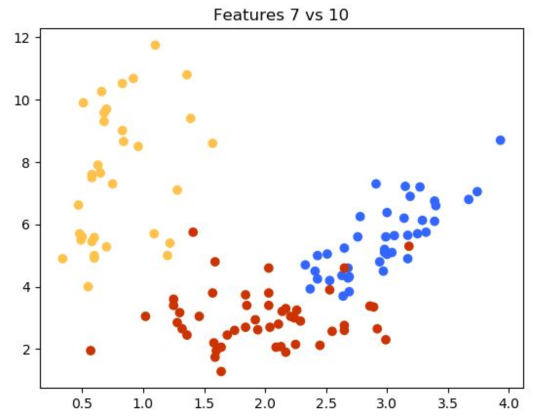
Report for SPS coursework 2

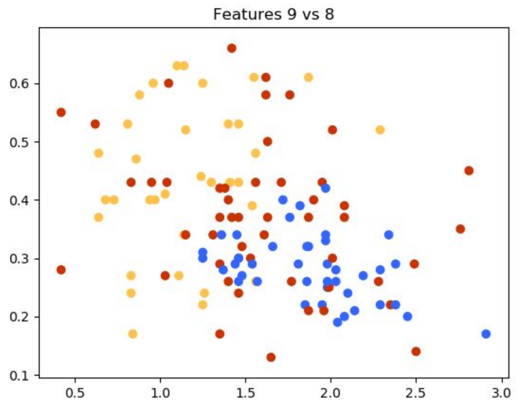
Introduction

This report demonstrates the process of data selection from the known dataset with its labels, by selecting features based on a group of graphs, then the creation of different classifiers that will be classify the future data according these features' training. Besides this, the accuracy is tested using extra test dataset and with its labels.

Feature Selection

Before we start it, one thing have to be clear in the mind, what is feature? Modern machine learning rarely uses raw data input to perform learning tasks. And the raw input is usually transformed into a more powerful representation--features. That's why we have to find feature transform function to help us define our own features from dataset. In this case, we just choose f manually based on the observation of the graphs shown. That's always efficient and working well when our dataset is not very huge. In order to see all of the features, we had to produce a set of scatter plots of all of the features. Then, we are able to see which pair of features separate the data the most, and which pairs have clusters with the least amount of variation within the clusters. Below is the scatter plot of the two features we selected (7v10) and a comparison of one we did not (9v8).



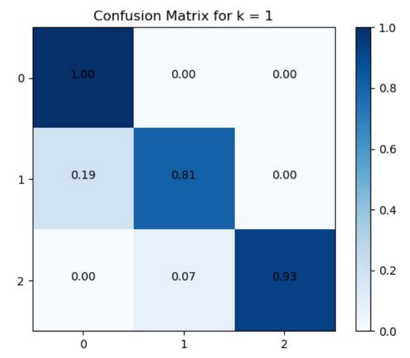


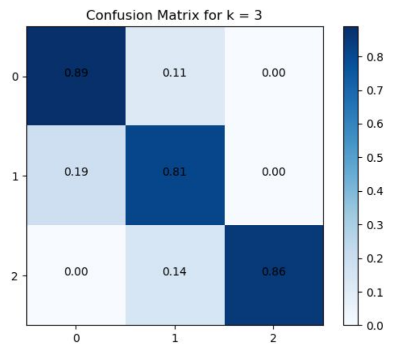
Features 7 and 10 (6 vs 9 in index form) have a good distinction. Although the yellow class is quite spread out, the red and blue have less intra-cluster variation and so we thought this would be a reasonable choice for our features. More distinct and separated data clusters means that there are fewer points that are within the clusters of the class they do not belong too, meaning that they are less likely to affect the classifier selection and produce an incorrect classification. In comparison, features 9 and 8 have very poor distinction and therefore if we used these to build our classifier we would not get accurate results.

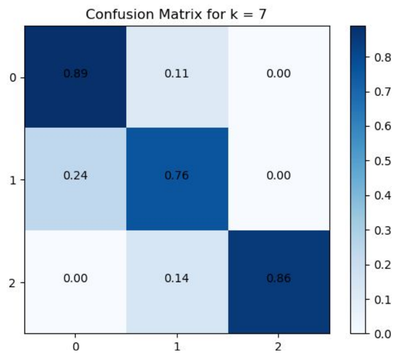
Knn-classifier

We are given a set of labeled training set, then assign the test set distance to the class of its k nearest neighbours. First we calculate the distances from test point to all of the points in the train set, and then we select the k nearest points. Out of these, we select the majority class and assign this to the test data. If there is no clear majority, we try again with k-1, and so we simply get rid of the furthest neighbour and try to find a majority. This continues until a class is assigned. The knn classifer is useful as it does not require us to make any assumptions about our data, and easily evolves if we collect more data. However, there are some disadvantages to using it. Firstly, if the train data does not have a balanced number of each class then it becomes easier for a piece of test data to be incorrectly classified as the data in which there is a majority in the train data. It also has no capability of classifying data that does not lie within the range of the train data. Also, choosing the k that provides the optimal accuracy is not clear cut and for new data requires evaluating rather than having a set answer.

For our data, varying the value of k changed our results, albeit not by much. Although the percentage varied across an approximately 10% range, the amount of test data was relatively small and so one error in classification of the test data made a large difference in the accuracy.
This was due to the fact that class would only change when you increased k if there was a large amount of different class nodes introduced relative to k, and so a new majority would be picked.
For small values of k, if the test node happened to be near an outlier, this could affect the choice especially if the nearest neighbour was this outlier. However for large values of k if a test node was on a boundary or near a boundary between two classes, there may be a larger majority of the wrong class for that data piece. This is why we tested it for many different values of k. Due to the fact that the clusters for our features were not all next to each other (if you see the graph above you can see that class 1 and class 3 are not overlapping at all), we found that for all values of k the amount of class 1 being incorrectly labelled as 3 and vica versa was always 0. This was to be expected given our feature selection, and it is partly why we picked those features. The confusion matrices give us a good representation of how accurate the classification was rather than just a percentage, as we can see how each class performed individually and how they were classified.
Our results for the knn classifier are at the end of the document in a table, with the knn for our selected features being in the column labelled knn. We have also included some confusion matrices below to give a taste of how it performed over varying k. The graphs were mostly similar for all k, and so we included 4 as a selection. These are for k=1,3,7.







Naive Bayes Classifier

For Naive Bayes Classifier, to make sure all the class independent, firstly we just separate all the class based on the unique number of labels with selected features. Each class appends with the selected features.

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

For prior(probability density function) probability calculation, which is a pre-condition for Naive Bayes model, we want to check the total number of test data points matched in three class, then get the percentage of these data point in the length of total training data. This is the process of converting the data set into a frequency table.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Then we need to create likelihood by finding the probabilities of each feature for all 3 class, we achieve this by using normal distribution, actually, using the different value of mean and standard deviation.

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

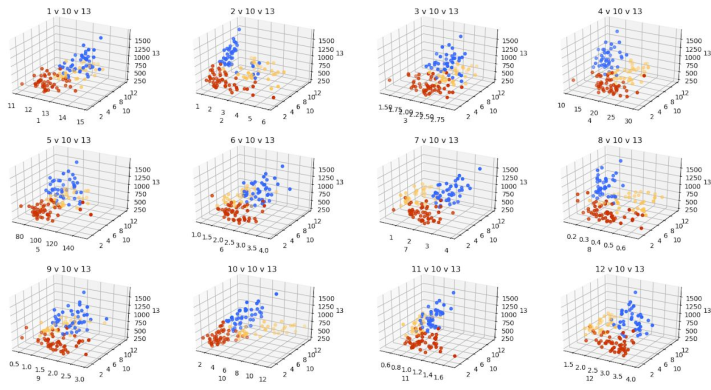
To obtain the complete likelihood we need to obtain the values for each by looking at the dataset and substitute them in the probabilities function, the get a multiple of probabilities of each feature, and for all entries in the dataset, the denominator does not change, it remain static, therefore it can be removed and a proportionality can be introduced.

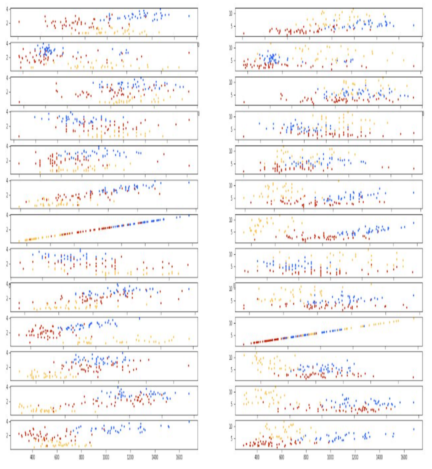
To avoid substituting the likelihood so many times assume that we have a large group of features, we use dynamic method to append our results altogether.
Finally, we can estimate the accuracy of the model by making predictions for each data instance in our test set, To look for the largest probabilities and return its associated class, we use argsort to select the posterior value we want. Thereafter, the predictions can be compared to the values in the dataset, by comparing the labels associated with, then count the number of correct ones, we got the exact accuracy, for feature [7,10] is 75.471698113%.
In comparison with K-NN, Naive Bayes classifier cares more about the global dataset, if we got outliers which are so far from the test nodes, K-NN would ignore it and it will not make difference. But Naive Bayes classifier does.for the current result, it says that K-NN looks more effective than Naive Bayes one, that's because our feature is not comprehensive enough, and if a variable in test set which iis not observed in training data, the model will assign a 0 probability and will be unable to make predictions.and the other limitation for Naive bayes is the assumption of independent predictors, however, it's almost impossible to get a set of predictors which are completely independent. But it's still a reliable choice, it's easy and fast to predict class of test set even multiple class.

Three features selection
Using 3 features means that we have a 3 dimensional graph in which we select the k nearest neighbours, and so the distance measure had to be editing to be Euclidean distance in 3 directions. This meant we had to have a new distance measure function but after could use the same functions from the 2-dimensional knn.

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2}.$$

The main difficulting was visualising the data in 3 directions so that we could pick a feature. To do this we tried two approaches. Firstly we plotted all of the features against our two features individually, so produced 2 sets of 13 graphs, and then we produced a set of 3d graphs of each feature against our two features we thought had good separation. These are below. We picked a third feature that had good separation when plotted against each feature individually and both together, and we picked feature 13. This meant we had features 7v10v13 in final.
The accuracies we achieved with 3 features were not better than with 2 features, in fact much worse by about 10-15% on average. This could be because we increased the dimensions, and hence had issues with the 'curse of dimensionality', and so as the volume of space increased by adding another feature the data became more sparse. It was also harder to pick a third feature as visualising the data is trickier, and so this could have added to the lack of accuracy also.

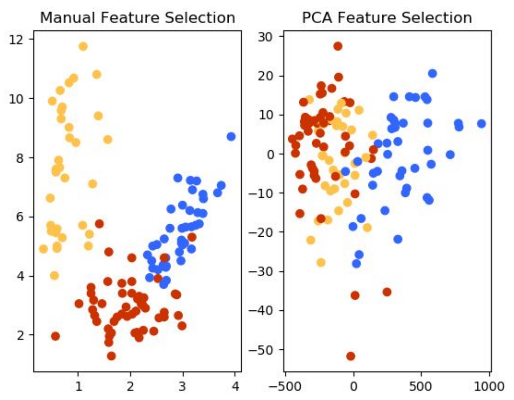


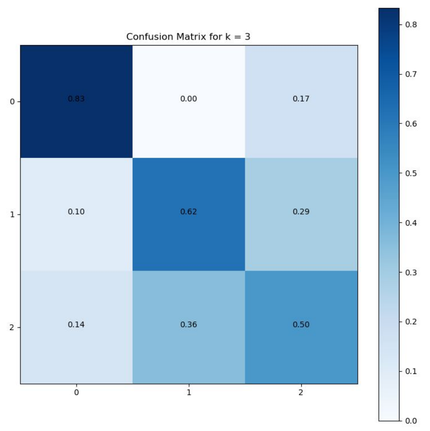


PCA

Principal component analysis uses the eigenvalues and eigenvectors of the covariance of the data to reduce it to get rid of redundant functions by minimizing the projection error. This is to solve the issue of the curse of dimensionality, which is when the dimension number increases, the performance may decrease.

We used scipy's pca function to reduce our data to 2 features, and then ran this data through our knn function. We produced a scatter plot comparing this to our manually selected features.





As shown in the final table in the column Knn_PCA, the results obtained are far worse than with our manually selected features. Looking at the scatter plot above this makes complete sense: the features chosen are not well separated and do not have small intra-cluster variation. The confusion matrix also shows that some of class 1 and 3 have been misclassified as each other, which is something we completely avoided with our manually chosen features. This is clear from the scatter plot - there are instances of overlap from all three classes in all other classes, and so is expected. Although PCA is helpful when you have a very large amount of features and manual selection is not possible, when you are able to select the features it seems to yield better results, for our data certainly. It is a useful tool, but not for our data set as it did not perform well.

K	KNN Accuracy	KNN_3d Accuracy	KNN_PCA Accuracy	Naive Bayes Accuracy
1	90.566%	75.472%	71.698%	75.472%
2	86.792%	69.811%	64.151%	
3	84.906%	77.358%	66.038%	
4	86.792%	75.472%	71.698%	
5	79.245%	69.811%	75.472%	
6	84.906%	79.245%	81.132%	
7	83.019%	75.472%	75.472%	

In conclusion, the highest accuracy achieved was when we chose 2 features manually and used K-nearest neighbours to classify the data. A value of k=4 provided a high percentage here, and so we could pick this. Although k=1 also provided a better result, we think that any stray outliers could easily affect the classification greatly and so may not consistently perform well across different data sets, and so this is why we would not choose this if we needed to pick one to implement for other data, however for our data and similar k=1 would be what we would pick.