UDACITY CAPSTONE PROJECT
Kaggle: Santander Customer Satisfaction

Juho Salminen

Definition

Problem overview

The problem for this capstone project comes from Kaggle machine learning competition site (Kaggle 2016). Consumer bank Santander wants to predict which customers are satisfied and which are not, based on information they have about the customers. The problem is important, because unsatisfied customers will not use the services as much as satisfied customers, will not recommend the company to their friends, and may change service provider, causing losses to the company. Furthermore, they usually do not voice their concerns, making it difficult for the companies to respond. Solving this problem would also benefit the customers. When Santander is able to identify customers that are not satisfied with its services, it can intervene and try to fix the issues from which the unsatisfied customers are suffering. The task is to predict if a customer is satisfied or dissatisfied with their banking experience, based on several hundred anonymized features.

Problem statement

As a solution to the problem described above, a machine learning algorithm needs to be implemented that can predict customer satisfaction based on other information Santander has about them. In other words, the task is to classify customers to two groups. Most customers (roughly 96 %) are satisfied with their experience, which complicates the evaluation slightly, and makes the choice of metric even more important. If for instance correct predictions is used as a measurement, even a naive classifier that classifies all cases in one class will achieve impressive-looking 96 % accuracy. However, the selection of measurement is simplified, as Kaggle provides a measurement they use to evaluate competition submissions. The measurement is Area Under Receiver Operating Charasteristic Curve (AUROC).

Metrics and benchmark

Receiver operating characteristic (ROC) curve is a plot that illustrates the performance of a binary classifier as its discrimination threshold is varied (Wikipedia 2016). The curve is created by plotting the true positive rate against the false positive rate (FPR) at different threshold values.

The true positive rate measures how many correct positive results occur among all positive samples available during the test, while the false positive rate defines how many incorrect positive results occur among all negative samples available during the test.

$$TPR = \frac{\sum true\ positives}{\sum condition\ positive}$$

$$FPR = \frac{\sum true\ negative}{\sum condition\ negative}$$

A ROC space is defined by FPR and TPR as coordinate axes, and it shows the relative trade-off between true and false positives. The best ideal prediction would yield a point in the upper left corner of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives). A random guess gives a point along a diagonal line from the left bottom to the top right corners. Fittingly to current case, this result holds regardless of the base rates of positive and negative cases. Area under curve, or AUROC, is then the probability that a classifier ranks a randomly chosen positive instance higher than a randomly chosen negative instance. A random guess or all zeros could be used as a baseline prediction. Kaggle uses all zeroes in this case, giving AUROC score of 0.50, which corresponds to diagonal line. The implemented machine learning algorithm should do at least this well to be of any use for Santader in predicting customer satisfaction. Figure 1 depicts an example of ROC curve.

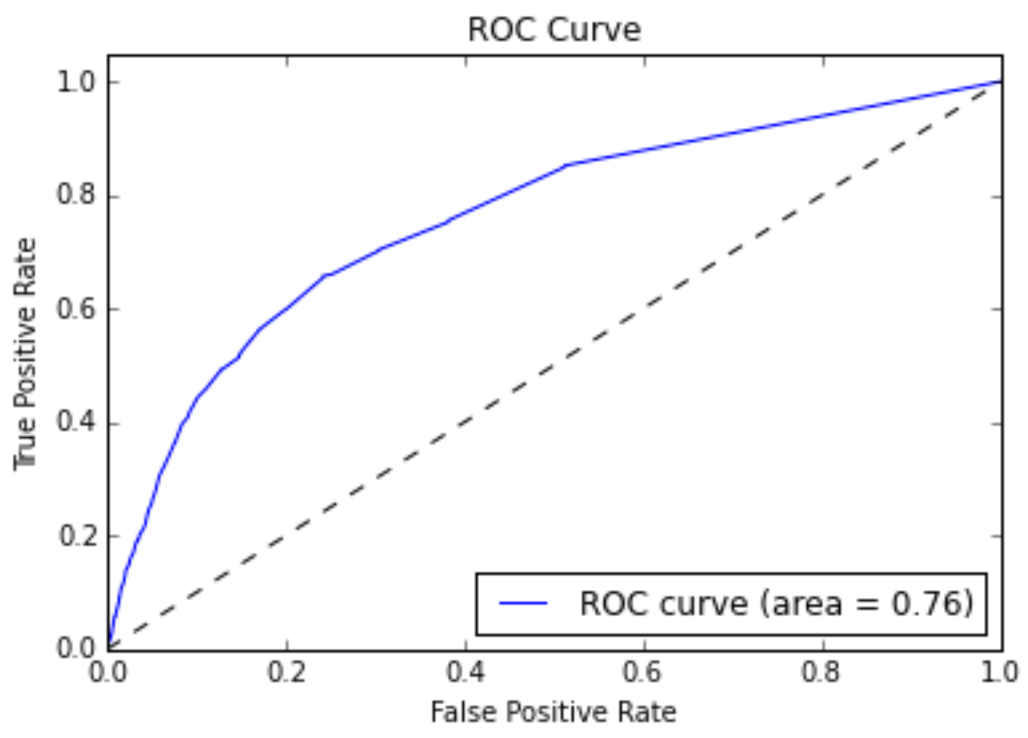


Figure 1. Example of a ROC curve. Diagonal dashed line implies a random guess and gives AUC score 0.5. Points above the line are better than random guesses. The blue line represents performance of a particular prediction algorithm when classification threshold is varied. The area under this curve is 0.76. Point 0,1 in the upper left corner of the diagram represents perfect classification: all predictions are correct.

Analysis

Data exploration and visualizations

The dataset provided by Kaggle consists of training and test datasets. Training set is used for training the algorithm, and test set is reserved for creating a submission to the competition. The training set consists of anonymized data on 76020 Santander customers. In addition to ID and TARGET variables there are 369 variables, with no missing values. Information on what the variables represent is not available. Such a large number of variables makes visual exploration challenging; R was used for initial exploration of all variables, but here only the main findings are reported. Figures 2-5 present histograms of different types of variables found in the dataset, grouped by TARGET value. 258 variables are integers and 111 are continuous. The integer variables might represent categorical data, which make warrant transformation to dummy variables. Numerical variables tend to have skewed distributions. Depending on the learning algorithm normalizing them might be necessary. The measurement variable is provided by Kaggle, and it is called TARGET. This variable is either 0 or 1 depending on if the customer is satisfied or unsatisfied. A quick visual exploration of distributions shows that there are not major differences between the distributions of variables whether the TARGET is one or zero. Therefore, the difference between satisfied and unsatisfied customers is likely an interaction between two or more variables, which makes visual exploratory investigation of relationships challenging when there are several hundred features. No single feature sticks out as particularly promising for predicting customer satisfaction. Main use of exploratory analysis in this case is therefore identifying possible errors in variables and data points.

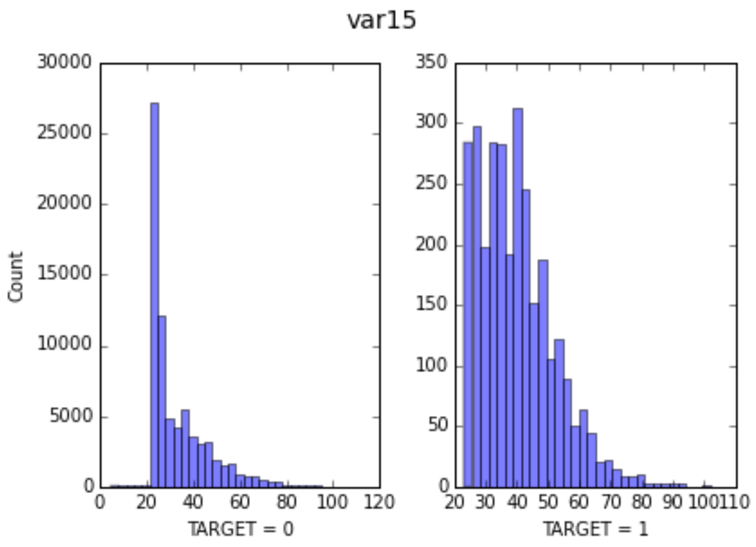


Figure 2. Right-skewed distributions of a continuous variable found in the Santander dataset.

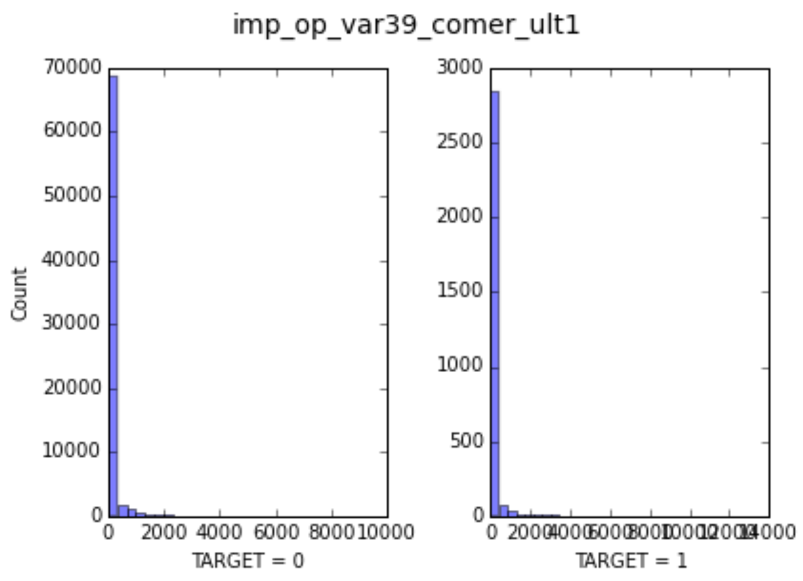


Figure 3. Extremely skewed distribution of a continuous variable found in the Santander dataset. Most observations are 0 with only a few larger values. There is no apparent difference between TARGET classes.

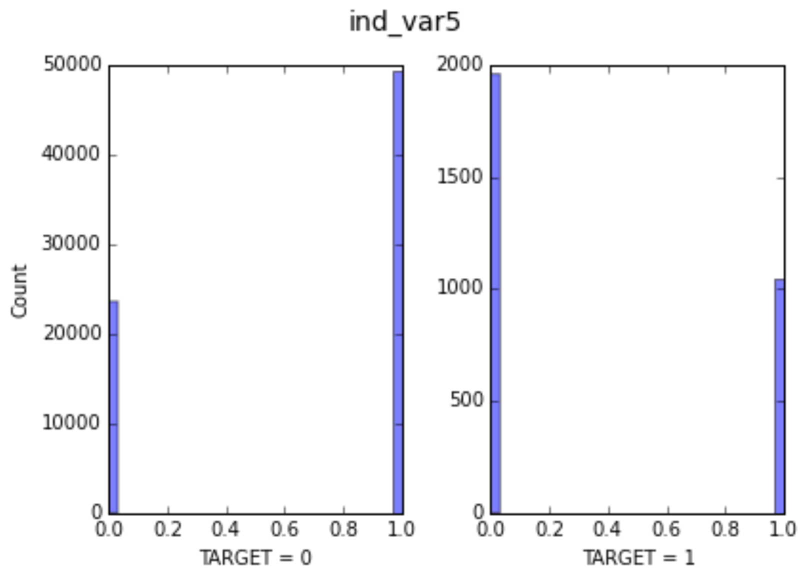


Figure 4. Distributions of an integer (binary) variable found in the Santander dataset. Observations of TARGET class 1 are more likely to have 0 value for this variable.

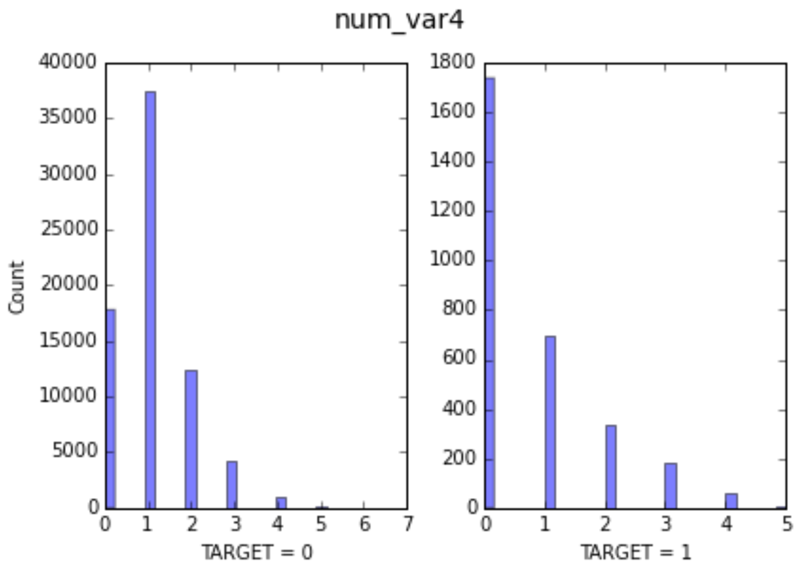


Figure 5. Distributions of an integer (non-binary) variable found in the Santander dataset. Observations of TARGET class 1 appear to be more likely to have 0 value for this variable, while 1 is the most typical value for TARGET class 0.

Algorithms and techniques

Suitable machine learning algorithms should predict customer satisfaction classification (binary variable) given other information on the customer. Many of the variables are categorical. Therefore, any algorithm that can classify observations in two groups and can deal with categorical variables could potentially be used. The range of applicable algorithms is thus huge, from variations of linear regression to support vector machines, random forests, Bayesian approaches, and even neural networks or "deep learning". As it seems there are no simple relationships between single variables and target classification, the chosen algorithm should take into account the interactions and possible non-linearities. Possible algorithms fitting these criteria include random forest, support vector machine and gradient boosting. Random forest is known as a well-performing "black box" algorithm. It is not easy to understand what is going on between the variables, but prediction performance is often good. Random forest can also deal with non-linearities and interactions between the variables. Support vector machine are also suitable for non-linearities, and it was considered to be one of the state-of-the art algorithms before the emergence of deep learning. Running time with the amount of data at hand might be an issue, though. Gradient boosting uses somewhat similar approach to random forest, and has performed well on Kaggle competitions in the past.

Data provided by Kaggle has already been transformed to machine-learning friendly format. There are no missing values, but some of the variables contain apparent errors or have always the same value. In one case there were only a few suspicious values, and they were imputed with median value. Other problematic variables were removed. Possible categorical variables are stored as integers and these might need converting. There is likely a lot of irrelevant information in the dataset. A subset of data may need to be selected for machine learning purposes. It is not clear what variables are the most suitable, but based on the visual exploration there is likely lots of correlation between several variables. Feature reduction is likely necessary, for instance by principal component analysis.

Methodology

Preprocessing

Data preprocessing included imputations on one variable and removal of several other, as described in table 1. A decision was made to initially use random forest algorithm, which makes it unnecessary to scale or normalize the features. Random forest can also be used for feature selection by plotting feature importances, which could be useful for improving performance. These considerations make random forest a good first-pass solution. Data was split to training and test sets, with 30 % of data retained for testing. Stratified sampling was used to retain the base-rates of different classes after the splitting.

Variable	**Preprocessing**	**Justification**
var3	Values less than zero imputed with median	Only few values are less than zero, and they all are -999999. Likely an error in data, and median is a good guess.
Variables with zero variance	Removed	Do not contain information that helps discriminate between the cases.
(Duplicate columns. None left after previous steps.)	Removed	Contain only redundant information.

Table 1. Data preprocessing.

Implementing and refining random forest

As a first-pass solution almost-default random forest from Scikit-learn Python package was implemented (Scikit-learn 2016). Default parameters were used, apart from setting number of estimators to 100 (more typical than default 10), and pre-specifying the initial random state for repeatability. AUROC score on the test set of the first-pass solution was not too impressive at 0.759. Figure 6 shows the ROC curve and cumulative feature importances for this solution. Plotting feature importances (figure 7) showed that only a few features had large contributions to prediction performance.

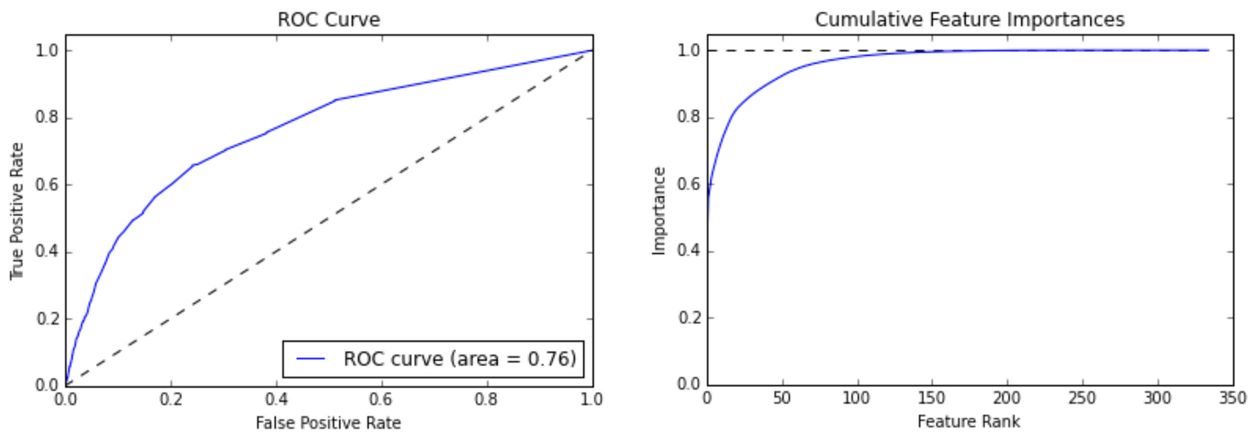


Figure 6. ROC curve and cumulative feature importances for a (almost) default random forest classifier. Practically all information is contained in about 150 most important features.

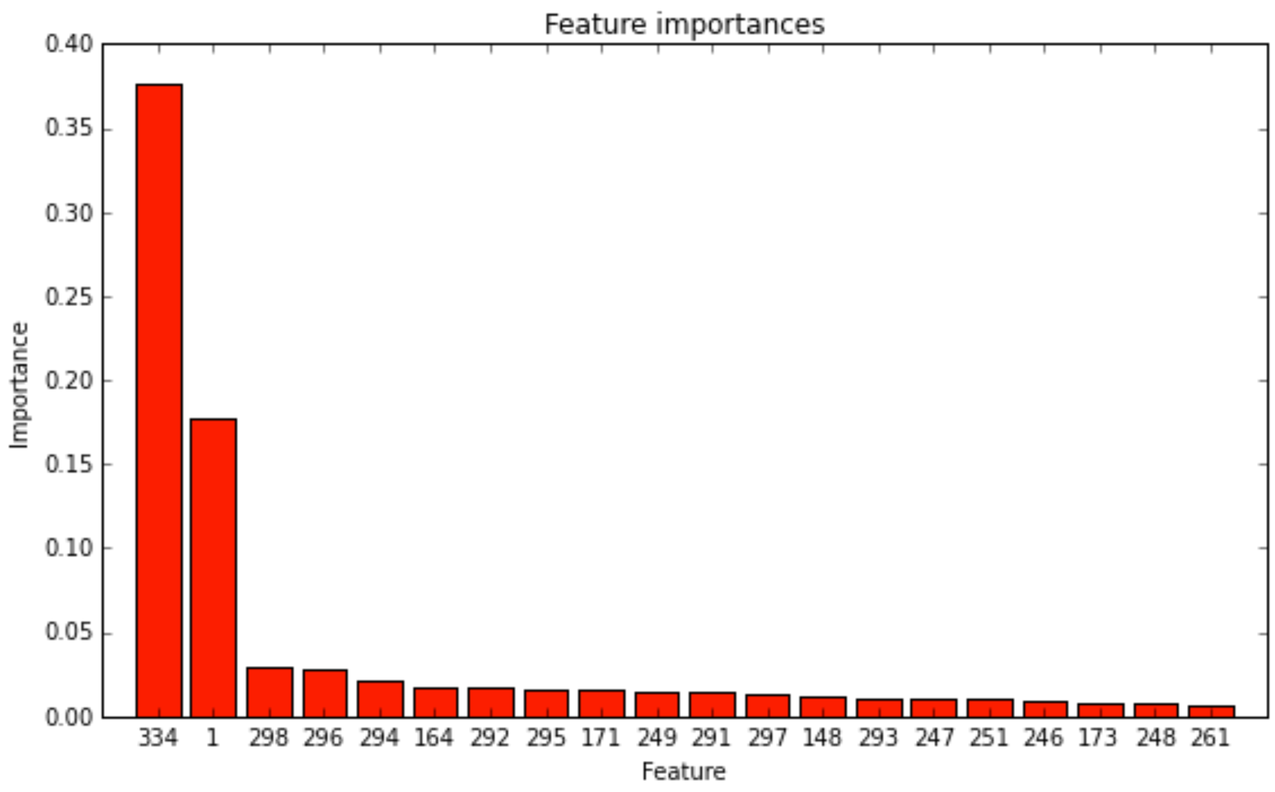


Figure 7. Random forest feature importances. Large part of the information is carried by the two most important features. Variable 334 is called saldo_medio_var44_ult3 and variable 1 is var3 in Santander dataset.

As a second pass solution only 150 most important features (see attached iPython notebook for a list of selected features). As figure 6 shows these features carry most of the information contained in the dataset. Additionally, the random forest algorithm was set to use balanced class weights. The balanced mode adjusts class weights automatically inversely proportional to class frequencies, which is helpful when the class frequencies are very different, as is the case here. Other parameters were kept as before. There was no improvement on AUROC score on the test set, as figure 8 demonstrates.

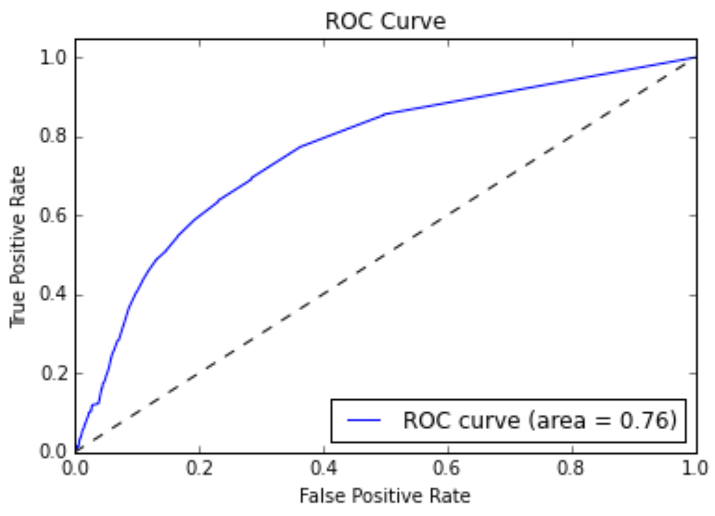


Figure 8. ROC curve for a random forest using 150 most useful features.

Next dimension reduction with principal component analysis (PCA) was implemented. Plotting the cumulative rate of unexplained variance for the most useful components showed that already the first 15 components contain in practice all variance in the dataset (figure 9). However, when training a random forest on transformed data and plotting the feature importances, it turned out that about 220 components were useful for the algorithm. The data was thus transformed using PCA to have 220 dimension, and a new random forest was trained.

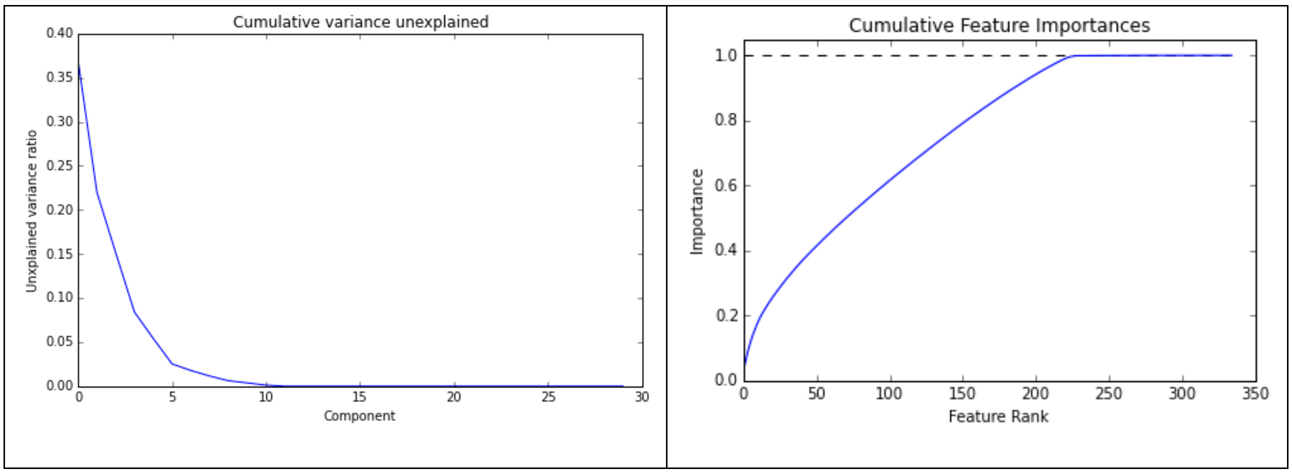


Figure 9. Cumulative variance unexplained by PCA components and cumulative feature importances of random forest after PCA. After 15 components practically all variance in the dataset has been accounted for. However, the random forest can use information in approximately 220 components.

After implementing dimension reduction with PCA the performance on the test set decreased. AUROC score on the test set was now 0.752, but the feature importance plot (figure 11) looked better: importance is more equally distributed. It appears that PCA might do a better job at extracting information from the dataset than merely selecting the most useful raw features.

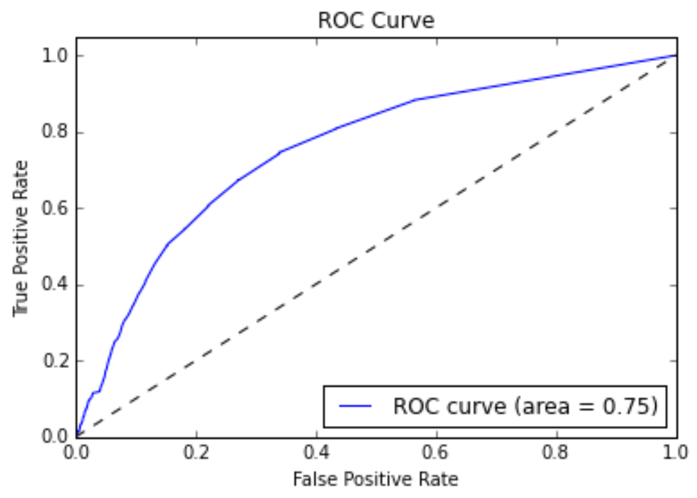


Figure 10. ROC curve for random forest after dimension reduction with PCA with 220 components.

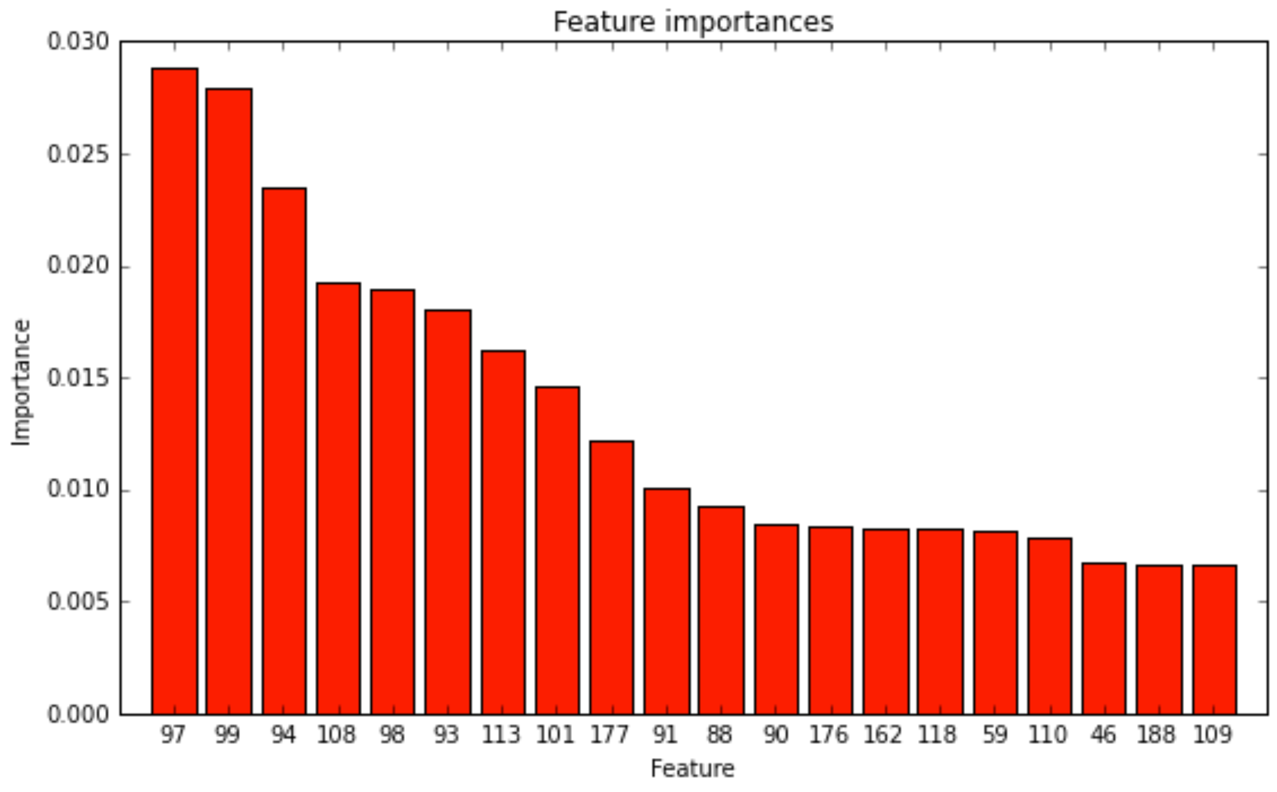


Figure 11. Feature importances for random forest after dimension reduction with PCA (220 components).

I tried to improve the algorithm's performance by parameter optimization. Grid search was used to find best values for number of features to consider at each split of trees, maximum depth of trees, and number of trees to train. To shorten the training time, first maximum number of features and maximum depth of trees is optimized, and then a suitable number of trees was searched for using the optimized parameters from the previous step. These optimization steps did not yield significant improvements. Slight variation in performance is likely due to random effects. Another approach was needed.

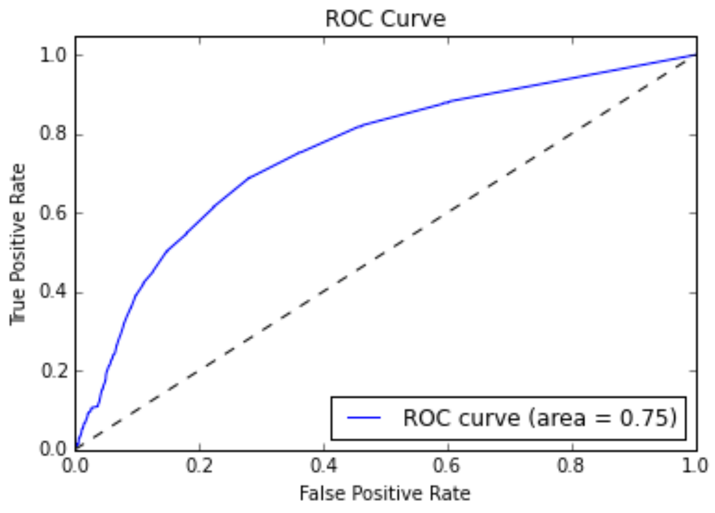


Figure 12. ROC curve for random forest with optimized parameters. Parameter optimization did not improve performance markedly.

Implementing and refining Gradient boosting classifier

Gradient boosting classifier is similar to random forest in that it also uses a set of weak learners (decision trees) to make predictions. The main difference is that each new learner added to the set gives higher weight to cases misclassified by previous learners. The algorithm focuses on challenging cases, which tends to improve performance. Here gradient boosting classifier from Scikit-learn Python package (Scikit-learn 2016) was used. First three gradient boosting classifiers with default settings were trained on unprocessed data, 150 features selected with random forest, and data transformed with 220 component PCA. In each case the AUC score on test set was much better than with random forest at the range of 0.84-0.85. Performance on the full dataset was the best, but as the difference to subset of 150 features was minimal, the smaller number of features was used in tuning the parameters of the algorithm. The smaller feature set should make the parameter tuning run faster.

After choosing which data to use, a thorough optimization of gradient boosting classifier was performed following the instructions from Jain (2016). As a first approximation suitable parameters were guessed based on commonly used values:

- Min samples split: 500
- Min samples leaf: 50
- Max depth: 8
- Max features: square root of number of features
- Subsample rate: 0.8

Parameter tuning progressed as follows. A suitable combination of learning rate and number of estimators for parameter tuning was searched for first. The goal was to find an optimal number of estimators in the range of 30 to 70 for a given learning rate. This way the running time will not become excessively long during the optimization of other parameters. Once good values for other parameters have been found the number of estimators can be increased and learning rate decreased, which usually improves performance. Suitable learning rate for parameter tuning was found to be 0.09, which results in optimal number of estimators being 60.

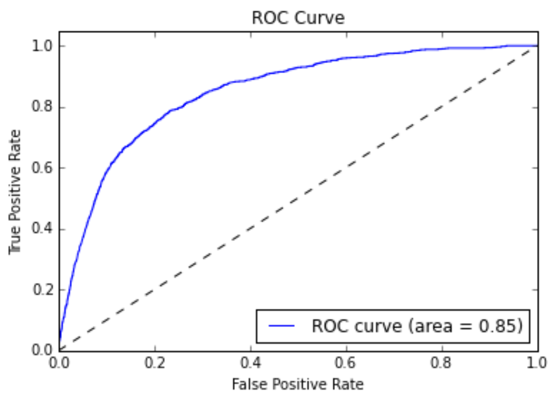
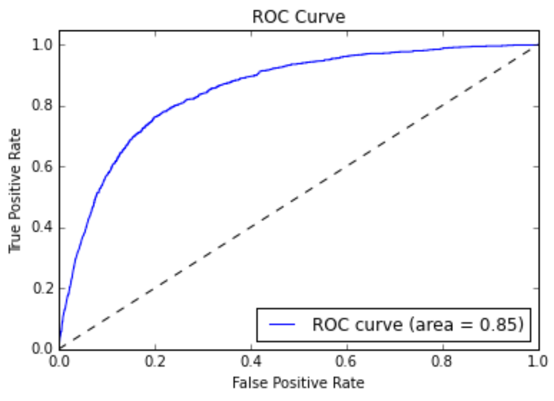
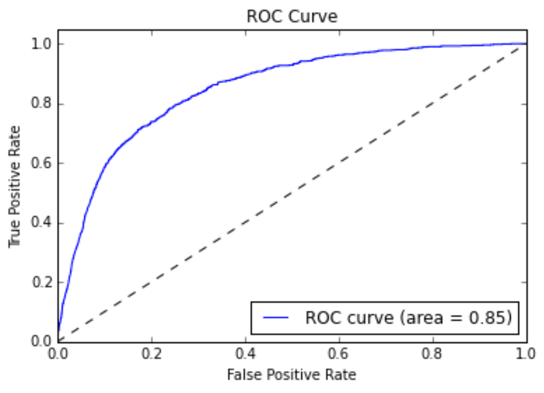
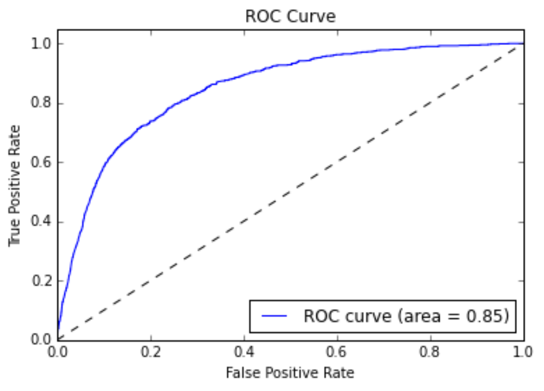
Next good combination of min samples split and max depth was searched for. Perhaps reflecting the rarity of unsatisfied customers, optimal trees are rather deep at 10 levels, and minimum number of samples to split is 2. Maximum depth of 10 was taken as optimal, while min samples split was reconsidered in combination of min samples leaf. This tuning round kept min samples split at 2 and min samples leaf at 50. During the following tuning rounds best value for maximum number of features to consider at each split was found to be 12 and for subsample rate 0.8. As table 2 in results section shows, these optimization rounds did not improve markedly the AUC score on the test set compared to default gradient boosting classifier with default settings. As a final step of parameter tuning number of estimators was increased while learning rate was decreased in same proportion. This increased the AUC score on the test set until 600 estimators at learning rate 0.01. With 1200 estimators at learning rate 0.005 AUC score on training set still improved, but the score on test set started decreasing. The model was likely starting to overfit. Final tuned parameters for gradient boosting classifier were chosen to be:

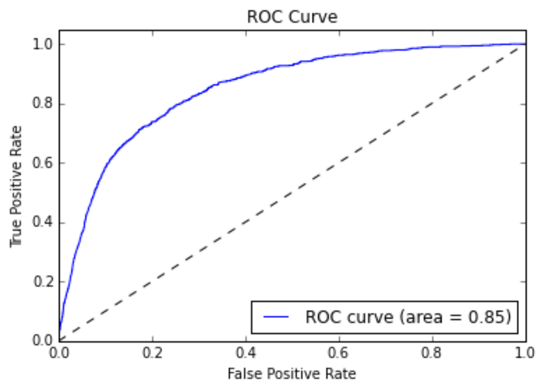
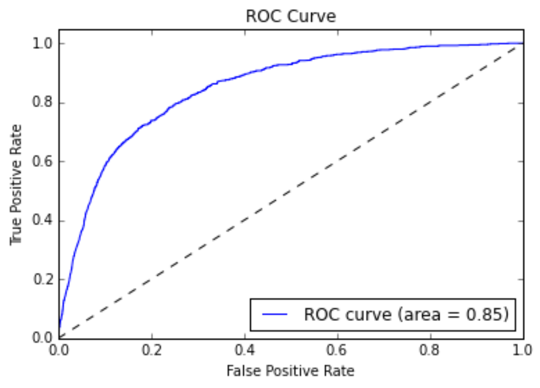
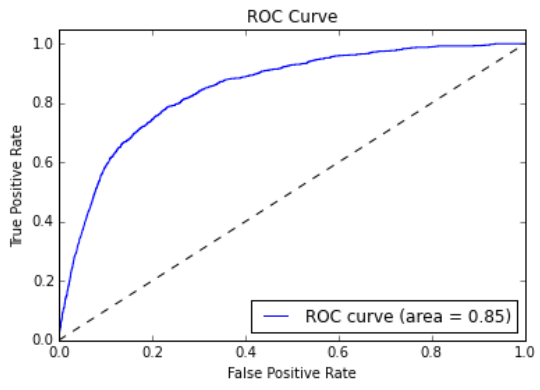
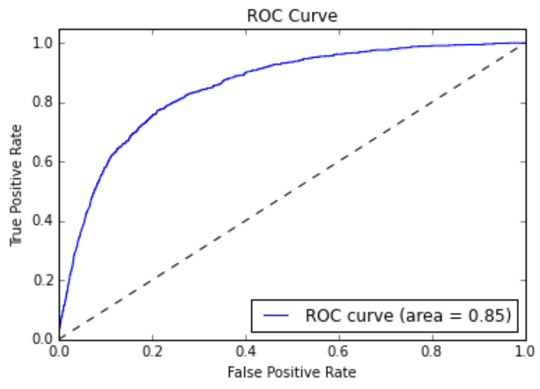
- Learning rate: 0.01
- Number of estimators: 600
- Min samples split: 2
- Min samples leaf: 50
- Max depth: 10
- Max features: 12
- Subsample rate: 0.8

Results and conclusions

Model evaluation and validation

Table 2 shows the results of the optimization steps with corresponding AUC scores and ROC curves. A final model was then trained and evaluated. A gradient boosting classifier with above parameters was trained on all training data using 150 most useful features. Five-fold cross-validation shows that the final model achieves on average AUC score of 0.84 (95 % confidence interval +- 0.02) on unseen data, and the performance does not change much with different data sets. This gives us confidence that the gradient boosting classifier should have decent performance also on the separate test data provided by Kaggle. As a final step predictions were made with the final model on Kaggle test set and predictions were submitted to the competition (although it had already officially ended). Submission scored 0.825358 on Kaggle scoreboard (6. Jun 2016), ranking around 1802/5123. Highest ranked submission scored 0.829072. Although far from the top in ranking, the difference in AUC scores is small at only 0.00314 in absolute terms, or 0.4 %. The solution implemented here should therefore be useful for its intended purpose.

Tuned parameters	**Training set AUC score**	**Test set AUC score**	**Test set ROC-curve**
Baseline model with default settings.	0.8556	0.8480	
Learning rate: 0.09 Estimators: 60	0.8719	0.8503	
Max depth: 10 Min samples split: 2	0.8955	0.8476	
Min samples leaf: 50 Min samples split: 2	0.8955	0.8476	

Max features: 12	0.8955	0.8476	
Subsample rate: 0.8	0.8955	0.8476	
Learning rate: 0.05 Estimators: 120	0.8997	0.8477	
Learning rate: 0.01 Estimators: 600	0.9009	0.8508	

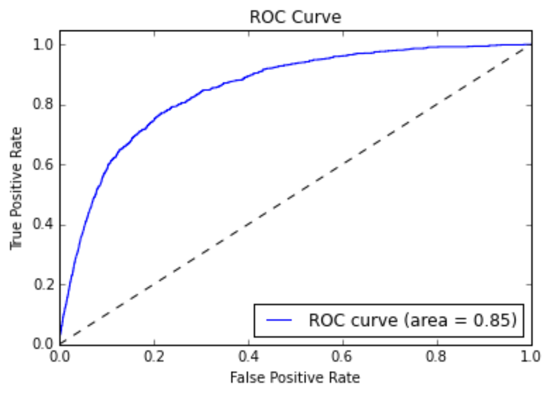
Learning rate: 0.005 Estimators: 1200	0.9014	0.8502	

Table 2. Results of parameter tuning of gradient boosting classifier.

Reflection

This report has dealt with the problem of Santander consumer bank trying to identify unsatisfied customers based on three hundred some variables they have collected. A solution to this problem could be a machine learning algorithm able to classify customers to two categories significantly better than random guess baseline. First the data provided by Santander was explored visually using histograms. Particularly useful features were not identified, but some issues in data could be found. These problems were fixed by imputing a few extreme outliers that were likely errors, and by removing clearly useless features.

After the data set was clean, it was split to training and test sets. A basic random forest classifier was fit on training data and feature importances were assessed. It turned out the algorithm was able to use information from about 150 features. Another random forest was fit on this subset of data. Dimension reduction with principal component analysis was also tried. 220 as a number of components was found suitable, but even parameter tuning did not markedly improve the performance of random forest classifier over the basic version of the algorithm used on raw data.

When the options for improving random forest were exhausted, I turned to gradient boosting classifier, which has a reputation of good performance on Kaggle competitions. First I trained gradient boosting classifiers on the three versions of training data: full set of features, 150 most useful features selected by random forest, and data transformed with 220 component PCA. Classifier using 150 feature-strong subset of data almost equaled the performance on full data set, while PCA-transformed data worked slightly worse. Subset of features was chosen to reduce the training time during parameter tuning. A thorough parameter tuning resulted in improvement on basic gradient boosting classifier. I found the systematic approach to parameter tuning gradient boosting classifier the most interesting aspect of this project. This was new to me, and despite being a lot of work and resulting in long running times, I was able to improve classifier's performance eventually.

The biggest challenge for me in this project was the large number of features. It made it difficult to explore or select features manually. Random Forest turned out to be a good algorithm also for exploration, as it can produce feature importances and thus identify promising variables. Another challenge was parameter tuning of gradient boosting classifier. Heavy use of grid search for suitable parameter values was time consuming due to relatively long running times and numerous interacting parameters. Fortunately, Jain (2016) provided practical and easy to follow instructions for parameter tuning. I followed the instructions successfully and was able to improve the performance of gradient boosting classifier. The final model beats the random guess baseline by a wide margin: AUC score 0.84 +-0.02 compared to 0.5. Furthermore, the model is not significantly worse than best solutions on Kaggle public leaderboard, losing only by 0.4 % (0.825358 vs. 0.829072).

Improvement

Possible improvements on the final model include feature engineering, model stacking, and deep learning. Feature engineering refers to transformation of data to representation better suited for machine learning algorithm (Brownlee 2014). It can include construction of new features from raw data, feature selection, decomposing categorical features, and reframing data. An example of feature engineering is extraction of week day from a date. In this case feature engineering is challenging due to lack of information on what the given features represent: human knowledge cannot be used to improve on the representations.

Stacking refers to different ways of ensembling various learning methods. MLWave (2015) describes many ways to do this. Examples include averaging the predictions of several different machine learning algorithms, taking a majority vote, and using predictions from one model as features for another. As a proof of the power of stacking, the classic Netflix data science competition was won with a very heavily stacked mode. In this case predicted probabilities from random forest could have been fed to gradient boosting classifier as one of the features, giving the classifier a new view to the data.

Deep learning can be considered as a kind of stacking combined with automatic feature creation. Deep learning algorithms are neural networks with multiple layers, where each layer uses the results of the previous layer to create more abstract representations of the information. They are inspired by the way how brain works and have achieved impressive results in recent years from Google Image Search to speech recognition. For instance, TensorFlow, an open source software library from Google, could be used to implement a modern deep learning system to make predictions on Santander data.

References

Brownlee J. (2014) Discover feature engineering, how to engineer features and how to get good at it. http://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/

Jain A. (2016) Complete guide to parameter tuning in Gradient Boosting (GBM) in Python. http://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/

Kaggle (2016) Santander Customer Satisfaction. https://www.kaggle.com/c/santander-customer-satisfaction

Scikit-learn (2016) Scikit-learn: Machine learning in Python. http://scikit-learn.org/stable/

Wikipedia (2016) Receiver operating characteristic. https://en.wikipedia.org/wiki/Receiver_operating_characteristic