Date......../........./.........

Page...........................

Archit Matta — Homework 2

am5500 — 28/2/21

(1) (a) Since only one part of the objective depends on π

$$\hat{\pi} = \arg\max_{\pi} \sum_{i=1}^{n} \ln p(y_i \mid \pi)$$

$$y_i \overset{iid}{\sim} \text{Bern}(\pi) \Rightarrow p(y_i \mid \pi) = \pi^{y_i}(1-\pi)^{1-y_i}$$

$$\hat{\pi} = \arg\max_{\pi} \sum_{i=1}^{n} \ln\left(\pi^{y_i}(1-\pi)^{1-y_i}\right)$$

Taking derivative w.r.t π and setting it to 0

$$\sum_{i=1}^{n} \frac{d}{d\pi}\left(\ln\left(\pi^{y_i}(1-\pi)^{1-y_i}\right)\right) = 0$$

$$\Rightarrow \sum_{i=1}^{n} \frac{d}{d\pi}\left(y_i \ln\pi + (1-y_i)\ln(1-\pi)\right) = 0$$

$$\Rightarrow \frac{\sum y_i}{\pi} - \frac{\sum(1-y_i)}{1-\pi} = 0$$

$$\Rightarrow \boxed{\hat{\pi} = \frac{\sum_{i=1}^{n} y_i}{n}}$$

(b) Since only one part of the objective depends on λ

$$\hat{\lambda}_{0,d} = \arg\max_{\lambda}\left[\ln p(\lambda_{0,d}) + \sum_{i=1}^{n} \ln p(x_{i,d} \mid \lambda_{0,d})\right]$$

$$\lambda_{0,d} \overset{iid}{\sim} \text{Gamma}(2,1) \Rightarrow p(\lambda_{0,d}) = \frac{\lambda_{0,d}\, e^{-\lambda_{0,d}}}{1}$$

$$x_{id} \mid y_0 \sim \text{Pois}(\lambda_{0,d}) \Rightarrow p(x_{id} \mid y_0) = \frac{e^{-\lambda_{0,d}}(\lambda_{0,d})^{x_{i,d}}}{x_{i,d}!}$$

Taking derivative w.r.t $\lambda_{0,d}$ and setting it to 0 :

$$\frac{d}{d\lambda_{0,d}}\left[\ln\left(\lambda_{0,d}\, e^{-\lambda_{0,d}}\right) + \sum_{i=1} \ln\left(\frac{e^{-\lambda_{0,d}}\,(\lambda_{0,d})^{x_{i,d}}}{x_{i,d}!}\right)\right] = 0$$

$$\Rightarrow \frac{1}{\lambda_{0,d}} - 1 + \sum_{i=1}^{n}\left[\frac{x_{i,d}}{\lambda_{0,d}} + (-1)\cdot I\{y_i=0\}\right] = 0$$

$$\Rightarrow \frac{1 + \sum_{i=1}^{n} x_{i,d}\, I\{y_i=0\}}{\lambda_{0,d}} = 1 + \sum_{i=1}^{n} I\{y_i=0\}$$

$$\Rightarrow \boxed{\hat{\lambda}_{0,d} = \frac{1 + \sum_{i=1}^{n} x_{i,d}\{y_i=0\}}{1 + \sum_{i=1}^{n} I\{y_i=0\}}}$$

Since our expression is symmetric w.r.t $\lambda_{0,d}$ & $\lambda_{1,d}$ $\Rightarrow$

$$\boxed{\hat{\lambda}_{y,d} = \frac{1 + \sum_{i=1}^{n} x_{i,d}\, I'\{y_i=y\}}{1 + \sum_{i=1}^{n} I\{y_i=y\}}}$$

(2) (a) $L(\omega) \approx L'(\omega) \equiv L(\omega_t) + (\omega - \omega_t)^T \nabla L(\omega_t) + \frac{1}{2}(\omega - \omega_t)^T \nabla^2 L(\omega_t)(\omega - \omega_t)$

$$\omega_{t+1} = \arg\max_{\omega} L'(\omega)$$

Taking the derivative and equating it to 0

$$\frac{\partial}{\partial \omega}\left[L(\omega_t) + (\omega - \omega_t)^T \nabla L(\omega_t) + \frac{1}{2}(\omega - \omega_t)^T \nabla^2 L(\omega_t)(\omega - \omega_t) \right]$$

$$\Rightarrow 0 + 1 \times \nabla L(\omega_t) + \frac{1}{2} \times 2 \times (\omega - \omega_t)\nabla^2 L(\omega_t) = 0$$

$$\Rightarrow (\omega - \omega_t)\nabla^2 L(\omega_t) = -\nabla L(\omega_t)$$

$$\Rightarrow (\omega - \omega_t) = -[\nabla^2 L(\omega_t)]^{-1} \nabla L(\omega_t)$$

$$\Rightarrow \boxed{\omega_{t+1} = \omega_t - [\nabla^2 L(\omega_t)]^{-1} \nabla L(\omega_t)}$$

$$\nabla L(\omega) = X^T \cdot (Y(1-\sigma))$$

Taking the derivative w.r.t. ω

$$\nabla^2 L(\omega) = X^T (1-\sigma)^T (-\sigma)^T X$$

$$= -X^T (1-\sigma)^T \sigma^T X$$

$$= -[\sigma(1-\sigma)X]^T X$$

Problem 2

Part a - Naive Bayes

In [11]:

```
accuracy = "\033[1m" + 'Prediction Accuracy = ' + str((df_confusion_nb[0][0] 
print(accuracy)
df_confusion_nb
```

Prediction Accuracy = 0.8693478260869565

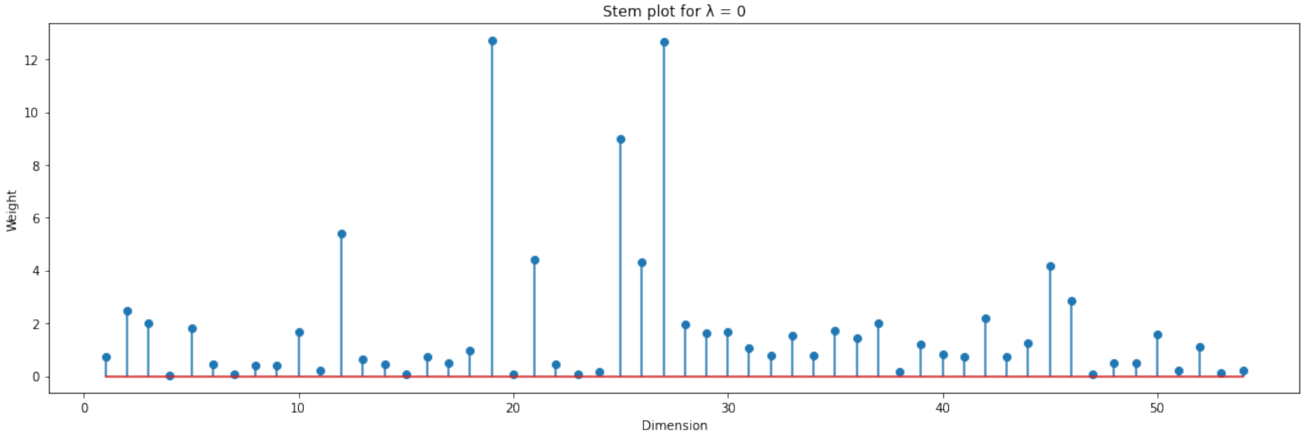
Out[11]:

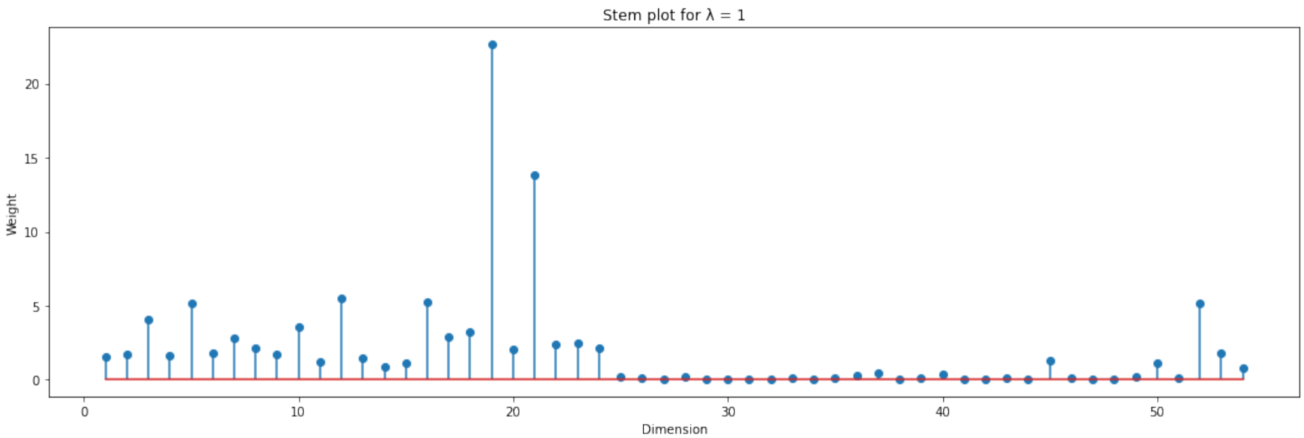
Predicted	0	1
Actual		
0	2296	491
1	110	1703

Part b - Naive Bayes

In [13]:

```
fig = plt.figure()
fig.set_figheight(12)
fig.set_figwidth(18)
plt.rcParams["figure.figsize"] = (20,10)
ax1 = fig.add_subplot(211)
ax2 = fig.add_subplot(212)
ax1.set_title('Stem plot for \u03BB = 0')
ax1.set_xlabel('Dimension')
ax1.set_ylabel('Weight')
ax1.stem(numbers, lamda_0_average)
ax2.set_title('Stem plot for \u03BB = 1')
ax2.set_xlabel('Dimension')
ax2.set_ylabel('Weight')
ax2.stem(numbers, lamda_1_average)
plt.show()
```





In [15]:

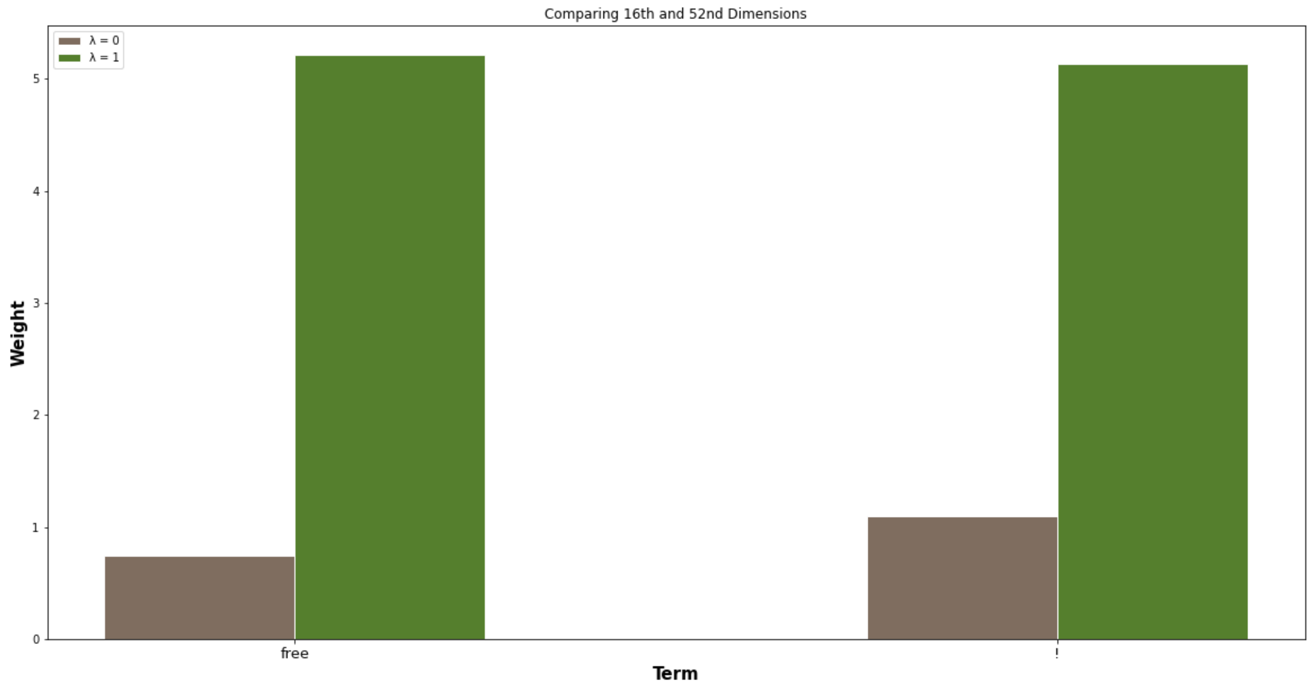
```
barWidth = 0.25

# Set position of bar on X axis
r1 = np.arange(len(lamda_0_subset))
r2 = [x + barWidth for x in r1]

# Make the plot
plt.bar(r1, lamda_0_subset, color='#7f6d5f', width=barWidth, edgecolor='white
plt.bar(r2, lamda_1_subset, color='#557f2d', width=barWidth, edgecolor='white

# Add xticks on the middle of the group bars
plt.title('Comparing 16th and 52nd Dimensions')
plt.xlabel('Term', fontweight='bold', fontsize=15)
plt.ylabel('Weight', fontweight='bold', fontsize=15)
plt.xticks([r + (barWidth/2) for r in range(len(lamda_0_subset))], label_subs

# Create legend & Show graphic
plt.legend()
plt.show()
```



16th and 52nd Dimension refer to the terms 'free' and '!'. These terms are seen having more weight for the case $\lambda = 1$ than $\lambda = 0$. This is expected as the terms 'free' and '!' are more likely to present in spam emails.

Part c - Logistic Regression

In [20]:

```
accuracy = "\033[1m" + 'Prediction Accuracy = ' + str((df_confusion_sa[-1][-1
print(accuracy)
df_confusion_sa
```

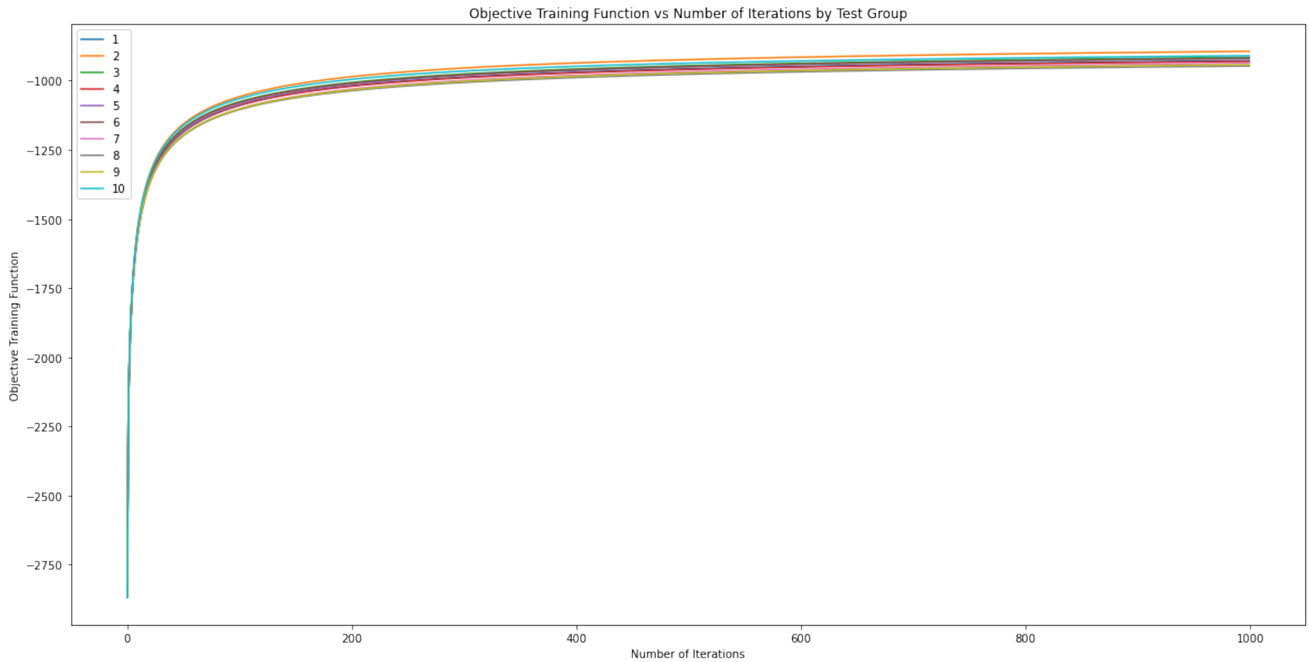
Prediction Accuracy = 0.927608695652174

Out[20]:

Predicted	-1.0	1.0
Actual		
-1	2615	172
1	161	1652

In [21]:

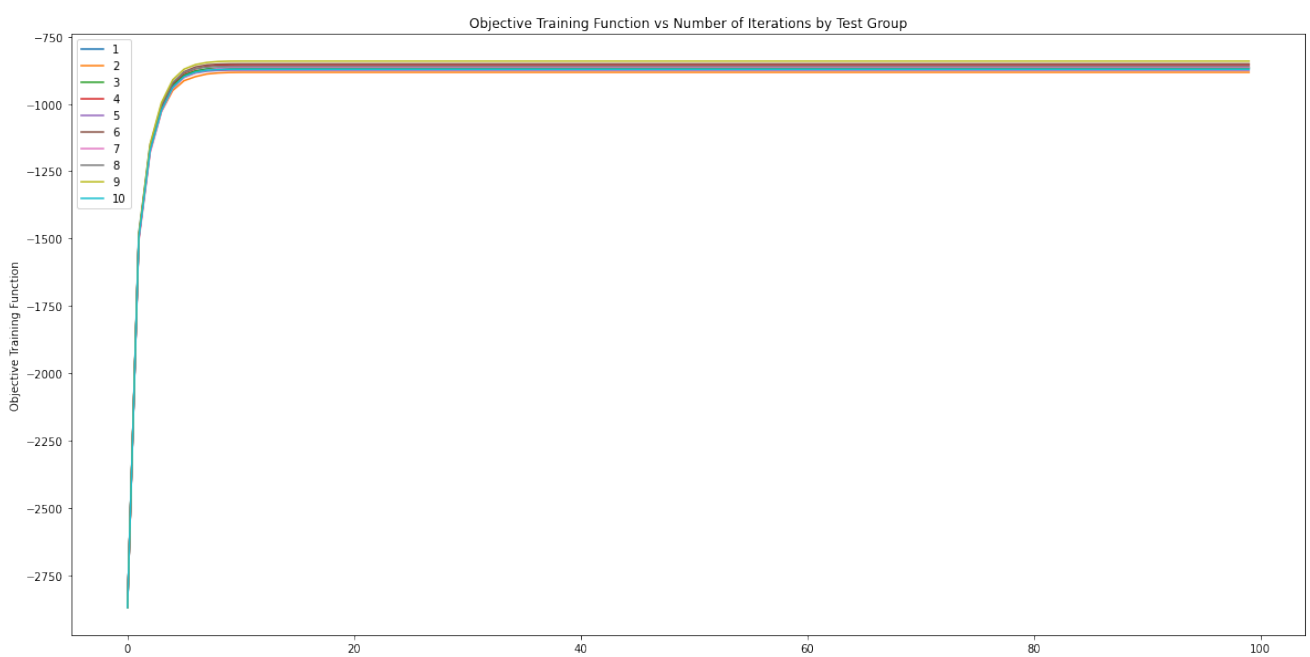
```
objective_all_sa_df = pd.DataFrame(objective_all_sa)
objective_all_sa_df.columns = [1,2,3,4,5,6,7,8,9,10]
ax = plt.gca()
ax.set_title('Objective Training Function vs Number of Iterations by Test Gro
ax.set_xlabel('Number of Iterations')
ax.set_ylabel('Objective Training Function')
objective_all_sa_df.plot(kind='line', ax = ax)
plt.show()
```



Part - d Newton's Method

In [25]:

```
objective_all_nm_df = pd.DataFrame(objective_all_nm)
objective_all_nm_df.columns = [1,2,3,4,5,6,7,8,9,10]
ax = plt.gca()
ax.set_title('Objective Training Function vs Number of Iterations by Test Gro
ax.set_xlabel('Number of Iterations')
ax.set_ylabel('Objective Training Function')
objective_all_nm_df.plot(kind='line', ax = ax)
plt.show()
```



Part - e Newton's Method

In [24]:

```
accuracy = "\033[1m" + 'Prediction Accuracy = ' + str((df_confusion_nm[-1][-1
print(accuracy)
df_confusion_nm
```

Prediction Accuracy = 0.9221739130434783

Out[24]:

Predicted	-1.0	1.0
Actual		
-1	2646	141
1	217	1596

Question 3

Part - a Gaussian Process

In [32]:

```
rmse_matrix
```

Out[32]:

b	5	7	9	11	13	15
Sigma^2						
0.1	1.938372	1.940853	1.941556	1.941658	1.941612	1.94159
0.2	1.933602	1.934726	1.935956	1.93735	1.938869	1.940463
0.3	1.930739	1.933197	1.936128	1.939142	1.942123	1.945042
0.4	1.930309	1.934424	1.938905	1.943281	1.947501	1.951609
0.5	1.931505	1.937161	1.942999	1.948611	1.954042	1.959397
0.6	1.933778	1.940821	1.947915	1.954755	1.961467	1.968194
0.7	1.936804	1.945119	1.953441	1.961562	1.969661	1.977894
0.8	1.940386	1.949906	1.959475	1.968956	1.978548	1.988404
0.9	1.944405	1.955098	1.965958	1.976881	1.988065	1.999638
1.0	1.948782	1.960644	1.972848	1.985292	1.998149	2.011511

Part - b Gaussian Process

In [33]:
```
b_min = 0
sigma_min = 0
min = rmse_matrix.to_numpy().min()
for b_i in b:
    for sigma_i in sigma:
        if rmse_matrix[b_i][sigma_i] == min:
            b_min = b_i
            sigma_min = sigma_i
            break
print('Minimum RMSE :', min)
print('Ideal b :', b_min)
print('Ideal \u03C3^2 :', sigma_min)
```

```
Minimum RMSE : 1.9303087263142182
Ideal b : 5
Ideal σ^2 : 0.4
```

The minimum RMSE in Assignment 1 was 2.100110972109874 while the minimum RMSE using the Gaussian Process Approach is 1.9303087263142182. As we can see, Gaussian Process has a lower RMSE value and is thus better at making predictions. Some drawbacks of using this approach are - Higher Computation Time, Higher Prediction Variance and Difficulty in Interpreting Feature Importance (Due to Kernel based Approach).

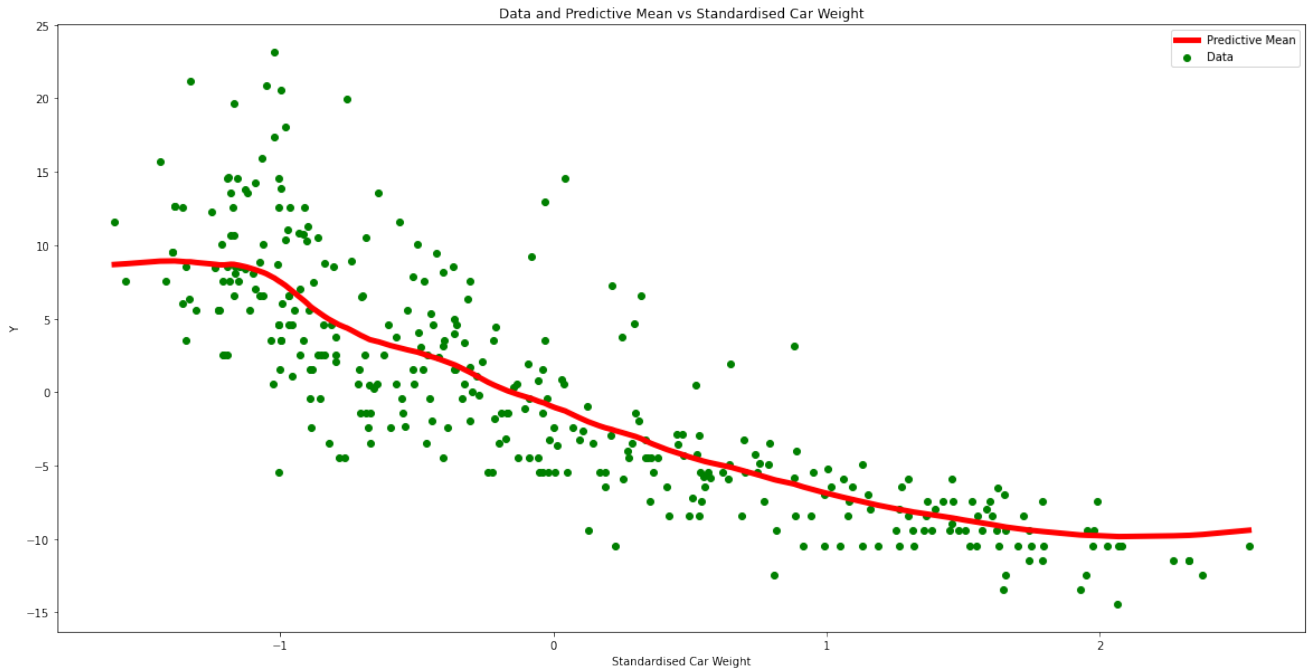
Part - c Gaussian Process

In [35]:
```
rmse_matrix_weight
```

Out[35]:

b	5
Sigma^2	
2	4.199018

In [36]:
```
plt.scatter(df_3_x_train_np[:, (3)], df_3_y_train_np, c = 'green', label = 'D
plt.plot(df_3_x_train_np[:, (3)][np.argsort(df_3_x_train_np[:, (3)].reshape(-
plt.legend()
plt.title('Data and Predictive Mean vs Standardised Car Weight')
plt.xlabel('Standardised Car Weight')
plt.ylabel('Y')
plt.show()
```


Data and Predictive Mean vs Standardised Car Weight
Predictive Mean
Data
Y
Standardised Car Weight
25
20
15
10
5
0
−5
−10
−15
−1
0
1
2