**Introduction**

For the complexity models cross validation is used to reduce variance due to noise.

**Chess Set**

***Why is it interesting?***

In this data set the attributes are made up of discrete values. It is a binary classification set.

* + For ANN
  + You can see it has a steeper learning curve compared to some of the others. It needs more data to make an accurate model
  + If you have time generate graph on pg 110. Can do so by increasing number of training epochs for x axis.

***Artificial Neural Network***

For the Aritficial Neural Network I chose the MultiLayer Perceptron. The learning curve for the mutilayer perceptron is depicted in Figure 1. This curve was generated using 100 epochs and 2 hidden layers. Looking at the false positive rate and false negative rate curves we can see at around 90% the model starts to overfit. This is indicated by the the trajectory of the training and testing curves. The training error continues to fall while the testing error begins to rise around 90%. We can see the algorithm needs a significant chuck of the data before it starts performing well approaching the training set error around 70%. The MultiLayer Perceptron performs well on this type of data set since it can adjust the weights and filter out the less important attributes.

* Fix figure 2 test error shouldn’t start at zero

***Decision Tree***

For the decision tree I chose the J48 algorithm. It is a modified version of the ID3 algorithm discussed in the lectures. The learning curve is shown in Figure 2. This graph was produced using subTree Raising and Collapse tree pruning. The gap between the error rates remains steady no matter the amount of data used. J48 performs well on this type of data set since it is discrete. The discrete nature allows for easy node splitting.

For IBK

* + 1 overfits training data. Actually I think its because the model it creates is perfect. It’s looking for an instance that is as close as possible to instance you are testing. But if you have 1 neighbor and are retesting on the training set that distance will be zero.
  + 2 doesn’t overfit but doesn’t do any better on test set
  + 3 has about 20% less false positives
  + 4 and above does worse
  + Manhattan and Minkowski distance has a small effect

***Nearest Neighbors***

For the nearest neighbors algorithm the IBk algorithm was used. The learning curve is shown in Figure 3. For the curve shown k was set to 3. Euclidean distance was used with no distance weighting. Here the training error remains low and the test error starts out extremely high. The test error never drops below 10%. The IBk algorithm does not perform as well as the J48 or Multilayer Perceptron. This has to do with the discrete nature of the data set. A nearest neighbor will tend to perform better on a data set with continuous numeric features which lends itself well to a nearest distance calculation and distance weighting. For example in the case of attribute 1 bkblk there are only two values that this attribute can take on. This means IBk either gets a perfect match or nothing. The neighbor with the closest distance is the one with the highest number of matching attributes. On the other hand if this attribute were a continuous value the concept of distance would be more applicaple as we are not left with all or nothing. In this data set each feature has a finite set of values. The nearest neighbor algorithm has a bias which treats all attributes equally. Even if a certain set of attributes is not important for classification the IBK may match an instance from the test set to an instance in the training set based on these attributes. Distance weighting does not work well since the features are discrete. Figure 15 shows the effect of changing k on the error rate. For the false positive rate the variance starts off high and the bias low. This is shown by the large gap between the test set error and the training set error. Even though the training error is low it is not a very good predictor for the test set thus it has high variance. As k increases the variance decreases and the bias increases. On the right side of the graph the training error has gone up while the test error has stayed relatively the same. This indicates the model is generating incorrect target values i.e. high bias. Variance is higher for a smaller number of neighbors. This is because you only get one data point to compare your test point to. Nearest neighbors algorithms are considered more complex with smaller k.

* For polykernel
  + For SMO changing the exponent to 5 in polykernel options gives better results. Moving it to 15 gives 50% FPR on test set. 10 does better than 15 but worse than 5. 5 seems to be ideal.
  + w/ exponent of 5 it performs perfectly on training data. I think this means it has a wide margin that is also a perfect separator of the data.
  + Maybe use FNR for this because w/ change in exponent there is a 80% reduction in FNR compared to 50% reduction in FPR
* For RBFKernel
  + Changing gamma to 0.1 cuts FPR in half on test set.
  + Chaning gamma to 0.3 cuts FPR by factor of 10 on test set
  + Gamme to 0.5 does even better
  + With gamme of 1 it doesn’t perform as well as 0.5

***SVM***

For this algorithm SMO was used with a PolyKernel and RBFKernel. Figure 4 shows the learning curve when a PolyKernel is used. Figure 5 shows the learning curve when RBFKernel is used. Both of these figures show an extremely low error for the training set. For the PolyKernel even though the training error remains at zero the test error continues to decrease as we add more data. For each iteration of the data size the SMO builds a perfect model for the training data but the “perfect” model varies with each size of the data. The model still tends to improve its fitting to the test with increasing data set size even though the training error always remains at zero. This same process is happening in Figure 6 with the RBFKernel.

* For Ada it doesn’t make sense that we are getting 0 erros because our test error is continually decreasing
  + Perhaps it is because ada builds a model and subsequently tests that model on test set. There are many different models that could perform perfectly on training set but when each model is applied to the test set it gives different results. This is probably high variance. More in stanford packet.

***Boosting***

For boosting the AdaBoostM1 algorithm was used with J48 as its weak classifier. Figure 6 shows the learning curve.

-------------------------------------------------------------------------------------------------------

**Tic Tac Toe Data Set**

***Artificial Neural Network***

The learning curve is shown in figure 7. The Multilayer Perceptron performs very well with this data set after about 60% of the data is used. This curve was generated using 100 epochs and 2 hidden layers. The MultiLayer Perceptron performs well on this type of data set since it can adjust the weights and filter out the less important attributes. In figure 16 we can see a graph of error vs. model complexity. As the number of hidden layers increases the variance increases. Initially at small numbers of hidden layers the training error is close to the test error. This indicates bias in the model. The model tends to misclassify by aiming for the wrong spot. As the model complexity grows, as the number of hidden layers increases, the training error drops significantly while the test eroor drops only slightly. This large gap between the error rates indicates variance. The model tends to misclassify based on a spread from the target value. The variance is caused by noise in the data. Most likely the test error rate will not decrease any more no matter how many hidden layers we add. This is standard. Typically as model complexity increase variance increases while bias decreases.

* For J48
  + Turning off pruning actually cuts FPR in half for test set.
  + Increasing minNumobj decreases size of the tree
    - This one decreases faster than chess set

***Decision Tree***

The learning curve is shown in in Figure 8. From the learning curve we can see the gap in error rates remains steady. Thuis indicates that more data may be needed before we start to see a closure in the gap. The complexity model in figure 17 shows the bias-variance relationship. The x axis indicates the number of nodes in the tree. As the size of the tree grows the training error increases and then decreases for the largest tree sizes. The test error, on the other hand, increases with increasing tree size. This seperation indicates variance. Even though the model can classify the training data fairly well it does not do well on the test set. The different realizations of complex models still misclassify a large amount of instances on the training set. The smaller trees have a large number of misclassifications on both the test and training set. The less complex models will tend to be weaker and misclassify a larger number of examples. This indicates bias. The model classifies data points, on average, the wrong way.

***Nearest Neighbors***

The learning curve is shown in figure 9. Similar to the IBk algorithm for the Chess data set the IBk also performs poorly on this data set. This is because the data set contains attributes with discrete values. Reviewing the error rate vs number of epochs the error bottoms out at 4.6% after about 50% of the data.

***SVM***

The learning curve for the SMO using Polykernel is show in figure 10. The learning curve for the SMO using RBFKernel is shown in figure 11. Comparing the two figures we can see the difference between the Polykernel and the RBFKernel. With the PolyKernel the error is caused by the bias since the training and testing error are close together. With the RBFKernel the error is caused more by variance. The training error is low but the test error is high.

***Boosting***

The learning curve is show in figure 12. The bossting algorithm does not perform as well on the Tic Tac Toe set. This is due to the fact that the J48 weak classifer used already had a high amount oferror. However, comparing the two error curves we see that the boosting model significantly performs better than the J48 alone.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Chess** | | **Tic Tac Toe** | |
|  | Time to build model | Time to apply | Time to build model | Time to apply |
| **J48** | 0.03 seconds | 0 seconds | 0 seconds | 0 seconds |
| **MultiLayer Perceptron** | 0.5 seconds (2 hidden) (100 epoch) | 0.01 seconds | 0.12 seconds | 0,01 seconds |
| **SMO PolyKernel** | 1.4 seconds (exponent = 5) | 0.18 seconds | 0.35 seconds | 0.1 seconds |
| **SMO RBFKernel** | 3.44 seconds (gamma = 0.5) | 0.43 seconds | 0.24 seconds | 0.06 seconds |
| **ADABoostM1** | 0.25 seconds (10 iterations) | 0.01 seconds | 0.03 seconds | 0.01 seconds |
| **IBk** | 0 seconds | 1.1 seconds (3NN) | 0 seconds | 0.04 seconds |

Figure : Time to build and apply model

|  |  |  |
| --- | --- | --- |
|  | **Chess** | **Tic Tac Toe** |
| **J48** | 21 kB | 21 kB |
| **MultiLayer Perceptron** | 35 kB | 20 kB |
| **SMO PolyKernel** | 776 kB | 169 kB |
| **SMO RBFKernel** | 792 kB | 173 kB |
| **ADABoostM1** | 129 kB | 173 kB |
| **IBk** | 162 kB | 67 kB |

Figure : Space required for model

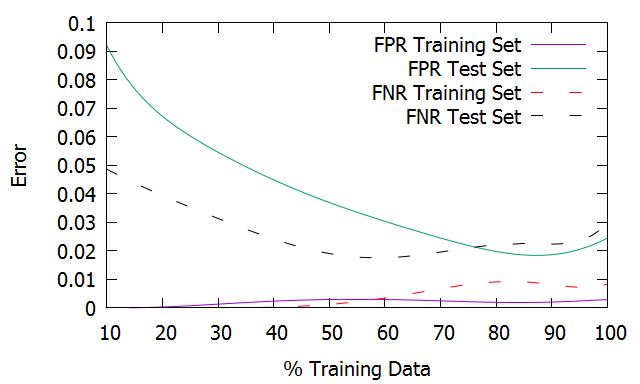


Figure 1: Multilayer Perceptron learning curve for Chess Set

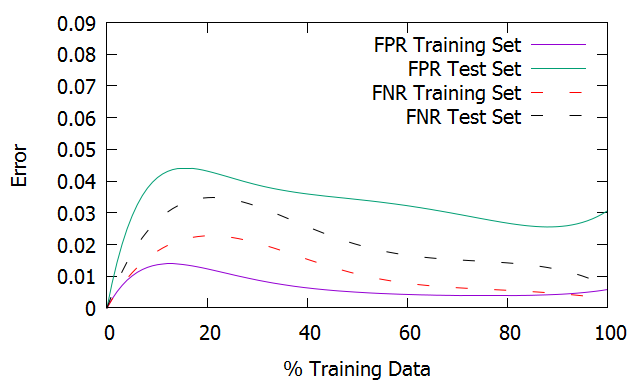


Figure 2: Decision Tree learning curve for Chess Set

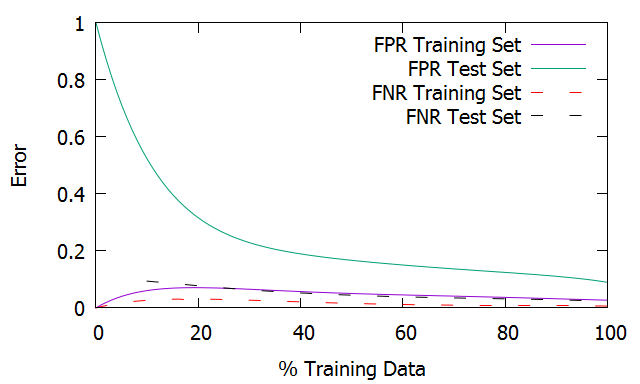


Figure 3: Nearest Neighbors learning curve for Chess Set

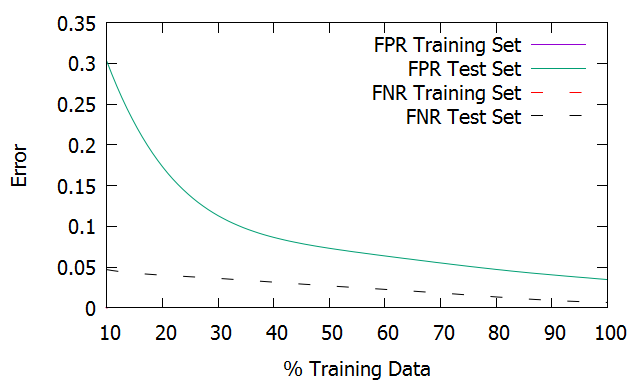


Figure 4: SVM with PolyKernel learning curve for Chess Set

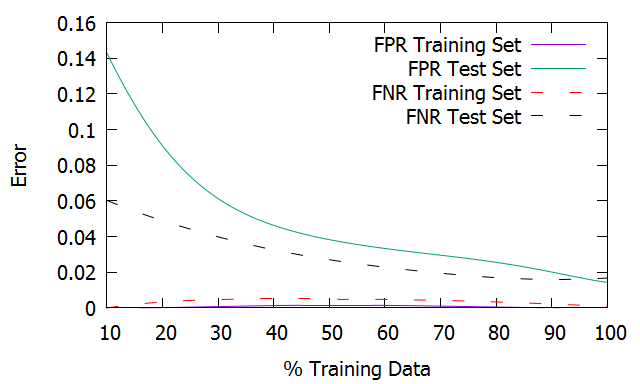


Figure 5: SVM with RBFKernel learning curve for Chess Set

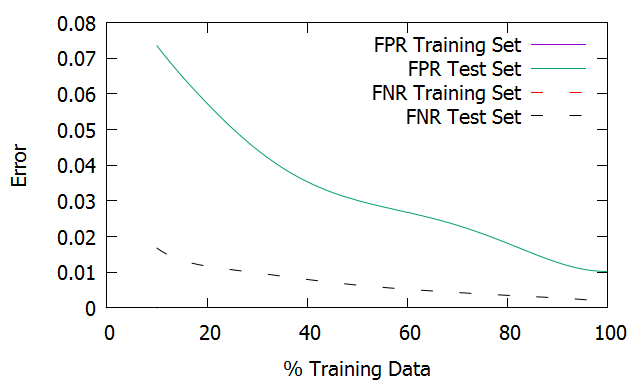


Figure : Nearest Neighbors learning curve for Chess Set

Figure 6: Boosting learning curve for Chess Set

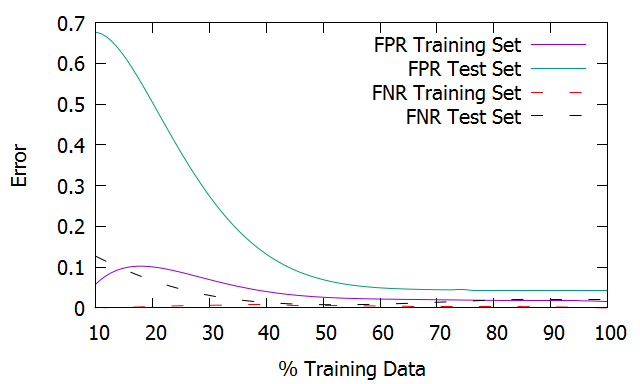


Figure 7: Multilayer Perceptron learning curve for Tic Tac Toe Set

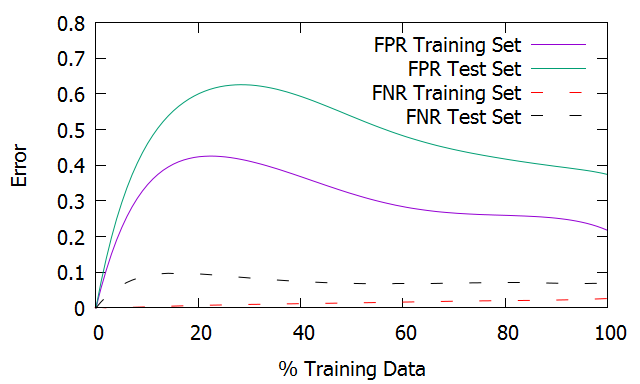


Figure 8: Decision Tree learning curve for Tic Tac Toe Set

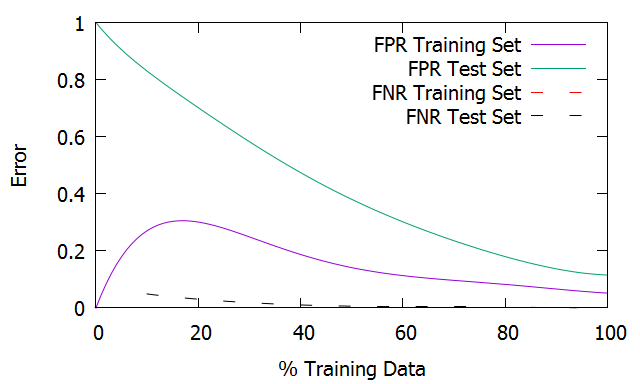


Figure : Nearest Neighbors learning curve for Chess Set

Figure 9: Nearest Neighbors learning curve for Tic Tac Toe Set

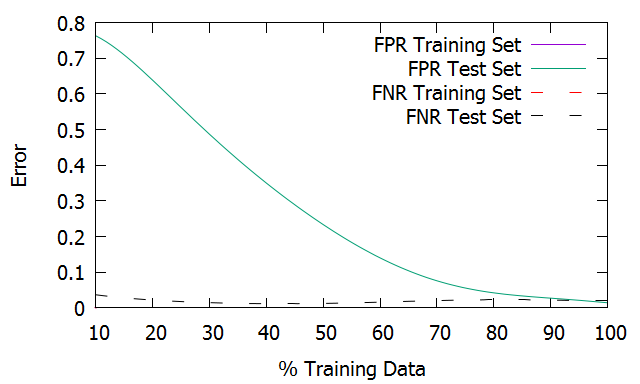


Figure 10: SVM with PolyKernel learning curve for Tic Tac Toe Set

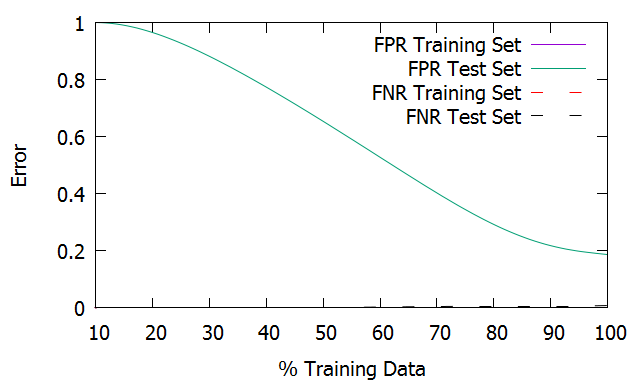


Figure 11: SVM with RBFKernel learning curve for Tic Tac Toe Set

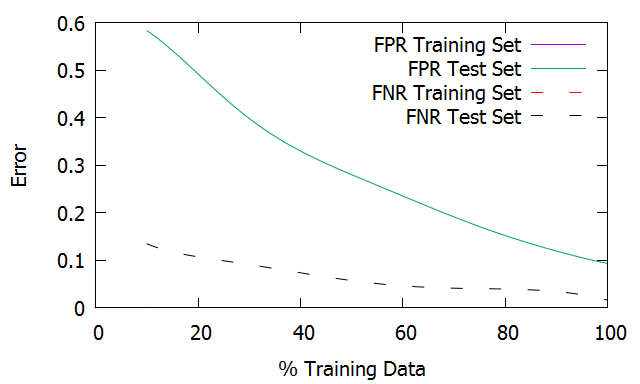


Figure : Nearest Neighbors learning curve for Chess Set

Figure 12: Boosting learning curve for Tic Tac Toe Set

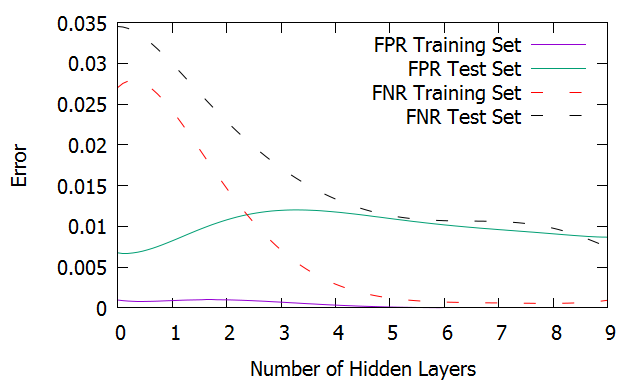


Figure 13: ANN, Error vs number of hidden layers for Chess Set

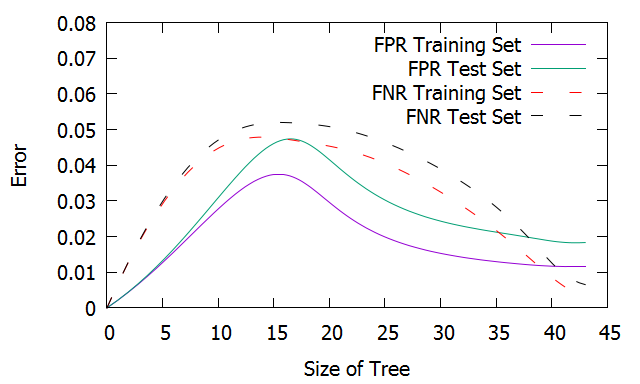


Figure 14: J48, Error vs. Size of Tree for Chess Set

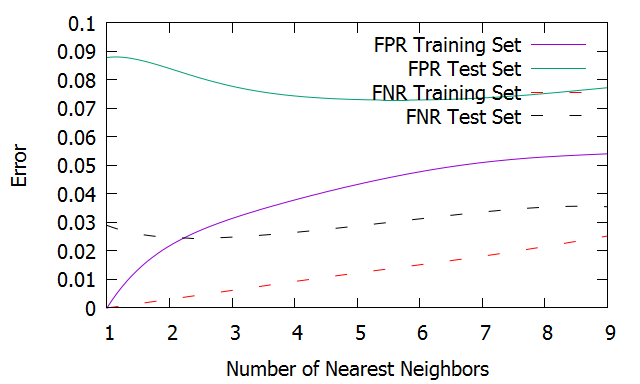


Figure : Nearest Neighbors learning curve for Chess Set

Figure 15: IBk, error vs. number of nearest neighbors for Chess Set

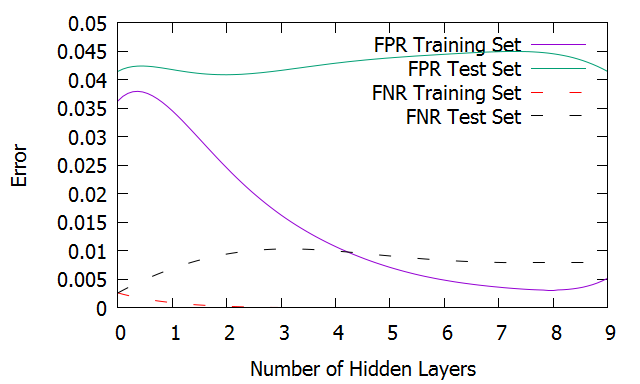


Figure 16: ANN, Error vs number of hidden layers for Tic Tac Toe Set

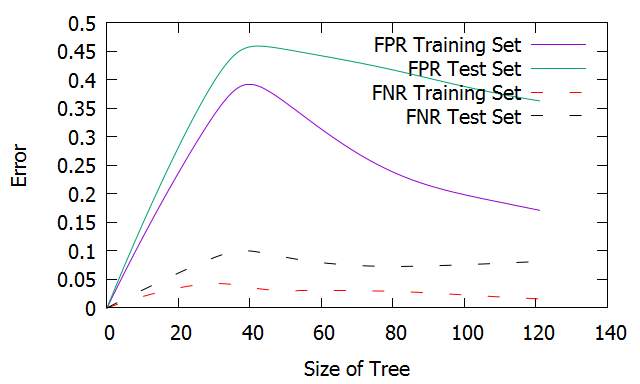


Figure 17: J48, Error vs. Size of Tree for Tic Tac Toe Set

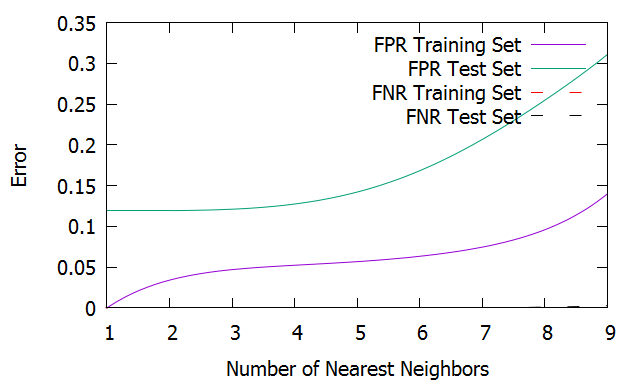


Figure : Nearest Neighbors learning curve for Chess Set

Figure 18: IBk, error vs. number of nearest neighbors for Tic Tac Toe Set

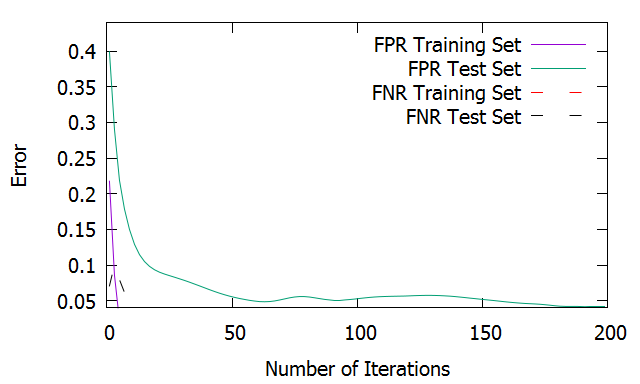


Figure 19: Error vs Number of Iterations for AdaBoost for Tic Tac toe set

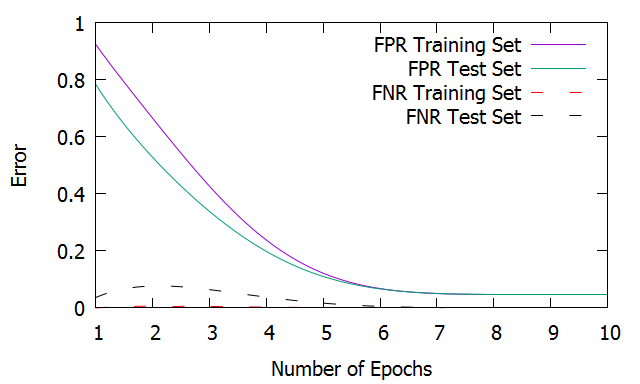


Figure 20: Error vs Number of Iterations for MultiLayer Perceptron for Tic Tac toe set

Figure : Nearest Neighbors learning curve for Chess Set

Figure 21