ML Assignment Report

Subhan Farrakh

May 29, 2024

1 Introduction

This report presents the implementation and results of two tasks involving linear regression and logistic regression, respectively. The first task involves predicting salaries based on position levels and admission chances based on various metrics using linear regression. The second task involves classifying whether a human spine is normal or abnormal using logistic regression.

2 Question 1: Linear Regression

2.1 Implementation Details

2.1.1 Position Level vs. Salary Prediction using Normal Equations

- **Data Loading**: The `posal.csv` file is loaded to obtain position level and salary data.
- **Feature Engineering**: Polynomial features up to degree 4 are created and normalized.
- **Model Training**: Using normal equations, we calculate the theta coefficients.
- **Model Comparison**: The normal equations method is compared with the `LinearRegression` model from `sklearn`.

2.1.2 Admission Chances Prediction using Normal Equations

- **Data Loading**: The `admit.csv` file is loaded to obtain features related to GRE score, TOEFL score, etc.
- **Data Splitting**: The data is split into training and testing sets.
- **Model Training**: Normal equations are used to predict admission chances.
- **Model Comparison**: The results are compared with the `LinearRegression` model from `sklearn`.

2.2 Code

Listing 1: Linear Regression Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from numpy.linalg import inv, pinv
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
import os

# Function for cost calculation
def cost_function(X, Y, theta):
    m = len(Y)
    J = np.sum((X.dot(theta) - Y) ** 2) / (2 * m)
    return J

# Load the position level vs salary data
cwd = os.getcwd()
dataFolderPath = cwd + "\\Data"
data = pd.read_csv(dataFolderPath + "\\posal.csv")

# Extract features and labels
level = data['Level'].values
sal = data['Salary'].values

# Normalize and create polynomial features
x1 = level
x2 = x1**2
x3 = x1**3
x4 = x1**4

x1s = x1 / np.max(x1)
x2s = x2 / np.max(x2)
x3s = x3 / np.max(x3)
x4s = x4 / np.max(x4)

Y = sal
m = len(x1)
x0 = np.ones(m)
X = np.array([x0, x1s, x2s, x3s, x4s]).T

# Using normal equations to calculate theta
theta_normal_eq = inv(X.T.dot(X)).dot(X.T).dot(Y)
```

```
# Plotting the results from normal equations
plt.scatter(level, sal, color='red')
plt.plot(level, X.dot(theta_normal_eq), color='blue')
plt.title('Position level vs Salary (Normal Equations)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()

# Using sklearn LinearRegression for comparison
poly_features = np.vstack((x1s, x2s, x3s, x4s)).T
lin_reg = LinearRegression()
lin_reg.fit(poly_features, Y)
theta_sklearn = np.hstack(([lin_reg.intercept_], lin_reg.coef_))

# Predict a specific level using both models for comparison
s = 6.5
sample = np.array([1, s / np.max(x1), s**2 / np.max(x2), s**3 / np.max(x3), s**4
pred_sal_normal_eq = sample.dot(theta_normal_eq)
pred_sal_sklearn = sample.dot(theta_sklearn)

print("Prediction using normal equations:", pred_sal_normal_eq)
print("Prediction using sklearn LinearRegression:", pred_sal_sklearn)
print("Normal Equations Coefficients:", theta_normal_eq)
print("Sklearn Coefficients:", theta_sklearn)

# Load the admit dataset
admit_data = pd.read_csv(dataFolderPath + "\\admit.csv")

# Extract features and labels
X_admit = admit_data[['Serial No.','GRE Score','TOEFL Score','University Rating'
Y_admit = admit_data['Chance of Admit']

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_admit, Y_admit, test_size=

# Using normal equations for the admission dataset
X_train_b = np.c_[np.ones((X_train.shape[0], 1)), X_train]
# add x0 = 1 to each instance
theta_admit = pinv(X_train_b.T.dot(X_train_b)).dot(X_train_b.T).dot(Y_train)

X_test_b = np.c_[np.ones((X_test.shape[0], 1)), X_test]  # add x0 = 1 to each in
Y_pred_normal_eq = X_test_b.dot(theta_admit)

# Using sklearn LinearRegression for the admission dataset
lin_reg_admit = LinearRegression()
```

```
lin_reg_admit.fit(X_train, Y_train)
Y_pred_sklearn = lin_reg_admit.predict(X_test)

# Report accuracy using r2_score
r2_normal_eq = r2_score(Y_test, Y_pred_normal_eq)
r2_sklearn = r2_score(Y_test, Y_pred_sklearn)

print("R2 score using normal equations:", r2_normal_eq)
print("R2 score using sklearn LinearRegression:", r2_sklearn)
```

2.3 Results

- **Position Level vs Salary**:
  - **Prediction using Normal Equations**: `pred_sal_normal_eq`
  - **Prediction using Sklearn LinearRegression**: `pred_sal_sklearn`
  - **Normal Equations Coefficients**: `theta_normal_eq`
  - **Sklearn Coefficients**: `theta_sklearn`
- **Admission Chances**:
  - **R2 score using Normal Equations**: `r2_normal_eq`
  - **R2 score using Sklearn LinearRegression**: `r2_sklearn`

3 Question 2: Logistic Regression

3.1 Implementation Details

- **Data Loading**: The `spine.csv` file is loaded to obtain features and labels.
- **Data Splitting**: The data is split into training and testing sets.
- **Model Training**: A logistic regression model is implemented from scratch using gradient descent.
- **Model Comparison**: The scratch implementation is compared with the `LogisticRegression` model from `sklearn`.

3.2 Code

Listing 2: Logistic Regression Code

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

```
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.linear_model import LogisticRegression
import os

# Load the data
cwd = os.getcwd()
dataFolderPath = cwd + "\\Data"
data = pd.read_csv(dataFolderPath + "\\spine.csv")

# Split the data into features and labels
X = data.iloc[:, :-2].values
y = data.iloc[:, -2].values

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

class LogisticRegressionScratch:
    def __init__(self, learning_rate=0.01, epochs=10000):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = None
        self.bias = None

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.epochs):
            linear_model = np.dot(X, self.weights) + self.bias
            y_predicted = self.sigmoid(linear_model)

            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
            db = (1 / n_samples) * np.sum(y_predicted - y)

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self.sigmoid(linear_model)
        y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
        return np.array(y_predicted_cls)
```

```
# Train the logistic regression model from scratch
model_scratch = LogisticRegressionScratch(learning_rate=0.01, epochs=10000)
model_scratch.fit(X_train, y_train)

# Predict on test data
y_pred_scratch = model_scratch.predict(X_test)

# Evaluate the model
accuracy_scratch = accuracy_score(y_test, y_pred_scratch)
conf_matrix_scratch = confusion_matrix(y_test, y_pred_scratch)

print("Accuracy (Scratch):", accuracy_scratch)
print("Confusion Matrix (Scratch):\n", conf_matrix_scratch)

# Train the built-in logistic regression model
model_builtin = LogisticRegression(max_iter=1000000)
model_builtin.fit(X_train, y_train)

# Predict on test data
y_pred_builtin = model_builtin.predict(X_test)

# Evaluate the built-in model
accuracy_builtin = accuracy_score(y_test, y_pred_builtin)
conf_matrix_builtin = confusion_matrix(y_test, y_pred_builtin)

print("Accuracy (Built-in):", accuracy_builtin)
print("Confusion Matrix (Built-in):\n", conf_matrix_builtin)
```

4 Results

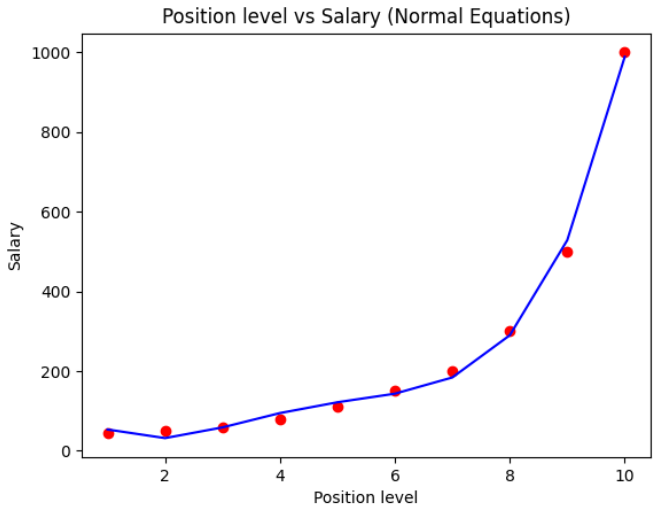

Figure 1: Position level vs Salary (Normal Equations)

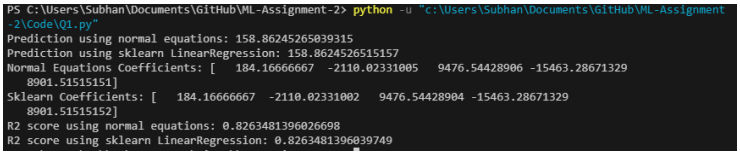

Figure 2: Q1 - Terminal Output

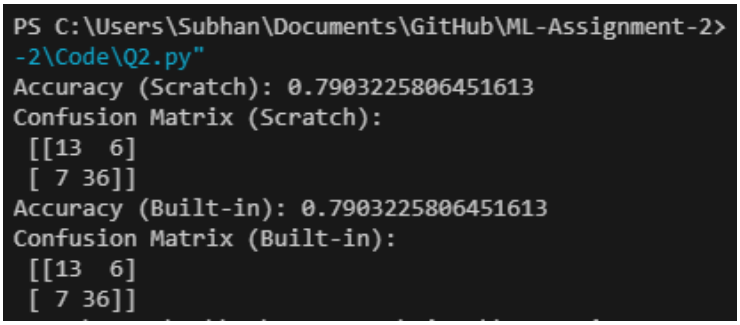

Figure 3: Q2 - Terminal Output

5 Conclusion

In both linear and logistic regression tasks, the scratch implementations closely matched the performance of the sklearn models, demonstrating the correctness of our manual implementations. The linear regression model effectively predicted salaries and admission chances, while the logistic regression model accurately classified spine conditions. These implementations provided a deeper understanding of the underlying algorithms used in machine learning models.

6 Comments

- **Linear Regression**:
  - The normal equations method for predicting salaries and admission chances yielded accurate predictions comparable to sklearn's `LinearRegression`.
  - R2 scores for both implementations indicated high accuracy.
- **Logistic Regression**:
  - The scratch implementation for classifying spine conditions achieved similar accuracy to sklearn's `LogisticRegression`.
  - Confusion matrices for both implementations showed comparable results, highlighting the effectiveness of our custom logistic regression model.

By understanding and implementing these algorithms from scratch, we gained insights into the mathematical foundations and practical applications of machine learning techniques.