Feature Selection in Imbalance Domain Learning Problems: A Case Study on Scrapping of Tires

Pedro Unas
Master in Data Science
Department of Computer Science
Faculty of Sciences, University of Porto
2022

Supervisor
Rita P. Ribeiro, Assistant Professor, Faculty of Sciences, University of Porto

External Supervisors
José Luís Mourão, Lúcia Moreira

MSc

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Sworn Statement

I, Pedro Manuel Moreira Unas, enrolled in the Master Degree Data Science at the Faculty of Sciences of the University of Porto hereby declare, in accordance with the provisions of paragraph a) of Article 14 of the Code of Ethical Conduct of the University of Porto, that the content of this dissertation reflects perspectives, research work and my own interpretations at the time of its submission.

By submitting this dissertation, I also declare that it contains the results of my own research work and contributions that have not been previously submitted to this or any other institution.

I further declare that all references to other authors fully comply with the rules of attribution and are referenced in the text by citation and identified in the bibliographic references section. This dissertation does not include any content whose reproduction is protected by copyright laws.

I am aware that the practice of plagiarism and self-plagiarism constitute a form of academic offense.

Pedro Manuel Moreira Unas

16th of December 2022

Abstract

In the tire-producing factories of the company Continental, one of the major issues they face is the scrapping of tires which contributes to increased waste and loss of profit. The data that can be used to determine whether a tire should be scrapped preemptively is high-dimensional and hard to interpret. In this work, we study how filter out key features and sampling can produce models to help better determine if a tire should be scrapped. We develop a pipeline that comprehends a two-stage approach for the pre-processing strategies: feature selection and sampling. Through an experimental study on benchmark data and real data, we study how the order of application of these strategies can impact the performance of models on the prediction of scrap tires.

Keywords: Industry 4.0; Data Science; Machine Learning; Feature Selection; Sampling; Tire Production.

Resumo

Nas fábricas de produção de pneus da empresa Continental, um dos principais problemas que enfrentam é o descartar de pneus, o que contribui para o aumento do lixo causado pela produção e a perda de lucros. Os dados que podem ser utilizados para tentar determinar antecipadamente se um pneu deve ser descartado são grandes em dimensão e difícieis de interpretar. Neste trabalho estudamos como filtrar *features* chave e como o uso *sampling* pode potencialmente produzir modelos que ajudem a melhor determinar se um pneu deve ser descartado. Desenvolvemos uma pipeline que utiliza uma abordagem de dois passos para as estratégias de pré-processamento: *feature selection* e *sampling*. Através de um estudo experimental em dados *benchmark* e dados reais, estudamos como a ordem de aplicação destas estratégias pode ter impacto na *performance* dos modelos e na previsão da descartagem de pneus.

Palavras-Chave: Indústria 4.0; Ciência de Dados; Machine Learning; Feature Selection; Sampling; Produção de Pneus.

Acknowledgments

I would like to thank Rita P. Ribeiro for her supervision of this dissertation and the countless hours of meetings and discussions during the dissertation's completion. I would also wish to thank Lúcia Moreira and José Luís Mourão who dedicated their time, effort and knowledge of the tire production business and the field of Data Science and were always present whenever necessary.

To my parents, whose support, love and interest were always present in my studies and in particular this dissertation.

A special thank you to Inês, my girlfriend, who motivated and encouraged me to keep going, even at the most difficult moments.

Contents

List of Tables

List of Figures

Acronyms

.csv Comma-Separated-Value

APS Air Pressure System

AUC Area Under Curve

AWS Amazon Web Services

CNN Condensed Nearest Neighbors

CPU Central Processing Unit

CRISP-DM CRoss Industry Standard Process for Data Mining

EC2 Elastic Compute

FCQ F-test correlation quotient

FS Feature Selection

kNN k-Nearest Neighbors

MCC Matthews Correlation Coefficient

mRMR Maximum Relevance and Minimum Redundancy

MI Mutual Information

NDA Non-Disclosure Agreement

RAM Random Access Memory

RFE Recursive Feature Elimination

ROC Receiver Operating Characteristic

ROS Random Oversampling

RUS Random Undersampling

S2N Signal 2 Noise Ratio

SBFFS Sequential Backward Floating Feature Selection

SBFS Sequential Backward Feature Selection

SFFFSS Sequential Forward Floating Feature Selection

SFFS Sequential Forward Feature Selection

SFS Sequential Feature Selection

TN True Negatives

Chapter 1

Introduction

For any for-profit company, one of its main goals is to improve its profit margins and reduce any type of wasted or scrap being produced. In the case of tire production, tens of millions of tires are manufactured each year, and with this vast amount of production, there is bound the be a large number of tires being scrapped each year for several reasons. Reducing the percentage of scrap would be beneficial for the company in two major ways: first, it would mean less raw material being wasted, decreasing the overall cost of producing each tire, and secondly, it would reduce the company's carbon footprint as less garbage would be produced something critical given the current environmental concerns.

The main problem the company faces however is that there is no real way to predict if a tire is scrapped or not until it reaches the end of its production and the decision can be made using mostly a tire's weight compared to the target weight, where it is decided if it should move forward or be scrapped. It would be in the company's best interest to develop a way to preemptively scrap a tire if necessary to avoid waste of time and resources.

1.1 Objectives

In this dissertation, our overall goal is to resort to machine learning techniques to leverage production data that is currently collected to help with the prediction of scrapping tires. Still, two major data problems pose additional challenges to the machine learning task: the high-dimensionality and the imbalanced domain. Sensors and operators record data and thus suffer from high-dimensionality and data quality issues. At the same time, the proportion of scrap tires is much smaller than the all-produced tires. The ultimate goal of this case study is to create a tool that tackles these challenges indicating a set of features that will be selected with streaming data to determine which tires should be scrapped earlier in the process and deployed in a real-time scenario.

Nonetheless, it is important to study beforehand how different machine learning strategies can help in this task. For that purpose, in this dissertation, our objective is to analyze how

different feature selection and sampling techniques impact the performance of models if taken in a different order as pre-processing steps. With that intent, we design an experimental comparison on benchmark data and on real data from the scrapping tires case study.

1.2 Organization

This dissertation is organized as follows. In Chapter 2, we showcase some fundamental concepts necessary for the work developed, highlighting the different types of feature selection and sampling techniques. We also briefly overview the tools used to develop the work. In Chapter 3, we present the preliminary study of our proposed pipeline on benchmark datasets. In Chapter 4, we apply the developed pipeline to the data provided by the company. It addresses several issues related to the business and data and discusses the obtained results. Chapter 5 concludes with an overall analysis of the study and point possible future work directions.

Chapter 2

State of the Art

In this chapter, we introduce some of the key concepts surrounding this work regarding feature selection and imbalanced learning, highlighting some state-of-the-art methods. We finish by referring to the tools used in the implementation.

2.1 Feature Selection Methods

2.1.1 Overview

The *curse of dimensionality*, a term created by Richard Bellman in 1957, was coined to express the difficulty of using brute force to optimize a function with too many input variables. Currently, the meaning of this curse can be applied to high dimensionality datasets whose sheer size presents complications when working with them. As the dimension of the data increases, it tends to become sparser, making it more difficult for machine learning models to separate different clusters of data, leading to worse performance. Another issue in data with many features is overfitting, where a model will be well fitted to the training data but will not perform well when new data deviates from the data it has seen during the training process.

Data with high dimensionality also hinders computational performance since as the number of features and entries increases, so does the memory required to store it and the computational horsepower to perform tasks, such as training a machine learning model. Reducing the number of features can be important when working with real-life datasets since they can have many features. There are two major fields of work in this area: feature extraction and feature selection. The first transforms the original high-dimensionality feature set into a dimensionality set of new features based on the data and will not be explored further in this study. Feature selection takes the original set of features and removes a given number, with the features and the number of features removed depending on the method one uses.

The advantages of reducing the dimensionality of the data are mostly centered around computational performance, both in terms of training speed and reducing the memory required

to store the data. However, it can also help to train models with less likelihood to overfit as models trained with high dimensionality data can have subpar performance on new, unseen data.

Feature selection can be used for both regression and classification problems, and its goal is always the same, to select a subset of features that can produce an approximation of the target value in regression problems or predict the correct class in classification problems. Feature selection methods have three broad categories: filter, wrapper, and embedded, and the feature selection process varies depending on what category the method being used belongs to. It is better to divide the data into a train and test set in all three categories, but from there, the process diverges.

In filter methods, the selection process is not related to the training portion of the pipeline as features are selected based on the data itself and its relation to the target variable to determine the importance of a feature. These methods are, in most cases, the most efficient process speed but can cause some machine learning algorithm loss of performance as the selected features are not tailored to any specific algorithm. Filter methods first rank all the features via some evaluation criteria, which varies from method to method, before filtering out the bottom-ranked features, which may vary from removing a given number of features or removing features whose importance is below a certain threshold.

Wrapper methods, unlike filter methods, use the performance of a machine learning model to measure the quality of the selected subset of features. The specifics will vary, but wrapper methods have two big steps in their pipeline that are repeated until a stopping criterion is reached. The first step is to generate a subset of features to train and test a model with, the second step is to evaluate the performance of a model on that subset of features. The stopping criteria can be a specific number of features to select, of which the best subset can be selected, or stopping when there are no more performance gains, similar to early stopping in neural networks. Compared to filter methods, the time it takes to produce the selected subset of features in wrapper methods can be quite large in particular on datasets with high dimensionality.

Embedded methods combine the positive qualities of filter and wrapper methods by integrating the feature selection process in the machine learning portion of a given method. A typical embedded method will contain ways to quantify the importance of a feature and ways to select the features it considers most important, removing redundant or unimportant features. This process helps fix the wrapper's major downside, which is the time taken to retrain the machine learning algorithms constantly.

2.1.2 Maximum Relevance and Minimum Redundancy

Initially proposed by Peng et al. [19] Maximum Relevance and Minimum Redundancy (mRMR) are a type of filter feature selection methods that try to select features taking into account their relevance in predicting the target variable and the redundancy of the feature itself. There are some methods based on this principle, with Zhao et al. [26] providing an overview of a selected

number of methods as seen in Table 2.1. The various methods differentiate themselves in the way they calculate the two parts of the problem, the relevance, and the redundancy, as well as the scheme they combine the two values.

Table 2.1: Different mRMR methods [26], based on Relevance, Redundancy and Scheme. The Scheme is Difference when Relevance - Redundancy and Quotient when Relevance / Redundancy.

Method	**Relevance**	**Redundancy**	**Scheme**
MID	Mutual Information	Mutual Information	Difference
MIQ	Mutual Information	Mutual Information	Quotient
FCD	F Statistic	Correlation	Difference
FCQ	**F Statistic**	**Correlation**	**Quotient**
FRQ	F Statistic	RDC	Quotient
RFCQ	Random Forests	Correlation	Quotient
RFRQ	Random Forest	RDC	Quotient
RF	Random Forests	N/A	N/A

For this work, we selected the FCQ (F-test Correlation Quotient) proposed by Ding and Peng [5], which uses the F-test value between each feature and the target variable to calculate the relevance of the feature and calculates the Pearson correlation of each feature compared to the others to calculate the redundancy of that feature before calculating the quotient of those two values, as shown in Equation 2.1.

$$f^{FCQ} = F(Y, X_i)/[\frac{1}{|S|}\sum_{X_s \in S} \rho(X_s, X_i)] \tag{2.1}$$

where Y is the target variable, X_i is the feature, S is the set of features, F the F-test score between the target variable and a given feature and ρ the correlation between a given variable and the rest of the variables in the feature set.

Starting with an empty set of selected features, this method will iteratively calculate the FCQ score of each non-selected feature and select the feature whose score is the largest until the given number of features is selected (cf. Figure 2.1).

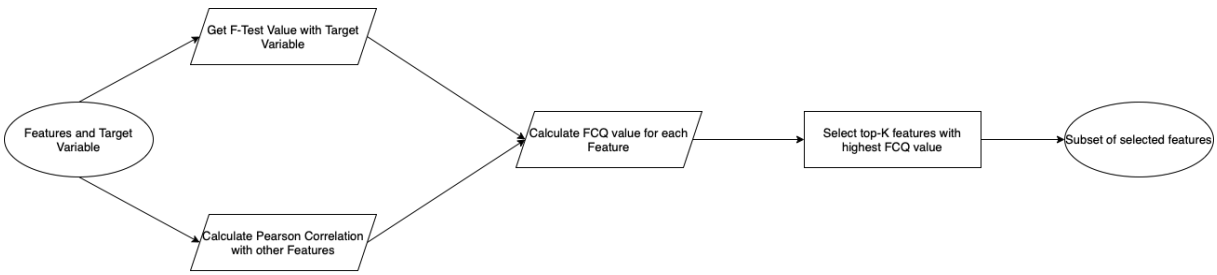

Figure 2.1: F-test Correlation Quotient (FCQ) Flowchart.

This method was used as it is constantly among the top performers in terms of AUC (Area

Under Curve) in [26] and is computationally efficient, as it calculated both the F-test value and Pearson correlation values for each feature before starting the iterative part.

2.1.3 Mutual Information

Mutual Information (MI) is used to measure the dependence between two random variables by using the concept of entropy, a measure of randomness or surprise that tries to quantify the how deterministic a variable is and how much information there is in it. Its values range from 0 to 1 where lower values indicate more certain events, i.e. if there is a 100% chance of an outcome and 0% of the contrary outcome then the entropy value will be 0, and more random events scoring higher entropy, i.e. a balanced coin flip has an entropy value of 1.

Mutual information was first proposed in 1987 by Claude Shannon [22] with the term Mutual Information only being given later by Robert Fano [13] and was originally used only for datasets with discrete variables. The work done in [20] allowed the use of Mutual Information in datasets where the non-target and target variables can be discrete or continuous. It measures the difference between the entropy of a feature and the entropy of a feature given the target variable, as can be seen in Equation 2.2 with an overview visible in Figure 2.2.

$$MutualInformation(X, Y) = H(X) - H(X|Y) \tag{2.2}$$

where $H(X)$ represents the entropy value of a given feature and $H(X|Y)$ the entropy value of the feature given the target variable.

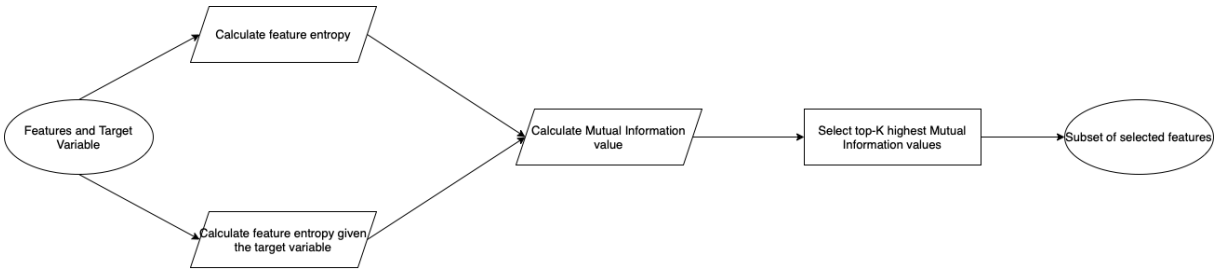

FIGURE 2.2: Mutual Information Flowchart.

2.1.4 Signal 2 Noise

Signal 2 noise ratio (S2N) is a measure used in signal processing and statistics to describe the ratio of a signal to the amount of noise in the signal. In the context of feature selection, S2N can be used as a metric to evaluate the relevance of a given feature in relation to the outcome of interest. For example, in a dataset used to predict the stock price of a company, the S2N of a given feature could be used to determine whether that feature is a strong predictor of the stock price, or whether it is dominated by noise and therefore not useful for prediction. Generally,

features with a high S2N are considered to be more relevant and useful for prediction, while features with a low S2N are less useful and may be removed from the dataset.

2.1.5 Boruta

Boruta is a wrapper filter selection method proposed by Kursa & Rudnicki [14] back in 2010 with a differentiating premise where instead of measuring the performance of features against other features, in Boruta features compete against randomized versions of themselves, which are called "shadow features". After creating these random "shadow features" they are appended to the dataset creating a new one with twice the number of features of the original. The method employs the Random Forest Classification algorithm to measure the performance, fits the new dataset and target variable to the algorithm, and retrieves the feature importance values of each feature, only considering that a feature is useful then it must be more important than the best of the randomized features. To make sure that a feature actually matches this idea it runs a number of iterations of the method and records each time a feature performs better than the best shadow feature as a "hit", as can be seen in Figure 2.3 where a single run of Boruta is showcased on a synthetic dataset.

	age	height	weight	shadow_age	shadow_height	shadow_weight
feature importance %	39	19	8	11	14	9
hits	1	1	0	-	-	-

FIGURE 2.3: Example of a single run of Boruta on a synthetic dataset [18]

The selection criteria for this method can vary, with the original proposal using a two-sided test of equality with the maximum scoring shadow feature to determine if a feature is important or not, removing those that are considered non-important. In our implementation, as we wanted to select a certain number of features, instead of using the two-sided test of equality we instead recorded the number of hits and the average feature importance value recorded in the Random Forest Classifier model and selected the top-K features with the highest number of hits, using the average feature importance value in case of a tie (cf. Figure 2.4).

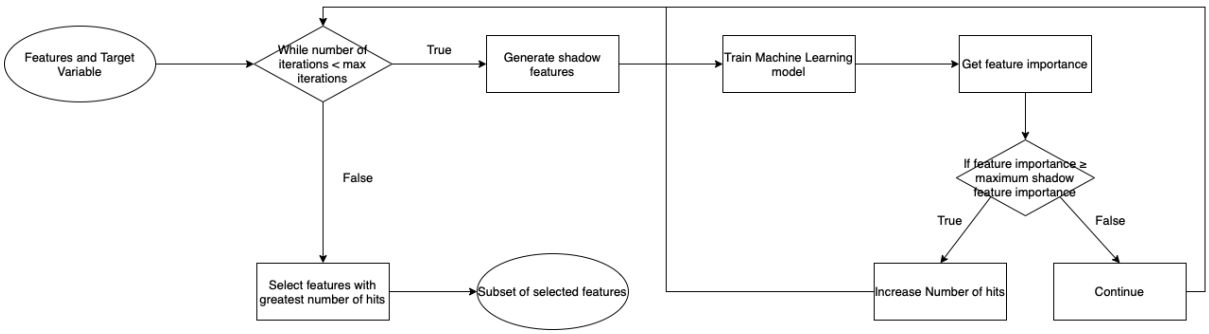

FIGURE 2.4: Boruta Flowchart.

2.1.6 Sequential Feature Selection

Sequential Feature Selection (SFS) is a family of wrapper methods that perform a greedy search to select a certain number of features proposed by Ferri et al. [6]. The two most used Sequential Feature Selection methods are Sequential Forward Feature Selection (SFFS) and Sequential Backward Feature Selection (SBFS), whose approach is the opposite of each other, with SFFS starting with an empty set of features, and at each iteration, it selects the feature that maximizes the cross-validated score of a given estimator, with the method stopping when the given number of features to select is reached. On the other hand, SBFS starts with the full set of features and at each iteration removes the feature that provides the biggest increase in the cross-validated score of a given estimator, i.e. the method calculates for each feature the cross-validated score of the estimator without that given feature.

A more complex variation for both Sequential Forward Feature Selection and Sequential Backward Feature Selection introduces a floating concept where features that were once removed from consideration can be added back to the selected subset of features. In Sequential Forward Floating Feature Selection (SFFFS) after the original step of selecting the best feature is complete, a second step is done where we can remove already selected features from the selected subset if they improve the performance of the model when removed. This step does not result in the removal of a feature if there is no performance gain or if the number of selected features is two. In Sequential Backward Floating Feature Selection (SBFFS) the new step can bring back a removed feature if it improves the performance of the model with the current subset of selected features. Just like in SFFFS it not is mandatory to add a removed feature if the performance of the model is not improved or the number of selected features is equal to two. The obvious disadvantage of the floating variations is the increased computational hit of having to train new models for each feature in the second step.

2.1.7 Recursive Feature Elimination

Similar to Sequential Backward Feature Selection discussed in the previous subsection, Recursive Feature Elimination (RFE) [7] is a simple to use wrapper method that starts with the full set of features in the dataset and, as the name suggests, recursively removes features until only a given number of them remain. Much like in the SFS methods, it is necessary to train a model to know which features to remove but, unlike in SFS methods, RFE requires the use of models that calculate the importance of a feature, like Random Forest, since the performance of the model is not measured but instead the importance of each feature according to the model, similar to what Boruta does.

But unlike Boruta there is no comparison with other features or “shadow” features with the logic of RFE simply removing the features that have the lowest importance value according to the model. This is done until there is a subset of features equal to the required number. Much like most wrapper methods the main downside of RFE is the necessity to re-fit the model at

each iteration, which depending on the current number of features, can be very computationally expensive.

2.2 Imbalanced Learning

One of the major issues that arise when working on classification problems with a real-world dataset is the imbalance in the distribution of the classes in the target variable. Imbalanced domains pose challenges for machine learning algorithms because the most relevant cases are underrepresented in the training set. Standard machine learning algorithms try to optimize the performance in the overall domain. This means that the model will disregard the performance in rare cases as they have a residual impact on the overall performance. Nonetheless, such models are useless from the domain perspective.

Strategies to tackle this type of problem include data-level, algorithm-level and hybrid approaches that ultimately bias the models for being accurate at predicting the less frequent but more relevant, target values. Data-level approaches act on the data distribution before giving it as input to any machine learning algorithm. Algorithm-level approaches act on changing the preference criterion of the machine algorithm that focuses on the performance in rare cases. Many approaches have been proposed that combine the two. Because data-level approaches allow for the use of any machine learning algorithm, they are far more explored within the research community, like in Li, et al [15]. For the same reason, this is also the approach we will use in our study and will detail more.

Another very important topic when tackling an imbalanced domain learning problem is the choice of appropriate evaluation metrics to estimate the performance of models. In the case of binary classification tasks, a specific set of metrics is commonly used, such as precision, recall, AUC, and MCC, among others. Precision calculates the ratio of true positive values predicted concerning the total number of predicted positive values. Recall calculates the ratio of positive predictions concerning all the positive examples. The Area Under Curve (AUC) uses the Receiver Operating Characteristic (ROC) curve and calculates its area to measure how well the model can separate a positive example from a negative example, considering multiple thresholds. F1-Score is another commonly used metric that combines precision and recall by the harmonic mean to leverage the tradeoff of these two metrics. Its idea is taken further by Matthews Correlation Coefficient (MCC) [17] that considers true positives and negatives and false positives and negatives, meaning that it can be used even for highly imbalanced data.

2.2.1 Data-level Approaches Overview

A common approach to counteract highly imbalanced data is to sample the data to remove or add entries to, most commonly, produce equal distributions or classes or another ratio that one desires. There are two major techniques in data sampling, undersampling and oversampling. In

the next two subsections, more detail will be given on both major groups.

A commonly used metric to measure the imbalance of the dataset is the imbalance ratio. The notation of the imbalance ratio varies but in this work, we consider the imbalance ratio as the quotient of the total number of entries in the minority over the total number of entries in the dataset as shown in Equation 2.3. Using this metric, a binary dataset can be considered balanced if its ratio is 0.5.

$$imbalance = \frac{m}{N} \tag{2.3}$$

where m represents the number of entries in the minority class and N the total number of entries

2.2.1.1 Undersampling

Undersampling methods remove entries in a dataset to balance the classes. These types of methods are traditionally less used as the loss of information is a concern that comes with removing entries in the data. However if the dataset one is working it is large enough, it can be more or less cut down mainly for computational reasons and remove redundancy in the data that is sure to happen with big data. In most undersampling techniques applied to binary classification problems, the more prevalent class will have some of its entries removed until the target class balance ratio is reached, usually 0.5 This means that if a binary classification dataset has ten thousand entries, with the majority class being in nine thousand of them and the minority class being the rest one thousand then if we want to reach a 0.5 ratio, the majority class will lose eight thousand entries until we are left with a two thousand entry dataset. The logic by which entries are removed varies from method to method, varying from totally random to using distance metrics as we will see next.

2.2.1.2 Oversampling

Contrary to undersampling, oversampling adds new entries to the dataset to balance the classes. This is more commonly used to increase the level of information present by creating synthetic data. The major drawback of oversampling is that the new entries, however similar to the existing ones, are not real which means that oversampling may not give any benefits even in highly imbalanced datasets. Another drawback is that the larger dataset created by oversampling will hinder computational performance. For the majority of oversampling techniques in binary classification problems, the methods will create fake entries of the minority class until the desired balance ratio is reached, so in the same ten thousand entry dataset with a 0.1 imbalance ratio, the minority class will this time gain eight thousand entries, creating a new dataset with eighteen thousand entries and a 0.5 ratio. Like undersampling, the sampling logic varies from method to method, with some key methods being explored as will see next.

2.2.2 Random Undersampling (RUS) and Oversampling (ROS)

The most basic type of sampling possible is random sampling, which can be applied to undersampling and oversampling. Random Undersampling (RUS) simply removes entries from the majority class or classes until the desired ratio is achieved. This method does not employ any type of heuristic. The biggest downside is that it can easily remove entries containing important information and result in a more noisy dataset, making it harder to apply a classifier further down the pipeline.

Random Oversampling (ROS) random selects examples from the minority class or classes with replacement and copies them several times to achieve the target ratio, with entries possibly being copied multiple times. Much like RUS, there is no kind of logic involved in this method.

2.2.3 ClusterCentroids

ClusterCentroids [1] is an undersampling method that takes the original data and creates, as the name suggests, centroids based on a clustering method, more commonly kMeans [16]. To start there needs to be a definition of the number of clusters N to generate, which if class balance is the goal will equal the number of entries of the minority class. Then the kMeans method, or any other clustering method, will generate these N clusters for the majority class before selecting the centroid of each cluster of the majority class and removing the remaining entries, achieving a class balance that way. A visual representation of this can be seen in Figure 2.5.

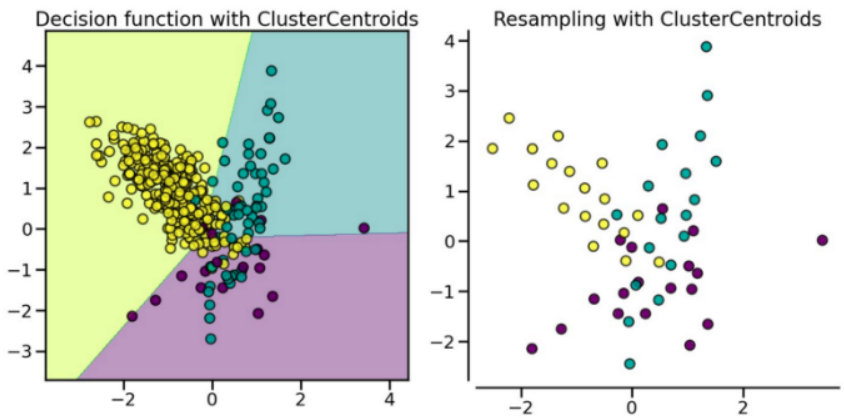

FIGURE 2.5: Example of the spacial distribution of a dataset before and after ClusterCentroids [12]. On the left we see the distribution of a target variable with 3 possible values, where the yellow dots are the majority. On the right, we see the distribution after ClusterCentroids is applied to the data and all three classes have the same number of entries.

One of the advantages of ClusterCentroids is that it can remove redundant entries in the data as multiple extremely similar entries will most likely be removed with only one being left. The major downside is the possible loss of outliers as there is little chance of outliers being the centroid of any cluster and will be removed.

2.2.4 Condensed Nearest Neighbors (CNN)

Condensed Nearest Neighbors (CNN) [9] is another undersampling method used in conjunction with the k-Nearest Neighbors (kNN) machine learning algorithm for classification. Unlike most sampling methods, this one does not offer the guarantee that an equal number of samples from each class will be achieved, as its focused purpose for use with kNN means that its goal is to produce a dataset where a 1NN classifier (kNN classifier using 1 neighbour) can produce almost similar performance with the sampled data as it would with the full training data.

CNN does this by creating a subset of entries U and by iteratively searching through all elements of the dataset, only those of the minority class by default, and selecting entries that have its nearest neighbour with a different class label. It then removes this entry from the original data and adds it to the U subset, repeating this process until no more entries can be added to U. The largest disadvantage of Condensed Nearest Neighbors is its focus on one machine learning method, meaning its results may not apply to other machine learning methods. Another disadvantage is not being able to achieve class balance ratio if it is desired.

2.2.5 Tomek Links

An undersampling method introduced by Ivan Tomek in 1976 [23], Tomek Links is a modification of the Condensed Nearest Neighbors (CNN) method that differentiates itself from CNN by introducing a set of rules that must be followed for an entry to be removed. Instead of randomly selecting features whose nearest neighbor is of a different class, Tomek Links looks at selected samples in pairs *(i, j)* and these pairs must follow three rules: The entry i is the nearest neighbor of entry j, the entry j is the nearest neighbor of entry i and finally entries i and j belong to opposing classes. If selected pairs of entries follow all three rules then they are considered to be a Tomek Link and the entry that belongs to the majority class is then removed.

This process will be repeated until no more pairs are available to be selected, meaning that much like Condensed Nearest Neighbors it does not guarantee a class equilibrium. Still, since this method is not fully focused on being used in conjunction with kNN it does also have that disadvantage.

2.2.6 Synthetic Minority Oversampling Technique (SMOTE)

Synthetic Minority Oversampling Technique or SMOTE, as it is more commonly referred to, is the go-to approach to oversampling in the current day. First proposed back in 2002 by Chawla, et al. [3], SMOTE synthesizes new entries to a dataset based on existing examples, balancing the data by randomly selecting a minority class entry and finding its nearest k neighbours that are also in the minority class. Then one of its nearest neighbours is again chosen randomly, and a new entry is created by "drawing" a line in the feature space from the original chosen entry to the selected nearest neighbours and stopping at a certain point in that line. This process, which

can be visually seen in Figure 2.6, is then repeated as many times at it is required to achieve class balance and is very effective at oversampling as new entries are not the same as previous ones, unlike in Random Oversampling, but instead are entries that are generated from plausible values based on the data.

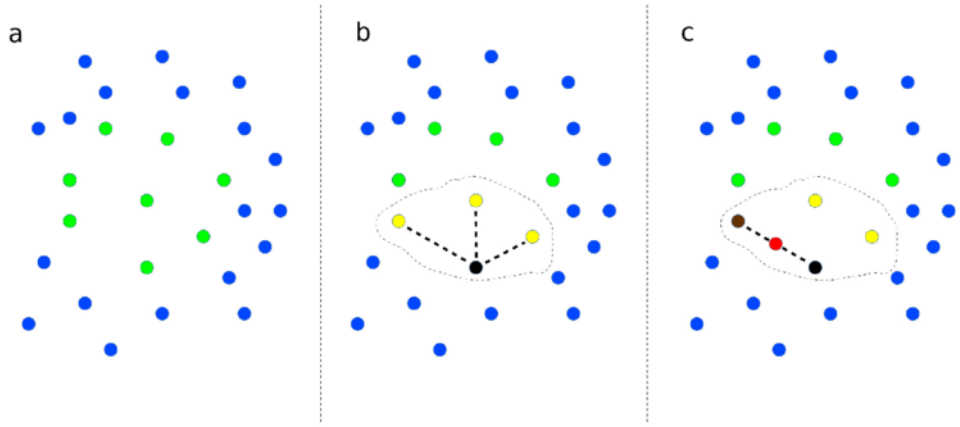

FIGURE 2.6: Graphical representation of SMOTE [21]. a) represents the set of points in the minority class (green) and majority class (blue). b) selects a point of the minority class (black) and its 3 nearest neighbours (yellow, k=3). c) selects the nearest neighbours (brown), and a new synthetic point is created (red) from the line between the black and brown point.

The major downside with SMOTE is that if a selected point only has majority class points as its nearest neighbours, it will introduce new synthetic minority class points inside a cluster of majority class points, thus making separating classes harder to discern in the feature space. With SMOTE as a basis, there are also many variations, some of which were used during this study and are described next.

2.2.7 Borderline SMOTE

This variation was proposed in [8] and aims to fix the major downside of SMOTE, previously discussed, of sometimes causing the border between classes to begin to disappear. Borderline SMOTE classifies some of the minority class points in the feature space into Border points and Noise points. Border points are points with both majority and minority class neighbours and sit on the border between the two classes, while noise points are points whose closest neighbours belong to the majority class. This method ignores all noise points it selects and only selects border points, hoping to improve the separation of classes. The downside is that it only pays attention to these points, neglecting most of the points in the minority class (cf. Figure 2.7).

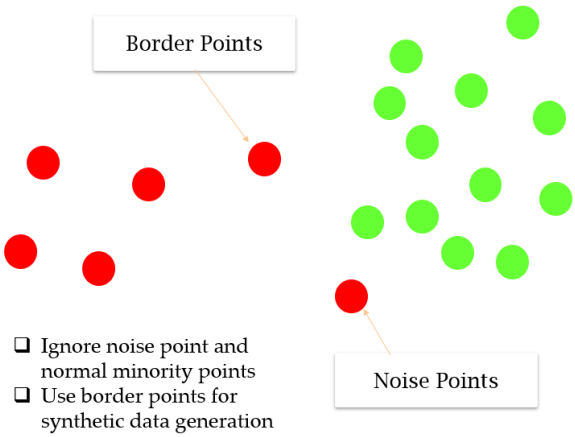

FIGURE 2.7: Graphical representation of Borderline SMOTE [11].

2.2.8 SMOTE Tomek Links

This method combines undersampling and oversampling methods, in this case, Tomek Links and SMOTE as the name suggests. It was introduced by Batista et al. [2] and combines the synthetic data creation of a SMOTE with the ability to select entries whose closest neighbour is of the opposing class of Tomek Links, thus aiming to solve the separation of classes problem that SMOTE has. The first step of the method is to apply the standard SMOTE method, creating synthetic entries until the class balance or other class proportion, as explained in subsection 2.2.6. Once this process is complete, we then move on to the Tomek Link portion, which like the standalone method searches for pairs of entries that follow all three rules outlined in subsection 2.2.5, but in SMOTE Tomek Links the difference is that both entries in the Tomek Link pair are removed instead of only removing the majority class entries, thus maintaining the class balance achieved with SMOTE.

2.3 Tools

The work developed in this thesis was implemented using Python 3.9, one of the world's more widely used programming languages. It was the language selected for three primary reasons: the students' previous knowledge and comfort with the language, the ease of integration with the infrastructure provided by the company, which will be outlined next, and the suitability of the language for the project in terms of availability of libraries.

The coding was done using Project Jupyter notebooks, an interactive computing platform used for developing and executing Python code; Jupyter allows for code to be independently run using cells inside documents called notebooks that each execute a standalone Python kernel. This allows for a better development process as it removes the need to execute a whole Python code file each time a small change is done, making the coding process more efficient.

The most popular Cloud service provider, Amazon Web Services (AWS), was used to develop

and execute the Jupyter notebooks created. The company provided the ability to create Elastic Compute (EC2) instances designed to specifically run Jupyter notebooks by launching a Jupyter service when the instance was active. These instances provided massive computing power and could be customized to suit the current needs in terms of virtual central processing units (CPU), random access memory (RAM) and storage space.

The most important Python library used for the development phase was pandas, the go-to library for data handling. It provided an easy and powerful way to parse, manipulate and analyze the data used for this work and create new structured files that suited our needs. It was almost exclusively used with comma-separated-value files (.csv) but also for text and Excel files.

To help implement the feature selection methods, sampling methods and machine learning algorithms used, two libraries were essential: scikit-learn and Imbalanced-learn. The first one provided the user with a wide variety of methods to use in machine learning projects and was used as the basis for some of the strategies implemented in the pipeline's feature selection and machine learning portion and the evaluation portion. The second key library, Imbalanced-learn, is a library that uses scikit-learn as a base to provide tools for dealing with imbalanced data. Its implementation of various sampling methods was used in this work. Finally, to help produce some of the visualizations seen in the later chapters of this work, Matplotlib, and more specifically, its pyplot module, was used.

Chapter 3

Feature Selection in Imbalanced Binary Classification

In this chapter, we present the pipeline designed for this study. We devise an experimental study on benchmark data and discuss the obtained results.

3.1 Pipelines for Feature Selection and Sampling

We aim to study how different feature selection methods and sampling techniques can interact to tackle the challenges of high-dimensionality and imbalanced data.

For this purpose, our initial idea was to follow two different approaches to feature selection. The idea of the first approach was to introduce a sampling step to create a balanced distribution of the target variable before applying any feature selection method, with the hope of counteracting the original imbalanced data on a machine learning algorithm that could be applied further down the pipeline. For this approach, the feature selection methods would be "traditional ones", i.e. not particularly suited for imbalanced data. The second approach would ignore the sampling part of the pipeline and instead focus on applying feature selection methods for imbalanced data before training and evaluating a machine learning algorithm.

What we found was that there was a large availability of feature selection and sampling methods that we could apply to our data if we were not focused on methods more suitable for highly imbalanced, as it was hard to find methods for this type of data and even harder to implement them in a reasonable time fashion.

For this reason, we decided to focus on the study of the first approach. The first step in implementing this approach was determining which methods would be used for Sampling and Feature Selection. On the first iterations of the pipeline, we focused primarily on sampling methods, all available in the Imbalanced-learn Python package, to implement six different sampling methods more uniformly and correctly. These six methods were: Random Undersampling, Cluster

Centroids, Random Oversampling, SMOTE, BorderlineSMOTE, and SMOTE Tomek Links. As for the feature selection methods, the initial pipeline only contained two, F-Test Correlation Quotient and Boruta, as these were the first two methods we looked at and implemented in Python.

To quantify the results of our pipeline we employed a Random Forest Classifier machine learning model from the scikit-learn Python library with default hyperparameters. This model is one of the more used for classification problems and was used for its simplicity in implementation. The first metric to measure the model's performance was the Matthews correlation coefficient.

In [25], the authors propose a direct comparison between two approaches applied to a similar problem. Still, instead of only considering sampling and feature selection in this order, the authors compare the performance of doing feature selection before sampling, something we had not yet done. Based on this work, we decided to devise two different pipelines for our initial approach: one where feature selection and then sampling was applied (FS $\rightarrow$ S) and the reverse happened (S $\rightarrow$ FS). The goal was to directly compare the performance of the two pipelines to try to narrow down the best course to apply to the real-world data subject of this study.

Using our initial approach as the starting point, there was a clear necessity to increase the number of feature selection methods being evaluated to provide a complete overview of the field of study. The methods we found and applied to both approaches were: Mutual Information and Recursive Feature Elimination. The first new method was implemented by ourselves while the other Recursive Feature Elimination used scikit-learns' built-in method to simplify the process. As for the sampling methods, this did not change.

3.2 Experimental Study

3.2.1 Experimental Setup

The first step for experimentally testing our pipelines was to determine what data we would be given in terms of the number of features, the types of those features and the total number of entries. In the early stages, a generic overview of the case-study-data was outlined, and we determined that we would be working on a binary classification problem with a highly imbalanced target variable that had a mix of continuous and discrete variables supporting it. For this, we resort to one of the better resources for datasets, the University of California, Irvine Machine Learning Repository [24] that contains over 600 datasets, some uploaded in 1987. After looking inside this collection of datasets, we found one dataset stood out as it resembled our future data regarding the number of features and the imbalance of the target variable. The APS Failure at Scania Trucks Dataset uploaded in 2016 by Tony Lindgreen, and Jonas Biteus focuses on the Air Pressure System (APS) of a Scania truck. The dataset contains 171 features and 60000 entries with a binary target variable that is positive if a failure is connected to an APS component or negative if it is not connected, the dataset has 1000 positive entries and 59000 negative entries.

For additional testing, we felt it was also a good idea to find additional datasets that were binary classification problems but varied in terms of the distribution of the target variable. In the same University of California, Irvine Machine Learning Repository we found three additional datasets. King's Rook vs King's Pawn dataset, a chess-related dataset, has 3196 entries and 36 features and an imbalance ratio of 0.48 (the most balanced of the four datasets). German Credit is a small dataset with 100 entries, 24 features and an imbalance ratio of 0.30. And finally Bank Fraud, a transactional analysis dataset, with 20468 entries, 112 features and an imbalance ratio of 0.27. In Table 3.1 we can observe this information summarized.

Table 3.1: Information about the datasets used for the early development of the pipeline.

Dataset	**Entries**	**Features**	**Imbalance Ratio**
King's Rook vs King's Pawn	3196	36	0.48
German Credit	1000	24	0.30
Bank Fraud	20468	112	0.27
APS Failure at Scania	60000	170	0.02

Creating the pipeline was done in Python using Jupyter Notebooks hosted a r6g.4xlarge instance in the companies' AWS Datalake with 16 virtual CPUs and 128GiB of RAM. The data was then split into a train/test split of 70% data for training and 30% for testing using scikit-learn's train_test_split method, using stratification to ensure that the distribution of the target variable was the same in both the training and testing data. After that, we defined five different thresholds for the number of features that would be selected with these thresholds being: 1% of features being selected, 25% of features being selected, 50% of features being selected, 75% of features being selected and 100% of features being selected.

As for the pipelines FS → S and S → FS experimental testing, we decided for each threshold of selected features, take all of the available feature selection and sampling methods and apply them to the data in pairs, first doing feature selection then sampling and next the other way around. Afterwards, the resulting data set would be used to train a machine learning model, Random Forest Classifier. This model was then used on the testing data to make predictions, which were recorded to be compared to the true values.

For our initial approach, we used four different feature selection methods, the two used in the initial version of the two-step pipeline previously mentioned and two new methods, Mutual Information and Recursive Feature Extraction. As for the sampling methods the same ones were used except for Cluster Centroids, whose subpar results when compared to the rest led to their removal, meaning that our sampling methods were: Random Undersampling, Random Oversampling, SMOTE, Borderline SMOTE and SMOTE Tomek Links.

3.2.2 Results

In this section, we will discuss the results obtained from both the preliminary approach and the two proposed pipelines (FS→S and S→FS) to the four benchmark datasets.

3.2.2.1 Preliminary Results

We started by testing our initial approach. The results, depicted in Figure 3.1, show us that despite being able to match and even surpass the results of our control test, none of the sampling methods could produce a better MCC score on the most imbalanced dataset, the APS Failure at Scania dataset. More so, only undersampling could improve the time taken in feature selection and model training. Still, undersampling methods produced the worst MCC values of the tested methods meaning that the gain in computational performance came at the cost of prediction performance. Our early experiments concluded that the Boruta, being a wrapper method, was slower than FCQ. This conclusion matches up with the current literature when comparing the computational performance of wrapper methods and filter methods.

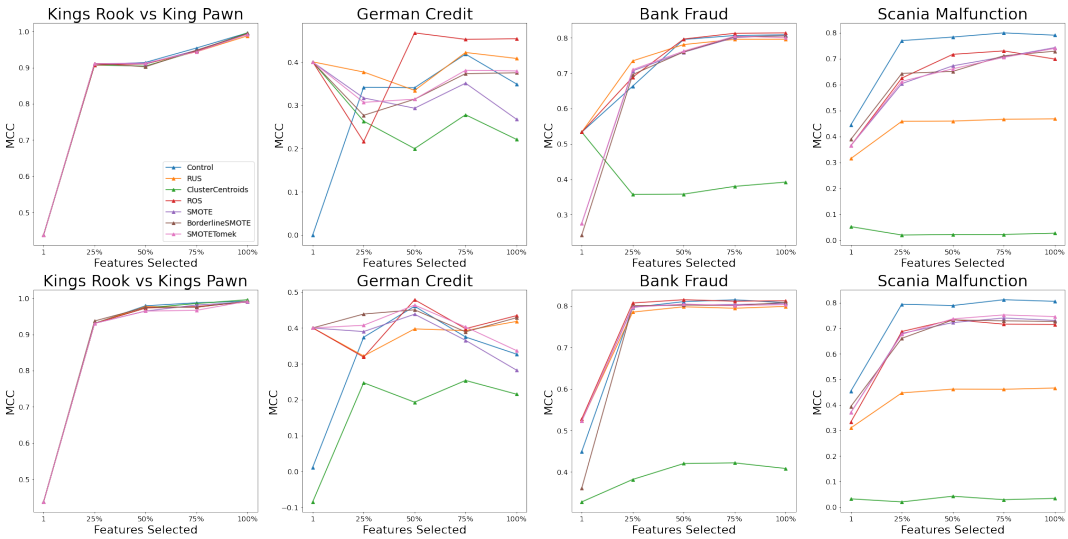

FIGURE 3.1: Results of the benchmark datasets on different combinations of feature selection and sampling methods. The results for FCQ are in the first row, and the results for Boruta are in the second.

3.2.2.2 Results obtained by FS→S and S→FS Pipelines

The results being discussed are of a single run of the pipeline and each combination of sampling method, feature selection method, order and number of features selected. Random Forest obtains the predicted values for test data.

To make the comparison more uniform the same results graphic was produced for each dataset and will be the backbone of this results analysis. The graphic aims to highlight differences between different types of sampling and feature selection methods, with the same combinations on all the graphics, with each subplot representing the average result score of the feature selection and sampling methods that are a part of that combination in the different order of operations, this approach was inspired by a similar aggregation of results performed in [25]. The only change between them is the dataset and range of the Y-axis, representing the F1-Score, to fit each dataset better. For each dataset, we aim to highlight which combination of methods and the order performs best for each feature percentage. We consider the control result, the result of applying the machine learning model to each dataset without any type of feature selection and sampling being applied, in any order.

King's Rook vs King's Pawn

Being our most balanced dataset, 0.48 imbalance ratio, with some amount of information in terms of the number of entries it is somewhat expected that a robust machine learning model can perform well for this dataset and looking at Figure A.1 we see that with our control F1-Score (marked as the red star) being near the perfect value of 1, having a value of approximately 0.997. Looking at the figure, we can also observe that all of the combinations of methods performed extremely well in terms of F1-Score values from the selection of 25% of features and up and were able to reach the F1-Score of the control for both orders of the pipeline when selecting 100% of the features.

Another conclusion we can draw is that the wrapper methods generally perform better than filter methods regardless of the sampling type used, particularly in the 25% and 50% selected feature threshold. For this dataset the order of operations, i.e. sampling then feature selection or feature selection then sampling did not matter with the differences between the two being hard to discern. As for the combination of methods and order that performed best for this dataset, we can observe that in Table 3.2, where we see that there is no clear best combination for all of the % features with Borderline SMOTE and Recursive Feature Extraction (RFE) performs best for a lower percentage of selected features while Mutual Information performs best for the 75% features selected.

Table 3.2: Best performing pairs and orders for each selected feature percentage for the King's Rook vs King's Pawn dataset.

% of Features	Sampling Method	FS Method	Order	F1-Score
1%	Borderline SMOTE	RFE	FS → S	0.498
25%	Borderline SMOTE	RFE	FS → S	0.985
50%	RUS	Boruta	S → FS	0.992
75%	Borderline SMOTE	Mutual Information	FS → S	0.994

German Credit

In the case of our smallest dataset in terms of entries and number of features the results in Figure A.2 show that as can be expected, the small amount of information that a dataset of this scale can have affected the machine learning model with both the control F1-Score and the pipeline results being significantly worse than in the previous dataset. Observing the results however we see despite reducing the information even further, under-sampling proved effective at providing the best results overall, at least for our single run. An explanation is that the random seed for our pipeline run helped remove some outliers of the former majority class and thus helped the Random Forest classifier a little bit.

In terms of feature selection methods, there is no clear difference between filter and wrapper methods and the same can be said for the order of operations different pairings present different winners and no clear conclusion can be reached. Looking at Table 3.3 we see that the results match the conclusion from the graphics that under-sampling was most effective here. Still, the 1% of features had RUS as the sampling method in the best-performing entry.

Table 3.3: Best performing pairs and orders for each selected feature percentage for the German Credit dataset.

% of Features	Sampling Method	FS Method	Order	F1-Score
1%	SMOTE	Boruta	FS → S	0.585
25%	RUS	Boruta	FS → S	0.586
50%	RUS	FCQ	S → FS	0.621
75%	RUS	Mutual Information	S → FS	0.614

Bank Fraud

Having a nice amount of information in terms of entries and number of features and an average imbalance ratio of 0.27, Figure A.3 shows that this dataset produced consistent results between the different pairings and had a nice performance in terms of F1-Score. As was the case with the first dataset in this section, we see no clear difference between the different types of sampling. Still, a difference is visible in the comparison between filter and wrapper methods in the 25% feature range, with the wrapper methods proving more effective for this range of features.

The order of operations only presented a noticeable difference when 25% of the features were selected with sampling and then feature selection had the best average performance. Looking at Table 3.4 we see that despite not being visible in the graphics, ROS was the sampling method on all of the best pairs, with the two wrapper methods here also being present in all of the pairs. We can also observe that the order in the best performing pair is always sampling then feature selection and that the best result was achieved when selecting 25% of features.

Table 3.4: Best performing pairs and orders for each selected feature percentage for the Bank Fraud dataset.

% of Features	Sampling Method	FS Method	Order	F1-Score
1%	ROS	Boruta	S → FS	0.663
25%	ROS	RFE	S → FS	0.866
50%	ROS	RFE	S → FS	0.865
75%	ROS	RFE	S → FS	0.864

APS Failure at Scania

Being our largest dataset in terms of entries and features and the most imbalanced, this dataset presents the results closest to the ones we expect when applying the pipeline to the real dataset. Looking at Figure A.4 we see that overall the pipeline results cannot match those of the control test, with under-sampling, in particular, creating subpar results. As is the case in other datasets we see, those wrapper methods have a strong relative performance when selecting 25% of features, with them, combined with oversampling being the ones closest to reaching the control F1-Score at 25%.

The results also show that all of the cases feature selection then sampling performs better than the other order, a conclusion that is also supported by Table 3.5 where all of the best-performing pairs are in the feature selection then sampling order, however, we can see also see that the best performing pair overall in terms of F1-Score was the pair FCQ and ROS at 25% of selected features, opposing what the graphic in Figure A.4 showcased, while also being the best pair for 50% and 75% features, leading to the conclusion that Mutual Information performed poorly in this dataset and thus reduced the mean of the filter method results.

Table 3.5: Best performing pairs and orders for each selected feature percentage for the APS Failure at Scania dataset.

% of Features	Sampling Method	FS Method	Order	F1-Score
1%	ROS	Boruta	FS → S	0.391
25%	ROS	FCQ	FS → S	0.775
50%	ROS	FCQ	FS → S	0.768
75%	ROS	FCQ	FS → S	0.769

3.3 Discussion

Looking at our results on the used datasets in this section, we can see mixed results, varying from dataset to dataset. In our most balanced dataset, King's Rook vs King's Pawn, we see that in the majority of cases the results of the pipeline at the 25% and 50% thresholds and the

Bank Fraud dataset surpass those of the control pipeline or come extremely close to matching, an observation that also occurs in the Bank Fraud dataset, a more imbalanced dataset but one with more information.

When there is not enough data and a small feature set, as in the German Credit dataset, we can see that the results are hard to interpret and draw conclusions from, as they are better in some cases and worse in others. Finally, for the dataset that most resembles ours, APS Failure at Scania, we see that some combinations of pairs provided a lot more effective than others, coming close to matching the control even at lower selected feature thresholds.

Overall, we see that this systematic study can help obtain insight conclusions. Still, there is no clear answer on the best combinations of methods to use and the order since, for the most part, the "best" order of operations varies from dataset to dataset, from pairing to pairing and threshold to threshold.

Chapter 4

Case Study on Scrapping of Tires

In this chapter, we focus on the scrapping tires problem. We perform a basic exploratory data analysis on the data and discuss how it was transformed from raw data to the data used in our proposed pipelines. Finally, we discuss some of the key results we found.

4.1 Methodology

The methodology followed in this work can be described as a modified Cross Industry Standard Process for Data Mining (CRISP-DM) methodology. CRISP-DM [10] is commonly used in the field of data science and consists of six key phases:

1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modeling
5. Evaluation
6. Deployment

It is important to highlight we did not concentrate so much on the modelling step. The main focus of this study is on applying the developed pipeline to the data and not on creating and optimizing a traditional machine learning model. The second note is that no deployment will be discussed in this work as it was not carried out.

4.2 Business Understanding

The first part of working with real-world data is defining our goal. Since our data is related to the scrapping of tires, the obvious goal was to produce a pipeline of feature selection and/or sampling that could improve the computational performance of a given machine learning algorithm without minimal or even without a hit in key performance metrics. Ideally, this work would cover both the feature selection pipeline and the model training portion.

The next step was to determine which of the many Continental tire production plants to use for applying the comparison pipeline developed in the previous chapter. In Figure 4.1 we can see some of the company's locations. After some discussion, we settled on two factories, chosen for opposite reasons. The first choice was the plant located in Lousado, Portugal as this plant produced a large number of tires each year and had a scrap rate, i.e. the percentage of tires that are sent back to be scrapped, of about 2%, meaning that it was among the best-performing factories of the whole Continental group. On the opposite end of the spectrum, there is the Clinton plant located in the United States of America. This plant produces much fewer tires yearly than the Lousado plant and has a much higher scrap rate of around 10%.

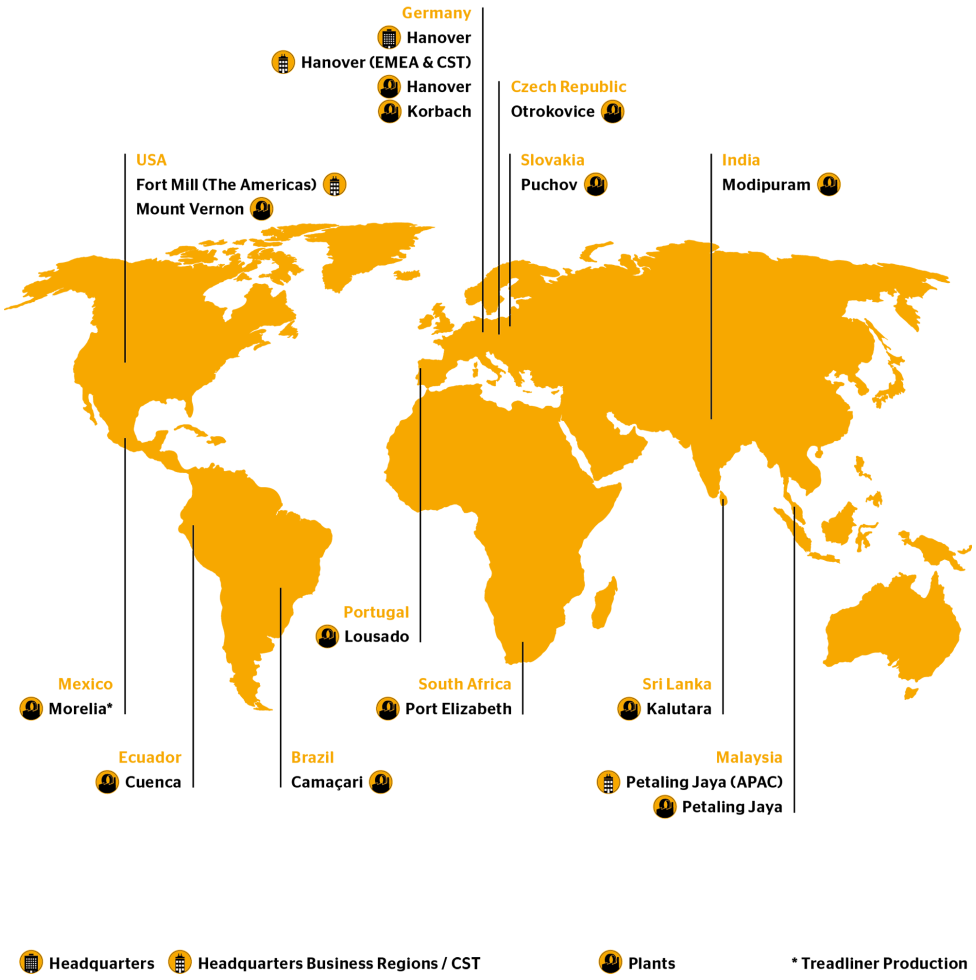

FIGURE 4.1: Map of the different Continental factories and offices around the world. [4]

This opposing factor regarding scrap rate meant that reducing the scrap rate in both plants would be for different reasons. For the Lousado plant, due to its sheer volume of production, reducing the scrap rate would result in a massive amount of tires being saved each year, further increasing the quality of production in the plant and saving money for it the company. On the other hand, the Clinton factory having its scrap rate reduced even for a slight amount would allow for the plant to have its scrap rate close to the company average among all factories, which is around 6%.

We settled on applying the pipeline first on the Clinton plant, mostly due to its lower volume of data, hoping to draw conclusions on the best methods to use both for feature selection and sampling.

4.3 Data Understanding

The data we dealt with in this work was obtained from a combination of tables inside SQL databases that the company has for each of its plants. This data is extremely large in size both in terms of entries and in terms of possible features to use. Due to the large nature of the feature space and the complex distribution of SQL tables the relationship between the data was not studied in the scope of this work.

The SQL databases were housed in the companies' AWS Datalake and queried using a previously developed Python script, being stored as *csv* file using pandas to make it easier to use and discarding the need for a query every time we wanted to use the data. For development and testing in the Clinton plant, we queried every entry whose curing date was between the 1st of January and the 31st of December 2021, producing a complete dataset for the calendar year of 2021. The resulting dataset contained 295 features and 247199 entries and become the first iteration of the data that would be used in the pipeline.

With our raw dataset being established there was early filtering of features done to remove those not relevant to the problem applying the knowledge from members of the Continental team. This process was done by hand and reduced the number of features from the initial 295 to 82 features. After this filter of features, we ran a basic missing value analysis on the remaining features, discovering that all but five columns had missing values. We grouped the columns into their percentage of missing values, grouping them into five different groups: 0% to 24%, 25% to 49%, 50% to 74% and 75% to 100%, with the results of this being visible in Table 4.1, here we can observe that around 40% of the original columns had more than 75% of its values as missing values, meaning that these columns if kept for the final dataset would have little to no meaningful information due to their lack of data.

After repeating the same missing value analysis for the entries of the dataset, we saw that all of them had missing values which meant that at least some of the features were completely empty. We confirmed this and found that out of the 82 features we were currently working with, 28 of them were full of missing values and should be removed. With removing these features, we

Table 4.1: Number of columns with a given percentage of missing values in the initial iteration of the Clinton dataset.

Percentage of Missing Values	**0%-24%**	**25%-49%**	**50%-74%**	**75%-100%**
Number of Columns	46	3	0	33

had 54 remaining.

After this iteration of the dataset, we decided to remove the columns that had more than 75% features missing, resulting in the removal of another five features. Finally to achieve the dataset used for the pipeline another manual removal of the columns was done by the team at Continental. To finalize the dataset we would be working on, we also decided to remove entries from the dataset that had more than 75% of its columns empty and at the end of it we were left with a dataset containing 32 features plus the target variable (*overallgrade*) and 245562 entries. The list and short description of the variables can be found in Appendix B.1.

Having defined our dataset we can perform some exploratory data analysis on it. To start, we can look at the distribution of the target variable to determine the imbalance ratio of the data using the metric defined in Eq. 2.3, and we see that for our dataset we have 222161 entries with the label 0 (meaning no scrapping) and 23401 entries with the label 1 (meaning that they were scrapped), giving us an imbalance ratio of around 0.10.

We can observe the correlation between the variables by looking at Figure 4.2 showcasing the Pearson Correlation between the variables displayed, where a value closer to 1 indicates a strong positive linear correlation, a value closer to -1 indicates a strong negative linear correlation and values close to 0 indicates no linear correlation between variables. The first conclusion we can draw is there is no clear variable that is highly correlated with the target variable (overallgrade), with the highest positive correlation being with the variable ct_workcenter with a value of around 0.014 and the highest negative correlation being with the variable bead_shift_date having a value of around 0.011.

Overall, we can see that there exists a strong positive correlation between a lot of variables that are related to the shift data, work center or lot of the various processes that occur during the production of a tire. Only the variable bead_workcenter appears to present a negative correlation with most variables with the remaining variables where negative correlation seems to occur being close to zero. Calculating the average correlation value for each variable (excluding itself), we see that our target variable has an average coefficient value of -0.001 and that 11 of our variables have an average correlation higher than 0.40, with the highest being first_ply_lot with a value of 0.445. On the other hand, no variable has a negative correlation lower than -0.40 as the highest is bead_workcenter having a value of -0.350, almost -0.30 lower than the second variable with the highest negative correlation.

The dataset has 1153 duplicate rows present, possibly due to the removal of unique identifier variables. Our 33 variables are divided into 23 numerical and 10 categorical variables. In

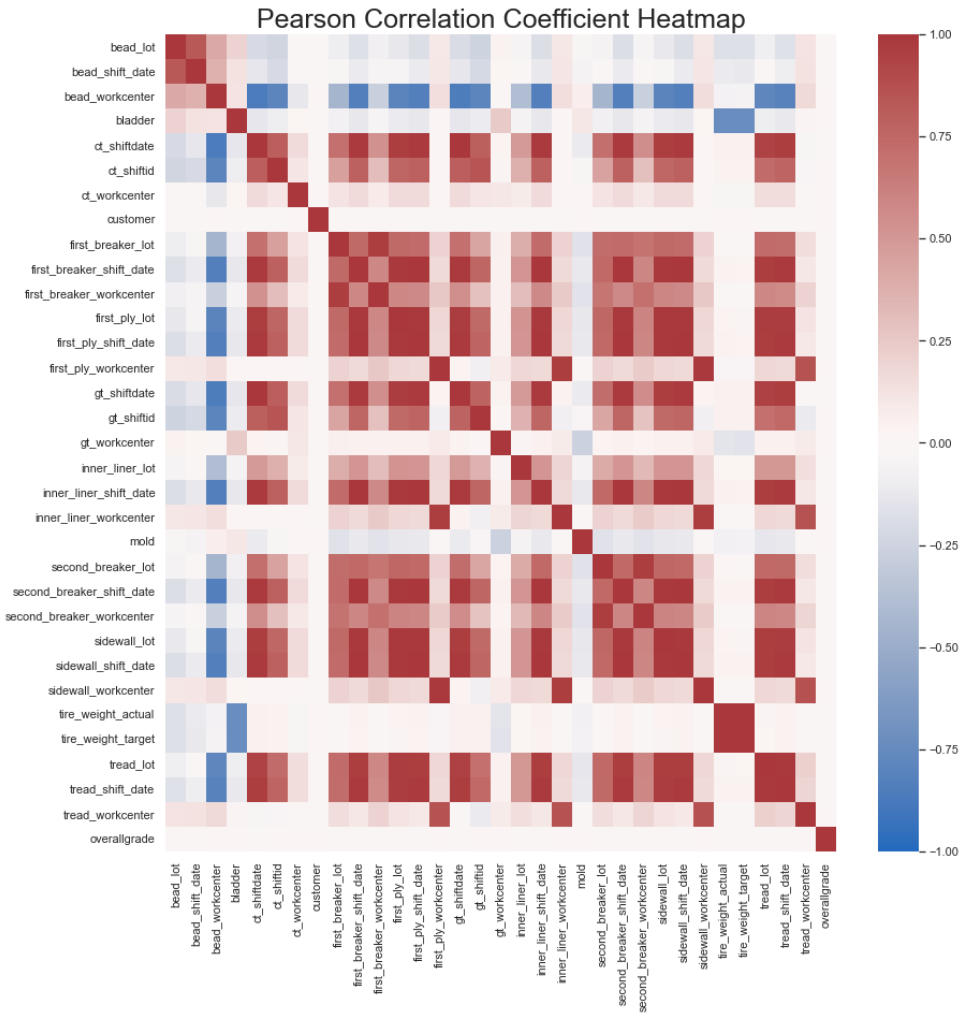

FIGURE 4.2: Pearson Correlation matrix for the variables of the Clinton data.

Appendix B.2 we can see the bar plots and histograms for all of the variables of the data. In the bar plots, we see that some of the variables do not have a lot of unique values within themselves as it is the case with most workcenter related variables. As for the histograms, using the default setting for the bin construction, which is 10 bins, we can see that some of the variables appear to present a more or less even distribution of variables, such is the case with some of the variables representing the lots used in the various processes, while others have a skewed distribution. Taking a closer look at the two variables that in theory have an impact if a tire is scrapped or not (tire_weight_target and tire_weight_actual) in Figure 4.3 we can see that their distributions are somewhat close, something that is confirmed by their correlation which is around 0.995.

4.4 Data Preparation

Preparing our data for ingestion in the pipeline in the pipeline meant that the first step was to make sure it was ready to be handled by all methods, both feature selection and sampling as well as our Random Forest Classifier model. Since a portion of these methods only works for features

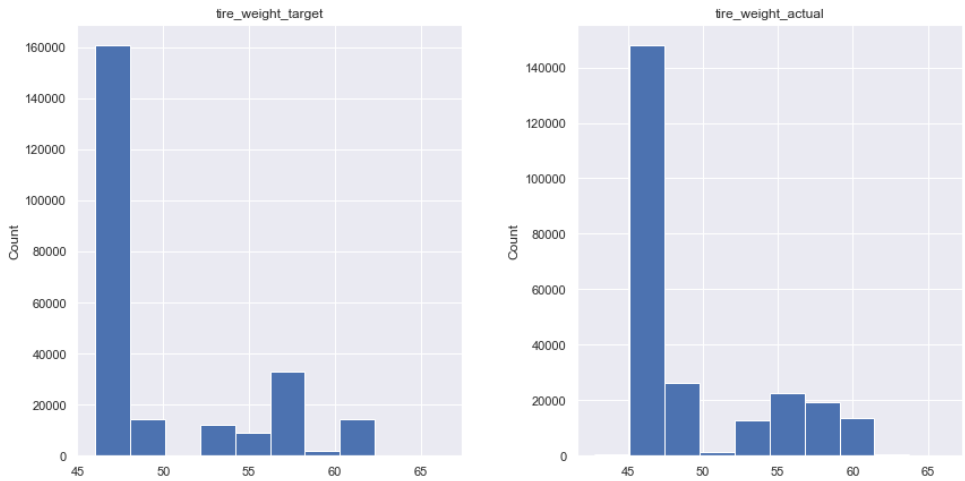

FIGURE 4.3: Histogram for the variables related to the tire weight.

containing only numeric values, it meant that we had to try and convert our non-numeric values to numerical ones. Our solution was to use Label Encoding, using the scikit-learn implementation, to convert our non-numerical categorical features into numerical categorical features, as Label Encoding assigns each category of a feature a sequential number. This solved the problem with ingesting as all we had to do was ensure all of the features were numeric.

As for the missing values, there were some options for us to use in this case. With the more common one we could have used mean or median value imputation. However, it does not work very well for encoded features and can lead to confusing results in this case. Another option considered and discarded was to use k-Nearest Neighbors or a similar neighbouring algorithm to approximate these missing values using the closest entries to the one with the missing values. This was discarded as the number of missing values was quite considerable, and this would decrease the "real life" aspect of the data in the sense that it could cause a lot of the entries to be more synthetic than real. So we settled on the most simple approach that still uses all of the available entries and performed zero imputation, where each missing value was replaced by zero.

4.5 Modeling

With our dataset finalized and ready to be processed we settled on applying a similar approach to the one done for the benchmark data as discussed in Section 3.2 by creating a stratified train-test split with a 70/30 distribution for training and testing using the scikit-learn package. We kept the same percentage of the number of features to be selected of 1%, 25%, 50%, 75% and 100%, which in this case represented 1, 8, 16, 24 and 32 features respectively. We used the same sampling methods but introduced two new feature selection methods: S2N and SFS. The execution of the developed pipeline is as straightforward as it had already been built and used on similar data in the last chapter and the same r6g.4xlarge AWS instance was used, hoping its computational power would achieve quick results. However, this was not the case since, as

expected, as the number of features being selected increased the total time for each pair of feature selection and sampling methods to run increased, something to discuss further in this chapter. This meant that the pipeline took over two weeks to produce the complete results and that mistakes found after the execution had concluded had major time costs as it could mean running the complete pipeline again.

The pipeline would, for every pair of feature selection and sampling methods, store two *csv* files, where one contained the selected columns and another that contained the predicted values for the testing data as the output of our Random Forest Classifier model. In addition, a log containing the time taken for three key stages of the pipeline (feature selection, sampling and training and testing the model) was also recorded in a structure that allowed for easy ingestion for analysis. In the end, the pipeline produced 300 *csv* files of selected columns and 300 *csv* for the predictions.

4.6 Evaluation

In this section, we discuss the results achieved when applying the developed comparison pipeline to the clean Clinton plant data for the year 2021. To facilitate the understanding and discussion of the results, this section will be divided into three subsections: Performance, Time and Selected Features. In each of the three subsections, we will discuss in more detail the results in those particular categories.

4.6.1 Performance

To begin our analysis of the performance of the pipeline in terms of metrics, we can look at performing a similar analysis to the one done in Section 3.2.2 by observing the F1-Score of the possible combinations of types of feature selection and sampling methods and plotting them to try and draw any conclusions. The control model, where no feature selection and sampling was applied to the original data, did not produce a very good F1-Score, approximately 0.153. In observing Figure 4.4 we can see that all pairings of methods managed to outperform the control F1-Score at the 50% and 75% selected feature mark, with all but three pairings: All sampling methods + only wrapper methods (FS $\rightarrow$ S), combined sampling methods + wrapper methods (FS $\rightarrow$ S) and under-sampling and filter methods (S $\rightarrow$ FS), outperforming the control F1-Score at the 25% feature selected mark.

Overall, Figure 4.4 did not showcase any clear winner in terms of the pairing of methods. However, Table 4.2 shows that SMOTE Tomek Links was a common factor in the best pair per percentage of selected features, discarding the 1% selected features. This sampling method and Mutual Information combined proved effective for the 25% and 50% marks only being beaten by Sequential Feature Selection at the 75% mark, with the sampling then feature selection proving to be the best strategy for the three percentages of selected features.

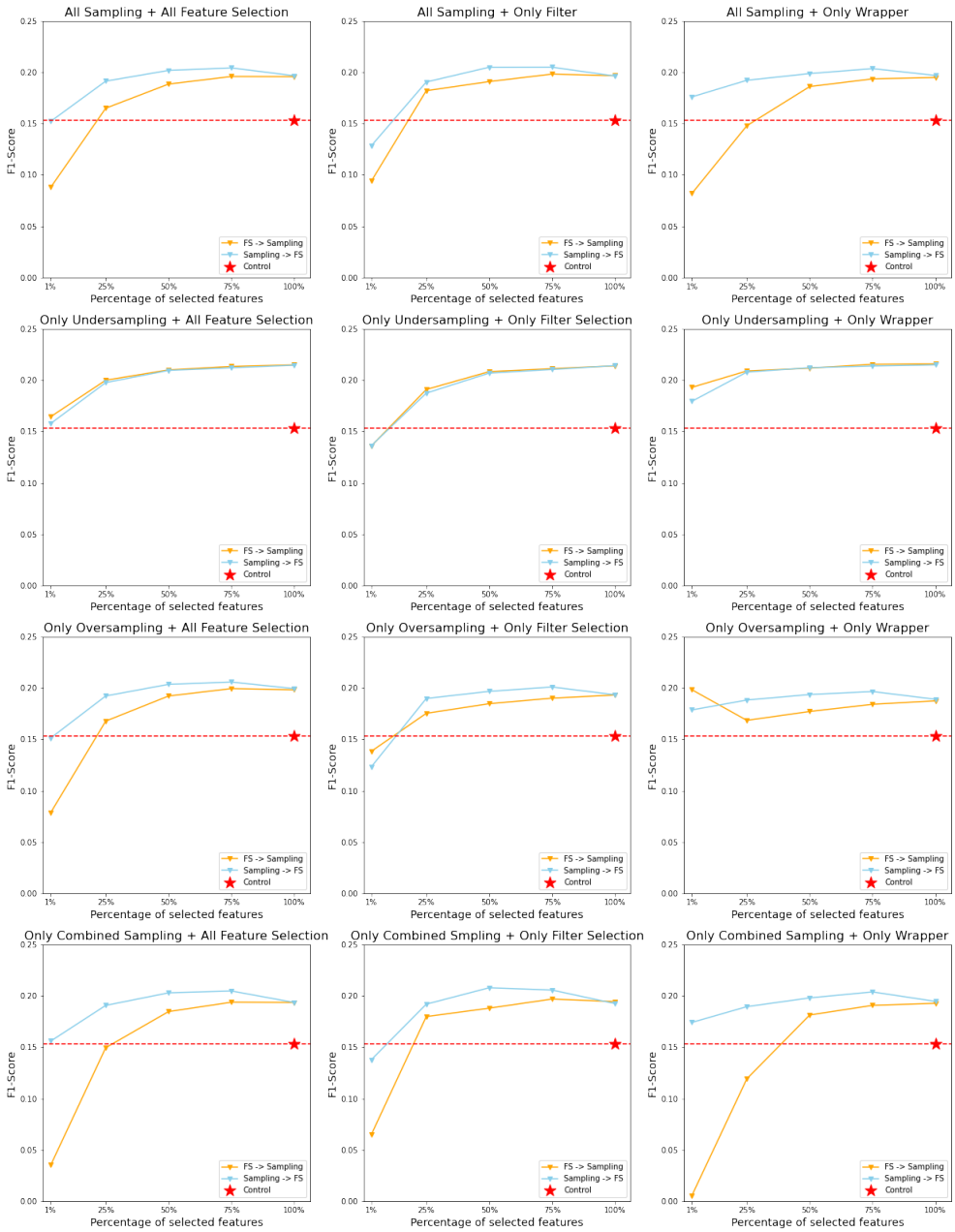

FIGURE 4.4: Results of the pipeline for the Clinton 2021 calendar year data using F1-Score as a metric.

Table 4.2: Best performing pairs and orders for each selected feature percentage for the Clinton 2021 dataset. The control value appears in the last row of the table.

% of Features	Sampling Method	FS Method	Order	F1-Score
1%	ROS	RFE	FS → S	0.217
25%	SMOTE Tomek Links	Mutual Information	S → FS	0.223
50%	SMOTE Tomek Links	Mutual Information	S → FS	0.222
75%	SMOTE Tomek Links	Sequential FS	S → FS	0.226
100%	N/A	N/A	N/A	0.153

Looking at the complete results in Appendix C, we can see that 274 out of the 300 entries (60 entries for each percentage of selected features) or approximately 91.33% of entries outperformed the control F1-Score. We can breakdown these figures further by calculating the percentage of features that surpasses the control score at each percentage of selected features, and we see that for 1% of selected features that value is around 63.33%, for 25% the value was around 93.33%. It was 100% for the remaining percentages of selected features, which was also concluded before looking at Figure 4.4.

To make sure that the results are actually an overall improvement over the control results, we can use a different metric that is more suited for imbalanced datasets, it being Matthews Correlation Coefficient (MCC). This metric is similar to F1-Score since it uses the confusion matrix as a basis for its produced value. However, it does not ignore the True Negatives (TN) entries of the confusion matrix like F1-Score does, and also has a larger interval of values ranging from -1 to +1, where the value of 0 is considered to be a random predictor and -1 an inverse predictor.

Applying Matthews Correlation Coefficient to our results we obtain a control MCC value of around 0.126, meaning that our control Random Forest model is a little better than a random predictor (MCC=0) according to the metric. In Figure 4.5 we can see that unlike the results of the F1-Score metric, using MCC indicates that the pipeline produced results slightly worst than the control model with the control MCC value only being beaten on 19 out of the 300 entries, with 11 of those entries being when 75% of the features were selected. None of them was when 1%, 25% and 50% of the features were selected.

Overall the best-performing entry in terms of Matthews Correlation Coefficient value occurred when selecting 75% of the features using SMOTE Tomek Links first followed by Sequential Feature Selection having a value of around 0.131. At the same time, the worst entry excluding those who selected 1% of features, was Boruta followed by Borderline SMOTE when selecting 25% of features, obtaining an MCC value of 0.003, being pretty much equal to randomly predicting outcomes according to the metric.

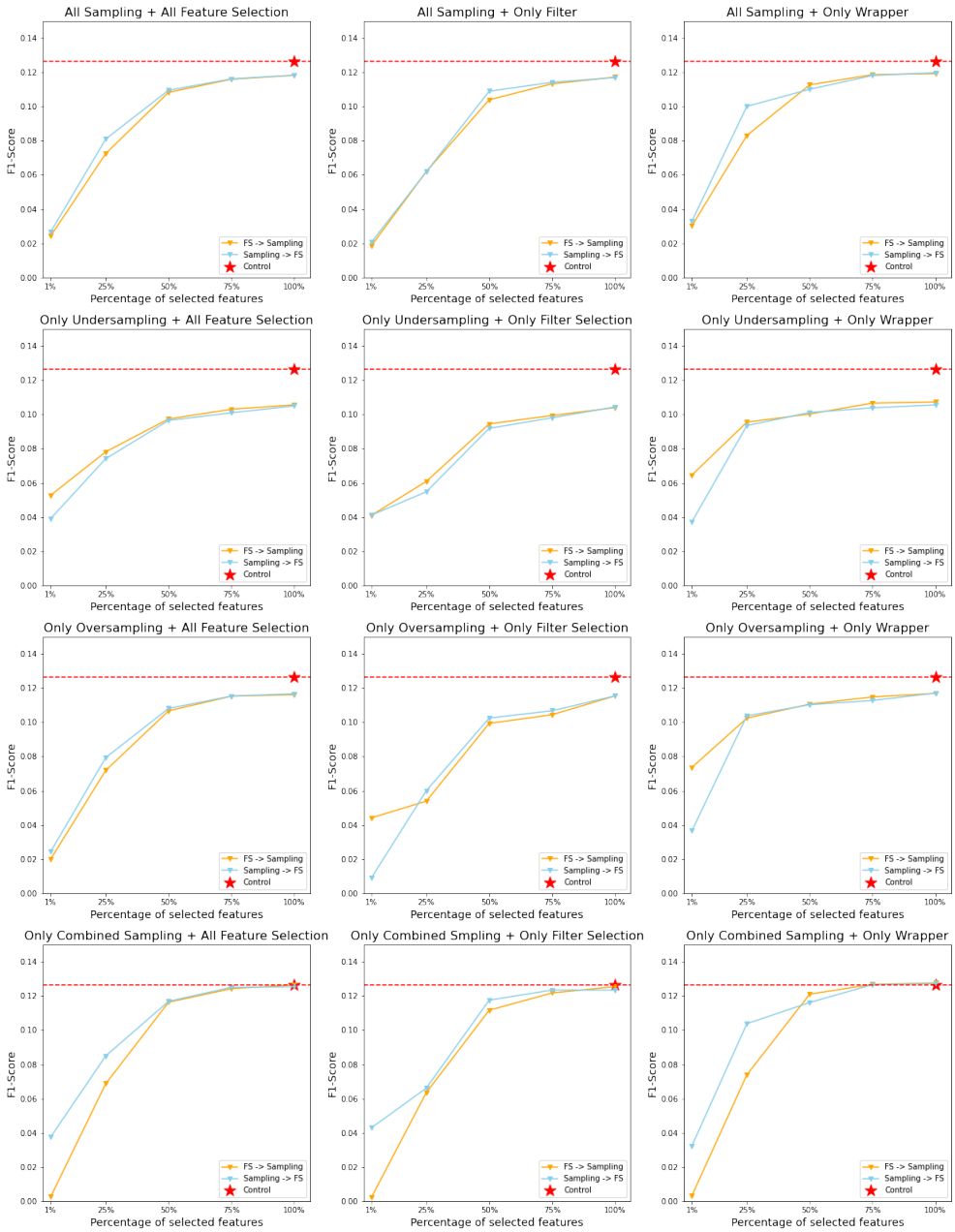

FIGURE 4.5: Results of the pipeline for the Clinton 2021 calendar year data using MCC as a metric.

4.6.2 Execution Time

We now highlight some results obtained regarding the execution time taken to perform each pipeline portion. Due to constraints in terms of the overall time taken for the pipeline to run, the results showcased here are of a single run of the pipeline for six different feature selection methods and five sampling methods.

For clarity reasons the large portion of the graphics showcased in this subsection will be in pairs or two-by-two grids, where the left graphics refer to results obtained when doing feature selection first and sampling second, while graphics on the right refer to the contrary order. This however will always be highlighted both in the title of each graphic and its caption, not excluding the appearance of graphics that follow other guidelines. For each pair of feature selection methods and sampling methods in the pipeline the time taken to select the features, sample the dataset, train and infer a Random Forest Model, as well as total time (a sum of the previous values) was recorded for each order of operations.

Our first aim is to determine if a difference exists in the total execution time taken for the pipeline to execute between the two orders of operations. Visually we can do this by averaging the total time taken for all entries of the results of each order, breaking down the total time into each specific task to help draw further conclusions. In Figure 4.6, we see that sampling caused the feature selection time to increase by a lot skewing the difference between the two orders greatly when selecting 25% of more features. This effect is expected to be much more noticeable on wrapper methods as the increased dataset size will cause them to take longer to complete.

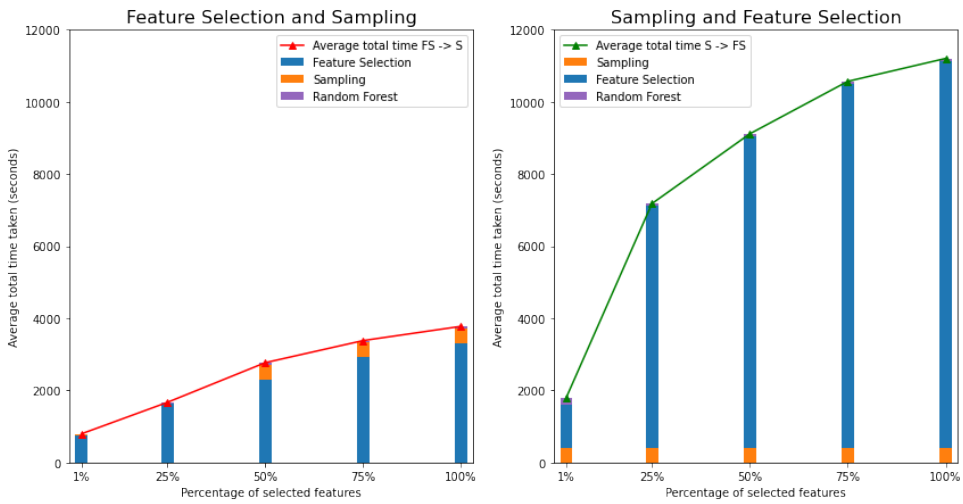

FIGURE 4.6: Breakdown of the average time taken for each order of the pipeline to complete, aggregated by the percentage of selected features.

Taking a detailed look at the sampling results presented in Figure 4.7 where only sampling time is taken into account and split by the sampling method used, we can see that, as expected, in the right-hand graph showing sampling first entries the time taken for each method is linear, and that SMOTE Tomek Links had a much worse performance when looking at time taken than

the rest of entries. As for the other methods, we can also see that Borderline SMOTE was the worst, with the remaining being nearly indistinguishable on that figure.

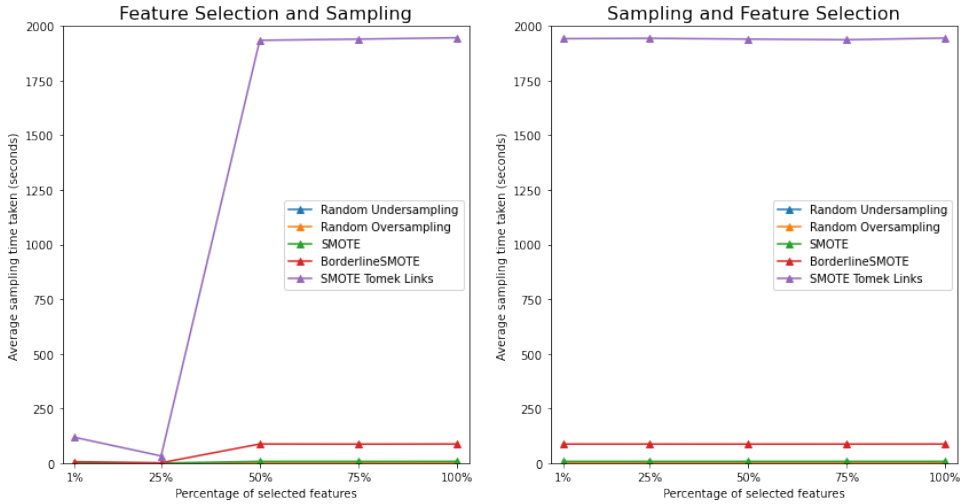

FIGURE 4.7: Average time taken for each individual sampling method to complete the sampling process, aggregated by the percentage of selected features.

Using Table 4.3 we see that both random sampling methods (RUS and ROS) are extremely close in terms of time taken while SMOTE falls slightly behind both. These results indicate to us that if time is any concern when applying any of these methods, then the execution time showcased here by SMOTE Tomek Links is not fit for any application of that type. As for the rest of the methods if there is the need for a more robust method that is not pure randomness for oversampling, then SMOTE or Borderline SMOTE appears to be the most correct method among the tested ones.

TABLE 4.3: Average sampling time for each sampling method, aggregated by percentage of selected features and order.

	Percentage of Selected Features	25%		50%		75%		100%	
	Order	FS → S	S → FS	FS → S	S → FS	FS → S	S → FS	FS → S	S → FS
Samp. Method	RUS	0.04	0.06	0.05	0.07	0.05	0.07	0.07	0.07
	ROS	0.06	0.18	0.11	0.18	0.14	0.18	0.19	0.19
	SMOTE	0.20	7.84	7.73	7.84	7.71	7.82	7.83	7.85
	Borderline SMOTE	1.38	87.1	87.35	86.93	86.82	87.12	87.44	87.24
	SMOTE Tomek Links	32.71	1943.27	1934.07	1939.61	1939.51	1936.68	1945.86	1944.31

Applying a similar approach to the feature selection methods, done in Figure 4.8, it becomes clear that the performance of Sequential Feature Selection was abysmal in terms of time taken, in particular when performing sampling first, with its average feature selection time surpassing the 15-hour mark when selecting 75% or more features.

These results for SFS are so poor that removing the entries where this method was used for the data and plotting the average time taken similar to Figure 4.6 without the breakdown, we see that Figure 4.9 show us that both orders are now much closer and in much more reasonable

time ranges.

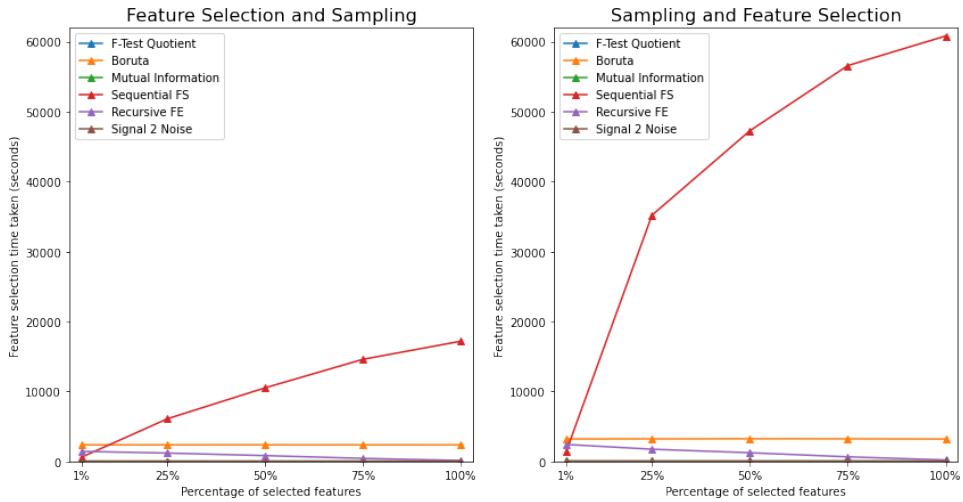

FIGURE 4.8: Average time taken for each individual feature selection method to complete the feature selection process, aggregated by the percentage of selected features.

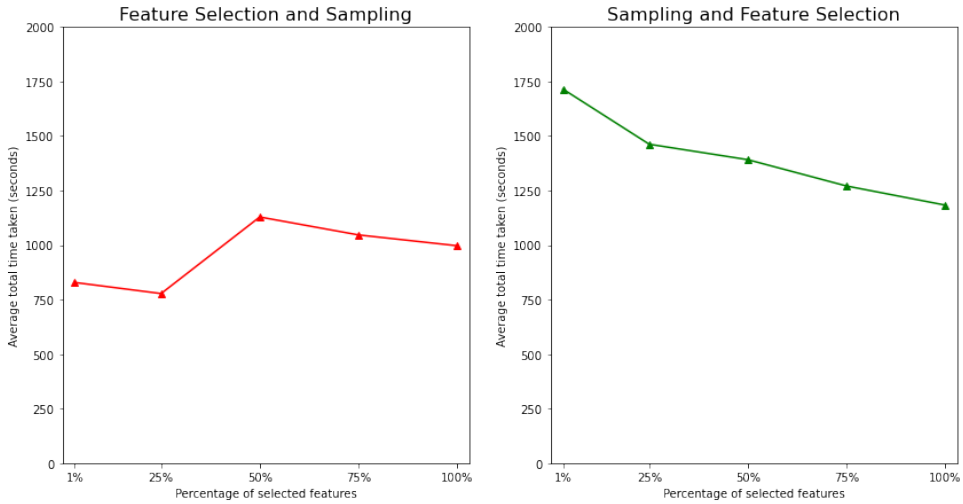

FIGURE 4.9: Average time taken for each order of the pipeline to complete, excluding entries where SFS was used, aggregated by the percentage of selected features.

Looking at Table 4.4, where the results are shown in a table format, we see that, as expected, all wrapper methods required more time than the filter methods, with Recursive Feature Elimination being faster as the percentage of selected features increases, due to starting with the full set of features as opposed to other methods. We can also see that for other methods the percentage of features selected does not affect them as they always produce a complete feature set and select the top-K features. F-Test Quotient's result is extremely fast and paired with a random sampling method can be a productive and efficient two-step pipeline in any given order.

Finally, we can look at our pipeline's effect when training and testing a machine learning model. As expected, the only entries that beat the recorded control time for a Random Forest model were those that used RUS as its sampling method. These entries decreased the dataset

TABLE 4.4: Average feature selection time for each feature selection method, aggregated by the percentage of selected features and order.

	Percentage of Selected Features	25%		50%		75%		100%	
	Order	FS → S	S → FS	FS → S	S → FS	FS → S	S → FS	FS → S	S → FS
FS Method	F-Test Quotient	0.62	0.91	1.11	1.63	1.42	2.07	1.55	2.26
	Mutual Information	35.45	53.57	35.55	53.59	35.49	53.45	35.61	53.56
	Signal 2 Noise	37.29	51.01	37.44	51.32	37.7	51.9	37.63	51.64
	Boruta	2379.92	3226.47	2383.03	3241.46	2374.47	3234.70	2379.95	3207.48
	Sequential Feature Selection	6091.97	35192.04	10528.10	47264.20	14601.00	56579.61	17190.07	60815.73
	Recursive Feature Selection	1197.42	1749.22	835.76	1243.61	436.56	659.79	138.47	211.82

size, making it faster for the model to train with. Observing Figure 4.10 where we separate entries that use RUS from the rest, we can conclude that some other entries that didn't use RUS managed to be faster than the control time at 25%, 50% and 75% of selected features. In total, 28 of these entries outperformed the control time at these feature selection steps.

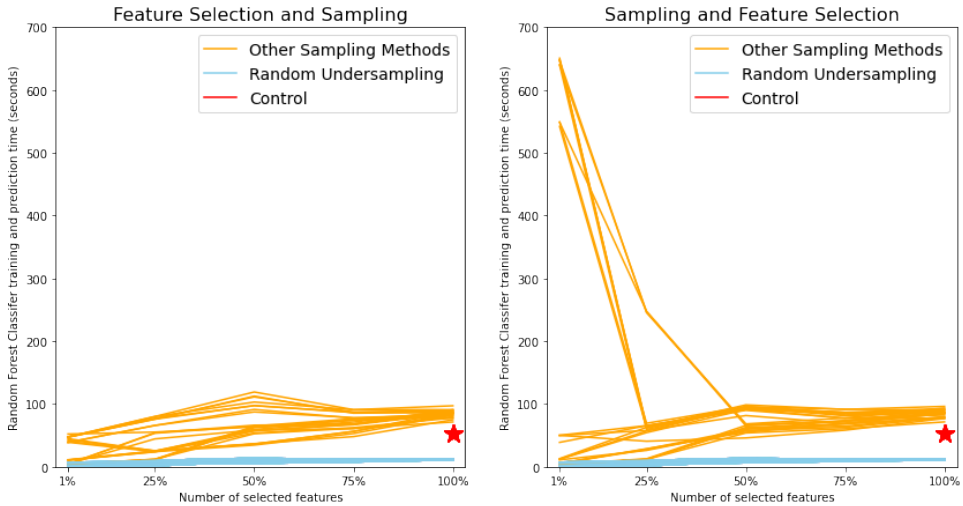

FIGURE 4.10: Comparison between average Random Forest Classifier training and testing time of entries using RUS as the sampling method and the rest, aggregated by the percentage of features selected.

4.7 Discussion

Our results showed us that different metrics could draw different conclusions regarding performance. The results using the F1-Score metric appeared to as a positive result on the pipeline, as we saw a large majority of entries at multiple thresholds being able to outperform a poor control F1-Score, sometimes providing close to a 33% boost in performance when compared to the control, with these improvements even going as far as occurring in the 1% threshold in some cases. The F1-Score results also highlighted some apparent differences in the order of operations in some combinations, most visible in Combined Sampling + Wrapper Methods, which was not as visible in the results of the previous chapter.

The positive results, however, were not as visible when using the MCC metric, with only 19 entries beating the control value. The results also showcased a much steeper curve in terms of MCC value increase over the feature selection thresholds when compared to the F1-Score results, as in almost all of the combinations, we saw a large increase, at least for our interval range, in MCC value when looking at the 1%, 25% and 50% thresholds. In contrast, in the F1-Score results, there was a little slope after reaching the 25% feature threshold in most cases.

In terms of time, as discussed previously, performing sampling first caused a visible increase in the time taken for the feature selection methods to execute, except FCQ. The gap in run time between filter and wrapper methods in terms of time was noticeable, with iterative methods being the worst of them all. The results in modelling and testing time were as expected, with the Random Undersampling's smaller dataset helping increase the model training and testing speed.

Overall we can conclude that if the single pairing of the feature selection method and sampling method is implemented, any of the filter methods could be used in combination with SMOTE Tomek Links, as it provided the best overall performance in both metrics. If a streaming solution was required, then SMOTE Tomek Links could be replaced with Random Undersampling combined with F-Test Quotient, providing the fastest oversampling solution.

Chapter 5

Conclusion

This section concludes our study by highlighting the main contributions and outlining some possible future research directions.

5.1 Main Contributions

Our study led us to conclude that the applied pipeline did not provide a massive increase in terms of performance when observing the F1-Score and did not provide almost any improvement if the considered metric was Matthews Correlation Coefficient, when applied to the Clinton plant 2021 data. However, many features were removed from this data due to the initial manual filtering of features. As such, the feature selection portion became less critical since the dimension of the data in terms of features became smaller.

Another interesting point was the fact that for most cases of both the benchmark data presented in Chapter 3 and the real data presented in Chapter 4, we could see many of the combinations of feature selection and sampling methods achieving similar results in terms of F1-Score, or even surpassing, the control results when the percentage of selected features was 25% or 50% leading us to believe that the pipeline could have the possibility to work well in feature-rich datasets that are not as imbalanced as the real-world data or the APS Failure at Scania dataset.

The work done here was able to showcase, in particular, the massive difference that exists in terms of time between the wrapper methods used here and the filter methods, as it was clear observing the results in terms of time taken for the Clinton 2021 data that filter methods were the ones to be used if any streaming solution was considered, in particular with F-Test Quotient being able to select the features quickly. The easier to discard methods for any future work would have to be Sequential Feature Selection, in terms of feature selection methods, and SMOTE Tomek Links, in terms of sampling, with the first being extremely slow at selecting the features and the latter, despite being usually among the best-performing methods, was visibly slower than the remaining sampling methods.

5.2 Future Work

In terms of future work that could be done using this work as a basis, the most obvious would be to apply the pipeline, or a more narrowed-down selection of methods, to the data provided by the Lousado plant as the data itself, should be more evident in terms of missing values and provides an even more imbalanced challenge for any model.

Another interesting challenge could create a solution hosted in the cloud that could predict the tires being produced in real-time. This could have the features being selected previously and filtered out as the data is ingested or as an evolving set of selected features being determined as the data comes in.

The other initial approach of feature selection specially devoted to imbalanced classification scenarios was tried but not followed through. The reason was that not enough methods were applied to our case study and could be implemented reasonably. Future work could conduct more thorough research on these methods and implement some.

And finally, a more intensive exploratory data analysis could have provided a better understanding of the data overall and have allowed for more ways of tackling this problem and different ways to pre-process the real-world data.

Appendix A

Benchmark Results

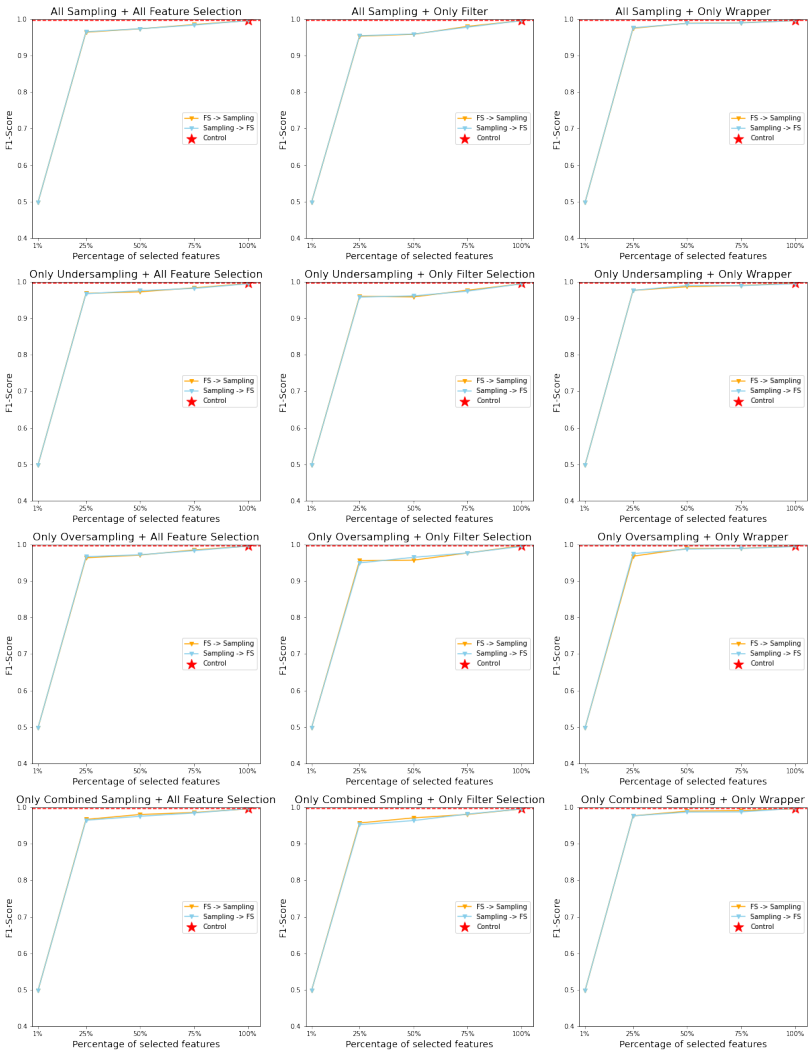

FIGURE A.1: Results of the pipeline for the King's Rook vs King's Pawn dataset.

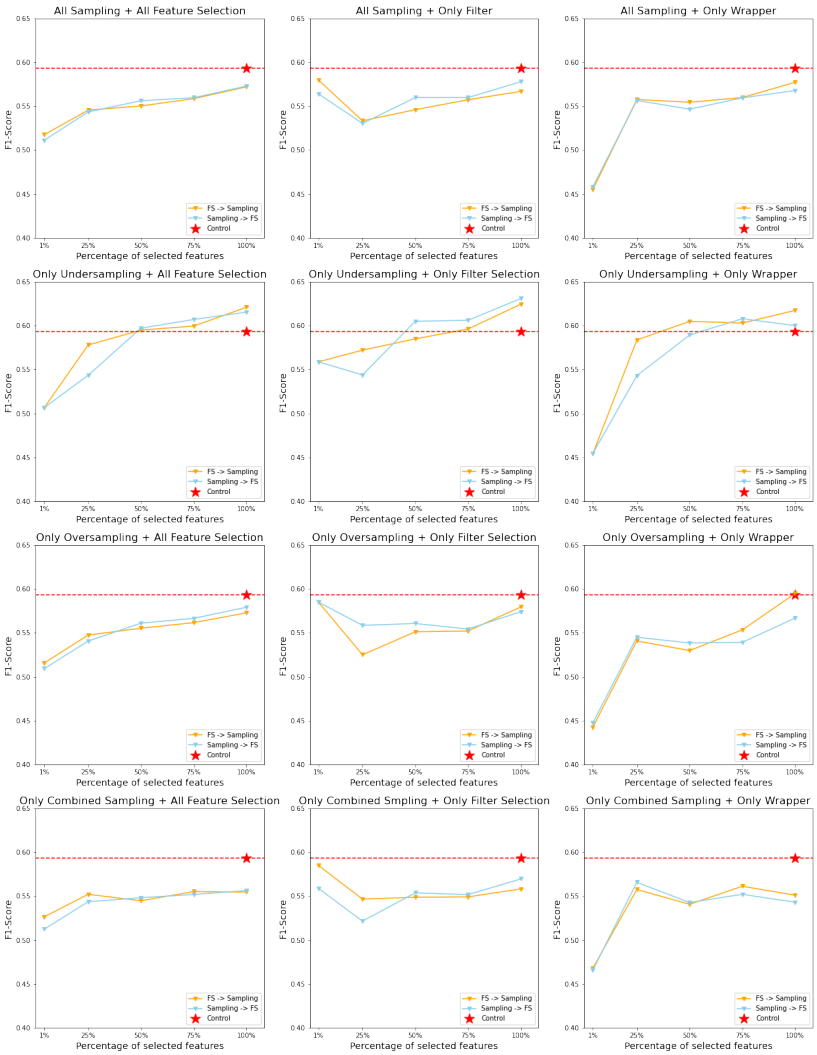

FIGURE A.2: Results of the pipeline for the German Credit dataset.

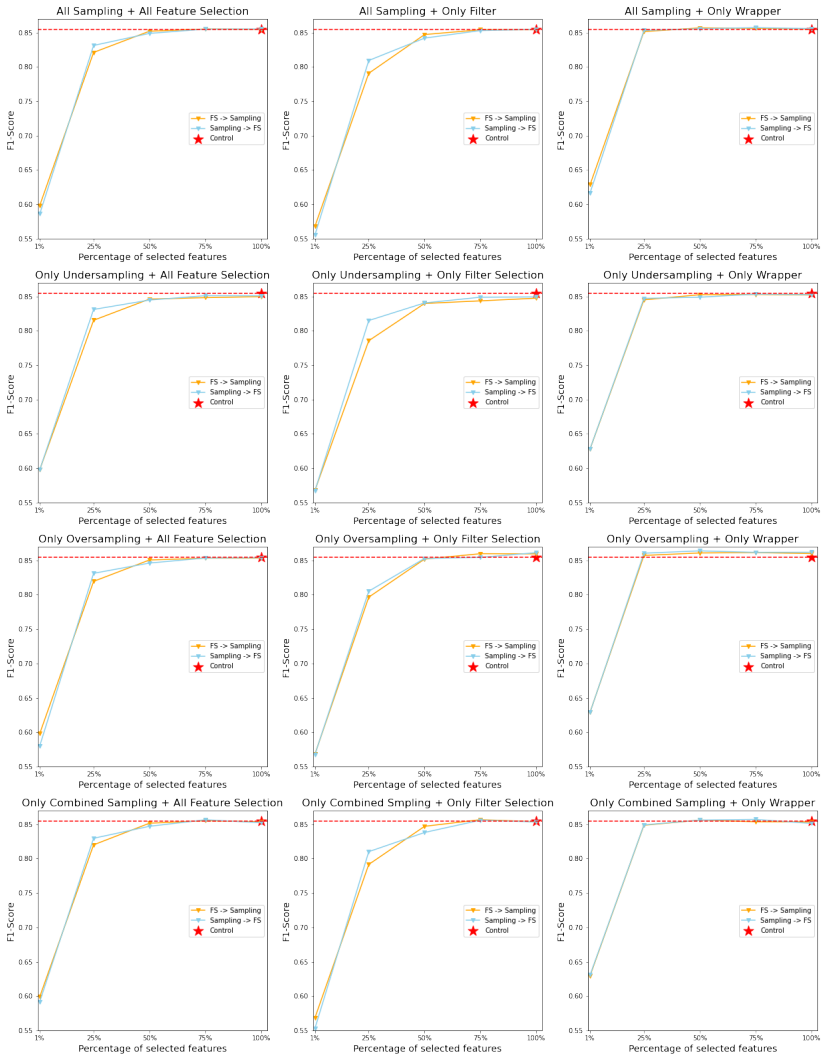

FIGURE A.3: Results of the pipeline for the Bank Fraud dataset.

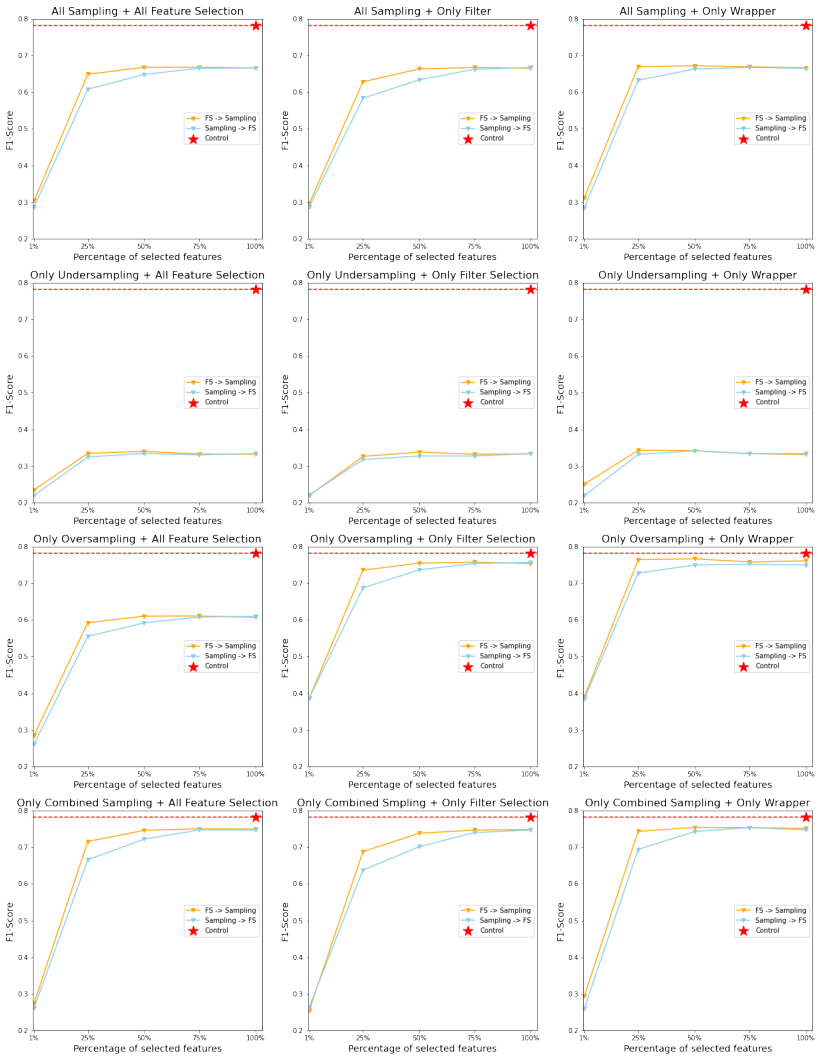

FIGURE A.4: Results of the pipeline for the APS Failure at Scania dataset.

Appendix B

Clinton 2021 Data

B.1 Variables Description

Table B.1: Description of the variables in the Clinton 2021 dataset.

	Variable Name	Description
0	bead_lot	Lot of the bead used for the tire
1	bead_shift_date	Recorded date of the bead production process
2	bead_workcenter	Equipment ID of the machine where the bead process
3	bladder	Identifer for the blader used when molding the tire
4	ct_shiftdate	Recorded date of when the tire was cured
5	ct_shiftid	Shift during which the tire was cured
6	ct_workcenter	Identifier of the machine where the tire was cured
7	customer	Company or customer that the tire is being produced for.
8	first_breaker_lot	Lot used in the first breaker process
9	first_breaker_shift_date	Recorded date for the first breaker process
10	first_breaker_workcenter	Equipment ID of the machine in the first breaker process
11	first_ply_lot	Identifier for the lot used in the first ply process
12	first_ply_shift_date	Recorded date for the first ply process
13	first_ply_workcenter	Equipment ID used in the first ply process
14	gt_shiftdate	Date of the last operation on the green tire (before curing)
15	gt_shiftid	Shift ID that made the last operation on the green tire
16	gt_workcenter	Equipment ID used in the last operation on the green tire
17	inner_liner_lot	Lot of the sheet used in the calendering process
18	inner_liner_shift_date	Recorded date of the calendering process
19	inner_liner_workcenter	Equipment ID of the calendering process
20	mold	Identifer for the mold used when molding the tire.
21	second_breaker_lot	Lot used in the second breaker process
22	second_breaker_shift_date	Recorded date for the second breaker process
23	second_breaker_workcenter	Equipment ID of the machine in second breaker process
24	sidewall_lot	Identifer for the lot used in the sidewall process

Continued on next page

Table B.1:Description of the variables in the Clinton 2021 dataset (cont.).

	Name	Explanation
25	sidewall_shift_date	Recorded date of the sidewall process
26	sidewall_workcenter	Equipment ID used in the sidewall process
27	tire_weight_actual	Measured weight for the tire.
28	tire_weight_target	Ideal weight for the produced OE tire.
29	tread_lot	Lot of the tread used in the tire
30	tread_shift_date	Recorded date that the tread was added to the tire
31	tread_workcenter	Equipment ID of the machine used to add the tread
32	overallgrade	Determines whether a tire is scrapped, replaced or approved.

B.2 Variables Distribution

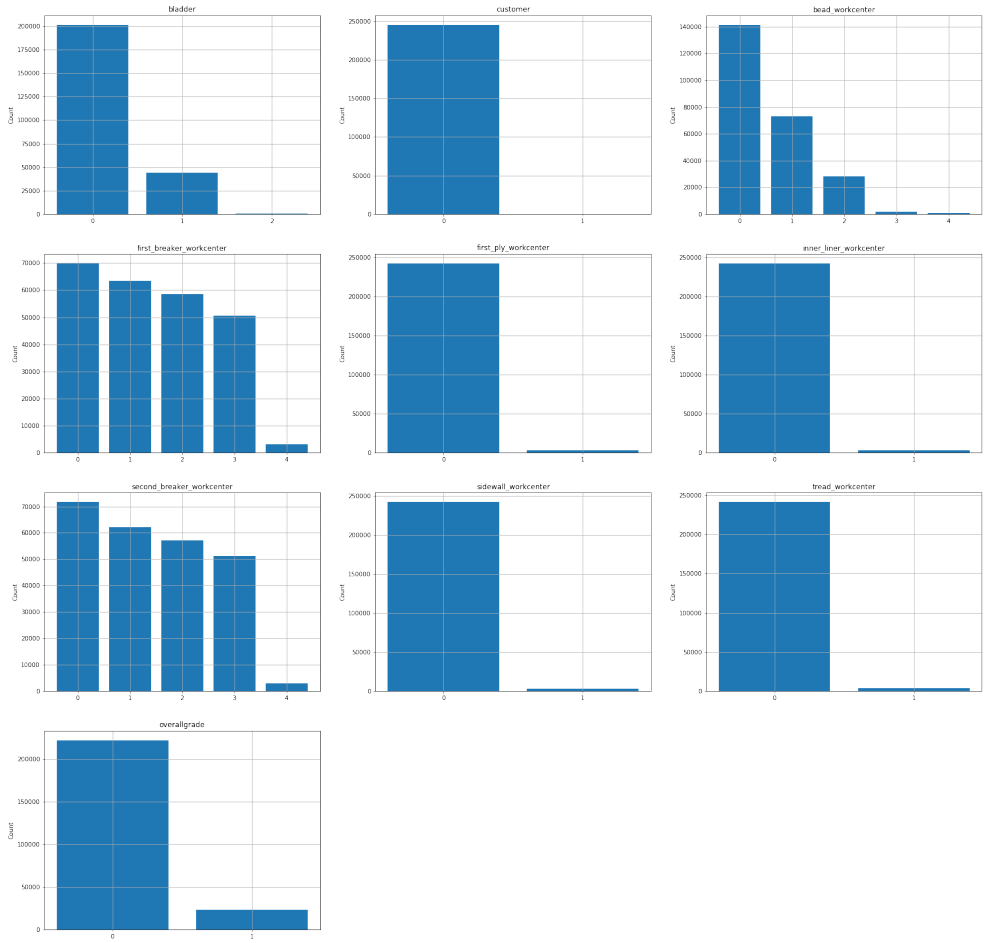

FIGURE B.1: Bar plots of the categorical variables in the Clinton 2021 dataset.

FIGURE B.2: Histograms of the numerical variables in the Clinton 2021 dataset.

Appendix C

Clinton 2021 Results

Table C.1: Results of the pipeline for the Clinton 2021 dataset when selecting 1% of features.

		F1-Score		MCC	
FS Method	Sampling Method	FS → S	S → FS	FS → S	S → FS
F-Test Quotient	RUS	**0.172465**	0.172452	**0.027966**	0.026404
	ROS	0.170290	0.170290	0.021811	0.021811
	SMOTE	0.162798	0.169844	0.002649	0.021447
	Borderline SMOTE	0.162798	0.164848	0.002649	-0.001430
	SMOTE Tomek Links	0.162798	0.169844	0.002649	0.021447
Mutual Information	RUS	0.207270	0.208283	0.093168	0.095152
	ROS	**0.216959**	0.171615	**0.108752**	0.003266
	SMOTE	0.005018	0.171615	0.003370	0.003266
	Borderline SMOTE	0.005023	0.171615	0.004136	0.003266
	SMOTE Tomek Links	0.003917	0.215842	0.001888	0.105423
Signal 2 Noise	RUS	0.027917	0.027917	0.001756	0.001756
	ROS	0.027917	0.027917	0.001756	0.001756
	SMOTE	0.027917	0.027917	0.001756	0.001756
	Borderline SMOTE	0.027917	0.027917	0.001756	0.001756
	SMOTE Tomek Links	0.027917	0.027917	0.001756	0.001756
Boruta	RUS	**0.164387**	0.164175	**0.007663**	0.006976
	ROS	0.161791	0.163188	0.004018	0.006752
	SMOTE	0.008512	0.157089	0.004521	0.001342
	Borderline SMOTE	0.008512	0.160354	0.004521	-0.000426
	SMOTE Tomek Links	0.008512	0.156327	0.004521	-0.000412
SFS	RUS	0.206540	0.209375	0.091168	0.096359
	ROS	**0.216273**	0.210219	**0.107572**	0.097134
	SMOTE	0.005299	0.209090	0.004827	0.095730
	Borderline SMOTE	0.005027	0.199782	0.004828	0.079075
	SMOTE Tomek Links	0.003920	0.209281	0.002699	0.095976
RFE	RUS	0.208137	0.164722	0.094738	0.008060
	ROS	**0.217244**	0.162741	**0.109250**	0.005619
	SMOTE	0.005582	0.156572	0.006328	-0.000191
	Borderline SMOTE	0.004747	0.160633	0.003543	0.000049
	SMOTE Tomek Links	0.003919	0.157063	0.002346	0.001382

Table C.2: Results of the pipeline for the Clinton 2021 dataset when selecting 25% of features.

		F1-Score		MCC	
FS Method	Sampling Method	FS → S	S → FS	FS → S	S → FS
F-Test Quotient	RUS	**0.193933**	0.186541	0.068411	0.054342
	ROS	0.187208	0.188697	0.061771	0.060633
	SMOTE	0.181523	0.186327	0.076704	0.057912
	Borderline SMOTE	0.188499	0.185719	**0.089116**	0.057639
	SMOTE Tomek Links	0.181075	0.185631	0.076259	0.056749
Mutual Information	RUS	0.208946	0.207816	0.096026	0.093999
	ROS	0.171508	0.213457	0.082148	0.101427
	SMOTE	0.196024	0.222381	0.101612	0.123587
	Borderline SMOTE	0.199068	**0.222530**	0.105649	0.122373
	SMOTE Tomek Links	0.194722	0.223074	0.103490	**0.124162**
Signal 2 Noise	RUS	**0.170013**	0.167874	**0.017980**	0.015926
	ROS	0.167526	0.167522	0.017852	0.017843
	SMOTE	0.163946	0.167456	0.010799	0.017646
	Borderline SMOTE	0.163112	0.165998	0.010446	0.006073
	SMOTE Tomek Links	0.163946	0.167418	0.010799	0.017471
Boruta	RUS	0.205871	**0.206824**	0.089848	0.091575
	ROS	0.175897	0.176979	0.105865	0.106633
	SMOTE	0.165289	0.177696	0.102945	0.103404
	Borderline SMOTE	0.165327	0.180096	0.103685	0.106708
	SMOTE Tomek Links	0.167231	0.178969	**0.111808**	0.106938
SFS	RUS	**0.214365**	0.209468	**0.104955**	0.096343
	ROS	0.152895	0.210341	0.094396	0.097310
	SMOTE	0.030153	0.209751	0.007663	0.096511
	Borderline SMOTE	0.027148	0.202125	0.005071	0.083176
	SMOTE Tomek Links	0.026149	0.209649	0.003442	0.096324
	RUS	0.206831	**0.207393**	0.091605	0.092535
RFE	ROS	0.176870	0.177792	0.107049	0.107400
	SMOTE	0.167792	0.176301	0.105288	0.102231
	Borderline SMOTE	0.173009	0.179689	0.105781	0.1061990
	SMOTE Tomek Links	0.164559	0.180196	0.106436	**0.108125**

Table C.3: Results of the pipeline for the Clinton 2021 dataset when selecting 50% of features.

		F1-Score		MCC	
FS Method	Sampling Method	FS → S	S → FS	FS → S	S → FS
F-Test Quotient	RUS	0.209719	**0.210011**	0.096545	0.097033
	ROS	0.176380	0.173474	**0.110602**	0.105717
	SMOTE	0.168562	0.19117	0.096448	0.107196
	Borderline SMOTE	0.156971	0.190668	0.078917	0.109928
	SMOTE Tomek Links	0.168215	0.194084	0.102885	0.110426
Mutual Information	RUS	0.210572	0.207379	0.098653	0.092961
	ROS	0.174981	0.213347	0.087185	0.100834
	SMOTE	0.192478	0.221322	0.120778	**0.129116**
	Borderline SMOTE	0.191538	0.220336	0.121053	0.125038
	SMOTE Tomek Links	0.191495	**0.221604**	0.120431	0.127539
Signal 2 Noise	RUS	0.204785	0.203318	0.088131	0.085511
	ROS	0.203004	0.203578	0.100264	0.100891
	SMOTE	0.205232	**0.208600**	0.112022	0.114606
	Borderline SMOTE	0.205905	0.206875	0.113410	0.112699
	SMOTE Tomek Links	0.204645	0.208333	0.111686	**0.114662**
Boruta	RUS	**0.212520**	0.212504	0.101427	0.101416
	ROS	0.182919	0.182175	0.112248	0.111364
	SMOTE	0.186737	0.181730	0.121390	0.109981
	Borderline SMOTE	0.185588	0.183374	0.120412	0.112263
	SMOTE Tomek Links	0.189339	0.184839	**0.127274**	0.115747
SFS	RUS	0.209412	0.211399	0.096456	0.100315
	ROS	0.162196	0.215186	0.102677	0.104761
	SMOTE	0.167017	0.216989	0.104197	0.106282
	Borderline SMOTE	0.169435	0.215630	0.106703	0.104287
	SMOTE Tomek Links	0.169636	**0.219511**	**0.112332**	0.110017
RFE	RUS	**0.213358**	0.212701	0.102841	0.101652
	ROS	0.186544	0.183878	0.116958	0.114708
	SMOTE	0.185777	0.184679	0.120769	0.115815
	Borderline SMOTE	0.185667	0.187040	0.120560	0.119915
	SMOTE Tomek Links	0.185452	0.190103	**0.123605**	0.122849

Table C.4: Results of the pipeline for the Clinton 2021 dataset when selecting 75% of features.

		F1-Score		MCC	
FS Method	Sampling Method	FS → S	S → FS	FS → S	S → FS
F-Test Quotient	RUS	**0.214072**	0.213927	0.104099	0.103820
	ROS	0.190332	0.188049	0.117435	0.114892
	SMOTE	0.187490	0.197048	0.117741	0.119136
	Borderline SMOTE	0.188476	0.199082	0.121948	**0.121994**
	SMOTE Tomek Links	0.187553	0.199833	0.121115	0.121895
Mutual Information	RUS	0.211208	0.208737	0.099421	0.095308
	ROS	0.177683	0.213646	0.090563	0.101264
	SMOTE	0.193616	**0.221743**	0.122143	**0.130757**
	Borderline SMOTE	0.194755	0.221219	0.125120	0.128508
	SMOTE Tomek Links	0.195094	0.222096	0.125174	0.130844
Signal 2 Noise	RUS	0.208494	0.208697	0.094572	0.094998
	ROS	0.202542	0.201510	0.105291	0.104051
	SMOTE	**0.209868**	0.191337	**0.120097**	0.112718
	Borderline SMOTE	0.204454	0.192852	0.115477	0.114468
	SMOTE Tomek Links	0.208397	0.195468	0.118983	0.117706
Boruta	RUS	**0.215344**	0.215041	0.106342	0.105765
	ROS	0.185875	0.187996	0.115402	0.117966
	SMOTE	0.187746	0.188682	0.120475	0.121095
	Borderline SMOTE	0.188419	0.191650	0.121968	0.125199
	SMOTE Tomek Links	0.191448	0.192038	**0.126965**	0.126785
SFS	RUS	0.215117	0.211749	0.105998	0.100843
	ROS	0.178532	0.214824	0.111056	0.103447
	SMOTE	0.187258	0.225667	0.121074	0.124402
	Borderline SMOTE	0.188676	0.225556	0.123465	0.125572
	SMOTE Tomek Links	0.189746	**0.225684**	**0.127083**	0.124149
RFE	RUS	**0.216074**	0.214629	0.107528	0.104967
	ROS	0.188210	0.187101	0.118009	0.116925
	SMOTE	0.188894	0.189664	0.122222	0.122653
	Borderline SMOTE	0.191391	0.189707	0.125588	0.123475
	SMOTE Tomek Links	0.191396	0.194208	0.126354	**0.129041**

Table C.5: Results of the pipeline for the Clinton 2021 dataset when selecting 100% of features.

		F1-Score		MCC	
FS Method	Sampling Method	FS → S	S → FS	FS → S	S → FS
F-Test Quotient	RUS	**0.216257**	0.216443	0.107840	0.108144
	ROS	0.188744	0.189650	0.118289	0.118574
	SMOTE	0.188058	0.190802	0.120682	0.123402
	Borderline SMOTE	0.189527	0.189908	0.122769	0.123473
	SMOTE Tomek Links	0.194526	0.192826	**0.129009**	0.127116
Mutual Information	RUS	**0.214701**	0.213945	0.105057	0.103806
	ROS	0.188260	0.188036	0.117873	0.117582
	SMOTE	0.187963	0.187010	0.120141	0.119488
	Borderline SMOTE	0.188401	0.187783	0.121560	0.121041
	SMOTE Tomek Links	0.193276	0.191649	**0.128071**	0.126065
Signal 2 Noise	RUS	0.210992	**0.212273**	0.098952	0.101092
	ROS	0.202767	0.202906	0.110039	0.110154
	SMOTE	0.196347	0.194367	**0.119782**	0.118067
	Borderline SMOTE	0.194531	0.194770	0.118708	0.118838
	SMOTE Tomek Links	0.195773	0.193363	0.119263	0.117192
Boruta	RUS	**0.216593**	0.215532	0.108415	0.106579
	ROS	0.187652	0.190437	0.116560	0.119977
	SMOTE	0.191658	0.187859	0.124245	0.120071
	Borderline SMOTE	0.189815	0.191315	0.122808	0.124828
	SMOTE Tomek Links	0.195192	0.194245	**0.129986**	0.128986
SFS	RUS	**0.215513**	0.214166	0.106542	0.104293
	ROS	0.187996	0.187820	0.117966	0.112961
	SMOTE	0.186933	0.198327	0.118980	0.125846
	Borderline SMOTE	0.188724	0.199550	0.122912	**0.128154**
	SMOTE Tomek Links	0.191992	0.198179	0.126953	0.125987
RFE	RUS	**0.215720**	0.215139	0.106900	0.105868
	ROS	0.187103	0.189081	0.116248	0.118334
	SMOTE	0.186582	0.189463	0.118947	0.122652
	Borderline SMOTE	0.191382	0.191363	0.125447	0.124916
	SMOTE Tomek Links	0.191473	0.191832	0.125991	**0.126910**

Bibliography

[1] Hakan Altincay and Cem Ergün. Clustering based under-sampling for improving speaker verification decisions using adaboost. volume 3138, pages 698–706, 08 2004. ISBN: 978-3-540-22570-6. doi:10.1007/978-3-540-27868-9_76.

[2] Gustavo Batista, Ana Bazzan, and Maria-Carolina Monard. Balancing training data for automated annotation of keywords: a case study. pages 10–18, 01 2003.

[3] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, jun 2002. doi:10.1613/jair.953.

[4] Continental. Headquarters & plants. https://www.continental-tires.com/transport/company/businessunit/headquarters-plants. Accessed: 21/09/2022.

[5] C. Ding and H. Peng. Minimum redundancy feature selection from microarray gene expression data. In *Computational Systems Bioinformatics. CSB2003. Proceedings of the 2003 IEEE Bioinformatics Conference. CSB2003*, pages 523–528, 2003. doi:10.1109/CSB.2003.1227396.

[6] F.J. Ferri, Pavel Pudil, and M. Hatef. Comparative study of techniques for large-scale feature selection. *Pattern Recognition in Practice, IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, 16, 12 2001. doi:10.1016/B978-0-444-81892-8.50040-7.

[7] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422, 01 2002. doi:10.1023/A:1012487302797.

[8] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: A new over-sampling method in imbalanced data sets learning. In De-Shuang Huang, Xiao-Ping Zhang, and Guang-Bin Huang, editors, *Advances in Intelligent Computing*, pages 878–887, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN: 978-3-540-31902-3.

[9] P. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516, 1968. doi:10.1109/TIT.1968.1054155.

[10] IBM. Crisp-dm help overview. https://www.ibm.com/docs/en/spss-modeler/saas?topic=dm-crisp-help-overview. Accessed: 12/12/2022.

[11] Bob Rupak Roy II. Borderline smote. https://bobrupakroy.medium.com/borderline-knn-svm-and-adaysn-smote-1d74756fb049. Accessed: 12/07/2022.

[12] Imbalanced-learn. Compare under-sampling samplers. https://imbalanced-learn.org/stable/auto_examples/under-sampling/plot_comparison_under_sampling.html#sphx-glr-auto-examples-under-sampling-plot-comparison-under-sampling-py. Accessed: 06/07/2022.

[13] J. Kreer. A question of terminology. *IRE Transactions on Information Theory*, 3(3):208–208, 1957. doi:10.1109/TIT.1957.1057418.

[14] Miron Kursa and Witold Rudnicki. Feature selection with boruta package. *Journal of Statistical Software*, 36:1–13, 09 2010. doi:10.18637/jss.v036.i11.

[15] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, and Huan Liu. Feature selection. *ACM Computing Surveys*, 50(6):1–45, nov 2018. doi:10.1145/3136625.

[16] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28 (2):129–137, 1982. doi:10.1109/TIT.1982.1056489.

[17] B.W. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442–451, 1975. ISSN: 0005-2795. doi:https://doi.org/10.1016/0005-2795(75)90109-9.

[18] Samuele Mazzanti. Boruta explained exactly how you wished someone explained to you. https://towardsdatascience.com/boruta-explained-the-way-i-wish-someone-explained-it-to-me-4489d70e154a. Accessed: 06/07/2022.

[19] Hanchuan Peng, Fuhui Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005. doi:10.1109/TPAMI.2005.159.

[20] Brian C. Ross. Mutual information between discrete and continuous data sets. *PLOS ONE*, 9(2):1–5, 02 2014. doi:10.1371/journal.pone.0087357.

[21] Max Schubach, Matteo Re, Peter Robinson, and Giorgio Valentini. Imbalance-aware machine learning for predicting rare and common disease-associated non-coding variants. *Scientific Reports*, 7, 06 2017. doi:10.1038/s41598-017-03011-5.

[22] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi:10.1002/j.1538-7305.1948.tb01338.x.

[23] Ivan Tomek. Two modifications of cnn. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(11):769–772, 1976. doi:10.1109/TSMC.1976.4309452.

[24] Irvine Machine Learning Repository University of California. Uci machine learning repository. https://archive.ics.uci.edu/ml/index.php. Accessed: 21/09/2022.

[25] Chongsheng Zhang, Jingjun Bi, and Paolo Soda. Feature selection and resampling in class imbalance learning: Which comes first? an empirical study in the biological domain. In *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 933–938, 2017. doi:10.1109/BIBM.2017.8217782.

[26] Zhenyu Zhao, Radhika Anand, and Mallory Wang. Maximum relevance and minimum redundancy feature selection methods for a marketing machine learning platform. In *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 442–452, 2019. doi:10.1109/DSAA.2019.00059.