Shawn Vega

09/24/2017

Machine Learning

Machine-learning Project 1

For this project the problem given to us was to implement 5 learning algorithms. We were asked to implement decision trees with some sort of pruning, neural networks, boosting, support vector machines, and K Nearest Neighbors.

Starting on this project, I encountered my first problem. I needed to find a good data set. On my first attempt I picked a data set of spam text messages and wanted to train a machine-learning algorithms to differentiate between spam and ham text messages. Once I got my Python environment set up I realized that I did not know how to do any natural language processing so I gave up on that attempt. I sought out machine-learning examples which included numerical data only. At http://archive.ics.uci.edu/ml/datasets.html I found two datasets for me to use.

One machine-learning dataset is called the "Iris data set"[1]. The Iris dataset is named after the Iris plant flower. The data set contains three classes of flowers with 50 samples each. The data set includes four attributes: the septal length in centimeters, the septal width in centimeters, the pedal length in centimeters, and the petal width in centimeters. In addition to those attributes, it contains one of three labels per row of data as follows, "Iris Setosa", "Iris Versicolour", and "Iris Virginica". This dataset has 150 samples which I wasn't sure if that was enough data to train on, but after writing and trying the code it seems to be enough data for the machine learning algorithms to learn accurate prediction functions.

Another dataset I found is called "Human Activity Recognition Using Smartphones" found at http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones [2]. I found this dataset to be especially interesting because I make Android applications for a living and use sensor data

all the time but have never actually used it to recognize human activities such as the ones described by this data set. The dataset was created with 30 volunteers and each person performed six activities such as walking, walking upstairs, walking downstairs, sitting, standing, and laying down. They used two sensors, accelerometer and a gyroscope. Signals were pre-processed for machine learning with sliding windows of 2.56 seconds and a 50% overlap. This dataset is a better measure of performance due to the fact that the training set has 7,353 examples and 561 attributes per row. The testing set has 2947 rows.

I wrote my code in Python 3.5 because it is the version of Python that is compatible with TensorFlow library which I plan to use later. Many people in the class were suggesting the Scikit-learn library for machine learning, so I decided to use it.

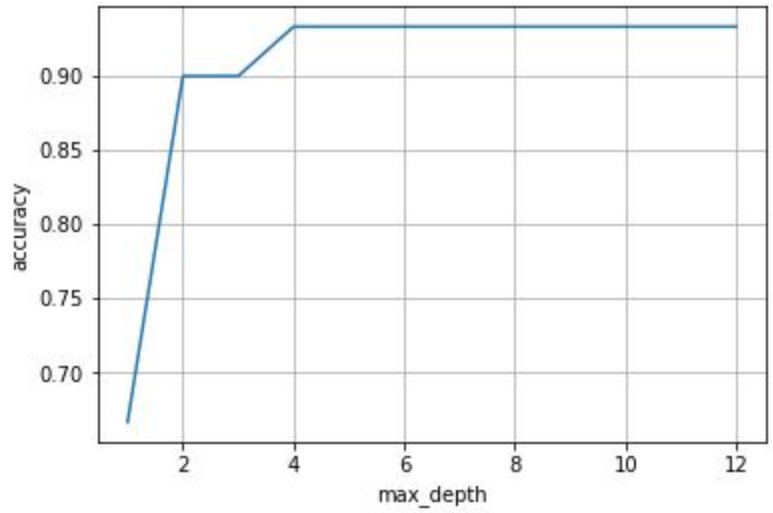
The raw data in the Iris dataset was in a text file in the CSV format with columns separated by commas and rows separated by the newline character. Similarly the Human Activity dataset was in CSV format, but used spaces instead of commas to separate columns. Preprocessing the data for the Iris dataset consisted of taking the data and mapping the text labels to integers. After the text labels are mapped we separate the attributes and target variables into separate arrays and convert the attributes to float values. The Human Activity dataset was already preprocessed, but for some reason I had to filter out a few empty items from the rows. Afterwards for Iris I split the data into a training set and a testing set. One-fifth of the data was in the testing set and four-fifths of the data was placed into the training set. In the case of the Human Activity data it was already split into training and test data so I left it as is.

Decision Trees:

For my Decision Tree implementation, I used the Scikit-learn decision tree classifier. It was fairly simple to setup and use with a constructor to pass the settings of the tree, a function to pass the data to fit the tree, and a function to test the validation set of data. SciKit learn also included a function that calculated the accuracy percentages. I implemented pre-pruning by varying the maximum depth of the

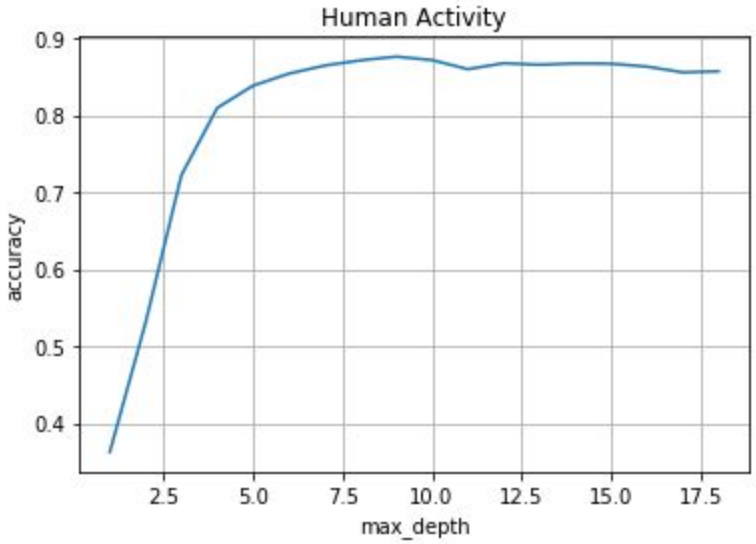
tree. Different trees with different depths seemed to produce very different results. Trees with a depth set to 1 on the Iris data only achieved an accuracy of 66.6%, whereas; if it was greater than one it seemed to perform better until a maximum height of four where the accuracy was 93.3% and remained the same with even greater max heights. A table and a graph of the results with a tree max height to 12 are included below for the Iris dataset.

max depth	1	2	3	4	5	6	7	8	9	10	11	12
Accuracy %	66.6	90	90	93.3	93.3	93.3	93.3	93.3	93.3	93.3	93.3	93.3



While working with the Human Activity dataset and implementing the machine learning algorithm known as decision tree, I noticed that the decision tree did take longer with the larger data set but it still only took less than 5 seconds. Unlike the Iris dataset, the Human Activity dataset achieved maximum accuracy of 87.6 percent at a max height of nine.

max depth	1	2	3	4	5	6	7	8	9	10
Accuracy %	36.3%	53.1%	72.2%	81.0%	83.8%	85.4%	86.5%	87.1%	87.6%	87.2%



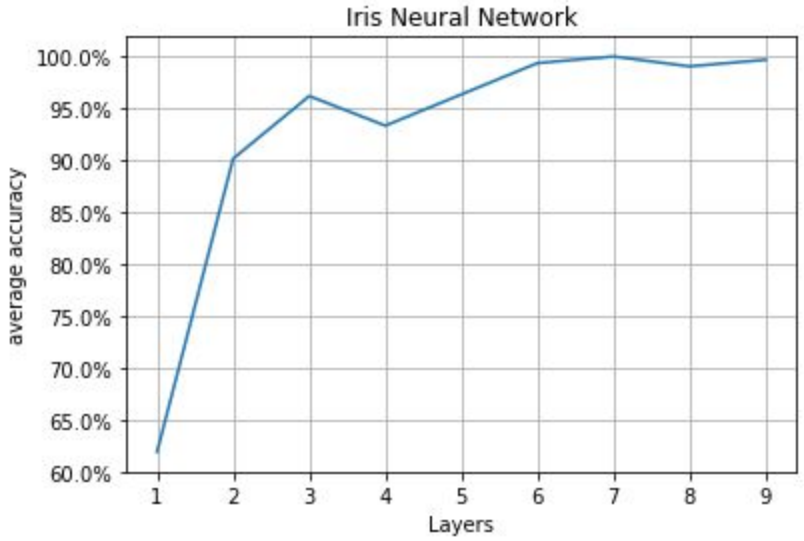
After implementing both decision trees and examining the performance of the two trees, it is clear that it is very easy to over fit the data with decision trees and that many maximum heights must be tried in order to fit the training data without overfitting it.

Neural Networks:

I then proceeded to implement the neural networks for the Iris data set. The implementation for the neural network using the Iris data set was done with Scikit-learn neural network with their MLP classifier and the LBFGS solver. I tried different hidden layer sizes and different random initial states with one hidden layer.

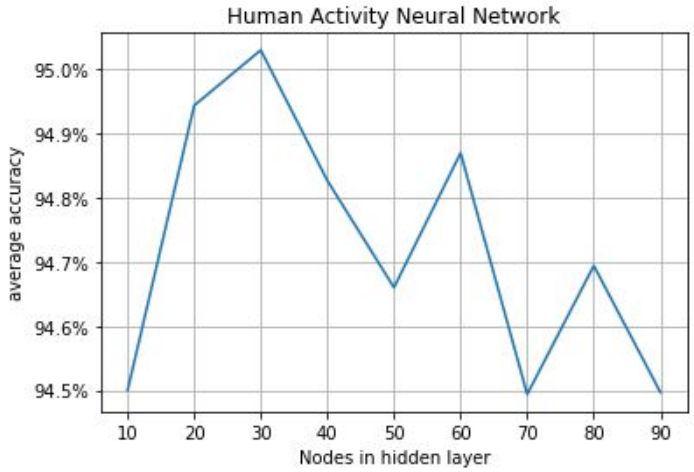
When implementing the neural network on the Iris data set, I noticed that there were few big factors when training the neural network. It seemed like one of the biggest factors was the random initial state of the neural network. Depending on the random state sometimes a neural network would only get 33% accuracy whereas other times they would get a 100% accuracy when the data was trained with only one node in the hidden layer. For the Iris data set, it seemed to only need one hidden layer so long as the proper random initial state was set to achieve a 100% accuracy. With more layers that seemed more likely to learn a function for the data set that was 100% accurate on its predictions. That is shown in the graph

below which shows up to 9 layers with 20 trials for each set of layers each of the twenty trials starting with a different random initial state.



For the Human Activity data set the neural network was similar to the Iris dataset except it never seem to achieve 100% accuracy. It would achieve different levels of accuracy based on the number of layers and different random numbers. For example, with 1 node it was able to achieve a 70% accuracy whereas with two nodes it was able to achieve a 93% accuracy. After five nodes we were able to achieve a 97% accuracy, but these were maximums given different random numbers for the initial state. For the average given 10 random numbers, the accuracy kept going up as we proceed from 1 to 30 nodes where it peaked, as I added more nodes the average tended to go downward. I proceeded to run the experiment from 10 to 90 nodes. The results are displayed in the graph below. From the results a person can infer that too many nodes can overfit a dataset. Another lesson learned from this series of experiments is that only a small amount of nodes in a hidden layer can achieve a high accuracy given the right random number, and

will achieve a low accuracy given the wrong random number.



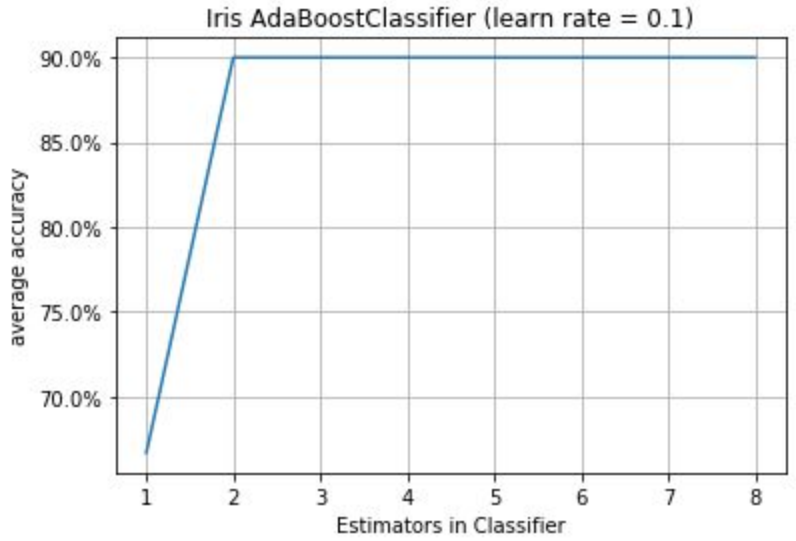
Another noteworthy observation about the neural networks is that they seem to depend less on the random number with an appropriate step size, denoted alpha, which can depend on the data. The neural networks were started with an alpha of 0.00001 but they were later changed to 0.1 for the Iris data based on achieving a better average result through trial and error. The same alpha was used for the Human Activity dataset. My last observation about neural networks is that they take a long time to run, and the more nodes the longer they take to run.

Boosting:

For boosting I used the Adaboost classifier built in to scikit learn, I tried different learning rates and a different number of estimators. The default learning rate of one didn't work well for either data set and had to be adjusted for both datasets. Tuning also had to be done for the number of estimators. When using Adaboost it was observed to be one of the slowest of all the algorithms. I believe this is do to the fact that it is an ensemble learning algorithm which means that we have to train n number of estimators which in this case was decision trees so not only did we have to train n decision Trees, but we also had to have each of the decision trees perform the prediction and multiply them by their respective weights. In

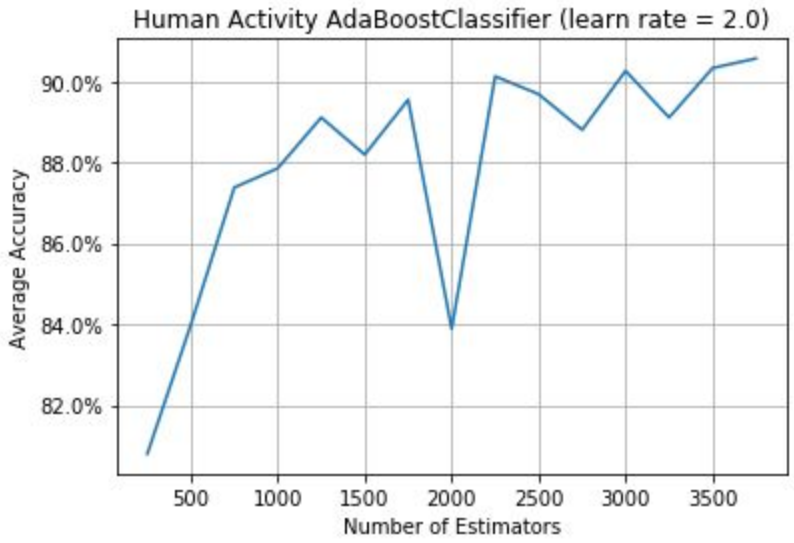
practice this resulted in waiting minutes to train the classifier when the number of classifiers was greater than about 300.

When boosting the Iris dataset with Adaboost the default learning rate of 1.0 performed poorly when the number of estimators was less than two, but when using two to four or more than fifteen estimators the algorithm performed with a 90% accuracy. When the learning rate was set to 0.1 it seemed to get a 90% accuracy always with two or more estimators as shown in the graph below.



When boosting the Human Activity dataset, the default learning rate of 1.0 always seemed to produce predictions that were accurate to 53.1% no matter the number of estimators. With this dataset something strange was observed. The unusual thing about Adaboost was that the higher the learning rate the more estimators it needed to produce more accurate results. That is that the results would become more accurate with a higher learning rate, but only given more estimators. With a learning rate of 2.0, the Adaboost algorithm was able to achieve a maximum prediction accuracy of 90.5%. To achieve that accuracy, 3750 estimators were needed. In Adaboost and boosting in general, the more estimators the longer it takes to run, for example; to run 250 estimators on the Human Activity dataset it only took 77 seconds whereas to run 2500 estimators it took 219 seconds. This long run time is why the graph does not

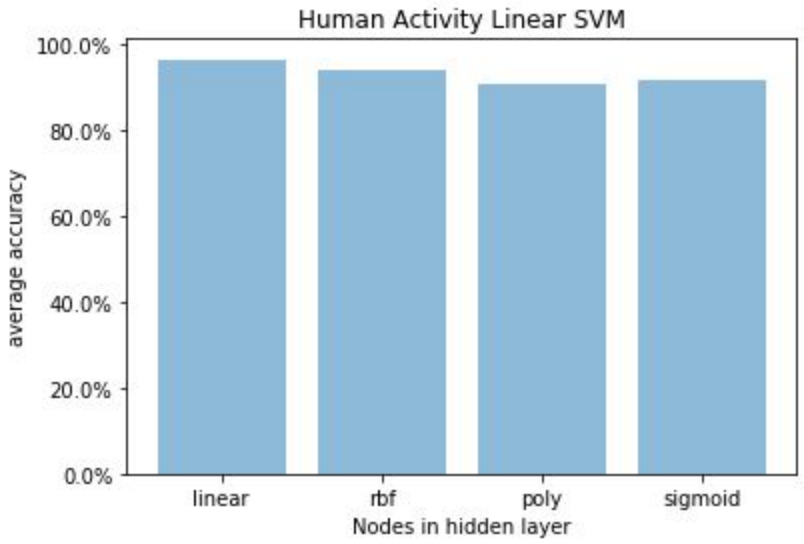
go further in addition to the fact that both 4000 and 4250 estimators were tried and achieved a lower accuracy than the 3750 estimators.



Support Vector Machines:

There were several Support Vector Machine algorithms that were used on the Iris dataset. The four types of SVM kernels used are linear, RBF, poly, and sigmoid. It seems that the four types of SVM were able to easily classify the Iris test data with 100% accuracy.

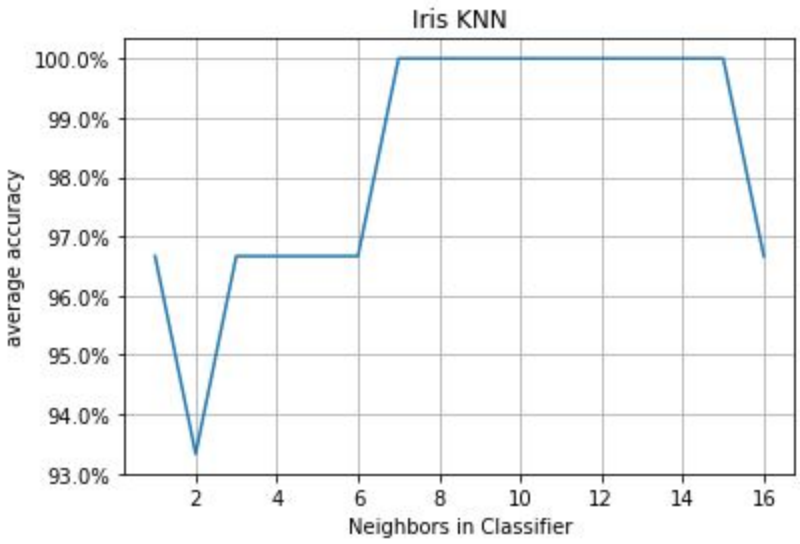
When applying the four types of SVM algorithms to the Human Activity dataset, the four types of SVM kernels performed with different accuracies. The linear kernel was 96.4% accurate whereas the other kernels did not perform as well with RBF coming in second at 94.0% accuracy, sigmoid came in third at 91.7% and polynomial came in last at 90.7% accuracy. Different penalty parameter C of the error term were tried. When C was a large number the linear classifier would do worse, but the other kernels would perform better when C was equal to 1,000,000 the rbf kernel's accuracy increased to 96.9%. Different tolerances for stopping criterion were tried for the SVMs, and it seemed to make the linear kernel slightly worse and make the others better up to the number 1 where all the kernels started to do worse. When weight class was set to balanced the sigmoid function accuracy got bumped to 92.3%.



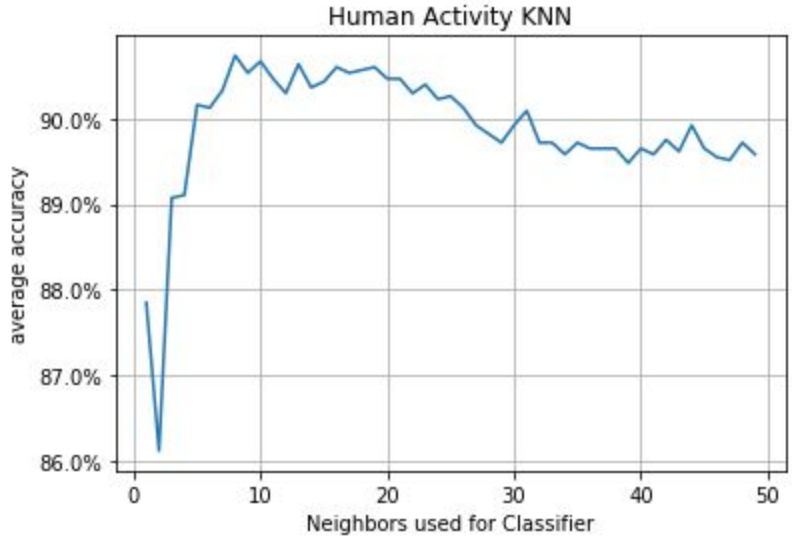
K-Nearest Neighbors:

For the K-Nearest Neighbors classifier the Scikit-learn implementation called KNeighborsClassifier was used to perform classification. It seemed to perform training quickly while performing slowly for predictions. This was the opposite to most other algorithms most of which performed slowly on the training, but quickly for the predictions. This is probably due to the fact that the KNN algorithm works by just storing the training data which is relatively a fast operation compared to how KNN performs predictions. When KNN performs predictions, it has to calculate the distance to all samples in the training set and then find the closest samples which is a O(n*log(n)) operation.

With the Iris dataset, the KNN classifier seemed to work almost instantly but this is expected as the dataset is small. As seen in the graph below, the number of neighbors that performed the best was from 7 to 15 neighbors which resulted in a 100% accuracy. Whereas in this limited trial, two nearest neighbors performed the worst.



When using the KNN classifier with the Human Activity data, that's when the slower predictions described above was truly observed due to the test dataset being much larger than the Iris dataset. For the Human Activity dataset the KNN classifier performed its worst with k equal to 2 with a 86.1% accuracy and performed the best with k = 8 with a 90.7% accuracy.



Conclusion:

In experimenting with five classifiers on the two datasets, a couple of major results can be gleaned from the results. For the Iris dataset the a few classifiers were easily able to achieve a 100%

accuracy. Those classifiers were the Neural Network, the SVM, and the KNN. If given the Iris dataset it would appear that either of those three would perform equally well.

With the Human Activity dataset the classifiers performed with the following maximums. Decision Tree was accurate 87.6% of the time, Neural Networks were accurate 97% of the time, Boosting with Adaboost was accurate 90.5% of the time, Support Vector Machines predicted accurately 96.9%, and K-Nearest Neighbors was accurate 90.7% of the time. With that said the Neural Networks seemed the best even though they take a longer time to train than some of the other algorithms. The easiest algorithm to reach maximum was the SVM because it didn't require much tuning. Neural Networks were on top on both dataset, this is probably because they are adept at not only learning linear datasets, but also because they can learn nonlinear functions as well.

In the future I'd like to experiment with more machine learning algorithms perhaps something like neural networks with more hidden layers and seeing if possibly deep learning neural networks perform better. I suspect that those will take even more time to run than the single hidden layer. Another interesting thing to try would be reinforcement learning although I'm sure I'll be doing something with that in future assignments.

[1] Creator: R.A. Fisher

Donor: Michael Marshall (MARSHALL%PLU '@' io.arc.nasa.gov)

[2] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L. Reyes-Ortiz. A Public Domain Dataset for Human Activity Recognition Using Smartphones. 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2013. Bruges, Belgium 24-26 April 2013.