Assignment 4: Data Preprocessing and Modeling

DSCI 6601: Data Science

Submission Instructions

The objective of this assignment is to analyze and justify the observed discrepancies in model accuracy when running different versions of the provided Python code. You are expected to explain why one version of the code might achieve perfect accuracy while another version does not. The Python code used for this assignment is available in the shared Google Drive folder under **Assignment4.ipynb**.

Instructions

Run the provided Python code cells, observe the model performance, and analyze the results. Discuss the potential reasons behind perfect accuracy in one case and lower accuracy in another. Identify any issues or overfitting concerns and justify your findings.

Detailed Analysis Questions

To thoroughly understand the dataset and model performance, answer the following questions with detailed explanations:

1. Number of Features in the Dataset

Analyze and explain the number of features in the dataset. Specifically, you need to address the following outputs and justify any discrepancies observed:

- Number of columns in the dataset: 22,278.
- As part of dimension of X we get : 49,112.
- Number of unique numeric feature columns: 14,236.

Explanation and Justification

1. Why do we observe a difference between the total number of numeric feature columns and the number of unique numeric feature columns? Explain the

potential reasons behind duplicate or redundant features and their impact on data analysis.

2. Which of these counts should be relied on for building machine learning models, and why? Provide a detailed justification for selecting the most appropriate count of feature columns for model training.

3. Discuss the implications of using non-unique columns in a dataset. How might this affect model performance, and what steps would you recommend to handle such a situation?

Analysis Task: Justify Discrepancies in Model Performance

As part of this assignment, analyze and justify the observed discrepancies in the performance of the Decision Tree Classifier between the first and second code cells.

Questions for Analysis

1. **Explain why the first code cell results in a perfect accuracy of 1.00 and the following confusion matrix:**

   ```
   [[15  0]
   [ 0 19]]
   ```

   Discuss whether this perfect accuracy could be due to overfitting, data distribution, or specific properties of the dataset and training/test split.

2. **Compare and explain why the second code cell results in a lower accuracy of 0.76 with the following confusion matrix:**

   ```
   [[13  3]
   [ 5 13]]
   ```

   Identify factors that might contribute to the decrease in accuracy, such as data complexity, randomness in the train-test split, or preprocessing differences.

3. **Provide a detailed justification** for the discrepancies between the two results, including an analysis of feature distributions, potential overfitting, and the impact of different random states on the dataset split.

Submission Instructions

Your submission should include the following:

- Comprehensive responses to all the tasks outlined above, accompanied by the Python code and outputs from your Jupyter Notebook.
- Clear and concise explanations of each step taken, with well-documented text and comments included in your Jupyter Notebook for every step.
- A thorough analysis of the results within the Jupyter Notebook.

Python Code Cell 1

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

dataset_path = '/content/drive/My Drive/data_code_DSCI6601/data/GSE44861'  # Update the path

# Check if the file exists
if not os.path.exists(dataset_path):
print(f"File not found: {dataset_path}")
else:
# Read the dataset from a pickle file
data = pd.read_pickle(dataset_path)

# Print dataset specifications
print("Dataset Specifications:")
print(f"Number of samples: {data.shape[0]}")
print(f"Number of columns: {data.shape[1]}")
print(f"Number of labels: {len(data.select_dtypes(include=['object', 'category']).columns)}")

# Print the dimensions of the data
print(f"Dimensions of the dataset: {data.shape}")

# Use iloc to access the columns and print the number of columns
number_of_columns = data.iloc[0, :].size
print(f"Number of columns in the dataset: {number_of_columns}")

# Check for normalization requirement (if any feature has variance > 1, normalization is need
feature_columns = data.select_dtypes(include=[np.number]).columns.tolist()
unique_feature_columns = pd.Index(feature_columns).unique()

# Print the total number of feature columns
print(f"Total number of numeric feature columns: {len(feature_columns)}")
print(f"Number of unique numeric feature columns: {len(unique_feature_columns)}")

X = data[feature_columns]
target_column = data.columns[-1]
y = data[target_column]
```

```
# Print dimensions of X and y
print(f"\nDimensions of X: {X.shape}")
print(f"Dimensions of y: {y.shape}")

print("\nChecking if normalization is required...")
if X.var().max() > 1:
print("Normalization is required.")
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)
else:
print("Normalization is not required.")
X_normalized = X.values

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y, test_size=0.3, random_st

# Train a Decision Tree Classifier without PCA
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Print the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"\nDecision Tree Classifier Accuracy: {accuracy:.2f}")

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)

# Plot the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.u
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

Python Code Cell 2

```
# dataset_path = '/content/drive/My Drive/data_code_DSCI6601/data/GSE44861' # Update the pat
```

```
# Check if the file exists
if not os.path.exists(dataset_path):
print(f"File not found: {dataset_path}")
else:
# Read the dataset from a pickle file
data = pd.read_pickle(dataset_path)

# Check for normalization requirement (if any feature has variance > 1, normalization is need
feature_columns = data.select_dtypes(include=[np.number]).columns.tolist()
features = data[feature_columns]

# Assuming the last column contains labels
target_column = data.columns[-1]

# Separate features (X) and target labels (y)
X = data.iloc[:, :-1]  # All columns except the last column
y = data[target_column]  # The last column is assumed to be the target label

# Print the number of samples per label type
label_counts = y.value_counts()
print("\nNumber of samples per label type:")
print(label_counts)

print("\nChecking if normalization is required...")
if features.var().max() > 1:
print("Normalization is required.")
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)
else:
print("Normalization is not required.")
X_normalized = X.values  # No normalization applied

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y, test_size=0.3, random_st

# Train a Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Print the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"\nDecision Tree Classifier Accuracy: {accuracy:.2f}")
```

```
# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)

# Plot the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.u
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

# Feature importance from the Decision Tree model
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]

print("\nFeature Importance:")
max_features = min(100, X_train.shape[1])

# Plot feature importances for the first 100 features
plt.figure(figsize=(15, 5))
plt.title('Feature Importances (Top 100 Features)')
plt.bar(range(max_features), importances[indices][:max_features], color="r", align="center")
plt.xticks(range(max_features), indices[:max_features], rotation='vertical')
plt.xlim([-1, max_features])
plt.xlabel('Feature Indices')
plt.ylabel('Importance')
plt.show()

# Number of features used
n_features_used = np.sum(importances > 0)
print(f"Number of features used to build the model: {n_features_used}")
```