**11676 – Big Data Analytics**

**lneves**

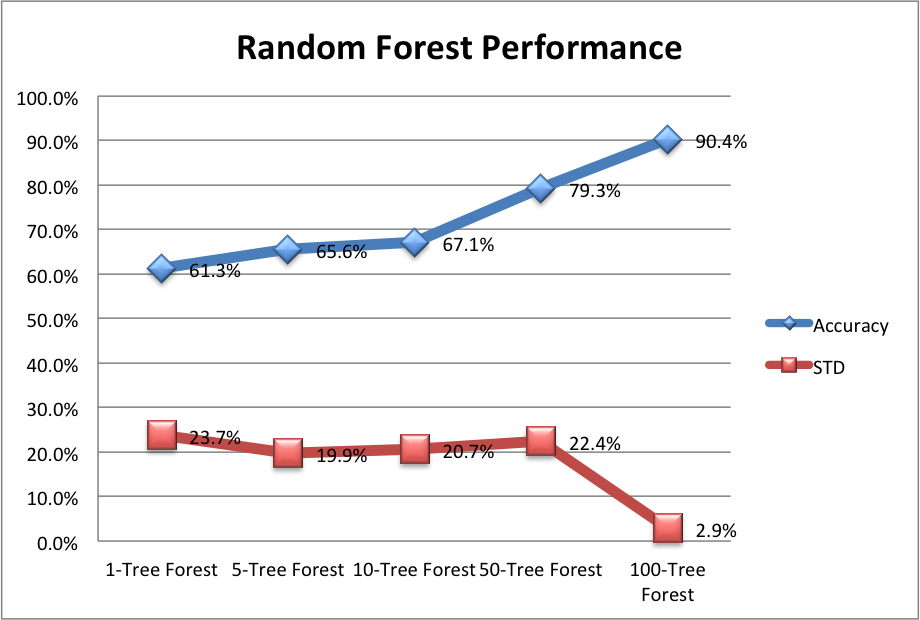
**HW 3 – Random Forest**

For homework 3, I have enhanced my python code from the data preparation part. For HW2, I have used the linear interpolation available on Panda, but the result of my random forest was getting really bad when choosing the “Artificial” features, thus, creating a really high standard deviation of accuracy even on large forests.

I have them implemented the idea that the professor gave in class. I have used the existing features as input for a linear regression model and tried to create the new features as a regression task. Each iteration, I would generate the values for the new feature and add them as features for the next iteration. You can check the code on my github repo under python\_code.

After having a more reliable value for my features, I have tuned my random forest implementation. As I had already built the decision tree structure, I have just created a way to split data randomly into 2/3 and 1/3 clusters and, from 1 to N, I have added a decision tree to the forest, training it on a random 2/3 set of the whole data and using only sqrt(F) features, being F the number of features. After training the current tree, I would test the whole random forest on the remaining 1/3 of this current split and store the error. I would train my forest on a subset of the total data containing 92.076 instances and, after my forest was built, test it on a 23.088 instances different test set. For deciding the label, I would make the trees vote and get the label voted by the majority.

I have also used Google’s GSON library to persist and load the model into a JSON format.

For my accuracy studies, I have first run the calculation 10 times on forests with 1, 5, 10, 50 and 100 trees and computed the average and standard deviation. I did that so that I could compare the accuracy with my previous approach that was only one tree with accuracy 76%.

My results were as follows:

|  |  |  |
| --- | --- | --- |
|  | Accuracy(Mean) | STD |
| 1-Tree Forest | 61.3% | 23.7% |
| 5-Tree Forest | 65.6% | 19.9% |
| 10-Tree Forest | 67.1% | 20.7% |
| 50-Tree Forest | 79.3% | 22.4% |
| 100-Tree Forest | 90.4% | 2.9% |

This result is as expected. Having less features and less data would make my accuracy worst when having only one tree, but as we grow the number of trees, the accuracy gets much higher and the standard deviation, smaller.

As of code organization, I have created a random forest class that would have an array list of decision trees. The rest of the code is pretty similar, since I had a decision tree and a decision tree node class that I am calling multiple times now. I have just fixed a bug on one of the files. I have also created a TreeSerializer class that uses GSON to read and persist the model. One upgrade on my code is that now I delete the instances after the code was created, so I need less memory to run my code, thus being able to run it on larger data and having a smaller JSON value.

After the class on Tuesday, October 6th, I have changed my accuracy computation to the one set on the handout.

This image shows that, on a 200 trees forest, my error was of 6%. It seems to have stabilized after 45 trees, with error floating around 6-7%. Also, when trying to predict the values on the separate test set, the model got 89% accuracy, just as the first experiment showed.