Peter Fulweiler, Andres Cojuangco
Prof. Manfredi
Machine Learning: Final Project Report
4/11/22

Introduction

The main goal of the project was to compare and contrast the effectiveness of an ensemble of decision trees versus an ensemble of perceptrons for a linearly separable data set and a non-linearly separable data set. The algorithm for the ensemble of decision trees is called the Random Forest algorithm and the algorithm for the ensemble of perceptrons is called the Random Perceptron Generator. Both algorithms take algorithms we learned in class, such as the ID3 algorithm and the basic perceptron algorithm, and turn them into more complex algorithms that are, in theory, more resistant to overfitting and have different ways to reduce bias. We know that decision trees are more expressive than perceptrons since they can handle non-linearly separable data. So, we thought it would be interesting to see if they could outperform the perceptrons even with linearly separable data.

We thought this project would be challenging and important since part of the job of machine learning scientists is to test different models and find new ways to get the best results. It feels very rewarding to know that we have created our own unique model and through this process we hoped to learn if combining different models can create stronger, more efficient models. This is extremely important because it shows an algorithmic approach to creating stronger and better models.

About the Data Sets

The sonar data set had sixty attributes where their rows of integer values corresponded to sonar signals that were obtained through varying angles. These signals were trying to find out if the objects around the sonar source were rocks or mines. These made up the class column. This data had around 500 examples. After running tests with a single perceptron, the results revealed that the weights were not converging and reaching the global minima. Thus, this data set was identified as non-linearly separable.

The banknotes data set is used to authenticate banknotes based on photos. It had four integer features: variance, skewness, curtosis, and entropy. Its class label values were 0 for real banknotes and 1 for fake ones. This data set had around 1300 examples. Using the perceptron, it was determined that this data set was linearly separable since a perceptron was able to converge and find the global minima.

Structure of the Algorithms

The ID3 Random Forest Algorithm and the Random Perceptron Generator are supervised learning models that make binary classifications. The general idea for their structure is that they use an ensemble technique called "bagging". This technique takes random samples from the train data to create smaller datasets and for each data set, a classifier is trained. Then, these trained classifiers make predictions for each example and the majority prediction of the models (i.e. the mode of the predictions for that example) becomes the final prediction for the example.

Our algorithm's structure consists of three components. The first component was bootstrapping the data set. Bootstrapping is the process of getting random samples from the data set with replacement to create smaller dataframes. The number of bootstrapped data and its number of rows depends on how many classifiers the user chooses and the performance of the model on the data set. A large number of classifiers can bring stability from using the majority prediction. The second component was getting random subspaces of the features to train the predictor models. In other words, each predictor only uses a reduced number of features that are chosen at random. This reduces the expressiveness of the classifiers which makes the algorithms less prone to overfitting. The randomness of the features also increase the diversity of the models, making each classifier almost independent of each other. For the ID3 Random Forest Algorithm, only decision trees were generated using the random features. For the Random Perceptron Generator, only perceptrons were trained. Lastly, the third component was getting the majority vote of the classifiers to make a prediction for an example.

ID3 Algorithm Random Forest Algorithm Details

For the ID3 Random Forest Algorithm, we first created a function that generates a list of trees from bootstrapped data. Each tree was created using the ID3_decision_tree function. This function uses information gain and entropy similar to the one in class. However, the function was modified to take a random subset of the features. Thus, it creates a diversity of features in the forest. Then, we had to create a function that gets a prediction for one example using a tree so we can eventually get predictions for all examples. In our homework, we never actually got the predictions themselves. So, we created a function called ID3_decision_tree_prediction and get_random_forest_predictions. ID3_decision_tree_prediction takes in the example and uses its values to recursively go down the tree that was generated in order to get its prediction. The function get_random_forest_predictions creates a dataframe where the columns are the names of the classifiers and the rows are their predictions for each example. This function then takes the mode of the rows and returns a single column that has the final predictions of the ID3 Random Forest Algorithm for each example.

Training and Testing ID3 Random Forest Algorithm

For both data sets, the difficulty was tuning the hyperparameters due to complexity issues. The way our code was set up, using lists and generating trees, would make the run time extremely long. Thus, we had to make some assumptions about the hyperparameters.

The hyperparameters for the ID3 Random Forest Algorithm are the following: maximum depth of the trees, number of classifiers, number of examples to bootstrap, and number of random features to sample. First, we set the number of classifiers to fifty. Because random forests use majority vote, a large number of fifty classifiers would likely bring stability in testing. However, not being able to use more than fifty trees rules out the possibility of reaching higher accuracies for the test set. Then, for the number of features to sample, we used the log(number of features + 1) for the sonar data and the square root of the number of features for the banknotes data. These numbers are often used for random forests. Both methods were tested for each data set and we used the method that yielded the best results. By reducing the number of features, we restrict the hypothesis space to avoid overfitting. Although, this assumption does not guarantee that it is the optimal number of features without tuning. Next, the number of examples to bootstrap were based on how much variance occurred. For the banknotes, we had to increase the number of bootstrapped data due to overfitting. So, its training split was set to 75/25 for train and test, while the sonar data had a 70/30 split. Finally, we used a get_hyperparameters function to test different depths. This function checked depths up to the number of features for each classifier and returned the depth that yielded the highest accuracy. This depth was used to calculate the average accuracies of the model on train and test data. In addition, it was also used to compare the train and test accuracies. The instructions on how to test and train for each dataset are described in the README.md file.

Random Perceptron Generator

In a broader sense, our Random perceptron Generator is a collection of sub-perceptron models that “vote” on predictions. Our Random Forest Generator comprises 3 major components. The first component is the bootstrapping of the dataset, the second component is creating a random subspace of features for the sub-decision tree, and the third component is having each sub-decision tree vote on the final prediction. First, in our bootstrapping of the dataset for each submodel essentially what you are doing is picking random examples of the dataframe to train that specific model. So that way, each submodel is trained on a slightly different subset of the dataset and this way randomness is introduced. For our bootstrap algorithm we decided to bootstrap using half the dataset. This way, each submodel would be trained on a random sample of the dataset half its size. We did this to increase randomness while still staying large enough to have enough examples to effectively train each submodel. Our random subspace algorithm involved training each submodel with a subset of features, this would also create randomness and would potentially weed out unhelpful features. Since we weren’t sure what the best number of features to use, in our get_hyperparameters function we tested a series of different numbers of features to try and find the best one. Finally, the last

portion of our Random Perceptron Generator was getting each tree to vote on a prediction. This was relatively straightforward as for each sub-perceptron, we would calculate predictions for the entire dataset and then transpose the sub-perceptrons and find the mode of each row. This was essentially finding the most common "vote" for each example.

Training and Testing Random Perceptron Generator

Once we had a functional Random Perceptron Generator, we then had to tune its hyper parameters. For our Random Perceptron Generator we used a combination of strategies learned in class such as cross validation and new methods using randomness. First, we decided straight off the bat to keep the learning_rate at 1. We had assumed this in the perceptron HW and we had many other hyperparameters to tune and we worried that the run time would be impacted by testing different learning rates. Additionally, as mentioned above we also assumed the number of examples to bootstrap as half the dataset. First, to get the best number of iterations, we simply just ran cross-validation on a single perceptron to get the best number of iterations. We figured if the best number of iterations worked for a single perceptron on the same dataset we figured they would work for a perceptron forest. Once we had our best number of iterations, we then used that to train different perceptron forests with different numbers of models and features using a double for loop. This was different from the Random Forest where we simply assumed the random subspace. This was our algorithmic approach: 1. Get the best number of iterations via cross-validation 2. Get the best number models and random features via bootstrapping and comparing accuracies of the model with those models and random features. Next, once we had the best number of iterations, models to create, and features to use in the random subspace, we ran our tuned perceptron on the testing dataset and got the accuracies and this is how we tested and trained for each dataset. The instructions on how to test and train for each dataset are described in the README.md file.

Results and Discussion

For the ID3 Random Forest algorithm, we had to put the values of the data set into bins. Bin strategy 1 binned values higher than the column mean as "higher" and values lower as "lower". On the other hand, bin strategy 2 did the same but with the column median instead of the mean.

For the sonar data set, bin strategy 1 had a higher average test accuracy of 73.87%. It also did better than the cross-validated ID3 algorithm in class whose highest accuracy was 70%. However, the results reveal some overfitting across all models since most yielded a 18 to 20% difference in accuracy, where training accuracies were around 90%. The source of the overfitting may be caused by our assumptions. We were not able to explore how different numbers of features would affect the test prediction nor were we able to explore larger numbers of

classifiers. This may have also been a result of the binning strategy since it groups specific values by range.

For the Random Perceptron Generator, the average test accuracy after finding the best hyper parameters was 71.29%. Its training accuracy was 91.09%, showing more signs of overfitting for both models. This is most likely because when your training accuracy is much better than the testing it means you are overfitting to the training data and not generalizing for all possible data. In addition, the cross validated single perceptron outperformed our model with an average test accuracy of 75.8%. This is disheartening because it shows that the perceptron forest does not produce better results than a single perceptron with cross-validation and in fact the perceptron forest is much less efficient than a single perceptron.

We thought since ID3 was more expressive, the Random Forest Algorithm would work better, especially since this dataset is non-linearly separable. But overall, the two models had similar results, with the ID3 Random Forest Algorithm performing slightly better. This may be due to the fact that all the models had some overfitting issues for this data set. Thus, with better hyper parameter tuning and binning, the ID3 Random Forest Algorithm could be effective.

The results for the banknotes were more promising in terms of avoiding overfitting. For both algorithms, their models were generally scored accurately on train and test that were not too far apart. For the ID3 Random Forest Algorithm, bin strategy 2 had the best performance on the testing data with an average accuracy of 81.62%. Although, the ID3 algorithm done in class performed better than the random forest with an accuracy score of 84.22%.

Compared to the random forest, the Random Perceptron Generator scored much higher with an average test score of 98.15%. Still, a single perceptron scored 1% higher than the ensemble technique for the test score.

In summary, it appears that using a perceptron is the best model for this data set. The scores of both perceptron models were similar but with more effective hyperparameter tuning, the Random Perceptron Generator could do better. One could even argue that neural networks may be even more suitable for this type of data set given the effectiveness of one perceptron.

Further Research:

There were many issues we encountered throughout the course of this project. One of the biggest issues that we ran into dictated our work. It was that perceptron only works on numerical values and ID3 only works on ordinal/nominal data. Therefore, we had to bin the integer values for them to work for decision trees. However, we were only able to explore two methods of

binning. Therefore, one way to take this exploration further would be to add more bins and see how the results change.

Another issue that we encountered was the complexity of our programming. For functions like bootstrapping and making ID3 decision trees with recursion, we had naive implementations and used lists. This made runtimes extremely long making it infeasible to code functions for hyperparameter tuning. Thus another path for exploration would be to modify parts of the code to run parallely to improve complexity, making it easier to experiment with different values for hyperparameters.

Lastly, a better way to go about this project was looking into the overfitting issues using charts and graphs. Further research can include calculating recall and the precision. This would give us a better understanding of the data and how we can make the appropriate tweaks to the model to get better results and generalize well.

In summary, this project was a great eye-opener to the challenges of running, comparing, and developing new models. It also gave us great insight to the benefits and hardships of the collaborative aspect in coding.

Photos of Results

ID3 Random Forest Algorithm (Sonar Data)

- Bin Strategy 1 (mean)

```
Average Train Accuracy: 0.9227586206896552
Average Test Accuracy 0.7387096774193548
```

- Bin Strategy 2 (mode)

```
Average Train Accuracy: 0.9410958904109589
Average Test Accuracy 0.7258064516129032
```

ID3 Algorithm from Class (Sonar Data)

- Bin Strategy 1

```
Train Accuracy for sonar_bin1: 0.8551724137931035
Test Accuracy for sonar_bin10.6774193548387096
```

- Bin Strategy 2

```
Train Accuracy for sonar_bin1: 0.9041095890410958
Test Accuracy for sonar_bin10.7096774193548387
```

ID3 Random Forest Algorithm (Banknotes Data)

- Bin Strategy 1 (mean)

```
Average Train Accuracy: 0.8188541666666665
Average Test Accuracy 0.8036407766990292
```

- Bin Strategy 2 (mode)

```
Average Train Accuracy: 0.846875
Average Test Accuracy 0.81626213592233
```

ID3 Algorithm from Class (Banknotes Data)

- Bin Strategy 1

```
Train Accuracy for bank_bin1: 0.846875
Test Accuracy for bank_bin10.8422330097087378
```

- Bin Strategy 2

```
Train Accuracy for bank_bin2: 0.846875
Test Accuracy for bank_bin20.8422330097087378
```

Random Perceptron Generator and Single Perceptron (Sonar Data)

```
Average Training Accuracy for one perceptron:  0.8835616438356164
Average Testing Accuracy for one perceptron:  0.7580645161290323
```

```
Average Train of for random perceptron generator with best hyperparameters:  0.9109589041095891
Average Test of for random perceptron generator with best hyperparameters:  0.7129032258064516
Average number of Models:  1
Average number of features:  50
Average best num of iterations:  1000
```

Random Perceptron Generator and Single Perceptron (Banknotes Data)

```
############# FINAL HYPERPARAMETERS #################
Average Training Accuracy for one perceptron:  0.9958333333333333
Average Testing Accuracy for one perceptron:  0.9902912621359224
Average Train of for random perceptron generator with best hyperparameters:  0.9933333333333334
Average Test of for random perceptron generator with best hyperparameters:  0.9815533980582524
Average number of Models:  2
Average number of features:  5
Average best num of iterations:  1000
```