Expertise
and insight
for the future

Mariia Ustilkina

Predictive modelling for low-quality data

Metropolia University of Applied Sciences

Bachelor of Engineering

Software Engineering

Bachelor's Thesis

7 March 2020

Author	Mariia Ustilkina
Title	Predictive modelling for low-quality data
Number of Pages	41 pages + 1 appendix
Date	7 March 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen, Head of Department

The major objective of this paper is to examine possible solutions that can be potentially beneficial in dealing with noise and insufficiency of data in the domain of predictive modelling, and to perform regression analysis for the project dataset using these findings in order to improve performance.

The project is implemented in Python with an aid of Scikit-Learn machine learning library. Various outlier detection techniques are reviewed and tested in the project, as well as dimensionality reduction and oversampling with SMOTE adapted for regression problems. Prediction analysis is performed with ensemble models, particularly Random Forest and Gradient Boosting Machine.

Both models demonstrate good results in the conditions of data noise and scarcity, avoiding overfitting, however performance of these models does not differ significantly from each other. Outlier detection techniques, especially Local Outlier Factor and Elliptic Envelope, as well as SMOTE oversampling for margin values are proven to be beneficial for the chosen task.

Keywords	Machine learning, predictive modelling, outlier detection, SMOTE

Contents

List of Abbreviations

ANOVA	Analysis of Variance
CART	Classification and Regression Trees
GBM	Gradient Boosting Machine
IQR	Interquartile range
LOF	Local Outlier Factor
LOOCV	Leave One Out Cross Validation
MSE	Mean Square Error
PCA	Principle Component Analysis
QLF	Quality Loss Function
RF	Random Forest
RFE	Recursive Feature Elimination
SVD	Singular Value Decomposition
SMOTE	Synthetic Minority Over-Sampling Technique
SMOTER	Synthetic Minority Over-Sampling Technique for Regression

1 Introduction

Data scarcity and noise are issues of high importance in predictive analytics, as data is the core element in the domain of machine learning. Unfortunately, collecting information inside single organisation can be problematic for various reasons, leading to insufficiency of data compared to open source pools. Although in certain cases this problem is solvable by transfer learning utilizing open source data, some tasks are highly specific to the particular company and do not match available datasets.

The project used as a base for this paper aims to predict hour rates of employees in a consultancy company based on their profile and project specifications. It is important to point out that the case is naturally prone to noise in the data, due to a human factor and specificity of sales processes. Moreover, it is impossible to obtain a dataset of sufficient size for the reason that it is limited by the number of employees and sales in the last years. The data used in the project will be edited and obfuscated due to privacy reasons.

Currently there is an abundance of data pre-processing methods aiming to solve different problems as well as a high number of various prediction models. However, depending on the data features and complications, efficiency and suitability of the same methods can vary. The major objective of this paper is to examine possible solutions that are known to be specifically useful with limited and noisy datasets and perform prediction analysis using these findings.

2 Theoretical Background

2.1 Potential problems caused by poor quality of data

A well-known quality engineering expert Dr. Genichi Taguchi defined the interconnection between poor quality and overall loss. Dr. Taguchi introduced a quality loss function (QLF), measuring the loss which correlates with poor quality characteristics. According to QLF, loss is associated with quality parameter deviating from the target. Taguchi goes further and describes it as a loss to the society, which does not go unnoticed. In fact, unsatisfactory levels of quality increase the risk of making wrong decisions, potentially leading to negative effects such as company failures, bankruptcies and system shut-downs. [5] It becomes clear that quality, in particular quality of data is crucial in the modern world and the goal to ensure good quality becomes imperative. Ideally, establishing a data management function responsible for high quality of data is advisable. However, data quality provision is out of the scope of this work, as it highly depends on management levels of a company, hence the focus will be maintained on possible solutions for already available data.

Before describing possible solutions, problems brought by low quality and scarcity of data in the domain of predictive analytics should be discussed. Clearly, lack of data is likely to affect a model performance, moreover insufficiency is highly linked to the data being underrepresentative, which leads to poor performance of the model due to the fact that it does not generalize on the new data well. Furthermore, it is commonly argued that the smaller the dataset, the more significant is the effect of noise in the data. However, even for large datasets noise and outliers can decrease the performance significantly if not taken care of, therefore data cleaning is a crucial step in predictive analytics. In fact, a survey conveyed by data scientists revealed that over 80% of project time is spent on data collecting, cleaning and preparation [4].

In the next sections, possible solutions for different stages of predictive modelling in the context of low-quality data will be examined, including data preparation, model selection and tuning, and finally performance measurement.

2.2 Data manipulation

First of all, to use the data efficiently preparation is highly important. The common steps to consider are data cleaning, normalization and categorical data encoding. This is a general approach, so it will not be examined here, instead the methods specific to the problem of data quality and scarcity will be considered.

However, one of the data cleaning aspects worth mentioning separately is outlier detection. Although outliers are not precisely the noise but instead individual data points falling off the expected data behavior, they can affect the performance of certain prediction models significantly. Thus, it is important to attend to this issue if possible.

2.2.1 Outlier detection

Outlier detection can be univariate and multivariate. Univariate outlier detection implies that variables are examined separately. Basic statistic methods suffice for univariate outlier detection, either by visualizing variables on a box plot and dropping values out of the bound accordingly, or by calculating outliers with value smaller than 25^{th} quartile – 1.5*IQR or higher than the 75^{th} quartile + 1.5*IQR (where IQR is Interquartile Range, 75^{th} quartile – 25^{th} quartile) [1]. However, datasets usually contain more than one feature, hence multivariate outlier detection will be the main focus on this section.

Multivariate outlier detection algorithms vary significantly, depending on the type of data and presence of the labels. Assuming that all the features in the dataset form Gaussian distribution, *Elliptic Envelope* fitting can be considered a suitable solution. To provide a basic description, algorithm attempts to define an ellipse that includes majority of the data by assessing the distance of each sample with regards to the total mean [1]. Elliptic Envelope technique is simple and effective; however, the disadvantage of the method is a strong assumption on the normal distribution of the dataset, which limits its usage to highly specific cases.

In contrast, *Isolation Forest* does not require normal distribution of the data. Isolation Forest is essentially a random forest, however single tree in this forest represents a split on a random feature, and a random value from the chosen feature range is chosen for

each split. Scheme represented in Appendix 1 illustrates the mechanism of isolation forest clearly.

Isolation Forest operates by calculating a number of steps required to isolate a single data point. In the scheme, this measure is called path length. Naturally, as normal data points tend to be located closer to each other, it takes more splits to isolate each of them. Outliers, on the contrary, have a shorter path length, thus creating a measure to assess anomaly of data points. This method is known to be efficient on high-dimensional datasets, however on with the low number of features it might be not the most suitable algorithm. [15.]

Similar to isolation forests, density-based outlier detectors perform well with non-Gaussian distributions. One of the common examples of density-based algorithms is Local Outlier Factor (LOF). LOF labels outliers based on the local density of a sample point (concentration of other points in the immediate area around sample point). Sample is considered outlier if its local density is significantly lower than density of closest n neighbors. One of the advantages of LOF is the fact that it can detect outliers around clusters with different densities, because only local neighbors are taken into account, not the whole dataset, therefore distance measure is not fixed. [15.]

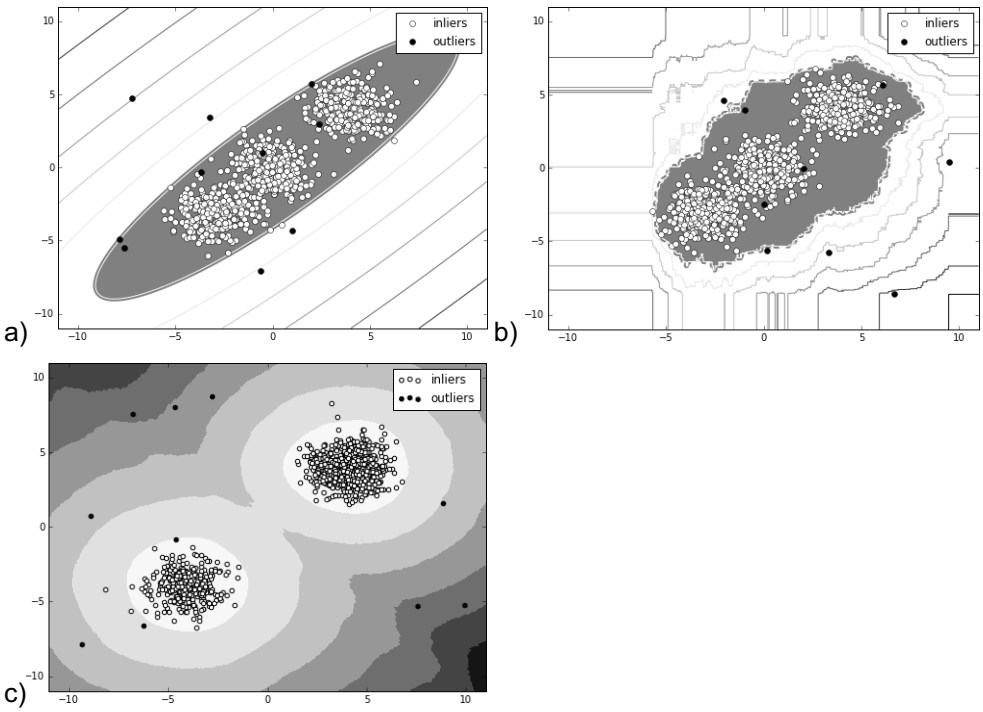

Figure 1. Outlier detection performance with a) Elliptic Envelope; b) Isolation Forest; c) LOF

It can be observed from figure 1 that LOF demonstrates the highest performance identifying outliers in clustered data while isolation forest and elliptic envelope tend to unify several clusters in a single inlier area, which leads to higher number of errors.

To sum up, it is important to analyze data types and distributions and prepare data before performing outlier detection to achieve best results.

The rest of this section will mostly focus on manipulating with a size of our dataset, and it can be divided into two categories: operating with a sample size or with a feature space. Feature manipulation will be examined first.

2.2.2 Feature selection

It is important to point out, not all the datasets have perfectly suitable features initially. In fact, if features appear to be non-linear, it is rather difficult for a classifier to capture the signal, which may lead both to overfitting and underfitting, depending on the strategy. Therefore, it can be useful to increase the dataset by adding features derived from original ones [1]. Such an increase is achievable by using *Polynomial Features* which returns a matrix of polynomial combinations of the features with a specified degree. The goal of the manipulation is not to increase the dimensions of dataset for the sake of a size, but rather to discover new features which would make a better contribution in model performance.

Another case with feature space is a comparatively high number of features in relation to the overall dataset size. This leads to the so-called curse of dimensionality: as the feature space increases, data becomes sparse and training the model becomes both critically slow and complex [3]. It is particularly important in algorithms such as K-Nearest Neighbors, the logic of which relies on the distance between data points in space. Apart from that insignificant features can simply contribute noise to the model, affecting the quality of a prediction. Moreover, high correlation between these features can become another obstacle. Fortunately, there are several methods to tackle this problem.

One of the approaches is feature selection and it can be divided into statistical-based type and model-based type. Statistical-based feature selection depends on statistical

tests without regards to a chosen model, while model-based approach automates the process of model training with different sets of features to get the best result.

A common technique for model-based feature selection is Recursive Feature Elimination. It acts by recursively running a model, measuring the accuracy and removing least useful features until the required number of features left or highest accuracy achieved. Figure 2 illustrates the procedure of RFE with Random Forest model.

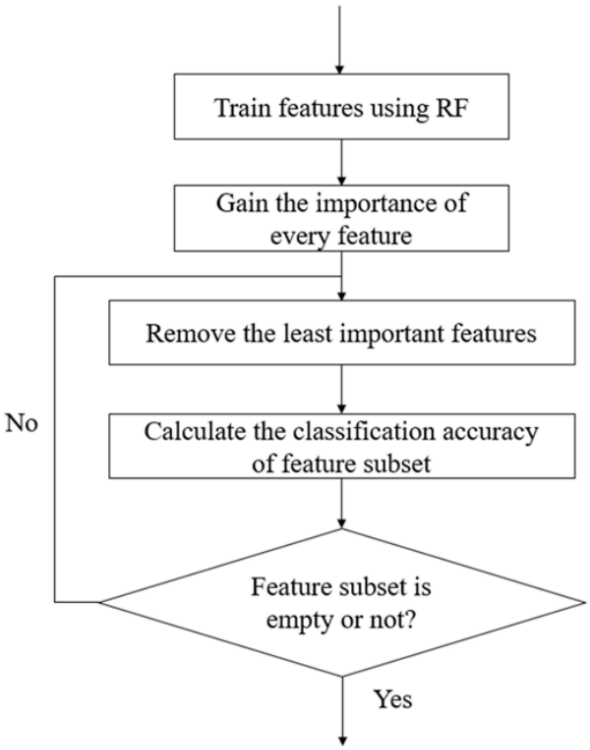

Figure 2. Recursive Feature Elimination algorithm scheme [13]

From the figure 2 one can conclude that process continues until all features are eliminated, and then the most successful set is chosen. However, implementations of the algorithm vary and while in some cases highest accuracy can be chosen as a stopping mechanism for RFE, there are situations when implementation expects a desired number of features as an input [13]. Without knowing what number would be the most beneficial for the performance, it is difficult to make that choice. Qi Chen et al. [14] suggest square root of total number of features as a default number, but another approach would be to test RFE with different numbers, although on the downside it is more time consuming.

Overall RFE is an efficient tool, however Qi Chen et al. state in the research that it is not the best solution for big feature space with high-correlated features [14]. In general, feature selection is limited to the existing features and hence can lose some information by dropping the least useful columns. Feature transformation approach, on the contrary, attempts to combine all features in the most efficient way to create new columns [4].

Probably most well-known transformation technique is Principal Component Analysis, and the main idea behind the mechanics of this method is to select several correlated columns in order to project them onto the new coordinate system where new features, called principal components, would demonstrate correlation of a lesser degree [4]. Figure 3 represents an example of mapping original features onto new axes.

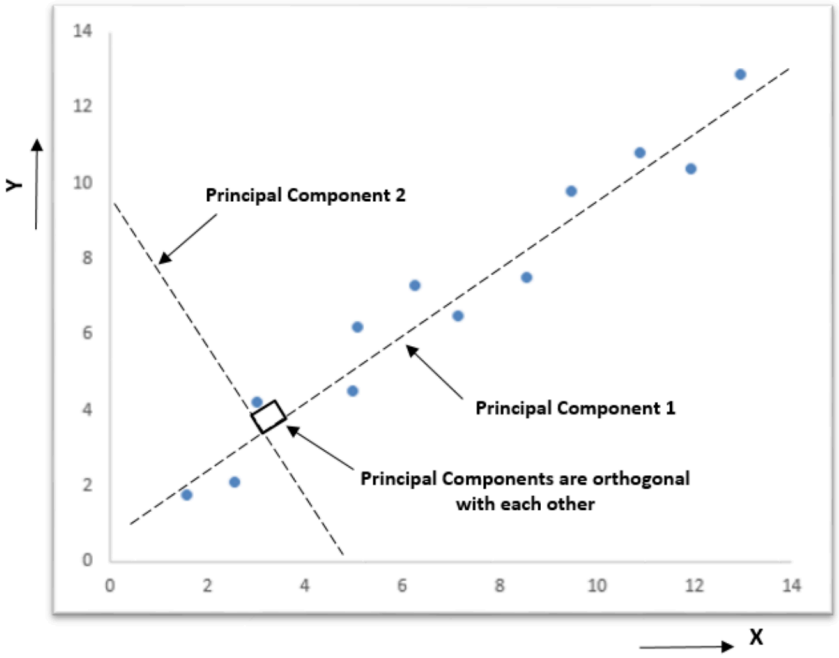

Figure 3. Feature transformation example by PCA. [16]

As figure 3 presents, projection space is chosen in the way, so the first principal component captures the most of variance. The second component is orthogonal to the first and contains most of the remaining variance. By analogue the rest of principal components follow the same mechanism, capturing most of the remaining signal from the previous component [7].

Unfortunately, this approach is not suitable for sparse data, as PCA implies centering of data as a first step, and it requires a significant amount of computational resources. In the case of sparse data, however, it is advised to use `TruncatedSVD` for dimensionality reduction [4]. SVD stands for Singular Value Decomposition and often happens to be a part of PCA implementation itself. To describe SVD more specifically, certain definitions are required.

A feature engineering textbook describes it in the following manner: "Let A be an n × n matrix. If there is a vector *v* and a scalar *λ* such that $Av = \lambda v$, then *v* is an eigenvector and *λ* an eigenvalue of A. Let A be a rectangular matrix. If there are vectors *u* and *v* and a scalar *σ* such that $Av = \sigma u$ and $A^T u = \sigma v$, then *u* and *v* are called left and right singular vectors and *σ* is a singular value of A." [8]

Technically any matrix can be decomposed into three particular matrices as presented in the formula:

$$A = U\Sigma V^T$$

Here *U* and *V* are orthogonal matrices, where *U* contains columns of left singular vectors, *V* contains right singular vectors and *Σ* is a diagonal matrix of the singular values. If out matrix *A* has dimensions of n x d, then *U* also has dimension n x d, while *Σ* and *V* have dimensions d x d. Basically, matrix *V* contains the unit vectors defining principal components. [8.]

Once principal components are identified, dimensionality reduction is performed by projecting dataset on a hyperplane defined by first *k* principal components, where *k* is a desired feature size. Algebraically it is described like this:

$$AV_k = U\Sigma V^T V_k = U_k \Sigma_k$$ [8]

The next question is how to choose *k.* To visualize this, scree plot presented on figure 4 can be useful.

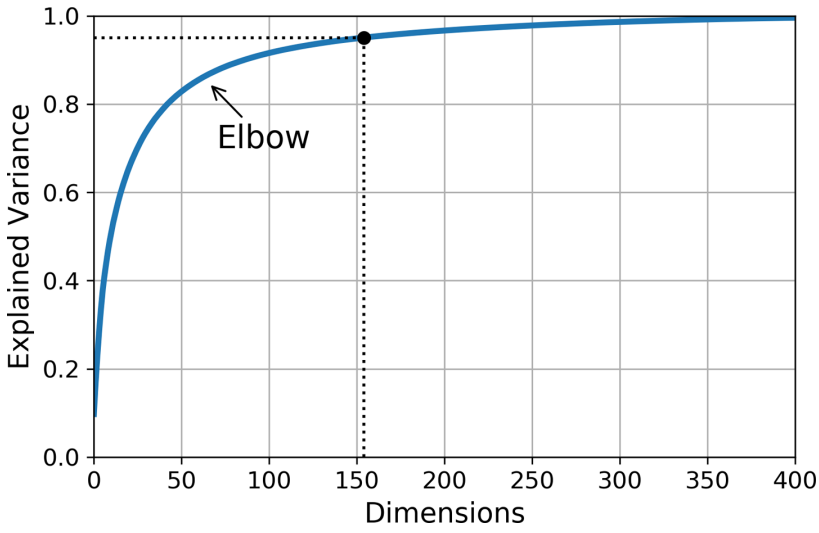

Figure 4. Scree plot of total variance explained by the number of principal components [9]

The plot on the figure 4 shows variance explained by each of the principal components as well as the sum of these. There are two main methods to choose k by consulting this plot. First is to choose a number of components explaining enough variance (this percentage is defined differently in various sources, from 80 to 95%). Another approach is to choose the number of components where the curve of total variance explained starts to grow more insignificantly (an elbow).

2.2.3 Data augmentation

As was mentioned previously, datasets can also be increased by the sample size. Data augmentation is widely used in the field of image classification, where lack of observations is a common issue. In this case datasets can be enlarged by transforming original samples with scaling, rotating, skewing, adding noise etc. Although the data produced in described way is synthetic, it still clearly represents target objects and is likely to be an efficient solution. [1]

Furthermore, data augmentation is beneficial in other cases of classification, especially when the dataset is imbalanced. It is important to have a distribution of targets close to uniform, otherwise it can lead to poor performance of the predictor, which has a risk to

be left unnoticed [1]. To provide an example, assume a dataset with 950 samples labeled as 0 and 50 samples labeled as 1. Even if the predictor classifies every sample as 0, accuracy is still 95%, which is commonly considered rather high. For this reason, it is important to increase the number of samples for an underrepresented class. Apart from that it is also essential to use a varied set of metrics when evaluating a model, which will be discussed in later sections.

One of the most common ways to augment the dataset is to use resampling. The technique is trivial; however, it is limited to the usage of the existing data. Therefore, an increased dataset will contain exactly the same data repeated several times [1]. Moreover, if a certain number of samples contain noise or irrelevant information, duplicating these items will increase bias in the dataset and decrease performance of a model.

A more reliable approach has been introduced by Chawla et al. (in SMOTE: Synthetic Minority Over-Sampling Technique, Chawla N. V., Bowyer K. W., Hall L. O., Kegelmeyer W. P., Journal of Artificial Intelligence Research, 16/2002). Synthetic Minority Over-Sampling Technique, as opposed to a previous approach, aims to create new samples based on distribution of the minor class. Figure 5 illustrates the main idea behind SMOTE approach.

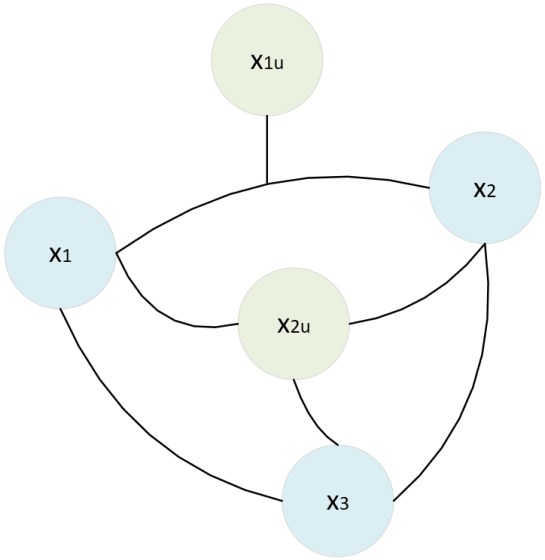

Figure 5. Illustration of SMOTE work principle. [1]

As can be seen from figure 5, described technique takes into account relationship between existing samples (samples X1, X2, X3), belonging to the same neighbourhood. The term *neighbourhood* refers to a group of points with mutual distances below a certain threshold. After neighbours were selected, new samples (X1u, X2u) are generated along the segments, connecting original data points. Contrary to resampling approach, SMOTE allows to have higher variance in the dataset, hence it becomes easier for a classifier to define a separation hypersurface. [1] Figure 6 represents a result of SMOTE data augmentation.

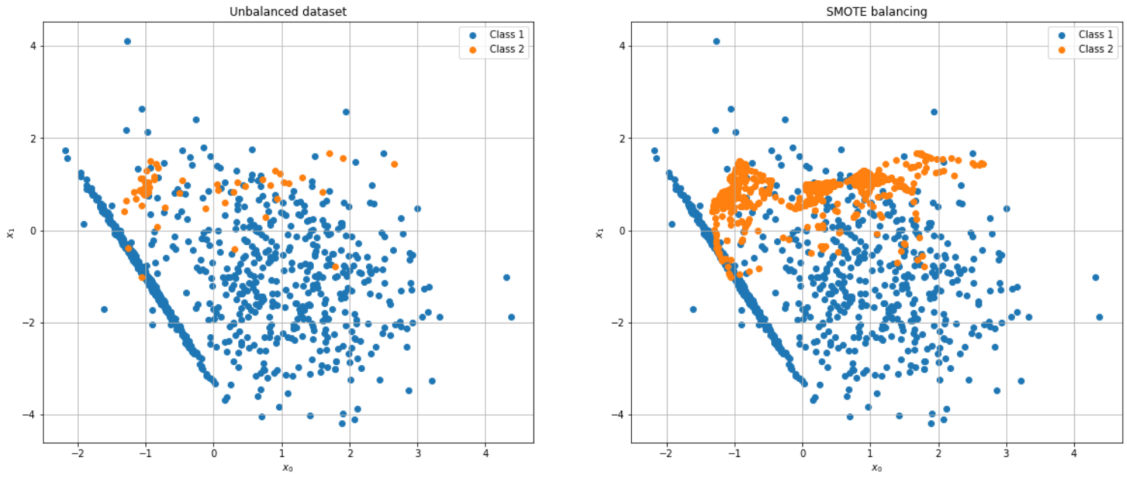

Figure 6. Original imbalanced dataset on the left and dataset balanced by SMOTE on the right [1].

Figure 6 illustrates that minor class significantly increased along the same distribution, clarifying the distinction between classes, even though overlap of the classes is present in this dataset.

The solution described above is designed to handle datasets with class labels, but in some cases, up-sampling is also required in regression problems, when faced with insufficiency of training data to predict extreme values, especially in complex models. This task is less trivial than SMOTE, since apart from the feature values, the value of the target for a synthetic sample itself needs to be defined being a continuous variable. Torgo et al. [10] offers a solution by adapting SMOTE approach for regression. Dataset is passed to the algorithm, threshold of relevance is defined by the user (which determines a set to be up-sampled), along with the number of neighbors used to create a new sample, and a percentage of over and under-sampling. To assign a new target variable it is

suggested to use a weighted average of the k original samples, where weights are defined as an inverse function of the distance between synthetic sample and each of these k samples. It should be noted, that in the case of up-sampling for both extreme high and low cases, these procedures should be made separately, as it is practically equal to belonging to different categories. [10] Unfortunately, up-sampling for regression is yet to be implemented in python libraries and thus have to be performed manually.

2.3 Model selection

Choosing a model is an important step of predictive analytics. While it is possible to test all the most common models and choose the best, it is time consuming and considered a bad practice. Basic understanding of a problem and specifics of the dataset can rule out several approaches and help to focus on the most suitable ones.

When dealing with low-quality datasets and lack of data in general, it is important to choose models more resistant to noise in the data. Random Forests are a clear example of a robust model, while simple linear regression is considered sensitive to outliers. Moreover, when working with small datasets it is advised to focus attention on relatively simple models, as complex approaches like neural networks usually require a significantly higher number of observations to find a pattern in the data.

2.3.1 Decision Tree and Random Forests

One of the most powerful algorithms nowadays is Random Forest. It is an ensemble method based on decision trees, so they will be explained first.

Decision Tree is a versatile method which suits both for classification and regression problems and is able to deal with complex tasks as well. One of the advantages of the method is the fact that decision trees do not require excessive preparation, neither feature scaling nor centering [9]. Moreover, decision trees do not depend on linear relationship between features and target variable like linear regression would.

There are several different Decision Tree implementations introduced by today, including ID3, C4.5, CART and CHAID. CART (Classification and Regression Trees) algorithm will be described in detail, as it is available in python library `scikit-learn` and able to perform both classification and regression. Figure 7 illustrates a simple decision tree on a well-known Iris dataset.

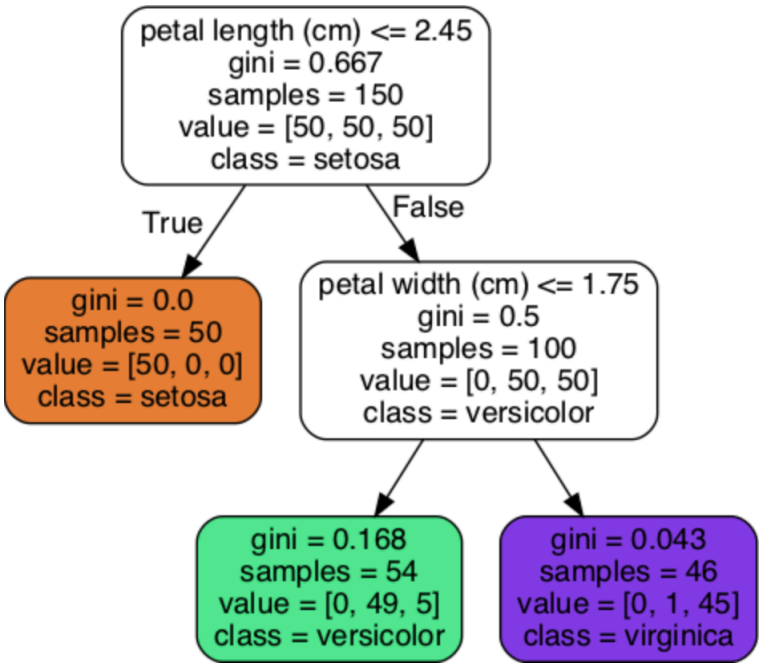

Figure 7. Decision tree for Iris dataset [9]

As can be observed from Figure 7, CART algorithm produces only binary trees, which means that each leaf can have only two children. The mechanics of the algorithm is to split data by a single feature k and threshold t_k, which produce the purest subsets, weighted by their size. The cost function of the algorithm is presented by the following equation:

$$J(k\ t_k) = \frac{M_{left}}{m} G_{left} + \frac{M_{right}}{m} G_{right}$$

where G_{left} and G_{right} measure the impurity of corresponding subsets, and m_{left} and m_{right} are the sizes of subsets. [9]

In classification trees two impurity measures are usually used: Gini and entropy. Gini index of a subset is calculated by this formula:

$$G_i = \sum_{k=1}^{n} {p_{i,k}}^2,$$

and entropy is calculated as:

$$H_i = \sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log\ (p_{i,k}),$$

where in both cases $p_{i,k}$ is the ratio of class k samples to the whole number of samples in the subset *i*. As can be seen from the formulas, subset is considered pure, when impurity index is 0 [9].

Usually the choice of the impurity measure does not change a performance significantly, but Gini index is considered to be faster to compute, if the speed of the predictor is important. Regression decision tree acts similarly, but using mean square error as an impurity measure instead [9].

Splitting each subset into new subsets continues recursively, until it is impossible to reduce impurity, and the maximum depth defined by a user is reached.

It is clear that CART is a greedy algorithm: while it searches for a best split for a current level, it does not account for the possible splits coming after it, therefore even if the split is not the most beneficial in the long run, model cannot know about it. Unfortunately building the most optimal tree is known to be an intractable problem due to its computational complexity. However, the greedy algorithm is still considered to be a reasonably efficient model.

One of the main disadvantages of decision trees is that when not pruned, they build overly complex structures around training datasets which lead to overfitting. Ensemble models are made to solve this problem, in particular, a random forest model.

Random forests and ensemble models in general use a group of estimators instead of single one and choose an output based either on majority vote of estimators (called hard-voting), or on the highest probability averaged from all estimators (called soft-voting).

Two aspects should be considered in ensemble models:

1. Each model should perform better than null model, even if it is weak.

2. Model prediction process is independent from other predictors. In other words, predictors should ideally be diverse and uncorrelated. [2]

While a simple voting model can consist of a number of different prediction models, random forest represents a set of decision trees, therefore additional measures are considered to ensure independence of singular predictors. First of all, bagging is used to pick a training set for each predictor. Bagging is defined as sampling with replacement, which ensures that predictors are not trained on completely unique datasets. Next, random feature selection is performed for each predictor [11]. If the overall number of features is n, then for one predictor square root of n is chosen in case of classification, and from p/3 to p in case of regression [2].

This model is beneficial for low-quality data, as bagging allows to decrease an effect of noise onto predictions, and voting also improves generalization ability of an algorithm. Even though computational cost is considered one of few RF disadvantages, especially with high number of trees, it is not an issue with small datasets.

2.3.2 Gradient Boosting Machines

Another side of ensemble methods is boosting. Unlike bagging, it does not use sampling with replacement, instead, it uses the same training set and sequentially trains estimators to improve the performance of previous estimator and adds it to the final model (this approach is also known as Forward Stage-wise Additive Modelling) [1]. Gradient Boosting, employing the described technique, is one of the most effective algorithms, used in many winning kaggle competitions.

To be more specific, each following estimator, normally represented by a Decision Tree, attempts to fit the residuals of the previous estimator in order to minimize a global cost function by employing gradient descent method [1]. It is important to note, that each estimator is multiplied with a constant variable 'learning rate' to control the contribution of estimators into the final model. This method is related to regularization techniques and will be covered in the next section.

2.4 Model tuning and evaluation

When dealing with noisy datasets, it is imperative to tune the model, especially use regularization to improve generalization ability of a model and to stop it from overfitting to the noisy values.

Learning rate is a parameter to regularize GBM by controlling the contribution of each estimator into the final model, and this method is called shrinkage. Figure 8 illustrates the performance of GBM with different learning rates.

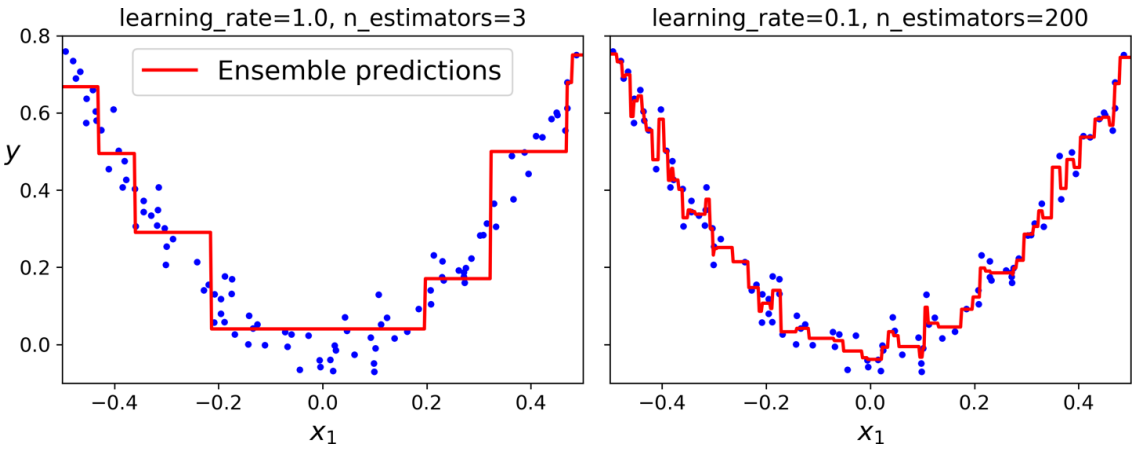

Figure 8. GBM performance with different learning rates [1]

It is clear that the less is a contribution of a single estimator into the model, the more estimators are required overall. Figure 8 represents two cases where combination or learning rate and a number of estimators is not optimal. In the first case, the model is overly generalized, while in the second case it is excessively precise and overfitting [1]. Moreover, an increased number of estimators requires more computational power, hence the goal of model tuning is to find the best combination of parameters to minimize

the testing error along with avoiding overly complex models. Fortunately, small datasets allow for extra complexity, therefore the main concern should be in avoiding overfitting.

Unfortunately, GBM is still more prone to overfitting in noisy environments than Random Forests. One of the solutions to avoid this is to use subsampling, which specifies a number of samples to be used in training of each estimator [1]. Evidently, this approach might not be optimal for small datasets, because the amount of training data is limited already, however it is worth a try.

Regarding RF, random set of features for estimators is used to regularize the model, as well as constraining the maximum depth of the trees to a chosen number.

2.5 Prediction intervals

One of the regression problems includes a prediction of a single point value without any confidence measure, unlike classification, which usually provides probability of the class correctness. While in noisy environments point predictions can be prone to error, confidence and prediction intervals can be used to increase certainty in the predictions.

Confidence interval is a range of values containing a true population value with chosen degree of certainty. In predictive modelling, confidence intervals can often be used to describe uncertainty in model parameters, for example regression slope coefficients [2].

The idea of prediction intervals is more important in predictive modelling, as it aims to quantify confidence for a future outcome. Practically it represents a range where future prediction is expected to fall with certain probability. [12]

In machine learning, quantile regression is often used to generate prediction intervals. In `scikit-learn` it is implemented in Gradient Boosting Machines algorithm by estimating conditional quantiles. To generate a prediction interval with confidence level, three estimators need to be created: prediction of the lower boundary, prediction of the higher boundary and finally prediction of the median. Figure 9 presents an example of quantile regression in action.

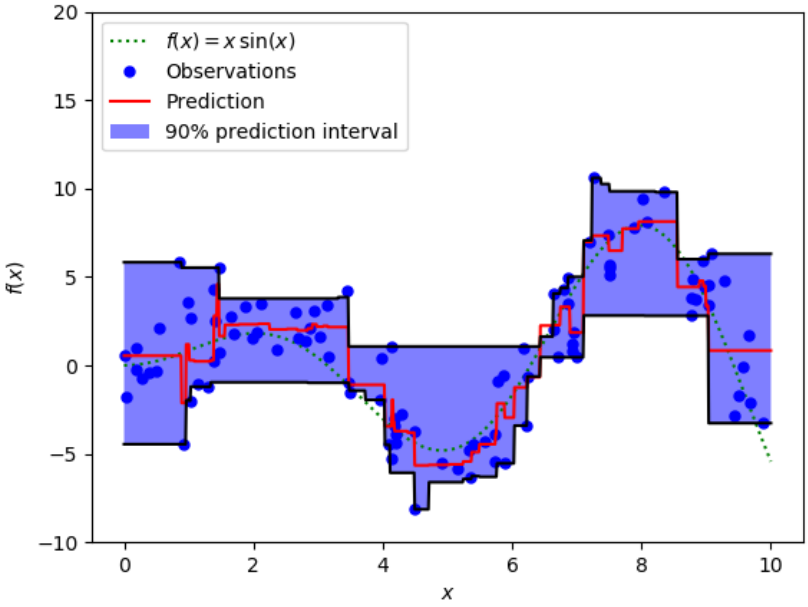

Figure 9. Quantile regression with 90% prediction interval

On figure 9 one can see upper and lower boundaries for prediction intervals, containing observation, as well as an actual prediction line. Depending on the task following result can appear extremely useful.

3 Methods

3.1 Tools

Language chosen for project implementation is Python, due to being more software oriented and suitable for further development of the prediction algorithm as a microservice. Furthermore, there is a high number of packages and libraries available for implementing data science projects in Python.

`Numpy` provides instruments to create and manage multidimensional arrays, and *Pandas* expands it with the concept of dataframes, allowing to handle complex tables containing different data types [7]. Both tools are indispensable for data science projects, facilitating an exhausting and time-consuming stage of data preparation.

`Scikit-learn` is the machine learning core for Python. It provides instruments for data preprocessing, supervised and unsupervised algorithms, model evaluation etc. [7]. In the current project scikit-learn is utilized extensively.

`Matplotlib` and `Seaborn` are the main python visualization libraries helping to create high-quality plots.

Last but not the least is `statsmodels`, an important tool to conduct analysis and statistical tests. In this project, it is used in the stage of exploratory analysis.

3.2 Data collecting and exploration

First step of any data science project includes data collection. In this case it means fetching available data from several internal databases, regrouping, calculating additional information from available data where possible and finally constructing a structured dataset for further use.

Next step is to perform exploratory analysis in order to get initial insights from the data, which helps in defining further course of actions. Exploratory analysis may include visu-

alization of main characteristics and relations between the features, such as basic statistics, distributions, correlation matrices, as well as performing statistical tests. In the current project, one-way ANOVA (Analysis of Variance) test is utilized along with visualizations to establish the dependency between target variable and categorical features. ANOVA test is used to compare two or more different groups on a continuous variable by analyzing a contribution of variation between the groups and variation within the groups into total variation. If the result of this test is high enough, it can be concluded that tested categorical variable is statistically significant. Post-hoc test is conducted as well to detect the pairs of groups with significant difference [17]. A popular way to conduct a post-hoc test in Python is Tukey honestly significant difference (HSD) test.

3.3 Data preparation

Data preparation includes encoding categorical variables either by labeling them with numerical values or by one hot encoding. Labeling is used with ordinal data (data that has an order and can be associated with numbers and distances between them). A great example is categories such as “small”, “average” and “big”, which can be mapped to [1,2,3]. One hot encoding is suitable for nominal data, and implies splitting a feature into a set of new binary features representing presence or absence of each category [4].

If some values are absent from the dataset, they should be handled as well. There are several solutions to tackle this problem. One of these is to drop the observations or columns with absent values, while another is imputation with a certain value. Usually mean is used for numerical features, and most frequent value for categorical ones [4]. Choice of the strategy depends on a particular dataset.

Another important task already described in theoretical background is outlier detection, and discussed techniques such as Isolation Forest, Elliptic Envelope and LOF are tested in this project and compared by the performance of Random forest on an altered dataset.

3.4 Model comparison and evaluation

After the dataset is prepared, different prediction models can be tested, and their performance can be compared. In this project Random Forest and Gradient Boosting, previously discussed in detail, are used. To get a more reliable and stable metrics, k-fold cross validation is used in assessing model performance. Cross validation implies that data is divided into several segments, and model is tested iteratively with one of the partitions as a validation set and the rest of the data as a training set. Figure 10 below gives more intuition into the process.

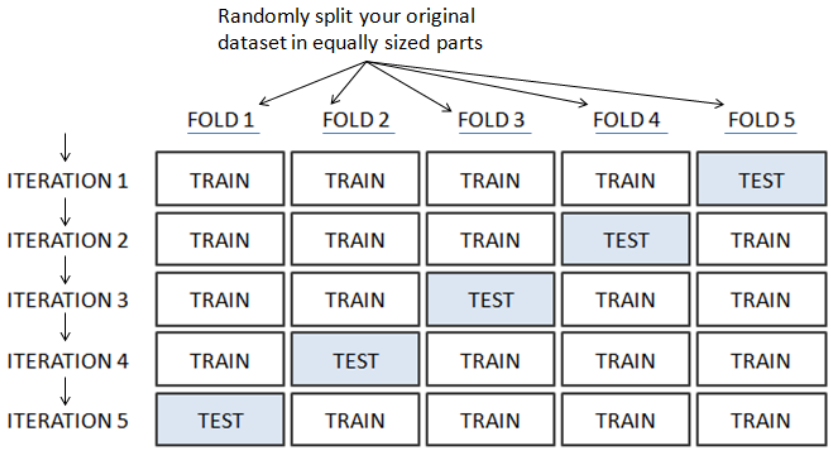

Figure 10. Cross validation mechanism [7].

As can be observed from figure 10, a different set is picked for testing in each iteration, which leads to a variation in performance scores between iterations. Thus, the average of these scores is a more reliable and illustrative measure. It is recommended to use at least 10 folds to secure training sets of a sufficient size, however in the case of particularly small datasets the number of folds can be increased up to the point where only one observation is left out for validation [7]. This approach is called Leave One Out Cross Validation (LOOCV).

Performance assessment for the project is based on two most common measures for regression: MSE (Mean Squared Error) and R^2 (Coefficient of Determination).

Mean Squared Error metrics is self-explanatory, calculating the mean of squared differences between predictions and target values. Mathematically it is defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - y\_pred_i)^2$$

However, this metric is badly interpretable due to the fact that it returns a squared unit, therefore it is common to take a square root of MSE, which is referred to as RMSE. It is also important to note, that large deviations in predictions, being squared, affect the metrics considerably, which, on the one hand, allows to penalize a model stronger for significantly wrong predictions, but on the other hand, it makes the metric less reliable. [18.]

R^2 or coefficient of determination represents proportion of variation in a target variable explained by a prediction model [18]. It is calculated through the following formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_{pred_i} - y_i)^2}{\sum_{i=1}^{N}(y_{pred_i} - \bar{y})^2}$$

In order to simplify model tuning, scikit-learn library provides a tool named GridSearchCV, which accommodates for testing different model parameter combinations with cross validation. To perform grid search, desired parameter ranges or options should be specified as well as a model and scoring function. Subsequently, the tool finds the best model and saves it for further use. The major disadvantage of the approach is low speed; hence parameter ranges should not be overly large, and basic understanding of model tuning for current case is still necessary.

4 Implementation and results

In order to demonstrate the project implementation, a partially synthetic dataset was used, for the reason that the initial dataset contained sensitive data. Furthermore, only a half of the dataset was used in a project, and the whole dataset was used afterwards to demonstrate the difference caused by the lack of data.

First, exploratory analysis of the dataset was performed. Figure 11 represents basic descriptive statistics for numeric features of the dataset.

```
df.describe().round(3)
```

	fa	fc	fj	fj_binned	fl	fp	fp_count	fr_act	fr_binned	target
count	654.000	654.000	654.000	654.000	654.000	519.000	519.000	654.000	654.000	654.000
mean	15.231	2.355	0.323	2.145	2.281	125.187	38.559	84.661	2.471	22.088
std	9.609	0.985	0.289	0.735	0.508	136.748	51.022	103.185	0.576	6.960
min	0.000	0.612	0.067	1.000	2.000	1.000	1.000	0.000	1.000	4.493
25%	8.000	1.550	0.133	2.000	2.000	31.000	6.000	16.667	2.000	16.400
50%	13.500	2.175	0.200	2.000	2.000	77.000	22.000	39.167	3.000	21.899
75%	21.000	3.138	0.467	3.000	3.000	173.000	45.000	120.000	3.000	26.900
max	36.000	5.600	1.467	3.000	4.000	921.000	293.000	635.000	3.000	46.057

Figure 11. Descriptive statistics of the project dataset

As can be observed from figure 11, the number of samples in the dataset amounts to 654, with missing values in several columns, such as `'fp'` and `'fp_count'`. Since it is impossible to use the majority of algorithms in scikit-learn with missing values, they were imputed, which is described further in the chapter. However, a test set was withheld from the dataset first, so that evaluation could be performed on untouched data.

Another issue evident from the figure is the fact that maximum values for the features `'fr_act'`, `'fp_count'` and `'fp'` deviate significantly from the rest of the data (distance to the mean is more than three standard deviations). These values are considered outliers and might represent errors in the data.

In order to examine the relationship between the features, and especially the relationship to the target value, correlation matrix was generated. This step, once again, is suitable only for numerical features and relies on Pearson correlation coefficient, which measures linear correlation between the feature vectors.

Following code in listing 1 calculates correlation matrix and plots it in the form of a heat map with the help of seaborn library.

```
corr = df.corr()
ax = sns.heatmap(corr, vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True, annot=True)
plt.show()
```

Listing 1. A Python code to display a correlation matrix

The result of this code can be seen on figure 12.

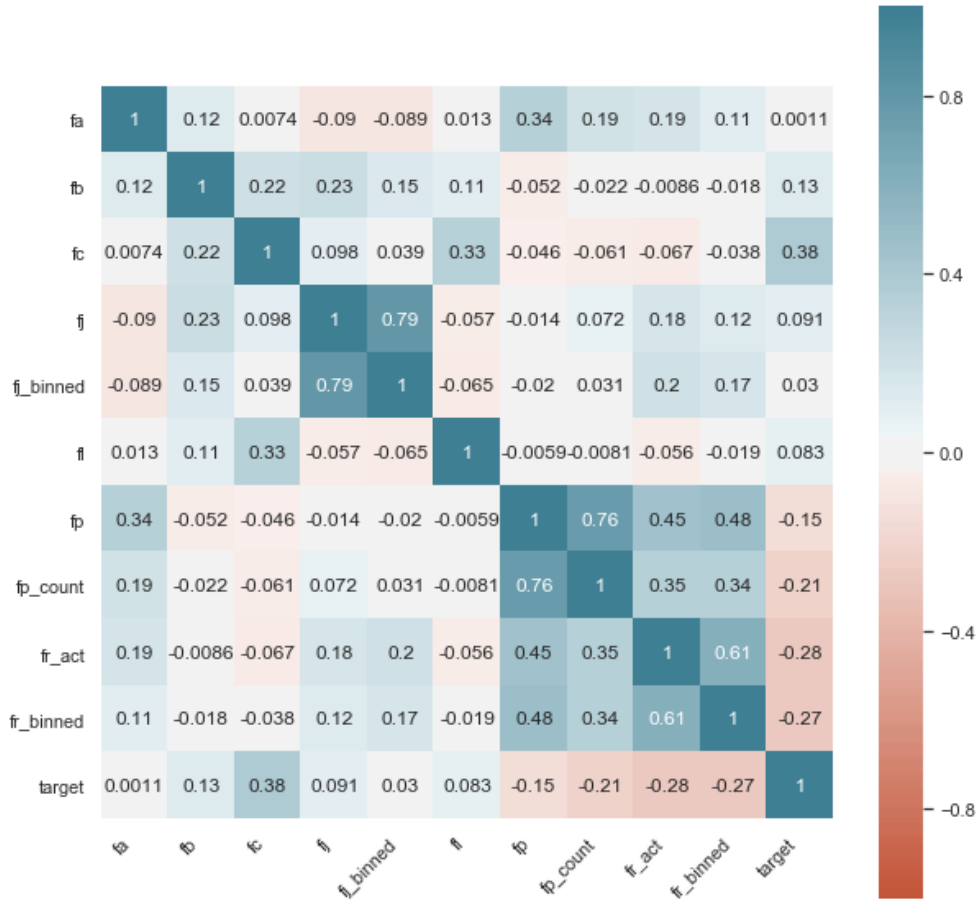

Figure 12. Correlation matrix of the project dataset

Evidently from figure 12, feature `fc` has the highest positive correlation with the target value, and feature `fr_act` has a relatively high negative correlation with target as well.

In fact, `fr_binned`, which shows almost the same result as `fr_act` is just a different representation of the same value, as can be concluded from the feature name. To be more precise, it represents `fr_act` values binned into three intervals, hence it also has a high correlation with `fr_act`. Similar situation is observed for the pairs `fj / fj_binned`, and `fp / fp_count`. While initially these features were modified to have several representations in order to see whether continuous or binned variable will be more efficient, using these at the same time is considered a bad practice, as it causes multicollinearity. Therefore, in the later steps one feature of each pair was omitted.

As was mentioned previously, Pearson coefficient only offers a glance into linear correlation, and cannot correctly describe the correlation between variables, if, for example, polynomial or logarithmic relation is present. To make sure a significant part of information is not lost, it is possible to outline a pairplot, as a quick tool to see scatterplots between all feature combinations simultaneously. Pairplot of the project dataset is presented in Figure 13.

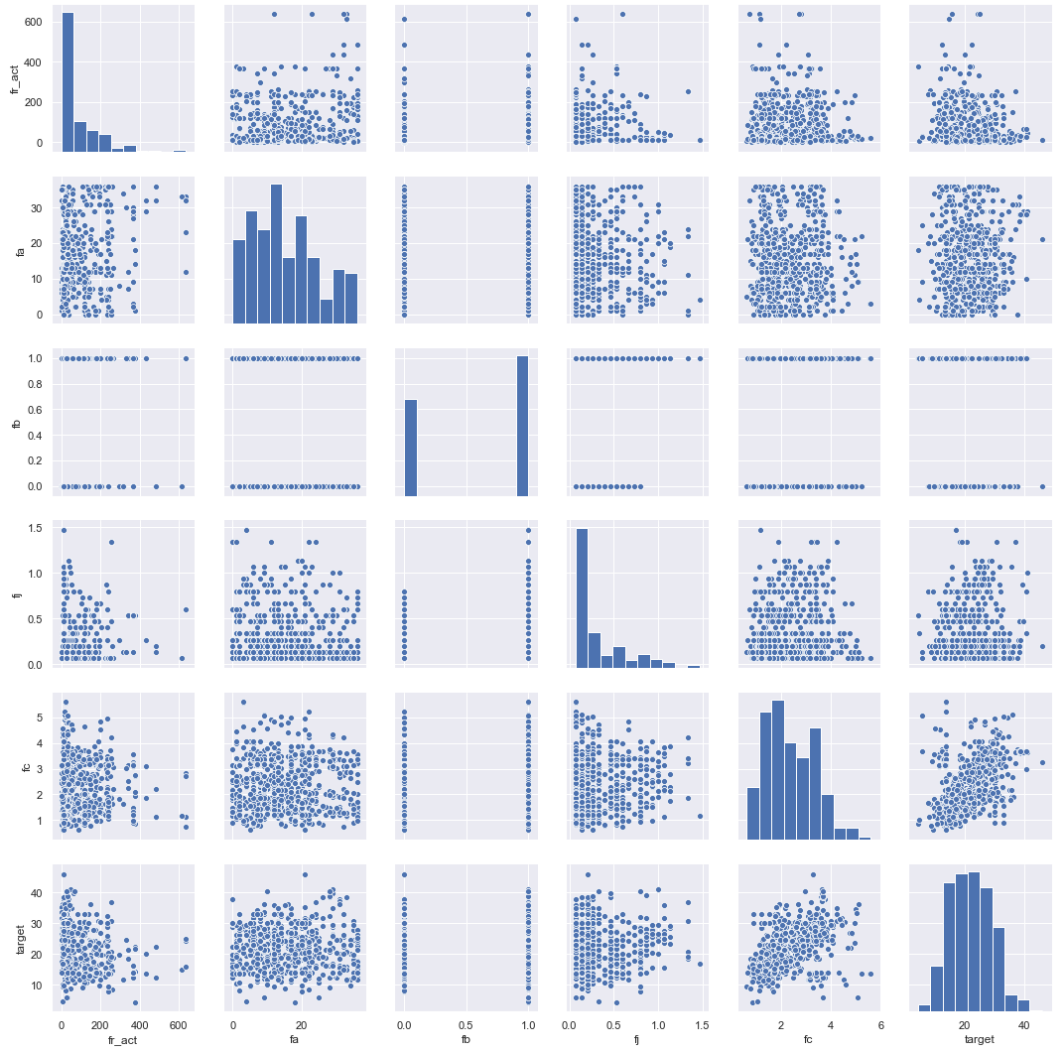

Figure 13. Pairplot of the project dataset

As can be observed from figure 13, there is a clear linear correlation between the `target` and `fc` feature, which was evident from the previous figure. However, there is no sign of any new and useful data relationship.

Next, contribution of categorical variables into the target was evaluated. ANOVA was used for that purpose, and it was implemented with `statsmodels` package. Output of ANOVA test is displayed on figure 14.

```
results = ols('target ~ C(fz)', data=df).fit()
aov_table = sm.stats.anova_lm(results, typ=2)
aov_table
```

	sum_sq	df	F	PR(>F)
C(fz)	11048.847077	7.0	49.527663	2.265451e-56
Residual	20587.499719	646.0	NaN	NaN

Figure 14. ANOVA result for categorical feature 'fz'

As can be seen from figure 14, p-value (PR) is small enough to say that the difference in category means is statistically significant. ANOVA was followed by `TukeyHSD` test in order to compare categories pairwise. The output of `TukeyHSD` is presented on figure 15.

```
mc = MultiComparison(df['target'], df['fz'])
mc_results = mc.tukeyhsd()
print(mc_results)
```

```
Multiple Comparison of Means - Tukey HSD,FWER=0.05
=============================================
group1 group2 meandiff  lower    upper   reject
---------------------------------------------
 cat1   cat2  -11.7771 -14.9032  -8.651   True
 cat1   cat3  -15.6657  -18.79  -12.5414  True
 cat1   cat4  -8.4851  -11.8487 -5.1216   True
 cat1   cat5  -13.0065 -16.2616 -9.7514   True
 cat1   cat6  -11.1083 -15.3657 -6.8508   True
 cat1   cat7  -20.3719 -24.4822 -16.2616  True
 cat1   cat8  -11.3914 -15.6913 -7.0915   True
 cat2   cat3  -3.8886  -5.7872   -1.99    True
 cat2   cat4   3.292    1.0212   5.5628   True
 cat2   cat5  -1.2294  -3.3363   0.8775  False
 cat2   cat6   0.6689  -2.7908   4.1286  False
 cat2   cat7  -8.5948  -11.8716 -5.3179   True
 cat2   cat8   0.3858   -3.126   3.8975  False
 cat3   cat4   7.1806   4.9122   9.449    True
 cat3   cat5   2.6592   0.5549   4.7635   True
 cat3   cat6   4.5575   1.0994   8.0156   True
 cat3   cat7  -4.7062  -7.9813   -1.431   True
 cat3   cat8   4.2744   0.7642   7.7845   True
 cat4   cat5  -4.5214  -6.9668   -2.076   True
 cat4   cat6  -2.6231  -6.2988   1.0525  False
 cat4   cat7  -11.8867 -15.3908 -8.3827   True
 cat4   cat8  -2.9062  -6.6309   0.8185  False
 cat5   cat6   1.8983  -1.6784   5.475   False
 cat5   cat7  -7.3654  -10.7655 -3.9652   True
 cat5   cat8   1.6152  -2.0119   5.2422  False
 cat6   cat7  -9.2636  -13.633  -4.8942   True
 cat6   cat8  -0.2831  -4.8313   4.2651  False
 cat7   cat8   8.9805   4.5698  13.3912   True
---------------------------------------------
```

Figure 15. Tukey HSD results for categorical feature 'fz'

'Reject' column, displayed on figure 15, shows whether the difference between two categories is statistically significant (with True being significant, meaning that null hypothesis can be rejected). From the tests it could be concluded that the categorical feature `fz` is correlated with the target variable and should be used further in prediction analysis.

With the means of performing exploratory analysis, the information regarding the features contributing into target variable the most is obtained, and technically it is possible to reject the least correlated features during this step, however there is a chance that these features affect the target variable in combination with other features, therefore it is more reliable to perform automated feature elimination with RFE.

First of all, data preparation was performed. Dataset was split into feature set and target vector. Next, in order to use categorical features in further process, they were encoded by one hot encoding, as shown in listing 2.

```
def encode_catdata(data, columns):
    for col in columns:
        one_hot = pd.get_dummies(data[col])
        data = data.join(one_hot)
    data.drop(columns, axis = 1, inplace = True)
return data

data = encode_catdata(data, ['ft', 'fz'])
```

Listing 2. Python function for categorical data encoding

In order to have pure untrained data for tests to have a less biased result, data was separated into training and test set. Suitable tool is available from `scikit-learn` model selection module.

```
X_train, X_test, y_train, y_test = train_test_split(data, target,
test_size=0.2, random_state=1)
```

Listing 3. Dataset splitting in Python with scikit-learn

Commonly data is split into 80% for training and 20% for testing, but with small datasets it is difficult to find the balance, since deficiency of training data might lead to high prediction variance, and lack of test data means performance metric will have greater variance, thus, less reliable. However, as cross validation score was used for evaluation, small testing set size was not as crucial. Next, missing values were handled, as was mentioned earlier, and imputer available from `scikit-learn` preprocessing module was used for this purpose.

```
imp = Imputer(missing_values=np.nan, strategy='mean')
X_train = imp.fit_transform(X_train)
X_test = imp.transform(X_test)
```

Listing 4. Python implementation of missing values handling

The code above substitutes NaN values in each column with mean values of these columns. This approach is acceptable when the percentage of missing values per column is not high, otherwise it would be more suitable to remove the columns with high number of missing values entirely.

Next step, outlier detection, utilized techniques described in detail in the literature review chapter. These techniques include isolation forest, LOF and Elliptic envelope. In order to compare the performance of listed approaches, untuned random forest with cross validation was performed on modified data. Detected outliers are presented in scatterplots in figure 16.

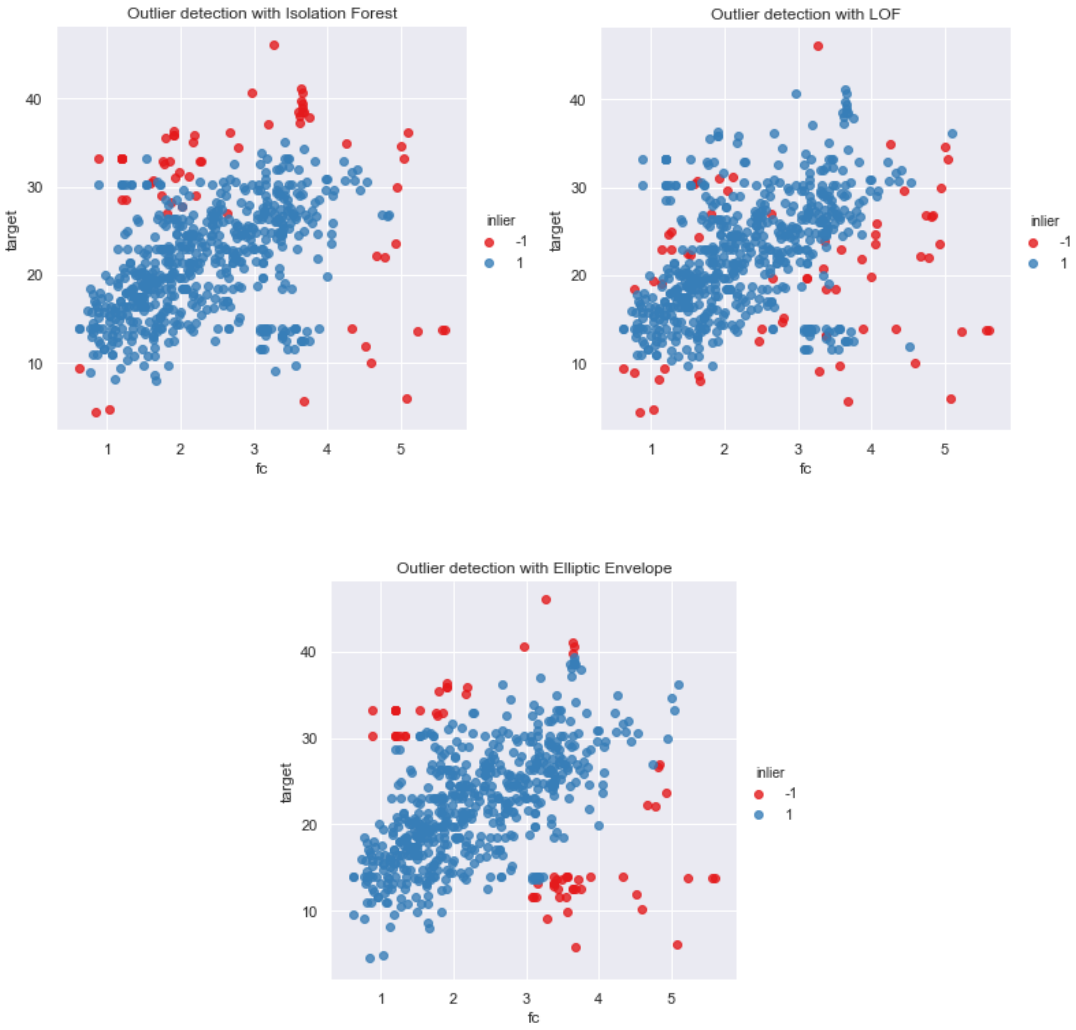

Figure 16. Performance of outlier detection algorithms of the project dataset

Figure 16 represents scatterplots of target variable as dependent from `fz`, feature, most correlated with the target. As can be seen from the figure, both Isolation Forest and LOF act in a similar way, defining a bigger and a smaller observation formations as clusters, omitting points around as outliers. Elliptic Envelope, on the other hand, defines an elliptic shaped area of inliers excluding everything around it. As it was not clear at this stage, whether the smaller formation contained valuable information, or was it a set of errors, performance evaluation was implemented, with the results presented in table 1.

Table 1. Performance scores with different outlier detection techniques

	CV R^2	CV RMSE	Test RMSE
No outlier detection / RF	0.47	4.99	5.23
Isolation Forest / RF	0.39	4.56	4.57
LOF / RF	0.53	4.49	4.89
Elliptic Envelope / RF	0.58	3.69	4.04

The first row of table 1 represents a performance of untuned Random Forest on unmodified dataset and is used as a baseline. The rest of the rows represent a performance with different outlier detection techniques applied beforehand. As can be observed from the table, isolation forest was the least effective for our case, decreasing the accuracy compared not only to other techniques, but even to a baseline. Elliptic Envelope, in its turn, showed the best result, improving accuracy score by 0.11 (23% of baseline accuracy) probably due to the fact that the data had mainly normal distribution.

Next step was to evaluate dimensionality reduction and feature selection contribution to the model performance. For this task data modified with the most successful outlier detection technique was used to assess a further improvement.

Several different options were tested for a number of features to select parameter in RFE along with feature importance provided by Random Forest. Figure 17 justifies a choice for aforementioned parameter.

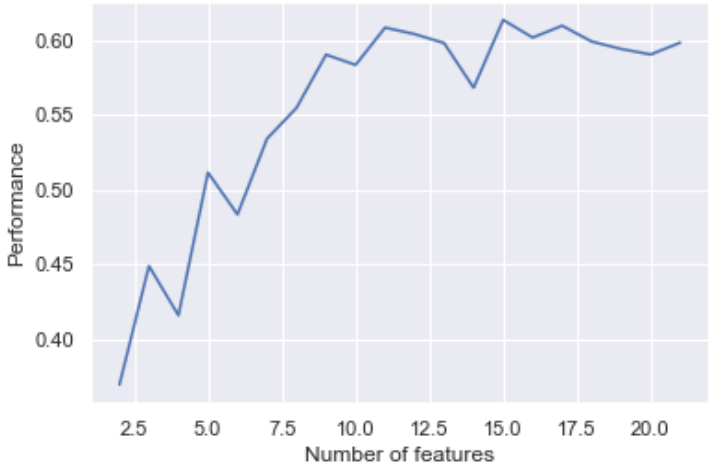

Figure 17. Performance of RFE based on number of features selected

Figure 17 represents a R^2 score of untuned Random Forest model after performing RFE with different numbers of features selected. Evidently, performance stops improving approximately after 10 of features selected, so using 10-12 features as a parameter was considered optimal for the current model.

Recursive feature elimination was performed by using the following code:

```
rfr = RandomForestRegressor()
rfe = RFE(estimator=rfr, n_features_to_select=12, step=1)
X_train_rfe = rfe.fit_transform(X_train, y_train)
X_test_rfe = rfe.transform(X_test, y_test)
```

Listing 5. Python implementation of RFE

The model uses untuned Random Forest as an estimator by utilizing feature importance provided by RandomForestRegressor tool. RFE model is trained on the training set, which is subsequently transformed, and after that test set is transformed by an already pre-trained model.

In order to perform PCA, data required scaling first. StandardScaler tool from `scikit-learn` preprocessing library was used for that purpose. Similar to the previous example, the model was trained on the training set, and after that modification for both training and test was performed. It is considered a standard practice when transforming data and is not explicitly mentioned in the text anymore.

```
scaler.fit(X_train)
X_trans = scaler.transform(X_train)
X_trans_test = scaler.transform(X_test)
pca = PCA(.99)
pca.fit(X_trans)
X_trans = pca.transform(X_trans)
X_trans_test = pca.transform(X_trans_test)
```

Listing 6. Python implementation of PCA with scaling

During PCA model creation desired explained variance proportion was set as a parameter, meaning that 99% of variance was to be explained by principal components created with PCA model. In order to compare performance results of RFE and PCA with each other as well as with previous results, the same approach of utilizing random forest was used again.

Table 2. Performance score comparison with RFE and PCA

	CV R^2	**CV RMSE**	**Test RMSE**
Elliptic Envelope / RF	0.58	3.69	4.04
Elliptic Envelope / RFE / RF	0.59	3.65	4.08
Elliptic Envelope / PCA / RF	0.51	4.02	5.41

Table 2 represents the effect of feature selection on the model performance. Accuracy of the model with Elliptic Envelope outlier detection was added to this table for a comparison purpose. As can be seen from the table, RFE did not not have a significant effect on model performance, however it did not decrease it either, while the number of features was reduced. It can be important in online model training, as reduced number of attributes can improve the model speed. On the other hand, PCA underperformed RFE, which was expected considering that the feature space was relatively normal. This approach is more suitable in the case where an especially high number of features with high collinearity interfere into model performance and should be controlled.

Next step was to tune Random Forest and GBM models to see how much further it could improve. GridSearhCV was used to automatize testing of a model with different settings.

The following piece of code in listing 7 provides an example of Grid Search configuration:

```
rf_reg = RandomForestRegressor()

params = [{'n_estimators': range(50,170,20),
     'max_depth': range(6,20,1),
     'bootstrap': [True, False],
     'max_features': [None, 'log2']
    }]
rf_grid_search = GridSearchCV(rf_reg, params, cv=10, refit='r2', scor-
ing=['neg_mean_squared_error', 'r2'])
rf_grid_search.fit(X_train, y_train)
```

Listing 7. Implementation of Grid Search with Random Forest Regressor in Python

First of all, the prediction model was chosen. Next, parameters to be tested were defined. After that, GridSearchCV was called with the following list of parameters: the estimator (rf_reg), parameters for the estimator, number of folds for cross validation and scoring measures. Both RMSE and R^2 are preferred metrics for cross validation, hence they were defined in an array. However, GridSearchCV can choose the most suitable parameters for an estimator based on only one metric, thus it was defined in a parameter called 'refit'. Finally, GridSearchCV was trained with training data. Essentially, it trained Random Forest with all combinations of parameters, and in order to access the estimator with the best parameters, `rf_grid_search.best_estimator_` can be called. This is a ready model that can be used to make further predictions. To view the chosen parameters, Grid search offers an attribute called `best_params_`, and to see the cross-validation results, `cv_results_` attribute can be used.

In the same way Gradient Boosting Regressor was trained, using its own set of parameters, displayed in listing 8.

```
params = [
    {'n_estimators': [60, 80, 100, 120, 140],
     'max_depth': range(3,8),
     'learning_rate': [0.01, 0.02, 0.05, 0.1, 0.5, 1.0],
     'loss': ['ls', 'lad', 'huber'],
    }
]
```

Listing 8. GBM parameters for Grid Search

Finally, in order to improve performance even further by enhancing accuracy for margin values, SMOTER (SMOTE for regression) was implemented. Listing 9 represents a SMOTER implementation in the project. `SmoteR` function takes in original dataset, upper

(`trH`) and lower (`trL`) thresholds, where upsampling is required, upsampling ratio (`rat`), and number of neighbors used in synthetic case generation (`k`). Next upsampling is performed and synthetic samples are returned. Originally, SMOTER also performs undersampling for the rest of the data to balance it out. However, for the purpose of the project undersampling was omitted in order to avoid excessive data shrinkage.

```
def knn_search(x, data, k):
    tree = spatial.KDTree(data)
    d, ind = tree.query(x, k=k+1)
    return data.iloc[ind][~(data.iloc[ind] == x).all(axis=1)].index

def oversample(data, rat, k):
    data_num = data.select_dtypes(include=[np.number])
    num_columns = data_num.columns
    newCases = pd.DataFrame(columns = data.columns)
    for i, case in data.iterrows():
        dists = [spatial.distance.euclidean(case[num_columns], r) for k,r in
data_num.iterrows()]
        max_d = np.median(dists)/2
        columns = case.index
        nns_ind = knn_search(case[num_columns], data_num, k)
        nns = data.loc[nns_ind]
        for p in range(1, rat):
            x = nns.iloc[random.randint(0,nns.shape[0]-1)]
            if spatial.distance.euclidean(case[num_columns], x[num_columns]) <
max_d:
                new = pd.Series(index=columns)
                for j in x.index:
                    if j != 'target':
                        if pd.api.types.is_number(x[j]):
                            diff = case[j] - x[j]
                            new[j] = case[j] + diff*random.random()
                        else:
                            new[j] = nns[j].mode()[0]
                d1 = spatial.distance.euclidean(new[num_columns].drop(la-
bels=['target']), case[num_columns].drop(labels=['target']))
                d2 = spatial.distance.euclidean(new[num_columns].drop(la-
bels=['target']), x[num_columns].drop(labels=['target']) )
                # weighted target value
                new['target'] = (d2*case['target'] + d1*x['target'])/(d1+d2)
                newCases = newCases.append(new, ignore_index=True)
    return newCases

def smoteR(data, trH, trL, rat, k):
    rareH = data[data['target'] > trH]
    rareL = data[data['target'] < trL]
    newCasesH = oversample(rareH, rat, k)
    newCasesL = oversample(rareL, rat, k)
    newCases = newCasesH.append(newCasesL)
    return newCases
```

Listing 9. SMOTER implementation in Python

Oversampling function, presented in the listing 9, takes in underrepresented partition of a dataset, upsampling ratio and number of neighbors used in synthetic sample

generation, passed from `smoteR` function. Next, dataset is stripped from non-numerical columns, as they cannot be used in calculating Euclidean distance. Synthetic samples are generated by taking one of dataset samples, one of its k neighbors, randomly picked, and randomly assigning each attribute a value between two chosen samples. For the categorical attributes a mode for all k neighbors for the current case is taken. In order to calculate target variable for a new synthetic case, Euclidean distance to two cases used in generation is calculated, and their distances are used as weights in target attribute calculation.

Lower and upper threshold were defined by approximately 5% of top and bottom of training data, which amounted to 56 samples in total. SMOTER algorithm, in its turn, generated 102 new samples. After training set was expanded with SMOTER, Grid Search for Random Forest and GBM was performed again. Table 3 lists the results for this test.

Table 3. Performance score comparison with / without SMOTER

	CV R^2	**CV RMSE**	**Test R^2**
Elliptic Envelope / RFE / Untuned RF	0.59	3.65	0.61
Elliptic Envelope / RFE / Tuned RF	0.68	3.32	0.69
Elliptic Envelope / RFE / Tuned GBM	0.69	3.27	0.69
Elliptic Envelope / RFE / SMOTER/ Tuned RF	0.81	2.8	0.77
Elliptic Envelope / RFE / SMOTER/ Tuned GBM	0.79	2.9	0.75

Table 3 shows a significant improvement in both tuning the model and applying SMOTER to the dataset. At the same time, there was no difference between the performance of Random Forest and Gradient Boosting Machine. From the test R^2 column we can see that the improvement with SMOTER was not as big as the CV R^2 column presents. It indicates that adding synthetic samples led to slight overfitting and might have been caused by using unrepresentative samples during the SMOTER process, which could introduce additional bias. As the gap between CV and test result is not wide, no action needed to be taken, however in general with increasing number of synthetic samples

overfitting also increases, so this factor should be taken into consideration. However, regardless of small dataset size, ensemble models such as Random forest and Gradient Boosting Regressor showed a good result of avoiding overfitting overall.

Finally, another method of increasing confidence in predictions is using prediction intervals. Three gradient boosting regressor models were trained for that purpose, where median was trained as usual, while upper and lower boundaries were trained with quantile loss parameter, and desired alpha parameter, setting confidence boundaries. To make prediction intervals with 90% confidence, lower alpha was set to 0.05 and upper alpha – to 0.95. Figure 17 visualizes results of quantile regression.

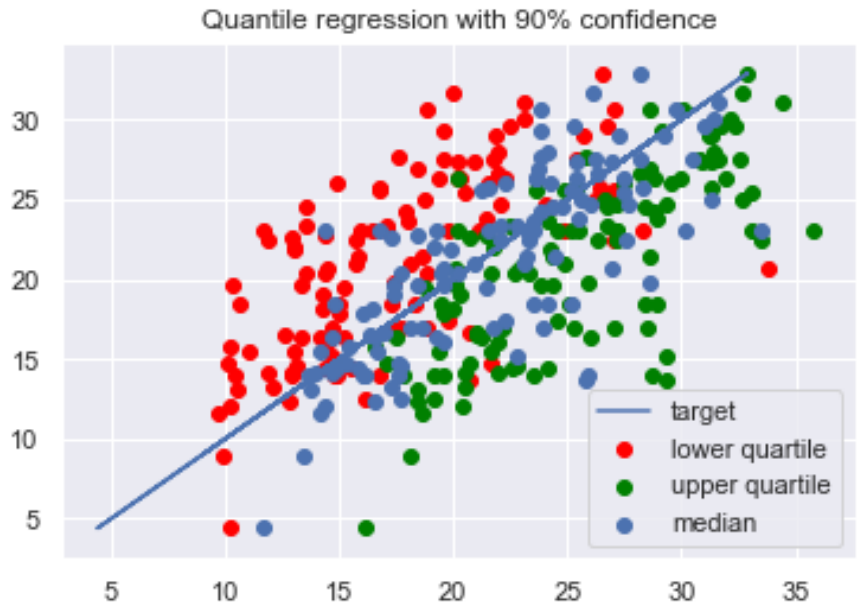

Figure 18. Prediction intervals with quantile regression

Maximum and minimum prediction values can be obtained from the plot presented on figure 18, as well as the median value, which is not different from gradient boosting regression results implemented earlier. Prediction intervals are lower than 15, which is a high value considering the target distribution. However, in certain cases prediction intervals with high confidence can be more valuable than a single value.

In fact, instead of prediction intervals, another custom metrics can be applied to determine the percentage of residuals falling into the acceptable range, and listing 10 presents, how it was implemented in the project.

```
residuals = (rf_pred - y_test).abs()
custom_metric = 100*(residuals < 7.5).mean()
print("Custom metric: {:,.2f}%".format(custom_metric))
```

Listing 10. Custom metrics implementation in Python

According to this measurement 97% of predictions from previously trained random forest regressor fell into ±7.5 from the target value. The main difference between these approaches is that prediction intervals are calculated based on desired confidence, while custom metrics uses acceptable range as a base instead, so the choice of a method depends on purpose.

5 Discussion

The techniques, tested in previous section, while improved performance significantly compared to the initial setup, still could not reach an objectively good performance that would be desired.

Initially a big part of the dataset was withheld for the research purpose – the idea was to improve the performance with a small dataset. However, this data was used for an experiment. Dataset was incrementally increased and used with an untuned Random Forest model without SMOTER application. The result of this experiment is presented on figure 19.

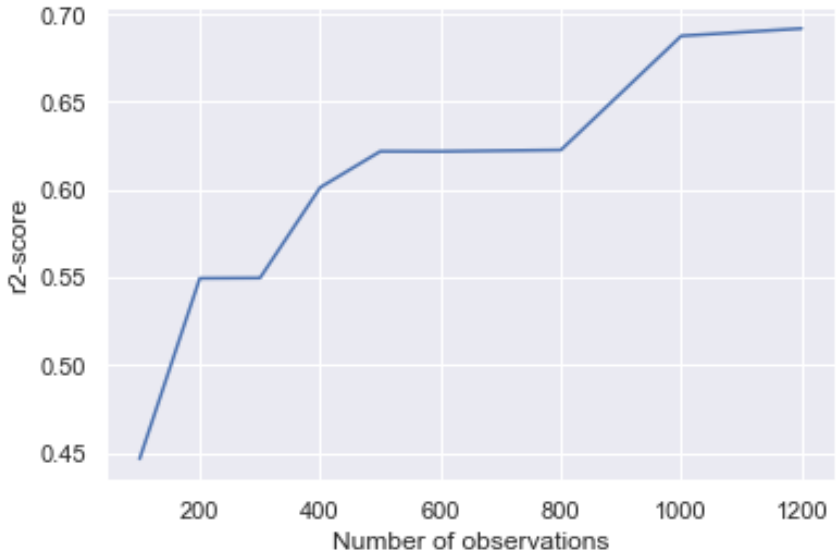

Figure 19. Model performance with different training set sizes

As can be observed from figure 19, the performance can be improved, by simply increasing dataset size. Size effect is well-known, but sometimes is underestimated. That being said, one of the approaches to consider in similar projects is to invest time into attempting to obtain more data. For instance, in the current project a lot of the observations were lost due to a missing target value. It could be avoided, if the process of filling in and collecting these values would be improved. Not only does it affect the size of the dataset, but also its quality, which leads to better results. Therefore, before attempting to improve the models it is important to make all the possible steps to obtain a better dataset.

6 Conclusion

The main goal of this thesis was to explore techniques improving predictive analysis performance with datasets lacking quality and size, as well as apply these methods to predict employee hour rates. Data obfuscation was performed for security purposes, in order to demonstrate the techniques in this paper.

The objective was achieved, all the steps to make predictions and improve the performance, found out in the research were made. Among the outlier detection techniques, LOF and Elliptic Envelope demonstrated good results, while Isolation Forest, surprisingly, failed. Ensemble models showed a great result of working with the small dataset prone to mistakes and overfitting. Moreover, synthetic minority over-sampling technique managed to improve the performance even further, showing the importance of sufficient representation of margin cases in the dataset.

References

1. Bonaccorso Giuseppe. Machine Learning Algorithms - Second Edition [e-book]. Packt Publishing; 2018

2. Kakade Sunil, Ozdemir Sinan. Principles of Data Science - Second Edition [e-book]. Packt Publishing; 2018

3. Harrison Matt. Machine Learning Pocket Reference [e-book]. O'Reilly Media, Inc.; 2019

4. Ozdemir Sinan, Susarla Divya. Feature Engineering Made Easy [e-book]. Packt Publishing; 2018

5. Jugulum Rajesh. Competing with High Quality Data: Concepts, Tools, and Techniques for Building a Successful Approach to Data Quality [e-book]. John Wiley & Sons; 2014

7. Massaron Luca, Boschetti Alberto. Python Data Science Essentials - Third Edition [e-book]. Packt Publishing; 2018

8. Zheng Alice, Casari Amanda. Feature Engineering for Machine Learning [e-book]. O'Reilly Media, Inc.; 2018

9. Géron Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition [e-book]. O'Reilly Media, Inc.; 2019

10. Torgo Luís, Ribeiro Rita, Pfahringer Bernhard, Branco Paula. SMOTE for Regression [conference paper]. Neural Networks to Predict Human Health Impacts of Traffic Emissions (pp.378-389); 2013
URL: https://www.researchgate.net/publication/257364616_SMOTE_for_Regression

11. Breiman Leo. Random Forests [article]. Machine learning. 2001, 45, pp 5-32
URL: https://link.springer.com/article/10.1023/A:1010933404324

12. Shrestha DL, Solomatine DP. Machine learning approaches for estimation of prediction interval for the model output [article]. Neural Networks. 2006 Mar 1;19(2):225-35.
URL: https://www.sciencedirect.com/science/article/abs/pii/S0893608006000153

13. Chen Q, Meng Z, Liu X, Jin Q, Su R. Decision Variants for the Automatic Determination of Optimal Feature Subset in RF-RFE [article]. Genes. 2018 Jun;9(6):301.
URL: https://www.researchgate.net/publication/325800934_Decision_Variants_for_the_Automatic_Determination_of_Optimal_Feature_Subset_in_RF-RFE

14. Darst BF, Malecki KC, Engelman CD. Using recursive feature elimination in random forest to account for correlated variables in high dimensional data [article]. BMC genetics. 2018 Sep;19(1):65.
URL: https://bmcgenet.biomedcentral.com/articles/10.1186/s12863-018-0633-8

15. Chio C, Freeman D. Machine Learning and Security: Protecting Systems with Data and Algorithms [e-book]. O'Reilly Media, Inc.; 2018

16. Yu A, Chung C, Yim A, Dangeti P. Numerical Computing with Python [e-book]. Packt Publishing; 2018

17. Pace L, Beginning R: An Introduction to Statistical Programming [e-book]. Apress; 2012

18. Fuentes A, Hands-On Predictive Analytics with Python [e-book]. Packt Publishing; 2018

Isolation forest scheme