Polynomial_Regression_w_Regularization

Summary

The code defines a function `get_regression_analysis()` that performs polynomial regression with various types of regularization (L1, L2, ElasticNet) and selects the model configuration with the lowest error. It iterates through different combinations of regularization parameters, polynomial degrees, and data splits (training and testing) using scikit-learn's regression models.

Finally, it plots the data points and the regression line for the best-performing model configuration based on the lowest error obtained.

Unlike my previous implementations, I am using sklearn's models rather than custom implementation. I did attempt to implement L1, L2, and combined regularization and was making progress, but after hours of work I was still unable to get any meaningful information from my model.

Implementation Overview

Imports

```
# Imports
import operator
import matplotlib.pyplot as plt
import pandas as pd
import random
import warnings
```

Helpers

These are functions to do small things.
load_data() simply grabs the datasets from Google Drive.

```
# Load Data
def load_data():
  drive.mount('/content/drive')
  df_training_set = pd.read_csv('/content/drive/MyDrive/kaggle/train.csv').dropna()
  df_testing_set = pd.read_csv('/content/drive/MyDrive/kaggle/test.csv').dropna()
```

```
  return df_training_set, df_testing_set
```

get_xy_values() extracts the 'x' and 'y' values from the training and testing sets, reshapes them into arrays with a single feature column, and returns those arrays.

```
# Get X,Y values from data
def get_xy_values(df_training_set, df_testing_set):
  x_train = df_training_set['x'].values.reshape(-1, 1)
  y_train = df_training_set['y'].values.reshape(-1, 1)
  x_test = df_training_set['x'].values.reshape(-1, 1)
  y_test = df_training_set['y'].values.reshape(-1, 1)

  return x_train, y_train, x_test, y_test
```

generate_polynomial_features() generates polynomial features up to a specified degree for each feature in the input array X. I ended up not using this, as sklearn had it already.

```
# Generate Polynomial Features
def generate_polynomial_features(X, degree):
    num_samples, num_features = X.shape
    X_poly = np.zeros((num_samples, num_features * degree))
    for i in range(num_samples):
        for j in range(num_features):
            for d in range(1, degree + 1):

                # Replace value at location with poly
                X_poly[i, j * degree + d - 1] = X[i, j] ** d
    return X_poly
```

Running the Program

We call get_regression_analysis() to run everything. To start, we initialize some stuff, make a call to load_data() and get_xy_values(), and ignore some warnings. There was also LinAlgWarning but trying to ignore that started causing other problems, and ignoring warnings is bad practice so I left it in. In the future, I plan to explore why those warnings were arising in the first place.

```
def get_regression_analysis():
  # Initialize some stuff
  iterations = 100000
  run_types = ['train', 'test']
  reg_types = ['L1', 'L2', 'L12']
```

```
  reg_params = [random.uniform(0, 10) for _ in range(10)]
  polynomial_degrees = 5
  smallest_abs_error, smallest_squ_error, smallest_config = float('inf'), float('inf'),
None

  # Get our initial data
  df_training_set, df_testing_set = load_data()
  x_train, y_train, x_test, y_test = get_xy_values(df_training_set, df_testing_set)
  # Ignore sklearn warnings
  warnings.filterwarnings("ignore", category=ConvergenceWarning)
```

Next, we have some loops so we can iterate through various parameters and datasets. We call PolynomialFeatures(degree=d) where d is defined in the loop. After that, we have the sklearn model implementations. Each of these will be evaluated to find the best model for the dataset.

```
for run in run_types:
    for reg_param in reg_params:
      for reg_type in reg_types:
        for d in range(0, polynomial_degrees):

          def switch(run_type):
            return {
                'train': (x_train, y_train),
                'test': (x_test, y_test)
            }.get(run_type, 'train')

          # Generate Polynomial Features
          x = switch(run)[0]
          y = switch(run)[1]
          polynomial_features = PolynomialFeatures(degree=d)
          x_poly = polynomial_features.fit_transform(x)

          # Figure out what kind of regularization to do
          if reg_type == 'L1':
            model = Lasso(alpha=reg_param, max_iter = iterations)
          elif reg_type == 'L2':
            model = Ridge(alpha=reg_param, max_iter = iterations)
          elif reg_type == 'L12':
            model = ElasticNet(alpha=reg_param, l1_ratio=0.5, max_iter = iterations)
          else: # None given
            model = LinearRegression()
```

```
            model.fit(x_poly, y)
            y_prediction = model.predict(x_poly)
```

At this point, we want to track which set of parameters gives the lowest error and save that information for later. Once all iterations are complete, we print the lowest error obtained along with a graph of data and predictions where the title is the parameters used to obtain the error and graph.

```
            # Get errors and make updates as needed
            abs_error = mae(y, y_prediction)
            squ_error = mse(y, y_prediction)

            if abs_error < smallest_abs_error or smallest_abs_error == -1:
              smallest_abs_error = abs_error
              smallest_squ_error = squ_error
              smallest_config = {'reg_type': reg_type, 'reg_param': reg_param,
'polynomial_degrees': d}
              smallest_x, smallest_y, smallest_y_prediction = x, y, y_prediction

  # Plot the best configuration
  print("Lowest Error Obtained:")
  print(f"absolute error: {smallest_abs_error}. squared error: {smallest_squ_error}.\n")

  plt.scatter(smallest_x, smallest_y)
  sort_axis = operator.itemgetter(0)
  sorted_zip = sorted(zip(smallest_x, smallest_y_prediction), key=sort_axis)
  smallest_x, smallest_y_prediction = zip(*sorted_zip)
  plt.plot(smallest_x, smallest_y_prediction, color='m')
  plt.title(smallest_config)
  plt.show()
```

To run everything, simple run the final cell in colab:

```
get_regression_analysis()
```

The output:

```
Lowest Error Obtained: absolute error: 2.2271536714188254. squared error:
7.867752736510159.
```

{'reg_type': 'L2', 'reg_param': 1.1111695142414946, 'polynomial_degrees': 1}