Open Lab I

Class: DE-42 CE-A | **Subject: EC 350- AI Decision Support**

Time Allowed: 2hr | **Total Marks: 20**

Date: Jan 12, 2024.

Instructions:

- Submit the report on LMS in .docx/pdf format. Add the code and output in the report.

- Do your own work. Any kind of plagiarism found will result in negative marks.

Statement:

Algorithms trained on vast datasets to identify patterns and anomalies that might signal the early stages of the disease. While not a crystal ball, such systems could revolutionize early detection, allowing for earlier interventions and potentially saving countless lives. But it's not just about individual risk; these systems can also guide public health initiatives, tailoring screening programs and preventative measures to those most susceptible. While ethical considerations and limitations in accuracy remain, the potential of cancer prediction systems to become a powerful weapon in the fight against this devastating disease is undeniable. Design an AI model which will predict the risk of developing cancer in individuals. You can use any algorithm which you found suitable for the given data set.

Tasks to do:

- Perform a 6-fold cross validation on the given dataset.
- Apply any 1 ML/AI algorithm.
- Plot a confusion matrix.
- Discuss and analyze the results and the misclassifications in the confusion matrix.

Results / Outcomes:

- Predicted class of the input data
- Accuracy of the model
- Confusion Matrix

Dataset:

You are provided with a dataset csv file.

Things to submit:

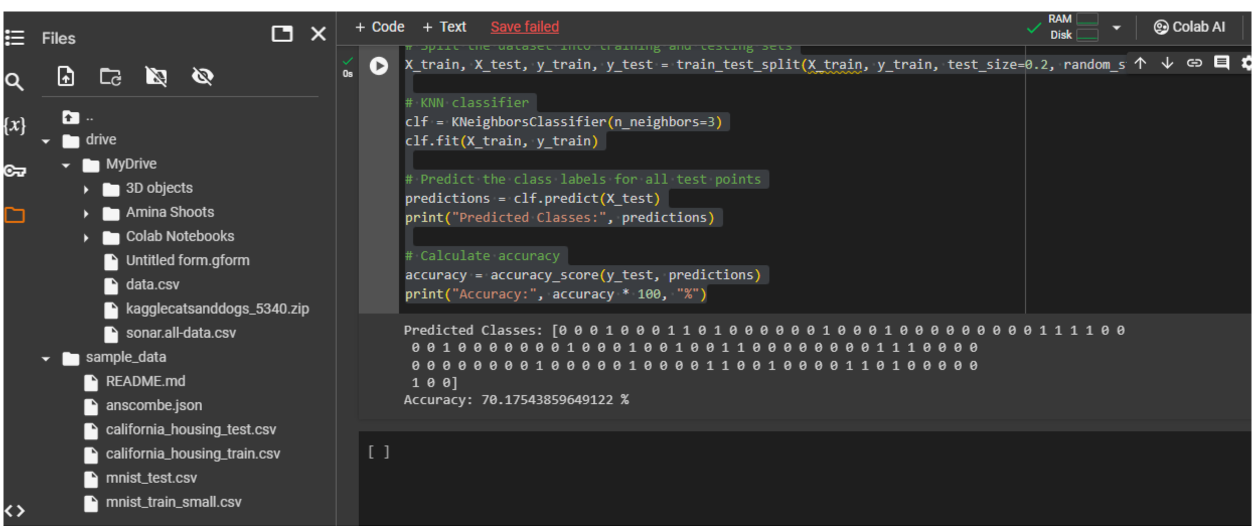
You are to submit a word file with your code, results and plots for confusion matrix and a methodology defining your code, your logic and discussion.

Code:

```
# Initialize the SVM model (or replace with your preferred algorithm)
model = SVC(kernel='linear')

# Perform 6-fold cross-validation and get predicted labels
predicted_labels = cross_val_predict(model, X, y, cv=6)

# Calculate accuracy
accuracy = np.mean(cross_val_score(model, X, y, cv=6, scoring='accuracy'))

# Calculate confusion matrix
conf_matrix = confusion_matrix(y, predicted_labels)

# Plot confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# Display results
print("Predicted class of the input data:", predicted_labels)
print("Accuracy of the model:", accuracy)
print("Confusion Matrix:")
print(conf_matrix)

import numpy as np
import pandas as pd
from sklearn.model_selection import cross_val_predict, cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import seaborn as sns

from google.colab import files

diabetes_df = pd.read_csv('/content/drive/MyDrive/data.csv')
```

```python
diabetes_df = np.array(diabetes_df)


# Function to calculate Euclidean distance between two points
def euclidean_distance(x1, x2):
    distance = np.sqrt(np.sum((x1 - x2) ** 2))
    return distance

# KNN Class
class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        predictions = [self._predict(x) for x in X_test]
        return np.array(predictions)

    def _predict(self, x):
        # Calculate distances to all points in the training set
        distances = [euclidean_distance(x, x_train) for x_train in
self.X_train]

        # Get indices of k-nearest training data points
        K_neighbors_indices = np.argsort(distances)[:self.k]

        # Get the labels of the k-nearest training data points
        k_neighbor_labels = [self.y_train[i] for i in K_neighbors_indices]

        # Return the most common class label among the k-nearest neighbors
        most_common = np.bincount(k_neighbor_labels).argmax()
        return most_common




from sklearn.preprocessing import LabelEncoder

# Assuming 'diagnosis' is the last column
diagnosis_labels = diabetes_df[1:, -1]
```

```
# Use LabelEncoder to convert 'M' to 1 and 'B' to 0
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(diagnosis_labels)

label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(diagnosis_labels)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(diagnosis_labels)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,
test_size=0.2, random_state=42)

# KNN classifier
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)

# Predict the class labels for all test points
predictions = clf.predict(X_test)
print("Predicted Classes:", predictions)

# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy * 100, "%")
```



```python
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_predict, cross_val_score
from sklearn.metrics import confusion_matrix

X_train = diabetes_df[1:, :2] # Exclude the first row for testing
y_train = diabetes_df[1:, 2]
X_test = diabetes_df[:, :2]

# Assuming 'diagnosis' column is the last column
X_test = diabetes_df.iloc[:, :-1].astype(float)
y_test = diabetes_df.iloc[:, -1]

# Use LabelEncoder to convert 'M' to 1 and 'B' to 0
label_encoder = LabelEncoder()
y_test = label_encoder.fit_transform(y_test)

# Initialize the SVM model (or replace with your preferred algorithm)
model = SVC(kernel='linear')

# Perform 6-fold cross-validation and get predicted labels
predicted_labels = cross_val_predict(model, X_train, y_train, cv=6)

# Calculate accuracy
accuracy = np.mean(cross_val_score(model, X_test, y_test, cv=6,
scoring='accuracy'))

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, predicted_labels)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
# Calculate confusion matrix
conf_matrix = confusion_matrix(y_train, predicted_labels)

# Plot confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

```python
# Display results
print("Predicted class of the input data:", predicted_labels)
print("Accuracy of the model:", accuracy)
print("Confusion Matrix:")
print(conf_matrix)
```