Assignment 3

Gary Saavedra

Gsaavedra3@gatech.edu

Introduction

The algorithms were implemented in Weka. For K-Means the SimpleKMeans algorithm was used. For Expectation Maximization the EM was used.

**Tic-Tac-Toe Data Set**

Why is it Interesting?

The second 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

In WEKA in the output there is a table which shows the majority value of each attribute.

Usually only needs a few iterations to get best error rate it can get. According to properties the error is monotonically non-increasing meaning with each iteration error should decrease or stay the same.

For the K-means algorithm k was set to 2. This gave the best performance. Increasing beyond 2 only increased classification error. Often the extra clusters do not end up with a class. 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. Euclidean distance was used, however no difference was noticed when using Manhattan distance. The initialization method used was canopy. With random start misclassification error was approximately 45%. Using canopy brought this down to 35%. Only 3 iterations were needed for convergence to 35% error but 100 were used to be safe. Depending on the random starting point the error varied.

Figure 1 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 majority of x’s is caused by the x player going first. They are more likely to win and also with an odd number of board position the x player with always have 5 position filled compared to the o players 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 mean and gravitates towards a mean 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 (zero?) 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. 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.

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.

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

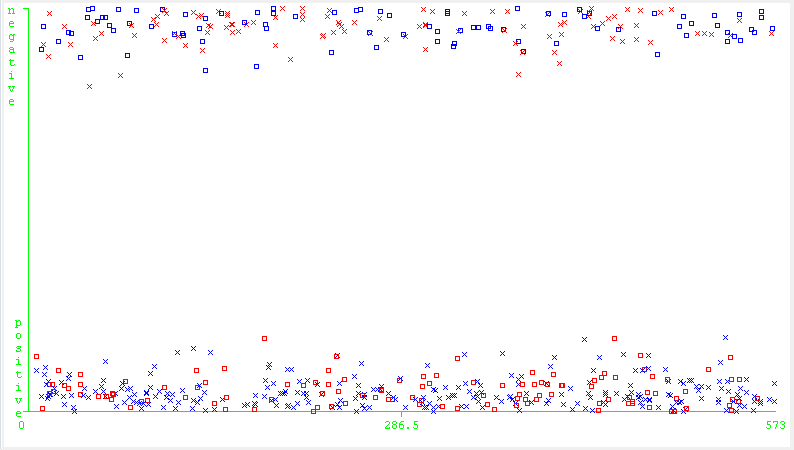
pos9 x x o

class positive positive negative

*Figure 1: Matrix of mode values for k-Means*

The Jitter function is used in Figure 2 and 3 to help visualize the number of points in a cluster. In reality all the points lie on top of each other at each tick mark of the axis. Possibly include attribute vs attribute graph.

Figure 2 shows a cluster plot for classes vs. instance. The blue shapes correspond to cluster 0 and the red shapes correspond to cluster 1. The bottom portion of the graph represents the positive class. It can be seen in the graph that the bottom portion contains a larger portion of the cluster points. This is an indication that a larger percentage of the instances are classified as positive.

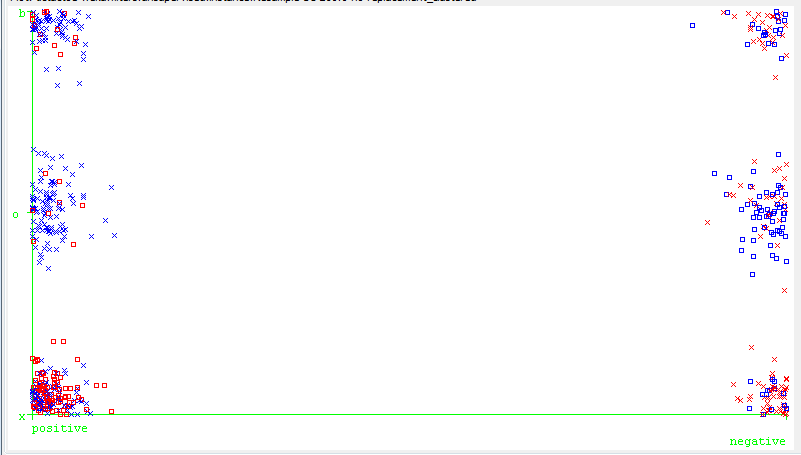


*Figure 2: Cluster plot, K-Means, class vs. instance*

Figure 3 shows a cluster plot for pos1 attribute values vs. the class value. 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 3 the o position is occupied by mostly cluster 0. The x position is occupied mostly by cluster 1. Figure 3 also shows that the majority of cluster points are classified as positive.

Figure 2 and 3 also indicate that the clustering algorithm is not creating clusters that align with the class values. 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 values. For this data set this makes sense. A board can have a matching attribute 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 value. Even though 8 of the 9 attributes match the class value is different. These instances may be responsible for most of the misclassification. Get a numeric value for this.

Similar to k nearest neighbors. That algorithm also did bad in Assignment 1. Worse performance because no distance weighting and instances on edge of cluster are still considered part of the class even though they may be far away.



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

**Expectation Maximization**

Does a little worse than k means.

There is some variation in error with respect to number of runs. Might make a good graph.

For EM 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. This resulted in a higher error. The number of folds used was 10 and 10 K-Means runs were used. Only 56 iterations were needed to converge to a local maximum.

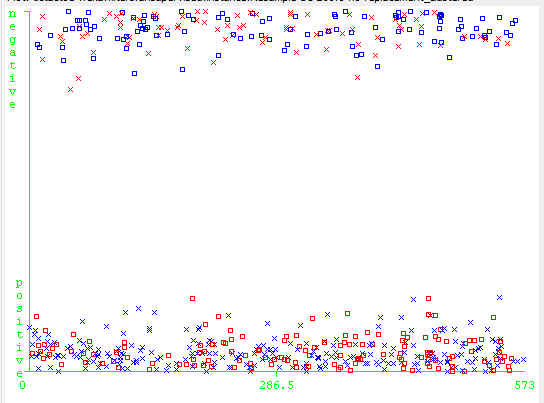
The EM did not perform as well as k-Means when classifying data. The lowest error achieved was 43%.

How does the soft clustering affect performance in this case?

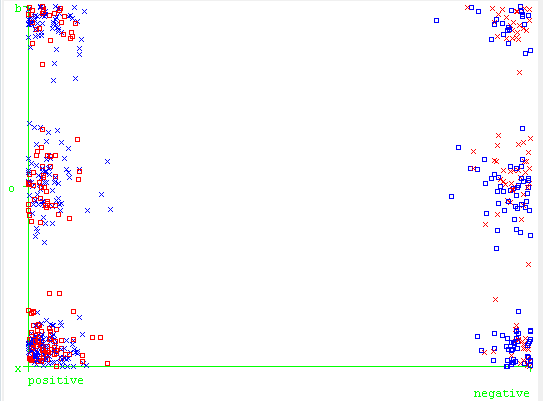
EM automatically turns data into Gaussian distribution. How does this affect results?

For both EM and k-means cluster 0 is the positive class and cluster 1 is negative.

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 class contains instances from both clusters. This is an indication that EM is not clustering the classes well.



*Figure 4: Cluster plot, EM, class vs. instance*



*Figure 5: Cluster plot, EM, pos1 vs class*

**Part 2**

**PCA**

Variance covered option tells how much of original interesting stuff in the data we want to keep.

Correlation matrix: Diagonal is standard deviation. Matrix is symmetric. Each row reps feature. Each column reps feature. Entries represent correlation between features. 0 means no correlation. The rank represents the amount of variance that attribute covers. Keep the top attributes amounts to keeping the most variance. The addition of the different features tells us the relative importance of each. Higher proportion means its more important.

Use Ranker numToSelect to choose best attributes

Intuitive understanding of why variance is good for classification. If you have attribute with all the same value it provides no information when trying to classify, its variance is zero. When variance is high there are many values to work with for our classifiers.

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 (after final length, see if you can fit it). The correlation matrix shows that all the attributes have a very low correlation with each other. This means each of the new attributes offers a much needed dimension to the problem.

The eigenvalues for the top 5 attributes are 2.3306, 2.295, 2.147, 2.074, 2.013. After the 5th attribute the eigenvalues drop below 2.

**Random Projection**

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.

**Other Feature Selection alg**

Possibilities:

PartitionedMultifilter

RandomSubset

**Part 3**

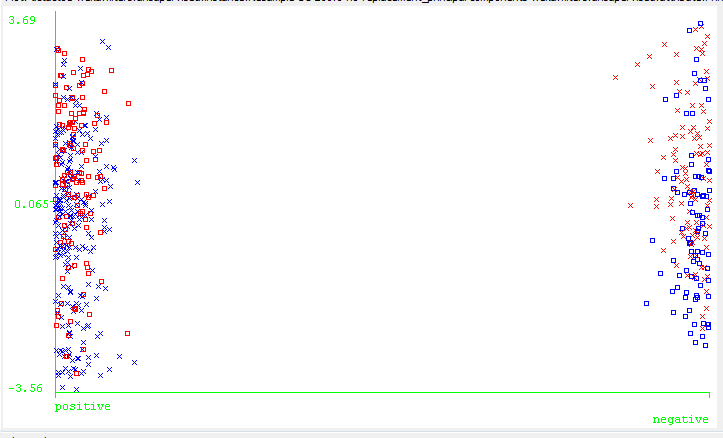
**PCA**

**K-Means**

Top 5 and also top 6 attributes give about 36% error 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. Get this measure for other part.

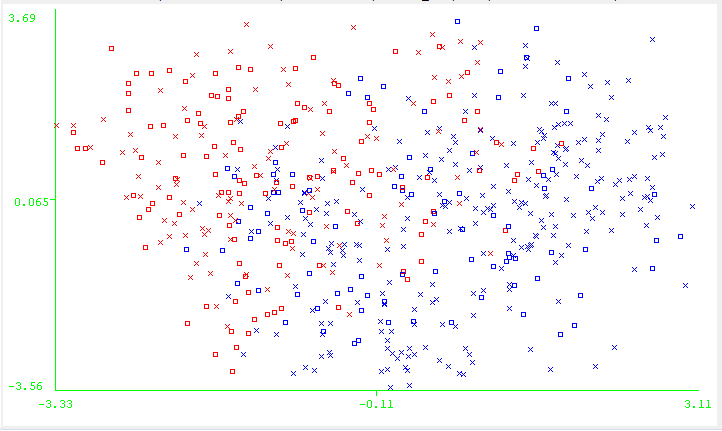
Only 11 iterations are need for convergence. T1 and T2 need to be bigger than previous part. Now they are both 10.

Figure 6 shows a cluster plot of the first selected attribute versus the class. 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. The jitter function was used here to bring the points out of a straight line on the axis for better visualization.



*Figure 6: K-means attribute 1 vs. class*

Figure 7 shows a cluster plot of attribute 1 vs attribute 4. This plot shows a clear division into 2 clusters with some overlap. The arrangement of points can be seen to come from two different Gaussian distributions. The cluster for attribute values vs each other are similar and show a clear division. This particular graph shows that large values of attribute 4 coupled with small values of attribute 1 come from a distinct Gaussian. In addition, large values of attribute 1 couple with small values of attribute 4 come from another distribution.

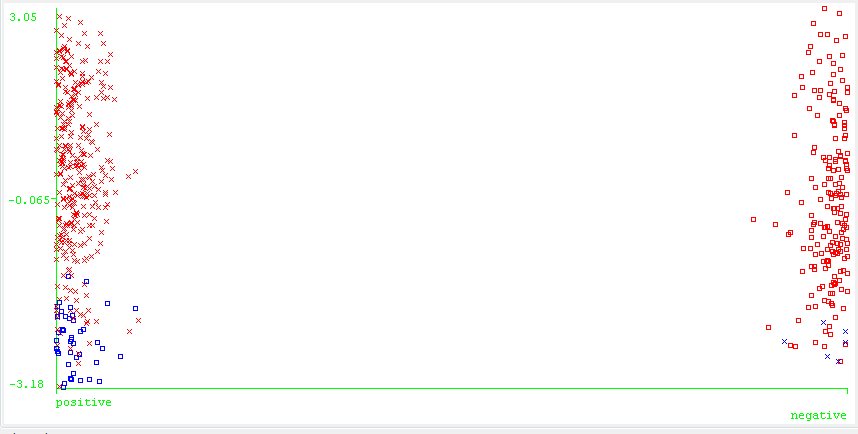


*Figure 7: k-Means, attribute 1 vs attribute 4*

**EM**

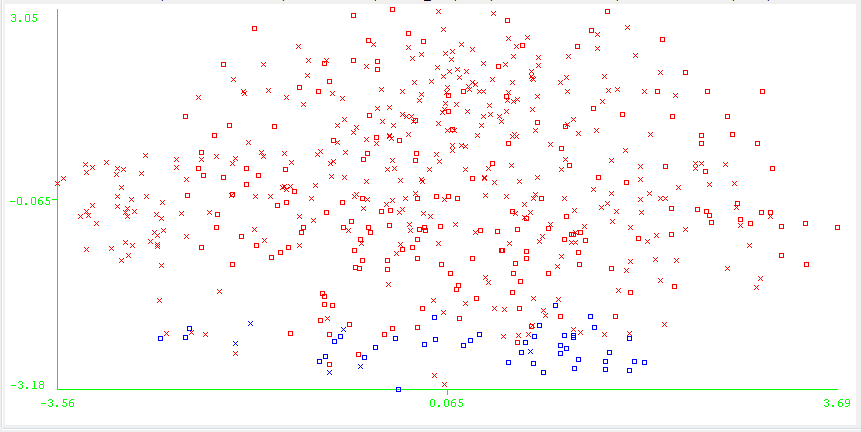
Cluster 0 corresponds to the negative class and cluster 1 corresponds to the positive class. The EM converges to a minimum error of 39.7% after 156 iterations. The clusters here are much more imbalanced with cluster 0 only containing 41 of the 574 instances.

Figure 8 shows the cluster plot for the third attribute vs the class. The Blue is cluster 0 and the red is cluster 1. All the cluster 0 points are in the bottom left corner indicating that if attribute 3 value is small and the point belongs to cluster 0 then likely the classification is positive. The figure shows that the two clusters are not as mixed for the EM compared to the k-means in figure 7.



*Figure 8: EM, PCA, Attribute 3 vs class*

Figure 9 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.



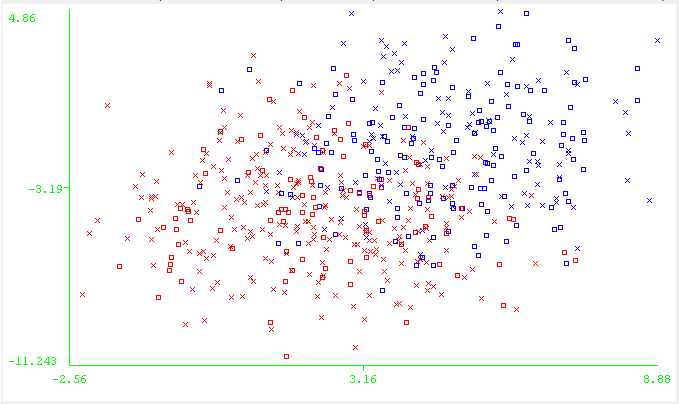
*Figure 9: EM, PCA, Attribute 1 vs Attribute 3*

**Random Projections**

**k-Means**

For Random Projection the distribution used was Gaussian. The number of attribute values was reduced to 7. Different attribute reduction values were tested with differing random seeds. 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 10 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.

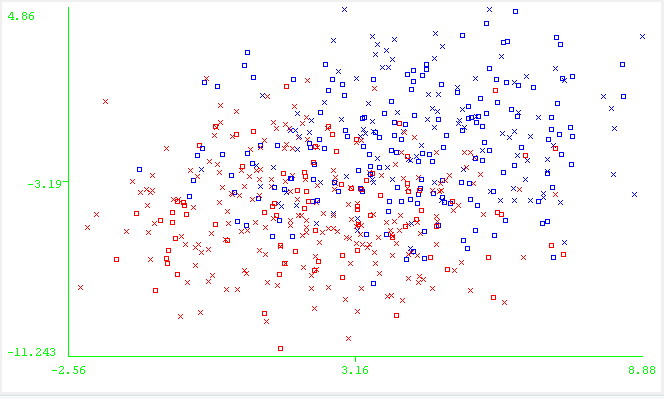


*Figure 10: k-Means, RP, K5 vs K2*

**EM**

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 11 shows the cluster plot for attribute 5 vs attribute 2. Here we can the clusters form two distinct Gaussian distributions similar to the k-Means. However, the EM has created distributions that are 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.



*Figure 11: EM, RP, K5 vs K2*

**Part 4**

**Part 5**

It sounds like AddCluster in the filters in weka will do this part, possibly also ClusterMembership.

**Conclusion**

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

Chess Data Set

Why is it Interesting?

The first 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. 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. For this data set a false postive is characterized as a “nowin” being classified as a “win”. A false negative is a “win’ being classified as a “nowin”.

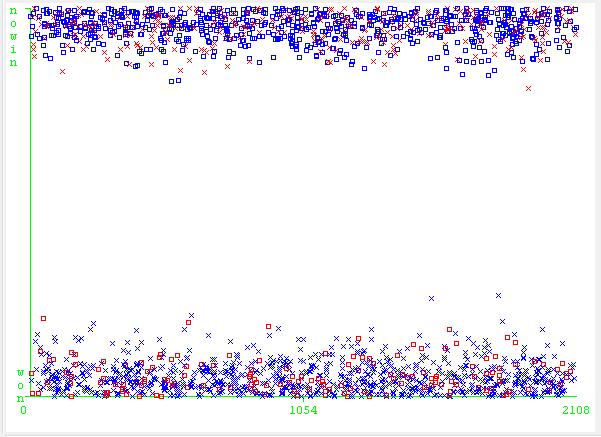
**Part 1**

**k-Means**

For the k means 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 12 shows a cluster plot for the class vs instance.



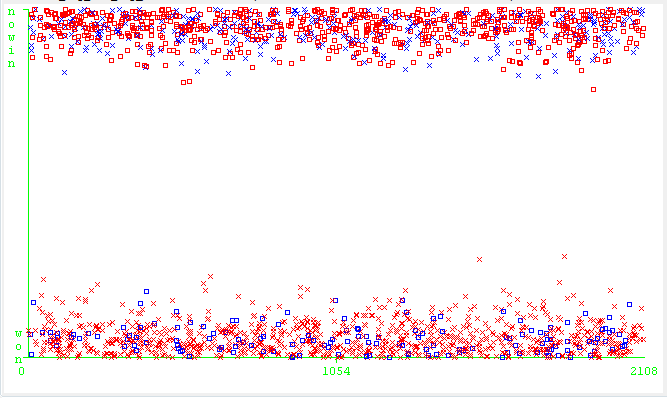
*Figure 12: Chess set, k-Means, class vs instance*

**EM**

EM was initialized using k=2. Only 46 iterations were needed. 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. Varying these parameters resulted in higher classification error with the highest being 65%.

In both the k-means and EM figures one cluster dominates. Both have about a 3 to 1 ratio.

Figure 13 shows the cluster plot for class vs instance number.



*Figure 13: Chess Set, EM, class vs variance*