Fit Data Assignment

January 25, 2019

Author: Brenten Canfield, Spring 2019

Directly below is the code used for model creation and analysis. Below that is the actual analysis of the results.

```
In [1]: import numpy as np
        import pandas as pd
        import sklearn.model_selection
        import sklearn.preprocessing
        import sklearn.linear_model
        import sklearn.metrics
        from prettytable import PrettyTable
        from joblib import dump, load

        def loadData( fileName ):
            return pd.read_csv(fileName, index_col=0)

        def separatePredictorsAndLabels( data ):
            predictors = data.drop("labels", axis=1)
            labels = data["labels"].copy()
            return predictors, labels

        def getScaler(predictors):
            scaler = sklearn.preprocessing.StandardScaler()
            scaler.fit(predictors.astype("float64"))
            return scaler
        def scaleData(data, scaler):
            data = data.astype("float64")
            data = scaler.transform(data)
            return data

        def trainModel(predictors, labels):
            model = sklearn.linear_model.LinearRegression()
            model.fit(predictors, labels)
            return model

        def errorTest(predictors, labels, model):
            predictedLabels = model.predict(predictors)
```

```python
    meanSquaredError = sklearn.metrics.mean_squared_error(labels, predictedLabels)
    rootMeanSquaredError = np.sqrt(meanSquaredError)
    return meanSquaredError, rootMeanSquaredError

def displayResults(model, MSE, RMSE):
    headers = ["X", "Theta", "Theta Value"]
    table = PrettyTable(headers)
    for i in range(len(model.coef_)):
        x = "x_" + str(i+1)
        theta = "Theta_" + str(i+1)
        value = model.coef_[i]
        table.add_row([x, theta, value])
    print(table)
    print("Y-intercept:", model.intercept_)
    print("Mean Squared Error:", MSE)
    print("Root Mean Squared Error:", RMSE)

def saveModel(model, fileName):
    dump(model, fileName)
def loadModel(fileName):
    return load(fileName)

def report( ):
    trainData = loadData( "train-data.csv" )
    trainXraw, trainY = separatePredictorsAndLabels( trainData )
    scaler = getScaler(trainXraw)
    trainX = scaleData(trainXraw, scaler)
    model = trainModel( trainX, trainY )

    testData = loadData( "test-data.csv" )
    testXraw, testY = separatePredictorsAndLabels( testData )
    testX = scaleData(testXraw, scaler)
    MSE, RMSE = errorTest( testX, testY, model )

    displayResults(model, MSE, RMSE)
    return model

def reportWithModel(model):
    trainData = loadData( "train-data.csv" )
    trainXraw, trainY = separatePredictorsAndLabels( trainData )
    scaler = getScaler(trainXraw)

    testData = loadData( "test-data.csv" )
    testXraw, testY = separatePredictorsAndLabels( testData )
    testX = scaleData(testXraw, scaler)
    MSE, RMSE = errorTest( testX, testY, model )

    displayResults(model, MSE, RMSE)
```

```
In [2]: # display report and get model
        model = report()
        # save model with joblib
        saveModel(model, "linear.joblib")
```

```
+-----+---------+---------------------+
|  X  |  Theta  |     Theta Value     |
+-----+---------+---------------------+
| x_1 | Theta_1 | -18.326866848615648 |
| x_2 | Theta_2 | -60.488285226256664 |
| x_3 | Theta_3 | -0.5002880659381503 |
| x_4 | Theta_4 | 0.44341680815857387 |
| x_5 | Theta_5 | -16.201565565095663 |
| x_6 | Theta_6 | -44.768884944380105 |
| x_7 | Theta_7 | -205.20882017371724 |
+-----+---------+---------------------+
Y-intercept: -1307.7683466494066
Mean Squared Error: 465.8066694026524
Root Mean Squared Error: 21.582554746893436
```

Previous predictions on feature to label correlation

- x_1: Mostly uniform, with little to no correlation
- x_2: A slight negative correlation
- x_3: Similar to X1, with little to no correlation
- x_4: Mostly a clustered blob, no correlation

- x_5: Similar to X1 and X3, wwith little to no correlation

- x_6: A blob like cluster with a slight negative correlation
- x_7: A negatively correlated line, with noise along line

Actual importance analysis

The table above provides the actual relevance of each feature for label prediction. The magnitude of the each theta value gives us how import each feature is. Furthermore, a negative theta value shows a negative correlation, and vise-versa. Below is my analysis of each feature, comparing my visual estimate to the actual result of theta values.

- x_1: There is more correlation than previously predicted. There is a negative correlation.
- x_2: This is the second most relevant feature, with a negative correlation.
- x_3: There is indeed little to no correlation with this feature. Although less than x_1.
- x_4: Prediction was acturate. There is basically no correlation.
- x_5: This feature has less relevance than x_1 and x_3.
- x_6: Prediction was fairly acturate, with a slight negative correlation.
- x_7: With this being the most relevant feature, the prediction was correct.

In summery, my predictions were fairly acurate. Although there were some features that turned out to be more relevant than expects, especially feature x_1.

```
In [3]: # test model load and use
        model = loadModel("linear.joblib")
        reportWithModel(model)
```

```
+-----+---------+---------------------+
|  X  |  Theta  |     Theta Value     |
+-----+---------+---------------------+
| x_1 | Theta_1 | -18.326866848615648 |
| x_2 | Theta_2 | -60.488285226256664 |
| x_3 | Theta_3 | -0.5002880659381503 |
| x_4 | Theta_4 | 0.44341680815857387 |
| x_5 | Theta_5 | -16.201565565095663 |
| x_6 | Theta_6 | -44.768884944380105 |
| x_7 | Theta_7 | -205.20882017371724 |
+-----+---------+---------------------+
Y-intercept: -1307.7683466494066
Mean Squared Error: 465.8066694026524
Root Mean Squared Error: 21.582554746893436
```

As we can see the model successfully saved, and can be used to attain the exact same results as before.

```
In [ ]:
```