Project 3 Report: Lending Club Loan Status

CS 598 Practical Statistical Learning

Graham D Chester, 15-Nov-2018

Method

In summary, the overall approach to this project was to firstly to review course materials and Kaggle for information to support Exploratory Data Analysis (EDA) and modelling. A detailed EDA was then performed to obtain a good understanding of the available data, then a model selection process was run on a subset of the data to establish likely high performance models, and finally three selected models were tuned on various subsets of the data to obtain the lowest log-loss scores.

The technology used for this project was Jupyter Notebooks, Python, with Scikit-learn and XGBoost (0.81) libraries. The modeling was run on an iMac quad-core i7 4.2GHz, 40GB RAM, and the runtime of the Python code is approximately 80 minutes. The metric used for this report is log-loss as required.

Data Exploration and Feature Engineering

Each data field was explored for patterns, null values, anomalies, cross-correlations and relationship to the loan default rate. Log transforms were tried on the skewed numeric fields but this was of minimal or no benefit to so this idea was dropped. Consolidation of values in the 'emp_title', 'title' and 'zip_code' fields was attempted but also resulted in no performance improvement, and so since these fields contained many values with only a very weak correlation with loan default rates, they were also dropped.

Correlations were then checked, and Grade and 'fico_range_high' were dropped as they were very highly correlated with sub-grade and 'fico_range_low' respectively. 'Loan_amount' was 0.95 correlated with installment, but both were retained as there may still be some information available from both. Several fields ('dti', 'revol_util', 'mort_acc', 'pub_rec_bankruptcies') contained NA values, and after careful checking, these NA's were replaced with zeros. The 'earliest_cr_line' field was converted to an integer offset from Jan-1950 to aid modelling.

Algorithms

A broad range of algorithms were applied (with basic tuning) using a 10% sample of the train/test data from each of the three folds provided, to determine their potential – results shown in Figure 1 below. Clearly log-loss performance was the main criteria, but run-time was also important due to the relatively high volume of data. Some classifiers were near or even below the log-loss levels required, even with only basic tuning.

K-Nearest Neighbors performance was not as good as Logistic Regression and was almost 100 times slower, so was dropped. Both linear and RBF kernel-based SVM models run-time was much too long to even appear in the preliminary results. The Decision Tree classifier and Random Forest performance was middle-ground, though Random forest was very slow. Adaboost was the worst performer, and Gaussian Naïve Bayes was also not a good performer. Both Quadratic Discriminant Analysis and Linear Discriminant Analysis reported warnings regarding collinear columns and were relatively slow.

The three models chosen were Logistic Regression for its simplicity, good performance, minimal tuning, and very short run-time. Scikit-Learn's Gradient Boosted Classifier (GBM), and the XGBoost classifier were also chosen for their excellent performance, despite the relatively long run times.

Initial Comparative Classifier Performance

```
SUMMARY:
LogisticRegression              mean logloss=   0.45642
KNeighborsClassifier            mean logloss=   0.49338
DecisionTreeClassifier          mean logloss=   0.47388
RandomForestClassifier          mean logloss=   0.45368
AdaBoostClassifier              mean logloss=   0.68068
GaussianNB                      mean logloss=   0.52045
LinearDiscriminantAnalysis      mean logloss=   0.45700
GradientBoostingClassifier      mean logloss=   0.44583
XGBClassifier                   mean logloss=   0.44482
CPU times: user 20min 52s, sys: 18.7 s, total: 21min 10s
Wall time: 24min 43s
```

Figure 1

Hyper-Parameter Tuning

Tuning the logistic regression model could be done on an entire data fold due to its fast performance and limited tuning parameters (penalty='l1' or 'l2', and C).

However tuning GBM and XGBoost was performed initially on only 10% of the available data (due to long run-times) to get ball-park estimates for the best hyper-parameters. This initial tuning used grid search with 5 fold cross-validation (CV) on learning rate and number of estimators, then on maximum tree depth and minimum samples for a leaf, then on gamma, then on subsample and colsample by tree, and finally on the lambda and alpha regularization parameters. The parameters for GBM for tuning were slightly different to the above, but the same process was followed.

These parameters resulted in models that comfortably beat the baseline required, however it was possible to optimize further by running a randomized search on ranges of parameters around the best ones found as described above, and on the first full test fold. This search ran for approximately 18 hours, and resulted in a small but significant improvement across all three folds.

Results

Once the tuning was complete, the logistic regression, GBM and XGBoost classifiers were run on the 3 folds provided in the assessment instructions with the results shown in Figure 3 below. The hyper-parameters for these final models are included in the Appendix.

Model Performance Summary

Model	Test1	Test2	Test3	Average
Logistic Regression	0.4555	0.4565	0.4558	0.4559
Scikit-learn GBM	0.4400	0.4417	0.4404	0.4407
XGBoost	0.4390	0.4406	0.4396	0.4396

Figure 2

Conclusion

The performance of the simplest model, Logistic Regression, was not quite as good as the two Gradient Boosted models, but it performed exceptionally well considering its simplicity and very short run-time. Somewhat surprisingly, it came close to achieving a lower mean log-loss (0.4559) than required.

The best performing models were XGBoost and GBM, with XGBoost having the best performance at a mean log-loss of 0.4396 over the 3 folds. It should be noted that performance results only slightly worse than those achieved could be obtained with model run-times less than a quarter of what these models took.

Acknowledgements

Course materials and Kaggle (https://www.kaggle.com/pileatedperch/predicting-charge-off-from-initial-listing-data) provided useful background, along with documentation for scikit-learn and XGBoost. All Python code, Jupyter notebooks, parameters etc. are my own.

Appendix

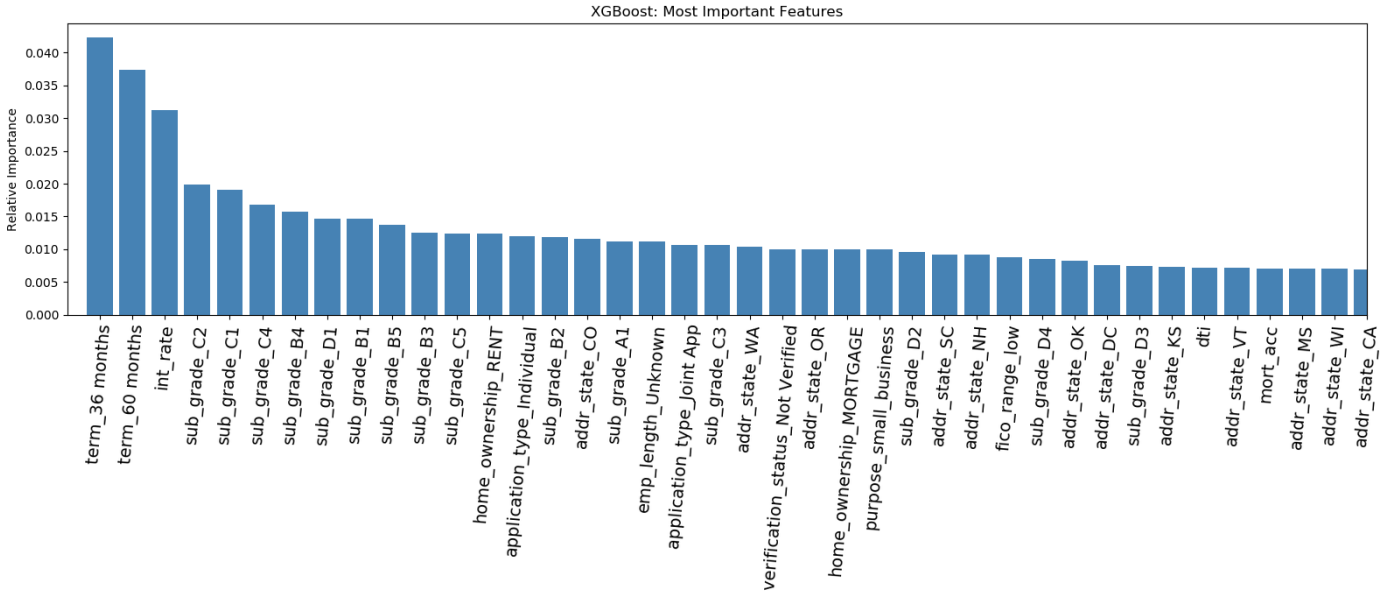
A) Model Hyperparameters

1) LogisticRegression(penalty='l1',C=1)

2) GradientBoostingClassifier(max_features='sqrt', learning_rate=0.055, n_estimators=780, max_depth=7, min_samples_leaf=2, subsample=0.9, min_samples_split=4, min_weight_fraction_leaf=0)

3) xgb.XGBClassifier(learning_rate=0.037, n_estimators=860, min_child_weight=8, max_depth=7, gamma=0.3, subsample=0.52, colsample_bytree=0.92, reg_lambda=0.67, reg_alpha=0.03)

B) Most Important Features (XGBoost)



C) Detailed performance and Run-times

```
LogisticRegression         Fold=0, Runime=    3.28 seconds, logloss= 0.45552
GradientBoostingClassifier Fold=0, Runime=  807.26 seconds, logloss= 0.44001
XGBClassifier              Fold=0, Runime=  576.60 seconds, logloss= 0.43903

LogisticRegression         Fold=1, Runime=    3.20 seconds, logloss= 0.45645
GradientBoostingClassifier Fold=1, Runime=  767.21 seconds, logloss= 0.44165
XGBClassifier              Fold=1, Runime=  569.36 seconds, logloss= 0.44060

LogisticRegression         Fold=2, Runime=    3.20 seconds, logloss= 0.45575
GradientBoostingClassifier Fold=2, Runime=  774.41 seconds, logloss= 0.44041
XGBClassifier              Fold=2, Runime=  567.80 seconds, logloss= 0.43956

SUMMARY:
LogisticRegression         mean logloss= 0.45591
GradientBoostingClassifier mean logloss= 0.44069
XGBClassifier              mean logloss= 0.43973
CPU times: user 3h 59min 44s, sys: 52.2 s, total: 4h 36s
Wall time: 1h 8min 42s
```

Performance on fold 'test1' using supplied R script 'evaluate.R'.

```
graham@Grahams-iMac: ~/Documents/UIUC/CS598_Practical_Statistical_Learning/project3
$ time python mymain.py
Reading and processing data files...
Starting Modelling...
Created mysubmission1.txt, rows= 253200 in 8.11 secs
Created mysubmission2.txt, rows= 253200 in 1288.99 secs
Created mysubmission3.txt, rows= 253200 in 901.95 secs

real    37m23.426s
user    93m59.616s
sys     0m23.732s
graham@Grahams-iMac: ~/Documents/UIUC/CS598_Practical_Statistical_Learning/project3
$ Rscript evaluate.R
[1] 0.4555830 0.4400147 0.4390305
graham@Grahams-iMac: ~/Documents/UIUC/CS598_Practical_Statistical_Learning/project3
$ cat proj_3.csv
0.455582976222659
0.440014740429023
0.439030453284695
1.01725260416667e-06
graham@Grahams-iMac: ~/Documents/UIUC/CS598_Practical_Statistical_Learning/project3
$
```