EE658/758 Machine Learning in Engineering

Assignment #1: Linear Regression
Due Date: Monday, February 12th, 2024

Analyzing Expenditures on Insured Customers for an Insurance Company

In this project, our focus will be on examining the data related to customers with insurance coverage. We aim to utilize a dataset comprising the profiles of insured clients to forecast the financial outlay incurred by the insurance company on these customers.

The Dataset

The insured customers' data is in a csv file. It has information consisting of:

1. Age (integer)
2. Sex (categorical variable): “female”, “male”
3. BMI (float)
4. Children (integer)
5. Smoker (categorical variable): “yes”, “no”
6. Region (categorical variable): “northeast”, “northwest”, “southeast”, “southwest”
7. Expenses (float): numerical value representing the label or target variable

1. **Data Preprocessing:**
    - Load the dataset and print a sample number of rows.
    - Find the number of rows with any missing values. Remove any row with a missing value.
    - Convert 'Gender', 'Smoker', and 'Region' into numerical values suitable for regression analysis (e.g., using one-hot encoding for 'region' and binary encoding for 'sex' and 'smoker').
    - Normalize the features using Min-Max scaling.
2. **Splitting the Data:**
    - Divide the data into “features” and “target” subsets.
    - Split the data into training and testing subsets (commonly a 70/30, 75/25, or 80/20 split).
3. **Gradient Descent Implementation:**
    - Implement the gradient descent algorithm (without the Scikit-Learn library) to find the regression line. Initialize parameters randomly and update them iteratively to minimize the loss function. Record the loss value for each iteration:

```
X_b = np.c_[np.ones((X_train.shape[0], 1)), X_train]
m = X_train.shape[0]  # number of data points
n = X_train.shape[0]   # number of features

alpha = 0.01              # Learning rate
n_iterations = 10000   # Number of iterations
W = np.random.randn(X_train.shape[1]+1,1)  # Weight matrix

loss = []                      # Loss value for each iteration
for iteration in range(n_iterations):
   gradients = 1/m * X_b.T.dot(X_b.dot(W) - y_train)
   W = W - alpha * gradients
   predictions = X_b.dot(W)
   loss.append(mean_squared_error(y_train, predictions))
```

- Show the coefficients and intercept of the model.
- Modify the code to implement the exponential decay method for the learning rate.
- Plot the loss values as a function of the number of iterations for the constant and decaying learning rates.

4. **Model Evaluation:**
    - Predict the expenses for the testing dataset using the trained model.
    - Compute the Mean Absolute Error (MAE) and Mean Squared Error (MSE) of the predictions.
    - Plot a histogram of the error distribution.
5. **Learning Rate Analysis:**
    - Demonstrate the effect of varying the learning rate on the convergence of the gradient descent algorithm.
6. **Scikit-learn Implementation:**
    - Repeat the regression using the **linear_model.LinearRegression** class from scikit-learn.
    - Compute MAE and MSE for comparison.
7. **Normal Equation Implementation:**
    - Use the normal equation method to find the regression line directly.
    - Compare the MAE and MSE with previous methods.
8. **Comparison:**
    - Compare the three solutions in terms of MAE, MSE, and computational efficiency.

Notes:

1. **Reading a CSV file with the Pandas library:**

```
import pandas as pd

# Replace 'your_file.csv' with the path to your CSV file
df = pd.read_csv('your_file.csv')
```

2. **Converting categorical variables into numeric values:**

Let's consider an example where we have a categorical variable with three distinct values, say "Red", "Blue", and "Green". We can convert this categorical variable into numeric format using the get_dummies() function from the Pandas library in Python.

```
import pandas as pd
data = {'Age': [35, 22, 28, 42, 21],
      'Color': ['Red', 'Blue', 'Green', 'Red', 'Green']}
df = pd.DataFrame(data)
# Convert the categorical variable into dummy/indicator variables
dummies = pd.get_dummies(df['Color'])
# The new DataFrame with dummy variables
print(dummies)
print(df)
df2 = pd.get_dummies(df, columns=['Color'])
print(df2)
```

A categorical variable with two distinct values, say "Red" and "Blue":

```
data = {'Age': [35, 22, 28, 42, 21],
      'Color': ['Red', 'Blue', 'Blue', 'Red', 'Red']}
df = pd.DataFrame(data)
df['Color_encoded'] = df['Color'].map({'Red': 0, 'Blue': 1})
print(df)

df = df.drop(['Color'], axis=1)
print(df)
```

3. **Normalizing the data**

Min-Max scaling is a technique used to normalize the features in your data. It scales the range of features to be between 0 and 1.

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np
# Example data
data = np.array([[100, 0.5],
                 [80, 0.1],
                 [120, 0.3]])

# Create the MinMaxScaler object
scaler = MinMaxScaler()
# Fit the scaler to the data and transform it
scaled_data = scaler.fit_transform(data)
# The scaled data
print(scaled_data)
```

4. Splitting the Data

Dividing data into features and target, and then splitting it into training and testing sets:

```
import pandas as pd
from sklearn.model_selection import train_test_split
# Example DataFrame
data = {
    'feature1': [1, 2, 3, 4, 5],
    'feature2': [10, 20, 30, 40, 50],
    'feature3': [100, 200, 300, 400, 500],
    'target': [0, 1, 0, 1, 0]
}
df = pd.DataFrame(data)
# Dividing the data into features and target
X = df.drop('target', axis=1)  # Features
y = df['target']  # Target
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# The resulting sets
print("X_train:\n", X_train)
print("\nX_test:\n", X_test)
print("\ny_train:\n", y_train)
print("\ny_test:\n", y_test)
```