Machine Learning Engineer Nanodegree

Capstone Project
Miloš Ranđić
September 6th, 2017

I. Definition

Project Overview

Nowdays, telecommunications industry is the phase where customer needs are in the center of attention. The greatest challenge is to align specific product offers to precise group of customers, in order to make customers satisfied with existing products and services, and to maximize company's revenue as well. Since the focus is on the customer, CRM(*Customer Relationship Management*) is one of the main points of interest for telco industry, especially when applying with machine learning[1].
It seems that over the years customer data has been collected and increased exponentialy in its volume, which means the user behaviour can be tracked and can teach us how customers think, what their needs are, and based on those rules, encourage us to conclude what product they should be offered. Telco industry is just a great place to make a research of user behaviour and create various product offers using supervised machine learing techniques. Supervised learning as a concept has been in the industry for a while, but nowdays its applications are growing exponentially[2].
Personal motivation lies in the need to make a research on behaviour of customers in telco industry, in order to build machine learning model for product recommendation with aim to generate more profits from customer services consumption[3].
The dataset considers *Telenor business customers* behaviour data available on a monthly basis. Data is available for a period of 16 months (04/16 to 07/17) . **Important notice:** Identity of the users and their true behaviour are **anonymized**, for privacy and user protection purposes declared by Telenor. Entire usage/revenue data presented in this dataset has been scaled using unique secret coefficient, avoiding privacy policy to be affected in that way.

Problem Statement

For this project, I have chosen the challenging problem about offering roaming add-on services to customers who travel to foreign countries. Since the roaming traffic is heavily billed while using international telco operator services, Telenor ensures that users who activate roaming add-on service won't be billed for additional fees as long as they consume mobile traffic within the add-on package. Of course, customers pay additional fee for this product at the moment of activation of the product, but the price is far cheaper than using mobile traffic without the add-on. Database that will be proposed tracks two groups of customers, on a monthly basis: users that buy this product when travel abroad and users that don't buy this product when travel abroad.
The goal of this project is to build machine learning model that could solve the problem of defining which users are more likely to buy this product in order to send them an offer for activating the service. On high level, the potential solution would consider building machine learning model using available data with appropriate tests using test data, in order to examine the performance of the trained model.
Since this is a two-category classification problem, one of the evaluation metrics that will be considered is confusion matrix, from which will be derived different metrics for measuring performance (precision, recall, accuracy and F-score).
Firstly, naive predictor should be built, using descriptive statistics in order to build naive classification critetia.

The model with highest performance on the test set should be used for generating campaigns for targeting users to buy add-on service. This campaign test will be used for demontration purposes only, since this model is not in the production yet, hence, it does not affect user behaviour from which we could measure our true campaign success rate. Only as-is success rate, without impact of our model will be observed. Take rate and roaming rate (explained in following sections) will be evaluation metrics for our campaign success.

Metrics

For this project, Accuracy and F-score measures will be used as evaluation metrics. Using these metrics naive predictor solution will be compared to learning algorithm soultion. Precission and recall will also be considered as evauation metics in order to track how our algorithm correcltly classifies users to one of two classes in test set.

$$precision = \frac{TP}{TP+FP};\ recall = \frac{TP}{TP+FN};\ accuracy = \frac{TP+TN}{TP+TN+FP+FN};\ F_\beta = (1+\beta^2)\frac{precision*recall}{(\beta^2*precision)+recall}$$

Selected optimal model will be used for creating target list in month *t*, in order to predict the potential buyers for the month *t+1*. Users who are selected by model as buyers in a month *t* will be assumed to buy the product in the next month, *t+1*. Take rate is calculated as ratio of actual takers intersected with target list, and actual roamers in month *t+1*. In this way, we calculate take rate as a ratio of users who activated add-on from target list and users from target list who were actually in roaming in the next month. Roaming rate explains the percent of correctly predicted users from target list, that happend to be in a roaming in the next month.

$$takeRate = \frac{count(targetedUsers_t \cap takers_{t+1})}{count(targetedUsers_t \cap (takers_{t+1} + noTakers_{t+1}))};\ roamingRate = \frac{count(targetedUsers_t)}{count(takers_{t+1} + noTakers_{t+1})}$$

II. Analysis

Data Exploration

Dataset contains following variables:

VARIABLE NAME	DESCRIPTION
TIME_ID	Year-Month of recorded behaviour.
SUBSCRIPTION_ID	Unique ID of a user.
SERVICE_SEG	Segmentation of user. There are four ordinal segments: N(New), Br(Bronze), Si(Silver), Go(Gold).
AP_NAME	Type of tariff plan. There are two possible roaming tariff plans: Travel Sure 10 and Travel Sure 50.
ACT_MONTH	Month of activity. This variable indicates the month [1..12] of the user being in roaming.
ACT_VOL	Entire monthly usage generated within international network, measured in points (mixed SMS+VOICE+INTERNET).
REV	Entire monthly revenue generated within international network.
NUM_OF_MONTHS	How many months in the past 4 months the user has been in roaming(within international network). Possible values: [0..4]
VOICE_VPN_MIN	Total monthly VPN minutes spent.
VOICE_ONNET_MIN	Total monthly minutes spent inside Telenor network.
VOICE_OFFNET_MIN	Total monthly minutes spent outside of Telenor network, towards national networks.
VOICE_INT_MIN	Total monthly minutes spent towards international networks, within national network.
SMS_NAT	Total number of SMS messages sent towards national networks, within national network.
SMS_INT	Total number of SMS messages sent towards international networks, within national network.
GPRS_NAT_MB	Total number of MBs spent within national network.
OS	Operating system user runs on a mobile device.

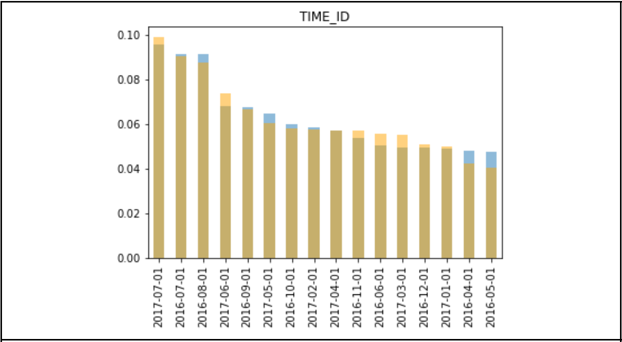
MF_REV	Total fixed monthly revenue from package bundle.
US_REV	Total monthly revenue from over the bundle usage.
FLAG	Is the user taker or no_taker of the add-on in particular month. Possible values [TARGET, NO_TARGET]. This is our target variable.

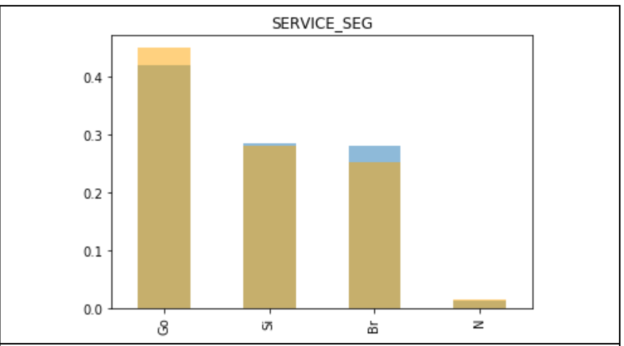
Table below shows count of records per group. We may notice this dataset is not balanced, only 4% of our data belongs to users who have already activated the add-on service.

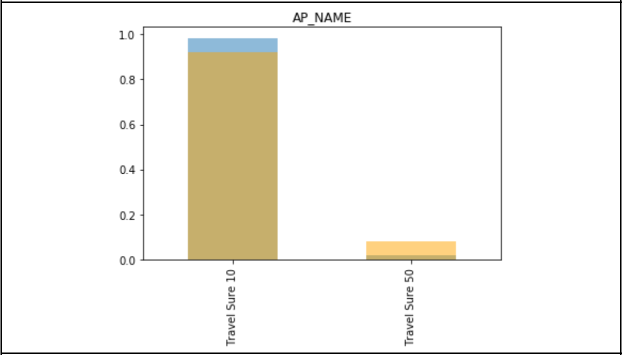
Description	Value
Total number of records	367.869
Total number of NO_TARGET	353.087
Total number of TARGET	14.782
Percent of NO_TARGET	96%
Percent of TARGET	4%

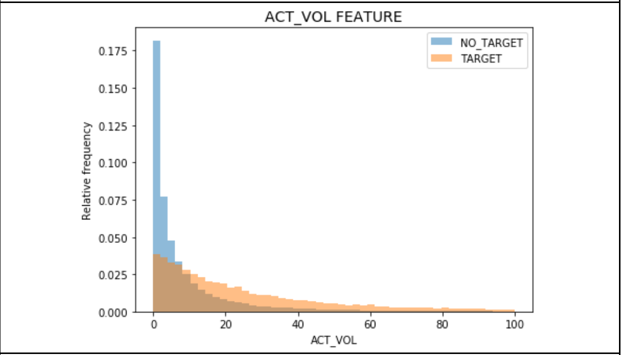
Exploratory Visualization

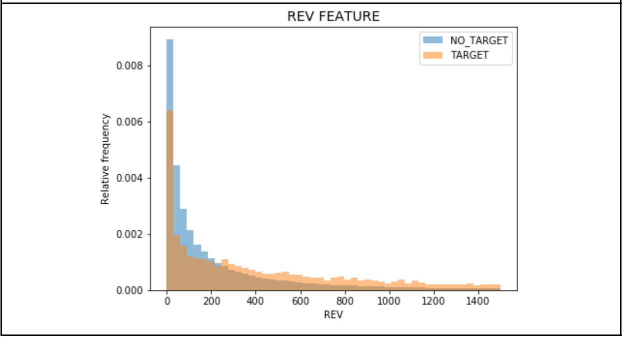
In this section, detailed visual observation of our features will be shown. Relative frequrncy histograms with comparisons between TARGET and NO_TARGET group for each feature will be shown (Orange bar – TARGET, Blue bar – NO_TARGET)

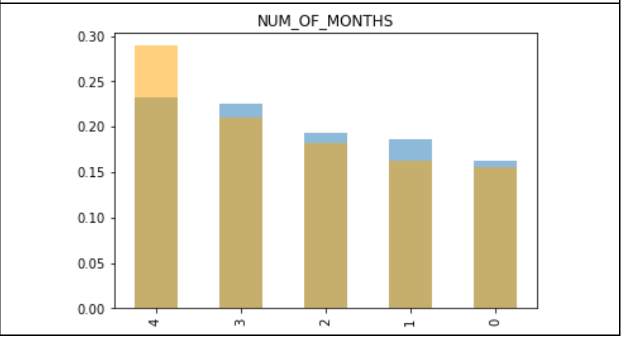


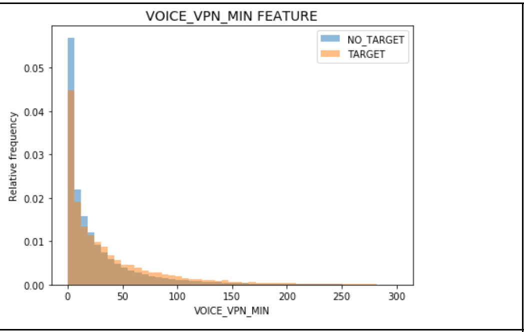
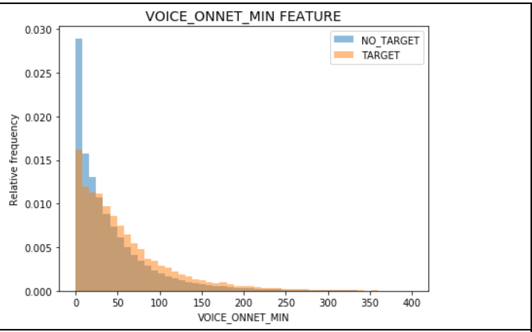
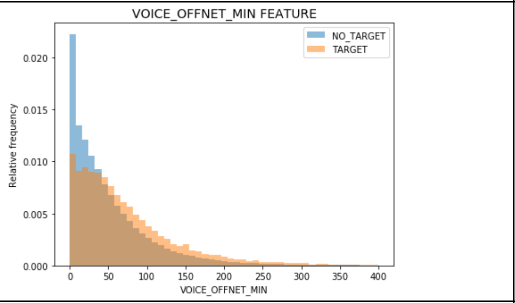
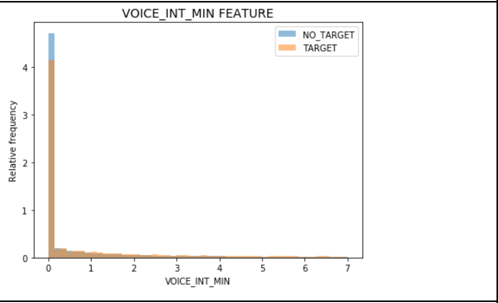
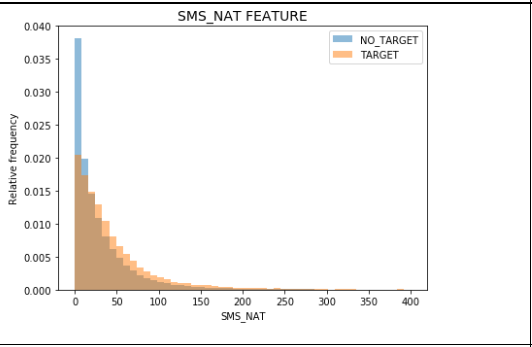
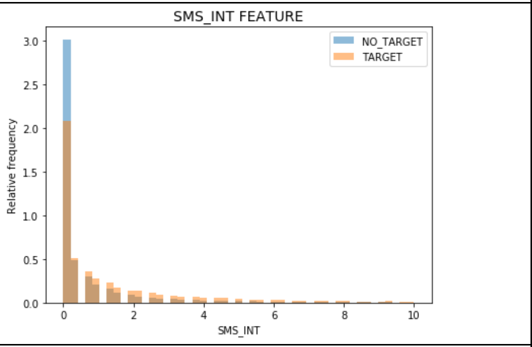
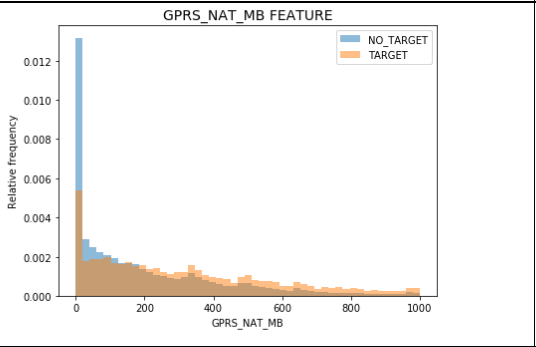
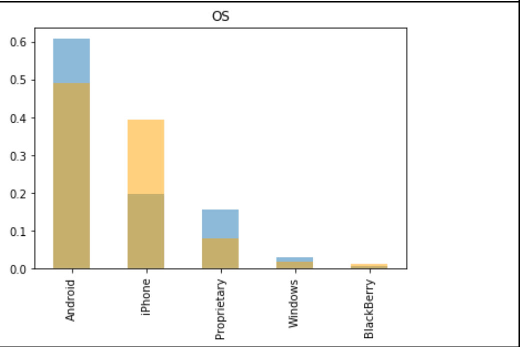
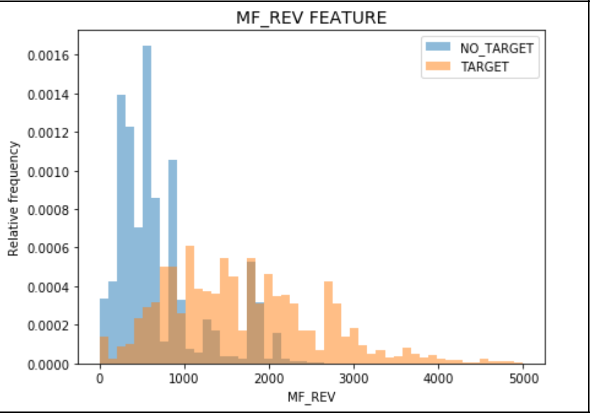
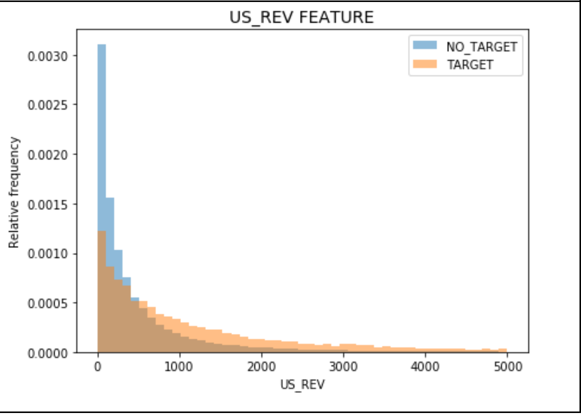











VOICE_VPN_MIN FEATURE
NO_TARGET
TARGET
Relative frequency
VOICE_VPN_MIN

VOICE_ONNET_MIN FEATURE
NO_TARGET
TARGET
Relative frequency
VOICE_ONNET_MIN

VOICE_OFFNET_MIN FEATURE
NO_TARGET
TARGET
Relative frequency
VOICE_OFFNET_MIN

VOICE_INT_MIN FEATURE
NO_TARGET
TARGET
Relative frequency
VOICE_INT_MIN

SMS_NAT FEATURE
NO_TARGET
TARGET
Relative frequency
SMS_NAT

SMS_INT FEATURE
NO_TARGET
TARGET
Relative frequency
SMS_INT

GPRS_NAT_MB FEATURE
NO_TARGET
TARGET
Relative frequency
GPRS_NAT_MB

OS
Android
iPhone
Proprietary
Windows
BlackBerry

MF_REV FEATURE
NO_TARGET
TARGET
Relative frequency
MF_REV

US_REV FEATURE
NO_TARGET
TARGET
Relative frequency
US_REV

FEATURE NAME	DESCRIPTION (Orange bar – TARGET, Blue bar – NO_TARGET)
TIME_ID	Graph shows seasonality patterns among both groups in dataset. We clearly see that during summer months, number of roaming users is higher (especially during July and August), comparing to other months presented in dataset. Users who have activated the service are more likely to buy it in March, June, July, November and December. This actually makes sense, because users travel more frequently during these months for e.g. vacation reasons.
SERVICE_SEG	Graph shows that users from Go(Gold) segment are more likely to activate roaming add-on, comparing to other segments. Users who belong to Go segment are more valuable ones (they pay higher for the mobile services), so it is expected from them to be more rational when using mobile services in roaming.
AP_NAME	Graph shows us that users who have activated Travel Sure 50 tariff plan are more likely to activate roaming add-on, which is expected, since users pay higher in order to use services for this tariff plan.
ACT_VOL	Graph shows us how roaming usage (SMS + VOICE + GPRS_MB) separates takers and no_takers. We clearly see that users who use more than 10 units are more likely to buy the add-on.
REV	Graph shows us how roaming revenue (SMS + VOICE + GPRS_MB) separates takers and no_takers. We see that users who spend more than 250 price units are more likely to buy the add-on.
NUM_OF_MONTHS	Graph shows us that users who have been in roaming in last 4 months are more likely to activate roaming add-on in following month. This information makes sense, because users who travel abroad more often are more rational in the sense of traffic usage.
VOICE_VPN_MIN	Graph shows us how voice vpn usage within national network separates takers and no_takers. We see that users who spend more than 25 minutes are more likely to buy the add-on.
VOICE_ONNET_MIN	This graph shows us how voice usage within Telenor network separates takers and no_takers. We see that users who spend more than 25 minutes are more likely to buy the add-on.
VOICE_OFFNET_MIN	Graph shows us how voice usage outside Telenor network separates takers and no_takers. We see that users who spend more than 40 minutes are more likely to buy the add-on.
VOICE_INT_MIN	Graph shows us how voice usage towards international network, from national network, separates takers and no_takers. We see that users who spend more than 0 minutes are more likely to buy the add-on. This feature obviously doees not clearly separates takers from no_takers.
SMS_NAT	Graph shows us how number of sent SMS messages within national network separates takers and no_takers. We see that users who send more than 25 messages are more likely to buy the add-on.
SMS_INT	Graph shows us how number of SMS messages sent towards international network, from national network, separates takers and no_takers. We see that users who send more than 0 messages are more likely to buy the add-on. This feature obviously doees not clearly separates takers from no_takers.
GPRS_NAT_MB	Graph shows us how internet usage within national network separates takers and no_takers. We see that users who use more than 180 MB are more likely to buy the add-on.
OS	Graph shows which users are more likely to buy add-on regarding OS they run on their mobile phones. It seems that iPhone and Blackberry users are more likely to activate this add-on comparing to other groups.
MF_REV	Graph shows us some nice information on how fixed monthly fee (in-tariff bundle) that users pay, clearly separates takers and no_takers. We clearly see that users who pay higher then 1000 price units (p.u.) are more likely to buy the add-on.
US_REV	Graph shows us how revenue over the bundle separates takers and no_takers. We clearly see that users who pay higher then 500 price units (p.u.) are more likely to buy the add-on.

Algorithms and Techniques

This section covers algorithms and techniques that are used in this project. Overview of each technique is shown. For all techniques, features from train set will be used for training purposes, and the same test data for all models will be used to perform tests on unseen data, in order to track performance and make comparisons against benchmark.

Naive predictor

This algorithm will use classification criteria based on visual observations of dataset. The algorithm should target customer as a TARGET if the value of a selected feature corresponds to defined value range. A list of combined conditions by using features would give a final result from this algorithm that should be compared with actual state of class variable in order to calculate evaluation metrics (precision, recall, F-Score).

Gaussian Naive Bayes

The reason why this algorithm has been chosen for this problem is:

- Performs well when there are a lot of features
- Computationally fast
- Simple to implement
- Good performance with high dimensional data

The reason this algorithm might not be a good option is for potential dependence of input data and data not normally being distributed.

Random Forest

The general idea behind this boosting algorithm is dividing data into several portions. Then, a relatively weak classifier/regressor is used to process, and then all of them are combined. Random forest is flexible and can enhance the accuracy/performance of the weak algorithm to a better extent, at the expense of heavier computational resources required. Random Forests train each tree independently, using a random sample of the data. This randomness helps to make the model more robust than a single decision tree, and less likely to overfit on the training data. There are typically two parameters in RF - number of trees and no. of features to be selected at each node.

Gradient Boosting

This ensamble method build trees one at a time, where each new tree helps to correct errors made by previously trained tree. With each tree added, the model becomes even more expressive. There are typically three parameters - number of trees, depth of trees and learning rate, and the each tree built is generally shallow.
Training process takes longer time because of the fact that trees are built sequentially. However benchmark results have shown GBDT are better learners than Random Forests.

Campaign Simulation

This technique will help to evaluate potential possbilities of our optimized model. This additional section should provide an insight on how this trained model could potentially be used in practice. Since we did not apply this algorithm for making real offers, we cannot track real effect of the campaign success. In that way, with this algorithm we can only track as-is situation, and measure the effect without machine learning algorithm being applied. In that way, we can measure as-is take-rate scores and expect that, when the model prediction from this project is applied, the take-rate should be higher.
In order to make target list for the month t+1, users that were selected as buyers by machine learning algorithm in month t should be selected as potential target for month t+1 (these are TP and FP groups of users from previous month, if we look at confusion matrix). Take rates from TP and FP groups should be observed separately, in order to see in what percent users that activated the add-on in month t (TP group) activate add-on in the next month as well. Users from FP group should be considered as potential targets (new buyers, since this group of users did not activate

the service in previous month). FP group of users could be a measure of campaign success, if we maximize number of takers during real campaigns using machine learning. One additional rate that will be considered is roaming rate - percentage of users from target list from month t that were in roaming in month t+1 (not minding if that user has activated service or not). Since we have 4 months of test data, 3 campaigns will be created (1-2, 2-3, and 3-4 month pairs) in order to test how our model predicts potential buyers for the next month. Model prediction scores of target lists should be plotted accordingly.

Benchmark

In this section, results from naive predictor benchmark and unoptimized learning algorithms will be demonstrated.

Naive Predictor

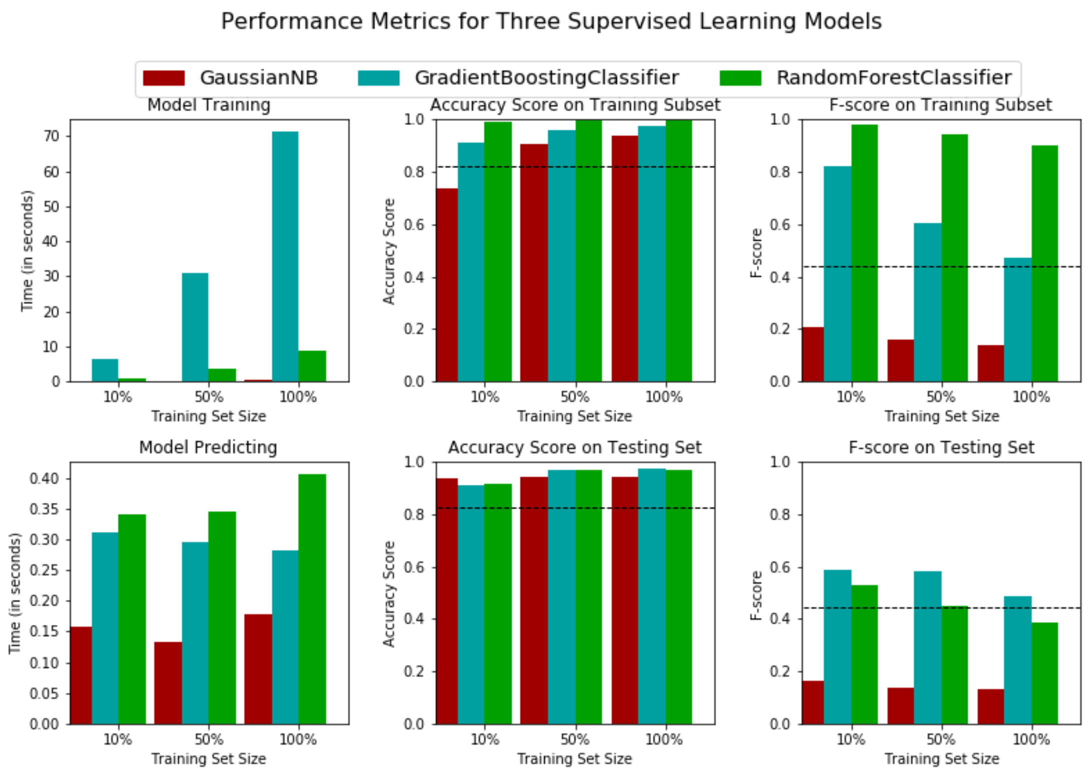
The results clearly show that naive predictor has best F-Score when only MF_REV feature is used as a classification feature.

FEATURE NAME	CRITERIA	PRECISION	RECALL	ACCURACY	F2-SCORE
MF_REV	>1.000	0.1632	0.7729	0.8230	0.4424
US_REV	>500	0.0918	0.8751	0.6290	0.3234
ACT_VOL	>10	0.0838	0.9188	0.5723	0.3072
REV	>250	0.0769	0.9328	0.5241	0.2892

From table above, we may notice drop in all evaluation metrics, except for recall, as we add more classification criteria to the model. Precision drops down because there are a lot of FP(False Positive) cases as we add more criterias. On the other hand, our naive predictor classifies excelent TARGET class (buyers of add-on service), with recall=0.9328 in final result. Just because we used criterias that are dominant comparing to other class after a certain value, we need to take into consideration how much the precision of our model will suffer if we just say that users with more than e.g. 1000 price units spent on MF_REV will definitely buy the add-on. The cost of bad precission is exactly the area of NO_TARGET class under the curve for MF_REV with value greater than 1000. As we add more feature criterias, this precission drops down for the same reason as for MF_REV. There is clearly no visible criteria that will retain both precision and recall metrics to a reasonable ratio. The idea of machie learning algorithms is to find this precission/recall trade-off in order to make more consistent predictions on our test sets.

Learning algorithms

Three learning algorithms were used for training on different percent of training data. In order to test model training and prediction performance, train set is downscaled by randomly subsampling NO_TARGET class to a lower percentage of data. Subsampling is done for each month separately, with same percentage, in order to retain the seasonality of number of NO_TARGET users in each month. For this project, subsampling is done using 10% , 50% and 100% of NO_TARGET training data.



From the comparison above, we see a horizontal line – it represents performance scores of our naive predictor compared to three learning algorithms. The goal of this benchmark section is to select the model that gives the best score and to set that model as a benchmark model, in order to compare scores obtained from this section to final optimized solution that is to be proposed.
From the graph we clearly see that, during tests on train data (300.000 samples of train data), Random Forest performs the best among the rest of the models. It has the highest accuracy (this metric is not so important in this project since we have an imbalanced dataset). F-Score of Random Forest during train procs is the highest. But during test process, Gradient Boosting method provided highest F-Score. Very important fact is that, with reducing the size of a NO_TARGET class, F-Score increases. If we look more carefully in, e.g. performance metrics of Gradient Boosting method when changing the size of train set, we see following:

% of NO_TARGET	PRECISION	RECALL	ACCURACY	F2-SCORE
10%	0.2937	0.7870	0.9110	0.5891
50%	0.6509	0.5662	0.9688	0.5814
100%	0.7750	0.4454	0.9710	0.4868

When increasing NO_TARGET class size in train set, precision rate increases and recall rate decreases. Since we observe the nature of users who buy the add-on, we want to emphasize recall more, since we want to know in what percent our algorithm trully detects users that actually bought add-on. For that reason, F2-Score was used, in order to reduce impact of lower precision rate to F-Score. The same trend happened using Random Forest classifier.
Our naive predictor gave a F2-Score of 0.4424 in best scenario, using only one feature for classification. Gradient Boosting model resulted with F2-Score of 0.5891 with only 10% of NO_TARGET class size in training set.

Having all this in mind, our benchmark model will have following parameters:

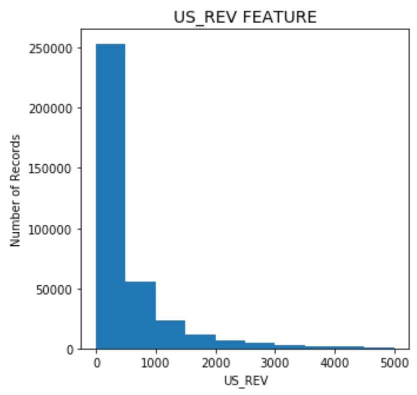
Model	Gradient Boosting
Precision	0.2937
Recall	0.7870
F2-Score	0.5891

Optimized solution will have to outperform this threshold criteria.

III. Methodology

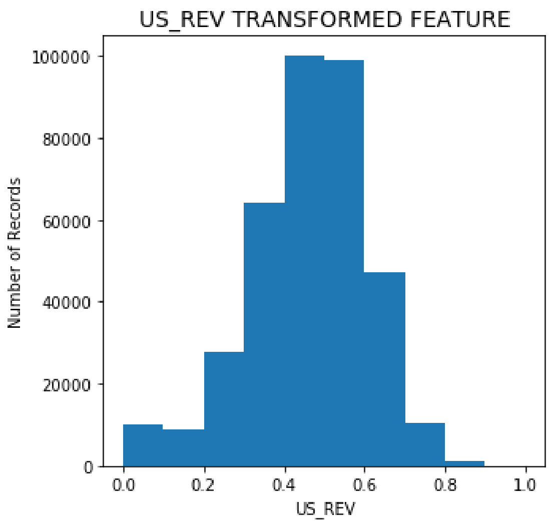
Data Preprocessing

In order for learning algorithms to perform properly, data pre-processing was done, for both continuous and categorical features. For categorical features, One-Hot encoding was performed, in order to transform categorical features into numerical features. For numerical features, each one was checked for skewed distribution. At the end, all of continuous features were transformed using the log-transform function. Finally, all continuous features were scaled to [-1,1] range. The example of skewed and then processed feature US_REV, is listed below.



The image above shows frequency histogram of number of users over revenue range. This feature is clearly skewed, so it needs to be transformed.

Figure below shows the same feature after log-transform function *log(1+x)*. Value 1 is added to x to ensure *log(0)* will never be calculated, when US_REV=0, since it would generate infinite value of that function. Also, min-max scaling function is applied to feature, retaining the shape of processed distribution. After preprocessing, our dataset has 39 features. Initially, there were 19 features, before One-Hot encoding was performed.



Dataset is divided into two parts:

- Train set: from 04/16 to 03/17
- Test set: from 04/17 to 07/17

Implementation

There are four realized implementations for this project.

Naive predictor

The implementation of this predictor considered building criteria for classification, regarding visual observations from data exploratory part. The predictor was tested on test data set in order to measure its performance using defined evaluation metrics. After each criteria was added, model classification performance was reported using naive_predictor function.

```
#Define naive test set range
start_naive_dt  = dt.datetime.strptime("2017-04-01", '%Y-%m-%d').date()
end_naive_dt    = dt.datetime.strptime("2017-07-01", '%Y-%m-%d').date()
naive_test_set['NAIVE_PRED']='NO_TARGET'

#Set first criteria
naive_test_set.loc[naive_test_set['MF_REV'] > 1000, ['NAIVE_PRED']] = 'TARGET'
fn.naive_predictor(naive_test_set)

#Set second criteria
naive_test_set.loc[naive_test_set['US_REV'] > 500, ['NAIVE_PRED']] = 'TARGET'
fn.naive_predictor(naive_test_set)

#Set hird criteria
naive_test_set.loc[naive_test_set['ACT_VOL'] > 10, ['NAIVE_PRED']] = 'TARGET'
fn.naive_predictor(naive_test_set)

#Set fourth criteria
naive_test_set.loc[naive_test_set['REV'] > 250, ['NAIVE_PRED']] = 'TARGET'
fn.naive_predictor(naive_test_set)
```

```
def naive_predictor(data):
    tp = float(len(data[(data['NAIVE_PRED']=="TARGET") & (data['FLAG']=="TARGET")]))
    tn = float(len(data[(data['NAIVE_PRED']=="NO_TARGET") & (data['FLAG']=="NO_TARGET")]))
    fp = float(len(data[(data['NAIVE_PRED']=="TARGET") & (data['FLAG']=="NO_TARGET")]))
    fn = float(len(data[(data['NAIVE_PRED']=="NO_TARGET") & (data['FLAG']=="TARGET")]))

    #Calculate precision
    precision = tp/(tp+fp)

    #Calculate recall
    recall    = tp/(tp+fn)

    #Calculate accuracy
    accuracy = (tp+tn)/(tp+tn+fp+fn)

    #Calculate F-score using the formula above for beta = 2
    beta = 2
    fscore = (1+np.square(beta))*(precision*recall)/(np.square(beta)*precision+recall)

    # Print the results
    print("TP={}".format(tp))
    print("FP={}".format(fp))
    print("FN={}".format(fn))
    print("TN={}".format(tn))
    print("Precision={}".format(precision))
    print("Recall={}".format(recall))
    print "Naive Predictor: [Accuracy score: {:.4f}, F-score: {:.4f}]".format(accuracy, fscore)
```

Learning algorithms

For three described algorithms: GaussianNB, GradientBoostingClassifier and RandomForestClassifier, train_predict function is invoked. This function downsamples NO_TARGET class, if specified with sample_frac scaling coefficient less than 1. Then, the model is trained on selected train data and tested on the test data. All evaluation metrics are recorded for each case in order to perform comparison between models/train set size for choosing optimal model among three models.

```
def train_predict(learner, sample_frac, train_set, test_set):

    results = {}
    #Number of samples from train set to be tested on trained model
    num_of_samples = 30000

    train_set_no_target = pd.DataFrame(train_set[(train_set['FLAG'] == 0)])
    train_set_target    = train_set[(train_set['FLAG'] == 1)]

    train_set_no_target = train_set_no_target.groupby(['TIME_ID'])
    train_set_no_target = train_set_no_target.apply(lambda x: x.sample(frac=sample_frac/100.0))

    train_set = pd.concat([train_set_no_target, train_set_target])

    #Shuffle train set
    train_set = train_set.sample(frac=1)

    train_set_features = train_set.drop(['FLAG', 'TIME_ID', 'SUBSCRIPTION_ID'], axis = 1)
    train_set_labels   = train_set['FLAG']

    test_set_features  = test_set.drop(['FLAG', 'TIME_ID', 'SUBSCRIPTION_ID'], axis = 1)
    test_set_labels    = test_set['FLAG']

    # Fit the learner to the training data
    start = time() # Get start time
    learner.fit(train_set_features, train_set_labels)
    end = time() # Get end time

    # Calculate the training time
    results['train_time'] = end - start

    # Get the predictions on the test set,
    # then get predictions on the first 30000 training samples
    start = time() # Get start time
    predictions_test = learner.predict(test_set_features)
    predictions_train = learner.predict(train_set_features[:num_of_samples])
    end = time() # Get end time

    tn, fp, fn, tp = confusion_matrix(test_set_labels,predictions_test).ravel()#

    tn=float(tn)
    fp=float(fp)
    fn=float(fn)
    tp=float(tp)

    precision = tp/(tp+fp)
    recall    = tp/(tp+fn)

    #Calculate accuracy
    accuracy = (tp+tn)/(tp+tn+fp+fn)

    #Calculate F-score using the formula above for beta = 2
    beta = 2
    fscore = (1+np.square(beta))*(precision*recall)/(np.square(beta)*precision+recall)

    #Calculate the total prediction time
    results['pred_time'] = end-start

    #Compute accuracy on the first 300 training samples
    results['acc_train'] = accuracy_score(train_set_labels[:num_of_samples],predictions_train)

    #Compute accuracy on test set
    results['acc_test'] = accuracy_score(test_set_labels,predictions_test)

    #Compute F-score on the the first 300 training samples
    results['f_train'] = fbeta_score(train_set_labels[:num_of_samples],predictions_train,beta=2)

    #Compute F-score on the test set
    results['f_test'] = fbeta_score(test_set_labels,predictions_test,beta=2)

    print ("{} trained on {} % of dataset with {} samples.".format(learner.__class__.__name__, sample_frac, 
train_set_labels.shape[0]))
    print ("tn={}, fp={}, fn={}, tp={}, precision={}, recall={}, accuracy={}, fscore={}.".format( tn, fp, fn, 
tp, precision, recall, accuracy, fscore ))

    # Return the results
    return results
```

Fine-tunning

After the best model is selected, fine tunning of parameters will be realized. Sine GMB model is chosen as the best model, optimization of this model using GridSearchCV is shown:

```
# Import 'GridSearchCV', 'make_scorer', and any other necessary libraries
from sklearn.metrics import make_scorer
from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import fbeta_score, accuracy_score, recall_score, precision_score


# Define train and test set ranges
start_train_dt = dt.datetime.strptime("2016-04-01", '%Y-%m-%d').date()
end_train_dt   = dt.datetime.strptime("2017-03-01", '%Y-%m-%d').date()

start_test_dt  = dt.datetime.strptime("2017-04-01", '%Y-%m-%d').date()
end_test_dt    = dt.datetime.strptime("2017-07-01", '%Y-%m-%d').date()


# Initialize the classifier
gbc = GradientBoostingClassifier(random_state=100)

# Create the parameters list for tunning
parameters = dict(n_estimators=[100,150],
                  learning_rate=[0.1, 0.01, 0.001, 0.0001],
                  max_depth=[4,6,10,12])

# Make an fbeta_score scoring object
scorer = make_scorer(fbeta_score, beta=2)

# Perform grid search on the classifier using 'scorer' as the scoring method
grid_obj = GridSearchCV(gbc, param_grid = parameters,scoring=scorer)

# Fit the grid search object to the training data and find the optimal parameters
grid_fit = grid_obj.fit(train_set[1], train_set[2])

# Get the estimator
best_clf = grid_fit.best_estimator_

test_set_features  = test_set.drop(['FLAG', 'TIME_ID', 'SUBSCRIPTION_ID'], axis = 1)
test_set_labels    = test_set['FLAG']

# Make predictions using the unoptimized and model
predictions = (gbc.fit(train_set[1], train_set[2])).predict(test_set_features)
best_predictions = best_clf.predict(test_set_features)

# Report the before-and-afterscores
print "Unoptimized model\n------"
print "Accuracy score on testing data: {:.4f}".format(accuracy_score(test_set_labels, predictions))
print "Precision score on testing data: {:.4f}".format(precision_score(test_set_labels, predictions))
print "Recall score on testing data: {:.4f}".format(recall_score(test_set_labels, predictions))
print "F-score on testing data: {:.4f}".format(fbeta_score(test_set_labels, predictions, beta = 2))
print "\nOptimized Model\n------"
print "Final accuracy score on the testing data: {:.4f}".format(accuracy_score(test_set_labels, 
best_predictions))
print "Final Precision score on the testing data: {:.4f}".format(precision_score(test_set_labels, 
best_predictions))
print "Final Recall score on the testing data: {:.4f}".format(recall_score(test_set_labels, 
best_predictions))
print "Final F-score on the testing data: {:.4f}".format(fbeta_score(test_set_labels, best_predictions, beta 
= 2))
```

Campaign offer simulation system

This system serves for calculating campaign success metrics, in order to observe how our model predicts potential buyers of add-on for the next month.

Trained model will be tested on separate months from the test set, in order to generate campaign offers. For month t, there will be actual and predicted classes of our target variable. Using FP and TP groups, we will create target lists for the month t+1 and see how many customers from these 2 separate groups have activated the service. Since our dataset consists of 4 months of data, 4 test sets will be created:

```
test_sets = fn.generate_test_sets(data, start_test_dt, end_test_dt)
test_1 = fn.test_predict(1, best_clf, test_sets[1], test_sets[2], test_sets[3])
test_2 = fn.test_predict(2, best_clf, test_sets[1], test_sets[2], test_sets[3])
test_3 = fn.test_predict(3, best_clf, test_sets[1], test_sets[2], test_sets[3])
test_4 = fn.test_predict(4, best_clf, test_sets[1], test_sets[2], test_sets[3])
```

```
def test_predict(test_month_index, clf, test_sets_features, test_sets_labels, test_sets_ID):
    #generate probabilities of classification
       pred_prob  = clf.predict_proba(test_sets_features[test_month_index])
    #generate predicted class
       pred_class = clf.predict(test_sets_features[test_month_index])

       #generate confusion matrix
    tn, fp, fn, tp = confusion_matrix(test_sets_labels[test_month_index], pred_class).ravel()

    tn=float(tn)
    fp=float(fp)
    fn=float(fn)
    tp=float(tp)

    #Calculate precision
    precision = tp/(tp+fp)

    #Calculate recall
    recall    = tp/(tp+fn)

    #Calculate accuracy
    accuracy = (tp+tn)/(tp+tn+fp+fn)

    #Calculate F-score using the formula above for beta = 2
    beta = 2
    fscore = (1+np.square(beta))*(precision*recall)/(np.square(beta)*precision+recall)

    # Print the results
    print("TP={}".format(tp))
    print("FP={}".format(fp))
    print("FN={}".format(fn))
    print("TN={}".format(tn))
    print("Precision={}".format(precision))
    print("Recall={}".format(recall))
    print "ML Predictor: [Accuracy score: {:.4f}, F-score: {:.4f}]".format(accuracy, fscore)

    return(pd.DataFrame({'C0_SCORE':pred_prob.T[0],'C1_SCORE':pred_prob.T[1], 'PRED_CLASS':pred_class,
'ACT_CLASS':test_sets_labels[test_month_index], 'SUBSCRIPTION_ID':test_sets_ID[test_month_index]}))
```

This function is invoked in order to generate campaign offer results. Take rate and roaming rate implementations are listed in code block below.

```
def campaign_simulation(target_data, activations_data, act_class, pred_class, score_treshold ):

       #Define from which group (FP or TP), target list for the t+1 month will be created, in month t.
       #ACT_CLASS is the actual class in month t
       #PRED_CLASS is the predicted class in month t
       #Knowing those two vectors, we can select TP or FP users in order to observe whether users from one
of these groups in month t will activate the service in the month t+1.
       #Minimum score of prediction is 0.5.
       #Score threshold can be increased, in order to track take rate and roaming rate of users with higher
prediction score.

    target_list    = target_data[ (target_data.ACT_CLASS==act_class) & (target_data.PRED_CLASS==pred_class) &
(target_data.C1_SCORE>score_treshold)]
```

```
        #Extract IDs of all users from the month t+1
        roamers        = pd.DataFrame(activations_data['SUBSCRIPTION_ID'])

        #Match users from target list to list of roamers, in order to find number of actual roaming users
from previous month t
        actual_roamers = pd.merge(target_list, roamers, on=['SUBSCRIPTION_ID'], how='inner')

        #Match users that activated add-on service
    activations    = activations_data[activations_data.ACT_CLASS==1]

        #Select IDs only
    activations    = pd.DataFrame(activations['SUBSCRIPTION_ID'])

        #Find actual takers from our target list
    activations    = pd.merge(target_list, activations, on=['SUBSCRIPTION_ID'], how='inner')

        #Plot score histogram of takers that were correctly classified by our algorithm
    plt.hist(activations['C1_SCORE'])

        #Print campaign statistics
    print("Total number of targeted users: \t\t\t{}".format( len(target_list)))
    print("Total number of roaming users in targeted month: \t{}".format( len(roamers)))
    print("Total number of correctly targeted roaming users: \t{}".format( len(actual_roamers)))
    print("Correctly targeted roaming users rate: \t\t\t{}%".format(
float(len(actual_roamers))/float(len(target_list))*100))
    print("Total number of activations: \t\t\t\t{}".format( len(activations)))
    print("Take rate: \t\t\t\t\t\t{}%".format( float(len(activations))/float(len(actual_roamers))*100))
```

Sample code of invoking the *campaign_simulation*, for 3 campaigns is:

```
#April-May target list
#Evaluation from FP target list
fn.campaign_simulation(target_data = test_1, activations_data = test_2, act_class = 0, pred_class = 1,
score_treshold = 0.5)

#Evaluation from TP target list
fn.campaign_simulation(target_data = test_1, activations_data = test_2, act_class = 1, pred_class = 1,
score_treshold = 0.5)


#May-June target list
#Evaluation from FP target list
fn.campaign_simulation(target_data = test_2, activations_data = test_3, act_class = 0, pred_class = 1,
score_treshold = 0.5)

#Evaluation from TP target list
fn.campaign_simulation(target_data = test_2, activations_data = test_3, act_class = 1, pred_class = 1,
score_treshold = 0.5)


#June-July target list
#Evaluation from FP target list
fn.campaign_simulation(target_data = test_3, activations_data = test_4, act_class = 0, pred_class = 1,
score_treshold = 0.5)

#Evaluation from TP target list
fn.campaign_simulation(target_data = test_3, activations_data = test_4, act_class = 1, pred_class = 1,
score_treshold = 0.5)
```

Refinement

Among three selected models, the optimal one was chosen, Gradient Boosting model. Improvements of model included change of following parameters:

- Train set NO_TARGET class downsampling – the train set was downscaled to 10% of NO_TARGET class, in order to maximize recall rate and train the model with additional increased parameters in a reasonable span of time.
- GridSearchCV method of fine-tunning the model using the predefined set of potential parameters.

Initial solution had lower scores, regarding precision, recall and F-Score. The improved solution provided higher evaluation metric values, so the optimized model trully gave better classification results. The compromise between precision/recall is reached, with aim to put more emphasize on recall rate, since we want to be sure our model correctly recognizes users who are currently buying the product. The precision rate might improve during the time if we costantly send offers to *FP* group of users in order for them to activate the add-on service. Also, during time user behaviour might change, so as the rules of buying the product. Then, the model shoul be retrained in order to follow the underlaying law of customer behaviour that might change during the time, so the precision and recall as well.

IV. Results

Model Evaluation and Validation

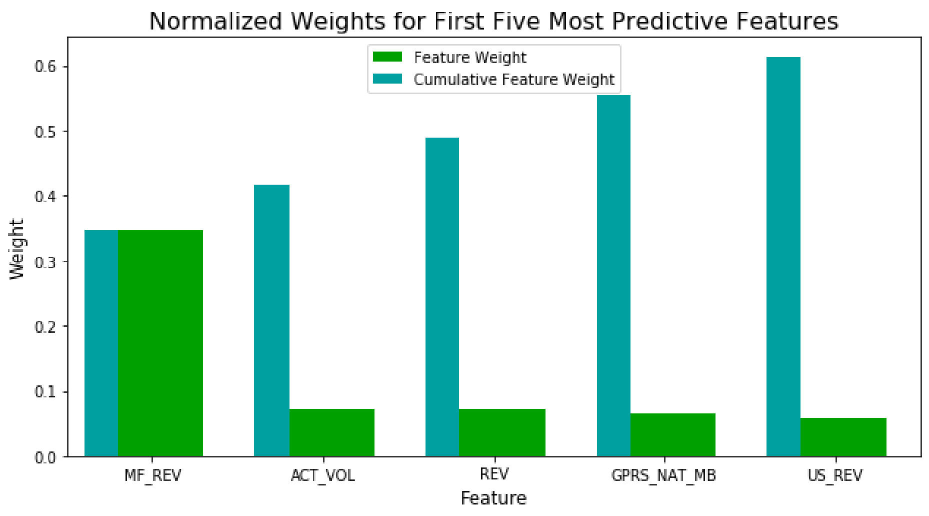
Initially, the goal was to obtain better classification performance using Gradient Boosting algorithm. The scores that should be outperformed are listed in table below.

Model	Gradient Boosting Unoptimized
Precision	0.2937
Recall	0.7870
F2-Score	0.5891

Fine-tuning of Gradient boosting model was performed. Grid search was used (GridSearchCV) in order to tune GBM with different combination of parameters. For optimization purposes, following list of GBM parameters with value combinations is used:

Parameter name	Set of possible values
n_estimators	[100,150]
learning_rate	[0.1, 0.01, 0.001, 0.0001]
max_depth	[4,6,10,12]

From the table listed above, there are total 2x4x4=32 possible configurations of a GBM model for training. Among 32 possible options, GridSearchCV found an optimized solution.
The idea of parameter tuning using GridSearchCV is to find optimal set of model parameters, in order to outperform performance score from benchmark.
GBM model was chosen as a final model because it outperformed Naive predictor, Random forest and GaussianNB in test set, using default parameters.
This optimization process was performed using pertubations of trainig set, by subsampling it to a lower ratio few times, and this process did not affect significantly the performance score on test set.
From our trained optimized model we may extract feature importance in building the model.



From the image above, we see how optimized GBM model ranked input features by importance. It seems that MF_REV played a huge role in classification, as we could expect, since we have noticed nice separation of classes using visual observations. REV parameter, ACT_VOL, GPRS_NAT_MB and US_REV were also selected as improtrant predictive features.

In order to make use of this evaluated results, some tests were performed by sending simulated campaign offers to customers. This is described in the next section.

Justification

This section will provide final performance scores of developed solution that will be compared against benchmark scores that needs to be outperformed, in order to consider this solution successfull.

Model	Naive_Predictor	GBM_Unomptimized	GBM_Optimized
Precision	0.1632	0.2937	0.3842
Recall	0.7729	0.7870	0.8230
F2-Score	0.4424	0.5891	0.6700

From this table we can clearly see how our optimized GBM model outperformed both naive predictor and unoptimized GBM model. Final solution has reached 0.67 F2-Score, which is better than previous two models. We clearly see that improvement came from the precision score increase. The recall score has improved as well, but in smaller rate, comparing to precision rate.

Optimized model was executed with test data in order to generate campaign offers. There were total 3 campaigns:

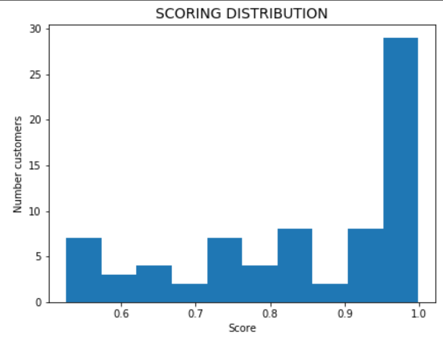
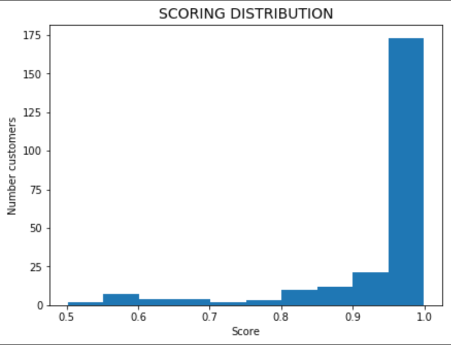
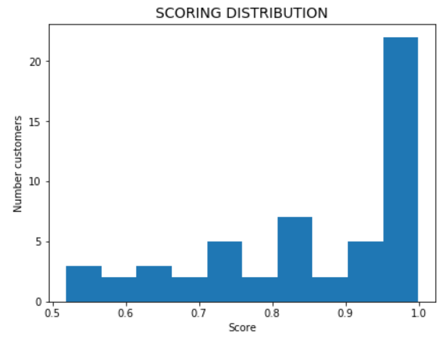
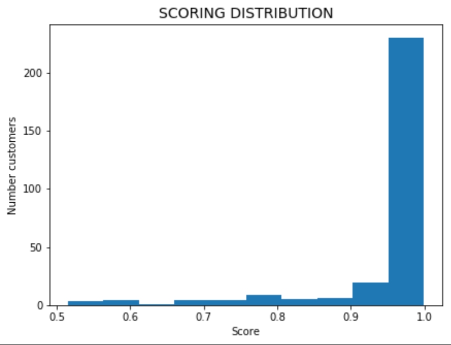
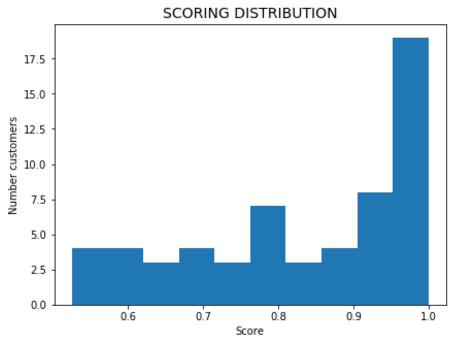
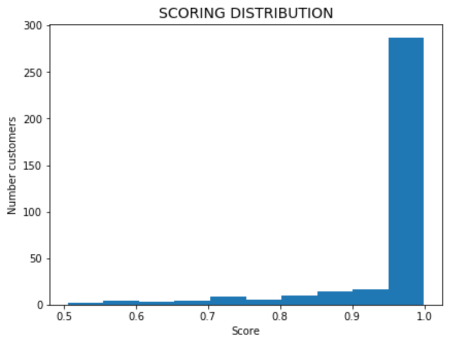
Campaign Number	Target list created	Target list sent
1	April	May
2	May	June
3	June	July

From table below, we may notice how take rate differs when targeting customers that already bought add-on in previous month (TP) from users that were selected by GBM as buyers but didn't buy the add-on in previous month (FP). Take rate of TP target list is much higher (above 50% in all 3 campaigns) , compared to FP target list (around 5%-9% take rate). This leads to conclusion that users that activate add-on in previous months are more likely to activate the add-on in next months – these users obviously know about existence of this add-on and its benefits. On the other hand, FP target list has much smaller take rate, which makes room for creating campaign offers for them in order to increase number of activations and revenue from activations as well. Considering roaming rate, in all cases, the model predicted that around 70% of target list will actually be in roaming in the next month, which is a good prediction score.

Campaign Number	Description	FP Target List	TP Target List
1	Total number of targeted users	1819	610
	Total number of roaming users in targeted month	21511	21511
	Total number of correctly targeted roaming users	1312	440
	Correctly targeted roaming users rate	72.12%	72.13%
	Total number of activations	74	238
	Take rate	5.64%	54.1 %
2	Total number of targeted	892	726

	users		
	Total number of roaming users in targeted month	25018	25018
	Total number of correctly targeted roaming users	589	509
	Correctly targeted roaming users rate	66.03%	70.11%
	Total number of activations	53	285
	Take rate	8.99%	55.99%
3	Total number of targeted users	1146	913
	Total number of roaming users in targeted month	33708	33708
	Total number of correctly targeted roaming users	857	707
	Correctly targeted roaming users rate	74.78%	77.43%
	Total number of activations	59	354
	Take rate	6.88%	50.07%

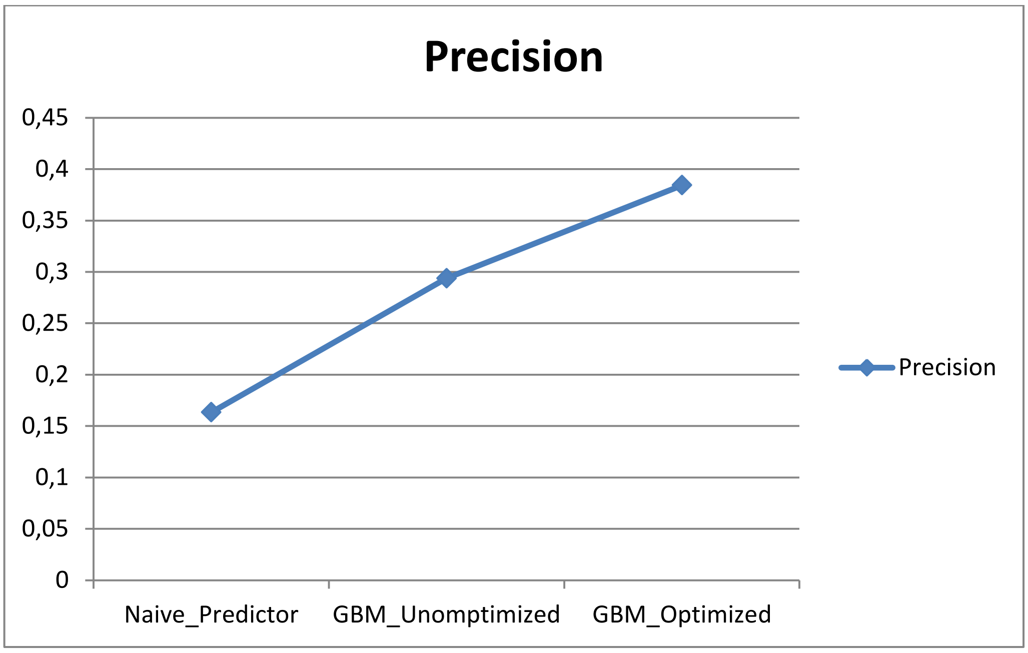
If we take a closer look into users who have activated add-on from our two target lists, we may observe scoring distribution of our takers. In the case of both target list, we see nice increase of number of takers when the score increases. This increase is stronger in TP type of target list. One of the potential reasons for this is because recall rate is higher than precision rate in our test set.

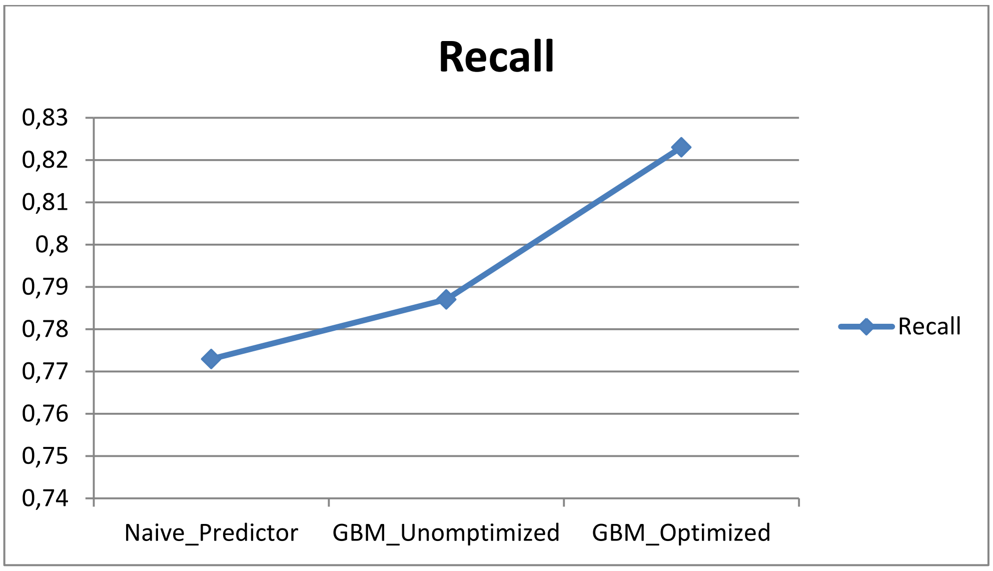
Campaign Number	FP Score Distribution	TP Score Distribution
1		
2		
3		

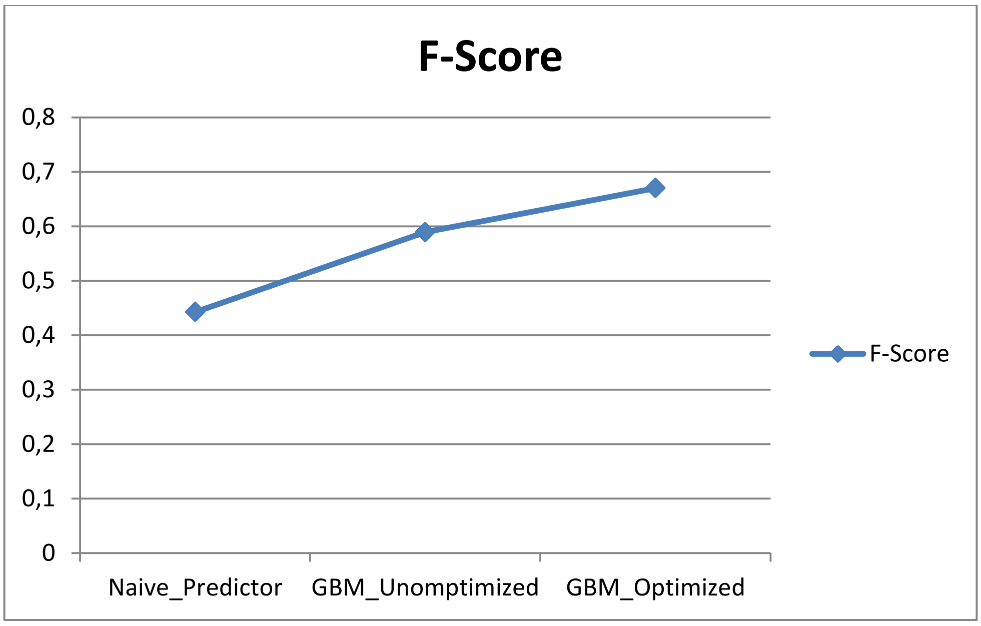
V. Conclusion

Free-Form Visualization

This section will provide visual representations of this project achievement.







In figures above, we clearly see how our optimized model outperformed other two models from benchmark. This leads us to conclusion that our model is successfull in terms of performance score.

Reflection

In summary, on high level, this project reflects in following workflow:

- **Dataset exploration** – exporatory visualizations were observed, in order to get to know the data.
- **Building naive predictor** – preditor was built using different criterias, in order to create basic classifier.
- **Data pre-processing** – Data was preprocessed using log-transform, minMax scaler and One-Hot encoding.
- **Train machine learning algorithms** – Three learning algorithms were trained and the best one was chosen.
- **Optimizing the learning algorithm** – The best algorithm was optimized using GridSearchCV method.
- **Final benchmark** – At the end, optimized model performance on the test set was compared to benchmark model performance.
- **Demontration on making campaign offers** – Using optimized model, a set of simulated campaign offers was created in order to track potential buyers and takers and measure as-is take rate of users, without affecting the population of customers with our model. True effects of this model will be tracked after this model is released in production.

The interesting aspect found in this research is that users who have activated the add-on in previous months are more likely to buy the same add-on in the next month, comparing to user that do not activated this servie earlier. This gives us more space for targeting users who have similar behaviuor as takers, in order to send them offers. Also, MF_REV is found to be the most importan feature for classification among other features in dataset. Machine learning will be of a great help in this step of optimizing campaign offers, but it will be difficult to gain model success in the beginning of the development, since we need to target more customers in order to make a promotion of this add-on. The other challenge is to investigate why users who have similar behaviour as buyers do not activate the add-on. In this phase, different campaign offer channels should be considered as well (SMS, email, mobile app), in order to increase probability of our potential buyers to buy the product.

Improvement

The model that was created in this project has a good performance metric, and it might get even more effective, as we add more months of user behaviour to our dataset.
There are various ways to make the improvement to this model, e.g.

- Change the existing train set size, by changing the downscaling coefficient
- Add more data
- Add some new features that were not considered by this dataset
- Try some other learning algorithms, in order to increase performance

VI. References

[1] M. Cioca et al., *"Machine Learning and Creative Methods Used to Classify Customers in a CRM Systems"*, Applied Mechanics and Materials, Vol. 371, pp. 769-773, 2013
[2] Hastie T., Tibshirani R., Friedman J., *"Overview of Supervised Learning",* The Elements of Statistical Learning. Springer Series in Statistics. Springer, New York, NY, 2009
[3] L. Zhao et al., *"An Overview of the Recommender System"*, Applied Mechanics and Materials, Vol. 302, pp. 787-791, 2013