Machine Learning Engineer Nanodegree Capstone Project

Dario Culig-Tokic
August 4th, 2018

Contents

I. Project Definition

Project Overview

The project is based on a Kaggle [1] competition Santander Value Prediction Challenge [2]. Santander Bank is asking Kagglers to help them identify the value of transactions for each potential customer.

This is a first step that Santander needs in order to personalize their services. According to Epsilon research, 80% of customers are more likely to do business with you if you provide a personalized service. Banking is no exception. The digitalization of everyday life means that customers expect services to be delivered in a personalized and timely manner. Often before, they have even realized they need the service. Santander Bank aims to go a step beyond recognizing that there is a need to provide the customer a financial service, and intends to determine the amount or value of the customer's transaction. This means anticipating customer needs in a more concrete, but also simple and personal way. With so many choices for financial services, this need is greater now than ever before.

Banking has been one of early adopters of machine learning due to large amount of data at their disposal and complexities involved in the day-to-day business. One of the paramount problems is predicting the value of transactions for each potential customer. This is a basis on which other personalized services can be provided. Value of transactions is a continuous variable and its prediction is a regression problem.

The data is publicly available through Kaggle website and is provided by Santander Bank. It has been already separated into training and testing datasets within a train.csv and test.csv files. Data is anonymized.

Training dataset contains a target variable (value of transition per customer) and 4911 feature variables. There are 4459 train data points and 49342 test data points. One has to use only the data provided within the train.csv file to train the regression algorithm. Use of other external source will result in disqualification.

Feature variables also come with no description and with names like “48df886f9” that provide little meaning as to what they represent. One can guess that they are possibly aggregated transactions (by day, by week, by month…) or an individual transaction which could be further grouped by some financial instrument (debit, credit, loan, …).

Problem Statement

This project aims to solve a high-dimensional sparse regression problem. The variable that we attempt to predict is value of transactions for each potential customer from 4991 feature variables. The overall error of a predictor variable will be evaluated using several error metrics.

There are a number of well know regression algorithms within the field of machine learning. The algorithms we will examine are LinearRegression (ordinary linear regression), Lasso (linear regression with L1 regularization), Ridge (linear regression with L2 regularisation), ElasticNet (linear regression with L1 and L2 regularisation), KNeighborsRegressor (regression based on k-nearest neighbours), MLPRegressor (multi-layer neural network regression) within the ScikitLearn library and LightGBM (gradient boosted decision tree regression) which is a gradient boosting decision tree algorithm. Each one of these algorithms can be used to find a solution to the above stated problem.

The evaluation metric used is RMSLE - Root Mean Squared Logarithmic Error. It is defined in equation (1).

$$e = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\log(p_i + 1) - \log(a_i + 1)\right)^2} \tag{1}$$

Where:

e is the RMSLE value
n is the total number of observations in the dataset
p_i is the prediction of the target variable
a_i is the actual target
log(x) is the natural logarithm of x

Evaluation metric RMSLE is usually used when you don't want to penalize huge differences in the predicted and the actual values when both predicted and true values are large numbers. The RMSLE penalizes the under estimate more than over estimate. It was defined as the competition error metric by the competition sponsor.

In some algorithms, other error metric are used such as RMSE - Root Mean Squared Error that are already predefined within the algorithm. RMSE - Root Mean Squared Error is defined in equation (2):

$$e = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2} \tag{2}$$

Where:

e is the RMSE value
n is the total number of observations in the dataset
p_i is the prediction of the target variable
a_i is the actual target

Benchmarking our model in Kaggle environment is straightforward. In Kaggle environment, after we train our algorithm on the training data, we generate the predication for the testing data.

The predicted dataset is then uploaded to the Kaggle website and compared to the true target variable. As a feedback, one receives an error score. This error score can be compared to error scores of other participants. In such a way, we can get a valuable feedback about the true error our algorithm makes.

The project was competed using a workflow generated at the start of the project. During the project, additional sections have been explored but the core workflow remained the same. The individual steps of the workflow follow the general layout of a machine leaning project.

The project will involve following steps:

Data Exploration

- Target statistics
- Feature statistics
- Sparsity of the data

Data Visualization

- Target visualization
- Feature visualization
- Target correlation visualization
- Feature correlation visualization

Data Preprocessing

- Removing columns with no variation
- Applying natural log function to target variable
- Applying a natural log function to features

Regression Algorithm Exploration

- Naïve Predictor
- Linear Regression
- Lasso
- Ridge
- ElasticNet
- MLPRegressor
- LightGBM
- Selecting the most promising algorithm

Feature Engineering and Selection

- Dimensionality reduction using PCA - Principal Component Analysis, RSP – Random Sparse Projection, FastICA – Fast Independent Component Analysis, tSVD – Truncated Single Value Decomposition, NMF - Non-Negative Matrix Factorization
- Feature selection using univariate linear regression test
- Feature expanding by adding a mean, standard deviation, and other statistical function for each row
- Feature selection using feature importance
- ✓ Algorithm Testing

Algorithm Tuning

- Random search through parameter space
- Selecting the best algorithm
- ✓ Algorithm Testing

Final Score Report

- Cross-validation score
- Leadership board score

II. Analysis

Exploratory Data Analysis

The data has been separated into a training and testing datasets by the Santander Bank. Training dataset contains a target variable (value of transition per customer) and testing dataset does not. There are 4459 train data points and 49342 test data points. Datasets contain 4911 feature variables.

Target variable and Features

Statistical description of target variable provided by the Pandas [3] library summary function can be seen in Table 1. The target variable has been transformed using a natural logarithm as explained in the next paragraph.

Table 1. Target variable distribution and transformation

Function	Target Value
count	4,459.00
mean	5,944,923.00
std	8,234,312.00
min	30,000.00
25%	600,000.00
50%	2,260,000.00
75%	8,000,000.00
max	40,000,000.00

Logarithm transform $\Rightarrow$

Function	Target Value
count	4,459.00
mean	14.49
std	1.75
min	10.31
25%	13.30
50%	14.63
75%	15.89
max	17.50

The distribution of the target variable can also be seen on a distribution plot [Figure 1].

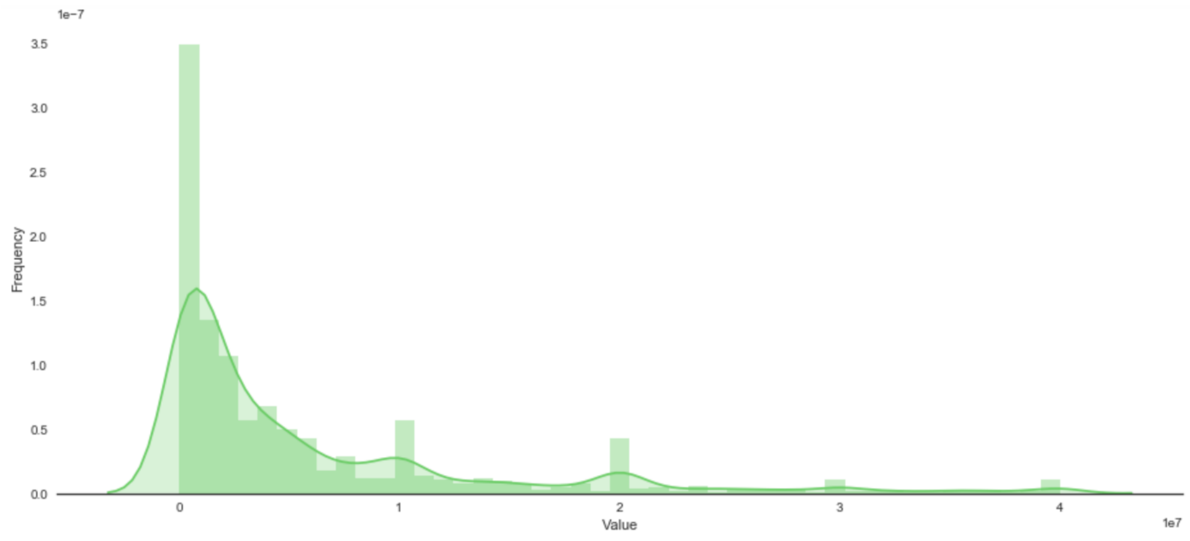

Figure 1. Target variable distribution

As we can see the target variable distribution is not normal. It seems to be a log-normal distribution. Given the distribution of the target variable we will perform a natural logarithm transformed in the preprocessing step to make it more Gaussian which is beneficial for training of most machine learning algorithms. Figure 3. shows target variable distribution after natural logarithm transformation.

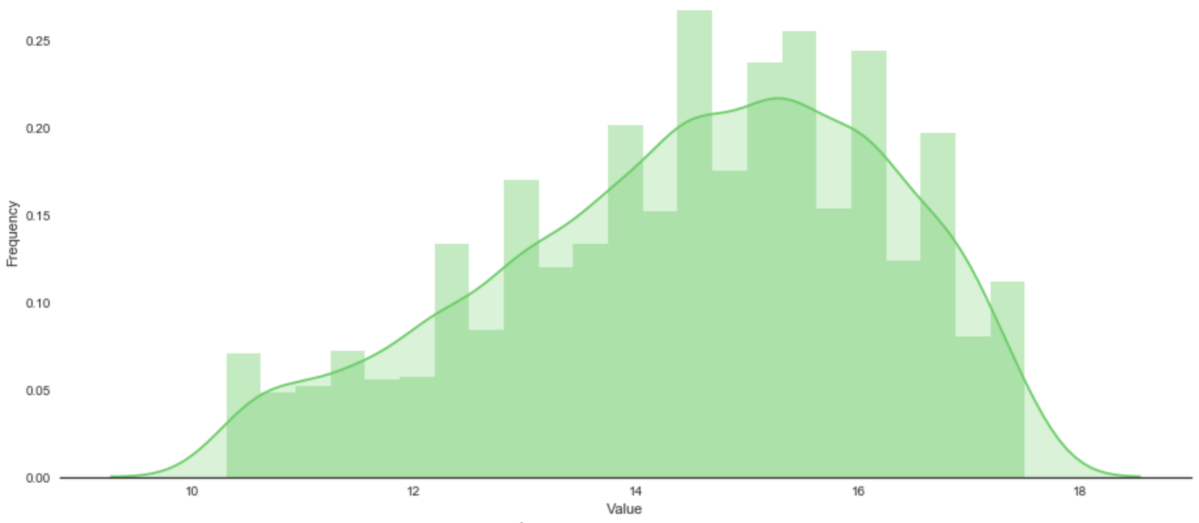

Figure 2. Transformed target variable distribution

We can now take a look at the features. Figure 3. shows the values of 7 random features belonging to training dataset. As can be seen on the Figure 3. feature data are very sparse.

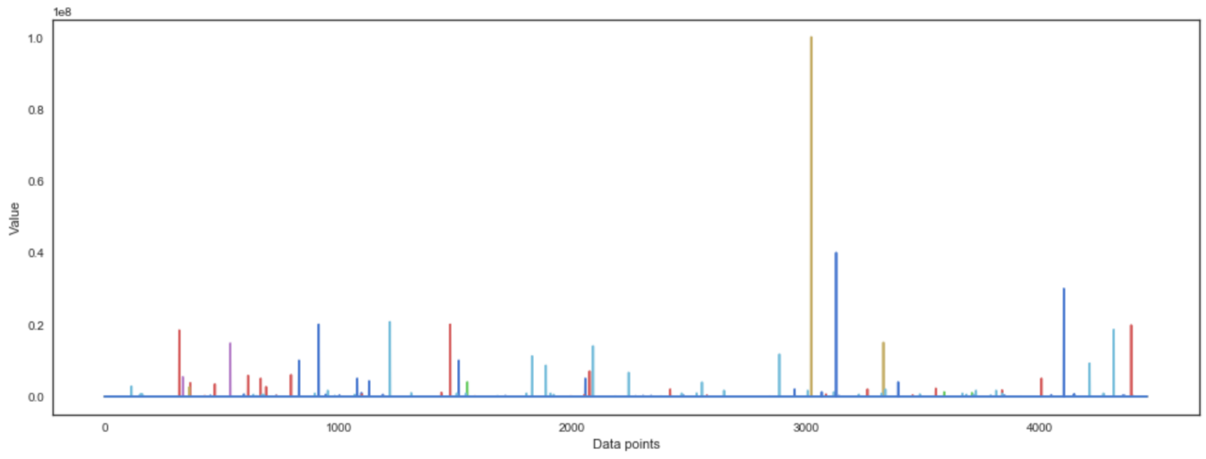

Figure 3. Distribution of several features

One usually preforms mean and variance centering steps during preprocessing of the training data but in the case of sparse data that is not recommended. We will instead perform a natural logarithm transform. After the transform the feature values features are plotted on Figure 4.

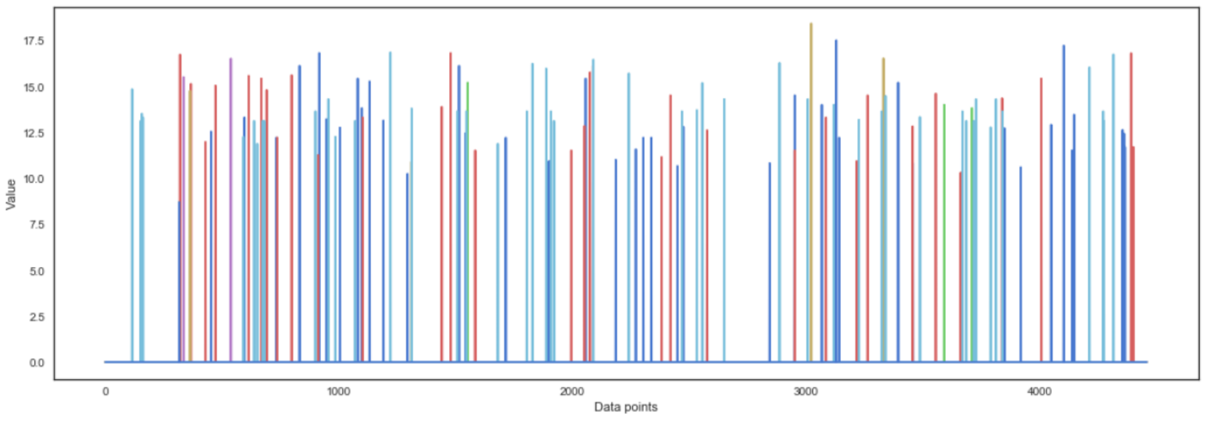

Figure 4. Distribution of several transformed features

As can be seen from above examples the data is sparse. There are in total 22,254,869 values out of which 21,554,760 are zeros. The data sparsity is 96.85%.

We can calculate and plot the distribution of zero elements both column-wise and row-wise to get some additional information about the data. Figure 4 shows the distribution of the zero values column-wise in the training dataset.

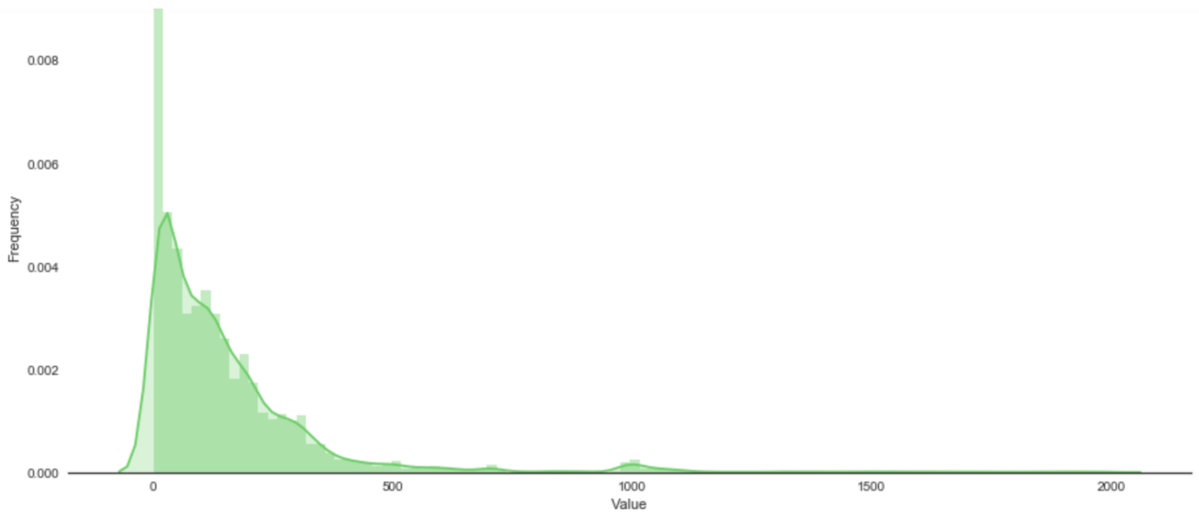

Figure 5. Distribution of row data sparsity

We can also apply some dimensionality reduction technique like PCA - Principal component analysis and then plot the data in two or more dimensions. This will help us get a better feel for the data structure.

Figure 6. Show the scatter plot of the first two PCA components.

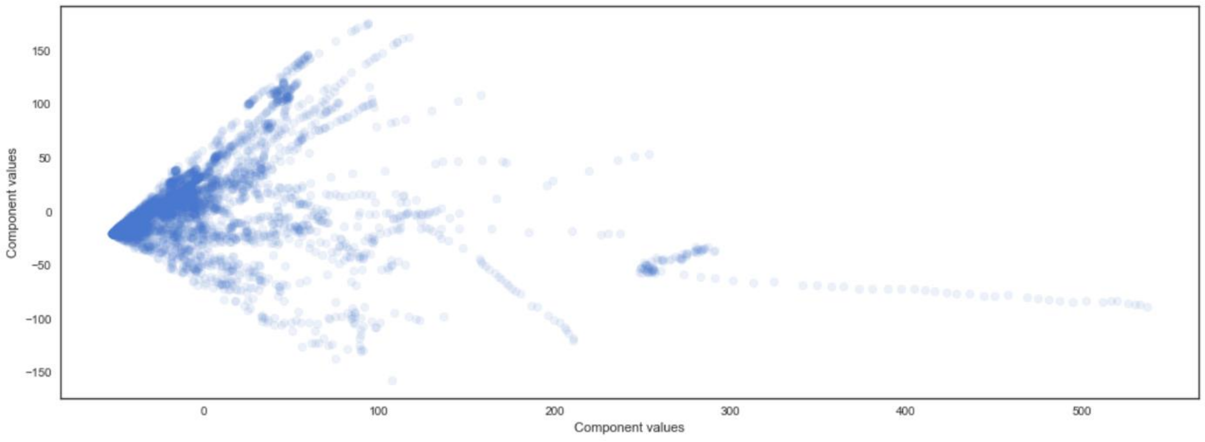

Figure 6. First two components of Principle Component Analysis

The above examples of PCA generate a relatively dense matrix from the data. Given that we have sparse data source we can also use SRP - Sparse Random Projection algorithm to output a sparse matrix. Figure 7. Show the scatter plot of the first two SRP components.

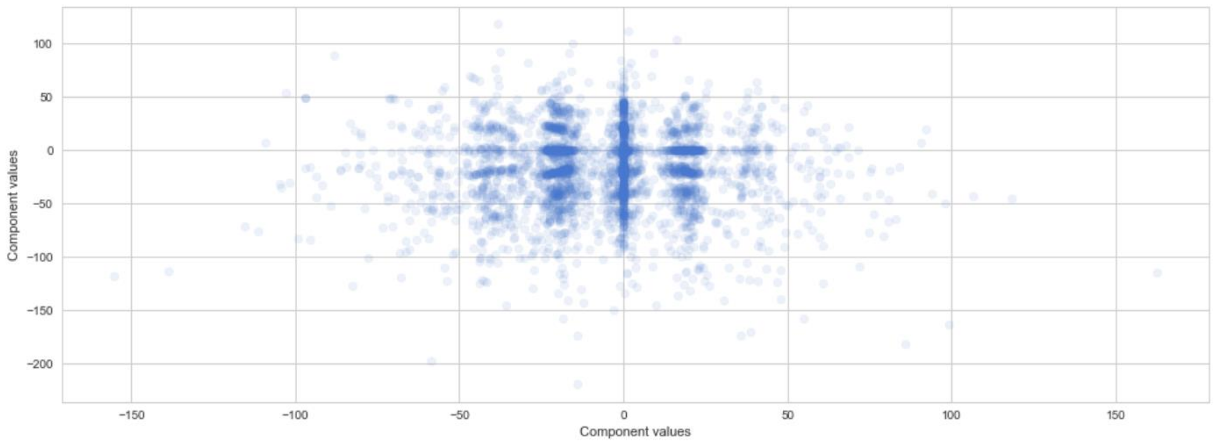

Figure 7. First two components of Sparse Random Projection

First 40 Sparse Random Projection components can be visualized using Seaborn [4] heatmap across all of the datapoints. Datapoint values have been scaled to between 0 and 1.

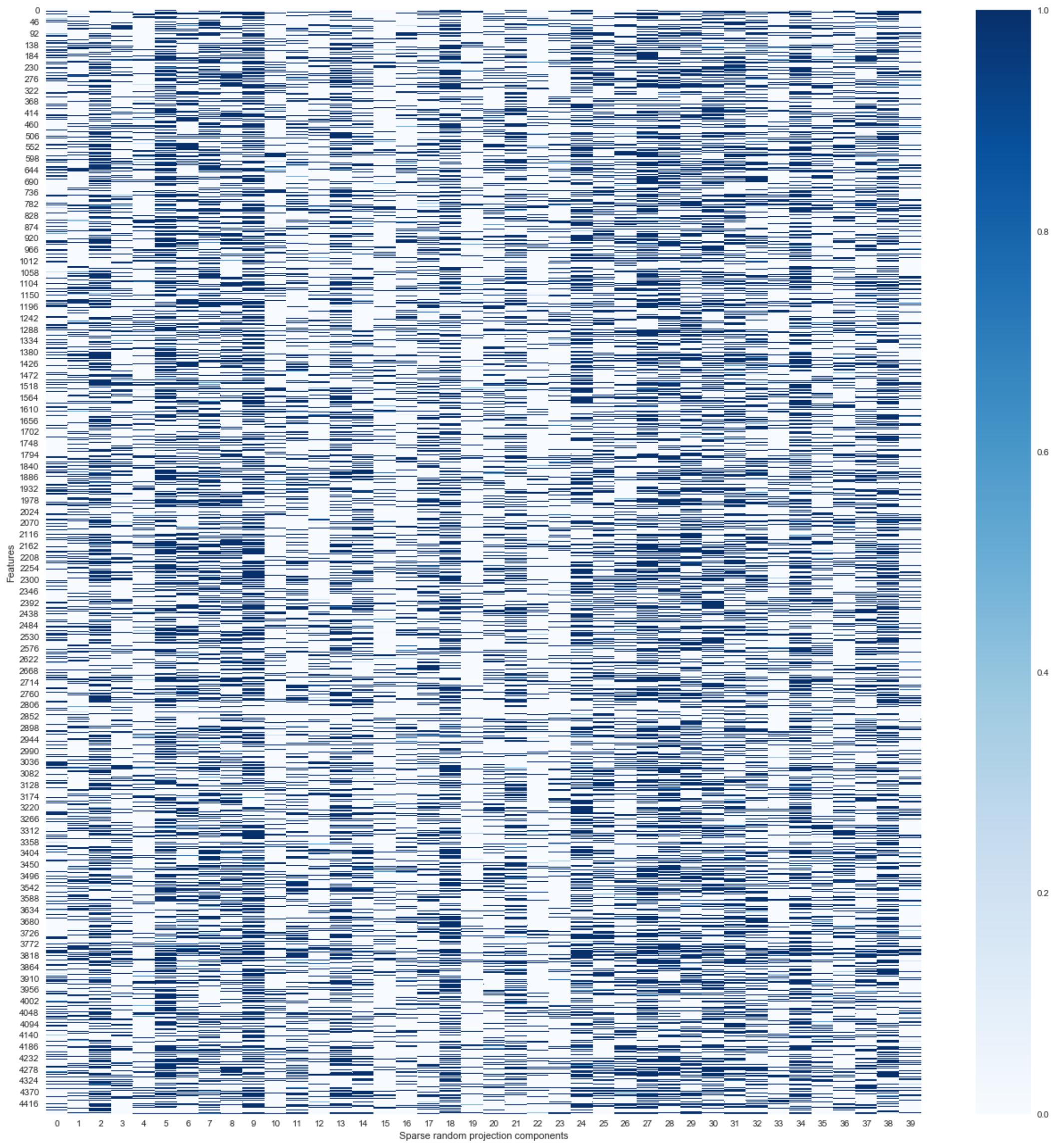

Figure 8. First 40 components of sparse random projection across all data points

The Sparse Random Projection algorithm produced a much sparser matrix than PCA. In the feature engineering section we will use both to expand our dataset.

Correlation between features and target variable

Mutual information regression

Correlation between features and target variable can be estimate using mutual_info_regression function from SciKit Learn library. Mutual_info_regression is suitable metric for sparse data. Mutual information between two random variables is a non-negative value, which measures the dependency between the variables. It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency. Figure 5. Shows the correlation between features and the target variable using mutual_info_regression function. The features have been sorted by values of mutual information values.

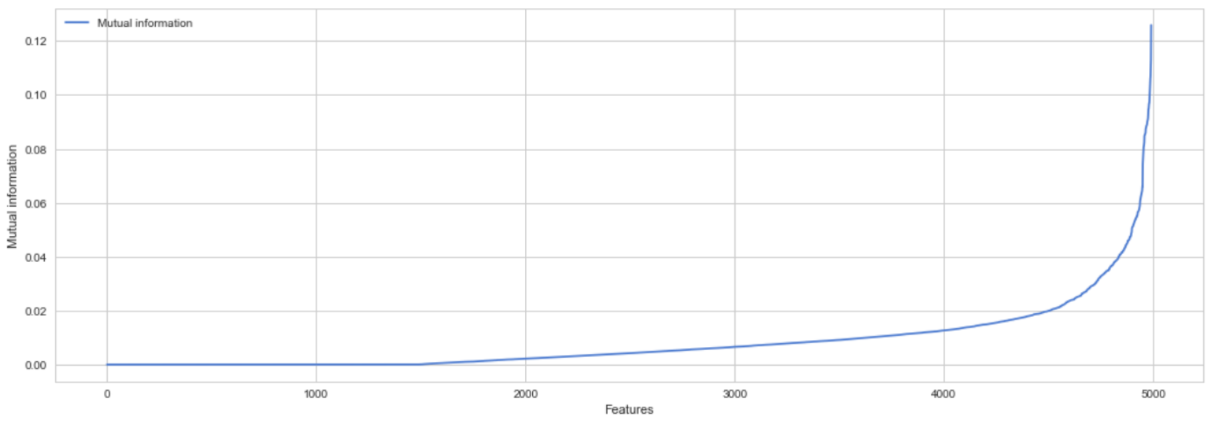

Figure 9. Mutual information between features and targe variable

As we can see from Figure 9 some features have a strong correlation with the target variable while many have almost no correlation. This information can be used when preforming dimensionality reduction in feature engineering section.

Univariate linear regression test

We are also going to use univariate linear regression test to investigate the similarity between features and target variable. The sorted results of the analysis are show on Figure 8.

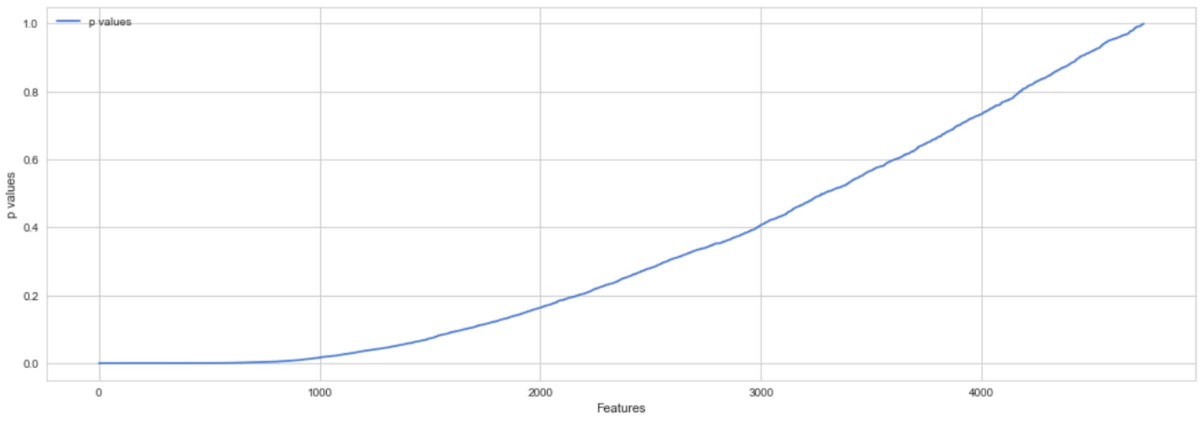

Figure 10. Univariate linear regression test between features and target variable

This test also reveals that there are many variables with 0 correlation with target variable. But there are quite a few that correlate relatively well. We will make a list of all these features and their correlation with the target value which will be used in feature engineering.

Correlation between features

High correlation between features is not a desirable property of training data for machine learning algorithms. Highly correlated features causes the algorithm to relay to heavily on them compared to the other features and thus can result in overfitting. It is necessary to remove these features from the dataset if there are any. One way to test this is by using the pandas correlation function that calculates the correlation between all individual features. Features with high correlation (above 0.99) have been identified. These features should normally be removed but we did not removed them here because we used dimensionality reduction and feature selection in feature engineering section.

Algorithms and Techniques

The used algorithms have already been defined in both the Capstone Proposal and in in the Problem Statement chapter. Here we will go through each of the algorithms and state their main characteristic and intended use. In general, it is useful to try several different machine learning algorithm on a single problem to get a better intuition what works and what does not work.

LinearRegression is a SciKit Learn [5] library ordinary least squares regression algorithm that is estimating the target variable with equation (3).

$$y_i \approx \beta_0 + \sum_{j=1}^{p} \beta_j X_{ij} \tag{3}$$

In linear regression, we assume that the response is well modeled as a linear combination of the features. The model is parameterized by an intercept $\beta_0 \in R$ and a vector of weights $\beta \in R^p$, where p is the number of features.

Let us assume that we have n data points consisting of a value of the target a_i and the corresponding values of the features $X_{i1}, X_{i2}, \ldots, X_{ip}$, $1 \leq i \leq n$.

The algorithm is minimizing the square difference between the target variable and the predicted variable, which is defined by the equation (4).

$$f(MSE) = \min\left(\frac{1}{2n}\sum_{i=1}^{n}\left(a_i - \beta_0 + \sum_{j=1}^{p}\beta_j X_{ij}\right)^2\right) \tag{4}$$

where a_i is the actual target variable.

Lasso is a SciKit Learn library ordinary least squares regression algorithm with L1 regularization that is estimating the target variable with equation (3) same as LinearRegression algorithm.

The algorithm is minimizing the square difference between the target variable and the predicted variable, which is defined by the equation (5).

$$f(\mathrm{MSE}) = \min\left(\frac{1}{2n}\sum_{i=1}^{n}\left(a_i - \beta_0 + \sum_{j=1}^{p}\beta_j X_{ij}\right)^2 + \lambda|\beta_j|\right) \tag{5}$$

Variable λ is a regularization parameter that controls the trade-off between the fit to the data and the L1 norm of the weights. If λ is very small, then the lasso estimate is similar to the least-squares estimate. For values in between, the lasso yields a sparse model containing the coefficients corresponding to the relevant features. L1 regularization is used to prevent overfitting to the training dataset.

Ridge is a SciKit Learn library ordinary least squares regression algorithm with L2 regularization that is estimating the target variable with equation (3) same as LinearRegression algorithm.

The algorithm is minimizing the square difference between the target variable and the predicted variable, which is defined by equation (6).

$$f(\mathrm{MSE}) = \min\left(\frac{1}{2n}\sum_{i=1}^{n}\left(a_i - \beta_0 + \sum_{j=1}^{p}\beta_j X_{ij}\right)^2 + \lambda|\beta_j|^2\right) \tag{6}$$

If a subset of the predictors are highly correlated, then some of the singular values will have very small values, which results in noise amplification. Ridge regression is an estimation technique that controls noise amplification by introducing an L2-norm penalty on the weight vector. Variable λ is a regularization parameter that controls the trade-off between the fit to the data and the L2 norm of the weights.

ElasticNet is a SciKit Learn library ordinary least squares regression algorithm with L1 and L2 regularization that is estimating the target variable with same equation (3) as LinearRegression algorithm.

The algorithm is minimizing the square difference between the target variable and the predicted variable, which is defined by the equation (7):

$$f(\mathrm{MSE}) = \min\left(\frac{1}{2n}\sum_{i=1}^{n}\left(a_i - \beta_0 + \sum_{j=1}^{p}\beta_j X_{ij}\right)^2 + \lambda_a|\beta_j| + \lambda_b|\beta_j|^2\right) \tag{7}$$

Variable λ_a and λ_b is a regularization parameter that controls the trade-off between the fit to the data and the L2 norm of the weights. They can be tuning to minimize overfitting of the algorithm to the training dataset.

KNeighborsRegressor is a SciKit Learn library regression algorithm based on k-nearest neighbors. The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set. All points in each neighborhood can be weighted equally or they can be scaled so that closer neighbors of a query point will have a greater influence than neighbors which are further away.

MLPRegressor is a SciKit Learn library regression algorithm based on multi-layer perceptron network (artificial neuron network). Multi-layer perceptron network is a supervised learning algorithm that learns a mapping function (from features to target variable) by training on a dataset. Given a set of features X = {x_1, x_2, ..., x_m} and a target y, it can learn a non-linear function approximator for either classification or regression. It can have, in that between the input and the output layer, one or more non-linear layers, called hidden layers. The leftmost layer, known as the input layer, consists of a set of neurons {x_1, x_2, ..., x_m} representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1 + w_2x_2 + ... + w_mx_m$, followed by a non-linear activation function like the RELU – Rectified linear unit or hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.

LightGBM is a gradient boosted decision tree regression algorithm. Gradient boosting algorithms builds an additive model in a forward stage-wise fashion and it allows for the optimization of arbitrary differentiable loss functions. LightGBM uses a novel technique of Gradient-based One-Side Sampling (GOSS) to filter out the data instances for finding a split value.

All of the mentioned algorithms can take input features as Numpy [6] ndarray matrix and target as numpy array vector. Most of the algorithms have various hyperparameters that can be tuned to optimize the performance of the algorithm.

Benchmark

First threshold that was calculated was using the naïve predictor using mean target value for all data points. The RMSE calculated by naïve predictor is 1.75. All of the algorithm should perform better than the set benchmark. If that is not the case than that algorithm is either not well suited for the problem at hand, the data is not preprocessed correctly for that algorithm or the hyperparamers of the algorithm are set improperly. The data itself could also be of such complexity or lack of correlation that it very difficult to model.

Benchmark can be also set using the Kaggle website were the one can see the leadership board of people participating in the competition. Currently the best score on the leadership board is 1.36 using unscrambled data [7, 8]. It is important to mention that this score is on test dataset for which no target variable was provided and can only be estimated though Kaggle website. The cross-validation score on the training dataset is likely lower than the reported 1.36.

III. Methodology

Data Preprocessing

As already discussed in the exploratory data analysis section the target variable is not normally distributed. We have thus preformed a natural logarithm transformation of the target variable.

Features constitute a sparse high-dimensional matrix. Considering the sparsity of the data, standard preprocessing (mean and variance centering) is not advised so features have also been transformed using a natural logarithm. (Before applying the natural logarithm one was added to each data point to avoid taking the logarithm of number 0)

Some of the features have zero variance so they have been removed.

Several features have identical values and copies of these features have been removed.

Feature Engineering

Decomposition algorithms

Several different dimensionality reduction techniques have been used on the data. For each technique a LightGBM algorithm has been fitted sequentially on different numbers of the generated components to find the ones most beneficial.

The Principal Component Analysis component test is shown in Figure 11.

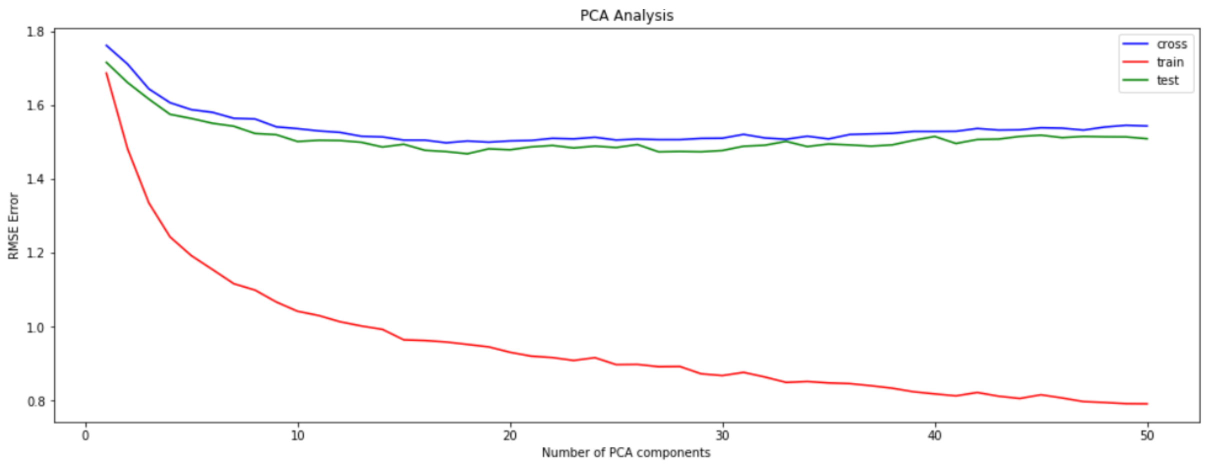

Figure 11. Testing PCA components using LightGBM

The Figure 11. Shows that there is no decrease in cross-validation error after 20 components. The RMSE error actually starts to increase slightly.

The following decomposition components have been identified to produce the best results:

Table 2. Decomposition algorithms and selected number of components

PCA - Principal Component Analysis	First 20 components produce the lowest error on cross-validation with smallest overfitting.
RSP – Random Sparse Projection	First 75 components produce the lowest error on cross-validation with smallest overfitting.
tSVD – Truncated Single Value Decomposition	First 20 components produce the lowest error on cross-validation with smallest overfitting.
FastICA – Fast Independent Component Analysis	Did not produce good results and so was omitted from further analysis.
NMF - Non-Negative Matrix Factorization	First 40 components produce the lowest error on cross-validation with smallest overfitting.

Components that have been shown to best represent the data have been concatenated and will be used to expand the dataset. FastICA was not used due to poor results while tSVD was not used due to high correlation to already identified PCA components.

Feature Selection

We can search for the optimal number of features using univariate feature regression test from SciKit Learn library. First we calculate F1 score between the target variable and our features sort them by correlation and then select subsets of feature and test them using the LightGBMT algorithm. The analysis has been shown on Figure 12.

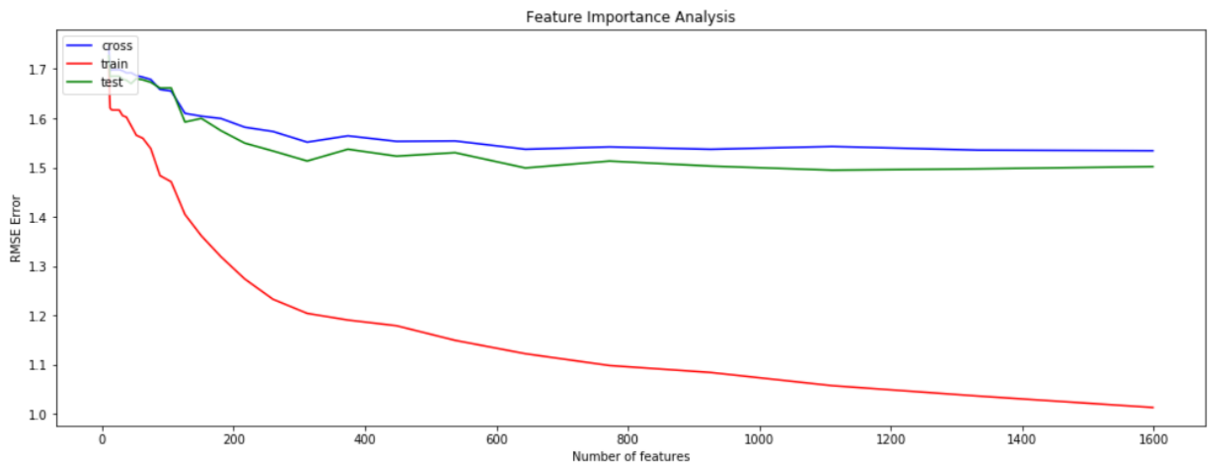

Figure 12. Feature selection using univariate feature regression

From the above diagram, we can see that more than 800 features do not contribute much to the explanatory power of the algorithm. We can thus use this information to reduce the number of features from the original dataset to 800 features.

Statistical Data Expansion

The data was also expanded row-wise using various statistical function like mean, standard deviation, min and max values, sum of the row, skewness, kurtosis, moment around mean, interquartile values from SciPy [9] library for each row. In total 10 new features were added.

Data Concatenation

The concatenated data set comprises form the preprocessed data with 800 features selected using univariate feature regression test and concatenated with the 135 selected decomposition components and 10 statistical row-wise expansion features. The dataset than has 945 features.

Dimensionality Reduction

Additional feature selection was performed using LightGBM algorithm. After fitting the algorithm, it can calculate feature importance based on the training data.

This has been used to try 30 different subsets of features and train and cross-validate a new algorithm on each subset of features. The resulting analysis has been plotted on Figure 13.

On Figure 13. it can be seen that the optimal number of features seems to be 150. After 150 feature not even the training error continuous to decrease. The dataset has been transformed so that it includes only these first 150 features.

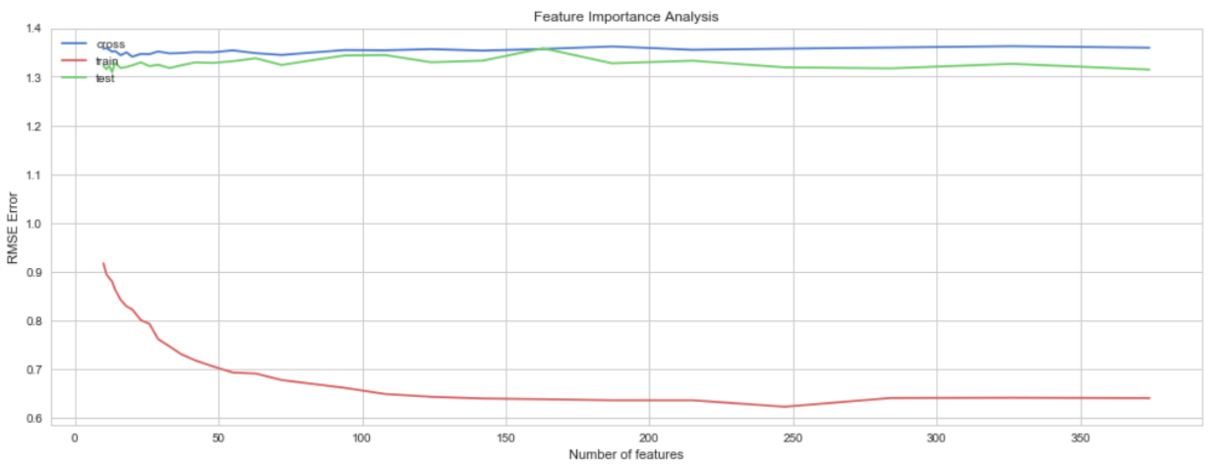

Figure 13. Feature importance analysis

Implementation

After the preprocessing the data several algorithms were tested to find the most promising candidate algorithm before the final tuning.

Linear regression algorithm cannot model the data well. The error is worse even compared to the naïve predictor with RMSE error metric of 1.75 using mean as a prediction for all data points. Lasso, Ridge and ElasticNet are all algorithms based on linear regression with regularization. Lasso with L1 regularization, Ridge with L2 regularization and ElasticNet with L1 and L2 regularization. These algorithms preform considerably better than the ordinary linear regression. The Lasso algorithm has a cross-validation error of 1.68 same as ridge repressor. The EasticNet algorithm error is similar at 1.69. KNeighborsRegressor performed pretty badly as well having the test error higher than the naive predictor. The MLPRegressor also has even worse cross-validation error then the naïve predictor. The LightGBM algorithm has the lowest cross validation error of 1.46.

For all the algorithms tested, a simple tuning of hyperparameters was performed to get a sense of the algorithms ability to model the data. Most algorithm performed pretty badly compared even to the naïve predictor even after algorithm tuning and data preprocessing. This indicates the high complexity of the data at hand.

The analysis preformed above shows that the most promising algorithm for this task is LGBMRegressor. All of the following analysis were made with LightGBM algorithm.

Feature engineering was performed as described in the chapter Feature engineering. During the feature engineering the Light GBM algorithm was used to find the best set of features to use in the final dataset.

The LGBMRegressor cross-validation RMSE error metric after feature engineer is 1.36. After the final dataset has been established LightGBM hyperparameters have been tuned using random search.

Algorithm Hyperparameter Tuning

The Light GBMT algorithm was selected for detailed parameter tuning. The algorithm has many hyperparameters which can be tuned in wide range of values. The tuning was performed using random search in 100 steps through predefined parameters. The predefined parameters can be seen in the code appendix. The search resulted in the algorithm scores ranging from 1.33 to 1.75 which can been seen on Figure 14.

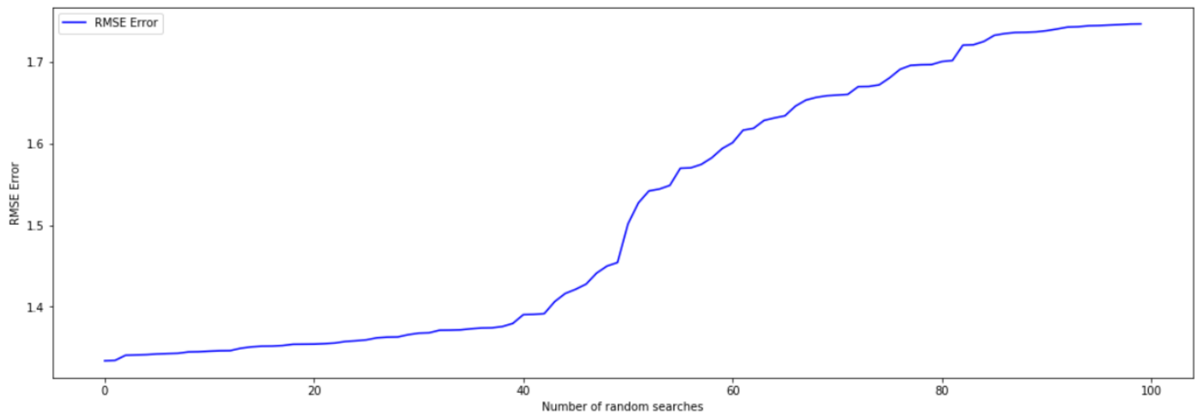

Figure 14. Hyperparameter tuning with random search

After the random search, the optimal algorithm was found:

LGBMRegressor(bagging_fraction=0.84, bagging_freq=5, boosting_type='gbdt', class_weight=None, colsample_bytree=1.0, feature_fraction=0.76,is_training_metric=True, lambda_l1=0.4, lambda_l2=0.15, learning_rate=0.01, max_bin=250, max_depth=11, metric='rmse', min_child_samples=20, min_child_weight=0.001, min_data_in_leaf=28, min_split_gain=0.0, n_estimators=1000, n_jobs=-1, num_iterations=435, num_leaves=110, objective='regression', random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True, subsample=1.0, subsample_for_bin=200000, subsample_freq=0, verbose=-1)

The tuned algorithm RMSE cross-validation error is 1.33. This is an improvement from the cross-validation score of 1.36.This algorithm was subsequently used in the final prediction.

IV. Results

Model Evaluation and Validation

The model cross-validation error is 1.33 on the data for which target variable is available. The predefined benchmark was set at 1.75 using a naïve predictor that uses mean target value for all data points. Another benchmark is the Kaggle Leadership Board score of 1.36.

After the prediction on the true test data (for which target variable is not available) the result was uploaded to Kaggle website for verification. The error on the test dataset is 1.41.

The increase in the error on the unseen data is reasonable considering that we had to train our data on only 4459 train data points and the test was performed on 49342 test data points. In conclusion, the algorithm generalizes to unseen data well and is robust.

However, the error is still relatively high compared to the naïve predictor which points to initial data complexity and potentially to still unexplored feature engineering or algorithm selection and tuning to produce a lower error and better fit to the data.

The solution presented here is based on the anonymized data and its precise meaning and categorization so it is difficult so say with certainty how much better the solution could be if one knew more about the data.

Predicted values for the training set and corresponding target values are shown on Figure 15. The values were previously sorted from smallest to largest.

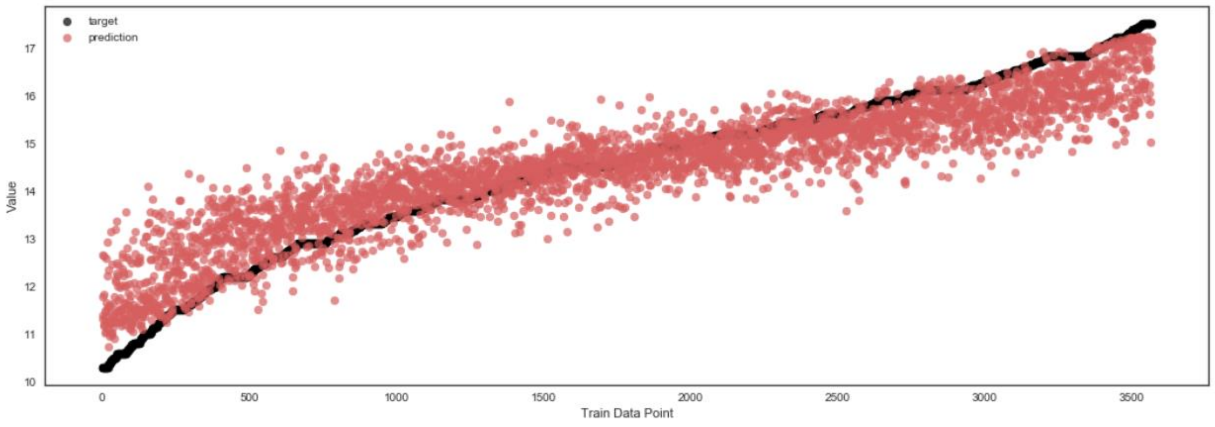

Figure 15. Traing data predictions

We can see that even on the training data set the algorithm is still not preforming that well. Especially on the data smallest and largest target values. This is due to initial data complexity and preventing overfitting on test data.

Predicted values for the test set and corresponding target values are shown on Figure 16.

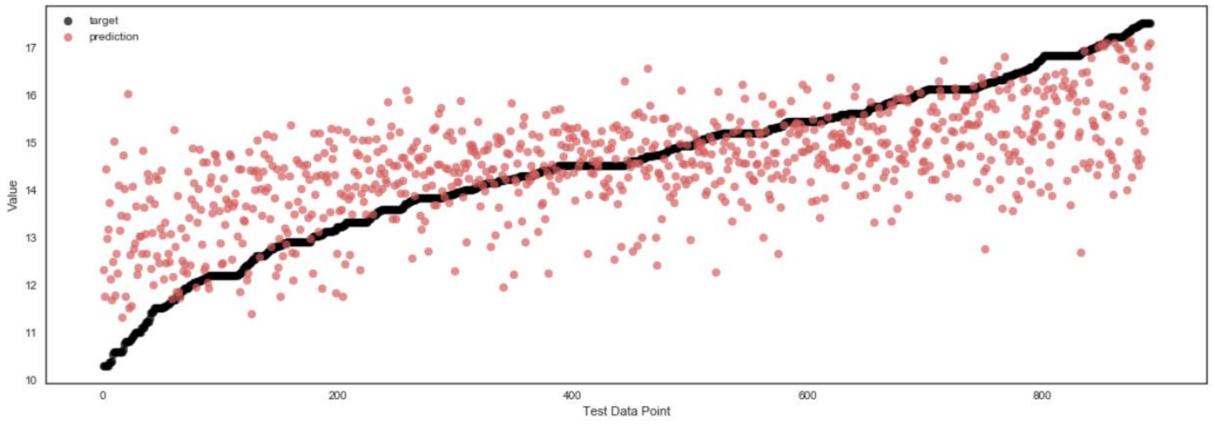

Figure 16. Testing data predictions

The testing data prediction plotted against the true target data shows the true complexity of the problem at hand. After all the steps involving data preprocessing, algorithm selection, feature selection and algorithm tuning the best prediction from the available features is one shown on Figure 16. Some of the predicted data points are quite far from the true target values. The prediction did capture the overall data trend but the errors especially near smallest and largest target values are quite substantial.

One can easily reduce the training error of the model by decreasing the min_data_in_leaf hyperparameter. After the change of the parameter the predicted values for the training are shown on Figure 17. However the final cross-validation error increases from 1.33 to 1.34.

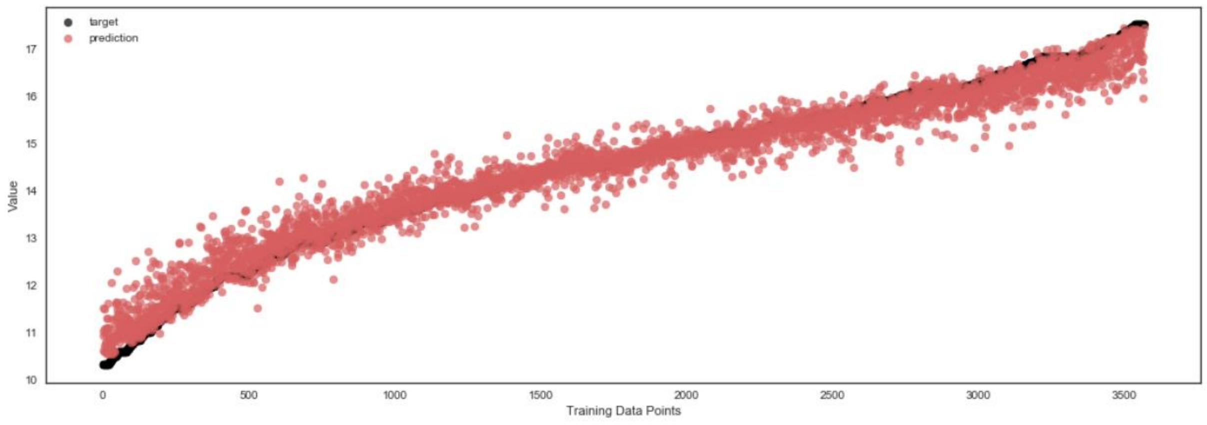

Figure 17. Reduction of minimal data in leaf

This is the main challenge of every machine learning task, learning as much as possible from the training data without overfitting. This means that the algorithm must not learn random noise which does not truly represent the data or it will have much lower score on the unseen data then on the one it trained on.

V. Conclusion

Work on this project proved full of challenges. The data is very sparse and high-dimensional and the distribution of the data including the target variable is not normal. This required special approach in preprocessing of the features because mean and variance centering, being standard techniques, are not recommended on sparse data. The data has been thus transformed using a natural logarithm which performed feature scaling while preserving the scarcity of the data. The target variable has also been transformed using a natural logarithm. Another challenge included the anonymized feature data including the names and meaning behind individual features.

One of the challenges was selecting an algorithm that performs well on the data. Several well know regression algorithms have been selected. The algorithms we have examined are LinearRegression (ordinary linear regression), Lasso (linear regression with L1 regularization), Ridge (linear regression with L2 regularization), ElasticNet (linear regression with L1 and L2 regularization), KNeighborsRegressor (regression based on k-nearest neighbors), MLPRegressor (multi-layer neural network regression) within the ScikitLearn library and LightGBM (gradient boosted decision tree regression) which is a gradient boosting decision tree algorithm.

None of the algorithms performed particularly well and some had RSME errors higher than the naïve predictor error of 1.75 using mean as a prediction for all data points.

Linear regression algorithm cannot model the data well. The Linear regression has cross-validation RMSE error 13.30 which is worse even compared to the naïve predictor. Lasso, Ridge and ElasticNet are all algorithms based on linear regression with regularization. Lasso with L1 regularization, Ridge with L2 regularization and ElasticNet with L1 and L2 regularization. These algorithms preform considerably better than the ordinary linear regression. The Lasso algorithm has a cross-validation RMSE error of 1.68 same as Ridge regressor. The EasticNet algorithm cross-validation RMSE error metric is similar at 1.69. KNeighborsRegressor performed pretty badly as well having the cross-validation error higher than the naive predictor. The MLPRegressor also has cross-validation worse than the naïve predictor. The LightGBM algorithm has the lowest cross-validation RMSE error of 1.46. This algorithm was used in all further analyses in the project.

Feature selection has been performed on the data to reduce its high dimensionality using univariate feature regression test and the resulting dataset has been expanded using a combination of decomposition algorithms (PCA, SRP, and NMF) and statistical functions (sum, min, max, interquartile values, skewness, and others) applied row-wise.

The final feature selection was performed using LightGBM algorithm feature importance function reducing the final data set to just 150 features. The LightGBM RMSE cross-validation error on the final dataset is 1.36.

The next challenge included the hyperparameter tuning of a LightGBM algorithm which has many hyperparameter. The algorithm has many hyperparameter. To find the best once a random search through predefined set of hyperparameters was used. After 100 steps the optimal algorithm was found. The RMSE cross-validation error of the optimal algorithm is 1.33.

The full end to end solution including data preprocessing pipeline, feature engineering and selection, algorithm selection and tuning with corresponding RMSE cross-validation error for each step involved is shown schematically in the Figure 18.

The final algorithm preforms well and produces well behaved results. The cross-validation error of the final algorithm is 1.33 on the data for which target variable is available. The error on the test dataset is 1.41 determined by prediction on the true test data via Kaggle website (for which target variable is not available).

The increase in the error on the unseen data is reasonable considering that we had to train our data on only 4459 train data points and the test was performed on 49342 test data points. Although, the error is still relatively high compared to the naive predictor which points to initial data complexity and potentially to still unexplored feature engineering or algorithm selection and tuning to produce a lower error and better fit to the data.

The supplied data could be some form of very sparse time-series data so some feature engineering like running mean or differences between data point points could be added to test this out. Expanding the algorithm selection to include Naïve Bayes, SVM – Support Vector Machine, or some alternative form of artificial neural network than the one used here could also hold potential for a better solution. Overall the algorithm preforms well and provides a robust solution to the problem at hand. However, the error is still relatively high which points to initial data complexity and potentially to still unexplored feature engineering or algorithm selection and tuning to produce a lower error and better fit to the data.

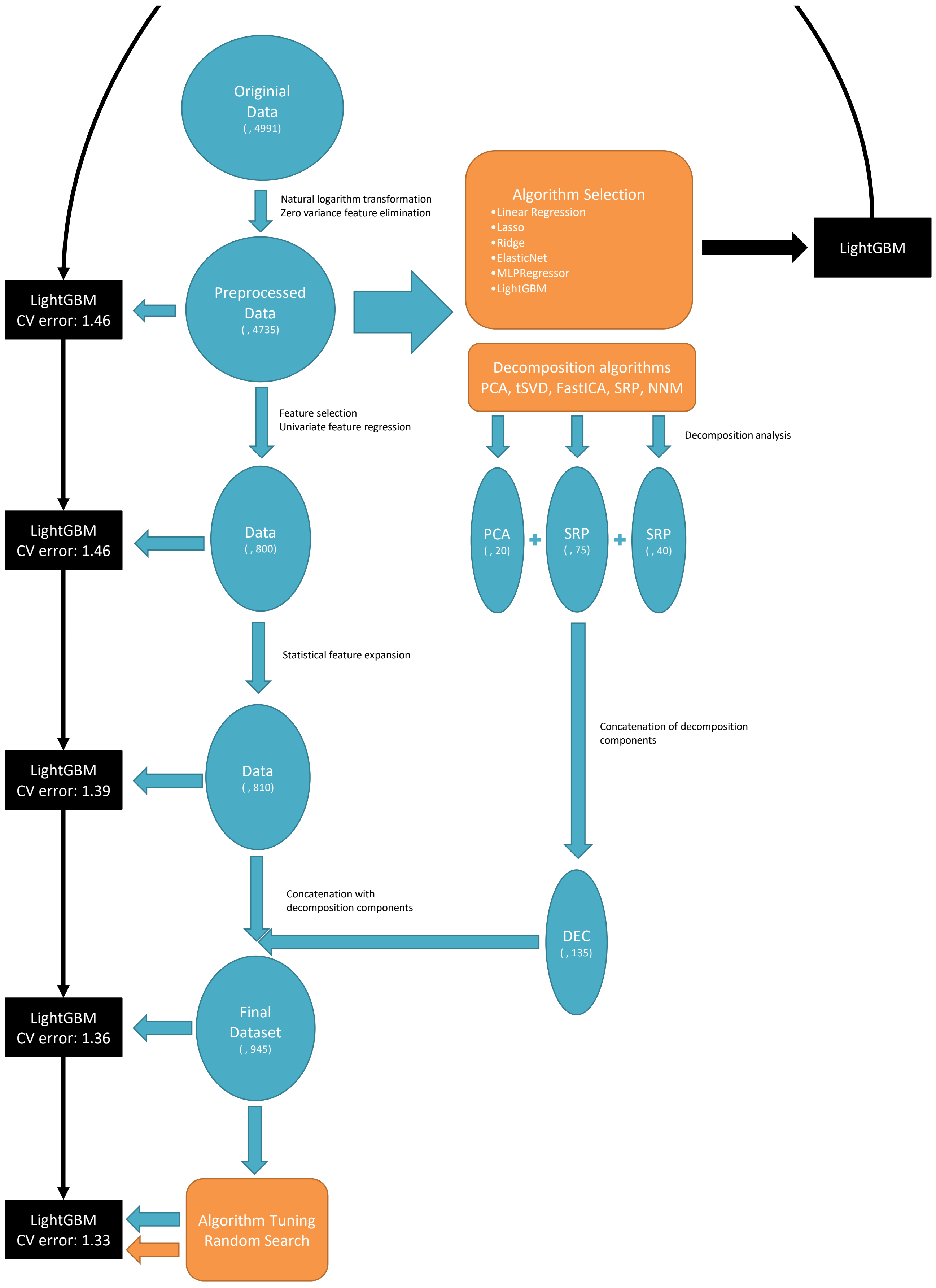

Figure 18. End to End Solution

VI. References

[1] Kaggle https://www.kaggle.com/

[2] Santander Value Prediction Challenge https://www.kaggle.com/c/santander-value-prediction-challenge

[3] Pandas - Python Data Analysis Library https://pandas.pydata.org/

[4] Seaborn - Statistical Data Visualization https://seaborn.pydata.org/index.html

[5] SciKit Learn - Machine Learning in Python http://scikit-learn.org/stable/

[6] NumPy - Scientific Computing in Python http://www.numpy.org/

[7] The Data "Property" https://www.kaggle.com/c/santander-value-prediction-challenge/discussion/61329

[8] Appeal to Kaggle https://www.kaggle.com/c/santander-value-prediction-challenge/discussion/62498

[9] SciPy https://docs.scipy.org/doc/scipy/reference/index.html

VI. Code

See attached file: Machine Learning Capstone Project.html