Sean Chua
Schua8
903191859

CS 4641 - Assignment 1

DATASETS

Breast Cancer Dataset - This is a dataset of 569 instances that describe characteristics of breast cancer data, provided by researchers at the University of Wisconsin. This dataset contains 32 attributes, which include ID, diagnosis, and ten features that were calculated for each breast cancer cell nucleus, such as radius, texture, concavity, symmetry, and more. This dataset also includes the mean, standard error, and largest (mean of the three largest values) of these features, which results in 30 different features. Therefore, even though the breast cancer dataset seems to have a high degree of dimensionality, that is mitigated by the strong relationships between our numerous features. Along with these 32 features, there are 2 class types of “Benign” and “Malignant”, which signify the doctor’s diagnosis. In addition, this dataset has 357 benign diagnoses and 212 malignant diagnoses, which make it imbalanced. As a result, I will be using the F1 score as the final metric to evaluate the models.

I chose this dataset because creating a machine learning algorithm that could diagnose breast cancer as well or even better than a doctor could would have vast implications for fighting breast cancer. This is because using a machine learning algorithm, as opposed to hiring a doctor to do a diagnosis, is much less financially expensive and could lower the barrier for people to check on their health. This could help diagnose early forms of breast cancer, save patients’ lives, and save them a lot of financial and physical problems. Lastly, I wanted to work on a significantly smaller dataset than the other one that I chose in order to see how training size and complexity can affect learners.

Handwritten Digits Dataset - This dataset has 5620 instances of handwritten digits (ranging from 0 to 9) that were extracted by researchers from the Bogazici University using preprocessing programs to yield normalized bitmaps of handwritten digits from a preprinted form. A total of 43 people contributed their writing to this dataset, 30 of which were included in the training set and 13 of which to the test set. These 32x32 bitmaps were divided into 4x4 blocks (which don’t overlap), and the number of “on” pixels are counted in each block. This yields an 8x8 input matrix in which each element is an integer in range [0,16]. As a result, there are only 64 input attributes and one class attributes, which is a great reduction of dimensionality. This dataset is also much more complex and larger than the breast cancer dataset, so it will be interesting to see the effect on testing error as a function of training size for the numerous algorithms.

I found this dataset particularly interesting because it represents something that humans find intuitive but is much more difficult for a computer to classify. Image classification has also become one of the most interesting applications of machine learning nowadays, as technologists have been able to implement it in facial recognition to unlock phones, handwriting recognition, and more. As I was looking through the description of the handwritten digits

dataset, I found it intriguing to see how much clever preprocessing it took to create such a dataset so that a machine could learn how to recognize handwritten digits. Handwritten digits are somewhat standardized across cultures, but I would love to explore this space further with an analysis on other forms of handwriting (not just within English).

ALGORITHM RUNS

Throughout my analysis, the training set was arranged to contain 70% of the data, while the test set contained the remaining 30%, in order to ensure valid comparisons among all five of the algorithms. Each model also had a five-fold cross validation performed on the training set through GridSearch in order to find the optimal combination of hyperparameters (e.g. how much to prune the decision tree, the number of hidden layers in the neural network, etc.). Each algorithm had at least two hyperparameters that were being changed to experiment with multiple combinations. Using a five-fold cross validation on each of the algorithms helped ensure that these algorithms are not relying on information from the testing set and helped choose the right hyperparameters that would be able to generalize well, but minimize overfitting.

Once the results from the GridSearch were finished, the final model with the tuned hyperparameters was tested against the test set to see its accuracy.

Pruned Decision Trees

Decision tree learning is a highly intuitive supervised learning algorithm that attempts to take a dataset and infer a number of rules or questions. Similar to the game “20 Questions”, each question and answer corresponds to if-then constructs, wherein the root node is an initial question, and the answer allows one to progress further down the tree, until one finally arrives at an answer. The algorithm is fairly simple to understand, but can be unstable, creating completely different trees for the most minute changes to the data. Furthermore, decision trees can possibly create overly complex trees with so many nodes that it leads to overfitting. For this analysis, we used scikit’s DecisionTreeClassifier to implement this type of learning.

Pruning: Pruning in machine learning is a technique for decision trees that involves removing sections of the tree that don’t provide much information to classifying instances. This reduces the complexity of the final classifier, and therefore improves predictive accuracy by reducing the chance of overfitting. The pre-pruning method that I decided on was to set a minimum number of samples in a node before splitting. This would prevent the tree from creating too many nodes and limited the complexity of the decision tree. Too high of a minimum could lead to a high testing error, but too low of a minimum could lead to an overly complex decision tree with the potential to overfit.

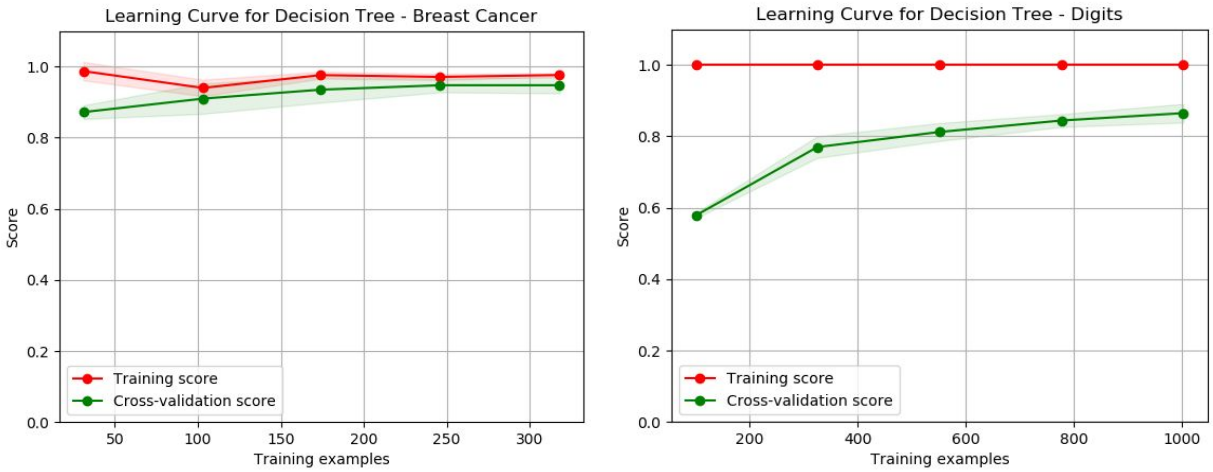
Max Depth: Another hyperparameter that I tweaked to control the complexity of the decision tree was the maximum depth of the tree. If there is nothing set, then the nodes would expand until all leaves were pure or until all leaves were less than the minimum number of samples in a node before splitting (specified above in “Pruning”).

Sean Chua
Schua8
903191859

For both datasets, I ran a GridSearch multiple times using the min_sample_split and maximum depth while using entropy as the criterion to measure the quality of a split. This means that the decision tree would be making splits based on the highest information gain. For the breast cancer dataset, different sets of hyperparameters performed optimally each time and produced very similar levels of accuracy throughout each iteration. I re-ran the model multiple times and chose the pair of hyperparameters that had the highest accuracy the most times. My final decision tree model for the breast cancer dataset had a maximum depth of 6 and a minimum sample split of size 18 with a weighted F1 accuracy of 91%. However, as I ran through multiple examples with the dataset, I found numerous other pairs of hyperparameters that led to similar levels of accuracy. This leads me to believe that this dataset is relatively simple as the algorithm could prune relatively aggressively (i.e. it had a relatively high minimum of 18 samples to split on a node) without sacrificing any accuracy.

As I mentioned before, the digits dataset is a bit more complex and thus requires a more complex decision tree in order to effectively classify it. My final decision tree model for the handwritten digits dataset had a maximum depth of 15 and a min_sample_split value of 2 with an overall accuracy of 87%. As the model was run through more iterations, maximum depth didn't seem to have as much of an impact, as it would move up or down a few levels, while the min_sample_split value would always stay optimal at 2. Regardless of the maximum depth, the accuracy did not change as well.

Below are the two finalized models plotted as a function of training size to see how well the models learned with more training examples.



In both of the finalized models, cross validation score increased as the training size increased. This makes sense as the model is able to generalize better with more and more of the data. However, the breast cancer decision tree seems to perform relatively well with few training examples and only makes slight improvements given a few more. In addition, after a certain

point its cross validation score seems to plateau, which again points to the notion that the breast cancer dataset is relatively simple. This is interesting because it seems that the decision tree doesn't require much data to perform well, as opposed to the digits dataset. The digits dataset seemed to have a much steeper learning curve and thus required more data, as a result of being much more complex. Even at the latter end of the graph for the digits dataset decision tree, it's still improving its accuracy, so it's possible that passing even more data could increase its accuracy further.

It is interesting to note that the decision tree algorithm for the breast cancer dataset was much more accurate than for the digits dataset. This can be explained by the complexity of each dataset's features and classes. Because the breast cancer dataset is a binary classification model, even a simple decision tree is able to separate the dataset into its benign and malignant classes. On the other hand, the digits dataset has ten classes and thus needs a much more complex decision tree to model it accurately. The decision criteria for the breast cancer also intuitively makes more sense, whereas pixel density doesn't have as much tangible decision making.

If I were to improve this model, I would consider running more combinations with different hyperparameters, in addition to the ones I have already chosen. This would include the minimum samples per leaf, which specifies the minimum number of samples required to be at a leaf node, as well as max_features, which represents the number of features to consider when looking for the best split. These could improve the algorithm's performance, especially for the digits dataset, because each split could be a *combination* of factors that my algorithm is not currently considering.

Boosted Decision Tree

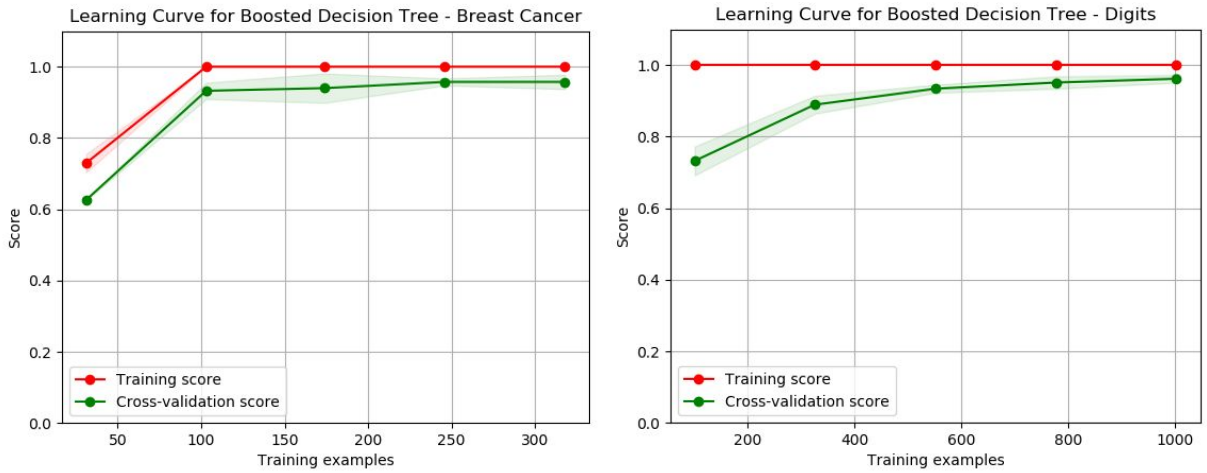
Now we will move on to experiment with boosted decision trees, in particular with scikit's GradientBoostingClassifier. At its core, boosting is under the same umbrella of a concept called ensemble learning, in which an algorithm takes simple rules and combines them to create a complex rule that works better. This is done by calculating the accuracy of each simple rule and taking the weighted average to return a final amalgamation of an algorithm. Because of this weighted average, boosting is able to create more complex models than the models that it is using.

Looking back at our decision tree algorithms, the breast cancer dataset was accurately modeled, but the algorithm's performance on the digits dataset was not optimal, as a result of higher complexity. However, by combining multiple weak decision trees, we can actually achieve higher predictive performance. After each weak learner is added, the weights are adjusted and tuned allowing for future weak learners to focus on the more difficult problems and "boost" the accuracy on these areas that previous weak learners misclassified. Similar to the previous decision trees, I also added some pruning with the same min_samples_split feature. In

addition, I changed the number of estimators to use in the model, which seemed to have the most impact. Because this boosting algorithm is an average of all these weak estimators (i.e. underfitting and overfitting decision trees), the boosting model should be able to maintain high levels of accuracy and not overfit, even with a high number of estimators.

The optimal hyperparameters for the breast cancer dataset was 200 estimators and 34 for the min_sample_split, with a weighted F1 accuracy of 97%. For the digits dataset, the optimal combination had 100 estimators and min_sample_split of 22, with an overall accuracy of 95%. Both of these datasets could aggressively prune with such high min_sample_splits only because they were being used as weak learners in the boosting algorithm. This matches my expectation as boosted trees are better at generalizing, whereas individual decision trees have a higher chance to overfit.

Below are the two finalized models plotted as a function of training size to see how well the models learned with more training examples.



In both of the finalized models, accuracy increased as the training size increased. As with the decision tree algorithm, this makes sense as the learner is getting more and more training examples to generalize with. It's interesting that the breast cancer dataset seems to perform worse with very few examples, but it makes sense as we are using weak learners. The digits dataset also performs significantly better with boosting and experiences the same steep, positive learning curve as the number of training examples increases.

For the boosted decision tree, I would improve this algorithm by altering more of the hyperparameters within each weak learner, including max_features and min_samples_leaf. I would also want to experiment with different learning rates, as this will determine the impact of each tree in the final outcome.

Sean Chua
Schua8
903191859

Support Vector Machines

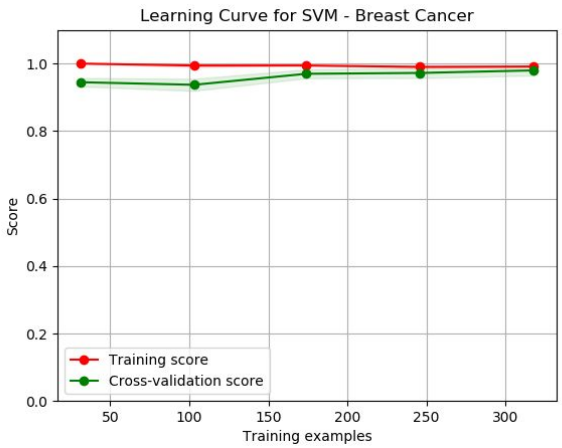
Support Vector Machines are able to categorize data by finding the best fit parameters for a specifically chosen kernel function, which will try to map various spaces of the data as belonging to a certain category. It separates the data by creating a hyperplane that maximizes the margin between all classes to avoid generalization to classify these data points. Similar to the implementation for neural networks, the datasets were normalized to both decrease the runtime of the algorithm and to avoid giving some features more importance than others.

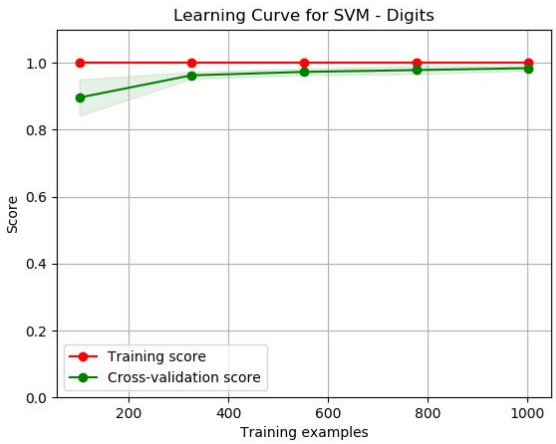
C: Penalty parameter C of the error term. This tells the SVM optimization how much you want to avoid misclassifying each training example. For larger values of C, the algorithm will chose a smaller-margin hyperplane if that hyperplane is more accurate.

Kernel: Similar to the distance metric in KNN, the kernel function is also a representation of similarity. This is where we can inject any domain knowledge we may have.

The optimal combination of hyperparameters for the SVM of the breast cancer dataset was C = 1 and a linear kernel of 95%. For the digits dataset, the best combination was C = 10 and the radial basis function (RBF) kernel, with an overall accuracy of 98%. Throughout multiple iterations, both datasets did not have any variation in the optimal combination of hyperparameters. The digits dataset most likely had C = 10 because setting it at such a high value prioritizes more accurate hyperplanes (although it does put the model at a higher risk of overfitting). The breast cancer dataset is able to be classified with a simple linear kernel, but the digits dataset works best with the RBF kernel. Because of the kernel effect, we can transform a given space into some other (usually very high dimensional) space and model the digits dataset very well with this kernel.

Below are the two finalized models plotted as a function of training size to see how well the models learned with more training examples.





Sean Chua
Schua8
903191859

In the finalized model for the breast cancer dataset, the accuracy seemed to stay around the same level, which is similar to other algorithms. The model for the digits dataset increased with more training examples, as a result of the higher complexity of the dataset.

If I were to improve this algorithm, some hyperparameters that I would try tweaking would include gamma and degrees for the polynomial kernel. Gamma is a parameter for non linear hyperplanes, so it would be interesting to see its effect on the digits dataset. Degrees for the polynomial could make the model a bit more complex and may even increase the accuracy for both datasets a bit.

K-Nearest Neighbors

We now move on to experiment with the K-Nearest Neighbor learning algorithm, a comparatively lazy algorithm compared to the others discussed in this paper. However, this algorithm is one of the best performing algorithms in a number of problem domains for its simplicity and even does well with the breast cancer dataset and the digits dataset. The algorithm makes predictions for a point by looking at the *k* nearest data points and making a decision on what to predict based off of those k nearest points. The notion of the "nearest" points is based on their similarity and is given by a distance metric. The potential for tuning, then, lies in the hyperparameters of the value of *K*, as well as the method by which we are assuming similarity - the distance metric. We will be using scikit's KNeighborsClassifier to implement this algorithm.

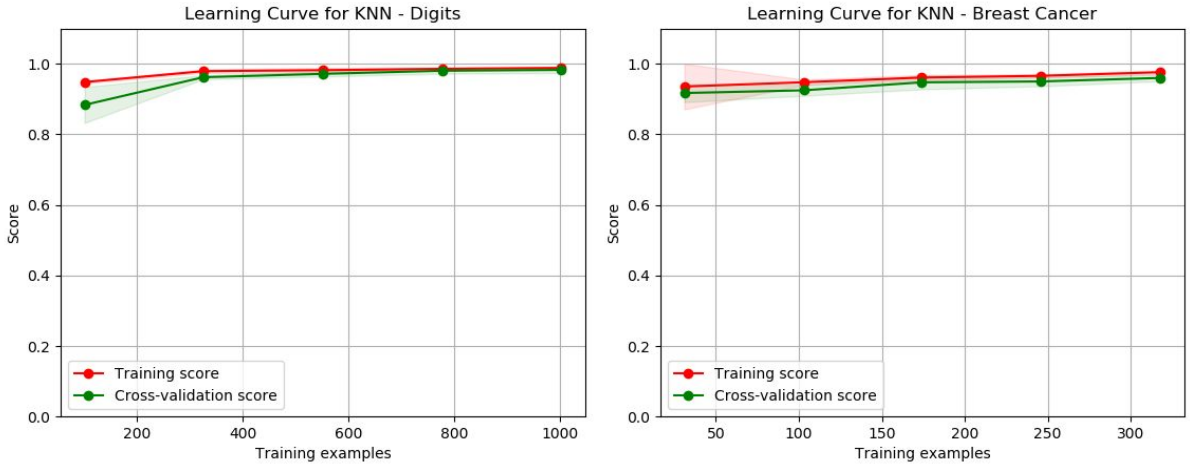
N_neighbors: the K number of neighbors we are looking at.

Distance Metric: the distance metric to use for the algorithm. This is one of the most important parts of the algorithm in which we should insert any domain knowledge. The metrics that we will be testing include: Euclidean, Manhattan, Minkowski, and Canberra. The Canberra distance is particularly interesting because it is often used to sort plants and animals into groups that are more closely or distantly related to each other. I'm curious to see how this fares with the breast cancer dataset.

The optimal hyperparameters for the breast cancer dataset was 7 for number of neighbors and Canberra distance for the similarity metric, with a weighted F1 accuracy of 96%. For the digits dataset, the optimal combination was 3 neighbors and Euclidean distance for the similarity metric, with an overall accuracy of 99%. Across multiple iterations for the breast cancer dataset, 7 nearest neighbors and using the Canberra distance seemed to always perform the best. This is a prime example of where domain knowledge can come into good use; Canberra distance is primarily used within the biology field, and its application here made the algorithm even more accurate. The digits dataset also did not vary and had the same optimal combination of parameters throughout multiple iterations.

Sean Chua
Schua8
903191859

Below are the two finalized models plotted as a function of training size to see how well the models learned with more training examples.



In both of the finalized models, cross validation score increased very little as the training size increased. For the breast cancer dataset, accuracy is almost identical throughout added training examples, while the digits dataset's accuracy increased marginally in the beginning and plateaued later on. In comparison to all of the other algorithms, KNN doesn't seem to need as much data to perform well.

Another thing to note was that this model took much less time than other models because it is less computationally intense.

If I were to improve this model, I would consider running more combinations with different hyperparameters, in addition to the ones I have already chosen. This would include hyperparameters such as the *weights*, which is the weight function used in prediction (includes functions such as uniform, distance, and more). I was also thoroughly pleased with how well the Canberra distance metric worked and would love to see if there were other distance metrics that would give even better senses of similarity, as that would further optimize the model.

Neural Networks

Neural networks represent a very powerful supervised learning algorithm, for which we used scikit's multilayer perceptron (MLP) classifier, a feedforward neural network model. In this model, the edges are either reinforced or reduced by each piece of data that is added to be learned from. There is an input layer, an output layer, and a varying number of hidden layers within. For this analysis, this will be one of the hyperparameters.

Hidden Layer Sizes: Specifies the number of hidden layers and number of nodes within each one.

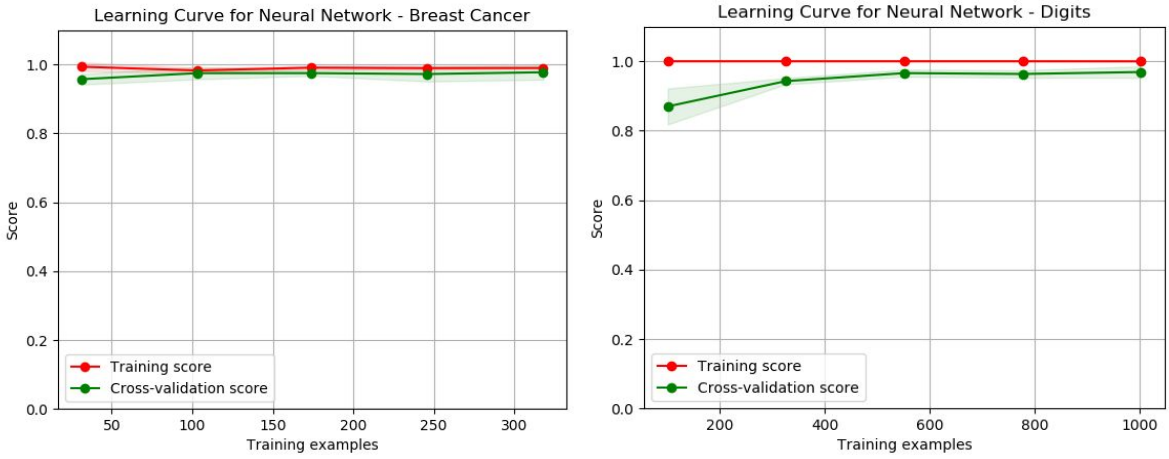
Sean Chua
Schua8
903191859

Activation: Each perceptron has an activation function that calculates a number to be evaluated against a threshold. The ones that will be tested include: Identity, Logistic (Sigmoid function), Tanh, and Relu.

At first, this algorithm was run without normalizing the breast cancer data, and it output graphs with very inconsistent patterns. This is because normalization helps to eliminate scale factors that might exist between the variables in my data. For example, if one of my features was on the order of 100s, then it would have much more (unwanted) impact than a feature on the order of 10s.

The optimal hyperparameters for the breast cancer set was 20 nodes for one hidden layer and tanh for the activation function, with a weighted F1 accuracy of 97%. For the digits dataset, the optimal combination was 80 nodes for one hidden layer and relu for the activation function, with an overall accuracy of 98%. However, the breast cancer dataset had a wide variety of combinations that were optimal throughout multiple iterations. Multiple activation functions such as the identity and logistic function, as well as numerous layer sizes came up. This could possibly be due to the simplicity of the dataset. For the digits dataset, the activation function was more set with relu and required a lot more nodes within the hidden layer (which again can be explained by the higher complexity of this dataset). Relu normally serves as a good choice for the activation function as some of the other ones (tanh, sigmoid) are more prone to vanishing gradient problems, which can make learning harder.

Below are the two finalized models plotted as a function of training size to see how well the models learned with more training examples.



In both of these finalized models, the accuracy increased as training size increased. As with most other models, the neural network learned relatively quickly with the breast cancer dataset and plateaued with very few training examples (due to simplicity of the dataset), and the opposite was true of the digits dataset. Just by comparing the graphs for the digits dataset, SVMs seem to need much more data than KNN in order to perform as well.

Sean Chua
Schua8
903191859

If I were to improve this algorithm, I would alter a few more of the hyperparameters, such as the number of epochs and different learning rate policies. An epoch is defined as a full pass of the data set; too few epochs don't give your network enough time to learn good parameters, and too many can lead to potential for overfitting. With a learning rate policy, the policy will change the learning rate over time, achieving better results since the learning rate can "slow down" to find closer local minima for convergence.

CONCLUSION

Upon closer examination of all the tested algorithms, it seems that the neural network works best for the breast cancer dataset with an F1 accuracy of 98%, while KNN is the most accurate for the handwritten digits dataset, with an overall accuracy of 99%. Here, I am defining the "best" algorithm as the one with the highest accuracy on the testing set. However, almost all of the algorithms were relatively similar in accuracy, so it may be useful to evaluate some of the benefits and disadvantages of each.

In regards to decision trees, the main advantage that they offer is interpretability. For any problem that needs comprehensibility, such as with the breast cancer data, it may be more useful. They contain knowledge that can be expressed in a readable form, while SVM and neural networks are somewhat like black boxes. However, it does lack in the complexity that the aforementioned models afford. Especially with neural networks and SVMs, they usually outperform nearly every other algorithm but can't offer any explanation as to how they did it. This is why a lot of banks don't use a Neural Network to predict whether a person is creditworthy - they need to have explanations (e.g. for a customer not getting a loan). KNN is a bit more intuitive and simple to understand than these algorithms and performs better than the pruned decision tree(96% for KNN vs 91% for Decision Trees). One disadvantage of the algorithm is the Curse of Dimensionality, as it normally works well with small numbers of input variables, but as the number of variables grow, KNN struggles to predict the output of a new data point.

Considering all of these factors, I would recommend the use of the most accurate algorithms provided there is no need for an explanation or understanding of the results. For the breast cancer dataset, it may be wise to consider KNN and decision trees, as they offer an intuitive form of understanding the results.

Training Times:

- KNN - < 1 minute for both
- SVM - ~1.5 minutes, 3 minutes for Digits Dataset
- Decision Trees - < 1 minute
- Neural Network - 3 minutes for both
- Boosted Decision Trees - ~2 minutes