Machine Learning & Data Mining
Miniproject 1

Caltech CS/CNS/EE 155
Released January 28th, 2017

Introduction

- **Group members**
  Steven Brotz, Andrew Wang, and David LeBaron
- **Team name**
  Double Boost
- **Division of labour**
  Each group member experimented with different models to see which model would work best. Once one model was identified to work the best, all members focused on that one model, observing its results and suggesting parameter adjustments.

  Andrew began by training decision trees with bagging, and after that he implemented a random forests model.

  David handled the data preprocessing, and worked on decision trees trained with the AdaBoost algorithm.

  Steven first trained a neural network, and then fit a model with logistic regression. He is also the primary author of this report.

Overview

- **Models and techniques tried**
  - **AdaBoost:** We tried sklearn.ensemble.AdaBoostClassifier.
  - **Neural Network:** We tried the Keras Sequential model with tensorflow backend.
  - **Bagging Decision Trees:** We tried sklearn.ensemble.BaggingClassifier
  - **Random forests:** We tried sklearn.ensemble.RandomForestClassifier.
  - **Logistic Regression:** We tried sklearn.linear_model.LogisticRegression.

  The first three models listed were all implemented at the same time, one by each group member. We then trained a random forests model, and proceeded with that model because it showed the highest validation accuracy. Logistic regression was implemented after we had already thoroughly tweaked the random forests model.
- **Work timeline**
  - **Week 1:** We spent the first week deciding how to prune the dataset. In specific, we wanted to eliminate some of the input parameters because there were so many. Additionally, we considered what approaches would work best given the nature of the problem i.e. size of input vector, number of training examples, etc.
  - **Week 2:** We implemented all of the models listed above during the second week. We conducted testing of the original three models (AdaBoost, neural networks, bagging), and achieved the best results with the bagging model and the boosting model. While the random forests model

was implemented, more adjustments and improvements were made to the AdaBoost model. Eventually the highest accuracy was achieved with the random forests approach, and so that was the model we spent the most time configuring.

Approach

- **Data processing and manipulation**
  - **Eliminating parameters:** We noticed that the dataset had more than 300 input parameters, and figured that probably many of them have very low correlation to the output. We pruned the dataset first by removing all columns that mainly consist of blanks. In addition to that, we removed all columns that consist of only one value, because any such parameter cannot be learned from because it does not vary at all with the output. Through both of these modifications, we removed 169 of the original 328 parameters, keeping only 159.

    Of course, all the same parameters that were removed from the training set were also removed from the test set so that the model trained on the training set could predict on the test set. We wrote a program to process the data in this way that loads 'train_2008.csv' and 'test_2008.csv' as Pandas DataFrames, prunes the datasets as indicated above, and creates two new csv files, which were given to models we trained.
  - **One-hot vector encoding:** As explained in lecture, it is important to one-hot vector encode the output if the output is over a set of real numbers but is meant to be categorical. This was achieved using keras.utils.np_utils.to_categorical
  - **Training/Validation Split:** Because there were so many training examples (almost 64000), cross-validation was not deemed absolutely necessary. However, we still employed it in one of our models, the boosted decision tree. When cross-validation was not used, the input data was randomly divided according to an 80/20 split. These training/validation splits were achieved using either of two functions from the sklearn.model_selection module: test_train_split or KFold.
- **Details of models and techniques**
  - **AdaBoost:** We thought we should start with AdaBoost with decision trees because it is a generally a good classifier in practice, as described in lecture. The parameters we varied for this model included the number of base estimators, the learning rate, and the learning algorithm.
  - **Neural Network:** Before we had implemented any learning models, we decided that one of us should try training a neural network on the data. We used the same libraries as were used on homework 4, the Keras Sequential Model with Tensorflow backend. We tested many variations on the activation type, regularization, number of layers, number of nodes, number of epochs, and optimizer. However, the best we could do with any combination of those parameters that we tested was getting the model to guess every output to be 1. The ensemble methods showed more promising results, and so we did not try to do better with neural networks and focused on those instead.

- **Bagging:** We used the bagging method for decision trees before we implemented the random forests model. We varied parameters such as number of base estimators, percentage of samples to use, and percentage of features to use. The promising results of this model in addition to those of the boosted models led us to try random forests, as it is essentially a combination of the two methods.
- **Random Forests:** The idea to use random forests was prompted by the success of both the boosting and bagging models. For this model, we varied max depth, minimum samples per leaf node, number of trees in forest, and percentage of features to use.
- **Logistic Regression:** After we had mostly settled on our decision to use random forests as our final model, we decided to try logistic regression, mostly out of curiosity to see how well it would do in relation to the other methods. Using the 'newton-cg' solver with a maximum of 200 iterations, we achieved an accuracy of 0.767537161966 on the validation set. Upping the maximum iterations increased the accuracy only marginally.

Model Selection

- **Scoring**
  For all scoring, we used the default scoring functions provided by the respective library of each model, which were all measures of the accuracy. We mostly cared about validation scores, and did not pay much attention to training scores. We achieved the lowest validation score, 0.742307097572, with the neural network, which guessed every output to be 1. The highest validation score we achieved was 0.785758466, which was given by our random forests model. After we achieved this score, we used cross validation with the same parameters for the model, and achieved an average accuracy with 5-fold cross validation of 0.778140343.

- **Validation and Test**
  Because there were so many samples in the training set, we reasoned that cross validation was not always absolutely necessary, as we could just split the data into a training and a validation set while still maintaining large numbers of samples in each. We generally divided the input data into the two sets with an 80/20 split, with 80% of the data designated for training. However, we still did use cross-validation in some cases.

  Score on the validation set was the main factor that led us in choosing parameters for any given model. Namely, for each parameter we wished to adjust (from the library default), we held all other parameters constant, and looped over values for that parameter over some reasonable range, and selected whichever gave the highest validation accuracy.

  In our bagging model, we noticed that including more than roughly 45 base estimators provided inconsistent, and at best marginal, increases in accuracies. Here we see this in the following graphs

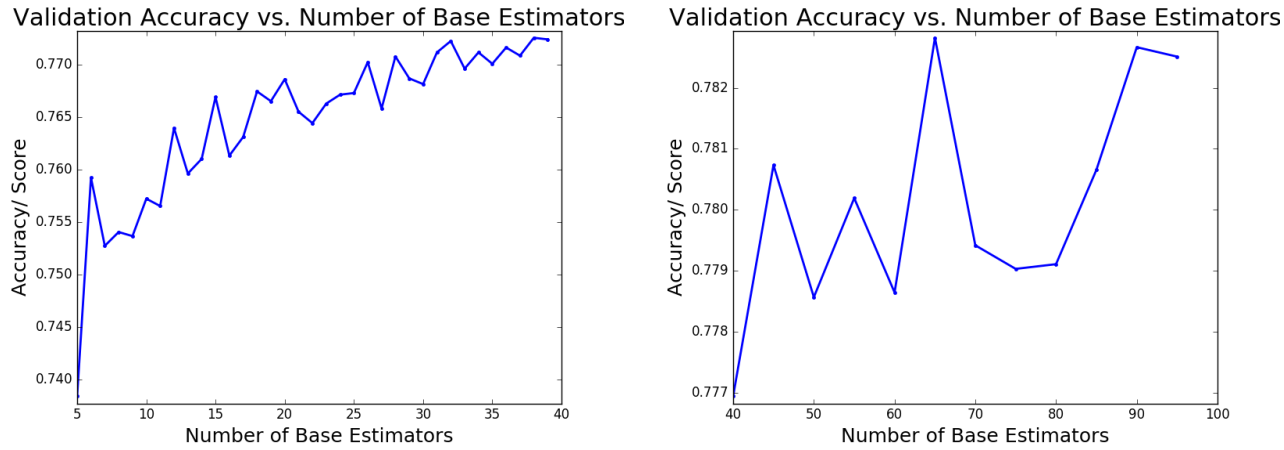

Figure 1: In the bagging model, we see that after roughly 45 estimators, accuracy does increase as much as it did for smaller numbers of estimators. Note the different y-scales.

We see a similar trend in the validation accuracy vs. number of estimators for the random forests model, however the plateau occurs around roughly 80 estimators instead of 45.

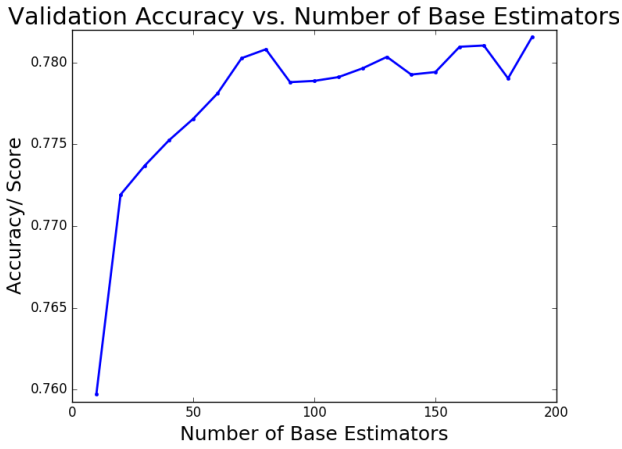

Figure 2: In the random forests model, we see that after roughly 80 estimators, the accuracy increase is small and inconsistent.

This phenomenon occurs at a larger number of estimators in the random forests model than it does in the bagging model. From this we may infer that random forests require more estimators than bagging methods to achieve their maximum accuracy.

We varied several other parameters of the bagging model, such as fraction of features and samples to train on. We see their respective effects on validation accuracy in the following graphs

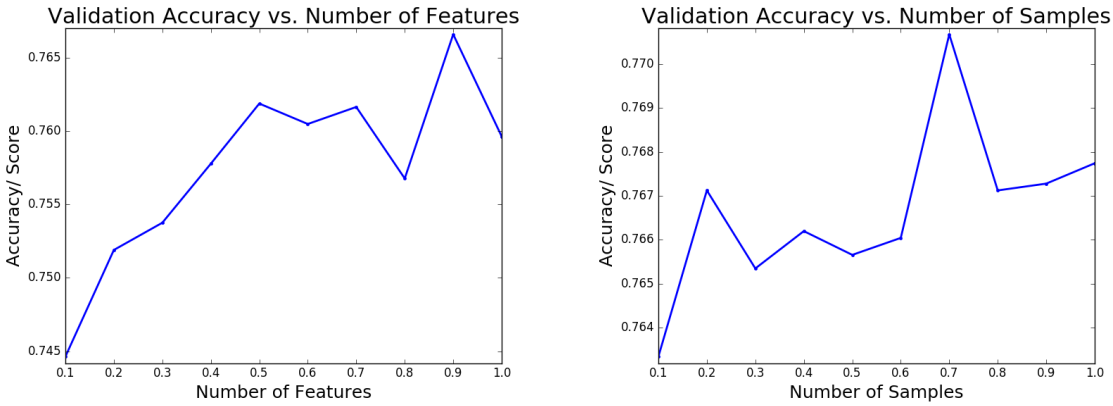

Figure 3: In the bagging model, we see the trends in the validation accuracy produced by varying (all else equal) either the fraction of features or fraction of samples to train on.

While there are notable peaks and valleys, we may observe that validation accuracy generally increases with the number of samples and features included. Conversely, with the random forests model, we do not see a positive correlation between number of features and validation accuracy. In fact, it appears to be weakly negative. This is evident in the following graph.

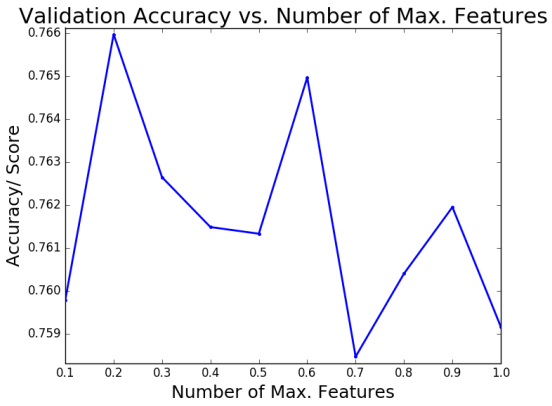

Figure 4: In the random forests, model, as fraction of features to train on increases, the validation accuracy slowly declines.

The validation accuracy in the random forests model experiences sharp peaks and valleys over a varying fraction of included features. Nonetheless, the validation accuracy appears to weakly trend downward.

Some of the other parameters we tweaked in the random forests model were maximum depth and minimum samples per leaf node. The graphs below illustrate their respective effects

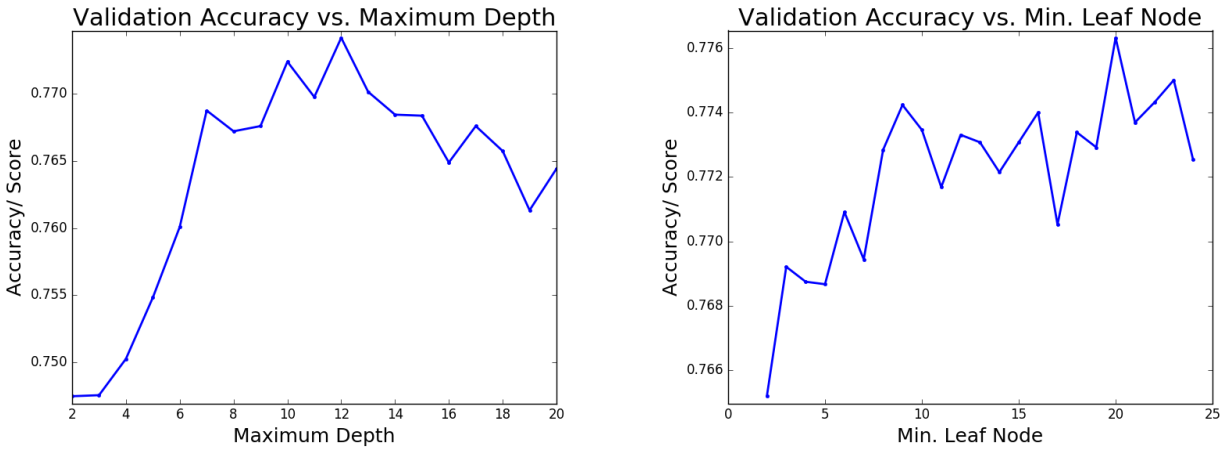

Figure 5: Here we see the effects of various forms of regularization for the random forests model.

As these are both forms of regularization, we expect to see a somewhat concave trend for the validation accuracy as we move from overfitting, to a proper range, and then eventually to underfitting, where the accuracy will drop once again. We do observe this in the maximum depth graph, noting that after a maximum depth of roughly 12, the model begins to underfit. However, in the second graph, it is unclear whether the accuracy is actually beginning to decline, or if it just another valley, which is clearly a common feature of the relationship between accuracy and minimum samples per leaf node. To determine this, testing validation accuracies against higher values for the minimum samples per leaf node would be necessary.

Here is a table of some of the highest accuracies we achieved, along with their parameters

Fraction of Features	Maximum Depth	Validation Accuracy
0.1	10	0.777253720
0.1	12	0.778800062
0.1	13	0.780114427
0.1	14	0.781196846
0.1	15	0.781815370
0.225	11	0.782124633
0.225	13	0.782433895
0.35	15	0.782588526
0.35	16	0.782665842
0.475	10	0.783748260
0.475	15	0.785758466

Table 1: These are some of the highest accuracies we achieved with the random forests model. Note: minimum number of samples per leaf and number of estimators were kept constant at 13 and 70, respectively.

Conclusion

- **Discoveries**
  We learned many practical things while doing this project. We learned that it is very important to preprocess our dataset. If we had trained with all those input parameters we had deemed irrelevant, our data would have been noisier and our models would likely be less accurate. In regards to models, we learned that neural networks, at least in all configurations we tried, are not particularly well suited for problems of this nature. Additionally, while better than simply choosing the most common classification in the training (as the neural network did), logistic regression is not the best model for this multi-class problem. Decision trees, particularly when used in a random forests algorithm, are effective classifiers for a problem with this many samples, input parameters, and output classes.

- **Challenges**
  The main challenges were in finding the ideal parameters for the random forests model. Specifically, each trial run took at least several minutes, and due to the rather large number of combinations of parameters, we had to wait long periods of time before we could consider the results and make progress. Additionally, there was initially some unfamiliarity with the libraries we chose to use for preprocessing the data.

- **Concluding Remarks**
  While we are happy with the model we were able to produce, we have some thoughts for potential improvements. Namely, we would further prune the input dataset to remove even more parameters, perhaps by calculating correlation coefficients to the output separately for each parameter, and removing those which do not meet some chosen cutoff correlation. Additionally, we might consider using a voting classification method, in which we would take several of our best models, and by majority rule among the predictions of each of those models, choose the final prediction.