Konrad Rauscher

Introduction to Data Mining

Project 3 - Choice 2

4/27/16

Status:

Completed

Approx. Time Spent on Project:

60 hours

Things I wish I had been told prior to being given assignment:

NA

Design:

Sources: *Scikit-Learn*: scikit-learn.org, *Pandas, Collections, Matplotlib, Scipy*

Preprocessing:
1) Load train.csv as a panda data frame.
2) Replace missing values
    1) Replace missing categorical values with the mode of the relevant attribute.
    2) Replace missing numeric values with the mean of the relevant attribute
3) Convert categorical values to decimal encoding
4) Use OneHotEncoder to encode the categorical attributes that had been subjected to numerical encoding
5) Feature Selection via SelectPercentile (scikit-learn) —> The top 20% of attributes in terms of how informative an attribute is for determining class values are kept.
6) The data set is then normalized

Classification:
After extensive testing and parameter tweaking, the four best classifiers kept (as determined by LogLoss and AUC) were:
1) Bagging
2) NaiveBayes
3) RandomForest
4) Voting (comprised of an ensemble of 1 and 3)

Evaluation:
The above four classifiers are subjected to a stratified k-fold cross validation and for each fold the following evaluation metrics are calculated and output:
1) Classification report: per-class and average precision, recall, f1-score, and support
2) Confusion matrix
3) Log Loss
4) ROC curve that is graphed once all folds have been computed

Analysis:

Preprocessing:
Because the class distribution in the train set is not particularly skewed (~1:4), I did not feel the need to take steps that would balance the exposure of classes to classifiers. Attributes were not labeled with descriptive names (v1, v2, v3, etc.), which made being able to determine important features by eye or validate feature selection more difficult than would otherwise have been the case. Missing values were replaced based on their type: mean for numeric as the assumption was made that the ordinality of these numeric attributes was important. Given that the average number of unique values for numeric features was considerably higher then for categorical features (70 vs 20), this seems like a reasonable assumption as a feature expressing magnitude could be expected to have more unique features than a feature expressing categories, Therefore, an average of a given attribute was considered an appropriate strategy to replace missing numeric values. The mode of categorical features was chosen to fill their respective missing values because of similar reasoning.

Once missing values had been replaced, categorical features were subjected to decimal encoding so that the entire data set would be comprised of numeric attributes, which was required for the classifiers that I was expecting to use. Next, OneHotEncoder was used to encode categorical attributes that had been encoded numerically in order to remove the misleading (seemingly ordinal) distribution of numerically encoded categorical attributes (i.e. 0, 1, 2, 3, etc. could be encoded to 10, 15, 40, 5, 0, for example). This increases the dimensionality of the dataset from 131 to 435. After this, only the top 20% informative features are selected for future use in classifiers, bringing the number of features in the data set down to 87. This is done to boost performance by removing useless/uninformative features that would likely dilute the impact of useful features on class prediction. Finally, the dataset is normalized so as to mitigate the effects of outliers and weight all features equally in their representation within a classifier.

The method used for feature selection SelectPercentile(p = 20) was decided upon after comparisons of the subsequent performances of classifiers after subjecting the training set to different tunings of PCA, SelectFDR, and SelectPercentile, as well as my own feature selection method that removed features which had an information gain of 0.

Classification:
Numerous classifiers were implemented in order to find which approaches worked best (all from *Scikit-Learn)*: SVC, KNeighbors, GaussianNB, Dummy, Bagging, RandomForest, and Voting.

SVC (C-Support Vector Classification)
Implemented all kernel options: 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed', all of which classified almost all (except 1-3 examples) as class 1. I then tried different weighting schemes to compensate for this (increasing the cost of misclassification of class 0), but none of these tunings resulted in performance better than GaussianNB, Bagging, RandomForest, or Voting. Another issue with SVC was the complexity of the fit process (which is more than quadratic) resulted in extremely long fit times (over half an hour) for my large train sets (even with k cross-fold validation). Setting max iterations was tried, but again, classification performance was not outstanding. Further, because Support Vector Machines do not directly provide probability

estimates (which is necessary for LogLoss calculations), I decided to spend more time on other classifiers.

KNeighbors

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='distance')
             precision    recall  f1-score   support

          0       0.41      0.25      0.31      9119
          1       0.79      0.88      0.83     28988

avg / total       0.70      0.73      0.71     38107

[[ 2325  6794]
 [ 3360 25628]]
Area under ROC:  0.569526067583
Log Loss:  3.458347251
```

Weighting points by distance performed marginally better than a uniform weighing scheme, indicating that the distance of a point is somewhat informative in class prediction. the number of neighbors didn't make much of a difference in terms of performance (n = 3, 5, 7, 9 were tried). Nothing particularly interesting about the results from this classifier. There were a good number of other classifiers that performed a good deal better. A possible reason for this is the dimensionality of the pre-processed data set (83 attributes) makes distance calculations less effective than approaches involving linear separation.

GaussianNB

```
             precision    recall  f1-score   support

          0       0.31      0.68      0.42      9119
          1       0.84      0.52      0.64     28988

avg / total       0.71      0.56      0.59     38107

[[ 6190  2929]
 [13995 14993]]
Area under ROC:  0.598008259934
Log Loss:  15.3393495612
```

GaussianNB had the highest class 1 precision of all classifiers (.84), although a relatively low recall. However, because of the nature of the task (predicting accelerated claims), high precision for class 1 is very desirable. Therefore, I decided to keep this classifier for attempted use in ensemble methods, despite its relatively poor performance in terms of other performance metrics.

I did not use MultinomialNB or BernouliNB because they are more suitable for classification with discrete features.

Dummy

```
          precision    recall  f1-score   support

        0       0.24      0.23      0.24      9119
        1       0.76      0.76      0.76     28988

avg / total       0.63      0.64      0.64     38107

[[ 2129  6990]
 [ 6845 22143]]
Area under ROC:  0.498668194725
Log Loss:  12.53967934
```

Implemented this method as a baseline classifier to compare other results against a classifier that generates predictions by respecting the training set's class distribution. A prediction scheme that always predicts the most frequent label in the training set was another option, but I thought it best to use the stratified approach (prediction by class distribution) as it performed the best of the two.

Bagging

```
BaggingClassifier()
          precision    recall  f1-score   support

        0       0.45      0.32      0.37      9119
        1       0.80      0.88      0.84     28988

avg / total       0.72      0.74      0.73     38107

[[ 2888  6231]
 [ 3519 25469]]
Area under ROC:  0.597653166336
Log Loss:  1.1223643239
```

The good performance of Bagging with decision trees as a base classifier was somewhat expected due to the previously observed better performances of tree-based classifiers with this data set.

Interestingly, I found that a halving of the number of max features led to better results (.05 more AUC and .04 higher class 1 precision). Because this occurred with a decision tree as the base classifier, a reasonable interpretation is that this reduction in dimensionality lead to shorter trees and thus helped maintain generalization (avoiding over-fitting).

Using GaussianNB as a base classifier led to particularly similar results to my GaussianNB, which makes sense because GaussianNB is a stable classifier, meaning that relatively similar models are produced from each subset in the bagging process.

RandomForest

```
n_estimators=10
          precision    recall  f1-score   support

        0       0.44      0.33      0.38      9119
        1       0.80      0.87      0.84     28988

avg / total       0.72      0.74      0.73     38107

[[ 2990  6129]
 [ 3750 25238]]
Area under ROC:  0.599261477495
Log Loss:  8.95408908437
```

```
n_estimators=50
          precision    recall  f1-score   support

        0       0.55      0.23      0.32      9119
        1       0.80      0.94      0.86     28988

avg / total       0.74      0.77      0.73     38107

[[ 2084  7035]
 [ 1704 27284]]
Area under ROC:  0.584875442899
Log Loss:  1.07286603783
```

After 10, increasing the number of estimators did not have a meaningful affect on class 1 accuracy, but LogLoss and class 0 precision do continue to increase up to 50 estimators (although at the cost of AUC and class 0 recall), which seems to indicate that the ensemble of trees grown with introduced begin to converge quite strongly in terms of class 1 classification once the forest reaches a size of 10.

Voting

```
          precision    recall  f1-score   support

        0       0.47      0.33      0.39      9119
        1       0.81      0.88      0.84     28988

avg / total       0.73      0.75      0.74     38107

[[ 3040  6079]
 [ 3376 25612]]
Area under ROC:  0.60845395139
Log Loss:  0.67516328978
```

```
    clf1 = GaussianNB()
    clf2 = BaggingClassifier()
    clf3 = RandomForestClassifier(n_estimators=100)
```

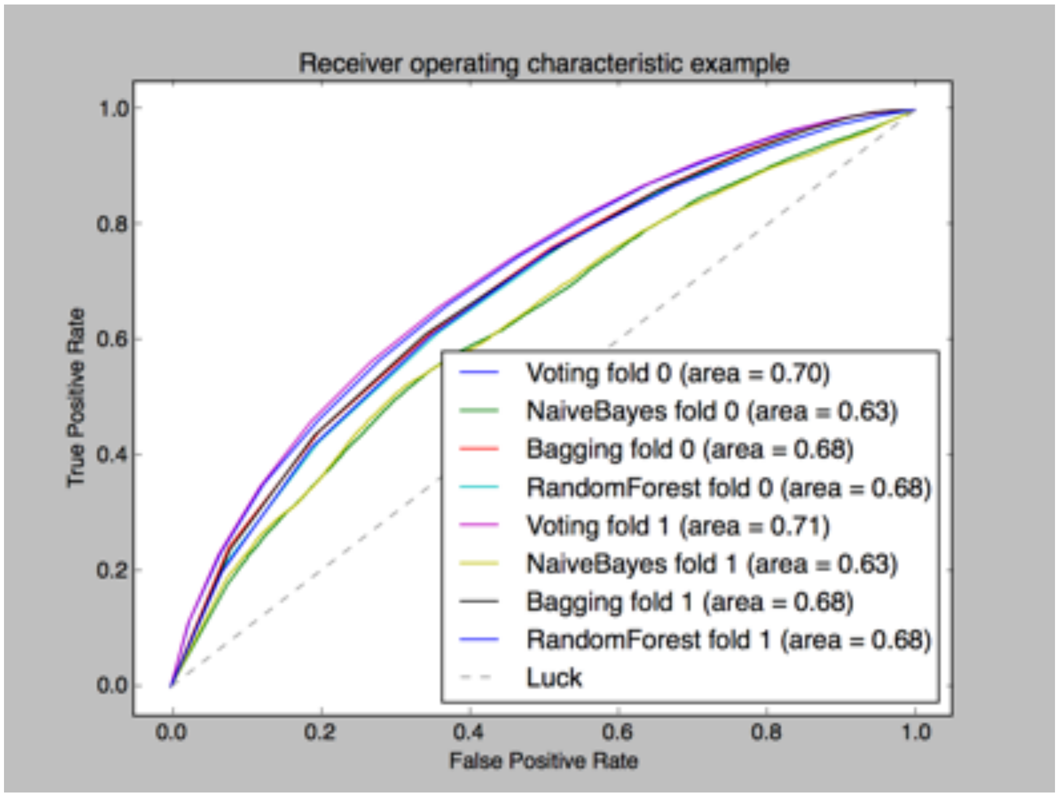
My best classifier based on class 1 precision and area under the ROC. My thought process in determining which and the number of classifiers to incorporate in this ensemble method was that I should choose my two best classifiers in terms of AUC, class 1 precision, and class 1 recall (RandomForest and Bagging) as well as a third classifier characterized by results involving the highest class 1 precision and a relatively low class 1 recall (GausianNB) that would resolve cases in which the first two alone would lead to ambiguous results.

However, simply the use of RandomForest and Bagging led to the best results (an improvement in LogLoss by over .5, which seems to indicate that GausianNB was merely diluting the better classification ability of the other two classifiers.

One concern here is that a voting scheming in which both Bagging and RandomForests are used is somewhat redundant (will predict similarly) as the versions that I used of both use decision trees as their base classifiers. Further, they are both averaging ensemble methods. However, seeing as how these are my two best classifiers (excluding voting), it made sense for me to combine them in an ensemble method, and the result was my best classifier.

Evaluation:
Although LogLoss (simply put, a measure of the uncertainty of the probabilities produced by a model) is the specified evaluation metric for this assignment, I found it worthwhile to use additional metrics. For example, because of the nature of this task (classification of accelerated claims), the classification accuracy for class 1 is a particularly important performance metric, as a false positive for a class 1 can be expected to have a much higher cost than a false negative.

	Precision: class 1	Recall: class 1	AUC	LogLoss	Macro f1
Voting fold0	0.82	0.78	0.69	0.683476203501	0.71
NaiveBayes fold0	**0.84**	0.52	0.63	2.76050875081	0.59
Bagging fold0	0.81	0.86	0.68	1.1223643239	**0.73**
RandomForest fold0	0.80	**0.87**	0.67	1.07286603783	0.72
Dummy fold0	0.76	0.76	0.5	12.4931988723	0.64
Voting fold1	0.83	0.78	**0.70**	**0.67516328978**	0.72
NaiveBayes fold1	**0.84**	0.51	0.63	2.78514145086	0.58
Bagging fold1	0.81	0.86	0.68	1.14342137335	0.73
RandomForest fold1	0.81	**0.87**	0.68	1.05109589847	**0.73**
Dummy fold1	0.76	0.76	0.5	12.4825409861	0.64



An observation pertinent across all classifiers was the poorer classification of class 0 relative to class 1. A possible explanation is that while the train set has only 2 classes, the test has 11. Although I only used the train set (with k-fold cross validation), the class 0's are seemingly actually comprised of 10 different type of claims, which could make classification of class 0 more difficult based on how different the characteristics (important features, the manner in which feature values are important, etc.) of the 10 classes are from each other.

In terms of my how my best LogLoss compare to others on Kagggle, the 0.67516328978 from soft Voting with randomForest and Bagging puts me at 2764th place.

Going forward, I am confident that I could improve my results by improving my voting scheme with the addition of different classifiers and the implementation of a tuned weighted voting scheme.