CSE 802 Project Report

By Abhiram Durgaraju

Introduction/Description:

The goal of the study is to gain as much insight as possible for a given dataset and various classifiers. The task is to perform a multi-class classification with a large set of features.

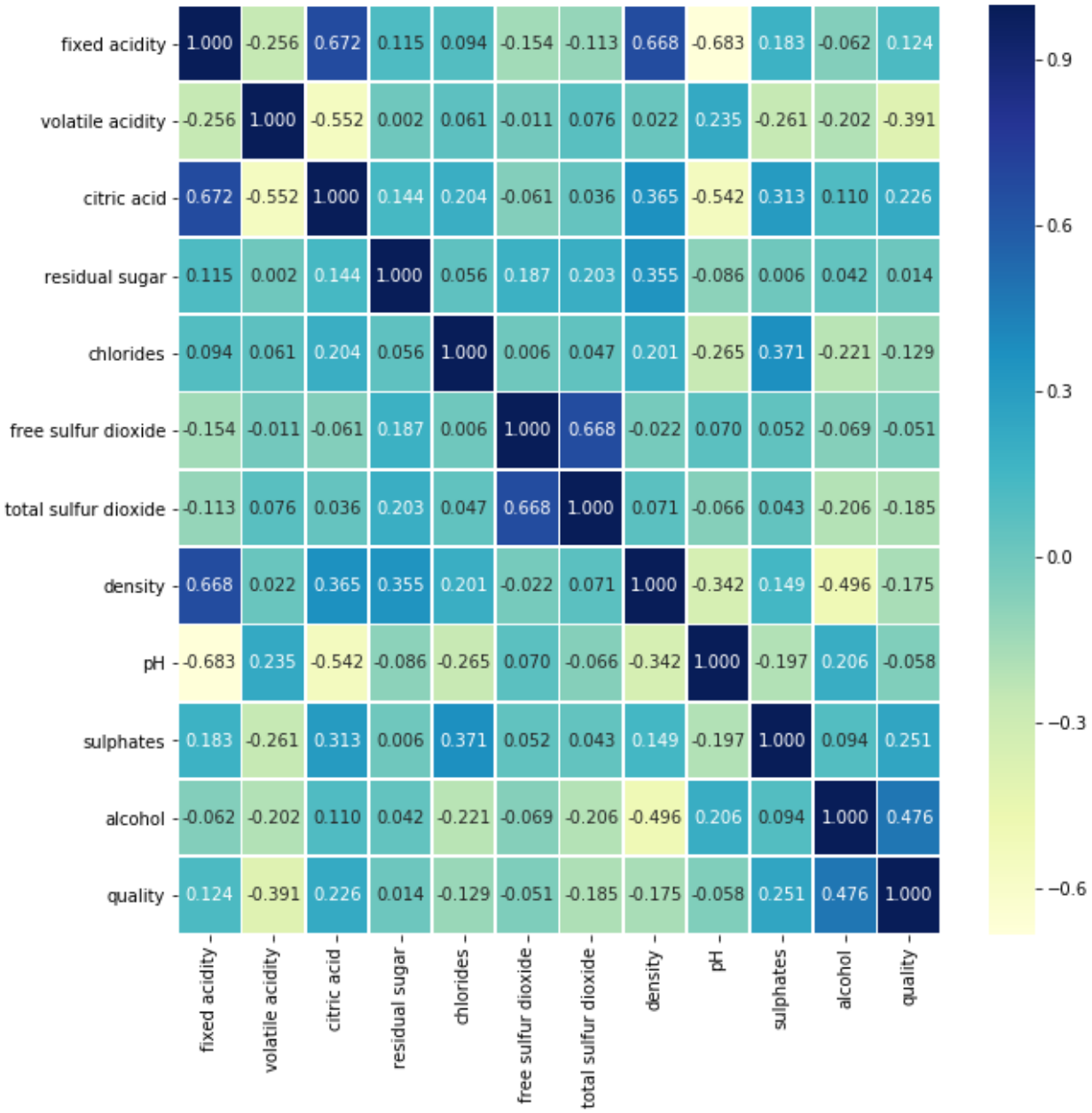
The Wine Quality Data Set from the UCI Machine Learning Repository was chosen for this task. This actually contains two separate data sets containing samples of Red Wine and White Wine variants of the Portuguese "Vinho Verde" wine. For this study, the focus was on Red Wine only. The dataset contains 11 features (fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol) and 1 target label (Wine quality).

The dataset is an imbalanced one with much more normal quality wines (quality=5 or quality=6) than poor or high quality wines.

Classifiers of interest for this dataset include python's sklearn packaged classifers such as SVM, Random Forests, and K-nearest neighbors. Self-developed classifiers using Bayes classification and Parzen window estimation are also goals for this study.

Results/Analysis:

This data set has a large number of features, 11 to be exact. So, it is critical to see the relationship between each of them before any preprocessing to continue. Below is a correlation matrix for all 11 features and the class labels (quality):



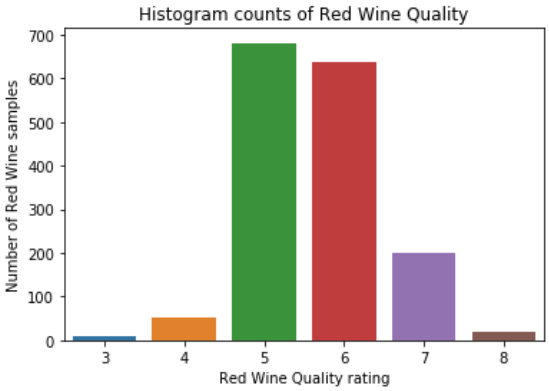
The correlation matrix was calculated using the seaborn library in python. A correlation of 1 means there is a strong positive relationship between the two features. A correlation of -1 means that there is a strong negative relationship between the two features. On the other hand, a correlation of 0 suggests no relationship between the two features.

One can gain the following insights into the data from the above correlation matrix:

- There is a relatively strong positive correlation between alcohol content and quality of the wine
- There is a positive correlation between sulphates, citric acid, fixed acidity and the quality of the wine
- There is almost no correlation between residual sugar, free sulfur dioxide, pH, and the quality of the wine
- There is a relatively strong negative correlation between volatile acidity and quality
- There is a strong positive correlation between density and fixed acidity
- There is a strong positive correlation between citric acid and fixed acidity
- There is a strong positive correlation between free sulfur dioxide and total sulfur dioxide
- There is a strong negative correlation between pH and fixed acidity
- There is a strong negative correlation between citric acid and volatile acidity
- There is a strong negative correlation between density and alcohol
- There is a strong negative correlation between citric acid and pH
- There is almost no correlation between chlorides and fixed acidity
- There is almost no correlation between alcohol and fixed acidity
- There is almost no correlation between residual sugar and volatile acidity

Above are the major insights we can gain. The first three bullets are most important since one can get a general idea of the usefulness of a feature for classification.

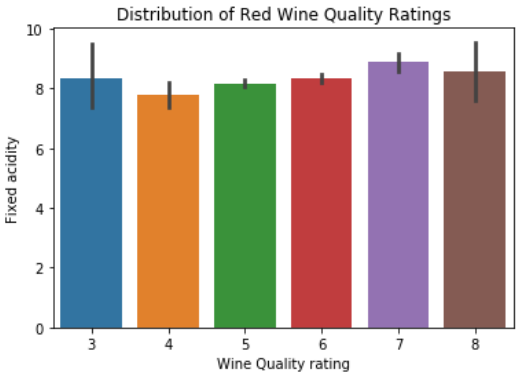
It is also important to see the distribution of samples relative to the target label since that is the interest for classification. Below is a histogram of wine quality vs. number of red wine samples:



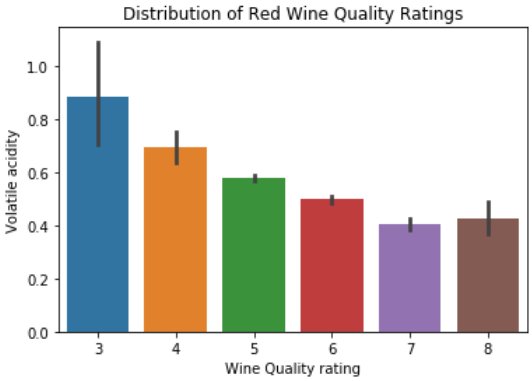
It can be seen that the majority of red wine samples were given a quality rating between 5 and 6. So, there is clearly an imbalance of classes in the distribution. This may affect classification accuracy since it is relatively safe for the classifier to simply learn the difference between wine quality of 5 and 6 and still perform relatively well on the entire data set.

With a really low number of samples from samples of quality 3 and 8, its pertinent that splitting the dataset into training and testing should take into account the distribution of classes. Or else, its possible the one of the datasets may not contain any samples from these classes. This was the primary reason for using Stratified split.

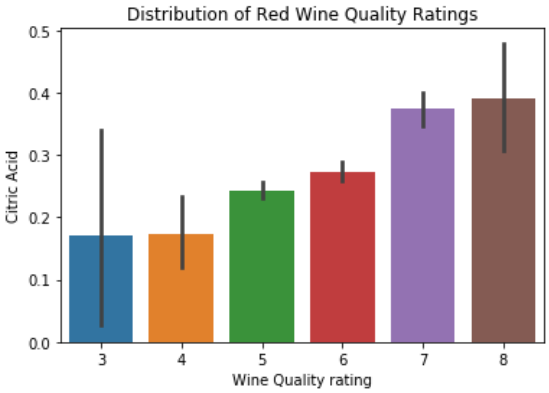
It is also important to see the range of feature values in relation to red wine quality to help with the decision of data normalization based on outliers. Below are 11 plots for each feature vs. wine quality:



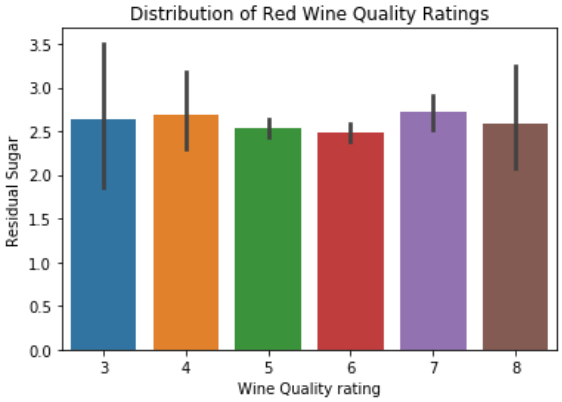
As seen from the correlation matrix, there is a positive correlation between fixed acidity and quality (corr = -0.124). The feature values range from 4.6 to 15.9 for the entire data set, with a mean of 8.31. The majority of this variation is seen at quality rating of 3 and 8.



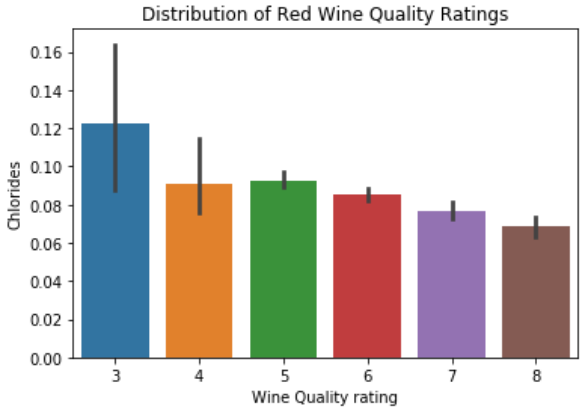
As seen from the correlation matrix, there is a relatively strong negative correlation between volatile acidity and quality (corr = -0.391). The feature values range from 0.12 to 1.58 for the entire data set, with a mean of 0.52. The majority of this variation is seen at quality rating of 3 and 8.



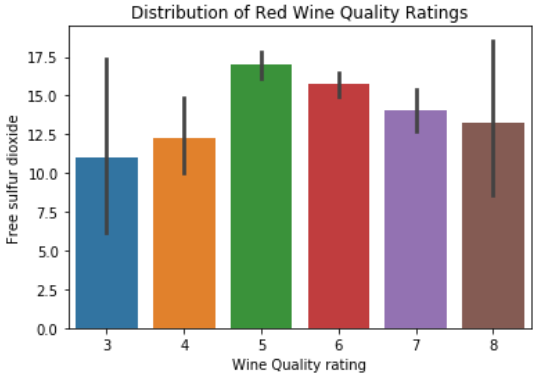
As seen from the correlation matrix, there is a positive correlation between citric acid and quality (corr = 0.226). The feature values range from 0 to 1 for the entire data set, with a mean of 0.27. The majority of this variation is seen at quality rating of 3,4 and 8.



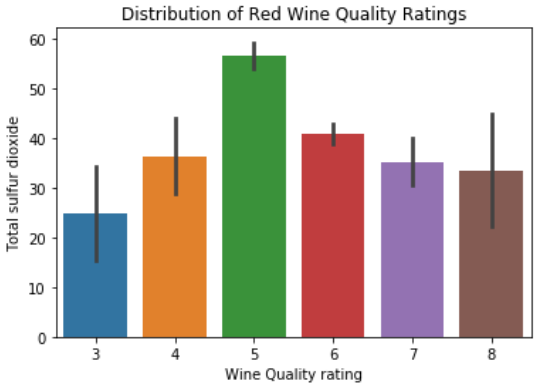
There is almost no correlation between residual sugar and red wine quality (corr = 0.014). The feature values range from 0.9 to 15.5, with a mean of 2.53. The majority of the variation is seen at quality rating of 3,4, and 8. This is a good candidate for removal during feature selection



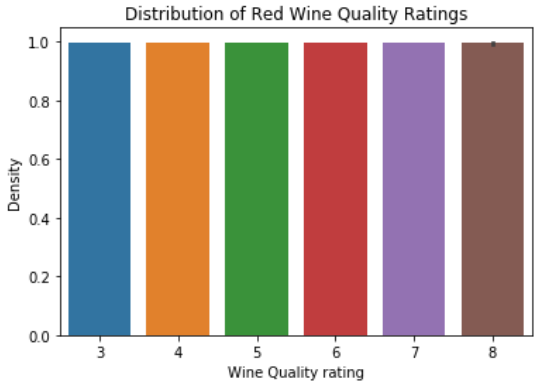
There is a negative correlation between chlorides and red wine quality (corr = -0.129). The feature values range from 0.12 to 0.611, with a mean of 0.087. The majority of the variation is seen at quality rating of 3 and 4



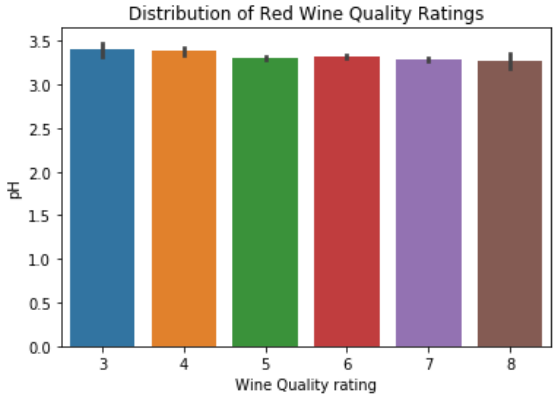
There is almost no correlation between free sulfur dioxide and red wine quality (corr = -0.051). The feature values range from 1 to 72, with a mean of 15.8. The majority of the variation is seen at quality rating of 3,4, and 8



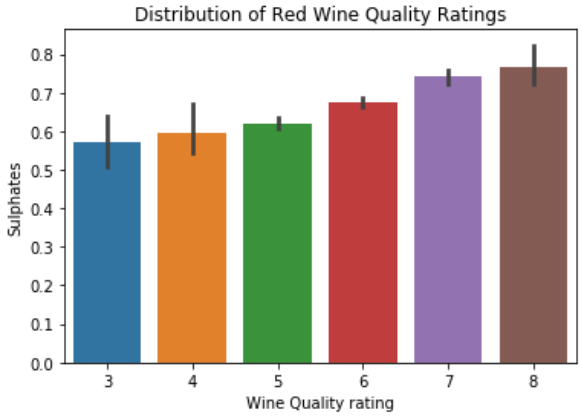
There is a slight negative correlation between total sulfur dioxide and red wine quality (corr = -0.185). The feature values range from 6 to 289, with a mean of 46.4. The majority of the variation is seen at quality rating of 3,4,7 and 8. The wide range of values in each class makes this an unstable feature to use for classification



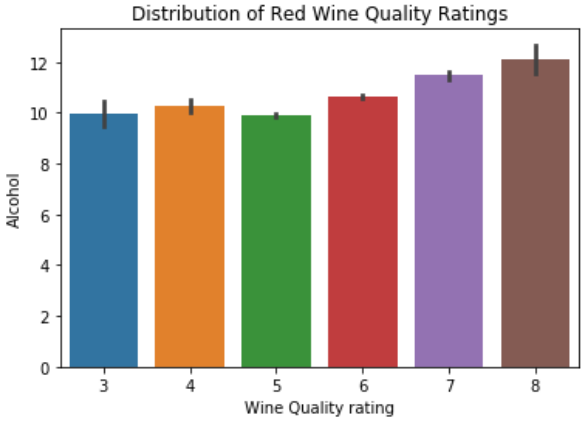
There is a slight negative correlation between density and red wine quality (corr = -0.175). The feature values range from 0.990 to 1.003, with a mean of 0.996. This is unique in that there is very little variation in feature values, but shows a slightly negative correlation.



There is a almost no correlation between pH and red wine quality (corr = -0.051). The feature values range from 2.74 to 4.01, with a mean of 3.31. While there is almost no correlation, it is worth nothing pH is inherently a logarithmic scale and that small variations actually mean large differences in acidity of the wine.



There is a relatively strong positive correlation sulphates and red wine quality (corr = 0.251). The feature values range from 0.33 to 2.0, with a mean of 0.658. The majority of the variation is seen at quality rating of 3,4, and 8.



There is a strong positive correlation alcohol and red wine quality (corr = 0.476). The feature values range from 8.4 to 14.9, with a mean of 10.42. The majority of the variation is seen at quality rating of 3 and 8.

So, as seen from features such as density and pH, it is clear that some form of data normalization is required to capture of the importance of each feature. But, there are so many forms of normalization. This study will explore the performance of three different normalization techniques on a baseline model, such as SVM.

An empirical approach was taken to preprocessing the data set. The data was randomly shuffled and split into a training and testing sets (80/20) and were stratified so that both sets have equal proportion of samples from each class.

First, a single model was chosen as a baseline. In this case, SVM was randomly chosen. For this model, an exhaustive grid search with cross validation of k=3 was chosen to obtain a sub-optimal set of hyperparameters. In this way, the training set is essentially split into 3 folds, where two folds are used for training and 1 fold is used for validation, and this process repeats k times where the validation set changes to a different fold. The performance of the model was then judged based on the performance on the best average validation accuracy across 3 folds. After the model parameters with the best average validation accuracy across 3 folds was selected, it was then used to predict the label of the true test set (which the model has never seen). Then, the original partitioning exercise of splitting the data into 80/20 was repeated with a different random seed (repartitioning seed) to analyze the variation in test classification accuracy.

This process was initially done with no data-preprocessing, which resulted in an extremely long training time and very poor validation and test accuracy as shown below:

Validation accuracy: 56.09%

Test accuracy: 54.12%

Variance in error: 8.40

Therefore, it became evident that some pre-processing had to be done. Data normalization was attempted using the sklearn StandardScaler(), MinMaxScalar(), and the Normalizer() functions.

Overall, the best validation accuracy was achieved using sklearn scale:

Repartition seed	Best average validation accuracy (k=3 folds)	Parameters used	Test accuracy (%)
1	62.47	{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}	60.31
2	63.33	{'C': 100, 'gamma': 1, 'kernel': 'rbf'}	61.25
3	63.64	{'C': 100, 'gamma': 1, 'kernel': 'rbf'}	65.93
4	62.23	{'C': 1, 'gamma': 1, 'kernel': 'rbf'}	63.12
5	62.23	{'C': 10, 'gamma': 1, 'kernel': 'rbf'}	66.25
Mean accuracy			**63.37**
Mean error			**36.63**
Variance of error			**7.19**

The classifier with best classification accuracy for StandardScaled data (66.25%) was chosen. Its confusion matrix is shown below.

	0	1	2	3	4	5
0	0	0	2	0	0	0
1	1	0	4	5	1	0
2	0	1	101	31	3	0
3	0	0	32	88	7	1
4	0	0	4	12	23	1
5	0	0	2	1	0	0

Note that python displays confusion matrices based on index. Index 0 corresponds to wine quality 3, index 1 corresponds to wine quality 4 etc. The predicted class labels are vertical, and true class labels are horizontal.

Here are the results for using MinMaxScalar():

Repartition seed	Best average validation accuracy (k=3 folds)	Parameters used	Test accuracy (%)
1	62.07	{'C': 1, 'gamma': 10, 'kernel': 'rbf'}	60.62
2	63.56	{'C': 1, 'gamma': 10, 'kernel': 'rbf'}	59.68
3	61.06	{'C': 1, 'gamma': 10, 'kernel': 'rbf'}	64.37
4	61.37	{'C': 1, 'gamma': 10, 'kernel': 'rbf'}	62.18
5	62.29	{'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}	65.62
Mean accuracy			**62.50**
Mean error			**37.50**
Variance			**6.29**

The classifier with best validation test accuracy for data using MinMaxScalar (65.62%) was chosen. It's confusion matrix is shown below:

	0	1	2	3	4	5
0	1	0	1	0	0	0
1	0	0	9	1	1	0
2	2	2	103	25	4	0
3	0	0	30	88	9	1
4	0	0	1	21	18	0
5	0	0	1	2	0	0

Here are the results for using Normalizer():

Repartition seed	Best average validation accuracy (k=3 folds)	Parameters used	Test accuracy (%)
1	58.24	{'C': 100, 'gamma': 10, 'kernel': 'rbf'}	57.81
2	57.62	{'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}	57.81
3	58.87	{'C': 100, 'gamma': 10, 'kernel': 'rbf'}	60.62
4	59.18	{'C': 100, 'gamma': 10, 'kernel': 'rbf'}	60.00
5	58.87	{'C': 100, 'gamma': 10, 'kernel': 'rbf'}	65.00
Mean accuracy			**60.62**
Mean error			**39.38**
Variance			**8.66**

The confusion matrix of the classifier with best test accuracy for Normalized data (65.00%) is shown below

	0	1	2	3	4	5
0	1	1	0	0	0	0
1	0	0	8	3	0	0
2	0	1	104	28	3	0
3	0	0	35	90	3	0
4	0	0	1	26	13	0
5	0	0	0	2	1	0

Since the average classification accuracy was highest using sklearn StandardScaler(), it was used for the rest of the study. At its core, this function preprocesses the data such that it is standardized, i.e has a zero mean and unit variance.

In contrast, MinMaxScaler() scales the data so that each feature has values only in between 0-1. The pitfall with this method is that it is very sensitive to outliers, which our dataset seems to suffer from.

Lastly, Normalizer() scales each individual sample such that they have unit norm. This process could be useful with classifiers using polynomial kernel. It did not perform well with the red wine quality dataset.

On to the task of feature selection. Performing a grid search of optimal hyperparameters of SVM for every subset of features considered by Sequential Forward Selection (SFS) drastically increased the training time. Therefore, to simplify the process, the best parameters for standard scaled data calculated earlier ({'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}) were used as a baseline when performing SFS. Again, cross validation was set to k=3. Below is a table summarizing the findings for every d, where d is the number of desired features using SFS.

Number of desired features (d) using SFS	Best validation accuracy	Features used	Mean test accuracy across 5 repartitions (%)	Best test accuracy across 5 repartitions (%)	Mean test error (%)	Variance in mean test error (%)
1	56.83	[10]	53.93	55.62	46.07	1.73
2	58.95	[9,10]	57.12	59.68	42.82	4.95
3	62.00	[6,9,10]	60.31	64.37	39.69	10.00
4	62.70	[3,6,9,10]	59.93	61.25	40.07	4.95
5	63.17	[3,5,6,9,10]	60.62	63.12	39.38	3.07
6	64.50	[0,3,6,8,9,10]	60.18	62.50	39.82	7.64
7	64.50	[0,3,5,6,8,9,10]	61.37	66.56	38.63	10.96
8	63.80	[0,3,4,5,6,8,9,10]	61.56	65.93	38.44	8.78
9	63.25	[0,2,3,4,5,6,8,9,10]	61.50	64.06	38.50	4.95
10	62.24	[0,1,2,3,4,5,6,7,9,10]	61.00	65.00	39.00	9.88
11(all)	63.17	all	61.31	65.93	38.69	10.32

Note that since best validation accuracy across 5 repartitions are used, it is possible for the optimal feature subset of d to contain features that are not in d+1.

Feature #	Label	Feature #	Label	Feature #	Label
0	Fixed acidity	4	Chlorides	8	pH
1	Volatile acidity	5	Free sulfur dioxide	9	Sulphates
2	Citric acid	6	Total sulfur dioxide	10	Alcohol
3	Residual sugar	7	Density	11	Quality

The mean classification accuracy starts to plateau at d = 7. While it is perfectly reasonable to use the subset listed in the table of this page, some more options are available in weeding out features.

The process above was repeated, but correlation matrix was used instead to decide what feature to remove. Initially, all feature |corr| < 0.1 were removed, and the threshold was slowly increased to get desired results. For the sake of brevity, the optimal set of features found was [0,1,2,4,5,6,8,9,10], with mean test accuracy 61.75%, best test accuracy of 66.56%, and variance of 10.86%. These results mimic the finds of the correlation matrix. The two most useless features were residual sugar and density for classifying wine quality.

Dimension reduction techniques such as PCA and LDA were employed as well. Below are the results:

Number of desired dimensions (d) using PCA	Mean test accuracy across 5 repartitions (%)	Best test accuracy across 5 repartitions (%)	Mean test error (%)	Variance in mean test error (%)
1	44.75	45.31	55.25	1.14
2	49.75	54.06	50.25	7.49
3	56.93	59.06	43.07	2.21
4	57.18	60.93	42.82	4.73
5	58.68	60.00	41.32	1.63
6	58.62	58.75	41.38	1.98
7	58.93	60.93	41.07	3.54
8	60.18	63.75	39.82	8.47
9	61.00	64.37	39.00	6.90
10	60.87	65.62	39.13	13.49
11(all)	61.31	65.93	38.69	10.32

As seen above, any dimensionality reduction using PCA resulted in worse mean test classification accuracy. Though, it is interesting to note that PCA reduction into d < 8 resulted in much lower variance in mean test error.

Number of desired dimensions (d) using LDA	Mean test accuracy across 5 repartitions (%)	Best test accuracy across 5 repartitions (%)	Mean test error (%)	Variance in mean test error (%)
1	60.06	62.50	39.94	5.04
2	59.68	60.93	40.32	2.58
3	59.18	61.87	40.82	5.88
4	59.31	61.87	40.69	7.39
5	59.37	62.18	40.63	11.03

LDA can sometimes be better than PCA because it tries to best separate the various classes. PCA, on the otherhand, tries to find features of maximal variance. In this case though, LDA performed as poorly as PCA. Since the best test classification accuracy resulted from scaled data

with feature subset [0,1,2,4,5,6,8,9,10] and no dimensionality reduction, these preprocessing settings were used for the rest of the study on all packaged classifiers.

Now that the preprocessing settings are frozen, it is time to train and tune various classifiers. SVM has already been tuned, and the results are summarized below:

Repartition seed	Best average validation accuracy (k=3 folds)	Parameters used	Test accuracy (%)
1	62.31	{'C': 1, 'gamma': 'auto', 'kernel': 'rbf'}	58.75
2	63.95	{'C': 1, 'gamma': 1, 'kernel': 'rbf'}	60.93
3	63.40	{'C': 1, 'gamma': 1, 'kernel': 'rbf'}	65.00
4	61.84	{'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}	58.75
5	62.86	{'C': 1, 'gamma': 1, 'kernel': 'rbf'}	66.87
Mean accuracy			**62.06**
Mean error			**37.94**
Variance in error			**13.75**

The confusion matrix of the classifier yielding the highest testing accuracy (66.87%) is shown below. The parameters were tuned to{'C': 1, 'gamma': 1, 'kernel': 'rbf'}.

	0	1	2	3	4	5
0	0	0	2	0	0	0
1	0	0	4	7	0	0
2	0	0	100	33	3	0
3	0	0	29	95	4	0
4	0	0	3	18	19	0
5	0	0	0	3	0	0

C is the regularization parameter or the penalty parameter. It is used to tell the kernel how important misclassification is. A high value of C will tell SVM to classify as many training points correctly as possibly, at the cost of possibly overfitting. The low value of C can do the opposite, and possibly underfit the data. At C=1, the model resulted in the highest accuracy for this dataset. Gamma is a hyperparameter that determines the influence of points near the boundary of separation. A high value of gamma will tell the model to only consider data samples

near the boundary, whereas a low value of gamma will tell the model to consider many data samples even if they are further away from the boundary. For the current data, this value did not seem important, as this was different for each repartitioning of the data, with the majority being gamma=1. Lastly, the kernel used can determine the shape of the decision surface. RBF (radial basis function) is a kernel that transform the data into infinite dimensions, which yielded the best accuracy in for all repartitions. Linear kernel had a much longer training time resulted in worse classification accuracy. It would be interesting for future work to test other parameters such as using polynomial kernel for various degrees and comparing this with the infinite dimensional estimation of the RBF kernel.

Next, random forest classifier was attempted:

Repartition seed	Best average validation accuracy (k=3 folds)	Parameters used	Test accuracy (%)
1	67.16	{'max_features': 1, 'n_estimators': 500}	67.50
2	67.94	{'max_features': 1, 'n_estimators': 500}	64.37
3	66.77	{'max_features': 2, 'n_estimators': 200}	73.43
4	66.45	{'max_features': 1, 'n_estimators': 300}	66.25
5	66.37	{'max_features': 2, 'n_estimators': 300}	72.18
Mean accuracy			**68.75**
Mean error			**31.25**
Variance in error			**15.18**

The classifier with the best test accuracy was 73.43%. It's confusion matrix is shown below:

	0	1	2	3	4	5
0	0	0	2	0	0	0
1	0	0	7	4	0	0
2	0	0	107	28	1	0
3	0	0	18	106	4	0
4	0	0	1	17	22	0
5	0	0	0	0	3	0

The two most important parameters for tuning are the number of trees in the forest (n_estimators) and the number of features (max_features) to consider when looking for the best split. As the number of trees in the forest grows, the validation and test accuracies improve, at the cost of drastically increasing training time, especially at 1000+ trees. The optimal value for our dataset is between 300-500 trees, though with more training time, this value can tuned further. The maximum number of features to consider for the best split was best at m=1 or m=2 for this dataset. It would be interesting to consider other hyperparameters to tune with more time, such as max depth of each tree or minimum number of samples in each leaf node.

Next, K-NN was used for the task of classification:

Repartition seed	Best average validation accuracy (k=3 folds)	Parameters used	Test accuracy (%)
1	67.55	{'n_neighbors': 96, 'p': 1, 'weights': 'distance'}	67.18
2	67.63	{'n_neighbors': 61, 'p': 1, 'weights': 'distance'}	66.56
3	65.74	{'n_neighbors': 86, 'p': 1, 'weights': 'distance'}	69.68
4	66.92	{'n_neighbors': 116, 'p': 1, 'weights': 'distance'}	65.62
5	65.98	{'n_neighbors': 81, 'p': 1, 'weights': 'distance'}	71.87
Mean accuracy			**68.18**
Mean error			**31.82**
Variance in error			**6.51**

The classifier with the best testing accuracy has 71.87%. It's confusion matrix is shown below:

	0	1	2	3	4	5
0	0	0	2	0	0	0
1	0	0	7	4	0	0
2	0	0	103	32	1	0
3	0	0	19	104	5	0
4	0	0	0	17	23	0
5	0	0	0	3	0	0

The most important parameter to tune for K-NN is the value of k and the metric used to discern similarity between two points. In practice, k is commonly set to the square root of total number of training samples. But, due to the divergent property of k, it isn't always guaranteed that a higher k will lead to better performance. For this dataset, ranges of k from 1-500 were tested. The optimal k in this case was around ~80 as seen in the above table. The next important hyperparameter to consider is the similarity between points. Most commonly, Euclidean distance is used (p=2 in the Minkowski metric). But, for this dataset, the best performance was achieved using Manhattan distance (p=1 in the Minkowski metric) instead. It would be interesting to test the classifier using perhaps Mahalonobis distance or correlation as a similarity measure etc.

Next Bayes classifier using own code was employed:

Using the preprocessed data detailed in this study led to poor classification accuracy. To improve accuracy, data standardization was not done for this classifier. Below are the results of Bayes classification:

Repartition seed	Test accuracy with no preprocessing (%)	Test accuracy with standardized scaled data (%)
1	43.12	35.62
2	45	32.18
3	47.18	22.81
4	47.18	24.68
5	50.31	35.93
Mean accuracy	**46.40**	**30.24**
Mean error	**53.60**	**69.76**
Variance in error	**9.53**	**37.80**

The classifier with the best test accuracy was 50.31%. Its confusion matrix is shown below:

	0	1	2	3	4	5
0	0	2	0	0	0	0
1	0	2	3	2	4	0
2	0	14	75	38	8	1
3	0	11	20	56	40	1
4	0	0	1	9	28	2
5	0	0	0	2	1	0

To test whether features are truly dependent, it was now assumed that the individual samples we i.i.d. Same personal Bayesian model was used, but instead with diagonal covariance matrix to represent feature independence. Below are its results:

Repartition seed	Test accuracy with no preprocessing (%)	Test accuracy with standardized scaled data (%)
1	33.12	33.12
2	28.75	28.75
3	35.62	35.62
4	39.68	39.68
5	35.31	35.31
Mean accuracy	**34.50**	**34.50**
Mean error	**65.50**	**65.50**
Variance in error	**15.92**	**15.92**

The best classification accuracy of 39.68% resulted during the 4th repartitioning of the data. It's confusion matrix is shown below:

	0	1	2	3	4	5
0	0	2	0	0	0	0
1	6	2	1	1	1	0
2	23	8	74	13	14	4
3	15	7	30	32	26	18
4	1	2	1	2	19	15
5	0	0	0	0	3	0

Interestingly, data processing had absolutely no affect on the outcome of this classifier with assumption of feature independence. This is likely because the scale of each feature is irrelevant when the covariance matrix simply assumes feature independence, as their relative distance from the mean sample is the only factor in determining the likelihood of a data samples being assigned to a certain class

To test non-parametric estimations (besides Random Forests and K-NN), personal code was used to test Parzen window estimation. The major hyperparameter to tune for this model is the window width (h), so the focus was on this. A Gaussian kernel was used as the window function. Due to training time, just scaled data was tested since it resulted in faster computation.

	h=0.1	h=1	h=10	h=100
Repartition seed	Test accuracy (%)			
1	6.87	38.12	41.87	41.87
2	13.43	24.68	25.00	25.00
3	12.81	44.37	45.62	45.62
4	10.31	34.37	36.25	36.25
5	10.62	34.06	39.37	39.37
Mean accuracy	**10.80**	**35.12**	**37.62**	**37.62**
Mean error	**89.20**	**64.88**	**62.38**	**62.38**
Variance in error	**6.66**	**51.31**	**61.56**	**61.56**

The best classification accuracy, of 45.62% resulted from setting the window width to at least 10, and using a Gaussian kernel. It's confusion matrix is shown below.

	0	1	2	3	4	5
0	0	0	1	1	0	0
1	0	0	0	9	2	0
2	0	0	42	85	9	0
3	0	0	10	75	43	0
4	0	0	2	9	29	0
5	0	0	1	0	2	0

As seen from the table above, the accuracies were very poor for a low window width. The jaggedness of the density is likely the cause of this, especially in a high-dimensional problem such as this. After h=10, the label predictions did not change since the window width as sufficiently large enough to get a smooth density, and therefore better adapted to the test data.

Partly the reason why Bayesian classification and Parzen window estimation performed poorly is because of unbalanced class distribution of our training set. The core of Bayesian classification is

the maximum a posterior (MAP) principle. The posterior probability is a combination of probability density and the priors. If the priors were adjusted based on the skewed class distribution, these classifiers would perform better.

Summary and Conclusions:

As stated before, the goal of the study was to gain insight into the data set. Data normalization techniques showed that scaling the data to zero mean and unit variance resulted in the best performance for the packaged classifiers (SVM, Random Forests, and K-NN). Though for personal coded Bayesian classifier, the performance was worse with data normalization. From the correlation matrix and feature selection, it was seen that residual sugar and density were the most useless features and so were discarded from evaluation of the packaged classifiers. The two most important features found using SFS were alcohol and sulphates. Dimensional reduction techniques such as PCA and LDA resulted in worse classification accuracy and therefore were avoided, though at n=8 dimensions and lower, the performance decreased only slightly whereas the variation in mean error over several repartitions of the data decreased as well.

Below is a summary of findings for each classifier:

Classier:	Parameters used	Best test accuracy (%)	Mean test accuracy (%)	Mean test error (%)	Variance in mean error
SVM	{'C': 1, 'gamma': 1, 'kernel': 'rbf'}	66.87	62.06	37.94	13.75
Random Forests	{'max_features': 2, 'n_estimators': 200}	73.43	68.75	31.25	15.18
K-NN	{'n_neighbors': 81, 'p': 1, 'weights': 'distance'}	71.87	68.18	31.82	6.51
Bayesian	None (no preprocessing or priors)	50.31	46.40	53.60	9.53
Bayesian (i.i.d)	None (just scaled data)	39.68	34.50	65.50	15.92
Parzen window	h=10, kernel=Gaussian	45.62	37.62	62.38	61.56

Overall, the best classification accuracy resulted from the packaged classifiers, especially Random Forests and K-NN. While Random Forests classifier performed the best in accuracy, it suffered from high variation in mean error over several repartitions of the data. K-NN had the least variance in mean error while still achieving high test accuracy, being the most stable classifier for the dataset. Bayesian and Parzen classifiers performed poorly on the dataset, likely due to the imbalanced distribution of classes and lack of utilization of priors. Parzen window was the most unstable classifier with the highest variance in mean error. It was also heavily reliant to the window size and converged to a Gaussian at $h \geq 10$.

Limitations of the study arise from preprocessing techniques. A sequential approach was taken to decide on preprocessing techniques such as method of normalization, feature selection, and feature extraction. A thorough analysis would use an exhaustive method to find the ideal preprocessing technique for each classifier, instead of just SVM. It is evident that no universal combination of preprocessing techniques will work well with every classifier, as an extension of the No Free Lunch Theorem. Likewise, there is no set of best features for a given problem as it is dependent on the classification task and classifier used (Ugly Duckling Theorem).

While grid search is an incredibly useful tool in python, there is still the element of deciding which hyperparameters to tune. There is a clear tradeoff between exhaustive search of every increment of every hyperparameter and training time using grid search. With more time, a better classification accuracy and more insight into the stability of the classifier could be gained with more hyperparameter tuning and increasing the number of folds in cross validation. Perhaps, a random search for a fixed amount of iterations over a wider range of hyperparameters may have yielded a better classification accuracy.

Imbalance of class distribution is a limitation to training the various classifiers. It might be useful to split the datasets into “good” or “bad” wine using some quality threshold. This simplifies the task of classification to a binary one and may have improved performance using logical classifiers like Logistic Regression. For the sake of keeping the study complex, this was not done, but can be a prospect for future studies.

While predicting red wine quality based on 11 attributes was educational, it would be interesting see how important these attributes are for white wine. More importantly, how are red and white wine different? Can one differentiate between red and white wine based on the 11 attributes? These are interesting questions that are potential goals for future studies.

Code used:

Grid search for prepackaged classifiers:

```
val_scores = []
test_scores = []
test_std_dev = []
parameter_list = []
confusionmat_list = []
subsets_list = []
for f in range(1,6): #for every seed or repartitioning of the dataset
    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 10+f, stratify=Y)
    #min_max_scaler = MinMaxScaler()
    #x_train_minmax = min_max_scaler.fit_transform(x_train)
    #x_test_minmax = min_max_scaler.transform(x_test)

    #normalizer = preprocessing.Normalizer().fit(x_train)
    #x_train_normalized = normalizer.transform(x_train)
    #x_test_normalized = normalizer.transform(x_test)


    scaler = preprocessing.StandardScaler().fit(x_train)
    x_train_scaled = scaler.transform(x_train)
    x_test_scaled = scaler.transform(x_test)

#    clf = GridSearchCV(svm.SVC(), {
#            'C': [0.001, 0.01, 0.1, 1, 10, 100],
#            'kernel': ['rbf', 'linear'],
#            'gamma': ['auto', 'scale', 0.001, 0.01, 0.1, 1, 10, 100]
#            }, cv=3, return_train_score=False)
    feat_cols = [0,1,2,4,5,6,8,9,10]
#    clf = GridSearchCV(tree.DecisionTreeClassifier(), {
#            'max_depth': [5, 10, 50, 100, 200, 300, 400, 500],
#            'max_features': [1,2,3,4,5,6,7,8,9],
#            }, cv=3, return_train_score=False)
#    clf = GridSearchCV(RandomForestClassifier(), {
#            'n_estimators': [5, 10, 50, 100, 200, 300, 400, 500],
#            'max_features': [1,2,3,4,5,6,7,8,9],
#            }, cv=3, return_train_score=False)
    clf = GridSearchCV(KNeighborsClassifier(), {
            'n_neighbors': list(range(1,500,5)),
            'weights': ['uniform','distance'],
            'p': [1,2]
            }, cv=3, return_train_score=False)

    clf.fit(x_train_scaled[:,feat_cols], y_train)
    #df = pd.DataFrame(clf.cv_results_)
    y_pred = clf.predict(x_test_scaled[:,feat_cols]) #use the best average validation score across 3 folds (their tuned parameters),
    test_accuracy = accuracy_score(y_test, y_pred)*100
    confusionmat = confusion_matrix(y_test, y_pred)
    val_scores.append(clf.best_score_*100)
    test_scores.append(test_accuracy)
    best_params = clf.best_params_
    parameter_list.append(best_params)
    confusionmat_list.append(confusionmat)
#mean_val_score = mean(val_scores)

mean_test_score = mean(test_scores)
test_var = statistics.variance(test_scores)

#results = [mean_val_score, mean_test_score, test_std]
```

SFS:

```
#n_comp_results = []
#for n in range(1,6):
val_scores = []
test_scores = []
test_std_dev = []
parameter_list = []
confusionmat_list = []
feature_list = []

for f in range(1,6): #for every seed or repartitioning of the dataset
    print("-------------f is---------------:", f)
    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 10+f, stratif
#
#
    scaler = preprocessing.StandardScaler().fit(x_train)
    x_train_scaled = scaler.transform(x_train)
    x_test_scaled = scaler.transform(x_test)
#
#
    clf = svm.SVC(C=10, gamma='scale', kernel='rbf');
    sfs1 = sfs(clf, k_features=10, forward=False, floating=False, verbose=2, scoring='accuracy', cv=3)
    sfs1 = sfs1.fit(x_train_scaled, y_train)
    feat_cols = list(sfs1.k_feature_idx_)
    feature_list.append(feat_cols)

#    feat_cols = [0,1,2,4,5,6,8,9,10]
#    clf.fit(x_train_scaled[:,feat_cols], y_train)
#    y_pred = clf.predict(x_test_scaled[:,feat_cols]) #use the best average validation score across 3 folds
#    test_accuracy = accuracy_score(y_test, y_pred)*100
#    test_scores.append(test_accuracy)
#
#
#mean_test_score = mean(test_scores)
#test_var = statistics.variance(test_scores)
```

PCA and LDA:

```
#n_comp_results = []
#for n in range(1,6):
val_scores = []
test_scores = []
test_std_dev = []
parameter_list = []
confusionmat_list = []
feature_list = []

for f in range(1,6): #for every seed or repartitioning of the dataset
    print("-------------f is---------------:", f)
    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 10+f
#
#
    scaler = preprocessing.StandardScaler().fit(x_train)
    x_train_scaled = scaler.transform(x_train)
    x_test_scaled = scaler.transform(x_test)
#
#



    #pca = PCA(n_components=0.95, svd_solver='full') #95% variance captured by 9 dimmension, best
    #pca = PCA(n_components=11)
    #pca.fit(x_train_scaled)
    #x_train_scaled_pca = pca.transform(x_train_scaled)
    #x_test_scaled_pca = pca.transform(x_test_scaled)

    lda = LinearDiscriminantAnalysis(n_components=5)
    lda.fit(x_train_scaled, y_train)
    x_train_scaled_lda = lda.transform(x_train_scaled)
    x_test_scaled_lda = lda.transform(x_test_scaled)

    clf = svm.SVC(C=10, gamma='scale', kernel='rbf');
    clf.fit(x_train_scaled_lda, y_train)
    y_pred = clf.predict(x_test_scaled_lda) #use the best average validation score across 3 folds
    test_accuracy = accuracy_score(y_test, y_pred)*100
    test_scores.append(test_accuracy)


mean_test_score = mean(test_scores)
test_var = statistics.variance(test_scores)
```

Bayes classification:

```
idx = y_train[:] == 3
mu_3 = np.mean(x_train.values[idx], axis=0)
cov_3 = np.cov(x_train.values[idx], bias=True, rowvar=False)
cov_3 = np.diag(np.diag(cov_3))

idx = y_train[:] == 4
mu_4 = np.mean(x_train.values[idx], axis=0)
cov_4 = np.cov(x_train.values[idx], bias=True, rowvar=False)
cov_4 = np.diag(np.diag(cov_4))


idx = y_train[:] == 5
mu_5 = np.mean(x_train.values[idx], axis=0)
cov_5 = np.cov(x_train.values[idx], bias=True, rowvar=False)
cov_5 = np.diag(np.diag(cov_5))


idx = y_train[:] == 6
mu_6 = np.mean(x_train.values[idx], axis=0)
cov_6 = np.cov(x_train.values[idx], bias='True', rowvar=False)
cov_6 = np.diag(np.diag(cov_6))


idx = y_train[:] == 7
mu_7 = np.mean(x_train.values[idx], axis=0)
cov_7 = np.cov(x_train.values[idx], bias='True', rowvar=False)
cov_7 = np.diag(np.diag(cov_7))


idx = y_train[:] == 8
mu_8 = np.mean(x_train.values[idx], axis=0)
cov_8 = np.cov(x_train.values[idx], bias='True', rowvar=False)
cov_8 = np.diag(np.diag(cov_8))


y_pred = np.zeros(shape=(x_test.shape[0],1))
for i in range(np.shape(x_test)[0]):
    p3 = multivariate_normal.pdf(x_test.values[i], mean=mu_3, cov=cov_3, allow_singular=True)
    p4 = multivariate_normal.pdf(x_test.values[i], mean=mu_4, cov=cov_4, allow_singular=True)
    p5 = multivariate_normal.pdf(x_test.values[i], mean=mu_5, cov=cov_5, allow_singular=True)
    p6 = multivariate_normal.pdf(x_test.values[i], mean=mu_6, cov=cov_6, allow_singular=True)
    p7 = multivariate_normal.pdf(x_test.values[i], mean=mu_7, cov=cov_7, allow_singular=True)
    p8 = multivariate_normal.pdf(x_test.values[i], mean=mu_8, cov=cov_8, allow_singular=True)
    pred_label = np.argmax([p3, p4, p5, p6, p7, p8])
    pred_label += 3
    y_pred[i] = pred_label

test_accuracy = accuracy_score(y_test, y_pred)*100
confusionmat = confusion_matrix(y_test, y_pred)
```

Parzen window estimation:

```
scaler = preprocessing.StandardScaler().fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)
h_n = 10 #window width
parzen_prediction = np.zeros(shape=(x_test.shape[0],1)) #label prediction for test data

for i in range(np.shape(x_test_scaled)[0]): #for every test sample
    pn_3 = 0
    pn_4 = 0
    pn_5 = 0
    pn_6 = 0
    pn_7 = 9
    pn_8 = 0
    for j in range(np.shape(x_train_scaled)[0]): #for every training sample
        if y_train.values[j] == 3:
            pn_3 = pn_3 + (1/h_n)* multivariate_normal.pdf((x_test_scaled[i]- x_train_scaled[j])/h_n,mean=[0,0,0,0,0,0,0,0,0,0,0], cov=np.identity(11))
        elif y_train.values[j] == 4:
            pn_4 = pn_4 + (1/h_n)* multivariate_normal.pdf((x_test_scaled[i]- x_train_scaled[j])/h_n,mean=[0,0,0,0,0,0,0,0,0,0,0], cov=np.identity(11))
        elif y_train.values[j] == 5:
            pn_5 = pn_4 + (1/h_n)* multivariate_normal.pdf((x_test_scaled[i]- x_train_scaled[j])/h_n,mean=[0,0,0,0,0,0,0,0,0,0,0], cov=np.identity(11))
        elif y_train.values[j] == 6:
            pn_6 = pn_4 + (1/h_n)* multivariate_normal.pdf((x_test_scaled[i]- x_train_scaled[j])/h_n,mean=[0,0,0,0,0,0,0,0,0,0,0], cov=np.identity(11))
        elif y_train.values[j] == 7:
            pn_7 = pn_4 + (1/h_n)* multivariate_normal.pdf((x_test_scaled[i]- x_train_scaled[j])/h_n,mean=[0,0,0,0,0,0,0,0,0,0,0], cov=np.identity(11))
        elif y_train.values[j] == 8:
            pn_8 = pn_4 + (1/h_n)* multivariate_normal.pdf((x_test_scaled[i]- x_train_scaled[j])/h_n,mean=[0,0,0,0,0,0,0,0,0,0,0], cov=np.identity(11))
    pn_3 = pn_3/h_n
    pn_4 = pn_4/h_n
    pn_5 = pn_5/h_n
    pn_6 = pn_6/h_n
    pn_7 = pn_7/h_n
    pn_8 = pn_8/h_n
    pred_label = np.argmax([pn_3, pn_4, pn_5, pn_6, pn_7, pn_8])
    pred_label += 3
    parzen_prediction[i] = pred_label
test_accuracy = accuracy_score(y_test, parzen_prediction)*100
confusionmat = confusion_matrix(y_test, parzen_prediction)
```