reduction in performance from LP-ROS and ML-ROS consistently increases.

In Figure 62, we observe line graphs representing the performance for Coverage on the MLC models for $K = 10$ labels, where each plot represents a different level of global density.

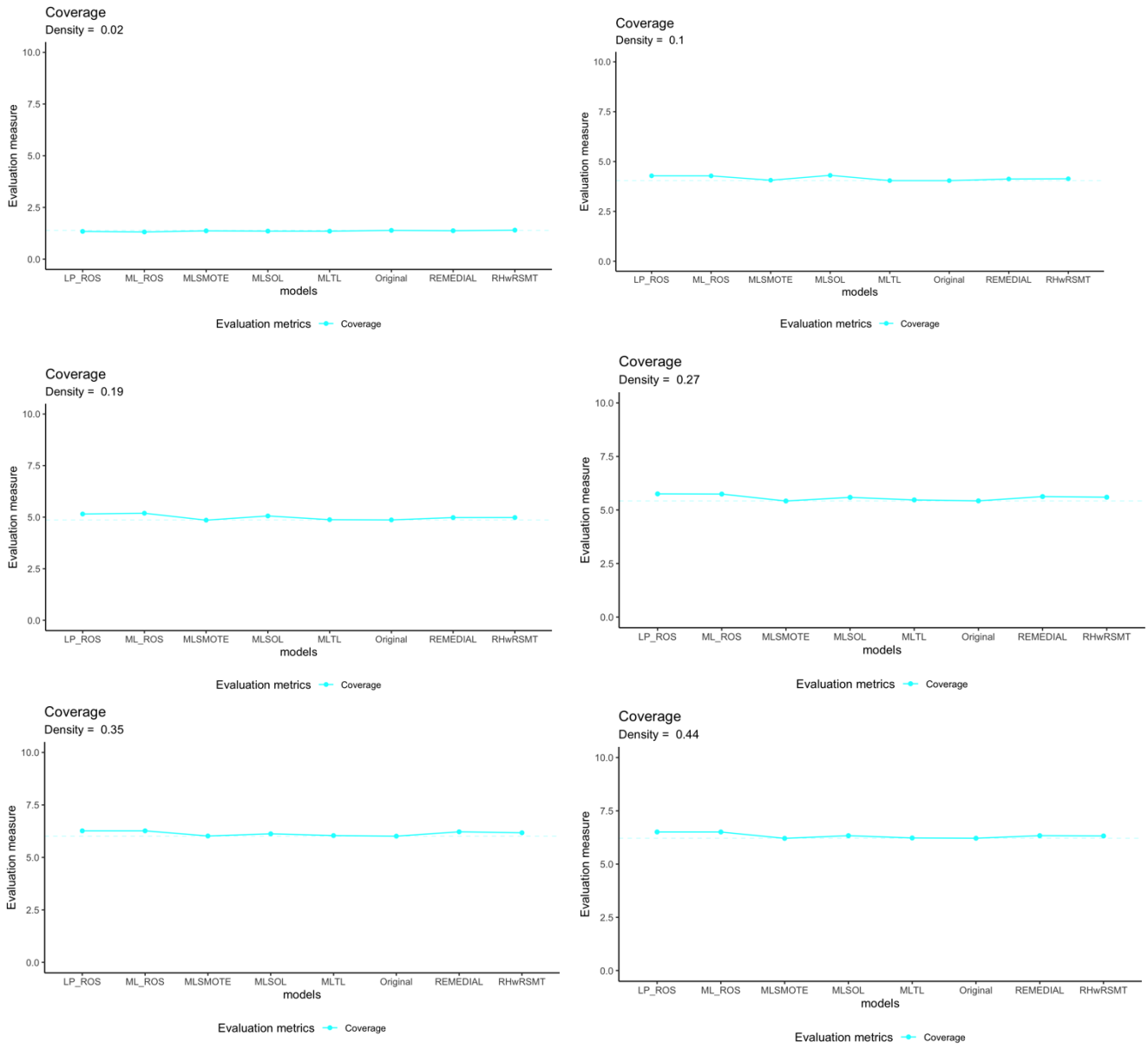


Figure 62: Coverage for $K = 10$

We observe that the deviations in performance are minimal at all levels of global density. Most of the algorithms tend to produce similar levels of Coverage at all levels of global density. For the "special case" with a very small global density, we observe that all the resampling algorithms successfully reduced the Coverage slightly. However, as the global density increases, most of the algorithms lead to a slight increase in Coverage, except for MLSMOTE, which led to a very small reduction in Coverage on two occasions.

7.4.3 $K = 20$

In Figure 63, we observe line graphs representing the average performance for One-Error and Ranking-Loss on the MLC models for $K = 20$ labels, where each plot represents a different level of global density.

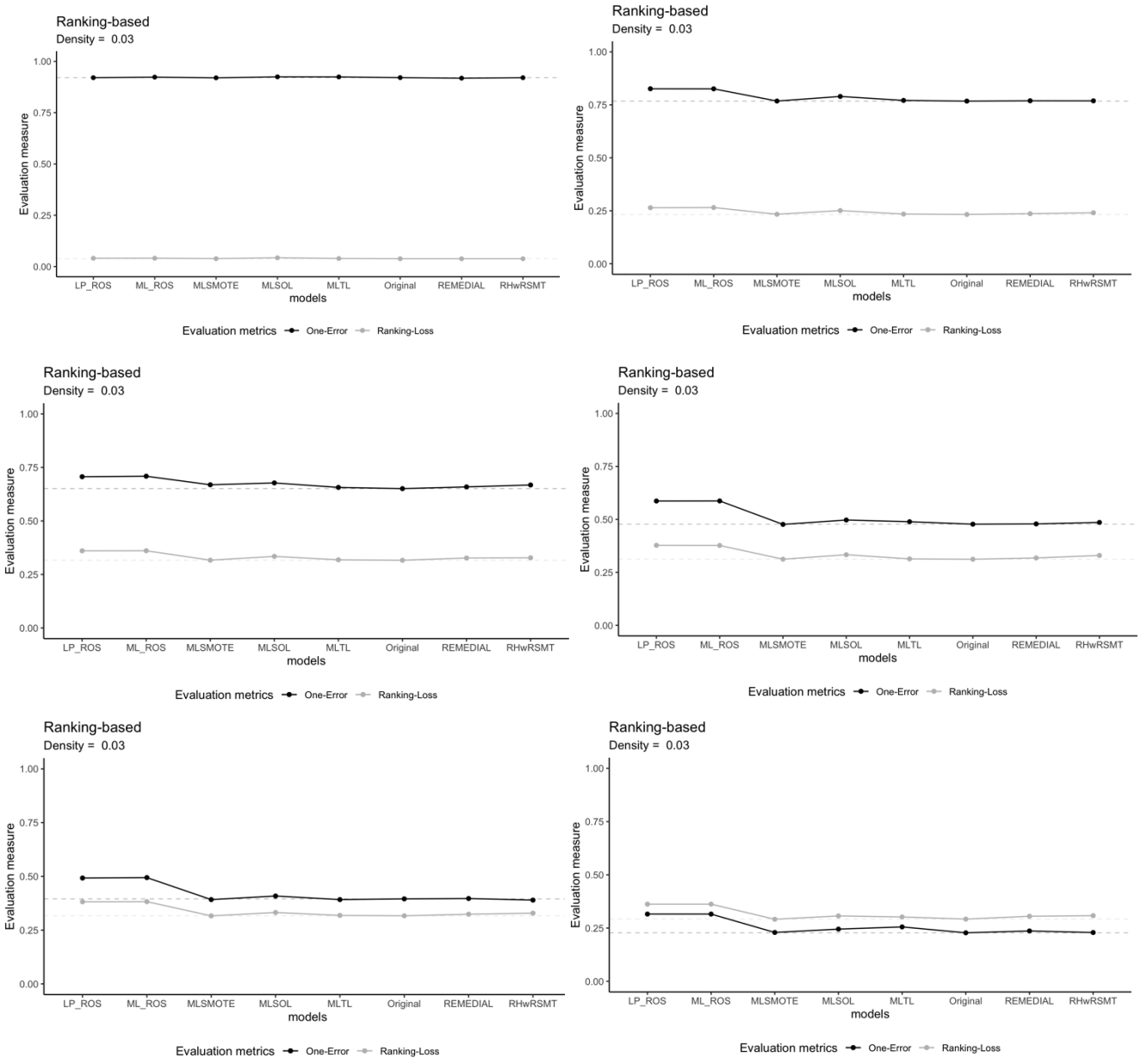


Figure 63: Ranking-based for $K = 20$

We observe that none of the resampling algorithms could improve MLC performance in terms of either of the metrics. The performance of the resampling algorithms was either the same as the original data or worse. LP-ROS, ML-ROS and MLSOL tended to have the worst performance and led to a reduction in performance at all levels of global density. This reduction in performance increased as the global density increased.

In Figure 64, we observe line graphs representing the average performance for One-Error and Ranking-Loss on the MLC models for $K = 20$ labels, where each plot represents a different level of global density.

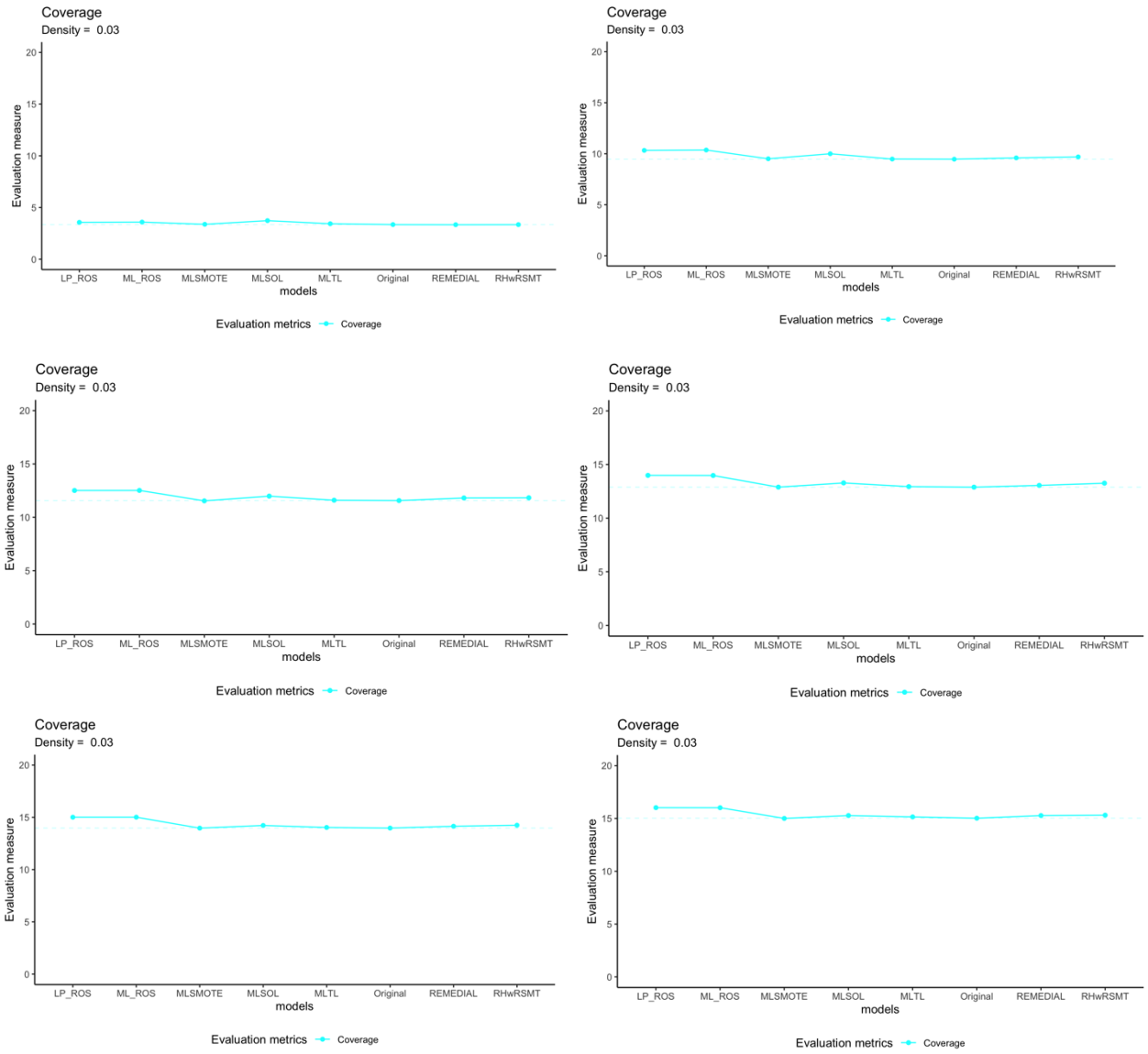


Figure 64: Coverage for $K = 20$

From Figure 64, we observe that most deviations in performance are very small. Generally, the resampling algorithms do not lead to significant deviations in Coverage. LP-ROS, ML-ROS and MLSOL were the models that struggled the most since they led to the largest increase in Coverage of all the resampling algorithms. We observe that MLSMOTE and MLTL were the

most successful resampling algorithms in terms of Coverage since they either matched the performance of the original data or led to a very slight decrease in Coverage.

7.4.4 Conclusions

The resampling algorithms were less successful at improving MLC performance in terms of the ranking-based metrics, compared to the example-based and label-based metrics before. In general, the deviations in performance tended to be small. Even as the global density increased, the deviations did not become very large as we experienced with the label-based and example-based metrics. Most of the resampling algorithms either retained the same performance as the original data or led to a slight reduction in performance.

MLSMOTE and MLTL were the two most successful resampling algorithms for the ranking-based metrics, although they rarely improved performance. They were able to consistently match the performance of the original data and on some occasions could make small improvemements in performance. Most of the resampling algorithms tended to struggle to improve the ranking-based performance, therefore at least matching the performance of the original data was a good property to have, when an algorithm could not make improvements in performance.

LP-ROS, ML-ROS and MLSOL were consistently the worst resampling algorithms in terms of the ranking-based metrics. These algorithms consistently led to a reduction in performance. The reduction in performance tended to increase as the global density increased. This result comes in contrast to Section 7.3, where these three algorithms were found to consistantly be the most successful algorithms at improving the label-based evaluation metrics.

The observation above, leads us to make a similar conclusion to that in Section 7.2.4 for Hamming loss. Undersamping could be a future area of research to improve the performance on the ranking-based metrics, instead of using oversampling. We would not recommend the resampling algorithms as a general mechanism to improve ranking-based performance since the resampling algortihms struggled to make improvements on the ranking-based metrics compared to the use of the original data, although the performance for most of the resampling algorithms tended to remain the same as for the original data.

Section 7.2.4 reccomends that LP-ROS, ML-ROS and MLSOL be used as resampling algorithms to improve the label-based evaluation metrics. The tradeoff in performance with the ranking-based metrics should be taken into consideration when these three resampling algorithms are used. The increase in label-based performance that these algorithms are able to generate will be accompanied with a reduction in ranking-based performance.

Chapter 8: Consolidation

8.1 Introduction

The second research goal addressed whether there is one form of resampling preferred to all others. This builds on the information we have seen in the preceding chapter. In Chapter 7 we observed how MLC performance changes when resampling algorithms are used to preprocess the data instead of just using the original data.

The information we have observed so far has left us with some mixed results. A clear answer to the question would be “No”. There is no form of resampling that is preferred to all others. This analysis was performed with a mix of different evaluation metrics to form a holistic approach that would provide a big picture of how the resampling algorithms affected MLC performance. Some algorithms are preferred to others in certain situations. However, no resampling algorithm emerges as the best in all scenarios.

The use of three categories of evaluation metrics allowed us to conclude that the different types of resampling algorithms affect performance differently. Some of the algorithms are very effective for example-based metrics but perform poorly for label-based metrics. The algorithms are also, in general, better at improving specific types of evaluation metrics. For instance, the resampling algorithms were very effective at enhancing label-based metrics but struggled to improve the performance for ranking-based metrics.

Therefore, when to use specific resampling algorithms depends on what the goal of the study is. Choosing the correct resampling algorithm would depend on which evaluation metric is important to that specific study. If Precision is important, we will select one of the resampling algorithms that effectively improve Precision without creating too much of a reduction in some of the other metrics. In the next section, we will explore which algorithms are the most effective for the different evaluation metrics and what the downside might be when the algorithms are used.

8.2 Recommendation

8.2.1 Example-based

We found that the resampling algorithms were effective at improving performance for the example-based metrics. However, the algorithms struggled to improve all three metrics simultaneously and could usually only improve one of the three metrics. The deviations in performance became larger as the global density increased.

Generally, MLSMOTE was the most effective resampling algorithm for the example-based metrics. MLSMOTE either improved performance or retained the same performance as the original data in the cases it could not improve performance. MLSMOTE was the most consistent algorithm across all the example-based metrics and rarely led to a reduction in performance in terms of any metrics.

MLSMOTE, REMEDIAL and RHwRSMT were the only algorithms that were able to improve Precision. However, this improvement usually came at the cost of a reduction in Recall. LP-ROS, ML-ROS, MLTL and MLSOL were effective at improving Recall. However, this came at the expense of a decrease in Precision. It was rare for any algorithm to improve the Hamming loss. There was no clear pattern in which algorithms are preferred in terms of Hamming loss. The changes in Hamming loss were negligible, even at larger global densities. Therefore, the resampling algorithms are practical at either improving Recall or Precision, and a trade-off exists between these two metrics to improve performance. It would therefore make sense to rather consider the F1 score as an evulation metric in future studies.

Therefore, in terms of a performance recommendation for the example-based metrics, MLSMOTE would be the safe choice. However, if Precision needs to be improved REMEDIAL or RHwRSMT could also be used. These algorithms do come with downside in terms of other metrics. If improving Recall is the objective LP-ROS, ML-ROS, MLTL or MLSOL could be considered.

8.2.2 Label-based

The resampling algorithms were highly successful at improving performance in terms of the label-based metrics. The algorithms that were able to improve all three metrics made considerable improvements in performance and were very consistent. Gains in performance for the label-based metrics provides encouraging signs for the efficacy of the resampling algorithms as a preprocessing tool to improve MLC performance. An improvement in performance from the original data shows that the resampling algorithms could better represent the minority classes and improve the classification performance of these underrepresented classes.

LP-ROS, ML-ROS and MLSOL were able to make large improvements in all three metrics at all levels of global density. Using these algorithms would be effective for label-based metrics. However, all three metrics also produced poor performance in terms of ranking-based metrics. Therefore, they effectively improve performance for the label-based metrics, but this comes at the cost of worse performance on the ranking-based metrics. MLTL and MLSMOTE

were also successful at improving performance in terms of the label-based metrics. However, they were not as consistent as LP-ROS, ML-ROS and MLSOL, and the performance improvements were smaller. But, MLSMOTE and MLTL come with fewer drawbacks and risk of reduced performance in other metrics.

REMEDIAL and RHwRSMT experienced a consistent reduction in label-based performance as the global density increased. These reductions in performance were also large. Therefore, we would advise avoiding these algorithms at large global densities. There was no trade-off in performance between the three metrics. Generally, if a resampling algorithm did well, it did well in terms of all three metrics, and if it did poorly, performance would decrease in terms of all three metrics.

To optimise performance in terms of the label-based metrics, LP-ROS, ML-ROS or MLSOL should be considered. All three algorithms do provide downside in terms of some other metrics. However, MLSOL is the safest in this regard and would be the best all round resampling algorithm of the three since LP-ROS and ML-ROS do have some considerable drawbacks in terms of the ranking-based metrics.

8.2.3 Ranking-based

All the resampling algorithms struggled to improve performance in terms of the ranking-based metrics. Deviations in performance remained small, even as the global density increased. Therefore, even though the algorithms did not improve performance, they also generally did not lead to large reductions in performance.

MLSMOTE and MLTL were the most successful algorithms in terms of ranking-based metrics. They were able to consistently match the original data's performance and improve performance on some occasions. Most of the other algorithms invariably led to small reductions in performance. Therefore, matching the performance of the original data could be seen as a success for the ranking-based metrics.

LP-ROS, ML-ROS and MLSOL consistently led to a reduction in performance for the ranking-based metrics. These reductions also increased as the global density increased. This is an interesting observation since these are the three very successful algorithms in terms of label-based metrics. Therefore, the gains in performance observed for the label-based metrics are somewhat offset by the reduction in performance for the ranking-based metrics. Although, the increase in performance seen for the label-based metrics are much larger than the reduction in performance for the ranking-based metrics.

Improving the performance on the ranking-based metrics through resampling might not be a feasible goal. Most of the algorithms struggled to improve performance, and when the algorithms did improve performance, they tended to be minor improvements. Therefore, we would not recommend resampling algorithms to improve ranking-based performance. However, if not reducing the ranking-based metrics is part of the goal, we recommend avoiding LP-ROS, ML-ROS and MLSOL. Generally, the deviations in performance for the ranking-based metrics tended to be minor, in a positive and negative sense.

8.3 Degree of resampling

In Chapter 7 we observed that the resampling algorithms' effect on performance became amplified as the global density increased. It is reasonable to assume that this is due to resampling algorithms becoming more aggressive in undersampling and oversampling the datasets as the global density increases. In Figure 65 below, we observe the average number of observations added or removed by the resampling algorithms at different levels of global density. All of the resampling algorithms have tuning parameters that control the degree of resampling performed. Throughout this thesis the default settings recommended by the relevant literature was used in all simulations.

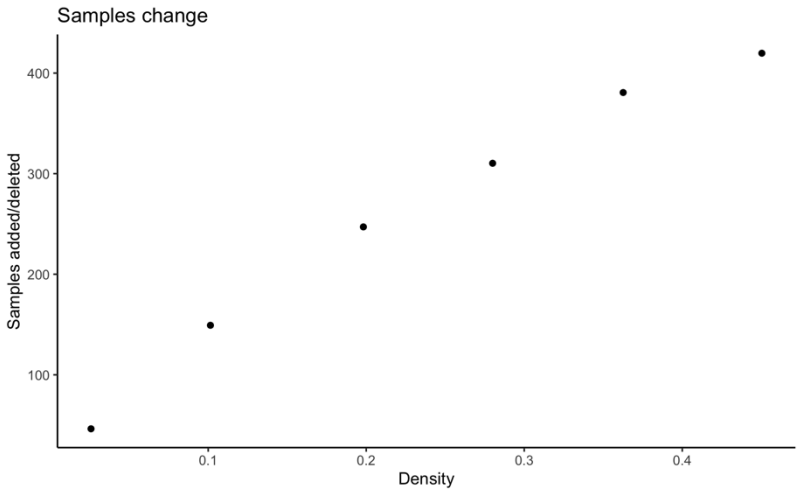


Figure 65: Degree of resampling

We observe that the average number of observations added or removed from the datasets increased as the global density increased. It is reasonable to assume that the increase in performance deviations experienced at larger global densities is due to the increase in observations that have been added or removed from the dataset as the global density increased. Adding or removing more observations from the dataset will have a more polarising effect on MLC performance. The direction of this deviation in performance depends on the quality of the observations generated or removed.

A future avenue of research could be to tune the relevant parameters for the resampling algorithms as the global density increases. This would reduce some of the polarity seen in the changes of performance. At large global densities, changes in performance became large in a positive and a negative sense. Using the tuning parameters of the resampling algorithms to control the trade-off between the degree of resampling and the performance of the algorithms could be an important part of unlocking the potential of the resampling algorithms at larger global densities.

8.4 Computational efficiency

Throughout the experiments the "run time" of the resampling algorithms were recorded using the Sys.time() function in R. The difference in time between the start and end of each resampling algorithm was recorded in seconds. We use the run time as an indication of the computational efficiency. All experiments were performed on a 2019 MacBook Pro, with a 1.4 GHz Quad-Core Intel Core i5 and 8 GB of 2133 MHz LPDDR3 RAM. It should be noted that all the resampling algorithms were coded from scratch, and it is not guaranteed that these run times represent the optimal run time that can be achieved for these algorithms. It is reasonable to assume that the run time can be lowered by improving the efficiency of the code.

In Figure 66 below we observe the run times in seconds for $K = 5$, $K = 10$ and $K = 20$ labels respectively. The run times are averaged across all the experiments.

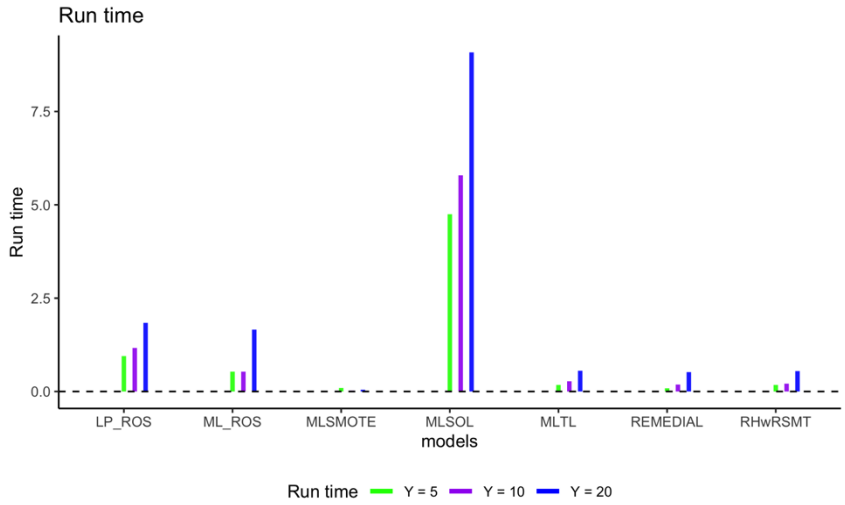


Figure 66: Run times for K = 5, 10 and 20

From Figure 66, we observe that the run times increased drastically for all the algorithms as the number of labels in the dataset increased. The increasing run times as the labels in the dataset increased was a limitation to our thesis since we had to fit resampling algorithms on many different datasets. The computational cost added up and led to experiments taking very long. For this reason, we did not include datasets with more labels in this analysis. This is something that should be considered in future studies. We observe that MLSOL had the longest run times of all the algorithms, and MLSMOTE had the shortest run times. MLTL, REMEDIAL and RHwRSMT were also relatively efficient. LP-ROS and ML-ROS were less efficient than MLTL, REMEDIAL and RHwRSMT, but was still much more efficient than MLSOL.

Referring to Chapter 3, we observed that MLSOL has a very elaborate algorithm involving many different algorithms used together. The algorithm also calculates the k-nearest neighbours to each observation in the dataset, which leads to very long run times. MLSMOTE, on the other hand, avoids iterating over all the observations in the dataset and instead only focuses on the minority observations.

In Figure 67, we observe the average run times of all the resampling algorithms as the global density increases.

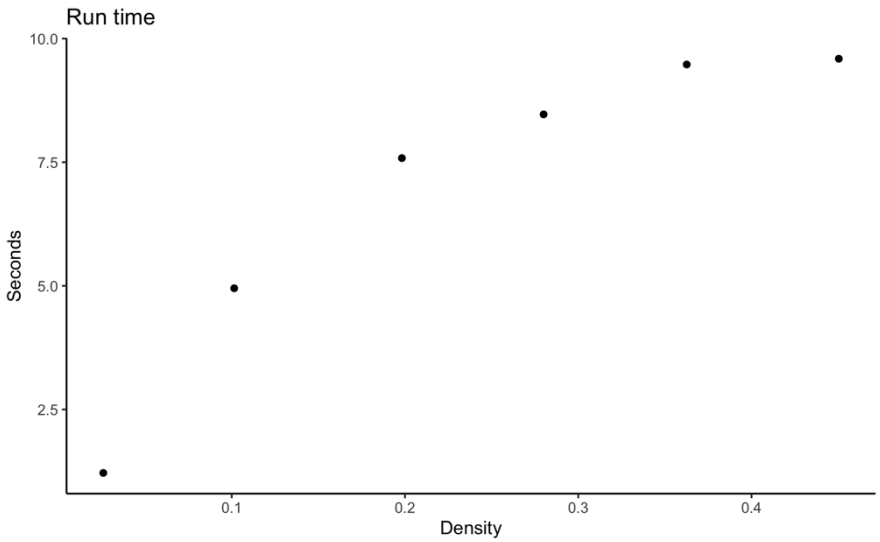


Figure 67: Run time vs global density

We observe that the run times of the algorithms generally increased as the global density increased. Initially, there was a large increase in the run times. However, this increase started to plateau as the global density increased further. It is reasonable to assume that this increase in run time, was due to the increase in observations being added and deleted by the resampling algorithms as the global density increased. The more samples that need to be added or removed, lead to more computations that need to be performed and in turn longer run times for the algorithms.

The run times of the resampling algorithms is something that should be kept in mind when considering it as part of a multi-label analysis. The run-time required for the resampling algorithms will increase as the number of labels increases and as the global density increases. Although changing the number of observations in the datasets fell outside the scope of this thesis, it is reasonable to assume that datasets with a larger number of observations will also lead to longer run times.

8.5 Evaluation metrics

In the literature on multi-label resampling algorithms, most papers make use of label-based evaluation metrics to measure the efficacy of the resampling algorithms as a preproccesing mechanism for multi-label data. The macro-F measure is often chosen as the evaluation metric that is used to rank the resampling algorithms. Using label-based metrics to measure the performance of the resampling algorithms is reasonable since the label-based metrics punish algorithms if they only perform well on the majority classes and perform poorly on the minority classes. This is a desirable characteristic since we would want the resampling algorithm to improve the performance of our MLC models on the minority labels.

However, in this study we have included a broad group of evaluation metrics including example-based, label-based and ranking-based metrics. The results are interesting, since we observe that the resampling algorithms are the most successful on the label-based metrics and less successful on the example-based and ranking-based metrics. A metric such as Hamming loss is extremely important in MLC and is often used to select the most appropriate MLC model. In this thesis we found that the resampling algorithms struggle to make improvements in terms of Hamming loss and often led to a slight reduction in performance in terms of Hamming loss.

Therefore, even though the resampling algorithms might be improving the classification performance on the minority labels, they might lead to a reduction in performance on the other labels and lead to a worse overall MLC model. In Chapter 7, we observed that LP-ROS, ML-ROS and MLSOL were very successful at improving the label-based metrics, however all three models also lead to a consistent reduction in the ranking-based performance measures.

We would recommend including both ranking-based and example-based metrics along with the label-based metrics in future studies to quantify the effect that the resampling algorithms will have on the overall MLC performance of the models and on the performance of the minority labels.

Chapter 9: Conclusion

In this thesis, we set out to address two research goals. These research goals allowed us to learn about the effect that resampling algorithms have on multi-label performance at changing levels of global density, where imbalance manifests itself as a disparity in local label densities. We made use of artificially generated data to mitigate the prevalence of domain bias. The artificial data allowed us to develop the necessary conditions without being dependant on benchmark datasets, which created a controlled environment for our experiments.

The thesis found resampling algorithms to be effective at all levels of global density. Although the algorithms performed well for the "special cases", which is a good representation of the scenarios the resampling algorithms were intended for, where imbalance manifests itself through a sparse data matrix. The improvements in performance made at these levels of global density were consistent but small. When the global density increased, we found that the resampling algorithms' effect on MLC performance was amplified. There were much larger deviations in performance created by the resampling algorithms at larger global densities, positively and negatively. The increase in performance deviations at larger global densities is due to the resampling algorithms becoming more aggressive in how it undersampled and oversampled the datasets. We found that more observations were added and removed from the datasets as the global density increased. This led to larger deviations in performance.

We observed that no resampling algorithm emerged as a dominant algorithm above all others. The use of many evaluation metrics showed that all the algorithms favour different evaluation metrics and can improve performance in different ways. We also found that the performance improvements for one metric were often paired with a reduction in performance for another.

For the example-based metrics we found that the resampling algorithms could not lead to a general improvement in MLC performance. The resampling algorithms were able to improve one or two of the metrics, but not all of them simultaneously. MLSMOTE was the most consistent resampling algorithm for the example-based metrics. The resampling algorithms were the most successful on the label-based evaluation metrics since they were able to consistently make large improvements at all levels of global density. Only REMEDIAL and RHwRSMT were unable to improve the label-based performance at all levels of global density. The most successful resampling algorithms for the label-based metrics were LP-ROS, ML-ROS and MLSOL. The resampling algorithms struggled to improve the ranking-based metrics and generally either retained the same performance as the original data or led to a reduction in performance. MLSMOTE and MLTL were the best performing algorithms and could on occasions lead to an improvement in the ranking-based performance. LP-ROS, ML-ROS and MLSOL were the worst performing resampling algorithms on the ranking-based metrics and led to larger reductions in ranking-based performance as the global density increased.

We found that the resampling algorithms do place a computational burden on the analysis. The computational cost of the resampling algorithms increased as the number of labels

increased. The computational cost also increased as the global density increased, due to an increase in the number of observations being added and removed from the datasets at larger global densities. Therefore, the number of labels and global density should be taken into consideration when resampling algorithms are considered for use in multi-label classification. MLSMOTE was the most efficient resampling algorithm and MLSOL was the least efficient resampling algorithm.

We would recommend expanding the research to include more observations, adding more labels, and including more iterations in the experimental design for future research. Many of the limitations placed on the thesis have been computational. Therefore, more computational power might allow future studies to broaden the scope of the analysis.

We found that the resampling algorithms became more aggressive in undersampling and oversampling as the global density increased. Throughout the experiments the default settings recommended for the algorithms in the literature were used. As a possible avenue of future research we would recommend to study the tuning of these parameters at changing levels of global density. Therefore, we could control the trade-off between the degree of resampling and the potential gain in performance from the resampling algorithms.

For future comparative studies, we would recommend including both ranking-based and example-based evaluation metrics along with the label-based metrics. Including more evaluation metrics will allow the study to show what effect the resampling algorithms have on the overall MLC performance as well as the effect that the resampling algorithms have on the classification of the minority labels.

We would also propose the use of MedianIR as an alternative to the MeanIR in the selection of majority and minority labels as a future area of research. The distribution of IRLbl across labels is often found to have an exponential distribution and is usually skewed to the right as seen in Section 3.1.2. Therefore, the MeanIR > MedianIR in these scenarios. Minority labels are those labels with IRLbl > MeanIR, so using the MedianIR instead of the MeanIR would lead to more of the labels being selected as minority labels. Therefore, more labels will be able to benefit from the oversampling mechanism.

Appendix

$K = 5$

Table 4: Results for ξ = 0 and K = 5

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.2581	0.0758	0.0705	0.0436	0.3880	0.0730	0.9240	0.6414	0.0146
LP_ROS	0.9487	71.4000	0.2577	0.0768	0.0716	0.0417	0.3961	0.0729	0.9233	0.6224	0.0144
ML_ROS	0.5320	40.0000	0.2580	0.0765	0.0713	0.0436	0.3867	0.0749	0.9237	0.6283	0.0146
MLSMOTE	0.0947	4.5000	0.2597	0.0732	0.0680	0.0338	0.3562	0.0596	0.9270	0.7003	0.0165
MLSOL	4.7485	210.0000	0.2573	0.0782	0.0735	0.0460	0.4509	0.0798	0.9210	0.5413	0.0138
RHwRSMT	0.1764	13.2000	0.2581	0.0757	0.0701	0.0739	0.3787	0.0730	0.9243	0.6555	0.0149
REMEDIAL	0.0857	8.7000	0.2588	0.0750	0.0696	0.0457	0.3587	0.0747	0.9250	0.6578	0.0151
MLTL	0.1764	4.4000	0.2575	0.0772	0.0718	0.0462	0.4077	0.0760	0.9230	0.5742	0.0143

Table 5: Results for ξ = 0.1 and K = 5

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.2339	0.1777	0.1310	0.1107	0.2374	0.1250	0.8223	1.7406	0.1049
LP_ROS	1.1258	77.1000	0.2380	0.1708	0.1301	0.1501	0.2742	0.1761	0.8280	1.7311	0.1047
ML_ROS	1.0368	70.8000	0.2376	0.1718	0.1305	0.1546	0.2743	0.1766	0.8263	1.7345	0.1047
MLSMOTE	3.0182	126.5000	0.2331	0.1793	0.1314	0.1258	0.2382	0.1236	0.8207	1.7671	0.1061
MLSOL	5.4402	210.0000	0.2379	0.1720	0.1297	0.1479	0.2830	0.1844	0.8267	1.7962	0.1089
RHwRSMT	3.6190	199.1000	0.2347	0.1753	0.1281	0.1382	0.2456	0.1475	0.8247	1.7855	0.1082
REMEDIAL	0.7237	72.6000	0.2346	0.1757	0.1275	0.1142	0.2313	0.1117	0.8243	1.7960	0.1098
MLTL	0.5394	33.2000	0.2331	0.1797	0.1321	0.1145	0.2431	0.1336	0.8203	1.7259	0.1031

Table 6: Results for ξ = 0.2 and K = 5

	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.2765	0.3423	0.2139	0.2282	0.2272	0.1740	0.6570	1.9710	0.1795
LP_ROS	1.1713	77.1000	0.2765	0.3435	0.2361	0.3131	0.2847	0.2630	0.6530	1.9617	0.1784
ML_ROS	1.0805	71.9000	0.2765	0.3433	0.2354	0.3043	0.2842	0.2619	0.6533	1.9618	0.1782
MLSMOTE	5.4579	239.7000	0.2734	0.3499	0.2225	0.2314	0.2315	0.1787	0.6500	1.9744	0.1800
MLSOL	5.6215	210.0000	0.2849	0.3271	0.2206	0.2760	0.2670	0.2448	0.6677	2.0270	0.1884
RHwRSMT	7.3247	396.0000	0.2790	0.3370	0.2095	0.2248	0.2328	0.1892	0.6630	2.0159	0.1869
REMEDIAL	1.6122	156.3000	0.2773	0.3413	0.2125	0.1905	0.2185	0.1573	0.6587	2.0182	0.1886
MLTL	1.2004	123.8000	0.2745	0.3475	0.2193	0.2900	0.2306	0.1796	0.6520	1.9869	0.1819

Table 7: Results for ξ = 0.3 and K = 5

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.2919	0.4929	0.3273	0.2625	0.2566	0.2218	0.4993	1.9388	0.1844
LP_ROS	1.0383	71.5000	0.3014	0.4745	0.3463	0.3679	0.3058	0.2981	0.5100	1.9944	0.1929
ML_ROS	1.0740	71.7000	0.2993	0.4795	0.3468	0.3741	0.3045	0.2969	0.5060	1.9865	0.1903
MLSMOTE	5.9213	254.8000	0.2919	0.4964	0.3263	0.2949	0.2544	0.2199	0.4980	1.9576	0.1876
MLSOL	5.6220	210.0000	0.3023	0.4720	0.3489	0.3917	0.3163	0.3085	0.5137	2.0033	0.1955
RHwRSMT	8.4278	471.0000	0.2993	0.4950	0.2822	0.3053	0.2197	0.1990	0.5050	1.9687	0.1891
REMEDIAL	2.2437	216.2000	0.3019	0.4887	0.2779	0.2763	0.2097	0.1832	0.5113	1.9969	0.1969
MLTL	1.9276	215.6000	0.2945	0.4900	0.3306	0.2673	0.2586	0.2175	0.5007	1.9862	0.1918

Table 8: Results for ξ = 0.5 and K = 5

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.3321	0.5556	0.3700	0.4660	0.3064	0.2988	0.4313	2.2494	0.2189
LP_ROS	1.0347	69.5000	0.3387	0.5335	0.3940	0.4723	0.3632	0.3756	0.4397	2.2855	0.2301
ML_ROS	1.0846	71.7000	0.3377	0.5330	0.3949	0.4682	0.3640	0.3751	0.4437	2.2807	0.2298
MLSMOTE	10.1565	436.5000	0.3360	0.5559	0.3460	0.4216	0.2773	0.2642	0.4330	2.2651	0.2231
MLSOL	5.3591	210.0000	0.3387	0.5314	0.4207	0.4860	0.3927	0.3938	0.4393	2.2807	0.2301
RHwRSMT	13.2540	684.2000	0.3460	0.5616	0.2867	0.4040	0.2146	0.2014	0.4383	2.3327	0.2391
REMEDIAL	2.5390	247.7000	0.3633	0.5188	0.2607	0.3167	0.2063	0.2100	0.4813	2.3933	0.2616
MLTL	2.0898	269.6000	0.3377	0.5411	0.4013	0.4279	0.3473	0.3283	0.4257	2.2673	0.2229

$K = 10$

Table 9: Results for ξ = 0 and K = 10

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.1541	0.0733	0.0595	0.0244	0.1815	0.0401	0.9267	1.3891	0.0239
LP_ROS	1.1663	69.8000	0.1530	0.0763	0.0628	0.0392	0.2020	0.0596	0.9237	1.3400	0.0224
ML_ROS	0.5319	33.3000	0.1527	0.0770	0.0638	0.0514	0.2076	0.0661	0.9230	1.3126	0.0220
MLSMOTE	0.0225	0.6000	0.1533	0.0750	0.0612	0.0496	0.1807	0.0511	0.9250	1.3683	0.0235
MLSOL	5.7905	210.0000	0.1540	0.0736	0.0623	0.0414	0.2193	0.0644	0.9253	1.3541	0.0226
RHwRSMT	0.2094	17.2000	0.1532	0.0750	0.0609	0.0324	0.1770	0.0433	0.9250	1.4009	0.0239
REMEDIAL	0.1860	16.6000	0.1527	0.0770	0.0631	0.0259	0.1758	0.0402	0.9230	1.3732	0.0232
MLTL	0.2738	5.2000	0.1530	0.0761	0.0625	0.0373	0.1809	0.0445	0.9237	1.3517	0.0229

Table 10: Results for ξ = 0.1 and K = 10

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.1597	0.2042	0.1226	0.0487	0.1034	0.0526	0.7957	4.0423	0.1547
LP_ROS	1.9422	86.0000	0.1682	0.1844	0.1168	0.1422	0.1359	0.1153	0.8147	4.2845	0.1651
ML_ROS	1.4911	72.4000	0.1678	0.1852	0.1176	0.1486	0.1353	0.1131	0.8137	4.2811	0.1645
MLSMOTE	4.4416	144.8000	0.1593	0.2062	0.1235	0.0522	0.1056	0.0553	0.7937	4.0634	0.1545
MLSOL	7.4379	210.0000	0.1676	0.1829	0.1121	0.1296	0.1237	0.1016	0.8167	4.3074	0.1667
RHwRSMT	6.1115	254.7000	0.1603	0.2007	0.1198	0.0535	0.1028	0.0555	0.7993	4.1359	0.1587
REMEDIAL	1.6830	109.9000	0.1607	0.1987	0.1171	0.0420	0.0995	0.0494	0.8013	4.1247	0.1591
MLTL	1.4625	64.5000	0.1603	0.2019	0.1208	0.0586	0.1043	0.0559	0.7977	4.0444	0.1542

Table 11: Results for ξ = 0.2 and K = 10

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.2311	0.3501	0.1612	0.0727	0.1023	0.0650	0.6490	4.8662	0.2535
LP_ROS	4.0143	169.1000	0.2497	0.3134	0.1891	0.2743	0.1873	0.2053	0.6827	5.1525	0.2782
ML_ROS	1.6250	72.1000	0.2492	0.3151	0.1882	0.2796	0.1876	0.2059	0.6777	5.1900	0.2807
MLSMOTE	12.4224	375.7000	0.2311	0.3502	0.1614	0.1008	0.1003	0.0607	0.6497	4.8544	0.2530
MLSOL	7.8638	210.0000	0.2421	0.3195	0.1684	0.2488	0.1401	0.1489	0.6760	5.0577	0.2686
RHwRSMT	16.6701	600.9000	0.2324	0.3447	0.1561	0.0946	0.1017	0.0700	0.6553	4.9813	0.2647
REMEDIAL	3.6521	225.2000	0.2317	0.3483	0.1561	0.0843	0.1009	0.0663	0.6517	4.9800	0.2637
MLTL	2.9713	172.0000	0.2316	0.3459	0.1612	0.1144	0.1035	0.0680	0.6540	4.8773	0.2550

Table 12: Results for ξ = 0.3 and K = 10

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.2851	0.4345	0.1669	0.1487	0.1078	0.0836	0.5650	5.4273	0.2941
LP_ROS	5.1427	213.0000	0.3073	0.3730	0.2494	0.3705	0.2539	0.2841	0.6097	5.7527	0.3252
ML_ROS	1.6354	72.6000	0.3074	0.3718	0.2492	0.3732	0.2529	0.2841	0.6110	5.7436	0.3252
MLSMOTE	19.4904	564.6000	0.2859	0.4353	0.1629	0.1057	0.1028	0.0768	0.5647	5.4208	0.2931
MLSOL	7.9937	210.0000	0.2926	0.4003	0.2120	0.3612	0.1820	0.2037	0.5850	5.5906	0.3081
RHwRSMT	24.3165	825.4000	0.2903	0.4133	0.1526	0.1467	0.0995	0.0855	0.5867	5.5965	0.3089
REMEDIAL	4.2993	260.8000	0.2884	0.4227	0.1561	0.1320	0.1011	0.0853	0.5773	5.6276	0.3121
MLTL	4.0975	251.7000	0.2862	0.4314	0.1679	0.1710	0.1099	0.0847	0.5683	5.4704	0.2966

Table 13: Results for ξ = 0.4 and K = 10

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.3352	0.5670	0.2823	0.3405	0.1895	0.1831	0.4090	6.0124	0.3031
LP_ROS	6.2887	251.1000	0.3603	0.4911	0.3994	0.4413	0.3625	0.3814	0.4527	6.2668	0.3332
ML_ROS	1.7955	72.9000	0.3611	0.4902	0.3978	0.4387	0.3619	0.3808	0.4533	6.2664	0.3339
MLSMOTE	23.2429	804.5000	0.3389	0.5758	0.2598	0.2455	0.1642	0.1487	0.4120	6.0202	0.3039
MLSOL	8.1470	210.0000	0.3422	0.5258	0.3629	0.4530	0.2974	0.3158	0.4267	6.1238	0.3139
RHwRSMT	28.7193	1111.5000	0.3479	0.5825	0.1771	0.1669	0.1012	0.0934	0.4170	6.1735	0.3207
REMEDIAL	5.5473	307.0000	0.3509	0.5664	0.1744	0.1660	0.1002	0.0974	0.4333	6.2197	0.3280
MLTL	5.6788	359.7000	0.3381	0.5489	0.3259	0.3760	0.2346	0.2277	0.4090	6.0405	0.3065

Table 14: Results for ξ = 0.5 and K = 10

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.3553	0.6203	0.5523	0.4519	0.4125	0.3715	0.2497	6.2133	0.2673
LP_ROS	7.9035	231.4000	0.3707	0.5901	0.5598	0.5097	0.4758	0.4730	0.3097	6.5074	0.3056
ML_ROS	3.2084	72.6000	0.3717	0.5878	0.5593	0.5052	0.4751	0.4720	0.3060	6.5067	0.3050
MLSMOTE	24.7175	710.7000	0.3548	0.6281	0.5334	0.3989	0.3888	0.3521	0.2490	6.2079	0.2675
MLSOL	11.8588	210.0000	0.3605	0.6059	0.5696	0.5062	0.4631	0.4480	0.2773	6.3338	0.2823
RHwRSMT	29.7924	982.0000	0.3973	0.7369	0.2110	0.2157	0.1120	0.1129	0.2567	6.3212	0.2815
REMEDIAL	5.2564	271.3000	0.3973	0.7234	0.2270	0.2288	0.1229	0.1251	0.2640	6.3346	0.2832
MLTL	7.0586	407.0000	0.3696	0.5727	0.6827	0.4580	0.5527	0.4546	0.2623	6.2258	0.2700

$K = 20$

Table 15: Results for ξ = 0 and K = 20

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.0780	0.0790	0.0598	0.0145	0.0732	0.0195	0.9210	3.3508	0.0381
LP_ROS	1.8389	67.4000	0.0784	0.0792	0.0601	0.0439	0.0933	0.0412	0.9207	3.5662	0.0404
ML_ROS	1.6572	64.1000	0.0787	0.0766	0.0578	0.0200	0.0805	0.0281	0.9233	3.5927	0.0405
MLSMOTE	0.0478	0.6000	0.0779	0.0800	0.0607	0.0161	0.0772	0.0210	0.9200	3.3771	0.0385
MLSOL	9.0836	210.0000	0.0790	0.0750	0.0556	0.0310	0.0858	0.0348	0.9247	3.7319	0.0427
RHwRSMT	0.5465	27.6000	0.0779	0.0793	0.0604	0.0181	0.0763	0.0205	0.9207	3.3465	0.0381
REMEDIAL	0.5199	27.0000	0.0776	0.0817	0.0626	0.0217	0.0769	0.0200	0.9183	3.3375	0.0380
MLTL	0.5558	10.2000	0.0785	0.0757	0.0567	0.0163	0.0716	0.0194	0.9243	3.4328	0.0392

Table 16: Results for ξ = 0.1 and K = 20

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.1278	0.2323	0.1001	0.0288	0.0507	0.0262	0.7677	9.4677	0.2327
LP_ROS	7.5628	212.4000	0.1441	0.1720	0.0943	0.1763	0.1011	0.1131	0.8260	10.3305	0.2645
ML_ROS	2.4728	74.8000	0.1442	0.1730	0.0945	0.1788	0.0997	0.1116	0.8257	10.3661	0.2654
MLSMOTE	20.7488	424.6000	0.1278	0.2320	0.0985	0.0210	0.0501	0.0240	0.7680	9.5077	0.2336
MLSOL	11.9140	210.0000	0.1340	0.2092	0.0989	0.1672	0.0758	0.0787	0.7897	10.0021	0.2510
RHwRSMT	26.9769	619.1000	0.1279	0.2310	0.0988	0.0377	0.0506	0.0281	0.7690	9.6928	0.2407
REMEDIAL	4.9846	194.5000	0.1279	0.2307	0.0992	0.0206	0.0496	0.0247	0.7693	9.5962	0.2363
MLTL	4.0855	114.4000	0.1281	0.2292	0.0978	0.0223	0.0502	0.0251	0.7710	9.4811	0.2345

Table 17: Results for ξ = 0.2 and K = 20

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.2069	0.3490	0.0966	0.0549	0.0517	0.0387	0.6510	11.5695	0.3162
LP_ROS	13.9112	363.3000	0.2382	0.2826	0.1657	0.2445	0.1510	0.1784	0.7067	12.5145	0.3603
ML_ROS	2.5772	73.9000	0.2394	0.2783	0.1648	0.2431	0.1505	0.1776	0.7090	12.5189	0.3607
MLSMOTE	31.2447	709.2000	0.2087	0.3310	0.0886	0.0561	0.0490	0.0380	0.6690	11.5431	0.3166
MLSOL	12.4594	210.0000	0.2130	0.3203	0.1132	0.2293	0.0786	0.0947	0.6777	11.9840	0.3343
RHwRSMT	33.2199	987.8000	0.2086	0.3320	0.0905	0.0703	0.0495	0.0391	0.6680	11.8325	0.3276
REMEDIAL	7.7566	278.6000	0.2077	0.3410	0.0928	0.0549	0.0504	0.0375	0.6590	11.8140	0.3268
MLTL	8.1275	206.5000	0.2075	0.3433	0.0959	0.0508	0.0503	0.0390	0.6567	11.6032	0.3183

Table 18: Results for ξ = 0.3 and K = 20

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.2725	0.5078	0.1422	0.0899	0.0667	0.0573	0.4773	12.8880	0.3120
LP_ROS	21.0749	520.0000	0.3246	0.3777	0.2824	0.3173	0.2463	0.2714	0.5867	13.9870	0.3775
ML_ROS	2.7116	74.6000	0.3239	0.3764	0.2781	0.3168	0.2425	0.2687	0.5870	13.9766	0.3769
MLSMOTE	34.6270	909.7000	0.2720	0.5118	0.1483	0.0589	0.0675	0.0532	0.4763	12.8915	0.3121
MLSOL	12.8526	210.0000	0.2814	0.4685	0.2064	0.3310	0.1342	0.1560	0.4970	13.2827	0.3331
RHwRSMT	17.5373	1198.3000	0.2737	0.5143	0.1114	0.0805	0.0501	0.0451	0.4857	13.2571	0.3297
REMEDIAL	8.4230	288.6000	0.2730	0.5208	0.1140	0.0537	0.0504	0.0434	0.4787	13.0527	0.3180
MLTL	12.7550	337.8000	0.2730	0.5049	0.1484	0.1154	0.0724	0.0652	0.4890	12.9398	0.3136

Table 19: Results for ξ = 0.4 and K = 20

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.3339	0.5708	0.2361	0.1737	0.1330	0.1178	0.3953	13.9758	0.3166
LP_ROS	26.7643	619.2000	0.3826	0.4539	0.3857	0.3844	0.3291	0.3475	0.4923	15.0136	0.3818
ML_ROS	2.8790	73.5000	0.3838	0.4507	0.3837	0.3810	0.3262	0.3448	0.4943	15.0162	0.3825
MLSMOTE	29.5702	1085.0000	0.3344	0.5670	0.2349	0.1480	0.1322	0.1166	0.3920	13.9707	0.3162
MLSOL	12.9062	210.0000	0.3422	0.5269	0.3135	0.3778	0.2164	0.2315	0.4087	14.2267	0.3320
RHwRSMT	15.4582	1357.4000	0.3396	0.6099	0.1073	0.0864	0.0505	0.0431	0.3897	14.2488	0.3288
REMEDIAL	8.1854	272.4000	0.3402	0.6037	0.1068	0.0894	0.0507	0.0456	0.3970	14.1501	0.3241
MLTL	16.7204	412.8000	0.3345	0.5467	0.2891	0.2382	0.1778	0.1590	0.3920	14.0350	0.3185

Table 20: Results for ξ = 0.5 and K = 20

			Example based			Label based			Ranking based		
	Time	Samples change	Hamming-loss	Precision	Recall	Macro-Precision	Macro-Recall	Macro-F1	One-Error	Coverage	Ranking-Loss
Original	0.0000	0.0000	0.3594	0.6258	0.4794	0.3349	0.3304	0.2866	0.2277	15.0227	0.2921
LP_ROS	31.0439	666.3000	0.4022	0.5522	0.5101	0.4716	0.4404	0.4505	0.3157	16.0249	0.3622
ML_ROS	3.2021	72.7000	0.4027	0.5508	0.5111	0.4706	0.4415	0.4505	0.3157	16.0182	0.3623
MLSMOTE	11.9473	1583.4000	0.3616	0.6252	0.4715	0.3185	0.3236	0.2780	0.2293	15.0050	0.2913
MLSOL	15.5112	210.0000	0.3674	0.6000	0.5166	0.4881	0.3987	0.3907	0.2450	15.2769	0.3070
RHwRSMT	7.6253	1865.5000	0.4041	0.7590	0.1437	0.1325	0.0753	0.0729	0.2290	15.3115	0.3083
REMEDIAL	9.5214	282.1000	0.3999	0.7436	0.1718	0.1512	0.0920	0.0949	0.2363	15.2758	0.3054
MLTL	22.9402	454.7000	0.3880	0.5584	0.6692	0.3936	0.5376	0.4158	0.2553	15.1480	0.3021

References

Bernardini, F. C., Silva, R. B. da, Rodovalho, R. M., & Meza, E. B. M. (2014). Cardinality and Density Measures and Their Influence to Multi-Label Learning Methods. *L&NLM*, *12*(1), 53–71. 10.21528/lnlm-vol12-no1-art4

Boutell, M. R., Luo, J., Shen, X., & Brown, C. M. (2004). Learning multi-label scene classification. *Pattern Recognition*, *37*(9), 1757–1771. 10.1016/j.patcog.2004.03.009

Charte, F., & Charte, D. (2015). Working with Multilabel Datasets in R: The mldr Package. *The R Journal*, *7*(2), 149. 10.32614/rj-2015-027

Charte, F., Rivera, A., del Jesus, M. J., & Herrera, F. (2015a). Resampling Multilabel Datasets by Decoupling Highly Imbalanced Labels. *International Conference on Hybrid Artificial Intelligence Systems*, 489–501. 10.1007/978-3-319-19644-2_41

Charte, F., Rivera, A. J., del Jesus, M. J., & Herrera, F. (2014). MLeNN: A First Approach to Heuristic Multilabel Undersampling. International Conference on Intelligent Data Engineering and Automated Learning, 1–9. 10.1007/978-3-319-10840-7_1

Charte, F., Rivera, A. J., del Jesus, M. J., & Herrera, F. (2015b). Addressing imbalance in multilabel classification: Measures and random resampling algorithms. *Neurocomputing*, *163*, 3–16. 10.1016/j.neucom.2014.08.091

Charte, F., Rivera, A. J., del Jesus, M. J., & Herrera, F. (2015c). MLSMOTE: Approaching imbalanced multilabel learning through synthetic instance generation. *Knowledge-Based Systems*, *89*, 385–397. 10.1016/j.knosys.2015.07.019

Charte, F., Rivera, A. J., del Jesus, M. J., & Herrera, F. (2019a). Dealing with difficult minority labels in imbalanced mutilabel data sets. *Neurocomputing*, *326–327*, 39–53. 10.1016/j.neucom.2016.08.158

Charte, F., Rivera, A. J., del Jesus, M. J., & Herrera, F. (2019b). REMEDIAL-HwR: Tackling multilabel imbalance through label decoupling and data resampling hybridization. *Neurocomputing*, *326–327*, 110–122. 10.1016/j.neucom.2017.01.118

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Jair* , *16*, 321–357. 10.1613/jair.953

Clare, A., & King, R. D. (2001). *Knowledge Discovery in Multi-label Phenotype Data*. *European Conference on Principles of Data Mining and Knowledge Discovery*, 42–53. 10.1007/3-540-44794-6_4

CRAN - Package E1071. Retrieved September 16, 2021, from https://cran.r-project.org/web/packages/e1071/index.html

CRAN - Package Mldr.datasets. Retrieved August 30, 2021, from https://cran.r-project.org/web/packages/mldr.datasets/index.html

Elisseeff, A., & Weston, J. (2002). A kernel method for multi-labelled classification. Advances in Neural Information Processing Systems 14: Proceedings of the 2001 Conference. 10.7551/mitpress/1120.003.0092

Everton Alvares-Cherman, Jean Metz, Maria Carolina Monard (2012). Incorporating label dependency into the binary relevance framework for multi-label classification, *Expert Systems with Applications, Volume 39, Issue 2*, 1647-1655, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2011.06.056.

Flags: Dataset With Features Correspoinding To World Flags In Mldr.datasets: R Ultimate Multilabel Dataset Repository. (2019, January 17). https://rdrr.io/cran/mldr.datasets/man/flags.html

Fürnkranz, J., Hüllermeier, E., Loza Mencía, E., & Brinker, K. (2008). Multilabel classification via calibrated label ranking. *Mach Learn*, *73*(2), 133–153. 10.1007/s10994-008-5064-8

Godbole, S., & Sarawagi, S. (2004). *Discriminative Methods for Multi-labeled Classification*. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 22–30. 10.1007/978-3-540-24775-3_5

Goncalves, E. C., Plastino, A., & Freitas, A. A. (2013). A Genetic Algorithm for Optimizing the Label Ordering in Multi-label Classifier Chains. 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, 469–476. 10.1109/ictai.2013.76

Hastie, T. (2013). *The Elements of Statistical Learning* (2nd ed., pp. 417–455). Springer Science & Business Media.

Herrera, F. (2016). *Multilabel Classification* (pp. 50–51). Springer.

Hüllermeier, E., Fürnkranz, J., Cheng, W., & Brinker, K. (2008). Label ranking by learning pairwise preferences. *Artificial Intelligence*, *172*(16–17), 1897–1916. 10.1016/j.artint.2008.08.002

James, G. (2013). *An Introduction to Statistical Learning* (7th ed., pp. 337–355). Springer Science & Business Media.

Keren, Betty & Kalech, Meir & Rokach, & Lior. (2011). Model-Based Diagnosis with Multi-Label Classification.: Vol. 22nd International Workshop on Principles of Diagnosis.

Kubat, M. (2017). *An Introduction to Machine Learning*. Springer.

Langlog: Dataset With Data From The Language Forum Discussion . (2019, May 16). https://rdrr.io/github/fcharte/mldr.datasets/man/langlog.html

Liu, B., & Tsoumakas, G. (2020). Synthetic Oversampling of Multi-label Data Based on Local Label Distribution. Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 180–193. 10.1007/978-3-030-46147-8_11

Madjarov, G., Kocev, D., Gjorgjevikj, D., & Džeroski, S. (2012). An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition*, *45*(9), 3084–3104. 10.1016/j.patcog.2012.03.004

Min-Ling Zhang, Zhi-Hua Zhou. (2006). Multilabel Neural Networks with Applications to Functional Genomics and Text Categorization. *IEEE Trans. Knowl. Data Eng.*, *18*(10), 1338–1351. 10.1109/tkde.2006.162

Montañes, E., Senge, R., Barranquero, J., Ramón Quevedo, J., José del Coz, J., & Hüllermeier, E. (2014). Dependent binary relevance models for multi-label classification. *Pattern Recognition*, *47*(3), 1494–1508. 10.1016/j.patcog.2013.09.029

Moyano, J. (2021). *Multi-Label Classification Dataset Repository – Knowledge Discovery And Intelligent Systems – KDIS – University Of Córdoba*. https://www.uco.es/kdis/mllresources/

Pereira, R. M., Costa, Y. M. G., & Silla Jr., C. N. (2020). MLTL: A multi-label approach for the Tomek Link undersampling algorithm. *Neurocomputing*, *383*, 95–105. 10.1016/j.neucom.2019.11.076

Rauber T.W., Mello L.H., Rocha V.F., Luchi D., Varejão F.M. (2014) Recursive Dependent Binary Relevance Model for Multi-label Classification. In: Bazzan A., Pichara K. (eds) Advances in Artificial Intelligence -- IBERAMIA 2014. IBERAMIA 2014. Lecture Notes in Computer Science, vol 8864. Springer, Cham. https://doi.org/10.1007/978-3-319-12027-0_17

Read, J., Bifet, A., Holmes, G., & Pfahringer, B. (2012). Scalable and efficient multi-label classification for evolving data streams. *Mach Learn*, *88*(1–2), 243–272. 10.1007/s10994-012-5279-6

Read J., B. Pfahringer and G. Holmes, (2008). Multi-label Classification Using Ensembles of Pruned Sets. *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 995-1000. doi: 10.1109/ICDM.2008.74.

Read J., Pfahringer B., Holmes G., Frank E. (2009) Classifier Chains for Multi-label Classification. In: Buntine W., Grobelnik M., Mladenić D., Shawe-Taylor J. (eds) Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2009. Lecture Notes in Computer Science, vol 5782. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-04174-7_17

Read, J., Pfahringer, B., Holmes, G., & Frank, E. (2011). Classifier chains for multi-label classification. *Mach Learn*, *85*(3), 333–359. 10.1007/s10994-011-5256-5

Read, J., Pfahringer, B., Holmes, G., & Frank, E. (2021). Classifier Chains: A Review and Perspectives. Journal of Artificial Intelligence Research, *70*, 683–718. 10.1613/jair.1.12376

Rivolli, A., & Carvalho, A., C. ,P. ,L. ,F. ,de. (2019). The utiml Package: Multi-label Classification in R. *The R Journal*, *10*(2), 24. 10.32614/rj-2018-041

Schapire, R.E., Singer, Y. BoosTexter: A Boosting-based System for Text Categorization. *Machine Learning* **39,** 135–168 (2000). https://doi.org/10.1023/A:1007649029923

Synthetic Dataset Generator For Multi-label Learning (Mldatagen). Retrieved June 23, 2021, from http://sites.labic.icmc.usp.br/mldatagen/

Thomas G. Dietterich, Suzanna Becker, Zoubin Ghahramani (2002). A kernel method for multi-labelled classification, *Advances in Neural Information Processing Systems 14: Proceedings of the 2001 Conference*, 2002

Tomás, J. T., Spolaôr, N., Cherman, E. A., & Monard, M. C. (2014). A Framework to Generate Synthetic Multi-label Datasets. *Electronic Notes in Theoretical Computer Science*, *302*, 155–176. 10.1016/j.entcs.2014.01.025

Trohidis, K., Tsoumakas, G., Kalliris, G., & Vlahavas, I. (2011). Multi-label classification of music by emotion. *J AUDIO SPEECH MUSIC PROC.*, *2011*(1). 10.1186/1687-4722-2011-426793

Tsoumakas, G., & Katakis, I. (2007). Multi-Label Classification: An Overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3), 1-13. http://doi.org/10.4018/jdwm.2007070101

Tsoumakas G., Katakis I., Vlahavas I. (2009) Mining Multi-label Data. In: Maimon O., Rokach L. (eds) Data Mining and Knowledge Discovery Handbook. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09823-4_34

Tsoumakas, G., Katakis, I., & Vlahavas, I. (2011). Random k-Label-sets for Multilabel Classification. *IEEE Trans. Knowl. Data Eng.*, *23*(7), 1079–1089. 10.1109/tkde.2010.164

Tsoumakas, G., Eleftherios, S., Jozef, V., Ioannis, V. (2011), MULAN: A Java Library for Multi-Label Learning, Journal of Machine Learning Research 12 (2011) 2411-2414

Turnbull, D., Barrington, L., Torres, D., & Lanckriet, G. (2008). Semantic Annotation and Retrieval of Music and Sound Effects. *IEEE Trans. Audio Speech Lang. Process.*, *16*(2), 467–476. 10.1109/tasl.2007.913750

Wieczorkowska A., Synak P., Raś Z.W. (2006). Multi-Label Classification of Emotions in Music. In: Kłopotek M.A., Wierzchoń S.T., Trojanowski K. (eds) Intelligent Information Processing and Web Mining. Advances in Soft Computing, vol 35. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-33521-8_30

Write Your Best With Grammarly. Retrieved August 20, 2021, from https://grammarly.com

Zafer Barutcuoglu, Robert E. Schapire, Olga G. Troyanskaya (2006), Hierarchical multi-label prediction of gene function, *Bioinformatics*, Volume 22, Issue 7, 1 April 2006, Pages 830–836, https://doi.org/10.1093/bioinformatics/btk048

Yang J., Jiang L., Wang C. and Xie J.. (2004). Multi-label Emotion Classification for Tweets, *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, 2014, pp. 424-428, doi: 10.1109/ICTAI.2014.71.

Yang S., Kim S. and Man Ro Y. (2007). Semantic Home Photo Categorization, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 3, pp. 324-335, March 2007, doi: 10.1109/TCSVT.2007.890829.

Zhang Yi, Street W. N. and Burer S. (2005). Sharing classifiers among ensembles from related problem domains, *Fifth IEEE International Conference on Data Mining (ICDM'05)*, 2005, pp. 8 pp.-, doi: 10.1109/ICDM.2005.131.

Zhang, M.-L., Li, Y.-K., Liu, X.-Y., & Geng, X. (2018). Binary relevance for multi-label learning: an overview. *Front. Comput. Sci.*, *12*(2), 191–202. 10.1007/s11704-017-7031-7
Zhang, M.-L., & Zhou, Z.-H. (2007). ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, *40*(7), 2038–2048. 10.1016/j.patcog.2006.12.019

Zhou, L., Zheng, X., Yang, D., Wang, Y., Bai, X., & Ye, X. (2021). Application of multi-label classification models for the diagnosis of diabetic complications. *BMC Med Inform Decis Mak*, *21*(1). 10.1186/s12911-021-01525-7

Code Appendix:

Simulation code:

```
generate.data = function (N,p,pnoise,K,rho,pvek,Amat,signal)
{
Amat=matrix(Amat,nrow=p)
n=length(pvek)
theta=qnorm(pvek)
sigmax=matrix(0.5,p,p)
diag(sigmax)=1
apvek1=NULL
xmat=matrix(0,N,p+pnoise)
ymat=matrix(0,N,K)
apmat=matrix(0,K,K)
gem=rep(0,p)
sign.rho=sign(rho)

for (j in 1:p) apvek1[j]=sum(Amat[j,]*pvek)

for (k1 in 1:K) for (k2 in 1:K) {
          term1=rho*sqrt(pvek[k1]*(1-pvek[k1])*pvek[k2]*(1-pvek[k2]))
          term2=pvek[k1]*pvek[k2]
          term3= t(Amat[,k1])%*%Amat[,k2]
          apmat[k1,k2]=term3*(term1+term2)
     }

antwoord=sum(apvek1)+sum(apmat)-sum(diag(apmat))-(sum(apvek1^2))
c=sqrt(signal/antwoord)

itel = 0
while (itel < N)
     {
     if (rho>=0) {

     eps0=rnorm(1)
     eps=rnorm(n)
     u=rbinom(n,1,sqrt(rho))
     z=u*eps0+(1-u)*eps
     yvek=(z<=theta)+0
}

     if (rho<0) {

     eps=rnorm(n)
     z=eps
     u=rbinom(n,1,abs(rho))
     for (j in 2:n)
     {
          z[j]=sign.rho*u[j]*z[j-1]+(1-u[j])*eps[j]
     }
     yvek=(z<=theta)+0
}

if (sum(yvek)>=0)
{
itel=itel+1
```

```
for (j in 1:p) gem[j]=c*sum(Amat[j,]*yvek)

     xmat[itel,]=c(mvrnorm(1,gem,sigmax),mvrnorm(1,rep(0,pnoise),diag(pnois
e)))
     ymat[itel,]=yvek
}
}
return(list("Predictors" = xmat,"Labels" = ymat,antwoord,c))
}

#Function to return some basic information about the dataset
getInfo = function(D){
  output = list("NumOfInstances" = D$measures$num.instances, "NumOfLabels"
= NROW(D$labels), "Cardinality" = D$measures$cardinality, "Density" =
D$measures$density, "MeanIR" = D$measures$meanIR, "SCUMBLE" =
D$measures$scumble, "Percentage min labels" = 100*mean(D$labels$IRLbl >
D$measures$meanIR
))
  return(output)
}

#Create an mldr dataset from simlation code
create.mldr = function(data){
df = as.data.frame(cbind(data$Predictors, data$Labels))
new.mldr = mldr_from_dataframe(dataframe = df, labelIndices = (NCOL(df) -
ncol(data$Labels) + 1):NCOL(df))
return(new.mldr)
}

#Function to create a vector of densities
create.density.vector = function(K, num.majority, Xi.mins, Xi.majs){
  dens.vec = rep(0, K)
  j = 1
   if(Xi.mins > 0){
  omega.mins = Xi.mins/3
  alpha.mins = (1 - Xi.mins)*-180
  }else{
    omega.mins = 0.005
    alpha.mins = 50
  }
  while(j <= K){
    val = rsn(n = 1, xi = Xi.mins,omega = omega.mins, alpha = alpha.mins)
    if(val>0){
    dens.vec[j] = val
    j = j + 1
    }
    else{
      print(paste("reject", j))
    }
  }
   if(Xi.majs > 0){
  omega.majs = 0.05 + Xi.majs/5
  }else{
    omega.majs = 0.05
  }
  dens.vec
  if(num.majority > 0){
    index.taken = c()
  while(num.majority>0){
   index = round(runif(n = 1, min = 1, max = K),0)
   if(index %in% index.taken == FALSE){
```

```
     val = rsn(n = 1, xi = 0.005 + Xi.majs,omega = omega.majs, alpha =
183.4461481484)
     if(val > 0){
   dens.vec[index] = val
   index.taken = c(index.taken, index)
   num.majority = num.majority - 1}
     else{
       print("Reject", index)
     }

   }
  }
  }
  return(dens.vec)
}
#function to plot the label densities
plot.label.density = function(data){
  densities = data$labels$count/data$measures$num.instances
  barplot(height = densities, names.arg =
colnames(data$dataset[,data$labels$index]),
          las = 2, ylim = c(0,1),
          col = rainbow(data$measures$num.labels),
          main = "Density of labels")
}

#Create the A matrix with 10 labels
A.10 = matrix(nrow = 10, ncol = 10)
for(i in 1:nrow(A.10)){
  for(j in 1:ncol(A.10)){
    A.10[i,j] = round(rnorm(n=1,mean = 0.43,sd = 0.1),0)
  }
}

#Create the A matrix with 20 labels
A.20 = matrix(nrow = 10, ncol = 20)
for(i in 1:nrow(A.20)){
  for(j in 1:ncol(A.20)){
    A.20[i,j] = round(rnorm(n=1,mean = 0.43,sd = 0.1),0)
  }
}
```

Resampling code:

```
library(utiml)
library(XML)
library(mldr)
library(smotefamily)
library(imbalance)
library(neighbr)
library(e1071)
library(FNN)
library(mldr.datasets)



LP_RUS = function(D, P, plots){
  #Initialise variabels
  L = NROW(D$labels) #Object for the labels
  samplesToDelete = P*D$measures$num.instances #-> Number of samples to
delete P% size reduction
```

```
  lp.trans = mldr_transform(D, type = "LP") #-> Transform the dataset to
label powerset
  MeanIR = D$measures$meanIR #-> MeanIR for the whole dataset
  IRLbl = D$labels$IRLbl #-> IRLbl for the labels in the original dataset
  labelsetBag = list() #-> Create a bag of observtaions for each of the
possible label combinations
  majBag = list() #-> A list for the majority bags...element type mldr()
  minBag = list() #-> A list for the minority bags...element type mldr()

#Create a bag of the samples in each labelset
  print("Get labelsetbags")
 labelsets = rownames(D$labelsets)#-> Creata a vector with the label
combinations
 lengths.labelsetBag = c()#-> Vector to save the counts of observtaions
belonging to each labelset combination
 for(i in 1:length(labelsets)){#->Iterate over the labelsets
   labelsetBag[[i]] = D[lp.trans$classLabel == labelsets[i]]#-> Create a
labelsetbag for each combination of labels
   lengths.labelsetBag[i] = labelsetBag[[i]]$measures$num.instances#-> Save
the number of instances in each of the bags, used to choose the majority
classes
 }

#Calculate the average number of samples in each labels
 meanSize = (1 / length(labelsetBag))*sum(lengths.labelsetBag)

#Obtain the majority labels bags and the minority bags
 print("Get majority and minority bags")
 j = 1
 z = 1
 indexes = c()
 for(i in 1:length(labelsetBag)){#-> Iterate over the labelsetbat
   if(labelsetBag[[i]]$measures$num.instances > meanSize){#-> Identify the
majority bags
     majBag[[j]] = labelsetBag[[i]]#-> Add bag to the list of majority bags
     j = j + 1
   }
   else{
     minBag[[z]] = labelsetBag[[i]]#->Add bag to the list of minority bags
     z = z+1
     }
 }
 meanRed = round(samplesToDelete/length(majBag))#-> Used to calculate the
number of samples that need to be deleted
if(meanRed < 1){
  print("P too small")
}
#Process the majority bag from smallest to largest
#Calculate the number of instances to delete and remove them
 if(length(majBag) == 0){
   return("No majority bags")
 }
 print("Generate new samples")
 remainder = 0
 for(j in 1:length(majBag)){#-> Iterate over the majority bags
   rBag = min(majBag[[j]]$measures$num.instances - meanSize, meanRed)#->
The number of samples that need to be removed from each bag
   remainder = remainder +  meanRed - rBag #Calculate the remainder

#Delete samples in majBagi
   for(n in 1:rBag){#->Remove rBag samples from majBag(j)
```

```
     x = sample(1:majBag[[j]]$measures$num.instances, size = 1, replace =
F)#-> Randomly select a sample in majBag(j)
     majBag[[j]] = majBag[[j]][-x]#-> Remove sample x
   }
 }

 # #The remainder needs to be randomly shared among the minBags
 for(r in 1:remainder){
    b = sample(1:length(majBag), 1, replace = F)
    if(majBag[[b]]$measures$num.instances > 1){
   x = sample(1:majBag[[b]]$measures$num.instances, size = 1, replace =
F)#-> Randomly select a sample in majBag(j)
   majBag[[b]] = majBag[[b]][-x]#-> Remove sample x
    }
 }

#Create a new resampled dataset to give as output
#First add the majority bags
resampled = majBag[[1]]
for(j in 2:length(majBag)){
  resampled = resampled + majBag[[j]]
}
#Then add the original minority bags
for(z in 1:length(minBag)){
  resampled = resampled + minBag[[z]]
}

#plots(TRUE/FALSE) -> Barplot of the label counts before and after the
resampling has been applied
if(plots == TRUE){
  barplot(D$labels[,2], main = "Before resampling", names.arg =
rownames(D$labels), xlab = "labels", ylab = "Instances per label", ylim =
c(0, max(D$labels[,2])))
  barplot(resampled$labels[,2], main = "After resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(D$labels[,2])))
}

#-> Output for the function in list() format
output = list("Resampled dataset" = resampled, "Samples deleted" =
NROW(D$dataset) - NROW(resampled$dataset), "SCUMBLE before" =
D$measures$scumble, "SCUMBLE after" = resampled$measures$scumble, "MeanIR
before" = D$measures$meanIR, "MeanIR after" = resampled$measures$meanIR)
return(output)
}



### Parameters ###
#D -> dataset
#P -> Percentage
#plots -> (TRUE/FALSE) plots of the number of insatnces per label before
and after the resampling has been applied

### Output ###
#A pre-proccessed dataset
LP_ROS = function(D, P, plots){
  L = NROW(D$labels) #-> Number of labels
  samplesToGenerate = (1+P)*D$measures$num.instances -
D$measures$num.instances #-> Num of samples to generate
```

```
  lp.trans = mldr_transform(D, type = "LP")#-> Dataset in label-powerset
form
  MeanIR = D$measures$meanIR #-> MeanIR for the full dataset
  IRLbl = D$labels$IRLbl #-> IRLbl per label
  labelsetBag = list() #-> A list with bags of instances for each of the
possible combination of labels...label-powerset
  majBag = list()#-> List of majBag datasets in mldr() format
  minBag = list()#-> List of minBag datasets in mldr() format
  resampled = D

#Create a bag of the samples in each labelset
  print("Getting labelsetbags")
 labelsets = rownames(D$labelsets)#-> All of the combinations of labels LP
 lengths.labelsetBag = c() #-> A vector to store the number of instances in
each of the labelsetBags, used to choose the combinations of labels that
need to be resampled
 for(i in 1:length(labelsets)){
   labelsetBag[[i]] = D[lp.trans$classLabel == labelsets[i]]
   lengths.labelsetBag[i] = labelsetBag[[i]]$measures$num.instances
 }

#Calculate the average number of samples in each of the labelsets
 meanSize = (1 / length(labelsetBag))*(sum(lengths.labelsetBag) )

#Obtain the majority labels bags and the minority bags
 print("Obtaining minority and majority bags")
 j = 1
 z = 1
 indexes = c()
 for(i in 1:length(labelsetBag)){#-> Iterate over labelsetBags
   if(labelsetBag[[i]]$measures$num.instances > meanSize){#-> Identify the
majority bags
     majBag[[j]] = labelsetBag[[i]]#-> The majority bags added to a list()
majBag
     j = j + 1
   }
   else{
     minBag[[z]] = labelsetBag[[i]]#-> The minortiy bags added to a list()
minBag
     z = z + 1
     }
 }

 added = 0
 meanInc = round(samplesToGenerate/length(minBag)) #-> Used to decide how
many samples is added to each element in minBag
 if(meanInc < 1){
   return("P too small")
 }
 #Process the minority bags from largest to smallest
 if(length(minBag) == 0){
   return("No minority bags")
 }
 print("Generate samples")
 remainder = 0
 added = 0
 #Calculate the number of instances to delete and remove them
 for(j in length(minBag):1){#-> Iterate over the minority bags
     rBag = min(minBag[[j]]$measures$num.instances + meanSize, meanInc )#-
> The number of samples that need to be removed from each bag
```

```
   remainder = remainder + meanInc - rBag #-> Needs to be distributed
evenly among the bags
#Add samples to minBag
   for(n in 1:round(rBag)){#-> Generate n points that is added to minBag(j)
     x = sample(1:minBag[[j]]$measures$num.instances, size = 1, replace =
F)#-> Select random samples
     minBag[[j]] = minBag[[j]] + minBag[[j]][x]#-> Add the cloned points to
minBag(j)
     resampled = resampled +  minBag[[j]][x]#-> Add the samples to the
resampled dataset
     added = added + 1
   }
 }

 # #Remainder needs to be shared evenly maong the bags
 # for(r in 1:remainder){
 #   b = sample(1:length(minBag), 1, replace = F)
 #   x = sample(1:minBag[[b]]$measures$num.instances, size = 1, replace =
F)#-> Randomly select a sample in majBag(j)
 #   minBag[[b]] = minBag[[b]] + minBag[[j]][x]
 #   resampled = resampled +  minBag[[b]][x]#-> Add the samples to the
resampled dataset
 # }

#From plots(TRUE/FALSE) -> barplots of the label counts before and after
the resampling has been applied
if(plots == TRUE){
  barplot(D$labels[,2], main = "Before resampling", names.arg =
rownames(D$labels), xlab = "labels", ylab = "Instances per label", ylim =
c(0, max(resampled$labels[,2])))
  barplot(resampled$labels[,2], main = "After resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(resampled$labels[,2])))
}

#List of outputs for the function
output = list("Resampled dataset" = resampled, "Samples added" =
NROW(resampled$dataset) - NROW(D$dataset), "SCUMBLE before" =
D$measures$scumble, "SCUMBLE after" = resampled$measures$scumble,  "MeanIR
before" = D$measures$meanIR, "MeanIR after" = resampled$measures$meanIR)
return(output)
}


ML_ROS = function(D, P, plots){
#Initialis some variables
resampled = D #-> The dataset that the new samples will be added to...D-
>orginal
samplesToClone = (1+P)*D$measures$num.instances - D$measures$num.instance
#-> Num of samples to clone/add
L = D$labels #-> Matrix with label properties
MeanIR = D$measures$meanIR #-> MeanIR for the original dataset
minBag = list() #-> List to store the minority bags...elements type mldr()
labelIndex = D$labels$index
minLabelIndex = c() #-> Indexes of the minority labels

#Find the bags of minority samples
print("Finding minority bags")
 IRLbl = D$labels$IRLbl #-> IRLbl for the labels in the original dataset
 j = 1
for(l in 1:length(IRLbl)){#-> Iterate over the labels
```

```
 if(IRLbl[l] > MeanIR){#->Look for minority labels
   minBag[[j]] = D[D$dataset[,labelIndex[l]] == 1]#-> Add the bag of labels
to the minBag list...adding instances
   j = j+1
   minLabelIndex = c(minLabelIndex,l)#-> Indexes of the minority labels
 }
}

 #Stop if there are no minority bags in the dataset
 if(length(minBag) < 1){
   return("No minority bags")
 }

 print("Cloning samples")
while(samplesToClone > 0 ){#-> Iterate while there are still instances to
clone
  delete.index = c() #Save the indexes of the bags that need to be deleted

  #Clone a sample from each minority bag
   for(i in 1:length(minBag)){#-> Iterate over the bags
    x = sample(1:minBag[[i]]$measures$num.instances, size = 1, replace =
F)#->Select a random instance from minBag(i)
     minBag[[i]] = minBag[[i]] + minBag[[i]][x] #-> Add the instance to
minBag(i)
     resampled = resampled + minBag[[i]][x]#-> Also add the cloned sample
to the resampled dataset, to test if the labels of minBag(i) iare still
minorities

     #Check if the minority labels are still minorities, remove the label
if it is not a minoroty
     if( resampled$labels$IRLbl[minLabelIndex[i]] <=
resampled$measures$meanIR ){
        delete.index = c(delete.index, i) #Save the indexes of labels that
are no longer minorities
     }
     samplesToClone = samplesToClone - 1 #-> One less sample to clone
   }#-> Back to start of the for loop

  #Delete the labels that are no longer minorities, only after the for loop
    if(length(delete.index) > 0){
    minBag = minBag[-delete.index]
    minLabelIndex = minLabelIndex[-delete.index]
    }

  #Break if there are no minBags left
    if(length(minBag) == 0){
      break
    }

 }#-> Break the while loop


#From the plots(TRUE\FALSE) parameter, plots the label counts before and
after the resampling...barplots
if(plots == TRUE){
  barplot(D$labels[,2], main = "Before resampling", names.arg =
rownames(D$labels), xlab = "labels", ylab = "Instances per label", ylim =
c(0, max(resampled$labels[,2])))
  barplot(resampled$labels[,2], main = "After resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(resampled$labels[,2])))
```

```
}

#-> Give the output of the functiona as a list
output = list("Resampled dataset" = resampled, "Samples added" = NROW(
  resampled$dataset) - NROW(D$dataset), "SCUMBLE before" =
D$measures$scumble, "SCUMBLE after" = resampled$measures$scumble,  "MeanIR
before" = D$measures$meanIR, "MeanIR after" = resampled$measures$meanIR)
return(output)
}


ML_RUS = function(D, P, plots){
#Initialise variables to be used
resampled = D #-> Dataset from which we will delete instances in majority
labels
samplesToDelete = P*D$measures$num.instances #->Number of samples to delete
L = D$labels #-> Labels
MeanIR = D$measures$meanIR #-> MeanIR for the original dataset
majBag = list() #-> Empty list for the majority bags...list elements of
type mldr()
minBag = list() #-> Empty list for the minority bags..list elements of type
mldr()
labelIndex = D$labels$index
majLabelIndex = c() #-> Empty vector for the majority label indexes
minLabelIndex = c() #-> Empty vector for the majority label indexes

#Find the bags of minority samples and bags of majority samples
print("Finding minority bags")
 IRLbl = D$labels$IRLbl #-> IRLbl per label
 i = 1
 j = 1
for(l in 1:length(IRLbl)){ #-> Iterate over the labels
 if(IRLbl[l] <= MeanIR && IRLbl[l]>0){ #-> Find the majority labels and put
them in bags (list)
   majBag[[i]] = D[D$dataset[,labelIndex[l]] == 1] #-> Each bag is an
element in the list
   i = i+1
   majLabelIndex = c(majLabelIndex,l)#->Index of the majority labels
 }else if(IRLbl[l] > 0){#-> Find the minority labels and put them into bags
    minBag[[j]] = D[D$dataset[,labelIndex[l]] == 1]#-> Each bag is an
element in the list
   j = j+1
   minLabelIndex = c(minLabelIndex,l)#->Index of the minority labels
 }
}

 samplesToDelete.total = samplesToDelete
 #Instances cloning loop
 print("Samples to delete:")
 print(samplesToDelete.total)
 while(samplesToDelete > 0 ){#->Loop until samplesToDelete equals zero
   #Delete a sample from each majority bag
   delete.index = c() #Indexes for samples that are no longer minorities


   for(i in 1:length(majBag)){#> Select a random observation in majBag(i)
and delete it

     if(majBag[[i]]$measures$num.instances == 1){
        #If there is only one observation in bag, we cannot delete it
mldr() package
```

```
      print("break")
       break}

      x = sample(1:majBag[[i]]$measures$num.instances, size = 1, replace =
F)
     majBag[[i]] = majBag[[i]][-x]

     #Delete the same sample from the full dataset, to calculate if the
labels are still majorities
    resampled = resampled$dataset[rownames(resampled$dataset) !=
rownames(majBag[[i]][x]$dataset),]
    resampled = mldr_from_dataframe(resampled, attributes =
D$attributesIndexes, labelIndices = labelIndex)

    #Check if the majority label is still a majority label
     if(resampled$labels$IRLbl[majLabelIndex[i]] >
resampled$measures$meanIR ){
       #If the label is not a majority anymore delete it from majBag
        delete.index = c(delete.index, i)
     }
     samplesToDelete = samplesToDelete - 1#-> One less observation to
delete
   } #-> Back to top of for loop

     if(length(delete.index) > 0){
    print(majLabelIndex)
    majBag = majBag[-delete.index]
    majLabelIndex = majLabelIndex[-delete.index]
  }

    if(samplesToDelete/samplesToDelete.total == 0.05){
    print("5%")
  }else if(samplesToDelete/samplesToDelete.total == 0.1){
     print("10%")
  }else if(samplesToDelete/samplesToDelete.total == 0.2){
     print("20%")
  }else if(samplesToDelete/samplesToDelete.total == 0.3){
     print("30%")
  }else if(samplesToDelete/samplesToDelete.total == 0.4){
     print("40%")
  }else if(samplesToDelete/samplesToDelete.total == 0.5){
     print("50%")
  }else if(samplesToDelete/samplesToDelete.total == 0.6){
     print("60%")
  }else if(samplesToDelete/samplesToDelete.total == 0.7){
     print("70%")
  }else if(samplesToDelete/samplesToDelete.total == 0.8){
     print("80%")
  }else if(samplesToDelete/samplesToDelete.total == 0.9){
     print("90%")
  }

 }#Break the while loop


 #Use plots parameter (TRUE/FALSE) -> Barchart of the number of instances
per label before and after the resampling
if(plots == TRUE){
  barplot(D$labels[,2], main = "Before resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(D$labels[,2])))
```

```
  barplot(resampled$labels[,2], main = "After resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(D$labels[,2])))
}

 #Place the functions output in a list
output = list("Resampled dataset" = resampled, "Samples deleted" = NROW(
  D$dataset) - NROW(resampled$dataset), "SCUMBLE before" =
D$measures$scumble, "SCUMBLE after" = resampled$measures$scumble,  "MeanIR
before" = D$measures$meanIR, "MeanIR after" = resampled$measures$meanIR)
return(output)
}


MLeNN = function(D, HT, NN, plots){
L = D$labels #-> Labels
labelIndex = D$labels$index #Column indexes of the labels
IRLbl = D$labels$IRLbl
MeanIR = D$measures$meanIR
markForRemoving = c()


for(i in 1:D$measures$num.instances){#-> Iterate over the samples
  #Looking for samples that are majorities


  #Check to see if the observation is a majority
  for(l in 1:NROW(IRLbl)){ #-> Iterate over the labels
  if(IRLbl[l] > MeanIR){ #-> Check if the label is a majority
    if(D$dataset[i,labelIndex[l]] == 1){
      majority.i = FALSE #Preserve instance as a minority
    }
  }else{
    majority.i = TRUE #Observation is a majority
  }#End if else
  }#End for loop over labels




  #Only for the majority observations
  if(majority.i){

     #Find the NN nearest neighbors to the sample i
    nn = get.knn(data = data.matrix(D$dataset[, D$attributesIndexes]), k =
NN)
    index.kkn = nn$nn.index[i,]
    numDifferences = 0


    #Check to see if the sample should be removed
    for(n in 1:length(index.kkn)){#-> For each of the nearest neighbors
      if(hamming.distance(x = as.numeric(D$dataset[i, labelIndex]), y =
as.numeric(D$dataset[index.kkn[n], labelIndex])) > HT){
        numDifferences = numDifferences + 1
      }
    }#End for loop over neighbors


    if(numDifferences >= NN/2){#->Check if the observation should be
removed
```

```
      markForRemoving = c(markForRemoving, i)
    }

  }#-> End of the if statement checking for majority observations


  #Progress of algorithm
   if(i == 0.05*D$measures$num.instances){
    print("5%")
  }else if(i == 0.1*D$measures$num.instances){
     print("10%")
  }else if(i == 0.2*D$measures$num.instances){
     print("20%")
  }else if(i == 0.3*D$measures$num.instances){
     print("30%")
  }else if(i == 0.4*D$measures$num.instances){
     print("40%")
  }else if(i == 0.5*D$measures$num.instances){
     print("50%")
  }else if(i == 0.6*D$measures$num.instances){
     print("60%")
  }else if(i == 0.7*D$measures$num.instances){
     print("70%")
  }else if(i == 0.8*D$measures$num.instances){
     print("80%")
  }else if(i == 0.9*D$measures$num.instances){
     print("90%")
  }

}#-> End of the for loop iterating over the samples



if(length(markForRemoving) == D$measures$num.instances){
return("Increase HT")#ALgorithm wants to remove all observations
}else if(length(markForRemoving) == 0){
  return("Decrease HT")#No observations being removed
}else {
  resampled = D[-markForRemoving]#Remove observtaions
}



#Use plots parameter (TRUE/FALSE) -> Barchart of the number of instances
per label before and after the resampling
if(plots == TRUE){
  barplot(D$labels[,2], main = "Before resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(D$labels[,2])))
  barplot(resampled$labels[,2], main = "After resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(D$labels[,2])))
}


 #Place the functions output in a list
output = list("Resampled dataset" = resampled, "Samples deleted" = NROW(
  D$dataset) - NROW(resampled$dataset), "SCUMBLE before" =
D$measures$scumble, "SCUMBLE after" = resampled$measures$scumble,  "MeanIR
before" = D$measures$meanIR, "MeanIR after" = resampled$measures$meanIR)
```

```
return(output)
}

remedial = function(D, plots){


#Calculate the imbalance levels
IRLbl = D$labels$IRLbl
MeanIR = D$measures$meanIR
#Calculate SCUMBLE
SCUMBLEins = D$dataset$.SCUMBLE
SCUMBLE.mean = mean(SCUMBLEins)
L = D$labels
labelIndex = D$labels$index #Column indexes of labels
resampled = D


for(i in 1:D$measures$num.instances){#->Iterate over the samples


   if(SCUMBLEins[i] > SCUMBLE.mean){#-> Look for observations with high
concurrance ie. SCUMBLEins
   #If the instance has high concurrance, decouple the labels

    #Decoupling
    clone = D[i] #-> Clone the instance
    clone$dataset[,labelIndex] =
clone$dataset[,labelIndex]*(IRLbl>MeanIR)#->Maintain majority labels
    resampled$dataset[i,labelIndex] =
resampled$dataset[i,labelIndex]*(IRLbl<=MeanIR)#->Maintain minority labels
    resampled = resampled + clone#->Add clone to resampled dataset
   }


  #Progress of the algorithm
  if(i == 0.05*D$measures$num.instances){
    print("5%")
  }else if(i == 0.1*D$measures$num.instances){
     print("10%")
  }else if(i == 0.2*D$measures$num.instances){
     print("20%")
  }else if(i == 0.3*D$measures$num.instances){
     print("30%")
  }else if(i == 0.4*D$measures$num.instances){
     print("40%")
  }else if(i == 0.5*D$measures$num.instances){
     print("50%")
  }else if(i == 0.6*D$measures$num.instances){
     print("60%")
  }else if(i == 0.7*D$measures$num.instances){
     print("70%")
  }else if(i == 0.8*D$measures$num.instances){
     print("80%")
  }else if(i == 0.9*D$measures$num.instances){
     print("90%")
  }


  }#-> End for loop
```

```
#Use plots parameter (TRUE/FALSE) -> Barchart of the number of instances
per label before and after the resampling
if(plots == TRUE){
  barplot(D$labels[,2], main = "Before resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(D$labels[,2])))
  barplot(resampled$labels[,2], main = "After resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(D$labels[,2])))
}

#Place the functions output in a list
output = list("Resampled dataset" = resampled, "Samples added" = NROW(
  resampled$dataset) - NROW(D$dataset), "SCUMBLE before" =
D$measures$scumble, "SCUMBLE after" = resampled$measures$scumble,  "MeanIR
before" = D$measures$meanIR, "MeanIR after" = resampled$measures$meanIR)
return(output)

}


##############################################################################
##########################
newSample = function(sample, refNeigh, neighbors, D, k){
  L = D$labels
  labelIndex = D$labels$index #Column index of labels



  #Assign the features to the new synthetic sample
  synth.sample = sample
  for(j in 1:length(D$attributesIndexes)){#Iterate over the features
    if(D$attributes[j] == "NUMERIC"){#Check if the feature is numeric
      diff = as.numeric(refNeigh$dataset[,refNeigh$attributesIndexes[j]]) -
as.numeric(sample$dataset[,refNeigh$attributesIndexes[j]])
      offset = diff*runif(n = 1, min = 0 , max = 1)
      value = as.numeric(sample$dataset[,refNeigh$attributesIndexes[j]]) +
offset
    }else{#If the feature is not numeric
      #Most frequent value
      value =
tail(sort(table(refNeigh$dataset[,refNeigh$attributesIndexes[j]])), 1)
    }
    synth.sample$dataset[,refNeigh$attributesIndexes[j]] = value #Assign
the new attributes
  }



  #Label set assignment
  lblcounts = sample$dataset[,labelIndex] #Samples labels
  neigh.counts = c(rep(0, length(labelIndex))) #Empty vector
  for(i in 1:neighbors$measures$num.instances){ #Iterate over neighbors
    neigh.counts = neigh.counts + neighbors$dataset[i,labelIndex]#Sum the
labels
  }
  lblcounts = lblcounts + neigh.counts
  labels = c(1)*(lblcounts > (k + 1)/2)#Find the labels
  synth.sample$dataset[, labelIndex] = labels #Assign the labels
  return(synth.sample)#-> Return the new observation
}
```

```
#############################################################################
##########################
MLSMOTE = function(D, k, plots){
resampled = D
L = D$labels
MeanIR = D$measures$meanIR
IRLbl = L$IRLbl
minBag = list()
labelIndex = D$labels$index

i = 1
for(l in 1:NROW(L)){#Iterate over the labels

  if(IRLbl[l] > MeanIR){#Identify minority labels and add the instances to
a bag
    minBag[[i]] = D[D$dataset[,labelIndex[l]] == 1] #Minbag for label l
 if(minBag[[i]]$measures$num.instances > k){#If there are more than k
observations
   nn = get.knn(data =
data.matrix(minBag[[i]]$dataset[,D$attributesIndexes]), k = k)#get the
nearest neighbors
 for(j in 1:minBag[[i]]$measures$num.instances){#->Iterate over the samples
in minbag[i]
    index.kkn = nn$nn.index[j,]#Sorted neareast neighbors to observation j
    #Neighbor set selection
    neighbors = minBag[[i]][index.kkn]#k nn to observation j
    refNeigh = minBag[[i]][sample(index.kkn, size = 1, replace =
FALSE)]#Choose random neighbor as a reference
    #Feature set and labelset generation
    sample = minBag[[i]][j]#Current observation(j) in minbag[i]
    synthSmpl = newSample(sample = minBag[[i]][j], refNeigh, neighbors, D,
k)#New sample generated
    resampled = resampled + synthSmpl#add sample to the dataset
    }#End for
    }#End if  for k
    i = i + 1
      if(l == 0.1*NROW(L)){
    print("10%")
  }else if(l == 0.2*NROW(L)){
    print("20%")
  }else if(l == 0.3*NROW(L)){
    print("30%")
  }else if(l == 0.4*NROW(L)){
    print("40%")
  }else if(l == 0.5*NROW(L)){
    print("50%")
  }else if(l == 0.6*NROW(L)){
    print("60%")
  }else if(l == 0.7*NROW(L)){
    print("70%")
  }else if(l == 0.8*NROW(L)){
    print("80%")
  }else if(l == 0.9*NROW(L)){
    print("90%")
  }

}#End if over labels
} #End for over labels
```

```
#Use plots parameter (TRUE/FALSE) -> Barchart of the number of instances
per label before and after the resampling
if(plots == TRUE){
  barplot(D$labels[,2], main = "Before resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(resampled$labels[,2])))
  barplot(resampled$labels[,2], main = "After resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(resampled$labels[,2])))
}

#Place the functions output in a list
output = list("Resampled dataset" = resampled, "Samples added" = NROW(
  resampled$dataset) - NROW(D$dataset), "SCUMBLE before" =
D$measures$scumble, "SCUMBLE after" = resampled$measures$scumble,  "MeanIR
before" = D$measures$meanIR, "MeanIR after" = resampled$measures$meanIR)
return(output)
}

################################################################################
################
#Function that finds the k-nearest neighbours to each instance
find_knn_per_example = function(D,k){
L = D$labels
labelIndex = D$labels$index
nearest.neighbors = list()
neighbor.index = matrix(nrow = D$measures$num.instances, ncol = k)
#Find the k nearest neighbors for each instance
nn = get.knn(data = data.matrix(D$dataset[, D$attributesIndexes]), k = k )
for(i in 1:D$measures$num.instances){
  nearest.neighbors[[i]] = D[nn$nn.index[i,]]
}

  if(i == 0.05*D$measures$num.instances){
    print("5%")
  }else if(i == 0.1*D$measures$num.instances){
     print("10%")
  }else if(i == 0.2*D$measures$num.instances){
     print("20%")
  }else if(i == 0.3*D$measures$num.instances){
     print("30%")
  }else if(i == 0.4*D$measures$num.instances){
     print("40%")
  }else if(i == 0.5*D$measures$num.instances){
     print("50%")
  }else if(i == 0.6*D$measures$num.instances){
     print("60%")
  }else if(i == 0.7*D$measures$num.instances){
     print("70%")
  }else if(i == 0.8*D$measures$num.instances){
     print("80%")
  }else if(i == 0.9*D$measures$num.instances){
     print("90%")
  }
 print("100%")
  output = list("neighbor.index" = neighbor.index,"nearest.neighbors" =
nearest.neighbors)
}

################################################################################
################
```

```
#Calculate the proportion of neighbors having oposite class
calculateC = function(D,k, nearest.neighbors){
L = D$labels
labelIndex = D$labels$index

#For each observation we calcuate the propotion of neighbors having
opposite class/labels
C = matrix(nrow = D$measures$num.instances, ncol = NROW(D$labels))
for(i in 1:D$measures$num.instances){#Loop over observations
labs.i = D$dataset[i,labelIndex]#Labels of observation i in orginal dataset
labs.nn.i = nearest.neighbors[[i]]$dataset[,labelIndex]#labels for the nn
of obs i
s = rep(0, NROW(L))
for(z in 1:k){#Loop over neighbors
  s = (s + c(1)*(labs.i != labs.nn.i[z,]))/k
}
  C[i,] = s
}#End loop over observations
return(C)
}
##############################################################################
################
#Find the labels that are minority labels...to be used in the calculation
of w
#Crude way of finding minority labels...Consider replacing this with
IRbl>MeanIR
get_min_class_labels = function(D){
  L = D$labels
  minority_labels = c()
  minority_labels = c(rep(0,NROW(L)))
  labelIndex = D$labels$index #-> Col index of the labs

  for(l in 1:NROW(L)){
    class1 = sum(D$dataset[,labelIndex[l]] == 1)
    class0 = sum(D$dataset[,labelIndex[l]] == 0)

    if(class1 >= class0){#Idenitfied as a majority label
      minority_labels[l] = 0
    }
    else{#Identified as a minority label
      minority_labels[l] = 1
    }
  }
  return(minority_labels)
}

##############################################################################
################
#Aggregate the values of C per sample, to calculate the sampling weight
get_w_per_example = function(D, C){
  L = D$labels
  minority_labels = get_min_class_labels(D)
  w = c()
  sum_of_non_out_minority_examples_per_example = rep(0, NROW(L))
  labelIndex = D$labels$index #-> Col index of the labs

  for(j in 1:NROW(L)){#Iterate over columns
    for(i in 1:D$measures$num.instances){#Iterate over rows
      if(D$dataset[i,labelIndex[j]] == minority_labels[j] &  C[i,j]<1){
        sum_of_non_out_minority_examples_per_example =
sum_of_non_out_minority_examples_per_example + C[i,j]
```

```
      }#End if
    }#End for loop over columns
  }#End for loop over rows

  for(i in 1:D$measures$num.instances){#Iterate over rows
    sum = 0
    for(j in 1:NROW(L)){#Iterate over columns
      if(D$dataset[i, labelIndex[j]]==minority_labels[j] & C[i,j]<1){
        sum = sum + C[i,j]/sum_of_non_out_minority_examples_per_example[j]
      }#End if
    }#End columns for loop
    w[i] = sum
  }#End rows for loop

  return(w)
}

#############################################################################
#################
#C->Matrix that stores the proportion of KNNs
#k-> Number of nearest neighbors
#OUTPUT -> Type of instance T
initTypes = function(C, k, neighbor.index, D){
  L = D$labels
  T.mat = matrix(nrow = D$measures$num.instances, ncol = NROW(L))
  minority_labels = get_min_class_labels(D)
  labelIndex = D$labels$index #-> Col index of the labs

for(i in 1:D$measures$num.instances){#Iterate over the samples
  for(j in 1:NROW(L)){#Iterate over the labels
    if(D$dataset[i,labelIndex[j]] == minority_labels[j]){#y is in majority
class
       if(C[i,j] < 0.3){
      T.mat[i,j] = "SF"
    }
    else if(C[i,j] < 0.7){
      T.mat[i,j] = "BD"
    }
    else if(C[i,j] < 1){
      T.mat[i,j] = "RR"
    }
    else{
      T.mat[i,j] = "OT"
    }
    }else{
       T.mat[i,j] = "MJ"
    }
  }#End for loop over columns
}#End for loop over observations

change = TRUE
while(change){
  change = FALSE
  for(i in 1:D$measures$num.instances){#Iterate over observations
    for(j in 1:NROW(L)){#Iterate over the labels
      if(T.mat[i,j] == "RR"){#Check for type RR
        for(m in neighbor.index[i,]){#Iterate over the bag of nearest
neighbors
          if(T.mat[m,j] == "SF" || T.mat[m,j] == "BD"){
            T.mat[i,j] = "BD"
            change = TRUE
```

```
            break
          }#End of if
        }#End of for loop over nearest neighbors
      }#End of if checking for RR
    }#End of for loop over labels
  }#End of for loop over observations
}#End of while loop

return(T.mat)
}
#############################################################################
################
GenerateInstance = function(seed,ref,Ts, Tr, D){
L = D$labels
  synth.sample = D[1]#-> An initial starting point of type mldr()
  labelIndex = D$labels$index #-> Col index of the labs

for(j in D$attributesIndexes){#Interpolate the feature values of the
instance
   synth.sample$dataset[,j] = as.numeric(seed$dataset[,j]) + runif(1,min =
0, max = 1)*(as.numeric(ref$dataset[,j]) - as.numeric(seed$dataset[,j]))
}
ds = distance(x = as.numeric(synth.sample$dataset[,D$attributesIndexes]), y
= as.numeric(seed$dataset[,D$attributesIndexes]), measure =
"euclidean")#Distance from synth to seed
dr = distance(x = as.numeric(synth.sample$dataset[,D$attributesIndexes]), y
= as.numeric(ref$dataset[,D$attributesIndexes]), measure =
"euclidean")#Distance from synth to ref
cd = ds/(ds + dr)#indicates whether the synthetic instance is closer to the
seed (cd < 0.5) or closer to the reference instance (cd > 0.5)

#Label assignment
theta = 0
for(j in 1:NROW(L)){
  if(seed$dataset[,labelIndex[j]] == ref$dataset[,labelIndex[j]]){
    synth.sample$dataset[, labelIndex] = seed$dataset[, labelIndex]
  }
  else{
    if(Ts[j] == "Mj"){#Ensure seed(j) is a minority label
      #Switch around the reference and seed instances
      holder = seed
      ref = seed
      seed = holder
      #Swith around the distances
      holder = ds
      dr = ds
      ds = holder
      #Switch around the references
      holder = Ts[j]
      Tr[j] = Ts[j]
      Ts[j] = holder

      cd = 1 - cd
    }
    if(Ts[j] == "SF"){
      theta =  0.5
    }
    else if(Ts[j] == "BD"){
      theta = 0.75
    }
    else if(Ts[j] == "RR"){
```

```
      theta = 1.00005
    }
    else if(Ts[j] == "OT"){
      theta = -0.00005
    }
    if(cd <= theta){
      synth.sample$dataset[,labelIndex] = seed$dataset[, labelIndex]
    }
    else{
        synth.sample$dataset[,labelIndex] = ref$dataset[, labelIndex]
    }
  }
}
return(synth.sample)
}
##############################################################################
################
# Get the seed instance for the instance generation
get_seed_instance = function(w){
  seed.index = 0
  limit = runif(n = 1, min = 0, max = 1)*sum(w)
  temp_sum = 0
  for(i in 1:length(w)){
    temp_sum = temp_sum + w[i]
    if(limit <= temp_sum){
      seed.index = i
      break
    }
  }
  return(seed.index)
}
##############################################################################
################
                          #            FINAL FUNCTION            #
##############################################################################
################
MLSOL = function(D,P,k, plots){
GenNum = D$measures$num.instances*P #->Number of instances to generate
resampled = D
L = D$labels
labelIndex = D$labels$index #-> Col index of the labs

print("Finding nearest neighbors")
knn = find_knn_per_example(D,k)#Find the knn to each of the samples
neighbor.index = knn$neighbor.index#Index of the neighbors in the origianal
D
nearest.neighbors = knn$nearest.neighbors#List with the neighbors to each
observation

print("Calculating C")
C = calculateC(D,k, nearest.neighbors)#Proportion of neighbors having
oposite class

print("Caclulating w")
w = get_w_per_example(D, C)#Sampling weight per training example,
aggregated from C

print("Finding observation types")
T.mat = initTypes(C,k, neighbor.index, D)#Distinguish the observations into
types
```

```
tot = GenNum
print("Generating new instances")
while(GenNum > 0){#Generate the new instances
  seed.index = get_seed_instance(w)#Choose a random seed index using
weights
  seed = D[seed.index]#The seed observation
  x = sample(x = 1:k, 1, replace = FALSE)#Choose a random neighbor
  reference = nearest.neighbors[[seed.index]][x]
  new = GenerateInstance(seed,reference,Ts = T.mat[seed.index,], Tr=
T.mat[neighbor.index[i,x],], D) #Generate a new instance
  resampled = resampled + new #Add new instance to resampled dataset
  GenNum = GenNum - 1 #One less observation to generate
  if(GenNum == 0.1*tot){
    print("10%")
  }else if( GenNum == 0.2*tot){
     print("20%")
  }else if( GenNum == 0.3*tot){
     print("30%")
  }else if( GenNum == 0.4*tot){
     print("40%")
  }else if( GenNum == 0.5*tot){
     print("50%")
  }else if( GenNum == 0.6*tot){
     print("60%")
  }else if( GenNum == 0.7*tot){
     print("70%")
  }else if( GenNum == 0.8*tot){
     print("80%")
  }else if( GenNum == 0.9*tot){
     print("90%")
  }
}

#Use plots parameter (TRUE/FALSE) -> Barchart of the number of instances
per label before and after the resampling
if(plots == TRUE){
  barplot(D$labels[,2], main = "Before resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(resampled$labels[,2])))
  barplot(resampled$labels[,2], main = "After resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(resampled$labels[,2])))
}

#Place the functions output in a list
output = list("Resampled dataset" = resampled, "Samples added" = NROW(
  resampled$dataset) - NROW(D$dataset), "SCUMBLE before" =
D$measures$scumble, "SCUMBLE after" = resampled$measures$scumble,  "MeanIR
before" = D$measures$meanIR, "MeanIR after" = resampled$measures$meanIR)

return(output)
}



RHwRSMT = function(D, plots){
  print("Implementing remedial")
  remed = remedial(D, plots = FALSE)
  print("Implementing MLSMOTE")
  resampled = MLSMOTE(D = remed$`Resampled dataset`, k = 5, plots =
FALSE)$`Resampled dataset`
```

```
  #Use plots parameter (TRUE/FALSE) -> Barchart of the number of instances
per label before   and after the resampling
  if(plots == TRUE){
  barplot(D$labels[,2], main = "Before resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(resampled$labels[,2])))
  barplot(resampled$labels[,2], main = "After resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(resampled$labels[,2])))
}

  #Place the functions output in a list
  output = list("Resampled dataset" = resampled, "Samples added" = NROW(
  resampled$dataset) - NROW(D$dataset), "SCUMBLE before" =
D$measures$scumble, "SCUMBLE after" = resampled$measures$scumble,  "MeanIR
before" = D$measures$meanIR, "MeanIR after" = resampled$measures$meanIR)

  return(output)
}



################################################################################
###############                              #     Undersampling method      #
################################################################################
###############
undersamplingMethod = function(D){
  L = D$labels
  labelIndex = D$labels$index #-> Col index of the labs
  MeanIR = D$measures$meanIR
  #Formula to choose the threshold
  I = 1/sqrt(MeanIR)
  if(I >= 0.5){
    TH = 0.5
  }else if( 0.5 > I && I > 0.3){
    TH= 0.3
  }else if(I<0.3){
    TH = 0.15
  }
  majBag = list()
  majLabelIndex = c()
  TL = c()
  #Find the nearest neighbor
  neighbors = get.knn(data = data.matrix(D$dataset[,D$attributesIndexes]),
k = 1)

  #Get the majority labels
  IRLbl = D$labels$IRLbl #-> IRLbl for the labels in the original dataset
  i = 1
for(l in 1:length(IRLbl)){#-> Iterate over the labels
 if(IRLbl[l] <= MeanIR && IRLbl[l] > 0){#->Look for majority labels
   majBag[[i]] = D[D$dataset[,labelIndex[l]] == 1]#-> Add the bag of labels
to the majBag
   i = i+1
   majLabelIndex = c(majLabelIndex,l)#-> Indexes of the majority labels
 }
}

 for(j in 1:length(majBag)){#Iterate ove the majority bags
```

```
   for(i in 1:majBag[[j]]$measures$num.instances){#Iterate ove the
instances in majbag[j]
 #Check if the current sample has been removed already
    neighbor.index = neighbors$nn.index[i,1]#Find the nearest neighbor to
observation i
    NN = D[neighbor.index]#NN in the dataset
     if(sum(NN$dataset[,labelIndex]) > 0){#If the observation has labels
present
   adj.hamming.dist = sum(xor(NN$dataset[, labelIndex], D$dataset[i,
labelIndex]))/sum((NN$dataset[, labelIndex] | D$dataset[i,
labelIndex]))#Calculate the adjusted hamming distance
    }else{
      adj.hamming.dist = 0
    }
   if(adj.hamming.dist >= TH){#Check if the observation should be removed
     TL = c(TL, i)#Add the index to observations that need to be removed
      }#End of the small if
  }#End for loop over observations in majBagi

    if(j == 0.05*length(majBag)){
    print("5%")
  }else if(j == 0.1*length(majBag)){
     print("10%")
  }else if(j == 0.2*length(majBag)){
     print("20%")
  }else if(j == 0.3*length(majBag)){
     print("30%")
  }else if(j == 0.4*length(majBag)){
     print("40%")
  }else if(j == 0.5*length(majBag)){
     print("50%")
  }else if(j == 0.6*length(majBag)){
     print("60%")
  }else if(j == 0.7*length(majBag)){
     print("70%")
  }else if(j == 0.8*length(majBag)){
     print("80%")
  }else if(j == 0.9*length(majBag)){
     print("90%")
  }


}#End for loop over majBags

 return(unique(TL))
}#End of function

##############################################################################
#                            Cleaning method                                 #
##############################################################################
cleaningMethod = function(D){
  L = D$labels
  labelIndex = D$labels$index #-> Col index of the labs
  MeanIR = D$measures$meanIR
  #Formula to choose the threshold
  I = 1/sqrt(MeanIR)
  if(I >= 0.5){
    TH = 0.5
  }else if( 0.5 > I && I > 0.3){
    TH = 0.3
  }else if(I < 0.3){
```

```
    TH = 0.15
  }
  TL = c()#Vector holding observations to be removed
  #Find the nearest neighbors
  neighbors = get.knn(data = data.matrix(D$dataset[,D$attributesIndexes]),
k = 1)#get the nn to each of the observations
  for(i in 1:D$measures$num.instances){#Iterate over the observations
    neighbor.index = neighbors$nn.index[i,1]#Index of the nn to each
observation
    NN = D[neighbor.index]#The nearest neighbor
    if(sum(NN$dataset[,labelIndex]) > 0){#Check if labels are present
   adj.hamming.dist = sum(xor(NN$dataset[, labelIndex], D$dataset[i,
labelIndex]))/sum((NN$dataset[, labelIndex] | D$dataset[i,
labelIndex]))#Calculate the adjusted hamming distance
    }else{
      adj.hamming.dist = 0
    }
   if(adj.hamming.dist >= TH){#Check if the observation should be removed
     TL = c(TL, i)#Add index to observations that need to be removed
    }#End of if
    if(i == 0.05*D$measures$num.instances){
    print("5%")
  }else if(i == 0.1*D$measures$num.instances){
     print("10%")
  }else if(i == 0.2*D$measures$num.instances){
     print("20%")
  }else if(i == 0.3*D$measures$num.instances){
     print("30%")
  }else if(i == 0.4*D$measures$num.instances){
     print("40%")
  }else if(i == 0.5*D$measures$num.instances){
     print("50%")
  }else if(i == 0.6*D$measures$num.instances){
     print("60%")
  }else if(i == 0.7*D$measures$num.instances){
     print("70%")
  }else if(i == 0.8*D$measures$num.instances){
     print("80%")
  }else if(i == 0.9*D$measures$num.instances){
     print("90%")
  }
  }#End for loop over observations
  return(unique(TL))
}#End of function
##############################################################################

#D -> Dataset
#type -> "cleaning" or "undersampling"
MLTL = function(D,type, plots){
  resampled = D
  if(type == "cleaning"){
    TL = cleaningMethod(D)
  }
  else if(type == "undersampling"){
    TL = undersamplingMethod(D)
  }
  #Only remove the observations if there are observations to remove anf if
all the observatiosna are not being removed
  if(length(TL) != 0 && length(TL) != D$measures$num.instances){
  resampled = resampled[-TL]
}
```

```
  #Use plots parameter (TRUE/FALSE) -> Barchart of the number of instances
per label before   and after the resampling
  if(plots == TRUE){
  barplot(D$labels[,2], main = "Before resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(D$labels[,2])))
  barplot(resampled$labels[,2], main = "After resampling", names.arg =
rownames(resampled$labels), xlab = "labels", ylab = "Instances per label",
ylim = c(0, max(D$labels[,2])))
}

  #Place the functions output in a list
  output = list("Resampled dataset" = resampled, "Samples deleted" = NROW(
  D$dataset) - NROW(resampled$dataset), "SCUMBLE before" =
D$measures$scumble, "SCUMBLE after" = resampled$measures$scumble,  "MeanIR
before" =   D$measures$meanIR, "MeanIR after" = resampled$measures$meanIR)
  return(output)
}
```