Iris Flower Classification

Objective

The aim is to classify iris flowers among three species from measurements of sepals and petals' length and width.The iris data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.The central goal here is to design a model that makes useful classifications for new flowers or, in other words, one which exhibits good generalization.

Importing libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as  plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Importing Data from .csv file

```
columns=["SepalLengthCm","SepalWidthCm","PetalLengthCm","PetalWidthCm","Species"]
dataset = pd.read_csv('/content/iris_data.csv',names=columns)
dataset.head()
```

```
dataset.tail()
```

```
dataset.shape
```

Exploratory Data Analysis (EDA)

Histogram of petal length

```
import seaborn as sns
import matplotlib.pyplot as plt

g = sns.FacetGrid(dataset, hue="Species", height=8)
g.map(sns.histplot, "PetalLengthCm", kde=True, stat="density", alpha=0.4).add_legend()
plt.show()
```

Histogram of petal width

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.FacetGrid(dataset, hue="Species", height=8) \
    .map(sns.distplot, "PetalWidthCm") \
    .add_legend()
plt.show()
```

Histogram of sepal length

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.FacetGrid(dataset, hue="Species", height=8) \
    .map(sns.distplot, "SepalLengthCm") \
    .add_legend()
plt.show()
```

Histogram of sepal width

```
sns.FacetGrid(dataset, hue="Species", height=5) \
    .map(sns.distplot, "SepalWidthCm") \
    .add_legend()
plt.show()
```

2D scatter plot

```
dataset.plot(kind='scatter', x='SepalLengthCm',y='SepalWidthCm')
plt.show()
```

2D scatter plot with color-coding for each flower type/class.

```
sns.set_style("whitegrid")
sns.FacetGrid(dataset, hue="Species", height=4) \
    .map(plt.scatter, "SepalLengthCm", "SepalWidthCm") \
    .add_legend()
plt.show()
```

Pairwise Scatter Plot

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("whitegrid")
sns.pairplot(dataset, hue="Species", height=3, aspect=1)
plt.show()
```

Data Preprocessing

```
dataset["Species"].value_counts()
```

```
dataset.tail()
```

descriptive statistics of data set

```
dataset.describe()
```

Checking for Null Values

```
dataset.isnull().sum()
```

```
dataset['Species'].unique()
```

label Encoding

```
label_encoder = preprocessing.LabelEncoder()
dataset['Species']= label_encoder.fit_transform(dataset['Species'])

dataset['Species'].unique()
```

```
#X = dataset.drop(["Species"], axis=1)
#y = dataset["Species"]
```

Splitting the dataSet

```
X = dataset.iloc[:, [2,3]].values

y = dataset.iloc[:, 4].values
```

```
len(X)
```

```
len(y)
```

heatmap is to identify the highly correlated features

```
plt.figure(figsize=(7,4))
sns.heatmap(dataset.corr(),annot=True,cmap='cubehelix_r')
plt.show()
```

Model Development

splitting the dataset into train set and test set

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3, random_state = 2)
```

Feature Scaling

```
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)

print("X_train:", X_train)
print("X_test:", X_test)
```

DecisionTree Classifier

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

Predicting the test results

```
y_pred = classifier.predict(X_test)
```

Making the confusion matrix

```

```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Visualization

Visualising the Training set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('pink','skyblue','gray')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('pink', 'skyblue','gray'))(i), label = j)
plt.title('Decision tree Classifier (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green','blue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green','blue'))(i), label = j)
plt.title('Decision tree Classifier  (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

Accuracy

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from math import sqrt
print('The accuracy of the Decision Tree Classifier is : %.2f'%accuracy_score(y_pred,y_test))
rmse = sqrt(mean_squared_error(y_test, y_pred))
print("RMSE value = %.2f"%rmse)
print("R2 Score= %.2f"%r2_score(y_test, y_pred))
```

```
classification_report(y_test, y_pred)
```

KNN Algorithm

```
from sklearn import neighbors
model = neighbors.KNeighborsClassifier(n_neighbors=3)
```

```
model.fit(X_train,y_train)
```

```
predict = model.predict(X_test)
```

Accuracy

```
#for checking the model accuracy
print('The accuracy of the KNN is',accuracy_score(predict,y_test))
rmse = sqrt(mean_squared_error(y_test, predict))
print("RMSE value = %.2f"%rmse)
print("R2 Score= %.2f"%r2_score(y_test, predict))
```

SVM

```
from sklearn import svm
svc = svm.SVC(C=1.0, kernel='rbf')
```

```
svc.fit(X_train,y_train)
```

```
pred = svc.predict(X_test)
```

validation

```
print('The accuracy of the SVM is: %.2f'%accuracy_score(pred,y_test))
rmse = sqrt(mean_squared_error(y_test, pred))
print("RMSE value = %.2f"%rmse)
print("R2 Score= %.2f"%r2_score(y_test, pred))
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression # for Logistic Regression algorithm
model = LogisticRegression()
model.fit(X_train,y_train)
prediction=model.predict(X_test)
```

Accuracy

```
print('The accuracy of the Logistic Regression using Petals is:',accuracy_score(prediction,y_test))
rmse = sqrt(mean_squared_error(y_test, prediction))
print("RMSE value = %.2f"%rmse)
print("R2 Score= %.2f"%r2_score(y_test, prediction))
```

Naive_bayes

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train,y_train)
predicted= model.predict(X_test)
```

Accuracy

```
print('The accuracy of the naive_bayes is:',accuracy_score(predicted,y_test))
rmse = sqrt(mean_squared_error(y_test, predicted))
print("RMSE value = %.2f"%rmse)
print("R2 Score= %.2f"%r2_score(y_test, predicted))
```