Adhithya Ravishankar

Professor Isbell

Machine Learning

February 4, 2018

Supervised Learning

Introduction

Supervised learning is one of the simplest things to do in machine learning. It basically involves pre-identified data and using it to classify non-identified data. Supervised learning offers a way for machines to classify data based on a series of data points. There are a number of supervised learning methods such as Decision Trees, Neural Networks, Boosted Decision Trees, Support Vector Machines, and K-Nearest Neighbors. With these different types of learning methods, one can choose the learning method best designed for them.

We used Python for each of the classification problems. Python's Scikit-Learn library module has all of these classifiers with various properties for each classifier to help tweak the classifier. We used DecisionTreeClassifier for Decision Trees, MLPClassifier (Multi Layer Perceptron) for a Neural Network, GradientBoostingClassifier for a Boosted Decision Tree, LinearSVC for a Support Vector Machine, and KNeighborsClassifier for a K-Nearest Neighbors classifier. The results from the classifier are then inputted into either a learning curve or the various training datasets are fitted and testing datasets used to determine the accuracy of the current solution.

For the K-Nearest Neighbors, we decided to compare two k values: three and five to see if there were any major differences between the two. For the decision trees, a pruning mechanism is already implemented via a form of pre-pruning (stops the tree from growing the tree early in the process) done via the min_impurity_decrease variable, which only splits the tree

if the decrease in impurity (the measurement of how a random piece of data could be incorrectly labeled) is greater than the value input. This is done on all of Scikit's Decision Trees, including the two we used: DecisionTreeClassifier and GradientBoostingClassifier.

The data preprocessing is done via a number of libraries: Numpy, Pandas, and NLTK. To save on repeat processing, the processed data, before it is split into training and testing datasets, is saved into Numpy array files and loaded from there. That can save a minute or two on the processing time, so with multiple iterations and changes, that can save some time. Before it is sent to the classifier, it is separated into training and testing datasets using a percentage split, such as 0.5, 0.4, 0.3, or even 0.2, which indicates the percentage of records held over for testing.

For testing, we used two computers. The first was a Lenovo Thinkpad P50 with an Intel i7-6700HQ quad-core at 2.6Ghz with a max frequency of 3.5Ghz and a memory capacity of 32GB. The second was a custom-built PC with a Intel i7-8700K hexa-core at 3.7Ghz with a max frequency of 4.7Ghz and a memory capacity of 16GB. All the performance testing is done on the custom-built PC to ensure that they are all comparable.

First Classification Problem

The first classification problem is involving census data vs. income. It compares many demographics and social properties to the income that a person might have. These are very important statistics for a social or economics researcher as they can showcase income inequality across race, gender, and other socioeconomic and demographic factors. This first problem presents a variety of challenges as there is a large dataset to contend with that also has a large number of variables. To be exact, there are 199,523 records with 41 variables each, which brings the number of datapoints that we have to contend with at approximately eight

million. We are going to see how the performance of this compares to the performance of the other one, which has fewer records but a lot more variables in each record.

Second Classification Problem

The second classification problem is a more typical scenario in the computer science world: having to filter spam messages from the multitude of messages coming in. It uses word frequency of the top 5000 words in all of the messages combined. The way it does this is to add each word to a Python counter which automatically aggregates the words and the frequency in all of the messages. After this counter is done with all of the emails, the counter takes the five thousand most frequent words in all of the messages and compiles them into an ordered list. After stripping this ordered list of the frequency of the words, it is kept in its ordered fashion. Then you create a frequency matrix and a training labels: the former which is a frequency of the top 5000 words in the 37,822 emails and the latter which is the record if each of the 37,822 emails are spam or not.

Both of the above datasets are binary datasets in nature. They are a 0 or a 1 dataset. The census dataset is measuring whether you have income above or below $50K. The emails dataset is measuring if an email message is spam or not.

Results

For each of the two classification problems, the results are a compilation of a set of different results: a confusion matrix and accuracy score at 50%, 40%, and 30% cross validation. The percentages indicate the number of records left over for testing after the classifier has finished training. The second set of results are the learning curves for each classifier. This can showcase the change in percentage accuracy over a wider range of results.

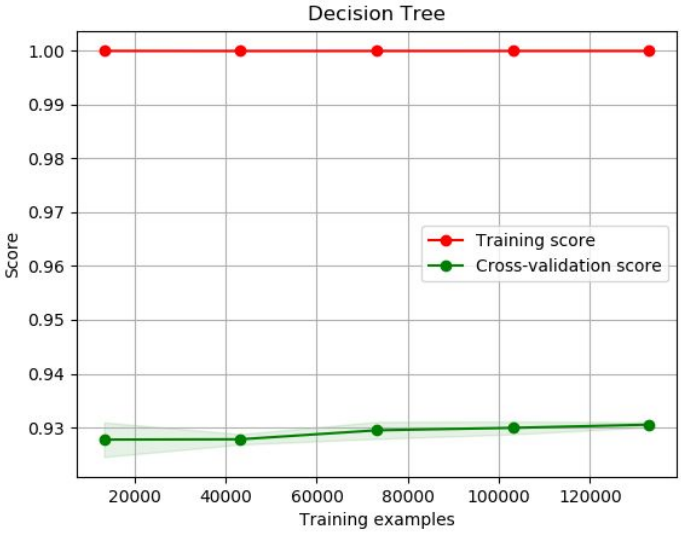
The first set of results are for the census income classification problem. The first test is showcased in the tables below. It showcases the accuracy of each of these methods as a

consistent over 90% accuracy is nothing to scoff at. It also showcases how a neural network has a steady but slow increase in its accuracy percentage increasing from 90.6% to nearly 94%, an improvement of over three percentage points.

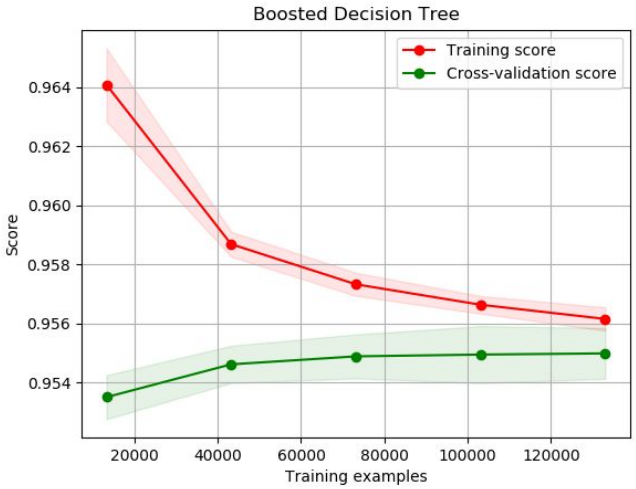
Testing Percentage	.5	.4	.3	.2
Decision Tree	92.99%	92.89%	93.19%	93.08%
Boosted Decision Tree	95.59%	95.46%	95.62%	95.51%
Linear SVM	94.17%	93.86%	91.35%	93.95%
Neural Network	90.60%	93.22%	93.34%	93.95%
K-Nearest Neighbors (k=5)	94.06%	94.16%	94.14%	94.28%
K-Nearest Neighbors (k=3)	93.65%	93.72%	93.67%	93.74%

Overall, other than the neural network, the accuracy percentage does not change that much with the testing percentage. I think the reason behind this is because there are tens of thousands of records already in the training set, therefore even a large change in the number of training examples do not change the accuracy that much. The most accurate was the Boosted Decision Tree, 5-Nearest Neighbors, 3-Nearest Neighbors, Linear Support Vector Machine, Decision Tree, and lastly a Neural Network. But, the range between the most and least accurate is less than three percentage points, so accuracy is not as much of a viable comparison.

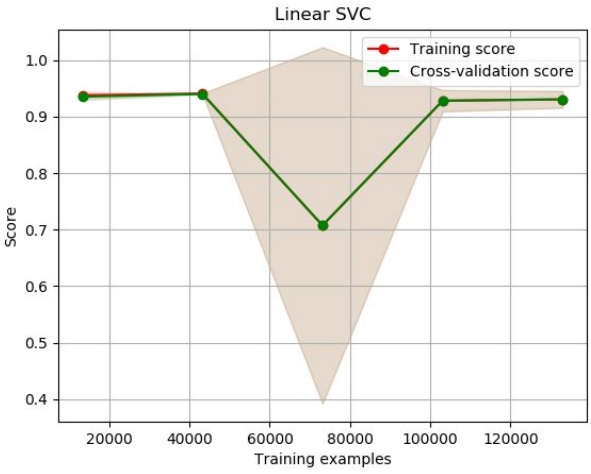
The second set of results show that the decision tree does not show any visible improvement over the number of training examples that are iterated over in this learning curve. Even with only 20,000 training samples, a decision tree can render a minimum of a little less than 93% accuracy, and an increase by



over a 100,000 samples yields less than a percentage point improvement. This shows the fixed nature of decision trees in general: after installing the number of leafs and nodes necessary for it to operate properly, the decision tree gains a high accuracy and is unable to improve it in fear of overfitting. The boosted decision tree shows a very different graph, however. The training score drops significantly at the beginning and slowly levels out as the frequency of the training examples increases. The cross-validation scores also slightly increase as the training examples increase, but the fascinating thing about the boosted decision tree is that the uncertainty increases as the training examples increases. In most other scenarios, the cross-validation curve uncertainty decreases as the training examples increase.

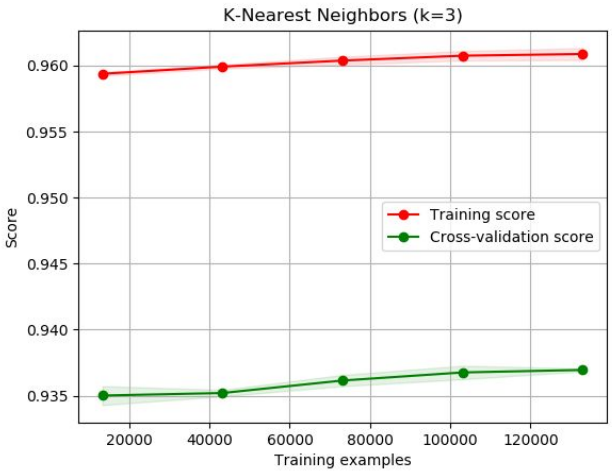


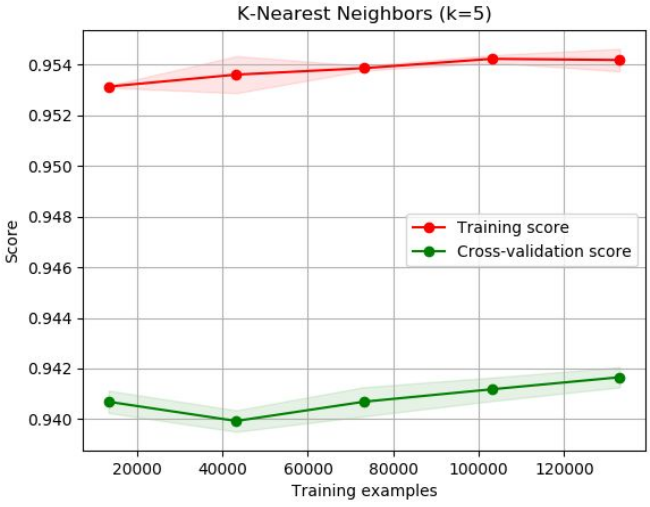
Next is the Linear Support Vector Machine, which has an unusual cross validation and training score. I'm not sure if the learning curve is faulty via some sort of processing error, which might explain the huge jump and decrease in between the 50,000 and 100,000, which then jumps back to a more normal position for the curve. Another possible reason for the decrease in scores is that the classifier overfitted for the problem. The Linear Support Vector Machine also shows an increase in uncertainty as the number of training



samples increase, thought that might be because of the huge drop in score in the middle of the chart.

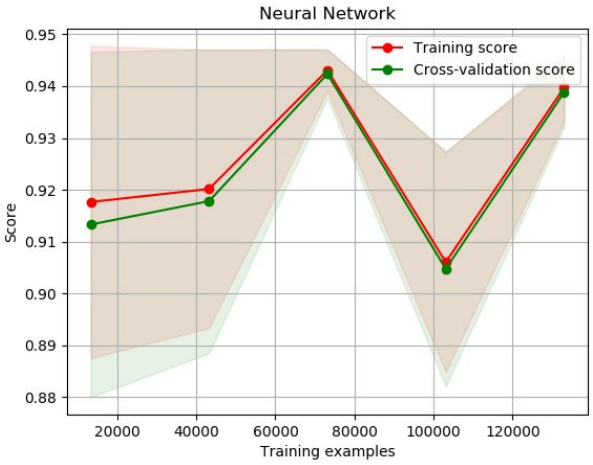
One of the most common Machine Learning classifiers used, K-nearest neighbors, calculates the score based on an arbitrary number of neighbors that are closest to it. This shows a relatively linear but slow increase as the number of training examples increase as well. Both curves do not exhibit more than a half of a percentage point increase as the number of training examples increases by over a hundred thousand. The uncertainty does decrease with the increase in training examples, even though there are points where it does slightly increase as well. The difference between the number of neighbors being three and five is not as noticeable in the above table and in the chart as well. The only difference is the decrease in the cross-validation score at one point in the five neighbors learning curve, but it rebounds back quickly and is still slightly above the three-neighbors chart at every single point, as the three-neighbors chart never even hits 94%.





The last learning curve to plot is the neural network curve. The neural network curve shows big swings in uncertainty and training, possibly due to the overfitting at some points. You

can see the scores jumping to over 94% at approximately 75,000 training examples, and then falling back down to approximately 90.5% at sometime over 100K training examples, and then rebounding back up to nearly 94% at the 130K mark. It's possible that it could show even more bounces like these with more training examples, so it would probably be unnecessary to try even more possibilities.



Next, we analyze the next classification problem: the emails. The emails are, as you might remember, a different type of problem set than the census data. First, the census data is more value orientated and the emails data is more frequency orientated. Second, there are more variables in the emails data than in the census data, while less records in the latter.

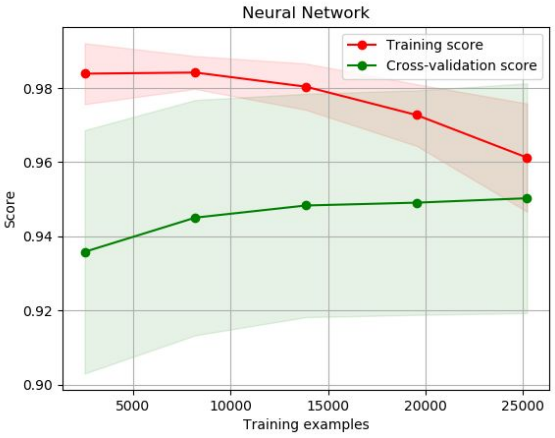
Classifier	0.5	0.4	0.3
Linear Support Vector Machine	95.01%	95.07%	95.18%
Neural Network	95.34%	95.49%	95.57%
Decision Tree	94.20%	94.45%	94.52%
Boosted Decision Tree	93.57%	93.78%	93.79%
K-Nearest Neighbors (k=3)	88.55%	89.60%	90.54%
K-Nearest Neighbors (k=5)	88.55%	88.46%	89.76%

I think the fascinating analytic from this one is that most if not all of the values are increases from the previous value. There is only one minor decrease from the previous value with K-Nearest Neighbors when k=5 from test size 0.5 to 0.4. However, the increase is not big enough in any of the cases to warrant the increase in testing size. You can only train on it with

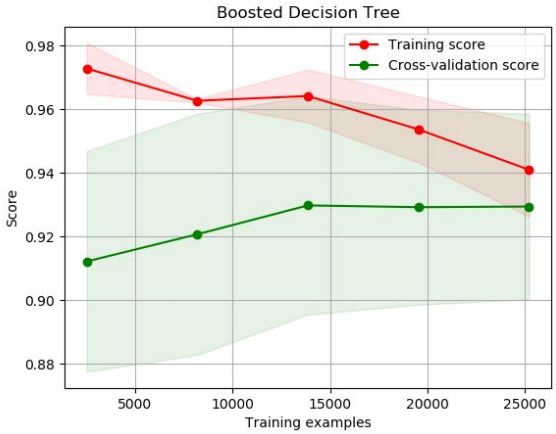
only half of the data and it should work very accurately. The classifiers from most accurate to least were Neural Network, Linear Support Vector Machine, Decision Tree, Boosted Decision Tree, and then K-Nearest Neighbors (5 then 3).

Both of these datasets have seen an ever so slight increase in accuracy when increasing the number of neighbors. When comparing the most accurate classifiers to the census data, the fascinating part is that none of the top three in this dataset were in the other or vice versa. This shows us that different datasets can yield different results in terms of which classifier is best, so it would be best to choose a classifier based on the dataset that you have. The range in this one in accuracies is nearly five and a half percentage points, so a bigger jump than the last one, but not a major jump. Some could argue that this range is bigger to the point where it needs to be looked at more for a possible accuracy boost. However, with this one, the more accurate classifiers are also the better performing ones, so there need not be a performance loss in exchange for more accuracy.

The set of results to analyze are from the second classification problem: whether an email message is spam. The learning curves for four of these look very similar: there are not that much of a difference between the four of these curves. The four that are in comparison at this point are the Boosted Decision Tree, the Decision Tree, a Neural Network, and a Linear Support Vector Machine. Each of the Training



Score curves starts at over 98%, and three of the four end at 96% at the 25,000 training examples mark. The only one that ends lower is the Boosted Decision Tree. For the cross validation score, the Decision Tree and Boosted Decision Tree make it to the same point at approximately 93%, which means that the Boosted Decision Tree did not receive that much of a



boost. Both the Linear Support Vector Machine and the Neural Network received a cross validation score of 95%, which is very good. The Linear Support Vector Machine was the fastest and one of the most accurate classifiers in the emails spam detection dataset.

The time it takes for each of the problems is shown in the table below.

Classifier	Time (Email)	Time (Census)
Support Vector Machine	1.60s	30.45s
Neural Network	139.11s	6.48s
Decision Tree	25.45s	1.06s
Boosted Decision Tree	506.89s (8.45m)	27.82s
K-Nearest Neighbors (k=3)	1618.47s (26.97m)	120.17s (2.00m)
K-Nearest Neighbors (k=5)	1611.33s (26.86m)	118.84s (1.98m)

None of these classifiers are in the same ballpark as eachother, but they also do not have any correlation with each other, which might show how the differences in the problem sets affect the times. If we can go back to the problem sets: the email problem set is a 5000 variables, 40K records whereas the census information is 40 variables, 200K records. One can see via these times that the Support Vector Machine is the only one of the classifiers that shows a noticeable increase in time between email and the census. All of the others show decreases in time. Now, this could be because there are more datapoints in the email than in census data, 200 million vs 8 million, which does not make this as fair of a comparison as it could be. A fairer comparison for time to see whether variables or records have more of an impact is to decrease the number of variables in the email to 200, or to increase the census data by about 25x, which would make the number of records about 5 million, so about so that each problem would have an equal number of datapoints.

There are many analyses that one can takeaway from this table. K-nearest neighbors is consistently the most time-consuming of the classifiers, probably due to the fact that it has to calculate the decision on each test record by comparing it to all the other training records it already has. Therefore when the training set increases by 1, the time also increases steadily, something not an issue with a classifier such as a decision tree. A boosted decision tree always has a increase in time in comparison to a regular decision tree, but does not have a comparable increase in accuracy. A neural network always seems to be middle of the pack in terms of time for both the census data and the emails. Last but not least are SVMs, which seem to be fast when there are a lot of variables involved but not much records.

All of these measurements and analyses should be taken as initial looks, because it is not as if we are measuring many iterations or varieties of datasets. I do not think that you can optimize many of these, but K-nearest neighbors, which was the slowest in our tests, also

seems to be the most optimizable. This was not because it was the slowest, but because it might be the most parallel, therefore it would be easier to run on a GPU. I might try next time to use more GPU-optimized ones to see how that compares to see the non-GPU optimized and also the others ones that might not require the usage of a GPU.

I think it might be fascinating to see how modifying the number of variables or records can impact the faster ones on the sheet. Modifying the number of most frequent words from 5000 to 4000 or 6000 would obviously decrease or increase the time needed for K-nearest neighbors to do its work. It would be fascinating to see how it would affect the performance of a classifier like Linear Support Vector Machines, which was already very fast. It only took 1.6 seconds for a Linear Support Vector Classifier to fit and predict tens of thousands of records with thousands of variables. One could measure how much of an increase in variables and/or records would be necessary to make that same time a minute and how much of a percentage increase in accuracy would you see based on that.

Conclusion

In conclusion, there are many great options in supervised learning that given enough data, they can provide accurate guesses to similar problems or test data. Supervised learning can not only provide with unique or interesting ways to look at data, but can also provide ways to find trends and analyses not found before.