kNN2 methods used to improve accuracy.

Distance method:

The first method I changed in order to improve the classification method was the distance calculation I was using in order to find the distances between the test data and the train data. I implemented two other formulas Manhattan and Minkowskis. Manhattan proved to be the most effective and therefore I used this in my kNN2.java program. The Manhattan distance function works by working out the sum of the absolute distance between the two points, in this case each test[i] value against every single train[j] value. This returned a double, which was then stored in the distances array.

Increasing the amount of K:

This involved increasing the amount of smallest distances being taken by the program in order to get a more accurate label as it takes in more data to decide what class the data should be set to. It also reduces the affect that any outliers may have in skewing the accuracy of the algorithm, as it will take a majority from the closest neighbours as oppose to just one. In order to implement this into my program I initially created a new array called indexes which stores index from 0 – size of data set. I then used a combination of Arrays.sort() and a Comparator. This worked as it effectively maps the indexes to the distances and then sorts the distances array in descending order. As the indexs are mapped to the distances they are also sorted in respect to their corresponding distance. This allowed me to see where the smallest distances were in the data set. Therefore, I could then see what the label was at that index and then add it to an array of size k(amount of nearest neighbours) and pass it into the majority method which would return the number with the most occurrences and set that as the prediction. Once this method was implemented, I ran it with different values as k, for example 3,5,7,9,11. However 3 returned the most accuracy so k = 3 in my final iteration of kNN2.java.

Removing noisy data

Another method I would've used to increase of accuracy of the kNN algorithm is to remove any noisy data which maybe be negatively affecting the algorithm by throwing it off. I opted for using the Z-score in order to identify and then remove the columns that are outliers. The z-score works by initially calculating the mean and the standard deviation. For the mean I looped through the feature columns individually and worked out mean, then using this mean to work out the standard deviation of each column. Standard deviation can used to see data which have large differences and therefore may not be suitable for accurate analysis. The z score combines the mean and standard deviation to give the amount of standard deviations the data is away from the mean. I would then again use a comparator to map the z scores to the different columns and then find the highest z scores and what column that relates to. This would then allow me to loop 10 times a call a remove column function on the columns with the highest z scores. A negative or positive score is returned. A threshold can then be set to remove the data is above the threshold therefore any data that is noisy gets removed.