schulzd_Lab6

October 26, 2020

1 Lab 6 - Cost Functions and Parameter Space

David Schulz

1.1 Introduction

The previous lab had us experiment with mathematical models and convert them into code. We approached modelling using a 5-step framework to help organize our thoughts when considering a model. These steps included: 1. Data Familiarity 2. Model Familiarity 3. Model Implementation 4. Model Output (and Visualization) 5. Error Between Model and Data

This lab will focus on converting this approach to a standardized form – the cost function – and further exploring the implications and approaches needed to solve for good model parameters. While in the previous lab we were required to make 'blind' guesses at good model parameters, the experiments that we will run in this lab will use a more directed approach - a grid search. The grid search will evaluate a set of model parameters, plot the model error in the parameter space, and find which model parameters gives minimal modelling error.

1.2 Questions

1. By looking at the provided cost_functions.py, use 1-2 sentences to describe in detail the purpose of each of the methods. To guide this description, discuss the method input, method output, and what function each method serves for the cost function.
    - First, the file is split into two classes: GaussianCostFunction and LinearCostFunction. The only difference between the two classes is the _predict method because the GaussianCostFunction predicts with 2 parameters and the gaussian function, while the LinearCostFunction predicts with 4 parameters and a multivariable linear system. The ___init___ method is the class instantiator. It takes the features and the true responses as inputs and saves them in the class for the cost function. The _predict method creates predicted responses as output from the given features and parameters as input. The _mse method calculates the mean-squared error as output from the given predicted responses and true responses. The cost method uses given parameters as input, as well as the features that were saved when the class was created, to predict the responses with the _predict method and then returns the calculated mean-squared error from the _mse method.
2. For the heatmaps that you generated for this lab, what do they describe? What do the "valleys" and "peaks" of this heat map represent?

- The x axis is the sigma parameter and the y axis is the mu parameter. The color at each point represents the calculated mean-squared error with the respective parameters at the point. The "valleys" are the ranges of parameters where the error was lowest and the "peaks" are where the error was greatest.

3. For experiment 2, you increased the number of samples within the specified range.
   - Describe how the heatmap representation changed due to this increase in sampling.
     - The number of samples did not increase. Instead, the range where mu and sigma were being tested decreased. This also narrowed the color gradient to show the difference in error more clearly.
   - What benefit did this higher sampling rate have for finding the set of parameters with the minimum error?
     - It decreased the area where the "valley" occurred, narrowing the range of possible parameters.
   - Was this sampling rate high enough? Defend your answer!
     - No it was not because the resulting heatmap still showed a comparatively large area where the color didn't show much difference.
4. The Gaussian distribution model is limited to two dimensions while the multivariate linear model implemented for this lab is 4 dimensional.
   - Describe a limitation of the grid search method as you add additional dimensions. Hint: Think about the time complexity required for the grid search as you add additional dimensions.
     - If there are m features and p grid points per feature, p^m points would need to be evaluated, so the runtime grows exponentially with the number of features/dimensions.
   - With time complexity in mind, can you derive a rule (mathematical expression) to estimate how many grid points are needed to evaluate all combination of parameters based on the number of dimensions.
     - With m features and p grid points per feature, there would be p^m points.
   - With this rule, compare 2-dimensional models with 4-dimensional models. 10-dimensional? 100-dimensional?
     - 4-dimensional models would run p^2 times longer than 2-dimensional. 10-dimensional models would run p^8 times longer. 100-dimensional models would run p^98 times longer.
5. In experiment 3 you plotted the line of identity in the figure that compared the given response variable to the model prediction.
   - What does this line represent and how is it useful?
     - The line represents the perfect solution where every predicted value equals its corresponding expected value. The closer the points are to lining up with it, the better the prediction was.
   - What does it mean for a value to lie above the line? Below the line?
     - A point above the line means it was an underestimation because the expected response value was greater than the predicted value. A point below the line would be an overestimation.
   - How would predictions that perfectly replicate the given data appear in this plot?
     - As I said, the points would line up evenly-spaced perfectly with the line.
6. What are the weaknesses of grid search? Why wouldn't we want to use it?
   - It's computationally expensive, even if using numpy. As I said earlier, if there are m features and p grid points per feature, p^m points would need to be evaluated, so the

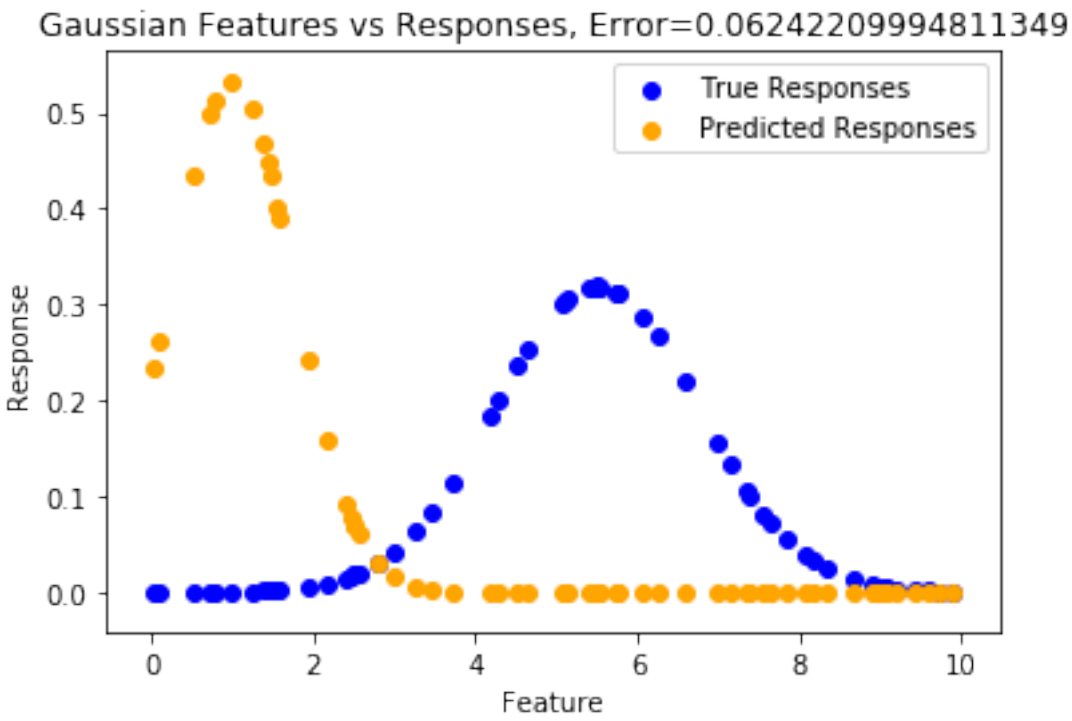
runtime grows exponentially with the number of features. The grid resolution is also hard to determine to find the absolute best parameters.

1.3 Experiment 1: Coarse Grid Search - Gaussian Distribution

1.3.1 Notebook Setup

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from cost_functions_stub import GaussianCostFunction

gauss = pd.read_csv('gaussdist.csv', header=None)
gauss = gauss.to_numpy()

features = gauss[:, 0]
responses = gauss[:, 1]

gcf = GaussianCostFunction(features, responses)
params = np.array([1.0, 0.75])
pred = gcf._predict(features, params)
error = gcf.cost(params)

plt.xlabel('Feature')
plt.ylabel('Response')
plt.title('Gaussian Features vs Responses, Error=' + str(error))
plt.scatter(features, responses, c='blue', label='True Responses')
plt.scatter(features, pred, c='orange', label='Predicted Responses')
plt.legend()
```

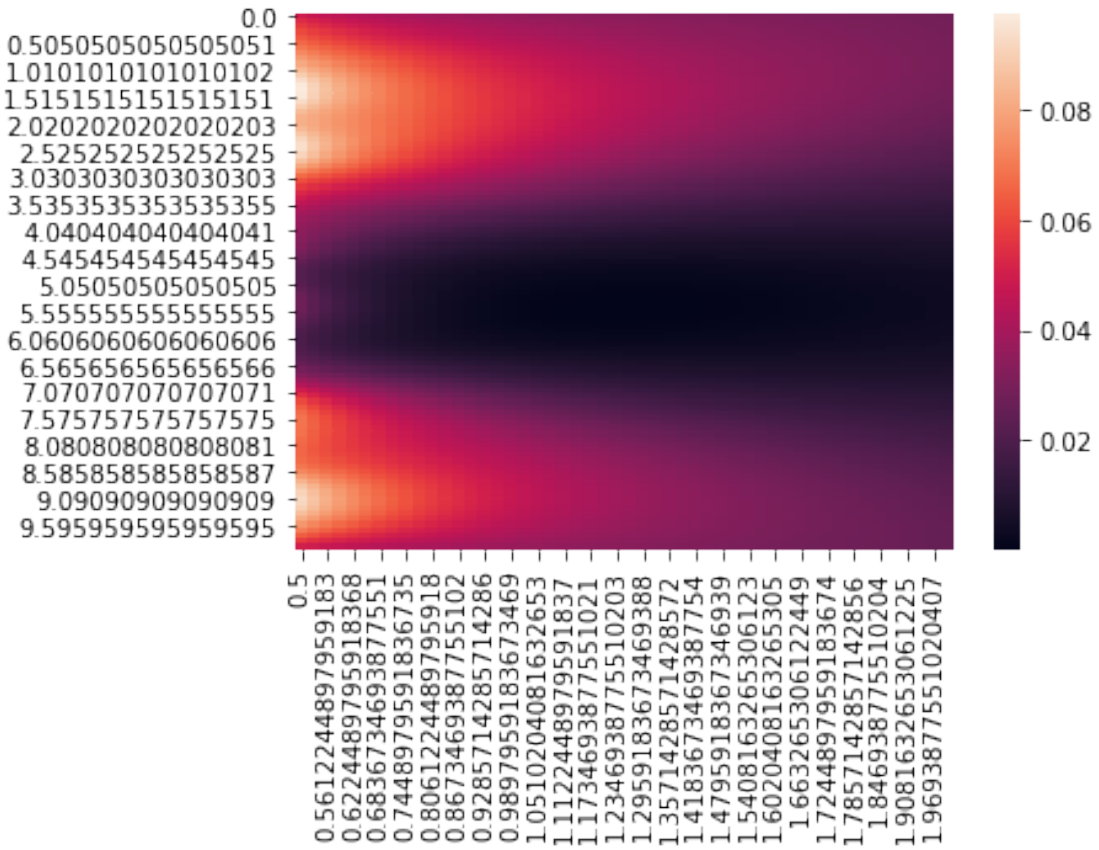
[1]:
```
<matplotlib.legend.Legend at 0x7fe843661fd0>
```



1.3.2 Grid Search - Coarse

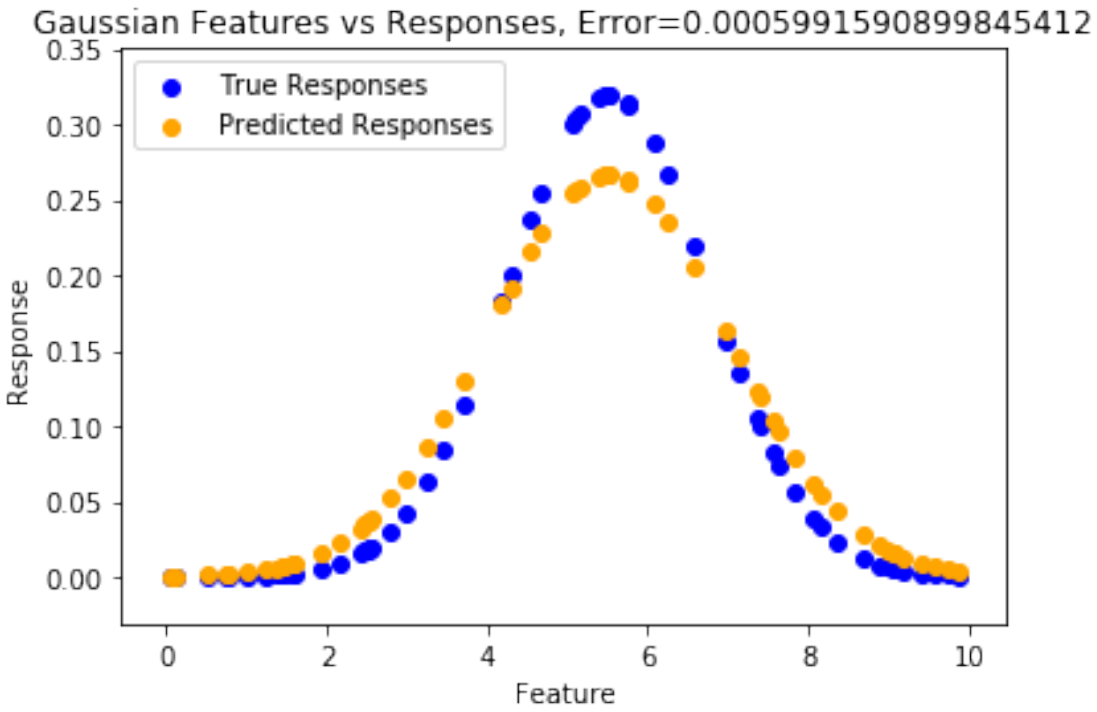
[2]:
```
from seaborn import heatmap

mus = np.linspace(0, 10, num=100)
sigmas = np.linspace(0.5, 2, num=50)

errors = np.ndarray((len(mus), len(sigmas)))
for i in range(len(mus)):
    mu = mus[i]
    for j in range(len(sigmas)):
        sigma = sigmas[j]
        error = gcf.cost(np.array([mu, sigma]))
        errors[i][j] = error

errors = pd.DataFrame(data=errors, index=mus, columns=sigmas)
heatmap(errors)
```

[2]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe8414bd4d0>
```



[3]:
```
params = np.array([5.5, 1.5])
pred = gcf._predict(features, params)
error = gcf.cost(params)

plt.xlabel('Feature')
plt.ylabel('Response')
plt.title('Gaussian Features vs Responses, Error=' + str(error))
plt.scatter(features, responses, c='blue', label='True Responses')
plt.scatter(features, pred, c='orange', label='Predicted Responses')
plt.legend()
```

[3]:
```
<matplotlib.legend.Legend at 0x7fe83d059f10>
```

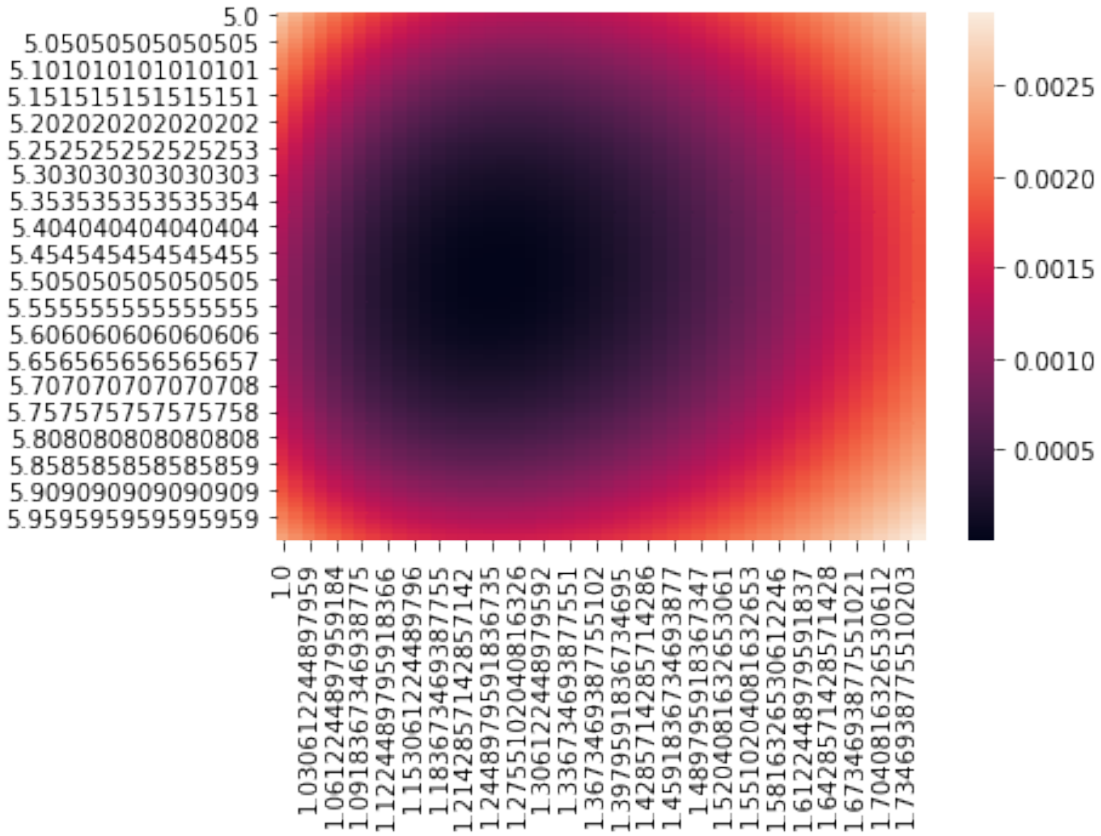


1.4 Experiment 2: Refined Grid Search - Gaussian Distribution

1.4.1 Grid Search - Refinement Pass

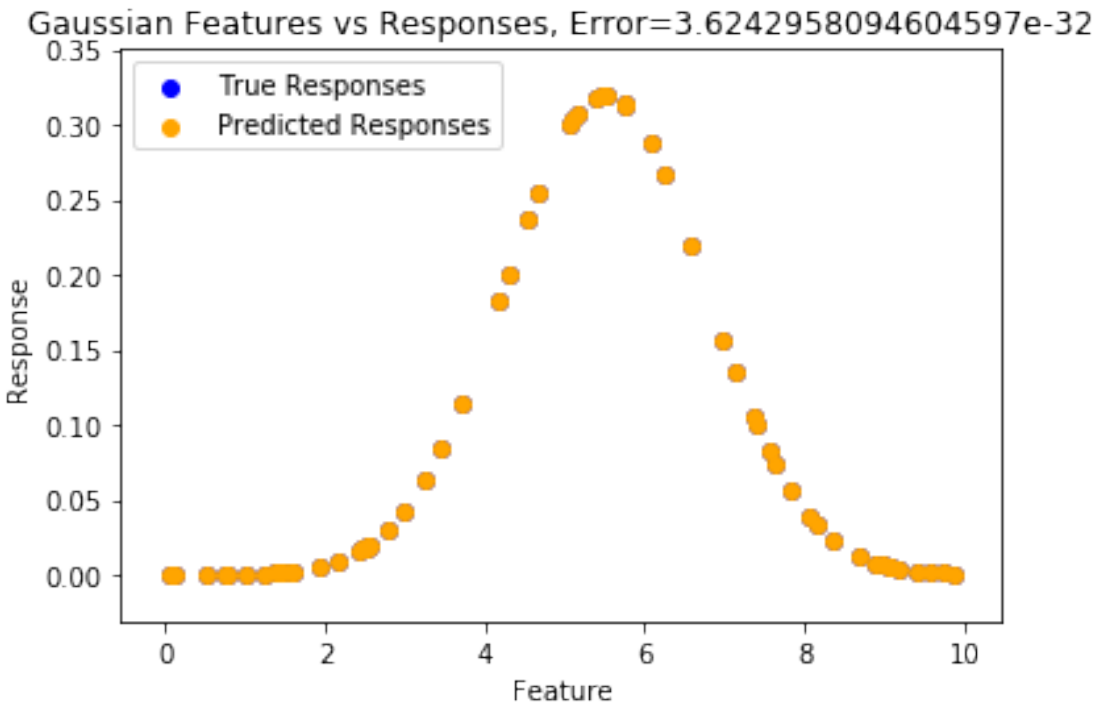
[4]:
```
mus = np.linspace(5, 6, num=100)
sigmas = np.linspace(1, 1.75, num=50)

errors = np.ndarray((len(mus), len(sigmas)))
for i in range(len(mus)):
    mu = mus[i]
    for j in range(len(sigmas)):
        sigma = sigmas[j]
        error = gcf.cost(np.array([mu, sigma]))
        errors[i][j] = error

errors_df = pd.DataFrame(data=errors, index=mus, columns=sigmas)
heatmap(errors_df)
```

[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe837787b90>
```



[5]:
```
params = np.array([5.5, 1.25])
pred = gcf._predict(features, params)
error = gcf.cost(params)

plt.xlabel('Feature')
plt.ylabel('Response')
plt.title('Gaussian Features vs Responses, Error=' + str(error))
plt.scatter(features, responses, c='blue', label='True Responses')
plt.scatter(features, pred, c='orange', label='Predicted Responses')
plt.legend()
```

[5]:
```
<matplotlib.legend.Legend at 0x7fe8436400d0>
```



[6]:
```
min_ind = np.unravel_index(np.argmin(errors, axis=None), errors.shape)
print("Error = " + str(errors[min_ind[0]][min_ind[1]]))
print("Mu = " + str(mus[min_ind[0]]))
print("Sigma = " + str(sigmas[min_ind[1]]))
```

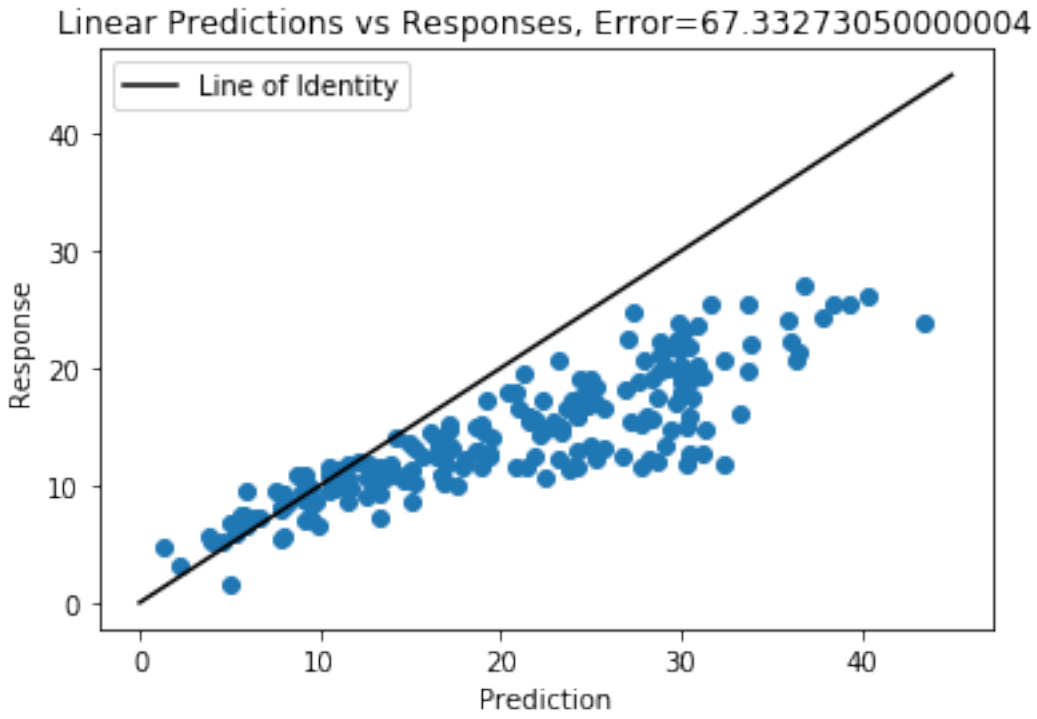
```
Error = 4.3307051270672576e-07
Mu = 5.505050505050505
Sigma = 1.2448979591836735
```

1.5 Experiment 3: "Blind" Grid Search - Multivariate Linear Model

[7]:
```
from cost_functions_stub import LinearCostFunction

advert = pd.read_csv('advertising.csv').drop('Unnamed: 0', axis=1)
advert = advert.to_numpy()

features = advert[:, :4]
sales = advert[:, 4]

lcf = LinearCostFunction(features, sales)
params = np.array([0.1, 0.1, 0.1, 0.1])
pred = lcf._predict(features, params)
error = lcf.cost(params)
```

```
plt.xlabel('Prediction')
plt.ylabel('Response')
plt.title('Linear Predictions vs Responses, Error=' + str(error))
plt.scatter(pred, sales)

x = np.linspace(0, 45)
plt.plot(x, x, c="black", label="Line of Identity")
plt.legend()
```

[7]: <matplotlib.legend.Legend at 0x7fe837661710>



[8]:
```
b0s = np.linspace(-1, 2, num=50)
b1s = np.linspace(-1, 1, num=50)
b2s = np.linspace(-1, 1, num=50)
b3s = np.linspace(-1, 1, num=50)

errors = np.ndarray((len(b0s), len(b1s), len(b2s), len(b3s)))
for i in range(len(b0s)):
    b0 = b0s[i]
    for j in range(len(b1s)):
        b1 = b1s[j]
        for k in range(len(b2s)):
```

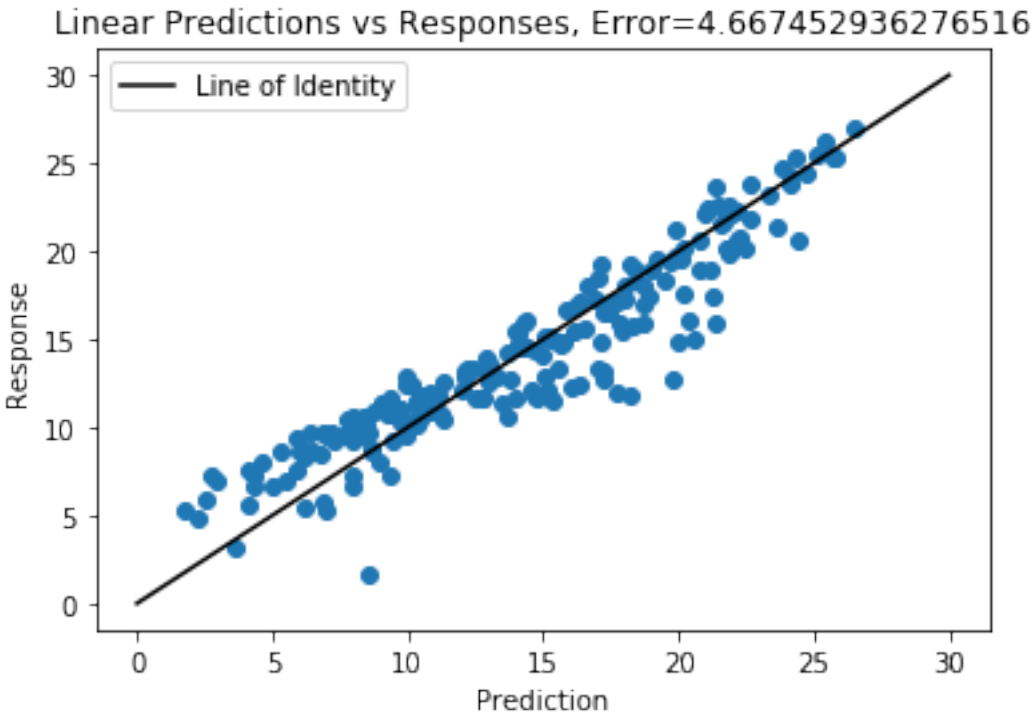
```
            b2 = b2s[k]
            for l in range(len(b3s)):
                b3 = b3s[l]
                error = lcf.cost(np.array([b0, b1, b2, b3]))
                errors[i][j][k][l] = error
```

[9]:
```
min_ind = np.unravel_index(np.argmin(errors, axis=None), errors.shape)
print("Error = " + str(errors[min_ind[0]][min_ind[1]][min_ind[2]][min_ind[3]]))
print("B0 = " + str(b0s[min_ind[0]]))
print("B1 = " + str(b1s[min_ind[1]]))
print("B2 = " + str(b2s[min_ind[2]]))
print("B3 = " + str(b3s[min_ind[3]]))
```

```
Error = 4.667452936276516
B0 = 1.387755102040816
B1 = 0.06122448979591821
B2 = 0.18367346938775508
B3 = -0.020408163265306145
```

[10]:
```
params = np.array([b0s[min_ind[0]], b1s[min_ind[1]], b2s[min_ind[2]],␣
 ↪b3s[min_ind[3]]])
pred = lcf._predict(features, params)
error = lcf.cost(params)

plt.xlabel('Prediction')
plt.ylabel('Response')
plt.title('Linear Predictions vs Responses, Error=' + str(error))
plt.scatter(pred, sales)

x = np.linspace(0, 30)
plt.plot(x, x, c="black", label="Line of Identity")
plt.legend()
```

[10]:
```
<matplotlib.legend.Legend at 0x7fe8375e6810>
```


Linear Predictions vs Responses, Error=4.667452936276516
Line of Identity
Response
Prediction
0
5
10
15
20
25
30