Credit Analysis

Rafa Martinez-Avial

November 11, 2020

1 Introduction

I will perform a classification problem with 101,503 test observations and 150,000 train observations. Each observation contains 10 explanatory variables and 1 response variable. The report consists of the following tasks:

1. Clean up the `train` and `test` datasets.
2. Try different learners on the `train` datasets and select the one with the lowest misclassification rate.
3. Apply the chosen learner to the `test` datasets.

The response variable, `SeriousDlqin2yrs`, states whether a person experienced 90 days past due delinquency or worse. I used 10 explanatory variables:

1. `Age`
2. `MonthlyIncome`
3. `NumberOfDependents`
4. `DebtRatio`
5. `NumberOfTimes90DaysLate`
6. `RevolvingUtilizationOfUnsecuredLines`
7. `NumberOfTime30-59DaysPastDueNotWorse`
8. `NumberOfOpenCreditLinesAndLoans`
9. `NumberRealEstateLoansOrLines`
10. `NumberOfTime60-89DaysPastDueNotWorse`

All of the code written for this task is included in this repository:
https://github.com/itsrafa/CreditAnalysis

2 Data Cleaning

My first challenge consisted of isolating and imputing the observations with missing values in some of the variables. I did this twice in two different ways.

- The first method consisted of two steps. First, simply fill in the missing values with the medians of the same explanatory variable in the other observations. Second, add an explanatory variable indicating whether an observation had any value missing.
- For our second method, I performed a linear regression in the missing entries of one of the variables with missing entries with respect to all the other variables. Then, did the same with a second variable with missing entries (this time with respect to every variable other than this second variable). I cycled back and for in a similar manner to the back-fitting algorithm, until I saw that the new regressions didn't update the values at all. I found that this happened after 5 iterations.

I decided to proceed with the analysis of the data after imputing it via the *first* method for the following observation. If missingness of a variable in an observation is actually an informative factor in predicting the output of our response variable (WLOG, the debtor defaults), the regression is likely to fill in the missing value with a value of the corresponding column that also increases the probability of default. Thus, by using the second method, we would be reinforcing the impact that a missing variable has on defaulting, potentially beyond its actual explanatory power. If, however, we fill in the missing entries with a neutral value (e.g. the median), we'd be letting the learner associate all the explanatory power of missingness to the newly created variable `MissingEntries` without reinforcing it.

3 Support Vector Machine

We used the support vector machine learner on a subset of the training data that ignored the ID column and response variable column. SVM delivered promising results, such as an misclassification error rate 93.3% on the training set. It's AUC, however, was only of 65.7%.

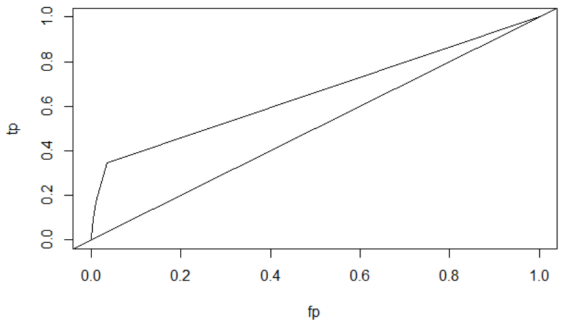


Figure 1: This is the ROC curve, showing SVM's performance as a classifier. `fp` is the false positive proportion and `tp` is the true positive proportion.

Unfortunately, this learner has a very long run time, especially if one tries to cross-validate over many values of gamma and cost. We hope that the misclassification error was similar in the testing set. In the interest of time, support vector machine is likely not the preferable learner, as it requires

being run overnight for many hours and has less impressive misclassification rate and AUC value than one of the following learners.

4 Generalized Linear Models

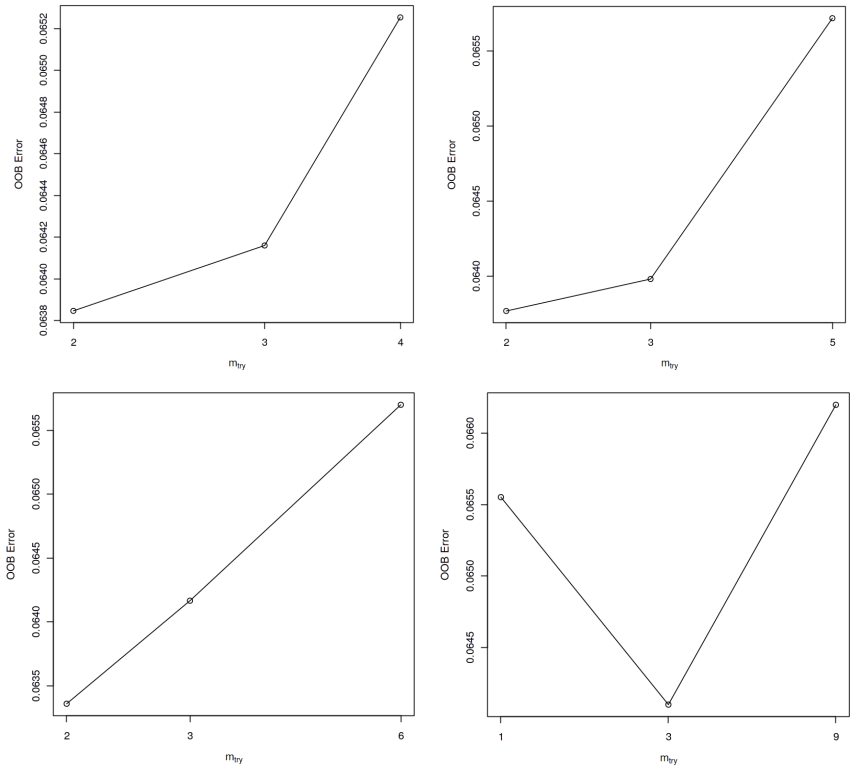
Generalized linear models are good at handling different types of data (those which fit a distribution that differs significantly from a multivariate normal). We used GLM on the training data, setting our response variable to `SeriousDlqin2yrs`. Our results, however, were not very promising. The majority of *p-values* for the relationships between our predictor variables and the response variable were significantly high. My analysis showed, surprisingly, that variables like `RevolvingUtilizationOfUnsecuredLines`, `age`, and `MonthlyIncome` were not significant.
To evaluate this learner, I looked at different parameters. The *Null Deviation* represents how well the response variable is predicted. A higher null deviation indicates that our predictions are not very good. In our case, we got a null deviation of 73616 on 149999, which is not very good at all. The *Fisher Score* tells us how many iterations a model takes to converge. In my case, it was 6. This is a pretty good number considering the volume of our data. *AIC* assesses the quality of the model by comparing it to other similar models, and penalizes one if the data is very complex. Considering the high dimensionality and volume of our data, it makes sense that our AIC is high. We then wanted to do a *Goodness of Fit* test to see how well our model fit the observed data. We decided to use the *Hosmer-Lemenshow Goodness of Fit*, which gave us a *p-value* less than 0.05, thus telling us that our model was not a good fit at all.

```
Call:
glm(formula = SeriousDlqin2yrs ~ ., family = "binomial", data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.2948  -0.3916  -0.3146  -0.2537   4.9464

Coefficients:
                                       Estimate Std. Error z value Pr(>|z|)
(Intercept)                          -1.341e+00  4.599e-02 -29.158  < 2e-16 ***
X                                     3.082e-07  2.472e-07   1.247  0.21247
RevolvingUtilizationOfUnsecuredLines -3.758e-05  6.120e-05  -0.614  0.53918
age                                  -2.848e-02  8.332e-04 -34.179  < 2e-16 ***
NumberOfTime30.59DaysPastDueNotWorse  5.033e-01  1.111e-02  45.318  < 2e-16 ***
DebtRatio                            -2.001e-05  1.258e-05  -1.591  0.11154
MonthlyIncome                        -3.594e-05  3.161e-06 -11.367  < 2e-16 ***
NumberOfOpenCreditLinesAndLoans      -7.508e-03  2.543e-03  -2.953  0.00315 **
NumberOfTimes90DaysLate               4.683e-01  1.522e-02  30.773  < 2e-16 ***
NumberRealEstateLoansOrLines          6.748e-02  1.066e-02   6.330 2.45e-10 ***
NumberOfTime60.89DaysPastDueNotWorse -9.396e-01  1.772e-02 -53.028  < 2e-16 ***
NumberOfDependents                    9.759e-02  9.630e-03  10.134  < 2e-16 ***
MissingEntries                        1.371e-02  3.730e-02   0.368  0.71318
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 73616  on 149999  degrees of freedom
Residual deviance: 67517  on 149987  degrees of freedom
AIC: 67543

Number of Fisher Scoring iterations: 6
```

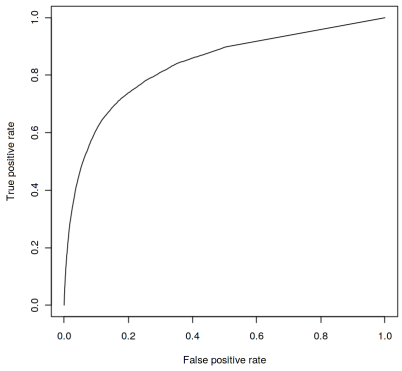
```
        Hosmer and Lemeshow goodness of fit (GOF) test

data:  train$SeriousDlqin2yrs, fitted(glm_model)
X-squared = 1350, df = 8, p-value < 2.2e-16
```

5 Random Forests

My implementation of random forests was straightforward. I decided to use the `tuneRF` method from the `randomForest` package. Namely, `tuneRF` finds the optimal value `mtry` (the number of variables we check during splits) which minimizes the OOB (Out-Of-Bag) error rate. My initial default m was $\sqrt{10}$ (since our data lives in $\mathbb{R}^{10}$). I grew 100 trees during the tuning step with a tolerance of .05. Below are the plots of the OOB error rate as a function of `mtry` using step factors of 1.5, 1.7, 2, and 3 respectively:
I found that using $m = 2$ allowed random forests to perform the best, yielding an OOB error rate



of 6.36 percent. Plotting the Receiver Operating Characteristic curve, I found an AUC of around .835:



6 Conclusion

Of the three methods, Random Forests was the learner that yielded the best results. It had fast convergence, which allowed for thorough tuning, and it did the best at predicting back `SeriousDlqin2yrs`

in the training set. We found that setting the random-forest parameter `mtry` to 2 yielded an out of bag error rate of 6.36 percent. Random forests also produced the best ROC of all our learners, attaining an AUC of .835. Future work to expand on this could involve modifying our random forest learner to minimize entropy instead of Gini impurity during successive splits.