COMS W4721 Spring 2020 Homework 1: Maximum Likelihood, Linear Regression, Bias-Variance Tradeoffs

Shuyu Huang (sh3967 @columbia.edu)

January 26, 2020

For this assignment I collaborated with the following people. Blank entries in this table means that I have worked on the corresponding parts on my own.

Problem	Collaborators with their UNIs	Part
Problem 2	Collaborator 1 (uni1@columbia.edu)	Part (a), (b)
	Collaborator 2 (uni2@columbia.edu)	Part (b)
Problem 3		
Problem 4	Collaborator 3 (uni3@columbia.edu)	Part (c)

Mingrui Liu ml4404 @ columbia.edu Problem 2. 3. 4

Zhiyi Guo zg2350 @ columbia.edu Problem 4.

Problem 1

Introduction: Please briefly describe your academic/career goals and your expectations about the course.

Answer:

Academic / Career Goal:

I hope to work in the industry as a data scientist / analyst, especially focus on Machine Learning. I do hope to utilize my insights in data in my career so that I do want to work at a tech position.

Expectations about the course:

Though I know that this course primarily focus on theoretical basis of Machine Learning, I hope it can somehow relate to practices. Such as when and how to utilize a specific methods, and also comparisons between similar methods.

Problem 2

In this problem we will review the principle of *maximum likelihood estimation*.

(a) We are given a coin which falls its heads up with probability $0 < \theta < 1$. Each throw is a Bernoulli random variable $x = \begin{cases} 1, \text{if falls heads up} \\ 0, \text{if falls tails up} \end{cases}$.

For a Bernoulli random variable x: the probability mass function of x is given by:

$$\Pr(x; \theta) = \theta^x (1-\theta)^{(1-x)}$$

Suppose we repeat the coin toss N times to collect the data $\{x^{(i)}\}_{i=1}^N$. Write the log likelihood function $\ln \mathcal{L}(\theta; \{x^{(i)}\}_{i=1}^N)$ and the maximum likelihood estimation of θ, $\hat{\theta}_{\mathsf{MLE}}$.

Solution:

$$L(\theta; \{x^{(i)}\}_{i=1}^N) = \prod_{i=1}^N P(\theta; x^{(i)})$$

$$= \prod_{i=1}^N \theta^{x^{(i)}} (1-\theta)^{(1-x^{(i)})}$$

$$= \theta^{\sum_{i=1}^N x^{(i)}} (1-\theta)^{\sum_{i=1}^N (1-x^{(i)})}$$

$$\ell(\theta; \{x^{(i)}\}_{i=1}^N) = \sum_{i=1}^N x^{(i)} \log \theta + \sum_{i=1}^N (1-x^{(i)}) \log(1-\theta)$$

To find $\hat{\theta}_{MLE}$:

$$\frac{\partial \ell}{\partial \theta} = \frac{\sum_{i=1}^N x^{(i)}}{\theta} - \frac{\sum_{i=1}^N (1-x^{(i)})}{1-\theta} = \frac{\sum_{i=1}^N x^{(i)}}{\theta} - \frac{N - \sum_{i=1}^N x^{(i)}}{1-\theta} \overset{\text{set}}{=} 0$$

$$\frac{\sum_{i=1}^N x^{(i)}}{\theta} = \frac{N - \sum_{i=1}^N x^{(i)}}{1-\theta}$$

$$(1-\theta) \sum_{i=1}^N x^{(i)} = N\theta - \theta \sum_{i=1}^N x^{(i)}$$

$$\sum_{i=1}^N x^{(i)} = N\theta$$

$$\hat{\theta}_{MLE} = \frac{\sum_{i=1}^N (x^{(i)})}{N} = \bar{x}$$

(b) Suppose instead we are given a die with K sides with each side falling its heads up with probability $0 < \theta_k < 1$. While we can represent the result of a throw using a categorical variable $x \in \{1, \cdots, K\}$, we can use 1-of-K encoding:

$$x = k \quad \Leftrightarrow \quad \mathbf{x} = [0, \cdots, \underbrace{1}_{\substack{\text{k-th} \\ \text{position} \\ := x_k}}, \cdots, 0]$$

Each throw is a categorical random variable $\mathbf{x} \sim \text{Categorical}(\theta_1, \cdots, \theta_K)$ such that $\Pr(x_k = 1; \theta) = \theta_k$ and $\sum_{k=1}^{K} \theta_k = 1$. The probability mass function at $\mathbf{x}$ is given by:

$$\Pr(\mathbf{x}; \theta_1, \cdots, \theta_K) = \prod_{k=1}^{K} \theta_k^{x_k}$$

Suppose we throw the die N times and obtain the data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$. Write the log likelihood function $\ln \mathcal{L}(\theta; \{\mathbf{x}^{(i)}\}_{i=1}^{N})$ and the maximum likelihood estimation of θ, $\hat{\theta}_{\mathsf{MLE}}$.

Hint: Once you obtain the log likelihood function $\mathcal{L}(\theta; \{\mathbf{x}^{(i)}\}_{i=1}^{N})$, you will need to add the Lagrangian multiplier part that takes the probability sum constraint $\sum_{k=1}^{K} \theta_k = 1$. For $\lambda \in \mathbb{R}$, the Lagrangian is:

$$\ln \mathcal{L}_\lambda(\theta; \{\mathbf{x}^{(i)}\}_{i=1}^{N}) = \ln \mathcal{L}(\theta; \{\mathbf{x}^{(i)}\}_{i=1}^{N}) + \lambda \left(1 - \sum_{k=1}^{K} \theta_k\right)$$

Take the partial derivative of $\ln \mathcal{L}(\theta; \{\mathbf{x}^{(i)}\}_{i=1}^{N})$ with respect to each θ_k and λ, set them to zero, and solve for each variable of θ_k. Appendix E of Bishop's book may be helpful for this problem.

Solution:

$$L(\theta; \{x^{(i)}\}_{i=1}^N) = \prod_{i=1}^N P(x_{(i)}; \theta_1, \cdots, \theta_K)$$
$$= \prod_{i=1}^N \prod_{k=1}^K \theta_k^{x_k}$$
$$= \theta_1^{\sum_{i=1}^N x_1^{(i)}} \cdots \theta_K^{\sum_{i=1}^N x_K^{(i)}}$$

$$\ell(\theta; \{x^{(i)}\}_{i=1}^N) = \sum_{i=1}^N x_1^{(i)} \log\theta_1 + \cdots + \sum_{i=1}^N x_K^{(i)} \log\theta_K$$
$$= \sum_{k=1}^K \sum_{i=1}^N x_k^{(i)} \log\theta_k$$

By Lagrangian,

$$\ell(\theta; \{x^{(i)}\}_{i=1}^N) = \ell(\theta; \{x^{(i)}\}_{i=1}^N) + \lambda\left(1 - \sum_{k=1}^K \theta_k\right)$$
$$= \sum_{k=1}^K \sum_{i=1}^N x_k^{(i)} \log\theta_k + \lambda\left(1 - \sum_{k=1}^K \theta_k\right)$$

$$\frac{\partial \ell}{\partial \lambda} = 1 - \sum_{k=1}^K \theta_k \overset{\text{set}}{=} 0 \rightarrow \boxed{\sum_{k=1}^K \hat{\theta}_k = 1} \quad ①$$

$$\frac{\partial \ell}{\partial \theta_k} = \frac{\sum_{i=1}^N x_k^{(i)}}{\theta_k} - \lambda = 0 \rightarrow \boxed{\hat{\theta}_k = \frac{1}{\lambda}\sum_{i=1}^N x_k^{(i)}} \quad ②$$

$$L(\theta; \{x^{(i)}\}_{i=1}^N) = \prod_{i=1}^N P(x_{(i)}; \theta_1, \cdots, \theta_K)$$
$$= \prod_{i=1}^N \prod_{k=1}^K \theta_k^{x_k}$$
$$= \theta_1^{\sum_{i=1}^N x_1^{(i)}} \cdots \theta_K^{\sum_{i=1}^N x_K^{(i)}}$$

$$\ell(\theta; \{x^{(i)}\}_{i=1}^N) = \sum_{i=1}^N x_1^{(i)} \log\theta_1 + \cdots + \sum_{i=1}^N x_K^{(i)} \log\theta_K$$
$$= \sum_{k=1}^K \sum_{i=1}^N x_k^{(i)} \log\theta_k$$

By Lagrangian,

$$\ell(\theta; \{x^{(i)}\}_{i=1}^N) = \ell(\theta; \{x^{(i)}\}_{i=1}^N) + \lambda\left(1 - \sum_{k=1}^K \theta_k\right)$$
$$= \sum_{k=1}^K \sum_{i=1}^N x_k^{(i)} \log\theta_k + \lambda\left(1 - \sum_{k=1}^K \theta_k\right)$$

$$\frac{\partial \ell}{\partial \lambda} = 1 - \sum_{k=1}^K \theta_k \overset{\text{set}}{=} 0 \rightarrow \boxed{\sum_{k=1}^K \hat{\theta}_k = 1} \quad ①$$

$$\frac{\partial \ell}{\partial \theta_k} = \frac{\sum_{i=1}^N x_k^{(i)}}{\theta_k} - \lambda = 0 \rightarrow \boxed{\hat{\theta}_k = \frac{1}{\lambda}\sum_{i=1}^N x_k^{(i)}} \quad ②$$

Side Note:

K sides w/ prob $0 < \theta_K < 1$

represent result:

$X = k \Leftrightarrow x = [0, \cdots, \underset{X_k}{1}, \cdots, 0]$

the roll returns k

Each Throw:

$x \sim$ Categorical $(\theta_1, \cdots, \theta_K)$

$P(X_k = 1; \theta) = \theta_k$

↳ the roll returns 1, the prob of roll $k = \theta_k$

$\sum \theta_k = 1 \rightarrow$ Total prob of one throw

$P(x; \theta_1, \cdots \theta_K) = \prod_{k=1}^K \theta_k^{x_k}$

θ_K: the prob of roll K

X_k: $\begin{cases} 1 & \text{if the roll returns } k \\ 0 & \text{else} \end{cases}$

(c) In class we derived a MLE estimator for the univariate Gaussian assumption. For $\{x^{(i)} \in \mathbb{R}\}_{i=1}^N$ i.i.d, we chose $p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$ and solved for $\hat{\mu}_{\mathsf{MLE}}$ and $\hat{\sigma}^2_{\mathsf{MLE}}$. Repeat this exercise for the multivariate case: now assume $\{\mathbf{x}^{(i)} \in \mathbb{R}^d\}_{i=1}^N$ and choose $p(\mathbf{x}|\mu,\mathbf{\Sigma}) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{\det(\Sigma)}} \exp\left(\frac{-(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)}{2}\right)$. Write the log likelihood function $\ln \mathcal{L}(\{\mu,\Sigma\};\{x^{(i)}\}_{i=1}^N)$ and solve for $\hat{\mu}_{\mathsf{MLE}}$ and $\hat{\Sigma}_{\mathsf{MLE}}$.

Solution:

$$p(\vec{x} \mid \mu, \Sigma) = (2\pi)^{-\frac{d}{2}} \det(\Sigma)^{-\frac{1}{2}} \exp\left(\frac{-(\vec{x}-\mu)^T(\Sigma)^{-1}(x-\mu)}{2}\right)$$

$$L(\vec{x} \mid \mu, \Sigma) = (2\pi)^{-\frac{Nd}{2}} \det(\vec{\Sigma})^{-\frac{N}{2}} \exp\left(\sum_{i=1}^{N} \frac{-(\vec{x_i}-\mu)^T\vec{\Sigma}^{-1}(\vec{x_i}-\mu)}{2}\right)$$

$$\ell(\vec{x}, \mu, \Sigma) = -\frac{Nd}{2}\log(2\pi) - \frac{N}{2}\log(\det(\vec{\Sigma})) - \sum_{i=1}^{N} \frac{(\vec{x_i}-\mu)^T\vec{\Sigma}^{-1}(\vec{x_i}-\mu)}{2}$$

$$\frac{\partial \ell}{\partial \mu} = \sum_{i=1}^{N} \frac{1}{2} \cdot 2(\vec{x_i}-\mu)\vec{\Sigma}^{-1} = \vec{\Sigma}^{-1}\sum_{i=1}^{N}(\vec{x_i}-\mu) = 0$$

$$\sum^{N} \vec{x_i} - N\mu = 0$$

$$\boxed{\hat{\mu}_{MLE} = \frac{1}{N}\sum_{i=1}^{N}\vec{x_i}}$$

$$\frac{\partial \ell}{\partial \Sigma^{-1}} = -\frac{N}{2}\frac{1}{\det\vec{\Sigma}} - \frac{1}{2}\sum_{i=1}^{N}(\vec{x_i}-\mu)^T(x_i-\mu) = 0$$

note that $\det(\vec{\Sigma})^{-1} = |\vec{\Sigma}|^{-1} = |\vec{\Sigma}^{-1}|$

$$\frac{N}{2}\vec{\Sigma} - \frac{1}{2}\sum_{i=1}^{N}(\vec{x_i}-\mu)^T(x_i-\mu) = 0$$

$$\boxed{\hat{\Sigma}_{MLE} = \frac{1}{N}\sum_{i=1}^{N}(\vec{x_i} - \hat{\mu}_{MLE})^T(\vec{x_i} - \hat{\mu}_{MLE})}$$

Problem 3

In this problem we will use a numerical optimization routine to obtain maximum likelihood estimate of parameters.

Suppose $\{x^{(i)} \in \mathbb{R}\}_{i=1}^{N}$ with $x^{(i)} \sim p(x; x_0, \gamma)$ defined as:

$$p(x; x_0, \gamma) = \frac{1}{\pi \exp(\gamma)\left[1 + \left(\frac{x - x_0}{\exp(\gamma)}\right)^2\right]}$$

(a) Prove that $p(x; x_0, \gamma)$ is a probability density function.

Solution:

$$P(x;\ x_0\ ;\ \gamma) = \frac{1}{\pi \exp(\gamma)\left[1 + \left(\frac{x - x_0}{\exp(\gamma)}\right)^2\right]}$$

$$\int_{-\infty}^{\infty} P(x; x_0; \gamma)\, dx$$

$$= \int_{-\infty}^{\infty} \frac{1}{\pi \exp(\gamma)\left[1 + \left(\frac{x - x_0}{\exp(\gamma)}\right)^2\right]} dx$$

$$= \int_{-\infty}^{\infty} \pi^{-1} \exp(-\gamma)\left[1 + \left(\frac{x - x_0}{\exp(\gamma)}\right)^2\right]^{-1} dx$$

$$= \pi^{-1} \tan^{-1}\left(\frac{x - x_0}{\exp(\gamma)}\right)\Big|_{-\infty}^{\infty}$$

$$= \pi^{-1}\left(\frac{\pi}{2} + \frac{\pi}{2}\right) = 1$$

(b) Prove that the mean $\mathbb{E}_{x \sim p(x; x_0, \gamma)}[x]$ is undefined.

Solution:

$$\mathbb{E}_{X \sim P(x; x_0; \gamma)}[X] = \int_{-\infty}^{\infty} x \frac{1}{\pi \exp(\gamma)\left[1 + \left(\frac{x - x_0}{\exp(\gamma)}\right)^2\right]} dx$$

$$= \int_{-\infty}^{\infty} \frac{1}{\pi \exp(\gamma)} \frac{x}{1 + \left(\frac{x - x_0}{\exp(\gamma)}\right)^2}$$

let $\frac{x - x_0}{\exp(\gamma)} = u$

$du = \frac{1}{\exp(\gamma)} dx$

$dx = \exp(\gamma)\, du$

$$= \int_{-\infty}^{\infty} \frac{1}{\pi \exp(\gamma)}\left[\frac{x - x_0}{1 + u^2} + \frac{x_0}{1 + u^2}\right] dx$$

$$= \int_{-\infty}^{\infty} \frac{1}{\pi} \frac{x - x_0}{\exp(\gamma)(1 + u^2)}\, dx + x_0 \underbrace{\int_{-\infty}^{\infty} \frac{1}{\pi \exp(\gamma)} \cdot \frac{1}{1 + \left(\frac{x - x_0}{\exp(\gamma)}\right)^2} dx}_{= 1 \text{ by } 3(a)}$$

$$= \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{x - x_0}{\exp(\gamma)} \cdot \frac{1}{1 + u^2} \cdot \exp(\gamma)\, du + x_0$$

$$= \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{u}{1 + u^2} \exp(\gamma)\, du + x_0$$

$$= \frac{\exp(\gamma)}{\pi} \frac{1}{2} \underbrace{\int_{-\infty}^{\infty} \frac{1}{1 + u^2} du^2}_{= \infty} + x_0$$

$\therefore \mathbb{E}[X]$ is also infinite.

$\mathbb{E}[X]$ is therefore not defined

(c) Write the log likelihood function $\ln \mathcal{L}(\{x_0, \gamma\}; \{x^{(i)}\}_{i=1}^N)$ and the expression for $\frac{\partial \ln \mathcal{L}(\{x_0,\gamma\};\{x^{(i)}\}_{i=1}^N)}{\partial x_0}$ and $\frac{\partial \ln \mathcal{L}(\{x_0,\gamma\};\{x^{(i)}\}_{i=1}^N)}{\partial \gamma}$. Plot the log likelihood value as a 3D surface plot: x-axis should run over x_0 and y-axis should run over γ. z-axis should correspond to the log likelihood value at the corresponding (x_0, γ) pair. Include the plot in your writeup. Do the stationary points (solutions to the maximum likelihood equations) have closed form solutions?

Solution:

$$L(\{x_0, \gamma\}; \{x^{(i)}\}_{i=1}^N) = \prod_{i=1}^N P(x_i; x_0, \gamma)$$

$$= \prod_{i=1}^N \frac{1}{\pi \exp(\gamma)\left[1+\left(\frac{x_i - x_0}{\exp(\gamma)}\right)^2\right]}$$

$$= \pi^{-N} \exp(-N\gamma) \prod_{i=1}^N \left[1+\left(\frac{x_i - x_0}{\exp(\gamma)}\right)^2\right]^{-1}$$

$$\ell = \ln L(\{x_0, \gamma\}; \{x^{(i)}\}_{i=1}^N) = -N \log \pi - N\gamma - \sum_{i=1}^N \log\left[1+\left(\frac{x_i - x_0}{\exp(\gamma)}\right)^2\right]$$

$$\frac{\partial \ln L(\{x_0, \gamma\}; \{x^{(i)}\}_{i=1}^N)}{\partial x_0} = \sum_{i=1}^N \frac{2\left(\frac{x_i - x_0}{\exp(\gamma)}\right)\left(\frac{1}{\exp(\gamma)}\right)}{1+\left(\frac{x_i - x_0}{\exp(\gamma)}\right)^2}$$

$$\frac{\partial \ln L(\{x_0, \gamma\}; \{x^{(i)}\}_{i=1}^N)}{\partial \gamma} = -N + \sum_{i=1}^N \frac{2\left(\frac{x_i - x_0}{\exp(\gamma)}\right)^2}{1+\left(\frac{x_i - x_0}{\exp(\gamma)}\right)^2}$$

Let loglikelihood function be 0, there always exists infinite sum if N is large enough.

Therefore, the stationary point doesn't have closed form

In [1]:
```
# Import modules.
import pandas as pd
import numpy as np
from scipy.optimize import minimize
import scipy.stats
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import random
from matplotlib import cm
```

Problem 3

(c)

plot log likelihood value as 3D surface plot:\ x-axis: run over x_0\ y-axis: run over γ\ z-axis: log likelihood value at the corresponding (x_0, γ)

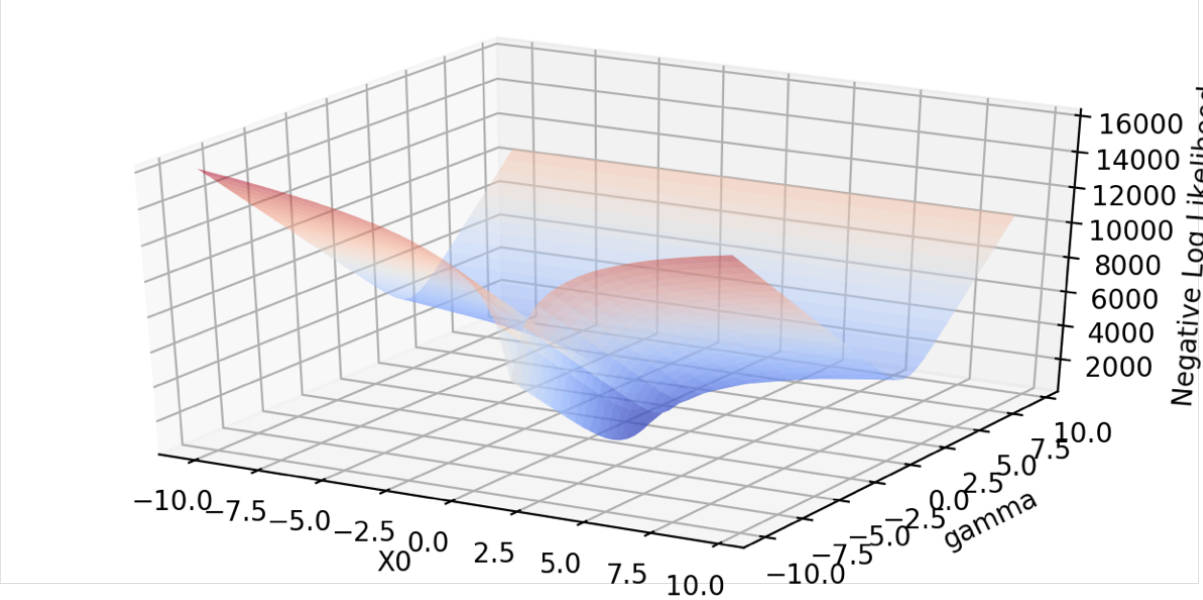
In [2]:
```
df = pd.read_csv('problem3.csv', header=None)
data = np.array(df)
```

In [3]:
```
def neg_loglikelihood(theta, data):
    num_points = data.shape[0]
    x0, gamma = theta
    loglikelihood = -num_points * np.log(np.pi) - num_points * gamma
    loglikelihood = loglikelihood - sum (np.log(1 + ((data-x0)/np.exp(
gamma))**2))
    return - loglikelihood
```

In [4]:
```
# Plot
fig = plt.figure(figsize=(8,4), dpi=200)
ax = fig.add_subplot(111, projection='3d')
x = np.arange(-10, 10, 0.5)
y = np.arange(-10, 10, 0.5)
X, Y = np.meshgrid(x, y)
#print(np.ravel(X).shape)
#print(x.shape)

def fun(x, y, data):
    return np.array([
    neg_loglikelihood((x_value, y_value), data) for x_value, y_value i
n zip(x,y) ])
zs = np.array(fun(np.ravel(X), np.ravel(Y), data))
Z = zs.reshape(X.shape)
print(Z.shape, Y.shape, X.shape)

ax.plot_surface(X, Y, Z, cmap=cm.coolwarm, alpha=0.5)

ax.set_xlabel('X0')
ax.set_ylabel('gamma')
ax.set_zlabel('Negative Log Likelihood')
```

```
(40, 40) (40, 40) (40, 40)
```

Out[4]:
```
Text(0.5, 0, 'Negative Log Likelihood')
```



(d) Write a program to obtain an estimate of $\theta = \{x_0, \gamma\}$ using the dataset **problem3.csv**. If you are using Python, it is helpful to utilize scipy.optimize.minimize function. Choose gradient descent optimizer or a quasi-Newton optimizer such as BFGS. List the optimizer that was chosen for this problem with the initial iterate. Tabulate the coordinates of the iterates of the optmization process and the final converged solution.

Solution:

(d)

obtain an estimator $\theta = (x_0, \gamma)$

In [5]:
```
def neg_loglikelihood_grad(theta, data):
    num_points = data.shape[0]
    x0, gamma = theta
    grad = np.zeros((2,))
    grad[0] = sum(2*(data-x0)/np.exp(2*gamma)/(1+((data-x0)/np.exp(gam
ma))**2))
    grad[1] = - num_points + sum((2*((data-x0)/np.exp(2*gamma)))/(1+((
data-x0)/np.exp(gamma))**2))
    return - grad
```

In [6]:
```
theta = np.array([0.0,1.0])
x0_array=[]
gamma_array=[]
def neg_loglikelihood_update(theta, data):
    global x0_array
    global gamma_array
    num_points = data.shape[0]
    x0, gamma = theta
    loglikelihood = -num_points * np.log(np.pi) - num_points * gamma
    loglikelihood = loglikelihood - sum (np.log(1 + ((data-x0)/np.exp(
gamma))**2))
    x0_array.append(x0)
    gamma_array.append(gamma)
    return - loglikelihood

res = minimize(neg_loglikelihood_update, theta, method='BFGS',
               jac = neg_loglikelihood_grad, args=(data,))
print(res)
```

```
      fun: 2066.661738096842
 hess_inv: array([[ 3.36690228, -3.36759245],
       [-3.36759245,  3.36927259]])
      jac: array([-847.22585653,  152.77414347])
  message: 'Desired error not necessarily achieved due to precision
loss.'
     nfev: 36
      nit: 1
     njev: 24
   status: 2
  success: False
        x: array([0.38377301, 0.06575256])
```

In [7]:
```
pd.DataFrame(data ={'X0':list(x0_array), 'gamma': list(gamma_array)})
```

Out[7]:

	X0	gamma
0	0.000000	1.000000
1	0.383773	0.065753
2	1.235930	-0.786618
3	4.644559	-4.196098
4	1.440424	-0.991162
5	1.618759	-1.169542
6	3.131659	-2.682820
7	1.689599	-1.240400
8	1.624146	-1.174931
9	1.619262	-1.170045
10	1.618806	-1.169590
11	1.618763	-1.169546
12	1.618759	-1.169542
13	1.618759	-1.169542
14	1.618759	-1.169542
15	1.618759	-1.169542
16	1.618759	-1.169542
17	1.618759	-1.169542
18	1.618759	-1.169542
19	1.618759	-1.169542
20	1.618759	-1.169542
21	1.618759	-1.169542
22	1.235930	-0.786618
23	2.088087	-1.638988
24	3.792401	-3.343728
25	2.334925	-1.885887
26	2.117255	-1.668162
27	2.091410	-1.642311
28	2.088465	-1.639365
29	2.088130	-1.639030

30	2.088092	-1.638992
31	2.088088	-1.638988
32	2.088087	-1.638988
33	2.088087	-1.638988
34	2.088087	-1.638988
35	2.088087	-1.638988

Problem 4

In this problem you will implement ridge regression estimator using gradient descent. Although the ridge regression does have a closed form solution, gradient descent form lets you avoid explicit matrix inversion and scale to larger data. We will see this in our subsequent lectures.

For this problem, we assume we are given the labeled data pair:
$\{(\mathbf{x}^{(i)} \in \mathbb{R}^d, y^{(i)} \in \mathbb{R})\}_{i=1}^N$. The regularized objective function in this case is given by:

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d} L(b, \mathbf{w}; \{(\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N) := \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - (b + \mathbf{w}^T \cdot \mathbf{x}^{(i)}))^2 + \lambda \cdot ||\mathbf{w}||_2^2$$

The pseudocode for performing gradient descent is given by the following algorithm. The main structure consists of a loop which continues for a given number of epochs T. η is the learning rate that controls the amount you want to step into the direction of the negative gradient, and λ is the regularization parameter.

function GDRidge($S_{\mathsf{train}} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$, T, η, λ)
  Initialize the bias term $b \leftarrow 0$ and the slope $\mathbf{w} \leftarrow \mathbf{0}$
  for $t = 1, \cdots, T$ **do**
    $b_{\mathsf{new}} \leftarrow b - \eta \cdot \frac{\partial L}{\partial b}$, $\quad \mathbf{w}_{\mathsf{new}} \leftarrow \mathbf{w} - \eta \cdot \frac{\partial L}{\partial \mathbf{w}}$
    $b \leftarrow b_{\mathsf{new}}$, $\quad \mathbf{w} \leftarrow \mathbf{w}_{\mathsf{new}}$
  end for
end function

(a) Write the expression for the gradient update for b and $\mathbf{w}$.

Solution:

$$\frac{\partial L}{\partial b}(b, \vec{w}; \{(x^{(i)}, y^{(i)})\}_{i=1}^N) = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - (b + \vec{w}^T \cdot \vec{x}^{(i)}))^2 + \lambda \cdot \|\vec{w}\|_2^2$$
$$= -\frac{1}{N}\sum 2\,(y^{(i)} - (b + \vec{w}^T \cdot \vec{x}^{(i)}))$$

$$\frac{\partial L}{\partial \vec{w}}(b, \vec{w}; \{(x^{(i)}, y^{(i)})\}_{i=1}^N) = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - (b + \vec{w}^T \cdot \vec{x}^{(i)}))^2 + \lambda \cdot \|\vec{w}\|_2^2$$
$$= \frac{1}{N}\sum 2\,(y^{(i)} - (b + \vec{w}^T \cdot \vec{x}^{(i)}))\,\vec{x}^{(i)} + 2\lambda\vec{w}$$

(b) Implement the function GDRidge described above. Please include your source code in the writeup.

Solution:

30	2.088092	-1.638992
31	2.088088	-1.638988
32	2.088087	-1.638988
33	2.088087	-1.638988
34	2.088087	-1.638988
35	2.088087	-1.638988

Problem 4

(b)

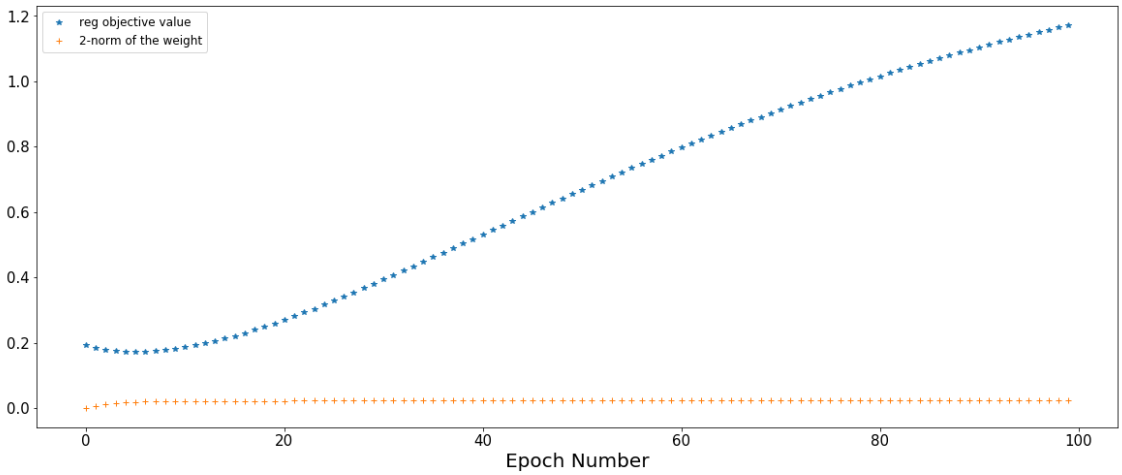
In [8]:
```
def GDRidge(x, y, T, eta, lam):
    n = y.shape[0]
    n_feature = x.shape[1]
    b = 0
    w = np.zeros(n_feature)
    w_array = []
    ridge_array = []
    for i in range(T):
        w_array.append(np.sqrt(np.dot(w, w)))
        ridge_array.append(L(x, y, n, w, b, lam))
        b_grad = 0
        w_grad = 0
        for j in range(n):
            b_grad += 2*(y[j]-b-np.dot(w, x[j]))
            w_grad += 2*(y[j]-b-np.dot(w, x[j]))*x[j]
        b_new = b + (eta/n)*b_grad
        w_new = w + eta*(w_grad/n - 2*lam*w)
        b = b_new
        w = w_new
    return b, w, ridge_array, w_array

def L(x, y, n, w, b, lam):
    n = y.shape[0]
    acc = 0
    for k in range(n):
        acc += (y[k]-b-np.dot(w,x[k]))**2
    Lr = acc/n + lam*np.sum(w**2)
    return Lr
```

(c) Use the Boston housing data (`https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html`) and scale the data appropriately: standardize the feature matrix and [0, 1] scale the y values. Choose $T = 100$ and $\eta = 0.01$.

For $\lambda = 0.1$, provide a x-y plot with the epoch number as x-axis and plot the following quantities on the y-axis:

- The value of regularized objective L at the start of each epoch.
- The 2-norm of the weight vector $\mathbf{w}$ at the start of each epoch.

Here is an example plot:



This will require you to modify the function written in Part (b) to compute the required quantities.

Solution:

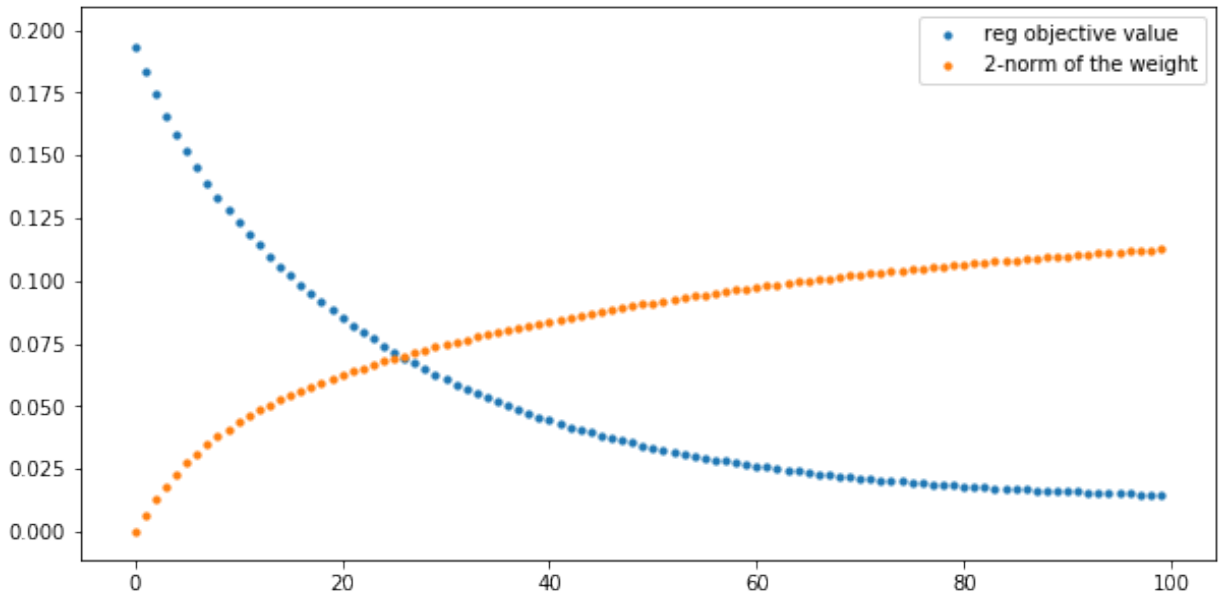
(c)

In [9]:
```
from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
boston = load_boston()
x = StandardScaler().fit_transform(boston.data)
y = MinMaxScaler().fit_transform(boston.target.reshape(-1,1))
df = pd.DataFrame(x, columns = boston.feature_names)
y = y.reshape(-1)
df['target'] = y
```

In [13]:
```
T = 100
eta = 0.01
lam = 0.
b, w, ridge_array, w_array = GDRidge(x, y, T, eta, lam)
```

In [14]:
```
plt.figure(figsize=(10,5))
plt.scatter(range(100), ridge_array, s = 10, label = "reg objective va
lue")
plt.scatter(range(100), w_array, s = 10, label = "2-norm of the weight
")
plt.legend(loc='best')
```

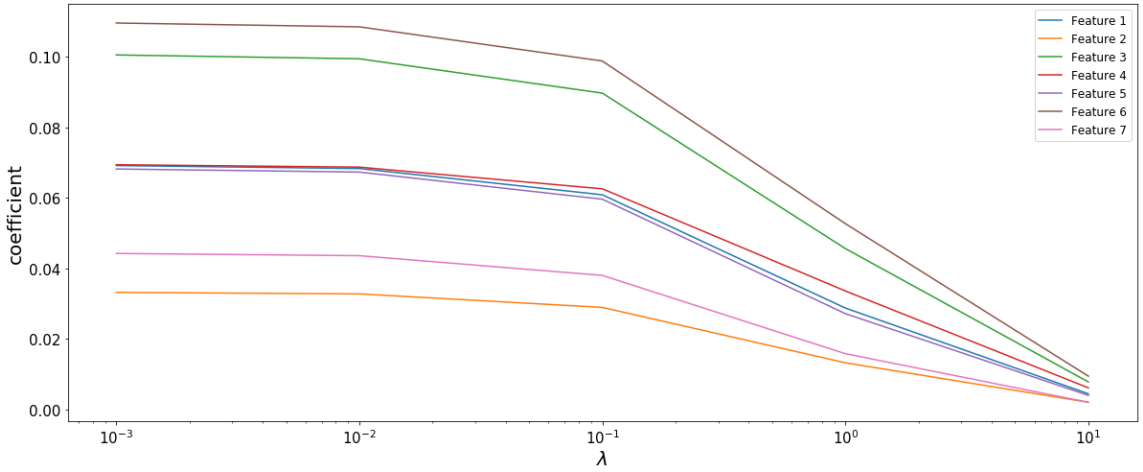
Out[14]:
```
<matplotlib.legend.Legend at 0x1a19c50c10>
```



(d) The next plot will examine how the coefficients for each of the features change as the regularization parameter is varied.

Provide a coefficient path plot for the Boston housing data for $\lambda = 10, 1, 0.1, 0.01, 0.001$. x-axis is for the regularization value and y-axis for the coefficient of the final converged iterate of your gradient descent algorithm. Make sure to scale the data appropriately. Choose $T = 100$ and $\eta = 0.01$.

Here is an example plot for a dataset with 7 features: there is a connected path for each of the 7 features as the regularization parameter is varied.



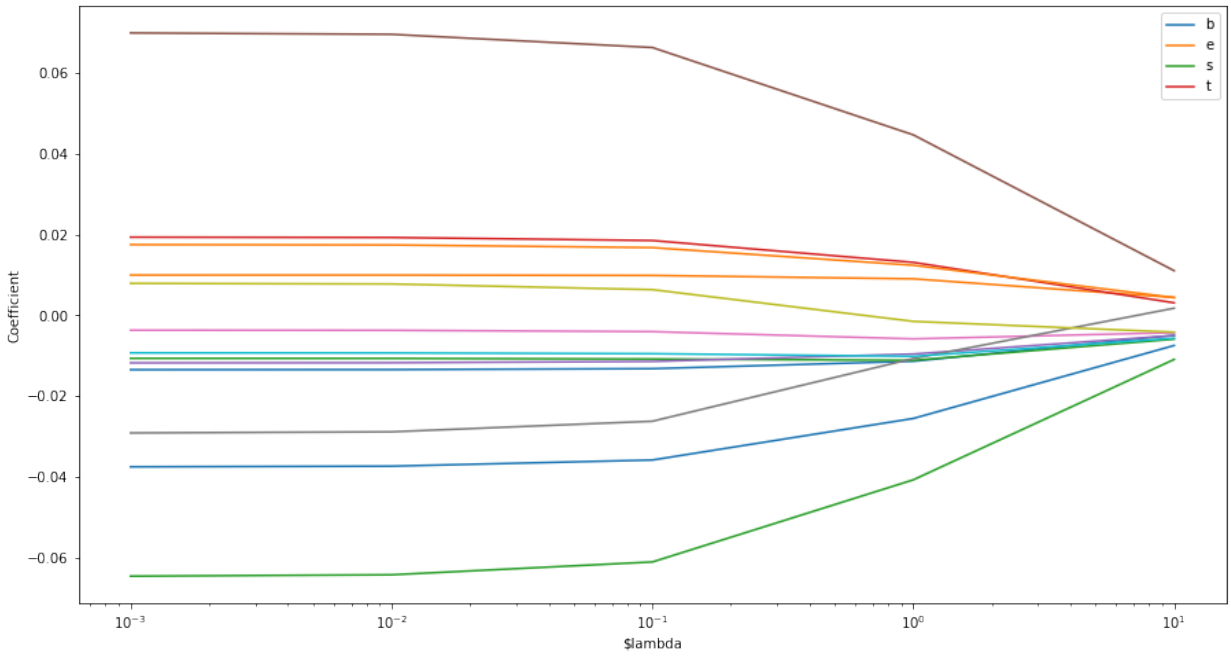
What do you notice about the behavior of the coefficients as the regularization is varied? Include the coefficient path plot and your analysis.

Solution:

(d)

In [17]:
```
l = [10, 1, 0.1, 0.01, 0.001]
n_feature = x.shape[1]
w_lst = []
for i in l:
    b, w, ridge_array, w_array = GDRidge(x, y, 100, 0.01, lam=i)
    w_lst.append(w)
w_lst = np.transpose(w_lst)

plt.figure(figsize=(15,8))
for k in range(n_feature):
    plt.plot(l, w_lst[k], label = 'Feature {}'.format(df.columns[k]))

plt.xscale('log')
plt.xlabel('$lambda')
plt.ylabel('Coefficient')
plt.legend('best')
```

Out[17]: <matplotlib.legend.Legend at 0x1a1a80edd0>



Coefficients converfes to 0 as the regularization parameter λ getting larger.