**Capstone Project 1: Building ML Models**

In this section we explore 6 commonly used classification models including tree-based methods such as Random Forest, XGboost and LightGBM, as well as classical models such as Logistic regression with regularization, K-nearest neighbors (KNN) and support vector machine / classifier (SVM).

Due to the nature of the imbalanced data, we tried 2 sampling methods - stratified sampling to maintain the default vs. non-default ratio when splitting the data into training and testing sets; down sampling of the non-default counts to be approximately the same as the default counts in the training data, while the test data is still a randomly held out 20% imbalanced dataset.

Within the Random Forest framework, we trained one model using the imbalanced training data with GridSearchCV and one model using the same imbalanced training data with RandomizedSearchCV, as well as one model using down-sampled balanced training data with RandomizedSearchCV in Scikit-learn to select the optimal hyperparameters. The results show that without balancing the training data, every data point will be classified as non-default loan. Also, unlike GridSearchCV that goes through every combination of points in the parameter spaces, RandomizedSearchCV gives the modeler the flexibility to pick a fixed pre-specified number of combinations of parameters in the specified hyperparameter space, it is more computationally efficient and also produces better performances. For these reasons, all the other 5 models are trained using the down-sampled balanced training data and RandomizedSearchCV is used in selecting the best hyperparameters.

All metrics reported are based on predictions for the independent random 20% held out test set.

1. **Random Forest**

Random Forest is a well known bagging algorithm which builds hundreds or thousands of trees on bootstrap samples of data and a random subset of features to reduce overfitting. The model uses majority of votes to determine the final classification labels.

**1.1a Stratified Sampling with GridSearchCV (Imbalanced training data)**

In this section we used the imbalanced training data to fit a Random Forest model, using GridSearchCV in Scikit-learn to select the best hyperparameters. We tuned number of trees and max depth parameters, searching from n = 100, 200 and 400 trees and max depth = 5, 10, 20. Totally 9 parameter combinations. Due to the long training time, we did not try n greater than 400, but it can be done in future work if computation power permits. In this case, the best parameters are n = 400 and max depth = 20.

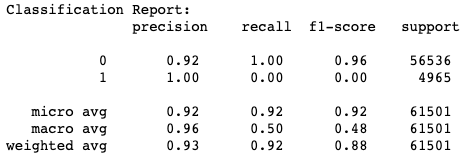
As shown in the confusion matrix below, the model classifies all but one point as non-default. This is not a useful model although the model accuracy is 92%, only one case of the actually defaulted loans is classified correctly.

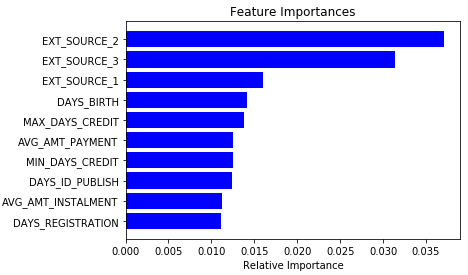
|  |  |  |
| --- | --- | --- |
| Confusion Matrix | Predicted | |
| Actual | Non-Default | Default |
| Non-Default | 56,536 | 0 |
| Default | 4,964 | 1 |

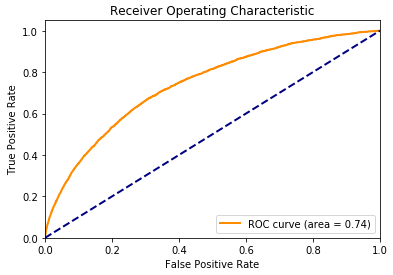
Detailed classification report, top 10 features selected by the model and the ROC curve are shown below.

The top 3 features are normalized score from external data source, which Home Credit did not reveal the sources and meaning. The rest important variables are age, maximum (aggregated out of all the Bureau IDs belonging to the same current application ID) number of days before current application that client apply for Credit Bureau credit, average amount of payment on previous credit, and so on.

The ROC curve can help analysts find a tradeoff point between the true and false positive rate that Home Credit’s risk policy can undertake. Using some financial assumptions such as how much Home Credit would lose if one default case is mistakenly predicted as non-default case, or how much profit Home Credit would miss if one non-default case is incorrectly classified as default case, the company can get a profit and loss forecast financial sheet and make the most appropriate decisions on what threshold to choose.



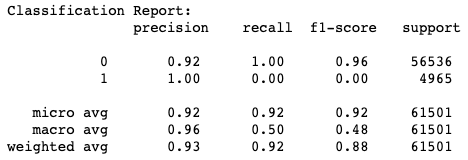


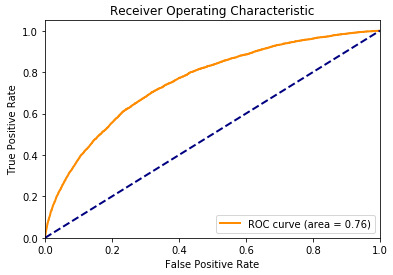


**1.1b Stratified Sampling with RandomizedSearchCV (Imbalanced training data)**

In this model we used the same imbalanced training data to fit a Random Forest model, using RandomizedSearchCV in Scikit-learn to select the best hyperparameters. We asked the model to select 5 combinations of parameters from n = 100, 200, 300, 400 and max depth ranging from 1 to 19, and the optimal parameter combinations chosen by randomized search cross validation is n = 300 and max depth = 17.

Similar to model 1.1a, the model classifies all but one cases as non-default. But the AUC in this case is slightly higher than 1.1a. Considering the computation cost and the higher AUC using RandomizedSearchCV, we choose to use RandomizedSearchCV in all the rest of the models.





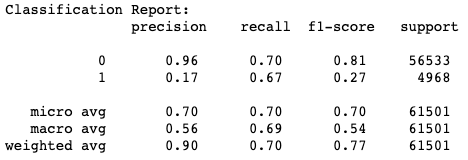
**1.2 Down Sampling with RandomizedSearchCV (Balanced training data)**

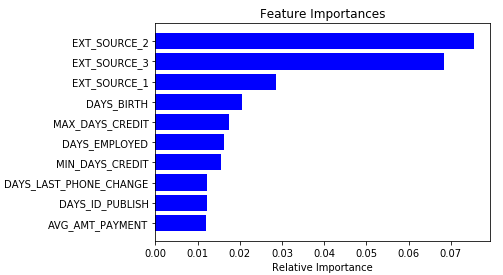
As we already mentioned in 1.1a and 1.1b, the model using imbalanced training data only classifies one default case correctly. In all of the upcoming models, we use down sampled balanced training data to fit the model, and apply the model on the independent imbalanced test data to evaluate the model performance.

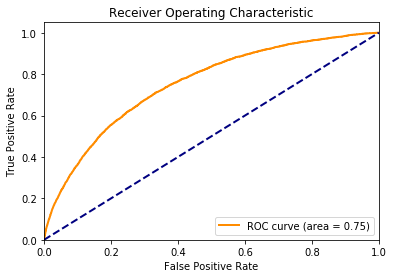
For this model, we used the same parameter space as 1.1b, and the best combination found by RandomizedSearchCV is n = 400 and max depth = 12.

In the following confusion matrix and report, we can see the precision and recall rates are higher than the imbalanced training data cases. Best 10 features selected are similar to model 1.1a.

|  |  |  |
| --- | --- | --- |
| Confusion Matrix | Predicted | |
| Actual | Non-Default | Default |
| Non-Default | 39,796 | 16,737 |
| Default | 1,636 | 3,332 |



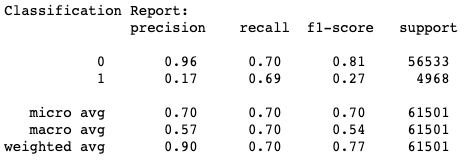


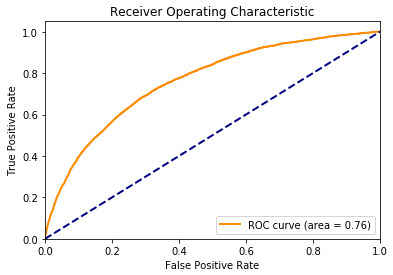


1. **Logistic Regression with Regularizations**

In this model we searched over L1, L2 penalty, and regularization strength parameter C in the list of [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1, 1, 10, 100]. It turned out L1 penalty with C = 0.1 is the best combination. The recall rate is slightly higher than model 1.2 using Random Forest.

|  |  |  |
| --- | --- | --- |
| Confusion Matrix | Predicted | |
| Actual | Non-Default | Default |
| Non-Default | 39,821 | 16,712 |
| Default | 1,554 | 3,414 |

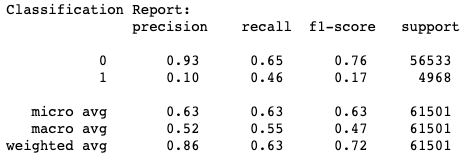


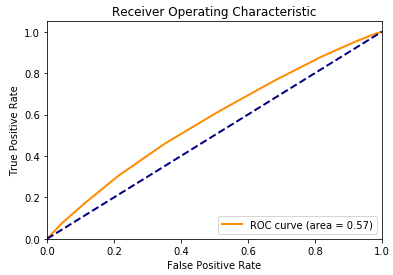


1. **K-Nearest Neighbors (KNN)**

In this model, we searched the best K from 1 to 19, and it turned out 14 is the best. The model performance is not as good as Random Forest and Logistic Regression, possibly due to the high dimension of the feature space, as it is known that algorithms that use distance as measure to classify “close by” points will suffer from curse of dimensionality.

|  |  |  |
| --- | --- | --- |
| Confusion Matrix | Predicted | |
| Actual | Non-Default | Default |
| Non-Default | 36,670 | 19,863 |
| Default | 2,686 | 2,282 |

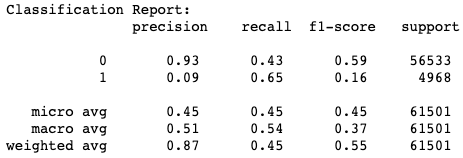


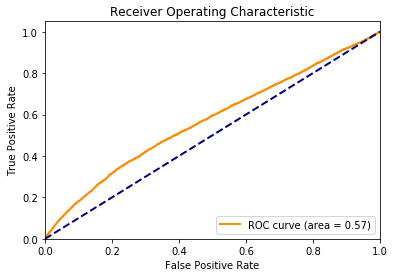


1. **Support Vector Machine (SVM)**
   1. **Linear SVM**

Due to the long training time, we used 2-fold CV and only tried 3 possible combinations of parameters. The penalty is fixed at L2 and the best regularization parameter C selected by RandomizedSearchCV is 9. As can be seen below, the performance is not very good.

|  |  |  |
| --- | --- | --- |
| Confusion Matrix | Predicted | |
| Actual | Non-Default | Default |
| Non-Default | 24,339 | 32,194 |
| Default | 1,734 | 3,234 |



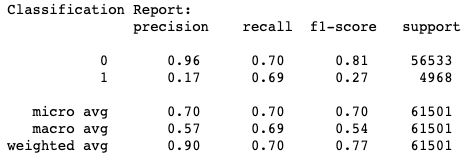


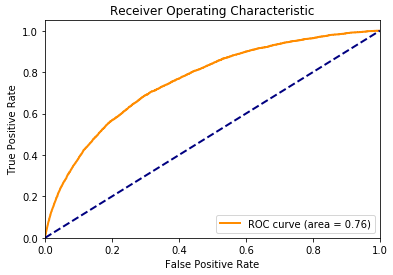
* 1. **Kernel SVM**

Kernel SVM is computationally intensive. To help convergence, we used MinMaxScaler in Scikit-learn to scale the features before training the model. Regularization parameter C is selected from range 1 to 100, kernel functions are chosen from RBF, Polynomial and Linear, and kernel coefficient gamma is chosen from the list [0.001, 0.0001]. The best parameters selected are C = 35, RBF kernel and gamma = 0.001.

The performance of Kernel SVM is better than Linear SVM, and is comparable with Random Forest and Logistic regression. Taking into account the computation time, the later 2 models are preferable.

|  |  |  |
| --- | --- | --- |
| Confusion Matrix | Predicted | |
| Actual | Non-Default | Default |
| Non-Default | 39,700 | 16,833 |
| Default | 1,556 | 3,412 |



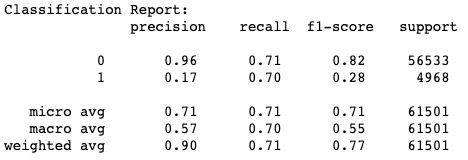


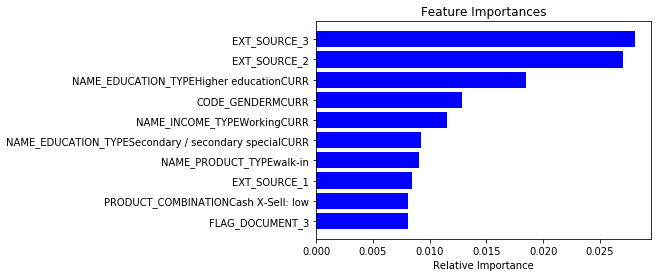
1. **XGboost**

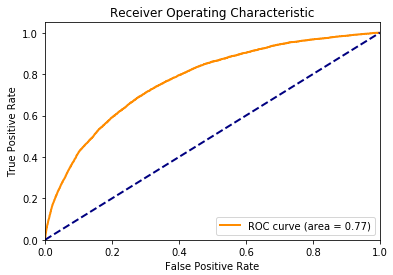
XGBoost is a scalable and accurate implementation of gradient boosting machines and it has proven to push the limits of computing power for boosted trees algorithms as it was built and developed for the sole purpose of model performance and computational speed.

In this model we tuned 4 parameters, number of trees, learning rate, max depth and minimum loss reduction required to make a further partition on a leaf node of the tree - gamma. The optimal n = 400, learning rate = 0.1, max depth = 5 and gamma = 0.01.

|  |  |  |
| --- | --- | --- |
| Confusion Matrix | Predicted | |
| Actual | Non-Default | Default |
| Non-Default | 40,172 | 16,361 |
| Default | 1,449 | 3,469 |







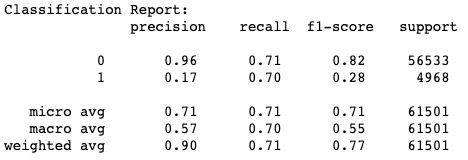
1. **LightGBM**

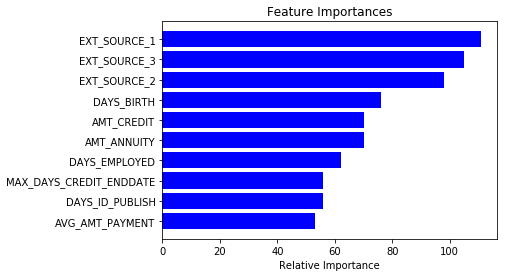
LightGBM uses XGboost as a baseline and outperforms it in training speed and dataset sizes it can handle. Some advantages of LightGBM:

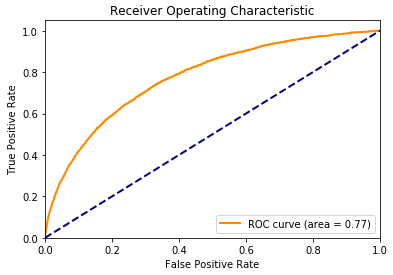
1. Faster training speed and higher efficiency.
2. Lower memory usage.
3. Better accuracy.
4. Support of parallel and GPU learning.
5. Capable of handling large-scale data

We tuned number of trees, learning rate and max depth and LightGBM produces the best results among all of the model we trained.

|  |  |  |
| --- | --- | --- |
| Confusion Matrix | Predicted | |
| Actual | Non-Default | Default |
| Non-Default | 40,188 | 16,345 |
| Default | 1,514 | 3,454 |







**Conclusions:**

Table below shows the accuracy and AUC score for all of the models we trained and tested. With imbalanced training data, model accuracy is not a good measure for performance, as a trivial model that classifies all points as non-default would have a high accuracy. Using balanced training data, LightGBM has the highest AUC and Linear SVM ranks last, only slightly better than random guess.

|  |  |  |
| --- | --- | --- |
| Model | Accuracy | AUC |
| Random Forest – 1.1a | 0.9193 | 0.7413 |
| Random Forest – 1.1b | 0.9193 | 0.7554 |
| Random Forest – 1.2 | 0.7013 | 0.7524 |
| Logistic Regression | 0.7030 | 0.7619 |
| KNN | 0.6334 | 0.5737 |
| Linear SVM | 0.4483 | 0.5715 |
| Kernel SVM - RBF | 0.7001 | 0.7597 |
| XGboost | 0.7096 | 0.7738 |
| LightGBM | 0.7096 | 0.7748 |