IMPROVING ON IMBALANCED DATA CLASSIFICATION BY FEATURE ENGINEERING COMBINED WITH RANDOM UNDER-SAMPLING

Levy Ardon

STUDENT NUMBER: 2019860

THESIS SUBMITTED IN PARTIAL FULFILMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN DATA SCIENCE & SOCIETY

DEPARTMENT OF COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE

SCHOOL OF HUMANITIES AND DIGITAL SCIENCES

TILBURG UNIVERSITY

Thesis committee:

Supervisor

Dr. Marie Postma

Second reader

Dr. Daniel Schad

Tilburg University

School of Humanities and Digital Sciences

Department of Cognitive Science & Artificial Intelligence

Tilburg, The Netherlands

May, 2020

I would like to thank Dr. Marie Postma for supervising my thesis, and for her excellent support during this project. Many thanks for the valuable feedback sessions and the various meetings we had!

Abstract

This Master thesis aims at improving the predictive performance of an AdaBoost and XGBoost classification algorithm on an imbalanced dataset. Classification on imbalanced data, or rare events, is a common phenomenon in real-life that can have significant effects on individuals or the society as a whole and is, therefore, an active area of research. According to the literature, two ways of combating an unbalanced label distribution in order to improve on the predictive performance of a classifier are on the algorithmic level, and on the data level. In this research, the data level approach is implemented by applying a novel strategy of random under-sampling in combination with adding additional engineered features to the dataset. The course of actions proposed are captured in the research question: *"What effect has feature engineering, in combination with random under-sampling on imbalanced data for classification tasks?".* A total of two experimental set-ups were proposed, only differing from each other in the number of feature engineering and selection runs adopted. The dataset consists of multivariate time series process log data of a pulp-to-paper manufacturer. It was found that the proposed experimental set-ups did not improve the predictive performance of the AdaBoost and XGBoost. However, the proposed experimental set-ups were tested also on the hybrid RUSBoost classifier that combines an AdaBoost with an automated random under-sampling strategy[1]. This classifier improved the benchmark F_1-score of 0.114 by 84.2% to 0.21.

[1] Random under-sampling was performed automatically by the RUSBoost classifier. So, no random under-sampling was performed during the pre-processing (preparing the experimental set-ups), only the engineering of features.

Table of Contents

1. Introduction

Data-driven process control is growing in popularity. In industrial applications, data science has an increasing role in the analysis of data, including the detecting of rare events. As the volume at which industries generate raw process data continues to grow, organizations acknowledge the potential of data analysis to benefit their core operations. The abundance of data, combined with sophisticated learning algorithms and improved computational power, has enabled companies to acquire new insights into their business processes and act accordingly. Organizations that effectively apply the capabilities of data analytics, can differentiate themselves by creating significant value (Henke, et al., 2016).

This project concerns a dataset of a pulp-to-paper manufacturer that contains multivariate time series process log data. In previous research performed by Ranjan et al. (2018), two machine learning algorithms (AdaBoost and XGBoost) were applied to this particular dataset in order to discover patterns in the data that occur before the rare event of a production process error. This research aims to further improve on the predictive performance of those two models in rare event prediction by implementing a resampling strategy in combination with adding additional engineered features to the data set. Ultimately, a machine learning model that can reliably establish hazardous patterns of rare events will enable process operators to take timely measures against a predicted error once such a pattern is detected in a production run. The usage of the data will contribute to optimizing the production process as errors could be avoided, resulting in a reduction of downtime of machines and an overall increase in productivity of the factory.

Rare event prediction remains a topic of discussion in the scientific community. Although various research has been performed (Krawczyk, 2016), there is no one-size-fits-all solution to what is the best machine learning method for rare event prediction; a certain machine learning algorithm might be superior in one rare event prediction task, but fails catastrophically in another (Mair, et al., 2000). Additionally, recently the focus has shifted from traditional machine learning algorithms to more sophisticated deep learning models. Karim et al. (2017) successfully applied Long Short Term Memory Fully Convolutional Network (LSTM-FCN) and Attention Long Short Term Memory Fully Convolutional Network (ALSTM-FCN) to rare event prediction in univariate time series data. Further research by Karim et al. (2019) enabled multivariate time series data as input. The results showed accuracy rates that outpaced a selection of respected benchmark machine learning algorithms. Nevertheless, while deep learning is a promising method for rare event prediction, deep learning models are regarded as “black boxes” that are hard to understand and to interpret (Zhang, Tan, Han, & Zhu, 2017).

In comparison to deep learning models, traditional machine learning algorithms are less complicated, better to understand, and have a proven record of success (Zhang, Tan, Han, & Zhu, 2017), and are therefore the method of choice in this project. Also, since Ranjan et al. (2018) only performed minor pre-processing modifications in their research to the dataset, it is projected that by implementing a resampling strategy[2] in combination with adding additional engineered features to the data set, the predictive performance of the machine learning models will increase. This research will use the pulp-to-paper dataset in order to answer the following research question:

"What effect has feature engineering, in combination with random under-sampling on imbalanced data for classification tasks?"

According to the available literature, this particular question has not been answered yet. Answering this question will contribute to the existing scientific knowledge regarding the effect that certain methods have on the predictive behaviour of machine learning algorithms on imbalanced data classification tasks. This approach differs from the one tested by Qazi et al. (2012) as instead of a multiclass classification problem it is a binary classification task. The proposed course of action also is in line with the suggestions made by Ranjan et al. (2018) regarding their recommendations on how to potentially increase the predictive performance of the AdaBoost and XGBoost.

Additionally, a better understanding of how to enhance predictive performance on rare event data will have a positive impact on society as well. Although situations of rare events are scarce in real life, their consequences to society can be significant. Early detection of a potential oil-spill could prevent an environmental disaster from happening, a reliable predictor of rare diseases could potentially save the lives of thousands or millions of people, and a credit card fraud detector can benefit customer trust in banks (Ganganwar, 2012). Moreover, rare event situations have a significant economic impact too. It is estimated that on average, a factory loses between 5 - 20% of its production capacity due to downtime (Fitchett & Sondalini, 2017). Industry experts claim that an estimated 80% of organizations fail to accurately predict downtime, causing a 200-300% increase of true downtime costs compared to forecasted numbers (Fitchett & Sondalini, 2017).

In this research, it was found that the proposed courses of action did not result in a significant increase in the predictive performance of the AdaBoost and XGBoost classifier. Nevertheless, the hybrid RUSBoost algorithm, which essentially is an AdaBoost algorithm that automatically performs random under-sampling, yielded a promising F_1-score of 0.21.

[2] The choice for random under-sampling will be further explained in chapter 3.4.

2. Related Work

This section of the paper provides an overview of related work in regards to imbalanced datasets, or rare events. First the term "rare events" will be defined, after which a brief historical background is provided. Next, some more in-depth information regarding various remedies for handling imbalanced datasets is reported.

2.1 Rare Events

One definition of rare event data is a dataset of binary dependent variables with dozens to thousands of times fewer observations belonging to the minority class than to the majority class (King & Zeng, 2001). Rare events "are patterns in data that do not conform to a well-defined notion of normal behaviour" (Chandola, Banerjee, & Kumar, 2009). This anomalous behaviour is of great interest to accurately predict as it indicates exceptions to the rules (Chandola, Banerjee, & Kumar, 2009). Some examples of anomalous events are fraudulent transaction detection (Bolton, 2002), rare disease classification (Santoro, et al., 2015) ,and faulty telecom network behaviour (Sasisekharan, Seshadri, & Weiss, 1996).

2.1.1 Brief History of Rare Event Prediction

One of the earliest recorded studies regarding the detection of rare event observations in data originates back to the 19th century (Edgeworth, 1887). Early research built on engineering and physical models to calculate the probability of rare events that started in the late 1970s (Rackwitz & Fiessler, 1978). The need for sophisticated methods that could handle uncertainties in the design process of structural systems grew as safety requirement standards increased. In research regarding catastrophic failures prediction (Peck, 1969), system data was collected by observation in order to obtain useful information regarding the rare event of a technical failure. The Bayesian approach proved to be an appropriate framework to analyse the large quantities of data as it incorporated an inverse problem-solving technique that, given the output variable, determined the probability of any input variable.

2.1.2 Problems with Traditional ML-models

As indicated by King & Zeng (2001), the rare event of interest is inadequately represented in the dataset with only the minority of all observations belonging to the positive class, while the majority is among the negative class. Because of the relatively few positive observations in the dataset, there is an unbalanced data distribution with the learning algorithm being highly biased towards the majority class (Wah, Rahman, He, & Bulgiba, 2016).

As traditional algorithms assume a normal distribution of the data, machine learning performance will usually be low on imbalanced data (Wah, Rahman, He, & Bulgiba, 2016). As the majority class has a larger volume than the minority class, it will have a greater influence on the training stage of the

machine learning model (Hastie, Tibshirani, & Friedman, 2009). Usually, the machine learning model will yield a good accuracy score on the majority class, but performs poorly on the minority class. Achieving a minimal error rate on a test set of data is the aim of most algorithms, which is achieved by emphasising on the majority class. In imbalanced data classification tasks, a high accuracy score can be achieved by only predicting the majority class (Hastie, Tibshirani, & Friedman, 2009). However, although this method can result in a minimal error rate, the misclassification of the minority class is neglected as misclassification costs are assumed to be equal. The assumption of equal misclassification costs per class should be carefully examined, as this might not be applicable (Ganganwar, 2012). An example of unequal misclassification costs is to wrongfully classify a healthy person as unhealthy, and misclassify a sick person as healthy. In the latter case, the person that has been classified as healthy (false negative) will not receive appropriate treatment, which might lead to serious complications (Ganganwar, 2012).

2.2 Remedies for Imbalanced Datasets

To overcome the problem of imbalanced data distribution, two solutions were proposed by Abdi and Hashemi (2015), and include an algorithmic level, and a data level approach.

One proposed solution to rectify class imbalance on an algorithmic level is to implement a cost-sensitive method. This method assigns different misclassification costs to the different classes (He & Garcia, 2009). Bagging and Boosting are two examples of ensembled learning algorithms that incorporate a cost-sensitive method (Geiler, Hong, & Yue-Jian, 2010). Boosting algorithms place more emphasis on the weak learners (positive class), and have been widely used for imbalanced class prediction problems (Li, et al., 2017). Ranjan et al. (2018) used an AdaBoost and XGBoost algorithm to predict rare events in their research. However, the cost-sensitive method may find difficulties in implementation as misclassification costs are not always available or cannot be generated at all (Zong, Huang, & Chen, 2013). Also, boosting can lead to poor test results as the noise might be captured due to over-emphasising on noisy observations in the data (Maclin & Opitz, 1997). Nevertheless, ensemble methods have been applied to numerous real-world machine learning problems, including class imbalance data, and are regarded as highly versatile and effective by the data science community (Polikar, 2012).

Resampling is a technique that aims to repair the unbalanced distribution on the data level (Wasikowski & Chen, 2010). Popular resampling techniques include over and under-sampling. Oversampling is a method applied to overcome class imbalance by selecting and duplicating observations that belong to the positive class. This process continues until the minority and majority class have equal observations. Minority class observations can be duplicated in a random or focused manner. The random selection (ROS) randomly duplicates minority class observations without any

further requirements, where the focused approach (FOS) randomly duplicated the border observations of the minority class (Wah, Rahman, He, & Bulgiba, 2016). Under-sampling is the opposite of oversampling and balances the dataset by excluding (RUS for randomly selected and FUS for focused selection) from the negative class. A research performed by Park and Jung (2019) on different methods of resampling techniques showed that RUS resulted in the highest sensitivity, with oversampling through an adaptive synthetic sampling approach as the next-best method. Nevertheless, data level solutions may lead to overfitting of the learning algorithm (oversampling), or misleading the learning algorithm by potentially removing instances that contain useful information (under-sampling) (Abdi & Hashemi, 2015).

Ranjan et al. (2018) recommended to collect additional data, and so increase the training data as another data level approach to balance the data. In recent research by Zhu et al. (2016), the question was addressed if adding more training data would increase performance. To test this hypothesis, an SVM-model was trained and tested on a number of differently sized training sets. The results were contradictory to the suggestion made by Ranjan; under certain circumstances, increasing the training set can decrease model performance. Models tend to perform worse when more training examples are added if the additional data has an unbalanced distribution. As the majority of the additionally added examples belong to the negative class, the model will skew even more towards this class. This will ultimately lead to poor performance regarding correctly classifying the positive class (Zhu, Vondrick, Fowlkes, & Ramanan, 2016). Research by Li et al. (2017) showed that an increase in data could lead to an improvement in the recall score. However, the research also showed that a more balanced data distribution has a greater effect on the results while fewer training observations are required.

2.3 Feature Selection for Combating Class Imbalance

Feature selection is the process of selecting a subset of *j* features from the original set of features (Wasikowski & Chen, 2010). Zheng et al. (2004) describe feature selection as selecting only those features that effectively discriminate between two classes (in a binary classification problem), assigning a positive value to one feature (belonging to class X), and a negative value to another feature (belonging to class Y). Feature selection is highly important, yet often neglected, since it filters those features that are informative in regards to the Y-variable from the features that do not contribute to the classification of the dependent variable. Guo et al. (2006) dedicated research to the development of a framework to evaluate the reliability of sensor data as they argued that sensor data is inherently unreliable caused by technical failures as well as the tendency to capture noise. Again does this confirm the importance of feature selection as not all variables contribute to the dependent variable, including sensor data.

Evidence was found by van der Put and van Someren (2004) that feature selection has significantly more influence on the performance of a machine learning model on classification problems then the chosen algorithm itself. Brownlee (2014) emphasises the importance of feature selection too and claimed that only features that describe the inherent structures of the dataset will lead to good model performance. Moreover, even if an appropriate machine learning model is chosen, as the features do not describe the inherent structure of the data correctly, the performance will always be poor (Brownlee, 2014). Wasikowski et al. (2010) performed specific research on feature selection for imbalanced datasets and found that feature selection benefits machine learning model performance on unbalanced data classification tasks. Feature selection on text classification tasks yielded similar results, with an overall improvement of the predictive performance of the machine learning model on an imbalanced dataset (Wasikowski & Chen, 2010).

In research by Qazi and Raza (2012) the effects of feature selection in combination with Synthetic Minority Over-Sampling (SMOTE) or under-sampling were tested on imbalanced data. This particular research concerned the highly unbalanced multiclass KDD CUPP99 dataset, from which only 0.845% of the total data accounted as the class of interest (the rare event). The researchers acknowledged the importance of feature selection as they assumed that a set of appropriate features enables a classifier to accurately identify patterns. Moreover, the class imbalance was expected to be overcome by applying various major and minor class manipulation techniques. Several different classifiers were utilized in this research, including a decision tree, Naïve Bayes, Radial basis neural network, and a support vector machine. It was found that feature selection in combination with under-sampling was more effective than applying SMOTE. Moreover, the decision tree and Naïve Bayer proved to be the most accurate classifiers of the four. Nevertheless, the paper concluded that first feature selection and then resampling the data by means of under-sampling, in combination with either a decision tree or Naïve Bayes classifier does not perform well on the multiclass classification task.

2.4 Feature Engineering

The second recommendation made by Ranjan et al. (2018) to improve predictive performance was to engineer new features. In research by Bahnsen et al. (2016), it was proposed to add additional features to an unbalanced dataset to overcome the poor predictive performance of machine learning models (including cost-sensitive models) on fraud detection. In their research, a set of new features was engineered which were derived from the original dataset. One of the new features was accumulated by aggregating individual customer transactions over a predetermined time interval. This new feature allowed for further investigation of a custom's spending patterns as e.g. transaction types and countries of purchase could be examined. The results showed an increase of over 200% regarding predictive performance (Bahnsen, Aouada, Stojanovic, & Ottersten, 2016). Another research by Wind

(2014) reported on the importance of feature engineering in a case study regarding predictive modelling competitions on Kaggle.

3. Experimental Setup

3.1 Data Overview

The dataset contains a total of 20,458 observations, with each observation consisting out of 63 attributes. Each data point equals a two-minute interval observation of various sensors that are equipped lengthways the pulp-to-paper machine. The observations include a date/time stamp and are chronologically ordered from May 1st,1999 - 0:00h up to and including May 31st,1999 – 23:58h. The dataset is characterized by an unbalanced data distribution with 20,323 observation belonging to the negative class, and only 135 observations belonging to the positive class, which leads to an error rate ratio of 0.66%. The features represent two kinds of measurements: product-specific variables for raw inputs (various input measures of chemical component like sulphite and silicon carbide), and measurements regarding the production-specific setting (e.g. fan speed, pressure, and couch load). The columns are made up of both integer as well as float values. The dataset is in a CSV-format and was imported into the Jupiter environment using the read_csv function from Pandas (API Reference, sd). Please see appendix I for all features. The dataset is available upon request and can be downloaded from here.

Before any pre-processing actions were taken, the data was checked for NA-values (no NA-values were present in the dataset). Moreover, the "DataTime", "EventPress" and "Grade&Bwt" columns were dropped from the data set as those columns either contained a timestamp or a categorical value.

3.2 Early Prediction

One of the motivations to research improving on rare event prediction is to better enable stakeholders (e.g. process operators in the pulp-to-paper factory) to take timely measures in order to anticipate upon the predicted error. Predicting the error, the dependent variable, is not the goal of this research. Instead, this research aims at learning patterns in the data that indicate an error before happening. Figure 3.1 illustrates the concept of early prediction.

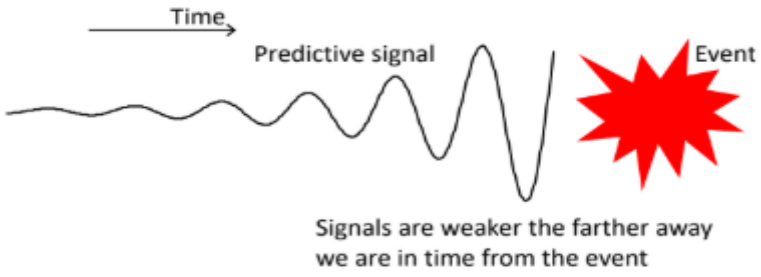


Figure 3.1: illustration of early prediction. Image used with permission from Ranjan (2020).

According to Ranjan (2020), the further away the observation is from the event of interest, the weaker the predictive signal becomes. Nevertheless, since the impact of a rare event can have significant consequences, the stakeholders should be alarmed well in advance. In the exploration of the data, it was discovered that the minimal gap between two errors is two observations of non-errors. The choice was therefore made to select the two observations that happen before the event of an error. To achieve this, the class column ("SheetBreak") Y should be moved up to the prior two observations. Consider the event of an error as Y_t, and the steps back in time are denoted as Q = 1, 2, etc. The "new" event of interest will be $Y_t - Q = Y_t$. Having Q = 2 results in moving back four minutes in time as one time step equals two minutes. An additional benefit of this approach is that the number of Y-variables increases from 135 to 270, which is beneficial for the class imbalance. Moreover, the "old" events of interest can be discarded from the dataset as they do not serve a purpose anymore.

3.3 Feature Engineering and Selection

After the Y-variable has been moved back four minutes in time, the next step in the pre-processing process was performed; feature engineering. Feature engineering by hand is a time consuming process of iterative engineering and testing of features and requires extensive domain knowledge (Nargesian, Samulowitz, Khurana, Khalil, & Turaga, 2017). To overcome the lack of domain knowledge and the limited amount of time available for this project, the choice was made to automate the feature engineering process. A number of open-source automated feature engineering libraries are available, including FeatureTools, tsfresh, and AutoFeat (Horn, Pack, & Rieger, The autofeat Python Library for Automated Feature Engineering and Selection, 2019). Nevertheless, not all libraries were found to be useful for this particular research as assumptions were made by the various libraries regarding the format of the input datasets to engineer features on. The FeatureTools library required input data to be in a relational database format, where the Tsfresh library was designed to engineer features for time-series datasets. The choice was made to deploy the AutoFeat, a recently developed general-purpose feature engineering library (Horn, Pack, & Rieger, 2019).

The engineering of new features was automatically performed by AutoFeatClassifier, which is a function from the open-source library Autofeat. In each user-specified iteration, the AutoFeatClassifier generates a batch of new features. The newly engineered features are derived from multiple non-linear transformations performed on the original input features. After the function has generated a large pool of newly engineered features, automatic feature selection is performed in order to select those features that contain relevant information regarding the dependent variable. Highly correlated features and noisy features are removed from the set of features first before a multi-step L1-regulized logistic regression model performs wrapper methods feature selection strategy. In each step (default feature selection run is 5), the model is trained, fit, and tested on the features. The coefficients of the

features are then evaluated in order to determine their contribution to the prediction of the dependent variable. Features that have a coefficient greater than the largest known coefficient of a noisy feature are selected, those features that fail to meet this requirement are discarded.

For this research, two feature engineering and selection strategies were proposed. The first strategy consists of one feature engineering step in combination with five feature selection runs. The second strategy consists of two feature engineering steps in combination with one feature selection run. Horn et al. (2019) showed in their research that one feature engineering step, in general, outperforms the linear model which is based on the original feature set only. This motivated the choice for the first strategy. Moreover, three feature engineering steps were found to overfit on the data, hence the choice for two feature engineering steps. Since the computational time and RAM-requirements for the feature selection grows exponentially at two feature engineering steps, it was recommended by Horn (2020) to set the "feature_run" parameter to one. The engineered features in the first strategy are limited to seven non-linear transformations, including 1/X, exp(X), log(X), abs(X), √(X), X^2, X^3. The second strategy contains both the seven non-linear transformations as well as combinations of combined pairs of features (original and engineered) with different mathematical operators (e.g. +, -, ÷). For both strategies, the test size was set to 0.20. This means that 80% of the data was used to engineer and test[3] the new features on, and the remaining 20% for validation purposes. Please see table 3.1 for the two proposed strategies.

Table 3.1: Parameter setup for the two proposed strategies

PARAMETERS	EXPERIMENTAL SET-UP 1	EXPERIMENTAL SET-UP 2
"FEATENG_STEPS"	1	2
"FEATSEL_RUNS"	5	1
"TEST_SIZE"	0.2	0.2

Experimental set-up 1: After one feature engineering run a total of 270 new features were engineered and added to the data frame. Next, the data were scaled and five feature selection runs were performed. The AutoFeatClassifier selected a total of 45 engineered features out of the pool with generated features. The final data frame, called "data_1step_5runs", contains 20,323 rows and 104 feature columns. The data frame contains 58 original features, 45 engineered features, and the dependent variable. The new data frame yielded an accuracy score of 0.433 on the training data and an accuracy score of 0.441 on the validation data. The top three engineered features with the highest

[3] Tested in the sense that the coefficients of the newly developed features are evaluated on their contribution to the prediction of the dependent variable. This is necessary in order to filter useful features from features that do not contribute.

coefficient are “CouchSpd**3” (0.000154), “x1PrsTopDrw**3” (0.000032), and “x1PrsTopSpd**3” (0.000022). Please see appendix II for a complete list with all the engineered and selected features.

Experimental set-up 2: The combined number of engineered features in two feature engineering steps totalled up to 53,040. The next action involved filtering the features for noise and correlation, after which 41,023 features remained. After scaling the features, the wrapper strategy performed one feature selection run. From the +40,000 engineered features, the feature selection run found two engineered features that contributed to the classification of the dependent variable. The two engineered features that were found useful include “exp(UpprHdTmpRL)/BleachedGWDFlow” (0.031250) and “1/(CT1BLADEPSI*RSBWSCANAVG)” (0.000295). After the engineered features were added to the original data frame, the new data frame called “data_2steps_1run” contains 61 columns and 20,323 rows. The new data frame achieved an accuracy score of 0.935 on the training data, and 0.931 on the test data.

3.4 Balancing the Data

The final step of the experimental setup preparation is to resample the data. As shown in figure 3.2, it is clear that the class label distribution is highly unbalanced. In other words, there is an imbalance in the data with roughly 99.3% of the observations labelled as a non-error, and the remaining observations as an error. The consequences of imbalanced data have extensively been covered in previous chapters.

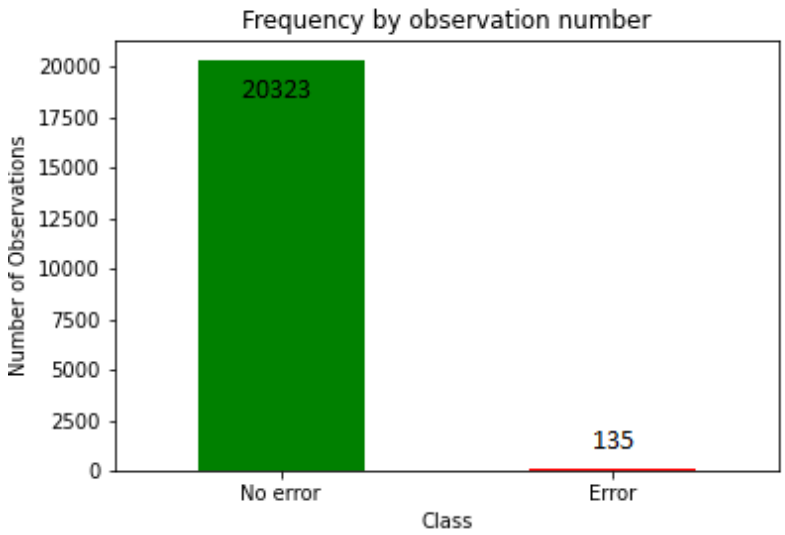


Figure 3.2: Distribution of class labels

To balance the label distribution, the choice was made to use an under-sampling strategy[4]. The under-sampling strategy reduces the majority class by randomly selecting a predefined number or ratio by

[4] Under-sampling is preferred over oversampling as oversampling was found to be an ineffective strategy that contributes little to predictive performance (Drummond & Holte, 2003).

the user, and ignoring the remaining observations. One advantage of under-sampling is that it is less likely to lead to overfitting of the data, and contributes to a reduction in computational time as the overall size of the dataset is vastly reduced (Liu, Wu, & Zhou, 2009). The data was under-sampled via the RandomUnderSampler function from imbalanced-learn Python library[5] (imblearn.under_sampling.RandomUnderSampler, sd). This sampling function first divides all majority class observations into similar clusters before an even proportion of observations are discarded from all clusters until the dataset is balanced. After resampling the training data, the final training set contains 210 error observations (SheetBreak = 1), and 210 non-error observations (SheetBreak = 0). Please see figure 3.3 for the rebalanced label distribution of the training set.

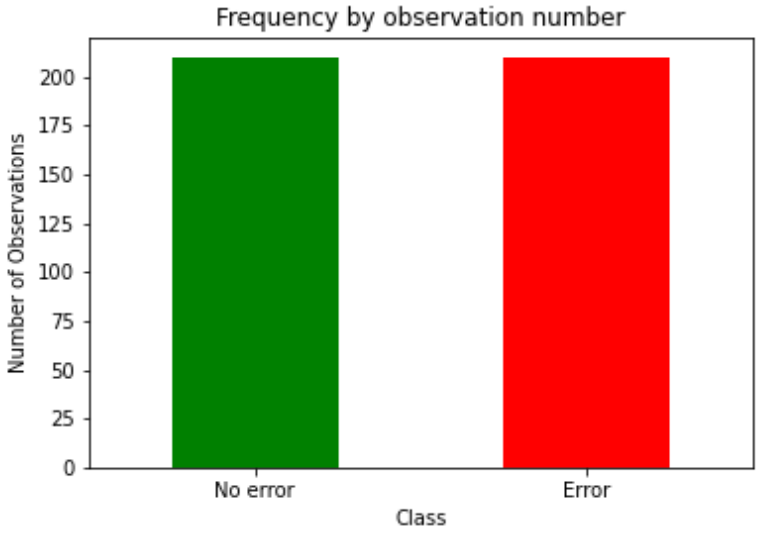


Figure 3.3: Distribution of class labels in training set after resampling

In addition to the random under-sampling strategy, a promising hybrid approach proposed by Seiffert et al. (2010) was applied as well. Seiffert et al (2010) developed an algorithm called RUSBoost that combines an AdaBoost with an automatic random under-sampling strategy. This algorithm was developed in order to alleviate the problem of class imbalance and the poor performance of a more traditional algorithm. The RUSBoost algorithm was tested on 15 datasets and competed with four base learners plus a SMOTEBoost algorithm. The results showed that the RUSBoost algorithm outperformed all four base learners, and performed better than the SMOTEBoost in most of the tasks. Seiffert et al. (2010) highly recommend the RUSBoost algorithm in imbalanced data classification and prediction tasks.

[5] The data was first split into a train (80) and test set (20) (covered in chapter 4). Then the training set was rebalanced by random under-sampling in order to rebalance the label distribution. The training set therefore contains a total of 420 observations (210 errors and 210 non-errors). The test set was not rebalanced (Santos, Soares, Abreu, Helder, & Santos, 2018), and contains 4065 observations (55 errors and 4010 non-errors).

3.5 Summary

In this chapter, all steps regarding preparing the data for the experiment have been covered. The first step taken was to check the original data set for NA-values and to discard some of the original columns (three in total). The next step involved shifting all SheetBreak = 1 observations four minutes back in time. This meant that when an observation was an error, the two observations that came prior to it had to be remarked as the error (y = 1), while the error observation had to be discarded. The step following was the feature engineering and selection process, which resulted in two different strategies with both a different number of independent variables (combination between the original as well as the newly engineered variables). The last step involved the resampling of the training data in order to balance the label distribution.

4. Methods

After the pre-processing of the data, the various strategies were evaluated on their effectiveness and compared accordingly. In order to prepare the data for training and testing, the data first had to be split into X and y variables. The X variables contain all features, and y represents the outcome variable. The data was split into X_train, X_test, y_train and y_test sets with the train_test_split function of the sklearn.model_selection library (sklearn.model_selection.train_test_split, 2020). The train-test split was 80/20 (or 16,258-4065 observations), with both sets containing an equal ratio of the binary y variable. The final step before the three classifiers could be trained and tuned was to scale the features. Scaling was accomplished with the StandardScaler function of the sklearn.preprocessing library. This function standardizes the features by removing the mean and scaling to unit variance (sklearn.preprocessing.StandardScaler, 2020).

4.1 Models

As this research aims at testing how the predictive performance of an AdaBoost and XGBoost machine learning algorithm on rare event data are affected after resampling training data in combination with adding additional engineered features, the pre-processed data is trained, tuned and tested on an AdaBoost and XGBoost classifier. Moreover, the RUSBoost classifier, an algorithm that automatically combines random under-sampling with an AdaBoost, is trained, tuned and tested as well.

Both the AdaBoost and XGBoost are ensemble models, which are basically combinations of series of low performing classifiers. By combining a series of weak models, a stronger model is created. Ensemble models usually have a higher accuracy rate than base classifiers have individually (Geiler, Hong, & Yue-Jian, 2010). In an ensemble model, all individual classifiers get to vote for the label to predict. The label that receives the majority of votes is returned as the final predicted label. An advantage of ensemble models is that base estimators can be parallelized by distributing them over

different machines. Also, a variety of different algorithms can be applied as base estimators for a single model, making it a meta-algorithm that predicts based on several machine learning methods. Finally, an ensemble method can introduce bias by applying a boosting tactic, which benefits the learning of weak learners. The RUSBoost is similar to the AdaBoost classifier, apart from that it automatically performs random under-sampling, and will, therefore, receive the same parameter settings.

Adaptive Boosting, or abbreviated as AdaBoost, was first introduced by Freund and Schapire (1997). This ensemble boosting method adjusts the weights of the base estimators according to their performance on uncommon observations. In each iteration, the model is trained on a subset of the training data, after which its performance is evaluated. Base estimators that predict well receive more weight and those that perform poorly are downsized in weight. Moreover, observations that were predicted wrongfully are increased in weight as well, increasing their probability. In accordance with the ensemble philosophy, the final vote is constructed by a majority voting. The process of refining the weight continues until a perfect fit is found, or until the user-specified iterations are met. According to Albon (2017), the most important hyperparameters to tune are the "base_estimator", "n_estimators" and the "learning_rate". For both the AdaBoost as well as the RUSBoost, a total of three base estimators will be tested, and include the default decision tree classifier, random forest classifier, and a support vector machine.

Decision Tree: the first base estimator is the default Decision Tree Classifier. Since this is the default base estimator, no additional background information will be provided here. The four parameters that are considered most important to tune include "max_depth", "min_samples_split", "min_samples_leaf", and "max_features" (Fraj, 2017).

Random Forest Classifier: the second base estimator is the Random Forest Classifier (RFC). The RFC was chosen as the second base estimator as research by Lin et al. (2017) proved the effectiveness of an RFC in imbalanced data analysis tasks. The RFC shares similarities with the Decision Tree classifier, but instead of a single tree, a forest of several trees that slightly differ from each other are fitted on the data. The trees are different from each other since a tree is fitted on data that is randomly selected, hence the name Random Forest. The final verdict of this classifier is an average (or majority vote) of all the trees. The RFC is less prone to overfitting (in comparison to a single tree classifier), yet utilizes the predictive power of the single trees combined. The most important parameters to tune include "n_estimators", "max_depth", "min_samples_split", "min_samples_leaf", and "max_features" (Fraj, 2017).

Support Vector Machine: the third base estimator is the Support Vector Machine (SVM). The SVM is proposed as a base estimator by Li et al. (2008) as it performed well on imbalanced classification

problems. The SVM projects the non-linear input data (training data) to a high-dimensional feature space by performing a kernel trick. Ultimately this operation enables a linear separation between the different classes by fitting an optimal hyperplane that separates the various classes (Vapnik, 2013). The parameter that C is a regularization parameter that regularizes the complexity of the model. After an SVM has been trained, unobserved data is classified by calculating the distance between the data point and the support vectors. The distance measuring process is performed by the Gaussian kernel, which is the exponent of the Euclidean distance and Gamma parameter (determining the width of the Gaussian kernel) squared product. According to Li et al. (2008), the Gamma parameter σ has the greatest effect on the performance of the SVM. A small σ value results in a large width of the Gaussian kernel, a large value enlarges the width. To conclude, the parameters to tune for the SVM base estimator are the C and σ parameter.

XGBoost, which is the abbreviation for Extreme Gradient Boosting, is a decision tree algorithm that implements gradient boosting and was introduced first in 2016 (Chen & Guestrin). The XGBoost algorithm mainly focusing on speed and performance by paralysing the training process over all the available CPUs. Similar to the AdaBoost and RUSBoost, the XGBoost is an ensemble technique. When the current model makes a mistake, an additional model (a decision tree) is added to the model in order to correct for the mistake made. This process continues until the model's performance does not improve anymore. The newly added models aim to minimize the loss by applying a gradient descent, hence the name Gradient Boosting. There is an abundant number of parameters to tune when implementing an XGBoost, the most significant include "learning_rate", "n_estimators" and "gamma" (Revert, 2018).

4.2 Parameter Tuning

In order to determine the parameter values for the proposed algorithms, a cross-validated (k=5) grid search was performed on the user-specified values for the various parameters. This grid search was performed via the GridSearchCV function from the Scikit-Learns library. See table 4.1 for an overview of all the algorithms and the corresponding parameters.

Table 4.1: overview of the various algorithms and the parameters to tune.

Model	**Parameters**	**Values**
AdaBoost	base_estimator	[DT, RFC, SVM]
	n_estimators	[10, 25, 50, 100, 200]
	learning_rate	[0.001, 0.01, 0.1, 0.5, 1]
RUSBoost	base_estimator	[DT, RFC, SVM]
	n_estimators	[10, 25, 50, 100, 200]

XGBoost	learning_rate	[0.001, 0.01, 0.1, 0.5, 1]
	learning_rate	[0.001, 0.01, 0.1, 0.5, 1]
	n_estimators	[10, 25, 50, 100, 200]
	gamma	[0, 1, 2, 3]

Note: DT is Decision Tree, RFC is Random Forest Classifier and SVM is Support Vector Machine.

As mentioned in chapter 4.1, the various base estimators have plenty of hyperparameters to tune as well. The tuning of hyperparameters is an important part of machine learning since the hyperparameter setup can significantly contribute to the model its performance (Bardenet, Brendel, Kégl, & Sebag, 2013). The hyperparameters that were tuned correspond to those that are in the description of the various base estimators. For each classifier – base estimator combination, a grid search was performed in order to discover the optimal parameter values. Since not every hyperparameter turned out to contribute to the performance of a certain classifier- base estimator combination, there is no dedicated table with all hyperparameter values used for optimizing the base estimators. The optimal hyperparameter values used can be found in the codebook, but do not include the various values that were experimented with. Moreover, the best performing classifier- base estimator combinations will be presented and discussed in chapter 4, including the set of hyperparameters values.

4.3 Evaluation & Benchmark

The performance of the proposed strategies was evaluated by benchmarking the largest F_1-score achieved in this research with the F_1-score of 0.114 that was achieved by Ranjan et al. (2018). in their research. The choice to use the F_1-score as a measure of performance was motivated by the assumption that this method provides the most reliable indication of performance in rare event classification tasks (Ranjan, Reddy, Mustonen, Paynabar, & Pourak, 2018). Since the data set is highly unbalanced, the accuracy score along would yield a false impression of a very high score (Chen Y. , 2009). The F_1-score is the harmonic mean of the averaged precision and recall, with scores ranging from 0 to 1 (on a scale from 0 to 1, 0 being a terrible F_1-score, and 1 is considered a perfect score). The calculation for the F_1-score is shown below.

$$\boldsymbol{F_1 - score} = \frac{2*(\text{Recall}*\text{Precision})}{\text{Recall}+\text{Precision}}$$

In addition to the F_1-score, the precision score of the best performing model (derived from the largest F_1-score) will be evaluated as well against a benchmark score. The precision score (which belongs to the F_1-score of 0.114) achieved by Ranjan et al. (2018) is 0.071 and will be used as the benchmark score. The comparison between the precision score that belongs to the largest F_1-score of this research

and the above-mentioned precision score is essential in order to provide a complete view of the effectiveness of the best performing strategy. The motivation for this decision is that a potential improvement in one measurer should not be at the expense of another measurer. Both the precision score and F_1-score were automatically generated via the "classification_report" function of the sklearn.metric library (Pedregosa, et al., 2011).

For each classifier-strategy combination, both the F_1-score and the corresponding precision score are collected. The final evaluation verdict is a statement regarding the effectiveness of the best performing classifier-strategy combination. This verdict is based on whether or not the best-achieved F_1-score exceeds the benchmark score, and on how this affects the precision score.

4.4 Software

The software used for the analysis of this research is Python version 3 in the Jupiter Notebook environment. In Table 4.2 an overview is given of the various libraries and functions used.

Table 4.2: overview of the various libraries and functions used in this research.

Library	**Specific**
Autofeat	AutoFeatClassifier
Imbalanced-learn	imblearn.ensemble.RUSBoostClassifier imblearn.under_sampling.RandomUnderSampler
NumPy	*
Pandas	*
Scikit-learn	sklearn.ensemble.AdaBoostClassifier sklearn.ensemble.RandomForestClassifier sklearn.svm.LinearSVC sklearn.preprocessing.StandardScaler sklearn.model_selection.train_test_split sklearn.model_selection.GridSearchCV sklearn.metrics
Time	*
Xgboost	XGBoost
Matplotlib	PyPlot

5. Results

In this section of the report, the results will be presented. Since two experimental set-ups were proposed, both results of each set-up will be covered individually in separate sections. Each set-up was tested on three ensemble classifiers, which are a RUSBoost, AdaBoost, and XGBoost. Additionally, the RUSBoost and AdaBoost were tested on three different base estimators, which are the default decision tree (DT), a random forest classifier (RFC), and lastly a support vector machine (SVM). Each experimental set-up generated seven F_1-scores and corresponding precision scores. The results in the tables represent the largest F_1-scores achieved with optimal hyperparameter settings.

5.1 Experimental Set-Up 1 Results

Experimental set-up 1 concerns a random under-sampling strategy combined with features engineered in one feature engineering step and five feature selection rounds. Table 5.1 displays the F_1-and precision scores that correspond to the various classifier-base estimator combinations. The scores represent the results obtained by the trained classifiers on unseen test data. The same test set was applied to all classifiers-base estimator combinations. Moreover, the values in table 5.1 represent the highest scores obtained by hyperparameter optimization for all classifier – base estimator combinations. This time-consuming process involved tuning the various parameters, training and testing the tuned model, and comparing the newly acquired scores with the scores generated by other parameter value combinations.

Table 5.1

	F_1-score			Precision		
Model	*RUSBoost*	*AdaBoost*	*XGBoost*	*RUSBoost*	*AdaBoost*	*XGBoost*
DT (default)	0.10	0.09	0.06	0.07	0.05	0.03
RFC	**0.21**	0.08	-	**0.16**	0.04	-
SVM	0.07	0.06	-	0.04	0.03	-

Note: XGBoost has no base estimator. The F_1-scores are font coloured black, and the precision scores are font coloured red. The bold underlined numbers represent the highest scores.

As can be observed from the results in table 5.1, the largest overall F_1-score of 0.21 is achieved by the RUSBoost classifier with an RFC-base estimator. The largest overall precision score of 0.16 is achieved by the same RUSBoost – RFC classifier combination. Both those scores exceed the benchmark thresholds of 0.114 and 0.071. The F_1-score improved by 84.2% ((0.21-0.114)/0.114), and precision improved by 125.4% ((0.16-0.071)/0.071). After a five-fold cross-validation grid search with different learning rates, the number of estimators, and various other hyperparameters, the optimal hyperparameter values for this particular combination were found to be the default settings for both

the RUSBooster as well as the RFC. A total of 18 out of the 55 rare events of an error were classifier correctly by the classifier combination. See figure 5.1 for the corresponding classification report.

Figure 5.1: Classification report of the RUSBoost – RFC classifier combination.

	Precision	Recall	**F_1-score**	Support
No error (0)	0.99	0.98	0.98	4010
Error (1)	0.16	0.33	0.21	55
Accuracy	*	*	0.97	4065
Macro avg	0.57	0.65	0.60	4065
Weighted avg	0.98	0.97	0.97	4065

The largest F_1-score achieved by the AdaBoost classifier is 0.09 and was realized with the default decision tree as for the base classifier. The score of 0.05 resembles the largest precision score achieved by the AdaBoost classifier, again with the decision tree as the base estimator. None of the AdaBoost – base estimator combinations exceeds the benchmark thresholds, including the best performing AdaBoost – DT combination. The hyperparameters used and tuned in this classifier - base estimator combination include “n_estimators” and “learning_rate” for the AdaBoost classifier, and “max_depth” and “min_samples_leaf” for the base estimator. The corresponding classification report is shown in figure 5.2.

Figure 5.2: Classification report of the AdaBoost – DT classifier combination.

	Precision	Recall	**F_1-score**	Support
No error (0)	1.00	0.73	0.84	4010
Error (1)	0.05	0.93	0.09	55
Accuracy	*	*	0.73	4065
Macro avg	0.52	0.83	0.46	4065
Weighted avg	0.99	0.73	0.83	4065

Note: base_estimator__max_depth (7), base_estimator__min_samples_leaf (1), learning_rate (1), n_estimators (200).

The F_1-and precision scores achieved by the XGBoost are 0.06 and 0.03 respectively. Both scores do not exceed the benchmark threshold values set. The parameters that were found to yield the best results are “learning_rate” (0.1) and “n_estimators” (200). From the 55 errors, the XGBoost classifier 48 correctly. See figure 5.3 for the classification report.

Figure 5.3: Classification report of the XGBoost.

	Precision	Recall	**F_1-score**	Support
No error (0)	1.00	0.65	0.79	4010
Error (1)	0.03	0.87	0.06	55
Accuracy	*	*	0.65	4065
Macro avg	0.52	0.76	0.43	4065
Weighted avg	0.98	0.65	0.78	4065

5.2 Experimental Set-Up 2 Results

Experimental set-up 2 applied two rounds of feature engineering, with each round having a single feature selection run. Similar to table 5.1, table 5.2 shows the best F_1-score results achieved by the different classifier-base estimator combinations on the test set of data.

Table 5.2

	F_1-score			Precision		
Model	*RUSBoost*	*AdaBoost*	*XGBoost*	*RUSBoost*	*AdaBoost*	*XGBoost*
DT (default)	0.15	0.09	0.07	0.17	0.05	0.04
RFC	**0.20**	0.08	-	**0.24**	0.04	-
SVM	0.04	0.05	-	0.02	0.03	-

Note: XGBoost has no base estimator. The F_1-scores are font coloured black, and the precision is font coloured red. The bold underlined numbers represent the highest scores.

The largest F_1-and precision scores for experimental set-up 2 were obtained by the RUSBoost - RFC classifier combination. Both the F_1-and precision score surpassed the benchmark threshold values, with the F_1-score improving by 75.4% ((0.20-0.114)/0.114) and the precision score resulting in an improvement of 238% ((0.24-0.071)/0.071). The three hyperparameters tuned include “n_estimators” (100) and “learning_rate” (1) for the classifier, and “min_samples_split” (2) for the base estimator. Out of the 55 observations of errors, a total of nine were classified correctly. See figure 5.4 for the classification report.

Moreover, the RUSBoost – DT classifier combination exceeded the benchmark threshold values too. The F_1-score of 0.15 improved by 31.6% ((0.15-0.114)/0.114), and precision score experienced an improvement of 139.4% ((0.17-0.071)/0.071). The hyperparameters selected and modified for the classifier are the “learning_rate” (0.1) and “n_estimators” (100). The base estimator was tuned only on the “min_samples_leaf” (3) hyperparameter in order to achieve the best result.

Figure 5.4: Classification report of the RUSBoost – RFC classifier combination.

	Precision	Recall	**F_1-score**	Support
No error (0)	0.99	0.99	0.99	4010
Error (1)	0.24	0.16	0.20	55
Accuracy	*	*	0.98	4065
Macro avg	0.62	0.58	0.59	4065
Weighted avg	0.98	0.98	0.98	4065

The largest F_1-and precision scores achieved by the AdaBoost classifier is in combination with the DT, 0.09 and 0.05. Nevertheless, both results do not surpass the benchmark threshold values. The remaining classifier – base estimator combinations score below the benchmark threshold too. The hyperparameters values for the AdaBoost – DT classifier combination are "n_estimators" (150), "learning_rate" (0.1), and the base estimator hyperparameter "min_samples_leaf" (2). Figure 5.5 provides an overview of the classification report.

Figure 5.5: Classification report of the AdaBoost – DT classifier combination.

	Precision	Recall	**F_1-score**	Support
No error (0)	1.00	0.74	0.85	4010
Error (1)	0.05	0.89	0.09	55
Accuracy	*	*	0.74	4065
Macro avg	0.52	0.82	0.47	4065
Weighted avg	0.99	0.74	0.84	4065

The final model tested concerns the XGBoost. The F_1-score realized by this classifier is 0.07, which is below the benchmark threshold value. Likewise, the precision score of 0.04 does not exceed the benchmark threshold set. The hyperparameters tuned for this classifier include "n_estimators" (25), "learning_rate" (0.5), "gamma" (0) and "max_depth" (8). From the 55 observations that represent an error, the XGBoost classifier 46 correctly. Please see figure 5.6 for the classification report.

Figure 5.6: Classification report of the XGBoost classifier.

	Precision	Recall	**F_1-score**	Support
No error (0)	1.00	0.72	0.84	4010
Error (1)	0.04	0.84	0.07	55
Accuracy	*	*	0.72	4065
Macro avg	0.52	0.78	0.45	4065
Weighted avg	0.98	0.72	0.82	4065

Table 5.7 shows an overall overview of the results achieved by each strategy on the different classifier – base estimator combinations. The black coloured font signifies the F_1-score, and the red coloured font indicates the precision score.

Table 5.7	**Experimental set-up 1**			**Experimental set-up 2**		
Model	*RUSBoost*	*AdaBoost*	*XGBoost*	*RUSBoost*	*AdaBoost*	*XGBoost*
DT (default)	0.10/0.07	0.09/0.05	0.06/0.03	**0.15/0.17**	0.09/0.05	0.07/0.04
RFC	**0.21/0.16**	0.08/0.04	-	**0.20/0.24**	0.08/0.04	-
SVM	0.07/0.04	0.06/0.03	-	0.04/0.02	0.05/0.02	-

6. Discussion

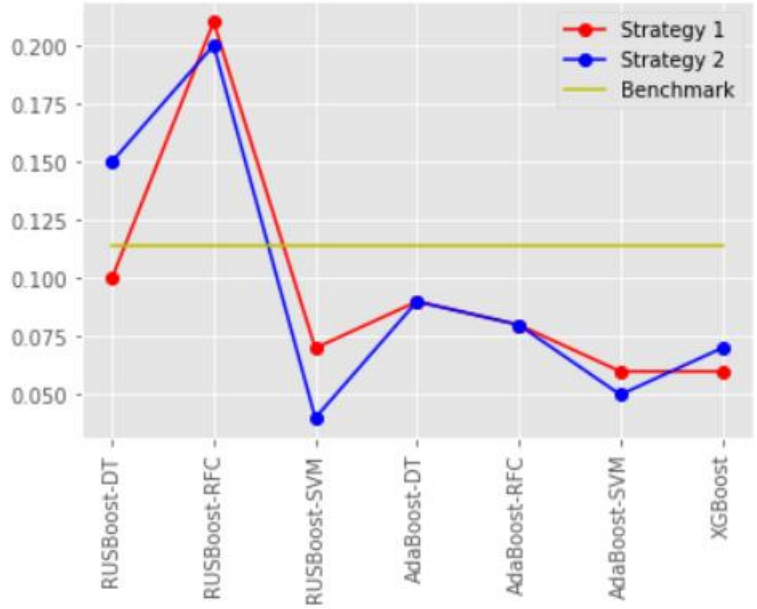
In this section, the results of this research will be evaluated in relation to the research question. The aim of this project was to find whether or not a resampling strategy (random under-sampling) in combination with adding additionally engineered features to an unbalanced dataset could improve on the predictive performance of an AdaBoost and XGBoost ensemble classifier. Improvement of the predictive performance was measured by benchmarking the F_1-score of 0.114 achieved by Ranjan et al. (2018) to the F_1-score results of this research. Two experimental set-ups were proposed and tested on seven different classifier – base estimator combinations. See table 5.7 in the previous chapter for a complete overview of the results per strategy on the different classifier – base estimator combinations.

The results in table 5.7 show that three out of the 14 classifier – base estimator combinations have exceeded the benchmark threshold value. Experimental set-up 2 appears to be the superior of the two proposed strategies as two out of the three best results were generated after applying experimental set-up 2. Nevertheless, the overall best F_1-score of 0.21 was realized by applying experimental set-up 1. None of the proposed strategies however managed to exceed the benchmark threshold value with the AdaBoost or XGBoost classifier, but only if applied to the RUSBoost classifier. It is noteworthy to find that, even though the RUSBoost is an AdaBoost that automatically performs random under-sampling, the performance of the RUSBoost is significantly better. Although this outcome might be contradictory, it is in accordance with the findings of the research by Seiffert et al. (2010) who found in their experiment that the RUSBoost outperformed four base learners and a SMOTEBoost on 15 test datasets. Those test datasets were highly imbalanced, similar to the dataset used for this research. It should, therefore, be no surprise that the RUSBoost indeed performs better than the AdaBoost and XGBoost on this particular dataset as it is in accordance with earlier findings by Seiffert et al. (2010).

A remarkable discovery is the similarities between scores achieved by both strategies for the AdaBoost – base estimator combinations and XGBoost. Please see figure 6.1. It seems from the results that the

effect of the proposed strategies differ little from each other as the results are almost similar. There are some marginal differences between the results achieved per strategy on the AdaBoost – SVM combination and XGBoost, but those are negligible. From figure 6.1 it can be observed that there is a clear trend regarding the performance of a classifier in combination with an SVM as the base estimator. For both strategies goes that when an SVM is applied as the base estimator, the performance goes down. A similar outcome was observed by Qazi and Raza (2012), where the support vector machine performed worst as compared to the other classifiers tested. Moreover, the expected outcomes in accordance with the observations by Qazi and Raza (2012) remained absent. The proposed strategies did not yield the desired outcomes if applied to the AdaBoost or XGBoost.

Figure 6.1: line graph that plots the F_1-scores per strategy on different classifiers.



The results for the performance of the AdaBoost and XGBoost are somewhat surprising. Although the recommendations made by Ranjan et al. (2018) were implemented, the F_1-scores did not improve. On the contrary, the results deteriorated. There are three possible explanations for this outcome. The first explanation is that by applying a random under-sampling technique, valuable information got lost. This information loss could have contributed to the decrease in the ability of the classifiers utilized to discriminate between both classes. Abdi et al. (2015) warned for this potential consequence when applying a random under-sampling technique in their research. The second possible explanation for the decrease in performance might be derived from the fact that the classifiers used are ensemble methods purposed to handle well on imbalanced data classification and regression tasks. This research implemented a random under-sampling technique in order to create a better balanced dataset. It might be possible that this newly created balance is the cause of the poor performance of the classifiers

as those were designed to deal with unbalanced datasets. The last possible explanation is the quality of the engineered features. As stated by Horn et al. (2019), one feature engineering round will probably only yield marginal improvements, and three feature engineering rounds will likely overfit on the data, hence two feature engineering rounds. Nevertheless, since the limited time available for this research, experimental set-up 2 only received one feature selection run (five feature selection runs are default). Perhaps this might have contributed to the poor performance of the AdaBoost and XGBoost as well.

Although the F_1-scores did not improve, the results contribute to the knowledge on how to handle unbalanced datasets. It is obvious from the results of this research that adding additional engineered features to an unbalanced dataset in combination with a random under-sampling technique does not improve on the predictive performance of an AdaBoost or XGBoost. It is important to acknowledge this discovery so that other researchers who work with datasets that have an imbalanced label distribution apply other techniques apart from the one proposed in this research. This research has also provided additional evidence regarding the performance of the RUSBoost classifier over other classifiers. In addition to the research by Seiffert et al. (2010), this research has proven on yet another unbalanced dataset the effectiveness of the RUSBoost classifier. This might be of interest to researchers who seek to compare the performance of various classifiers on unbalanced datasets. To the pulp-to-paper manufacturer who seeks to reduce downtime, it is recommended to implement the experimental set-up 1 in combination with a RUSBoost – RFC base estimator classifier as this yielded a vast improvement in the prediction of errors.

7. Conclusion

This research intended at improving the predictive performance of an AdaBoost and XGBoost classifier on an unbalanced dataset from a pulp-to-paper producer. Two experimental set-ups were proposed to increase on the benchmark F_1-score of 0.114 and involved adding additionally engineered features to the dataset while resampling the data by means of random under-sampling.

It was found that no improvement in the predictive performance of the originally proposed classifiers was made by applying the different experimental set-ups. One can, therefore, state that random under-sampling in combination with feature engineering is not a method to improve on imbalanced data classification tasks. On the other hand, if the RUSBoost classifier is taken into consideration, then a significant improvement on the benchmark F_1-score has been made. Figure 6.1 in the previous chapter shows the similarity between the F_1-scores achieved by both experimental set-ups as they are almost parallel to one each other. Nevertheless, since experimental set-up 1 yielded the overall best F_1-score of 0.21, and did require significantly less time to engineer and select features (30 minutes for experimental set-up 1 compared to almost 26 hours for experimental set-up 2 in feature engineering and selection), it should be considered the superior of the two, although experimental set-up 2 generated two above-benchmark scores. Nevertheless, since Ranjan et al. (2018) did not include the RUSBoost classifier in their research, an objective comparison between F_1-scores cannot be made, and so it is unclear whether the F_1-score of this classifier is as a result of the proposed experimental set-ups, or the RUSBoost classifier algorithm itself.

For future research, it is recommended to test the two proposed strategies on the RUSBoost and AdaBoost with different base estimators. Due to time restrictions, this research limited the implemented base estimators to a decision tree, random forest classifier, and support vector machine. Improvement in the predictive performance of the classifier could potentially be made by testing on other base estimators. Moreover, the feature engineering via the AutoFeat library using two rounds of feature engineering steps combined with five feature selection runs (similar to experimental set-up 2, except than with more feature selection runs) could potentially enhance the predictive performance of the classifiers as well.

References

Abdi, L., & Hashemi, S. (2015). To Combat Multi-Class Imbalanced Problems by Means of Over-Sampling Techniques. *IEEE Transactions on Knowledge and Data Engineering*, 238-251.

Albon, C. (2017, 12 20). *Adaboost Classifier*. Retrieved from www.chrisalbon.com: https://chrisalbon.com/machine_learning/trees_and_forests/adaboost_classifier/

API Reference. (n.d.). Retrieved from www.pandas.pydata.org: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

Bahnsen, A. C., Aouada, D., Stojanovic, A., & Ottersten, B. (2016). Feature engineering strategies for credit card fraud detection. *Expert Systems with Applications, Vol. 1*, 134-142.

Bardenet, R., Brendel, M., Kégl, B., & Sebag, M. (2013). Collaborative hyperparameter tuning. *International Conference on Machine Learning.* Atlanta: JMLR.

Bierman, C. J. (1993). *Essentials of pulping and papermaking.* San Diego: Academic Press.

Bolton, R. H. (2002). Statistical Fraud Detection: A Review. *Statistical Science*, 235-249.

Brownlee, J. (2014, 09 26). *Discover Feature Engineering, How to Engineer Features and How to Get Good at It*. Retrieved from machinelearningmastery: https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 1-58.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785-794). San Francisco : Association for Computing Machinery, New York, NY, United States.

Chen, Y. (2009). *Learning Classifiers from Imbalanced, Only Positive and Unlabeled Data Sets.* Ames: Department of Computer Science - Iowa State University.

Drummond, C., & Holte, R. C. (2003). *C4.5, Class Imbalance, and Cost Sensitivity: Why Under-Sampling beats Over-Sampling.*

Edgeworth, F. Y. (1887). XLI. On discordant observations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, Series 5, Vol 23*, 364-375.

Fitchett, D., & Sondalini, M. (2017). *True Down Time Cost Analysis - 2nd Edition.*

Fraj, M. B. (2017, 12 21). *In Depth: Parameter tuning for Random Forest*. Retrieved from www.medium.com: https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb7e920d

Fraj, M. B. (2017, 12 20). *InDepth: Parameter tuning for Decision Tree*. Retrieved from www.medium.com: https://medium.com/@mohtedibf/indepth-parameter-tuning-for-decision-tree-6753118a03c3

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 119-139.

Ganganwar, V. (2012). An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering*, 42-47.

Geiler, O. J., Hong, L., & Yue-Jian, G. (2010). An Adaptive Sampling Ensemble Classifier for Learning from Imbalanced Data Sets. *The International MultiConference of Engineers and Computer Scientists.* Hong Kong: IMECS.

Guo, H., Shi, W., & Deng, Y. (2006). Evaluating Sensor Reliability in Classification Problems Based on Evidence Theory. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 970-981.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction.* New-York City: Springer.

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on knowledge and data*, 1263-1284.

Henke, N., Bughin, J., Chui, M., Manyika, J., Saleh, T., Wiseman, B., & Sethupathy, G. (2016). *The age of analytics: competing in a data-driven world.* McKinsey & Company.

Horn, F. (2020, 04 07). Question regarding AutoFeatClassifier. (A. L., Interviewer)

Horn, F., Pack, R., & Rieger, M. (2019). The autofeat Python Library for Automated Feature Engineering and Selection. *arXiv preprint arXiv:1901.07329*.

imblearn.under_sampling.RandomUnderSampler. (n.d.). Retrieved from www.imbalanced-learn.readthedocs.io: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under_sampling.RandomUnderSampler.html

Kadlec P., G. B. (2009). Data-driven Soft Sensors in the process industry. *Computers & Chemical Engineering*, 795-814.

Karim, F., Majumdar, S., Darabi, H., & Chen, S. (2017). Lstm fully convolutional networks for time series classification. *IEEE Access*, 1-7.

Karim, F., Majumdar, S., Darabi, H., & Harford, S. (2019). Multivariate LSTM-FCNs for time series classification. *Neural Networks*, 237-245.

King, G., & Zeng, L. (2001). Logistic Regression in Rare Events Data. *Oxford Journals, Volume 9, Issue 2*, 137-163.

Krawczyk, B. (2016). Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 221–232.

Li X., W. L. (2008). AdaBoost with SVM-based component classifiers. *Engineering Applications of Artificial Intelligence, Vol 21, Issue 5*, 785-795.

Li, H., Liu, Y., Zhang, X., An, Z., Wang, J., Chen, Y., & Tong, J. (2017). Do we really need more training data for object localization. *IEEE International Conference on Image Processing (ICIP)* (pp. 775-779). Beijing: IEEE.

Lin, W., Wu, Z., Lin, L., Wen, A., & Li, J. (2017). An Ensemble Random Forest Algorithm for Insurance Big Data Analysis. *IEEE Access, vol. 5*, 16568-16575.

Liu, X. Y., Wu, J., & Zhou, Z. H. (2009). Exploratory Undersampling for Class-Imbalance Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 539-550.

Maclin, R., & Opitz, D. (1997). An Empirical Evaluation of Bagging and Boosting. *The Fourteenth National Conference on Artificial Intelligence.* Rhode Island: AAAI Press.

Mair, C., Kadoda, G., Lefley, M., Phalp, K., Schofield, C., Shepperd, M., & Webster, S. (2000). An investigation of machine learning based prediction systems. *Journal of Systems and Software, Vol. 53, Issue 1*, 23-29.

Marr, B. (2018, 5 21). *How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read*. Retrieved from www.forbes.com: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#35179f0c60ba

Massis, B. (2012). Using predictive analytics in the library. *New Library World, Vol 133 No. 9/10*, 491-494.

Nargesian, F., Samulowitz, H., Khurana, U., Khalil, E. B., & Turaga, D. (2017). Learning Feature Engineering for Classification. *International Joint Conference on Artificial Intelligence*, (pp. 2529-2535).

Neethu, C. M., & Abraham, A. (2019). Customer Segmentation From Massive Customer Transaction Data. *International Research Journal of Engineering and Technology (IRJET)*, 432-436.

Park, G. U., & Jung, I. (2019). Comparison of resampling methods for dealing with imbalanced data in binary classification problem. *The Korean Journal of Applied Statistics*, 349-374.

Peck, R. B. (1969). Advantages and limitations of the observational method in applied soil mechanics. *Geotechnique*, 171-187.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research, vol 12*, 2825-2830. Retrieved from www.scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

Polikar, R. (2012). *Ensemble Machine Learning.* Boston: Springer.

Qazi, N., & Raza, K. (2012). Effect of Feature Selection, SMOTE and under Sampling on Class Imbalance Classification. *2012 UKSim 14th International Conference on Computer Modelling and Simulation* (pp. 145-150). Cambridge: IEEE.

Rackwitz, R., & Fiessler, B. (1978). Structural reliability under combined random load sequences. *Computers & Structures, Vol 9, Issue 5*, 489-494.

Ranjan, C. (2020). Early Prediction. In R. C., *Rare Event Prediction with Deep Learning* (p. 21).

Ranjan, C. (2020). Early Prediction. In R. C., *Rare Event Prediction with Deep Learning* (p. 21).

Ranjan, C., Reddy, M., Mustonen, M., Paynabar, K., & Pourak, K. (2018). *Dataset: Rare Event Classification in Multivariate Time Series.*

Revert, F. (2018, 08 10). *Fine-tuning XGBoost in Python like a boss*. Retrieved from www.towardsdatascience.com: https://towardsdatascience.com/fine-tuning-xgboost-in-python-like-a-boss-b4543ed8b1e

Santoro, M., Coi, A., Lipucci Di Paola, M., Bianucci, A., Gainotti, S., Mollo, E., . . . Bianchi, F. (2015). Rare Disease Registries Classification and Characterization: A Data Mining Approach. *Public Health Genomics*, 113-122.

Santos, M. S., Soares, J. P., Abreu, P. H., Helder, A. J., & Santos, J. A. (2018). Cross-Validation for Imbalanced Datasets: Avoiding Overoptimistic and Overfitting Approaches. *IEEE Computational Intelligence Magazine 13*, 59-76.

Sasisekharan, R., Seshadri, V., & Weiss, S. M. (1996). Data mining and forecasting in large-scale telecommunication networks. *IEEE Expert, vol. 11, no. 1*, 37-43.

Seiffert, C., Khoshgoftaar, T. M., van Hulse, J., & Napolitano, A. (2010). RUSBoost: A Hybrid Approach to Alleviating Class Imbalance. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART A: SYSTEMS AND HUMANS*, 185-197.

sklearn.model_selection.train_test_split. (2020, 04 10). Retrieved from www.scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

sklearn.preprocessing.StandardScaler. (2020, 04 10). Retrieved from www.scikit-learn.org/: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

van der Putten, P., & van Someren, M. (2004). A Bias-Variance Analysis of a Real World Learning Problem: The CoIL Challenge 2000. *Springer Machine Learning*, 177–195.

Vapnik, V. (2013). *Statistics for Engineering and Information Science: The nature of statistical learning theory.* New-York: Springer.

Wah, Y. B., Rahman, H. A., He, H., & Bulgiba, A. (2016). Handling imbalanced dataset using SVM and k-NN approach. *AIP Conference Proceedings 1750.* AIP Publishing.

Wasikowski, M., & Chen, X. W. (2010). Combating the Small Sample Class Imbalance Problem Using Feature Selection. *IEEE Transactions on Knowledge and Data Engineering*, 1388-1400.

Wind, D. K. (2014). *Concepts in Predictive Machine Learning.*

Zhang, L., Tan, J., Han, D., & Zhu, H. (2017). From machine learning to deep learning: progress in machine intelligence for rational drug discovery. *Drug Discovery Today, Vol. 22, Issue 11*, 1680-1685.

Zheng, Z., Wu, X., & Srihari, R. (2004). Feature Selection for Text Categorization on Imbalanced Data. *ACM SIGKDD Explorations Newsletter, Vol. 6*, 80-89.

Zhu, X., Vondrick, C., Fowlkes, C. C., & Ramanan, D. (2016). Do We Need More Training Data? *International Journal of Computer Vision volume 119*, 76-92.

Zong, W., Huang, G., & Chen, Y. (2013). Weighted extreme learning machine for imbalance learning. *Neurocomputing*, 229-242.

Appendices

Appendix I: all original variables in the dataset.

"1"	"DateTime"	Timestamp
"2"	"SheetBreak"	Y-variable
"3"	"RSashScanAvg"	
"4"	"CT.1.BLADE.PSI"	
"5"	"P4.CT.2.BLADE.PSI"	
"6"	"Bleached.GWD.Flow"	
"7"	"ShwerTemp"	
"8"	"BlndStckFloTPD"	
"9"	"C1.BW.SPREAD.CD"	
"10"	"RS.BW.SPREAD.CD"	
"11"	"C1.BW.SPREAD.MD"	
"12"	"RS.BW.SPREAD.MD"	
"13"	"C1.BW.SCAN.AVG"	
"14"	"RS.BW.SCAN.AVG"	
"15"	"CoatBrkFlo"	
"16"	"Clay.Flow"	
"17"	"CouchLoVac"	
"18"	"COUCH.VAC"	
"19"	"X4PrsTopLd"	
"20"	"X4PrsBotLod"	
"21"	"CalndrDrw"	
"22"	"X2DryrDrw"	
"23"	"X3DryrDrw"	
"24"	"X4DryrDraw"	
"25"	"X1PrsTopDrw"	
"26"	"X4PrsBotDrw"	
"27"	"FanPmpSpd"	
"28"	"FlBxHdrVac"	
"29"	"FlatBxVac"	
"30"	"Grade.Bwt"	
"31"	"UnblGWDFlo"	
"32"	"Hdbox.pH"	
"33"	"HdBxLiqLvl"	
"34"	"TotHead."	
"35"	"HorzSlcPos"	
"36"	"KraftFlow"	
"37"	"CouchLoad"	
"38"	"C1MoSprdCD"	
"39"	"RSMoSprdCD"	
"40"	"C1MoSprdMD"	
"41"	"RSMoSprdMD"	
"42"	"RL.MoisAct"	
"43"	"PrScrRjFlo"	
"44"	"RwBrkFlo"	
"45"	"RcycFbrFlo"	
"46"	"RetnAidFlo"	
"47"	"RUSH.DRAG"	
"48"	"Rush.Drag"	
"49"	"SilicaFlo"	
"50"	"HBxSlcTemp"	
"51"	"SodAlumFlo"	
"52"	"CouchSpd"	
"53"	"MachSpd"	
"54"	"X1PrsTopSpd"	
"55"	"X4PrsBotSpd"	
"56"	"WtNStarFlo"	

"57"	"BasWgtFlo"
"58"	"TMP.Flow"
"59"	"HBxTotHead"
"60"	"TrayCons"
"61"	"UpprHdTmpRL"
"62"	"VertSlcPos"
"63"	"EventPress"

Variables coloured in blue represent sensor data of machine settings. The yellow coloured variables are measures of input quantities various ingredients required for the production of the paper (Bierman, 1993).

Appendix II: Table with all the engineered features that were added to the data frame with feateng_steps=1, and featsel_runs=5.
'Abs(CouchLoVac)'
'BlndStckFloTPD**2'
'HdboxpH**2'
'CoatBrkFlo**3'
'exp(COUCHVAC)'
'HBxSlcTemp**2'
'x1PrsTopSpd**2'
'x1PrsTopSpd**3'
'Abs(x1PrsTopDrw)'
'Abs(RSBWSPREADMD)'
'Abs(P4CT2BLADEPSI)'
'1/CouchLoad'
'RwBrkFlo**3'
'CouchSpd**3'
'SilicaFlo**3'
'BasWgtFlo**3'
'RSMoSprdMD**2'
'Abs(RwBrkFlo)'
'Abs(KraftFlow)'
'Abs(SilicaFlo)'
'exp(HBxTotHead)'
'RSBWSPREADCD**2'
'exp(FlBxHdrVac)'
'RSBWSPREADMD**2'
'Abs(SodAlumFlo)'
'Abs(x4PrsBotLod)'
'Abs(HdboxpH)'
'1/HBxTotHead'
'CouchLoad**2'
'Abs(CouchSpd)'
'FlBxHdrVac**2'
'UnblGWDFlo**3'
'x1PrsTopDrw**3'
'Abs(RSMoSprdMD)'
'exp(HBxSlcTemp)'
'exp(C1BWSCANAVG)'
'BleachedGWDFlow**3'
'x2DryrDrw**2'
'ShwerTemp**3'
'WtNStarFlo**2'
'x4DryrDraw**3'
'Abs(HBxTotHead)'
'C1BWSPREADCD**3'
'Abs(RSBWSPREADCD)'
'BlndStckFloTPD**3'