CSE 5243 Introduction to Data Mining

Lab 3: Comparing Classifiers
(Executive Summary)

Bharat Suri (suri.40@osu.edu)
Nithin Senthil Kumar (senthilkumar.16@osu.edu)
Taught By: Michael Burkhardt
Date: 10/18/2019

1. Introduction

In this lab, six different Scikit-Learn classifiers were implemented and tested to predict the grape variety of wines based on 13 feature attributes. The lab used the "Wine" dataset from the UCI machine learning repository. The data collected stems from a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars, as stated in the dataset description by C. Blake. The analysis determined the quantities of 13 constituents found in each of the three types of wines. In total, there were 59 instances belonging to class 1, 71 instances belonging to class 2, 48 instances belonging to class 3, where each class represented a different grape variety.

All 13 attributes are continuous and are listed below:

1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10)Color intensity
11) Hue
12) OD280/OD315 of diluted wines
13) Proline

Several libraries were used to visualize, process and predict the data. The key libraries used for visualization are seaborn and matplotlib. Seaborn is based on matplotlib but provides a higher level of abstraction allowing one to build attractive, informative graphs and implement complex visualizations. Numpy and pandas libraries are used for fast data manipulation, computation and processing tasks. The

robust python machine learning library, scikit-learn, provides a range of supervised learning algorithms, among other algorithms, which were used in this lab.

The 6 classifiers that were built and investigated are:

1) Logistic Regression (sklearn.linear_model.LogisticRegression)
2) k-Nearest Neighbors (sklearn.neighbors.KNeighborsClassifier)
3) Decision Tree (sklearn.tree.DecisionTreeClassifier)
4) Naïve Bayes (sklearn.naive_bayes.GaussianNB)
5) Support Vector Machine (sklearn.svm.LinearSVC)
6) Ensemble Classifier (sklearn.ensemble.RandomForestClassifier)

For each classifier, a default model was fitted using the training set. Predictions were made on the training set and test set which were then evaluated using metrics such as accuracy and F-1 score.

2. Exploratory Data Analysis

A number of methods were used to understand and explore the characteristics of the records, attributes and the relationships between them. The outcome of this preliminary analysis aided the data preprocessing step. This step explores summary statistics, data quality issues, missing values and attribute correlations.

2.1 Summary Statistics

The data is described to obtain the key statistical attributes that can convey important information about features and the spread of data. It is evident that the range of the attributes vary considerably between the features. Therefore, a standard scaler needs to be applied to normalize the data before comparison.

	class	alcohol	malic_acid	ash	alcalinity	magnesium	phenols	flavanoids	nonflav_phenols	proanth	color	hue
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	1.938202	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	5.058090	0.957449
std	0.775035	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.572359	2.318286	0.228572
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.280000	0.480000
25%	1.000000	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.250000	3.220000	0.782500
50%	2.000000	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.555000	4.690000	0.965000
75%	3.000000	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	6.200000	1.120000
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000

Table 1. Basic Statistics

From Table 1, it was observed that the attributes ‘magnesium’, ‘color’, ‘alkalinity’ and ‘malic_acid’ have relatively higher variance. This indicates that these attributes hold the potential to be more informative and significant.

2.2 Data Quality Issues

NULL values and duplicates

The names file states that there are no missing values in the dataset. This was corroborated by checking for NULL values in the data set (Figure 2.1). None were present. No duplicate entries were found either.

```
RangeIndex: 178 entries, 0 to 177
Data columns (total 13 columns):
alcohol             178 non-null float64
malic_acid          178 non-null float64
ash                 178 non-null float64
alcalinity          178 non-null float64
magnesium           178 non-null int64
phenols             178 non-null float64
flavanoids          178 non-null float64
nonflav_phenols     178 non-null float64
proanth             178 non-null float64
color               178 non-null float64
hue                 178 non-null float64
dil                 178 non-null float64
proline             178 non-null int64
```

Figure 2.1 Attribute information

Outlier detection with BoxPlots

One of the most common methods to detect outliers is to use boxplots. In figure 2.2 below, boxplots are plotted for each of the attributes to visualize the spread of data and the existence of outliers. Once again, as inferred from the summary statistics, it can be clearly seen that the range of data is widespread amongst all the feature attributes. There were no prominent outliers, however, scaling was done to make the plot more readable.

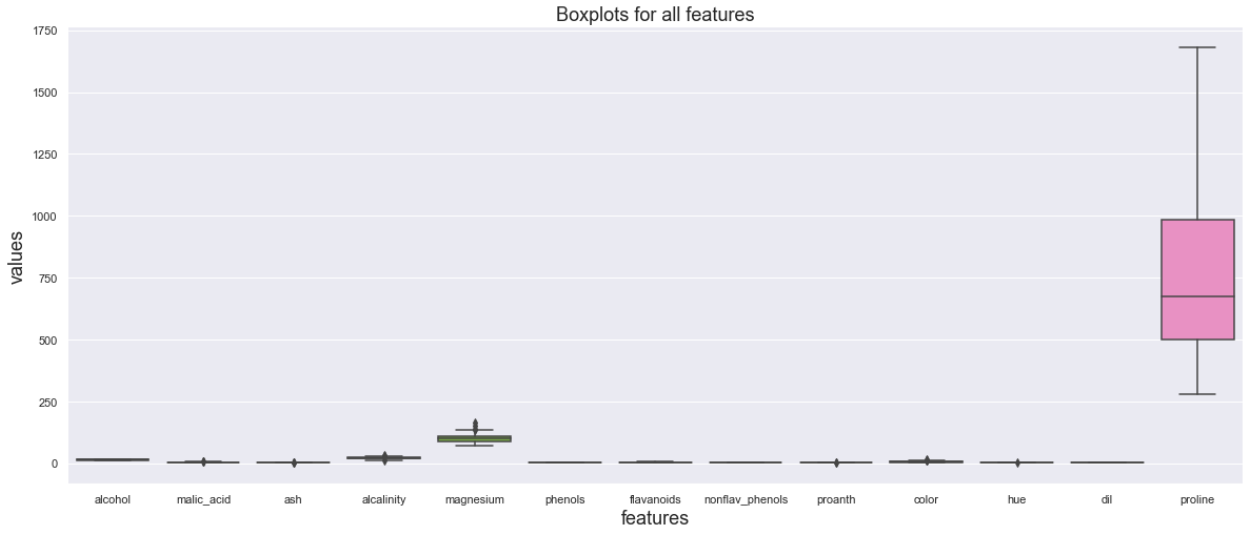

Figure 2.2 Boxplot for outlier detection

Correlation matrix as a heatmap

Since the box plots indicated the existence of a widely varying range between different attributes, it is necessary to normalize or scale the attributes to all have values within similar ranges. This allows the correlation coefficients to be more meaningful. The heatmap in Figure 2.3 serves as an interesting visualization tool since it is easy to observe colored areas and use this to identify highly correlated attributes and attributes of minimal to no correlation.

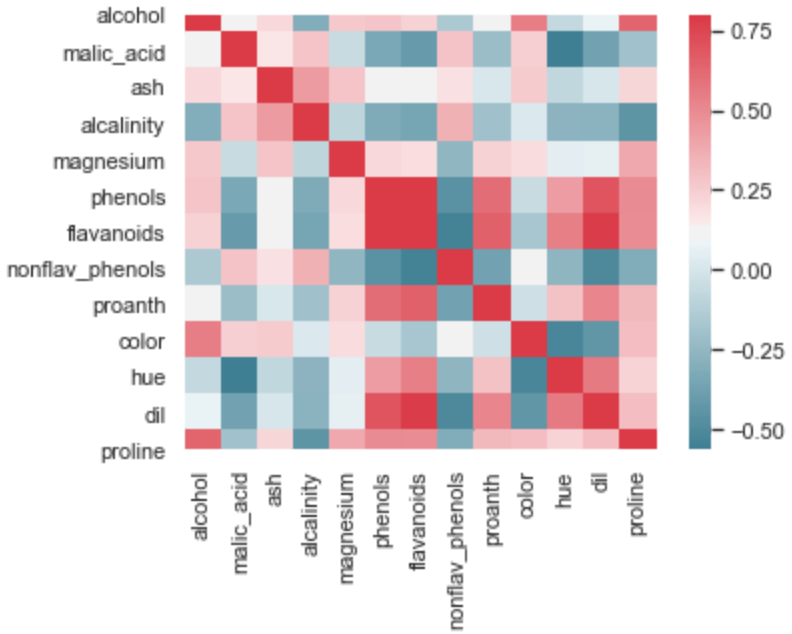

Figure 2.3 Heatmap for correlation matrix

From the heatmap, the following interesting relationships between the features were observed:

1) phenols and flavanoids (Positive correlation)
2) color and hue (Negative Correlation)
3) dil and flavanoids (Positive Correlation)

The use of a scatter plot to plot the relationship between these pairs of attributes and the class they belong to may provide insightful information on how separate the classes are from each other. Figure 2.4 visualizes the relationships between the pairs of features obtained from a pairplot computed on the dataset.

Though the distinction between the three classes are soft for all four plots, we can see that class 1 is well separated from class 2 and class 3 in all four of the plots. However, the boundary between class two and three is not clear from the pairs of attributes above.

Pairplots

To see if any single pair of attributes are able to clearly distinguish between the three classes, we can use the pairplot function in the SeaBorn library. This is a powerful illustration which outputs a grid of plots that shows the relationship between every possible pair of attributes and the three classes. The figures below are the most interesting from the pairplots and show the clearest separation between the three classes.

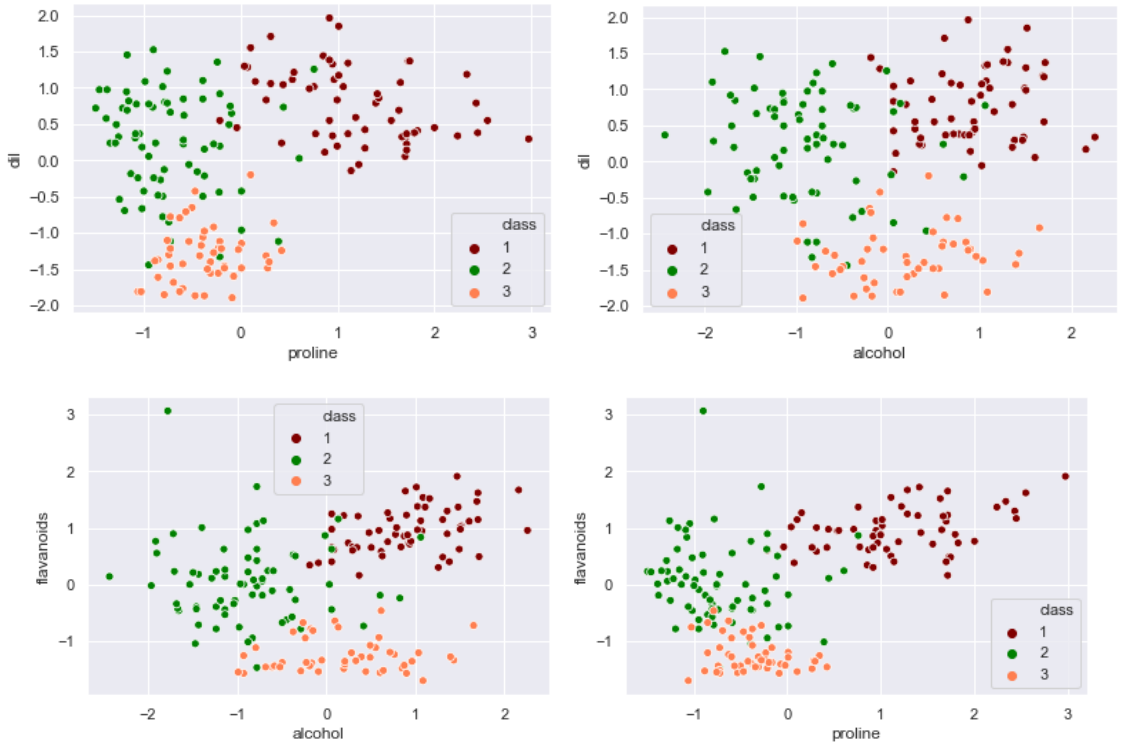

Figure 2.4 Interesting scatterplots from the pairplot

While pair plots can only illustrate the relation amongst two attributes taken at a time, the classifiers that will be studied in section 4 will take into consideration the contribution of all the attribute values in the prediction of class.

3. Data preprocessing

3.1 Standardization

Standardization of the values of continuous attributes is an important step in data preprocessing. The need for standardization was identified by analyzing the box plots in Figure 2.2 as well as the values of mean of features in the dataset. More importantly, without any domain knowledge, it is difficult to comment on the units of each of those features. Therefore, it was decided that the dataset will be scaled before proceeding with the next step. Scikit-learn's StandardScaler function was used to accomplish the task of standardization. According to Scikit-learn's documentation, Standardization refers to the "process of standardizing features by removing the mean and scaling to unit variance."

The standard score of a sample x is calculated as:

$$z = (x - u) / s$$

Where ***u*** is the mean of the training samples, and ***s*** is the standard deviation of the training samples. The Boxplots for outlier detection after standardization are shown in Figure 3.1. From the figure, it can be inferred that no prominent outliers were found.

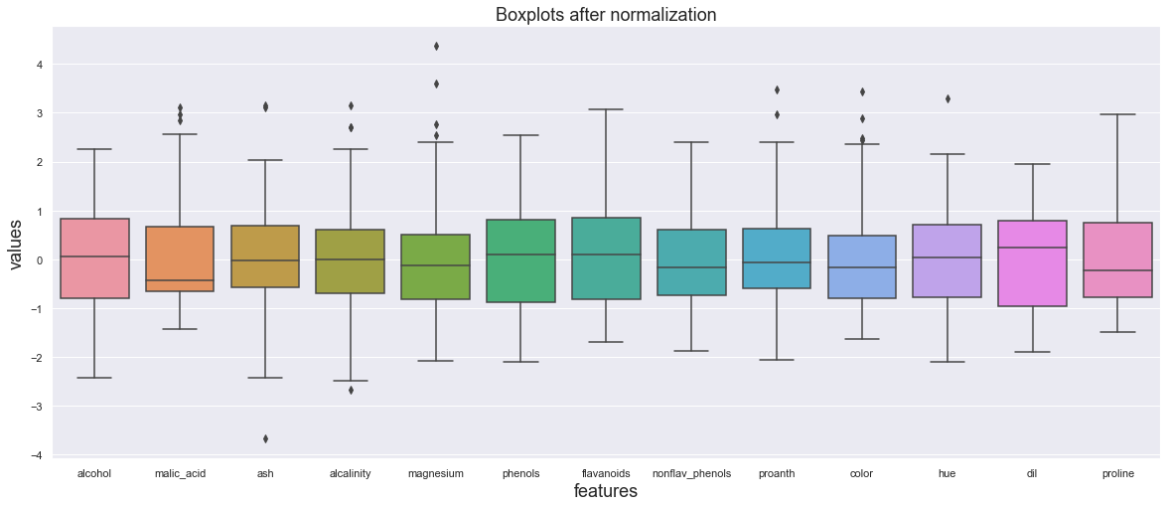

Figure 3.1 Boxplots after normalization

Feature selection and creation

It was also observed through a multi-tree classifier that various features were being computed as the important features. This was done by checking the feature importance after fitting the training data. Due to the varied output, no particular feature subset was selected as it may require more domain knowledge to make such a decision. Same was the case for feature creation so the only preprocessing that was done was the scaling of the dataset using StandardScaler.

3.2 Test-Train split

Here, the data is split into training and testing sets using the sklearn.model_selection train_test_split splitter method. The ratio of the split was 1:1, which means 50% of the records were used for training and the remaining 50% were used for testing. After splitting, it was found that the training data had 29 instances of class 1, 38 instances of class 2 and 22 instances of class 3. The test data had 30 instances of class 1, 33 instances of class 2 and 26 instances of class 3.

4. Training of classifiers

There are a total of 6 classifiers whose performances were evaluated. These classifiers are:

1. Support Vector Machine
2. k-Nearest Neighbors Classifier
3. Logistic Regression
4. Naive Bayes Classifier

5. Decision Tree Classifier
6. Ensemble Classifier

To train the classifiers, the model is fit using the training set only. Then, the predictions were made using the training set data and the performance metrics for the training set were recorded. These metrics were accuracy, recall, precision, weighted average, and F-1 score. Furthermore, the default hyperparameters were used for each classifier. The ***classification_report*** function from the sklearn library was used to generate score report on the different performance metrics mentioned above.

4.1 Support Vector Machine (SVM)

SVM is a discriminative classifier that separates classes based on a decision boundary characterized by a hyperplane. The Linear Support Vector Classification function was used which is equipped with a linear kernel. Figure 4.1 illustrates the difference between using a linear kernel and one that is not linear. The maximum number of iterations that are run by default is 100 and the loss function is by default the square of the hinge loss.

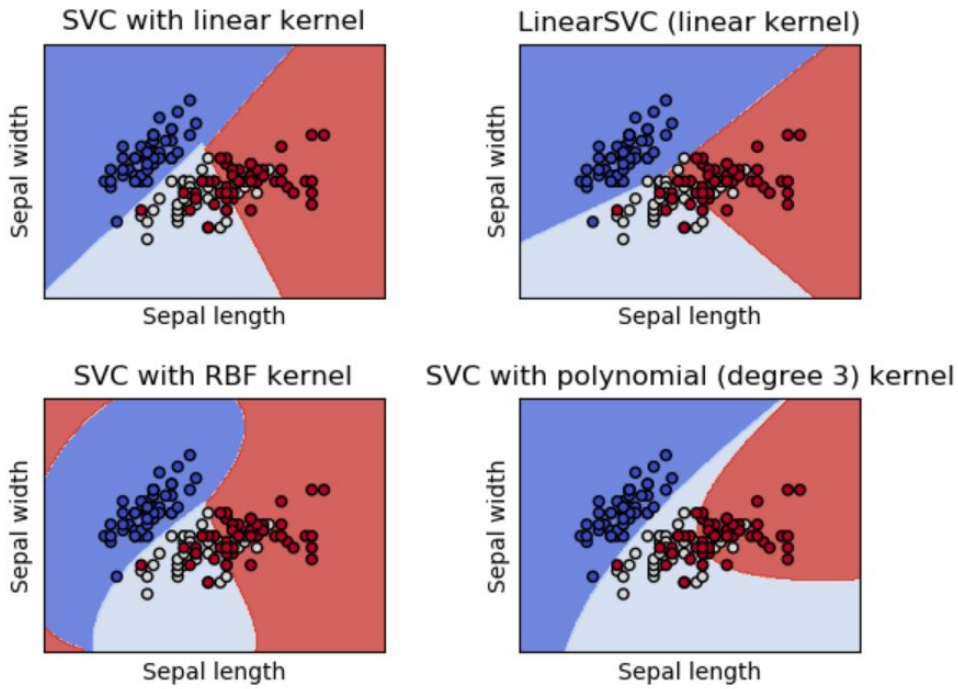

Figure 4.1 General SVC examples[6]

When making predictions on the training data, the model achieved a score of 1.0, that is, the model was 100% accurate in identifying the records from the training data. The score function of LinearSVC returns the mean accuracy on the given test data and labels. The other important performance metrics such as f1-score, precision and recall are also given below in Figure 4.2

	precision	recall	f1-score	support
1	1.00	1.00	1.00	29
2	1.00	1.00	1.00	38
3	1.00	1.00	1.00	22
accuracy			1.00	89
macro avg	1.00	1.00	1.00	89
weighted avg	1.00	1.00	1.00	89

Figure 4.2 Training score of SVC

4.2 k-Nearest Neighbors

The classification report for the KNN classifier showed an accuracy of **98%** on the training set. Number of neighbors to use by default for k-nearest neighbors is 5. By default all points in each neighborhood are weighted equally. Furthermore, Euclidean distance is considered for Minkowski metric. The report in Fig. 4.3 also shows that the errors made by the classifier are on class 1 and class 2. For class 1, the Sensitivity is 1.0 that is there are no false negatives, and for class 2, the Positive Predictive Value is 1.0 thus there are no false positives. The reason for this could be the higher number of instances of class 1 and 2 than class 3 in training.

	precision	recall	f1-score	support
1	0.94	1.00	0.97	29
2	1.00	0.95	0.97	38
3	1.00	1.00	1.00	22
accuracy			0.98	89
macro avg	0.98	0.98	0.98	89
weighted avg	0.98	0.98	0.98	89

Figure 4.3 Training score of K-Nearest Neighbors

4.3 Logistic Regression

The Logistic Regression function allows for a multiclass classifier. Out of the default parameters that the function takes, there is a need to choose a different value for the parameter “solver”. This parameter defines the algorithm to use in the optimization problem[1]. LBFGS is chosen which is the Limited Memory Broyden–Fletcher–Goldfarb–Shanno algorithm. It is a popular algorithm for parameter estimation in machine learning. The algorithm's target problem is to minimize a function f(x) over unconstrained values of the real-vector x where f is a differentiable scalar function[2]. L-BFGS can be thought of as a way to find a (local) minimum of an objective function, making use of objective function values and the gradient of the objective function

1) For small datasets, ‘liblinear’ is a good choice, whereas ‘sag’ and ‘saga’ are faster for large ones.
2) For multiclass problems, only ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ handle multinomial loss; ‘liblinear’ is limited to one-versus-rest schemes.
3) ‘newton-cg’, ‘lbfgs’, ‘sag’ and ‘saga’ handle L2 or no penalty

4) ‘liblinear’ and ‘saga’ also handle L1 penalty
5) ‘liblinear’ does not handle no penalty

For the parameter multiclass, the value ‘multinomial’ is chosen so that the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. A maximum of 100 iterations are taken for the solvers to converge. The score function returns the mean accuracy on the given test data and labels. We get a score of 1 when the model is fit to and tested on the training data. Since the score is a measure of the mean accuracy, the accuracy of the model below on the training data is **100%**. The f1-score, precision and recall are in Figure 4.4

	precision	recall	f1-score	support
1	1.00	1.00	1.00	29
2	1.00	1.00	1.00	38
3	1.00	1.00	1.00	22
accuracy			1.00	89
macro avg	1.00	1.00	1.00	89
weighted avg	1.00	1.00	1.00	89

Figure 4.4 Training score of Logistic Regression

4.4 Naive-Bayes

GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

The GaussianNB function is used with its default parameters. The score once again refers to the mean accuracy of the classifier. On the training data, an accuracy score of 0.99 was obtained. The model has an F-1 score of 0.99 and 0.98 for classes 1 and 2 respectively as seen from Fig. 4.5

	precision	recall	f1-score	support
1	1.00	1.00	1.00	29
2	1.00	0.97	0.99	38
3	0.96	1.00	0.98	22
accuracy			0.99	89
macro avg	0.99	0.99	0.99	89
weighted avg	0.99	0.99	0.99	89

Figure 4.5 Training score of Naive-Bayes

4.5 Decision Tree

Decision Trees (DT) are non-parametric supervised learning methods used for classification and regression. The goal is to "create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features", as stated in the documentation of Scikit-learn. There are many advantages to using decision trees. They are simple to interpret and can be visualized easily. They require little data preparation and can work on both numerical and categorical attributes. However, decision trees come at a cost where in some cases, they do not generalize well. Decision trees with a greater depth may have been over fitted to the training data. By default, the decision tree classifier uses the gini index to measure the quality of a split. The best split is chosen at each node. The maximum depth of the tree is node specified which means that the trees are allowed to grow until there are only leaf nodes which are all pure. The minimum number of samples required to split an internal node is 2 by default. The score method returns the mean accuracy on the given test data and labels. The mean accuracy of the decision tree classifier on the training data is 10 % and the classifier secures a perfect score on the other performance metrics.

	precision	recall	f1-score	support
1	1.00	1.00	1.00	29
2	1.00	1.00	1.00	38
3	1.00	1.00	1.00	22
accuracy			1.00	89
macro avg	1.00	1.00	1.00	89
weighted avg	1.00	1.00	1.00	89

Figure 4.6 Training score of Decision Tree

4.6 Ensemble Classifier

The Random Forest Classifier is used as an example of an ensemble classifier. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default)[5]. The number of trees in the forest are 10 by default and the function to measure the quality of a split is Gini. All classes are weighted equally and have weight = 1. The score method returns the mean accuracy on the given test data and labels. A mean accuracy of 100% is obtained for the training data and a score of 1 is obtained for all other performance metrics shown in Figure 4.7

	precision	recall	f1-score	support
1	1.00	1.00	1.00	29
2	1.00	1.00	1.00	38
3	1.00	1.00	1.00	22
accuracy			1.00	89
macro avg	1.00	1.00	1.00	89
weighted avg	1.00	1.00	1.00	89

Figure 4.7 Training score of Random Forest Classifier

4.7 Choosing a classifier

Since the size of the dataset is small, and all classifiers had similar performance metrics, there is not much room to choose a classifier based on their performance of predictions on the training data. However, considering the characteristics of data set and the advantages and disadvantages associated with each classifier, one can be chosen as a preferred classifier based on how their merits stack up.

Occam's Razor implies that the simpler explanation(classifier) is the better choice when the accuracies are comparable for all the classifiers. However, k-Nearest Neighbors which is simple in its approach was outperformed on the training data by the other classifiers. Decision trees and random forests were not chosen since they can suffer from overfitting in higher dimensional space which could be the reason why they all performed exceptionally well when it came to predicting the data that the classifier was trained on. Between logistic regression, support vector machines and Naive Bayes, support vector machine was chosen as the preferred classifier because it is more robust than Logistic Regression and is simple to use for such a small dataset.

Support vector machines are known to find globally optimal solutions. Given that the cost functions and the kernel are provided, they are able to balance model complexity and training error resulting in better generalization performance. They are redundant to irrelevant attributes. Since there are no missing fields in the data set, their inability to handle missing values with ease is not of concern. Support vector machines are also robust to noise points. Since we have chosen a linear kernel and the dataset is small in size, the computational complexity for building the model is also insignificant. Finally, SVMs have the added advantage of fast classification and prediction.

From Figure 4.2, it can be observed that support vector machines had a score of 1 for all performance metrics measured. This means that all the training data were correctly predicted. The metrics measured were precision, recall, accuracy and f1 score. The true test of its performance however, will stem from its performance on the test data set and its ability to have a better generalization error. This will be observed in Section 5.

5. Evaluating performance on the test dataset

In this section, the classification report is presented for each of the 6 classifiers and their performances are compared when predicting the test data set. The performance metrics that are measured are precision, recall, f1-score and accuracy.

5.1 Support Vector Machine

Though the accuracy of the model on the test set has decreased compared to the training set, the performance of the model is consistent across multiple runs of different test train splits. This shows that SVM has generalized well and is consistent in its performance and predictions. In general, it is also known to work well with scaled data as seen from the figure.

	precision	recall	f1-score	support
1	1.00	0.97	0.98	30
2	0.97	1.00	0.99	33
3	1.00	1.00	1.00	26
accuracy			0.99	89
macro avg	0.99	0.99	0.99	89
weighted avg	0.99	0.99	0.99	89

Figure 5.1 Test score of SVC

5.2 k-Nearest Neighbors

k-Nearest Neighbors has been one of the most inconsistent models on repeated runs of the training and prediction. This performance on the test data below shows that the accuracy dropped to 97% from 98% but it has even showed accuracy as low as 90%. This is because the performance of the model is dependent on the points that are present in the training data which are used to predict the test data. Noise or outliers in the training data will have a negative impact on the test data points. It is sensitive to the overlapping points which result in such misclassification errors. This is shown by the F-1 scores of classes 1 and 2, both having higher number of instances in the test set.

	precision	recall	f1-score	support
1	0.91	1.00	0.95	30
2	1.00	0.91	0.95	33
3	1.00	1.00	1.00	26
accuracy			0.97	89
macro avg	0.97	0.97	0.97	89
weighted avg	0.97	0.97	0.97	89

Figure 5.2 Test score of K-Nearest Neighbors

5.3 Logistic Regression

Logistic regression performs consistently on repeated runs of prediction between different test train splits. However, given the number of attributes, logistic regression is at a risk of overfitting and hence sometimes performs poorly when compared to support vector machines. In this case, it has shown to generalize well by achieving 99% accuracy on the test set.

	precision	recall	f1-score	support
1	1.00	0.97	0.98	30
2	0.97	1.00	0.99	33
3	1.00	1.00	1.00	26
accuracy			0.99	89
macro avg	0.99	0.99	0.99	89
weighted avg	0.99	0.99	0.99	89

Figure 5.3 Test score of Logistic Regression

5.4 Naive-Bayes Classifier

Naive-Bayes Classifier performs well in high dimensional space and is robust to isolated noise points. However, it was decided not to choose Naive Bayes as the preferred classifier because of the lack of domain knowledge w.r.t the attributes. It is difficult to determine if all the attributes are conditionally independent. The classifier however performs consistently and has accuracies upwards of 95%. Here, it achieved an accuracy of 98% on the test set.

	precision	recall	f1-score	support
1	1.00	0.97	0.98	30
2	0.97	0.97	0.97	33
3	0.96	1.00	0.98	26
accuracy			0.98	89
macro avg	0.98	0.98	0.98	89
weighted avg	0.98	0.98	0.98	89

Figure 5.4 Test score of Naive-Bayes

5.5 Decision Tree Classifier

The decision tree classifier had the worst performance in terms of its ability to generalize. Its exceptional performance on the training data can be attributed to overfitting. The accuracy of 92% on the test data shows that the model has overfitted to the training data. Furthermore, since all the attributes are continuous in nature, decision trees might not be the right choice.

	precision	recall	f1-score	support
1	0.90	0.93	0.92	30
2	0.91	0.88	0.89	33
3	0.96	0.96	0.96	26
accuracy			0.92	89
macro avg	0.92	0.92	0.92	89
weighted avg	0.92	0.92	0.92	89

Figure 5.5 Test score of Decision Tree

5.6 Ensemble Classifier

A random forest classifier constructs a set of uncorrelated decision tree classifiers, then aggregates the results. Though it has better performance that decision trees and works with continuous attributes, it is slower to train and is outperformed by support vector machines as its accuracy measure was 98%.

	precision	recall	f1-score	support
1	0.97	0.97	0.97	30
2	0.97	0.97	0.97	33
3	1.00	1.00	1.00	26
accuracy			0.98	89
macro avg	0.98	0.98	0.98	89
weighted avg	0.98	0.98	0.98	89

Figure 5.6 Test score of Random Forest Classifier

6. Improving a Model

In order to demonstrate the effects of hyper-parameter tuning on the accuracy of a model, we have chosen the decision tree classifier and the k-Nearest Neighbors because of its inconsistent and relatively poor performance compared to the other classifiers. Upon repeated fitting of the model to different samples of test and train data, decision trees have an accuracy of around **85-92%** and k-Nearest Neighbors had accuracy in the range of **90-97%**. To improve the model, a hyperparameter optimizer is applied and its performance evaluated on the same split of train data. When an optimal model is obtained, predictions are rerun on the test set.

The function provided by scikit learn to optimize hyperparameters is model_selection.GridSearchCV. This function runs an exhaustive search over the specified parameter values for an estimator. GridSearchCV implements a “fit”, “score” and “predict” method. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

6.1 Decision Trees Classifier

The hyperparameters being tuned in the case of decision trees are the following:

1) Maximum Depth of the decision tree
2) Maximum number of features: [‘sqrt’, ‘log2’]

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')

Default classifier:  0.9325842696629213

GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort=False, random_state=None,
                                              splitter='best'),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
                                       16, 17, 18, 19],
                         'max_features': ['sqrt', 'log2']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)

Parametrized classifier:  0.9550561797752809
Parameters:  {'max_depth': 8, 'max_features': 'log2'}
```

Figure 6.1 Output of GridSearchCV for Decision Tree

An improvement of 2% was improved in the performance of the classifier on the test set when the default classifier was compared with the optimal one.

6.1 k-Nearest Neighbors Classifier

The hyperparameters being tuned in the case of k-Nearest Neighbors are the following:

1) Number of neighbors, or k
2) Weights for calculating closeness, uniform or using distance

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')

Default classifier:  0.9662921348314607

GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid={'n_neighbors': [3, 5, 7, 9, 11, 13, 15, 17, 19, 21],
                         'weights': ['uniform', 'distance']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)

Parametrized classifier:  0.9775280898876404
Parameters:  {'n_neighbors': 3, 'weights': 'uniform'}
```

Figure 6.2 Output of GridSearchCV for k-Nearest Neighbors

Accuracy measure of the optimal model was greater than the default model by 1%, which is not that high but there is a difference.

7. Conclusion

Below are the main observation made in this lab:

- Dataset was clean, except for the different measurement scales of different features.
- Scaling was done in order to deal with the different features.
- There were no outliers, and the classes were also balanced.
- More domain knowledge could translate to better feature engineering of this problem.
- The models that performed best in training were:
    - Support Vector Machine
    - Logistic Regression
    - Decision Tree
    - Random Forest
- This assertion is based on the classification report of the predictions made by the models on the train set after training was complete. All of them had perfect accuracy on the train set.
- The models that generalized well (Performed best on train set):
    - Support Vector Machine
    - Logistic Regression
    - Random Forest
- Again, this assertion is based on the classification reports of the models on the test set after they were trained on the train set only. Thus, this gives us a good measure of the generalization error in an optimistic way.
- Using GridSearchCV showed a slight improvement in the performance(~2%) of Decision Tree and k-Nearest Neighbors classifier. However, this was again dependant on the train-validation split for each of the validation runs in cross-validation. All the outputs needed to make this assertion have been included in the report (section 6).

8. References

[1] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[2] https://en.wikipedia.org/wiki/Limited-memory_BFGS
[3] https://scikit-learn.org/stable/modules/naive_bayes.html
[4] https://scikit-learn.org/stable/modules/tree.html
[5] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[6] https://scikit-learn.org/stable/modules/svm.html