Clustering and Dimensionality Transformation

Gary Saavedra

gsaavedra3@gatech.edu

Introduction

The difficulty in clustering comes from measuring what a good cluster is. A cluster can perform well according to one measure and poorly according to another measure. This paper will attempt to make sense of the clusters produced from two data sets: a Tic-Tac-Toe data set and a Chess data set. Both of these data sets are from Assignment 1. For both data sets we do not need to normalize the features before running the clustering algorithms. Normalization is only needed when using features that have different units of measurement. The features of each data set are all measured in the same units with respect to that data set. This paper is divided into 5 parts for each data set. Each part answers the directive associated in the assignment prompt.

All the algorithms were implemented in Weka. SimpleKMeans was used as the k-Means algorithm and the EM algorithm was used for Expectation Maximization. Principal Components was used for PCA. Independent Components was used for ICA. Random Projection was used for RPA. The fourth dimensionality reduction algorithm chosen was Targeted Projection Pursuit. The parameters and results of each algorithm have been organized into tables.

**Tic-Tac-Toe Data Set**

Why is it Interesting?

The first data set used is a complete compilation of all possibilities of a Tic-Tac-Toe end game board. It is a binary classifier with “positive” or “negative” values. Positive indicates a win for the player, and negative indicates a loss. This data set contains 958 instances and 9 attributes. Each attribute represents a position on the board and has three values: ‘x’, ‘o’, or ‘b’ where b indicates a blank. Each instance is an board configuration at the end of a played game where the ‘x’ player has gone first. This data set differs from the Chess data set in a few different ways. First, it is much smaller at about one third the size of the chess data set. Second, it is unbalanced with about 65% “positive” classifications and 35% “negative” classifications. Analysis of this data can be used for improving performance at tic tac toe. Analysis of the data also shows the positive effect going first has on winning a match, as well as the most important locations to occupy on the board.

**Part 1**

K-Means Clustering

Figure 1 shows that for the k-Means algorithm k was set to 2. This gave the best performance in matching the class labels resulting in a 34.3% misclassification error. Increasing k beyond 2 only increased misclassification error as the extra clusters do not end up with a class label.

Euclidean distance was used, however no difference was noticed when using Manhattan distance. The initialization method used was Canopy. With random start initialization the misclassification error was approximately 45%. Using Canopy brought this down to 34.3%. Only 3 iterations were needed for convergence to 34.3% error. One of the properties of k-Means is that error is monotonically non-increasing with respect to iterations. Therefore, after 3 iterations the error stops decreasing and the algorithm terminates shortly thereafter. Depending on the random starting point the error varied. Increasing k did decrease the within cluster sum of squared errors.

The algorithm also performed well with k=1 however the result is misleading. With only 1 cluster every instance is classified as the majority class resulting in a misclassification error of 35%, the number of negative instances.

Figure 2 shows the mode matrix using 2 clusters. Each row represents the mode value for the attribute in that row. The column labeled Full Data shows the mode for each attribute for the full data set. The column labeled 0 shows the mode for each attribute for cluster 0. The column labeled 1 shows the mode for each attribute for cluster 1. The Full Data column shows that the mode for all the attributes is x for the whole training set. The mode value of ‘x’ for each attribute is caused by the ‘x’ player going first, thus they are able to occupy more positions on the board. With an odd number of board position the ‘x’ player will always have 5 position filled compared to the ‘o’ player’s 4 positions when the game ends in a draw. Cluster 0 centers on a majority of x’s while cluster 1 centers on a majority of o’s. The starting point of cluster 0 is x, x, x, x, x, x, x, x, x. The starting point of cluster 1 is x, x, o, x, b, o, x, o, b.

The way the algorithm works for a nominal data set is by matching attributes values. A value of 8 x’s and 1 o will be close to a value of all x’s. On the other hand, a value of all o’s would be far away from a value of all x’s. Thus cluster 0 starts with all x’s as its center and gravitates towards a center that is mostly x’s with a few o’s. This version of the algorithm is very limited and is responsible for the poor performance. The algorithm has an all or nothing approach. An attribute value with a match has a small distance and a mismatch has a large distance. There is also no variation in the distance for different types of mismatch. For example, both an ‘o’ and a ‘b’ will receive the same distance for mismatching without distinguishing between the two. The inability to have an actual distance metric limits the algorithm on this data set similar to k-Nearest Neighbors. In assignment 1 k-Nearest Neighbors also performed poorly on this data set. The poor performance is caused by the lack of distance weighting. Also instances on the edge of the cluster are still considered part of the class even though they may be far away.

Being limited to a mode of a data set severely limits the k-Means performance. With a numeric data set the mean can be chosen to be any point in the plane. It is not required that the point be one of the instance values. However, with the discrete version of k-Means we are forced to choose an instance value as our mode.

It is possible that the algorithm could be improved by stating that certain mismatches have greater distances than others. For instance a b could be closer to an x than an o is. This is because a blank can still be filled by an x.

Final cluster centroids:

Cluster#

Attribute Full Data 0 1

(574.0) (351.0) (223.0)

===================================

pos1 x o x

pos2 x x x

pos3 x x o

pos4 x o x

pos5 x x o

pos6 x x o

pos7 x o x

pos8 x x o

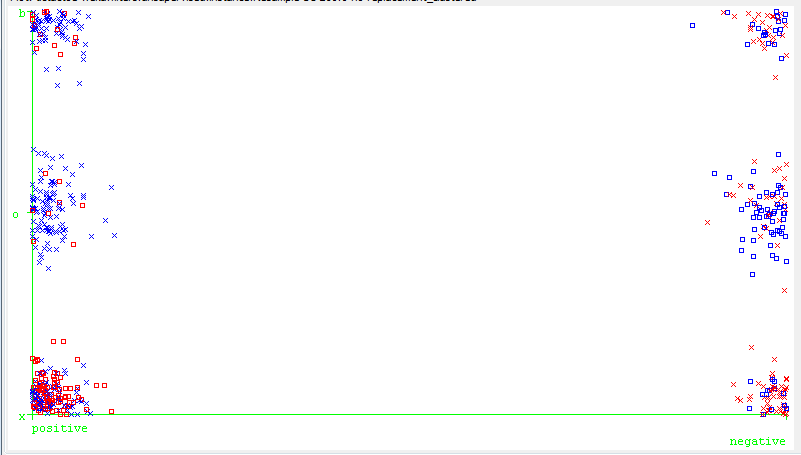
pos9 x x o

class positive positive negative

*Figure 2: Matrix of mode values for k-Means*

Figure 3 shows a cluster plot for the pos1 attribute value vs. the class label. The blue shapes correspond to cluster 0 and the red shapes correspond to cluster 1. Figure 3 provides a visual for the pos1 row in Figure 1. The majority value for cluster 0 for pos1 is ‘o’. In Figure 4 the ‘o’ position is occupied by mostly cluster 0, therefore the pos1 attribute will have a mode of ‘o’. Similarly the ‘x’ position is occupied mostly by cluster 1. Figure 3 also shows that the majority of cluster points are classified as positive.

Figure 3 indicates that the clustering algorithm is not creating clusters that align with the class labels. Instead it is creating clusters based on similarities between the starting instance and the neighboring instances. The fact that nearby instances do not necessarily share the same classification means the clusters will not align with the class labels. For example, an instance can have matching attribute values for all but 1 attribute but if that attribute was one of the 3 in a row creating a player win changing it will change the class label. Even though 8 of the 9 attributes match the class label is different.



*Figure 3: Cluster plot, k-Means, pos1 vs. class*

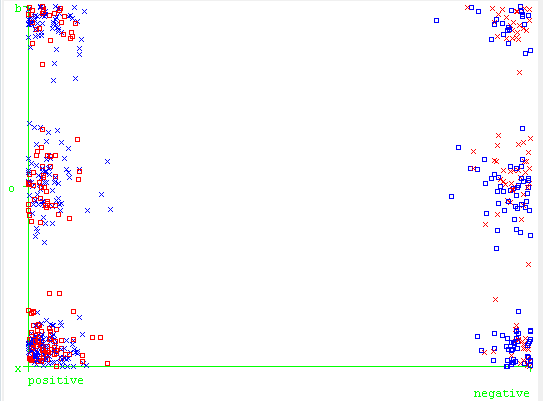
**Expectation Maximization**

The parameters and output for the EM are shown in Figure 1. For EM the k was set to 2. This gave the best output. If a higher value of k was used the extra clusters created were not used to classify which resulted in a higher error. The number of folds used was 10 and 10 K-Means runs were used. Overall the EM did not perform as well as k-Means when classifying the data. In comparison to the 3 iterations needed by k-Means, EM needed 86 iterations to converge to a local maximum. The time needed was 0.12 seconds in comparison to 0.001 seconds needed by k-Means, an increase by a factor of 100. The lowest misclassification error achieved was 43%.

How does the soft clustering affect performance in this case?

EM automatically turns data into Gaussian distribution. How does this affect results?

Figure 4 shows a cluster plot for the class value vs instance. Blue represents cluster 0 and red represents cluster 1. Figure 4 shows that the majority of clusters belong to the positive class. However, the positive and negative class contains instances from both clusters. This is an indication that EM is not clustering the classes well.



*Figure 4: Cluster plot, EM, pos1 vs class*

**Part 2 and 3**

**PCA**

After running PCA the 9 attributes are transformed into 16 attributes. The new attributes are linear combinations of the old attributes. The correlation matrix is 16 x 16 and too large to fit. The correlation matrix shows that all the attributes have a very low correlation with each other. This means that each of the newly transformed attributes are not repeating information from the other attributes.

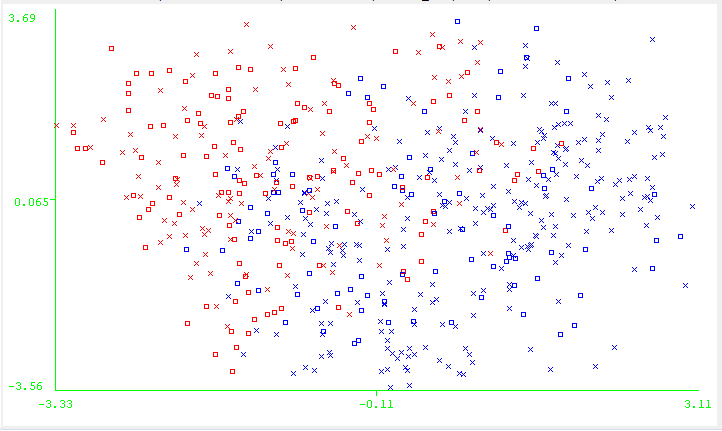
The scree plot is shown in Figure 5. The elbow in the plot is not clear but it appears to be at 6. In addition, the eigenvalues for the top 5 attributes are 2.3306, 2.295, 2.147, 2.074, and 2.013. After the 5th attribute the eigenvalues drop below 2. The larger eigenvalues indicate that the associated principal component lies in the direction of a large variance. Choosing attributes with a large variance guarantees we will see how that attribute value affects classification. Conversely attributes with low variance offer very little information about the class label.

*Figure 5: Scree plot, PCA, tic-tac-toe set*

***K-Means with PCA***

Whether using the top 5 or top 6 attributes the misclassification error is about 36% with simple k-Means. The within cluster sum of squared errors is 113.226 for 5 attributes. The within cluster error for 6 attributes is higher at 136.79. Only 11 iterations are need for convergence. T1 and T2 need to be bigger than previous part.

Figure 6 shows a cluster plot of attribute 1 vs attribute 4. There are a few key differences in this plot compared to the plot from the k-Means before PCA was applied. Before the plots shows the cluster with respect to a nominal attribute pos1. Now the attributes are linear combinations of the original attributes. This creates a difference in the scatter of the points. They now have a continuous spectrum they can occupy instead of the data points occupying one of three discrete values. This plot shows a clear division into 2 clusters with some overlap. The cluster plots for attribute values vs each other are similar and show a clear division. For this particular cluster plot there appears to be a diagonal plane that splits the two clusters. Once again the clusters are not being created based on class label. Even after the principal components transformation neighbors do not necessarily correspond to a particular class label.

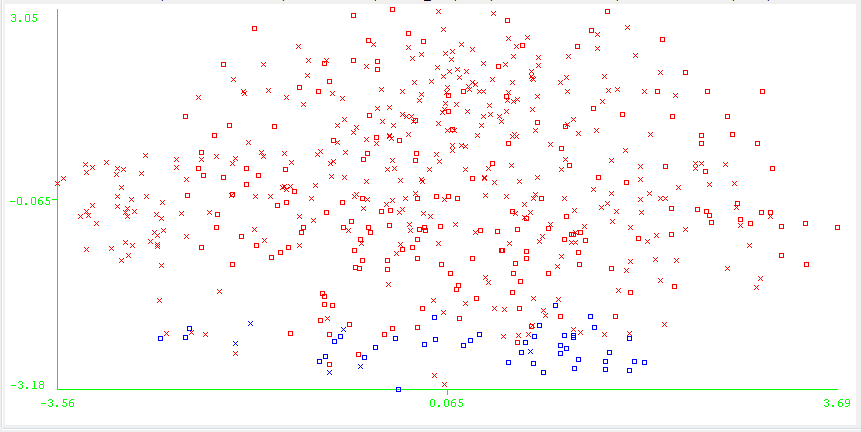


*Figure 6: k-Means, attribute 1 vs attribute 4*

***EM with PCA***

The EM converges to a minimum error of 39.7% after 156 iterations. The clusters here are much more imbalanced with cluster 0 only containing 41 of the 574 instances. Overall the EM does not perform as well as the k-Means. The time taken is 0.28 seconds compared to 0.005 seconds. The number of iterations needed is much larger as well, 156 compared to 11.

Figure 7 shows the cluster plot of attribute 1 vs attribute 3. Here we can see a clear division in the Gaussian distributions that generated the clusters. Figure 7 shows that the two clusters are not as mixed for the EM compared to the k-means in Figure 6. There is much more clear division in the clusters. The clusters appear to be divided by attribute 1 being less than or greater than -2.



*Figure 7: EM, PCA, Attribute 1 vs Attribute 3*

**Random Projections**

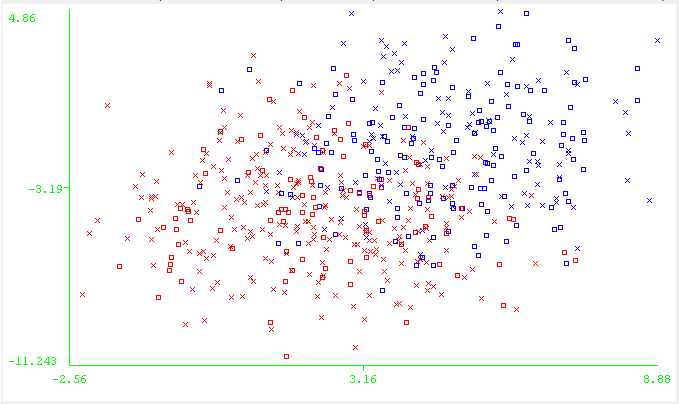
The Random Projection algorithm constructs a matrix with each row being an instance. The columns represent the attributes. To reduce the number of attributes we multiply this matrix by a scalar and another random matrix of the size we want to reduce to. In this case we have 574 instances and 9 attributes so we start with a 524 x 9 matrix. To reduce it down to 7 attributes we multiply it by a random number matrix of size 9 x 7.

The algorithm transforms the data set from a nominal to a numerical set. Each attribute now has a mean associated with it. This may improve performance because we are no longer using the mode.

***k-Means with RP***

Using a Gaussian distribution RP reduced the number of attributes down to 7. Different attribute reduction values were tested with differing random seeds. K was set to 2 These parameters resulted in the lowest misclassification error of 41.8%. Other parameters had error that varied between 41.8% and 82%. The within cluster sum of squared errors is 108.97. Only 25 iterations were needed to converge to this error. T1 and T2 were set to 10. Canopy was used again.

Figure 8 shows the cluster plot for attribute 5 vs attribute 2. The plot shows that the clusters are being formed around Gaussian distributions. For this particular plot a point is more likely to belong to cluster 0 when attribute 2 and 5 are high. Likewise, a point is more likely to belong to cluster 1 when both are attributes are small.

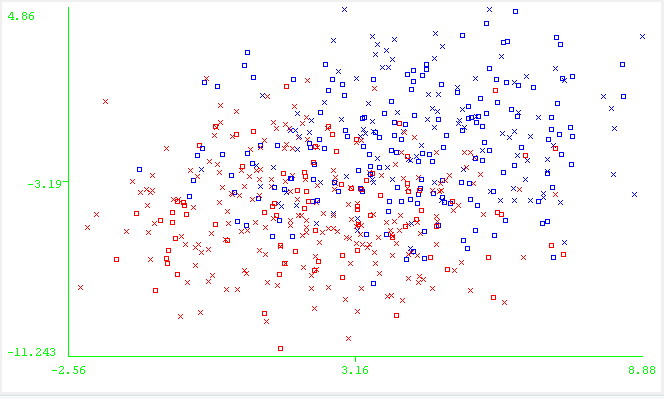


*Figure 8: k-Means, RP, K5 vs K2*

***EM with RP***

The distribution used was Gaussian. The number of dimensions was reduced to 5. K was set to 2. These parameters achieved the lowest misclassification error of 43.72%. Varying the parameters led to error as high as 72%.

Figure 9 shows the cluster plot for attribute 5 vs attribute 2. Here the clusters form two distinct Gaussian distributions similar to the k-Means. However, the EM has created distributions that are slightly more overlapping here. This is the result of soft clustering. With k-Means there is a sharp divide between the two clusters. With EM there is some probability that a point in the middle of cluster 1 could come from the distribution for cluster 0. In addition the EM treats the points as if there is an underlying Gaussian distribution. Those inferred distributions can be seen in Figure 9.



*Figure 9: EM, RP, K5 vs K2*

**ICA**

In order to run ICA on the discrete data set a preprocessing step is required. The data must be whitened. PCA accomplishes whitening while also adding a few other benefits. PCA de-correlates the data and creates partly independent variables which are required for ICA to operate. It can also discard small eigenvalues before whitening that will interfere with ICA. In this case, however, there are no trailing eigenvalues so all 16 attributes created by PCA are kept and fed into ICA.

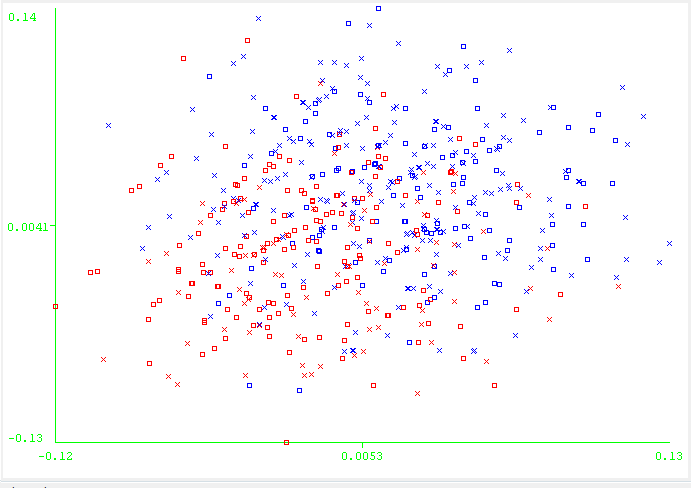
Figure 9 shows the kurtosis plotted against each of the attributes. ICA only operates on random variables that are non-Gaussian. Gaussian variables have a symmetric variance and give no information to ICA when trying to decompose the dependent variables. Since kurtosis is zero for Gaussian variables the absolute value can be used to determine non-gaussianity. The larger the absolute value of kurtosis the more non-gaussianity a random variable has. The Gaussian attributes that can be excluded from the clustering algorithms are 1, 5, 7, 9, 10, and 14.

*Figure 9: Kurtosis plot, Tic-Tac-Toe*

***k-Means with ICA***

For all 16 attributes random initialization was used. Canopy didn’t make a difference. 12 iterations and 0.01 seconds were needed to converge. The within cluster sum of squared error was 262.51. The misclassification error is 49.3%. The clusters are evenly balanced.

When excluding the 5 Gaussian attributes we get a similar classification error of 48.2%. The within cluster sum of squared errors is 179.81. It took 11 iterations and 0.01 seconds to converge. Clusters are less balanced at 336/238. Figure 10 shows the cluster arrangement for attribute 4 vs attribute 12. The clusters here appear to be more overlapping than the clusters created after applying PCA and RP. This is because ICA assumes the underlying distribution is not Gaussian. Therefore, the underlying properties of the cluster are not generated from a Gaussian distribution.

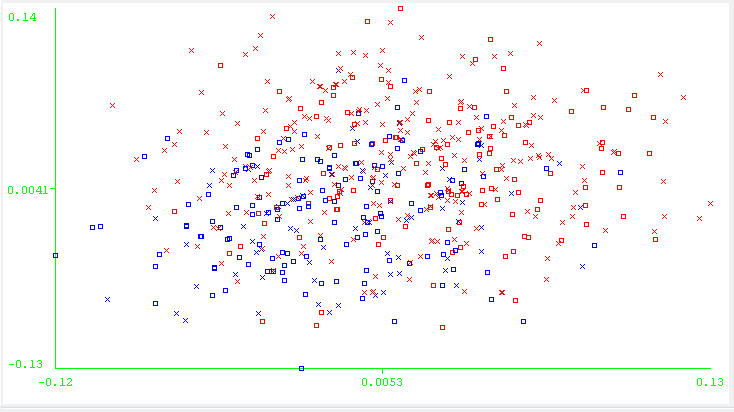


*Figure 10: Tic-Tac-Toe, ICA, k-Means, attribute 4 vs attribute 12*

***EM with ICA***

With all attributes being used the misclassification error is 49.6% the log likelihood is 27.09. It took 7 iterations and 0.07 seconds to converge. Clusters are about evenly balanced.

With the 5 Gaussian features removed the misclassification error is 44.5% and the log likelihood is 18.55. It took 35 iterations and 0.09 seconds to converge. There is less cluster balance at 197/377. Figure 11 shows the cluster plot for attribute 4 vs attribute 12. The clusters here appear to overlap more than in Figure 10. This I because EM uses soft clustering.



*Figure 11: Tic-Tac-Toe, ICA, EM, attribute 4 vs attribute 12*

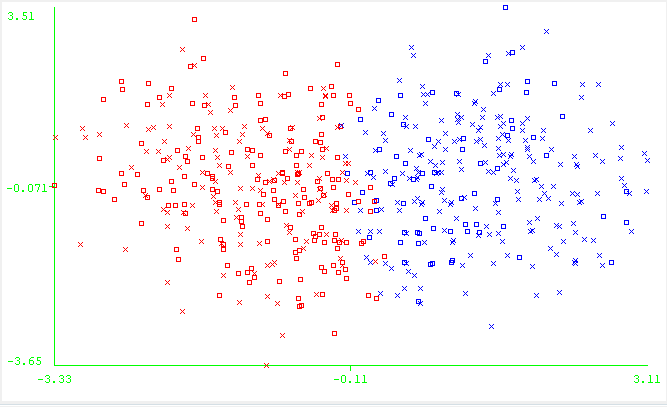
**Targeted Projection Pursuit**

TPP finds projections by reducing the number of projections from a high dimensional space into fewer projections in a lower dimensional space. TPP’s main asset is its ability to overcome the curse of dimensionality. TPP seeks to extract random variables that are non-Gaussian. PCA was run first on the data. This helps whiten the data as TPP cannot run on a discrete data set. The set was reduced from 16 PCA attributes down to 5 attributes. The PCA attributes chosen were 4, 7, 10, 13, 15.

***k-Means using TPP***

After using TPP, k was set to 2 and the canopy method was used with T1=10.25 and T2=10 for k-Means. The number of iterations was 9 and the time taken was 0.01 seconds. The misclassification was 42.8% and the within cluster sum of squared errors was 75.07. The clusters are almost evenly balanced at 273/301.

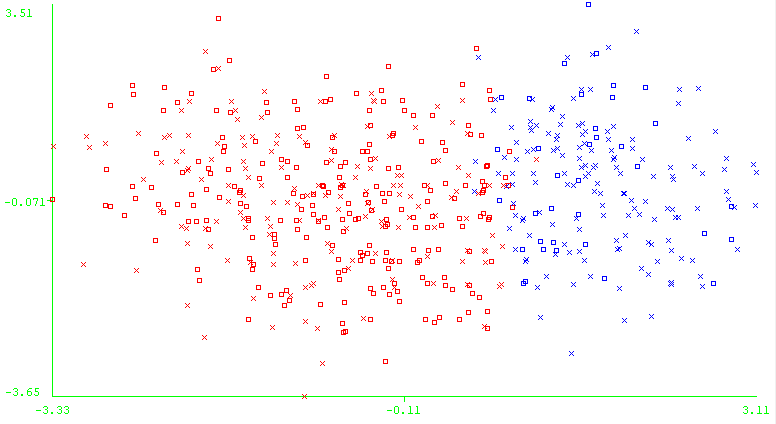
Figure 12 shows attribute 4 vs attribute 5 for TPP. The clusters are evenly splits down the middle at the mean of attribute 4. Any attribute vs attribute 4 gets perfectly split like this. This is not the case for any other attributes, the rest are overlapping. This shows that TPP is creating the clusters based solely on attribute 4. Redoing k-means when keeping only attribute 4 yields an error of 42.3%. The misclassification error remains relatively unchanged showing that the other attributes discovered by TPP were not as important. In addition the squared error within the cluster reduces down to 8.14.



*Figure 12: Tic-Tac-Toe, TPP, k-Means, Attribute 4 vs Attribute 10*

***EM using TPP***

Using all 5 attributes the misclassification error is 46.5% and the log likelihood it -7.96. It takes 122 iterations and 0.11 seconds to converge. The cluster balance is 186/388. Figure 13 is similar to Figure 12. Attribute 4 splits the clusters perfectly with all the other attributes. However, the split is no longer at the mean. Using just attribute 4 gives a misclassification rate of 43.9% and a log likelihood of -1.74. The EM’s performance in this case is similar to k-Means when comparing the misclassification. The time and iterations needed are still much greater though.



*Figure 13: Tic-Tac-Toe, TPP, EM, Attribute 4 vs Attribute 10*

Chess Data Set

Why is it Interesting?

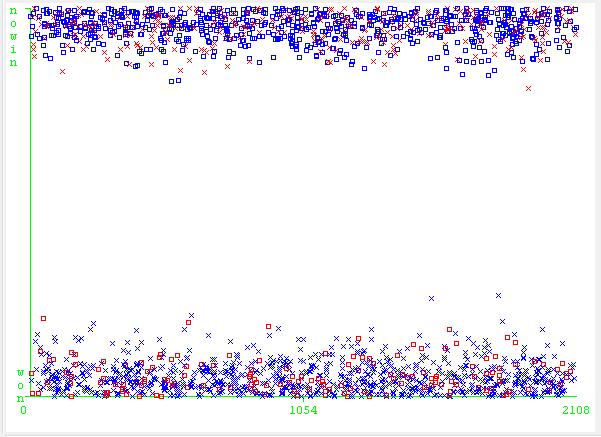
The second data set used here is a Chess data set with 3196 instances and 36 attributes. Each instance represents a board configuration in the end stage of a chess game. In this data set the attributes are made up of discrete values. It is a binary classification set with each instance being classified as “win” or nowin”. The data set is fairly balanced with 52% “win” classifications and 48% “nowin” classifications. The data represents one player with a king and a rook and an opponent with a king and a pawn. The opponent with the pawn is one step away from queening. If the opponent is able to queen it is considered a “nowin” for the player. This data set illuminates some differences in the operating principles of the different algorithms. Analysis of this data can be used to improve a player’s performance during the end stages of a chess match. It will allow a player to determine which moves truly matter when attempting to capture an opponent’s king.

**Part 1**

***k-Means***

For the k-Means before any dimensional reduction techniques were used, k was set to 2. Canopy mode was used with T1 = 5.25 and T2 = 5. Over many trials with different random number seeds these parameters were able to achieve a misclassification rate of 41.01%. Varying the parameters led to higher errors. Particularly varying k led to error rates as high as 85%. The within cluster sum of square errors was 13390.

Figure 14 shows a cluster plot for the class vs instance. Figure 14 shows the k-Means algorithm is not building clusters based on class labels. Similar to the Tic-Tac-Toe data set nearby instances do not necessarily share class labels. If you take an instance and change a single attribute value the new instance will be very near the original instance. However, this equates to changing the board layout which when trying to capture the pawn may completely change the result.

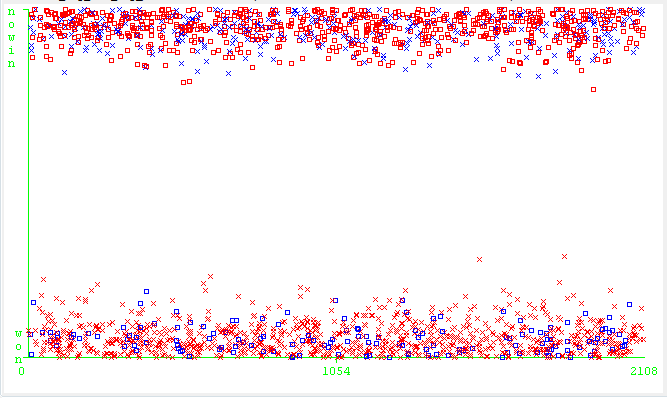


*Figure 14: Chess set, k-Means, class vs instance*

***EM***

EM was initialized using k=2. The number of folds used was 10. These parameters gave the best performance with a misclassification error of 40.77% and a log likelihood -14.17. The EM took 46 iterations to find the local minimum. Varying these parameters resulted in higher classification error with the highest being 65%. EM performs slightly better than k-Means when finding a misclassification error minimum, however the time and iterations needed are greater.

Figure 15 shows the cluster plot for class vs instance number. Once again EM is not building clusters based on the class label. It suffers from the same limitation that the k-Means suffers from. It builds clusters based only on physical distance when other distance measures may be more appropriate.



*Figure 15: Chess Set, EM, class vs variance*

**Part 2 and 3**

**PCA**

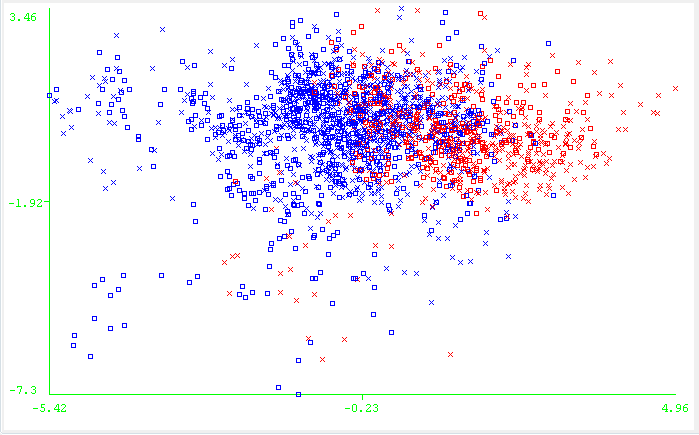
The 37 attributes are transformed into 31 attributes using a 0.95 variance coverage. The correlation matrix shows a low correlation between each of the new attributes. Figure 16 shows the scree plot. The elbow in the plot appears to be at about attribute 10. The sharp decrease in eigenvalues begins to flatten out at this point. Comparing the eigenvalues shows that the first 4 attributes are the most important. The first attribute has an eigenvalue of 4.02 while the next 3 eigenvalues are 2.89, 2.65, and 2.08. The large eigenvalue of the first attribute shows that the first attribute captures a large amount of variance.

*Figure 16: Scree plot of eigenvalues*

***k-Means using PCA***

Initially k-Means was run with all 31 attributes discovered by PCA and k was set to 2. T1 = 10.25 and T2 = 5. This led to the lowest error of 41.9%. The within cluster sum of squared errors was 891.37. 22 iterations were needed for convergence. Using less attributes resulted in a higher error. Using only the first 10 attributes the misclassification rate was 48.6%. There is a significant increase in misclassification error when using the elbow method to pick attributes. This indicates that another method, such as negentropy, may do a better job at picking the most important attributes when attempting to build clusters that match the class labels. The sum of squared errors within the cluster, however, does decrease with the elbow method with a value of 273.5.

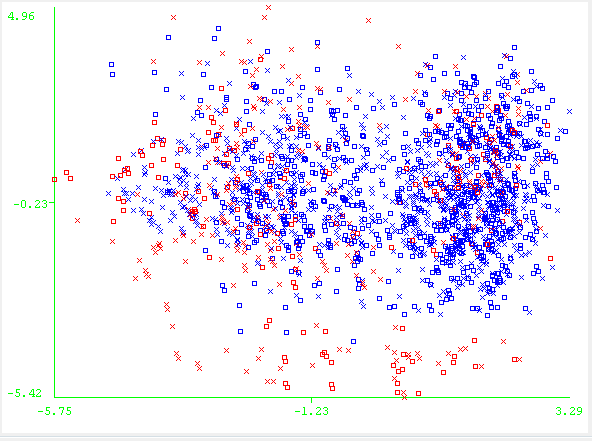
Figure 17 shows a cluster plot comparing attribute 3 and 5. The two clusters are somewhat overlapping thus the plane that cuts the two clusters is not entirely independent of attribute 3 and 5. All the cluster plots shows some overlap, thus the plane that cuts the two clusters is dependent on all the attributes.



*Figure 17: Chess set, k-Means, PCA, attribute 3 vs. attribute 5*

***EM with PCA***

After PCA was applied for the chess set k was set to 2. The number of folds used was 100 and 100 kMeansRuns were used. These setting gave the best misclassification rate. The misclassification rate was 44.3% and the log likelihood is -41.35. It took 28 iterations to converge. Using only the first 10 attributes according to the scree plot we get a 47.1% misclassification error and -15.13 log likelihood. Figure 17 shows a cluster plot for attribute 1 vs attribute 3. Cluster 0 contains about 3 times as many of the classification points as cluster 1. Compared to the k-Means plot there is not a sharp divide between the clusters.



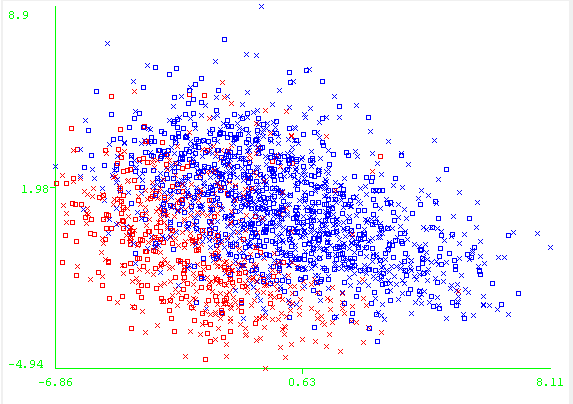
*Figure 17: EM, Chess Set, attribute 1 vs attribute 3*

**Random Projection**

Compared to the Tic-Tac-Toe set there are many more features to experiment with for RP. Several different projections were experimented with to find the best scheme for k-Means and EM. The distribution used for the randomizing matrix was Gaussian.

***k-Means with RP***

The best performance was obtained by reducing the dimensions from 36 down to 30 with k set to 2. Canopy mode was used with T1 = 10.25 and T2 = 10. Only 15 iterations were needed for convergence. The within cluster sum of squared errors is 1478.79. The misclassification error is 38.88%. Figure 18 shows the cluster plot attribute 3 and 7. The clusters are imbalanced with about 3 times as many points in cluster 0. Conversely k-Means created balanced clusters when using PCA.

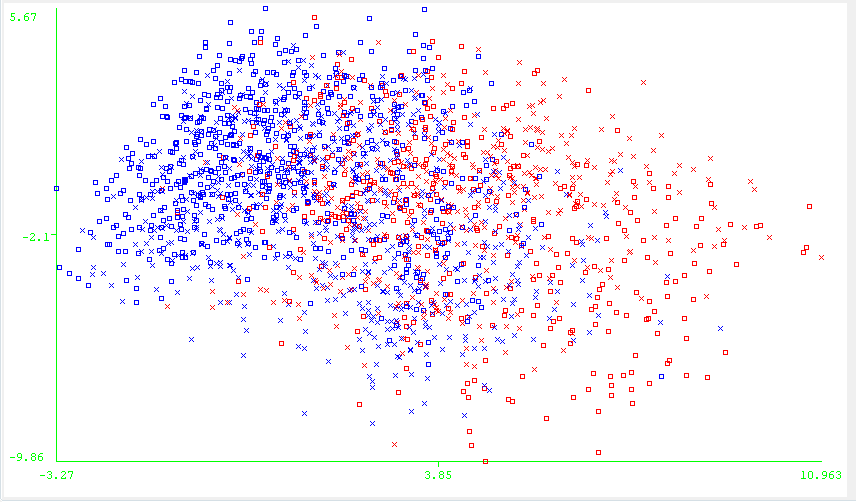


*Figure 18: k-Means, RP, Chess set, K3 vs K7*

***EM with RP***

The EM also performs best with 30 attributes and k=2. 100 kMeansruns were used with 10 folds. There was a log likelihood of -63.71 and misclassification rate of 49.17%. Only 21 iterations and 0.64 seconds were needed for convergence. With respect to misclassification the EM performs worse than the k-Means after using RP. However, with respect to number of iterations the EM performs comparably to k-Means.

Figure 19 shows the cluster plot for attribute 12 vs attribute 6. Compared to EM using PCA the clusters are much more balanced. The misclassification error is slightly worse than the results obtained from PCA which is not unusual. RP often does not outperform PCA with respect to error. Its usefulness lies in its speed. The slowness of PCA and ICA come from the many correlation and independence calculations they must make. For large data sets RP is useful but these data sets are small enough that RP’s advantage cannot be seen.



*Figure 19: Chess set, EM, RP, attribute 12 vs attribute 6*

**ICA**

ICA was run after PCA was run on the data. The data set was reduced down to 31 attributes. Running PCA whitens the data. The kurtosis plot is shown in Figure 20. The plot shows there are only a few attributes that are important: 4, 14, 21, 22, and 31. Together these attributes account for most of the original underlying independent variables. The attributes with low kurtosis are likely to be Gaussian noise. Attributes with high kurtosis will have low Gaussianity.

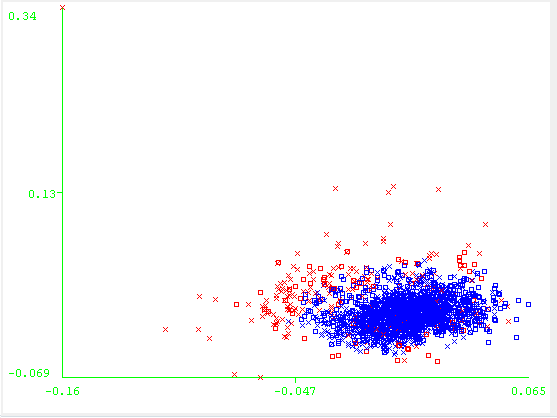
*Figure 20: Chess set, ICA, Kurtosis vs attribute*

***k-Means with ICA***

Using all 31 attributes generated by ICA the misclassification rate is 30.1% and the sum within clusters squared error is 590.1. The misclassification rate achieved by ICA is the lowest compared to all the other dimensional reduction techniques. ICA is able to extract some of the underlying hidden variables that the other dimensional reduction techniques miss. The clusters are nearly balanced with an equal number of points in each. It took 14 iterations to converge. Canopy method was used with T1=10 and T2=10. Using only the 5 attributes indicated by the kurtosis plot the misclassification error was 34%. Clusters are slightly more imbalanced at 60/40. The within cluster sum of squared errors is 35.17. Using only the 5 attributes makes little difference in the misclassification error and creates a huge drop in the within cluster sum of squared errors.

***EM with ICA***

Using all 31 attributes generated by ICA and k=2 there was a 38.1% misclassification rate and 74.52 log likelihood. It took 25 iterations to converge and 0.74 seconds. There was a 1400/700 split in clusters. Reducing to the 5 features indicated by the kurtosis plot the misclassification is 44.76%. The log likelihood is 12.21. The cluster is also more imbalanced 1800/200 which is most likely leading to the higher misclassification error. Overall, keeping only the 5 attributes has made a minor difference in classification error. Figure 21 shows the clusters are finding a similarity on points in the center and another similarity on points surrounding the outside. In contrast, the clusters created previously by k-Means were split more down the middle according to some plane. The figure also shows there is a large cluster imbalance.



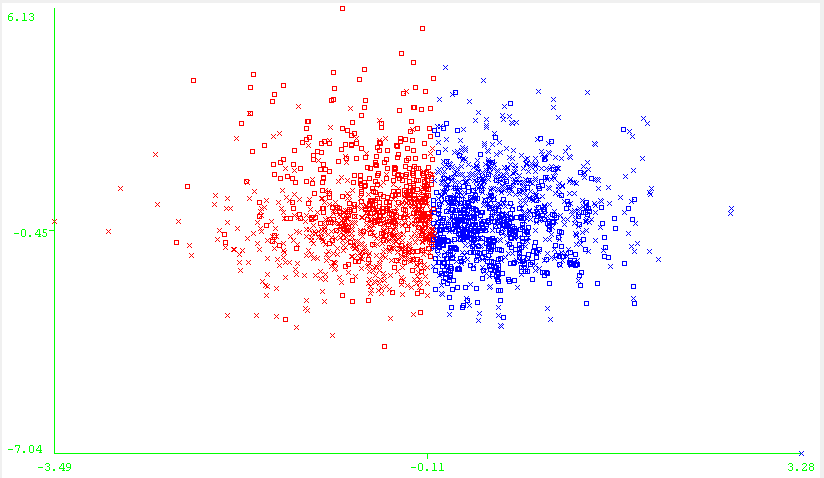
*Figure 21: Chess set, ICA, Attribute 14 vs attribute 21*

**Targeted Projection Pursuit**

PCA was run on the data set before TPP was run to whiten the data. TPP reduces the selection of 31 attributes obtained by PCA down to 5.

***k-Means with TPP***

With all 5 attributes k was set to 2 and canopy was used with T1=10.25 and T2 = 10. The within clusters sum of squared error was 67.36 and the misclassification error was 42.1%. The cluster balance was 1139/970. The number of iterations was 20 and the time taken was 0.02 seconds. Figure 22 shows, similar to Tic-Tac-Toe set, that the clusters get split on one attribute at the mean. Using only attribute 5 the misclassification error was 42.3% and the within cluster sum of squared errors was 10.47. The cluster balance was 1142/967. There is no change from using all 5 attributes to just using attribute 5. The projections created by TPP lead the k-Means algorithm to consider attribute 5 to be the deciding factor when forming the clusters.

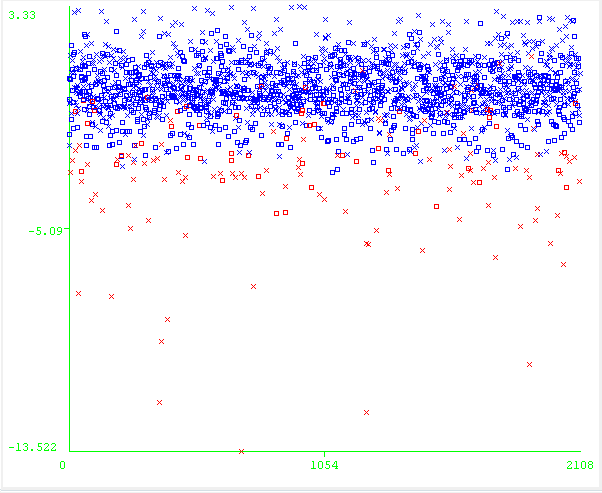


*Figure 22: Chess, k-Means, TPP, attribute 1 vs attribute 5*

***EM with TPP***

Using all 5 attributes the misclassification rate was 46.6% and the log likelihood was -7.1. The clusters are very imbalanced with cluster 0 containing 1950 points and cluster containing 159 points. Even though the cluster is very imbalanced its error rate isn’t far off from the k-Means which has more balanced clusters. The number of iterations needed was 34 and the time was 0.19 seconds. EM performs comparably to k-Means in both iterations and classification error.

Figure 23 shows that the clusters can be split perfectly across attribute 2. Using only attribute 2 the misclassification error is 46.51% and the log likelihood is -1.65. The clusters were still unbalanced with a ratio of 2001/108. The misclassification error remains relatively unchanged. One interesting difference between the EM and k-Means is that the perfect split happens on a different attribute for each.

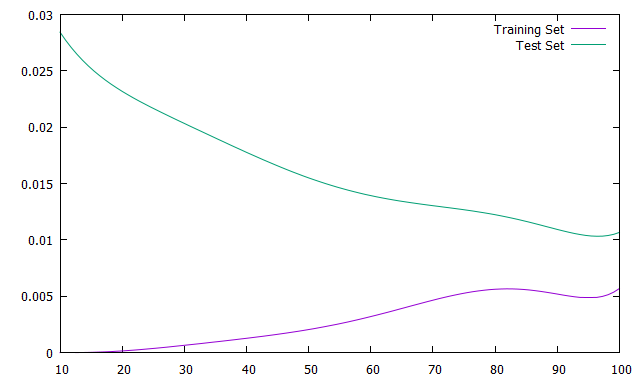


*Figure 23: Chess, TPP, EM, Instance vs Attribute 2*

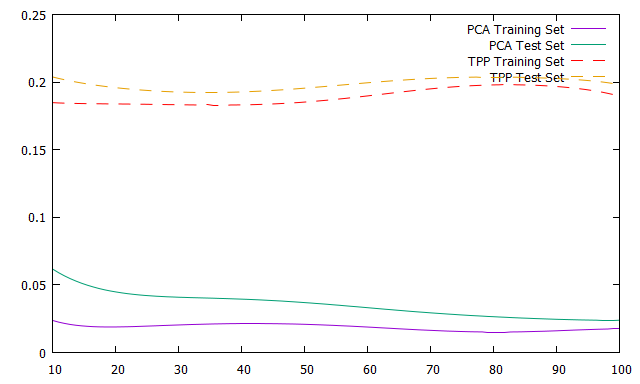
**Part 4**

Figure … shows the learning curve using the original attributes. The error rate is extremely low when using the original attributes. Figure … shows the learning curves for PCA and TPP. The ANN performs well with the attribute transformations from PCA. The error rate is only slightly higher. The ANN does not perform well with the TPP transformations with both the training and testing error around 20%. We can also see that the performance is about the same regardless of the data size.

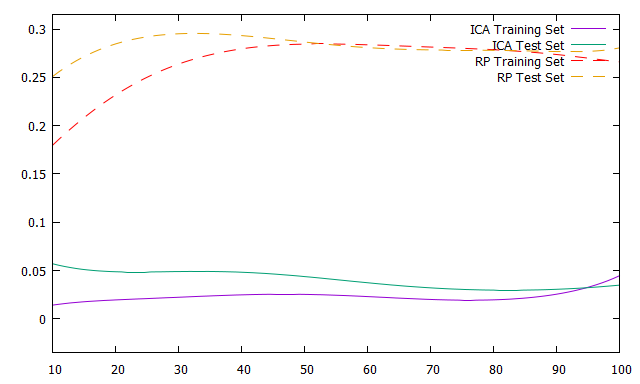
Figure … shows the learning curves for RP and ICA. The ANN performs well with the ICA transformations with an error of about 5%. The ANN does not perform well with the RP transformations. The error hovers above 25%. This makes the RP the worst transform for this data. The error also appears to initially increase as the data size grows and then levels out.



*Figure : Original Learning curve for Chess set*



*Figure : Learning curve for PCA and TPP*

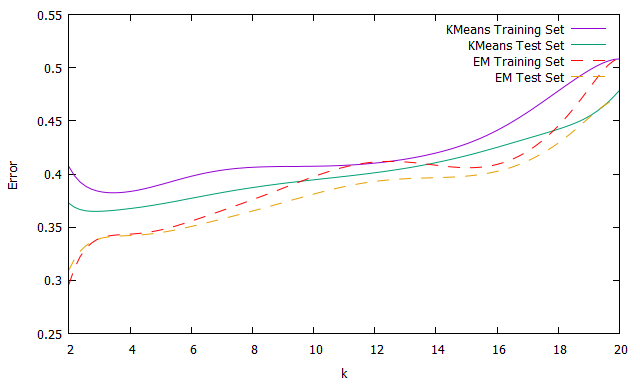


*Figure : Learning curve for ICA and RP*

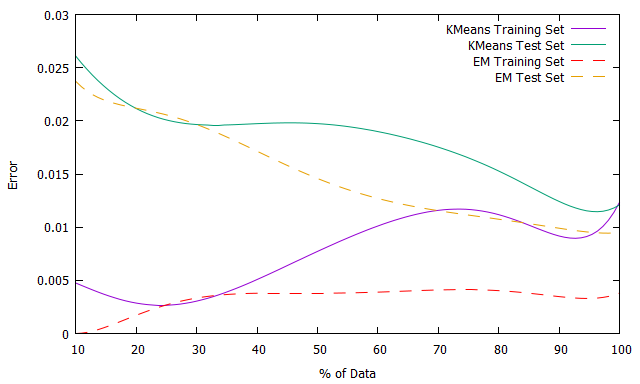
**Part 5**

Figure … shows the error rate of the ANN for different choices of k. In the plot only the clusters were kept as features. The original features were ignored. The graph shows that the best choices of k to reduce misclassification error using the neural net is k=2. The graph also shows something similar to the other parts of the assignment. Using only the clusters as classification our error rate for k-Means and EM are approximately 33% and 38% for k=2 respectively. This is similar to the results we achieved earlier. The ANN cannot perform any better because it is restricted to what the clusters think the class labels are.

Figure … shows the learning curve with the cluster being used as a feature in addition to the original features. The performance of the ANN is similar to Figure (from part 4). The error rate remains relatively unchanged. This means the patterns that the cluster algorithms are finding provide no additional information for classification.



*Figure : Error vs k for EM and k-Means*



*Figure : Learning curve for EM and k-Means*

**Conclusion**

These algorithms are not meant to classify data. Instead they find other patterns that would be missed by a supervised learning algorithm.

Overall these data sets do not lend themselves to these particular clustering algorithms. Physical distance metrics do not make sense with these data sets since neighbors can often have differing class labels. Similarly this is the same reason k-nearest neighbors performed poorly on the data sets in Assignment 1. One way to improve the clustering algorithms performance is to use a distance function that is not a physical distance. For example, the Tic-Tac-Toe set could use a distance function that classifies two or more symbols in a row as closer together than a randomly scattered board. Using this distance metric would give correct class labels to the clusters more often. For the Chess set neighbors that are physically close are not necessarily the same class. Often when a player is attempting to capture an opponent’s king the arrangement of pieces is important. Moving just one piece could alter the class label from a “win” to a “nowin.” However this would be an extremely near neighbor. An alternative distance function could be number of moves needed for the rook to capture the pawn. If the number of moves required to capture the pawn is greater than the pawn’s needed number of moves to queen then we know we will receive a “nowin” label. This distance function would eliminate all the board configurations that are neighbors that do not contribute to winning.