Logan Allen

902961062

Why are Chess Endgame Positions interesting?

The first data set that I chose to analyze contains chess endgame positions for games containing a white king and rook against a black king, where it's black's turn to move (Krkopt). There are six features: the rank and file of the white king, the rank and file of the white rook, and the rank and file of the black king. For the purpose of classification, we assume that both sides will play optimally, and we wish to use the features to predict the number of moves it will take for white to win. Krkopt has 18 different classes, and as this data set is made for supervised learning, each datum is labeled with its correct class.

Chess was a subject of machine learning research for many years. For a typical chess game of 40 moves, there are $\sim10^{120}$ possible permutations, which is far beyond reasonable brute computation. Companies such as IBM helped move the field further through their research into agents such as Deep Blue, which beat the chess world champion in 1997.

Krkopt is an interesting data set because the search space of possible chess games is very large, and the data set contains only a few features. We limit the size of the search space substantially by restricting our task to discerning possible outcomes at the end of a game with only three pieces on the board, but this is still a difficult task. There are a total of 28,056 examples in the data set, which only covers a subset of all of the possible positions the three pieces could take on the board. For the classes in which white is closer to winning, the data set becomes very sparse, containing only a few tens or hundreds of examples, whereas for many of the classes in which white is far from winning, there are thousands of examples. The performance of the supervised learning algorithms used in this project vary within a 13 percentile point range.

Why is Letter Image Recognition interesting?

The second data set that I chose to analyze contains capital English letters in various fonts that have been distorted in various ways to introduce variability (LIR). The features were extracted from raster scans of these letters. LIR has 16 features, all integers, which contain information about horizontal and vertical positions, edge counts, height and widths, total numbers of pixels, and statistical properties of pixel counts in rows and columns of the image.

LIR is an interesting data set because we use only numbers and statistical properties extracted from an image to create human-readable text. There are a large number of classes (26) that each example could be classified as. The 20,000 examples in this data set cover the space of possibilities quite well. Unlike Krkopt, there is an almost perfectly even distribution of examples between classes and the 16 features in the data set are much better at conveying human domain knowledge. The performance of the algorithms varies within a 20 percentile point range.

Why are the two data sets interesting together?

These two data sets allow excellent comparisons of the algorithms used, and of the contexts in which these algorithms perform well. Krkopt has a hypothesis space that is much more easily modeled as a tree of potential outcomes than as a function in a Euclidian space. It's intuitive to think of a chess game as being a chain of cause and effect — if the other player moves here, then I should move here — or in other words, a decision tree. On the other hand, the alphabetical letters in LIR are perceived by humans as connected lines in a 2-dimensional space, and can intuitively be modeled better by functions of those lines than as trees of relations of their properties.

Krkopt also differs from LIR in that the data in Krkopt is much more sparse in relation to the total size of the hypothesis space than that seen in LIR. Krkopt has a very unequal distribution of examples for each classification, where LIR has an almost perfectly equal distribution. Krkopt has only a few features containing the absolute bare minimum amount of domain knowledge, whereas LIR is much more rich in terms of number of features and the amount of feature engineering involved in determining which features to include in the data set.s

The first few experiments in this project used WEKA's percentage split feature in an attempt to have various training / testing splits (I tried 30, 50, and 70% respectively). However, I learned that this feature randomly selects a percentage of the data examples to be your training data, then tests against the entirety of the data set. Towards the end of this assignment, I learned how to correctly split up the data into separate training and testing set files using randomization and percentage filters. The tables that I did correctly contain both testing percentile and training percentile, whereas the others contain only percentile correct.

The results in the following table use the best hyperparameters found during my experiments.

krkopt (70% Training/ 30% Testing)	Testing % Correct	Training % Correct	Elapsed Time Training (s)
Boosting (J48 Pruned, 10 iterations)	70.56	98.66	4.72
J48 Pruned (C = 0.25)	65.7	83.74	1.19
NNet (700 epochs)	58.4	61.41	207.2
SMO (Puk Kernel)	57.7	71.36	395.84
k-NN k=3	57.43	80.97	0

letters (70% Training/ 30% Testing)	Testing % Correct	Training % Correct	Elapsed Time Training (s)
Boosting (J48 Pruned, 20 iterations)	97.83	100	25.8
k-NN k=1	95.48	100	0
SMO (Puk Kernel)	95.13	97.1	53.15
J48 Pruned (C = 0.25)	87.05	95.76	0.03
NNet (400 epochs)	81.93	83.93	95.08

Decision Trees (J48)

Decision trees performed exceptionally well relative to the other algorithms in modeling Krkopt and were almost 12x more accurate than random chance. I modified the hyperparameter of pruning to discern whether pruning would increase performance on this data set. I found no substantive difference in results from using pruning for Krkopt.

When pruning a decision tree, one is imparting an inductive bias for simpler trees. This is because subtrees that contribute little information gain could be caused by noise in the data and lead to overfitting. Thus, having a simpler model will often reproduce better.

The trees discovered for Krkopt had a very high number of leaves relative to their size, and are shallow. This tells us that the trees split at many of the possible values of each attribute. As the performance of the pruned and unpruned trees are incredibly similar, we know that the trees are not overfitting the data set.

Decision trees were one of the worst algorithms at modeling LIR. All of the features in LIR are integers, and as decision trees are best suited for nominal attributes in classification tasks, they produced less accurate results for LIR than every other algorithm except neural nets. Algorithms that consider numbers to be a part of some Euclidian space with dimensions are able to gain much more information from an integer feature than a decision tree, as decision trees are less inclined towards modeling relationships between numbers.

krkopt (70% Training/ 30% Testing)	Testing % Correct	Training % Correct	Tree Size	# of Leaves	Elapsed Time Training (s)
J48 Pruned	65.7	83.74	4593	2623	0.45
J48 Unpruned	65.68	86.03	5589	3238	0.6

letters (70% Training/ 30% Testing)	Testing % Correct	Training % Correct	Tree Size	# of Leaves	Elapsed Time Training (s)
J48 Pruned	87.05	95.76	2021	1011	0.7
J48 Unpruned	86.82	96.07	2219	1110	0.85

k-Nearest Neighbors:

k-Nearest Neighbors is a lazy learner, so it trains instantaneously, but takes longer to test against than the other algorithms; overall, it was very fast, and completed in about 10 seconds. k-Nearest Neighbors assumes that data examples correspond to some n-dimensional Euclidian space. Krkopt is very difficult to model in a Euclidian space without some form of feature engineering, or special way of representing domain knowledge numerically. This is verified by our results, as k-Nearest Neighbors was the worst performing algorithm for Krkopt.

On the other hand, k-Nearest Neighbors was the second best performing algorithm for LIR. LIR is made up of entirely numeric attributes, has many more features than Krkopt, and each class has an equally large number of examples. k-Nearest Neighbors functions very well with numeric attributes as it models each item as being in an n-dimensional Euclidian space. It was most likely also helpful that a single class of capital letter does not vary that significantly even when written in different fonts, so a new letter will most likely be of the same class as the most similar example in the training set.

	% Correct	Elapsed Time Training (s)
krkopt k=1	55.36	0
krkopt k=2	56.31	0
krkopt k=3	57.22	0
krkopt k=4	56.81	0
krkopt k=5	55.15	0
letters k=1	95.08	0
letters k=2	93.76	0
letters k=3	94.68	0
letters k=4	94.39	0
letters k=5	94.43	0

Boosting

Boosting with pruned decision trees was the most effective algorithm for Krkopt. Decision trees are weak learners for Krkopt, as they have an accuracy higher than random chance. Decision trees were already one of the highest performing algorithms for Krkopt, but boosting allows us to make them even more effective.

As boosting is an ensemble algorithm, we iteratively learn our classifier by generating multiple trees. At each iteration, the boosting algorithm focuses more on the harder problems that have not been correctly classified yet. Thus, over time, our trees get better and better at learning the harder to solve data from the data sets. We see that boosting performs better with J48 algorithms that implement pruning than those that do not, because this helps the decision trees in not overfitting the data. This shows that our induction bias for simpler hypotheses is imparting useful domain knowledge.

Boosting is also the best performer for LIR, performing significantly better than decision trees on their own. Boosting is held back from better performance by the poor fit between decision trees and integer features discussed earlier. Changing the base learner could improve the performance of boosting further for this problem. I decided to try out increasing the number of iterations for boosting, as the decision trees still qualify as weak learners, and to my surprise, the performance increased drastically.

krkopt (70% Training/ 30% Testing)	Testing % Correct	Training % Correct	Elapsed Time Training (s)
Boosting (J48 Pruned, 10 iterations)	70.56	98.66	6.21
Boosting (J48 Unpruned, 10 iterations)	69.93	99.53	4.66

letters (70% Training/ 30% Testing)	Testing % Correct	Training % Correct	Elapsed Time Training (s)
Boosting (J48 Pruned, 10 iterations)	94.75	100	9.4
Boosting (J48 Pruned, 20 iterations)	96.05	100	15.56
Boosting (J48 Pruned, 30 iterations)	97.83	100	25.8
Boosting (J48 Unpruned, 10 iterations)	94.67	100	6.53

Support Vector Machines

Support vector machines proved to be an effective technique that was toward the middle of the pack for both data sets. Kernels allow one to project from some other highly dimensional space onto a 2d space. Switching out kernels gives one great flexibility over the hypothesis space, allowing a more direct incorporation of domain knowledge into problem-solving.

In Krkopt, we saw that attempting to model possible checkmates in a chess game through separating data linearly with the Poly kernel was an exceptionally poor technique, as it is very difficult to model a chess game and all the possibilities that entails using a linear function. However, PUK as a kernel was a significantly higher performer in both data sets.

In LIR, while the Poly kernel was still the lower performing function, it was only slightly lower than Puk. This leads me to draw the intuitive conclusion that letters are modeled very well by functions in 2d space. The Poly kernel is much more accurate when the exponent parameter is increased and as both the testing percentage and training percentage it can be increased some further amount without overfitting.

SVM with the Puk kernel took a massive amount of time to train on Krkopt, in some cases almost as much time as all the other functions put together. If I had put more time into optimizing the hyperparameters or trying more kernels for SVM, I believe its performance could have been much higher for Krkopt.

	% Correct	Elapsed Time Training (s)
krkopt SMO (Puk)	55.59	1378.55
krkopt SMO (Poly exp=1)	34.65	90.06

letters (70% Training/ 30% Testing)	Testing % Correct	Training % Correct	Elapsed Time Training (s)
SMO (Poly exponent=1)	81.73	82.43	4.5
SMO (Poly exponent=2)	88.07	88.90	31.07
SMO (Poly exponent=3)	92.87	94.88	31.38
SMO (Poly exponent=4)	94.72	97.83	32.03
SMO (Puk)	95.13	97.1	43.44

Neural Networks:

Neural nets can theoretically model any hypothesis space, no matter how complex, given enough data and time. Neural nets are eager learners, as opposed to k-Nearest Neighbors, which is a lazy learner. An eager learner attempts to generalize from all the training data to a global function that it can use to classify new data and takes a while to train, but very little time to classify.

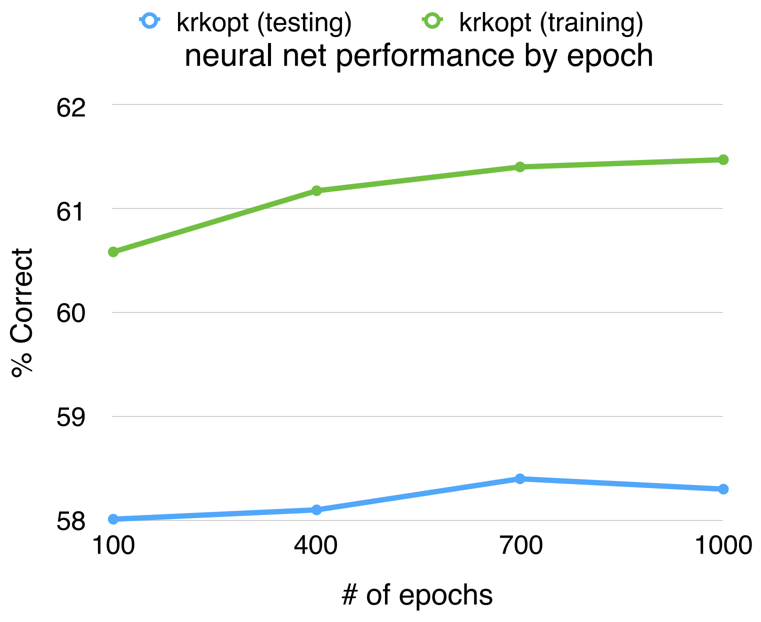
In Krkopt, neural nets were the middle performer. I modified the number of epochs to run the neural net for, and found a local maxima at 700 epochs before it began to overfit.

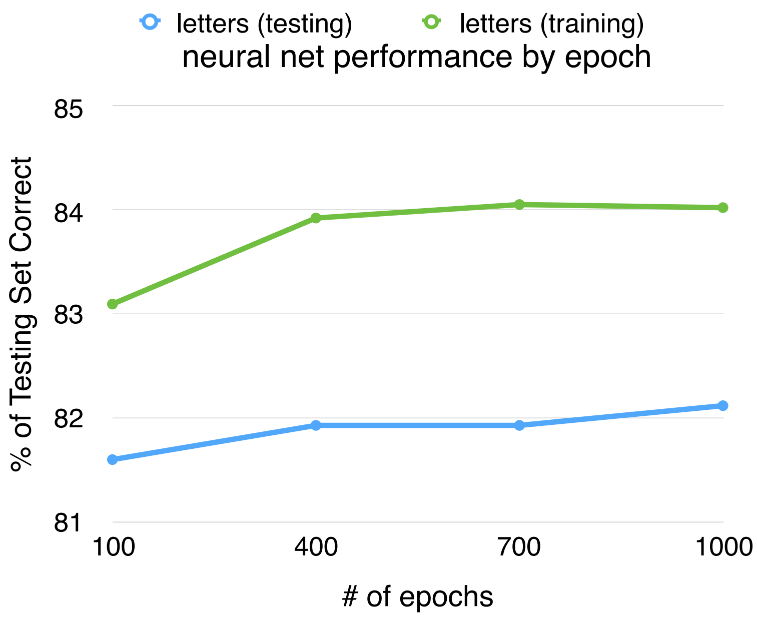
In LIR, neural nets were the algorithm with the worst performance. I modified the number of epochs to run the neural net for, and the best result I received was at 1000 epochs, the maximum number I tested. It could potentially be increased further without overfitting. I modified the momentum number from 0.2 to 0.3, and saw a decrease in accuracy by 2 percentile points.

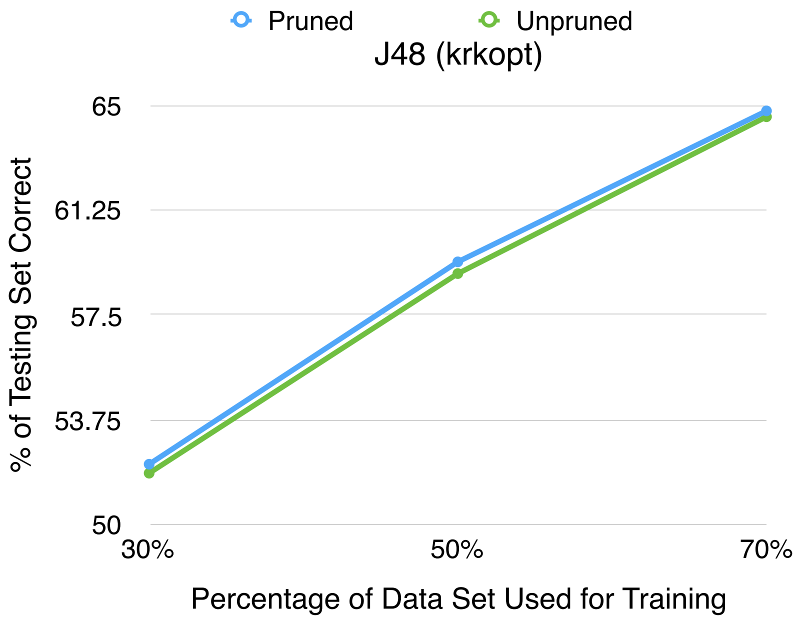
krkopt (70% Training/ 30% Testing)	Testing % Correct	Training % Correct	Elapsed Time Training (s)
NNet (100 epochs)	58.01	60.59	26.28
NNet (400 epochs)	58.1	61.18	126.74
NNet (700 epochs)	58.4	61.41	207.2
NNet (1000 epochs)	58.3	61.48	315.19

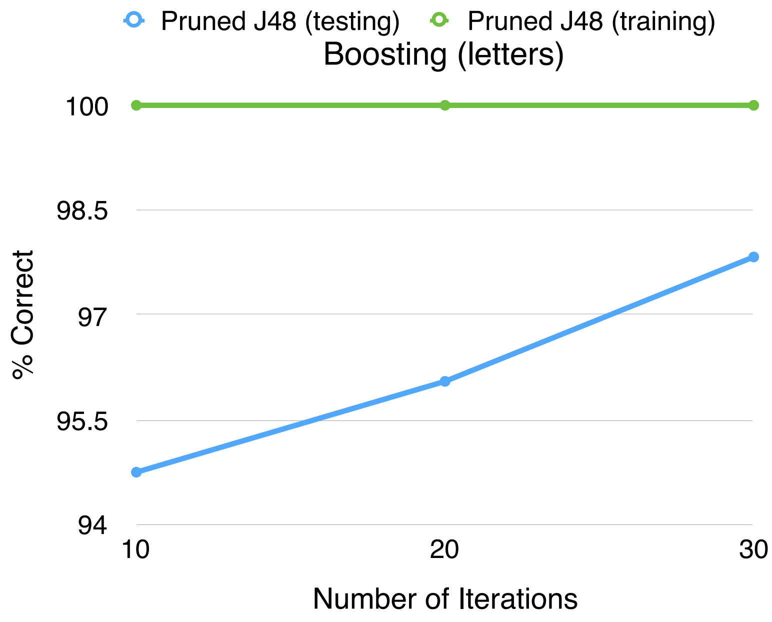
letters (70% Training/ 30% Testing) M=0.2	Testing % Correct	Training % Correct	Elapsed Time Training (s)
NNet (100 epochs)	81.6	83.1	29.7
NNet (400 epochs)	81.93	83.93	118.72
NNet (700 epochs)	81.93	84.06	273.73
NNet (1000 epochs)	82.12	84.03	362.46

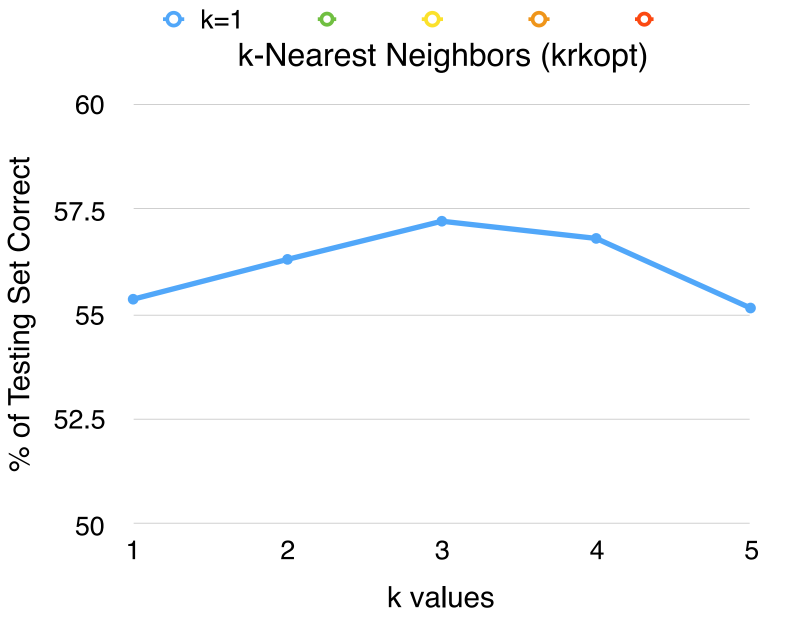
letters (70% Training/ 30% Testing) M=0.3	Testing % Correct	Training % Correct	Elapsed Time Training (s)
NNet (1000 epochs)	80.68	83.25	296.95

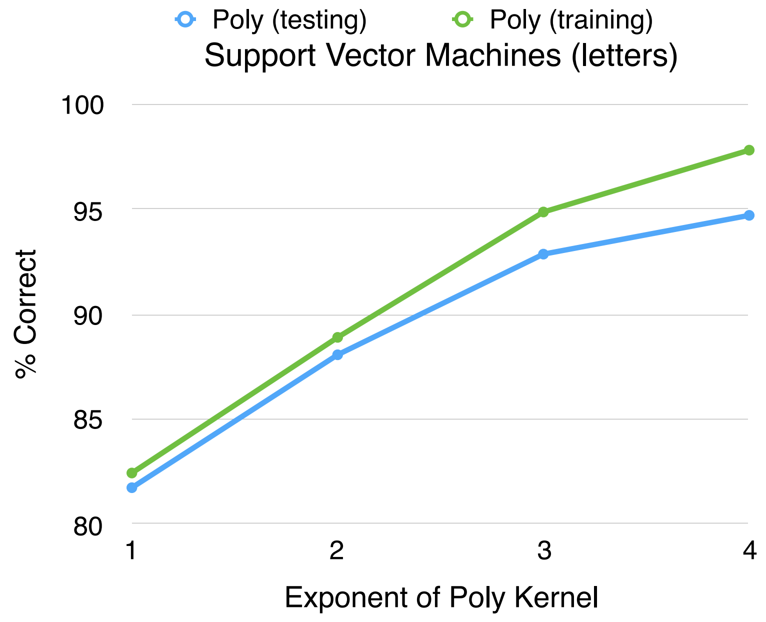

krkopt (testing)
krkopt (training)
neural net performance by epoch
% Correct
62
61
60
59
58
100
400
700
1000
# of epochs

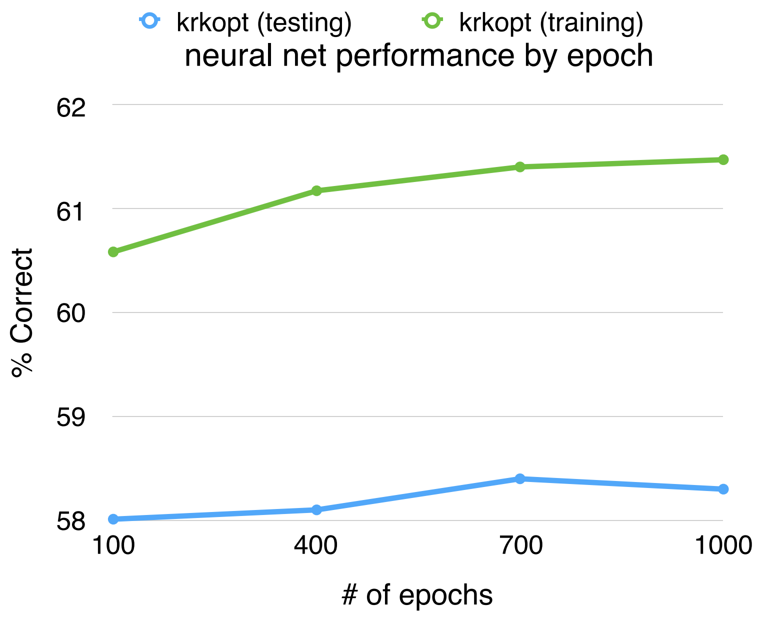

letters (testing)
letters (training)
neural net performance by epoch
% of Testing Set Correct
85
84
83
82
81
100
400
700
1000
# of epochs


Pruned
Unpruned
J48 (krkopt)
% of Testing Set Correct
65
61.25
57.5
53.75
50
30%
50%
70%
Percentage of Data Set Used for Training

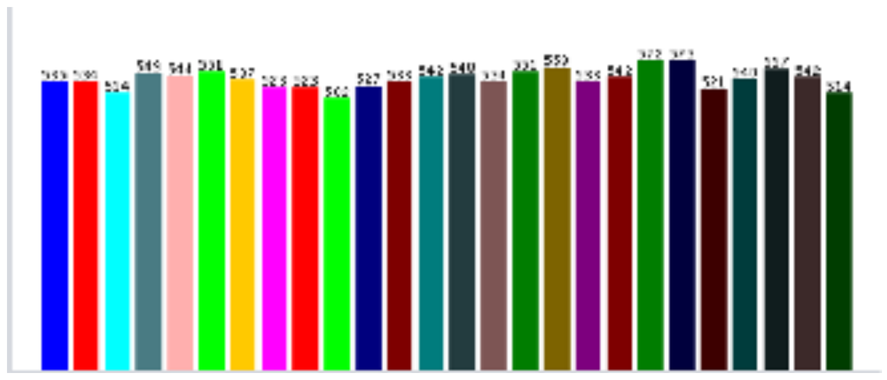

Pruned J48 (testing)
Pruned J48 (training)
Boosting (letters)
% Correct
100
98.5
97
95.5
94
10
20
30
Number of Iterations


k=1
k-Nearest Neighbors (krkopt)
% of Testing Set Correct
60
57.5
55
52.5
50
1
2
3
4
5
k values

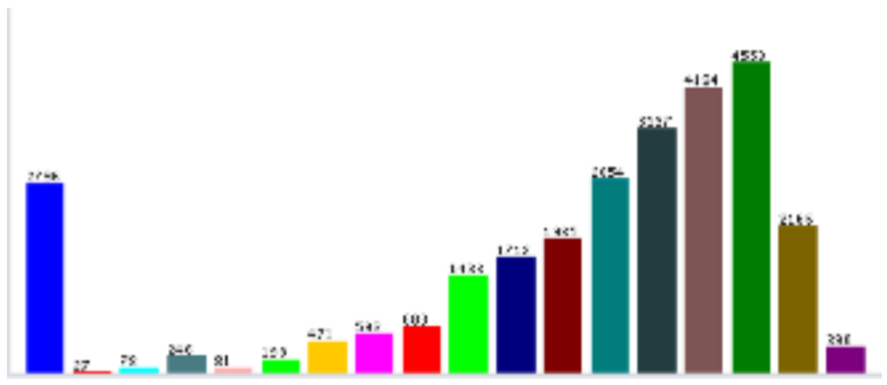

Poly (testing)
Poly (training)
Support Vector Machines (letters)
% Correct
100
95
90
85
80
1
2
3
4
Exponent of Poly Kernel



Letters Class Distribution:



Krkopt Class Distribution:


2796
27
78
246
81
198
471
592
683
1433
1712
1985
2854
3597
4194
4553
2166
390