Machine Learning HW2

Yara Alfouzan
443203087

DATA SET DESCRIPTION:

Customer Purchasing Behaviors dataset. The specific features included are userr_id, age, annual_income, purchase_amount, purchase_frequency, region, and loyalty_score. This dataset can be used to analyze and predict customer purchasing behaviors. In this hw it'll be used to predict whether a customer makes a high or low frequency of purchases.

Clear screenshots of the code, evaluations and outputs:

Homework_2(2).ipynb

File Edit View Insert Runtime Tools Help

Import Necessary Libraries

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score
```

Dataset Loading, Exploration & Preprocessing

```python
[3] from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
[4] # 1. Load the dataset

path="/content/drive/MyDrive/newCustomer_Purchasing_Behaviors(3).csv"
df=pd.read_csv(path)


# 2. Examine the dataset and identify the types of features (e.g., numerical, categorical).
print(df.head())
print(df.info())
print ("\n numerical features ", df.select_dtypes(include=['int64', 'float64']).columns)
print ("\n categorical features ", df.select_dtypes(include=['object']).columns)

# One-hot encode categorical variables
# 3. Convert all categorical features to numerical format using one-hot encoding.

df_onehot = pd.get_dummies(df.select_dtypes(include=['object']), dtype=int)
df_onehot = pd.concat([df_onehot, df.select_dtypes(include=['int64', 'float64'])], axis=1)

# 4. Discretizing Purchase Frequency into High and Low categories (based on mean)
```

0s completed at 2:24 PM

```python
mean_purchase_frequency = df_onehot['purchase_frequency'].mean()
df_onehot['purchase_frequency_category'] = pd.cut(df['purchase_frequency'], bins=[0, mean_purchase_frequency, float('inf')], labels=['Low', 'High'])
print(df_onehot.head())

# 5. Select purchase_frequency_category as the target variable (y) and use all other features except (purchase_frequency) as the input features (X).

y = df_onehot['purchase_frequency_category']
X = df_onehot.drop(['purchase_frequency', 'purchase_frequency_category'], axis=1)

# 6. Split the dataset into training (80%) and testing (20%) sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
   user_id  age  annual_income  purchase_amount  loyalty_score region  \
0        1   25          45000              200            4.5  North
1        2   34          55000              350            7.0  South
2        3   45          65000              500            8.0   West
3        4   22          30000              150            3.0   East
4        5   29          47000              220            4.8  North

   purchase_frequency
0                  12
1                  18
2                  22
3                  10
4                  13
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 238 entries, 0 to 237
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   user_id             238 non-null    int64
 1   age                 238 non-null    int64
 2   annual_income       238 non-null    int64
 3   purchase_amount     238 non-null    int64
 4   loyalty_score       238 non-null    float64
 5   region              238 non-null    object
 6   purchase_frequency  238 non-null    int64
dtypes: float64(1), int64(5), object(1)
memory usage: 13.1+ KB
None

 numerical features  Index(['user_id', 'age', 'annual_income', 'purchase_amount', 'loyalty_score',
       'purchase_frequency'],
      dtype='object')
```

0s completed at 2:24 PM

```
categorical features  Index(['region'], dtype='object')
   region_East  region_North  region_South  region_West  user_id  age  \
0            0             1             0            0        1   25
1            0             0             1            0        2   34
2            0             0             0            1        3   45
3            1             0             0            0        4   22
4            0             1             0            0        5   29

   annual_income  purchase_amount  loyalty_score  purchase_frequency  \
0          45000              200            4.5                  12
1          55000              350            7.0                  18
2          65000              500            8.0                  22
3          30000              150            3.0                  10
4          47000              220            4.8                  13

  purchase_frequency_category
0                         Low
1                         Low
2                        High
3                         Low
4                         Low
```

Logistic Regression Using Normal Equation

[5]

```python
# 1. Implement logistic regression using the normal equation formula.

# Sigmoid function for logistic regression
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Function to apply the normal equation to logistic regression
def logistic_regression_normal_equation(X, y):

    # Convert y to numerical values (0 and 1)
    y_numeric= np.where(y== 'High', 1, 0)

    # Add bias term (intercept) to feature matrix
    X_augmented = np.hstack((np.ones((X.shape[0], 1)), X))

    # Calculate theta using the normal equation
    theta = np.linalg.inv(X_augmented.T @ X_augmented) @ X_augmented.T @ y_numeric
    return theta
```

✓ 0s completed at 2:24 PM

```python
    return theta

# Function to make predictions using learned parameters
def predict(X, theta):
    # Add bias term (intercept) to test feature matrix
    X_augmented = np.hstack((np.ones((X.shape[0], 1)), X))

    # Calculate predictions using sigmoid
    return sigmoid(X_augmented @ theta)

# Train logistic regression using the normal equation
theta = logistic_regression_normal_equation(X_train.to_numpy(), y_train)

# 2. Predict probabilities on the test set
y_pred_prob = predict(X_test.to_numpy(), theta)

# Convert probabilities to binary predictions (0 or 1)
y_pred = (y_pred_prob >= 0.5).astype(int)

# Convert y_test to numerical values for accuracy and F1 score
y_test_numeric = np.where(y_test == 'High', 1, 0)
```

✓ 0s completed at 2:24 PM

```python
# 3. Calculate accuracy and F1 score
accuracy = accuracy_score(y_test_numeric, y_pred)
f1 = f1_score(y_test_numeric, y_pred)

print("Accuracy:", accuracy)
print("F1 Score:", f1)
```

```
Accuracy: 0.5625
F1 Score: 0.72
```

Logistic Regression Using Sklearn Library

```python
# 1. Implement logistic regression model using the LogisticRegression from sklearn.
model = LogisticRegression()
model.fit(X_train, y_train)

# 2. Predict using sklearn library
y_pred_sklearn = model.predict(X_test)

# 3. Calculate Accuracy and F1 score
accuracy_sklearn = accuracy_score(y_test, y_pred_sklearn)
f1_sklearn = f1_score(y_test, y_pred_sklearn, pos_label='High')

print("Accuracy (Sklearn):", accuracy_sklearn)
print("F1 Score (Sklearn):", f1_sklearn)
```

```
Accuracy (Sklearn): 0.9375
F1 Score (Sklearn): 0.9433962264150944
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Feature Engineering and Re-evaluation

```python
# 1. Create a new feature by squaring the purchase amount feature to capture a potential non-linear relationship.
df_onehot['purchase_amount_squared'] = df_onehot['purchase_amount'] ** 2

# 2. Re-split the data to include the new feature.
y = df_onehot['purchase_frequency_category']
X = df_onehot.drop(['purchase_frequency', 'purchase_frequency_category'], axis=1)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. Fit a new sklearn logistic regression model using the original features plus the new quadratic feature.
#model_sklearn_quadratic = LogisticRegression(max_iter=1000)
model_sklearn_quadratic = LogisticRegression()
model_sklearn_quadratic.fit(X_train2, y_train2)

# 4. Predict the test data and calculate Accuracy and F1 score for the new sklearn model.

y_pred_sklearn_quadratic = model_sklearn_quadratic.predict(X_test2)
accuracy_sklearn = accuracy_score(y_test2, y_pred_sklearn_quadratic)
f1_sklearn = f1_score(y_test2, y_pred_sklearn_quadratic, pos_label='High')

print("Accuracy (Scikit-learn):", accuracy_sklearn)
print("F1 Score (Scikit-learn):", f1_sklearn)
```

```
Accuracy (Scikit-learn): 1.0
F1 Score (Scikit-learn): 1.0
```

A table displaying printed results (e.g., model parameters and/or predicted values):

```
numerical features  Index(['user_id', 'age', 'annual_income', 'purchase_amount', 'loyalty_score',
       'purchase_frequency'],
      dtype='object')

categorical features  Index(['region'], dtype='object')
```

F1 score and accuracy.

Logistic regression using normal eq:

```
Accuracy: 0.5625
F1 Score: 0.72
```

Logistic regression using Sklearn:

```
Accuracy (Sklearn): 0.9375
F1 Score (Sklearn): 0.9433962264150944
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
```

Logistic regression using Sklearn after feature engineering:

```
Accuracy (Scikit-learn): 1.0
F1 Score (Scikit-learn): 1.0
```

Part5: Observation & Comparison

1.Explain why one-hot encoding is necessary in the context of linear regression

Linear regression models work best with numerical data. One-hot encoding is necessary to convert categorical data into a numerical format that the model can understand;This prevents the model from misinterpreting the categorical data as ordinal, ensuring that the model treats each category as a distinct entity without any implied order or relationship between them.

1. Compare the performance of the model with and without the new polynomial feature

The model with the squared feature performs better than the model without it.

Accuracy: Increased from 0.9375 to 1.0

F1 Score: Increased from 0.9434 to 1.0

This indicates that the squared feature likely captures a non-linear relationship between purchase amount and purchase frequency, which the original model could not capture. As a result, the model with the squared feature is able to make perfect predictions on the test set;but observing perfect accuracy and F1 scores, might indicate overfitting.

2. Provide observations on which model performs better and why.

The model implemented with scikit-learn performs significantly better than the model using the normal equation.

Accuracy: 0.9375 (scikit-learn) vs. 0.5625 (normal equation)

F1 Score: 0.9434 (scikit-learn) vs. 0.72 (normal equation)

This difference in performance could be due to that Scikit-learn's LogisticRegression includes regularization by default; moreover it likely uses more robust optimization algorithms that are less susceptible to numerical issues rather than solving normal equation directly.