Faculty of computers and artificial intelligence

Cairo University

ML Assignment 2

Team members

1. Aliaa Mohamed Saad 20201120
2. Walaa Hassan Saad 20201217
3. Bassant Mahmoud Mohamed 20201043
4. Mohamed Ahmed Mahmoud 20200425
5. Mohamed Hamdy Mohamed 20200442

Question 1

- First experiment
  - Data Preprocessing

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

#-------------Data Preprocessing----------#
# Load the dataset
df = pd.read_csv('drug.csv')

# Check for missing values
print("\nMissing values in each column:")
print(df.isnull().sum())

# Handle missing values
# For numerical columns, replace missing values with the mean
numeric_columns = ['Age', 'Na_to_K']
for col in numeric_columns:
    df[col].fillna(df[col].mean(), inplace=True)

# For categorical columns, replace missing values with the most frequent value (mode)
categorical_columns = ['Sex', 'BP', 'Cholesterol']
for col in categorical_columns:
    df[col].fillna(df[col].mode()[0], inplace=True)

# Verify that missing values have been handled
print("\nMissing values after handling:")
print(df.isnull().sum())

# Encode categorical variables using Label Encoding
label_encoder = LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])
df['BP'] = label_encoder.fit_transform(df['BP'])
df['Cholesterol'] = label_encoder.fit_transform(df['Cholesterol'])

# Display the updated dataset
print("\nUpdated dataset:")
print(df.head())

#--------------END Data Preprocessing------------------#
```

Question 1

- First experiment
  - Data Preprocessing output

```
Missing values in each column:
Age            0
Sex            0
BP             2
Cholesterol    2
Na_to_K        1
Drug           0
dtype: int64

Missing values after handling:
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64

Updated dataset:
   Age  Sex  BP  Cholesterol    Na_to_K   Drug
0   23    0   0            0  25.355000  drugY
1   47    1   1            0  13.093000  drugC
2   47    1   1            0  10.114000  drugC
3   28    0   2            0  16.126126  drugX
4   61    0   1            0  18.043000  drugY
```

Question 1

- First experiment
    - Training and decision tree code

```python
#------------First experiment: Training and Testing with Fixed Train-Test Split Ratio:-------#
# Set the number of experiments
num_experiments = 5
# Initialize a list to store the accuracies and sizes of each experiment
results_fixed_split = []
# Repeat the experiment five times
for i in range(num_experiments):
    # Split the data into training and testing sets with a fixed ratio
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=i)
    # Create a Decision Tree model
    model_fixed_split = DecisionTreeClassifier(random_state=42)
    # Train the model on the training set
    model_fixed_split.fit(X_train, y_train)
    # Make predictions on the testing set
    y_pred_fixed_split = model_fixed_split.predict(X_test)
    # Calculate accuracy
    accuracy_fixed_split = accuracy_score(y_test, y_pred_fixed_split)
    # Get the size of the decision tree
    tree_size_fixed_split = model_fixed_split.tree_.node_count
    # Store the results
    results_fixed_split.append({
        'experiment': i + 1,
        'train_set_size': len(X_train),
        'test_set_size': len(X_test),
        'tree_size': tree_size_fixed_split,
        'accuracy': accuracy_fixed_split
    })
    # Print the results for each experiment
    print(f"\nExperiment {i + 1} (Fixed Split):")
    print(f"Train set size: {len(X_train)}, Test set size: {len(X_test)}")
    print(f"Decision Tree Size: {tree_size_fixed_split}")
    print(f"Decision Tree Accuracy: {accuracy_fixed_split:.4f}")
# Sort the results based on accuracy in descending order
sorted_results_fixed_split = sorted(results_fixed_split, key=lambda x: x['accuracy'], reverse=True)
# Select the best-performing model (the one with the highest accuracy)
best_model_fixed_split = sorted_results_fixed_split[0]
# Print details of the best-performing model for fixed split
print("\nBest Performing Model (Fixed Split):")
print(f"Experiment {best_model_fixed_split['experiment']}:")
print(f"Train set size: {best_model_fixed_split['train_set_size']}, Test set size: {best_model_fixed_split['test_set_size']}")
print(f"Decision Tree Size: {best_model_fixed_split['tree_size']}")
print(f"Decision Tree Accuracy: {best_model_fixed_split['accuracy']:.4f}")
#-----------------------END First experiment --------------------------#
```

Question 1

- First experiment
    - Final output

```
Experiment 1 (Fixed Split):
Train set size: 140, Test set size: 60
Decision Tree Size: 15
Decision Tree Accuracy: 1.0000

Experiment 2 (Fixed Split):
Train set size: 140, Test set size: 60
Decision Tree Size: 15
Decision Tree Accuracy: 0.9667

Experiment 3 (Fixed Split):
Train set size: 140, Test set size: 60
Decision Tree Size: 11
Decision Tree Accuracy: 0.9833

Experiment 4 (Fixed Split):
Train set size: 140, Test set size: 60
Decision Tree Size: 11
Decision Tree Accuracy: 0.9667

Experiment 5 (Fixed Split):
Train set size: 140, Test set size: 60
Decision Tree Size: 15
Decision Tree Accuracy: 0.9667

Best Performing Model (Fixed Split):
Experiment 1:
Train set size: 140, Test set size: 60
Decision Tree Size: 15
Decision Tree Accuracy: 1.0000
```

Question 1

- Second experiment
  - Training and Testing with a Range of Train-Test Split Ratios: the range of 30% to 70% (increments of 10%).

```
#------------Second experiment: Training and Testing with a Range of Train-Test Split Ratios:-------#

# Set the range of training set sizes
train_set_sizes = np.arange(0.3, 0.8, 0.1)

# Initialize lists to store statistics
train_set_sizes_report = []
mean_accuracies_report = []
max_accuracies_report = []
min_accuracies_report = []
mean_tree_sizes_report = []
max_tree_sizes_report = []
min_tree_sizes_report = []

# Initialize lists to store data for plots
accuracy_vs_size_data = {'Train Set Size': [], 'Mean Accuracy': []}
tree_size_vs_size_data = {'Train Set Size': [], 'Mean Tree Size': []}
```

Question 1

- Second experiment
  - For each iteration, run the experiment with five different random seeds.
  - Code for making this.

```python
# Repeat the experiment for each training set size
for train_size in train_set_sizes:
    # Initialize lists to store statistics for each random seed
    accuracies = []
    tree_sizes = []
    print(f"Iteration {iteration_counter} for Train Set Size {train_size * 100}%:")
    # Repeat the experiment with five different random seeds
    for i in range(num_experiments):
        # Split the data into training and testing sets with a variable ratio
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1 - train_size, random_state=i)

        # Create a Decision Tree model
        model_variable_split = DecisionTreeClassifier(random_state=i)

        # Train the model on the training set
        model_variable_split.fit(X_train, y_train)

        # Make predictions on the testing set
        y_pred_variable_split = model_variable_split.predict(X_test)

        # Calculate accuracy
        accuracy_variable_split = accuracy_score(y_test, y_pred_variable_split)

        # Get the size of the decision tree
        tree_size_variable_split = model_variable_split.tree_.node_count

        # Store accuracy and tree size for each seed
        accuracies.append(accuracy_variable_split)
        tree_sizes.append(tree_size_variable_split)

        # Print the results for each experiment
        print(f"\nRandom Seed: {i + 1}")
        print(f"Train set size: {len(X_train)}, Test set size: {len(X_test)}")
        print(f"Decision Tree Size: {tree_size_variable_split}")
        print(f"Decision Tree Accuracy: {accuracy_variable_split:.4f}")
```

Question 1

- Second experiment
    - For 0.3 , output

```
Iteration 1 for Train Set Size 30.0%:

Random Seed: 1
Train set size: 60, Test set size: 140
Decision Tree Size: 11
Decision Tree Accuracy: 0.9786

Random Seed: 2
Train set size: 60, Test set size: 140
Decision Tree Size: 17
Decision Tree Accuracy: 0.9357

Random Seed: 3
Train set size: 60, Test set size: 140
Decision Tree Size: 11
Decision Tree Accuracy: 0.9714

Random Seed: 4
Train set size: 60, Test set size: 140
Decision Tree Size: 11
Decision Tree Accuracy: 0.9857

Random Seed: 5
Train set size: 60, Test set size: 140
Decision Tree Size: 15
Decision Tree Accuracy: 0.9643
```

    - For 0.4 , output

```
Iteration 2 for Train Set Size 40.0%:

Random Seed: 1
Train set size: 80, Test set size: 120
Decision Tree Size: 15
Decision Tree Accuracy: 0.9750

Random Seed: 2
Train set size: 80, Test set size: 120
Decision Tree Size: 17
Decision Tree Accuracy: 0.9417

Random Seed: 3
Train set size: 80, Test set size: 120
Decision Tree Size: 11
Decision Tree Accuracy: 0.9750

Random Seed: 4
Train set size: 80, Test set size: 120
Decision Tree Size: 11
Decision Tree Accuracy: 0.9833

Random Seed: 5
Train set size: 80, Test set size: 120
Decision Tree Size: 15
Decision Tree Accuracy: 0.9583
------------------------------------
```

- For 0.5, output

```
Iteration 3 for Train Set Size 50.0%:

Random Seed: 1
Train set size: 100, Test set size: 100
Decision Tree Size: 15
Decision Tree Accuracy: 0.9600

Random Seed: 2
Train set size: 100, Test set size: 100
Decision Tree Size: 15
Decision Tree Accuracy: 0.9700

Random Seed: 3
Train set size: 100, Test set size: 100
Decision Tree Size: 11
Decision Tree Accuracy: 0.9900

Random Seed: 4
Train set size: 100, Test set size: 100
Decision Tree Size: 11
Decision Tree Accuracy: 0.9800

Random Seed: 5
Train set size: 100, Test set size: 100
Decision Tree Size: 15
Decision Tree Accuracy: 0.9800
```

- For 0.6, output

```
Iteration 4 for Train Set Size 60.00000000000001%:

Random Seed: 1
Train set size: 120, Test set size: 80
Decision Tree Size: 15
Decision Tree Accuracy: 0.9875

Random Seed: 2
Train set size: 120, Test set size: 80
Decision Tree Size: 15
Decision Tree Accuracy: 0.9750

Random Seed: 3
Train set size: 120, Test set size: 80
Decision Tree Size: 11
Decision Tree Accuracy: 0.9875

Random Seed: 4
Train set size: 120, Test set size: 80
Decision Tree Size: 11
Decision Tree Accuracy: 0.9750

Random Seed: 5
Train set size: 120, Test set size: 80
Decision Tree Size: 15
Decision Tree Accuracy: 0.9750
-----------------------------------
```

- For 0.7, output

```
------------------------------------
Iteration 5 for Train Set Size 70.00000000000001%:

Random Seed: 1
Train set size: 140, Test set size: 60
Decision Tree Size: 15
Decision Tree Accuracy: 1.0000

Random Seed: 2
Train set size: 140, Test set size: 60
Decision Tree Size: 15
Decision Tree Accuracy: 0.9667

Random Seed: 3
Train set size: 140, Test set size: 60
Decision Tree Size: 11
Decision Tree Accuracy: 0.9833

Random Seed: 4
Train set size: 140, Test set size: 60
Decision Tree Size: 11
Decision Tree Accuracy: 0.9667

Random Seed: 5
Train set size: 140, Test set size: 60
Decision Tree Size: 15
Decision Tree Accuracy: 0.9667
```

Faculty of computers and artificial intelligence

Cairo University

Question 1

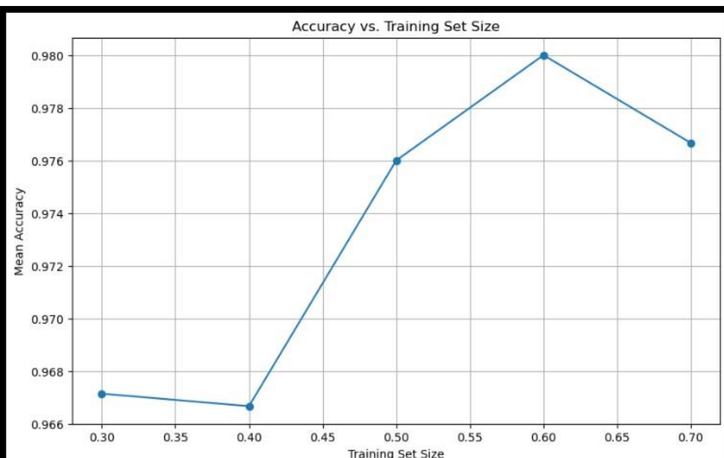
- Second experiment
  - Code for Experiment Report and Graph

```
# Display the report
print("\nExperiment Report:")
print(report_df)

# Save the report to a CSV file
report_df.to_csv('experiment_report.csv', index=False)

# Plot 1: Accuracy vs. Training Set Size
plt.figure(figsize=(10, 6))
plt.plot(accuracy_vs_size_data['Train Set Size'], accuracy_vs_size_data['Mean Accuracy'], marker='o')
plt.title('Accuracy vs. Training Set Size')
plt.xlabel('Training Set Size')
plt.ylabel('Mean Accuracy')
plt.grid(True)
plt.show()

# Plot 2: Mean Tree Size vs. Training Set Size
plt.figure(figsize=(10, 6))
plt.plot(tree_size_vs_size_data['Train Set Size'], tree_size_vs_size_data['Mean Tree Size'], marker='o', color='orange')
plt.title('Mean Tree Size vs. Training Set Size')
plt.xlabel('Training Set Size')
plt.ylabel('Mean Tree Size')
plt.grid(True)
plt.show()
```

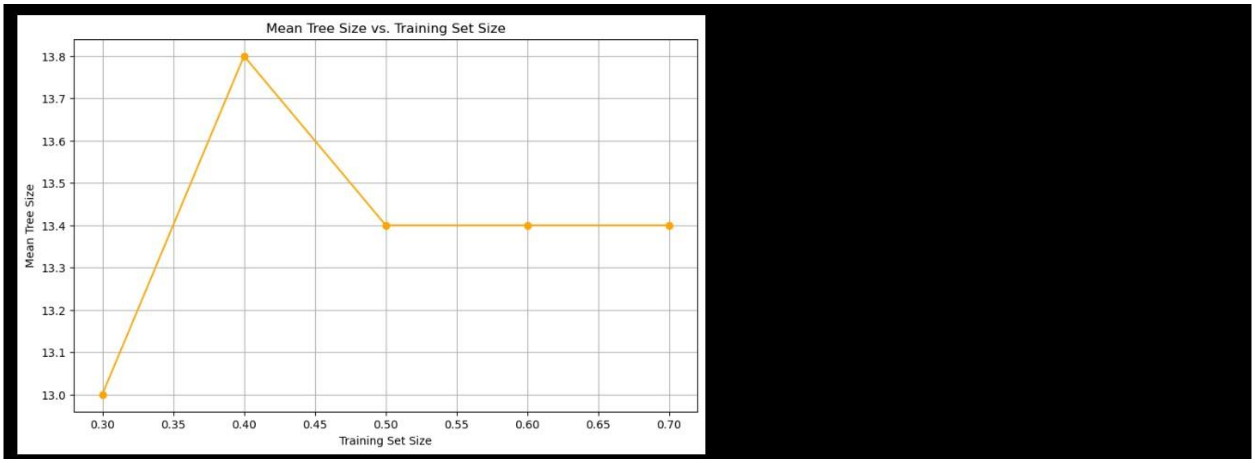
  - Report output

```
Experiment Report:
   Train Set Size  Mean Accuracy  Max Accuracy  Min Accuracy  Mean Tree Size  \
0             0.3       0.967143      0.985714      0.935714            13.0
1             0.4       0.966667      0.983333      0.941667            13.8
2             0.5       0.976000      0.990000      0.960000            13.4
3             0.6       0.980000      0.987500      0.975000            13.4
4             0.7       0.976667      1.000000      0.966667            13.4

   Max Tree Size  Min Tree Size
0             17             11
1             17             11
2             15             11
3             15             11
4             15             11
```

  - Polt for accuracy against training set size



- plot for the number of nodes in the final tree against training set size.



Question 2

- Data Preprocessing

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Load the dataset
data = pd.read_csv('diabetes.csv')

# Data Preprocessing

# Separate features and targets
X = data.drop('Outcome', axis=1)
y = data['Outcome']

# Split the data int training dataset and test data set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=22)

# Normalizing features using Min-Max Scaling
X_train_normalized = (X_train - X_train.min()) / (X_train.max() - X_train.min())
X_test_normalized = (X_test - X_test.min()) / (X_test.max() - X_test.min())

# Converting to numpy arrays to deal with it with code
training_data = np.column_stack((X_train_normalized.values, y_train.values))
test_data = np.column_stack((X_test_normalized.values, y_test.values))

print("Training Data after Preprocessing:")
print(pd.DataFrame(training_data, columns=data.columns).head())

print("\nTest Data after Preprocessing:")
print(pd.DataFrame(test_data, columns=data.columns).head())
```

Question 2

- Data Preprocessing output

```
Training Data after Preprocessing:
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0     0.882353  0.683417       0.573770       0.323232  0.130024  0.624579
1     0.352941  0.482412       0.000000       0.000000  0.000000  0.398990
2     0.235294  0.859296       0.590164       0.000000  0.000000  0.734007
3     0.000000  0.457286       0.655738       0.000000  0.000000  0.545455
4     0.058824  0.824121       0.672131       0.434343  0.079196  0.552189

   DiabetesPedigreeFunction       Age  Outcome
0                  0.032024  0.366667      1.0
1                  0.047822  0.116667      0.0
2                  0.171221  0.083333      1.0
3                  0.223313  0.100000      0.0
4                  0.112297  0.483333      0.0

Test Data after Preprocessing:
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0     0.214286  0.520202       0.631579       0.535714  0.298039  0.411326
1     0.071429  0.636364       0.526316       0.000000  0.000000  0.448584
2     0.214286  0.797980       0.666667       0.642857  0.480392  0.470939
3     0.357143  0.742424       0.684211       0.000000  0.000000  0.502235
4     0.000000  0.530303       0.561404       0.732143  0.278431  0.618480

   DiabetesPedigreeFunction       Age  Outcome
0                  0.310632  0.122449      0.0
1                  0.123959  0.530612      1.0
2                  0.369917  0.142857      1.0
3                  0.059775  0.897959      0.0
4                  0.037727  0.020408      0.0
```

Question 2

- Algorithm code

```python
# function which tests our knn algorithm
def evaluate_knn(k_values, training_data, test_data):
    # array to store all the accuaracies that used for calc avg later
    accuracies = []

    # loop over the k values
    for k in k_values:
        # see how many correct predictions our model make
        correct_predictions = 0

        # for each test instance in test data .. see whether it predict correctly or not
        for test_instance in test_data:
            predicted_class = knn_algorithm(training_data, test_instance[:-1], k)
            if predicted_class == test_instance[-1]:
                correct_predictions += 1

        accuracy = correct_predictions / len(test_data)
        accuracies.append((k, accuracy))

        print(f"k value: {k}")
        print(f"Number of correctly classified instances: {correct_predictions}")
        print(f"Total number of instances: {len(test_data)}")
        print(f"Accuracy: {accuracy * 100:.2f}%")

    # Output the average accuracy across all iterations
    average_accuracy = sum(accuracy for _, accuracy in accuracies) / len(accuracies)
    print(f"Average accuracy across all iterations: {average_accuracy * 100:.2f}%")

    return accuracies
```

Question 2

- Algorithm final output

```
k value: 2
Number of correctly classified instances: 162
Total number of instances: 231
Accuracy: 70.13%
k value: 3
Number of correctly classified instances: 167
Total number of instances: 231
Accuracy: 72.29%
k value: 4
Number of correctly classified instances: 166
Total number of instances: 231
Accuracy: 71.86%
k value: 5
Number of correctly classified instances: 165
Total number of instances: 231
Accuracy: 71.43%
k value: 6
Number of correctly classified instances: 167
Total number of instances: 231
Accuracy: 72.29%
Average accuracy across all iterations: 71.60%
```