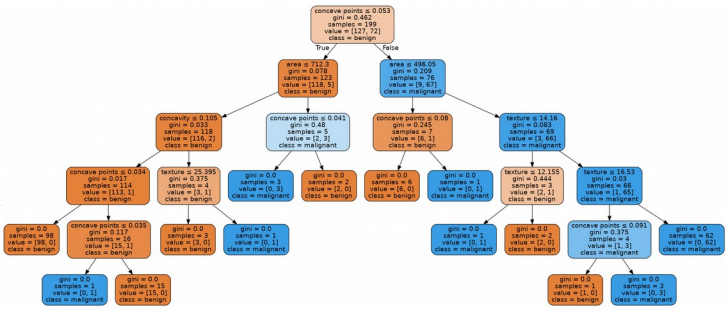
DECISION TREES

A great example of how machine learning can aid us in just about every facet of our lives is demonstrated when applying machine learning techniques to aid with breast cancer diagnoses. I pulled this data set on breast cancer diagnosis from the UCI Machine Learning Repository here (creators: W.H. Wolberg, W.N. Street, O.L. Mangasarian). This is a very good data set for decision tree classification as the target attribute is binary, the cancer can only be benign or malignant. The central question is whether or not we can diagnose future cases of breast cancer based on this data and if so, could it in practice stand in as a better and/or less expensive test over others?

The data set in question describes the various features of a cell nuclei pulled from the breast tissue via fine needle aspiration, where the features are computed from a digitized image of the cell nuclei. Some of these features include the cell radius, texture, perimeter, etc. Although the data set includes some other points such as the error on each computed feature and a “worst estimate”, for the purpose of this post I will not be using those and I will only use the computed feature values to generate the decision trees.

I have used the scikit-learn machine learning package available for python to generate decision trees for this problem. The following figure is an example of a decision tree that is generated using the decision tree algorithm in scikit-learn. This algorithm uses the GINI index to decide how to split attributes.



Immediately we can see some nice features of the tree. It’s not a large tree, only 6 nodes in depth, and also it is symmetric. Unfortunately, scikit-learn does not support pruning at this time, but I will be demonstrating boosting in a future post. So how well is the learned algorithm able to classify instances outside of the training set? In the generated decision tree, we can see that each instance in the test set is classified exactly correctly by the learned algorithm, suggesting that it may be overfit and unable to generalize beyond the training set.

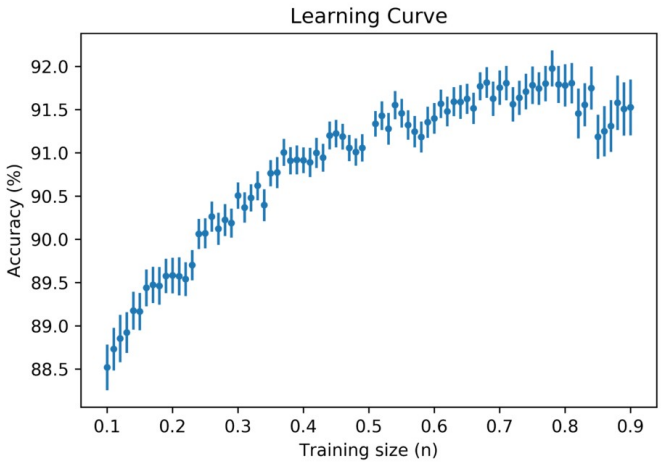
As it turns out, this is not the case and the learned algorithm performs very nicely in classifying new instances. For example, when trained using 65% of the available data (holding the other 35% as the validation set), the algorithm is able to correctly classify new instances about 90% of the time, as shown below.

	precision	recall	f1-score	support
benign	0.92	0.95	0.93	230
malignant	0.91	0.86	0.88	140
avg / total	0.91	0.91	0.91	370

Though perhaps this is not good enough for a medical setting, a result like this would not be bad at all for problems outside of medicine. However, it could be the case that the training set was not large

enough, maybe it is possible to see better results using a larger training set? To this end, I have used cross validation to determine whether or not a larger training set would make any difference. In short, I used many possible training set sizes and ran the decision tree classifier 100 times for each training set size (each time the data is randomly split), extracting the mean accuracy and standard error for each possible training set size. This way, the measured accuracy is dependent only on the training set size since the measured accuracy will not be biased by the ordering of the data always being the same each time the classifier is run.

The resulting graph is the learning curve, shown below.



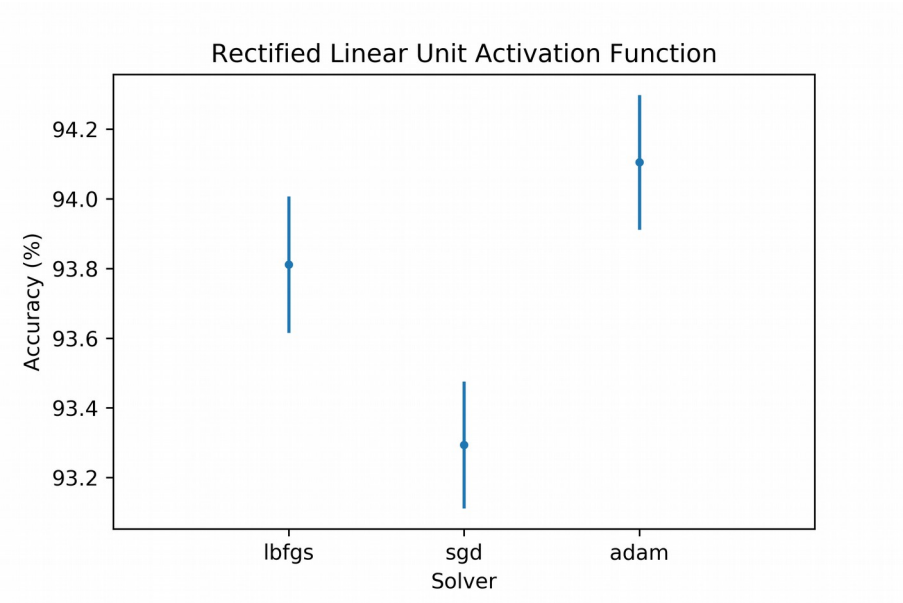
Clearly, there is an improvement in accuracy as the training set becomes larger, but it seems like it may be plateauing or even becoming worse with more data. The dip in accuracy at around n=470, which is about 85% of the total data, could be evidence of overfitting, either that or there is just not enough data left in the validation set to get a good measurement of the accuracy in those regions. To solve this definitely, I would require a larger data set but unfortunately the data set is quite small and I do not know where I can find more data like this. Still, even for the smaller training set sizes, the accuracy isn't too much worse than the best results.

NEURAL NETWORKS

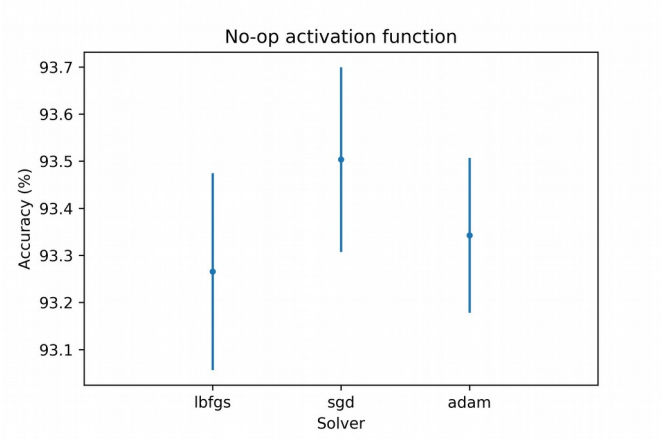
I have used the scikit-learn library in python to generate and train the neural network. Although the scikit-learn package is not so great for decision tree learning (because of the lack of support for pruning), scikit-learn does support many options for artificial neural networks. I was interested to see which options and settings will produce the best trained neural network for this problem.

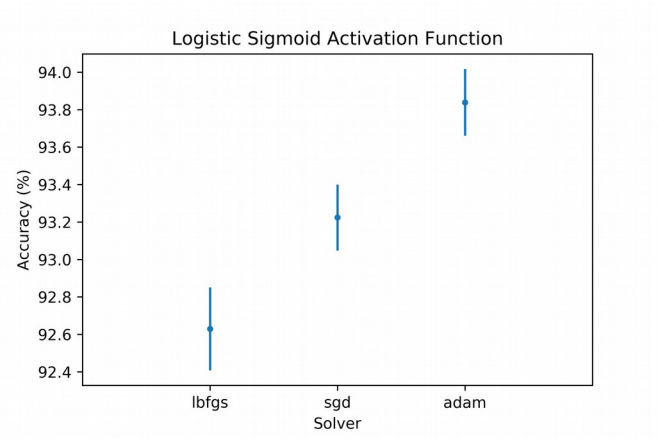
First, I wanted to find out which of the three solvers worked best with the data. There are three solvers available for neural networks in scikit-learn, a typical stochastic gradient descent (sgd) solver, the Adam solver (more info here), and a limited memory BFGS solver (lbfgs). For each solver I used a neural network with a single hidden layer of 10 nodes, the default rectified linear unit activation function ($f(x) = \max(0,x)$) for each node, and the maximum number of iterations to 10,000.

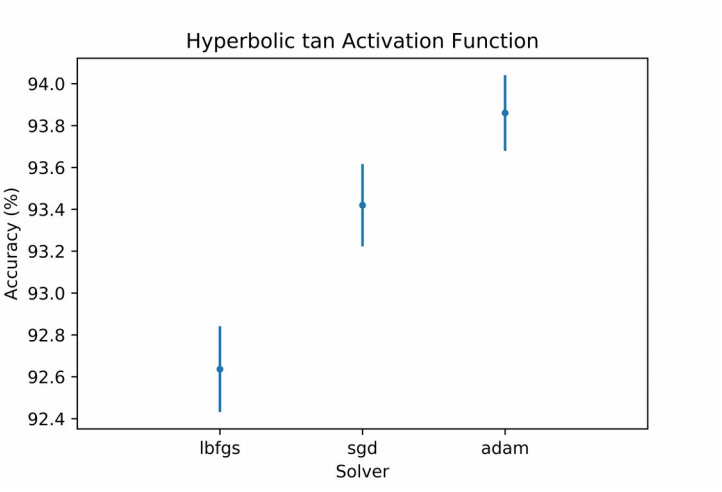
The rest of the settings, including learning rate, momentum, etc, were unchanged from default. I trained the neural network 100 times for each solver, each time randomly splitting the data between the test set and validation set so that the results are not biased by the ordering of the data always being the same. The trained neural networks are then tested with the validation set and the average accuracy and standard error on the accuracy is calculated. The results are shown in the following figure.



Evidently, the Adam solver performs the best, at least when using the rectified linear unit activation function for the nodes. Of course, I want the best possible results so I performed the same analysis using all the available activation functions. The three remaining functions are the hyperbolic tan activation function (f(x) = tanh(x)), the logistic sigmoid activation function (f(x) = 1/(1+exp(-x))), and the no-op activation function (f(x)=x). The results are shown in the following figures:

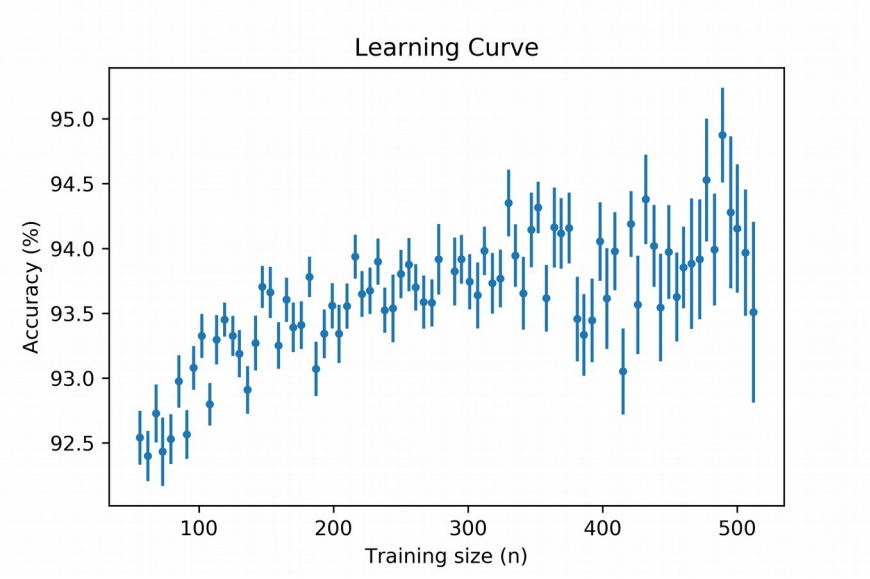






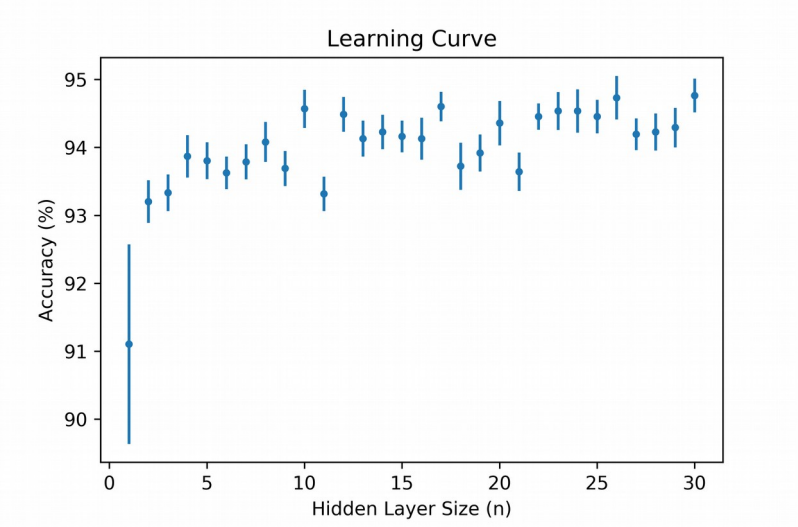
It looks like the Adam solver with the rectified linear unit activation function still produces the best results, with an average accuracy of about 94.1 +/- 0.2%. It may not be exactly the case that these are the best possible results since the stochastic gradient descent solver can vary the momentum as well as the learning rate. Although I have not tested systematically, I tried varying the momentum and learning rate by hand a bit to see if I could produce any better results and as it turns out I was able to get good results with stochastic gradient descent on par with Adam, but since the results were not significantly better than Adam, I will just be continuing with the Adam solver and the rectified linear unit activation function.

In the previous analysis, I used a training set size of three quarters of the total data set to generate the figures. I would like to see what difference it makes when varying the training set size. Therefore, I performed the same kind of analysis as before, where I trained and tested the neural network a number of times and averaged the results of each trained neural network, except this time I have varied the size of the training set. The results are shown below.



So there is clearly some increase in accuracy as the training size increases, however when the training set size is greater than 420 the accuracy is not consistent (the error bars are relatively larger) as compared to the smaller training set sizes where the error bars are relatively smaller. This may be due the fact that the data set has only 569 data points, so if the training set size is too large then the test set is too small and is not able to consistently test the accuracy of the trained neural network. Looking at the graph, I have chosen the training set size of n=364 to continue with the analysis as this value appears to produce consistently high accuracy results as compared to the other points.

Finally, I want to find an optimal size of the hidden layer so I re-ran the same kind of analysis as previously except leaving the training set size unchanged from n=364 and instead varying the hidden layer size. The results are shown to the right



Evidently, having more than 5 nodes in the hidden layer doesn't offer much benefit but since I wanted the best possible results I chose to use 10 nodes as it appears to be the smallest number of nodes with the best accuracy. As it turns out, I have been using 10 nodes the entire time so I have simply confirmed that 10 nodes was an optimal number. Printed below are the precision, recall, and f1-score of the trained neural network.

	precision	recall	f1-score	support
benign	0.96	0.96	0.96	128
malignant	0.94	0.94	0.94	77
avg / total	0.95	0.95	0.95	205

Evidently the results are very good. Compared to the use of decision tree, neural networks improve classification accuracy by 4-5%. On the other hand, training took significantly longer. For example, to generate the learning curve for decision tree learning happened in an instant, generating the learning curve for the neural network would take up to 6-8 minutes. At the same time, though the learning curve may have taken longer to generate, training the neural network just once was instantaneous so I am not too worried about the performance.