Homework 7: Linear Regression

Due Friday, October 31, 2025 at 11:59 pm ET

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. You have two weeks to work on this assignment.

Submit both the *CODE + PDF document (Write-up)*!

Submit **multiple** files named Gradescope (**NOT a single zip file**). The following items should be submitted: 1. A write-up document named `outcomes.pdf` 2. A completed `polyfit.py` 3. A completed `regularize_cv.py`

Failure to follow these guidelines will result in an incorrect formatting penalty

Grading Breakdown

This homework is worth 100 points total: - **Code (30 points)**: Your Python code (`polyfit.py` and `regularize_cv.py`) will be automatically tested for basic functionality to ensure you have implemented the required functions correctly. - **Writeup (70 points)**: Your written solutions in `outcomes.pdf` will be manually graded based on correctness, clarity, and completeness.

The autograder will run basic tests on your code to verify that your functions work as expected. Make sure to test your code thoroughly before submission!

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

Packages access

For this homework, you will have access to the following Python packages and using anything else may cause you to fail test cases:

For `polyfit.py` (problem 1):

```
numpy==1.24.4
matplotlib==3.7.2
pandas==2.1.0
```

For `regularize_cv.py` (problem 2):

```
numpy==1.24.4
matplotlib==3.7.2
scikit-learn==1.3.2
pandas==2.1.0
```

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:

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.

You would need to install the python's pandas module for this homework. Please note that the skeleton code provided has an example of CSV processing included. 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

Matrix Algebra in Python

Python offers computationally efficient functions for linear algebra operations through the numpy library. Suppose `nested_list` is a list of `m` lists, each having `n` numerical items. We can convert this to a numpy 2D array by calling `A = np.array(nested_list)`.
Numpy will treat `A` as an `m` $\times$ `n` matrix. If we want to transpose A, we can write:

```
import numpy as np

AT = np.transpose(A)
```

or

```
AT = A.T
```

if B is another m × n matrix, we can perform the matrix operation $A^\top B$ (where $A^\top$ denotes the tranpose of A) by writing:

```
ATB = np.matmul(np.transpose(A), B)
```

or

```
ATB = A.T @ B
```

*Note that this is different from the * symbol which will do an element-wise multiplication rather than matrix multiplication.*
Also, 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.

Additionally, there are certain restrictions on the the shape of matrices when multiplied together. The shape of a matrix `A` can be found using `A.shape` which will return a tuple of the dimensions of the matrix. Two matrices `A` and `B` can only be matrix multiplied together `A @ B` if the number of rows in `A` match the number of columns in `B`. Put another way, `A` must be an `n` × `m` matrix and `B` must be an `j` × `n` matrix to multiply `A^T` with `B`.

If `A` is a square `n` × `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.ht

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 `X_n` of new samples, we can predict the target variables (if we know the model is performing well) by applying the trained model:

```
y_n = model_lin.predict(X_n)
```

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 $\lambda = 0.2$ (named `alpha` in sklearn), 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

Instructions

Download starter code and writeup template

Download the zip file for the homework from Brightspace. If you are reading this, you would have already done that step. Aside from this readme, you should also see the following files:

1. `polyfit.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. `AAPL.csv`, a data file for Problem 2. It contains historical stock data for Apple Inc. (AAPL) from 12-12-1980 to 02-27-2022. The goal is to forecast next day's closing stock price.
5. `GOOG.csv`, historical stock data for Google over past five years. You would need this for completing the part 8 of Problem 2.
6. `outcomes_sample.pdf` is there to give you an idea as to what's expected from your solution PDF. You need to submit your solution PDF that you can create by writing your answers to the `outcomes.docx`. ***Only PDF file format submitted to Gradescope will be accepted.*** There will be a ***penalty*** if your submission includes any other file format.
7. `given_tests.py` is a test file you can run locally to verify your implementation before submitting. Run it with `python3 given_tests.py` to see if your code produces the expected outputs. The autograder on Gradescope will run similar (but not identical) tests.

Problem 1: Polynomial regression

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.

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 d-th degree polynomial being of the form

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

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 visually comparing the models created from different degrees against a scatterplot of the data, and make a prediction for an unseen sample.

More specifically:

(IMPORTANT NOTE: Your solutions need to be written in a writeup for problem 1 and 2. Only PDF file format will be accepted. There will be a penalty if your submission includes any other file format.)

For examples as to how your written answers should be formated, please see the `outcomes.docx` and `outcomes_sample.pdf` files.

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 = 1,2,3,4,5,6` on the `poly.txt` dataset. Write out the resulting estimated functions $\hat{y}_d(x)$ for each d. Note that the skeleton code for this part and the parts below are provided within the `if __name__ == '__main__':` statement.

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 $(y, \hat{y}_1(x), \ldots, \hat{y}_6(x))$. You may need to use the `feature matrix` function from part 1. Make 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.

(Note: You must fill in your code inside the if **name** == "**main**": section for this problem.)

4. If we measured a new datapoint x = 2, what would be the predicted value $\hat{y}$ (based on the polynomial identified as best fitting the data in Part 3 above)? This is to be answered in the writeup pdf.

Again, 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 (where d = [2,4]), it should output (to at least 3 decimal places rounded up):

```
y_hat(x_2)
[-1.26488664 27.02773667 88.44135383]
****************
y_hat(x_4)
[-2.25144937e-02  1.75588710e+00 -8.88289340e-01 -6.51881085e-01
  9.99239381e+01]
****************
```

> Note: if your output looks slightly different (marginally different decimals) that's ok. Your output should at least have the same shape as this expected output, and **your decimals should agree to at least 3 decimal places.**

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 `AAPL.csv` dataset (https://finance.yahoo.com/quote/AAPL/history?p=AAPL) here. It has Apple's stock history from 12-11-1980 to 02-27-2022 containing 10391 rows and seven columns corresponding to each day's stock price attributes (Date, Open, High, Low, Close, Adj Close, and Volume). We're interested in forecasting the next day's Closing (`Close`) Price. We will be using the `AAPL.csv` data to train our model to predict the closing price for new datasets, in this case the `GOOG.csv` data. > Note that we don't need the date column for forecasting the stock prices.

The `prediction` values are generated by shifting the values in `Close` column by 1 because we're interested in forecasting the next day's closing price.

We have already provided the code for preparing the input and output data in the `main` function as shown below.

```
#step 1 : read csv
df = pd.read_csv('AAPL.csv')

#step 2: identify the column(s) we want to remove
remove_features = ['Date']

#step 3: create extra column for prediction by shifting
#rows of `Close` columns by one to obtain next day's closing price
df['Prediction'] = pd.Series(np.append(df['Close'][1:].to_numpy(), [0]))

# step 4: drop the last row because it would have invalid value after the shift.
df.drop(df.tail(1).index, inplace=True)

# step 5: remove the columns identified in step 2
df.drop(remove_features, axis=1, inplace=True)

# step 6: create X by dropping the `Prediction` column
X = np.array(df.drop(['Prediction'], axis=1))

# step 7: Store `Prediction` column in Y array
y = np.array(df['Prediction'])
```

Another important thing to note is that stock market data is *time series data*. Therefore, we do not want to shuffle while splitting the data into training and testing sets. We can create the training and testing sets without shuffling using the following call

```
[X_train, X_test, y_train, y_test] = train_test_split(X, y, test_size=0.2, shuffle=False)
```

From the input data, you will train a ridge regression model on these six input attributes, i.e., excluding `Date` for different values of λ, find the best, and use the result to forecast the next day's closing price given a set of input features describing it.

(IMPORTANT NOTE:Your solutions need to be written in a writeup for problem 1 and 2. Only PDF file format will be accepted. There will be a penalty if your submission includes any other file format. Solution for problems 1 and 2 should be in a SINGLE PDF file.)

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 six columns of this matrix, `X = [x_1`

`x_2 ... x_6]`, 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. Make sure to account for the case where no training data is given (in this case, return empty numpy arrays for each output variable).

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`. **Be careful here, you have to use the mean and standard deviation of the train set for the test set too, since the test set is "unlabeled data" that you model the distribution for using the training set.**

3. Define the range of λ to test in `get_lambda_range` as `[1E-1.00,..., 1E3.00]`. This type of logarithmic scale is common for regularization. You should use the numpy function np.logspace to define this array (Hint: Use 51 points as num, 1e-1 as the starting value, and 1e+3 as the final value of the sequence. I.e., the numpy array returned by np.logspace should begin with 1e-1 and end with 1e+3. The inputs to np.logspace to get this range will **NOT** be start=1e-1 and end=1e+3). 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 `X_train`, `y_train`. You may use the `Ridge` class in `sklearn` to do this. Examples for `Ridge` are mentioned above. 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 `X_test` with true values `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`('lambda' is a reserved keyword in Python, therefore we need to name it in a different way). Make sure to include a title and axes labels with your plot. Use the Ridge regression model `train_model`. (Note: You must fill in your code inside the function main for problems 6, 7, and 8)

7. Using the coefficients (and intercept) $\beta = (a_1, a_2, ..., a_6, b)^T$ from the returned `model_best`, write out the equation $\hat{y}(x) = a_1x_1 + a_2x_2 + ... + a_6x_6 + b$ of your fitted model for a sample $\mathbf{x}$.

8. Now let's put our model to test by using our model to predict the closing values for the `GOOG.csv` data. Load `GOOG.csv` (contains historical stock data for Google over the past five years), similar to `AAPL.csv` as shown before. Let's call the input features and output array as `X_goog` and `y`. Note that we don't need to split this data. **Normalize** `X_goog` similar to `X_test` and call `y_hat = model.predict(X)` ***It's important to normalize the X_goog.*** Plot `y` and `y_hat` on the same plot and make sure to vary colors and include a legend so that each curve can be distinguished.

Once you have completed `regularize_cv.py`, if you set`lmbda = [1, 500, 1000, 2500]`, your output message should be:

```
Best lambda tested is 1000, which yields an MSE of 2.091694289307238
```

Note on precision

For Problem 1 and 2 your answers are expected to be correct up to three decimal points.

What and where to Submit

Upload multiple files to Gradescope that shows your code and your writeup: (i) For each problem, your completed version of the starter code (ii) a combined problem 1 & 2 writeup as PDF document. i.e.:

```
outcomes.pdf
polyfit.py
regularize_cv.py
```