Supervised Learning

Tiffany Xu

Classification Problems

1. **Letter Recognition** (referred to as Letters)
   The letter recognition dataset contains 20,000 instances of letters (A-Z, capitalized) with 16 attributes describing the physical properties of each letter. The attributes, such as x-box horizontal position of the box and y-ege mean edge count bottom to top, encapsulate the statistical moments and edge counts of each letter instance. The data was based on 20 different fonts, and each letter in a font was furthermore randomly distorted to produce the 20,000 instances. We will use the physical property attributes to identify which letter each instance is. This dataset was created by David J. Slate of the Odesta Corporation in 1991 and made available on the UCI Machine Learning Repository.

   I think this data set is interesting because as a kid, I had a random obsession with downloading cute fonts and trying to mimic other people's work with their unique typography. I used the website https://www.myfonts.com/WhatTheFont/ to identify which fonts they used so I could use them too. Many of the fonts used had to be bought, so I would use another website to enter the name of font and find free alternatives. At the time, I never thought about how these websites found the fonts, but now I realize they probably used machine learning to identify fonts and similar fonts. Additionally, when I travel to a different country with a foreign language, I often use Google Translate's photo-to-translation functionality to navigate my way around the foreign country. This feature definitely uses machine learning to analyze an image and identify the words in the image that can then be translated into another language.

2. **Adult Census Income** (referred to as Income)
   The adult census income dataset was derived from the 1994 U.S. Census bureau database by Ronny Kohavi and Barry Becker. The instances were chosen based on the conditions of ((AAGE>16) && (AGI>100) && (AFNLWGT>1)&& (HRSWK>0)) resulting in 32,561 instances. Some other attributes are sex, race, native country, occupation, education, relationship, etc. which will be used to predict if a person earns <= $50,000 or > $50,000 a year. This dataset was found on Kaggle.com.

   This data set is interesting especially in light of the information I've learned in sociology—that how successful people are is much more a factor of our environment and upbringing (e.g. culture, race, location) than our own specific traits (e.g. determined, kind). We often overestimate our personal traits and underestimate the outside affect. This dataset uses these "outside" effects (i.e. anything other than our own personality and traits) to determine if a person earns more or less than $50,000 a year. I would consider annual income to be one crude measurement of success, and this dataset can estimate if a person will be 'successful' or not based on these "outside" effects.

Overview

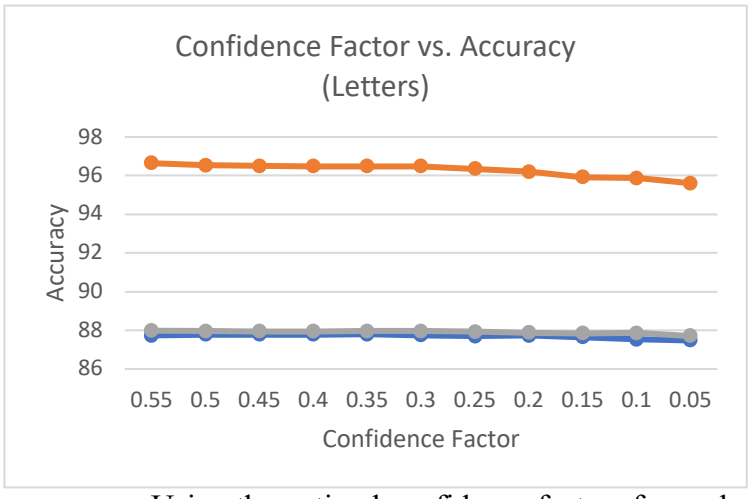
Both datasets were split 80% for training, and the remaining 20% for testing. Weka was used to run all five algorithms, and hyperparameters within each algorithm were systematically altered to record their corresponding effects on the accuracy on classification. The values of the hyperparameters versus the accuracy will be drawn on graphs, as well as the learning curve for the best configuration of hyperparameters for that algorithm and dataset. Accuracy was reported

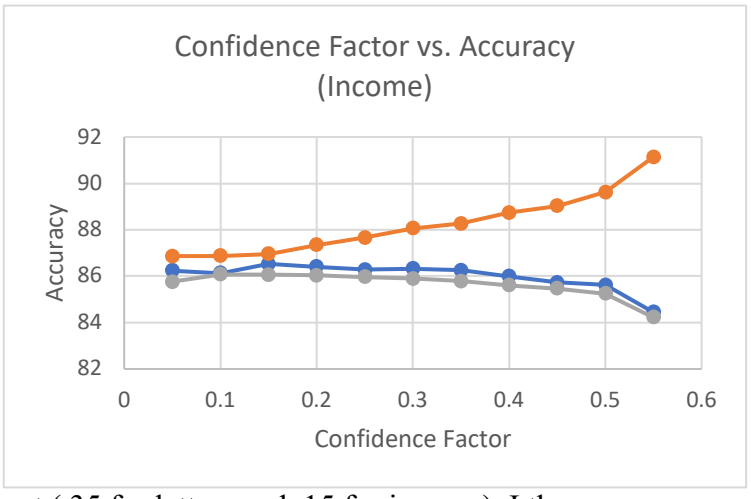
as opposed to metrics like precision, recall, or f1 score because the datasets used have a good distribution of instances into all classes. Additionally, the datasets don't have life-death consequences for incorrect classifications, so distinguishing between the errors (false positive or false negative) does not yield much information, so an overall accuracy measure is used.

Decision Tress

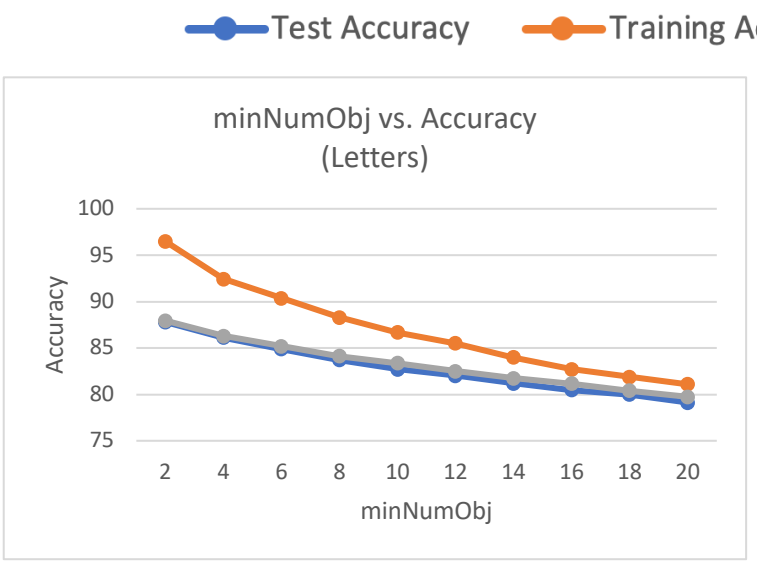
I ran the J48 algorithm on the datasets, which uses information gain to determine which attribute is best to split on. Before running the algorithm, I predicted that decision trees would work better for the income dataset, since some attributes seem more useful than others, such as occupation and education as opposed to relationship status. The letter dataset, on the other hand, has many physical trait descriptors that all probably matter somewhat equally. I altered the hyperparameters for pruning including confidence factor and minimum number of instances per leaf (minNumObj). The greater the confidence factor on your dataset, the surer you are that the data is representative of that specific problem space. That means, the higher the confidence factor, the less pruning, and vice versa. The minNumObj value tries to maximize the information gain and also causes pruning. If a large number is input for the minNumObj, then the current set of instances should be split up in way such that there is no trivial one kind classification, where each instance ends up having its own leaf. This parameter should also be greater for noisier datasets to avoid overfitting.

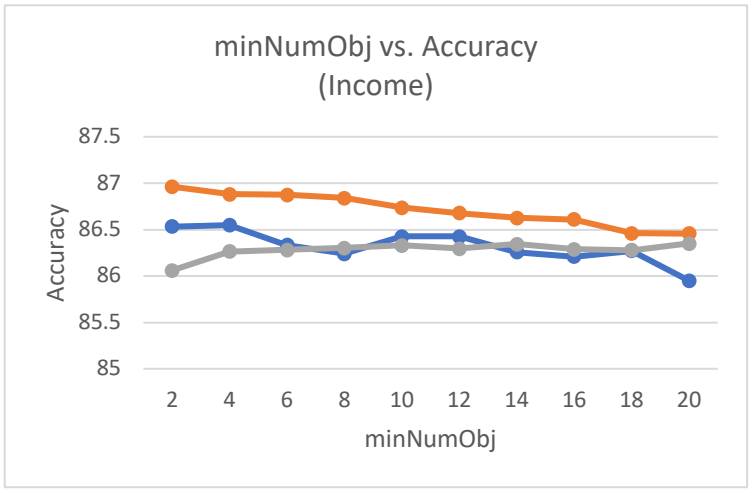
For letters, it seemed that varying the confidence factor did not have much effect on any of the test set, cross validation set, or the training set. For the income dataset, the training set accuracy increased with higher confidence factor while the cross validation and testing sets decreased in accuracy. This is a showing example of overfitting since allowing the tree grow in complexity due to the high confidence factor, while increases the training accuracy, loses some generalizability for unseen instances. There is a slight increase in accuracy, though, from 0.05 to .15.



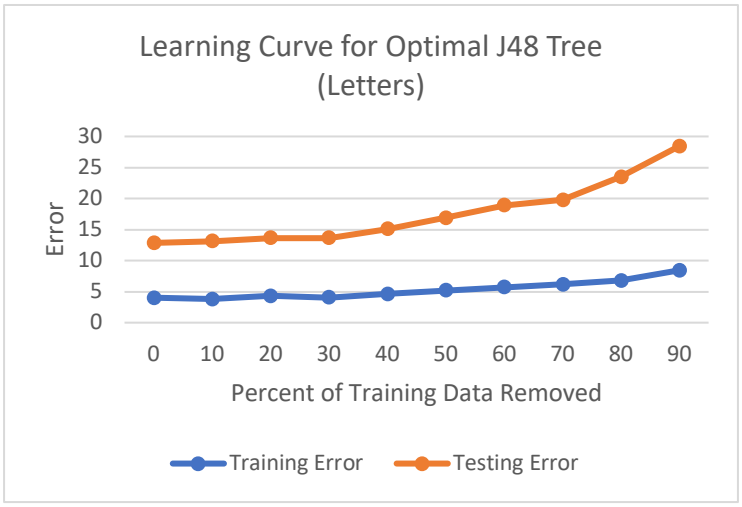


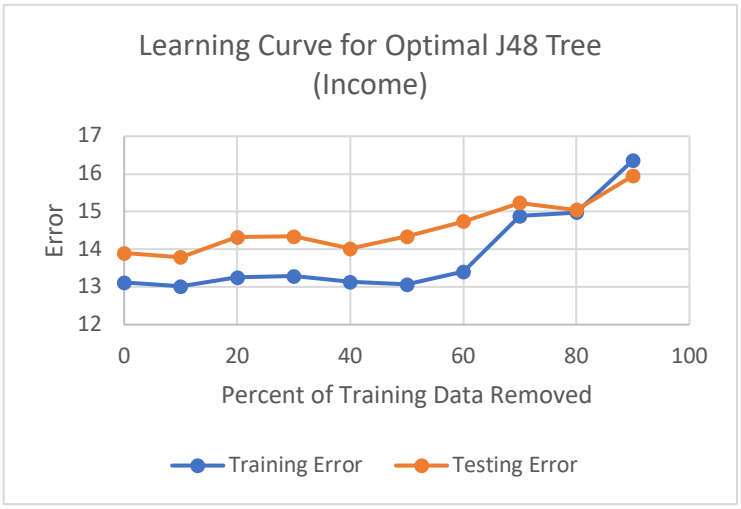
Using the optimal confidence factors for each dataset (.35 for letters and .15 for income), I then varied the minNumObj parameter. The letters dataset consistently decreased in accuracy as the minNumObj increased. It seems that a certain level of tree complexity is needed to accurately classify the letters, which makes sense since there are 26 classes to classify into. For the income, there doesn't seem to be a clear trend, since the test accuracy decreases, the cross validation accuracy seems to have only a slightly increasing trend, and the test accuracy varies at all points.





Increasing the training set size had the greatest impact on improving the accuracy for both the letter and income dataset. The letter dataset reached a constant accuracy with just 70% of the training data utilized, and the income dataset at around 90%.



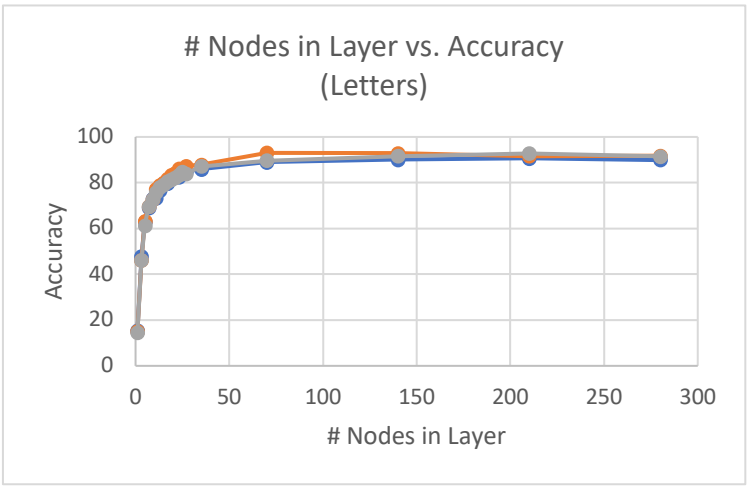


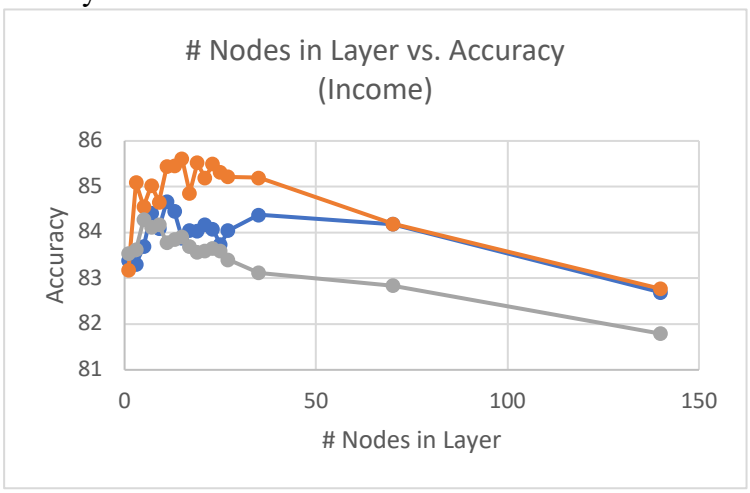
Neural Networks

I used the Multilayer Perceptron algorithm to implement the neural networks. This algorithm is a feedforward neural network that utilizes back propagation. The more nodes and layers in a neural network, the more complex it is, and therefore more vulnerable to overfitting. Neural networks are powerful because they can represent any arbitrary function but has a downside of taking exceptionally long to run, especially with more nodes and layers. The hyperparameters modified was first the number of nodes in a single layer, then the number of layers with the optimal number of nodes determined from the previous part, and finally the learning rate.

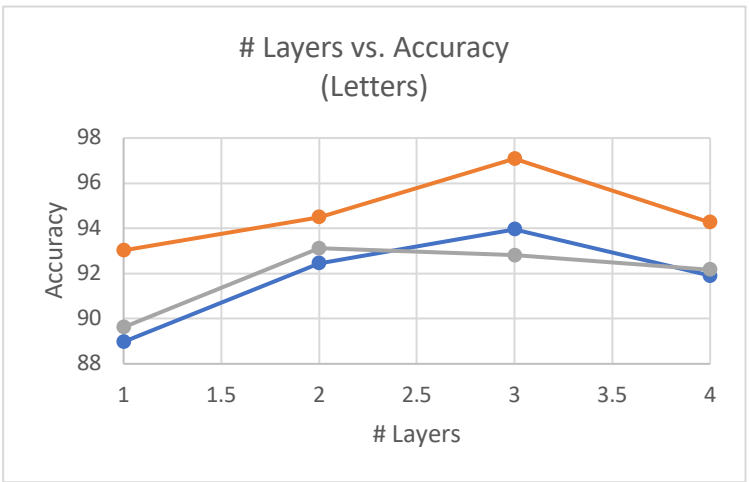
There was an exponential increase in accuracy with the initial increase of nodes from 1 to 70 for the letter dataset. After the 70th node, the accuracy wavered at around the same percentage; 70 nodes in a layer was determined to be optimal, especially since additional nodes greatly increased the runtime without much improvement in accuracy. The reason the subsequent increase in nodes did not increase accuracy is probably because letters recognition does not need a very complex network and additional nodes provided no additional information. Letters are

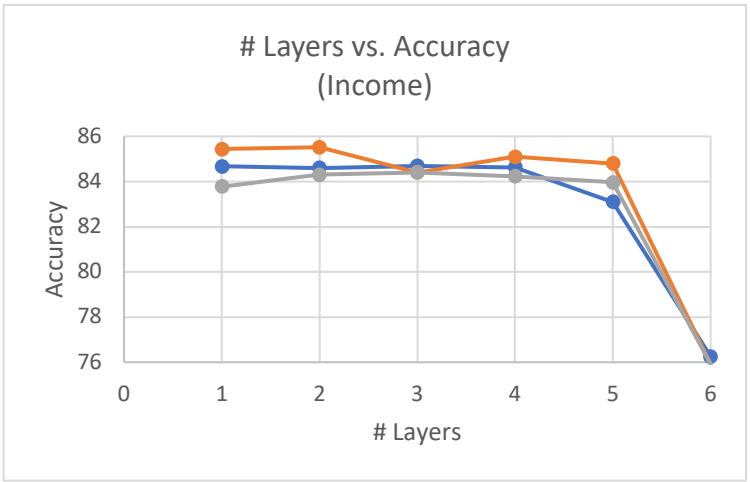
relatively static and comply to set of numerical rules and physical properties (although the dataset contained some random levels of distortions). That is, the letters in the alphabet will always look the same. The income dataset had a very different result. It had a less aesthetic trend, with very uneven accuracy up until the 35th node. After the 35th node, there was a decreasing trend in accuracy. Within the first 35 nodes, 11 nodes had the highest average accuracy among the training, testing, and cross validation set, so 11 nodes was deemed to be the optimal. The reason for the random jumps in accuracy within the first 35 nodes could be due to noise within the income dataset. Additionally, classifying income based on dynamic traits from marital status to native country is not as accurately defining as attributes such as width and height in the letter dataset. Many of the income attributes can have many interpretations and meanings. Sometimes married people are wealthier, but sometimes single people have more time for their careers. Within a country, there are many states and cities that vary greatly in their wealth, so a person from a nice town in the U.S.A. could be better set up for a higher income, but another person from a worse off town in the U.S.A. would be more likely to have a lower income.



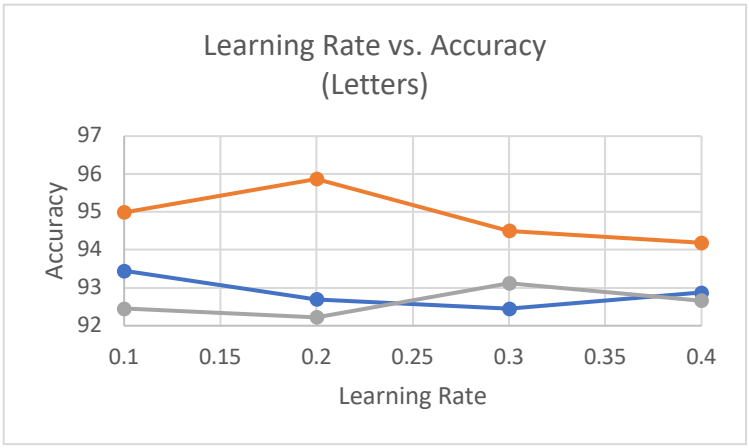


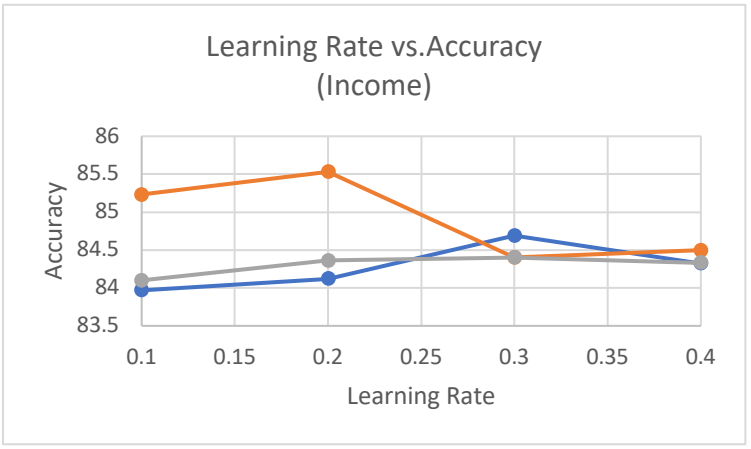
Using 70 nodes for the letter set, I then modified the number of hidden layers containing 70 nodes in each. The accuracy increased up until 3 layers, after which the neural net probably became too complex and began to overfit. Similarly, using 11 nodes for the income set, the optimal number of hidden layers with 11 nodes in each was 3. As you can see in the graph, the training, test, and cross validation sets all converged at 3 layers, where the accuracy is most stable across all sets.



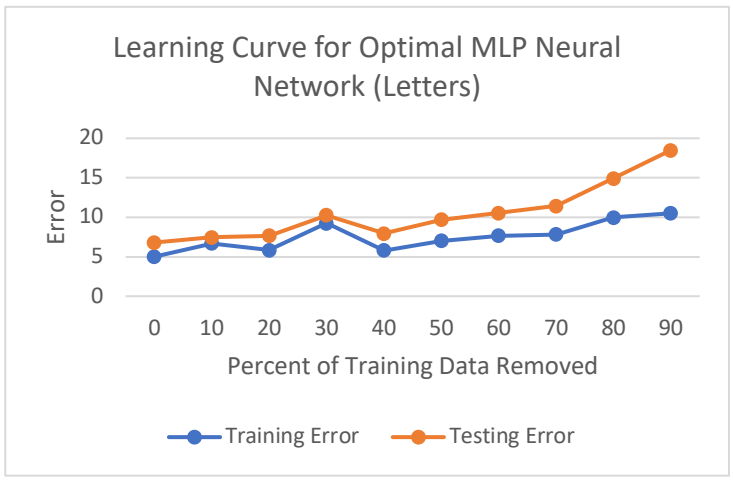


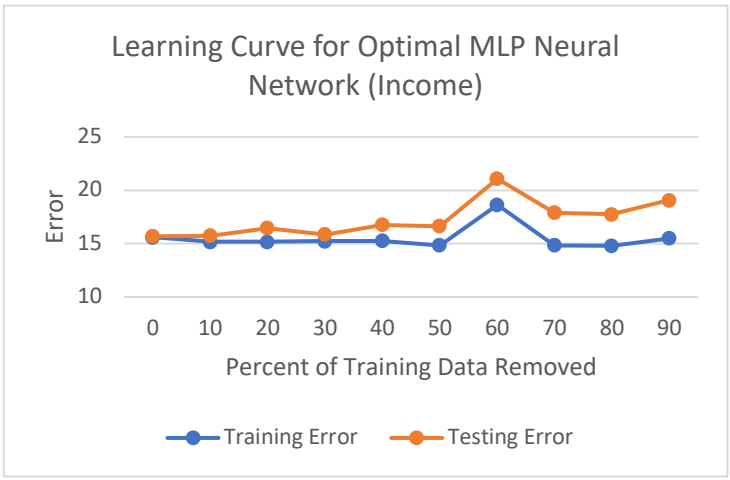
Finally, I then modified the learning rate. Intuitively speaking, the learning rate determines how quickly a network will take in new information in place of old information. Unfortunately, a trend wasn't detected for either of the datasets.





When examining the learning curves, the letter dataset seemed to benefit more from more training data, with an increase in almost 12% in accuracy for the test set. The income dataset only increased its accuracy by around 3.4%. This could be because the letter dataset has to learn to classify the instances into 26 different and needs more instances to learn more for each specific class. The income dataset only classifies into two classes, and is of a more abstract nature, so after a certain point even more data would not shed any additional light on how to better classify, and therefore led to a convergence on accuracy.

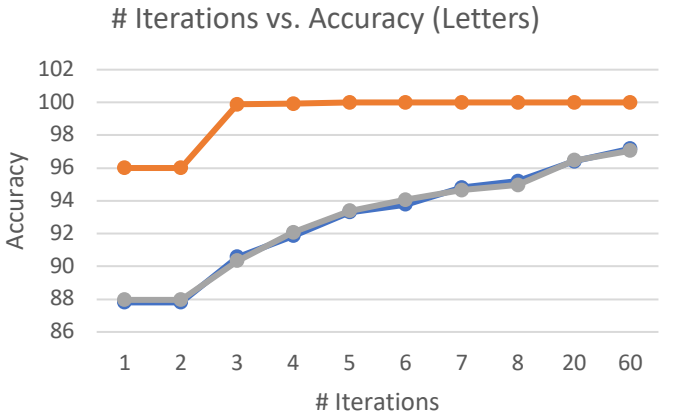


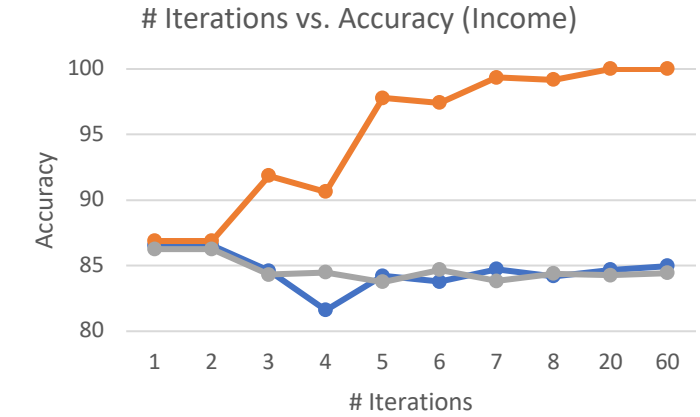


Boosting

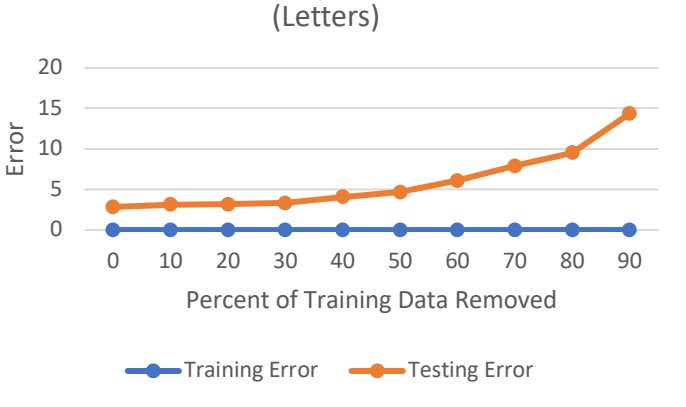
The AdaBoostM1 algorithm was used to implement boosting. Boosting is an ensemble learner method, where weak learners reduce the error at each iteration by ensuring it will do better than chance. The classifier used for each dataset is the optimal J48 decision tree discussed in the prior decision tree section. The hyperparameter was the number of iterations. For the letter dataset, there is a clear positive trend in accuracy with increasing number of iterations. This is expected, as weak learners must do better than chance, and so with each iteration, a previously incorrectly classified instance should be learned. However, as the accuracy nears 100%, more and more iterations are needed to only increase the accuracy very slightly. The increase from 20 iterations to 60 iterations is as much as the increase from 6 iterations to 7 iterations. For the income

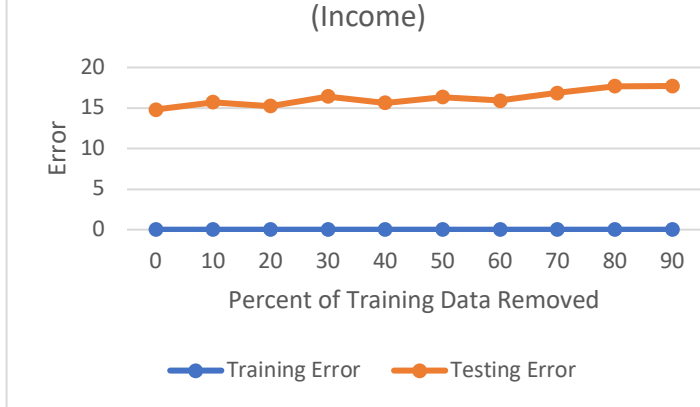
dataset, the training set clearly improves, but the cross validation and test set waver around the same percentage. When looking at the numbers, there is a minute increase in accuracy from 20 to 60 iterations, but it seems that a mammoth increase in iterations would be needed to reach a higher accuracy.





The effect of training size is greater on letters than on income. The testing error for the letters dataset continually decrease until it converges at around 80% of the training data utilized. The income testing set improves slightly from 10% to 40% of the training data used, but then the error wavers around at 15%.



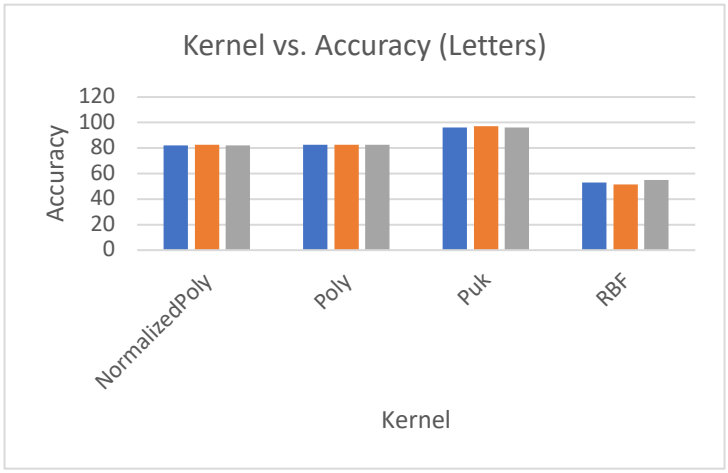


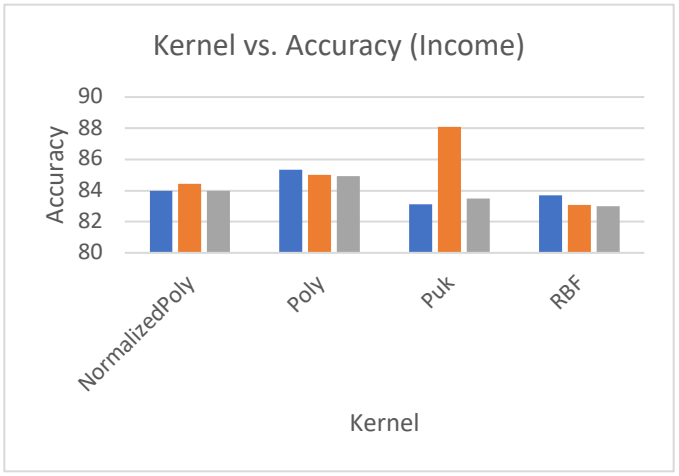
Support Vector Machines

To implement a support vector machine algorithm, I used the SMO algorithm. In SVMs, a hyperplane is constructed to separate the different classes. Out of all possible hyperplanes that classify the data, the best hyperplane is the one that is furthest from all the nearby training points; this allows for generalization in unseen data instances. Because support vector machines really only rely on the training data near the boundary of the hyperplane, these data points are the support vectors.

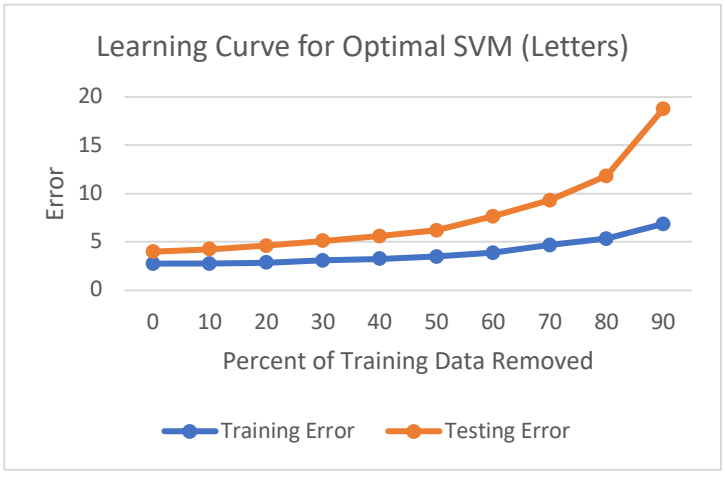
The way we inject domain knowledge is through the kernels, so that is the hyperparameter I modified. I used a normalized polynomial kernel, a polynomial kernel, the Pearson VII function-based universal kernel (Puk), and a radial basis function kernel (RBF). For the letter dataset, the

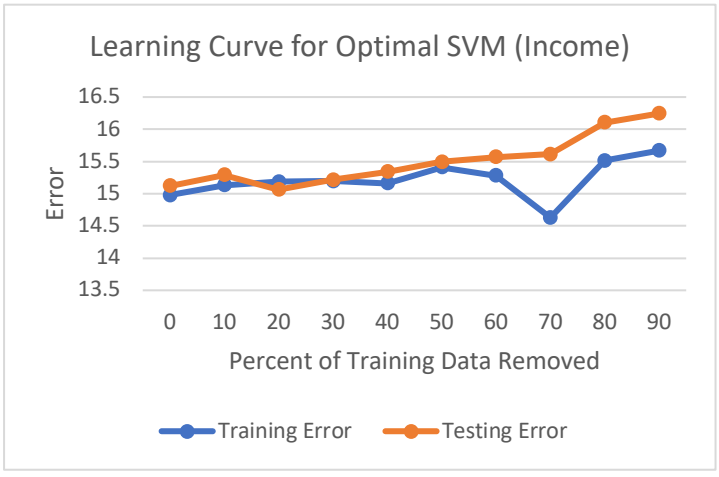
training, test, and cross validation sets all performed around equally for the same kernel, but the optimal kernel was the Puk. For the income dataset, there was more variation among the training, test, and cross validation sets for each kernel, with Puk's variation standing out the most. Puk was the lowest accuracy for testing and cross validation, but highest for training. Perhaps the training set coincidentally contained instances that did exceptionally well with Puk, but when generalized to other instances, failed to do as well. The overall optimal kernel for the income dataset was the polynomial kernel. Interestingly, the test accuracy for this kernel was actually highest out of the training and cross validation set (it is usually the other way around), meaning the hyperplane definitely did not overfit and was very nicely generalized to new data.





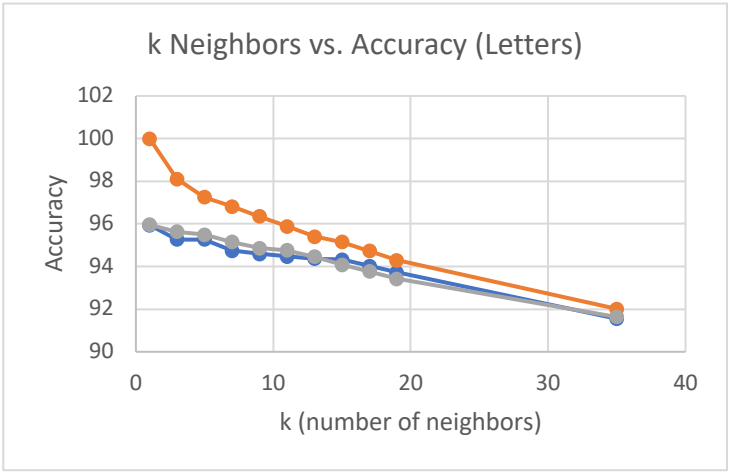
When looking at the learning curve, the letter dataset has a clean and clear trend; the more training data, the more accurate the classifier. The error stabilizes at around 90% of training data utilized. For the income dataset, there is a positive correlation between amount of training data and the accuracy, but the trend is not as smooth as the letters. There is a drop in error for the training set at 30% of that training data utilized, but because the subsequent percentages added follow along the trend of the other points, this drop is most likely an outlier due to a misrepresentation of the problem by the examples in that set. A downside to this algorithm is that it took a long time to run, ranging from an hour to 7 hours, but was still faster than neural networks.

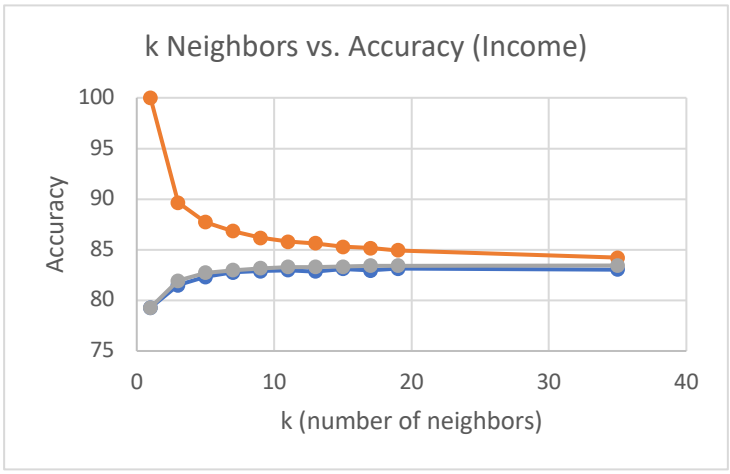




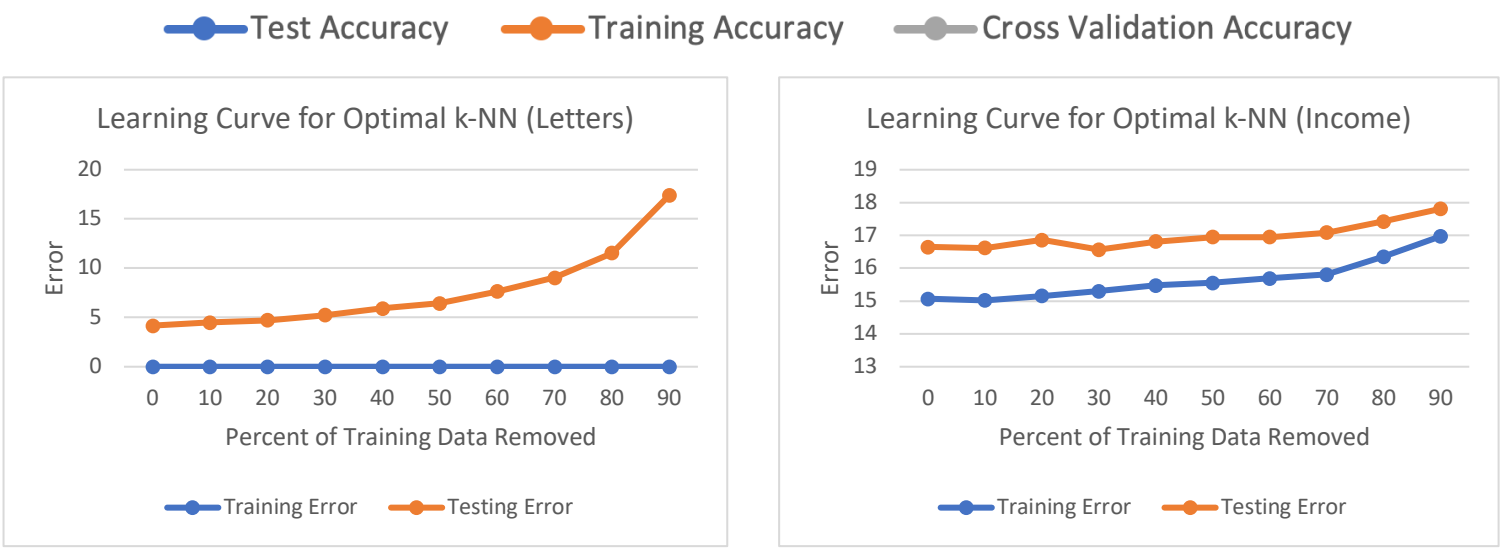
k-Nearest Neighbors

I used the IBk algorithm to implement the k-nearest neighbor algorithm. k-NN is an instance-based learner, also known as a lazy learner. This is because instead of going through the training set and building a model, and using the model to classify future instances, each instance is compared to the k nearest data points to predict a classification. Thus, the algorithm is "lazy" and only looks at the data points needed when it is needed. This also creates a really fast algorithm, but what is gained in speed is lost in space complexity, since all the training data needs to be kept at hand. The hyperparameter modified was the number of neighbors to look at (k). For the letter dataset, looking only at 1 neighbor yielded the highest accuracy. This makes sense, since letters are most probably in the same class as another letter that has the same physical properties as it. That is, if a letter looks like an 'A', it is mostly likely an 'A', especially since letters will always look the same. For the income dataset, however, the optimal number of neighbors was 19. While the training set did best with just 1 neighbor, the trend was completely opposite for the test and cross validation sets. The test and training accuracies began to converge at a similar accuracy as more neighbors were added. At 19 neighbors, the value was stabilized enough against all sets, and further neighbors resulted in a lower accuracy. The income dataset needed more neighbors to accurately predict a classification because income depends on attributes that are in some sense more complex than those of the letter dataset. Because the income attributes model some human behavior aspects, and human behavior is dynamic and vary greatly from people to people, what is indicative of one class for one instance might be the opposite for another instance.



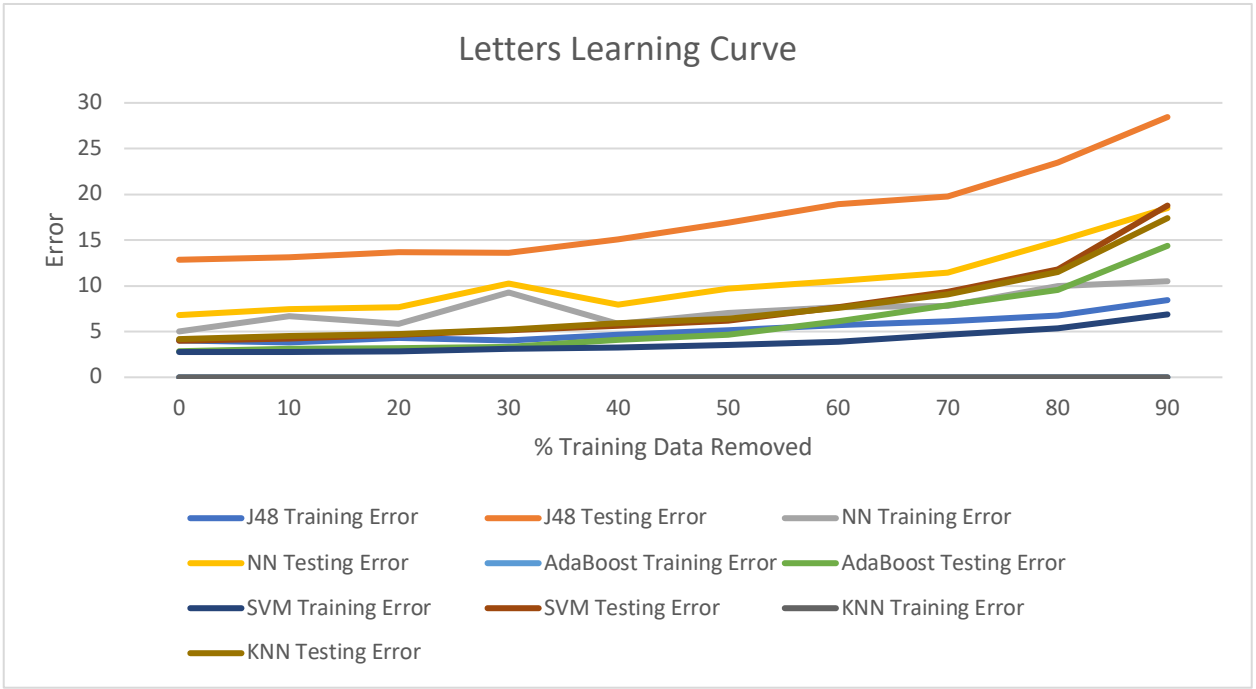


When looking at the learning curve for the letter dataset, it is interesting to note that for each of the five algorithms, the learning curves look almost the same. Because of this consistency through different algorithms, the letter dataset seems to need a considerable amount of training data in order to classify at a relatively high accuracy. Just as noted before, it may also be because there are 26 classes to classify the instances into, so a plethora of examples are needed for each class. The income dataset learning curve for k-NN has a positive correlation between more training data and higher accuracy. This learning curve is relatively smoother than the other learning curves for the income dataset. This makes sense, since having access to more data points, and possibly even closer neighbors, should yield a better accuracy.



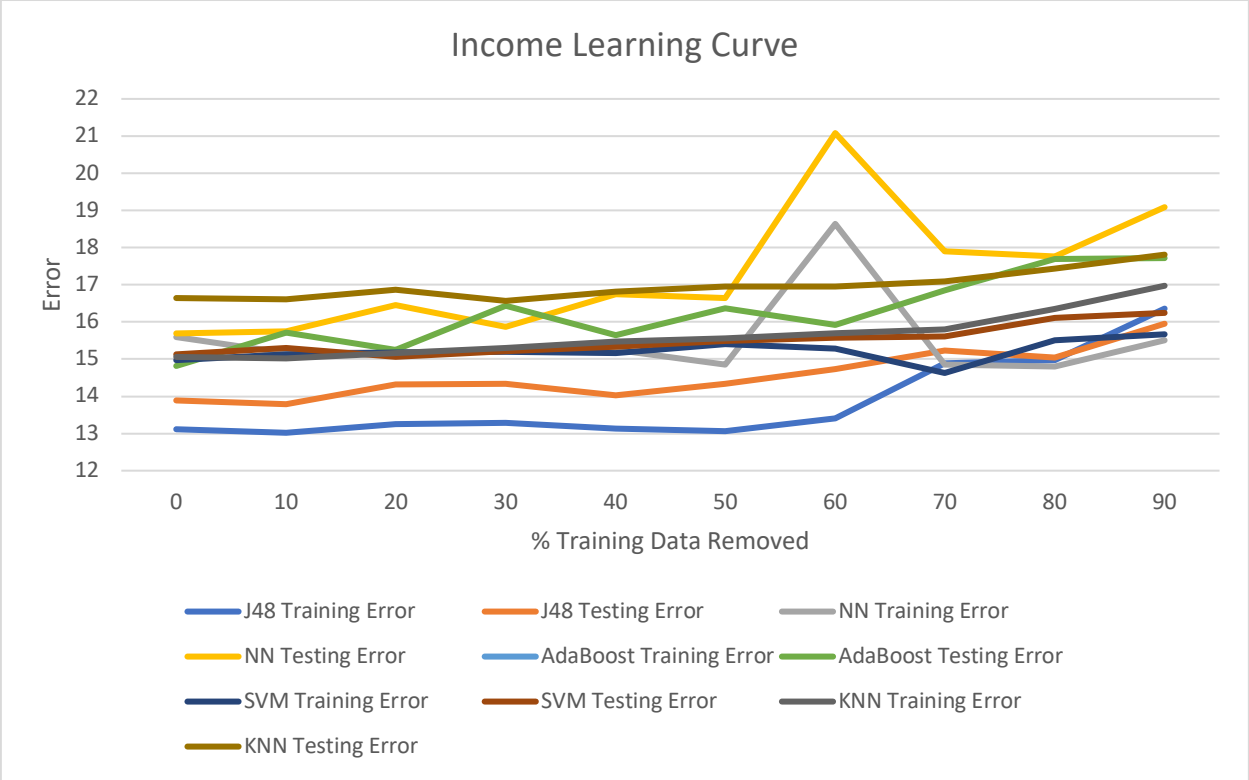
Conclusion

A fundamental mistake I made, which I realized too late into the project, was how I dealt with finding the optimal hyperparameters for algorithms where I changed more than one hyperparameter—J48 decision trees and multilayer perception neural networks. I performed a partial version of grid search, where I found the optimal value for one hyperparameter and then using that value, modified the other hyperparameter to find the overall optimal configuration. This is not right though, since finding an optimal value in one dimension does not mean that that is the space within which the overall maximum configuration is. I should have done a randomized set of parameters. In the neural networks, for example, I should have generated random number of layers and a random number of nodes for each layer and done that multiple times to find an optimal configuration. Grid search would have been impractical because of the number of combinations and amount of time a neural network takes to run, so using random hyperparameter values would increase the chance of finding an optimal value.



Above is the graph of all the learning curves for the letter dataset. The decision tree benefitted the most from more training data. The learning curve with the lowest error in the test set with all of the training data utilized is the boosting algorithm. When looking at the overall accuracies though, boosting, k-NN, and SVM all seem to have very high accuracies. Boosting has the highest test accuracy, at 97.175% with the hyperparameter set to 60 iterations. With more iterations, the accuracy is likely to increase, but additional number of iterations needed to considerably increase the accuracy would grow very fast. For example, doubling the iterations to 120 yields an accuracy of 97.375%, a 0.2% increase, but the time required almost tripled. However, because boosting yielded such a high accuracy, and only took a few minutes, AdaBoostM1 algorithm is the best algorithm for the letter dataset.

I still want to discuss the k-NN and SVM algorithms, which were close contenders. To decide which algorithm is the best among these two depends on the circumstances under which this classification is utilized. The maximum test accuracy for SVM is 95.95%, which is also the same maximum test accuracy for the k-nearest neighbors algorithm. However, the SVM algorithm can take a few hours to build a model, while the k-NN only takes seconds. If a faster classifier is needed, then k-NN would be the optimal algorithm. If space is a concern, though, then the SVM algorithm would be better, since the training data can be disposed of after training is completed.



Above is the graph of all the learning curves for the income dataset. At first glance, we can already see that that learning curves aren't as smooth as those of the letter dataset. There is a slight trend of decreasing error as more training data is utilized, but the slope is very small. The lowest overall testing error is for the J48 decision tree. The decision tree also only took seconds to run, so the optimal algorithm for the income dataset is the J48 decision tree.

If I were to redo this project, there are some changes I would carry out (in addition to fixing the mistake with multiple hyperparameters addressed earlier). For the neural networks on the income dataset, choosing 35 nodes per layer may have been a better choice, since that is when the accuracy depending on number of nodes began to stabilize. Additionally, I probably wouldn't vary the learning rate since that yielded no useful information but took a lot of time to run. For the decision trees for both datasets, I modified the confidence factor by increments of 0.05, but later found out the confidence factor ranges from 0 to 5. I should have varied the increments by more than just 0.05 since the range is so much bigger. Finally, I would have liked to randomize and shuffle my training set and re-run the learning curves for the income dataset, because there were some random jumps in error that might've been due to the set order in which the training data remained.

Table summary of the optimal hyperparameters determined for each classification algorithm

Classification Algorithm	**Hyperparameters**	
	Letters	**Income**
Decision Tree (J48)	Confidence factor = 0.35 minNumObj = 2	Confidence factor = 0.15 minNumObj = 4
Neural Networks (MLP)	# nodes in layer = 70 # layers = 3	# nodes in layer = 11 # layers = 3
Boosting (AdaBoostM1)	# iterations = 60	# iterations = 20
Support Vector Machines (SMO)	Kernel = Puk	Kernel = Polynomial
k-Nearest Neighbors (IBk)	# neighbors = 1	# neighbors = 19

Citations

Slate, David J., (1991). UCI Machine Learning Repository [https://archive.ics.uci.edu/ml/datasets/Letter+Recognition]. Irvine, CA: University of California, School of Information and Computer Science.

UCI Machine Learning (2017). Adult Census Income: Predict whether income exceeds $50K/yr based on census data. Retrieved January 20, 2019 from https://www.kaggle.com/uciml/adult-census-income/home.