Applying Machine Learning to Bank Churn Prediction

Rajbir Bhattacharjee

**Abstract** Regarding *Lorem Ipsum* and according to Wikipedia, in publishing and graphic design, *lorem ipsum* is common placeholder text used to demonstrate the graphic elements of a document or visual presentation such as font, typography, and layout. Although it resembles Latin, it does not have any inherent meaning. Publishers often use *lorem ipsum* when displaying typeface to direct focus to the presentation. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam sapien. Proin euismod metus id elit. Aliquam posuere orci nec lacus. Vivamus consectetuer, turpis non vulputate faucibus, nisl lectus pretium urna, at mollis turpis arcu non quam.

1 Introduction

The dataset considered in this paper is labelled sample of a bank’s customer data. The data is labeled whether the customer left the bank or not. The task is a classification task to predict whether a customer will leave the bank or not.[14]

The dataset has the following fields:

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Notes** |
| RowNumber | Integer | Row Number |
| CustomerId | Integer | Customer Id, no specific pattern noted |
| Surname | String | Surname of the customer |
| CreditScore | Integer | Credit score of the customer |
| Geography | String | Country of the customer, has 3 distinct values, France, Germany and Spain |
| Gender | String | Male or Female |
| Age | Integer | Age of the customer |
| Tenure | Integer | Number of years with the bank |
| Balance | Float | Account balance |
| NumOfProducts | Integer | Number of products customer is subscribed to |
| HasCrCard | Integer | Denotes whether a customer has a credit card or not |
| IsActiveMember | Integer | Denotes whether customer is active or not |
| Estimated Salary | Float | Estimated salary of the customer |
| Exited | Integer | Target class label: 0/1 indicating whether the customer exited the bank or not. |

The data set has 10000 examples with no missing values. However, the data set suffers from moderate class imbalance with about 25% of the samples have the label Exited=1, and the remaining 75% have the label Exited=0.

The motivation to build a machine learning model is to be able to predict which customers will exit the bank after training on the data. A further objective of the study is to compare how different models perform, along with techniques to remedy data imbalance. We will show that certain algorithms are more sensitive to data imbalance than others. We will also discuss the performance of several methods to mitigate data imbalance on this this dataset. Furthermore we attempt to find a combination of an algorithm and other techniques to mitigate data imbalance and try to maximize the F1-score for classification.

2 Research

In a classification problem, the ideal situation is where each target label has equal probability of occurrence in the real world, as well as the sample data-set. However, that is seldom the case, and in most real-world problems, the class labels are highly skewed. Consider the case of predicting cancer – only a small percentage of the population have cancer, and the vast majority of the population do not have cancer. Further imbalance maybe introduced by the sampling methodology.

If we are predicting two classes, and the chances of a sample being a +ve is 1%, and the chance of a sample being a -ve is 99%, a machine learning model can achieve an accuracy of 99% by simply predicting -ve all the time. However, such a prediction is of little use. Hence accuracy is not a good judge of the quality of prediction. Better measures of judging accuracy are measures like precision, recall and F-1 score. The F-1 score is particularly interesting because it uses just one number to indicate both false positives and false negatives. Another metric that is frequently used is the confusion matrix.[[1]](#footnote-1) Precision and recall values and ROC curve can also be used.

The goal of any model is to be able to accurately predict both +ve and -ve classes. However, since in real world phenomena, -ve examples vastly outnumber +ve examples, the model has very little opportunity and incentive to learn how to predict the minority class correctly.[[2]](#footnote-2) This generally results in a very good performance for predicting the majority class, but a very bad performance for the minority class. For such types of problems, the challenges introduced by data imbalance must be remedied.

Certain models are less sensitive to class imbalance, and that can be a method to mitigate class imbalance. The k-nearest neighbors class of algorithms are resilient to class imbalance. Boosting is another method which offers a degree of resilience to class imbalance though it may still benefit from other mitigation techniques. This is because at each stage of boosting, the weights of individual instances are readjusted sampling done with those weights. The weights are readjusted such that the misclassified instances have a higher probability of getting sampled. As the minority class is more likely to be miscategorized, the effective result is some level of protection against the problems of class imbalance. SVMs are also somewhat resilient against class imbalance problems compared to other methods, though they are still affected by class imbalance.[11] However, most of these models are not completely immune to imbalance problems and may still benefit from other techniques to mitigate imbalance.

One way to deal with data imbalance the use of alternate cut-offs in classifiers that output each prediction with a probability. This probability can be adjusted so that the minority class is favored. The use of alternate cut-offs does not result in any change in the model as such.[1] Alternate cut-offs are only applied after the model has done with its prediction of probabilities.

Another method of dealing with imbalance is to adjust the prior probabilities of the classes so that it results in an increase in the probability of the minority class being predicted. Such a method can help improve the performance of a Bayesian learner. Weiss and Provost (2001)[2] suggest that using prior probabilities that describe the true occurrence of a class in the real-world biases the model towards predicting the majority class, and using a more balanced prior probability is likely to improve the performance of the model.

Yet another way to deal with imbalance is to use unequal case weights. Ting (2002)[3] describes one method in which they adjust the weight of a sample to reflect the cost incurred in misclassifying it, with the minority samples having a greater weight than the majority class.

A similar effect also happens in boosting, where the probability distribution over all the examples is changed at each iteration, which affects the sampling, and results in more samples from the misclassified instances being drawn. Since the minority class is likely to be misclassified more initially, it is likely that later samples will contain more instances of the minority class, and the overall classifier will learn to classify the minority class correctly.[4] Because of this, boosting techniques offer some level of protection against class imbalance, although they also benefit from other techniques to mitigate imbalance.

Cost-sensitive training has been the subject of much inquiry. In such methods, the cost function modified such misclassification of an instance belonging to the minority class incurs a much heavier penalty than misclassification of the majority class. The modified cost function results in a change in the model so that the model learns to classify the minority class better. This contrasts with the use of alternate cut-offs where there was no change in the model as such. Another variation of alternate cut-offs is one which not only assigns different costs to misclassification of the majority or minority class, but also allows for different costs for certain types of errors.

Many classification tree models including CART can be modified to incorporate different costs for different classes. The cost can take into account the cost of the mistake, the probability of making a mistake, and the prior probability of the class for a sophisticated.[5]

Breiman, et.al. suggest using generalized Gini coefficient to mitigate class imbalance (instead of using the standard entropy measure).[6] The use of the Gini coefficient results in the cost of misclassification to be scaled depending on the probability of the misclassification. They suggest that this is sometimes equivalent to adjusting the prior probabilities.

A more direct way in dealing with class imbalance is to either artificially increase the number of instances in the minority class, or artificially decrease the number of instances in the majority class. This has given rise to a family of techniques called sampling. There are two types of sampling: oversampling and under-sampling; the former works by increasing the number of instances in the minority class and the latter works by decreasing the number of instances in the majority class. The goal of sampling is to achieve balance between majority and minority classes.

The simplest of all sampling techniques is random under-sampling in which balance is achieved by under-sampling the minority class. However, this has the drawback that it loses data instances, which thereby can affect the quality of the prediction. However, a boosting technique may be applied in conjunction with it to make a better classifier. RUSBoost[16] takes this approach of combining random under-sampling with a boosted classifier. The traditional boosting algorithm is modified in RUSBoost to first apply random under-sampling to create a temporary training dataset where N% of the instances are of the minority class. This is applied at each iteration. The result is a greater number of samples from the minority class being chosen. The use of a boosting algorithm also means that none of the instances of the majority class are thrown out, because they will be chosen in other iterations. The Tomek-Links[7] method removes data-items in the boundary between the majority and minority class, and works by improving class separation.

By contrast, over-sampling methods work by increasing the number of minority instances while leaving the majority instances untouched. The simplest of all methods is random over-sampling where random instances of the minority class are duplicated. Chawla, et.al. proposed SMOTE[8] which has been very successful. SMOTE works by synthetically creating new samples rather than just duplicating new samples. In SMOTE, an instance is chosen at random and its k nearest neighbors are determined. One of the k nearest neighbors is chosen at random, and the new instances is created at a random point between the two instances.

There are several variants of the basic SMOTE algorithm. Han, et.al. describe Borderline-SMOTE in which the minority class is oversampled but only at the decision boundary or borderline.[9] Nguyen, et.al[11] build up on this by proposing SVM-SMOTE which also oversamples instances only along the borderline. In SVM-SMOTE, the borderline is approximated by training an SVM and obtaining the support vectors. New instances are created by using both interpolation (new instances are created between two existing instances) like the traditional SMOTE algorithm, or extrapolation (in the line joining two instances but not between them). SMOTE can also be combined with an under-sampling method. TOMEK-SMOTE is one such method in which SMOTE oversampling is combined with TOMEK-LINK under-sampling at the boundaries.[12] Batista, et.al. propose SMOTE-ENN[17] which combines SMOTE with the *edited nearest-neighbor*[18] technique

The basic SMOTE can also be combined with a boosting learner. This is similar to RUSBoost, except that SMOTE oversampling is used at each stage instead of random under-sampling.[13]

Chen, et.al, propose a method for Balanced Random Forest where the following is done:[15] A bootstrap of instances is drawn from the minority class at every step of the iteration. The number is matched from the majority class by sampling with replacement. A CART Tree is used with the modification that only a random set of features is drawn at each iteration, and one feature from that reduced set is used to take the decision where to split.

Chan, et.al also propose a weighted Random Forest mechanism where different weights are assigned to the different classes.[15] The class weights are used in the algorithms in two places: at each iteration in the Gini criterion to find the splits, and finally the prediction is made by a weighted majority vote of all the classifiers.

3 Methodology   
  
*3.1 Establishing a baseline*

The data was first read as a pandas data frame and subjected to initial visual exploration. There were no missing data-points. Two columns were deemed to be irrelevant to the prediction and were dropped: RowNumber and Customer ID.

Generally, outlier detection and removal is performed before scaling, but in this case, standardization was performed on the columns 'CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary' before outlier detection. Early scaling was more efficient as it allowed us to visualize all the box-plots in the same scale. This was also deemed procedurally sound as scaling does not affect the frequency distribution and all data points scale by equal amounts so outliers remain outliers. The distribution was similar for the field CreditScore as well, and CreditScore was also allowed to remain as is for the same reasons.

A box-plot was plotted and visual examination of the box plots was undertaken to identify the outliers. The fields Age and NumOfProducts were found to have items beyond the whiskers in the outliers. The outliers in NumOfProducts was clipped to 1.5 IQR around the mean. For the fields ‘Age’ and ‘CreditScore’, there were a significant number of instances beyond the whiskers. However, these instances formed a continuum of points starting from the whiskers and extended beyond, hence it was decided that they are not true outliers, and were allowed to remain. *[Handle Outliers]*

The categorical features Gender and Country were converted to one-hot encoding. *[Engineer Features]*

An early attempt at feature selection was undertaken to get an idea of the importance of the features. This was revisited again later. *[Feature Selection Initial Exploration]*. Three different techniques for feature selection were used: SelectPercentile(), RandomForest classification followed by comparison of feature\_importances\_, and the greedy method using RFECV().

Throughout the study, several metrics were used to evaluate the performance of the models including Precision, Recall, accuracy and F1-Score. However, all decisions were taken using the F1-Score.

Also, throughout the study, 6-fold cross validation was used to evaluate all models. Ideally, we would have liked to use a higher number of folds, but we chose this number for the limitations on time.

Following this, a host of algorithms were compared with default parameters to see their performance on the dataset. *[Compare Basic Models].* The algorithms that were evaluated are as follows:

1. DecisionTreeClassifier
2. KNeighborsClassifier
3. NearestCentroid
4. NaiveBayes
5. Support Vector Machines
6. Random Forest
7. Ada Boost
8. Gradient Boost
9. Linear Discriminant Analysis
10. Quadriatic Discriminant Analysis
11. Logistic Regression
12. Ridge Classifier
13. Bagging Classifier
14. SGD Classifier
15. Passive Aggressive Classifier
16. Perceptron
17. Multi-Layer Perceptron

To check if we were correct in not clipping the outliers in the columns CreditScore and Age, the same algorithms were run after clipping these columns and seeing the accuracy scores. *[Does clipping CreditScore and Age give better results?]*

The same algorithms were then run again, this time after SMOTE oversampling was applied on the dataset. *[Evaluate the same models with SMOTE]*.

Hyper-parameter optimization were performed on the four best performing models using GridSearchCV (SMOTE was applied) *[Hyper Parameter Optimization]*:

1. Gradient Boosting *[Test GradientBoost with GridSearchCV]*
2. Random Forest *[Test RandomForest with GridSearchCV]*
3. Support Vector Classifier *[Test SVC with GridSearchCV]*
4. Multi-Level Perceptron *[Test MLP with GridSearchCV]*

***Gradient Boosting***

The Hyper-parameters tried for Gradient Boosting were optimized in two stages. First the following:

|  |  |
| --- | --- |
| n\_estimators | 10, 100, 1000 |
| max\_depth | 3, 10 |
| min\_samples\_split | range(2, 210, 40) |
| max\_features | None, auto, sqrt, log2 |

Once the best parameters were found, they were fixed, and a further search was performed for the following:

|  |  |
| --- | --- |
| n\_estimators | 10, 100, 1000, 10000 |
| learning\_rate | 0.01, 0.05, 0.1, 0.15, 0.2 |

***Random Forest***

Again, hyper parameters were optimized in two stages. The first stage:

|  |  |
| --- | --- |
| n\_estimators | 10, 100, 1000 |
| criterion | Gini, entropy |
| min\_samples\_split | Range(20, 200, 40) |
| max\_depth | None, 5 |
| class\_weight | Balanced, balanced\_subsample |

Once an optimal parameter set was found, the following were varied with the rest being set:

|  |  |
| --- | --- |
| n\_estimators | 1000, 10000 |
| min\_impurity\_decrease | 0, 0.00001, 0.0001, 0.001, 0.01, 0.1 |

The second optimization actually produced worse results than the first hyper-parameter set, and the second set of hyper-parameters were discarded.

***Support Vector Classifier***

The following hyper-parameters were tried:

|  |  |
| --- | --- |
| SVC kernel | Linear, poly, rbf, precomputed, sigmoid |
| SVC class\_weight | None, balanced |
| SVC gamma | Scale, auto, 0.001 |
| SMOTE k\_neighbors | 2, 20 |

**Multi-Level Perceptron:**

|  |  |
| --- | --- |
| Hidden layer sizes | (100), (100, 50) |
| Activation function | Relu, logistic |
| Solver | Adam |
| Learning Rate | Constant, Adaptive |

After this step, all evaluations were done only on the first three methods as Multi-Level perceptron was slow to train and also didn’t produce results that were as good as the first three methods even after hyper-parameter optimization.

A baseline was established with the first three methods

***3.2 Basic Experimentation***

The first experiment was to see if IsolationForest could be used to remove outliers.

Feature Selection was then applied to the models to see if it resulted in any benefit. *[Compare feature selection methods]*. A stage was added to the pipeline for feature selection. The following feature selectors were tested:

1. LinearSVC
2. LinearSVC with L1 penalty
3. Variance Threshold of 0.8
4. Greedy RFECV with RandomForest Classifier
5. Chi2 along with family-wise error-based selector
6. ANOVA F-value with family-wise error-based selector
7. ExtraTreeClassifier to select features

So far, standardization was applied to the data to standardize it to 0 mean and unit standard deviation. At this stage, it was tested if MinMaxScaling would make any difference or not. *[See if MinMax scaling makes a difference]*.

A third experiment was performed where PCA was applied along with feature selection. PCA was added to the pipeline after the feature selection and before the classifier stage. PCA with a varying number of axes [5, 14] were tried. *[Add PCA to the mix]*

***3.3 Research Experimentation***

This section contains experiments to mitigate the class imbalance. The first experiment tried was to experiment with the class weight parameter. The following variations of the class weight were tried: balanced, balanced\_subsample, 0.2:0.8, 0.6:0.4, 4:1, 4:1.33, 6:1, 0.9:0.1, 0.2037:0.7963, 0.7963:0.2037. These runs were conducted without up-sampling the instances. *[Experiment with weighted features in Random Forest]*

Next a grid-search was performed on the following hyperparameters on a RandomForestClassifier (without up-sampling). *[Grid search with different weights (no SMOTE)]*

|  |  |
| --- | --- |
| n\_estimators | 10, 100, 1000 |
| criterion | Gini, entropy |
| min\_samples\_split | Range(20, 200, 40) |
| class\_weight | {0: i, 1: 1-i} for i in np.arange(0, 1, 0.05), ), {1:4, 0:1.33}, {1:4, 0:1}, {1:5, 0:1} |

The BalancedRandomForestClassifier from imblearn was evaluated with the help fo a grid-search. *[GridSearch BalancedRandomForestClassifier from Imblearn]*

The following parameters were used:

|  |  |
| --- | --- |
| n\_estimators | 10, 100, 1000 |
| criterion | Gini, entropy |
| min\_samples\_split | Range(20, 200, 40) |
| class\_weight | {0: i, 1: 1-i} for i in np.arange(0, 1, 0.05) |
| bootstrap | True, false |
| oob\_score | True, False |

The next approach that was attempted was to use the traditional RandomForest model that we got from the baseline and use that to predict the class but with alternate cut-offs. The method predict\_proba() was called, and different values of cut-off were tested with 6-folds cross validation to see which ones performed the best. The same was repeated after adding SMOTE up-sampling to see how it behaved. *[Random forest with Adjusted Cutoff]*

The next step was to compare how different sampling techniques compare against each other, and how different feature-selection algorithms work in conjunction with sampling. *[Classification with Other Samplers]*

The first step was to just compare the sampling techniques without adding feature-selection. The following were evaluated:

None, RandomOverSampling, RandomUnderSampling, SMOTE, ADASYN, SMOTEENC, BorderlineSMOTE, SVMSMOTE, SMOTEENN, SMOTETomek.

The best samplers were chosen and combined with different feature selection strategies.  
combinations of the following were tried:

Feature Selection: Variance Threshold 0.8, RFECV Random Forest, FWE f\_classif, None

Sampling: SMOTE, ADASYN, SVM-SMOTE, SMOTETomek, Random over-smapling, Random under-sampling.

The combinations of feature selection and sampling were sorted on the basis of the best f1-score.

To remove any confusion whether variance feature selection made any difference, or whether variance feature selection just chose all the features, the variance threshold was changed within a range to see if it made any difference.

The last experiment attempted was to use RUSBoost and BalancedRandomForestClassifier from imblearn.

A grid-search for hyper-parameters was undertaken with RUSBoost in two stages (without up-sampling):

Stage 1:

|  |  |
| --- | --- |
| n\_estimators | 1000, 10000 |
| learning\_rate | 1, 0.1, 0.01, 0.001 |
| algorithm | SAMME, SAMME.R |

Stage 2:

|  |  |
| --- | --- |
| n\_estimators | 10000, 100000 |
| learning\_rate | 0.001, 0.0001 |
| algorithm | SAMME.R |

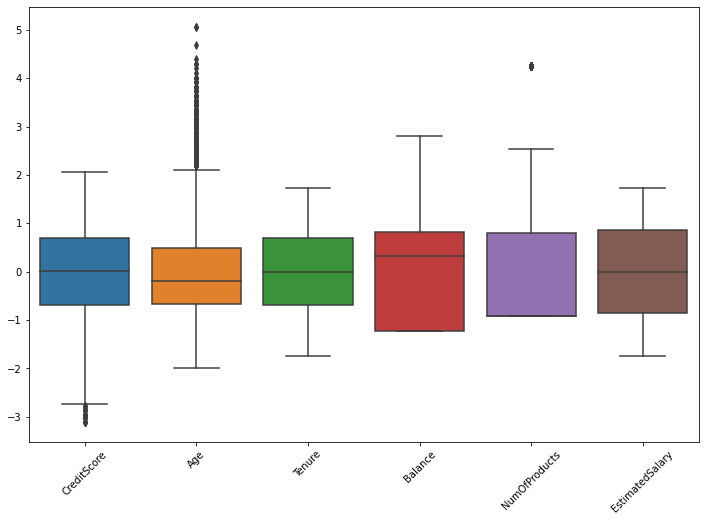
A grid search on BalancedRandomForestClassifier was undertaken with the following variations.

|  |  |
| --- | --- |
| n\_estimators | 10, 100, 1000 |
| criterion | entropy |
| min\_saples\_split | Range(20, 200, 40) |
| class\_weight | {0: i, 1: 1-i} for i in np.arange(0, 1, 0.1) |
| bootstrap | True, False |
| oob\_score | True, False |

4 Evaluation

Initial inspection of the data revealed no missing values. However, the heatmap showed moderate class imbalance, where 75% of the values had Exited=0 and 25% of the values had Exited=1.

There were several outliers in the data, as can be seen in the box-plot:

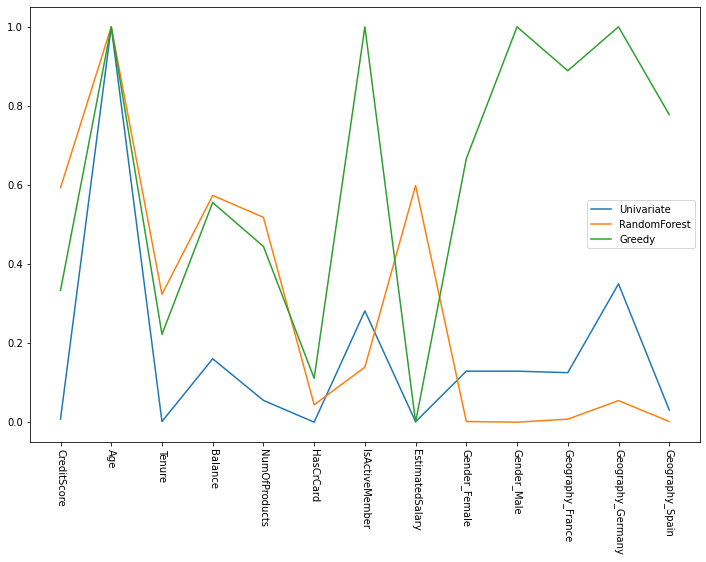


For the field NumOfProducts, the outliers were clipped, but for Age and CreditScore, outliers were not clipped because they form a continuum of points. In an experiment, when CreditScore and Age were clipped to within +/- 1.5IQR, the models performed worse than without clipping.

|  |  |  |
| --- | --- | --- |
| Algorithm | F1 score with Age and CreditScore Clipped | F1 Score without Age and CreditScore clipped |
| MLP | 0.590 | 0.599 |
| Gradient Boost | 0.577 | 0.580 |
| Random Forest | 0.572 | 0.574 |

While Random-Forest performed marginally better, overall other algorithms fared far worse. Random Forest’s better performance may also be down to the random choices. The over all conclusion here is that clipping the fields CreditScore and Age had a detrimental effect on the performance. This is also in line with what is expected since the data we clipped were not really outliers as they formed a continuum, and they have importance in deciding the outcome.

An initial exploration of feature importance with three methods resulted in no clear indication of redundant features. Below, the feature importances are plotted according to each algorithm. With all the features, using the three different methods to see which features are noisy, we don't get any clear indication about which features are important and which are not. HasCrCard is probably not important, but even that is not very clear, so we'll let things be. On the other hand, we know that Age, Balance, and IsActiveMember are probably very important.



Furthermore, when feature selection was added to the hyper-parameter optimized modules, no significant changes were noted. Below is a chart with the best feature selection algorithm applied to each of the hyper-parameter tuned models. In fact, in some cases feature selection reduced the F1 score slightly.

Also, when PCA was applied in addition to Feature Selection, the results didn’t change much, and in some cases became worse.

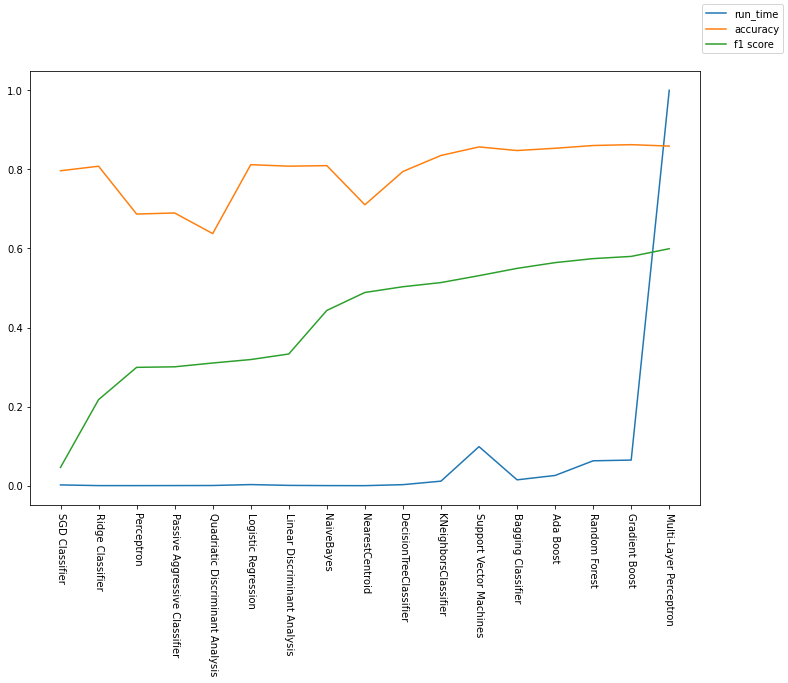


The addition of MinMaxScaling didn’t make much difference to the results.

It was evaluated whether IsolationForest could be used to remove outliers. However, IsolationForest turned out to be very sensitive to the random number passed to it, and the number of instances it treated as outliers varied wildly with the random number. Hence it was decided not to use IsolationForest to remove outliers. Below is the table describing the number of outliers predicted by IsolationForest. We only noticed this effect only because we once applied IsolationForest to remove outliers and then classified it – and the result we got was an F1-score approaching 1. That seemed too good to be true, so we investigated the cause of the dramatic improvement further. We suspect that IsolationForest removed all instances of the minority class in that instance, thereby giving a very good F1-score.

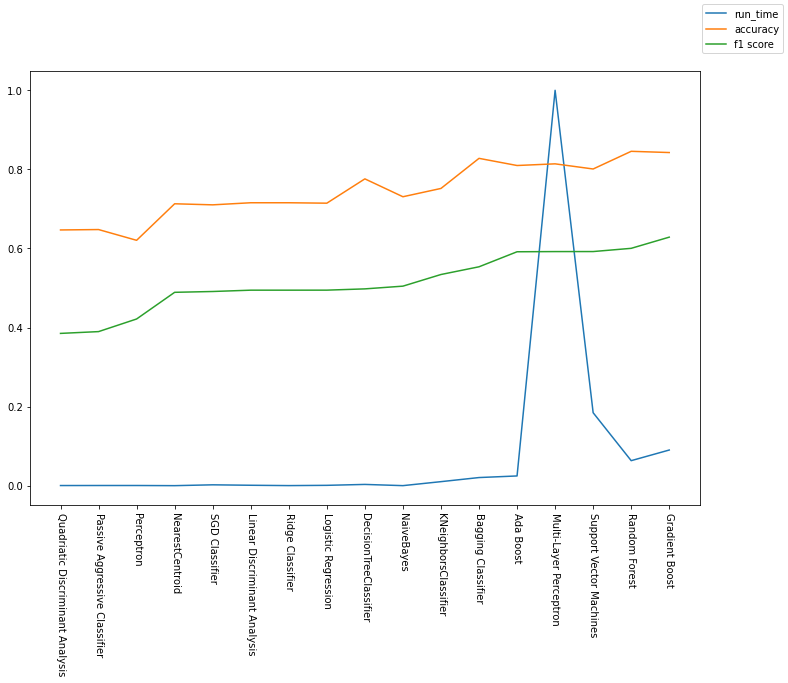


The basic comparison of the models without up-sampling gave the following output:



Here we notice that the run time of Multi-Layer Perceptron is several orders of magnitude higher than the rest of the algorithms. This additional cost of training and classification in terms of computation power and run time must also be accounted for. This can have implications in real-world applications regarding the choice of the algorithm.

The application of SMOTE led to an increase in the F1 score across the board (except for DecisionTree and Multi-Layer Perceptrons).



The F1-scores with and without SMOTE were compared, and the improvement as percentage sorted. What we observe is that SGD and LDA classifiers had a huge improvement. Ridge, Logistic Regression, Perceptron, Passive-Aggressive classifiers, and Quadratic Discriminant Analysis had good improvements. SVM, Bagging algorithms, Boosting algorithms, Nearest Neighbor/Centroid and Tree-based algorithms proved quite resistant to class imbalance. DecsionTree and Multi-Layer Perceptrons saw a small decrease in performance after introduction of SMOTE. In the data below, the benefit is calculated as a percentage:

|  |  |
| --- | --- |
|  |  |

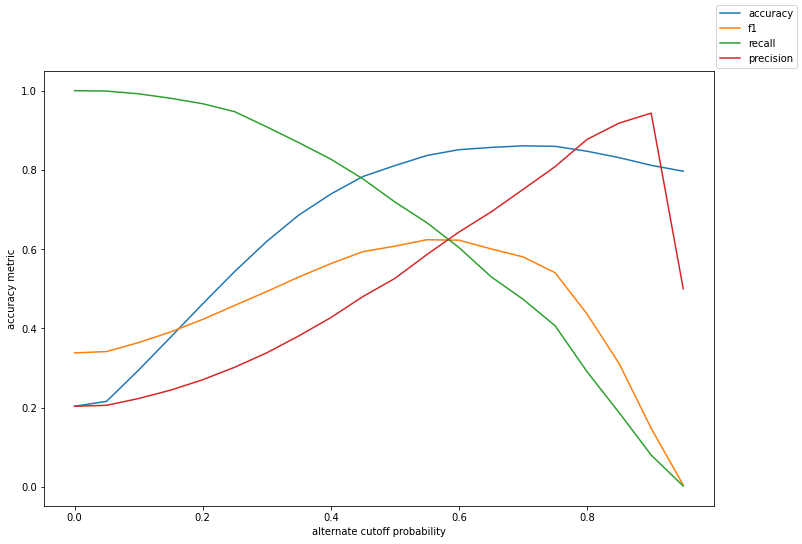
Basic testing with the default parameters ranked the algorithms as follows:



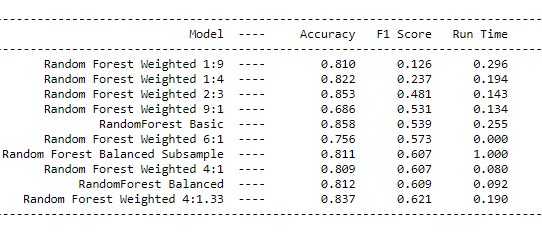
The best parameters after hyper-parameter optimization are as follows:

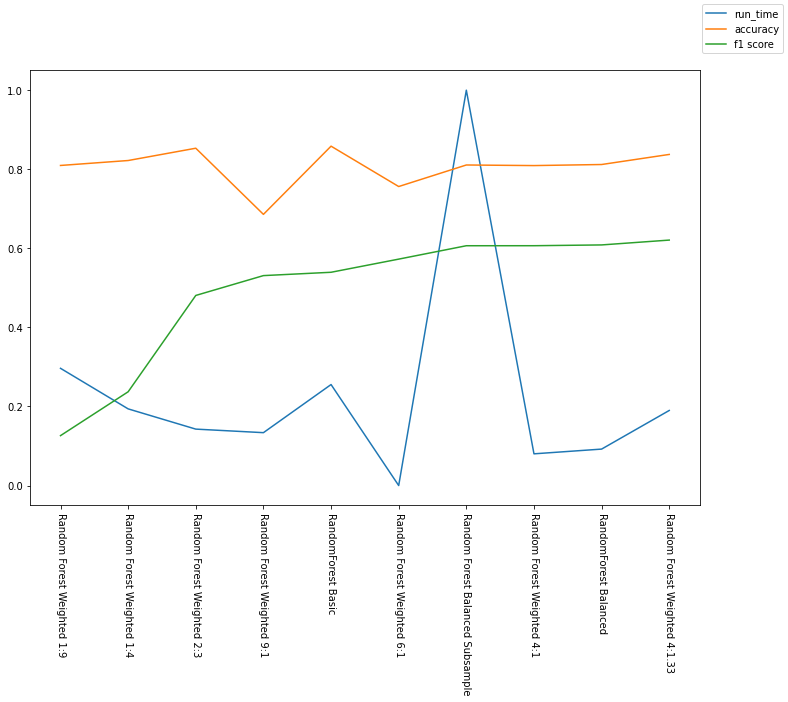
|  |  |  |
| --- | --- | --- |
| Algorithm | Params | best\_score\_ |
| Support Vector Machines | SMOTE\_k\_neighbors=20, class\_weight=None, gamma=’auto’, kernel=’rbf’ | 0.5958 |
| Gradient Boost | max\_depth=3, max\_features=None, min\_samples\_split=42, n\_estimators=100 | 0.6312 |
| Random Forest | class\_weight=’balanced\_subsample’, criterion=’entropy’, max\_depth=None, min\_samples\_split=100, n\_estimators=1000 | 0.6244 |

When alternate cut-offs were adjusted with Random Forest, the best performance was obtained at a cut off of 0.57 and the F1-Score was 0.6278. The curve plotted of the accuracy and F1 against the cut-off was as follows. We can see that there is a peak F1 score (at around 0.9) beyond which it drops sharply. The recall falls steadily as the cut-off is increased. The Precision increases steadily as the cut-off is increased and then drops drastically after a point.

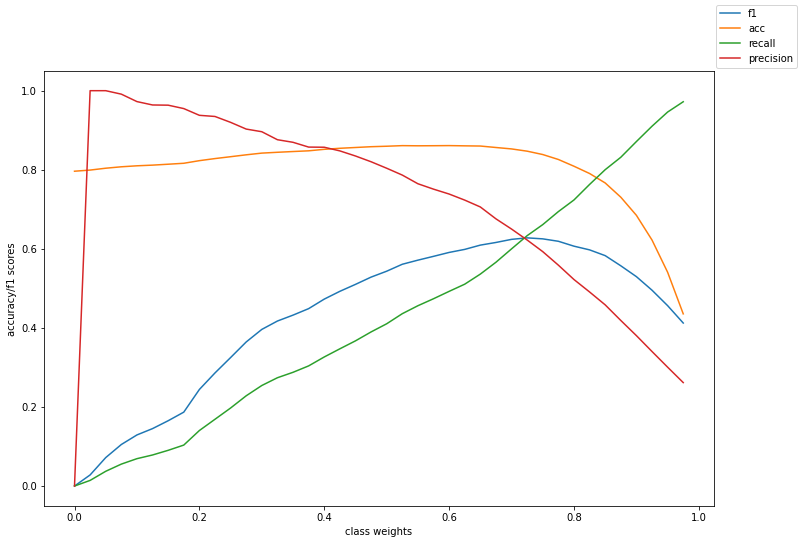


Random Forests with class weights also showed promise. When various parameters were tried, the best results were obtained from custom weights, and from the canned parameters: balanced, and balanced\_subsample. However, balanced\_subsample was much slower than all the other methods, although its accuracy was comparable to the balanced method.





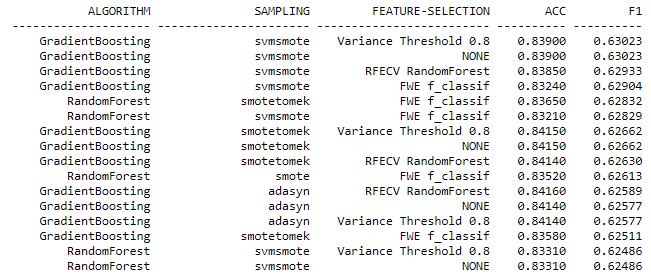
When the effect of different class weights were investigated further. The recall, accuracy and precision were plotted as the class weights changed. Here is the weight associated with the majority class. What we see here is that precision is highest when the weight associated with the majority class is low, and is at its best between 0.3 and 0.5 after which it starts falling. The recall steadily rises as the weight associated with the majority class is increased. The best F1 score is obtained at weight 0.75 (F1=0.6252). This is also roughly the ratio of the majority class in the sample. This opens up a possibility of combining various weighted Random Forest classifiers some with high low class weight to the majority class and others with a more balanced weight, and then pooling the results.



When all the sampling methods were compared, the performance of the different sampling methods in the following page. We see that SVMSmote performs the best of all the methods.

|  |  |
| --- | --- |
|  |  |
|  | |

When Feature Selection and Sampling were combined, the best performing combinations are listed below. Here Variance Threshold 0.8 and no feature selection gave identical numbers because both of them selected all the features.



The last experiment conducted was to use imblearn’s Balanced Random Forest Classifier as proposed by Chen, et.al[15]. This gave the best result of all the classifiers with an F1 score of 0.63291. RUSBoost was also tried, but its F1 score was low 0.5847.

The overall result is as follows.

|  |  |
| --- | --- |
| Algorithm | F1 Score |
| SVC |  |
| Unoptimized Support Vector Machines | 0.531 |
| Unoptimized SVM With SMOTE | 0.592 |
| Optimized SVM with SMOTE | 0.596 |
| Boosting Algorithms |  |
| RUSBoost | 0.585 |
| Gradient Boost | 0.58 |
| Gradient Boost With SMOTE | 0.629 |
| Optimized Gradient Boost with SMOTE | 0.631 |
| Gradient Boosting with SVMSMOTE | 0.63 |
| Random Forest Variations |  |
| Unoptimized Random Forest | 0.574 |
| Unoptimized Random Forest With SMOTE | 0.6 |
| Optimized Random Forest with SMOTE | 0.624 |
| Optimized Random Forest weighted 0.75:0.25 (NO SMOTE) | 0.625 |
| Optimized Random Forest with Adjusted Cut-offs | 0.628 |
| Optimized BalancedRandomForest (imblearn) | 0.633 |
| Optimized Random Forest with TOMEK-SMOTE | 0.628 |

**5 Conclusion**

We compared various methods of

References

1. Kuhn, M., Johnson, K., Applied Predictive Modeling, Springer, 2013
2. Weiss, G.M., Provost, F.: The effect of class distrubition on classifier learning. Technical report ML-TR 43, Department of Computer Scinece, Rutgers University, 2001
3. Ting, K.M., An instance weighting method to induce cost-sensitive trees. IEEE Transactions on knowledge and data engineering, Vol. 14, No. 3, 2002
4. Schapire, E., Freund, Y.: Experiments with a new boosting algorithm. Machine Learning: Proceedings of the Thirteenth International Conference, 1996.
5. Johnson, R.A. and Wichern D.W., Applied Multivariate StatisticalAnalysis, Pearson, 2007
6. Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C. J.: Classijication and Regression Trees. Wadsworth Books. 1984
7. Tomek, I. Two Modifications of CNN. IEEE Transactions on Systems Man and Communications SMC-6 (1976), 769–772.
8. Chawla et.al.: SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 2002
9. Han, H., et.al: A new sampling method in imbalanced data sets learning. Proceedings of the international conference on intelligent computing, 2005
10. Nguyen, et.al.: Borderline over-sampling for Imbalanced Data Classification, Fifth International Workshop on computational intelligence and applications, 2009
11. Japkowicz N. and Stephen, S.: The class imbalance program: a systematic study. Intelligent data analysis, vol 6(5), pp. 429-449, 2002
12. Batista, G., et.al.: Balanced training data for automatied annotation of keywords: a case study. WOB, 2003
13. Chawla et. al: SMOTEBoost: Improving Prediction of the Minority Class in Boosting. European conference on principles of data mining and knowledge discovery, 2003
14. https://www.kaggle.com/shrutimechlearn/churn-modelling
15. Chen, Chao, Andy Liaw, and Leo Breiman. “Using random forest to learn imbalanced data.” University of California, Berkeley 110 (2004): 1-12.
16. Seiffert, C. RUSBoost: Improving classification performance when training data is skewed. 19th International Conference on Pattern Recognition, 2008
17. Batista, E.A, et.al. A study of the behavior of several methods for balancing machine learning training data. ACM SIGKKD 2004
18. Wilson, D.L. Asymptotic properties of nearest neighbor rule using edited data. IEEE transactions on systems, man and cybernetics, 1972

1. In this discussion, we will talk about the case of a binary classification for simplicity. However, the same discussion can generalize to multi-class classification as well. [↑](#footnote-ref-1)
2. The reverse is also possible where -ve items are the minority and +ve the majority. However, for this discussion we will follow the convention that +ve cases are minority and -ve cases are the majority. This doesn’t change the core ideas. [↑](#footnote-ref-2)