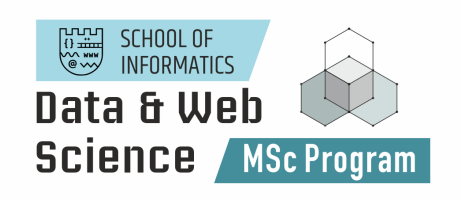
Aristotle University of Thessaloniki

MSc Data and Web Science



A "potpourri" of Machine Learning algorithms addressing advanced topics

Tourgaidis Dimitrios, 66*

Gkiouzelis Nikolaos, 62*

Michoulis Georgios, 82*

Moschopoulos Vasileios, 59*

*All authors contributed equally to this project

Our GitHub repository:
Enkhai/addressing-advanced-ML-topics

Thessaloniki, June 16, 2021

Contents

List of Tables

List of Figures

1 Introduction

The aim of this project was to examine the behaviour of various techniques concerning advanced topics in machine learning. In order to understand the behavior of these techniques for each distinct topic, we examined them using a number of machine learning models. The domain for which we addressed the advanced topics was **Natural Language Processing (NLP)**. More specifically, the advanced machine learning topics we examined were; **cost-sensitive learning**, **class-imbalance**, **multi-label learning** and **explainable learning**. The report is structured as follows:

- Firstly, we provide a detailed description and key elements of each selected dataset.
- Secondly, we provide insights and definitions of the utilized advanced machine learning techniques.
- Thirdly, we analyse and evaluate the produced results.
- And finally, we present our conclusions.

2 Data

This section is dedicated to the datasets we used for implementations and experiments. Since our focus was on Natural Language Processing, we collected three different datasets that suited our needs. For each dataset we briefly discuss its characteristics and present certain statistics extracted using Exploratory Data Analysis. In addition, we also discuss some of our observations and include correlograms to visualize the relationship between different labels.

The content of our datasets was in English and consisted of the following:

- **SMS Spam** was a dataset consisting of a spam or ham collection of SMS messages.
- **Toxic Comment Classification** was a dataset containing various types of toxic comments, with each comment classified in multiple labels.
- **TDMB5000** comprised movie overviews, with multiple labels, where the goal is the classification of movies to different labels. Each movie could be classified in more than one label, similarly to the Toxic Comment Classification Dataset.

2.1 SMS Spam collection

The first dataset we used in our project regards SMS spamming. We use this dataset to deal with 3 of the 4 fields we examine in our project; *class-imbalance*, *cost-sensitive*, *explainable machine learning*. This particular dataset is available in Kaggle and has been employed in several projects. Until the date that this

report had been written, 614 projects has been uploaded to Kaggle that adopt this dataset.

Concerning the characteristics of the dataset, it contains 5.574 messages in English, where each message is either a *spam* or a *ham* (legitimate). As for its structure, it comprises a *csv* file that contains 2 columns and each line in the file reflects a distinct message. The first column indicates if a message is a *spam* or a *ham*, consequently, the value in this columns for a message is either "spam" or "ham". The value of the second column contains the message itself in a string format. Table 1 shows the first 3 messages that are included in the current *csv* file.

	Type	Message
1	ham	Go until jurong point, crazy.. Available only ...
2	ham	Ok lar... Joking wif u oni...
3	spam	Free entry in 2 a wkly comp to win FA Cup fina...

Table 1: First 3 messages of the SMS spam collection dataset

Following, we present several important statistical data of the selected dataset. From the 5.574 messages that the dataset contains, 4.835 are *ham messages* and 747 *spam messages*. From the *hams*, 4.516 are unique messages and from the *spams* the corresponding number is 653. Next, the mean length of the *ham* messages is 71 words, while for the *spam* case it is higher; 139 words. In summary, all the aforementioned statistics are presented in Table 2.

	Message		
	count	unique	mean(length(message)))
ham	4825	4516	71
spam	747	653	139

Table 2: Important statistics of SMS spam collection dataset.

2.2 Toxic Comment Classification

The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

This dataset was created by the Conversation AI team and it was published online on Kaggle. The aim of using this dataset was to detect different types of of toxicity. The dataset consists of comments from Wikipedia's talk page edit.

The Toxic Comment Classification dataset was preferred for the *multi-label learning* field due to its nature and the fact that detecting different types of of toxicity, like threats, obscenity, insults, and identity-based hate, requires a model capable of dealing with multiple labels. This dataset provides a number

of 159571 comments. Moreover, the correlation graph displayed in Figure 1 shows a high dependency between the insult and obscene labels.

Figure 1: Correlogram of the 6 different types of toxic comments for the Toxic Comment Classification dataset

Following, we present some useful statistical data. Most comments were largely unlabelled, with a percentage of 89.8%. Out of the 159571 comments, only 16225 had any labels. For this reason we decided to drop unlabelled comments and used only the small subset of the labelled ones. In Table 3 we can see the number of comments categorized in each label.

toxic	severe toxic	obscene	threat	insult	identity hate
15294	1595	8449	478	7877	1405

Table 3: Number of comments for each label for the Toxic Comment Classification dataset

2.3 TMDB5000: Movie Genre Prediction

Due to the low number of labels of the Toxic Comment Classification dataset, we decided to find yet another dataset with higher number of labels for the *multi-label learning* field. The dataset we eventually used was the TMDB500. The purpose of this dataset is to predict the multiple genre labels using an overview

text. The dataset contains 20 different labels, albeit with roughly 5000 movie titles only, hence its name.

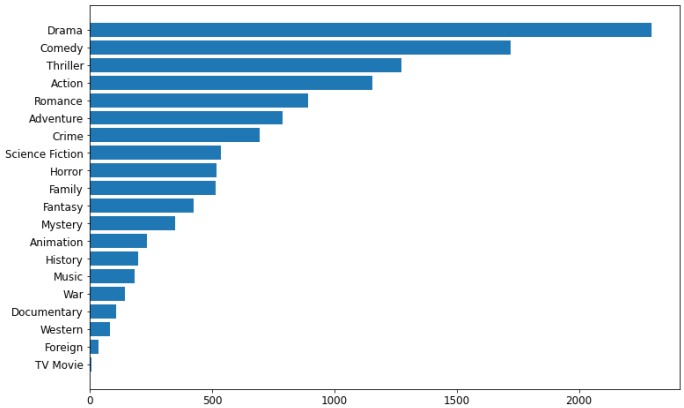


Figure 2: Barplot with the number of labels in the TMDB5000 dataset

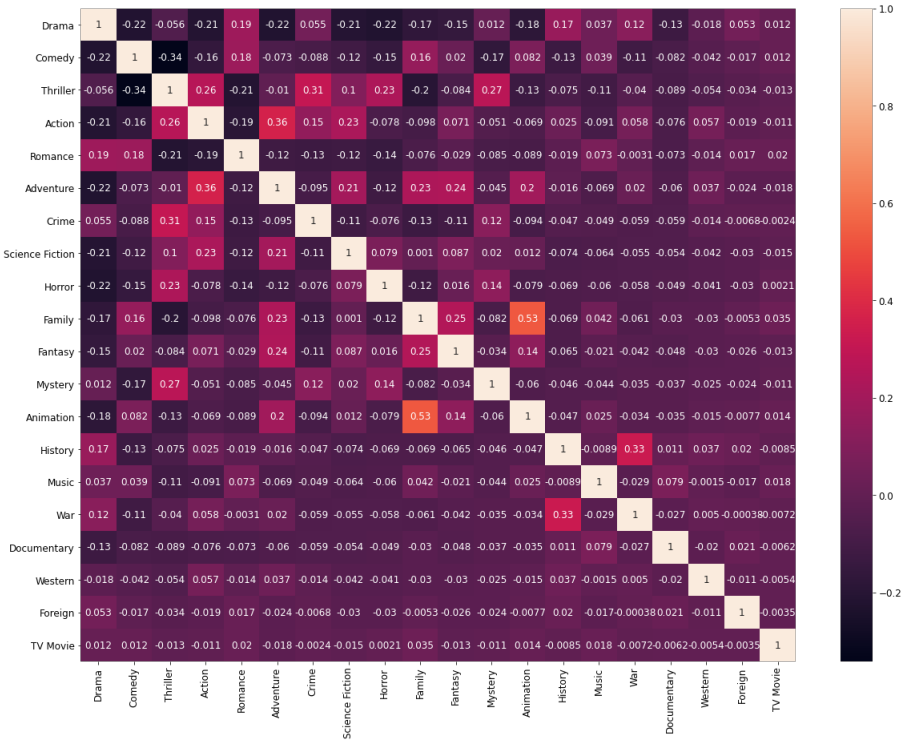


Figure 3: Correlogram of the 20 different movie genres for the TMDB5000 dataset

The exact number of movie overviews in the dataset were 4804, with Drama, Comedy, Thriller, Action and Romance being the top 5 most common labels. A barplot of the frequencies of each label can be seen in Figure 2 and a table containing the exact numbers of the top 10 most common labels can be seen in Table 4. We must stress here that due to a higher number of different labels and various correlations among labels, we cannot expect our multi-label learning algorithms to perform that well. The correlogram can be seen in detail in Figure 3.

Drama	Comedy	Thriller	Action	Romance
2296	1722	1274	1154	894

Adventure	Crime	Science Fiction	Horror	Family
790	696	535	519	513

Table 4: Top 10 number of movies for each label for the TMDB5000 dataset, in decreasing order.

3 Models

Due to the fact that not all readers will be familiar with the advanced machine learning topics that this report regards, we initially provide an introduction for each field and then continue to present the corresponding techniques we examined for each of them.

The report is structured in such a way that the reader can examine the topics one at a time by studying the techniques we implemented in section 3 and then proceeding with the results in section 4, corresponding to our experiments regarding those techniques. On the other hand, the reader also has the option to collectively examine all of the topics and the techniques we examine in each, by reading the entire section 3 at first, and then continuing with section 4 with the results of our experiments.

3.1 Cost sensitive

When we create a machine learning model, in most cases, we develop it to treat all the misclassification errors as equal. But in many real-life domains, such an assumption is wrong and could lead to fatal irreversible effects, analogous the domain's importance; *spam detection*, *diagnosis of medical condition*, *fraud detection*, *granting bank loans*. In spam detection, where a message is classified either as "spam" or "not spam", the misclassification of a "not spam" message as a "spam", could have more negative reverberations as the reversal occasion.For example, the accidental removal of a ham fellowship e-mail is much more significant than reading some spam e-mails.

In machine learning, there is an entire subfield, named **Cost-sensitive learning**, which regards the construction of models that are capable to be used

in such cases. These type of models attach different *costs* to different misclassifications types. Consequently, the substantial difference between a cost-sensitive and a cost-insensitive model, is that the former takes the *costs* of prediction errors (and potentially other costs) into account when it is trained and treats different misclassifications differently. Cost-sensitive learning can be applied in multiclass problems too. We proceed to present the cost-sensitive learning techniques we used to train our models in this task.

- **Weighting** is a cost-sensitive learning approach in which we weigh each example analogous to its misclassification cost from the corresponding cost-matrix.
- **Bayes Minimum Risk Classifier (BMRC)** is a decision model that utilizes probabilities and the *costs* of the decisions, in order to quantify the tradeoffs of various decisions. In the case of cost-sensitive learning, the *costs* are extracted from the **cost-matrix** (defined in 4.1).
- **Probability Calibration** is the procedure in which we try to enhance the produced probabilities from our selected classifiers. These probabilities are going to be utilized as inputs in various cost-sensitive approaches and as a result play a huge part in their overall performance. A pair of probability values like (0.4, 0.6) will be ambiguous for our cost-sensitive approach whereas a 'clear' probability pair of (0.9, 0.1) will surely produce better results. Many algorithms, namely logistic regression, Naive Bayes classifier, Support Vector Machine and Random Forest classifier have the drawback of producing 'bad' probabilities and need calibration. There are various calibration approaches like platt scaling, isotonic regression and sigmoid calibration.
- **Stratification Aka Rebalancing** is a cost-sensitive learning procedure in which we modify the data so that the frequency of each class is analogous to its misclafication cost. The desired class frequencies can be achieved through under-sampling, over-sampling and their combination.
  - In **Under-sampling** we select randomly observations from the majority class until the desired class frequencies are achieved by reducing the overall available data.
  - In **Over-sampling** we randomly duplicate examples in the minority class until the desired class frequencies are achieved. A drawback of this approach is that overfitting may occur due to duplicate examples
  - In **Combination** we utilize the under-sampling and over-sampling approaches in the same time combining both of these approaches until the desired class frequencies are achieved.

3.2 Class imbalance

In many cases, after examining the number of observations for each class in a dataset, we conclude that the observations are not balanced. Hence a particular

class may have more observations than the other class in an existing dataset. For example, we have a dataset with 20.000 observations, which correspond to the classes 0 and 1, where the 18.000 observations are of type 0 and the 2.000 of type 1. Therefore, the ratio between the two classes is 18.000:2.000 or 9:1. This imbalanced ratio leads to problems, which do not existing in binary classification cases only, but in multiclass classification cases too.

The problem that arises when dealing with imbalanced datasets is established as the *accuracy paradox*. More specifically, the models are producing high accuracy, but this accuracy is misguiding because it reflects only the underlying distribution of the dataset. The *accuracy paradox* results from the event that the models that have been trained with imbalanced datasets, learn to classify most of the observations of the majority class. Since the test observations, from which the accuracy is quantified, are most of the majority type, the final accuracy will be high, regardless the fact that many observations of the minority class are misclassified. Consequently, the models don't learn to classify correctly the observations that belong to minority classes.

Many techniques have been proposed to handle the class imbalance problem in the literature. For our case, in order to increase the efficiency of our models, compared to the corresponding imbalance-unaware models, we applied the following techniques.

- **Weighting** uses the values of y, namely class 0 or class 1 in binary classification problems, to automatically adjust weights inversely proportional to class frequencies in the input data and achieve to emphasize to the minority values of the corresponding dataset.

- **Cost-sensitive learning** is a method in which a cost-matrix is utilized to weight the observations inversely proportional to their class frequencies and achieve as a result a more balanced outcome.

- **Random under-sampling** is the procedure where we randomly delete examples in the majority class until the distributions are balanced. As a result the classifier will consider equally both the minority and the majority class. A drawback of this approach is that we may lose important observations / examples during the removal procedure.

- **Random oversampling** is the procedure where we randomly duplicate examples in the minority class until the distributions of the two classes are balanced. A drawback is that overfitting may occur due to duplicate examples.

- **Synthetic Minority Oversampling TEchnique (SMOTE)** is an oversampling technique that creates synthetic observation based on the minority classes observations. After the observations are mapped to the feature space, at first, a random observation is selected, and then its k nearest minority class neighbors are found. Afterwards n random neighbors are selected, and a line between those observations and the initial selected

observation are constructed in the feature space. Finally, the new synthetic observations are points that are on those lines. The aforementioned procedure can be replied as many times we want to generate any number of new minority class observations.

- **Borderline SMOTE** is an extension of the classic SMOTE. It produces synthetic observations based on the minority class observations only when their neighborhood contains more than half of the majority examples excluding the case in which all the neighbors are majority examples.
- **NearMiss** is a distanced-based under-sampling technique that extracts observation from the majority class. It resolves which observation of the majority class to keep, and then remove the rest. There are three variations of the technique.
  - In **NearMiss1** the majority class examples with the smallest average distance to the three ***closest*** minority class examples are selected to be kept.
  - In **NearMiss2** the majority class examples with the smallest average distance to the three ***farthest*** minority class examples are selected to be kept.
  - In **NearMiss3**, at first, the k nearest majority class observation for each minority class observation are selected. Afterwards, among these majority observations, the ones with the largest average distance to their three closest minority examples are finally kept.
- **Tomek Links** is an undersampling technique that extracts observations from the majority class, more precisely, to resolve which observations to remove. A Tomek link is considered as a "link" between an observation X and an observation Y that fulfill a specific requirement. The requirement is that observation X is the nearest neighbor of observation Y, Y is the nearest neighbor of X, and X and Y belong to different classes. The majority class observations that belong to a Tomek links are extracted. As result, boundary or noise observation of the majority class are extracted from the dataset.
- **Easy ensemble** is an approach, where at first, we randomly sample with replacement from the majority class as many examples as the minority class. Then we train a boosting model on the union of the above sample and the minority class examples. We repeat the aforementioned steps multiple times and then we combine the resulting models with voting.
- **Cluster-based Sampling** is a random over-sampling procedure in which at first, we cluster the data of each class separately using a baseline clustering algorithm like k-means and then we over-sample all clusters from the minority class and all clusters besides the most frequent one from the majority class. As a result the minority class has the same number of

observations as the majority class and in the same time the clusters inside both of these classes have the same distribution.

3.3 Multi-label Learning

Multi-label classification is a generalization of multi-class classification, which is the single-label problem of categorizing instances into precisely one of more than two classes; in the multi-label problem there is no constraint on how many of the classes the instance can be assigned to. The difference between multi-class classification and multi-label classification is that in multi-class problems the classes are mutually exclusive, whereas for multi-label problems each label represents a different classification task, but the tasks are somehow related.
Unlike normal classification tasks where class labels are mutually exclusive, multi-label classification requires specialized machine learning algorithms that support predicting multiple mutually non-exclusive classes or "labels." For this task, we will the scikit-multilearn[1] python lib, since it was created specifically for multi-label problems.

Many techniques have been proposed to handle the multi-label classification problem in the literature. For our case, we experiment with the models we believed that will produce the best performance and metrics on our datasets. The models are the following:

- **Classifier Chain** constructs a Bayesian conditioned chain of per label classifiers. This chain is an implementation of Jesse Read's problem transformation method with the same name. For L labels, the Classifier Chain trains L classifiers, ordered in a chain according to the Bayesian chain rule. The first classifier is trained on just the input space, while each next classifier is trained on the input space and all previous classifiers in the chain. The default order of the chain follows the same ordering as provided in the training set, i.e. label in column 0, then 1, etc. We used this model with a cross validation grid search, which searched between the following classifiers:
  - **MultinomialNB** or Naive Bayes classifier for multinomial models. The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.
  - **Random Forest Classifier**. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
- **Label Powerset** transforms multi-label problems to multi-class problems. Label Powerset is a problem transformation approach to multi-label

[1] http://scikit.ml/

classification that transforms a multi-label problem to a multi-class problem with 1 multi-class classifier trained on all unique label combinations found in the training data. The method maps each combination to a unique combination id number, and performs multi-class classification using the classifier as multi-class classifier and combination ids as classes. Similarly to the Classifier Chain we used, we performed cross validation grid search for the Label Powerset for a Multinomial Naive Bayes model and an SVM classifier.

  - **C-Support Vector Classification**. The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. The multiclass support is handled according to a one-vs-one scheme.

- **Binary Relevance** transforms a multi-label classification problem with L labels into L single-label separate binary classification problems using the same base classifier provided in the constructor. The prediction output is the union of all per label classifiers. We used this model with Gaussian Naive Bayes as a base classifier.

- **RakelD** or distinct RAndom k-labELsets multi-label classifier. RakelD divides the label space into equal partitions of size k, trains a Label Powerset classifier per partition and predicts by summing the result of all trained classifiers. We used this model using a Decision Tree as the base classifier.

  - **Decision Trees** (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

3.4 Explainable Machine Learning

All data scientists at some point in their careers will encounter a case where the need to explain the decision of a machine learning model will arise. Being able to understand the reasons why machine learning models classify certain items into certain classes or predict a certain value is crucial for gaining insight and providing adequate explanations to others.

Think of an example. Suppose there is a company that uses an MLP model to optimize the sales of their products, in which case we are talking about a recommender system. The system is good and the company is profitable. Nobody knows however the reasoning behind the model's decision. One day the sales department asks of the data science team to help them understand what exactly boosts sales of specific products in order to improve the way they advertise. In order for the data science team to provide adequate explanations,

it needs to find a way to explain the behaviour of their black-box model, since MLP weights are very hard to understand because of their multiple and complex hidden layers. In this case, we are talking about *Explainable Machine Learning* and *black-box interpretation* where the goal is to explain the behaviour of an otherwise uninterpretable model.

Now think of another company, a small news company. The company keeps 2-week records of the activity of users and applies, let's say Latent Dirichlet Allocation, to build vector representations of topic interests for their registered news readers in order to suggest articles of related topics to them, using a cosine similarity metric between user profiles and articles. One day the CEO of this small company has a great idea in order to attract attention to the small company. The idea involves adding a functionality to provide a detailed analysis to its readers of why they are suggested certain articles, as well as a detailed analysis of their predicted topic interests. The readers will also be able to modify their interests by hand, hence manually altering the vector representations of their own profiles to improve the content they are suggested. In this case, the reasoning behind why users are suggested articles is very simple and relies solely on the representation vectors of users and articles. This is a typical case of *Interpretable Machine Learning* and *white-box modeling* where the deployed models can be readily interpreted.

White-box modeling and black-box interpretation are the most common methods available for providing explanations and reasoning. As discussed earlier, our solutions are focused in NLP and text documents, for which we used a number of different algorithms in order to understand the characteristic traits of each label and how different words affect the decision of our algorithms. The algorithms we used can be summarized below.

- **White-box models**, a.k.a interpretable models, comprise simple models, such as Linear and Logistic Regression, and Decision Trees. Despite interpretability being a very hot topic nowadays, the trade-off for creating interpretable models involves reduced predictive power and model complexity. The more complex a model becomes, the less understandable its decisions become to humans. The two main properties of interpretable models involve **(i)** Linearity, and **(ii)** Monotonicity.
  - **Logistic Regression** was one of our implemented white-box models. Logistic regression is very similar to Linear regression, with the difference of a logit (sigmoid) function being appended to its end. In contrast to Linear regression, Logistic regression is suitable for classification, with an extension being possible of using the softmax function instead of the sigmoid for multi-label classification. Logistic regression is the number one interpretable model, due to the fact of having a single linear weight layer, the weights of which can be interpreted as the different importance factors of each input feature.
  - The **Decision Tree** is a very simple binary-tree-like model. The Decision Tree comprises a tree of nodes which are further split into

left and right and contain a classification condition. Each node aims to minimize the entropy at each level by selecting the appropriate feature and a suitable value, in order to classify points in the dataset into the left and right nodes. The Decision Tree ends when all points have been perfectly classified or the maximum depth level has been reached. The levels of the tree contain the most important features in decreasing order, allowing us to deduce the importance of different features.

- **Black-box model interpretation - Random Forest**. As discussed earlier, a trade-off between interpretability and predictive power exists, hence complex, but highly accurate models, can rarely be intrepreted. Our goal in this case is to *explain* the model, instead of trying to interpret it. For this task, simpler surrogate models exist, which can be used to explain the decision of a black-box model. In this work we selected a Random Forest model as our black-box model.
  - **Global surrogate model - Logistic Regression**. Global surrogate models are interpretable models that are trained on all of the predictions in the test set of a black-box model in order to explain its behaviour. As a global surrogate model, we decided to use Logistic Regression once more.
  - **Local surrogate model - Decision Tree**. In contrast to global surrogate models, local surrogate models are used to explain individual predictions, i.e., instances of the test set. Local surrogate models comprise interpretable models that are trained on a number of nearest neighbours of that instance, which are eventually used to explain why the black-box model decided to classify the instance to a specific label.
- **Variable importance** is a useful method commonly used against the phenomenon known as the Curse of Dimensionality. Variable importance helps us deduce the highest contributing features in the decision of a model and use them to build simpler models.
  - **Permutation importance** is one of the different variable importance methods. The permutation importance method is applied to a trained model and ultimately permutes features of the test data. Different predictions and metrics are drawn by various permutations, which the method collects and uses them to define the difference between the permutation metric and the baseline metric. This way, the method can decide which features contribute the most to the decision of the model and assign different weights to them.

4 Results

In this section we discuss the effectiveness of the implemented techniques described in the Section 3. Similarly to Section 3, we begin with Cost-Sensitive learning, follow with Class Imbalance, proceed with Multi-Label learning and end with Explainable Machine Learning. For each of the subsections, we present the results of our implementations, extract metrics, provide detailed plots, showcase comparisons and elaborate on our findings.

4.1 Cost-Sensitive learning

Cost-insensitive machine learning models are trained on a dataset and seek to minimize the error. In the procedure of their fitting, they resolve an optimization problem which seeks to minimize the error. On the other hand, in cost-sensitive learning, the notion of *cost* exist; *The penalty associated with an incorrect prediction*. Consequently, the objective of cost-sensitive learning is to minimize the *total cost* of a model on the training dataset. Different types of prediction errors have different predefined *costs*.

Cost-matrix is a mandatory component in cost-sensitive learning. It reflects the error cost for each type of prediction error. Table 5 shows a cost-matrix, for which the following apply.

1. Costs C(0,0) and C(1,1) are equal to 0. The model do not get penalized when it predicts observation correct.
2. Costs C(0,1) and C(1,0) are defined by domain experts and are not equal to 0. If the "important" class is labeled as "positive", then C(0,1) < C(1,0).
3. Total cost = C(0,1) * FP + C(1,0) * FN, where FP = number of False Positives, and FN = number of False Negatives that the model predicts.

	Actual Positive	Actual Negative
Predicted Positive	C(0,0), TP	C(0,1), FP
Predicted Negative	C(1,0), FN	C(1,1), TN

Table 5: Cost-matrix format.

As for the procedure regarding the training of our models for the cost-sensitive learning case, the first basic step is to determine the cost matrix. Considering the characteristics of our selected SMS spam collection dataset (2.1), it is worth mentioning that a possible misclassification of a spam message as a ham may cost us a little bit more reading time whereas a possible misclassification of a ham message as spam could cost us significantly higher.

In that respect, in our cost matrix in Table 6 we weight a possible ham message misclassification 10 times higher than a spam message misclassification.

We created various cost-sensitive learning approaches namely minimizing the cost function with the BMRC desicion model with and without calibration, example weighting, over-sampling, under-sampling and their combination and tested them in 4 different classifiers namely random forest, logistic regression, support vector classification and multinomial naïve bayes classifier to evaluate the overall performance of our approaches in a more holistic way.

	Actual Ham	Actual Spam
Predicted Ham	0	1
Predicted Spam	10	0

Table 6: The cost-matrix of the cost-sensitive learning case study.

Dataset preprocessing. In order to utilize the SMS spam collection dataset to train our models, at first, we had to preprocess the dataset. Since the dataset regards the NLP field, we applied classic NLP preprocessing techniques. We removed all the punctuation and stopwords, and then vectorized each message in a tf-idf representation. The same preprocessing procedure is applied to the class-imbalance case.

Regarding the **random forest** classifier case study, we can observe from Table 7 that all the proposed approaches except the minimizing the cost function with no calibration approach reduce the cost of the baseline random forest. Both the sigmoid and isotonic calibrations seem to work pretty well producing the highest cost decrease.

Technique	Total Cost
Random forest baseline	78
Weighting	60
BMRC without calibration	102
BMRC sigmoid calibration	51
BMRC isotonic calibration	53
Random undersampling	60
Random oversampling	57
Hybrid sampling	58

Table 7: Cost-sensitive results for the random forest classifier

In the **logistic regression** classifier, the low-cost value of 45 in the baseline logistic regression is highly competitive. Only 2 cost-sensitive approaches decrease even further this value, specifically the example weighting and the over-sampling approach.

Technique	Total Cost
Logistic regression baseline	45
Weighting	27
BMRC without calibration	94
BMRC sigmoid calibration	64
BMRC isotonic calibration	56
Random undersampling	49
Random oversampling	40
Hybrid sampling	45

Table 8: Cost-sensitive results for the logistic regression classifier.

Next, in the **SVM** classifier all the cost-sensitive approaches decrease the baseline cost value of 64, except the example weighting, contrary to the logistic regression case where example weighting was by far the best approach. Undersampling and minimizing the cost with and without calibration outperform the rest of the proposed approaches.

Technique	Total Cost
SVM baseline	62
Weighting	64
BMRC without calibration	40
BMRC sigmoid calibration	55
BMRC isotonic calibration	52
Random undersampling	38
Random oversampling	61
Hybrid sampling	55

Table 9: Cost-sensitive results for the support vector classifier

Following, the baseline **MultinomialNB** approach produces the high-cost value of 88 and leaves plenty room for improvement. All the proposed approaches decrease this amount by a respectable margin with the example weighting and minimizing the cost with isotonic calibration approaches achieving the highest total cost decreases of 58 and 56 respectively resulting in low-cost results.

Technique	Total Cost
Multinomial NB baseline	88
Weighting	30
BMRC without calibration	48
BMRC sigmoid calibration	40
BMRC isotonic calibration	32
Random undersampling	61
Random oversampling	66
Hybrid sampling	79

Table 10: Cost-sensitive results for the multinomial Naive bayes classifier

Overall, our proposed cost-sensitive framework affects all the different baseline classifiers.More specifically,as we can see from Figure 4, in most cases, the isotonic calibration works better than the sigmoid calibration and the example weighting technique is highly effective. Generally, we can state that various classifier - cost-sensitive learning approach couples achieve significant results, concluding in the fact that cost-sensitive learning algorithms need to be tested extensively in various classifiers in order to produce the optimized results.

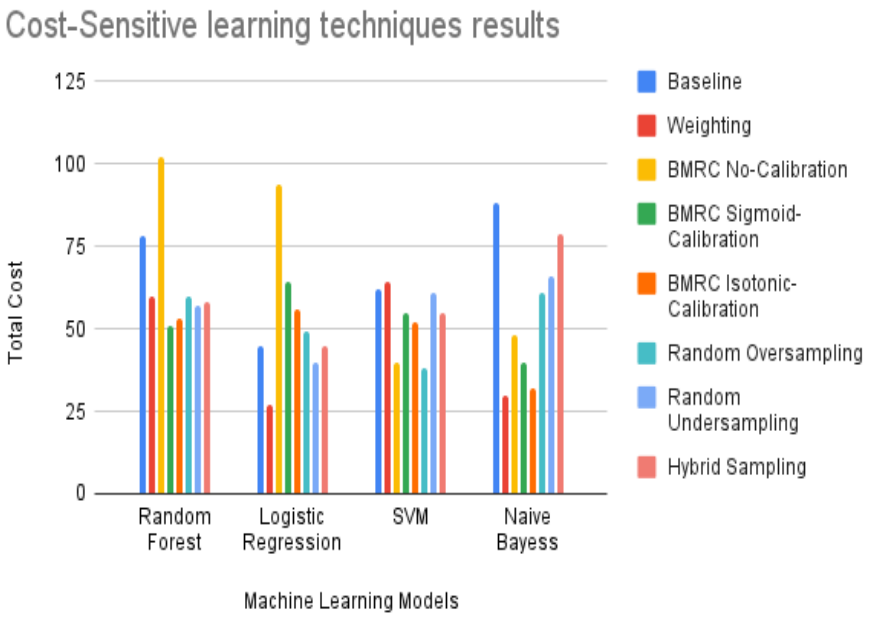


Figure 4: Cost-sensitive learning techniques results

4.2 Class Imbalance

Due to the *accuracy paradox*, in class imbalanced datasets, accuracy is not the primary metric that indicates how good the model performs. Instead, the primary metrics that are suggested are *balanced accuracy*, *G-mean*, *roc-curve area under the curve (ROCAUC)* and *precision-recall-curve area under the curve (PRCAUC)*. Consequently, in Tables 11, 12, 13, which present the results of the adopted class-imbalance techniques and classifiers, except of accuracy, the results of all the aforementioned metrics are presented too.We also included *F1-score* but it should not be considered as significantly as the prior suggested metrics.

If we examine carefully the characteristics of our selected SMS spam collection dataset 2.1, and especially the ham/spam classes ratio that is approximately 6/1, it is worth mentioning that this particular dataset is appropriate for adopting the aforementioned class-imbalance techniques (3.2) and metrics (4.2 par1). In order to have a more holistic view of the corresponding results we employed several classifiers namely multinomialNB, logistic regression and random forest.

Dataset preprocessing. We applied the same preprocessing procedure as we applied in the cost-sensitive learning case (4.1).

Regarding the **multinomialNB** classifier, Figure 5 indicates that the Naive bayes weighting scheme, Nearmiss-2 and cost-sensitive learning achieve the most significant results. We can calculate from Table 11 that the enhancement of the critical g-mean and balanced accuracy metrics is at a rate of approximately 1,5% in these best-performing techniques. This might not seem as an astonishing achievement but it is highly significant if we consider the initial values of 0.949 and 0.9478 of the balanced accuracy and g-mean metrics that the Naive Bayes imbalance unaware model had already achieved.

Technique	Accuracy	Balanced Accuracy	F1	ROCAUC	PRCAUC	G-mean
Imbalanced unaware	0.9820	0.9490	0.9311	0.99	0.97	0.9478
Weighting	0.9742	0.9624	0.9077	0.99	0.97	0.9624
Cost-sensitive learning	0.98062	0.9662	0.9291	0.98	0.96	0.9660
NearMiss1	0.9806	0.955	0.9272	0.97	0.97	0.9542
NearMiss2	0.9813	0.9621	0.93	0.98	0.96	0.9618
NearMiss3	0.9289	0.9318	0.7795	0.98	0.95	0.9318
Random oversampling	0.9734	0.9552	0.9039	0.98	0.96	0.955
Random undersampling	0.9519	0.9541	0.8424	0.99	0.97	0.9541
SMOTE	0.9792	0.9518	0.9218	0.98	0.96	0.951
Bordeline SMOTE	0.9755	0.9543	0.9105	0.98	0.96	0.9538
Tomek Links	0.9806	0.9662	0.9291	0.98	0.96	0.9660
Cluster-based sampling	0.9827	0.9584	0.9351	0.98	0.97	0.9578

Table 11: Class imbalance approaches comparisons in the Naive Bayes classifier case study

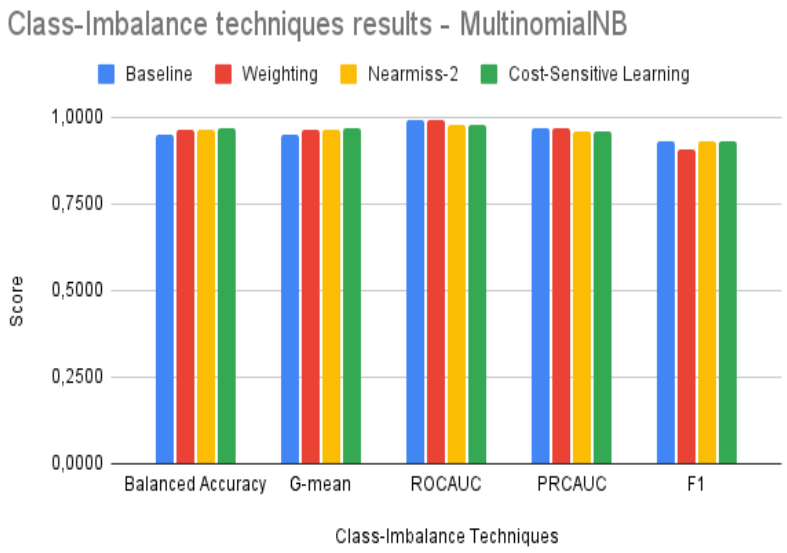


Figure 5: Class-imbalance techniques results for the multinomialNB classifier

Moving on to the **logistic regression** classifier, as we observe in Figure 6 different approaches clinch the top-3 steps in terms of performance this time. Borderline smote, tomek links and cost-sensitive learning outperform the imbalance unaware model by a margin of approximately ∼2% and ∼2,5% in the best-performing cost-sensitive learning case in terms of g-mean and balanced accuracy metrics as we calculated from the Table 12. In some of these cases the accuracy decreases, but this is not a concerning factor for our case study as accuracy is not an appropriate class-imbalance metric.

Technique	Accuracy	Balanced Accuracy	F1	ROCAUC	PRCAUC	G-mean
Imbalanced unaware	0.9806	0.9278	0.9221	0.99	0.96	0.925
Weighting	0.9813	0.9463	0.9282	0.99	0.96	0.945
Cost-sensitive learning	0.9785	0.9514	0.9194	0.98	0.96	0.9507
NearMiss1	0.9612	0.9347	0.8615	0.97	0.93	0.934
NearMiss2	0.962	0.935	0.8637	0.97	0.93	0.9344
NearMiss3	0.9576	0.9394	0.8529	0.97	0.93	0.939
Random oversampling	0.9828	0.9471	0.9333	0.99	0.96	0.9459
Random undersampling	0.9770	0.9415	0.9126	0.98	0.96	0.9403
SMOTE	0.9720	0.9477	0.8976	0.97	0.95	0.9471
Bordeline SMOTE	0.9727	0.948	0.9	0.97	0.95	0.9475
Tomek Links	0.972	0.9477	0.8976	0.97	0.95	0.9471
Cluster-based sampling	0.9791	0.9338	0.9183	0.98	0.95	0.9317

Table 12: Class imbalance approaches comparisons in the logistic regression classifier case study

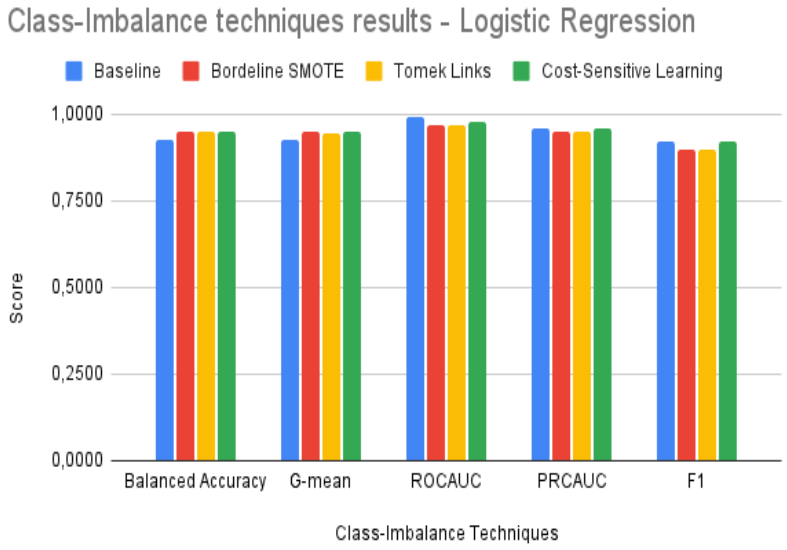


Figure 6: Class-imbalance techniques results for the logistic regression classifier

Considering the fact that both the **logistic regression** and **multinomialNB** imbalance unaware models had already achieved significant results and there was little room for improvement for our class imbalance approaches, we tried to create a weaker imbalance unaware model namely a **weak random forest** classifier tuning it with the small value of maximum depth of 20. The

scope of this relatively weaker than the rest classifier is to evaluate the wider, in this case, enhancement percentage of our class-imbalance approaches metrics results, prove their overall effectiveness and examine the wide margins between the various class-imbalance approaches. As Table 13 indicates all our approaches enhance the key balanced accuracy and g-mean metrics at an impressive rate of approximately 10%. Nearmiss-3, random over-sampling and cost-sensitive learning perform slightly better as we can observe in Figure 7 and prove the overall point that each classifier may be suited better to other class-imbalance approaches. Nevertheless, we can generally state that some approaches like cost-sensitive learning and random over-sampling seem to work well in various classifiers and could be considered as the most reliable.

Technique	Accuracy	Balanced Accuracy	F1	ROCAUC	PRCAUC	G-mean
Imbalanced unaware	0.9526	0.8235	0.7857	0.98	0.94	0.8044
Weighting	0.9734	0.9214	0.8958	0.98	0.94	0.9187
Cost-sensitive learning	0.967	0.9421	0.8802	0.98	0.95	0.9415
NearMiss1	0.9641	0.925	0.867	0.96	0.92	0.9235
NearMiss2	0.9742	0.9263	0.8994	0.97	0.93	0.9240
NearMiss3	0.9734	0.935	0.8991	0.97	0.94	0.9335
Random oversampling	0.9742	0.9263	0.8994	0.98	0.95	0.9240
Random undersampling	0.9677	0.9226	0.8774	0.98	0.94	0.9205
SMOTE	0.9713	0.9224	0.889	0.98	0.94	0.92
Bordeline SMOTE	0.9727	0.9232	0.8939	0.98	0.94	0.9208
Tomek Links	0.9713	0.9224	0.889	0.98	0.94	0.92
Cluster-based sampling	0.9655	0.8897	0.8596	0.97	0.93	0.8836

Table 13: Class imbalance approaches comparisons in the random forest classifier case study

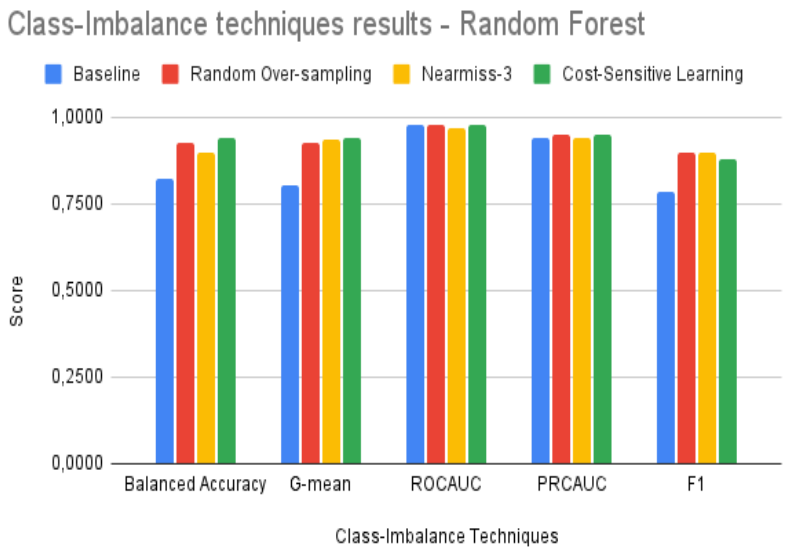


Figure 7: Class-imbalance techniques results for the random forest classifier

In order to present the improvements of our proposed approaches in a more intuitive manner, we created the confusion-matrix in the logistic regression imbalance unaware case study in Table 14 and the confusion-matrix after the adoption of the cost-sensitive learning class imbalance approach for the logistic regression classifier in Table 15. In Table 14 the model misclassifies 27 observations in total whereas in Table 15 it seems that the model works even worse misclassifying 30 observations. But the point is that in the first case we classify well all our majority class observations and misclassify a significant number of observations of the minority class leading to an untrusty classifier because its' high accuracy values are misleading. On the other hand, in the second case we may lose more observations in total but now our model is trustworthy misclassifying those observations in a more balanced way as both the minority and the majority class affect the overall performance of the algorithm in the same degree. Finally, we make the assumption that both the ham and spam observations are treated equally in terms of cost so the fact that we lose more ham observations which were crucial in the cost-sensitive learning case does not play a significant role in this current class-imbalance framework.

	Actual Ham	Actual Spam
Predicted Ham	1206	0
Predicted Spam	27	160

Table 14: Confusion-matrix for the imbalance unaware logistic regression classifier

	Actual Ham	Actual Spam
Predicted Ham	1192	16
Predicted Spam	14	171

Table 15: Confusion-matrix after the adoption of cost-sensitive learning class imbalance approach for the logistic regression classifier

4.3 Multi-label Learning

In this section we present the multi-label learning approach we followed in this work. The subsections that follow are divided based on our dataset. First, we will discuss about the pre-proccessing procedure in order to format our data for the training model. We used four multi-label models in total, i) Binary Relevance, ii) RakelD, iii) Label-Powerset and, iv) Classifier Chain.

Furthermore, in multi-label classification problems regular classification metrics cannot be commonly used, since labels are often imbalanced. A metric worth mentioning here is the Hamming loss, which computes the average Hamming distance between two sets of samples.

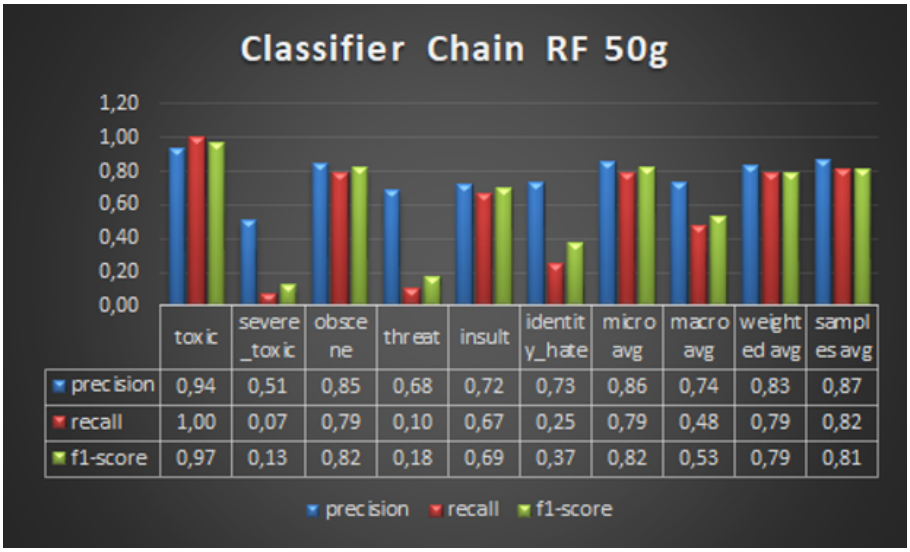
4.3.1 Toxic Comment Classification

The Toxic Comment Classification dataset provides one column of comments and 6 columns of binary labels. In order to use a multi-label technique, firstly we have to better understand our dataset. So, in this preprocessing step we had to find if there were any null values, or unlabelled (0 in all labels). There were none null values, however, 89.83% of our data was unlabelled, and eventually decided to drop the corresponding unlabelled rows. Furthermore, the shape of the character length distribution informed us that most comments were less than 1000 characters, with the most to be up to 500 characters.

As a next step, we used NLP techniques to normalize the text data (comments). The python libraries we used were re and nltk, which helped us set all characters to lower, format all data in same way (*i'm* to *i am*, etc.), we removed all non-alphanumeric values and non-word characters, and removed the stopwords. Then, we needed to vectorize each comment for the training process. For this purpose, we used TfidfVectorizer, since we observed that it produced the best performance in the training process. As parameters for the TfidfVectorizer we used the following: (max_features=10000, stop_words='english', strip_accents='unicode', sublinear_tf=True, token_pattern='w1,'). Because the dataset did not provide a proper testing set, we used the training set to split it into training and testing sets. After data preprocessing, we implemented our selected multi-label techniques.

Classifier Chain The first model we examined was the Classifier Chain. In Figure 8 we see the results of the Classifier Chain. We used a Multinomial Naive Bayes as the baseline algorithm and a Random Forrest Classifier for the comparison task. Also, we let the Classifier Chain search and calculate different

instances for each algorithm. For MultinomialNB we searched for a range of alpha parameter (0.7, 1.0) and for the Random Forest Classifier we searched for three values of the "n" estimator, 10, 20 and 50, based on a different criterion each time (gini or entropy). After that, we used a GridSearch algorithm to search and compare each instance of the algorithms. GridSearch then returned the instance that produced the best accuracy. In our experiments, this instance was the Random Forest classifier with 50 "n" estimator and gini criterion, with an overall accuracy of 0.48. Moreover, the algorithm also produced a Hamming Loss score of 0.13. Due to the fact that Hamming loss calculates the fraction of labels that are incorrectly predicted, we deem the value of the loss as very good. The last evaluation metrics were the average ROC-AUC, which was 0.63, a poor score, and precision_recall_fscore_support with average sampling which resulted in "(0.8743652945526251, 0.8153890395201709, 0.8080579910873232, None)".



	toxic	severe_toxic	obscene	threat	insult	identity_hate	micro avg	macro avg	weighted avg	samples avg
precision	0,94	0,51	0,85	0,68	0,72	0,73	0,86	0,74	0,83	0,87
recall	1,00	0,07	0,79	0,10	0,67	0,25	0,79	0,48	0,79	0,82
f1-score	0,97	0,13	0,82	0,18	0,69	0,37	0,82	0,53	0,79	0,81

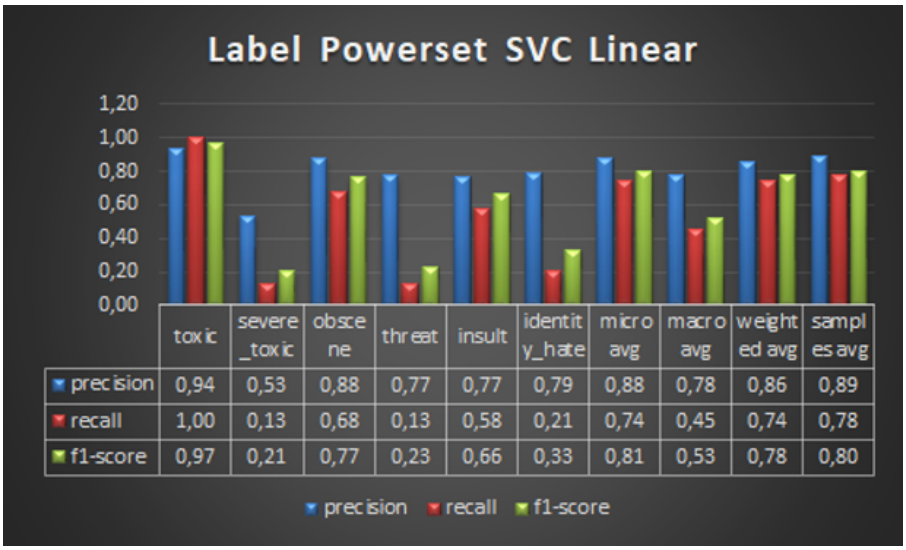
Figure 8: Classifier Chain Results

The results in Figure 8 can be quite confusing, which we attribute to the fact that our data was imbalanced, furtherly shown in Table 16 with the support of each label. Because many comments were labelled as toxic, and few comments as severe_toxic and threat, we saw an over-fit for recall on toxic labels and an under-fit for severe_toxic and threat. We believe that had we more balanced data, we would be able to produce better metrics.

label	support
toxic	3810
severe_toxic	418
obscene	2127
threat	129
insult	2022
identity_hate	378

Table 16: Label support results by Label Powerset

Label Powerset Besides Classifier Chain, we used another multi-label approach for this dataset, Label Powerset. Again, we used the same preprocessing and training as the Classifier Chain. We used Label Poweset with cross validation grid search for comparison between a baseline Multinomial Naive Bayes and an SVM classifier. The baseline Multinomial Naive Bayes was searched for a range of 0.7 and 1.0, for the alpha parameter, and the SVC with comparison between a linear and a sigmoid kerner. The GridSearch algorithm searched for the best accuracy score between the two above algorithms using the Label Powerset. The SVC model with Linear kernel produced the best accuracy (0.5) and its results are presented in Figure 9. These results are not much different than the Classifier Chain. The only difference between those two methods was on f1-score of the sever_toxic and threat labels, since Label Poweset produced 10% more than the Classifier Chain on those two labels. Again we see over-fitted and under-fitted labels due to class imbalance. The Hamming loss score and the macro averaged ROC AUC score was the same as the Classifier Chain. The most accurate metric was precision_recall_fscore_support with average sampling which resulted in (0.8931764029249856, 0.77757374085942, 0.7950152783157466, None).



	toxic	severe_toxic	obscene	threat	insult	identity_hate	micro avg	macro avg	weighted avg	samples avg
precision	0,94	0,53	0,88	0,77	0,77	0,79	0,88	0,78	0,86	0,89
recall	1,00	0,13	0,68	0,13	0,58	0,21	0,74	0,45	0,74	0,78
f1-score	0,97	0,21	0,77	0,23	0,66	0,33	0,81	0,53	0,78	0,80

Figure 9: Results of Label Powerset

4.3.2 TMDB5000

We tried another dataset with more labels and different models to test the scalability of multi-label classification with more labels. The process here was to classify the movie genre in the dataset based on an overview . We tokenized, removed stopwords and lemmatized the overview column in the dataset. As a final step for the data preprocessing we used Bag of Words and tf-idf to vectorize the data.

Since we have 20 different labels and various correlations, we cannot expect our models to yield great results in terms of classification metrics. We can however expect a considerable approximation, which is our main goal. Then, we split our data into training and test set.

Binary Relevance The first model we tested was the simplest one, Binary Relevance with Gaussian Naive Bayes. The results in Table 17 shows the total metrics of the model, which are poor, seemingly randomly predicted, something we already discussed when by seeing the confusion matrix at Figure 15.

Hamming	0.1989103101424979
Accuracy	0.02347024308466052
Precision	0.3792678868118935
Recall	0.5848381431365505
F1	0.41290374429519444
ROC AUC	0.5720369526620227

Table 17: Binary Relevance Metrics

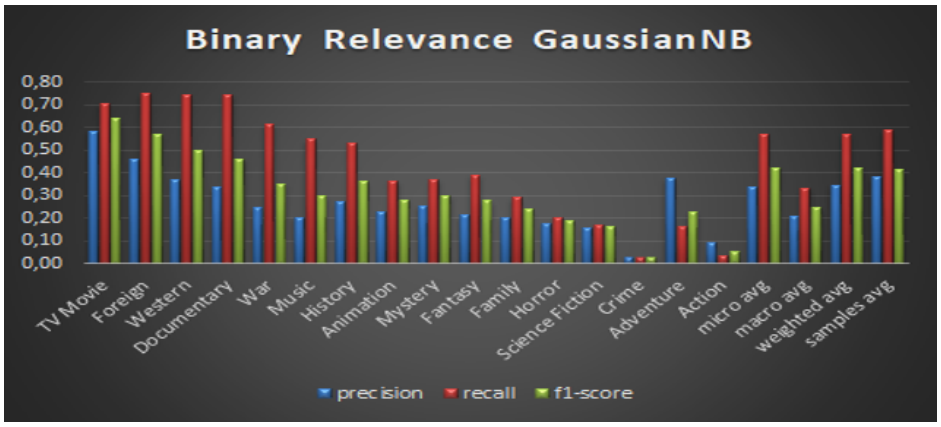


Figure 10: Binary Relevance Label Metrics

Furthermore, we present the prediction metrics for each label in Figure 10. We see, yet again, poor performance, with the highest score being recall of the "Foreign" label with a value of 0.67. The most interesting thing is that the most populated labels ("Thriller","Comedy", "Drama") produced 0 metrics, therefore there are not presented in Figure 10. The only decent metric was precision_recall_fscore_support which produced (0.3792678868118935, 0.5848381431365505, 0.41290374429519444, None).

RakelD The last model we experimented with was RakelD (Random K-Labelsets). The preprocessing here was the same as for Binary Relevance. As a base classifier we used a Decision Tree classifier. The total metrics the model produced are shown in Table 18. We see some poor metrics, nearly randomly predicted. Compared to previous total metrics, here we see a bit increased metrics but worse Recall. We attributed the improved metrics because of our use of Decision Tree classifier instead of Gaussian Naive Bayes.

Furthermore, the label metrics are shown in Figure 11, which are much worse than Binary Relevance. The highest metric was recall for the "TV Movie" label, with a value of only 0.61. Despite metrics being poor, they are more balanced than Binary Relevance. precision_recall_fscore_support with average sampling was the only decent metric with a score of (0.41016445136311025, 0.4059693449886241, 0.37502428185579906, None).

Hamming	0.149
Accuracy	0.041
Precision	0.41
Recall	0.405
F1	0.375
ROC AUC	0.582

Table 18: Overall Metrics with RakelD and Desicion Tree classifier

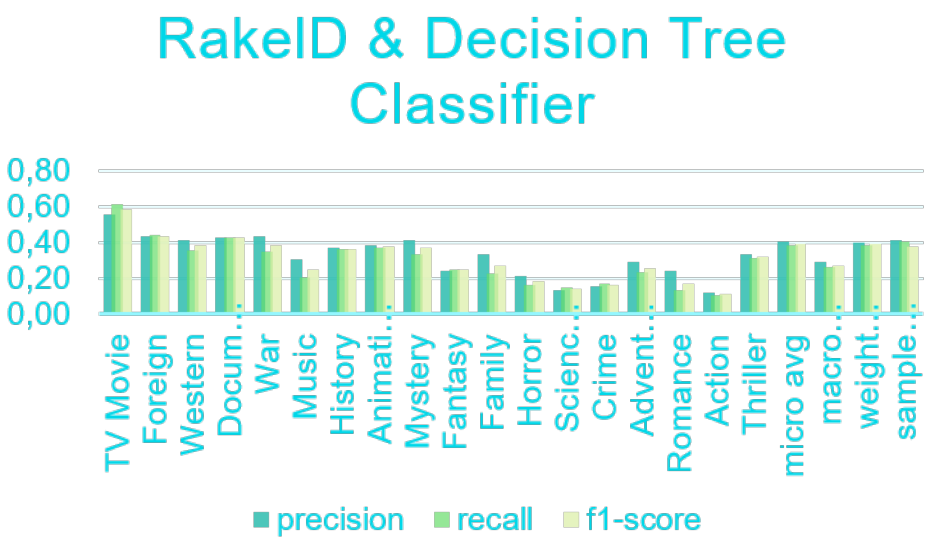


Figure 11: RakelD Labels performance

Based on our results We conclude to the following:

- Performance decreases with higher number of labels
- Despite the high number of labels in the dataset, having a low amount of data was what eventually produced the bad metrics
- We could not find a strong correlations in our predictions
- RakelD with the Decision Tree classifier performed better for the TMDB5000 dataset, while Label Powerset with the SVM classifier having a linear kernel performed better for the Toxic Comments dataset
- We experimented with sample average metrics, which is suggested for multi-label classification, and which produced better values than micro or macro average metrics

4.4 Explainable Machine Learning

Explainability of machine learning is a trending topic nowadays and highly important in the case of enterprise services and applications, mostly due to the right to explanation[2].

In this section we discuss various results from our tests with explainable and interpretable models with the SMS Spam dataset. We must, however, first address the dataset preprocessing procedure, since despite the dataset being used in the *cost-sensitive learning* and *class imbalance learning* fields as well, different preprocessing was applied.

Preprocessing was applied as follows:

- We dropped rows containing null values,
- Removed stopwords,
- Lemmatized and stemmed all words,
- Filtered words having less than 5 occurrences (rare words) or appearing in more than 50% of the messages (common)
- And finally, transformed SMS messages into a tf-idf format.

As discussed in the previous section, regarding white-box models, we used a Logistic Regression and a Decision Tree model. In Figure 12 we can see the top 30 words for ham and spam message classification according to our white-box Logistic Regression model. We can observe that the differences in importance between words become smaller and smaller as we get closer to the bottom.

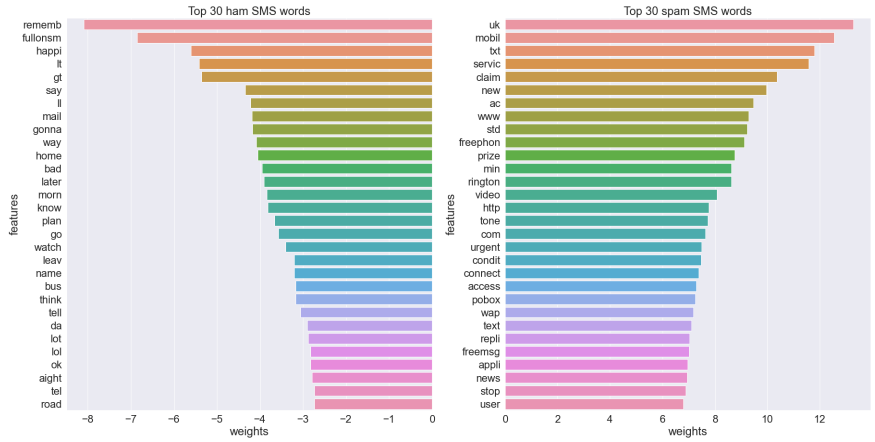


Figure 12: White-box models: The 30 most important words for ham and spam SMS messages, according to the white-box Logistic Regression model

[2]Wikipedia - Right to explanation

It is worth mentioning here, that we used a higher C number of 100 for our white-box and surrogate Logistic Regression models. This number C is the inverse of the regularization strength, which we used in order to strongly distinguish between different word importances.

Similarly to the white-box Logistic Regression model, in Figure 13 we can see the top 30 most important words according to the white-box Decision Tree model. One can observe that the highest ranked words are words typically contained in spam messages. This phenomenon aligns to the fact that certain spam words have a much higher weight than ham words, which is also displayed in Figure 12 for the Logistic Regression model. A notable difference in regard to Logistic Regression is the fact that we do not have negative (ham) or positive (spam) word importances, but instead absolute values of the different importances of words. Another observation is that word importances are not as uniformly distributed as in the Logistic Regression model, which is explained by the fact that in this case the model is structured as a tree, hence top level features are fewer and therefore much more heavily weighted compared to features of other levels.

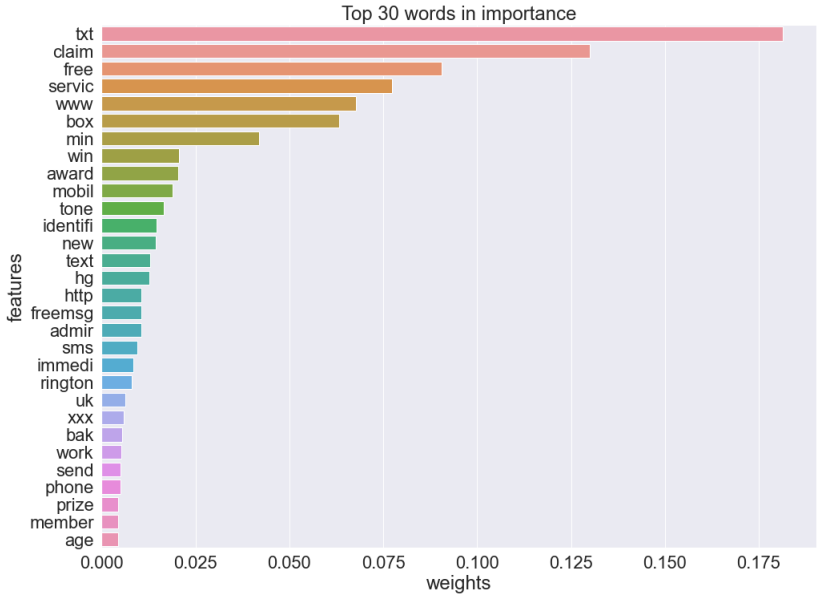


Figure 13: White-box models: The 30 most important words, according to the white-box Decision Tree model

Next up, we trained a black-box Random Forest model, which we attempted to explain. We used surrogate models for this task; a Logistic Regression model, as a global surrogate model, and a Decision Tree, as a local surrogate model. Surrogate models are trained on the predictions of the black-box model, using an accuracy metric to measure the percentage of equally predicted instances.

The accuracy metric in this case is dubbed *Fidelity*.

Global surrogate models are trained on the predictions of black-box models, of all the test instances. In Figure 14 we can see the top 30 words for ham and spam messages in a similar way as in Figure 12. As we can observe, the figures are very similar, with the top 30 words aligning almost perfectly.

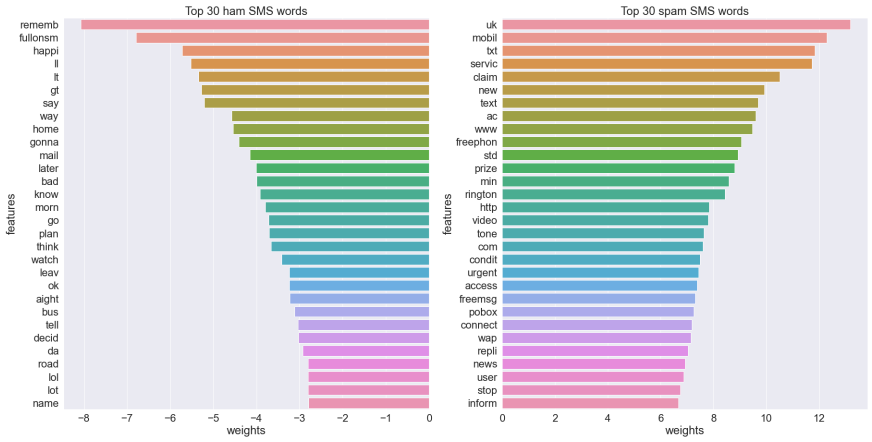


Figure 14: Black-box interpretation: The 30 most important words for ham and spam SMS messages, according to the global surrogate Logistic Regression model

On the other hand, local surrogate models obtain an instance from the test set and use a number of nearest neighbors of that instance in order to try and explain the decision of the black-box model around that instance. As a local surrogate model we used a Decision Tree. The tree was trained on the 100 nearest neighbours of the test instance with index 3 (2, starting from 0). We decided to use this instance due to the fact it belonged to the minority spam class. In order to reproduce obtaining the same instance every time, we used the same seed for splitting the data into training and test sets each time.

The procedure of training the local surrogate model was done as follows. After having trained the black-box Random Forest model, we:

- Trained a kNN classifier on the training data with their predicted labels, using a number of 100 neighbours,
- Selected we third instance in the test set, classified in the minority spam class,
- Obtained the 100 neighbours of the test instance using the kNN classifier,
- And trained a white-box Decision Tree model on its neighbours.

A detailed representation of the local surrogate Decision Tree model can be seen in Figure 15. As one can observe, the way the model classifies neighbouring

instances is very different compared to the other models we have seen so far. Despite this, a perfect fidelity score of 1 was achieved for the neighbours of the test instance.

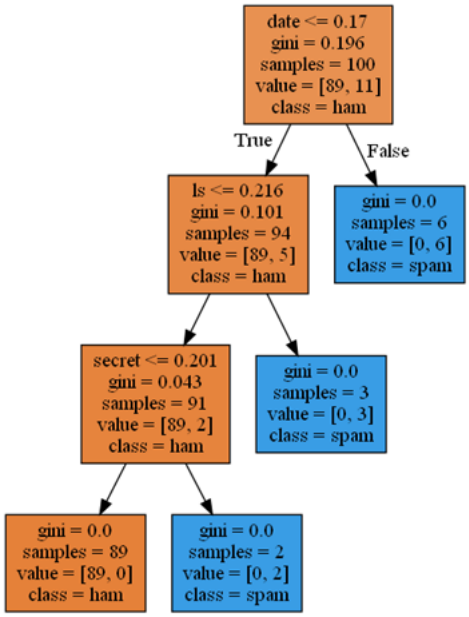


Figure 15: Black-box interpretation: The decision structure plot of the local surrogate Decision Tree model

As our final implementation, we used the Permutation Importance method on a new Logistic Regression model. Permutation Importance permutes different features in the test set and extracts accuracies for different permutations, which are then compared to the accuracy obtained using the original data of the test set. The higher the deviation of the accuracy by permuting a feature, the higher the weight it is assigned to it. This way we can extract different word importances and rank them based on their assigned weights.

In Figure 16 we can see the top 10 most important words for ham and spam classification, according to Permutation Importance. The figure can be interpreted as follows:

- The first number displays the reduction in model performance by the reshuffle of that feature
- The second number is the measure of randomness of the performance reduction for different reshuffles of the feature

We can observe that despite spam words mostly aligning with the models we have seen this far, ham words vary much more, while maintaining much lower weights.

0.0056 ± 0.0014	txt
0.0042 ± 0.0011	win
0.0039 ± 0.0007	text
0.0036 ± 0.0009	repli
0.0033 ± 0.0007	min
0.0029 ± 0.0009	privat
0.0029 ± 0.0009	code
0.0027 ± 0.0006	expir
0.0026 ± 0.0007	redeem
0.0024 ± 0.0011	identifi
-0.0003 ± 0.0007	direct
-0.0003 ± 0.0007	pound
-0.0004 ± 0.0011	week
-0.0004 ± 0.0007	day
-0.0006 ± 0.0014	phone
-0.0006 ± 0.0011	tone
-0.0007 ± 0.0000	wan
-0.0007 ± 0.0000	meet
-0.0007 ± 0.0000	cal
-0.0013 ± 0.0006	think

Figure 16: The top 10 most important words for spam and ham SMS messages, according to the Permutation Importance method

Overall, we can conclude in the following:

- All of the models displayed similar behaviour, with the exception of the local surrogate Decision Tree model, which used completely different words for classifying messages.
- The local surrogate model can be improved by increasing the number of neighbours or by using a larger dataset, since the dataset we used was rather small.
- For spam messages, the most common words identifiers were
  'txt', 'win', 'text', 'repli', 'min', 'privat', 'code', 'expir', 'redeem', 'identifi', 'servic', 'claim', 'new', 'free', 'ac', 'www', 'freephon', 'prize', 'rington', 'mobil' and 'uk'
  in non-specific order, all of which are commonly met in spam messages.
- For ham messages the most common word identifiers were
  'rememb', 'fullonsm', 'happi', 'it', 'gt', 'say', 'll', 'mail', 'gonna', 'way', 'home', 'bad', 'later', 'morn', 'know', 'plan', 'go', 'watch', 'leav', 'name', 'bus', 'think', 'tell', 'da', 'lol', 'ok', 'aight', 'tel', 'road', 'rememb', 'think', 'lot', 'direct', 'pound', 'week', 'day', 'phone', 'tone', 'wan', 'meet' and 'cal'
  in non-specific order, all of which are everyday common words.
- Spam message words displayed much higher weights/importance compared to ham message words.

- Most models agreed on spam words while ham words varied across models.

5 Conclusion

After having examined all these different topics, techniques, models, as well as datasets, we conclude that there is no "golden solution", since different solutions produce different results, better tailored to the needs of the current task. Therefore, the solution that one must select for a given problem depends solely on the task at hand, the domain to be applied and the characteristics of the available dataset. Consequently, analysis and experimentation is required for the respective "golden solution" to be found for each case.