Marketing: Predicting Customer Churn

rushingdragging
https://github.com/rushingdragging

HWANG7308
https://github.com/HWANG7308

philipppp000
https://github.com/philipppp000

S1887468
https://github.com/s1887468

Abstract

This project, derived from the KDD Cup 2009 competition, focuses on predicting an element of customer behaviour, customer churn, with preprocessing and classification on the provided anonymised data set. We adopted exploratory data analysis and explored the feature of the data set, and introduced several simple classifiers and ensemble methods, including Naive Bayes, Logistic Regression, Decision Tree, Bagging, Boosting and Voting classifiers. This project compared their performance and discussed the potential reasons for the achieved outcome.

1 Introduction

1.1 Motivation

The data set that forms the basis for this assignment was originally prepared for the KDD Cup Challenge 2009. The data was provided by the French Telecomms company Orange and the task represented a real-life business challenge: creating an effective *Customer Relationship Management* (CRM) model.

CRM is a strategic approach to managing company's interaction with current and future customers with the purpose of driving business growth through improved customer relationship. The concept of CRM emerged at the end of the 20th century and since then multiple models have been adopted by the business community. The key aspect of this approach is the development of a customer-centric business culture, which necessitates an in-depth understanding of customer profiles, needs and behaviours [1].

The complexity of the input space makes modelling such understanding a challenging multi-dimensional task. The objective of the 2009 KDD Cup competition was to predict three specific aspects of customer behaviour: the propensity of customers to switch the telecomms provider due to the loss of interest in the current service (customer churn); the tendency to sign up for a new product or service (appetency); and the inclination to upgrade or purchase add-ons (upselling) [2].

1.2 Background and Previous Work

The KDD Challenge is an annual competition that focuses on data mining/machine learning tasks. In 2009, an offer of a lucrative prize attracted a total of 450 competition entrants, all aiming to achieve better prediction results on the three target variables (*customer churn*, *appetency* and *upselling*) compared to the classification algorithm developed by Orange in-house. [2] offered a comprehensive summary of the top successful approaches; and the Cup organisers made the reports authored by the individual teams available for public use.

The analysis of the literature mentioned above highlighted the key themes about the task (heterogeneous sparse data, large number of training examples, unbalanced class distribution etc) as well as

the best practice in achieving a favourable task outcome (e.g. utilising ensemble learning methods). This information inspired the authors of this report to adapt the techniques described as best practice to our project.

1.3 Project Scoping

The KDD challenge organisers provided two distinct data sets: a large data set consisting of 15,000 features and a downsized version of the same data set consisting of 230 features. Both data sets offered a large volume of training examples - 50,000, and a separate test set.

While the teams on the leader board of the KDD challenge focused their modelling efforts primarily on the large data set, this assignment will concentrate on the exploration of the smaller data set version with the purpose of predicting a single aspect of customer behaviour, the choice of which will be made following the data exploration.

The goal of the assignment is to apply best practice techniques to pre-process the data set in such a way that it could be consumed by the selected training model and produce a reasonable level of performance for predicting the target variable. While the results achieved by the KDD Cup performers will serve as a useful benchmark for this activity, there is no expectation that this team will match the level of performance of the leader board entries.

2 Data Preparation and Exploration

2.1 Class distribution

The task in hand represented three distinct binary classification problems. Since the team's focus was on solving a single problem, the *exploratory data analysis* (EDA) began with establishing the class distribution for each of the targets: appetency, churn and upselling. A histogram of class distribution (Figure 1) confirmed a significant bias for negative class in all three target variables.

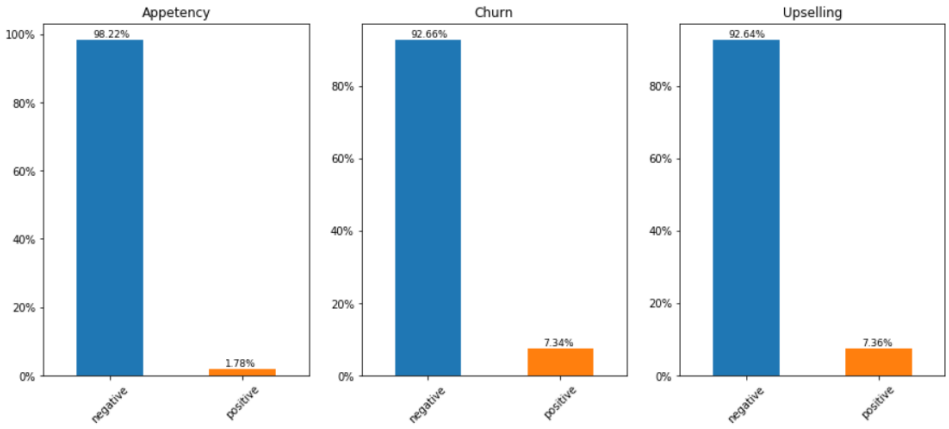


Figure 1: Class distribution in target metrics

Based on this analysis the team chose customer churn as the aspect of customer behaviour to be modelled. The analysis also highlighted that extra care needed to be taken when choosing the prediction model(s) and the evaluation mechanism, as both would need to be suitable for solving a binary classification problem with a significant class imbalance.

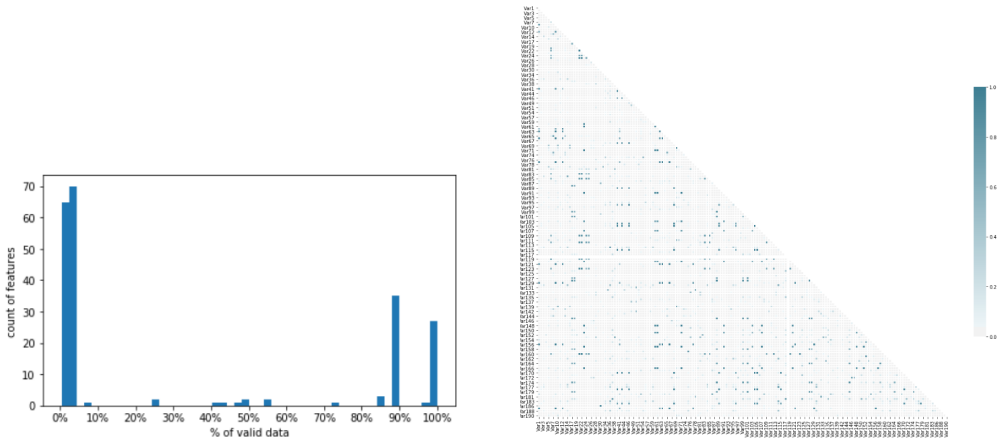
2.2 Feature exploration and cleaning

Feature exploration formed the second step of the EDA. The initial examination of the training data set established that it consisted of 230 features and 50,000 records. The raw data had been fully anonymised by the organisers since it contained information about private customers. All variable names followed a naming convention of 'Var*n*' (where *n* was the index of the column) and all

categorical values had been converted into random strings. This level of data anonymising, although necessary from a privacy perspective, limited the team's understanding of the domain and the ability to apply judgment to identify outliers, among other intuitive decisions. The preliminary exploration had also highlighted a high volume of missing data: almost 70% of all data were missing. 18 features were removed due to containing no data at all; the remaining feature set was checked for duplicate features but none were present. Following the first cleaning step, the data set was reduced to 212 features (173 with continuous numeric values and 39 categorical features).

The team intended to backfill the missing numeric values with feature means, however a decision had to be taken on whether to remove a subset of the features with low value coverage or, alternatively, expand the feature set to preserve the information about the missing data by creating new features indicating the presence/absence of data in the original features. To establish the suitable way forward the team visualised the distribution of features based on the percentage of valid data, presented in Figure 2(a). This analysis highlighted that 66 features (circa 31% of the cleaned feature set) had value coverage in excess of 80%, whereas 136 features (circa 64% of the feature set) had value coverage below 10%. Removing sparsely populated features would have resulted in a loss of nearly two thirds of the feature set, which the team considered too risky from an information loss perspective.

Following this assessment the data set was segregated by data type. Missing values for real-valued numeric features were imputed with feature means, and the feature set was expanded by binary encoded features indicating imputed vs originally present values. Subsequently, a correlation assessment of the numeric features was made, as shown in Figure 2(b), and features with correlation in excess of 80% were removed, reducing the numeric feature set by 48 features. The corresponding encoded features were also removed.



(a) Feature distribution by percentage of valid data (b) Feature correlation heatmap

Figure 2: Feature visualisations

Further exploration was then carried out for the categorical features. First, the top ten counts for each category for each feature was printed for every categorical feature, which was then visualised using a bar plot that counted for each value of the feature the number of entries in each class. Due to the extreme class imbalance, it was difficult to determine whether certain features would be a good way to distinguish between the classes, as the underlying distribution of feature categories usually consisted of one category value containing the majority of the entries for that feature. Figure 3 shows the distribution for a small selection of cateogircal variables. This analysis did reveal that some of the features contained only one category, with a similar distribution of the classes as in the data set as a whole (i.e. class distribution remained at around 10% positive); since these features would not contain any information to distinguish the classes, they were removed (5 features in total). Since we planned to use one-hot encoding in order to fit linear classifiers to the model, a number of preprocesing steps were required for the cateogircal variables. In many of the categorical variables, we discovered that there were a large number of distinct categories. These would greatly boost the size of the one-hot encoding, so the first pass of cleaning involved removing any variable for which there were upwards of 500 distinct categories. Binning proved to be useful for many of the competition entrants [2], so

we decided to implement it by creating a new category for each categorical variable, 'OTHERS'. In this category, we included every category which had less than 5% coverage of the total number of entries. Another problem with using one-hot encoding was that some categorical variables had a considerable number of missing values; these were replaced with a new category 'MISSING' which can easily be transformed using one-hot encoding. The effects of these cleaning steps on an example variable can be seen in Figure 4.

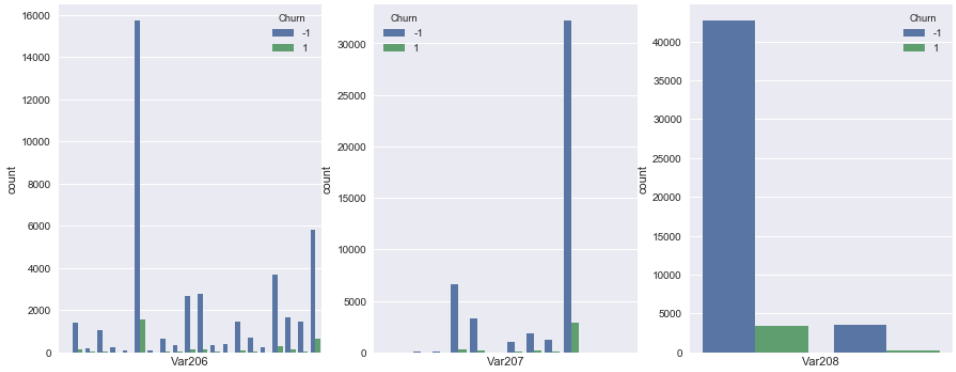


Figure 3: Class Distribution of three categorical features (Var206, Var207 and Var208)

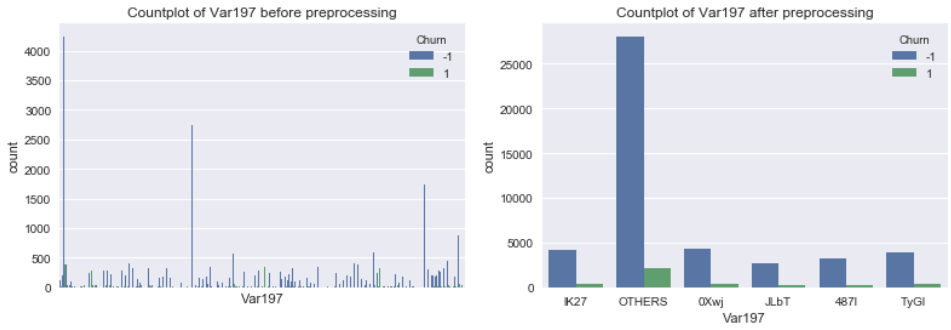


Figure 4: Countplots showing changes to Var197 after preprocessing on categorical variables

After above data pre-processing steps were completed, the data sets consisted of 125 numerical and 17/67 categorical features (before/after one-hot encoding was performed). There were additionally 125 features representing the binary missing value indicators for each numerical column. Since we were limited in terms of both resources available and training time for our models, we decided to use a traditional dimensionality reduction technique (standard PCA with *sklearn.decomposition.PCA*) as the last pre-processing step for categorical variables. This was also motivated by our concern that a high number of variables may lead to overfitting because of the sparsity of the data, particularly for logistic regression and decision tree classifiers (although for decision trees this can be counteracted using the max-depth hyper-parameter). Although many teams taking part in the challenge found little improvement using PCA [2], since we undertook our own preprocessing and encoding of the categorical variables that condenses the information, we decided to apply it to the categorical variables only. The PCA resulted in the top 32 components containing 90% of the explained variance, which we took as our cut-off point; as a result, the PCA reduced the number of categorical (encoded) features from 67 to 32.

For a train/validation/test split, since we only had access to labels for the training set, we set aside a test set consisting of 10,000 entries (20%) at this stage to compare the models on. This test set was generated with $sklearn.model\_selection.train\_test\_split$, with random state parameter set for repeatability.

3 Learning methods

In line with the summary of the methods used by successful competitors in the KDD Cup competition [2], the team decided to experiment with several different classifiers. The three types of classifier we implemented were Gaussian Naive Bayes (NB), Logistic Regression (LR) and Decision Tree (DT). Given the sparse nature of the data and the imbalanced class distribution, we chose to use these classifiers with established ensemble methods. Ensemble methods are algorithms which combine multiple machine learning techniques into a single predictive model to minimise variance (bagging), bias (boosting), or improve predictions (stacking and voting) [3].

Three ensemble methods were used with the classifiers: Bagging, Boosting and (soft) Voting Classifier (Voting). When Bagging and Boosting are used with Decision Trees or Logistic Regression, the model is trained iteratively. In the final classification all of the results provided by each iteration created using the ensemble method are weighted to obtain a more precise result.

Bagging [4] involves picking samples at random from the data set to be used in a training set, and was implemented with both the Decision Tree and Logistic Regression classifiers. More specifically, given a training data set D with n entries, Bagging generates m new data sets of size n by sampling from D uniformly at random with replacement. Each of the m created data sets are then used to train the models. When predicting classification, we obtain m predictions, which are then weighted.

Alternatively, Boosting [5] iterates the regression in the simple classifiers, with different weights being given to samples for each iteration, with the weighting being based on the results of the previous trained iteration - wrong predictions are given heavier weights. Boosting was applied to the Decision Tree classifier. In more detail, a data set D of n entries is used to first train a base model, which is used to predict on the data set D - any incorrect predictions are isolated. These incorrectly predicted entries are used together with D to train a second model, with weights given to the additional data in each step. This process continues iteratively to train m models, with weights given to predictions based on the accuracy produced on the training data set.

A key difference between these two ensemble methods is that boosting uses all training data, whereas bagging may miss some training entries due to the sampling method used. Also, since boosting has no effect on linear classifiers (the iterative step would retrain the same weights for a linear model each step), it is not appropriate to use with a Logistic Regression classifier.

In addition, a soft voting method was applied using three types of classifier for comparison. If we train three models (Decision Tree, Logistic Regression and Random Forest [6]) on the training data set, we can use the prediction from each model and take the final prediction to be the most common classification. A soft Voting classifier additionally applies a probability weight to each of the models for the final prediction.

To simplify notation in graphs and tables, we used the following names for our classifiers: Decision Tree/DT (no ensemble method applied), Random Forest/RF (DT with bagging), AdaBoost (DT with boosting), Logistic Regression/LR (no ensemble method applied), Bagging (Logistic Regression with bagging) and Voting Classifier (Soft Voting classifier).

In addition, we decided to train (and test) our models on four variants of our pre-processed dataset, each variant including certain combinations of pre-processed features. Since we had encountered and understood the mechanics of PCA from the course, we elected to train/test on the data both with and without applying PCA to see if we could explain any difference (although we weren't necessarily expecting any difference due to the results of the competition). We also trained/tested on the dataset with and without using the added missing value features (annotated as missing matrix) - this was motivated by our ability to easily separate the dataset this way, so we could see directly if this made any difference to each classifier.

4 Performance Analysis

Due to the significant class imbalance present in the data, we have reported not only the accuracy but the precision, true positive (TP) and false positive (FP) rates for each classifier across the four dataset variants in Table 1 and Table 2. As shown in Section 4, we provided a visualisation of the ROC curve for each model and labelled the area under each curve, as this is not only a widely used

statistic for model comparison, but was also used as the method for scoring the models in the original competition [2].

Approach	No PCA				PCA			
	ACC(%)	TPR(%)	FPR(%)	P(%)	ACC(%)	TPR(%)	FPR(%)	P(%)
NB	16.55	93.10	89.82	7.94	16.43	93.36	89.97	7.95
LR	92.32	0.00	0.01	0.00	92.31	0.00	0.01	0.00
DT	85.20	11.20	8.64	9.73	86.20	14.58	7.84	13.40
RF	92.32	0.00	0.00	-	92.32	0.00	0.00	-
Voting	92.31	0.00	0.01	0.00	92.31	0.00	0.01	0.00
AdaBoost	92.06	1.82	0.43	25.93	91.87	1.95	0.65	20.00
Bagging	92.31	0.00	0.01	0.00	92.31	0.00	0.01	0.00

Table 1: Result without missing matrix, accuracy (ACC), TPR, FPR, precision (P)

Approach	No PCA				PCA			
	ACC(%)	TPR(%)	FPR(%)	P(%)	ACC(%)	TPR(%)	FPR(%)	P(%)
NB	17.34	92.84	88.94	7.99	17.22	93.10	89.09	8.00
LR	92.31	0.00	0.01	0.00	92.31	0.00	0.01	0.00
DT	85.27	12.50	8.68	10.70	85.56	14.19	8.50	12.19
RF	92.32	0.00	0.00	-	92.32	0.00	0.00	-
Voting	92.31	0.00	0.01	0.00	92.31	0.00	0.01	0.00
AdaBoost	92.04	2.08	0.48	26.67	91.97	1.82	0.53	22.22
Bagging	92.31	0.00	0.01	0.00	92.32	0.00	0.00	-

Table 2: Result with missing matrix, accuracy (ACC), TPR, FPR, precision (P)

For our data, with its material class imbalance, a model which only ever predicts a negative class would produce a high accuracy of 92.3%. Therefore also reporting the precision, true positive and false positive allows us to see how well our models are identifying the positive class entries using various metrics, since we would want a classifier that classifies as many positive instances correctly, without also generating too many false positives. Therefore we look for a high True Positive rate and precision with as low a False Positive rate as possible.

The investigation into the use of PCA and the missing data indicators showed little change in the results. In general, we found that the use of PCA caused little or no improvement (and in a few cases, they were slightly worse) to each of the classifiers. For example, the Decision Tree classifier had a higher accuracy (85.2% to 86.2%), True Positive rate (11.2% to 14.58%) and precision (9.73% to 13.4%) with a lower False Positive rate (8.64% to 7.84%) (when not using the missing indicators), with similar small improvements also when using PCA with the missing indicators. However, many classifiers, including Logistic Regression, Random Forest, Bagging and Voting Classifier saw no change when using PCA. AdaBoost, interestingly, had a worse performance when using PCA with the missing indicators, although both the TP and FP rate both increased when using PCA without the missing indicators, so in both cases PCA seemed to have a small net negative effect on AdaBoost. Naive Bayes was the only classifier which showed worse performance in both cases when using PCA. Additionally, every classifier showed at most a 0.02 in area under the ROC curve improvement when using PCA. Overall, we conclude that PCA had little effect on the classifiers, with any change dependent on the classifier.

It was a similar outcome when using the missing indicators. The same classifiers, Logistic Regression, Random Forest, Bagging and Voting Classifier saw no change. Only small changes were observed for Naive Bayes. Decision Tree and AdaBoost - in each case, there was a trade-off between better

and worse rates when using the missing indicators, with only Decision Tree showing an improvement in every metric when not using the missing indicators.

The results for each classifier however showed a much more interesting outcome. Naive Bayes was unique in that it consistently showed a very low accuracy rate (around 16-17%) across all datasets - this was due to it classifying too many entries in the positive class. It tended to capture most of the positive entries (TP rate of 93%) but also classified around 90% of the negative class as positive. Four classifiers (Logistic Regression, Random Forest, Voting Classifier and Bagging) showed the complete opposite problem - they all classified (almost) everything as negative, as can be seen in the 0% values for the precision, TP and FP rates. They did report the highest accuracy amongst all classifiers, but this was only due to the class imbalance. The conclusion for these classifiers was that none of them offered an acceptable level of performance for the overall task - for Naive Bayes, too many false positives would be returned, whereas the others were equivalent to a base classifier that classified everything as negative.

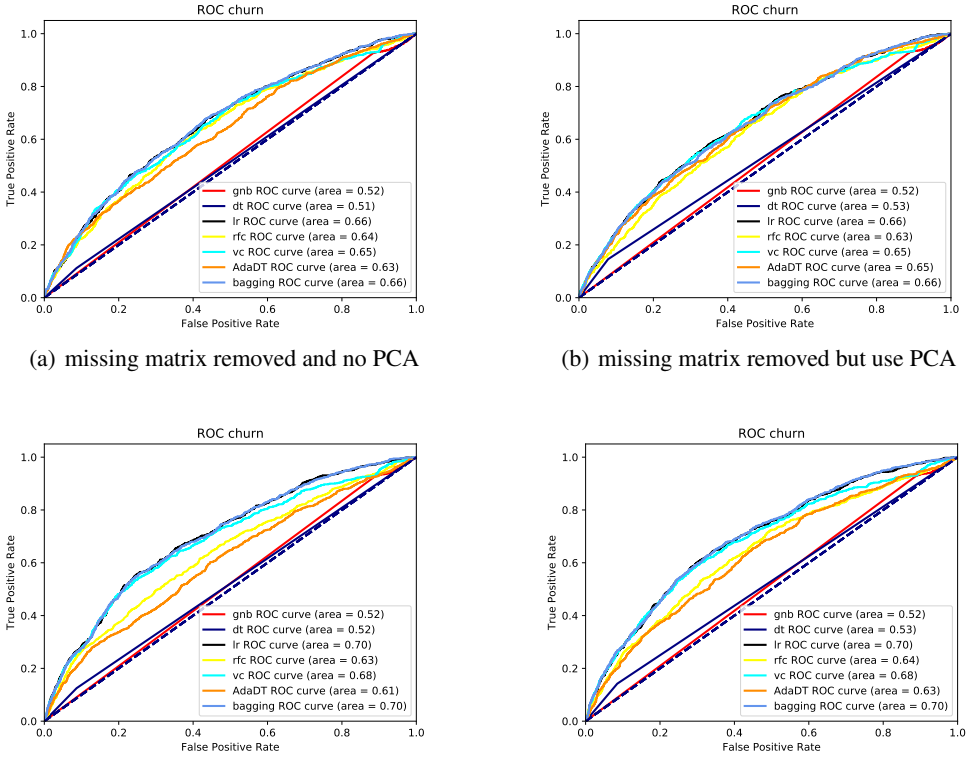
We found that two of the classifiers showed a small amount of promise - Decision Tree and AdaBoost. The Decision Tree classifier generated the highest TP rate of 11-14%, meaning this was the classifier that was finding the most correctly positive entries (outside the extreme result of Naive Bayes). However, this result came with a trade-off: The FP rate of 8-9%. Due to the class imbalance, this meant that the precision is only around 10-13%. For the task, this meant that the Decision Tree identified correctly a decent number of positive entries, but also falsely identified many more. For AdaBoost, the results showed that it was more reluctant to classify instances as positive, since it had a much lower TP rate (around 2%) but also a much lower FP rate (around 0.5%) than the Decision Tree classifier. This could also be seen in the higher precision rate that exceeded 20%. The AdaBoost classifier was therefore more sure when it made a positive decision; however, the TP rate was so much lower that it didn't correctly classify many positive entries at all. Finally, across all four different dataset combinations, the best performing Decision Tree was on the PCA without missing indicators data, whereas AdaBoost performed best on the no-PCA with missing indicators data. We found that, for the case of customer churn, the area under the ROC was a poor measure as to the usefulness of the model - the models exhibiting an expected behaviour from a business perspective had consistently scored a lower area under the curve than the classifiers predicting only the negative results.

Some of the results could be explained using the underlying mechanics of each of the models. Naive Bayes, for example, could show a strong bias towards positive class prediction because there was a lack of conditional independence between features (perhaps evidenced by the high correlation between numerical variables). The behaviour of Logistic Regression and its Bagging counterpart could be explained by the major class imbalance and the setting of the decision boundary using the logistic function - we set the decision boundary at 0.5, although a lower value may have been more appropriate in this case. Because of the high bias and low variance present in a normal Random Forest classifier, combined with our pre-processing steps, the Random Forest classified everything as negative. The AdaBoostClassifier put emphasis on the positive class labels during its iteration process, so it had a higher true positive rate than the Random Forest model, but it sacrificed for lower accuracy as it produced more false positives. The Voting Classifier, because it contained two all-negative classifiers, acted the same as an all-negative classifier due to the voting mechanism.

5 Conclusion

The results of our experiments show than none of the classifiers were well suited to the given task with the chosen pre-processing methods, in particular Logistic Regression and others ended up classifying everything as negative. We conclude that this outcome could be attributed to two main factors: first, our pre-processing techniques were not as powerful at maximising the information as we would have liked, either because we lost too much information or because the pre-processed data still contained too much noise. Secondly, the small dataset did not contain the combination of features valuable for more accurate predictions (we worked with 230 features vs. 15,000 in the large dataset). Notably, the leading competition entrants came to the same conclusion and discarded the small dataset in favour of the large one.[2]

However, two of the classifiers showed promise, namely Decision Tree and AdaBoost, as both had seemingly extracted some useful information from the data in order to classify some positive churn customers. In our best case scenario, Decision Tree correctly predicted around 15% of the customers



(a) missing matrix removed and no PCA

(b) missing matrix removed but use PCA

(c) missing matrix included but no PCA

(d) missing matrix included and use PCA

Figure 5: ROC curve

who could switch to a different provider, giving the company an opportunity to influence these customers through incentives. The downside was that they would also try to incentivise a large number of customers who had no intention to switch. AdaBoost carried a smaller rate of customers falsely classified as likely to switch, but also only predicted at most 2% of the total customers who were actually likely to switch. Some further cost-benefit analysis would be required to determine which of these two models was the most useful for the problem task, which was outside of our scope. With further time, we would perhaps focused on more powerful pre-processing methods, for example down sampling, as evidenced in [7], in order to reduce the severe class imbalance.

References

[1] Francis Buttle. Customer relationship management: Concepts and technology, 2009.

[2] Isabelle Guyon, Vincent Lemaire, Marc Boullé, Gideon Dror, and David Vogel. Analysis of the kdd cup 2009: Fast scoring on a large orange customer database. In *Proceedings of the 2009 International Conference on KDD-Cup 2009-Volume 7*, pages 1–22. JMLR. org, 2009.

[3] Oleg Okun and Giorgio Valentini (Eds). Supervised and unsupervised ensemble methods and their applications, 2008.

[4] Tom Fearn. Bagging. *NIR news*, 17(8):15–15, 2006.

[5] Peter Bühlmann and Bin Yu. Boosting. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(1):69–74, 2010.

[6] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[7] Pedro Domingos. Metacost: A general method for making classifiers cost-sensitive. In *KDD*, volume 99, pages 155–164, 1999.