Milind Lingineni CS 4641
Supervised Learning Assignment

Supervised Learning Report

Datasets

Adult Census Income Dataset: The Census Income dataset has 48,841 instances of records containing census information about adults to predict whether their income will exceed $50,000 per year or not. There are 14 features which are included such as age, work class, education, occupation, race, sex, hours per week along with several other potentially relevant statistics. Education and education-num ended up representing the same data so I deleted the education column.

Diabetes 130 US Hospitals from 1999-2008: This dataset contains information about 100000 patients with diabetes mellitus collected from 130 U.S. hospitals and integrated delivery networks over the course of 10 years (1998 – 2008). The task is to classify whether a given patient will be readmitted to a hospital within the next 30 days, greater than 30 days or won't be readmitted for diabetes based on their conditions and background. The types of the features include race, gender, age, weight, number and type of procedures, diagnoses, medications, and length of stay among many other factors. I chose this dataset to contrast with the Adult dataset because there are almost twice as many instances and almost four times as many features.

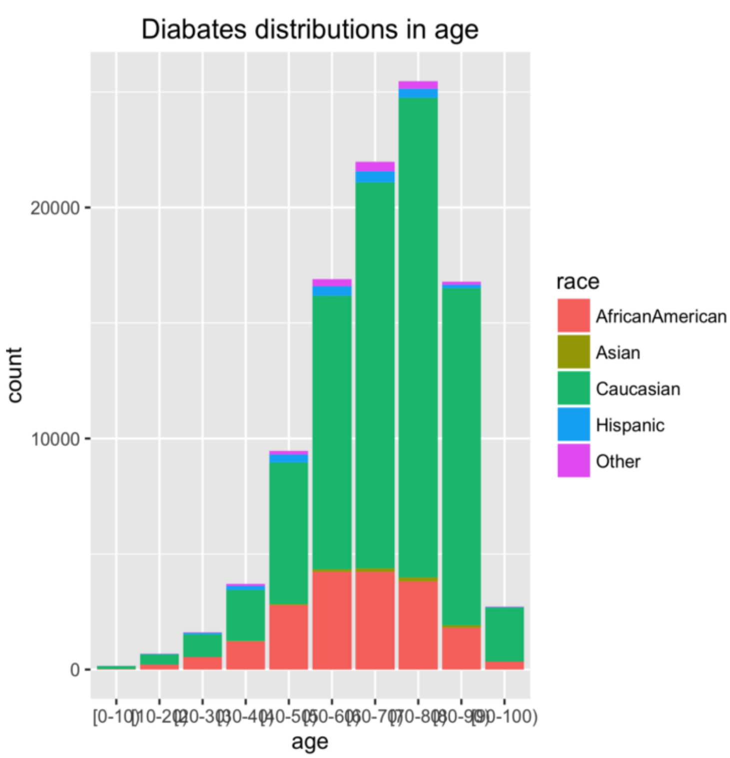


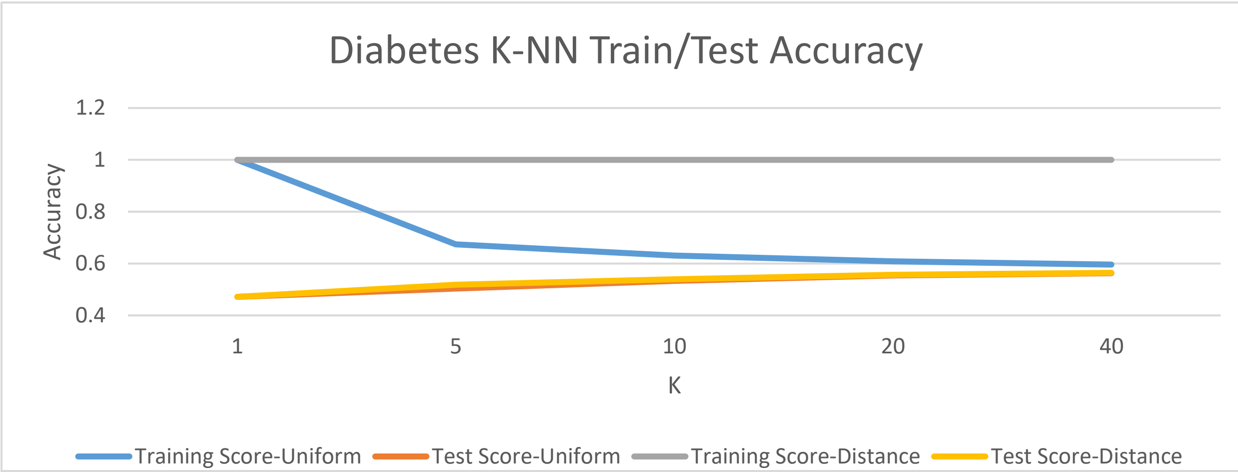
Figure 1. Visualization of the races and ages of patients represented in the diabetes dataset.

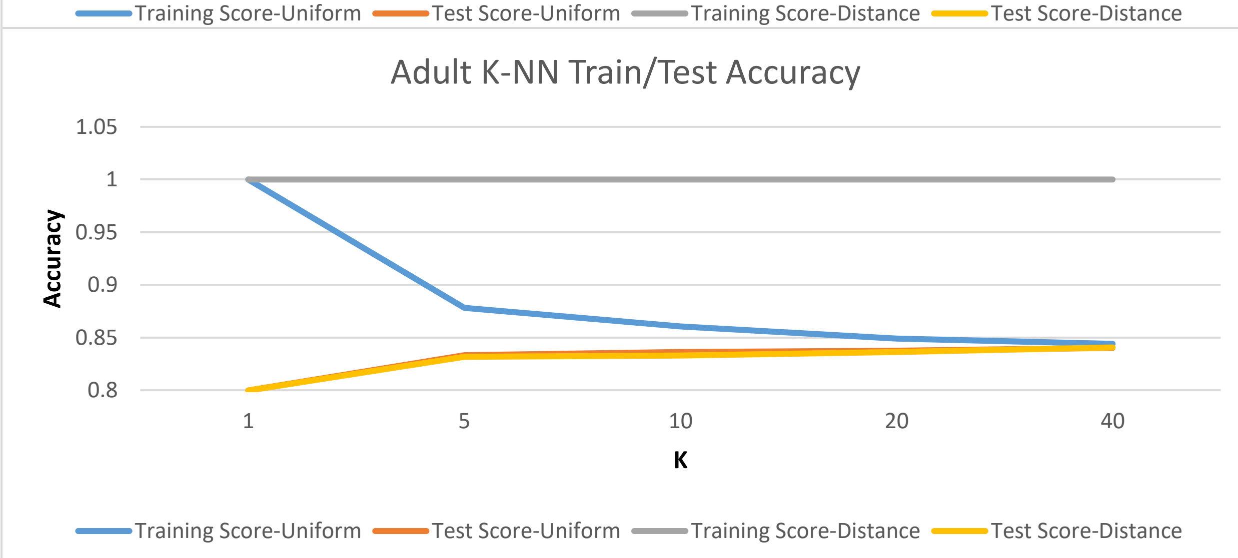
What makes the datasets interesting?

I found the adult dataset intriguing because I found it to be a practical application of machine learning to be able to make predictions based on U.S. census data. Considering that the census is performed every ten years and gathers a wealth of data, I thought it was an interesting usage to predict expected income, which might be useful in economics research for tracking predicting other information. I'm curious about the intersection of machine learning and finance so this might help gain insight into those types of problems.

The diabetes dataset is especially noteworthy to me because I've always been interested in applying machine learning to biology and disease detection/prediction but the difficulty in finding substantive datasets has been a great setback. This dataset provides me the opportunity to learn how to apply what I'm learning to a public dataset and to learn how effective certain algorithms perform on this type of data. With 55 features and 100,000 instances, this dataset will take much longer to run than the adult dataset and may face the curse of dimensionality but will be a fun challenge.

K-Nearest Neighbors:





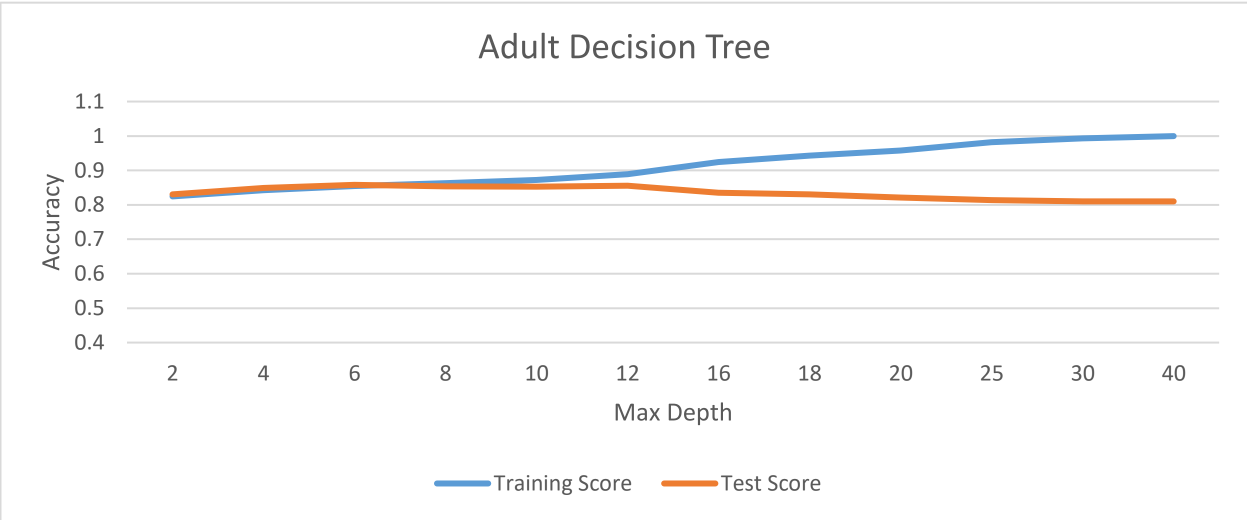
Adult/K	Weight	Training Score	Test Score	Train Time(s)	Test Time(s)
1	uniform	0.999956125	0.799774798	0.418576	0.860286474
5	uniform	0.878071253	0.833248029	0.3980951	1.389649391
10	uniform	0.860521236	0.836421333	0.4276392	1.557108641
20	uniform	0.849069849	0.837444979	0.3724897	1.929156542
40	uniform	0.844024219	0.840208824	0.4401507	2.318674326
1	distance	0.999956125	0.799774798	0.4261568	0.882848978
5	distance	0.999956125	0.831610196	0.4012616	1.317504406
10	distance	0.999956125	0.832838571	0.430697	1.516495705
20	distance	0.999956125	0.836318968	0.3830512	2.025887251
40	distance	0.999956125	0.840618282	0.4201863	2.170774221

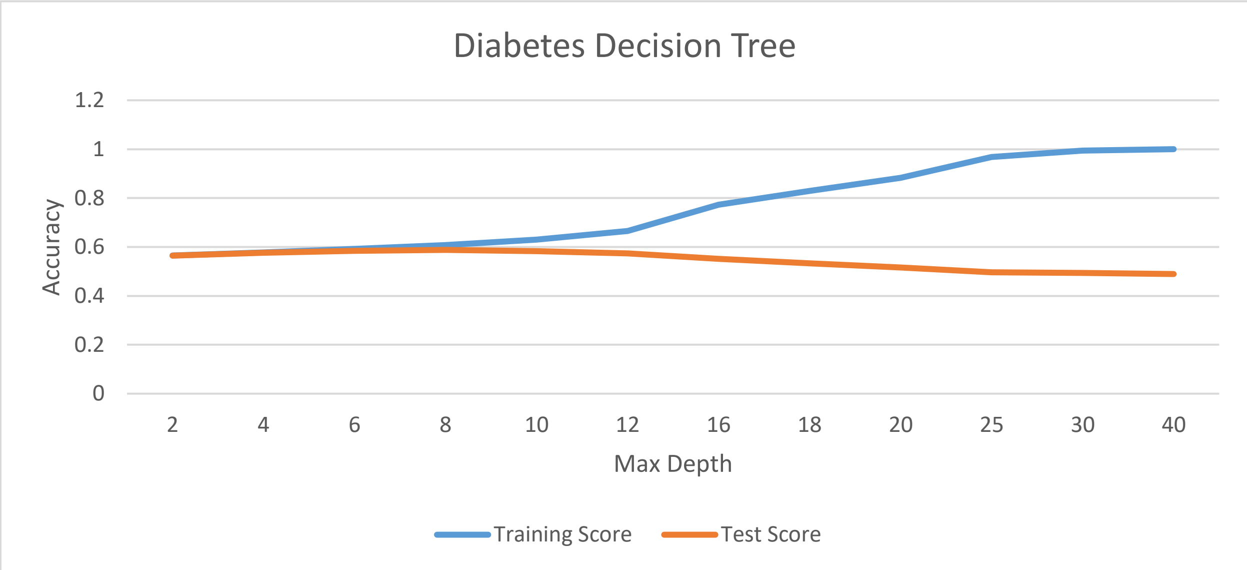
Diabetes/K	Weight	Training Score	Test Score	Train Time(s)	Test Time(s)
1	uniform	1	0.471634458	19.36810493	54.96591544
5	uniform	0.673774496	0.503373731	20.47715831	57.84559441
10	uniform	0.630594082	0.532721913	23.13471365	55.01594758
20	uniform	0.609130215	0.554110711	21.20039797	58.7419765
40	uniform	0.596103094	0.562659679	19.68697882	55.08915067
1	distance	1	0.471634458	20.36287546	57.84766889
5	distance	1	0.518964953	21.70937753	52.76754642
10	distance	1	0.539698657	22.69831467	53.67534304
20	distance	1	0.556763839	19.66303349	58.2211318
40	distance	1	0.564231903	21.98511624	57.17383766

The K-Nearest Neighbors algorithm underscores the issue of the curse of dimensionality. When using a distance-based weights for the nearest neighbors, the algorithms devolve to 1-NN where it gets almost 100% training accuracy because the neighbors are so far away distance-wise. With increasing K values, the test score went up increase around 5-9% from 1 neighbor to 40 neighbors. The fact that the diabetes dataset contained significantly more attributes might have contributed to such low test scores due to the curse of dimensionality which would require exponentially more instances for each attribute added. Another reason might be that I converted the categorical data to integers which creates a false assumption of ordinality and a one hot vector encoder would be better suited but also would increase the dimensionality of the data significantly as most features are categorical. K-fold cross validation could have been helpful to find optimal hypotheses.

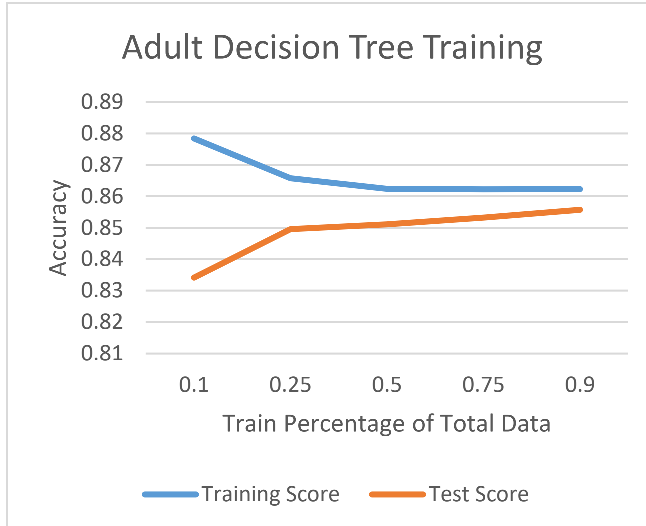
As KNN is an instance based learning algorithm and a lazy learner, I thought that increasing K would increase the test time; however, the diabetes dataset hovered around the same test time while the adult dataset grew somewhat linearly but for small values. The training times were low because no learning needs to be performed, just storing the points. As there are twice as many instance and nearly four times as many attributes for the diabetes dataset, it makes sense that the test times are much longer. I was surprised that increasing K consistently increased the test accuracy which might be due to the smoothing caused by higher K values.

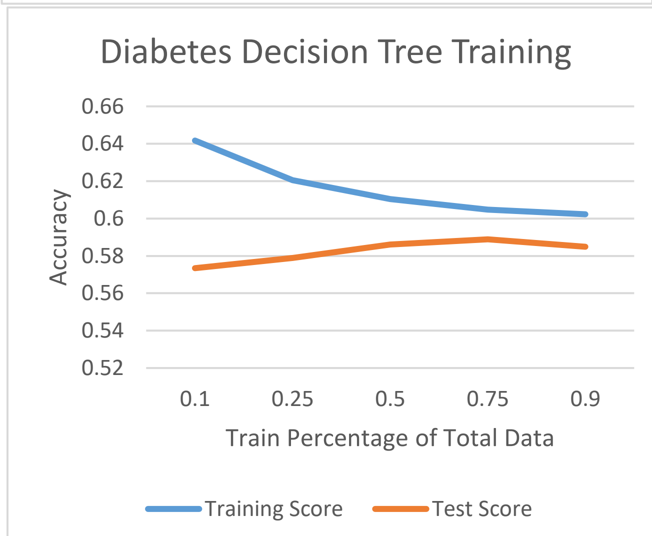
<u>Pruned Decision Tree</u>:





Scikit-learn, the Python machine learning library I used, doesn't support decision trees with pruning so I varied the max height to determine what would be the ideal height for varying train/test splits afterwards. The decision trees trained blazingly fast which surprised me as I thought it would be more complicated to train. The test error decreased up until around a height of 10 or 12 when it rebounded upwards because it started overfitting on the training data and did not generalize well due to its complexity. I recognize that I used accuracy not error for the learning curves so the graphs are essentially upside down of what the error graphs would be.

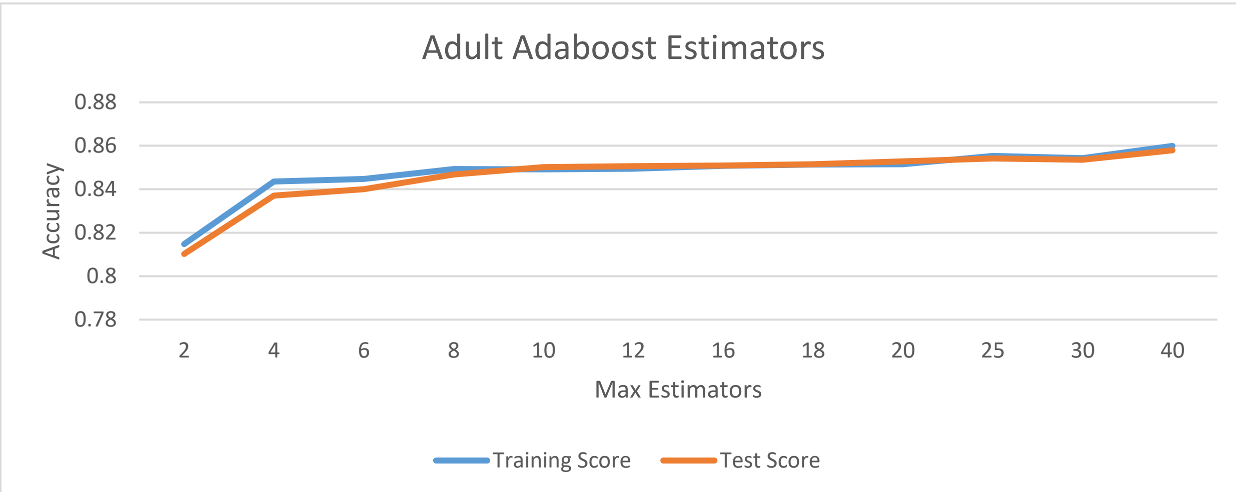


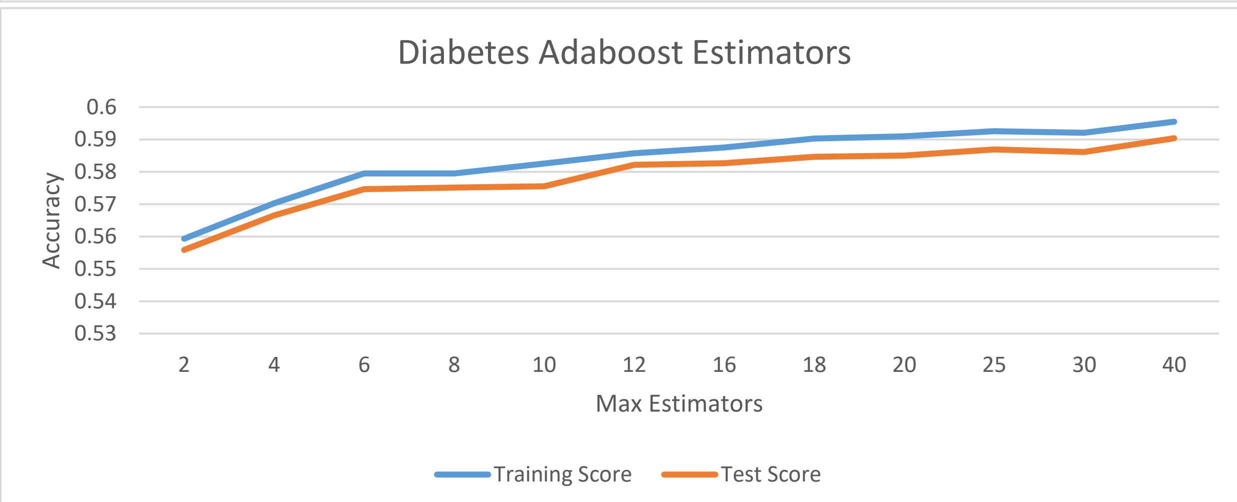


The pruned decision tree performed a few percentage points higher than KNN, which I believe to be because decision trees aren't as strongly affected by the curse of dimensionality as KNNs are. The medium sized trees performed the best and were used when testing the scores over different train/test splits because they learned much from information gain and reducing entropy without overcommitting to the data. Increasing training data generally lead to increases in test score until very high proportions (>.80) for the diabetes data which might be because there's more entropy in a large training set which helped the decision tree generalize more. The cost to generalize is decreasing training score because of increased entropy and necessary complexity.

Boosted Decision Trees

I used the AdaBoost ensemble learner from scikit-learn on decision trees to create a boosted learner. After learning about boosting, I was fascinated that by repeating and intelligently adding weak learners, we can guarantee that the error will decrease or remain the same. I was especially excited for these hypotheses because it's the first algorithm in which both training error and testing error can continually decrease or remain the same without overfitting. The speed at which AdaBoost performed also surprised me because I thought boosting would take much longer but because it's based on decision trees, it ended learning quite quickly.



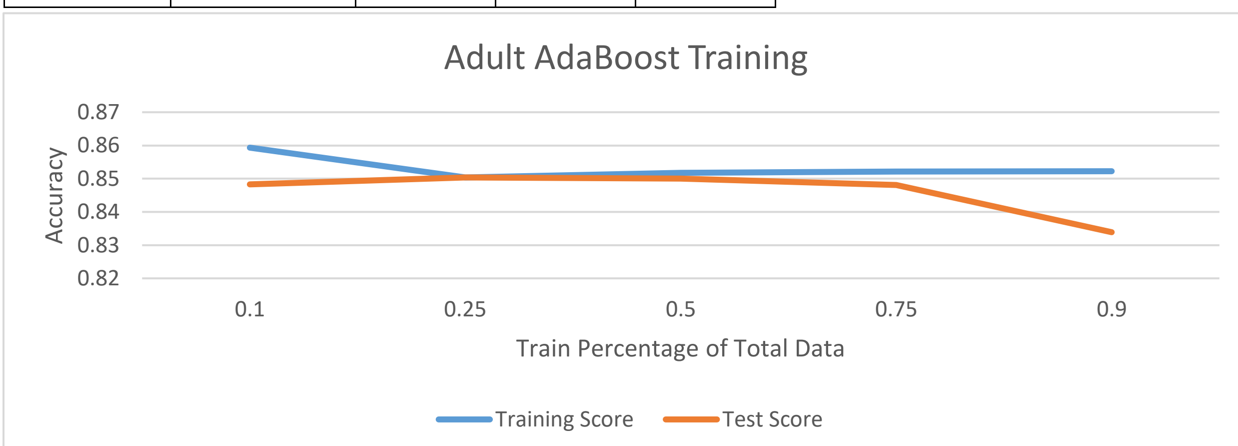


Instead of max height for pruning, I used max number of estimators for the initial change because that's all AdaBoost provided and it would also be stimulating to pursue a different avenue of pruning. Instead of restricting the height of decision trees themselves, I now restricted the maximum combination of weak learners for the boosted meta-ensemble AdaBoost learner. Despite the continual improvement, I chose 10 estimators for the adult dataset and 12 for the diabetes dataset because diminishing gains for increasing complexity.

Max Estimators	Training Score	Test Score	Train Time	Test Time
2	0.81476	0.810216	0.027018	0.003502
4	0.843498	0.837036	0.052038	0.005504
6	0.84477	0.840004	0.077553	0.008005
8	0.849245	0.84676	0.10357	0.010007
10	0.849201	0.850241	0.130088	0.012008
12	0.849465	0.85065	0.153604	0.014511
16	0.850825	0.850855	0.205138	0.019013
18	0.851351	0.851469	0.239163	0.021015
20	0.851483	0.8528	0.257174	0.023015
25	0.855344	0.854233	0.327722	0.02952
30	0.854335	0.853516	0.387268	0.034522
40	0.859951	0.857918	0.524857	0.04453

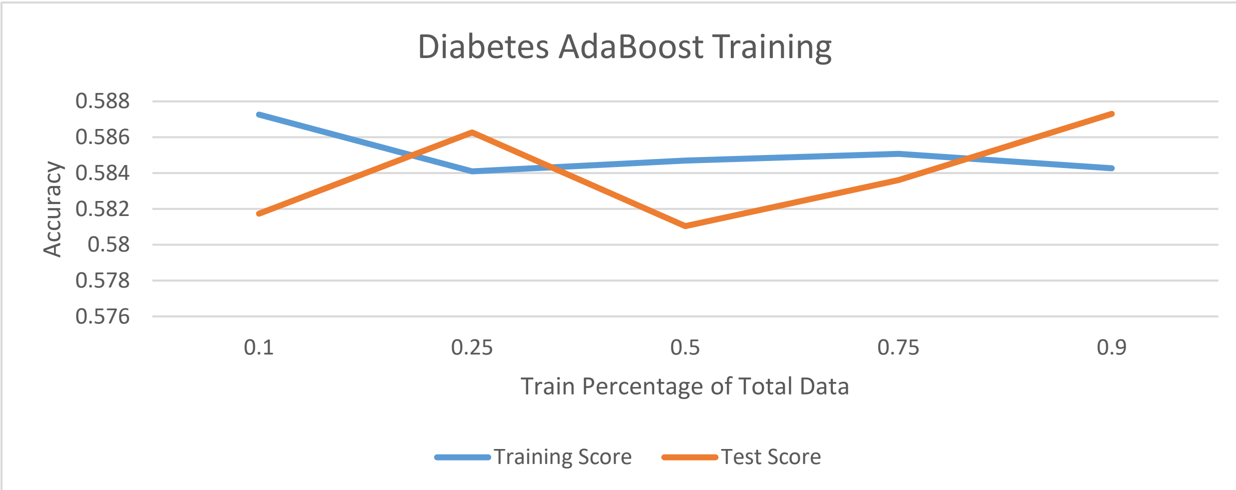
Max Estimators	Training Score	Test Score	Train Time	Test Time
2	0.559309899	0.555847	0.187172	0.019014
4	0.570287495	0.566525	0.352187	0.028519
6	0.579496322	0.574615	0.521354	0.039528
8	0.579440171	0.575139	0.690468	0.051035
10	0.582514459	0.575565	0.856081	0.061542
12	0.585715088	0.582149	1.026196	0.079054
16	0.587455781	0.582673	1.37193	0.094065
18	0.590305464	0.584671	1.533541	0.10761
20	0.59097928	0.585064	1.717665	0.116633
25	0.592579595	0.586865	2.10793	0.140596
30	0.592032119	0.586079	2.579254	0.202641
40	0.595485429	0.59037	3.37679	0.210643

With respect to the train/test split, the learning curves were erratic, suggesting that the success of a boosted learner is not dependent upon the size of the data but rather the luck in finding helpful weak learners. Boosted decision trees with pruning handedly beat out KNN and normal pruned decision trees and could continue improving with greater numbers of estimators.

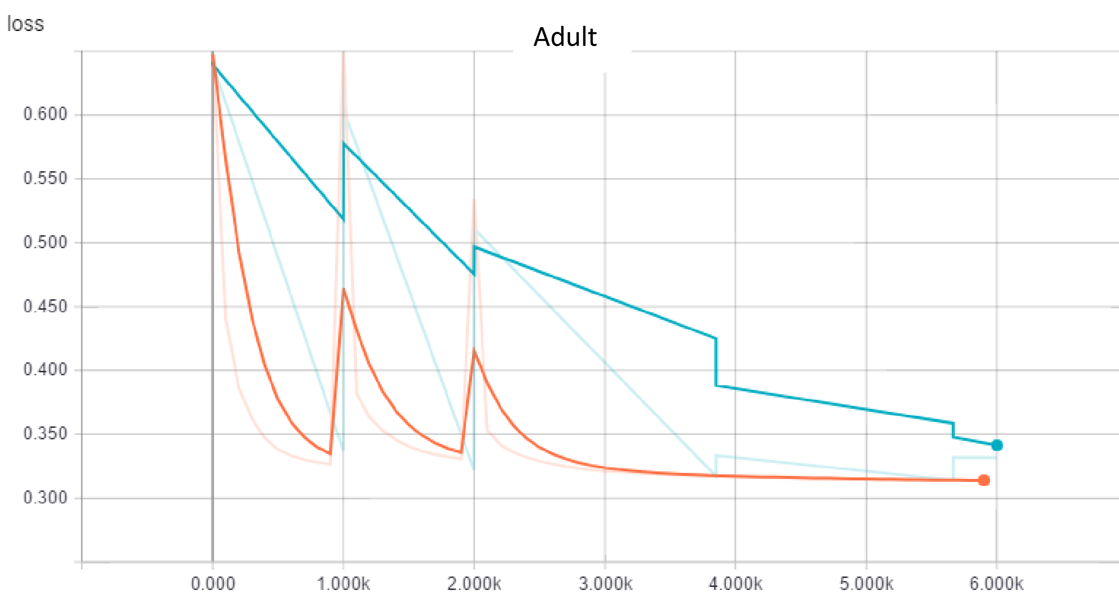


Neural Networks:

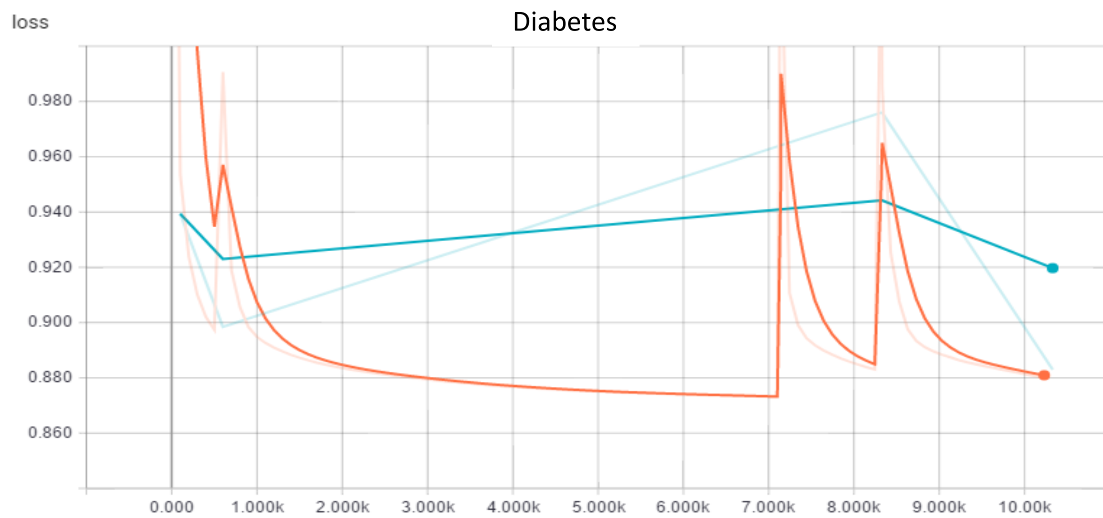
Neural networks and deep learning have been especially hot topics within the last few years so I was also ecstatic to apply them to real world datasets. Because neural networks are very computationally expensive and rely heavily on graphics cards for their parallel computation abilities, I wasn't able to run many different kinds of neural networks on my weak



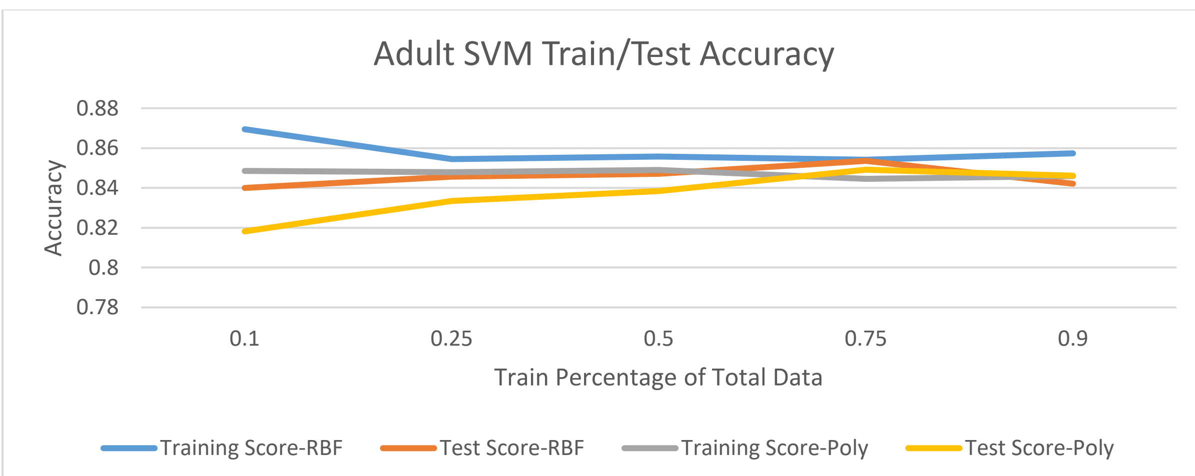
computer as it would take many many hours. I also didn't perform any hyperparameter optimization as it also would take too long on my computer. I used the default values from TensorFlow's high-level Learn API to create a simple neural network classifier with two hidden layers and with 10 nodes each and a ReLU activation function. I ran the adult dataset for 6000

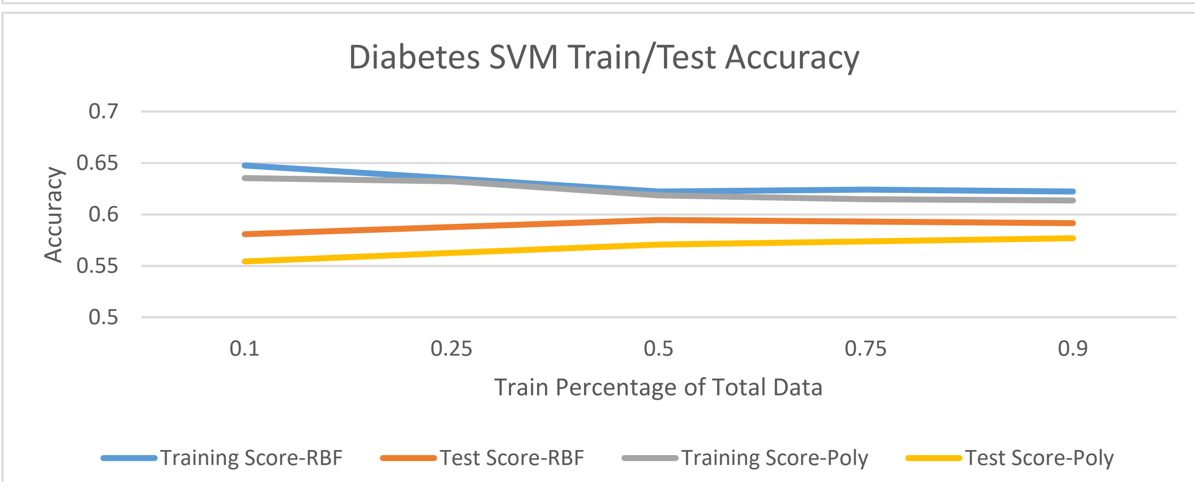


steps until it appeared to have plateaued and I ran the diabetes dataset for 10,000 steps. The adult neural network obtained an accuracy of 0.841744 and 0.587 for diabetes which are both less than AdaBoost but better than the other previous two. It also used an Adagrad optimizer and default learning rate of 0.1. I think that with better hyperparameter tuning and longer training time, the neural network's universal function approximation ability would enable it to outshine the others, though at the risk of overfitting. I found the seemingly random spikes in error to be an interesting quirk of neural networks as they try to find the optimal weights.



<u>Support Vector Machines</u>:





Adult/Kernel	Training Set Size	Training Score	Test Score	Train Time	Test Time
rbf	0.1	0.869471744	0.839993175	0.466423988	2.2841959
rbf	0.25	0.854545455	0.845583719	1.572182178	3.266815901
rbf	0.5	0.855712531	0.847060991	4.571765184	4.060728073
rbf	0.75	0.854135954	0.853580641	12.18820286	3.962385178
rbf	0.9	0.857289107	0.842186061	15.8130722	1.585304976
poly	0.1	0.848587224	0.818188023	0.361237049	0.875526905
poly	0.25	0.847911548	0.833422055	1.472840071	1.958359003
poly	0.5	0.849017199	0.838462011	5.982253075	1.770528078
poly	0.75	0.844553645	0.84915858	13.09980702	1.464980125
poly	0.9	0.846027846	0.846177464	20.62231112	0.955014944
Diabetes/Kernel	**Training Set Size**	**Training Score**	**Test Score**	**Train Time**	**Test Time**
rbf	0.1	0.647602201	0.580816683	8.308310032	56.69025302
rbf	0.25	0.634880704	0.58778906	69.91632295	128.4119751
rbf	0.5	0.622270699	0.594599375	331.6153278	174.970134
rbf	0.75	0.624194225	0.593113749	894.2965992	129.9199202
rbf	0.9	0.622258131	0.59152992	1507.624075	62.77286887
poly	0.1	0.635318396	0.554285402	7.209960938	38.73153806
poly	0.25	0.632286467	0.56273829	61.47342801	94.14712715
poly	0.5	0.618418725	0.570642454	299.7114019	121.3187411
poly	0.75	0.614786961	0.573893562	717.745086	105.1446509
poly	0.9	0.613599886	0.576889064	951.148952	51.32251406

Support vector machines took a long time to run like neural networks because their fit time is of quadratic complexity so it doesn't scale well. I decided to use a bagging classifier of support vector classifiers from scikit-learn instead because it runs faster in scikit. Fitting still took so long that I used a cluster of computers to compute the results instead of my own. I tested the polynomial and radial basis functions on the two datasets and returned results that were slightly better or on par with AdaBoost and neural networks. The polynomial kernel performed somewhat worse than the radial basis function. Over different test/train splits, test accuracy increased while training accuracy decreased with increasing training set sizes. The quickly increasing train times result from having to perform the quadratic programming operations over all the vectors to determine which support vectors are needed for the hyperplanes. Despite decent performance, the long training time makes it difficult to be practically applied as it will take too long.