Utkarsh Garg
CS 4641 - Machine Learning
Assignment #1
2/8/2015

Supervised Learning Analysis Report

Contents:

Classification Datasets

I have chosen two very different classification datasets; one is a letter recognition dataset based on provided attributes and the other is classifying DNA sequences into either of 2 different classes or as belonging to neither.

Letter Recognition Dataset:

The dataset itself:

This dataset contains labeled examples of letters of the english alphabet, followed by a list of 16 attributes. It contains 20,000 instances which were manually divided into testing and training sets using a 66% training ratio. There are no missing values. The attributes (all integers) used are defined below:

Horizontal position of box
Vertical position of box
Width of box
Height of box
Total # on pixels
Mean x of on pixels in box
Mean y of on pixels in box
Mean x variance
Mean y variance
Mean x y correlation
Mean of x * x * y
Mean of x * y * y
mean edge count left to right
Correlation of x-ege with y
Mean edge count bottom to top
Correlation of y-ege with x

The number and nature of attributes seems very extensive; the attributes include original readings from the data and then outputs of statistical functions of the same readings.

Attribution: David J. Slate, Odesta Corporation; 1890 Maple Ave; Suite 115, Evanston, IL 60201, from the UCI database.

Why it is interesting:

Having computers identify text from images is a very established machine learning problem with its complexity arising from the fact that there are tons of different fonts and even worse, many different handwritings that make it hard to identify a discernible pattern. Using the supervised learning algorithms that we are testing for this assignment on this problem is interesting because it gives an insight into how much accuracy basic algorithms can achieve on a legacy problem like this.

Splicing Dataset:

The dataset itself:

This dataset is used to identify two different types of splice junctions in DNA sequences; the exon/intron and the intron/exon junctions which are splices of DNA that are removed in protein formation. The dataset has been modified to eliminate useless information from the file(identification ids of each sequence). It now consists of a label (IE/EI/N), followed by a sequence of 60 characters, representing the DNA splice to be classified. There are 3175 instances and I have manually divided them into training and testing datasets (66% training). There are no missing values.

Attribution: G. Towell, M. Noordewier, and J. Shavlik, {towell,shavlik}@cs.wisc.edu, noordewi@cs.rutgers.edu, from the UCI database.

Why is it interesting:

Because of the vast amount of computational complexity, I have been interested in the applications of machine learning to genomics and sequencing, and machine learning as a field itself has been gaining a lot of traction in this area. It has been interesting to see how these legacy algorithms, not designed specifically for this purpose perform on this dataset.

Algorithms:

KNN

I ran KNN for both datasets in the following ways:

1. Run for $1 < k < 50$
2. Run for training set 66%, 33%, 50% of the total given data.

The report includes comparison graphs of the in-sample rms (root mean square) and the out-of-sample rms error plotted against k-values for the **letter prediction** dataset. For the **DNA splicing** dataset however, I have included scatterplots as it was a binary classification and a line graph did not make sense. It is very interesting to see how the effectiveness of KNN varies with the number of nearest neighbors; the two data sets give completely opposing conclusions.

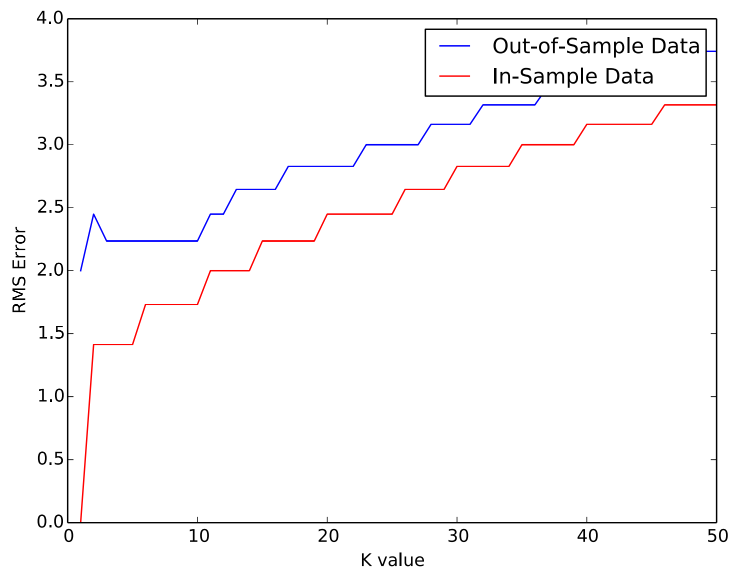


Figure 1

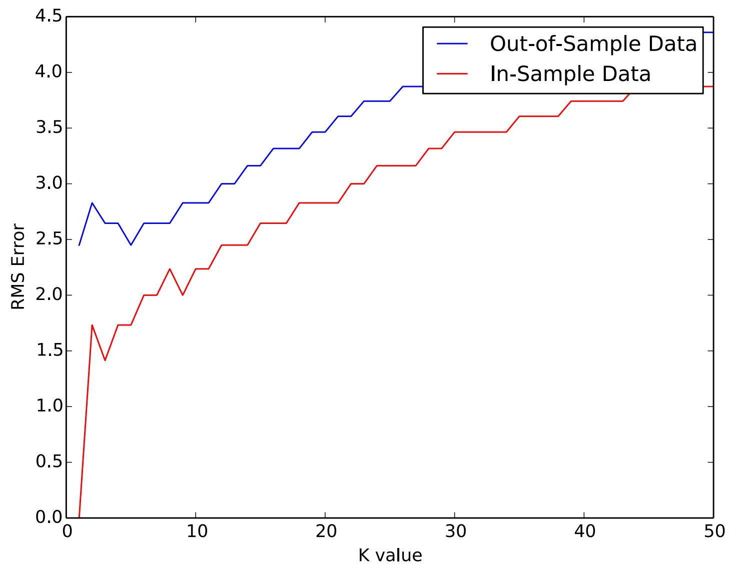


Figure 2

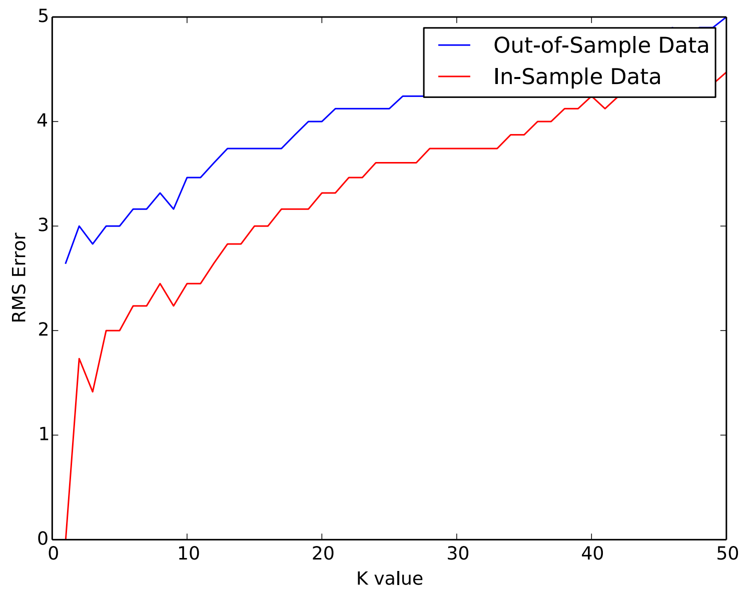


Figure 3

These are the graphs for the KNN algorithm for values of K from 1 to 50 for the **letter prediction** dataset. The graph maps k values to the root mean square error of predictions of the test set and the training set itself on the trained model. **Figure 1** represents training with 66% data, **Figure 2** represents training with 50% data and **Figure 3** represents training with 33% data.

As is apparent from all three graphs, the performance decreases with increasing k. This signifies that every element of the dataset may have very little correlation with every other element in the dataset, and definitely very little correlation with its immediate neighbors. The performance does not spike in between, just declines constantly from k=1. In all three cases in the table below, performance in terms of both accuracy and time declines with increasing k.

%Data Trained On	Min. Training Error (rms)	Max. training Error (rms)	Min. Testing Error (rms)	Max. Testing Error (rms)	Min. Time Taken (s)	Max. Time Taken (s)	Optimal K
66%	0	3.3166	2	3.7416	1.688	3.639	k = 1
50%	0	3.8729	2.4494	4.3588	1.568	3.586	k = 1
33%	0	4.4721	2.6457	5	0.737	1.666	k = 1

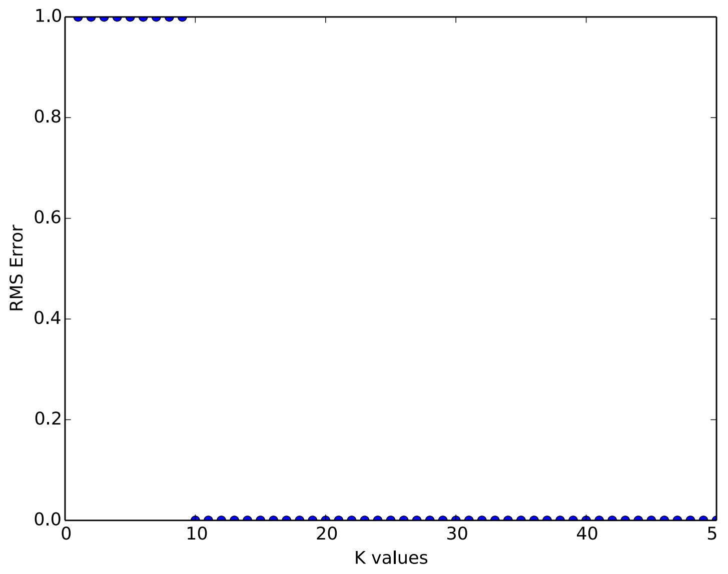


Figure 4

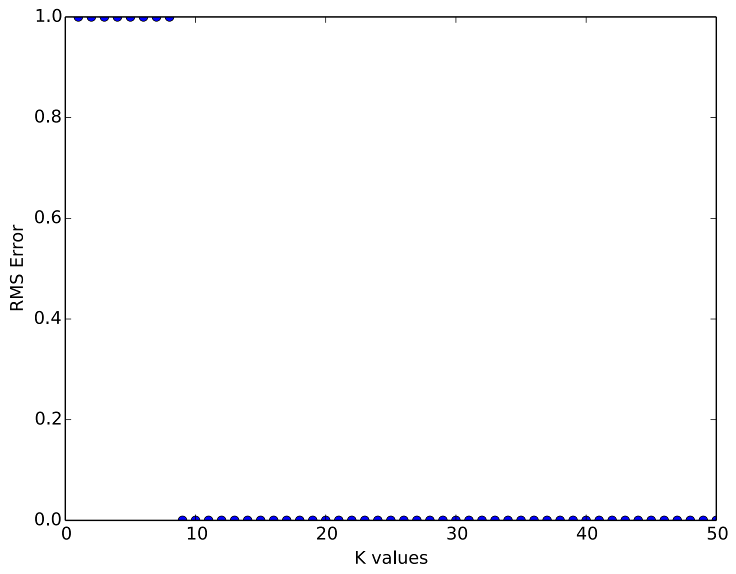


Figure 5

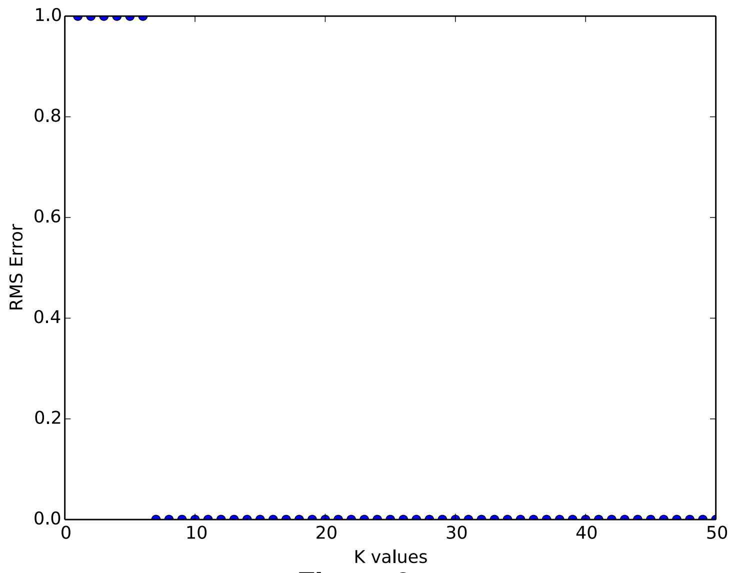


Figure 6

These are the graphs for the KNN algorithm for values of K from 1 to 50 for the **DNA splice classification** dataset. The graph maps k values to the error of predictions of the test set on the trained model.
Figure 1 represents training with 66% data,
Figure 2 represents training with 50% data and
Figure 3 represents training with 33% data.

The graphs are scatter plots representing the error in classification. As is apparent from all three graphs, they follow the same trend, up to a certain point, all classification is incorrect, but after different values of k(all within an error of 2), the classification is always correct. A generalized version may be that increasing k leads to increasing accuracy, though the time taken also increases. The first observation completely opposes that made for dataset 1, where increasing k constantly led to decreasing accuracy of the results.

%Data Trained On	Min. Training Error (rms)	Max. training Error (rms)	Min. Testing Error (rms)	Max. Testing Error (rms)	Min. Time Taken (s)	Max. Time Taken (s)	Optimal K
66%	0	0	0	1	0.433	0.550	k = 10
50%	0	0	0	1	0.388	0.444	k = 9
33%	0	0	0	1	0.154	0.319	k = 7

SVM (Support Vector Machine)

For the SVM experiments, I used the linear, radial bias function and sigmoid kernels for the **letter prediction** dataset and only the radial bias function and sigmoid for the **DNA splice classification** experiment. For some reason (and I haven't been able to figure out why), the linear kernel did not terminate for the **splice** dataset even after 12 hours of running. I used the sklearn implementation.

%Data Trained On	Kernel	Training Error	Testing Error	Time Taken (Train + Test) (s)
66%	Linear	3.316	3.872	8.827
66%	Radial Bias Function	0	1.732	20.763
66%	Sigmoid	7.483	7.483	24.005
50%	Linear	3.316	3.872	5.734
50%	Radial Bias Function	0	1.732	13.120
50%	Sigmoid	9.899	9.899	13.067
33%	Linear	3	3.872	2.885
33%	Radial Bias Function	0	2	6.620
33%	Sigmoid	12.806	12.845	5.980

Letter Prediction Dataset - As is visible from the table, I ran all three kernels with varying amount of training data. A very interesting thing to note is that the sigmoid function learnt absolutely nothing. Apart from the intriguing fact that its training error is so high, it is also interesting to note that the testing error is either greater than or equal to the training error. This means there was no learning involved with the sigmoid kernel. A very simple explanation for that is the behavior of the sigmoid function in contrast with the required classification. The required classification problem is to classify as one of 26 possible labels; a task a function as simple as the sigmoid function cannot cover.

Coming to the Linear kernel, the difference between the training and testing errors is minimal i.e. there is no or very little overfitting because the model fits the test data almost as well as it fits the training data. On the other hand, the radial bias function has no error for the training data, a sure shot sign of overfitting. This is further clear from the fact that the difference in the test error and the train error is very significant, compared to other kernels and on its own as well. In terms of time complexity, the linear kernel performs the best and the sigmoid and rbf rally together and don;t have much of a time difference. Another notable act is that as the amount of training data decreases, the training error in the linear kernel tends to decline whereas the testing error remains constant. This is the opposite behavior of what we expect from overfitting, thus further confirming its absence.

%Data Trained On	Kernel	Training Error	Testing Error	Time Taken (Train + Test) (s)
66%	Radial Bias Function	0	0.076	0.254
66%	Sigmoid	0	0.076	0.254
50%	Radial Bias Function	0	0.057	0.191
50%	Sigmoid	0.477	0	0.123
33%	Radial Bias Function	0	0.041	0.131
33%	Sigmoid	0.459	0	0.094

DNA splice classification dataset - Because of the reasons mentioned at the beginning of this section, the table only reports data on the rbf and sigmoid kernels. Very surprisingly, one can note here that the sigmoid function, at times, has higher training error than testing error, and overall, the error in the sigmoid kernel SVM is minimal. It performs well both on time and performance, as opposed to the previous dataset. This can be attributed to the difference in the two classification problems; while the **splice** dataset poses as binary classification problem, the **letter** dataset poses a multivariate classification problem. The difference of the core of the two problems and the inherent behavior of the sigmoid function can be held responsible for the almost opposing behaviors of the two datasets.

Decision Tree

For both, decision trees and boosting, I used weka's J48 implementation, as is, with and without pruning. All the data was run with a 66% training split. The two tables below resent the data from running the decision tree algorithm on my two datasets with varying confidence levels and with or without pruning.

Pruning	Confidence	# of leaves	Tree size	Accuracy	RMS error	Relative Absolute Error	Time (s)
Pruned	0.25	1226	2451	86.456%	0.0958	15.802%	1.28
Pruned	0.50	1267	2533	86.456%	0.0958	15.726%	1.14
Pruned	0.75	1303	2605	86.456%	0.0958	15.507%	26.93
Unpruned	-	1337	2673	86.397%	0.0958	15.467%	0.75

Letter Classification Dataset

For the **letter classification** dataset, Pruning evidently decreases tree size and number of leaves, and by the definition of pruning, this is the behavior we expect. on the other hand, this reduction in the tree size and the number of leaves has little to no effect on the reported error and accuracy of the classification. Also, pruning with higher and higher confidence simply increases the time it takes to form the model and does not improve results noticeably. Based on these facts, it is safe to conclude that decision trees, though are a decent model, because of their reasonable accuracy (~86%), don't have their performance improved by pruning in this case. This is expected behavior because of the large number of classes an element can be classified into, as opposed to just binary classification.

Pruning	Confidence	# of leaves	Tree size	Accuracy	RMS error	Relative Absolute Error	Time (s)
Pruned	0.25	136	181	93.698%	0.1997	14.600%	0.02
Pruned	0.50	205	273	93.049%	0.2024	13.447%	0.04
Pruned	0.75	346	461	90.918%	0.2347	14.800%	1.21
Unpruned	-	349	465	90.918%	0.2347	14.800%	0.02

DNA Splice Dataset

For the **DNA Splice** dataset, which is a binary classification problem, Both pruning and the confidence level cause high variability. Although pruning itself reduces the tree size and the number of leaves significantly, with a high confidence level, it does not do much for the accuracy of the classification and the errors reported. As the confidence level is decreased and the pruning is improved, the number of leaves and tree size further drop significantly and the accuracy increases by a fir amount. It is also interesting to note the apart from the anomaly of the time taken to model pruning with confidence 0.75, pruning does not increase the modeling time by much.

```
=== Confusion Matrix ===

   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   p   q   r   s   t   u   v   w   x   y   z   <-- classified as
 238   1   5   0   1   5   0   1   2   0   3   1   2   2   0   0   0   0   0   2   1  10   2   1   0   2 |   a = T
   2 213   2   0   1   3   2   0   7   0   3   0   0   5   0   0   0   0   1   2   0   0   1   0   0   2 |   b = I
   0   1 223   3   1   4   3   2   1   0   6   2   1   0   0   9   0   0   1   1   0   1   1   1   0   1 |   c = D
   0   0   8 256   0   0   0   1   0   1   0   1   1   1   0   0   2   0   1   1   4   0   0   3   1   0 |   d = N
   1   1   3   0 206   5   6   0   0   3   0   4   0   0   6   2   3   4   0   3   1   0   6   1   4   0 |   e = G
   4   6   2   1   2 192   7   0   0   0   3   4   2   2   1   3   0   4   0   4   0   1   2   0   0   5 |   f = S
   2   3   5   2   2   3 207   0   1   0   4   1   9   2   0   4   0   1   0   2   6   1   1   0   0   1 |   g = B
   0   0   4   2   3   0   0 259   0   0   1   0   1   0   0   1   0   0   0   0   0   0   0   0   1   1 |   h = A
   2   6   2   0   0   2   2   0 226   0   1   2   0   1   0   1   0   2   2   0   0   0   0   0   2   1 |   i = J
   0   0   2   4   0   0   2   1   0 260   0   1   2   2   0   0   1   1   0   1   1   0   0   3   4   0 |   j = M
   2   1   1   0   0   2   0   0   0   0 237   0   1   0   0   1   0   4   0   3   0   0   0   0   5   1 |   k = X
   2   0  12   1   4   2   1   0   0   2   3 218   3   0   0   1   1   0   2   0   0   0   4   4   0   0 |   l = O
   2   0   7   0   2   1   7   1   2   2   0   1 203   2   1   8   1   2   0   4   0   1   1   0   3   1 |   m = R
   2   6   1   0   1   1   6   1   1   1   1   0   1 237   2   0   0   0   9   1   1   6   0   0   0   0 |   n = F
   0   2   0   0   5   1   0   0   0   0   2   4   0   0 208   3   0   0   3   7   0   2   0   0   0   1 |   o = C
   0   0  13   2   1   0  11   1   0   1   3   3   7   0   0 204   0   0   1   0   0   1   0   2   5   0 |   p = H
   0   0   2   5   0   1   0   0   0   2   0   4   1   0   0   3 244   0   2   0   3   0   0   8   0   0 |   q = W
   0   3   1   0   1   0   0   1   0   0   1   0   2   0   0   0   0 246   1   1   0   0   1   0   0   0 |   r = L
   0   2   4   0   1   0   2   1   0   0   1   3   0   8   0   2   0   0 258   2   1   0   1   0   0   0 |   s = P
   2   0   1   0  13   3   1   0   0   1   3   0   0   0   4   3   0   0   1 223   0   0   6   0   3   2 |   t = E
   2   0   2   2   0   2   6   2   0   2   0   1   0   2   1   0   7   0   1   0 235   6   0   2   0   0 |   u = V
  11   3   0   0   0   1   1   2   0   0   0   0   0   2   1   2   0   0   2   0   7 205   0   1   1   2 |   v = Y
   0   0   3   0   1   1   2   0   0   0   2   5   0   0   1   2   0   2   1  10   3   2 215   0   0   1 |   w = Q
   2   0   1   9   0   0   1   0   1   5   0   2   2   0   7   3   2   3   0   1   1   0   0 253   2   0 |   x = U
   1   0   1   0   0   1   3   4   1   0   3   0   3   0   1  12   0   3   0   1   0   0   0   1 202   0 |   y = K
   2   0   1   0   0   6   1   0   2   0   1   0   1   3   1   2   0   3   0   3   0   1   2   0   0 211 |   z = Z
```

A confusion matrix from one of the decision tree algorithm results for classification into correct letters based on the provided attributes.

Boosting

For the boosting experiments, I used the J48 implementation of the weka decision tree along with AdaboostM1 boosting. For both datasets, experiments were run using varying confidence levels and with or without pruning.

Iterations	Confidence	Pruning	# of leaves	Tree size	Accuracy	Time to build (s)
10	0.25	Pruned	1312	2623	94.823%	11.84
10	0.5	Pruned	1375	2749	94.647%	11.47
10	-	Unpruned	1515	3029	94.588%	10.97
20	0.25	Pruned	1276	2551	95.823%	25.57
20	0.5	Pruned	1321	2641	95.985%	24.31
20	-	Unpruned	1488	2975	95.647%	25.06
30	0.25	Pruned	1303	2605	96.058%	37.94
30	0.5	Pruned	1334	2667	96.279%	35.89
30	-	Unpruned	1489	2977	96.088%	32.51

Letter Classification Dataset

Letter Classification Dataset - The most interesting aspect in the table above is the increasing accuracy of classification with the increasing number of iterations, regardless of confidence and pruning. It seems like confidence level and pruning fail to make an impact on the accuracy of the outcome the reported error. But, the number of iterations, when increased to 3 times increase the accuracy by about 1.5% which is a lot given the base accuracy of ~95.6%.

Iterations	Confidence	Pruning	# of leaves	Tree size	Accuracy	Time to build (s)
10	0.25	Pruned	190	253	94.3466%	0.27
10	0.5	Pruned	157	209	93.7905%	0.24
10	-	Unpruned	205	273	94.4393%	0.21
20	0.25	Pruned	7	9	94.532%	0.46
20	0.5	Pruned	43	57	94.9954%	0.44
20	-	Unpruned	118	157	94.532%	0.36
30	0.25	Pruned	13	17	94.81%	0.64
30	0.5	Pruned	40	53	94.7173%	0.68
30	-	Unpruned	85	113	94.9027	0.85

DNA Splice dataset - Unlike the letter classification dataset, the interesting portion of the above table is the massive reduction in tree size and the number of leaves on implementing pruning with a confidence level of 0.25. The increasing number of iterations only seem to increase the tim required to build the model and seem to do nothing for the accuracy of the model. In contrast, a good pruning (pruning with confidence level 0.25) reduces the tree size and number of leaves by magnitudes. This might be because of the limited number of classes an object can be classified into. Since only 2 classes exist, a good pruning greatly reduces the probability of a wrong classification and iterations with a learning rate help in this to a limited extent (that is why boosting has much better results for this data set than decision trees but increasing the number of iterations after 10 does not seem to have much effect).

Neural Networks

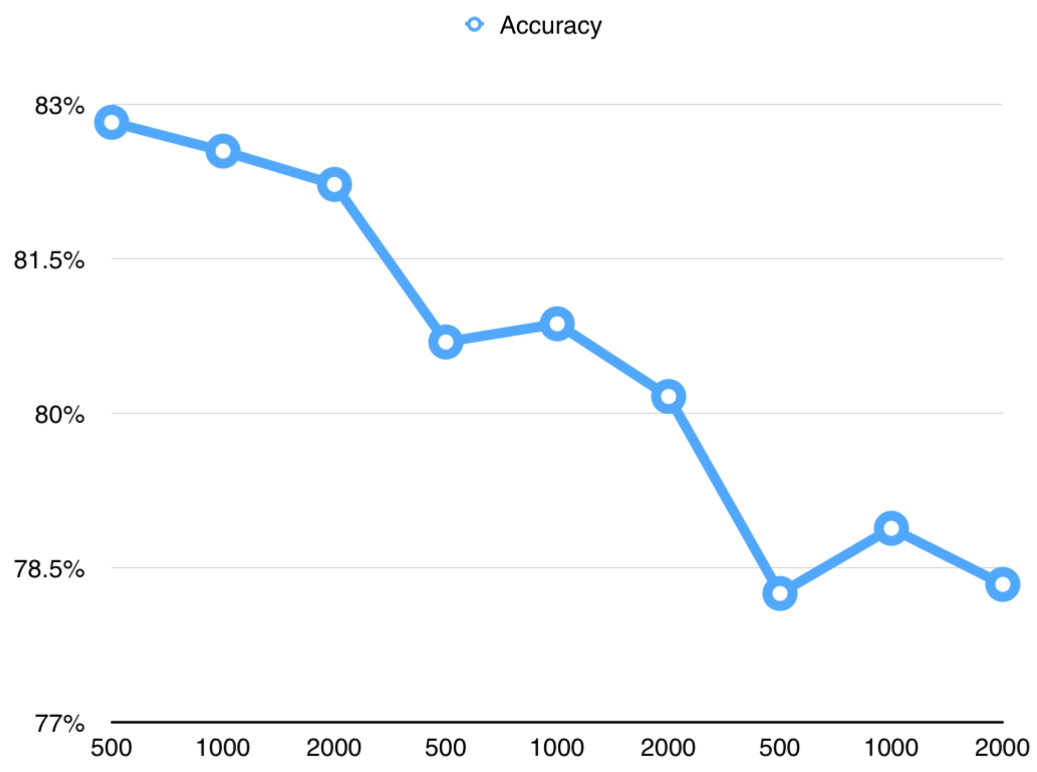
For the neural networks experiments, for both datasets I used weka's multilayer perceptron with varying number of epoch and varying learning rate and momentum.

# Epochs	Learning Rate	Momentum	Accuracy	Time (s)
500	0.3	0.2	82.8235%	220.59
1000	0.3	0.2	82.5441%	465.88
2000	0.3	0.2	82.2206%	1098.48
500	0.5	0.4	80.6912%	249.99
1000	0.5	0.4	80.8676%	473.4
2000	0.5	0.4	80.1618%	1121.35
500	0.9	0.7	78.25%	226.21
1000	0.9	0.7	78.8824%	587.94
2000	0.9	0.7	78.3382%	891.8

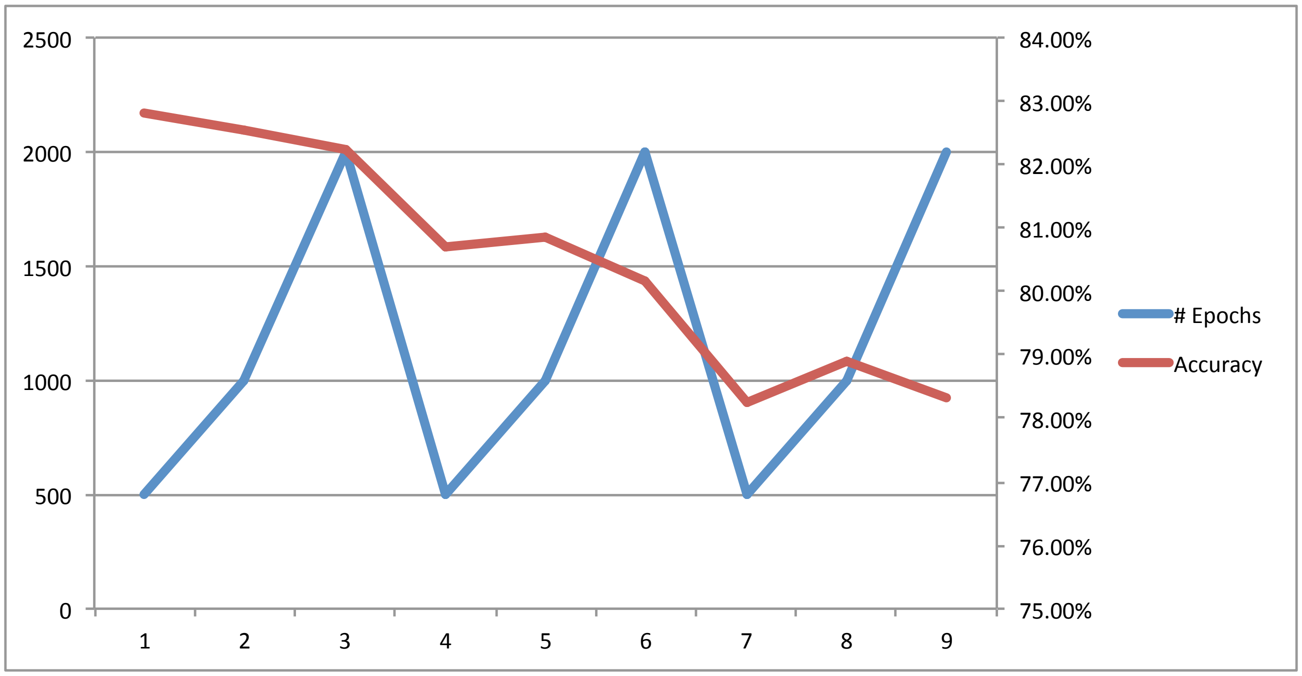
Letter Classification Dataset

There is a very discernible pattern in the table above, for the **letter classification** dataset. As the combination of the learning rate and the momentum increases, the overall accuracy declines. This happens regardless of the number of epochs. Within the epochs themselves there isn't much variation in the accuracy of the various experiments. The slight change however also follows a downward trend with increasing number of epochs. I attribute this to overfitting as the number of epochs are increasing by double every time. Also, with all three combinations of the learning rate and momentum, the time increases almost linearly with respect to the change in the number of epochs, as the number of epochs increase.

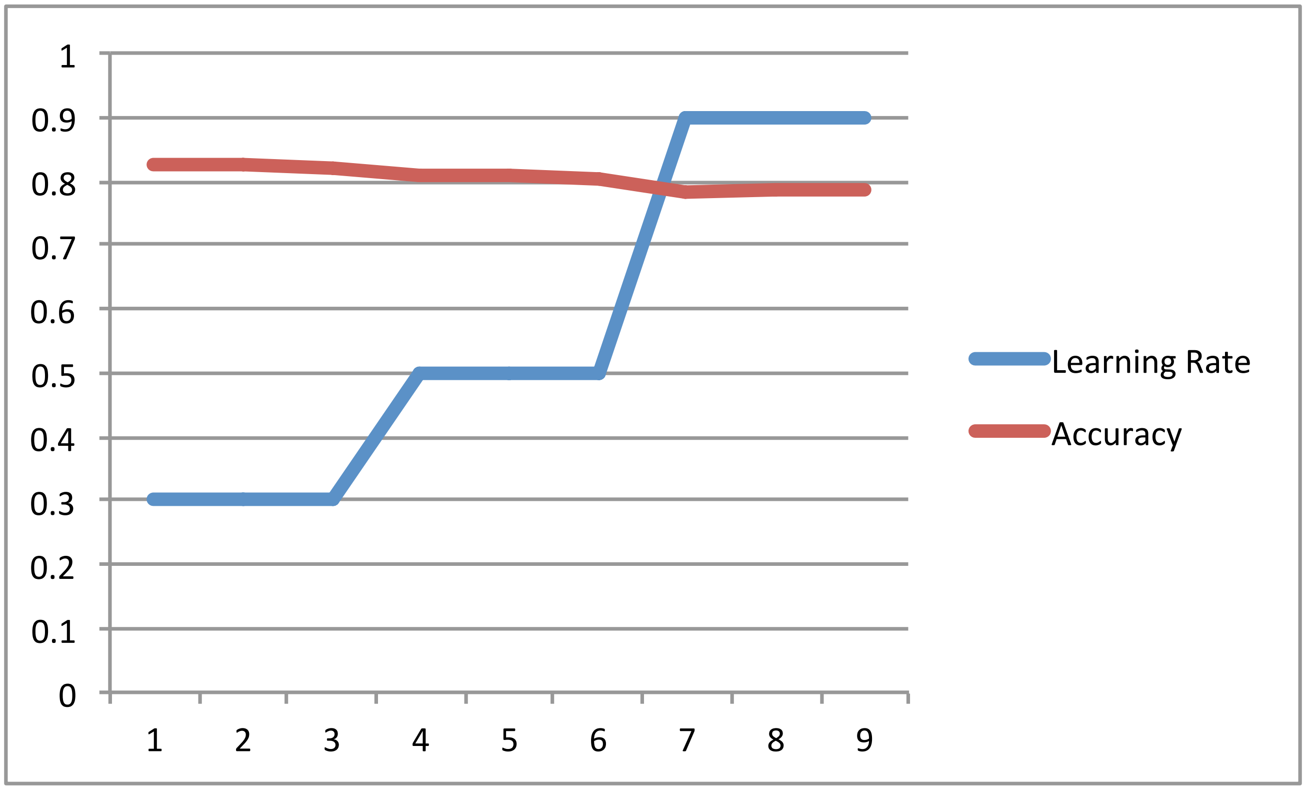
The graphs below further illustrate my point.



Graph: Accuracy vs Epochs, based on learning rate and momentum. As the parameters, learning rate and momentum increase, the accuracy constantly declines. Two of the combinations peak in accuracy for 1000 epochs, but fall overall at 2000 epochs.



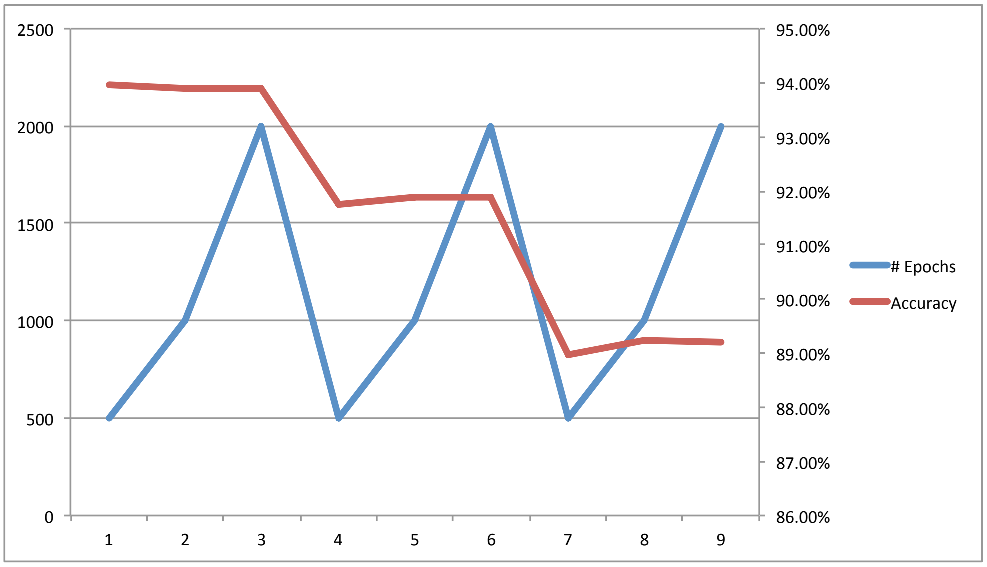
Same as above, but illustrated better with a clearly visible downward trend of accuracy, regardless of # epochs.



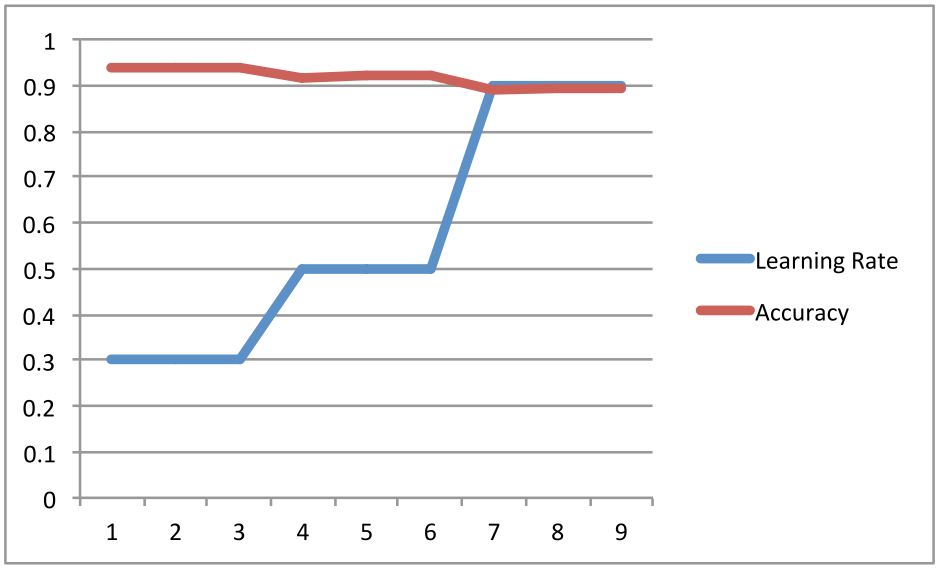
Graph illustrating the decline in accuracy with increasing learning rate.

# Epochs	Learning Rate	Momentum	Accuracy	Time (s)
500	0.3	0.2	93.9759%	746.28
1000	0.3	0.2	93.8832%	1443.22
2000	0.3	0.2	93.8832%	2848.16
500	0.5	0.4	91.7641%	780.7
1000	0.5	0.4	91.885%	1575.32
2000	0.5	0.4	91.885%	3014.21
500	0.9	0.7	88.9713%	752.2
1000	0.9	0.7	89.2532%	1755.55
2000	0.9	0.7	89.2172%	2683.12

As in the previous dataset, we observe here a decline in accuracy regardless of the number of epochs, with an increase in learning rate and momentum. Also, same as the previous case, the time increase by magnitudes with increasing number of epochs, but does not follow a linear pattern with increasing epochs as the previous dataset did. Interestingly, as compared to all other algorithms that showed opposing patterns for both datasets, which I attributed to the difference in the types of the two problems we are dealing with, neural networks showed almost a similar output and variance based on varying parameters. For both datasets, boosting with decision trees, with appropriate confidence (0.25) gave the best performance (most optimal accuracy/ least error). In terms of time, the SVM with a rbf kernel performed exponentially faster than any other algorithm, with a minimal error rate for the DNA splicing data set.



Declining accuracy regardless of epochs



Declining accuracy with increasing learning rate