Project 2: Predicting Gas Mileage with Multiple Linear Regression
COSC425, Fall 2020

Due: Oct 16 @ 11:59pm

In this project, you will be implement multiple linear regression and apply it to a dataset. This assignment follows the specification in the "Project Guidelines" document on Canvas.

I. General Information

Your project is responsible for implementing multiple linear regression. Your project uses a dataset that has one target variable and multiple input features. Your report should cover the relevant steps in the "Project Workflow", reporting what you did and the results. You are responsible for implementing your own regression functions, with the exception of basic numerical libraries.

II. Dataset

This project will use the Auto-Mpg dataset from the UCI Machine Learning repository. The dataset file is named `auto-mpg.csv`, and is available for download on Canvas. The data includes the following features: MPG (Continuous), Cylinders (Multivariate Categorical), Displacement (Continuous), Horsepower (Continuous), Weight (Continuous), Acceleration (Continuous), Model Year (Multivariate Categorical), Origin (Multivariate Categorical), and Car Name (String).

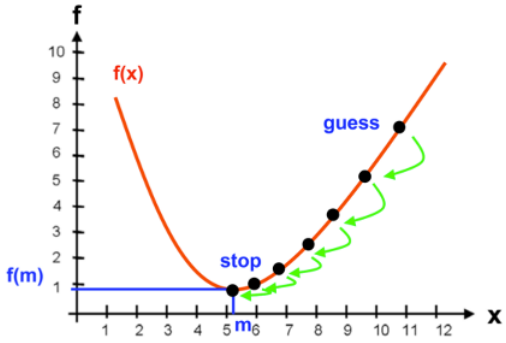

Figure 1: A visualization for the Gradient Descent approach.

III. Implementation Requirements

You should implement multiple linear regression with gradient descent. The Gradient Descent algorithm is an iterative approach to calculating coefficient "guesses" for a best-fit regression line. Your implementation cannot make use of machine learning libraries that perform regression for you (e.g. sklearn). You can, however, use them for supplemental activities (e.g. normalization).

IV. Implementation Recommendations

The Python library `NumPy` may be of substantial use. NumPy allows you to store data in matrices (see the `np.array()` function) and apply matrix operations to them. Consider a similar scenario in which we seek to implement linear regression for predicting performance in COSC 425 based on prior performance in MATH 251 and MATH 323. We could establish a simple set-up like so:

```python
import numpy as np

# student.csv is a CSV file with three columns representing student grades
# for COSC 251, MATH 323, and COSC 425. Each row represents a different student.
data = pd.read_csv('student.csv')

# separate the values from each column by column name
matrix_alg = data['MATH251'].values
prob_stats = data['MATH323'].values
ml_grades = data['COSC425'].values

# generate a matrix of ones for x0
m = len(matrix_alg)
x0 = np.ones(m)

# create an array of all features, i.e. X values, and transpose it
X = np.array([x0, matrix_alg, prob_stats]).T

# initialize coefficients
B = np.array([0, 0, 0])

# create an array for output variables
Y = np.array(ml_grades)

# establish a learning rate
alpha = 0.0001
# NOTE: This is an example for setting up your code. You do not have to use it.
```

Listing 1: An example with matrix manipulation in NumPy (Python 3.7.7)

Should you choose to make use of NumPy, you should be aware of the `dot` function that facilitates dot product calculations and the `np.sum` function, which facilitates summation. An example can be found here: https://pythonexamples.org/python-numpy-dot-product/.

IV. Evaluation Requirements

Using your implementation, you should evaluate two regression models:

- A regression model that uses the provided dataset as is.
- A regression model that uses a "standardized" version of the dataset.

"Standardization" is the process of (1) normalizing continuous features and (2) standardizing these features' mean to zero and variance to 1. As implementing standardization is not the goal of this project, I am permitting you to use the `MinMaxScaler` and `StandardScaler` classes in the `sklearn.preprocessing` library to perform standardization. A clear example of these can be found in Section 2 of this webpage.

The R^2 Metric

The most common way to evaluate regression models is through the R^2 metric, which measures how much of your data's variance is captured by your regression model. Generally speaking, your model's R^2 value should be higher because you want your model to explain as much of your data's variance as possible (i.e., has the best "fit"). In other words, the best-performing model is the one that yields the highest R^2.

Given that we didn't dive into evaluating regression models in class, I'm providing you with the implementation for calculating the R^2 metric with NumPy arrays:

```
import numpy as np

def r2_score(Y, Y_pred):
    '''Produces the R2 score between two NumPy arrays.'''
    mean_y = np.mean(Y)
    ss_tot = sum((Y - mean_y) ** 2)
    ss_res = sum((Y - Y_pred) ** 2)
    r2 = 1 - (ss_res / ss_tot)
    return r2
```

Listing 2: Implementation for R^2 calculations in NumPy (Python 3.7.7)

Alongside the R^2 metric, you should report the final coefficients in your models. To better understand your coefficients, it may also help to present information about how "cost" (i.e., Y_{pred} - Y) changes across iterations for each model. (e.g., At what point do your iterations start having little change in your cost?).

A Note on Plotting

Before you begin your implementation, it may be beneficial to visually explore you data (e.g. What is the distribution of each feature in your dataset?). Seaborn[1] is an extended Python library built on top of Matplotlib, and it is perfectly permissible to use in your project. Seaborn generally brings ease to the task of plotting complex information. I encourage you to explore the Seaborn Gallery[2], which demonstrates a variety of plots that the library can create out-of-the-box.

Extra Credit: Simplified Models and Polynomial Regression

The base requirement for this assignment is a comparison of two models. For extra credit, explore the possibility of adding polynomial terms into your models and examine how doing so affects the R^2 metric. Further, you may also explore dropping terms from your model. For example, how does a model that includes all continuous input features compare to a model that includes includes only a single continuous input feature?

V. Writing the Report

The "Project Guidelines" document outlines the general format that you should use for your format and the grading rubric used to evaluate your submission. Note that the report is a requirement for project submission.

Note: A submission without functional source code will result in a project submission of zero.

[1] https://seaborn.pydata.org/

[2] https://seaborn.pydata.org/examples/index.html