Machine Learning Engineer Nanodegree

Capstone Project

Andy Shkabko

December 14th, 2016

I. Reduce manufacturing failures in BOSCH production line

Project Overview

Bosch is one of the leading manufacturing companies decided to tackle the challenges in their production line, concretely, Kaggle competition was initiated with a goal to improve the output of the final products and reduce the manufacturing failures. There are different ways to reduce manufacturing failures like to stop the production lines and fix the errors on occurrence. However, due to the complexity of the production process there are some of the potentially failing parameters which can not be easily identified and have to be monitored on different measuring stations/points. Let's imagine the car engine that has slightly noisy (σ=2) numerical readings on one of the measuring stations. Despite the fact, the point is within the error margin, this error can result in even noisier next parameter on the following stations resulting the engine will not start at the end of the production line. Production line of Bosch company is very similar to the car example and require rigorous measurements of complex data, as samples move through the production lines. Since the data is very complex, machine learning algorithms should be used to pick and train all the possible variations of the features to successfully predict the output of the production line. Such prediction will potentially save a lot of money for final test or even perhaps help Bosch to detect times when things goes wrong during the production. Imagine the car production as in the case above. The final test will require the measurements of all the systems together on a test stand and in some cases even a testdrive necessity. The solution for this problem will be valuable predictions for the final car quality control even without doing any test stand measurements or a testdrive.

Bosch provided 3 sets of training data for each sample: numerical measurements, date(time) and categorical data. Each sample has a unique 'Id' and the goal is to predict which samples will fail quality control (represented by a 'Response' = 1). The dataset contains only anonymized features and they are

named according to a convention that tells the production line, the station on the line, and a feature number. E.g. L1_S11_F1111 is a feature measured on line 1, station 11, and is feature number 1111. The date features provide a timestamp for when each measurement was taken. Each date column ends in a number that corresponds to the previous feature number. E.g. the value of L0_S0_D1 is the time at which L0_S0_F0 was taken.

In addition to being one of the largest datasets (in terms of number of features) ever hosted on Kaggle, the training data for this competition is highly imbalanced with a fail rate ('Response' = 1) of about 0.5%.

Problem Statement

Based on the training data (numerical, date, categorical) the task is to predict the output (0 or 1) for the test data. The test data has similar structure and consists of 3 files: test_numerical, test_date, test_categorical and has the same amount of samples as for the training. The expected solution should have 1,183,748 predictions. The possible type of algorithms to consider for this competition is supervised learning classification algorithms, because the data has distinct outputs '1' and '0'. Since the amount of features is big(>3K), after data preprocessing: cleaning, feature selection, duplicate removal etc., we will use gradient boosted trees (xgboost) for each of the data types: numerical, categorical, date; train datasets independently and find the most relevant features by filtering: important features will be chosen for the next training step, while irrelevant or weak features will be removed away from the next stage. Following this way, the selected features will improve the training speed significantly, however, it can result in slight downturn of prediction rate. Since we will use desktop computer for this competition and would like to get decent results fast, this type of filtering is appropriate first step. Ones the small model is one might consider to train a full model as will be discussed later in the Refinement section.

Metrics

Submissions for public leaderboard are evaluated with 30% of data based on Matthews correlation coefficient (mcc) as a measure of the quality of binary classification error. The mcc metric is especially efficient when the data is imbalanced like in a Bosch case. The coefficient is derived from the confusion matrix (error matrix) and takes TP (true positive, as correctly predicted failed samples) and FP (false positive, as incorrectly predicted failed samples), false negatives (FN) and true negative (TN) with a combination as:

$$mcc = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

MCC range is [-1;1] with +1 for being perfect agreement, 0 – no better than a random and -1 as total disagreement between prediction and observation[(www.sciencedirect.com/science/article/pii/0005279575901099)].

II. Analysis

Data Exploration

1. Numeric features (Data_exploration.ipnb).

   File train_numeric.csv can be parsed into memory in 1min 11s and contains 1183747 samples and 968 features + 'Id' and 'Response' columns. The statistics on the traindata shows that failed samples (Responses = 1) of 6879 out of more than 1million samples! Very imbalanced dataset. There is lot of data for failed samples for some of the features like L0_S0_F0 (3608) and not much for others like L3_S50_F4251 (158). Splitting the features into lines, stations and features (not all but with 'F') gives 4 lines, 50 stations and 968 unique features. All the features are distributed within [-1,+1], however, some of them are in the [0,1] range.

2. Date features (Data_exploration.ipnb).

   File train_date.csv can be parsed into memory in 1min 49s and contains 1183747 samples and 1156 features + 'Id' column. Statistics on lines for the date features reveals the same 4 lines L0 - L3, but there are 2 more stations and no numeric data for them exist. Additionally, its worth to mention that the time and numeric features are asynchronous, since there are 968 numeric features and much more (1156) date features. The features are all distributed within [0,1800] range. If we compare stations in num and date data, there are 2 stations S42 and S46 where the numerical data was not given by unknown reasons. It might be that having only date data but not actual measurements can gives some clues about the failure.

3. Categorical features (Data_exploration.ipnb).

   File train_categorical.csv can be parsed into memory in 4min and contains 1183747 samples and 2140 features + 'Id' column. There are 18 stations which do not provide any categorical input data 'S0', 'S12', 'S13', 'S17', 'S19', 'S20', 'S33', 'S34', 'S37', 'S40', 'S41', 'S45', 'S48', 'S5', 'S50', 'S51', 'S7', 'S8' There are 93 unique categories in the categorical training set. Many of categories contain "T" symbol that was cut from the data to increase the loading speed.

Exploratory Visualization

1. Numeric features (Numeric_data_visualization.ipynb).

We decide to pick some of the important features (how to chose them will be discussed later) and visualize them. Fig. 1 shows the scatter matrix, which is a known good technique to visualize continuous features and find relationships between features simultaneously. Scatter matrix contains pairwise 26 scatter plots of the features plotted against each other. The plots are 2D and allow to roughly determine if features are linearly correlated.

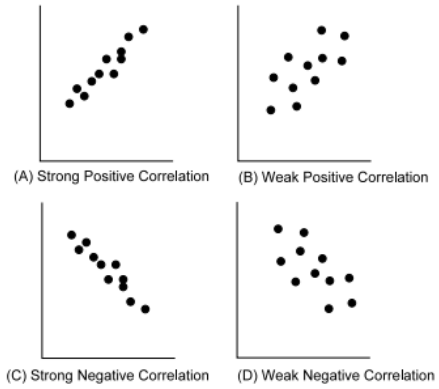


Fig. 0. Classification of correlations. (Gerstman, SJSU lecture notes)

From the figures its clear, that some of the features are uncorrelated, but some have positive corellations as represented by linear-like graphs with positive slope, L1_S24_F1846 and L1_S24_F1723. The diagonal plots are univariate distributions of each variable, which shows if data is likely e.g. normally, distributed or skewed and need some further preprocessing in order to be effectively used in machine learning algorithms.

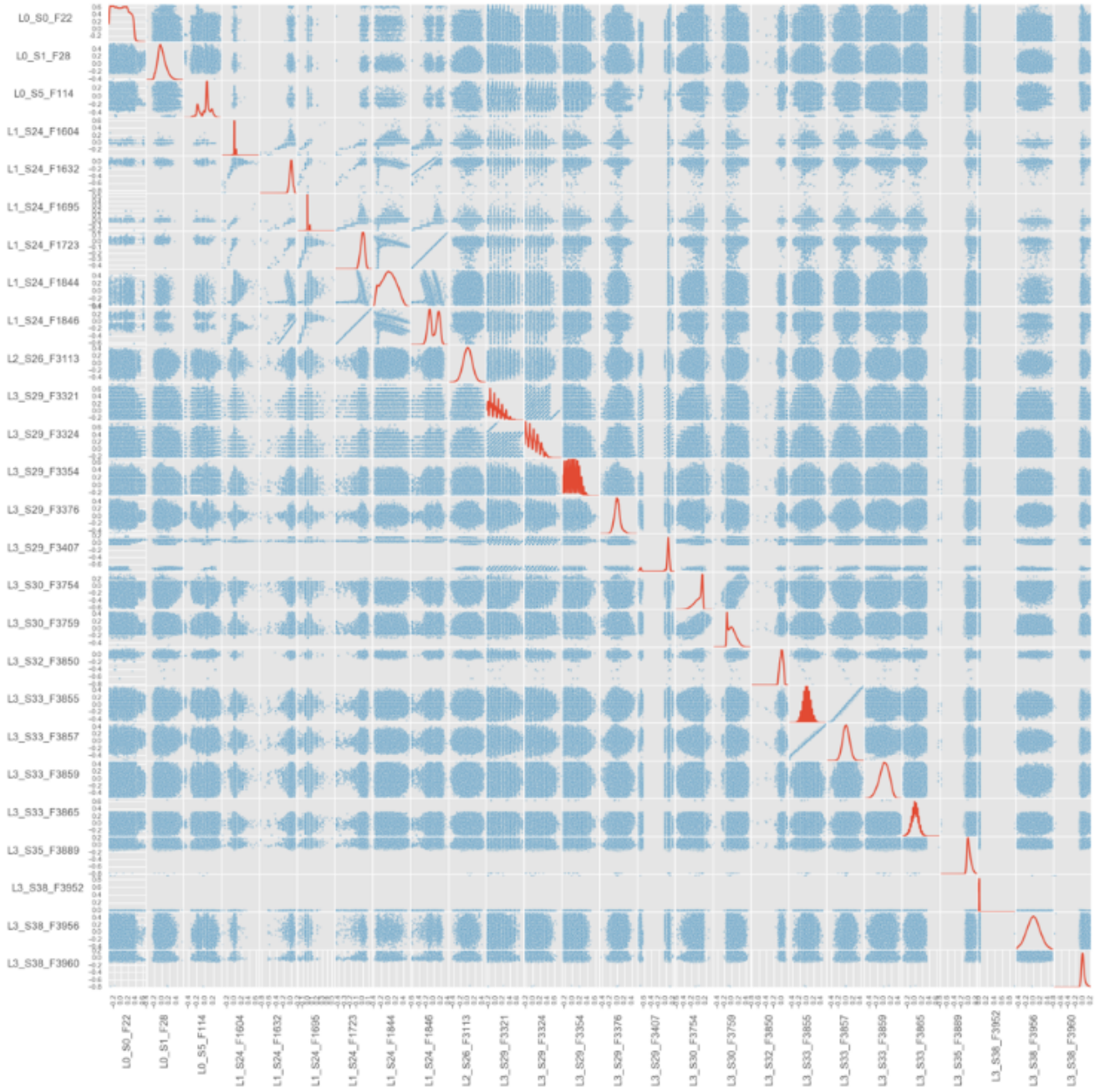


Fig.1 Scatter matrix plot for 26 selected numeric features.

In a next step we investigate if there are some of the differences in the distribution of the data for both classes: 0 and 1. For that 2 approaches are used: violin plots and histograms, which have some advantages and disadvantages. The Responses data for violin plots are split into two: 0 and 1, and plotted separately keeping the same areas for both classes. Looking at feature F3354 one suspect that broken samples have smooth distributions e.g. like one cannot switch them on the station, while

working devices have only discrete values.

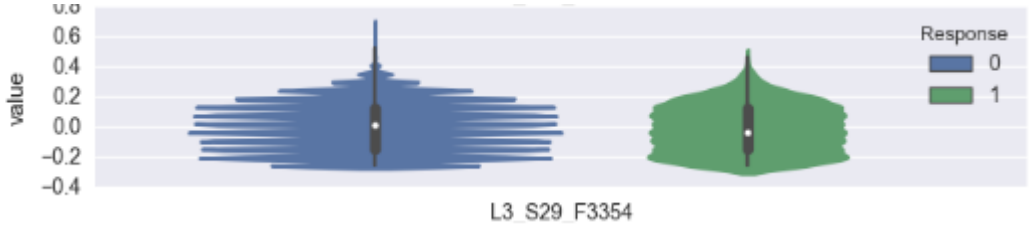
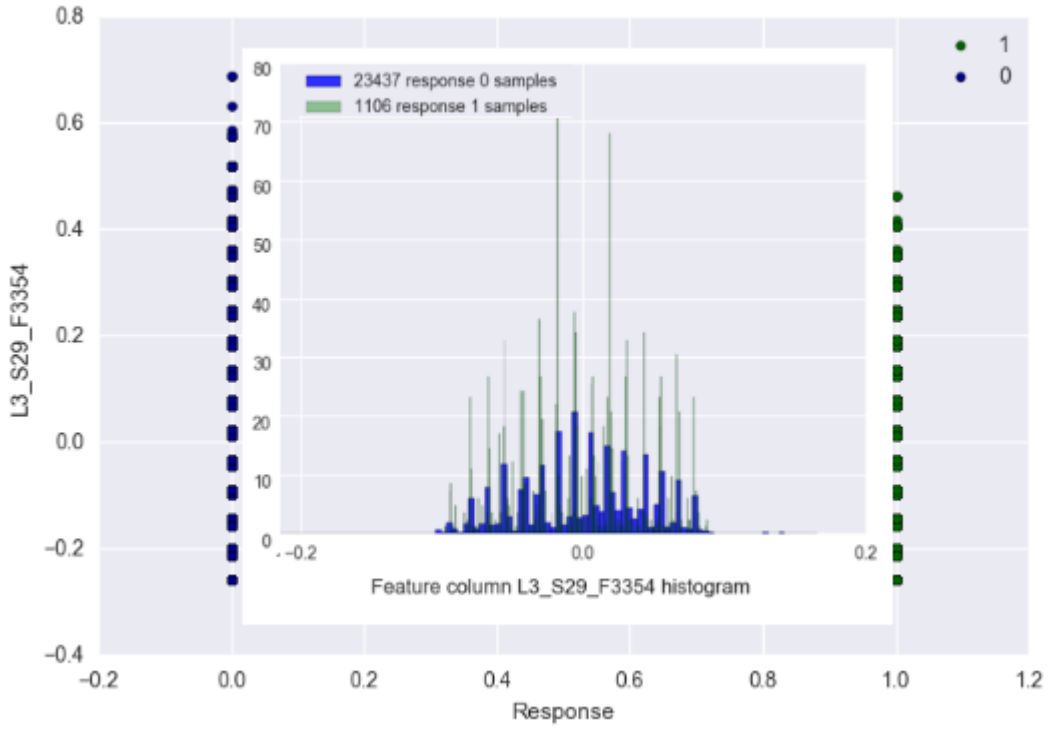
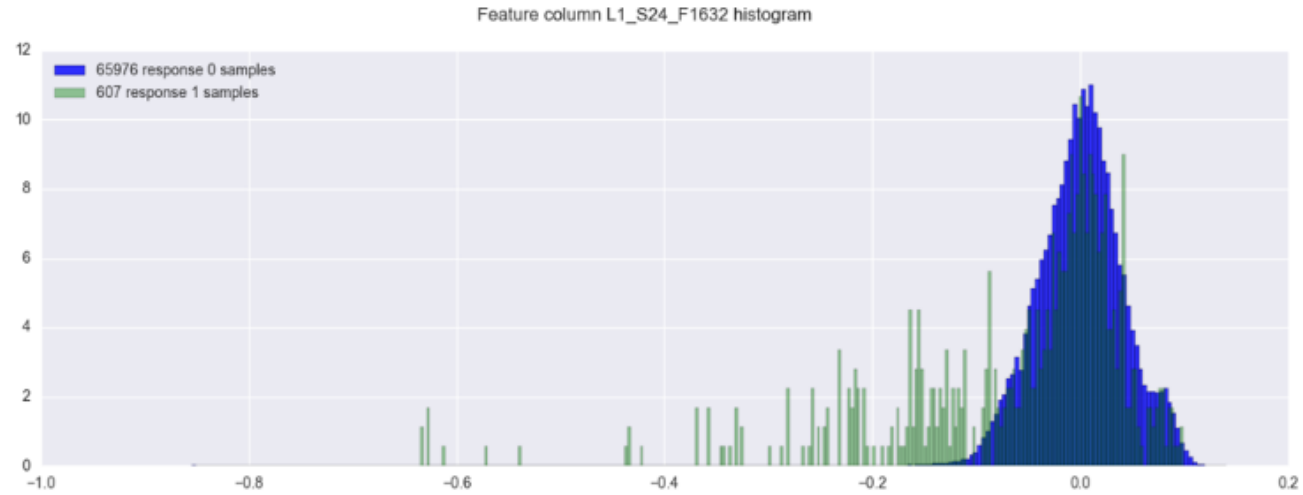


Fig.2. Violin plots for numeric feature for 2 classes (0 and 1).

To check this hypothesis the scatter plot for F3354 feature is plotted, Fig.3a and show all values are in fact discrete. However, working samples have wider range of the data compared to broken ones. Plotting the histograms for both classes, confirms this finding.



(a)



(b)

Fig.3a. Scatter plot and histograms for F3354 feature, b. Histograms for F1632 feature to show wide range (negative values tail) for broken samples (in green).

On Fig. 3b F1632 feature have distinct ('outlier') values in lower values range, thus if read values on this for this station has too low values, the sample is most probably broken.

To find some of the strong features with linear dependencies, Pearson correlation coefficient for the Response column is calculated. As a result, most of the features have ~0 correlation. Pearson correlation coefficient is basically covariance *cov* of the two features x,y (measure of how much two random variables change together) over their multiplied standard deviations, $\sigma_x \sigma_y$.

$$\rho_{x,y} = \frac{\sum(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum(x_i - \overline{x})^2 (y_i - \overline{y})^2}} = \frac{cov(x,y)}{\sigma_x \sigma_y}$$

For the correlations between the features, the correlation heatmaps are plotted for both classes. Some of the features are correlated indeed as was found in scatter matrix. Additionally, some of the features have linear correlations and these features are very close to each other forming the clusters. Similar situation with clustering is valid for NaN data as well.

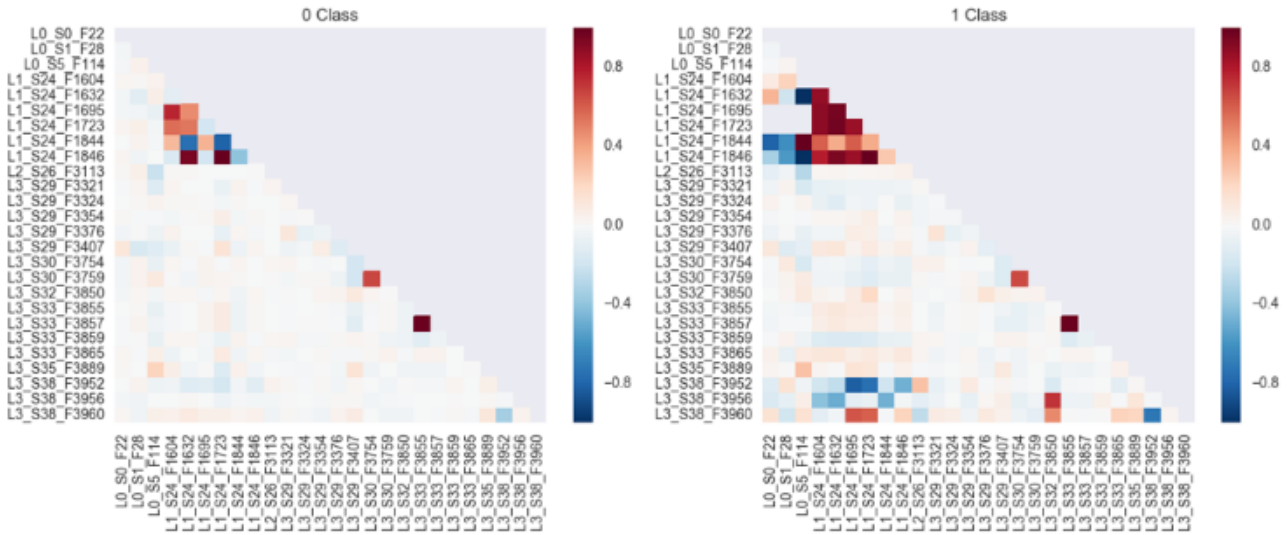
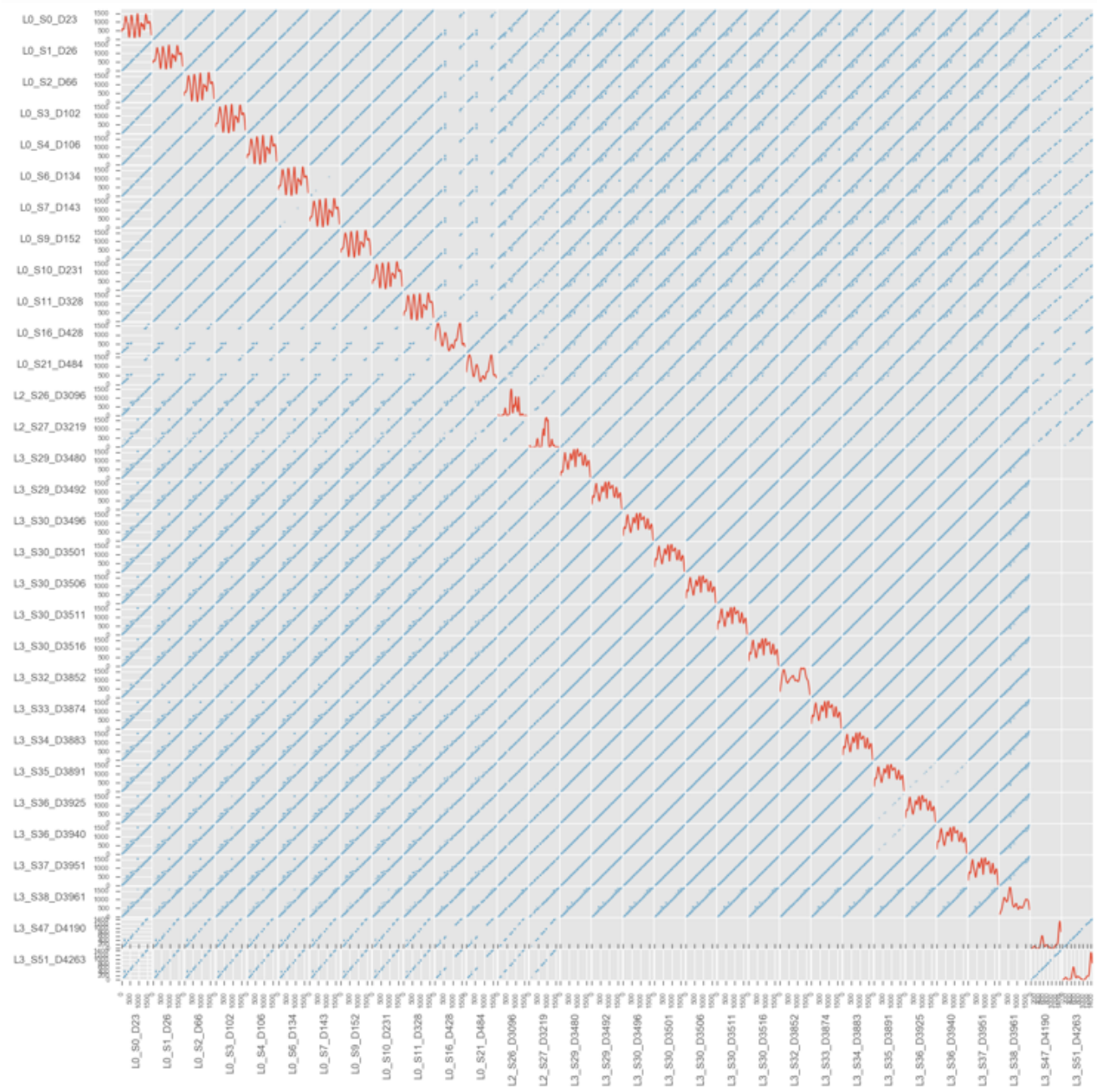


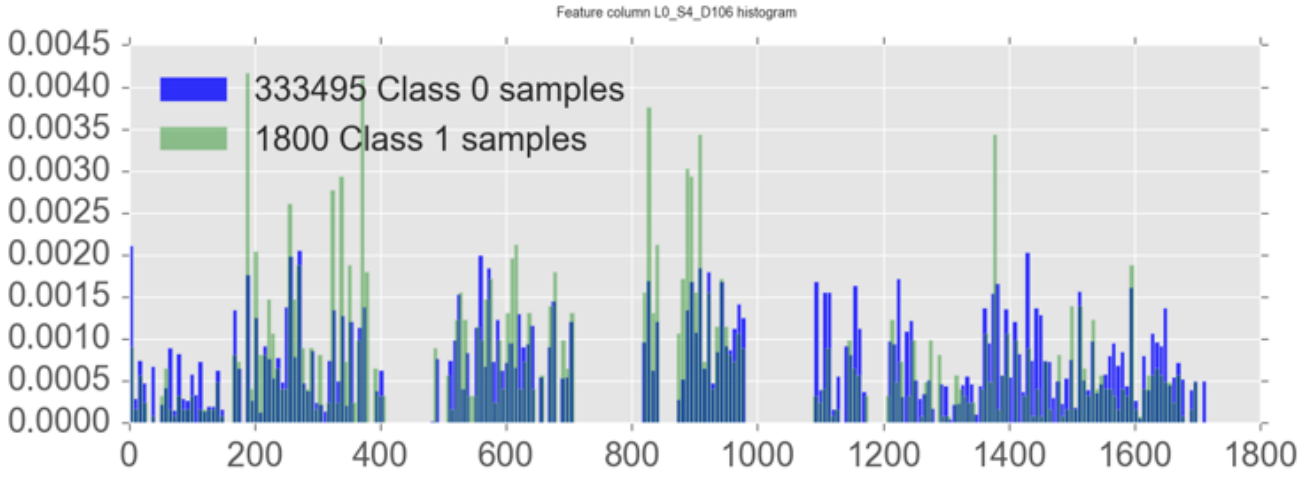
Fig.4. Heatmaps for 2 classes. Note, features are alphabetically ordered to show correlations with some features forming clusters on the maps.

2. Date features(Date_data_visualization.ipynb).

   All the features represent similar time series and are not normally distributed. They are all linearly correlated with some exceptions. As for univariate distribution, first investigations show that many features have similar distributions and might have duplicated data(which will be checked later). Ploting histograms confirm that many features have a time break between 800 and 1000, and some have more time breaks.


L0_S0_D23
L0_S1_D26
L0_S2_D66
L0_S3_D102
L0_S4_D106
L0_S6_D134
L0_S7_D143
L0_S9_D152
L0_S10_D231
L0_S11_D328
L0_S16_D428
L0_S21_D484
L2_S26_D3096
L2_S27_D3219
L3_S29_D3480
L3_S29_D3492
L3_S30_D3496
L3_S30_D3501
L3_S30_D3506
L3_S30_D3511
L3_S30_D3516
L3_S32_D3852
L3_S33_D3874
L3_S34_D3883
L3_S35_D3891
L3_S36_D3925
L3_S36_D3940
L3_S37_D3951
L3_S38_D3961
L3_S47_D4190
L3_S51_D4263

(a)



(b)

Fig. 5(a). Scatter matrix for date data and (b) typical histogram with about ~5 breaks (feature D106)

Algorithms and Techniques

In this work I will use Extreme Gradient Boosting (xgboost), which is a gradient boosted trees algorithm for supervised learning. It is well-established award winning and one of the most advanced algorithms of the last few years. It uses trees ensembles as a set of classification and regression trees (CART). The properties of xgboost are very suitable to our model, because xgboost

1. can handle big data
2. ignore redundant variables and handle missing data
3. good for mixed features: qualitative and quantitative

Compared to decision trees which contain only decisions, CART has the scores for each leaves and by building many trees and ensemble each leaves the leave's scores are summed, thus algorithm complement and improve the predictions. As for the case of predictive algorithms training, the objective for the xgboost is to optimize the cost function which consists of training error(loss) and regularization parameter. To build the cost function for the trees an additive strategy is used, when starting from the "weak learners"(can be toss coin) the prediction is made and scores are calculated. In a next step of training a new tree is added but only the one that optimizes the cost function. This process is called boosting and it results in improved predictions by adding more trees.

Since we deal with big data, which consist of 3 data types, the usage of ensemble learning is of big advantage. We can learn and train the system for each type of data separately by apply xgboost for each

dataset and then combine all of them and find rules for the whole data. An advantage of xgboost implementation is feature importance scores. The best features will be picked from each dataset and by combining them the final model will be trained and optimized for maximum prediction. Obviously, xgboost will handle sparse data, but in a preprocessing step, the categorical data are will be categorized and numeric features will be scaled.

For tuning the parameters of the xgboost we will use random grid search, which is very similar to the grid search, however gives some of the advantages. The parameters can be chosen from the distribution over possible parameters randomly, plus each parameter can have a distribution and not explicitly set parameter values. Grid search might give smaller precision in parameter determination as compared to grid search, however, it can save time. Random grid search is particularly good for the algorithms with plenty of parameters to tune like extreme gradient boosting of xgboost. In this project 25 gridsearch parameters were tested in each run, which significantly decreased amount of possible variations of the parameters.

Benchmark

The objective for the model is a binary logistic regression. Since xgboost gradient boosting uses gradient descent optimization, the proper benchmark for highly imbalanced dataset should be used, like mcc. Gradient descent of xgboost will return probabilities, which should be converted into binary predictions and at the same time be optimized for the highest possible mcc.

As a benchmark for the prediction we estimated the amount of true positives as an average for the training data. Since the data is very imbalanced and has only 0.0056% failures, we conclude our predictions should be about 6000 failed samples out of more than a million of test samples and mcc=0 for complete randomness in the predictions.

III. Methodology

Data Preprocessing

(Please note only part of the preprocessing is in Preprocess-num-date-cat.ipynb the rest is in main program with randomized crossvalidation)

1. Numeric features.

   Loaded the csv file and converted in into binary pkl to have faster loading. We also added box-cox preprosessing to correct the skewness of the data for the features with skewness above 0.2.

Before the skewness calculations and correction the train and test data were concatenated.

2. Date features.

   For date features an efficient code was used to find duplicates that resulted in many features appeared identical. The .csv was also converted to pkl. For all the date features the NANs were replaced with a mean values. For date features few additional features were used: minumum date when the sample entered the production, maximum date when the product was released and the difference between them.

3. Categorical features.

   For the categorical file at first symbol “T” was subtracted. After that duplicate columns were deleted. For all the cat features the NANs were replaced with a mean values.

Feature selection (Feature_Importance.ipynb).

As it was already mentioned, there are a lot of features in the dataset and as was stated on the Kaggle Bosch data page(the data is not only imbalanced but also largest dataset ever hosted on Kaggle in terms of numbers of features). So we decided to run xgboost and determine the most important features for the training sets and after use these features to predict the outcome for the big problem. Fig. 6 shows feature importance scores for numeric, date and categorical features. The preprocessing xgboost was run many times, and features were added manually to insure mcc score is maximized in the main program.

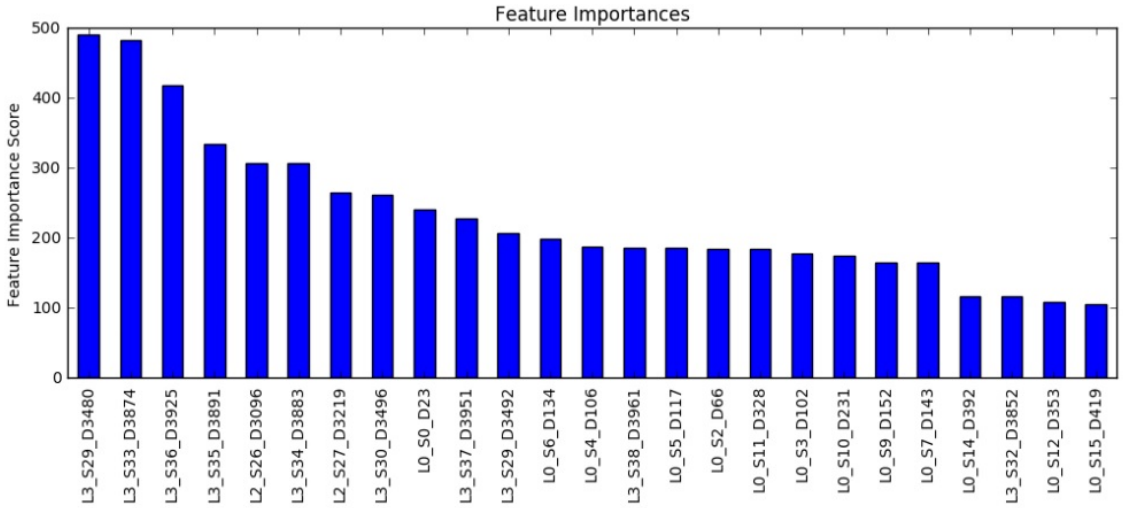


Fig. 6. Typical importance feature list (max 25) and their scores obtained by xgboost on date dataset.

```
if __name__ == "__main__":

    def mcc(y, preds):
        ## max mcc from thresholds and predictions
        thresholds = np.linspace(0.01, 0.99, 50)
        mcc_ = np.array([matthews_corrcoef(y, preds>thr) for thr in thresholds])
        best_threshold = thresholds[mcc_.argmax()]
        return mcc_.max()

    #start_time = timer(None)

    train, test = data()
    features = list(train.columns)
    features.remove('Response')
    features.remove('Id')

    X = train[features]
    y = train['Response']
    parameters = { 'n_estimators': [ 100, 150, 200],
                   'objective' : ['binary:logistic'],
                   'learning_rate' : [0.02, 0.05, 0.1],
                   'max_depth' : [9, 10, 12, 15],
                   'base_score' : [0.0056],
                   'min_child_weight': [ 2, 4, 6],
                   'gamma' : [0, 0.2, 1],
    }
    ssscv = StratifiedKFold(y, n_folds=3, shuffle=True, random_state=0)

    score = make_scorer(mcc, greater_is_better=True)
    grid = RandomizedSearchCV(XGBClassifier(),
                              param_distributions=parameters,
                              n_iter=23,
                              cv=ssscv,
                              scoring=score,)

    grid.fit(X, y)
    best = grid.best_estimator_
    for params, mean_score, scores in grid.grid_scores_:
        print("%0.3f (+/-%0.3f) for %r" % (scores.mean(), scores.std(), params))

    for g in grid.grid_scores_: print(g)
    print(grid.best_score_)
    print(grid.best_estimator_)
    print("Best params: {}".format( grid.best_params_))

    scores = np.array([[
            e[1],
            e.parameters['n_estimators'],
            e.parameters['gamma'],
            e.parameters['learning_rate'],
            e.parameters['max_depth'],
                   for e in grid.grid_scores_])
    plt.scatter(scores[:,1], scores[:, 0])
    plt.xlabel('n_estimators')
    plt.ylabel('MCC average')
    plt.show()
    plt.scatter(scores[:,3], scores[:, 0])
    plt.xlabel('learning rate')
    plt.ylabel('MCC average')
    plt.show()
    plt.scatter(scores[:,4], scores[:, 0])
    plt.xlabel('Max depth')
    plt.ylabel('MCC average')
    plt.show()
```

Implementation and Refinement (RandomSearchCV_3.ipynb)

For implementation the xgboost algorithm with binary:logistic objective is used with standard parameters plus base_score=0.0056 to take into account imbalanced classes.

Initially training on only numerical+date features gave mcc = 0.22. XGBoost and on categorical data gavee mcc=0.12 which confirmed that the separate datasets give positive results (mcc>0) and all datatypes should be combined.

The combined data is trained with cross validation n_folds=3 in a stratified manner to keep the ratio of both classes.

The parameters tuning during optimization is implemented in a randomized grid search and parameters for optimizations are number of estimators, learning rate, maximum depth of the trees, child weight and gamma, all of which can be tuned to find max mcc. The script in this capstone project is limited to about 78 features and can be increased to higher numbers for better predictions, however upscaling will result in significant increase of training time. We also see positive trend in number of the estimators, concluding that for the sets of optimized parameters the training can be improved!

Adding additional features, actually improved score by 0.05-0.1, as well as combining all the features and dropping redundant features.

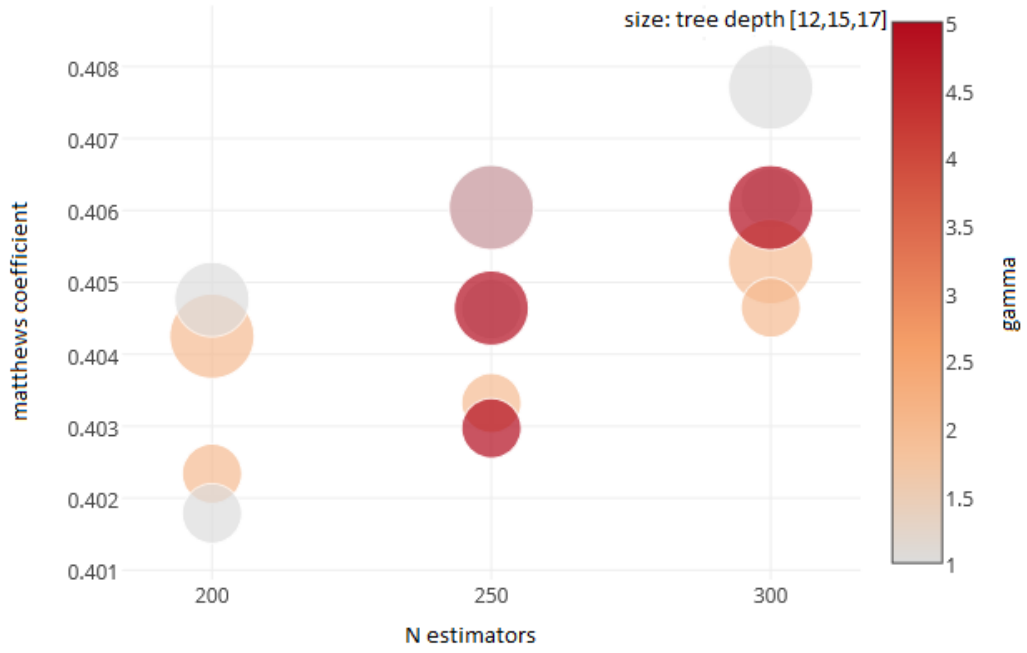


Fig. 7. 4 dimensional data for optimizations of 4 parameters for learning rate=0.05,

min_child_weight=4, as optimized on a previous steps.

Figure 7 represent last steps in parameters tuning after learning rate and min child weight showed optimum in the tuned parameters.

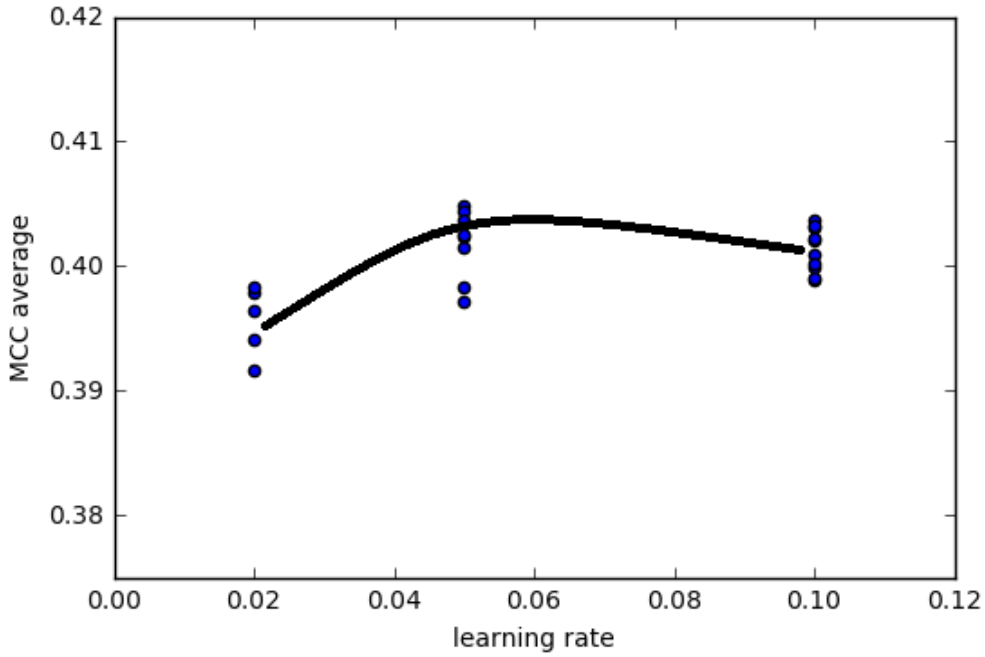


Fig.8. MCC optimization results as a function of learning rate. Black line is for guide eyes purposes.

As was already mentioned before, xgboost return probabilities, which has to be converted into 2 classes with a threshold [0,1] in order to have optimized mcc. After the training was performed the validation score is calculated at different thresholds (correlation score threshold) and the mcc curve is plotted, Fig. 9. It has a peak value above 0.4 which defines best threshold above which the predictions are 1 and below which the prediction are in a negative class 0.

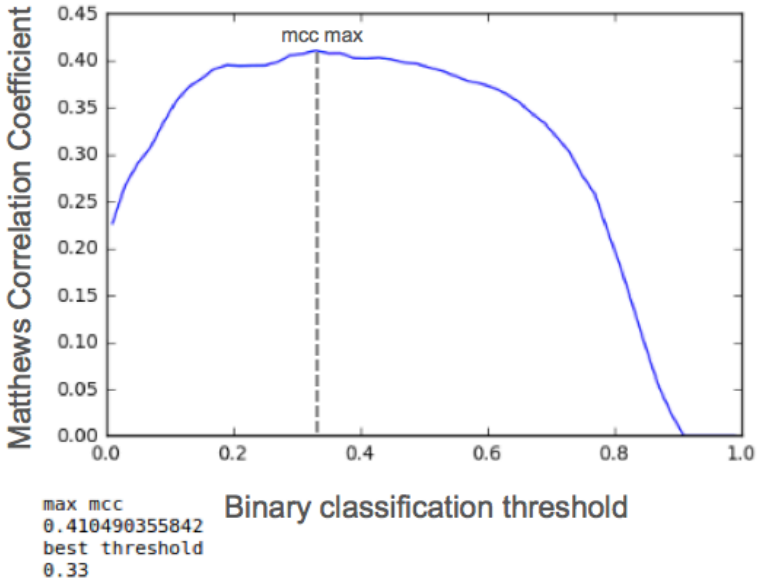


Fig.9. mcc as a function of threshold for validation data

IV. Results

Model Evaluation and Validation

We got mcc=0.41 for the data with limited amount of features, but this score can be easily improved by adding more features, especially date features. For example, an increase of features from 78 to 150-200 will result in increased computation for non-optimized parameters from 10mins to 1h and will still require optimization of all the parameters for higher mcc scores, Fig.7.

As we also find out an increase of tree depth and number of estimators also positively affect the mcc scores, but again and again, the time is additionally increased.

In my Kaggle submission I used 5 folds which is 4/5 parts for training and 1/5 for testing used 5 times and averaged, which also gave better scores of 0.43.

We also performed some sort of sensitivity analysis as described above by filtering the features important for mcc and final leaderboard.

Justification

The final mcc score as predicted on 70% of data on private leader board gave mcc = 0.44732 which is above 0 considered to be random. The final place is in a top 10% (bronze medal).

Fig.10. Kaggle leaderboard.

To increase the score to maximum level blending was used, when the predictions from slightly modified models with different feature sets/preprocessing strategies (feature engineering) were arithmetically summed and normalized without specific weight adjustments (considering the scores of different models within mcc ~ ±0.1)

V. Conclusion

In this project we have learned how to predict a line performance in a reasonable manner with a correct performance metric for binary classifiers (mcc). The mcc results are well above random predictions (benchmark=0) reaching 0.447 on the unseen data. To reach this goal different techniques were applied with some of steps summarized in Table 1.

Dataset/Action Step	Matthews Correlation Coefficient(on the LB)
Categorical/as is	~0.12
Date+Numerical/with 0.05% most important features	~0.21
Date+Numerical/ as is(unprocessed)	~0.25
Date+Num+Cat/~50 most important features	~0.27

Date+Num+Cat/50 + feature engineered	~0.36
Date+Num+Cat/50 + feature engineered+parameter tuning	~0.41
Data before plus 100 features plus blending 2 similar models	~0.447

Table 1. Step-by-step mcc improvements and steps/actions to improve prediction scores.

In the first steps since the amount of features is really big for this competition, every dataset was fit for prediction scores which gave first results and comparatively low scores 0.12-0.21. In the next steps all the data was cleaned, features unduplicated and most important features from different datasets created and merged to the new dataset for prediction which gave highest mcc scores.

The following conclusions can be drawn from the score dynamics in Table 1:

- Utilizing just a part of data gives already positive mcc, but adding more features and more data increase the prediction score
- Specifically engineered features gives a boost in prediction rates of ~0.1
- Blending of the results obtained by 2 different runs of similar xgbmodels gave a slight positive boost in the score.

Our solution provides pretty fast method of finding, visualizing and optimizing the set of numerous parameters of xgboost, but we think a lot of improvements can still be done. E.g. by increase the amount of estimators, depth of the trees, as can be seen from images of optimal parameters on Fig. 7.

Unfortunately, all the improvements mentioned will result in further increase of computation time. As for feature engineering, we showed that some of them are really useful, however, due to anonymized nature of some of the Kaggle competitions it's hard to engineer new quickly and efficiently. Releasing flaws in the productions or some details of the company's losses and revenues (as in the case of new Allstate claims severity https://www.kaggle.com/c/allstate-claims-severity) is against company's policies.

To summarize, I really enjoyed working on this dataset. It contains a lot of features and the data is imbalanced with only small portion of broken samples. The hardest was to find a way to handle different datasets memory efficiently. In general a lot of competitors used Amazon Web Services (AWS) with big memory instances to fit the whole datasets and do computation, however there always small model like the one here which should fit into 64Gb of RAM.

In general it was already memory difficult to run and factorize the sets of data like categorical data:

concatenate matrices, find duplicates. If the data will be bigger than that it will be obviously impossible to scale utilizing the same hardware.

For the computation time, 8 thread i7-6700K 4GHz had also very hard time with increased amount of features, so decreased feature set down to 100s from 1000s obviously saved a lot of computation time.

As further improvements for this work, we will recommend to run computation on machines with higher RAM, with more features and faster CPUs. As one more algorithm approach can be a deep learning approach without extensive feature engineering, because the amount of samples is pretty substantial (>1mil.) as well as there are a lot of features. DL solution can be blended with xgboost to provide better-improved solutions, however, the problems with data imbalance can be a big problem for DL. As for optimization for hyper-parameter tuning, Bayesian optimization will be a good option to improve times instead of gridsearch or randomgridsearch. In a last few days, GPU support for xgboost appeared as a plugin, which will allow training speedup up to 10 times (https://github.com/dmlc/xgboost/tree/master/plugin/updater_gpu).