Name :- Prateek Mahajan

EESII : Homework 3

Problem 1 :-

a) I have read and understood the general instructions at the top of HW3 and I formally declare that all work I turn in for everything in this course will not contain or involve any cheating at all.

Problem 2 :-

a) Consider a binary naive Bayes classifier with multivariate class conditional distributions $p(x|y)$

let class variance be equal.

Bayes Assumption :- $p(x|y) = \prod_{j=1}^{m} p(x_j|y)$

∴ since they are gaussian :-

$$p(x|y) = \prod_{i=1}^{m} \frac{1}{(2\pi\sigma_{y,i}^2)^{1/2}} \cdot \exp\left(\frac{-1}{2}(x_i - \mu_{y,i})^2/\sigma_{y,i}^2\right)$$

for "m" univariate gaussians

As mentioned in the slides :-

~~Log odds ratio~~ for the binary case :-

Posterior probability, $p(y=y_1|x) = \frac{1}{1+\exp\left(-\log\left(\frac{p(x|y=y_1)\cdot p(y=y_1)}{p(x|y=y_0)\cdot p(y=y_0)}\right)\right)}$

$$\Rightarrow p(y=y_1|x) = g\left(\sum_{i=1}^{m} \log\left(\frac{p(x_i|y=1)}{p(x_i|y=0)}\right) + \log\left(\frac{p(y=1)}{p(y=0)}\right)\right),$$

where the f^n inside g is the log ratio

Consider the term $\frac{p(x_i|y=y_1)}{p(x_i|y=y_0)}$

① $p(x_i|y=y_1) = \frac{1}{(2\pi\sigma^2_{y_1,i})^{1/2}} \cdot \exp\left(\frac{-1}{2}\frac{(x_i-\mu_{y_1,i})^2}{\sigma^2_{y_1,i}}\right)$

② $p(x_i|y=y_0) = \frac{1}{(2\pi\sigma^2_{y_0,i})^{1/2}} \cdot \exp\left(\frac{-1}{2}\frac{(x_i-\mu_{y_0,i})^2}{\sigma^2_{y_0,i}}\right)$

But class variances are equal, $\sigma^2_{y_0,i} = \sigma^2_{y_1,i} = \sigma^2_i \ \forall i$

$$\Rightarrow \frac{p(x_i|y=y_1)}{p(x_i|y=y_0)} = \exp\left(\frac{-1}{2\sigma_i^2}\left((x_i-\mu_{y_1,i})^2 - (x_i-\mu_{y_0,i})^2\right)\right)$$

$$= \exp\left(\frac{-1}{2\sigma_i^2}(2x_i - \mu_{y_1,i} - \mu_{y_0,i})(\mu_{y_0,i} - \mu_{y_1,i})\right)$$

$$= \exp\left(\frac{-1}{\sigma_i^2}\left(x_i - \frac{(\mu_{y_1,i}+\mu_{y_0,i})}{2}\right)(\mu_{y_0,i} - \mu_{y_1,i})\right)$$

$$\therefore \log\left(\frac{p(x_i|y=y_1)}{p(x_i|y=y_0)}\right) = \frac{(\mu_{y_1,i}-\mu_{y_0,i})}{\sigma_i^2}x_i + \frac{\mu^2_{y_0,i} - \mu^2_{y_1,i}}{2\sigma_i^2}$$

Now consider $\log\left(\frac{p(y=1)}{p(y=0)}\right)$

Clearly, the above term is a constant & independent of X

∴ the log ratio

$$\sum_{i=1}^{m} \log\left(\frac{p(x_i|y=1)}{p(x_i|y=0)}\right) + \log\left(\frac{p(y=1)}{p(y=0)}\right)$$

$$= \sum_{i=1}^{m} \frac{(\mu_{y_1,i}-\mu_{y_0,i})}{\sigma_i^2}x_i + \frac{\mu^2_{y_0,i} - \mu^2_{y_1,i}}{2\sigma_i^2} + \log\left(\frac{p(y=1)}{p(y=0)}\right)$$

$$= b_i x_i + c_i \quad \text{where :—}$$

$$b_i = \frac{(\mu_{y_1,i} - \mu_{y_0,i})}{\sigma_i^2}$$

$$c_i = \frac{\mu^2_{y_0,i} - \mu^2_{y_1,i}}{2\sigma_i^2} + \log\left(\frac{p(y=1)}{p(y=0)}\right)$$

b) If the class variances are unequal, consider eq" ① & ② from 2a).
From ① & ② :—

$$\frac{p(x_i|y=y_1)}{p(x_i|y=y_0)} = \frac{\sigma_{y_0,i}}{\sigma_{y_1,i}} \cdot \exp\left(\frac{-1}{2}\left[\frac{1 \cdot 1}{\sigma_{y_1,i}^2 \cdot \sigma_{y_0,i}^2}\right]\right)$$

$$\exp\left[\sigma_{y_0,i}^2 (x_i^2 + \mu_{y_1,i}^2 - 2\mu_{y_1,i} x_i) + \sigma_{y_1,i}^2 (x_i^2 + \mu_{y_0,i}^2 - 2\mu_{y_0,i} x_i)\right]$$

$$\Rightarrow \frac{p(x_i|y=y_1)}{p(x_i|y=y_0)} = \frac{\sigma_{y_0,i}}{\sigma_{y_1,i}} \cdot \exp\left[\frac{-1 \cdot 1 \cdot 1}{2\sigma_{y_1,i}^2 \sigma_{y_0,i}^2} \cdot \left[x_i^2(\sigma_{y_0,i}^2 - \sigma_{y_1,i}^2) + \right.\right.$$

$$x_i\left[2\mu_{y_0,i}\sigma_{y_1,i}^2 - 2\mu_{y_1,i}\sigma_{y_0,i}^2\right] + \sigma_{y_0,i}^2 \cdot \mu_{y_1,i}^2$$

$$\left.\left. - \sigma_{y_1,i}^2 \cdot \mu_{y_0,i}^2\right]\right]$$

Taking log on both sides :—

$$\log\left(\frac{p(x_i|y=y_1)}{p(x_i|y=y_0)}\right) = \log\left(\frac{\sigma_{y_0,i}}{\sigma_{y_1,i}}\right) + \frac{1}{2\sigma_{y_1,i}^2 \sigma_{y_0,i}^2}\left[x_i^2(\sigma_{y_0,i}^2 - \sigma_{y_1,i}^2) + \right.$$

$$2x_i(\mu_{y_1,i}\sigma_{y_0,i}^2 - \mu_{y_0,i}\sigma_{y_1,i}^2)$$

$$\left. + \sigma_{y_1,i}^2 \mu_{y_0,i}^2 - \sigma_{y_0,i}^2 \mu_{y_1,i}^2\right]$$

Now consider $\log\left(\frac{p(y=1)}{p(y=0)}\right)$

Clearly, as in 2a, this is constant & independent of X.

∴ the log ratio =

$$\sum_{i=1}^{n} \log\left(\frac{p(x_i|y=1)}{p(x_i|y=0)}\right) + \log\left(\frac{p(y=1)}{p(y=0)}\right)$$

$$= x_i^2\left(\frac{\sigma_{y_1,i}^2 - \sigma_{y_0,i}^2}{2\sigma_{y_1,i}^2\sigma_{y_0,i}^2}\right) + \left(\frac{\mu_{y_1,i}\sigma_{y_0,i}^2 - \mu_{y_0,i}\sigma_{y_1,i}^2}{\sigma_{y_1,i}^2\sigma_{y_0,i}^2}\right)x_i + \left(\frac{\sigma_{y_0,i}^2\mu_{y_1,i}^2}{2\sigma_{y_1,i}^2\cdot\sigma_{y_0,i}^2} - \frac{\sigma_{y_0,i}^2\mu_{y_1,i}^2}{2\sigma_{y_1,i}^2\cdot\sigma_{y_0,i}^2} + \log\left(\frac{\sigma_{y_0,i}}{\sigma_{y_1,i}}\right) + \log\left(\frac{p(y=1)}{p(y=0)}\right)\right)$$

$$= a_i x_i^2 + b_i x_i + c_i,$$

where :—

$$a_i = \frac{\sigma_{y_1,i}^2 - \sigma_{y_0,i}^2}{2\sigma_{y_1,i}^2\cdot\sigma_{y_0,i}^2}$$

$$b_i = \frac{\mu_{y_1,i}\sigma_{y_0,i}^2 - \mu_{y_0,i}\sigma_{y_1,i}^2}{\sigma_{y_1,i}^2\cdot\sigma_{y_0,i}^2}$$

$$c_i = \frac{\sigma_{y_1,i}^2\mu_{y_0,i}^2 - \sigma_{y_0,i}^2\mu_{y_1,i}^2}{2\sigma_{y_1,i}^2\cdot\sigma_{y_0,i}^2} + \log\left(\frac{\sigma_{y_0,i}}{\sigma_{y_1,i}}\right) + \log\left(\frac{p(y=1)}{p(y=0)}\right)$$

c) As mentioned in 2a :—

$$p(y=y_1|x) = \frac{1}{1+\exp\left(-\log\left(\frac{p(x|y=y_1)\cdot p(y=y_1)}{p(x|y=y_0)\cdot p(y=y_0)}\right)\right)}$$

$$= g\left(\sum_{i=1}^{m}\log\left(\frac{p(x_i|y=y_1)}{p(x_i|y=y_0)}\right) + \log\left(\frac{p(y=1)}{p(y=0)}\right)\right),$$

where $g = \frac{1}{1+e^{-z}}$ → logistic fn

& z → log ratio of gaussian class conditional distributions

∴ this classifier is essentially a logistic regression in the log ratio of the naive Bayes classifier.

For equal class variance (and assuming g is a logistic f^n): —

$p(y=1|x) = g(\theta^T x + \text{const})$ → linear logistic regression

For unequal class variance: —

$p(y=1|x) = g\left(\sum_{i=1}^{n} a_i x_i^2 + b_i x_i + c_i\right)$

↳ quadratic logistic regression, which can be made linear by consider the x_i^2 terms as new features

Problem 3: —

a) For $n=2$: —

$V_2(r) = \pi r^2$

$S_1(r) = 2\pi r$

For $n=3$: —

$V_3(r) = \frac{4}{3}\pi r^3$

$S_2(r) = 4\pi r^2$

b) In my opinion, the core intuition to remember here is that surface area represents the "boundary" or "skin" of a given volume. "dr" represents a small change in the radius of a volume.

∴ if you change the ~~outer surface of a volume~~ radius of a volume by "dr" the proportional change in volume, "dV" will be the surface area of the volume (especially since $dr \to 0$, which means surface area can be considered to be constant)

∴ $\frac{\text{change in volume (infinitely small)}}{\text{change in radius (infinitely small)}} = \text{Surface Area}$

$\Rightarrow \frac{d}{dr} V_m(r) = \alpha \cdot S_{m-1}(r)$ (eq as $dr \to 0$)

Now take $m = 2$:-

$V_2(r) = \pi r^2$

$\Rightarrow \frac{d}{dr} V_2(r) = 2\pi r = S_1(r)$

For $m = 3$:-

$V_3 = \frac{4}{3} \pi r^3$

$\Rightarrow \frac{d}{dr} V_3(r) = 4\pi r^2 = S_2(r)$

Clearly, this $\forall^n$ holds for $m = 2 \& 3$!!

c) $V_m(r) = K r^m$ (as mentioned in qsn.), where K is a constant

$\therefore S_{m-1}(r) = m \cdot K \cdot r^{m-1} = dV(r)/dr$

If $r = 1$:-

$S_{m-1}(r) = m \cdot K = \bar{S}_{m-1}$

$\Rightarrow K = \frac{\bar{S}_{m-1}}{m}$

$\therefore V_m(r) = \frac{\bar{S}_{m-1}}{m} \cdot r^m$

$\therefore S_{m-1}(r) = \bar{S}_{m-1} \cdot r^{m-1}$

d) $f_m(r) = \int_{x \in S_{m-1}(r)} p(x) \cdot dx$

$p(x) = \frac{1}{(2\pi\sigma^2)^{m/2}} \cdot \exp\left(\frac{-\|x\|_2^2}{2\sigma^2}\right)$

$\|x\|_2 = r$ & x in $S_{m-1}(r)$

$\therefore f_m(r)$ = probability density of sampled points lying on surface of $S_{m-1}(r)$

$$= \int_{x \in S_{m-1}(r)} p(x) \cdot dx$$

But $p(x) = \frac{1}{(2\pi\sigma^2)^{m/2}} \cdot \exp\left(\frac{-\|x\|_2^2}{2\sigma^2}\right)$

& since $\|x\|_2 = r$:-

$$p(x) = \frac{1}{(2\pi\sigma^2)^{m/2}} \cdot \exp\left(\frac{-r^2}{2\sigma^2}\right)$$

note that this is because the gaussian is isotropic

$$\therefore f_m(r) = \frac{1}{(2\pi\sigma^2)^{m/2}} \cdot \exp\left(\frac{-r^2}{2\sigma^2}\right) \cdot \int_{x \in S_{m-1}(r)} dx$$

this is essentially $S_{m-1}(r)$

$$= \frac{1}{(2\pi\sigma^2)^{m/2}} \cdot \exp\left(\frac{-r^2}{2\sigma^2}\right) \cdot S_{m-1} \cdot r^{m-1}$$

$$= \frac{1}{(2\pi\sigma^2)^{m/2}} \cdot \exp\left(\frac{-r^2}{2\sigma^2}\right) \cdot \overline{S_{m-1}} \cdot r^{m-1}$$

e) using the above $f_m(r)$,

set $\frac{d}{dr} f_m(r) = 0$

$$\frac{d}{dr} f_m(r) = \frac{d}{dr} \frac{\overline{S_{m-1}} \cdot r^{m-1}}{(2\pi\sigma^2)^{m/2}} \cdot \exp\left(\frac{-r^2}{2\sigma^2}\right)$$

$$= \frac{\overline{S_{m-1}}}{(2\pi\sigma^2)^{m/2}} \cdot \left[(m-1) r^{m-2} \cdot \exp\left(\frac{-r^2}{2\sigma^2}\right) - \frac{2r^m}{2\sigma^2} \exp\left(\frac{-r^2}{2\sigma^2}\right)\right]$$

$$= \frac{\overline{S_{m-1}}}{(2\pi\sigma^2)^{m/2}} \cdot r^{m-2} \cdot \exp\left(\frac{-r^2}{2\sigma^2}\right) \cdot \left[(m-1) - \frac{r^2}{\sigma^2}\right]$$

Setting $\frac{d}{dr} f_m(r) = 0$

$\Rightarrow (m-1) - \frac{r^2}{\sigma^2} = 0 \Rightarrow r^2 = (m-1)\sigma^2$

$\Rightarrow r = \pm\sqrt{(m-1)\sigma^2}$

But $m >>> 1 \Rightarrow m-1 \simeq m$ (i.e. m is large)

$\Rightarrow r \simeq \pm\sqrt{m}\,\sigma$

Consider $\frac{d^2}{dr^2} f_m(r)$

$= \frac{S_{m-1}}{(2\pi\sigma^2)^{m/2}} \cdot \exp\left(\frac{-r^2}{2\sigma^2}\right)\left[(m-2)r^{m-3}\cdot\left[(m-1) - \frac{r^2}{\sigma^2}\right] - 3r^{m-1}\frac{}{\sigma^2}\right]$

$= \frac{S_{m-1}}{(2\pi\sigma^2)^{m/2}} \cdot \exp\left(\frac{-r^2}{\sigma^2}\right) r^{m-3}\left[(m-2)(m-1) - \frac{(m+1)r^2}{\sigma^2}\right]$

For $r = \sqrt{m}\,\sigma$, clearly $\frac{d^2}{dr^2} f_m(r) < 0$

Cos $m(m+1)\sigma^2 = m(m+1)$, ~~(as (m-2)) (as (m+1)) mσ²~~ / σ²

which is greater than $(m-2)(m-1)$]

$\therefore r = \sqrt{m}\,\sigma$ is a maximum value.

Clearly $\frac{d}{dr} f_m(r) = 0$ only at $r = \sqrt{m}\,\sigma$

$\Rightarrow f_m(r)$ has a single maximum value at $\hat{r} \simeq \sqrt{m}\sigma$

f) As proven above, $\hat{r} = \sqrt{m}.\sigma$

~~f(x)~~

$f(r) = \frac{S_{m-1}}{(2\pi\sigma^2)^{m/2}} r^{m-1} \cdot \exp\left(\frac{-r^2}{2\sigma^2}\right)$

$\therefore f(\hat{r}+\epsilon) = \frac{S_{m-1}}{(2\pi\sigma^2)^{m/2}} \cdot (\hat{r}+\epsilon)^{m-1} \cdot \exp\left(-\frac{(\hat{r}+\epsilon)^2}{2\sigma^2}\right)$

Now consider $(\hat{r}+\epsilon)^{m-1}$

Using multinomial expansion :—

$(\hat{r}+\epsilon)^{m-1} = {}^{m-1}C_0 \cdot (\hat{r})^{m-1} \cdot \epsilon^0 + {}^{m-1}C_1 \cdot (\hat{r})^{m-2} \cdot \epsilon + \ldots$

$+ {}^{m-1}C_{m-2}\, r^1 \cdot \epsilon^{m-2} + {}^{m-1}C_{m-1} \cdot (\hat{r})^0 \cdot \epsilon^{m-1}$

Clearly, since $\epsilon << \hat{r}$, all terms are much smaller than ${}^{m-1}C_0 \cdot (\hat{r})^{m-1} = (\hat{r})^{m-1}$

$$\therefore (\hat{r}+\epsilon)^{m-1} \simeq (\hat{r})^{m-1}$$

$$\therefore \rho(\hat{r}+\epsilon) \simeq \frac{S_{m-1}}{(2\pi\sigma^2)^{m/2}} \cdot (\hat{r})^{m-1} \cdot \exp\left(\frac{-(\hat{r})^2 - \epsilon^2 - 2\hat{r}\epsilon}{2\sigma^2}\right)$$

$$\simeq \underbrace{\frac{S_{m-1}}{(2\pi\sigma^2)^{m/2}} \cdot (\hat{r})^{m-1} \cdot \exp\left(\frac{-(\hat{r})^2}{2\sigma^2}\right)}_{\text{this is } \rho(\hat{r})} \cdot \exp\left(\frac{-\epsilon^2 - 2\hat{r}\epsilon}{2\sigma^2}\right)$$

$$\simeq \rho(\hat{r}) \cdot \exp\left(\frac{-\epsilon^2}{2\sigma^2}\right) \cdot \exp\left(-\frac{\hat{r}\epsilon}{\sigma^2}\right)$$

$$\simeq \rho(\hat{r}) \cdot \exp\left(\frac{-\epsilon^2}{2\sigma^2}\right) \cdot \exp\left(-\frac{\sqrt{m}\cdot\epsilon}{\sigma}\right)$$

Using Taylor series expansion:-

$$\exp\left(-\frac{\sqrt{m}\epsilon}{\sigma}\right) = 1 - \frac{\sqrt{m}\epsilon}{\sigma} + \frac{m\epsilon^2}{2!\sigma^2} + \frac{m^{3/2}\epsilon^3}{3!\sigma^3} + \dots$$

But $\epsilon << \hat{r} \Rightarrow \epsilon << \sqrt{m}$ (as σ is constant)

$$\Rightarrow \sqrt{m}\cdot\epsilon \simeq 0.$$

$$\therefore \exp\left(-\frac{\sqrt{m}\epsilon}{\sigma}\right) \simeq 1.$$

$$\Rightarrow \rho(\hat{r}+\epsilon) \simeq \rho(\hat{r}) \cdot \exp\left(\frac{-\epsilon^2}{2\sigma^2}\right) \cdot 1$$

g) If we take a finite ~~sample~~ number of samples from a high dimensional gaussian distribution most of the points would reside at a distance / radius, $r = \sqrt{m}\cdot\sigma$

As shown in 3f), moving away from this point exponentially drops the mass density ρ

probability of sampling. This holds true whether we move from $r = \sqrt{m}\sigma$ towards both the origin & towards infinity.

For a low dimensional gaussian distribution, most points reside near $r_2 = (\sqrt{m-1}).\sigma$

=> For $m = 1$ they reside at the origin
For other low values of m as well, most points would be at small multiples of σ from the origin, and the drop would not be as dramatic as we move from the maxima

h) Probability density $f_m(r) = \frac{S_{m-1}}{(2\pi\sigma^2)^{m/2}} \cdot r^{m-1} \cdot \exp\left(\frac{-r^2}{2\sigma^2}\right)$

$\therefore f_m(0) = 0$

$f_m(r \pm S_{m-1}(\hat{r})) = f_m(\hat{r}) = \frac{S_{m-1}}{(2\pi\sigma^2)^{m/2}} \cdot (\sqrt{m}\sigma)^{m-1} \cdot \exp\left(\frac{-m\sigma^2}{2\sigma^2}\right)$

$= \frac{S_{m-1}}{(2\pi)^{m/2}} \cdot \frac{m^{(m-1/2)}}{\sigma} \cdot e^{-m/2}$

Clearly, probability density = 0 at the origin & is very very high at $r = \hat{r}$ (which is the curse of dimensionality)

i) Please refer to the end of the pdf for the graph & code.

Clearly, the mean of the radius increases as n increases, which is in line with our earlier observations, which show that the mass of the gaussian goes to a higher radius as the dimensionality increases (as depicted by this increase in mean).
Further, it was observed that even when the mass increased, the probability density at the maximum "r" dropped exponentially as you moved away from it in higher dimensions ⇒ the standard deviation should don't increase & be ≃ constant, as is observed in this graph.

Problem 5:-

a) Based on the scatter plot of the question the the decision boundary seems to be a non-linear & non-exponential one. Therefore I believe a low-dimensional linear SVM (note a higher dimensional may work) would not be a great choice which is also the case for logistic regression, as the decision boundary is not ~~ignorable~~ linear in nature.
∴ I would choose the gaussian kernel SVM.

(since it is probably better at representing a non-linear decision boundary, like the provided one)

b) Please refer to the end of this pdf.

c) Please refer to the end of this pdf for the accuracy answers.

The gaussian kernel SVM is better at representing non-linear decision boundaries than the others which are for linear boundaries. More so than anything else, this is a property of their mathematical formulation (which is "linear" in nature)

d) Logistic Regression: Linear Decision Boundary, as it is a linear mathematical formulation (i.e. it relies on a linear combination of input features)

Linear SVM: Linear Decision Boundary as it is a linear mathematical formulation (as it relies on a linear combination of input features)

Gaussian Kernel SVM:- Non linear decision boundary, as is a non-linear mathematical formulation

e) Please refer to the end of this pdf for visualisations. The experiment confirmed my hypothesis.

Problem 4:-

a) Please refer to the end of the pdf for the code & plots.
I took the soln for $\lambda = 25.0$

b) Please refer to the end of this pdf for this qn.

c) Please refer to the end of this pdf for this qn..

In the higher σ case, the initial weights & the change in weights is less consistent & more random.
As a consequence, the zero features take longer to converge to sparsity.
This occurs due to an increase in randomness in data and drop in consistency.

d) For n=50, m=75, λ = 25.0
For n=50, m=150, λ = 50.0
For n=50, m=1000, λ = 70.0
For n=100, m=75, λ = 25.0
For n=100, m=150, λ = 50.0.

I think n = O(m) is most the probable relationship here.

e) Please refer to the end of this pdf file for this qn.

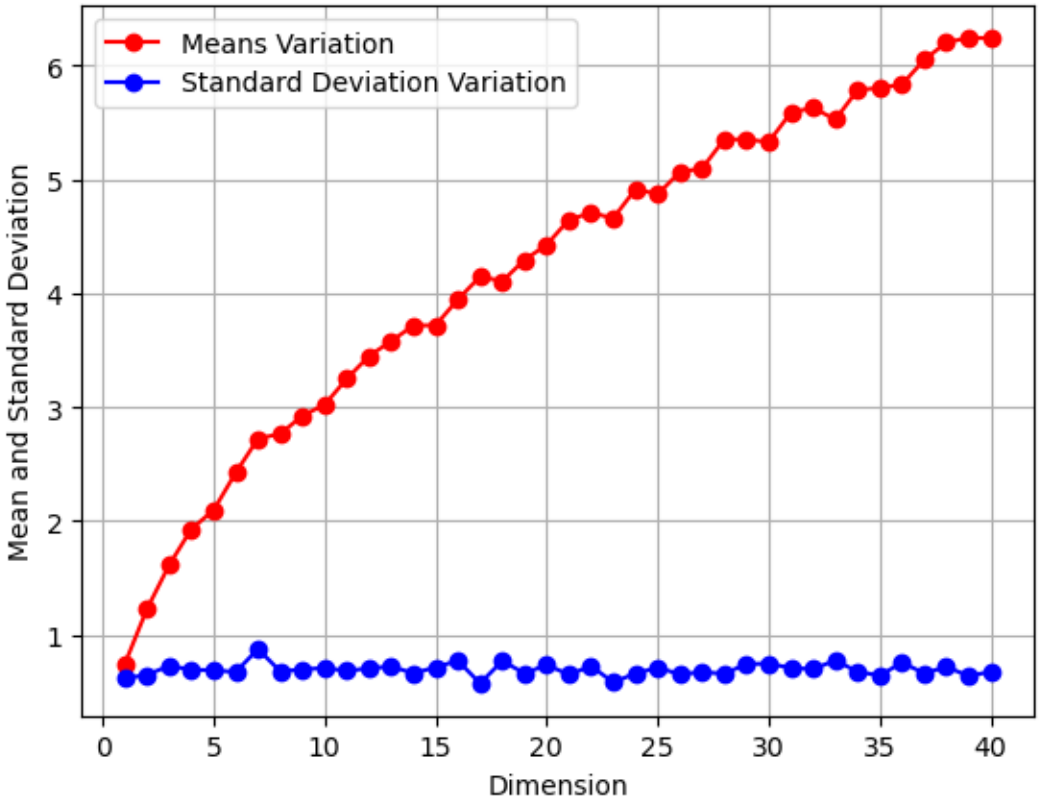
4ynougipo

February 23, 2024

[1]:
```
import sys
import numpy as np
import time

import matplotlib.pyplot as plt
```

[5]:
```
dimensions =␣
 ↪[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
radii = np.zeros(40)
radiiMeans = np.zeros(40)
radiiStd = np.zeros(40)
for m in dimensions:
    # Sample from an m-dimensional Gaussian distribution
    points = np.random.normal(0, 1, (100, m))

    # Compute the radii of the points
    points_radii_sq = np.sum(points**2, axis=1)
    #print(points_radii_sq.shape)
    points_radii = np.sqrt(points_radii_sq)

    # Compute the mean and standard deviation of the radii
    mean_radius = np.mean(points_radii)
    std_radius = np.std(points_radii)

    radiiMeans[m-1] = mean_radius
    radiiStd[m-1] = std_radius

cl = ['red', 'blue']
plt.plot(dimensions, radiiMeans, cl[0], ls = '-', marker = 'o', label = 'Means␣
 ↪Variation')
plt.plot(dimensions, radiiStd, cl[1], ls = '-', marker = 'o', label = 'Standard␣
 ↪Deviation Variation')
plt.grid()
plt.legend()
plt.xlabel('Dimension')
plt.ylabel('Mean and Standard Deviation')
```

[5]: Text(0, 0.5, 'Mean and Standard Deviation')



[ ]:

j887zeo78

February 23, 2024

1 Programming Problem: Model Comparison among Logistic Regression, Linear SVM and Gaussian Kernel SVM

In this problem, you are expected to implement and compare three models: (1) logistic regression, (2) a linear SVM, (3) a Gaussian kernel SVM. In this assignment, you are allowed to use the sklearn packages.

Data: we provide binary classification dataset camel_train.cvs and camel_test.cvs for training and testing respectively (these files are both in hw3_data_files.zip). The model is trained on training set, you are supposed to report accuracy both on the training set and on the test set. Each sample consists of input features $x^{(i)} \in \mathbb{R}^2$ and class lables $y^{(i)} \in \{-1, 1\}$. Note that here the labels are $\{-1, +1\}$ valued rather than $\{0, 1\}$ valued for convenience.

Part (a) [5 points] Visualize the training data in 2D space in Sec 1.1 and estimate or guess based on visualization which classifier you think will be most suitable for the dataset, and explain why. Write down this estimation **{before} you start your implementation and testing, and this is what you'll also turn in. Note that you will not be penalized for getting the wrong answer here, the main thing we would like to see is your prediction and your clear justification why.**

Part (b) [10 points] Implement a function in Sec 1.2 that can learn the Logistic Regression, Linear SVM, Gaussian kernel SVM models by setting the appropriate model type.

Part (c) [5 points] Report the accuracy of each model and explain why one method is superior to the others (that is, fill in Sec 1.3).

Part (d) [5 points] Discuss how the decision boundary of each model looks like in 2D space and why. Please be very clear and precise in your discussion.

Part (d) [10 points] Fill in the code to visualize the decision boundary of the learned model in Sec 1.4. Do they follow your assumptions and guesses above? If so, explain why the experiment confirmed your hypothesis. If the models did not behave consistently with your expectations explain why, and most importantly, explain how your hypotheses were adjusted and critically what you learned from this experiment.

[1]:
```
# Import Modules
import sklearn
import pandas as pd
import numpy as np
```

```
from sklearn import datasets
import matplotlib.pyplot as plt
```

[2]:
```
# Load dataset
# Read in the csv
df_train=pd.read_csv('camel_train.csv', encoding='utf-8')
df_test = pd.read_csv('camel_test.csv', encoding='utf-8')
# Difference between white rating and black rating - independent variable
df_train.head()
```

[2]:

	0.566992416097	0.84059391133	-1.0
0	0.297289	0.791696	-1.0
1	1.030462	0.429955	1.0
2	0.311769	0.867175	-1.0
3	0.883618	0.397090	1.0
4	0.362195	0.880017	1.0

[3]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99 entries, 0 to 98
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   0.566992416097  99 non-null     float64
 1   0.84059391133   99 non-null     float64
 2   -1.0            99 non-null     float64
dtypes: float64(3)
memory usage: 2.4 KB
```

[4]:
```
df_train.describe()
```

[4]:

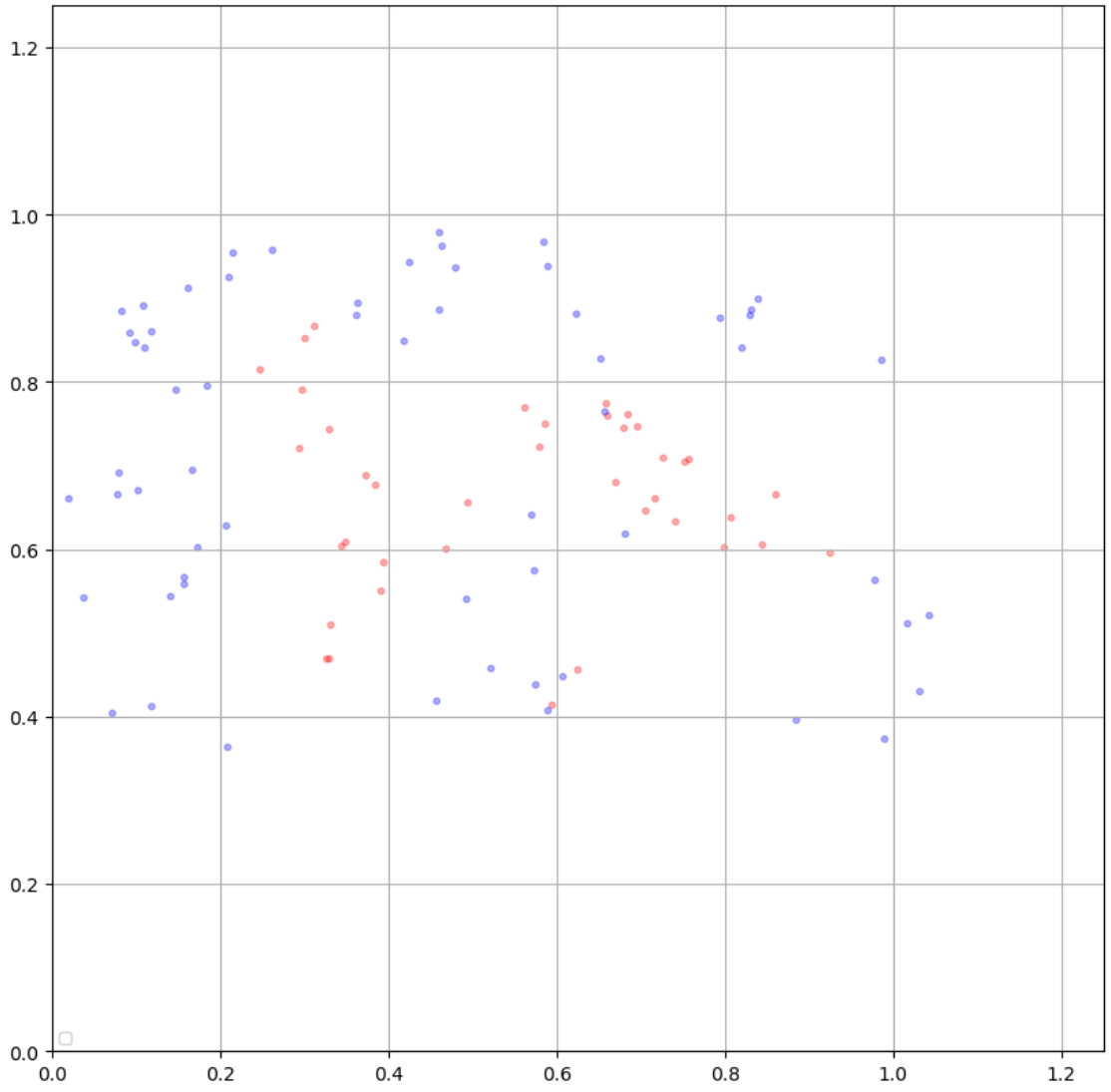
	0.566992416097	0.84059391133	-1.0
count	99.000000	99.000000	99.000000
mean	0.482068	0.694767	0.212121
std	0.277511	0.168909	0.982217
min	0.019830	0.363592	-1.000000
25%	0.230922	0.570406	-1.000000
50%	0.467246	0.691494	1.000000
75%	0.682227	0.848954	1.000000
max	1.041663	0.979701	1.000000

[5]:
```
# Select rating difference and turns as feature to predict the label
# training set
x_train = df_train.iloc[:, [0,1]].to_numpy()
y_train = df_train.iloc[:, 2].to_numpy()
```

```
# testing set
x_test = df_test.iloc[:, [0,1]].to_numpy()
y_test = df_test.iloc[:, 2].to_numpy()
```

[16]:
```
#@title Part (a)
# visualize training set
# TODO
fig, ax = plt.subplots(figsize=(10, 10))
num_samples, num_features = x_train.shape
for i in range(num_samples):
    if y_train[i] == -1 :
        ax.scatter(x_train[i][0], x_train[i][1], c='r', s=10.0, alpha=0.3)
    else :
        ax.scatter(x_train[i][0], x_train[i][1], c='b', s=10.0, alpha=0.3)
ax.grid()
ax.legend(('r', 'b'),
           ('y = -1', 'y = +1'),
           scatterpoints=1,
           loc='lower left',
           ncol=3,
           fontsize=8)
# Set x/y axis limits
ax.set_xlim([0, 1.25])
ax.set_ylim([0, 1.25])
fig.show()
```

```
/var/folders/c9/3q_3q36n4k9_t4gnwrhyd7s80000gn/T/ipykernel_16449/3689731530.py:1
2: UserWarning: Legend does not support handles for str instances.
A proxy artist may be used instead.
See:
https://matplotlib.org/stable/users/explain/axes/legend_guide.html#controlling-
the-legend-entries
  ax.legend(('r', 'b'),
/var/folders/c9/3q_3q36n4k9_t4gnwrhyd7s80000gn/T/ipykernel_16449/3689731530.py:2
1: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
  fig.show()
```



[27]:
```
#@title Part (b) implement
# Train logistic regression and print accuracy
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn import svm
from sklearn.metrics import accuracy_score, classification_report

# define training procedure for logistic regression and svm
def training(x_train, y_train, x_test, y_test, model_type = 'logistic',␣
 ↪kernel=None):
  # specify the model used to learn by model_type
```

```
  # model_type == 'logistic' -> using logistic regression model
    if model_type == 'logistic':
        model = LogisticRegression(max_iter=200)
        model.fit(x_train, y_train)
        y_pred_test = model.predict(x_test)
        y_pred_train = model.predict(x_train)
        acc_test = accuracy_score(y_test, y_pred_test)
        acc_train = accuracy_score(y_train, y_pred_train)
  # model_type == 'svm' --> using SVM model
    elif model_type == 'svm':
  # kernel == None  --> using linear kenel for SVM
        if kernel == None:
            model = LinearSVC(max_iter=10000)
            model.fit(x_train, y_train)
            y_pred_test = model.predict(x_test)
            y_pred_train = model.predict(x_train)
            acc_test = accuracy_score(y_test, y_pred_test)
            acc_train = accuracy_score(y_train, y_pred_train)
  # kernel == 'rbf' --> using gaussian kernel for SVM
        elif kernel == 'rbf' :
            model = SVC(kernel='rbf', gamma='scale')
            model.fit(x_train, y_train)
            y_pred_test = model.predict(x_test)
            y_pred_train = model.predict(x_train)
            acc_test = accuracy_score(y_test, y_pred_test)
            acc_train = accuracy_score(y_train, y_pred_train)
  # define models
    return model, acc_train, acc_test
```

[28]:
```
#@title Part (c)
# train logistic regression
# TODO
(model_log, acc_train_log, acc_test_log) = training(x_train, y_train, x_test,␣
 ↪y_test, model_type = 'logistic', kernel=None)
print("Logistic Training Accuracy : " + str(acc_train_log))
print("Logistic Testing Accuracy : " + str(acc_test_log))
# train linear SVM
# TODO
(model_lsvm, acc_train_lsvm, acc_test_lsvm) = training(x_train, y_train,␣
 ↪x_test, y_test, model_type = 'svm', kernel=None)
print("Linear SVM Training Accuracy : " + str(acc_train_lsvm))
print("Linear SVM Testing Accuracy : " + str(acc_test_lsvm))

# train Gaussian SVM
# TODO
(model_gsvm, acc_train_gsvm, acc_test_gsvm) = training(x_train, y_train,␣
 ↪x_test, y_test, model_type = 'svm', kernel='rbf')
```

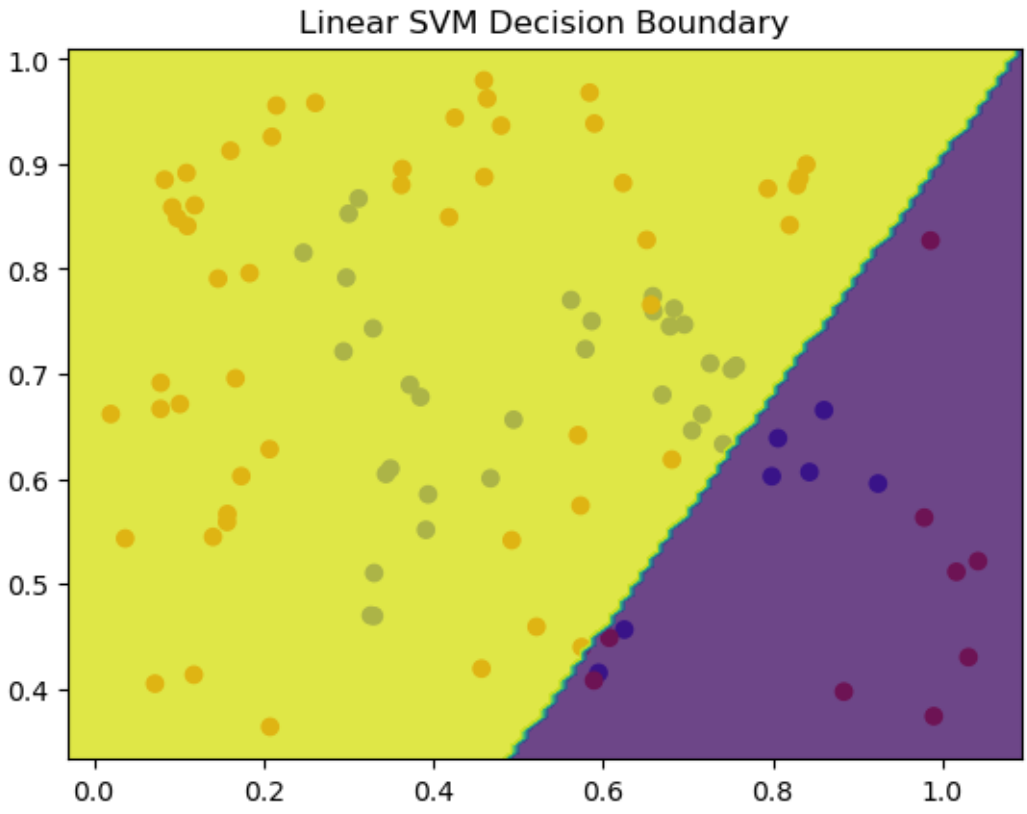
```
print("Gaussian SVM Training Accuracy : " + str(acc_train_gsvm))
print("Gaussian SVM Testing Accuracy : " + str(acc_test_gsvm))
# Report accuracies of the above three
```

```
Logistic Training Accuracy : 0.5555555555555556
Logistic Testing Accuracy : 0.531328320802005
Linear SVM Training Accuracy : 0.5858585858585859
Linear SVM Testing Accuracy : 0.581453634085213
Gaussian SVM Training Accuracy : 0.8585858585858586
Gaussian SVM Testing Accuracy : 0.7192982456140351
```

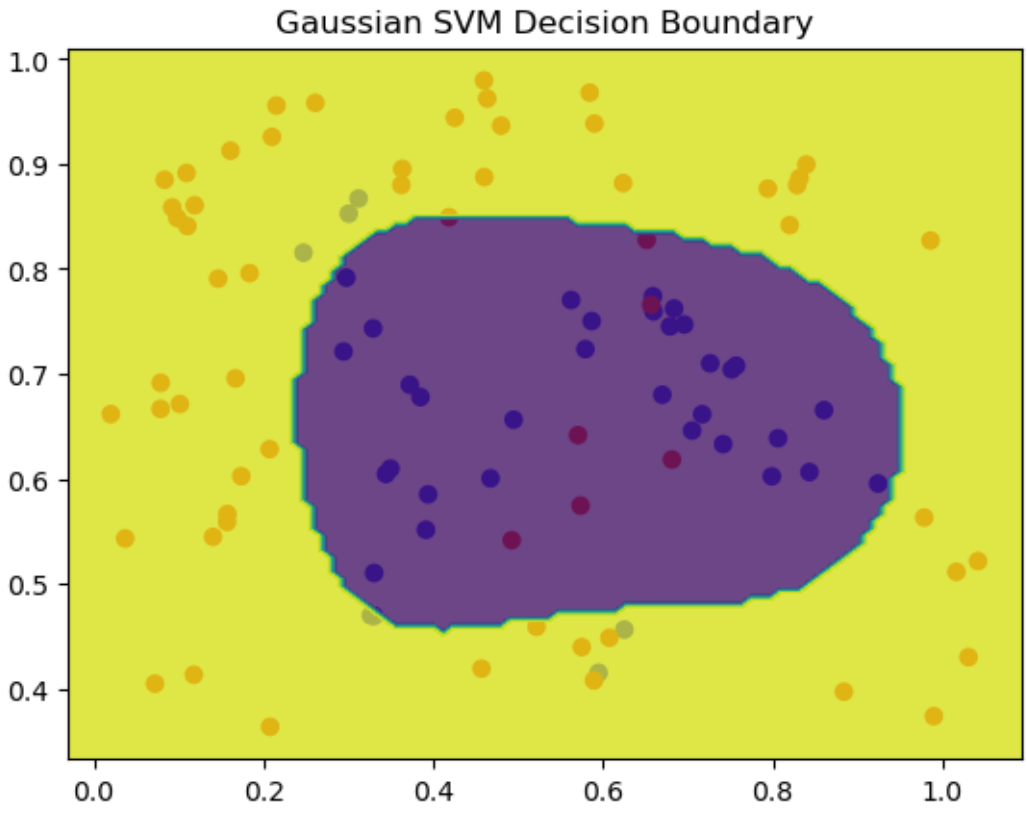
[39]:
```
# Part (d) plot decision boundary for learned models, add discussion in your␣
 ↪writeup.
class_colors = {-1: 'b', 1: 'r'}
def plot_decision_boundary(model, x, y, title='', output_path=None,
        file_name=None, class_colors={-1: 'b', 1: 'r'}, model_type= 'svm' ):
    colors = y > 0 if class_colors is None else [class_colors[c] for c in y]
    fig = plt.figure()
    plt.scatter(x[:, 0], x[:, 1], c=colors)

    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    xx = np.linspace(xlim[0], xlim[1], 100)
    yy = np.linspace(ylim[0], ylim[1], 100)
    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T
    Z = model.predict(xy)
    Z = Z.reshape(XX.shape)
    # TODO
    plt.contourf(XX, YY, Z, alpha=0.8)
    plt.title(title)
    plt.show()
    if output_path is not None and file_name is not None:
        fig.savefig(os.path.join(output_path, file_name))
```

[40]:
```
# Part (e)
# plot decision boundary for logistic regression
# TODO
plot_decision_boundary(model_log, x_train, y_train, title='Logistic Regression␣
 ↪Decision Boundary', output_path=None,
        file_name=None, class_colors={-1: 'b', 1: 'r'}, model_type= 'logistic' )
```

[41]:
```
# plot decision boundary for linear svm
# TODO
plot_decision_boundary(model_lsvm, x_train, y_train, title='Linear SVM Decision␣
 ↪Boundary', output_path=None,
       file_name=None, class_colors={-1: 'b', 1: 'r'}, model_type= 'svm' )
```



[42]:
```
# plot decision boundary for linear svm
# TODO
plot_decision_boundary(model_gsvm, x_train, y_train, title='Gaussian SVM␣
 ↪Decision Boundary', output_path=None,
       file_name=None, class_colors={-1: 'b', 1: 'r'}, model_type= 'svm' )
```


Gaussian SVM Decision Boundary
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
0.2
0.4
0.6
0.8
1.0

[ ]:

y1ugbd5qh

February 23, 2024

[1]:
```
# all the packages you need
from __future__ import division
import sys
import numpy as np
import time
import scipy.io as io
import scipy.sparse as sparse
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# synthetic data generator
# n is number of samples, d is number of dimensions, k is number of nonzeros in␣
 ↪w, sigma is std of noise,
# X is a n x d data matrix, y=Xw+w_0+noise is a n-dimensional vector, w is the␣
 ↪true weight vector, w0 is true intercept
def DataGenerator(n = 50, d = 75, k = 5, sigma = 1.0, w0 = 0.0, seed = 256):

    np.random.seed(seed)
    X = np.random.normal(0,1,(n,d))
    w = np.random.binomial(1,0.5,k)
    noise = np.random.normal(0,sigma,n)
    w[w == 1] = 10.0
    w[w == 0] = -10.0
    w = np.append(w, np.zeros(d - k))
    y = X.dot(w) + w0 + noise
    return (X, y, w, w0)
```

[3]:
```
# initialization of W for lasso by least square regression or ridge regression
def Initialw(X, y):

    n, d = X.shape
    # increment X
    if sparse.issparse(X):
        XI = sparse.hstack((X, np.ones(n).reshape(n,1)))
    else:
        XI = np.hstack((X, np.ones(n).reshape(n,1)))
```

```
    if sparse.issparse(X):
        if n >= d:
            w = sparse.linalg.lsqr(XI, y)[0]
        else:
            w = sparse.linalg.inv(XI.T.dot(XI) + 1e-3 * sparse.eye(d+1)).dot(XI.
 ↪T.dot(y))
            w = w.T
    else:
        if n >= d:
            w = np.linalg.lstsq(XI, y)[0]
        else:
            w = np.linalg.inv(XI.T.dot(XI) + 1e-3 * np.eye(d+1)).dot(XI.T.
 ↪dot(y))

    return (w[:d], w[d])
```

[4]:
```
# Helper and example function of sparse matrix operation for Problem 2.5
# W: a scipy.sparse.csc_matrix
# x: a vector with length equal to the number of columns of W
# In place change the data stored in W,
# so that every row of W gets element-wise multiplied by x
def cscMatInplaceEleMultEveryRow(W, x):
    W_out = W.copy()
    indptr = W_out.indptr
    last_idx = indptr[0]
    for col_id, idx in enumerate(indptr[1:]):
        if idx == last_idx:
            continue
        else:
            W_out.data[last_idx:idx] *= x[col_id]
            last_idx = idx
    return W_out
```

[5]:
```
# Problem 4(a).
# TODO: coordinate descent of lasso, note lmda stands for lambda
def calculate_cost(X, w, w0, y, lmda):
    cost = np.dot(X,w)
    cost = cost + w0*np.ones(cost.shape) - y
    cost = cost**2
    total_cost = np.sum(cost)
    total_cost = total_cost/2
    total_cost = total_cost + lmda*np.sum(np.abs(w))
    #print("Total Cost : ", total_cost)
    return total_cost

def lasso(X, y, lmda = 10.0, epsilon = 1.0e-2, max_iter = 100, draw_curve =␣
 ↪False):
```

```
    num_samples, num_features = X.shape
    #print(X.shape)
    w, w0 = Initialw(X,y)
    Cost = []
    prev_w = 100000*np.ones(w.shape)
    step = 0
    while((np.max(np.abs(w - prev_w)) > epsilon) and step <= 100):
        for i in range(w.size):
            prev_w[i] = w[i]
        for k in range(num_features):
            Xtheta_wok = np.zeros(num_samples)
            temp_w = np.zeros(w.shape)
            temp_w = w
            w[k] = 0
            Xtheta_wok = np.matmul(X,temp_w)
            R_k = y - Xtheta_wok
            C_total = np.sum(np.dot(R_k,X[:,k]))
            A_total = np.sum(np.dot(X[:,k],X[:,k]))
            if C_total < -lmda :
                w[k] = (C_total + lmda)/A_total
            elif C_total > lmda :
                w[k] = (C_total - lmda)/A_total
            else:
                w[k] = 0
        current_cost = calculate_cost(X, w, w0, y, lmda)
        Cost.append(current_cost)
        step += 1
    return (w,w0,Cost)
```

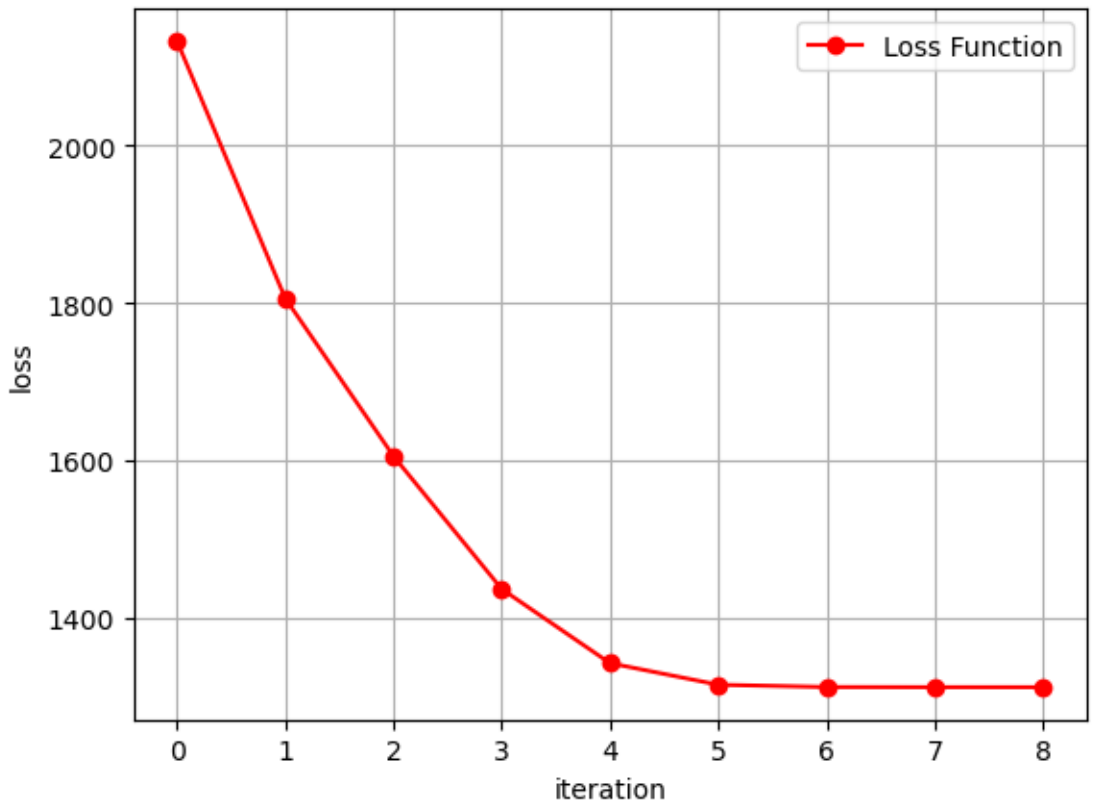
[6]:
```
# Problem 4(a): data generation
X, y, w_true, w0_true = DataGenerator(n=50, d=75, k=5, sigma=1.0)
# have a look at generated data and true model
print(X)
print(y)
print(w_true)
print(w0_true)
```

```
[[ 0.10430293 -0.55011253 -0.07271465 …  0.9858945   0.9762621
   0.66088793]
 [-1.00421694 -0.98028568  1.04231343 …  0.54423528 -0.12555319
   0.29833038]
 [-0.93920808 -0.88460697 -0.36846914 …  1.13839265 -0.17706563
  -1.1040073 ]
 …
 [ 0.22627269 -1.41473902 -1.38744153 …  0.40629811  1.81803336
   0.57718998]
 [-0.87827944 -1.1588945  -0.20821426 …  2.5616317   0.71706683
```

```
  -1.6834583 ]
 [ 1.18136184  0.97753967 -1.08284432 … -0.26515022  1.70874717
   1.25566562]]
[ -2.94661658  -9.2469922   -6.61852337  -8.71813976  -2.77082316
 -21.16384608   2.47720978  -8.18425969  17.12490003  13.69805685
  27.11926075 -35.71631086 -11.85971212  18.6242186  -10.34229026
 -26.02528015 -38.1950294   19.8767635    0.46858206  -3.92985654
   8.35960867  22.22456719 -63.25244103  -7.14048583   8.24525032
  23.62138731 -28.79749873  -3.8576642   18.13970725  43.72678802
 -24.73981649  -8.27834954  40.86565523  32.20353774  -7.46417913
  -1.43551809 -33.9853813   15.26040273   9.93183083   4.22152497
 -12.82174377  -3.78551444   0.33847136  14.91338771  22.9035117
  26.94902572 -18.02183139  44.98241912  24.73597308  -2.21765887]
[ 10. -10. -10.  10.  10.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.]
0.0
```

[7]:
```
# Problem 4(a): run lasso and plot the convergence curve
# TODO: run lasso for one synthetic data
w_lasso, w0_lasso, Cost = lasso(X, y, lmda = 25.0, epsilon = 1.0e-2, draw_curve
 ↪= True, max_iter = 100)
# have a look at the lasso model you got (sparse? where?)
plt.plot(Cost, c = 'red', ls = '-', marker = 'o', label = 'Loss Function')
plt.grid()
plt.legend()
plt.xlabel('iteration')
plt.ylabel('loss')
print(w_lasso)
```

```
[ 9.53051744 -9.4689961  -9.47056747  9.4387061   9.58339615  0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.14519184  0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.        ]
```



[8]:
```python
# Problem 4(b):
def root_mean_square_error(pred, y):
    diff_matrix = y - pred
    rmse = diff_matrix**2
    rmse = rmse.sum()
    rmse = rmse/np.size(y)
    rmse = np.sqrt(rmse)
    return rmse

def Evaluate(X, y, w_lasso, w0_lasso, w_true, w0_true):
    numerator = 0
    precision_dec = 0
    recall_dec = 0
    for i in range(w_true.size):
        if((w_true[i] != 0) and (w_lasso[i] != 0)):
            numerator += 1
        if(w_true[i] != 0):
            recall_dec += 1
        if(w_lasso[i] != 0):
            precision_dec += 1
    precision_w = 0
```

```
    if precision_dec != 0 :
        precision_w = numerator/precision_dec
    recall_w = numerator/recall_dec
    sparsity_w = precision_dec
    pred = np.dot(X,w_lasso)
    rmse = root_mean_square_error(pred, y)
    return (rmse, sparsity_w, precision_w, recall_w)
```

[9]:
```
# Problem 4(b)
# TODO: apply your evaluation function to compute precision (of w), recall (of␣
 ↪w), sparsity (of w) and training RMSE
(rmse, sparsity_w, precision_w, recall_w) = Evaluate(X, y, w_lasso, w0_lasso,␣
 ↪w_true, w0_true)
print("Precision : ", precision_w)
print("Recall :", recall_w)
print("Sparsity :", sparsity_w)
print("Root Mean Square Error :", rmse)
```

```
Precision :  0.8333333333333334
Recall : 1.0
Sparsity : 6
Root Mean Square Error : 1.401209192235464
```

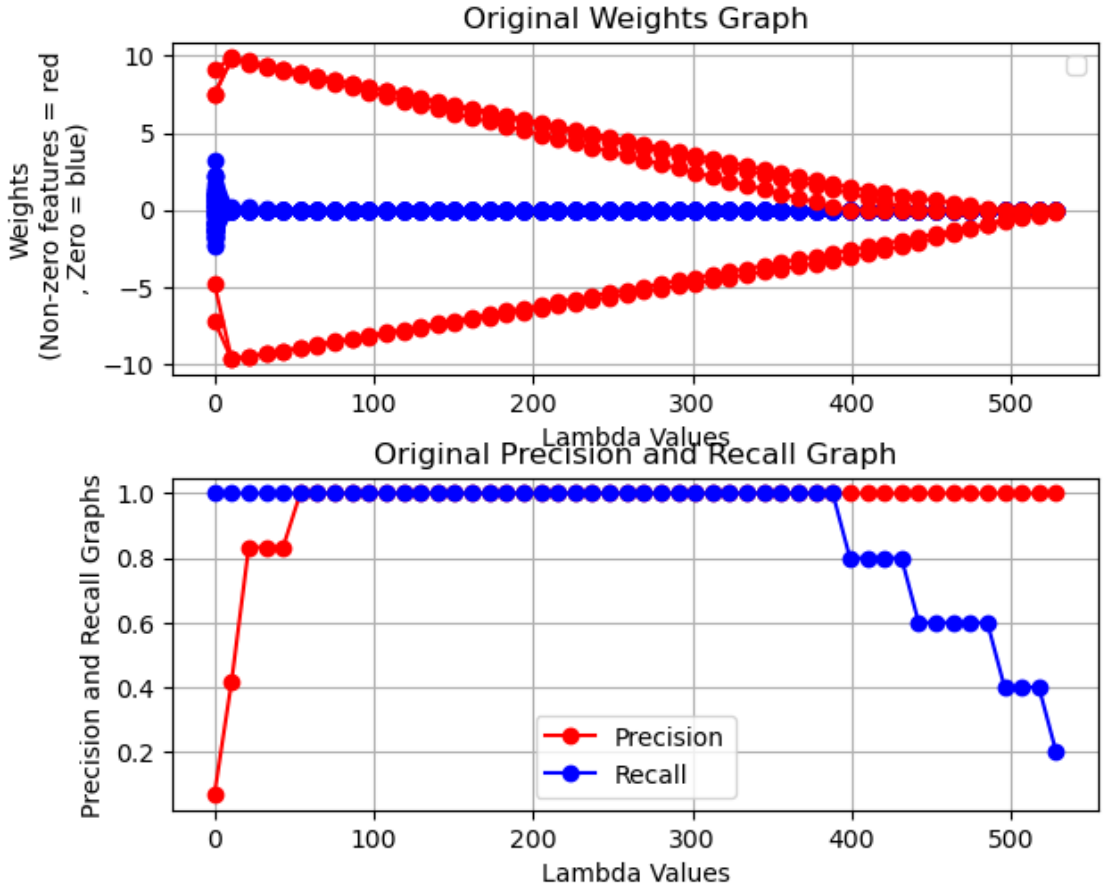
[10]:
```
# Problem 4(c), first part
# TODO: compute a lasso solution path, draw the path(s) in a 2D plot
def LassoPath(X, y):
    ##########TODO###########
    y_av = np.mean(y)
    num_samples, num_features = X.shape
    y_norm = y - y_av*np.ones(y.shape)
    lmdaMax = np.max(np.abs(np.dot(y_norm,X)))
    lmdaMin = 0
    Lmda = np.linspace(lmdaMin, lmdaMax, 50)
    #print(Lmda)
    W = []
    W0 = []
    for l in Lmda:
        w_lasso, w0_lasso, Cost = lasso(X, y, l, epsilon = 1.0e-2, draw_curve =␣
 ↪True, max_iter = 100)
        W.append(w_lasso)
        W0.append(w0_lasso)
    cl = ['red', 'blue']
    #print("Start plotting")
    for i in range(num_features-5):
        temp = np.zeros(50)
        for j in range(50):
            temp[j] = W[j][i+5]
```

```
        plt.plot(Lmda, temp, cl[1], ls = '-', marker = 'o')
    for i in range(5):
        temp = np.zeros(50)
        for j in range(50):
            temp[j] = W[j][i]
        plt.plot(Lmda, temp, cl[0], ls = '-', marker = 'o')
    plt.grid()
    plt.legend()
    plt.xlabel('Lambda Values')
    plt.ylabel('Weights\n (Non-zero features = red\n, Zero = blue)')
    return (W, W0, Lmda)
```

[11]:
```
# Problem 4(c), second part:
# TODO: create function to evaluate a given lasso solution path & draw plot of␣
 ↪precision/recall vs. lambda
def EvaluatePath(X, y, W, W0, w_true, w0_true, Lmda):
    ##########TODO###########
    Precision = []
    Recall = []
    Sparsity = []
    RMSE = []
    for l in range(50) :
        (rmse, sparsity_w, precision_w, recall_w) = Evaluate(X, y, W[l], W0[l],␣
 ↪w_true, w0_true)
        Precision.append(precision_w)
        Recall.append(recall_w)
        Sparsity.append(sparsity_w)
        RMSE.append(rmse)
    cl = ['red', 'blue']
    plt.plot(Lmda, Precision, cl[0], ls = '-', marker = 'o', label =␣
 ↪'Precision')
    plt.plot(Lmda, Recall, cl[1], ls = '-', marker = 'o', label = 'Recall')
    plt.grid()
    plt.legend()
    plt.xlabel('Lambda Values')
    plt.ylabel('Precision and Recall Graphs')
    return (RMSE, Sparsity, Precision, Recall)
```

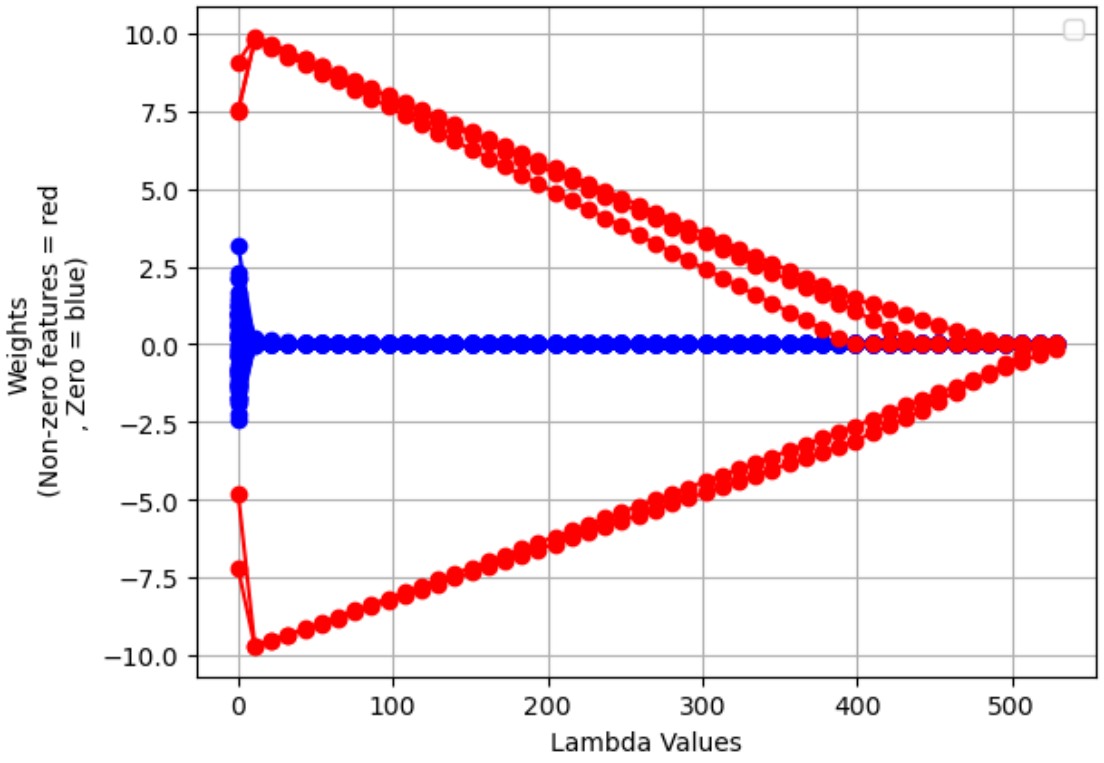
[12]:
```
plt.subplot(2,1,1)
plt.gca().set_title('Original Weights Graph')
W, W0, Lmda = LassoPath(X, y)
plt.tight_layout()
plt.subplot(2,1,2)
plt.gca().set_title('Original Precision and Recall Graph')
RMSE, Sparsity, Precision, Recall = EvaluatePath(X, y, W, W0, w_true, w0_true,␣
 ↪Lmda)
```

No artists with labels found to put in legend. Note that artists whose label start with an underscore are ignored when legend() is called with no argument.



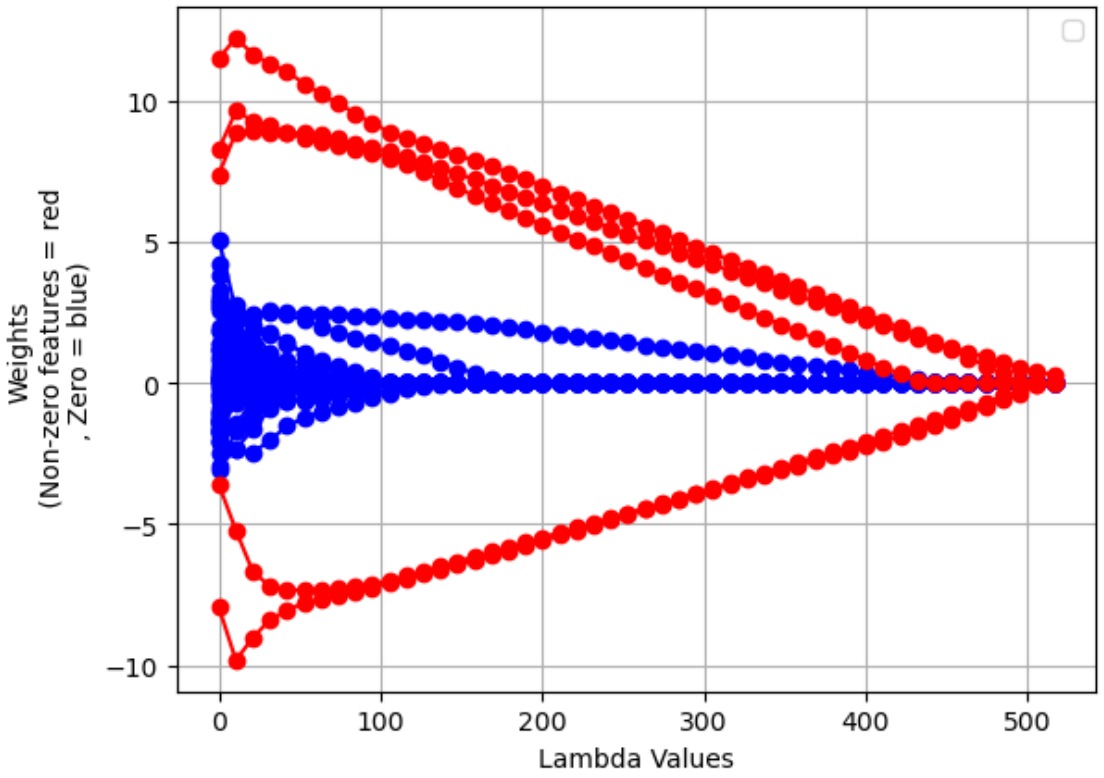
[13]:
```
# Problem 4(c), third part:
# TODO: using the above, draw lasso solution path and precision/recall vs. lambda curves
X, y, w_true, w0_true = DataGenerator(n=50, d=75, k=5, sigma=1.0)
W, W0, Lmda = LassoPath(X, y)
```

No artists with labels found to put in legend. Note that artists whose label start with an underscore are ignored when legend() is called with no argument.



[14]:
```
# Problem 4(c), noise standard deviation:
# TODO: try a larger std sigma = 10.0
X, y, w_true, w0_true = DataGenerator(n=50, d=75, k=5, sigma=10.0)
W, W0, Lmda = LassoPath(X, y)
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```



[15]:
```
# Problem 4(d):
# TODO: try another different choices of (n,m)
# draw lasso solution path and precision/recall vs. lambda curves, use them to␣
 ↪estimate the lasso sample complexity
n = [50, 50, 50, 100, 100, 100]
m = [75, 150, 1000, 75, 150, 1000]
#n = [50]
#m = [75]
plt.subplots(12,1,figsize=(15,50))
for i in range(6):
    print("n = ", n[i], " m = ", m[i])
    X, y, w_true, w0_true = DataGenerator(n=n[i], d=m[i], k=5, sigma=1.0)
    plt.subplot(12,1,2*i+1)
    W, W0, Lmda = LassoPath(X, y)
    plt.gca().set_title('Weights Graph : n = ' + str(n[i]) + ', m = ' +␣
 ↪str(m[i]))
    plt.tight_layout()
    plt.subplot(12,1,2*i+2)
    RMSE, Sparsity, Precision, Recall = EvaluatePath(X, y, W, W0, w_true,␣
 ↪w0_true, Lmda)
    plt.gca().set_title('Precision and Recall Graph : n = ' + str(n[i]) + ', m␣
 ↪= '+ str(m[i]))
```

```
    plt.tight_layout()
```

```
n =  50  m =  75
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```

```
n =  50  m =  150
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```

```
n =  50  m =  1000
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```

```
n =  100  m =  75
```

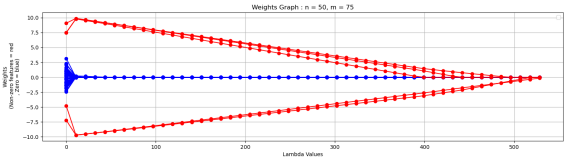
```
/var/folders/c9/3q_3q36n4k9_t4gnwrhyd7s80000gn/T/ipykernel_17115/3894678280.py:1
9: FutureWarning: `rcond` parameter will change to the default of machine
precision times ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  w = np.linalg.lstsq(XI, y)[0]
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```

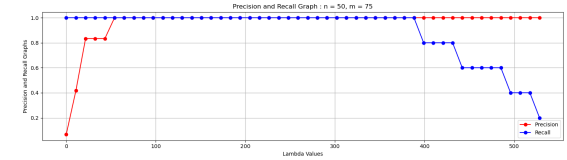
```
n =  100  m =  150
```

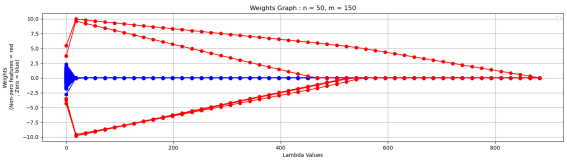
```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```

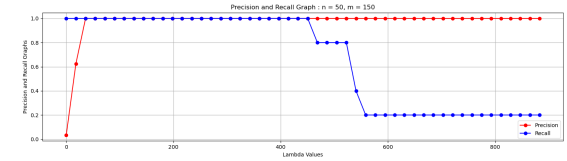
```
n =  100  m =  1000
```

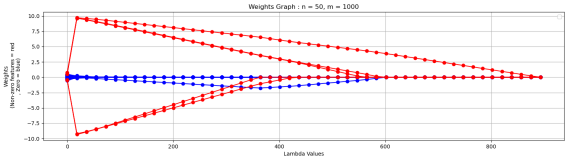
```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```

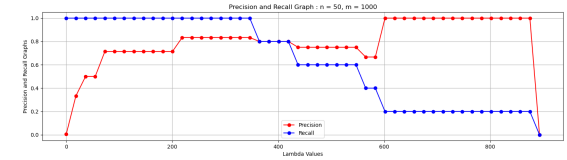

Weights Graph : n = 50, m = 75
Weights
(Non-zero Values = red
Zero = blue)
Lambda Values

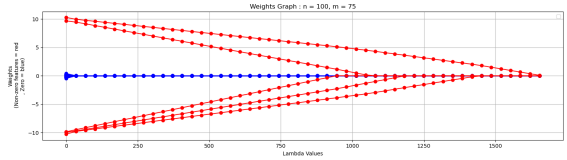

Precision and Recall Graph : n = 50, m = 75
Precision and Recall Graphs
Precison
Recall
Lambda Values

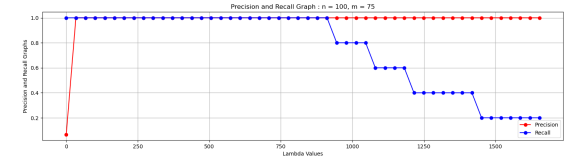

Weights Graph : n = 50, m = 150
Weights
(Non-zero Values = red
Zero = blue)
Lambda Values

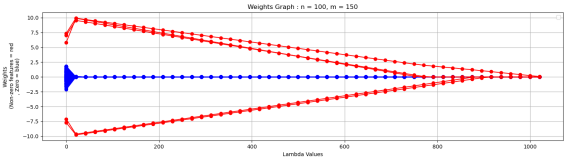

Precision and Recall Graph : n = 50, m = 150
Precision and Recall Graphs
Precison
Recall
Lambda Values

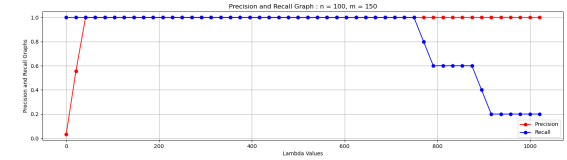

Weights Graph : n = 50, m = 1000
Weights
(Non-zero Values = red
Zero = blue)
Lambda Values

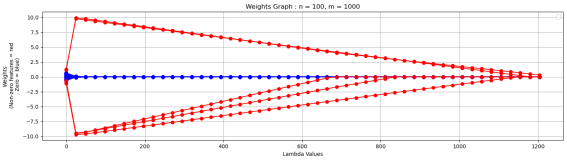

Precision and Recall Graph : n = 50, m = 1000
Precision and Recall Graphs
Precison
Recall
Lambda Values

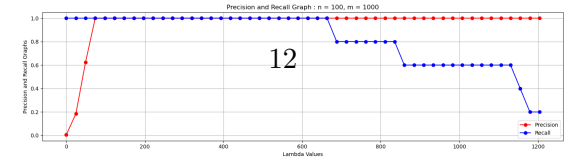

Weights Graph : n = 100, m = 75
Weights
(Non-zero Values = red
Zero = blue)
Lambda Values


Precision and Recall Graph : n = 100, m = 75
Precision and Recall Graphs
Precison
Recall
Lambda Values


Weights Graph : n = 100, m = 150
Weights
(Non-zero Values = red
Zero = blue)
Lambda Values


Precision and Recall Graph : n = 100, m = 150
Precision and Recall Graphs
Precison
Recall
Lambda Values


Weights Graph : n = 100, m = 1000
Weights
(Non-zero Values = red
Zero = blue)
Lambda Values


Precision and Recall Graph : n = 100, m = 1000
Precision and Recall Graphs
Precison
Recall
Lambda Values

[16]:
```
# Problem 4(e): predict reviews' star on Yelp
# data parser reading yelp data
def DataParser(Xfile, yfile, nfile, train_size = 30000, valid_size = 5000):

    # read X, y, feature names from file
    fName = open(nfile).read().splitlines()
    y = np.loadtxt(yfile, dtype=int)
    if Xfile.find('mtx') >= 0:
        # sparse data
        X = io.mmread(Xfile).tocsc()
    else:
        # dense data
        X = np.genfromtxt(Xfile, delimiter=",")

    # split training, validation and test set
    X_train = X[0 : train_size,:]
    y_train = y[0 : train_size]
    X_valid = X[train_size : train_size + valid_size,:]
    y_valid = y[train_size : train_size + valid_size]
    X_test = X[train_size + valid_size : np.size(X,0),:]
    y_test = y[train_size + valid_size : np.size(y,0)]

    return (X_train, y_train, X_valid, y_valid, X_test, y_test, fName)
```

[17]:
```
# Problem 4(e): predict reviews' star on Yelp
# TODO: evaluation funtion that computes the lasso path, evaluates the result,␣
 ↪and draws the required plots

# W: a scipy.sparse.csc_matrix
# x: a vector with length equal to the number of columns of W
# In place change the data stored in W,
# so that every row of W gets element-wise multiplied by x
def cscMulKthColumnWithVector(W, x, k):
    indptr = W.indptr
    sum = 0
    print(x.shape)
    last_idx = indptr[0]
    for col_id, idx in enumerate(indptr[1:]):
        if idx == last_idx:
            continue
        else:
            if col_id == k:
                sum += (W.data[last_idx:idx]*x[col_id])
            last_idx = idx
```

```
    return sum

def lasso_sparse(X, y, lmda = 10.0, epsilon = 1.0e-2, max_iter = 100,␣
 ↪draw_curve = False):
    num_samples, num_features = X.shape
    #print(X.shape)
    w, w0 = Initialw(X,y)
    prev_w = np.zeros(num_features)
    step = 0
    while((np.max(np.abs((w - prev_w))) > epsilon) and step <= max_iter):
        prev_w = w.copy()
        #print(w)
        for k in range(num_features):
            temp_w = np.zeros(w.shape)
            temp_w = w
            temp_w[k] = 0
            #print(temp_w.shape)
            Xtheta_wok_sparse = cscMatInplaceEleMultEveryRow(X,temp_w)
            Xtheta_wok = Xtheta_wok_sparse.sum(axis=1)
            #print(Xtheta_wok.shape)
            ynp = np.expand_dims(y, axis=-1)
            #print(ynp.shape)
            R_k = ynp - Xtheta_wok
            #print(R_k.shape)
            #print("Subtraction done")
            X_k = X.getcol(k)
            #print(X_k.shape)
            C_total = X_k.T.dot(R_k).sum(axis=0)
            #print(C_total)
            A_total = X_k.power(2).sum()
            #print(A_total)
            if C_total < -lmda :
                w[k] = (C_total + lmda)/A_total
            elif C_total > lmda :
                w[k] = (C_total - lmda)/A_total
            else:
                w[k] = 0
        #print(w.shape)
        #print(w)
        step += 1
    #print(w)
    return (w,w0)

def root_mean_square_error(pred, y):
    ynp = np.expand_dims(y, axis=-1)
    diff_matrix = sparse.csc_matrix(ynp) - pred
    #if sparse.issparse(diff_matrix) :
```

```
        #print(diff_matrix.shape)
    rmse = diff_matrix.power(2).sum()
    rmse = rmse/np.size(y)
    rmse = np.sqrt(rmse)
    return rmse

def Validation(X_train, y_train, X_valid, y_valid):
    ##########TODO###########
    y_av = np.mean(y_train)
    y_norm = y_train - y_av
    lmdaMax = np.max(np.abs(X_train.T.dot(y_norm)))
    #print(lmdaMax)
    lmdaMin = 0.1*lmdaMax
    num_samples, num_features = X_train.shape
    noOfLambas = 20
    Lmda = np.linspace(lmdaMin, lmdaMax, noOfLambas)
    w_lasso = []
    TrainingRMSEs = []
    ValidationRMSEs = []
    for l in Lmda :
        print("Starting Lambda : " + str(l))
        (w,w0) = lasso_sparse(X_train, y_train, l, epsilon = 1.0e-2, draw_curve
 ↪= True, max_iter = 50)
        #print("Sparse Lasso Done")
        w_lasso.append(w)
        #print("RMSE started")
        trainingPred = sparse.csc_array(cscMatInplaceEleMultEveryRow(X_train,w).
 ↪sum(axis=1))
        #if sparse.issparse(trainingPred) :
            #print(trainingPred.shape)
        trainingRmse = root_mean_square_error(trainingPred,y_train)
        TrainingRMSEs.append(trainingRmse)
        #print("Training Set stuff done")
        validationPred = sparse.
 ↪csc_array(cscMatInplaceEleMultEveryRow(X_valid,w).sum(axis=1))
        validationRmse = root_mean_square_error(validationPred,y_valid)
        ValidationRMSEs.append(validationRmse)
        #print("RMSE done")
    plt.subplots(3,1,figsize=(10,30))
    cl = ['red']
    plt.subplot(3,1,1)
    #print("Start plotting")
    for i in range(num_features):
        temp = np.zeros(noOfLambas)
        for j in range(noOfLambas):
            temp[j] = w_lasso[j][i]
        plt.plot(Lmda, temp, cl[0], ls = '-', marker = 'o')
```

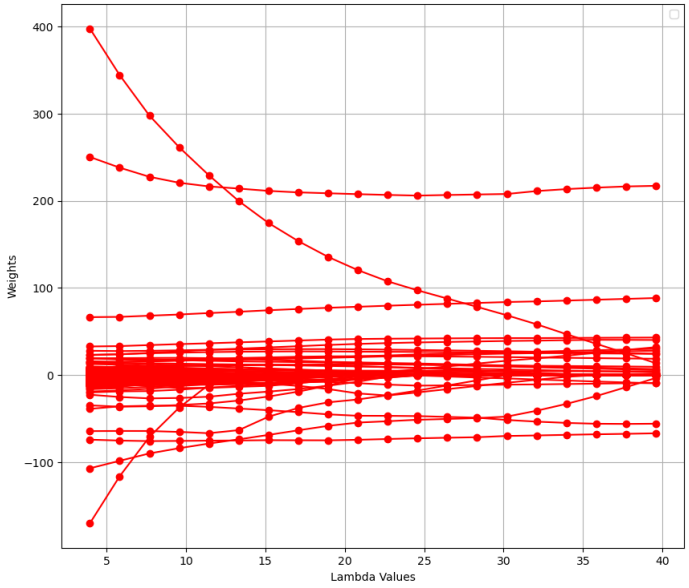
```
    plt.grid()
    plt.legend()
    plt.xlabel('Lambda Values')
    plt.ylabel('Weights')
    plt.subplot(3,1,2)
    #print("Start plotting")
    temp = np.zeros(noOfLambas)
    for j in range(noOfLambas):
        temp[j] = TrainingRMSEs[j]
    plt.plot(Lmda, temp, cl[0], ls = '-', marker = 'o')
    plt.grid()
    plt.legend()
    plt.xlabel('Lambda Values')
    plt.ylabel('Training RMSE')
    plt.subplot(3,1,3)
    #print("Start plotting")
    temp = np.zeros(noOfLambas)
    for j in range(noOfLambas):
        temp[j] = ValidationRMSEs[j]
    plt.plot(Lmda, temp, cl[0], ls = '-', marker = 'o')
    plt.grid()
    plt.legend()
    plt.xlabel('Lambda Values')
    plt.ylabel('Validation RMSE')
    minIndex = ValidationRMSEs.index(min(ValidationRMSEs))
    lmda_best_index = minIndex
    lmda_best = Lmda[minIndex]
    print("The best Lambda as per our evaulation is " + str(lmda_best))
    print("Validation Set RMSE : " + str(ValidationRMSEs[minIndex]))
    print("Training Set RMSE : " + str(TrainingRMSEs[minIndex]))
    return (w_lasso, w0_lasso, lmda_best, lmda_best_index)
```

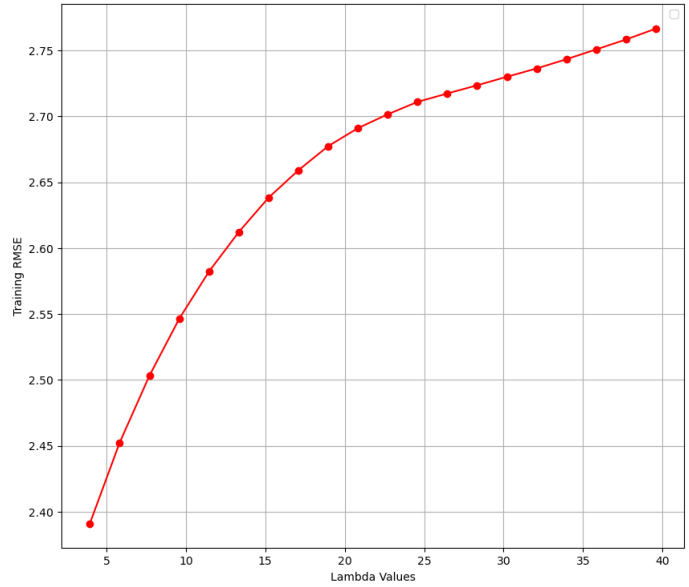
[18]:
```
# Problem 4(e): predict reviews' star on Yelp
# TODO: evaluation of your results

# load Yelp data: change the address of data files on your own machine if␣
 ↪necessary ('../data/' in the below)
from scipy.sparse.linalg import lsqr
X_train, y_train, X_valid, y_valid, X_test, y_test, fName = DataParser('../data/
 ↪star_data.mtx', '../data/star_labels.txt', '../data/star_features.txt',␣
 ↪30000, 5000)

# evaluation
w_lasso, w0_lasso, lmda_best, lmda_best_index = Validation(X_train, y_train,␣
 ↪X_valid, y_valid)
```

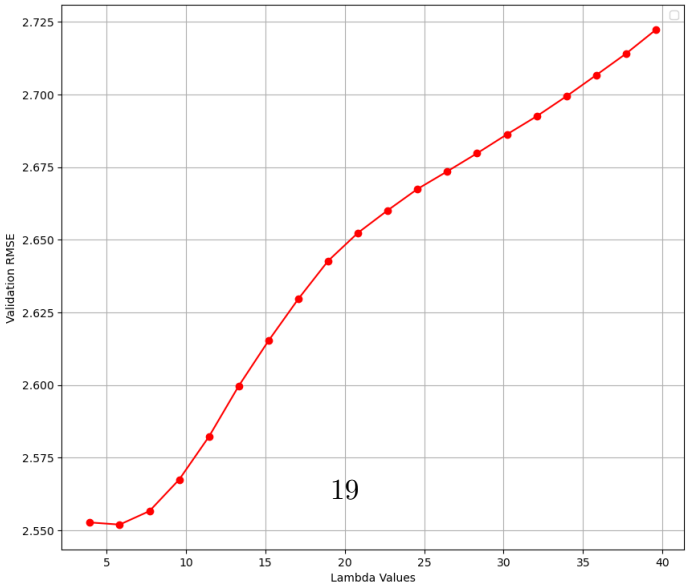
```
testingPred = sparse.
 ↪csc_array(cscMatInplaceEleMultEveryRow(X_test,w_lasso[lmda_best_index]).
 ↪sum(axis=1))
testingRmse = root_mean_square_error(testingPred,y_test)
print("Testing Set RMSE : " + str(testingRmse))

# print the top-10 features you found by lasso
idx = (-np.abs(w_lasso[lmda_best_index])).argsort()[0:10]
print('Lasso select features:')
for i in range(10):
    print(fName[idx[i]],w_lasso[lmda_best_index][idx[i]])
```

```
Starting Lambda : 3.957825401364829
Starting Lambda : 5.832584802011327
Starting Lambda : 7.707344202657826
Starting Lambda : 9.582103603304322
Starting Lambda : 11.456863003950822
Starting Lambda : 13.331622404597319
Starting Lambda : 15.206381805243817
Starting Lambda : 17.081141205890315
Starting Lambda : 18.955900606536815
Starting Lambda : 20.83066000718331
Starting Lambda : 22.705419407829808
Starting Lambda : 24.580178808476308
Starting Lambda : 26.454938209122805
Starting Lambda : 28.329697609769305
Starting Lambda : 30.2044570104158
Starting Lambda : 32.079216411062305
Starting Lambda : 33.9539758117088
Starting Lambda : 35.8287352123553
Starting Lambda : 37.703494613001794
Starting Lambda : 39.57825401364829

No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.

The best Lambda as per our evaulation is 5.832584802011327
Validation Set RMSE : 2.5518993445640437
Training Set RMSE : 2.4524218415861774
Testing Set RMSE : 2.603536820441196
Lasso select features:
the 344.1678544194271
and 238.00036897092048
dog food worst -115.99833178262747
```

```
the people -98.24193726466952
sometime -75.03374864214545
great 66.8468434285838
soaked -63.99805264776684
were -36.18642115055025
sure the -35.76581601794429
best 33.32704433594598
```


Weights
Lambda Values


Training RMSE
Lambda Values


Validation RMSE
Lambda Values

[ ]:

[ ]: