Naive Bayes, KNN, and Random Forest Comparison

Luke Fahmy
COGS 118A
A13849289

Abstract

In this paper, we conduct an empirical study on the performance of three supervised learning methods: Naive Bayes (NB), K Nearest Neighbors (KNN), and Random Forests (RF). We then compare our results to the findings of the well known Caruana and NiculescuMizil paper. We find that our results are consistent with the paper, with RF performing the best, while KNN is slightly behind, and NB cannot keep up in performance. We also experimented with balanced versus unbalanced datasets, but reached no conclusion on that front.

1 Introduction

In the last few years, the popularity of machine learning has continued to grow. This is in part due to the advent of GPUs, as well as the increased focus on building easy to use libraries. Combining these two factors with the advent of the personal computer era, we see that we have been perfectly placed in the most opportune time for data mining. Indeed, machine learning is now even done on laptops without dedicated graphics cards. This paper seeks to empirically analyze three algorithms that can be done on laptops without requiring long hours of training. For this reason, some of the more time-consuming methods have been omitted (like SVMs).

2 Methodology

We used the sci-kit learn implementation of all methods listed below, and GridSearchCV for all cross-validation.

2.1 Learning Algorithms

Naive Bayes: For Naive Bayes, we used the BernoulliNB, because we needed to be able to accept negative values. We ranged our values of alpha from 1e-7 to 1e3.

KNN: For KNN, we followed the Caruana and NiculescuMizil implementation very closely. We used 20 values of K, range from 1 to the size of the training set. However the first 10 values were from 1-10, then the rest of them scaled over the size of the set. This is because having 20 values scale over the entire set versus 10 values is not that much more advantageous. But having the values 1-10 is very important because KNN can usually be satisfied with a small K. For the distance we use the normal Euclidean.

Random Forest: For RF, we also followed the Caruana and NiculescuMizil implementation very closely. We had 1024 trees always, and the size of the feature set considered at each split is 1,2,4,6,8,12,16 or 20, depending on how many features are available.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

2.2 Performance Metrics

The metrics assessed are all the usual accuracy scores(testing accuracy, validation accuracy, training accuracy). Testing accuracy is the most important. We will look at it across different inputs. The inputs will be which dataset is used, which learning algorithm is used, and whether or not the set is balanced or unbalanced.

2.3 Datasets

We compared all of our algorithms on binary classification problems. If a dataset was not initially binary, we made it binary. The three datasets were: MAGIC Gamma Telescope Dataset (MAG), Abalone Dataset (ABA), and Letter Recognition Dataset (LET). MAG was already binary, since we only had to distinguish between gamma and hadron. ABA had three classes: male, female and infant. There were no even ways to split it, so we made an unbalanced and a balanced dataset with ABA. The unbalanced version was ABA the way it can be downloaded. The balanced was ABA without the males, since there were a similar number of females and infants. LET has 26 classes, for every letter in the alphabet. The balanced version of let was to make one class A-M, and the other class N-Z. The unbalanced version was to choose one letter as a class, and make every other letter the other class. All datasets were taken from the UCI Machine Learning Repository.

3 Experiment

First we will just show some of the raw data.

3.1 Testing Accuracy by Dataset

Average Accuracy on Magic: 0.766561111111
Average Accuracy on AbaloneB: 0.700974842767
Average Accuracy on AbaloneU: 0.625812263578
Average Accuracy on LetterB: 0.791661111111
Average Accuracy on LetterU: 0.982911111111

3.2 Testing Accuracy by Algorithm

Average Accuracy on Naive Bayes: 0.653513399066
Average Accuracy on KNN: 0.825848061308
Average Accuracy on Random Forest: 0.841390803433

3.3 Training Accuracy by Dataset

Average Training Accuracy on Magic: 0.882911115801
Average Training Accuracy on AbaloneB: 0.835920146025
Average Training Accuracy on AbaloneU: 0.79781436074
Average Training Accuracy on LetterB: 0.854243290385
Average Training Accuracy on LetterU: 0.988161189839

3.4 Validation Accuracy by Dataset

Average Validation Accuracy on Magic: 0.758116666667
Average Validation Accuracy on AbaloneB: 0.722782408849
Average Validation Accuracy on AbaloneU: 0.632267136619
Average Validation Accuracy on LetterB: 0.786844444444
Average Validation Accuracy on LetterU: 0.985711111111

3.5 Average Hyperparameter by Algorithm

Average Alpha on Naive Bayes: 140.74000017999995
Average # Neighbors on KNN: 80.26666666666667
Average # Features on Random Forest: 11.6

3.6 Best Hyperparameter for Algorithm by Dataset

Best Alpha for Naive Bayes on Magic: 500.00000005
Best # Neighbors for KNN on Magic: 8.0
Best # Features for Random Forest on Magic: 9.0

Best Alpha for Naive Bayes on AbaloneB: 550.0
Best # Neighbors for KNN on AbaloneB: 1.0
Best # Features for Random Forest on AbaloneB: 16.0

Best Alpha for Naive Bayes on AbaloneU: 5.5e-07
Best # Neighbors for KNN on AbaloneU: 148.0
Best # Features for Random Forest on AbaloneU: 8.0

Best Alpha for Naive Bayes on LetterB: 5.5
Best # Neighbors for KNN on LetterB: 4.5
Best # Features for Random Forest on LetterB: 13.0

Best Alpha for Naive Bayes on LetterU: 0.050000050000000004
Best # Neighbors for KNN on LetterU: 276.0
Best # Features for Random Forest on LetterU: 12.0

3.7 Analysis

Now we will offer our analysis of the data. First we look at the relationship between our findings and those of Caruana and NiculescuMizil. Our findings seem to be strikingly similar, emphasizing how empirical both studies were (since the findings were reproducible). The rankings are the same, with RF coming first, KNN second, and NB third. Also RB and KNN are both in the 80's (percent-wise), with KNN being low 80's. Our RF is about 4 percent lower than theirs, which leads to some questions about implementations and datasets. Maybe their implementations have something that genuinely bolsters accuracy. Both of our NB are at about 65 percent.
Next we look at how the algorithms performed on balanced versus unbalanced datasets. At first we see that the balanced abalone performed about 8 percent better than the unbalanced on average, and this seems to make sense. But then we see that the unbalanced letters performed almost 20 percent better than the balanced letters. In fact, the unbalanced letters had astoundingly high accuracy rates throughout this experiment. This might mean that it is very easy to say if a letter is an 'A' or not an 'A' instead of asking if it's 'A', 'B', 'C', 'D', etc. Maybe this will lend itself to new letter recognition problems where we compare to 25 other possibilities instead of trying to discern which letter we have in one loop.
Finally, we look at our hyperparameters. Looking at our alpha for NB, we see that the spread of most successful alphas is quite wide. This makes sense because it was not very successful, so it most likely would not have converged on a best hyperparameter. For # Neighbors in KNN, we see that even with big datasets, we tend to choose a smaller amount of neighbors (we even chose 1 neighbor for AbaloneB!), but large amounts of neighbors is not out of the question. However, we never get too close to the number of elements. For # Features in RF, we see that it tends to be close to the actual number of features, meaning it's not always good to eliminate too many features when building forests.

4 Conclusion

In conclusion, we have found two very good algorithms for working with data on a laptop and without a GPU: KNN and RF. Random Forests especially seem to be very impressive, as was also shown in the other study. They perform well on diverse sets. Naive Bayes however, did poorly in just about every scenario. In fact, recently we were working on another KNN project, but we were using the wrong array for the Y_train, and so our KNN was having very low accuracies. But these accuracies were still in the low 60's, which is very close to what NB produces without a bug in the code!
Also, our decision on balanced versus unbalanced sets cannot be made quite yet. This will require

a much more extensive empirical study. If we were to make a hypothesis, it would be that whether or not a balanced or unbalanced set helps or hurts accuracy, depends on the nature of the data itself. This was the case with the letter dataset.
Finally, we have given further evidence that a subset of the Caruana and NiculescuMizil results are empirically rigorous, since we have reproduced these results

References

[1] Savicky, P. (2017). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[2] Slate, David J. (2017). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[3] Waugh, Sam (2017). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.