David Vinegar / dvinegar3

An Analysis of Five Unique Classification Algorithms

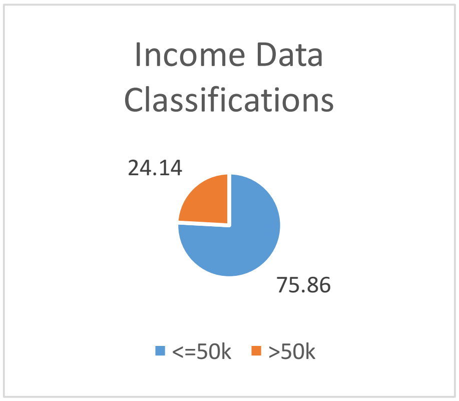
1) A description of your classification problems, and why you feel that they are interesting.

This report analyzes two classification problems. The first problem attempts to classify people into those who make greater than $50,000 annually and those who make less than or equal to that. Its inputs include age, education level, and hours worked per week. The data set includes 6,083 instances. It is interesting for a few reasons. Namely, it is quite different from my second data set, in that it has substantially more (about 3x) as many instances, and substantially fewer attributes (about 1/5 as many). Second, there are all kinds of reasons why it is interesting to know (or at least make a good guess at) how much money someone makes. Many people are reluctant to give out how much money they make to a complete stranger, but would gladly tell you whether or not they went to college, how many hours they worked last week, and how old they are. There are plenty of businesses, perhaps advertising agencies in particular, who would be able to better target more or less affluent customers based on this data.

The second classification problem examines housing data. Specifically, it attempts to classify houses from King County, CA, based on 14 different inputs into a set of price ranges. The data set has 2,042 instances. As mentioned previously, this should make for interesting comparisons, considering one data set has triple the instances and about a fifth the number of attributes. Additionally, anyone interested in building a house within King County, California, would want to know about how much it may be worth when it's done being built based on the inputs they know about.

It is worth noting that 75% of the income data belongs in a single class. Hence, an algorithm that just guesses that one class for everything would be right 75% of the time, which is quite comparable in performance to how each of the algorithms performed on it. The distribution of the housing data was between five different price ranges, with a more even distribution (thought certainly not an even distribution) that seen in the income data, so a similar comparison cannot be extended to it. Speaking of distributions, to ensure they remained consistent across training data and test data, stratified sampling was used.

It should also be noted that the predictors used in either dataset, the attributes, are not perfect predictors. Houses are worth as much as someone is willing to pay for them, which will create outliers. Similarly, although age, hours worked, and education are correlated with income, there are people out there who are young, barely work at all, with no education who earn very large sums of money.



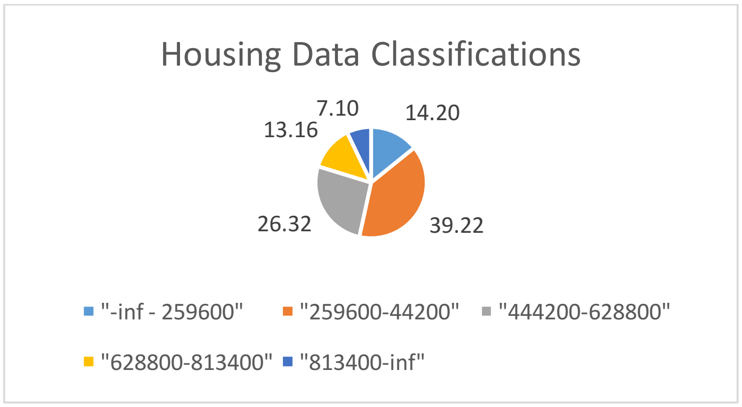
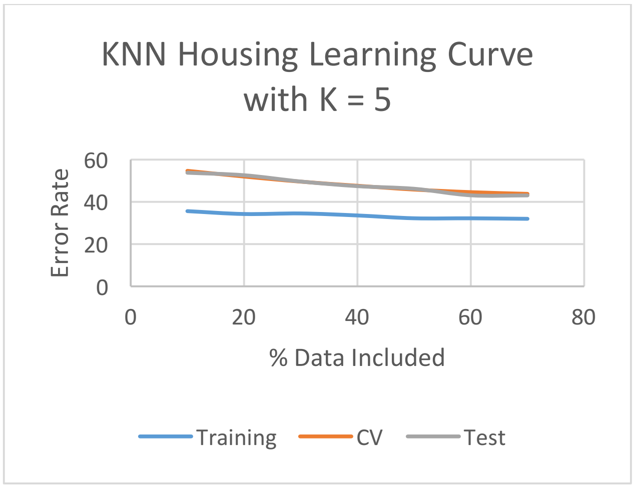


Figure 1

Figure 2

2) The training and testing error rates you obtained running the various learning algorithms on your problems, and--for the algorithms that are iterative--training times/iterations.



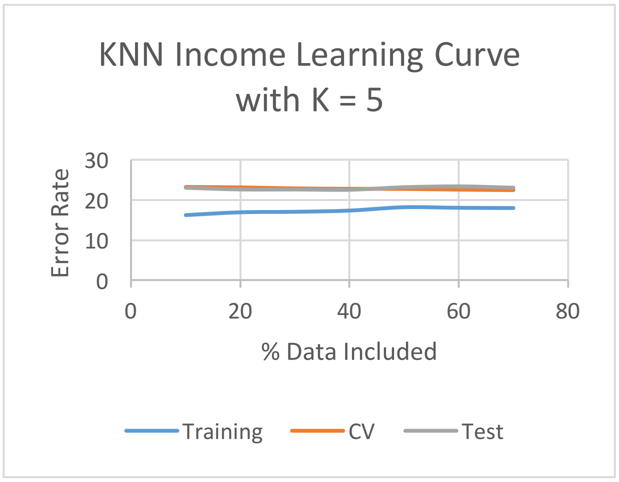
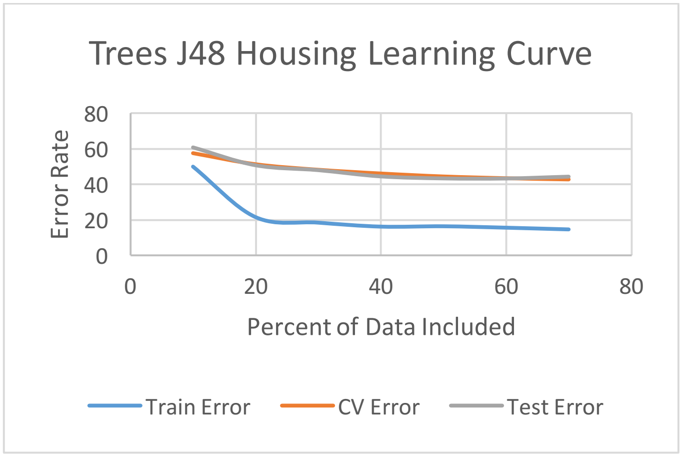


Figure 3

Figure 4



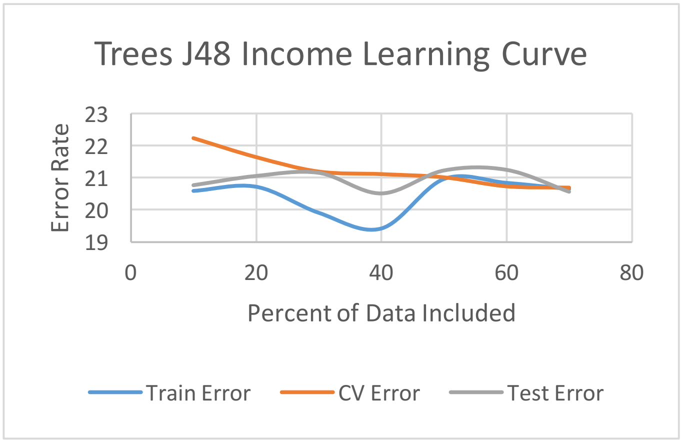


Figure 5

Figure 6

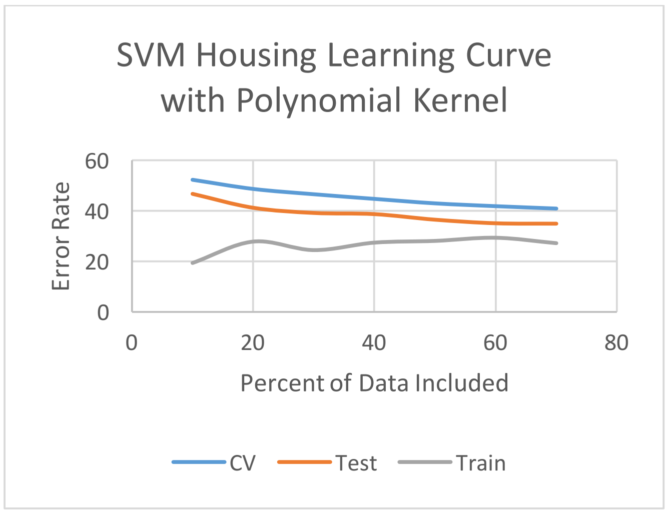


Figure 7

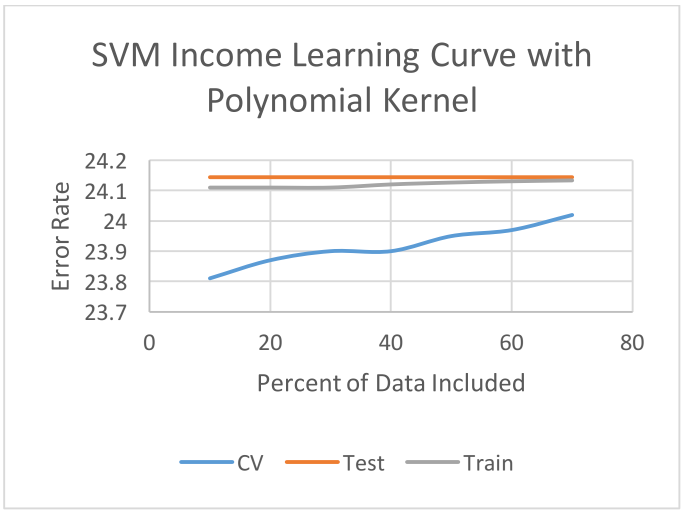


Figure 8

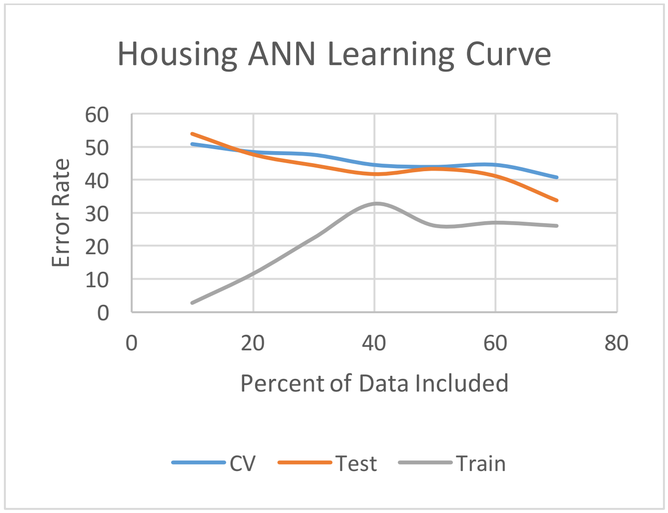


Figure 9

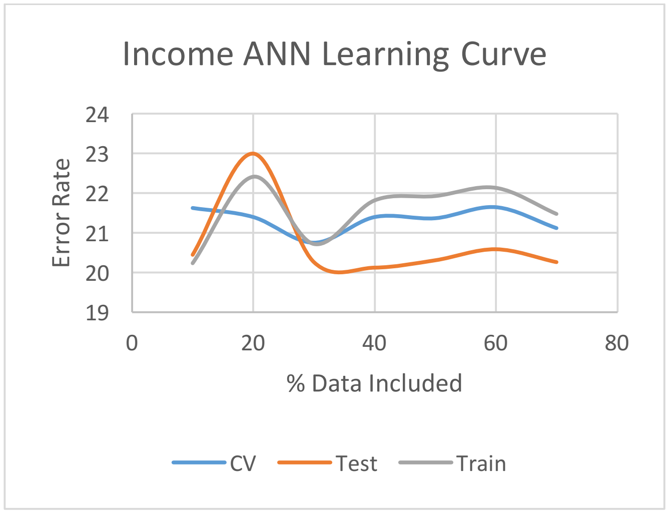


Figure 10

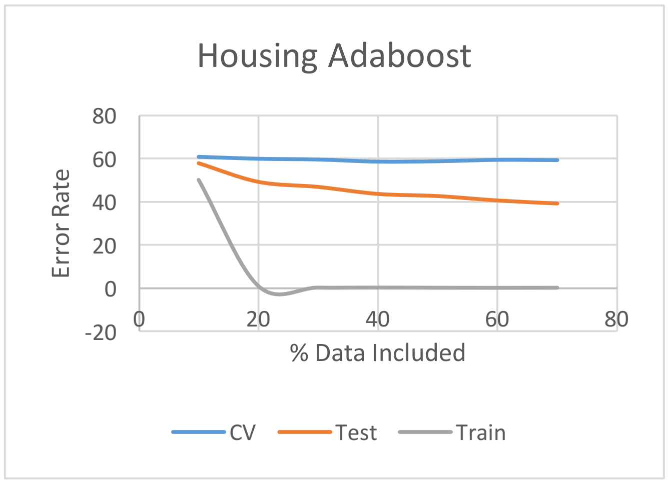


Figure 11

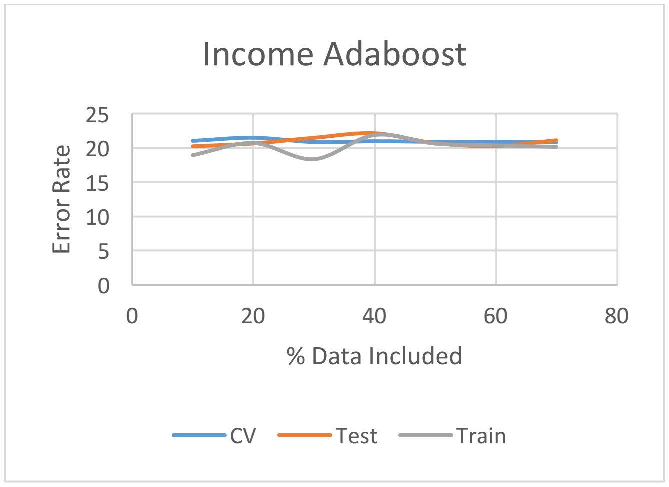


Figure 12

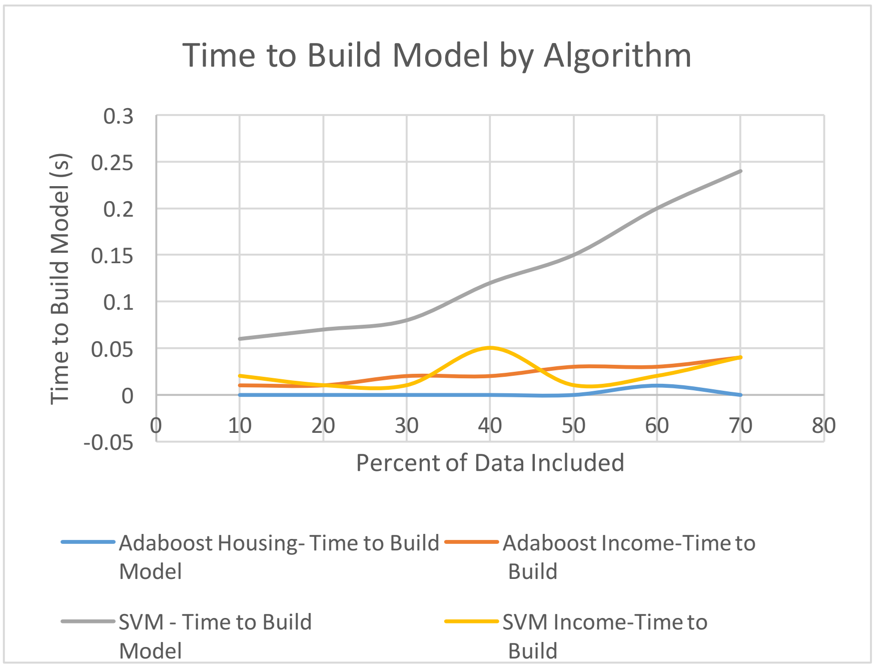


Figure 13

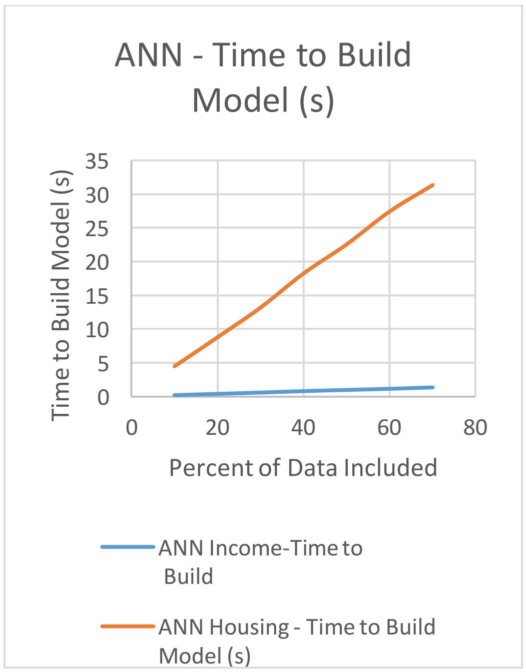


Figure 14

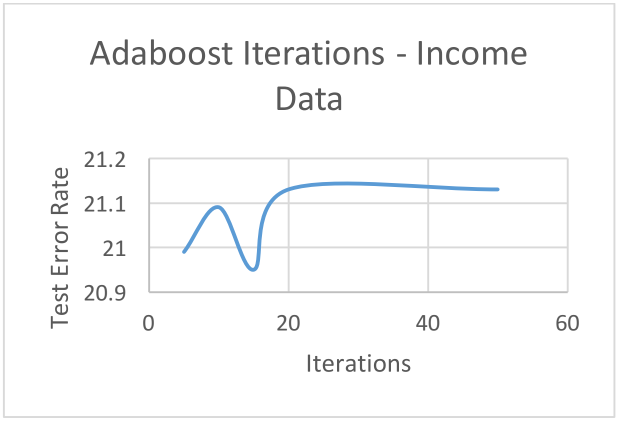


Figure 15

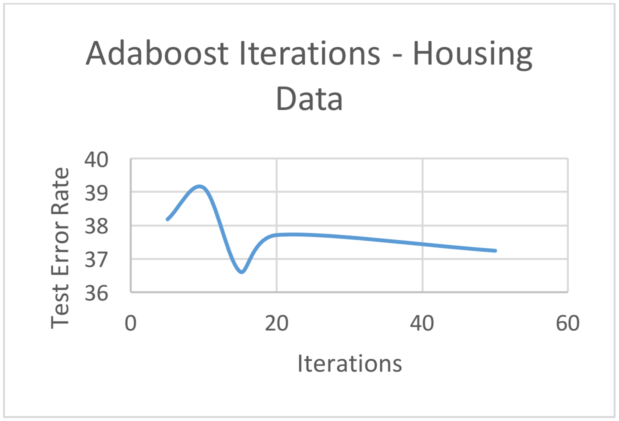


Figure 16

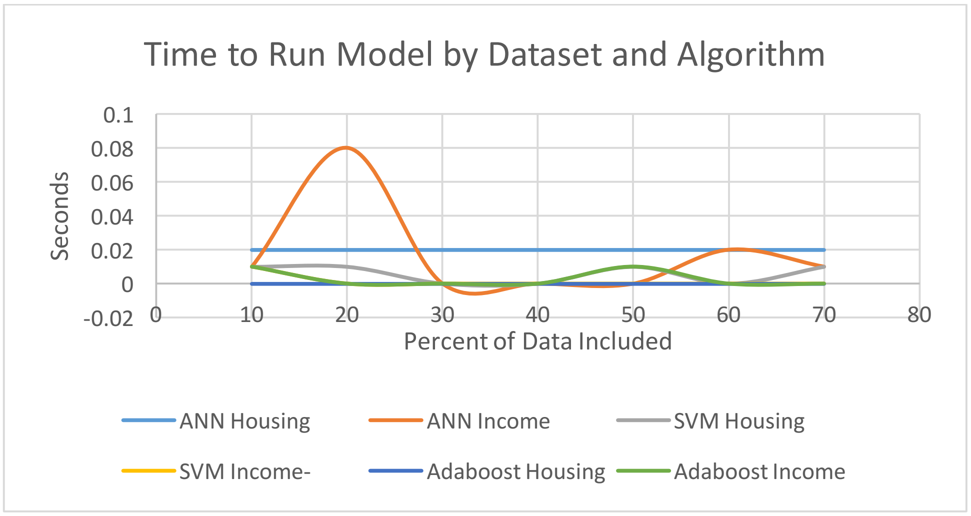


Figure 17

*Learning curve with ANN showing error rate and training time is omitted, since it did not affect the error rate.

3a) Why did you get the results you did? Compare and contrast the different algorithms.

The results are best represented by the two charts below on error rate and variance.

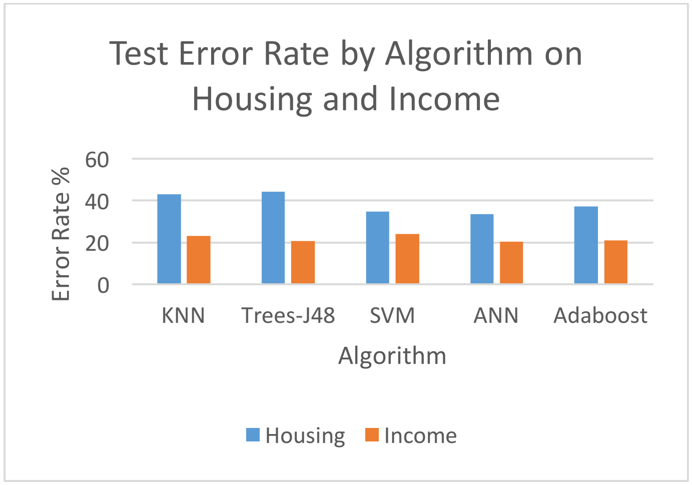


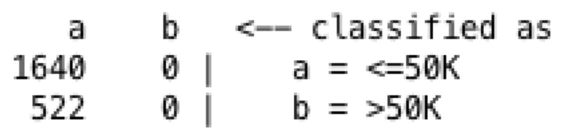
Figure 16

Figure 15: Variance by Algorithm and Dataset

Dataset	Housing Variance	Income Variance
KNN	6339.052596	121.6013647
Trees-J48	1.629090362	90096.31911
SVM	104846.6138	0.00063688
ANN	636490.1889	0.490870548
Adaboost	1596554.804	6.358591637

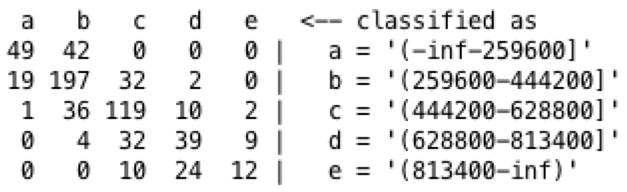
First, why is the error rate so much lower for the income data set? This can most likely be attributed to the income data set only having two potential classifications, versus five potential classifications for the housing data set. If the algorithm doesn't know how to properly classify a given instance in either data set, its chances of guessing correctly are 1 in 5 for the housing data and 1 in 2 for the income data. Interestingly, the worst algorithm in the pack for the income classification problem, Support Vector Machine (SVM), used just that strategy. It guessed class A (income <=50K) for every single instance of the test data. Although SVM did not work out that way for the Housing problem, with guesses in each category.

Confusion Matrix – SVM Income Data



```
    a     b   <-- classified as
 1640     0 |     a = <=50K
  522     0 |     b = >50K
```

Confusion Matrix – SVM Housing Data



```
   a   b   c   d   e   <-- classified as
  49  42   0   0   0 |   a = '(-inf-259600]'
  19 197  32   2   0 |   b = '(259600-444200]'
   1  36 119  10   2 |   c = '(444200-628800]'
   0   4  32  39   9 |   d = '(628800-813400]'
   0   0  10  24  12 |   e = '(813400-inf)'
```

It makes particular sense that SVM, Artificial Neural Network (ANN), and adaboost were the top three by error rate for the housing data because these three algorithms can reassign weights within various parts of the algorithms, which would be especially important in a data set with lots of attributes where some of them may be significantly more important than others. The housing dataset has 14 attributes. On the other hand, an algorithm like KNN weighs all attributes equally, so it makes sense that it would struggle more with lots of less meaningful attributes.

It is less clear why ANN was the best algorithm for the income data set. Its superiority over the other algorithms was certainly less pronounced within the income data set. It could be, similar to the previous hypothesis, that its ability to weight particular nodes was less important in a dataset with only 3 attributes, but perhaps it is just the most powerful classifier of the group regardless of dataset.

As far as variance, KNN varied the least on housing data, while it varied the most on income data. Support vector machine varied the least on income data, while adaboost varied the most on the housing data.

It is particularly interesting that KNN varied the least on the housing data, because its learning rate graph demonstrates high variance. There's a fairly large gap between the training data curve and the cross validation curve. The KNN error rate decreases as more data is added – another characteristic of high variance. It appears the reason KNN varied the least on the housing data is because it is underfitting the training data. This makes it perform more consistently regardless of the kind of data thrown at it. It is also worth mentioning that its error rate was the second highest among the five algorithms, so although its variance was low, its test error rate relative to the other algorithms was bad.

Why was the training error so low with adaboost on housing data, but cross validation and testing error were high? This is definitely due to overfitting, the opposite of what occurred with KNN on the housing data. The training error kept getting closer to 0 as more data was added, while cross validation error remained around 60%. Overfitting often goes hand in hand with high variance, which is also indicated by the large distance between the cross validation curve and the train curve on the Housing adaboost learning curve chart.

Another question worth asking is why the J48 trees algorithm performed so poorly? Its training error was quite low, and perhaps more significant is the fact that it went from extremely high, 50%, to very low, 14.49%, as more examples were added. This definitely implies overfitting, especially since we see that its training error was very low and its test error was so high.

3b) What sort of changes might you make to each of those algorithms to improve performance?

To improve performance on KNN, distance weighting could be used. The current form of the algorithm weighs all k neighbors equally, but weighting the k neighbors by distance could achieve superior results. Alternatively, a different way of finding the neighbors could have been used; the current algorithm uses Euclidean distance. Manhattan distance could have been tried, among other ways. Either of these methods, distance weighting or using a different neighbor finding method, could potentially decrease the error rate or decrease the running time. Lastly, as analysis showed, changing the value of K can increase performance.

For J48, the default Weka settings ended up being the best setting. This included splitting using information gain, where the attribute that provides the largest information gain (best sorts the data into its classification) is placed highest in the tree, at each level of the tree. There were some notable changes attempted that were able to reduce performance, however. Namely, setting binarySplits to True caused an increase in error rate by 3%. A 2% increase in error rate was seen when setting reducedErrorPruning to true. Probably the easiest way to improve the decision tree with pruning would be to try a version of it other than J48, perhaps some other form of ID3.

With SVM, a quick way to improve results was to switch the filter type. I originally used “Normalize the training data”. When changed to “Standardize training data”, the Test Error rate decreased to 33.9593% from 34.89% on the Housing data, but did not change the results of the Income data. I also tried using the filter type “No standardization/normalization”, but had to end the computations after letting them run for many hours with no end in sight. Perhaps this would have improved results even further. Switching the kernel to Puk was able to reduce income data error rate by 3.56% to 20.5828%. This same kernel switch worsened test error rate for the housing data by 3.59% to 38.49%.

ANN was unable to be improved for either data set. The default Weka settings were the best settings. Learning rate was altered within the range of 0.1 to 0.5 to no avail as well as momentum. Perhaps moving the two in concert may be one way to improve the results. Additionally, learning rate fluctuations were attempted with decays, which also was unable to beat Weka’s default setting results.

Adaboost improved upon adding in more iterations, as seen in Figure 15 and Figure 16. Weka’s default settings used 10 iterations. Test error rate was found to be optimal at 15 iterations; it lowered the test error rate by 2.51% from 39.12% to 36.61%. Additionally, more iterations could be completed. Additionally, weight thresholds were modified (not in concert with more iterations), but no change in performance was attained. Adaboost was improved using different weak learners than the one primarily used in the analysis, J48. This analysis is showcased in figure 25.

3c) How fast were they in terms of wall clock time? Iterations?

The following chart summarizes the wall clock times for both building the model and running the model on a training set of 70% of the training data.

Classifier	Housing - Time to Build Model (s)	Housing - Time to Run (s)	Income-Time to Build	Income-Time to Run	Iterations
KNN	0	0	0	0	N/A
J48	0	0	0	0	N/A
SVM	0.02	0	0.01	0	N/A
ANN	29.96	0.04	1.43	0.01	500 epochs
Adaboost	0.01	0	0.01	0	10

Figure 17

3d) Would cross validation help (and if it would, why didn't you implement it?)

Cross validation was implemented for each classifier and was very helpful for each classifier, because in conjunction with the training data, it was used for creating learning curves that enabled quick determination of bias and variance based on the distance between the two curves as well as how high each curve was in general. Cross validation was useful in determining overfitting or underfitting. That said, cross validation almost never improved performance in terms of error rate.

Additionally, k- fold cross validation in Weka, which was used with K=10, takes an average of each resulting classification/misclassification. It therefore can provide a helpful way to estimate what error will be on the test set.

3e) How much performance was due to the problems you chose?

Performance was heavily affected by the problems chosen for every classifier. This can be easily seen by just looking at figure 18 and seeing the large disparity between performance of the classifiers on the two different data sets. The closest in terms of error rate was SVM, which performed 10.74% better on the income data set than the housing data set.

Additionally, variance was significantly lower for the problem of classifying income than classifying housing prices. This is due to much smaller differences in training error and cross validation error, which implies much better fitting for the income data problem than the housing classification problem.

Interestingly, most of the algorithms were built and ran in less than a second. Only ANN needed more than that, but its performance as far as timing goes was much greater on the income data set. This implies that ANN has a much easier time with problems with larger number of instances than it does larger numbers of attributes.

3f) How about the values you chose for learning rates, stopping criteria, pruning methods, and so forth (and why doesn't your analysis show results for the different values you chose?)?

Analysis is largely excluded for data that does not improve default values, or does not change the outcome significantly.

For KNN, to showcase the learning curve, k = 5 was used. This number was determined somewhat arbitrarily. Error rate was lowest for different k's based on the dataset, so there was not one particular "best" k to choose for the learning curve that would best showcase each problem. k = 5 was a reasonably good pick in both datasets where error was relatively low for both datasets. Figures depicting the performance of various K's is below. From the figures, we can see that k was ideal for the income data at k = 40, while k was ideal for the housing data at k = 7. From this we can infer that frequently only the 7 closest data points were particularly relevant for the income data, while 40 closest data points were particularly relevant for the housing data. Additionally, error rate was improved by 0.14% on the income data set by varying distance weighting to "Weight by 1-distance" as well as using a different neighbor finding algorithm, called CoverTree, which improved the default by 0.19%.

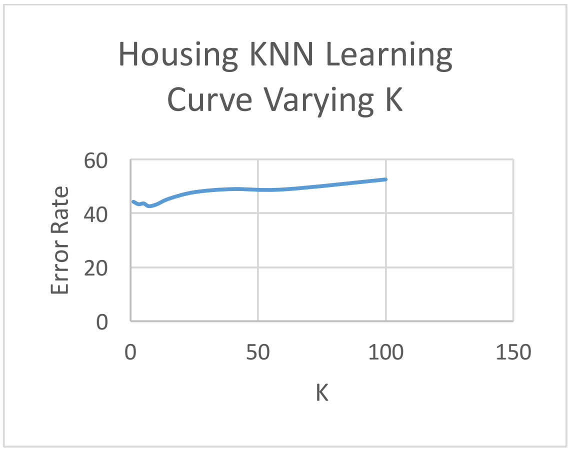


Figure 18

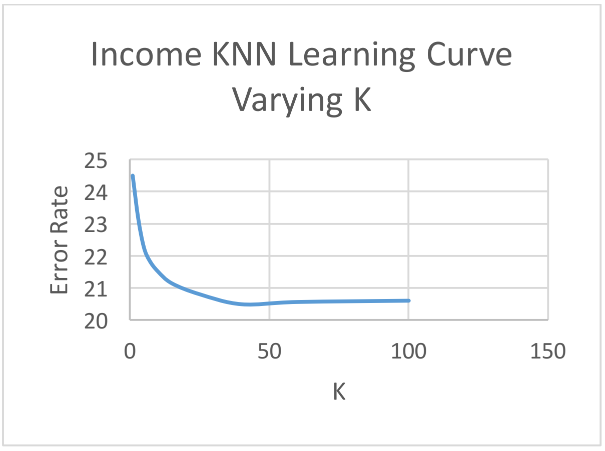
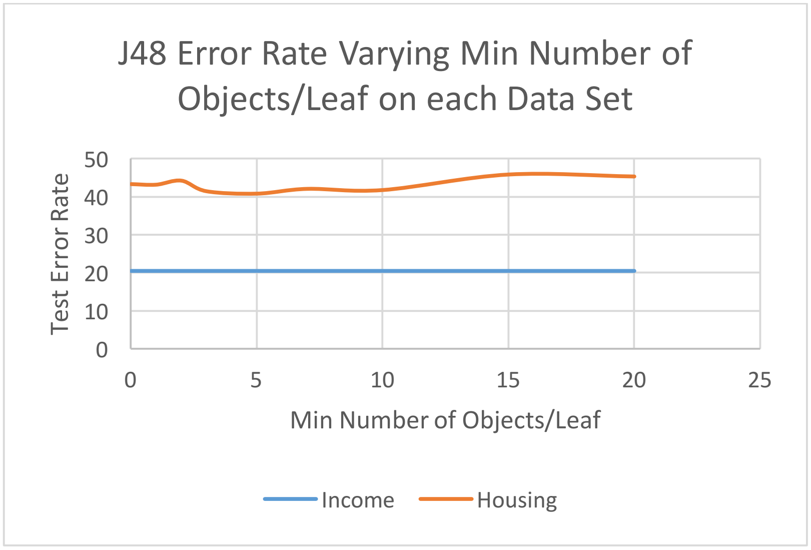


Figure 19

For J48, pruning was used since it was required for the project, but also because the performance was significantly better with pruning. The trees created with pruning were much smaller - 482 leaves and 695 nodes vs 596 leaves and 896 nodes. Additionally, percent correct was ~1% higher with pruning. Further analysis of this is excluded, since the error rate was so similar. However, performance was improved by changing the minimum number of objects per leaf from the default of 2 to 5 by 3.41%.



Some potential improvements for SVM could come by changing the filter type. Chart analysis shows the results of using two different kinds of kernels. Primarily, the polynomial kernel was used, but the normalized polynomial kernel was used. Of note though is that performance did vary significantly when using the RBFKernel on the housing data. The test error rate jumped to 60.87% on the housing data, while the test error rate stayed the same with the income data. Additionally, when using a Puk kernel, error rate was able to improve by 3.56% to 20.5828% for the Income data and worsened for the housing data by 3.59% to 38.49%. Asides from changing the kernel, the Weka setting “Normalize the training data” was initially used, but when changed to “Standardize training data”, the Test Error rate decreased to 33.9593% from 34.89% on the Housing data, but did not change the results of the Income data.

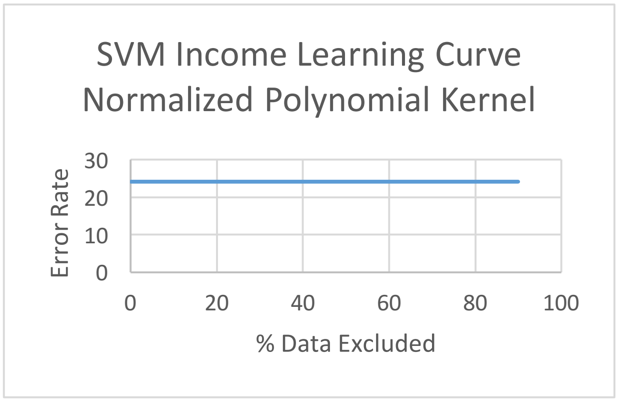


Figure 20

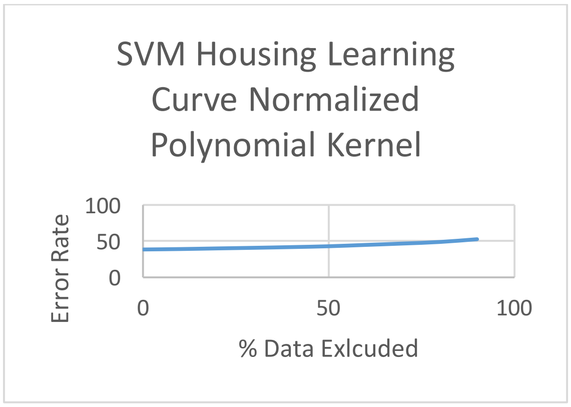


Figure 21

For ANN, the default settings were found to be best, which includes the learning rate, decay, momentum, training time, and the validation threshold.

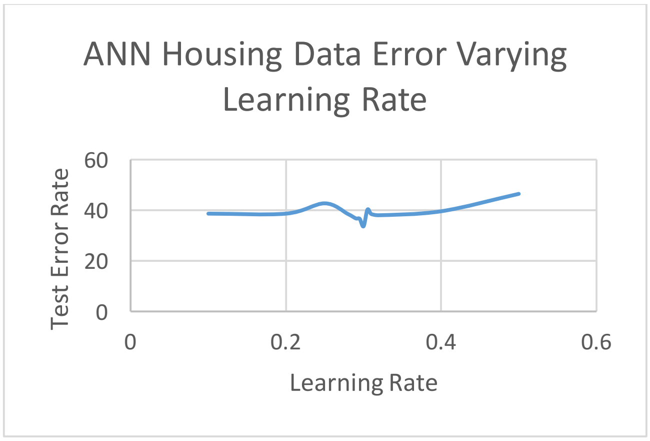


Figure 22

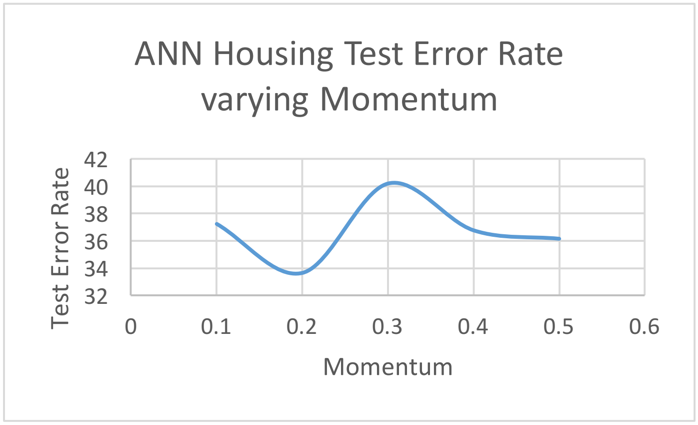


Figure 23

Adaboost results did not change as a result of changing the weight threshold, and using weight resampling barely changed the test error to 39.4366 from 39.12. On the other hand, increasing iterations was able to improve the performance of the housing data by a few percent, by decreasing the error rate from 39.12 to 36.61. A similar decrease in error rate was not seen with the income data, however. In contrast, income and housing data results changed drastically, using different weak learners, as portrayed in figure 25.

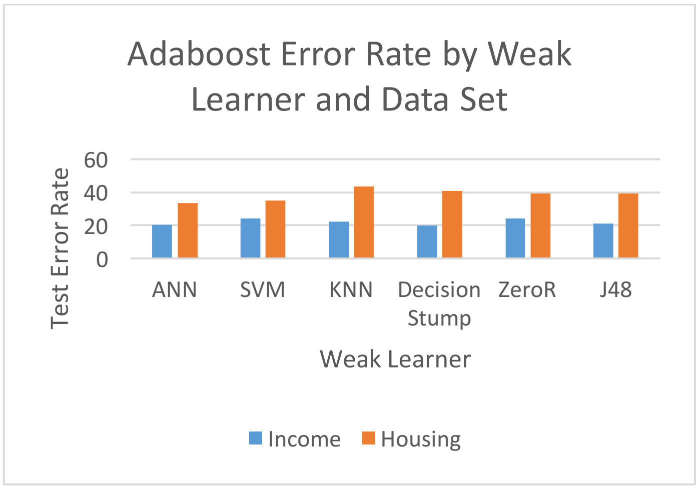


Figure 24

3g) Which algorithm performed best? How do you define best? Be creative and think of as many questions you can, and as many answers as you can.

The best algorithm could be defined as the algorithm that generalizes better than all the other algorithms. This could mean the model with the smallest bias and the least variance. For the purpose of this project, the bias will be measured by the test error rate, and the variance is measured by computing variance over the sum of squared errors between the training curve and the cross validation curve.

As far as classifying the highest percent of test examples, Multilayer Perceptron Artificial Neural Network (ANN) performed the best on both Income and Housing datasets. ANN didn't perform as well when looking at its variance, coming in at 3rd best for both data sets. K-Nearest Neighbor (KNN) had the best variance on the housing data, while support vector machine (SVM) had the best variance on the income data.

Because ANN classified the greatest percent of examples correctly and is in the middle of the pack of algorithms as far as variance goes, it is clearly the algorithm that best generalizes. To achieve the smallest bias in both data sets, its variance trade-off was still somewhat favorable. It is noteworthy though that ANN was the model that took the longest to build as well as execute on the test data.

<u>Data Sources – Works Cited</u>

1) UCI Machine Learning Repository [http://archive.ics.uci.edu/ml/datasets/Census+Income]. Irvine, CA: University of California, School of Information and Computer Science.

2) Kaggle. https://www.kaggle.com/harlfoxem/housesalesprediction

3) Weka. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1