Brunton and Kutz Problem 4.3: MNIST Classification with Multilinear Regression

Source Filename: /main.py

Rico A. R. Picone

Begin by loading the necessary libraries:

```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp
```

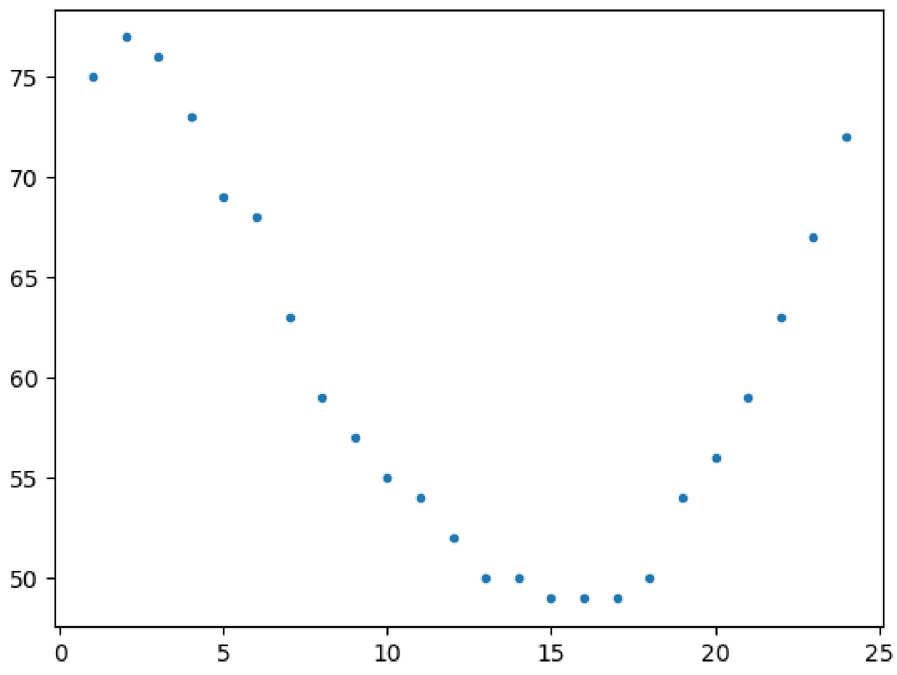
Define the data. We define two vectors: time and temperature.

```
temperature = np.array([
    75, 77, 76, 73, 69, 68, 63, 59, 57, 55, 54, 52, 50, 50, 49, 49, 49, 50, \
    54, 56, 59, 63, 67, 72
])
n_data = len(temperature)
time = np.arange(1, n_data + 1)
```

Visualize the data. The temperature data is plotted against time:

```
fig, ax = plt.subplots()
ax.plot(time, temperature, '.')
```

```
[<matplotlib.lines.Line2D at 0x15bcdc210>]
```



Create `n_samples` random corrupted data samples from corrupting by 90 percent a single original data point. The first sample is the original temperature data.

```
n_samples = 30
np.random.seed(4)  # Set random seed for reproducibility
y = np.zeros((n_samples, len(temperature)))
y[0] = temperature
for i in range(1, n_samples):
    y[i] = temperature
    jrand = np.random.randint(n_data)
    y[i, jrand] = y[i, jrand] * (1 + .9*np.array([-1,1])[np.random.randint(2)])
```

Define the objective functions. The L1 regularization term encourages sparsity in the coefficients, while the L2 regularization term encourages small coefficients. The elastic net loss is the sum of the squared residuals and the L1 and L2 regularization terms. The elastic net loss function trades off between L1 and L2 regularization. For a Lasso objective, set lambda2 = 0. For a Ridge penalty, set lambda1 = 0. For a standard least squares objective, set lambda1 = lambda2 = 0.

```
def least_squares_objective(y, x, beta):
    return cp.sum_squares(y - x @ beta)/y.shape[0]

def lasso_objective(y, x, beta, lambda1):
    return cp.sum_squares(y - x @ beta)/y.shape[0] + \
        lambda1 * cp.norm1(beta)

def ridge_objective(y, x, beta, lambda2):
    return cp.sum_squares(y - x @ beta)/y.shape[0] + \
        lambda2 * cp.sum_squares(beta)

def elastic_objective(y, x, beta, lambda1, lambda2):
    return cp.sum_squares(y - x @ beta)/y.shape[0] + \
        lambda1 * cp.norm1(beta) + lambda2 * cp.sum_squares(beta)
```

Solve the elastic net optimization problem for each corrupted data sample with all four objective functions: least squares, Lasso, Ridge, and elastic net. Fit to a 10th order polynomial with coefficients alpha.

```
def x_matrix(n_powers, time):
    return np.vander(time, n_powers + 1, increasing=True)
```

```
n_powers = 10  # Polynomial order
n_alpha = n_powers + 1
n_models = 4
alpha_values = np.zeros((n_samples, n_models, n_alpha))
lambda_amp = 0.1
lambda_pairs = lambda_amp * np.array([[0, 0], [1, 0], [0, 0.01], [0.99, 0.01]])
model_names = ['Least Squares', 'Lasso', 'Ridge', 'Elastic Net']
x = x_matrix(n_powers, time)  # Polynomial power matrix
for i in range(n_samples):
    for j in range(n_models):
        beta = cp.Variable(n_alpha)
        lambda1, lambda2 = lambda_pairs[j]
        if model_names[j] == 'Least Squares':
            objective = cp.Minimize(least_squares_objective(y[i], x, beta))
        elif model_names[j] == 'Lasso':
            objective = cp.Minimize(lasso_objective(y[i], x, beta, lambda1))
        elif model_names[j] == 'Ridge':
            objective = cp.Minimize(ridge_objective(y[i], x, beta, lambda2))
        elif model_names[j] == 'Elastic Net':
            objective = cp.Minimize(elastic_objective(y[i], x, beta, lambda1, lambda2))
        problem = cp.Problem(objective)
        problem.solve(
            solver=cp.SCS,
            verbose=False,
            # eps_rel=1e-10,
            # eps_abs=1e-9,
        )
        alpha_values[i, j] = beta.value
```

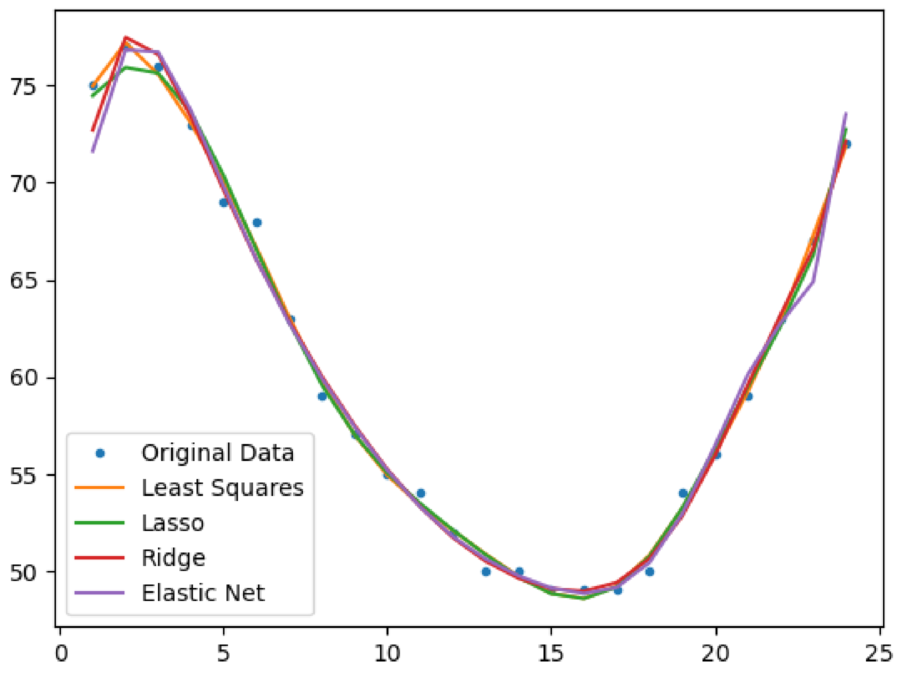
```
/Users/ricopicone/anaconda3/envs/595/lib/python3.11/site-packages/cvxpy/problems/problem.py
  warnings.warn(
print("Alpha values for the original data fit:")
print(f"Least Squares: {alpha_values[0, 0]}")
print(f"Lasso: {alpha_values[0, 1]}")
print(f"Ridge: {alpha_values[0, 2]}")
print(f"Elastic Net: {alpha_values[0, 3]}")
```

```
Alpha values for the original data fit:
Least Squares: [ 5.79073669e+01  3.19066710e+01 -2.05166991e+01  6.93431576e+00
 -1.44045979e+00  1.88958390e-01 -1.58515704e-02  8.46271739e-04
 -2.77723523e-05  5.10312685e-07 -4.01721740e-09]
Lasso: [ 7.18786216e+01  2.80678608e+00  4.41512624e-11 -2.20532232e-
01
 -3.46393123e-10  8.48372336e-03 -1.30910243e-03  9.16229784e-05
 -3.39211116e-06  6.41786500e-08 -4.85515256e-10]
Ridge: [ 5.71544201e+01  2.30174866e+01 -8.61312020e+00  1.15879066e+00
 -3.35503650e-03 -1.97376692e-02  2.85698202e-03 -2.03302157e-04
  8.12135632e-06 -1.73930247e-07  1.55626569e-09]
Elastic Net: [ 6.06875001e+01  1.24636900e+01  1.63707706e-11 -
2.14466603e+00
  7.17565944e-01 -1.17285140e-01  1.13360987e-02 -6.77236507e-04
  2.45696627e-05 -4.96057234e-07  4.27305246e-09]
```

Plot the fits for the original data:

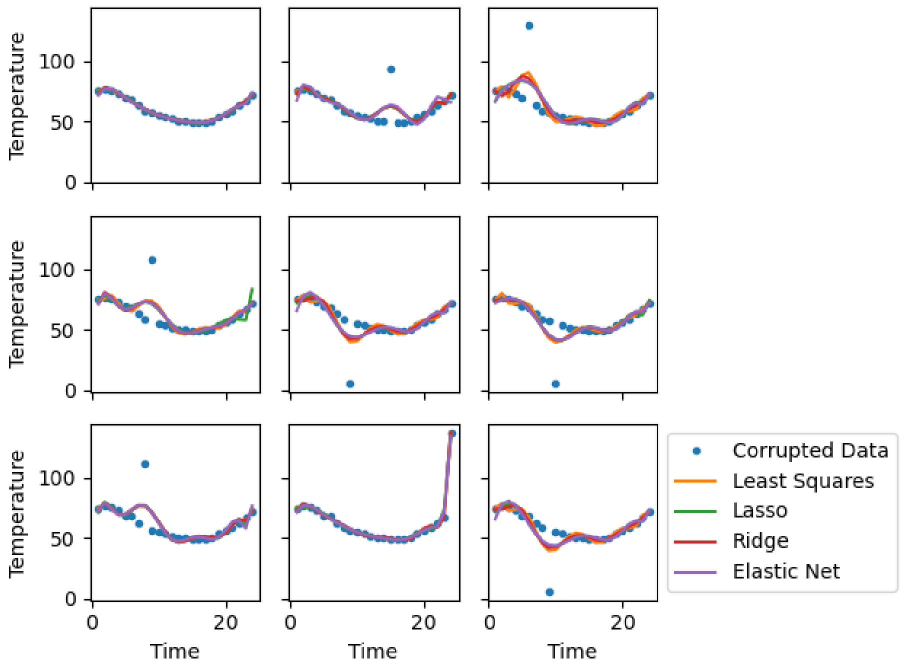
```
fig, ax = plt.subplots()
ax.plot(time, temperature, '.', label='Original Data')
for j in range(n_models):
    ax.plot(time, x @ alpha_values[0, j], label=['Least Squares', 'Lasso', 'Ridge', 'Elastic
ax.legend()
```

```
<matplotlib.legend.Legend at 0x15c2b9410>
```



All the fits are very similar for the original data. Now we will plot the fits for the original data with the corrupted data. Plot a 3x3 grid of samples and fits for the corrupted data:

```
fig, ax = plt.subplots(3, 3, sharex=True, sharey=True)
for i in range(3):
    for j in range(3):
        ax[i, j].plot(time, y[i * 3 + j], '.', label='Corrupted Data')
        for k in range(n_models):
            ax[i, j].plot(time, x @ alpha_values[i * 3 + j, k], label=['Least Squares', 'Las
            if i == 2:
                ax[i, j].set_xlabel('Time')
            if j == 0:
                ax[i, j].set_ylabel('Temperature')
ax[i, j].legend(loc='center left', bbox_to_anchor=(1, 0.5))
fig.tight_layout()
```

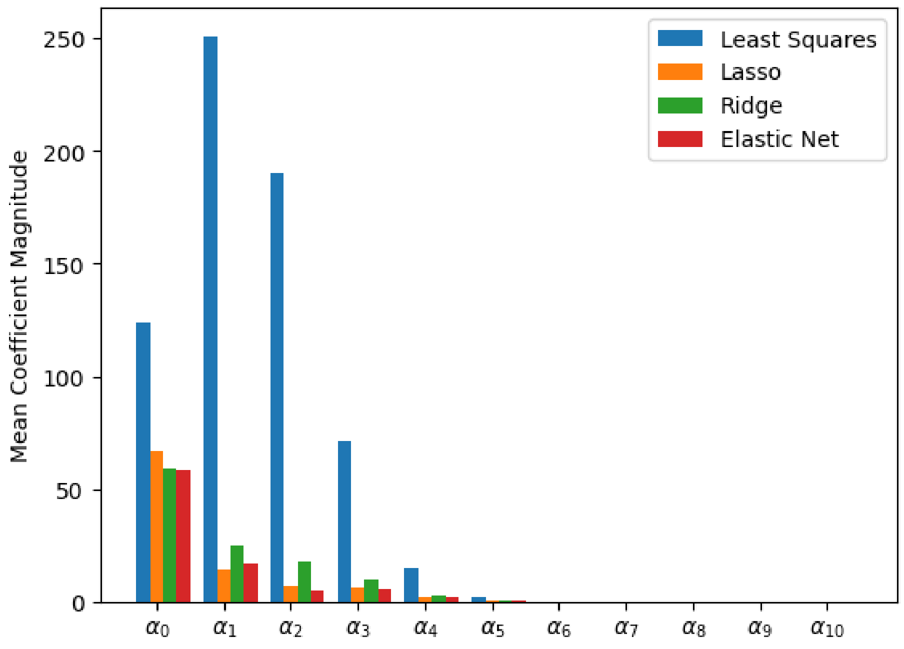


The fits are different. From this selection of samples, it's hard to evaluate the relative performance of the models.

To get a better sense of the magnitude of the coefficients, we can plot the mean of the absolute values of the coefficients for each model. Multi bar chart the mean magnitudes of the coefficients for each model:

```
fig, ax = plt.subplots()
bar_width = 0.2
bar_positions = np.arange(n_alpha)
for j in range(n_models):
    ax.bar(bar_positions + j * bar_width, np.mean(np.abs(alpha_values[:, j]), axis=0), bar_w
ax.set_xticks(bar_positions + bar_width)
ax.set_xticklabels([f'$\\alpha_{{{i}}}$' for i in range(n_alpha)])
ax.set_ylabel('Mean Coefficient Magnitude')
ax.legend()
```

```
<matplotlib.legend.Legend at 0x15c5d8290>
```

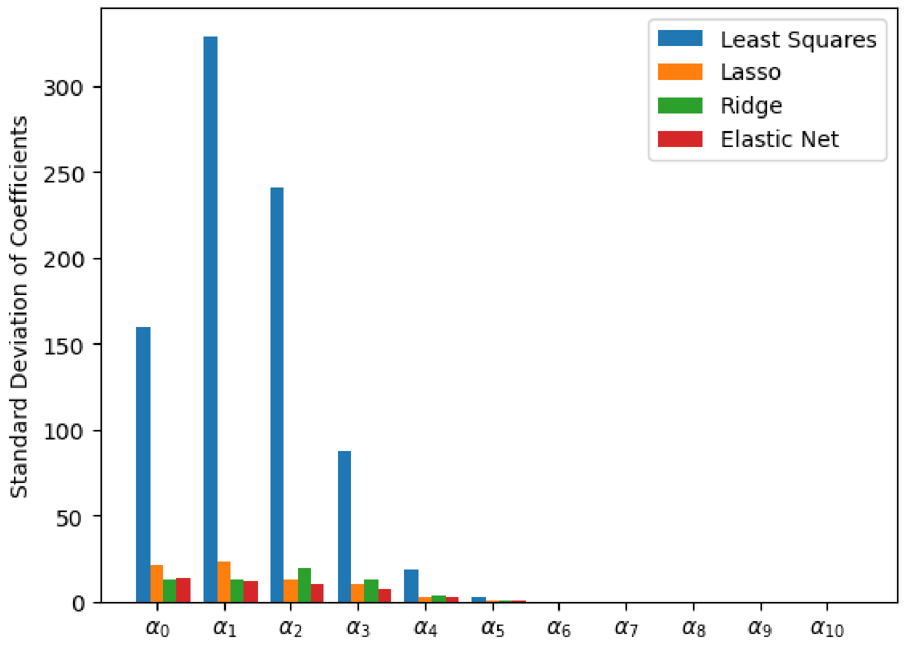


The least squares model has the largest coefficients, as expected. The lasso model is not as sparse as we might want, but weighting the L1 norm more heavily would increase sparsity. I didn't do so because it made the fits quite a bit worse.

To get a better sense of the variation in the coefficients, we can plot the standard deviation of the coefficients for each model. Multi bar chart the standard deviation of the coefficients for each model:

```
fig, ax = plt.subplots()
bar_width = 0.2
bar_positions = np.arange(n_alpha)
for j in range(n_models):
    ax.bar(
        bar_positions + j * bar_width, np.std(alpha_values[:, j], axis=0),
        bar_width,
        label=['Least Squares', 'Lasso', 'Ridge', 'Elastic Net'][j]
    )
ax.set_xticks(bar_positions + bar_width)
ax.set_xticklabels([f'$\\alpha_{{{i}}}$' for i in range(n_alpha)])
ax.set_ylabel('Standard Deviation of Coefficients')
ax.legend()
```

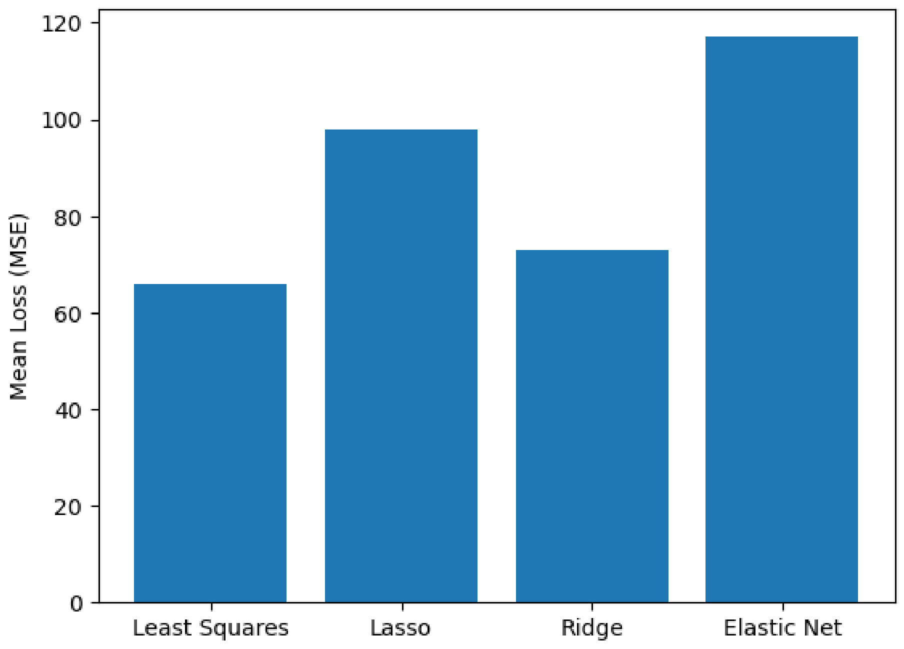
```
<matplotlib.legend.Legend at 0x15c89b210>
```



For most coefficients, the standard deviation is largest for the least squares model and smallest for the elastic net model. Having a small standard deviation is a guard against overfitting.

Bar chart the loss for each model:

```
def loss(y, x, alpha):
    # Use the squared error loss (i.e., least squares loss)
    return least_squares_objective(y, x, alpha)
loss_values = np.zeros((n_samples, n_models))
for i in range(n_samples):
    for j in range(n_models):
        loss_values[i, j] = loss(y[i], x, alpha_values[i, j]).value
fig, ax = plt.subplots()
bar_positions = np.arange(n_models)
ax.bar(bar_positions, np.mean(np.abs(loss_values), axis=0))
ax.set_xticks(bar_positions)
ax.set_xticklabels(['Least Squares', 'Lasso', 'Ridge', 'Elastic Net'])
ax.set_ylabel('Mean Loss (MSE)')
plt.show()
```



As we expect, the least squares model has the highest loss because it has prioritized fit over sparsity or small coefficients, which can lead to overfitting.