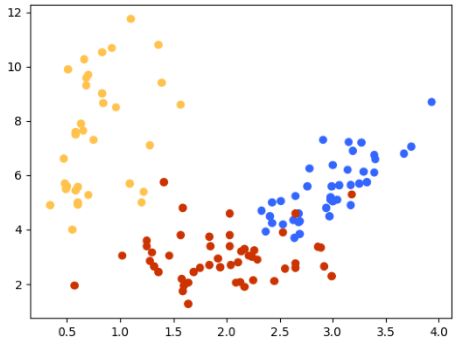
SPS - A Memory of Wine

Chanthru Uthaya-Shankar – cu17780
Jake Ramaer - jr17327

This report summarises the different classification methods implemented in wine_classifier.py and discusses the relationship between the shape of our supplied data and the prediction accuracies relative to each classification method.

Feature Selection

We initially chose our features by looking at the plots for each individual set and choosing features with minimum overlap. These features are the most linearly separable, and therefore the best data to use when training our classifiers.



[7,10]
Blue -1, Red - 2, Yellow – 3

After developing all the classification methods we found the optimal features by running KNN for every feature combination and calculating their accuracy with K=1. We found that the combination [7,10] gave the highest accuracy of 90.6%, and was therefore one of the better pairs to use for both our KNN and ALT classifier.

KNN

With features 7 and 10, the highest accuracies were produced when the value of K was set to 1, this is due to the classes already being in clusters with low overlap. As the value of k gets higher, the classifier considers more values of the training set when assigning a class to the specific test point and the classification process starts to become more general.

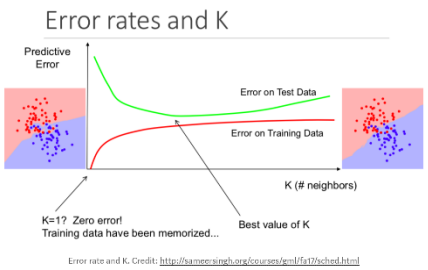


Fig 2

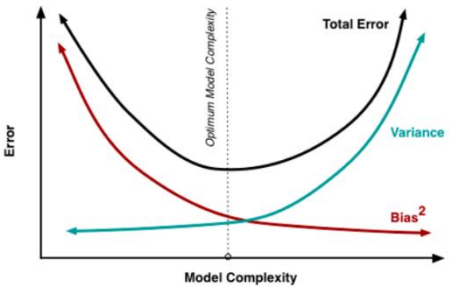
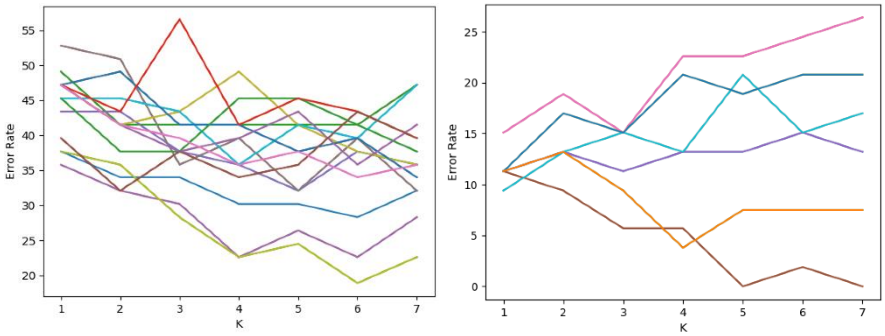


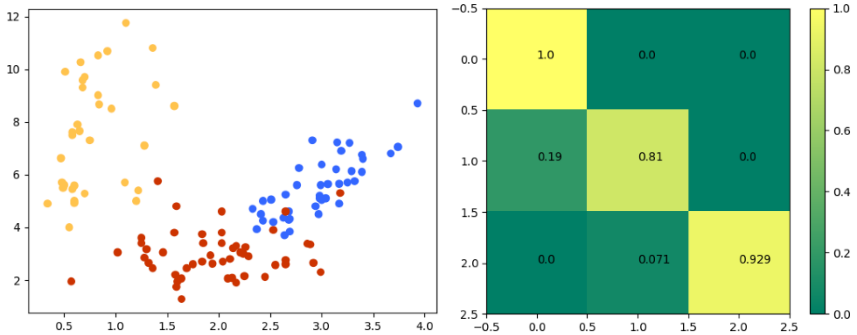
Fig 3

When we opt for features which have higher class overlaps, we find the value of k greatly affects the accuracy of our model. This is because when using features with lower separation we find more overlap between classes, which means a larger amount of values are closer to (and on the wrong side of) the classification line between the two classes. When K is low, we are overfitting on the training data, creating a more jagged boundary classification line, which has a higher variance when classifying the test data. As K increases, this jaggedness smooths, and the accuracy increases up to an optimum value. After this value, the classifying line becomes too smooth, and begins to underfit the training data, leading to a high bias when classifying the test data.



Above (right) was plotted by running all features with (K=1) accuracies below 55% with different K values. The curves created mimic the green 'Error on test data' curve in Fig 2. The trend is no longer apparent when using (K=1) accuracies above 85%, pictured left. This supports the notion that the value of K has less of a positive impact for features which are already well separated. By observing the above graph, we find the optimum value of K to be 5.

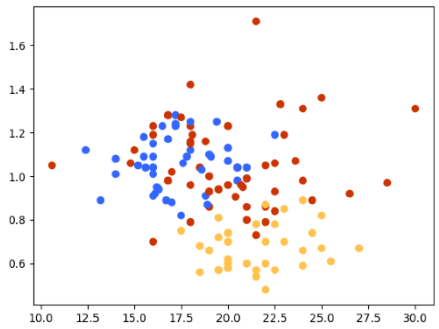
A major drawback of using KNN with larger values of K stems from the relative weights of each neighbour and their contribution to the overall classification. In an implementation without weighting (such as ours), training points which are further away contribute equally towards the classification as closer points do. This means that features which have higher variance and overlap will falsely classify the test points if the majority of its k nearest neighbours are of a different class, regardless of the distance from the test point.



To discuss the behaviour of our algorithm, we have chosen features [7, 10] again as they give a high visual overlap between 2 (red) and the other classes, but no overlap between 1 (blue) and 3 (yellow). The confusion matrix produced shows 0 misclassifications between 1 and 3, and a small error between 2 and the other classes. This is because KNN, being a lazy evaluator, works off its nearest points. Therefore, the further 2 classes are away from each other the less likely they are to misclassify.

ALT

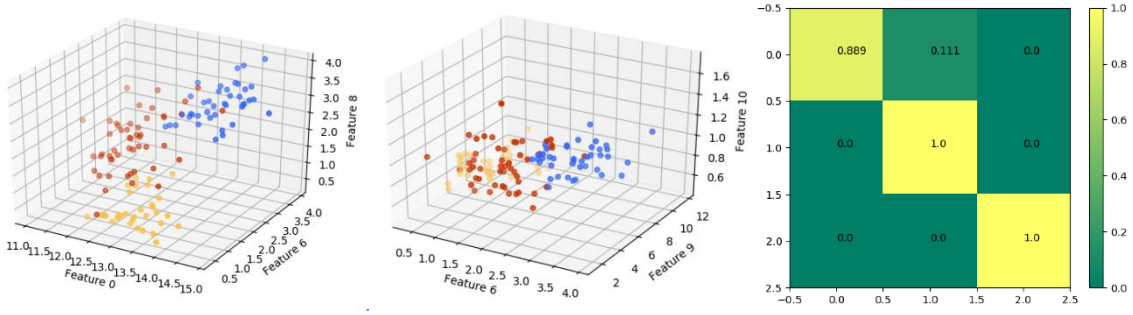
For our alternate classifier we chose to implement Naive Bayes. Using [7, 10], we obtained a final percentage accuracy of 84.9%, far poorer than KNN's accuracy of 90.6%. We believe KNN performs better in this case as it utilises lazy learning. This means the algorithm is instance-based and utilises only the data close to it - its k nearest neighbours. Naive Bayes, on the other hand, uses general evaluation, which takes every point in the dataset into account by considering entire class means & standard deviations. With this data, it creates a model that encompasses the whole dataset. Because we chose features which ensured the data was already well separated, a lazy evaluation algorithm will perform better, as points with identical classes will be closer to each other. Thus in our case KNN gives a higher accuracy than Naive Bayes. To further illustrate this point we ran our algorithms using features [4, 11], whose features highly overlap. Our Naive Bayes performed better than KNN (66.0% vs 52.8%).



In addition, Naive Bayes' (NB) accuracy is also largely affected by outliers. KNN (for small values of K), completely ignores outliers, again due to it being lazily evaluated. Our feature selection [7, 10] involves several of these outliers, which reduces accuracy of the class mean calculations in NB, and therefore its overall classification accuracy.

KNN 3D

When selecting the 3rd feature we had to consider what the data would look like when represented in 3D. Based on our decision when selecting 2 features, we initially chose features 7 and 10 to create a function that ran our KNN-3D classifier program with every other feature. 11 was selected as the best feature to use in conjunction with 7 and 10, producing an accuracy of 88.7%.



Later we decided to assess all possible combinations, which produced [1, 7, 9], giving an accuracy of 96.2%. We thought our 3 features would include both 7 and 10, but our system found that [1, 7, 9] were more linearly separable. Features 7 and 10 show that between them their values can best identify a class, as shown by the separability and clustering of classes in the plot in Fig 1. However, in 3 dimensions, [1, 7, 9] produced more separable clusters with less variance compared to features [7, 10, 11].

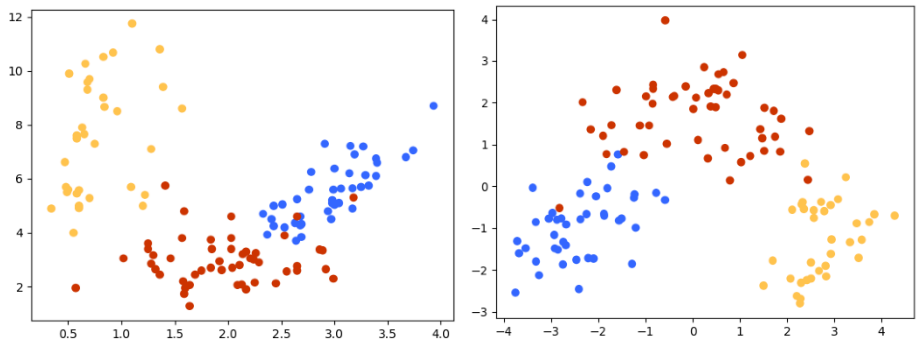
Using an extra feature was beneficial as it gave us more data by which to identify a specific class with. This resulted in a slightly higher accuracy than using the 2D classifier. However, the main drawback for using the extra 3rd feature was the overall runtime of our classifier. There was a noticeable increase in time to compute the classification, which shows that the

3D classification model would not scale well as the size of the data set becomes extremely large.

KNN_PCA

PCA is able to reduce all features of a dataset to 2 principal components by decreasing the impact of those which show less separability. Our implementation of PCA feature extraction produced a dataset with accuracy of 96.2% when using the KNN algorithm, as opposed to 94.0% with features [7, 10].

This implementation of PCA standardized the data. Often features with larger scale (<1000) can falsely appear to have more variation relative to features with a shorter scale (>1). By standardising the data, the model is rendered unitless, and all features are measured by their standard deviations away from their own class means.



KNN (left) vs PCA (right)

The features generated by the PCA function are superior to our chosen features [7, 10]. This is because PCA's class boundaries see less overlap, meaning the data is more separable. For example, observe the blue/red overlap with and without using PCA feature extraction. As previously explained, the more separable the classes are the more accurately KNN performs. Therefore, the PCA reduced data produces a more accurate classification when paired with KNN.