James Liu CS 4641
Assignment #1

Supervised Learning Report

Datasets

Krkopt: This is a dataset comprised of endgame scenarios in Chess that involve only the White King, White Rook, and Black King. The task is to predict the optimal number of moves it takes for the White player to achieve checkmate given the positions of the three pieces. Attributions for the dataset can be seen in the README.txt file.

The data was complete, contained 7 attributes (including class), and had 28,056 instances.

There are a total of 18 classes: draw, where it is impossible for white to achieve checkmate, and then zero to sixteen optimal moves for White to achieve checkmate. The class distribution can be seen in Figure 1.

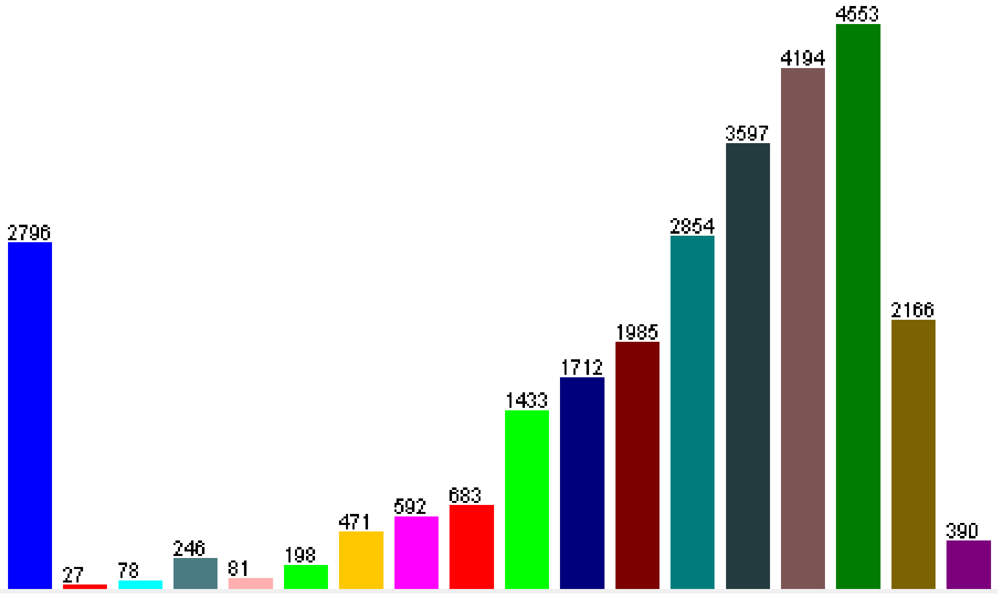


Figure 1. The krkopt data set sorted by class (draw on the farthest left, zero, one, two…, sixteen on the farthest right).

spambase: The dataset is comprised of email instances that have been sorted into whether it was considered to be spam by the recipient. Each instance contains whether the email was spam or not as its class, and a list of attributes describing the frequency of certain words, characters, or certain characteristics about the capital letters used in the email. The data is complete, has a total of 58 attributes (including class), and 4,601 instances.

Why are these datasets interesting?

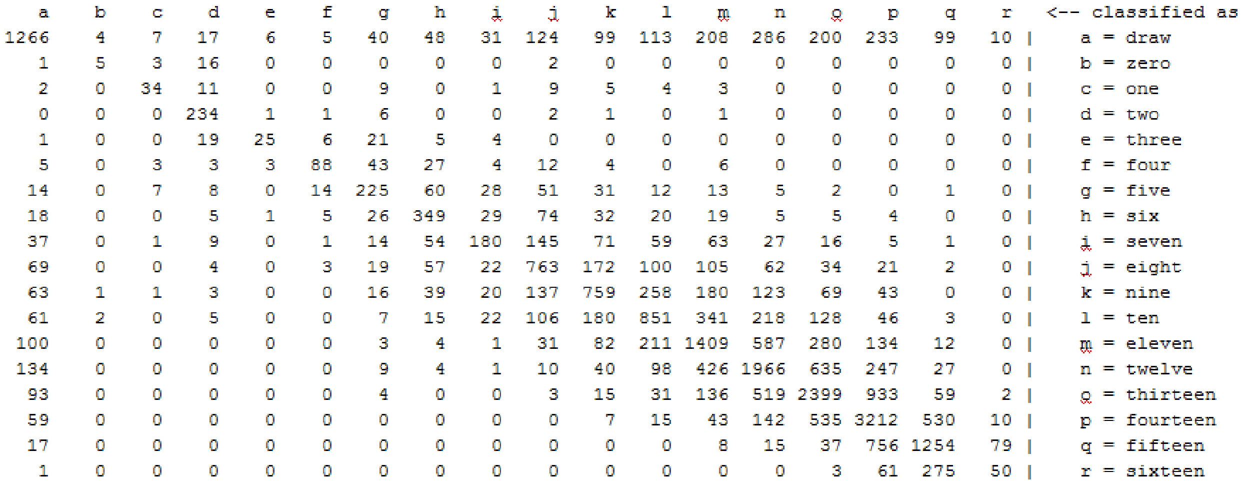
Both of these datasets are interesting from both a practical use perspective and a machine learning perspective. As for the krkopt dataset, Chess, and other complex strategy games, has long been the interest of artificial intelligence research. Companies like Intel have invested millions of dollars in developing supercomputers that could properly preform large scale searches that would allow computers to compete, and win, against even the best human players. Developing a heuristic to optimize search in endgame scenarios makes it more feasible to implement a similar chess playing AI in systems with less computing power. As for the spambase dataset, it is a common interest for all email systems to filter out spam, and thus so is finding an effective algorithm that makes few mistakes.

From a machine learning perspective, why are these datasets interesting? The krkopt dataset is of interest because of the noise in its output: many situations that lead to checkmate are very similar to others that lead to stalemate, and vice versa. Additionally, the dataset is large, with 28,000 instances, but not large enough to cover the entire state space of the problem. This also makes it interesting to see how the algorithms deal with the noisy output while limiting overfitting, as well as the effect of a large dataset has on both training and test times of each algorithm.

The spambase dataset is interesting because of how its low instance count respective to the number attributes. With only 4,600 instances and 58 attributes, the curse of dimensionality makes this dataset very sparse and thus it’ll difficult to produce low-error results. However, it should also should be noted that many of attributes have distributions that are heavily skewed to the right, and thus many of the attributes may be irrelevant to determining the output.

<u>Decision Trees</u>

Prune State	Confidence	Leaves	Tree Size	Training %	Test %	Training Time	Test Time
krkopt							
Pruned	0.125	3,529	4,033	62.6176%	53.7104%	0.16s	0.11s
Pruned	0.25	4,600	5,257	66.7486%	56.5833%	0.16s	0.11s
Pruned	0.5	6,546	7,481	71.8385%	57.9876%	0.16s	0.11s
Unpruned	N/A	8,751	10,001	74.583%	58.3191%	0.12s	0.11s
spambase							
Pruned	0.125	87	173	96.5877%	92.7842%	0.45s	0.06s
Pruned	0.25	104	207	97.1745%	92.9798%	0.42s	0.06s
Pruned	0.5	159	317	98.1743%	92.6103%	0.44s	0.06s
Unpruned	N/A	190	379	98.3265%	92.6538%	0.41s	0.06s



```
    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1266    4    7   17    6    5   40   48   31  124   99  113  208  286  200  233   99   10 |    a = draw
    1    5    3   16    0    0    0    0    0    2    0    0    0    0    0    0    0    0 |    b = zero
    2    0   34   11    0    0    9    0    1    9    5    4    3    0    0    0    0    0 |    c = one
    0    0    0  234    1    1    6    0    0    2    1    0    1    0    0    0    0    0 |    d = two
    1    0    0   19   25    6   21    5    4    0    0    0    0    0    0    0    0    0 |    e = three
    5    0    3    3    3   88   43   27    4   12    4    0    6    0    0    0    0    0 |    f = four
   14    0    7    8    0   14  225   60   28   51   31   12   13    5    2    0    1    0 |    g = five
   18    0    0    5    1    5   26  349   29   74   32   20   19    5    5    4    0    0 |    h = six
   37    0    1    9    0    1   14   54  180  145   71   59   63   27   16    5    1    0 |    i = seven
   69    0    0    4    0    3   19   57   22  763  172  100  105   62   34   21    2    0 |    j = eight
   63    1    1    3    0    0   16   39   20  137  759  258  180  123   69   43    0    0 |    k = nine
   61    2    0    5    0    0    7   15   22  106  180  851  341  218  128   46    3    0 |    l = ten
  100    0    0    0    0    0    3    4    1   31   82  211 1409  587  280  134   12    0 |    m = eleven
  134    0    0    0    0    0    9    4    1   10   40   98  426 1966  635  247   27    0 |    n = twelve
   93    0    0    0    0    0    4    0    0    3   15   31  136  519 2399  933   59    2 |    o = thirteen
   59    0    0    0    0    0    0    0    0    0    7   15   43  142  535 3212  530   10 |    p = fourteen
   17    0    0    0    0    0    0    0    0    0    0    0    8   15   37  756 1254   79 |    q = fifteen
    1    0    0    0    0    0    0    0    0    0    0    0    0    0    3   61  275   50 |    r = sixteen
```

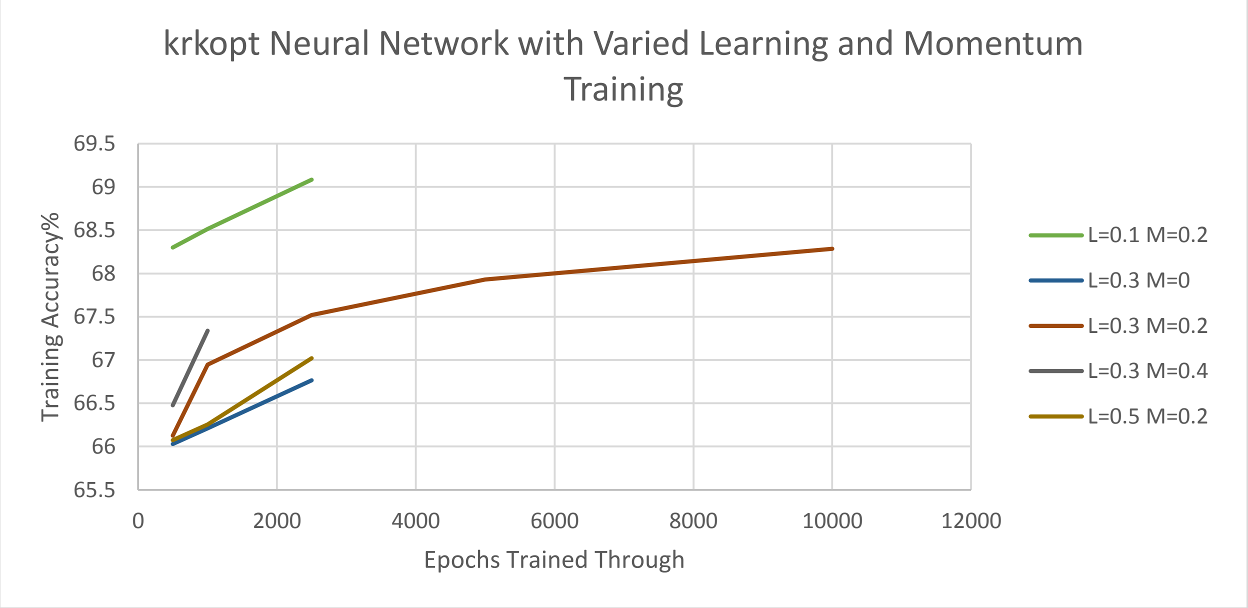
Figure 2. Testing Confusion Matrix for J48 on krkopt.arff, pruned at a 0.125 confidence level.

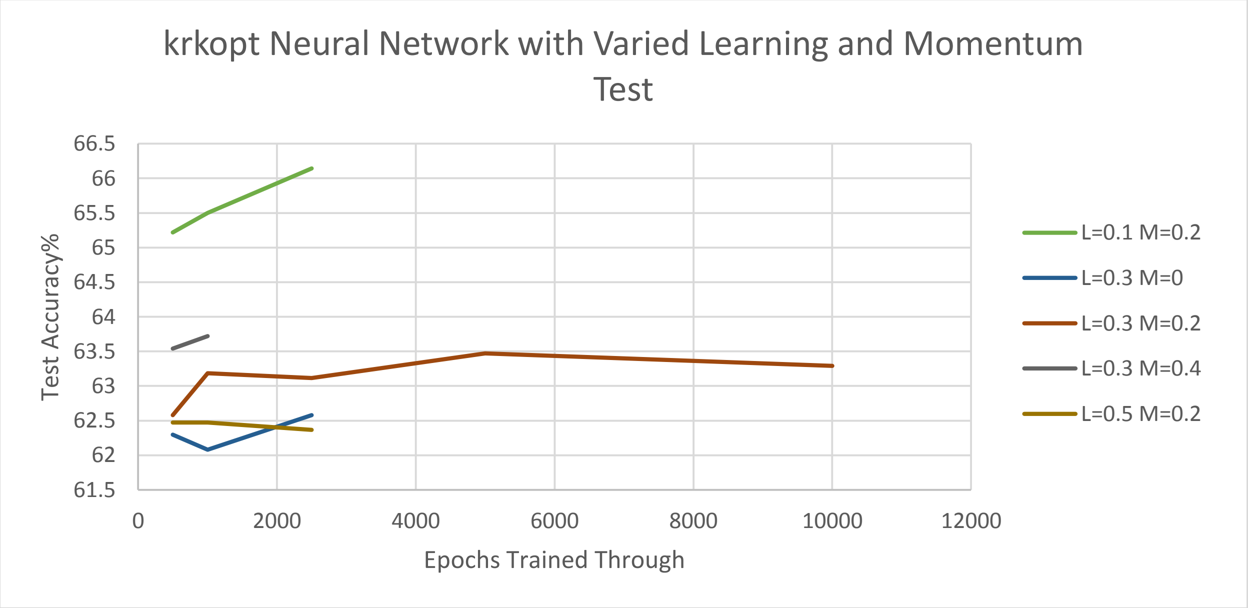
The first thing to note is the nature of the trees generated for krkopt. The trees are both large and most of the tree is constituted of its leaves: resulting in shallow and wide trees. Furthermore, from the confusion matrix shown in Figure 2, it can be noted that a large number of the misclassifications were related to the result of draws. These two facts can be explained by the fact that all of attributes for this model are nominal and that the output is determined by a combination of attributes in tandem rather than single attributes and thus noisy. Thus, the information gained from splitting on a single attribute is fairly insignificant in determining the output, which is why all of the trees tend to split every possible value of the attribute being evaluated into child nodes. Repeating this on every attribute would result in large, wide, and shallow. Furthermore, as a draw can result from many different starting configurations, causing them to trickle down to a wide variety of leaf nodes, rather than being restricted to one branch, which can account for the large number of misclassifications associated with it. However, it should also be noted that these trees are not overfit, as the test error continues to decrease as the trees grow larger.

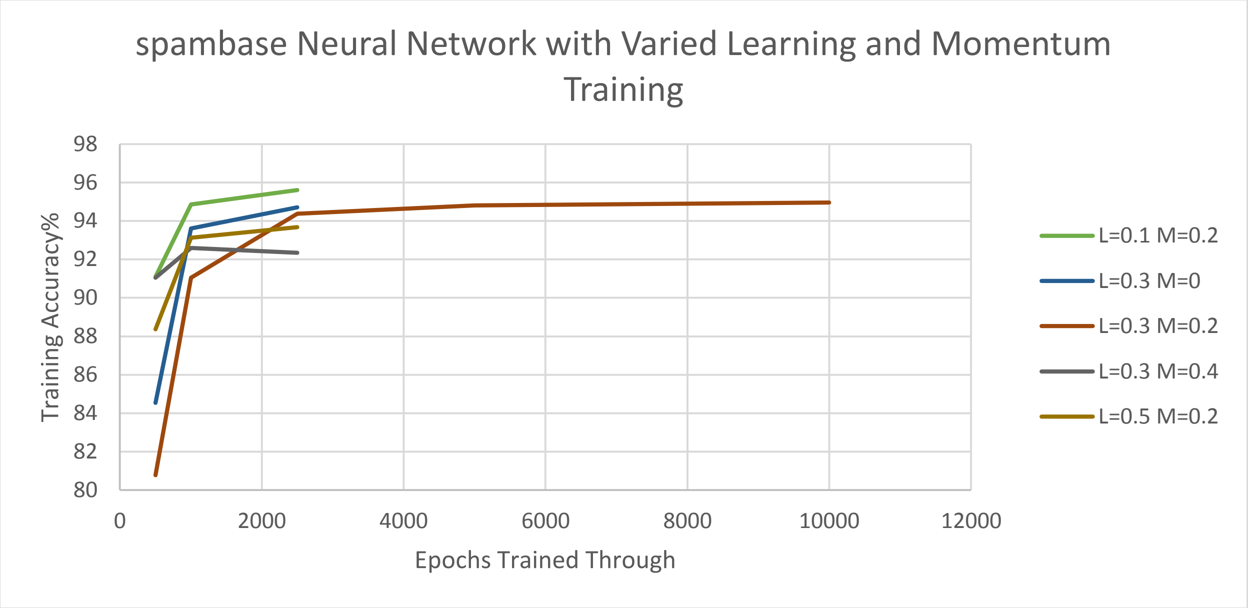
The trees produced by the spambase dataset are small, and do overfit as the trees grow larger, as evidenced by the increase in test error as the aggressiveness of pruning decreases, despite the increase in training accuracy. Additionally, decision trees produced surprising accurate results while keeping the trees small, which suggests that there are indeed a large number of irrelevant attributes in the dataset.

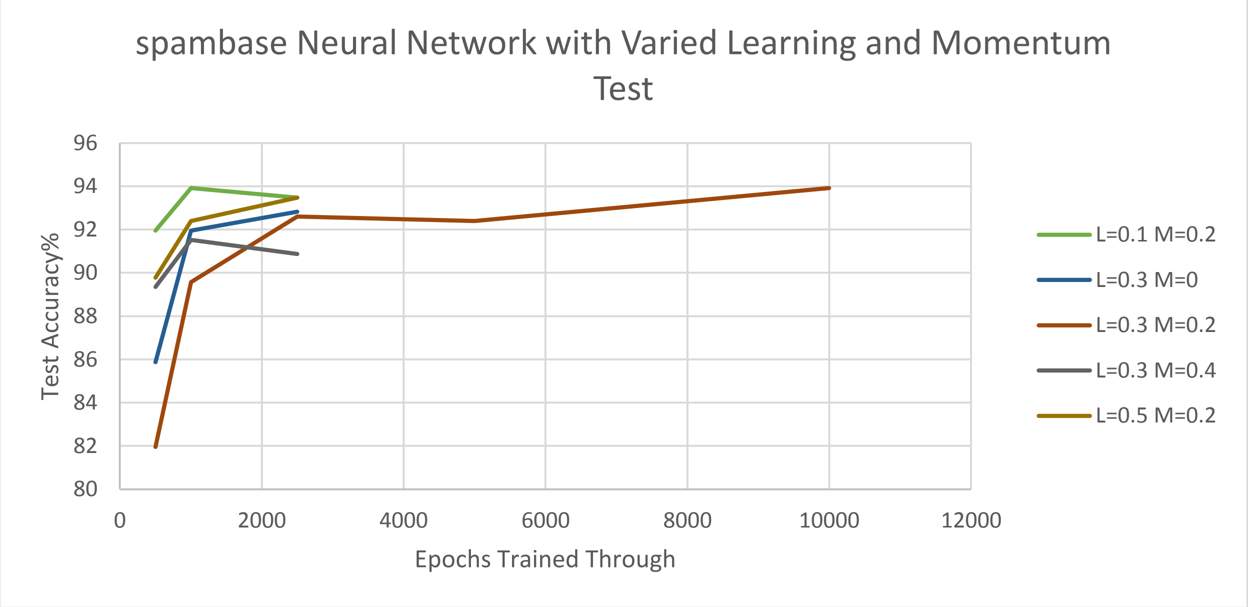
Finally, the values for both training and test times match up to what is expected of an eager learner: training times that are significantly higher than test times. However, compared to other eager learners, the training time is relatively short, and only increases with the addition of pruning and irrelevant attributes.

Neural Networks









For the purposes of this experiment, both datasets were run through a neural network with varying learning and momentum rates. The standard setting of L=0.3 and M=0.2 was run to a full 10,000 epoch, the others were terminated at 2,500 epochs. Additionally, for the sake of time, these experiments were not run with cross-validation, but rather a static split-percentage (90% training, 10% test). All of the numerical values attained from these experiments can be seen in the Neural Networks excel file included.

Neural networks are known to be able to describe any function, and thus are prone to overfit. This can be seen in the 10,000 epoch results for krkopt, as well as some of the 2,500 epoch results for spambase. The results for spambase at 10,000 epochs, however, contradict this. The transition from 2,500 to 5,000 suggest that the NN was already starting to overfit there, but the increase in test accuracy at 10,000 suggests otherwise. One explanation is that, because these experiments were run without cross-validation, the percentage split of the original dataset created two very similar datasets for training and testing, resulting in an anomaly.

Changing learning rate and momentum have significant effects on the resultant output. Decreasing learning rate tends to increase overall accuracy of the result, as evidenced by the 0.1 and 0.5 learning rate results, but also tends to overfit, as seen in the test accuracy results for spambase. Momentum has a similar effect, but can produce anomalous results such as the krkopt L=0.3, M=0.4, 1000 epochs data point, where there is an unexpectedly low test accuracy. This can be explained by the fact that the momentum is used to prevent the neural network from converging at a local minimum. Thus using too high of a momentum will result in overshooting the absolute minimum, and too low of a momentum will result in converging at local minimum. Both of which will result in less accurate results. Likewise, learning rate is used to determine magnitude of weight or bias changes during learning. Using a lower learning rate will result in more accurate results, at the cost of more time taken to reach a result.

The run times are exactly as expected from an eager learner. Training times were linearly proportional to the number of epochs trained through. 10,000 epochs on the krkopt dataset took 13976.37 seconds to complete, or just under 4 hours. Test times were, as expected, extremely short and increasing in proportion to the size of the test data set, averaging only 0.105 seconds for the spambase dataset and 0.685 seconds for the krkopt dataset.

If I could, I would repeat these expiriments with cross-validation, as it would likely produce more accurate measurements on the accuracy of the resultant neural networks.

Boosting

Prune State	Confidence	Boost Iterations	Actual Iterations Performed	Training %	Test %	Training Time	Test Time
krkopt							
Pruned	0.125	10	4	62.6176%	54.6621%	1.17s	0.24s
Pruned	0.125	20	4	62.6176%	54.6621%	1.17s	0.24s
Pruned	0.125	40	4	62.6176%	54.6621%	1.17s	0.24s
Pruned	0.25	10	10	83.6113%	61.2668%	2.42s	0.5s
Pruned	0.25	20	20	91.8591%	63.0917%	4.93s	1s
Pruned	0.25	40	40	98.4816%	64.5744%	9.44s	1.93s
Pruned	0.5	10	10	94.5644%	61.7943%	2.54s	0.52s
Pruned	0.5	20	20	99.4867%	62.242%	5.02s	1.02s
Pruned	0.5	40	40	100%	63.6299%	9.99s	2.05s
Unpruned	N/A	10	10	99.0555%	61.5697%	2.19s	0.54s
Unpruned	N/A	20	20	100%	62.7388%	4.28s	1.09s
Unpruned	N/A	40	40	100%	63.5622%	8.67s	2.24s
spambase							
Pruned	0.125	10	10	99.9348%	95.1967%	5.07s	0.06s

Pruned	0.125	20	20	99.9348%	95.2619%	6.93s	0.07s
Pruned	0.125	40	22	99.9348%	95.2619%	7.21s	0.07s
Pruned	0.25	10	10	99.9348%	95.1532%	5.76s	0.06s
Pruned	0.25	20	18	99.9348%	95.1532%	7.16s	0.08s
Pruned	0.25	40	18	99.9348%	95.175%	6.91s	0.07s
Pruned	0.5	10	10	99.9348%	94.9576%	6.95s	0.07s
Pruned	0.5	20	20	99.9348%	94.9794%	8.33s	0.07s
Pruned	0.5	40	24	99.9348%	94.9794%	9.3s	0.08s
Unpruned	N/A	10	10	99.9348%	95.0011%	4.81s	0.07s
Unpruned	N/A	20	20	99.9348%	95.1098%	6.43s	0.07s
Unpruned	N/A	40	40	99.9348%	95.1098%	8.07s	0.09s

These boosting experiments were run on the same decision tree algorithm used in the Decision Tree section of this report.

One of the most significant observations we can make from these results is that some runs ended prematurely, performing less boost iterations than it was originally set to. This is present in both krkopt and spambase datasets. This can be explained by examining the AdaBoost algorithm, which terminates prematurely if the absolute difference between 0.5 and the weighted error rate is less than a certain threshold. Decreasing this threshold would allow the algorithm to run for longer, adding more weak classifiers to the set, supposedly making the resultant classifier more accurate, but also lengthening the run time of the algorithm. The question is: why did this occur in only the pruned trees, and why did it become more common as pruning became more aggressive? The simplest answer is that the initial classifier output for more aggressively pruned trees have a weighted error closer to 0.5, and thus converge faster. This can be further explained by the fact that pruned trees have a lower accuracy than unpruned trees because of the cut branches, and thus the initial weighted error larger, and thus closer to 0.5. Unpruned or less aggressively pruned trees are generally more overfit, and have a training accuracy closer to 1, thus require more boost iterations to reach the threshold.

Furthermore, it should also be noted that boosting appears to take on the run times of the weak classifier, as many of the training and test times are linearly proportional to those shown in the Decision tree section of this report by a factor determined by the number of boost iterations executed.

k-Nearest Neighbors

Weighted	Nearest Neighbors	Training %	Test %	Training Time	Test Time
krkopt					
Unweighted	1	100%	73.8539%	0.02s	42s
Unweighted	5	80.3037%	73.0539%	0.02s	54.3s
Unweighted	10	80.3037%	73.0539%	0.02s	57.39s

Unweighted	15	80.3037%	72.9897%	0.02s	59.12s
Unweighted	20	79.876%	71.6852%	0.02s	60.77s
Weighted	1	100%	73.0539%	0.02s	41.8%
Weighted	5	100%	73.0539%	0.02s	54.33s
Weighted	10	100%	73.0539%	0.02s	57.54s
Weighted	15	100%	72.9933%	0.02s	59.21s
Weighted	20	100%	71.7636%	0.02s	60.74s
spambase					
Unweighted	1	99.9348%	90.7629%	0.01s	5.67s
Unweighted	5	93.0015%	90.3499%	0.01s	8.53s
Unweighted	10	91.8061%	89.8718%	0.01s	9.69s
Unweighted	15	90.5238$	88.6981%	0.01s	10.05s
Unweighted	20	90.3282%	88.6981%	0.01s	10.57s
Weighted	1	99.9348%	90.7629%	0.01s	5.4s
Weighted	5	99.715%	91.5453%	0.01s	8.43s
Weighted	10	99.5218%	91.1976%	0.01s	9.44s
Weighted	15	99.3697%	91.002%	0.01s	10.22s
Weighted	20	99.1958%	91.1758%	0.01s	10.42s

K-Nearest Neighbors is well known to deal with noise well, and is shown to be true in the results of its application to krkopt. Overall, on krkopt, kNN performed 9-14% better than decision trees, and by proxy boosting on decision trees, did. This is because instead of solely determining the output based on the combination of inputs, as decision trees do, kNN does a local vote using the nearest neighbors to determine the most likely output, which provides potential dissent from other local data points, rather than attempting to generalize an area by the most common result.

However, as evidenced by both datasets, overall accuracy in both training and testing, as well as the tendency to overfit, decreases as the number of neighbors voting increases. The decrease in testing accuracy can be remedied by weighting the neighbors' votes based on distance to the evaluated point, as evidenced by the weighted results from spambase, where the testing accuracy increases as the number of neighbors increased from 1 to 5. For the purposes of this experiment, weighting equal to (1/distance) was used, as the other option of weighting equal to (1 – distance) is more intended for normalized datasets, of which both krkopt and spambase are not.

Finally, the run time shown by kNN is exactly what is expected of a lazy learner: exceptionally short training times that only increase proportionally with the size of the training set, and much longer testing times. Testing times seemingly do not only scale with dataset size, but also with the number of neighbors voting.

Support Vector Machines

Kernel	Exponent/Gamma	Training %	Test %	Training Time	Test Time

krkopt					
Poly	1	44.3762%	42.9793%	765.34s	1.07s
Poly	2	88.6851%	82.3236%	3,645.03s	824.85s
Poly	3	100%	90.5203%	4,154.24s	8,031.75s
RBF	0.01	39.8574%	38.3108%	2,962.44s	9,678.47s
RBF	0.5	97.0297%	85.3885%	3,488.5s	11,150.49s
RBF	1.0	99.9644%	80.3991%	13,286.38s	8,888.31s
spambase					
Poly	1	90.5337%	90%	0.35s	0.02s
Poly	2	82.9027%	82.1739%	39.08s	4.24s
Poly	3	72.9051%	70.8696%	53.31s	5.7s
RBF	0.1	76.3101%	73.4783%	82.09s	11.63s
RBF	0.5	91.9343%	91.5217%	25s	5.53s
RBF	1.0	92.9969%	92.3913%	21.54s	4.66s

For the purpose of this experiment, I used two different kernels: a polynomial kernel varied on the degree of the polynomial and a Radial Basis Function kernel with varying gamma (which is the negative one half the inverse of the expected variance of the function). It is commonly stated that SVMs are very resistant to overfitting. However, it can be observed in these results that, they indeed can overfit if the kernel does not match the function described by the data, as seen in the krkopt tests, where an SVM with a RBF kernel of 1.0 gamma shows a decrease in test accuracy compared to that of one with a gamma of 0.5, despite an increase in training accuracy.

The most obvious observation from these results is the almost absurd run time of support vector machines on krkopt, where the relationship between input parameters, training time, and test time is inconsistent, but generally take an excessively long amount of time, even compared to that of neural networks. As such, these tests were not run with cross-validation, but rather with a static split-percentage (90% training, 10% test), for the sake of time. This suggests that krkopt is does not conform to the functions presented by the two kernels, and attempting to learn using them will ultimately produce overfit or erroneous results. It may also be because of the size of the dataset, which supposedly does not affect support vector machine runtime, as SVMs are only dependent on the edge case instances. However, due to the mismatch of the function described by the kernels and the actual function describing the King Rook King optimization problem, more piecewise parts are needed to properly describe the given data, increasing run time of both training and testing. Mismatched kernels can also decrease the accuracy of the result, as seen with the polynomial results with spambase. While the SVM produced increasingly accurate results as the exponent/gamma increased, the drastic increase in run time renders it useless in practice.