Homework 8: Linear Regression

This homework covers several regression topics, and will give you practice with the `numpy` and `sklearn` libraries in Python. It has both a coding and a writeup component.

1 Goals

In this homework you will:

1. Build linear regression models to serve as predictors from input data
2. Parse input data into feature matrices and target variables
3. Use cross validation to find the best regularization parameter for a dataset

2 Background

Before attempting the homework, please review the notes on linear regression. In addition to what is covered there, the following background may be useful:

2.1 CSV Processing in Python

Like .txt, .csv (comma-separated values) is a useful file format for storing data. In a CSV file, each line is a data record, and different fields of the record are separated by commas, making them two-dimensional data tables (i.e., records by fields). Typically, the first row and first column are headings for the fields and records.

Python's `pandas` module helps manage two-dimensional data tables. We can read a CSV as follows:

```
import pandas as pd
data = pd.read_csv('data.csv')
```

To see a small snippet of the data, including the headers, we can write `data.head()`. Once we know which columns we want to use as features (say 'A', 'B', 'D') and which to use as a target variable (say 'C'), we can build our feature matrix and target vector by referencing the header:

```
X = data[['A', 'B', 'D']]
y = data[['C']]
```

More details on Pandas can be found here: `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html`.

2.2 Matrix Algebra in Python

Python offers computationally efficient functions for linear algebra operations through the numpy library. Suppose A is a list of m lists, each having n numerical items. Numpy will treat A as an $m \times n$ matrix. If we want to transpose A, we can write:

```
import numpy as np
AT = A.T
```

if B is another $m \times n$ matrix, we can perform the matrix operation $A^T B$ by writing:

```
AB = A.T @ B
```

Note that if $n = 1$, i.e., A and B are both vectors with m elements, this operation takes the dot product between the vectors.

If A is a square $n \times n$ matrix, we can find its inverse (if it exists) with the following:

```
Ainv = np.linalg.inv(A)
```

Other useful matrix operations can be found here: https://docs.scipy.org/doc/numpy/reference/routines.linalg.html.

2.3 Linear Regression in Python

Python offers several standard machine learning models with optimized implementations in the sklearn library. Suppose we have a feature matrix X and a target variable vector y. To train a standard linear regression model, we can write:

```
from sklearn.linear_model import LinearRegression
model_lin = linear_model.LinearRegression(fit_intercept = True)
model_lin.fit(X, y)
```

Then, if we have a feature matrix Xn of new samples, we can predict the target variables (if we know the model is performing well) by applying the trained model:

```
yn = model_lin.predict(Xn)
```

And we can view the parameters of the model by writing:

```
model_lin.get_params()
```

There are also a few different versions of regularized linear regression models in sklearn. One of the most common is ridge regression, which has a single regularization parameter λ. To train with $\lambda = 0.2$, for instance, we can write:

```
from sklearn.linear_model import Ridge
model_ridge = linear_model.Ridge(alpha = 0.2, fit_intercept = True)
model_ridge.fit(X, y)
```

More regression models in Python can be found here: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning.

3 Instructions

3.0 Setting up your repository

Click the link on Piazza to set up your repository for Homework 8, then clone it.

Aside from this readme, the repository should contain the following files:

1. `plotfit.py`, a starter file with functions, instructions, and a skeleton that you will fill out in Problem 1.
2. `poly.txt`, a data file for Problem 1 where each row is a datapoint in the format: `x y`, with `x` being the explanatory and `y` being the target variable.
3. `regularize-cv.py`, a starter file with functions, instructions, and a skeleton that you will fill out in Problem 2.
4. `diamonds.csv`, a data file for Problem 2 where each row has 10 attributes corresponding to a diamond.

3.1 Problem 1: Polynomial regression

A common misconception is that linear regression can only be used to fit a linear relationship. We can fit more complicated functions of the explanatory variables by defining new features that are functions of the existing features. A common class of models is the **polynomial**, with a dth degree polynomial being of the form

$$\hat{y}_d(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + b$$

with the $d+1$ parameters $\beta = (a_d, ..., a_1, b)^T$. So $d = 1$ corresponds to a line, $d = 2$ to a quadratic, $d = 3$ to a cubic, and so forth.

In this problem, you will build a series of functions that fit polynomials of different degrees to a dataset. You will then use this to determine the best fit to a dataset by comparing the models from different degrees visually against a scatterplot of the data, and make a prediction for an unseen sample. More specifically:

1. Complete the functions in `polyfit.py`, which accepts as input a dataset to be fit and polynomial degrees to be tried, and outputs a list of fitted models. The specifications for the `main`, `feature_matrix`, and `least_squares` functions are contained as comments in the skeleton code. The key steps are parsing the input data, creating the feature matrix, and solving the least squares equations.
2. Use your completed `polyfit.py` to find fitted polynomial coefficients for $d = 1, 2, 3, 4, 5$ on the `poly.txt` dataset. Write out the resulting estimated functions $\hat{y}_d(x)$ for each d.
3. Use the `scatter` and `plot` functions in the `matplotlib.pyplot` module to visualize the dataset and these fitted models on a single graph (i.e., for each x, plot y, $\hat{y}_1(x)$, ..., $\hat{y}_5(x)$). Be sure to vary colors and include a legend so that each curve can be distinguished. What degree polynomial does the relationship seem to follow? Explain.
4. If we measured a new datapoint $x = 2$, what would be the predicted value $\hat{y}$ of y (based on the polynomial identified as the best fit in Question 3)?

Note that in this problem, **you are not permitted to use the `sklearn` library**. You must use matrix operations in `numpy` to solve the least squares equations.

Once you have completed `polyfit.py`, if you run the test case provided, it should output (rounded to 6 significant digits)

$$[[7.00158, 9.30386, -239.334], [0.00598796, 0.755218, 0.234560, 1.17636, -175.880]]$$

3.2 Problem 2: Regularized regression

Regularization techniques like ridge regression introduce an extra model parameter, namely, the regularization parameter λ. To determine the best value of λ for a given dataset, we often employ cross validation, where we compare the error of the trained model with different values of λ on a test set, and choose the one yielding lowest error.

In this problem, you will complete the starter code in `regularize-cv.py` that employs cross validation in selecting the best combination of model parameters β and regularization parameter λ for a predictor on a given dataset. We use the `diamonds.csv` dataset (`http://vincentarelbundock.github.io/Rdatasets/datasets.html`) here, which contains the prices and nine descriptive attributes (carats, cut, color, clarity, depth, table, x, y, z) of roughly 54,000 diamonds. From the input data, you will train a ridge regression model on these nine attributes for different values of λ, find the best, and use the result to predict the price of a new diamond given a set of input features describing it. More specifically:

1. Complete the function `normalize_train` that takes the training set **X_train** as input and returns a normalized feature matrix along with arrays of the means and standard deviations for each column. (The means and standard deviations for each column are needed for properly normalizing the testing data.) The nine columns of this matrix, $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_9]$, must each be normalized to have a mean of 0 and a standard deviation of 1. Recall that this can be accomplished, for each column, by calculating the column's mean and standard deviation and then subtracting that mean from each element in the column and dividing the result by the column's standard deviation.

2. Complete the function `normalize_test` that takes the testing set **X_test** and the training set means and standard deviations and returns a normalized feature matrix. Each column should subtract the mean of the corresponding column from **X_train** and divide by the standard deviation of the corresponding column from **X_train**. For example, each element in the first column of **X_test** should subtract the mean of the first column of **X_train** and divide by the standard deviation of the first column of **X_train**.

3. Define the range of λ to test in `main` as $[10^{-1.00}, 10^{-0.97}, 10^{-0.94}, 10^{-0.91}, 10^{-0.88}, \cdots, 10^{1.88}, 10^{1.91}, 10^{1.94}, 10^{1.97}, 10^{2.00}]$. This type of logarithmic scale is common for regularization. You should use the numpy function `np.logspace` to define this array (Hint: Use 101 points as `num`). See the documentation on `np.logspace` here: `https://docs.scipy.org/doc/numpy/reference/generated/numpy.logspace.html`.

4. Complete the function `train_model` to fit a ridge regression model with regularization parameter $\lambda = l$ on a training dataset $\mathbf{X}_{train}$, $\mathbf{y}_{train}$. You may use the `linear_model.Ridge` class in `sklearn` to do this. Note that the partition of the training and testing set has already been done for you in the `main` function.

5. Complete the function `error` to calculate the mean squared error of the model on a testing dataset $\mathbf{X}_{test}$, $\mathbf{y}_{test}$.

6. Complete the code in `main` for plotting the mean squared error as a function of λ, and for finding the `model` and `mse` corresponding to the best `lmbda`. Be sure to include a title and axes labels with your plot.

7. Using the coefficients (and intercept) $\beta = (a_1,\ a_2,\ \cdots\ a_9, b)^T$ from the returned `model_best`, write out the equation $\hat{y}(\mathbf{x}) = a_1x_1 + a_2x_2 + \cdots + a_9x_9 + b$ of your fitted model for a sample $\mathbf{x}$. What is the predicted price $\hat{y}$ for a 0.25 carat, 3 cut, 3 color, 5 clarity, 60 depth, 55 table, 4 x, 3 y, 2 z diamond?

Once you have completed `regularizecv.py`, if you set `lmbda` $= [1, 100]$, your output message should be 'Best lambda tested is 1, which yields an MSE of 1812351.1908771885'.

4 What to Submit

For each problem, you must submit (i) your completed version of the starter code, and (ii) a writeup as a separate PDF document named `problem1_writeup.pdf` and `problem2_writeup.pdf` respectively.

5 Submitting your Code

Push your completed `polyfit.py`, `problem1_writeup.pdf`, `regularize-cv.py`, and `problem2_writeup.pdf` to your repository before the deadline.