PCC-DS 391 Lab Assignment 6

Topic: Python Pandas and Classification algorithms of Machine Learning

1. Write a program to perform the following tasks using pandas .
    a. Create a Dataframe from a dictionary that has three keys such as “Name”, “Age”, “Department” and its corresponding values.
    b. Create a Dataframe from a csv file that has three columns such as “Name”, “Age”, “Department” and rows corresponding to values of the three columns.
    c. Concatenate the above two dataframes into one Dataframe.
    d. Select only the "Name" and "Age" columns from that Dataframe.
    e. Filter the DataFrame to include only rows where “Age” is greater than 25.
    f. Add a new column, "Salary", with some values to the DataFrame.
    g. Group the DataFrame by "Department" and calculate the average age for each Department.
    h. Sort the DataFrame by the "Age" column in descending order.
    i. Calculate the mean, median, and sum of the "Salary" column.
    j. Write the contents of a Dataframe into a csv file.

```
import pandas as pd

emp_dict = {
   'Name': ['Alice', 'Bob', 'Charlie'],
   'Age': [25, 30, 35],
   'Department': ['HR', 'Engineering', 'Sales']
}

df1 = pd.DataFrame(emp_dict)
print("Dictionary -> Datafrane: Contents of Dataframe1")
print(df1)

df2 = pd.read_csv('emp_data.csv') #keep this emp.csv file in the same folder where the code exists.
print("CSV -> Datafrane: Contents of Dataframe2")
print(df2)

df = pd.concat([df1, df2], ignore_index=True)
print("Dictionary and CSV Concatenated: Contents of Dataframe3")
print(df)

selected_df = df[["Name", "Age"]]
print("Name and Age Columns")
print(selected_df)

filtered_df = df[df["Age"] > 25]
print("Age > 25")
print(filtered_df)

df["Salary"] = [50000, 60000, 70000, 45000, 82000, 35000]
print("Add new column Salary")
print(df)
```

PCC-DS 391 Lab Assignment 6

```
grouped_df = df.groupby("Department")["Age"].mean()
print("Average Age for each Department")
print(grouped_df)

sorted_df = df.sort_values(by="Age", ascending=False)
print("Sort by the Age column in descending order")
print(sorted_df)

mean_salary = df["Salary"].mean()
median_salary = df["Salary"].median()
sum_salary = df["Salary"].sum()
print("Mean, median, and sum of the Salary column")
print("Mean:", mean_salary)
print("Median:", median_salary)
print("Sum:", sum_salary)
```

2. Write a program to run Support Vector Machines (SVM) on the Iris dataset and calculate its accuracy using scikit-learn. This code will train the SVM model and then evaluate its accuracy on the test set.

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data  # Features
Y = iris.target  # Target classes

# Split the dataset into training and test sets (70% train, 30% test)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=42)

# Create an SVM model
model = SVC(kernel='linear')  # You can try different kernels like 'rbf', 'poly', etc.

# Fit the model to the training data
model.fit(X_train, Y_train)

# Make predictions on the test set
Y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

PCC-DS 391 Lab Assignment 6

Explanation

1. SVC stands for Support Vector Classification, and it's a class in the scikit-learn library used to implement Support Vector Machines (SVM) for classification tasks.
2. **Load the Data**: The Iris dataset is loaded, and features (X) and targets (y) are extracted.
3. **Train-Test Split**: The dataset is split into a training set (70%) and a test set (30%).
4. **Create and Train the Model**: An SVM model is created using a linear kernel and trained on the training set.
5. **Make Predictions**: The model predicts the classes for the test set.
6. **Calculate Accuracy**: The accuracy is calculated using the accuracy_score function and printed.
7. **`random_state = 42`**: Every time you run this code, you'll get the same training and testing sets, making your experiments more reliable. It has become a common convention in programming examples to use the number 42 for arbitrary values, but you can use any integer for random_state.

3. Write a program to run Naïve Bayes on the Iris dataset and calculate its accuracy using scikit-learn. This code will train the Naïve Bayes model and then evaluate its accuracy on the test set.

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data  # Features
Y = iris.target  # Target classes

# Split the dataset into training and test sets (70% train, 30% test)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=42)

# Create a Naive Bayes model
model = GaussianNB()

# Fit the model to the training data
model.fit(X_train, Y_train)

# Make predictions on the test set
Y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

Explanation

1. **Load the Data**: The Iris dataset is loaded, and features (X) and targets (y) are extracted.

2. **Train-Test Split**: The dataset is split into a training set (70%) and a test set (30%).
3. **Create and Train the Model**: A Gaussian Naive Bayes model is created and trained on the training set.
4. **Make Predictions**: The model predicts the classes for the test set.
5. **Calculate Accuracy**: The accuracy is calculated using the accuracy_score function and printed.

4. Write a program to run Random Forest on the Iris dataset and calculate its accuracy using scikit-learn. This code will train the Random Forest model and then evaluate its accuracy on the test set.

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data  # Features
Y = iris.target  # Target classes

# Split the dataset into training and test sets (70% train, 30% test)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=42)

# Create a Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model to the training data
model.fit(X_train, Y_train)

# Make predictions on the test set
Y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

Explanation

1. **Load the Data**: The Iris dataset is loaded, and the features (X) and target labels (y) are extracted.
2. **Train-Test Split**: The dataset is split into a training set (70%) and a test set (30%).
3. **Create and Train the Model**: A Random Forest classifier is created with 100 trees and trained on the training data. So, the value of n_estimators is 100.
4. **Make Predictions**: The trained model predicts the classes for the test set.
5. **Calculate Accuracy**: The accuracy of the model's predictions is calculated using the accuracy_score function and printed.