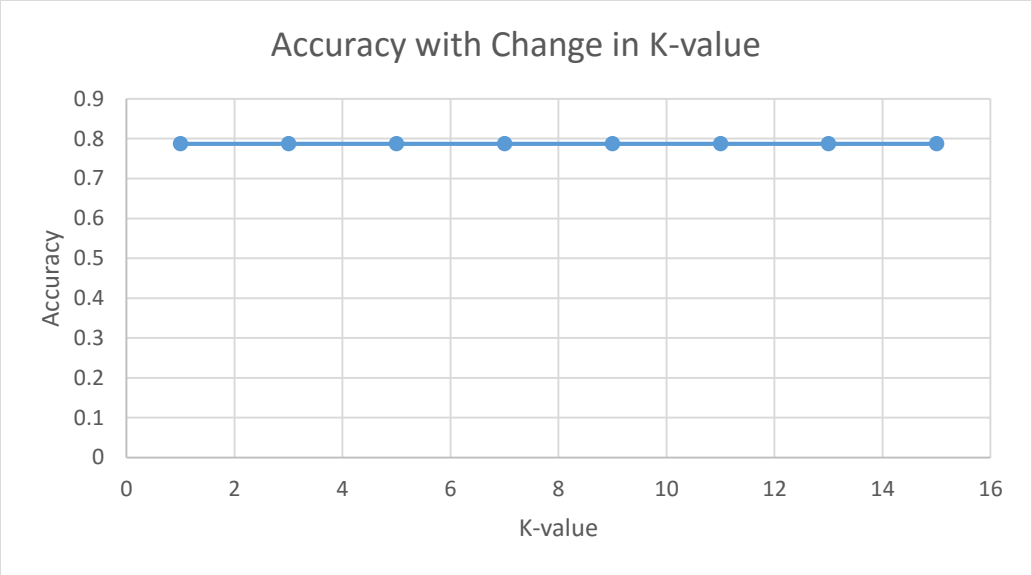
Instance-Based Learning Report

Wesley Ackerman

Magic Telescope Dataset

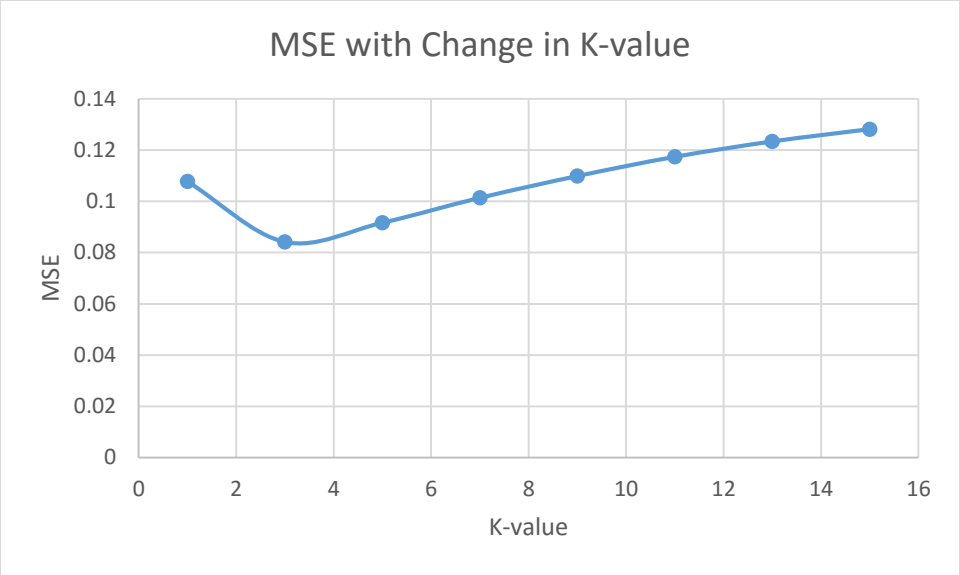
I implemented k-nearest neighbor on the supplied magic telescope training and test sets. Without normalizing the dataset, I got 78.56% accuracy. When I normalized the dataset, I saw a very slight improvement to 78.74% accuracy. Since the data values were fairly small in general (must were under 40, besides occasional data-points and the final feature column), I think normalizing didn't have as large of an effect as it would if the dataset contained very large, and more varied values.

When I changed the k-value for the nearest neighbor algorithm, I saw no change in accuracy. I always got the same value, whether I used 1, 15, or anything in between for k. I suspect that some of this is because I used the same data-points, in the same order, each time. Also, since there are only two possible output classes in the dataset, it makes sense that classifying an instance would rarely switch from one to the other when comparing it to its k neighbors, because they are likely to be the same class.



Housing Dataset

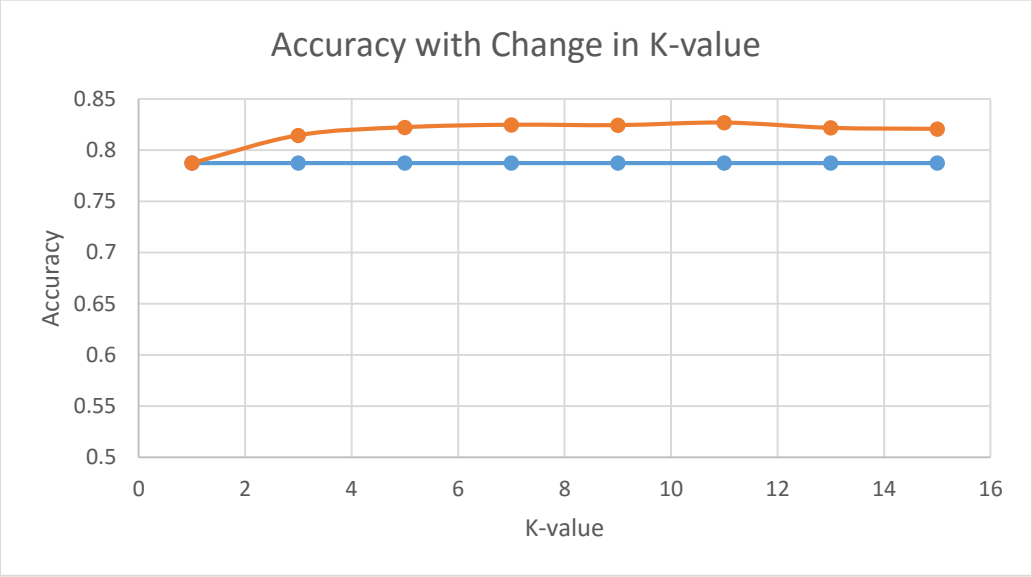
In experimenting with different values of k for the housing dataset, I found mean squared error was the lowest with a k value of 3. This is probably because the housing dataset is quite small, so as k gets larger, the algorithm begins to compare neighbors with the test instance that aren't nearly as similar. My results are shown below.



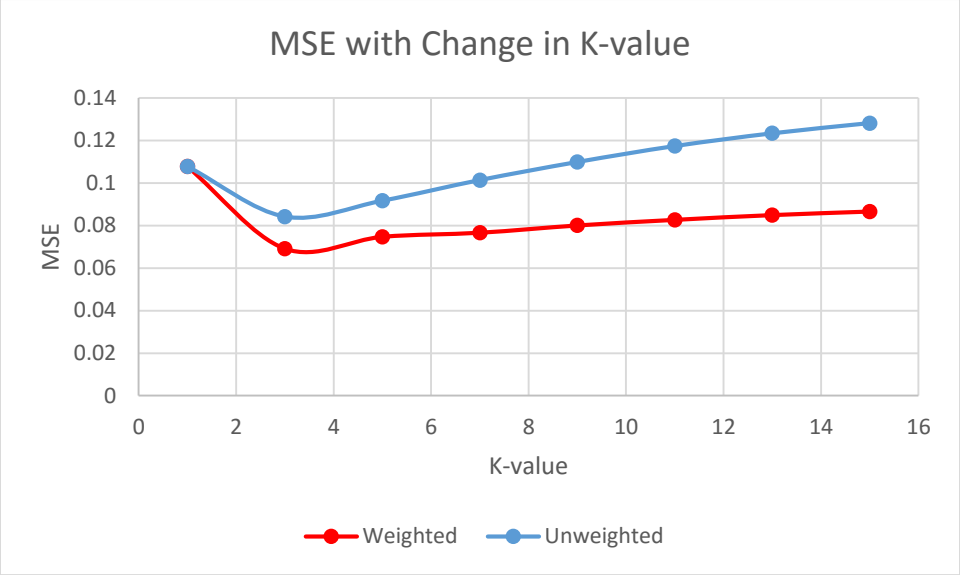
Distance-Weighted Voting

In adding distance-weighting the k-nearest neighbor algorithm, I found that my model's accuracy improved on the dataset. The magic telescope dataset saw a continued increase in accuracy as k increased, while the housing dataset had the lowest MSE at k=3, the same as the unweighted version. However, accuracy was improved on both datasets by using distance-weighting. This makes sense because it gives the closest neighbor instances the most say in what a newly-classified instance's label should be. This increases the probability that an instance will be labeled accurately, because the label is based more on its very closest neighbors.

On the magic telescope, unlike the unweighted algorithm, changing k for the weighted algorithm did have an effect on accuracy. It performed best where k was set to 11. The improvement from k=5 to k=11 was very small, but there was a significant jump in accuracy from k=1 to k=3.



On the housing dataset, the algorithm still performed best when a k value of 3 was used. However, the MSE was lower for all values of k but 1 when compared to the non-distance-weighted runs. Below, non-weighted and weighted voting are plotted together so that the improvement can be seen. The two curves have the same shape, but the weighted one has lower MSE values and increases less rapidly.



Credit Approval Dataset

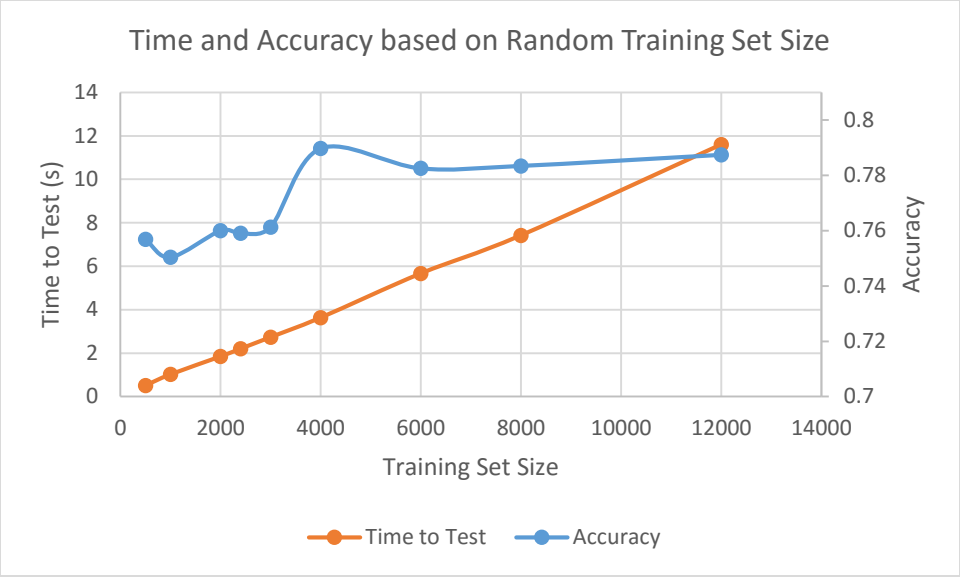
I chose the usual, Euclidean distance metric for continuous values. Since many of them are financial, it makes sense to measure distance as the difference between the financial values.

I chose a 1-0 distance metric for nominal attributes in the credit approval dataset. Instances with the same class of a nominal feature were given a distance of 0, while instances with different classes of a nominal feature were given a distance of one. I chose this metric because it is relatively simple to implement, and because it works well for attributes where it is unknown how the classes relate. Since I don't know if certain classes are more or less related to each other for a given feature, it made more sense to classify them as either the same or different.

For unknown attributes, I chose to always classify them as 0.75 distance away from the other instance. At first, I tried classifying them as 1 distance. Since it is impossible to say how closely related the unknown is to another instance, it made sense to me to say that they are completely different from other instances. I also tried other values, and found that a distance of .5 performed about the same of a distance of 1, and a distance of .75 performed better, on average. It makes sense to say that unknowns aren't completely unlike other instances (1), but they are still quite different. I got an average of 81% accuracy using a distance of 1, and an average of 84% accuracy using a distance of .75.

Experimentation

For my own experiment, I took a random subset of the training set to use as a new training set. I tried multiple subset sizes of the training set, and found that a training set of 4000 instances offered the best mix of time saved and accuracy saved. With 400 instances, testing took an average of 3.6 seconds and an average accuracy of 78.97%. That is about a quarter of the time the algorithm took with the full 12000 instances, with about the same amount of accuracy.



As the graph shows, there was a sizeable jump in accuracy from a set of size 3000 to a set of size 4000. The average accuracy is about as good as the accuracy with the full test set. Datasets with 3000 and less instances had significantly less accuracy. The graph also shows that time to test increases linearly with the size of the training set. 4000 data-points offered a good compromise of time and accuracy.