Machine Learning Case - Predictive modeling

Jonathan Ridenour
May 14, 2017

Methodology

Given that we have both categorical and numerical predictors and that we would like to predict probabilities associated with a binary response, I have decided to use logistic regression.

In order to choose a suitable subset of the predictors, I used a hybrid method of forward stepwise selection, including only those variables which showed correlation with `default` in a box-plot or one-at-a-time regression, and adding at each step that predictor which contributed the greatest decrease in AIC to the model as a whole. This left me with 8 predictors: 4 numeric and 4 categorical.

To handle the missing data, I let `R` predict the missing values with K-nearest neighbours (at $K = 10$). I use the `knnImputation()` function from `DMwR` (warning: this takes some time).

To validate this model, I have split the data into training and test sets (80/20). Note that the training data has not been imputed by KNN, since this would add noise to the system. Any `NA`s in the training data are simply left out when the model is fit. This means that the true training size is only around 8000.

Having used the above-described model to predict probabilities of default on the test data, I looked at the associated confusion matrix to validate the method. If I predict default for $Pr(\texttt{default} = 1|\mathbf{X}) > 0.006$ (the mean of the predicted probabilities), I get:

	true status	
test prediction	0	1
0	16149	140
1	1598	109

Under these conditions, out of 249 customers who defaulted in the test data, the model successfully predicted 109, giving an error rate of 56%, while misclassifying 1598 out of 17747 total non-defaults (9%). By further decreasing the threshold to 0.003, the model achieves error rates of 45% and 20% respectively. The threshold should be set based on the costs of misclassification for default vs non-default.

I also checked to see if adding more predictors to the model would decrease the test error rates. Adding more features did not improve the test error, so I stayed with the original 8 predictors. Finally, I retrained the model using all the available data and predicted the unknown values of `default`. Since the validation of the model is somewhat ad-hoc (due to time constraints) I felt it was worth it to retrain with the full data, in hopes of gaining some predictive power.

Reflections

To me the probabilities for predicting default seem extremely low. To achieve a decent error rate, the threshold must be set to less that 1%. However, this reflects the relative infrequency of defaults in the data (also 1%). In terms of ideas for improvement: a more rigorous analysis of the effect of KNN imputation should be done. Given the time-constraints, I haven't looked into how the modelling of missing data effects the model performance. A different method may yield a better result. Also, a data-based feature selection should be attempted (e.g. LASSO). Also, combinations of features could improve performance (e.g. principal components). A k-fold cross-validation would provide a more robust validation. Finally, since the program is quite slow, it should be implemented in parallel (e.g. Spark)!