Group - Room 6

Compiled by:
smck0583 510678236
kong7456 500459007
jdor8613 500490723
mlim2731 500308222

Domain and Dataset

The dataset used was collected from the UC Irvine Machine Learning Repository website under the title ‘Iris Data Set’. The data itself was originally recorded by R.A. Fisher in 1936 and is based on analysis of iris plants. The dataset contains 150 values with 5 attributes:

1. Sepal length
2. Sepal width
3. Petal length
4. Petal width
5. Class

The attribute to be predicted was chosen as the class attribute, relating to the class of each iris plant recorded in the dataset. The dataset was randomly split into a training dataset (iris_train.csv) and a test dataset (iris_test.csv) through R, where the training set contains 135 values and the test set contains 15 values.

Multiclass Gaussian Naive Bayes Classifier Method (510678236)

Within this stage, the team has decided to investigate the relationship between the classification of the Iris plant (Iris Virginica, Iris Setosa, and Iris Versiolour) and the plant's quantitative attributes including sepal length, sepal width, petal length and petal width. These qualities are explored in the “iris.data” entitled data set featured on the University of California - Irvine Machine Learning Website.

The Iris plant classification objective may be considered and solved as a multi-class machine learning classification problem. Although there are many methods within machine learning that deal with multi-class identification, one method that may be employed is the Gaussian Naive Bayes method. Using this method, all of the given variables within the Iris data set may be used to help classify the flower belonging to each row into an Iris type classification. The Gaussian Naive Bayes method was specifically chosen due to the type of data within the input variables. The variables, sepal length, sepal width, petal length and petal width, are all quantitative and it may be proposed that those quantitative qualities may be normally distributed. Given these qualities it may be believed that the Gaussian Naive Bayes method will work well with the data set to predict the Iris flower classification.

Gaussian Naive Bayes Iris Classification Method on Iris Data Set Code:

```
import pandas as pd
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report

train_df = pd.read_csv("iris_train.csv")
column_names = list(train_df.columns)
Iris_traits = column_names[0:4]
Iris_name = column_names[4]
X_train = train_df[Iris_traits]
y_train = train_df[Iris_name]

test_df = pd.read_csv("iris_test.csv")
X_test = test_df[Iris_traits]
y_test = test_df[Iris_name]



gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)

print("The Accuracy Score: ")
print(metrics.accuracy_score(y_test,y_pred))
print()
```

```
print("The Balanced Accuracy Score: ")
print(metrics.balanced_accuracy_score(y_test, y_pred))
print()
print("The Precision Score: ")
print(metrics.precision_score(y_test, y_pred, average='weighted'))
print()
```

Output:
shaemckenna1@Shaes-MacBook-Pro DataSciene % python3 Machine_learning.py
The Accuracy Score:
0.9333333333333333

The Balanced Accuracy Score:
0.9166666666666666

The Precision Score:
0.9444444444444445
To evaluate the Gaussian Naive Bayes Machine Learning classifier method for the Iris data, 3 accuracy methods were implemented including metrics.accuracy_score, metrics.balanced_accuracy_score and metrics.precision_score. From the 2 accuracy methods, it may be deduced that the Gaussian Naive Bayes model worked very well achieving an accuracy score of 93.3%, a balanced accuracy score of 91.67% and a precision score of 94.4% all on the test set. Therefore, it may be claimed that even when the accuracy score is adjusted for the imbalanced nature of the dataset (with balanced accuracy score) and the possibility of false positives (with precision score), the overall scores of the Gaussian Naive Bayes model still reaches over 90% accuracy. The variances within the scores may suggest that there is some imbalance within the data which lead the model to predict some Iris classifications better than others, which may simply be attributed to the fact that there is less data for certain classifications (in an already small data set) for the algorithm to work with. Furthermore, the slightly higher score for precision would likely suggest that the model does quite well in not predicting an incorrect class for any given Iris entry.

However, despite the Gaussian Naive Bayes model being relatively quite successful in its predictions, the characteristics of the data set and the nature of the training approach may have led the model to be less accurate than it could have been. Firstly, with only 150 rows, there is only a limited number of Iris flowers documented within the data set, especially after it is then divided into training and testing sets. This is particularly important for the Gaussian Naive Bayes model as it relies on each of the input variables columns being normally distributed, with a smaller data set the data is less likely to conform to the central limit theorem therefore less likely to be approximately normally distributed. Furthermore, this may also affect the training approach implemented, since the data set is quite small, the further constricting of the set into train and test sets would have likely decreased the performance of the model as it only has a limited number of data points to make predictions on.

Logistic Regression (500459007)

Another approach in predicting the flower class for analysis is through the linear regression algorithm. Using the scikit-learn library on Python, the algorithm takes in the dataset and all its values and utilises them in predicting a certain flower class from a sample case.

Logistic regression model code:

```
import pandas as pd
from math import sqrt
from sklearn import linear_model
from sklearn import metrics
from sklearn.model_selection import train_test_split

sir_choo_choo = pd.read_csv('logic train.csv')
column_name = list(sir_choo_choo.columns)
iris_thing = column_name[0:4]
iris_things = column_name[4]
X_train = sir_choo_choo[iris_thing]
y_train = sir_choo_choo[iris_things]
jarrod = pd.read_csv('logic test.csv')
X_test = jarrod[iris_thing]
y_test = jarrod[iris_things]

clf = linear_model.LogisticRegression(solver='liblinear').fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_pred_proba = clf.predict_proba(X_test)
last_sample_proba = y_pred_proba[-1]

last_sample = X_test.loc[list(X_test.index)[-1]]
print('Predicted class:',y_pred[-1])
print('Actual class:', y_test.loc[list(y_test.index)[-1]])
```

With the model, three measures of performance were used in assessing its ability in accurately predicting the flower class from the dataset; the coefficients from the model, a balanced accuracy measurement and an overall accuracy measurement of the entire model.

Measures of performance code:

```
print('Coefficients:')
print(clf.coef_)
print("Balanced accuracy:", metrics.balanced_accuracy_score(y_test, y_pred))
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

Output of entire model:

```
Predicted class: Iris-virginica
Actual class: Iris-virginica
Coefficients:
[[ 0.41986484  1.42783335 -2.22074987 -1.02119552]
 [ 0.43346862 -1.58234603  0.47052639 -1.13898706]
 [-1.61004024 -1.38230878  2.35247914  2.3087628 ]]
Balanced accuracy: 0.9259259259259259
Accuracy: 0.8666666666666667
```

The code predicted Iris-virginica in accordance with the sample case provided. In terms of the coefficients, the coefficient corresponding to both the petal width and length variable is larger than the others, suggesting that those two variables have a higher association with the flower class than the other two variables.

With the accuracy tests, the metrics.accuracy_score method was used in measuring an overall accuracy score for the model's prediction of flower class and the metrics.balanced_ accuracy_score method was used on the test data. The overall accuracy result for both the training and test data was at 86.6%, indicating that the model's predictions are strongly accurate in predicting the flower class from the dataset. The balanced accuracy score was slightly higher at 92%, supporting the overall accuracy score and indicating that the model is likely not unbalanced; the higher the score, the more balanced the data.

With the logistic regression model and its measurements, the accuracy measured could be a result of the small sample size of the dataset (150 values). As compared to other datasets with over thousands of values, a small one with 150 would not provide with a wide diversity or variety of recorded results as the range of data is limited, thus leading to a potentially inaccurate model as the result. On the other hand, if the dataset is of high dimensions, it would lead to overfitting on the logistic regression model as there are simply too many parameters in forming an accurate prediction. The model is also prone to outliers which can affect outcomes of predictions leading to inaccuracy.

Decision Tree (500490723)

Another approach in predicting the flower class is through the use of a decision tree classifier utilised through the scikit-learn library on Python.

```
import pandas as pd
from sklearn import metrics
from sklearn import tree
from sklearn.model_selection import train_test_split

df_train = pd.read_csv('iris_train.csv')
df_test = pd.read_csv('iris_test.csv')

column_names = list(df_train.columns)
Iris_stats = column_names[0:-1]
Iris_type = column_names[-1]

X_train = df_train[Iris_stats]
y_train = df_train[Iris_type]
X_test = df_test[Iris_stats]
y_test = df_test[Iris_type]

clf = tree.DecisionTreeClassifier(max_depth=2).fit(X_train, y_train)
y_pred = clf.predict(X_test)
train_pred = clf.predict(X_train)


#calculate accuracy of training data and test data
print(f"Training Set Accuracy: {round(metrics.accuracy_score(y_train, train_pred),2)}%")
print(f"Accuracy score: {round(metrics.accuracy_score(y_test, y_pred),2)}%")
print(f"Balanced Accuracy score: {round(balanced_accuracy_score(y_test, y_pred),2)}%")
```

```
Training Set Accuracy: 0.96%
Accuracy score: 0.93%
Balanced Accuracy score: 0.96%
```

For testing accuracy of my results metrics.accuracy_score was used for both the training data set and testing data set, as well as balanced_accuracy_score also being used on the test set. Firstly an accuracy score of 93% for the test data shows that the predictive model works very well in correctly predicting species of iris. A balanced accuracy score of 96% also confirms this and also mentions that the data may be ever so slightly imbalanced, though this imbalance does not reveal that there are issues with our predictive model(if balanced accuracy was low it would mean that the model was not properly predicting). By comparing the training set accuracy with the other two accuracy measurements it could be concluded that there aren't any real overfitting or underfitting issues. Decision trees are known to be prone to overfitting and originally the max_depth had no limit so the tree extended to 15 nodes with a depth of 5. The training set accuracy was the only score affected and this is a prime example of such overfitting. Therefore I limited the max_depth to 2, to get the most consistent results.

K-Nearest Neighbours Classification (500308222)

The dataset used is “iris.data.csv”, and the four attributes (sepal_length, sepal_width, petal_length and petal_width) are used to predict the class which has 3 levels which are coded in the column “class.val” as follows:

- 1: Iris-setosa
- 2: Iris-versicolor
- 3: Iris-virginica

The algorithm used is the K-Nearest Neighbours (KNN) classification, with the choice of K=5. Before running the algorithm, the attributes sepal_length, sepal_width, petal_length and petal_width were normalised to bring all the attributes to the same scale and improve model performance.

```
import pandas as pd
from math import sqrt
import numpy as np
from sklearn import metrics
from sklearn import neighbors
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

df_train = pd.read_csv("iris_train.csv") # Import Training Dataset
df_test = pd.read_csv("iris_test.csv") # Import Testing Dataset

# slice training dataFrame for input variables
X_train = df_train[['sepal_length','sepal_width','petal_length','petal_width']]
# Normalise training dataset input variables
normalised_X = preprocessing.normalize(X_train)
X_train = pd.DataFrame(normalised_X, columns = X_train.columns)

# slide testing dataset for input variables
X_test = df_test[['sepal_length','sepal_width','petal_length','petal_width']]
# Normalise testing dataset input variables
normalised_X_test = preprocessing.normalize(X_test)
X_test = pd.DataFrame(normalised_X_test, columns = X_test.columns)

y_train = df_train['class.val'] # slice training dataset for output variable
y_test = df_test['class.val'] # slice testing dataset for output variable

n_neighbours = 5 # set hyperparameter K=5

# run KNN algorithm
knn_model = neighbors.KNeighborsClassifier(n_neighbours).fit(X_train, y_train)
```

```
# use the KNN model to predict X_test
y_pred = knn_model.predict(X_test)
```

Model Evaluation:

```
# generate model accuracy for test set
print("Test Set Accuracy:",metrics.accuracy_score(y_test, y_pred))
# generate R-squared score
print("F1 Score:",metrics.f1_score(y_test, y_pred,average='weighted'),"\n")

# generate model accuracy for training set
train_pred = knn_model.predict(X_train)
print("Training Set Accuracy:",metrics.accuracy_score(y_train, train_pred))
```

Command output:
Test Set Accuracy: 1.0
F1 Score: 1.0

Training Set Accuracy: 0.9703703703703703

The test set accuracy score of 1.0 indicates that the model predicted the class in the test set with 100% accuracy. This may be due to the small size of the test set or pure luck. However, the extremely high accuracy rate is not surprising as the KNN classification algorithm was expected to perform well with this dataset, due to its low dimensionality, minimal noise and lack of categorical features besides the output variable. The accuracy is not indicative of overfitting or underfitting, especially as it can be seen that the training set accuracy produced by the same model is 97.77% which is less than 100% but still high.The F-score of 1.0 additionally indicates that the model has perfect precision and recall when predicting the test set. This is likely attributable to the same reasons as the perfect test set accuracy score.

The four machine learning methods used were Multiclass Gaussian Naive Bayes, logistic regression, decision tree, and K-nearest neighbour.

The K-nearest neighbours model achieved 100% accuracy in predicting the test set. However, this can be attributed to the fact that the Iris dataset only has 4 input variables and virtually no noise. The KNN model was performed extremely well with the Iris dataset as the model is suitable for low dimensional data, and the data was scaled in order to lessen the impact of differing scales of the input variables on the euclidean distance measure used in the model. The weakness of the KNN algorithm is that it depends heavily on the quality of the data. When used with datasets with high dimensionality, there is an increased risk of overfitting which can negatively impact model performance. Furthermore, the KNN classifier is particularly sensitive to noise in the data, which impacts its real-life application as noise from various sources is often prevalent in large scale databases. Regarding real-life applications, KNN is also slower and costlier in terms of speed and memory in the testing phase as the storage of the entire dataset for prediction generally requires a large amount of memory. Therefore, it is expected that the other models would outperform the KNN model in speed and accuracy when used in real-life applications where larger datasets with noise and higher dimensionality are more likely to be used in order to produce useful, data-driven insights.

The decision tree model achieved an accuracy of 93% in predicting the iris class within the test set, therefore being very accurate. Interestingly increasing the max depth of the tree past 2, did not actually change either of the accuracy results for the test set, only the training set accuracy increased, making the model only overfit the data. It is difficult to accurately say how well this model compares to other models due to the limited size of data, but in general all models worked comparatively well at predicting the right class, with the decision tree being on the slightly better end. A strength and also a weakness of decision trees is the ability to change the amount of rows of nodes. By changing the max_depth the decision tree will create another row of decisions to separate to data, making a more accurate prediction, though the issue with this is that if the value is to high or no max is given the decision tree will simply make a node for every single data point in the training set making the model completely overfit the train data.

The logistic regression model achieved an accuracy of 86% in predicting the flower class from the dataset, indicating a strong model in forming accurate predictions. The model offers an ease of use in implementation whilst producing well-calibrated outputs from datasets, enhancing the reliability of predictions made from the dataset. The model can also be conveniently extended to include multiple classes and variables (multinomial regression), offering a layer of flexibility with analysing different datasets. The model is also suited for the size of the dataset used as logistic regression models are less prone to overfitting with low dimensional datasets, as long as there is a sufficient amount of data in the training set. However, the model also has its weaknesses including its limited ability in forming accurate predictions with datasets of high and complex dimensions; the model is simply not as strong as compared to other models/algorithms such as k-nearest neighbours or neural networks. To compare logistic regression with other models analysing this dataset (decision tree, Gaussian Naive Bayes, k-nearest neighbours) would not lead to an accurate representation of which model performed best (or worse) as each model

have their own specific advantages and disadvantages with predictions from datasets. With the dataset provided, all models performed well with high accuracy scores. However, this result is still limited as the dataset was of small dimensions.

Although the Gaussian Naive Bayes model performed well with an accuracy score of 93.3%, it did not score as well as the K-nearest neighbors method which in large part is attributable to the failure of the dataset to uphold the assumptions of the model. It was previously discussed that the model assumes that the data given within the input columns is normally distributed which is unlikely to be exhaustively true given the small size of the data set. However, it is also quite likely that the data set did not uphold another Naive Bayes assumption which assumes independence between the input variables. Although it wasn't tested within this paper, it is likely that the iris petal and sepal measurements likely have interdependence.

Yet, given the high accuracy results, one may conclude that the violations of these assumptions by the Iris data themselves may have only been minor (perhaps a small number of outliers for instance) and hence did not negatively affect the Gaussian Naive Bayes model accuracy much. Furthermore, the lack of need to tune hyperparameters made the model extremely easy to use as it is generalizable. It also worked really well with the multi class data set which given the ease of use makes it readily generalizable. Although, it is likely that due to the failure of the Iris data set to uphold the assumptions, the model did not perform as well as the K-nearest neighbors model, whose assumptions were upheld quite well. With that being said, these conclusions have inherent limitations as it was only proposed that lack of interdependence and outliers may have limited normality within the data. Those accusations have limited validity without a followup study that investigates these properties within the dataset.