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.

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”.

**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

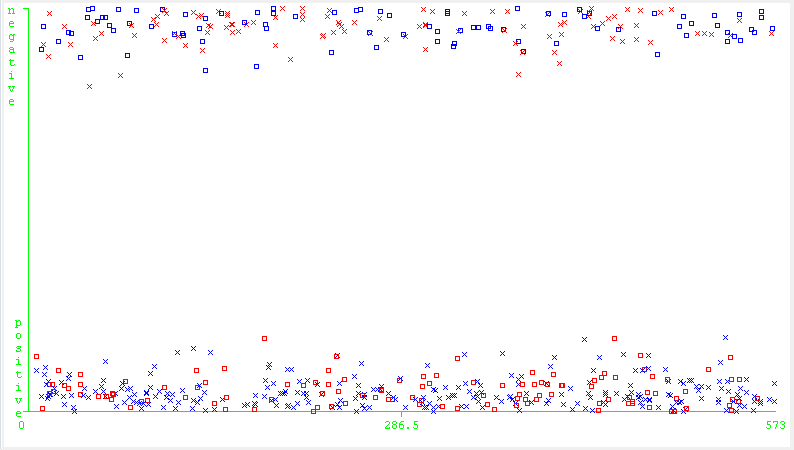
pos8 x x o

pos9 x x o

class positive positive negative

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

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.

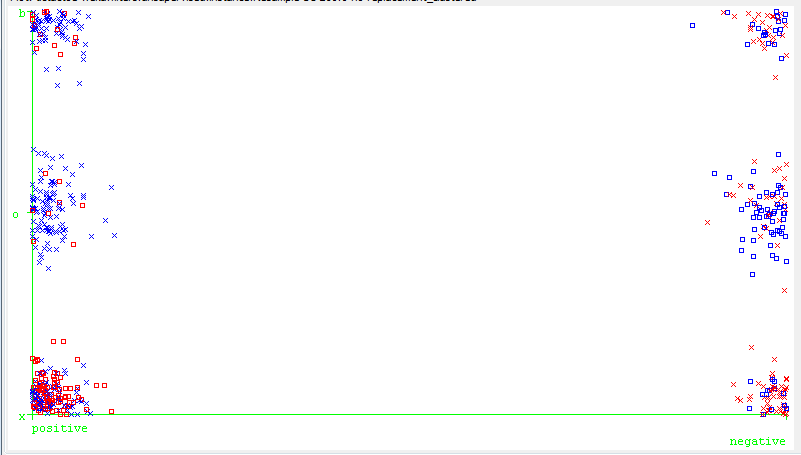


*Figure 2: Cluster plot, K-Means, class vs. instance*

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?

**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.

**Random Projection**

With a Gaussian and 5 attributes error got down to 43% with 13 seed.

**Other Feature Selection alg**

Possibilities:

PartitionedMultifilter

RandomSubset

**Part 3**

Top 3 attributes give about 39% error with simple k means.

Graph idea: error vs number of attributes

**Part 4**

**Part 5**

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