Homework 3

Kyle Hadley

In [2]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
import warnings
warnings.simplefilter('ignore')
```

1. Math Warm-up

Given that our multivariate Gaussian class-conditional distributions $p(x|y)$ is defined by the equation

$$p(x|y) = \frac{1}{|2\pi C_y|^{m/2}} exp\left(-\frac{1}{2}(x-\mu_y)^T C_y^{-1}(x-\mu_y)\right)$$

Note: Worked problem with Joaquin from class on this problem.

(a)

If we have an equal class variance case (i.e. $C_0 = C_1 = C$), then we write our log ratio as follows:

$$\log\frac{p(x_i|y=1)}{p(x_i|y=0)} + \log\frac{p(y=1)}{p(y=0)} = \log\frac{p(x_i|y=1)}{p(x_i|y=0)}$$

as we know that $p(y=0) = p(y=1)$, and $\log(1) = 0$. Now applying our relationship for $p(x|y)$,

$$\log\frac{p(x_i|y=1)}{p(x_i|y=0)} = \log\frac{\left(\frac{1}{|2\pi C_1|^{m/2}} exp\left(\dots\right)\right)}{\left(\frac{1}{|2\pi C_0|^{m/2}} exp\left(\dots\right)\right)}$$

where $\dots$ represents the value inside the numerator. Given that $C_0 = C_1 = C$ and it is just a constant, the term $\frac{1}{|2\pi C_y|^{m/2}}$ will cancel out from the top and bottom leaving us with,

$$= \log\frac{\left(exp\left(-\frac{1}{2}(x_i-\mu_1)^T C_1^{-1}(x_i-\mu_1)\right)\right)}{\left(exp\left(-\frac{1}{2}(x_i-\mu_0)^T C_0^{-1}(x_i-\mu_0)\right)\right)}$$

This can be simplified further to,

$$= \log\left(exp\left(-\frac{1}{2}(x_i-\mu_1)^T C_1^{-1}(x_i-\mu_1)\right)\right) - \log\left(exp\left(-\frac{1}{2}(x_i-\mu_0)^T C_0^{-1}(x_i-\mu_0)\right)\right)$$

$$= \left(-\frac{1}{2}(x_i-\mu_1)^T C_1^{-1}(x_i-\mu_1)\right) - \left(-\frac{1}{2}(x_i-\mu_0)^T C_0^{-1}(x_i-\mu_0)\right)$$

Again, given that $C_0 = C_1 = C$ and combining similar terms we find,

$$= -\frac{1}{2}(x_i-\mu_1)^T C^{-1}(x_i-\mu_1) + \frac{1}{2}(x_i-\mu_0)^T C^{-1}(x_i-\mu_0)$$

$$= -\frac{1}{2}\left(x_i^T C^{-1}x_i - x_i^T C^{-1}\mu_1 - \mu_1^T C^{-1}x_i + \mu_1^T C^{-1}\mu_1\right) + \frac{1}{2}\left(x_i^T C^{-1}x_i - x_i^T C^{-1}\mu_0 - \mu_0^T C^{-1}x_i +\right.$$

$$= -\frac{1}{2}\left(-\mu_1^T C^{-1}x_i - \mu_1^T C^{-1}x_i + \mu_1^T C^{-1}\mu_1\right) + \frac{1}{2}\left(-\mu_0^T C^{-1}x_i - \mu_0^T C^{-1}x_i + \mu_0^T C^{-1}\mu_0\right)$$

$$= (C^{-1}\mu_1 - C^{-1}\mu_0)^T x_i + \mu_0^T C^{-1}\mu_0 - \mu_1^T C^{-1}\mu_1$$

From this form of the log ratio, we can now equate $b_i = (C^{-1}\mu_1 - C^{-1}\mu_0)^T$ and $c_i = \mu_0^T C^{-1}\mu_0 - \mu_1^T C^{-1}\mu_1$ such that the log ratio is $b_i x_i + c_i$ (which can be written as $b_i x + c_i$ when we include the summation).

(b)

If we have an equal class variance case (i.e. $C_0 = C_1 = C$), then we write our log ratio as follows:

$$\log\frac{p(x_i|y=1)}{p(x_i|y=0)} + \log\frac{p(y=1)}{p(y=0)} = \log\frac{p(x_i|y=1)}{p(x_i|y=0)}$$

as we know that $p(y=0) = p(y=1)$, and $\log(1) = 0$. Now applying our relationship for $p(x|y)$,

$$\log\frac{p(x_i|y=1)}{p(x_i|y=0)} = \log\left(\frac{\left(\frac{1}{|2\pi C_1|^{m/2}} exp\left(-\frac{1}{2}(x-\mu_1)^T C_1^{-1}(x-\mu_1)\right)\right)}{\left(\frac{1}{|2\pi C_0|^{m/2}} exp\left(-\frac{1}{2}(x-\mu_0)^T C_0^{-1}(x-\mu_0)\right)\right)}\right)$$

We can isolate the first term from both the top and the bottom and apply our logarithimic properties such that,

$$\log\frac{p(x_i|y=1)}{p(x_i|y=0)} = \log\left(\frac{|2\pi C_0|^{m/2}}{|2\pi C_1|^{m/2}}\right) + \log\left(\frac{\left(exp\left(-\frac{1}{2}(x-\mu_1)^T C_1^{-1}(x-\mu_1)\right)\right)}{\left(exp\left(-\frac{1}{2}(x-\mu_0)^T C_0^{-1}(x-\mu_0)\right)\right)}\right)$$

$$\log\frac{p(x_i|y=1)}{p(x_i|y=0)} = \log\left(\frac{|2\pi C_0|^{m/2}}{|2\pi C_1|^{m/2}}\right) + \log\left(exp\left(-\frac{1}{2}(x-\mu_1)^T C_1^{-1}(x-\mu_1)\right)\right) - \log \left(exp\left(-\frac{1}{2}(x-\mu_0)^T C_0^{-1}(x-\mu_0)\right)\right)$$

$$\log\frac{p(x_i|y=1)}{p(x_i|y=0)} = \log\left(\frac{|2\pi C_0|^{m/2}}{|2\pi C_1|^{m/2}}\right) - \frac{1}{2}(x-\mu_1)^T C_1^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_0)^T C_0^{-1}(x-\mu_0)$$

Given that $m = 2$ is this situation, we can simply further and expand our quadratic relationships such that,

$$\log \frac{p(x_i|y=1)}{p(x_i|y=0)} = 2\log\left(\frac{C_0^2}{C_2^2}\right) - x^T C_1^{-1} x + \mu_1^T C_1^{-1} x + x^T C_1^{-1} \mu_1 -$$

$$\mu_1^T C_0^{-1} \mu_1 + x^T C_0^{-1} x - \mu_0^T C_0^{-1} x - x^T C_0^{-1} \mu_0 + \mu_0^T C_0^{-1} \mu_0$$

Combining like terms,

$$\log \frac{p(x_i|y=1)}{p(x_i|y=0)} = x^T (C_0^{-1} - C_1^{-1}) x + (C_1^{-1}\mu_1 - C_0^{-1}\mu_0)^T x + (\mu_0^T C_0^{-1} \mu_0 - \mu_1^T C_1^{-1} \mu_1) + 2\log\Big(\cdot$$

From this form of the log ratio, we can now equate $a_i = C_0^{-1} - C_1^{-1}$, $b_i = C_1^{-1}\mu_1 - C_0^{-1}\mu_0)^T$ and $c_i = (\mu_0^T C_0^{-1} \mu_0 - \mu_1^T C_1^{-1} \mu_1) + 2\log\left(\frac{C_0^2}{C_2^2}\right)$ such that the log ratio is $a_i x_i^2 + b_i x_i + c_i$.

(c)

A logistic regression and Gaussian class conditional distributions are similar and can be considered identical in the naive Bayes case. Assumptions need to be made to prove this relationship, as outlined in the article linked here: https://appliedmachinelearning.blog/2019/09/30/equivalence-of-gaussian-naive-bayes-and-logistic-regression-an-explanation/.

2. Gaussian Distribution and the Curse of Dimensionality

(a)

When $m = 2$, we know that a "sphere" in 2D space is actually a circle - thus we can define $S_{2-1}(r) = S_1(r)$ and $V_2(r)$ based on the known surface area and area (i.e. volume) equations for a circle,

$$S_1(r) = 2\pi r$$

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

When $m = 3$, a sphere follows the standard sphere equations - thus we can define $S_{3-1}(r) = S_2(r)$ and $V_3(r)$ based on the known surface area and volume equations for a sphere,

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

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

(b)

The equation $S_{m-1} = \frac{d}{dr} V_m(r)$ states that the derivative of the volume of a m-dimensional sphere is its surface area (i.e. $S_{m-1}). We can see that this realtionship holds true from our derived equations from part (a) such that,

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

$$\frac{d}{dr}V_3(r) = \frac{d}{dr}\left(\frac{4}{3}\pi r^3\right) = 4\pi r^2 = S_2(r)$$

(c)

Intuitively, we can see that the relationship for S_{m-1} can be defined as,

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

(d)

Note: Worked problem with Tess, Charlie, Dean, and Steve from class on this problem.

We can solve for $\rho_m(r) = \int_{x\in\delta_{m-1}(r)} p(x)dx$ by using our known equation for $p(x)$ first; thus,

$$\rho_m(r) = \int_{x\in\delta_{m-1}(r)} \frac{1}{(2\pi\sigma^2)^{m/2}} exp\left(-\frac{\|x\|_2^2}{2\sigma^2}\right)dx$$

We can extract our first term as it is not a function of x such that

$$\rho_m(r) = \frac{1}{(2\pi\sigma^2)^{m/2}}\int_{x\in\delta_{m-1}(r)} exp\left(-\frac{\|x\|_2^2}{2\sigma^2}\right)dx$$

We can now substitute the relationship defined by $\delta_{m-1}(r)$ in which $\|x\|_2 = r$; thus,

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

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

From part (c), we know that the integral of all points about the surface of the m-sphere is just the surface area of the m-sphere itself (i.e. $\int_{x\in\delta_{m-1}(r)} dx = \bar{S}_{m-1}r^{m-1}$); thus,

$$\rho_m(r) = \frac{exp\left(\frac{-r^2}{2\sigma^2}\right)}{(2\pi\sigma^2)^{m/2}}\bar{S}_{m-1}r^{m-1}$$

(e)

Note: Worked problem with Tess, Charlie, Dean, and Steve from class on this problem.

We can show that for large m, that $\rho_m(r)$ has a single maximum value $\hat{r}$ such that $\hat{r} \approx \sqrt{m}\sigma$, by taking the derivative of $\rho_m(r)$ with respect to r such that,

$$\frac{d}{dr}\rho_m(r) = \frac{d}{dr}\left(\frac{exp\left(\frac{-r^2}{2\sigma^2}\right)}{(2\pi\sigma^2)^{m/2}}\bar{S}_{m-1}r^{m-1}\right) = \frac{\bar{S}_{m-1}}{(2\pi\sigma^2)^{m/2}}\frac{d}{dr}\left(exp\left(\frac{-r^2}{2\sigma^2}\right)r^{m-1}\right)$$

Once we've taken the derivative, we will be setting it equal to zero to find our maximum value (at $\hat{r}$), so we can eliminate the constant term $\frac{\bar{S}_{m-1}}{(2\pi\sigma^2)^{m/2}}$ as they will be divided out. Thus, our resulting equation when set equal to zero is:

$$\frac{d}{dr}\left(exp\left(\frac{-r^2}{2\sigma^2}\right)r^{m-1}\right) = 0$$

Applying the product rule,

$$\frac{d}{dr}\left(exp\left(\frac{-r^2}{2\sigma^2}\right)\right)r^{m-1} + exp\left(\frac{-r^2}{2\sigma^2}\right)\frac{d}{dr}\left(r^{m-1}\right) = 0$$

$$exp\left(\frac{-r^2}{2\sigma^2}\right)\frac{-2r}{2\sigma^2}r^{m-1} + exp\left(\frac{-r^2}{2\sigma^2}\right)(m-1)r^{m-2} = 0$$

Simplifying this,

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

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

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

$$r^2 = (m-1)\sigma^2$$

$$r = \sqrt{(m-1)}\sigma$$

With our assumption, when we have a large m we can assume that $m-1 \approx m$; thus,

$$\hat{r} \approx \sqrt{m}\sigma$$

(f)

We can show that for large m and a small value ϵ such that $\epsilon \ll \hat{r}$, that $\rho(\hat{r}+\epsilon) \approx \rho(\hat{r})e^{-\frac{\epsilon^2}{\sigma^2}}$, by first relating $\rho(\hat{r}+\epsilon)$ and $\rho(\hat{r})$ through a fractional relationship such that,

$$\frac{\rho(\hat{r}+\epsilon)}{\rho(\hat{r})} = \frac{\frac{\bar{S}_{m-1}}{(2\pi\sigma^2)^{m/2}}exp\left(\frac{-(\hat{r}+\epsilon)^2}{2\sigma^2}\right)(\hat{r}+\epsilon)^{m-1}}{\frac{\bar{S}_{m-1}}{(2\pi\sigma^2)^{m/2}}exp\left(\frac{-\hat{r}^2}{2\sigma^2}\right)\hat{r}^{m-1}}$$

Simplifying by eliminating common terms from top and bottom we're left with,

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

We can combine all terms into the exponential such that,

$$\frac{\rho(\hat{r}+\epsilon)}{\rho(\hat{r})} = \frac{exp\left(\frac{-(\hat{r}+\epsilon)^2}{2\sigma^2}\right)exp(\ln((\hat{r}+\epsilon)^{m-1})}{exp\left(\frac{-\hat{r}^2}{2\sigma^2}\right)exp(\ln(\hat{r}^{m-1}))}$$

$$\frac{\rho(\hat{r}+\epsilon)}{\rho(\hat{r})} = exp\left(\frac{-(\hat{r}+\epsilon)^2}{2\sigma^2} + \ln((\hat{r}+\epsilon)^{m-1}) + \frac{\hat{r}^2}{2\sigma^2} - \ln(\hat{r}^{m-1})\right)$$

Combining like terms we find,

$$\frac{\rho(\hat{r}+\epsilon)}{\rho(\hat{r})} = exp\left(\frac{-(\hat{r}+\epsilon)^2+\hat{r}^2}{2\sigma^2} + \ln((\hat{r}+\epsilon)^{m-1}) - \ln(\hat{r}^{m-1})\right)$$

$$\frac{\rho(\hat{r}+\epsilon)}{\rho(\hat{r})} = exp\left(\frac{-2\hat{r}\epsilon-\epsilon^2}{2\sigma^2} + \ln((\hat{r}+\epsilon)^{m-1}) - \ln(\hat{r}^{m-1})\right)$$

$$\frac{\rho(\hat{r}+\epsilon)}{\rho(\hat{r})} = exp\left(\frac{-2\hat{r}\epsilon-\epsilon^2}{2\sigma^2} + \ln\left(\frac{(\hat{r}+\epsilon)^{m-1}}{\hat{r}^{m-1}}\right)\right)$$

$$\frac{\rho(\hat{r}+\epsilon)}{\rho(\hat{r})} = exp\left(\frac{-2\hat{r}\epsilon-\epsilon^2}{2\sigma^2} + (m-1)\ln\left(1+\frac{\epsilon}{\hat{r}}\right)\right)$$

We can apply a 2-term Maclauren expansion approximation of the form $ln\,(1+x)$ such that,

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

Substituting $\hat{r} \approx \sqrt{m}\sigma$ as found in part (e),

$$\frac{\rho(\hat{r}+\epsilon)}{\rho(\hat{r})} \approx exp\left(\frac{-2\sqrt{m}\sigma\epsilon-\epsilon^2}{2\sigma^2} + (m-1)\left(\frac{\epsilon}{\sqrt{m}\sigma} - \frac{\epsilon^2}{2m\sigma^2}\right)\right)$$

With our assumption, when we have a large m we can assume that $m-1 \approx m$; thus,

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

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

Simplifying we find,

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

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

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

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

Thus, we can see that for large m and a small value $\epsilon \ll \hat{r}$, $\rho(\hat{r}+\epsilon) \approx \rho(\hat{r})exp\left(\frac{-\epsilon^2}{\sigma^2}\right)$.

(g)

For the high dimensional Gaussian distribution (m is large), the majority of our points will reside at our value $\sqrt{m}$ as we see in our equation $\hat{r} \approx \sqrt{m}\sigma$ where σ is small compared to $\hat{r}$.

For a low dimensional Guassian distribution, most points will reside around the origin $x = 0$ because most of the mass is centered at the origin within the window of roughly σ.

(h)

Calculating the probability density at the origin, we find that

$$\rho_m(0) = \frac{exp\left(\frac{-(0)^2}{2\sigma^2}\right)}{(2\pi\sigma^2)^{m/2}}\bar{S}_{m-1}(0)^{m-1} = 0$$

Calculating the probability density at a point on the sphere,

$$\rho_m(\hat{r}) = \frac{exp\left(\frac{-(\hat{r})^2}{2\sigma^2}\right)}{(2\pi\sigma^2)^{m/2}}\bar{S}_{m-1}(\hat{r})^{m-1}$$

For large m, i.e. high dimensionality, we can approximate $\hat{r}$ as $\sqrt{m}\sigma$, such that

$$\rho_m(\hat{r}) = \frac{exp\left(\frac{-(\sqrt{m}\sigma)^2}{2\sigma^2}\right)}{(2\pi\sigma^2)^{m/2}}\bar{S}_{m-1}(\sqrt{m}\sigma)^{m-1}$$

$$\rho_m(\hat{r}) = \frac{exp\left(\frac{-m}{2}\right)}{(2\pi\sigma^2)^{m/2}}\bar{S}_{m-1}(\sqrt{m}\sigma)^{m-1}$$

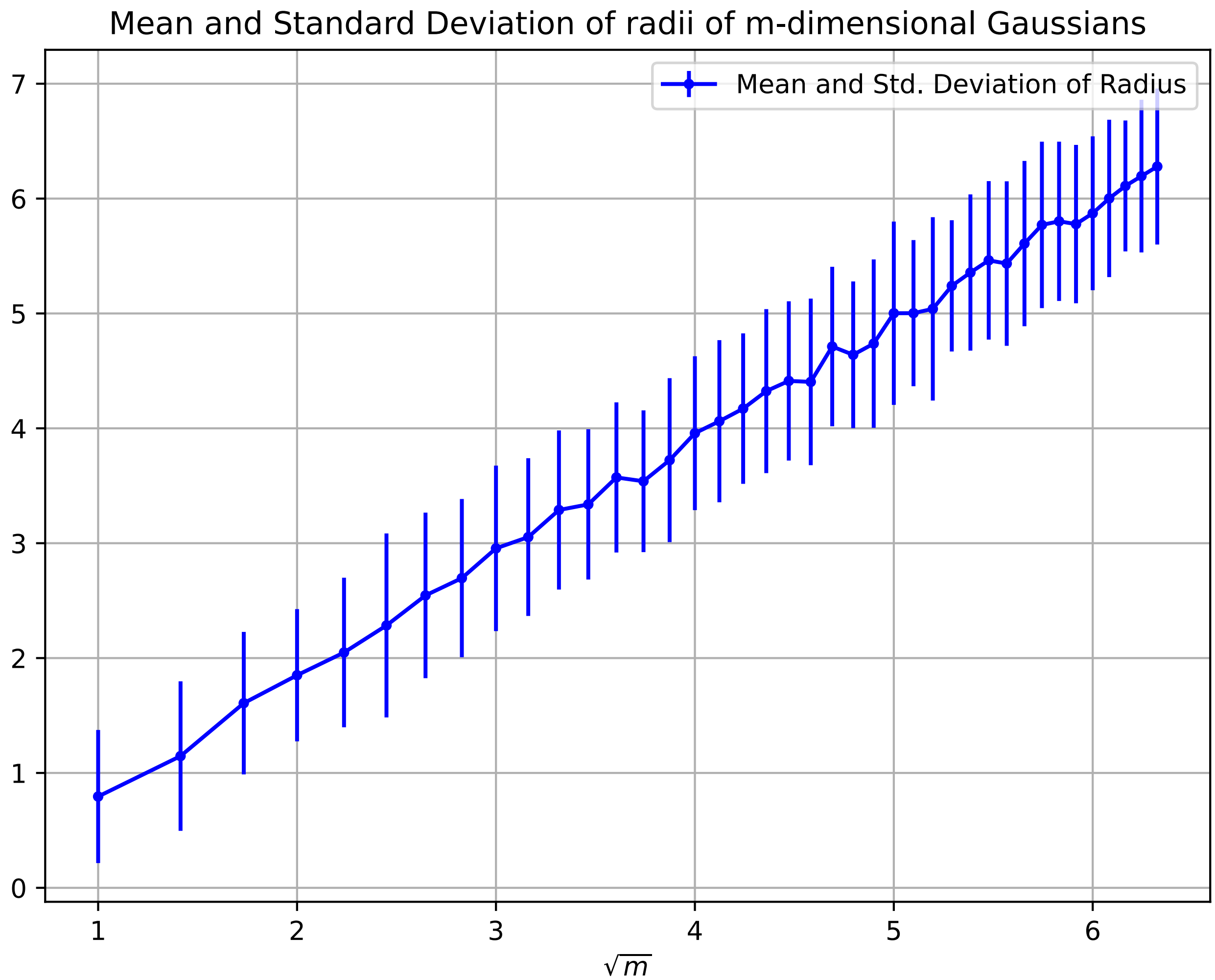
(i)

From the plot generated by the plot below, we can see that many of the relationships we calculated above hold true. We can see that there is a linear relationship between mean of our radii and the root of m.

This aligns with our logic as we defined $\hat{r} \approx \sqrt{m}\sigma$ as m increases. In the case outlined below, our σ was set at a value of 1 such that our $\hat{r}$ is directly a function of $\sqrt{m}$. It is expected that $\hat{r}$ should align with where the mean of our radii as this is the "maximum" of our distribution for the radii. This maximum should occur where the mean of the distribution occurs.

In [23]:

```
m = np.arange(40) + 1
sample_size = 100

# Set an arbitrary mean and std. deviation value for our m-dimensional Gaussian
mean_G = 0
sigma_G = 1

mean = np.empty(shape=(40))
std_dev = np.empty(shape=(40))

for m_i in m:
    # For each value of m (i.e. each m-dimensional Gaussian), generate a random sample
    cov_i = np.identity(m_i) * sigma_G**2
    mean_i = np.ones(m_i) * mean_G
    x_i = np.random.multivariate_normal(mean_i, cov_i, size=100)

    # Calculate the radii of the samples (i.e. the sqrt(x^2) = abs(x))
    r_i = np.sum(x_i**2, axis=1)**(1/2)
    #print('\n', x_i.shape, r_i.shape)

    # Calculate the mean and std deviation
    mean[m_i - 1] = np.mean(r_i)
    std_dev[m_i - 1] = np.std(r_i)

#print(mean.shape, std_dev.shape)

# Generate a 2D plot of mean/std dev. vs. m
fig, ax = plt.subplots(figsize=(8, 6))
#plt.plot(m, mean, ls = '-', marker = '.', c = 'blue', label = 'Mean')
#plt.plot(m, std_dev, ls = '-', marker = '.', c='red', label = 'Standard Deviation')
plt.errorbar(np.sqrt(m), mean, yerr=std_dev, marker='.', c='blue', label='Mean and Std.
#plt.errorbar(m, mean, yerr=std_dev, marker='.', c='red', label='Mean and Std. Deviatio
plt.grid()
plt.legend()
plt.xlabel('$\sqrt{m}$')
plt.title('Mean and Standard Deviation of radii of m-dimensional Gaussians')
fig.show()
plt.savefig('Problem_2_i.png') # If saving a file
```



3. Programming Problem: Lasso

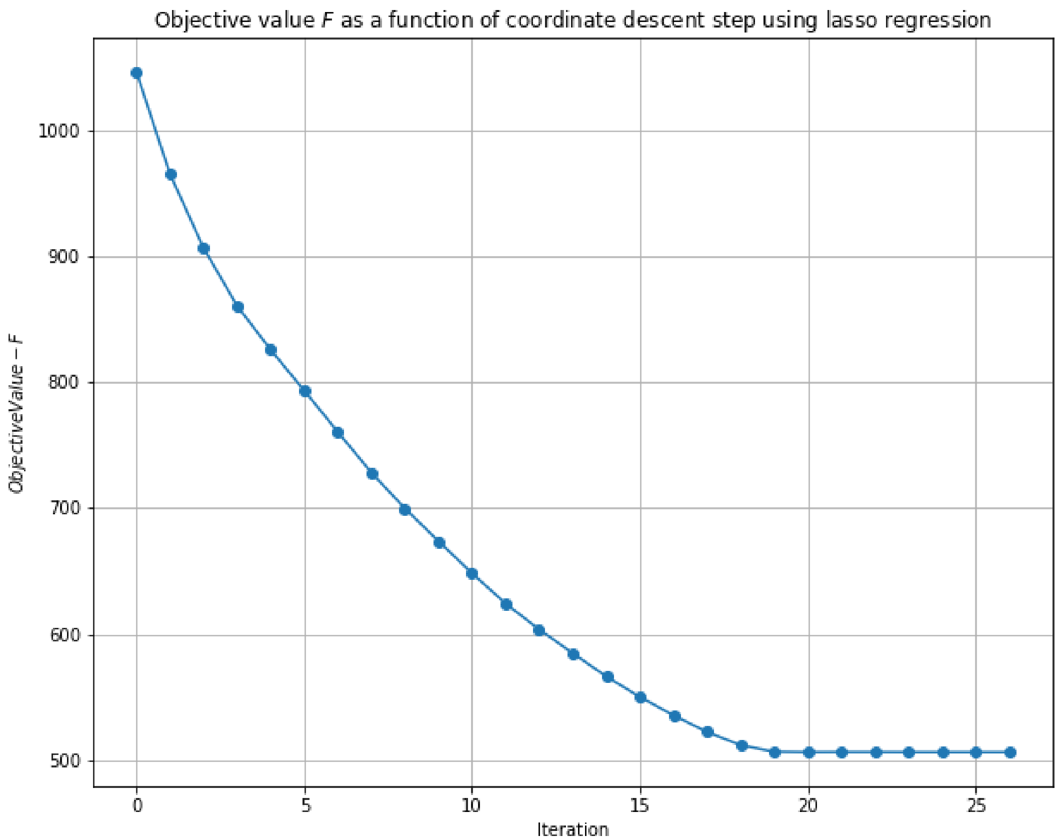
(a)

Captured within the *lass.ipynb* file is the coordinate descent algorithim.

From this algorithim, the reported non-zero weight entries are

$$\{\text{Non-zero indices of } \theta\} = \{0, 1, 2, 3, 4, 7, 11, 12, 14, 16, 21, 22, 23, 25, 34, 36, 48, 52, 54, 61, 63, 71\}$$

In addition, the plot below showing the objective value during gradient descent is captured below.

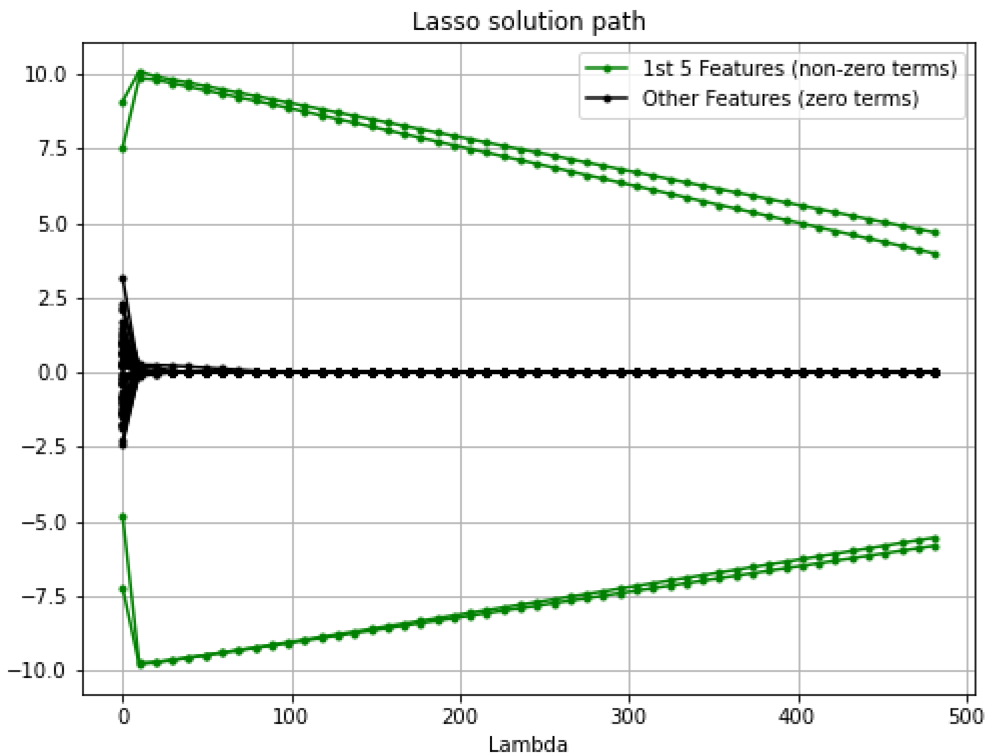


(b)

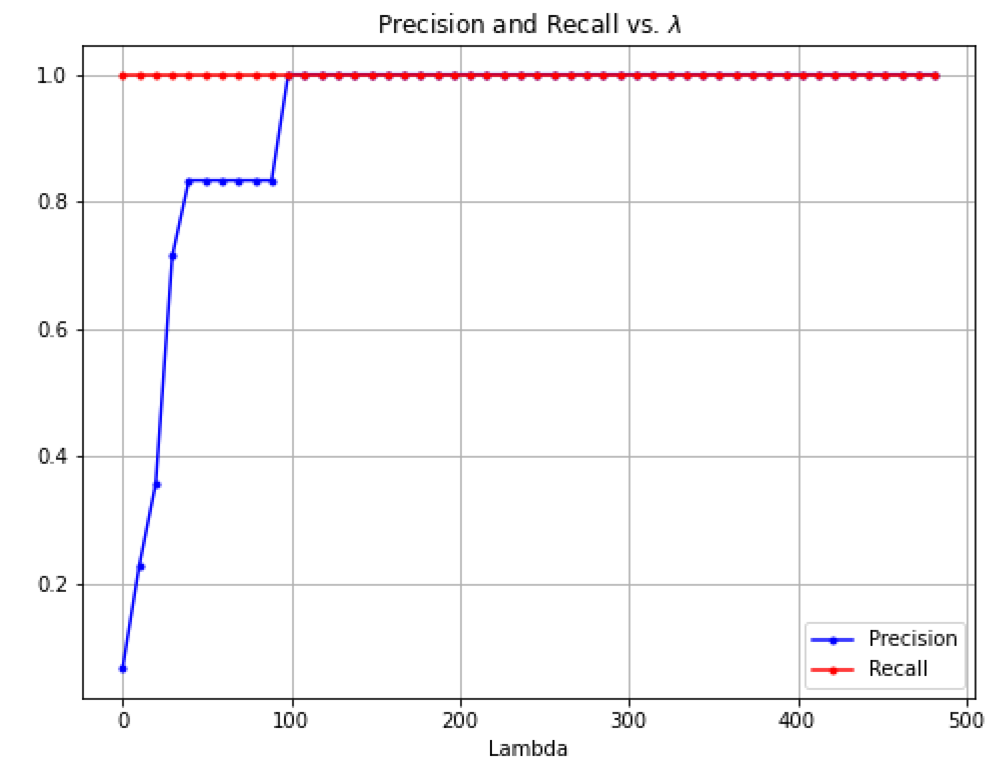
From the *lasso.ipynb*, $\text{precision} = 0.22727272727272727$ and $\text{recall} = 1.0$ for the lasso solution in part (a).

(c)

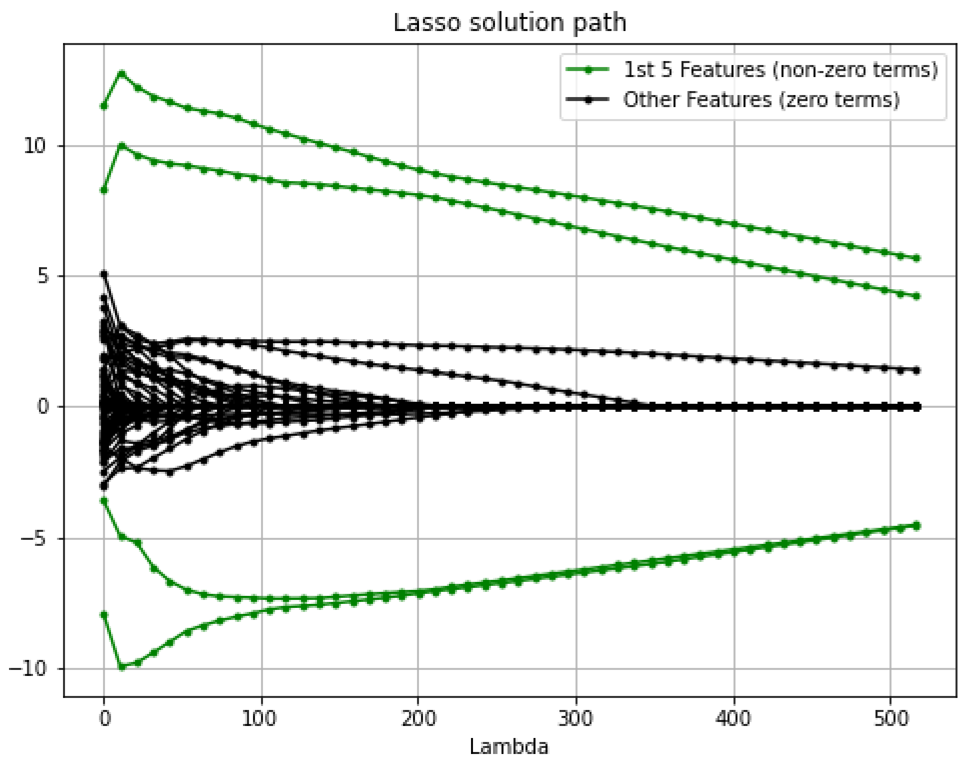
From the *lasso.ipynb*, we can plot the precision and recall vs. λ as shown in the plot below.



We can also plot the lasso solution path for the varying values of λ as shown in the plot below.



Varying our noise's standard deviation to $\sigma = 10$ through the DataGenerator function, we find the following results.



Comparing the two lasso solution paths, we can see the impact noise has on calculating our weights for our lasso solution. With an increases variance in the induced noise, we are more likely to see more impactful incorrect weights (color coded by black lines) at lower lambdas. For our lower noise std. deviation, these incorrect weights were nearly zero for the majority of our values for lambda. This was not the case when we increase the noise std. deviation to 10 as we see that multiply incorrect weights are non-zero for all lambdas.

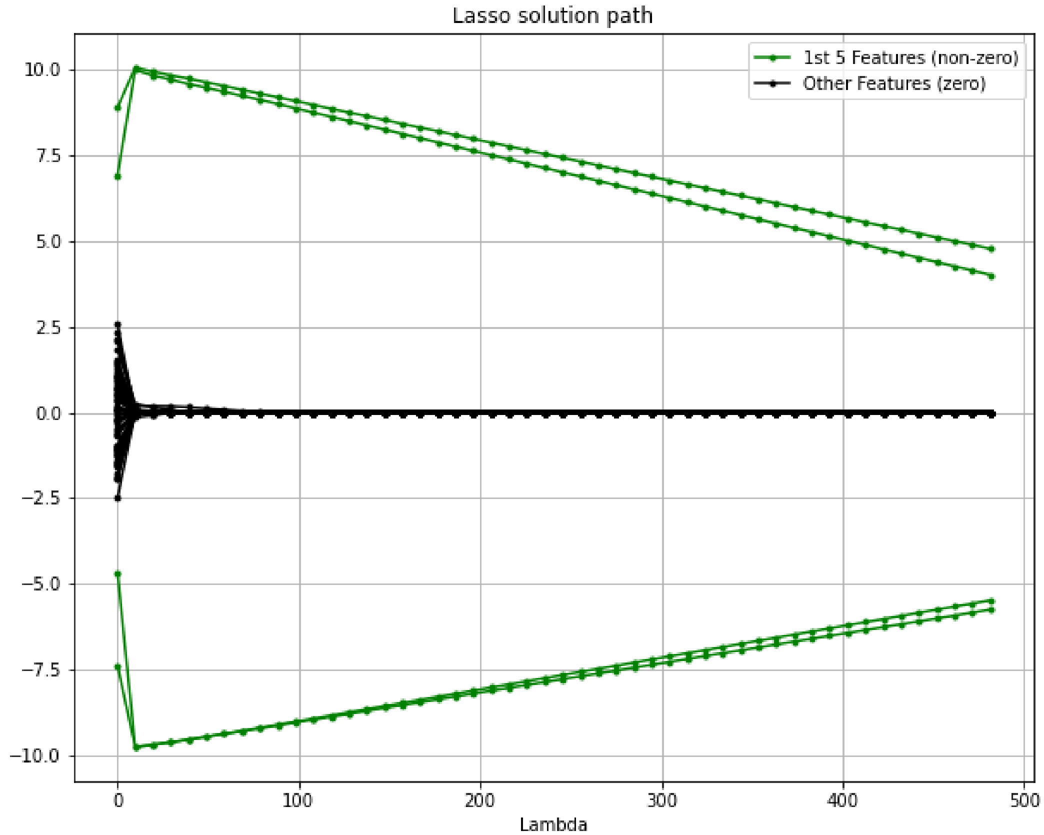
(d)

With varying input parameters for DataGenerator, below are the plots of both (1) precision and recall vs. λ and (2) the lasso solution path.

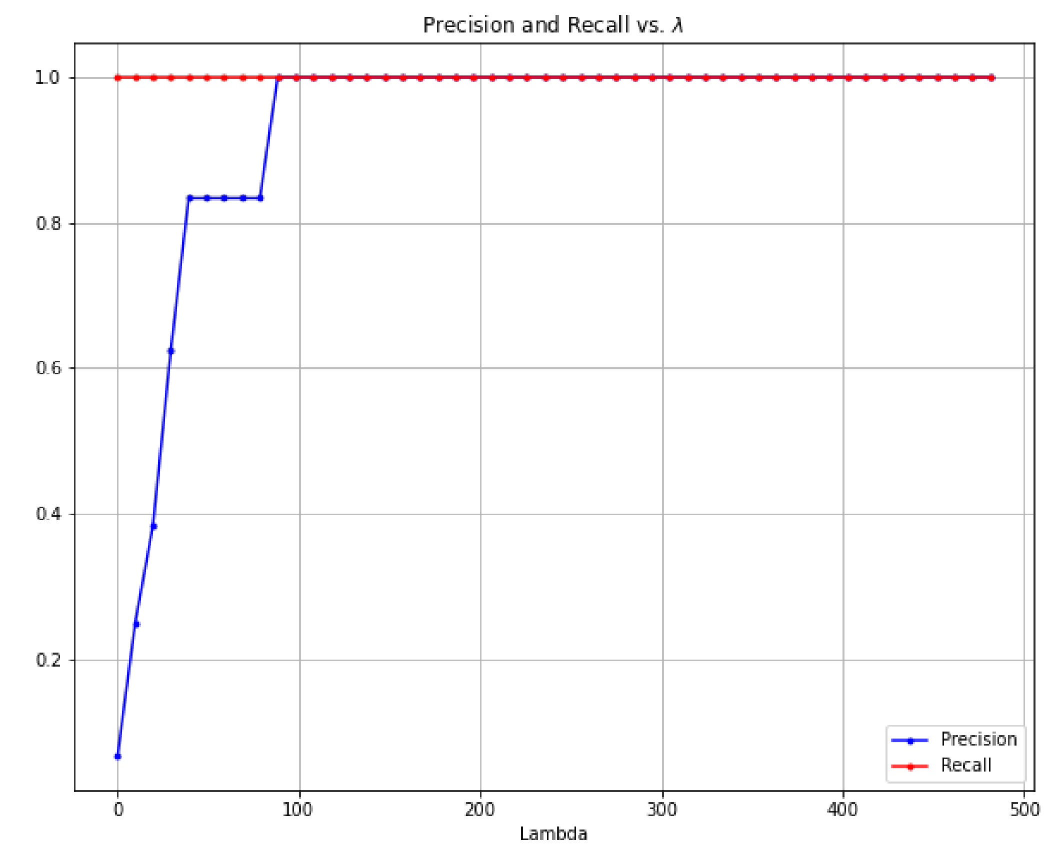
From the graphs below, we can surmise that an ideal λ for these various input parameters we when we can maximize both precision and recall.

Based on the graphs below, it seems that a good precision and recall value is obtained when $n = O(log(m))$. It seems that when $n = O(m^2)$ the precision and recall can exhibit strange behavior (as seen in our $(n = 50, m = 1000)$), and $n = O(m)$ yields precision and recall values that are undesirable when we have more accurate weight values (i.e. smaller lambda). Thus it seems that $n = O(log(m))$ is an desirable relationship.

For $(n = 50, m = 75)$,

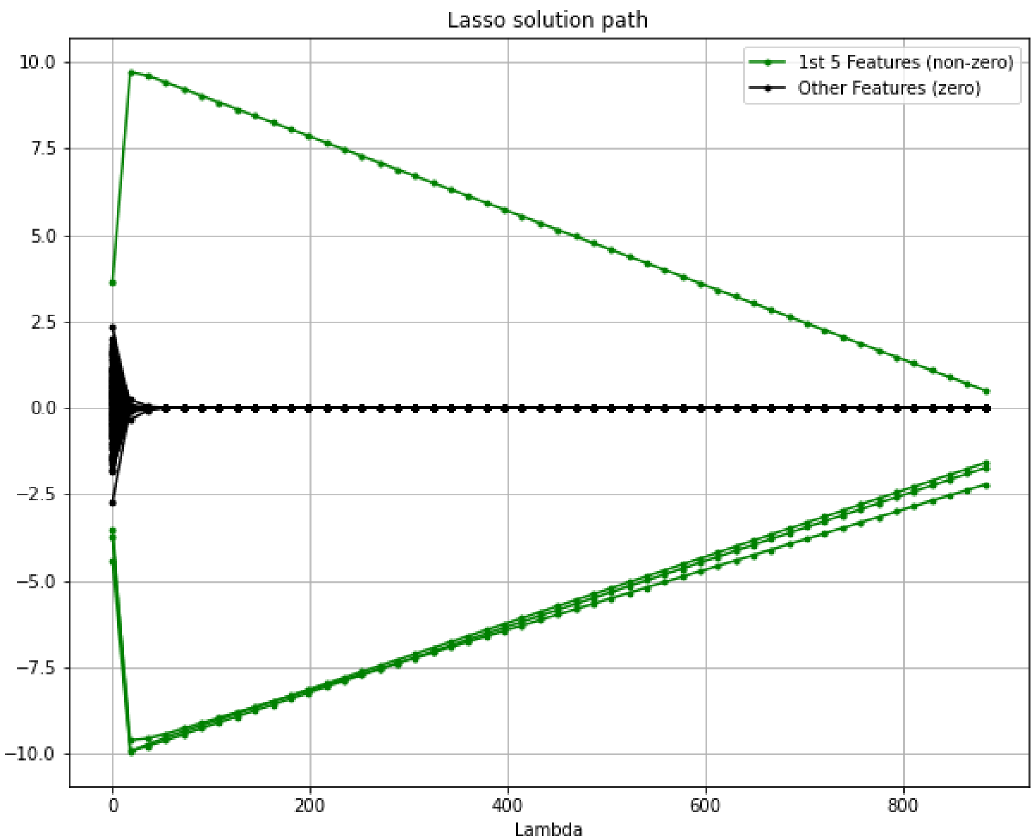


Lasso solution path
1st 5 Features (non-zero)
Other Features (zero)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
100
200
300
400
500
Lambda

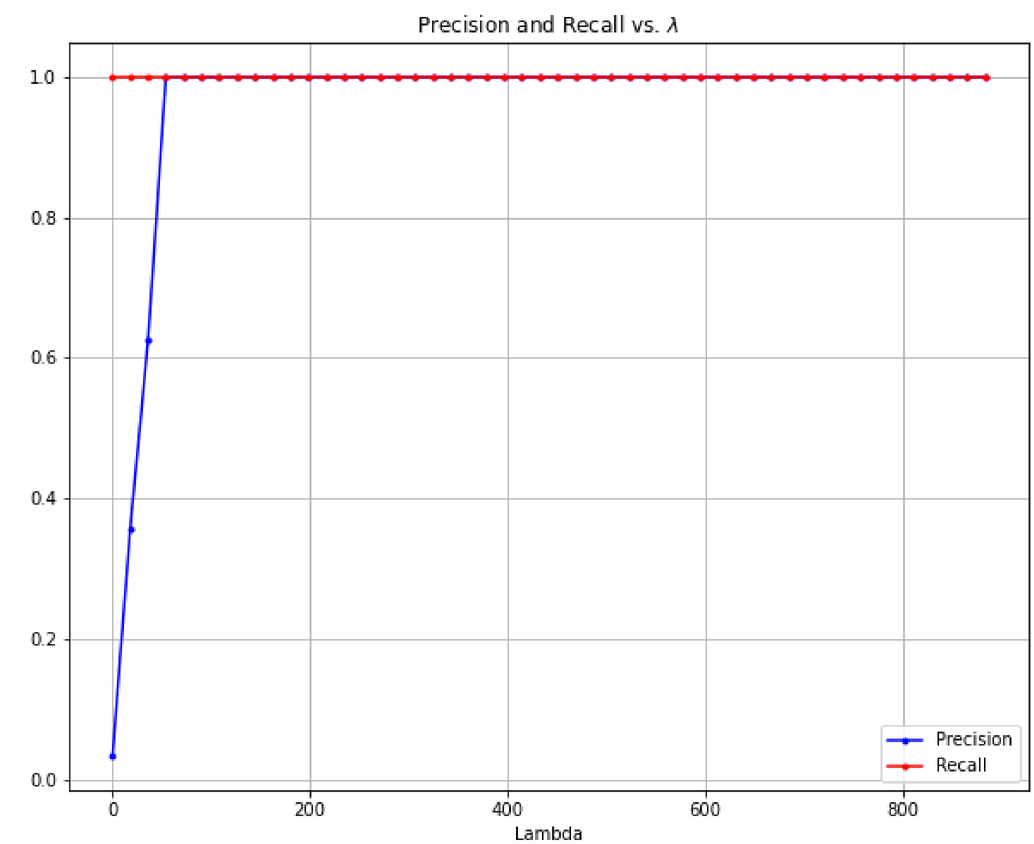


Precision and Recall vs. λ
1.0
0.8
0.6
0.4
0.2
Precision
Recall
0
100
200
300
400
500
Lambda

For $(n = 50, m = 150)$,

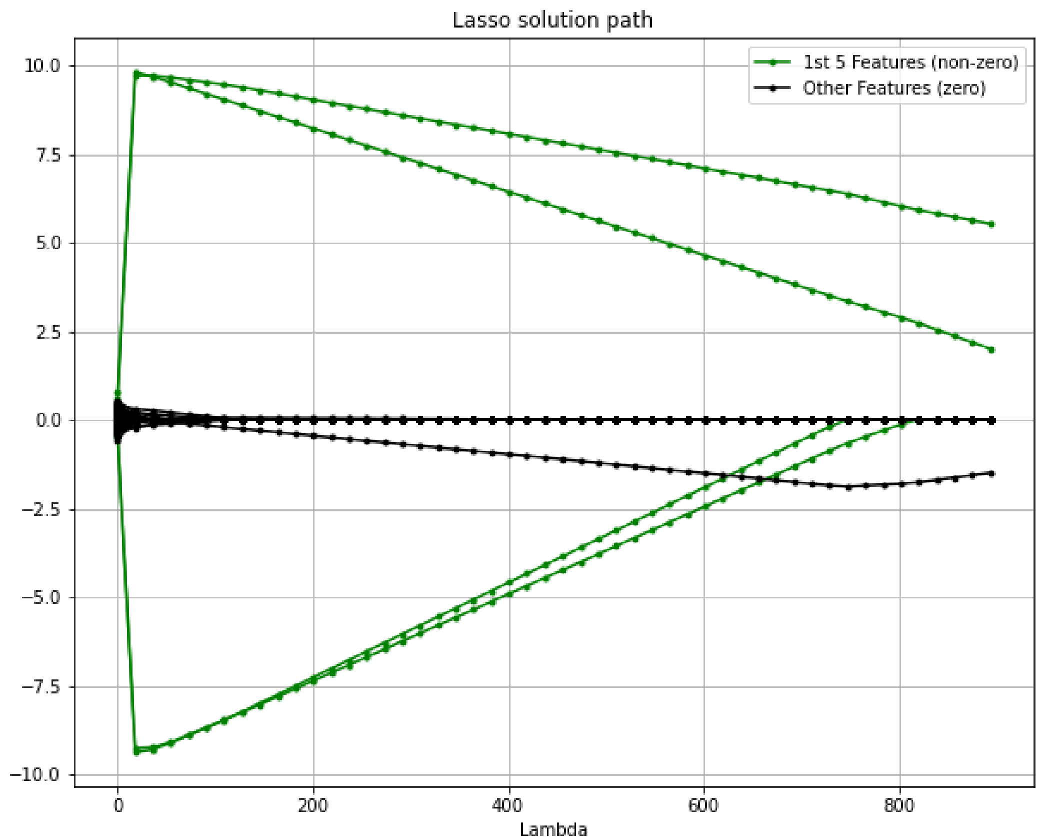


Lasso solution path
1st 5 Features (non-zero)
Other Features (zero)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
200
400
600
800
Lambda

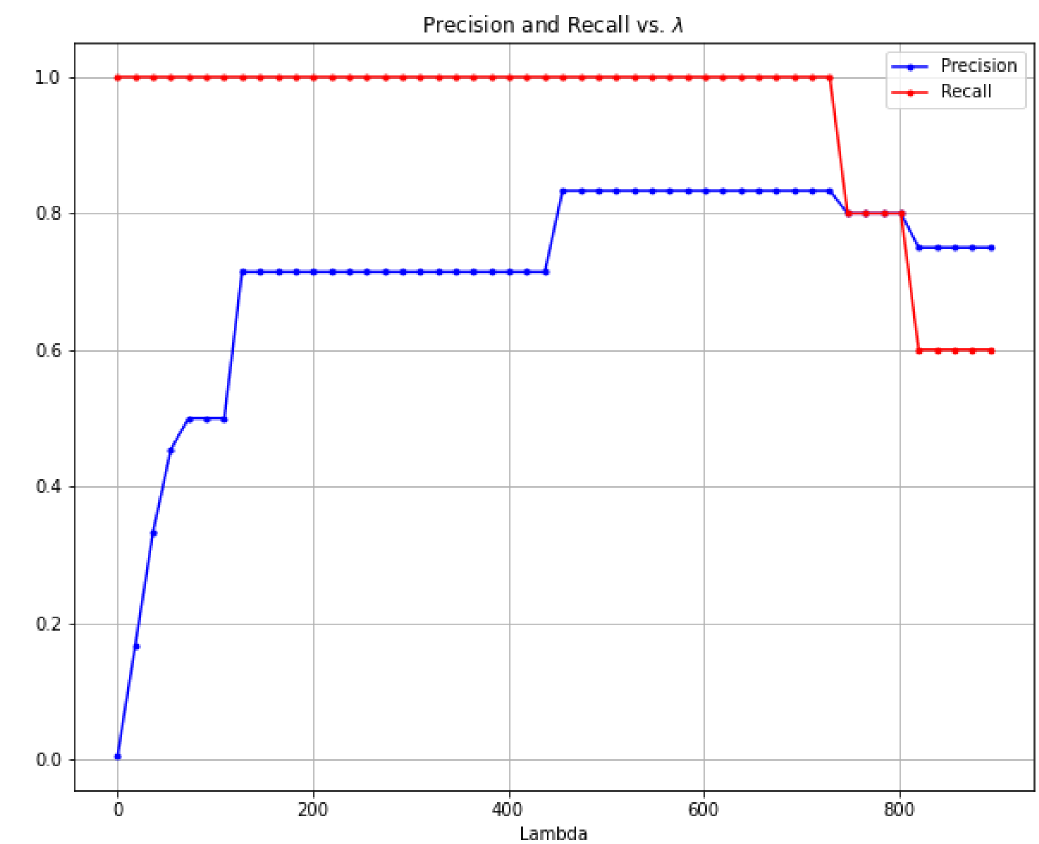


Precision and Recall vs. λ
1.0
0.8
0.6
0.4
0.2
0.0
Precision
Recall
0
200
400
600
800
Lambda

For $(n = 50, m = 1000)$,

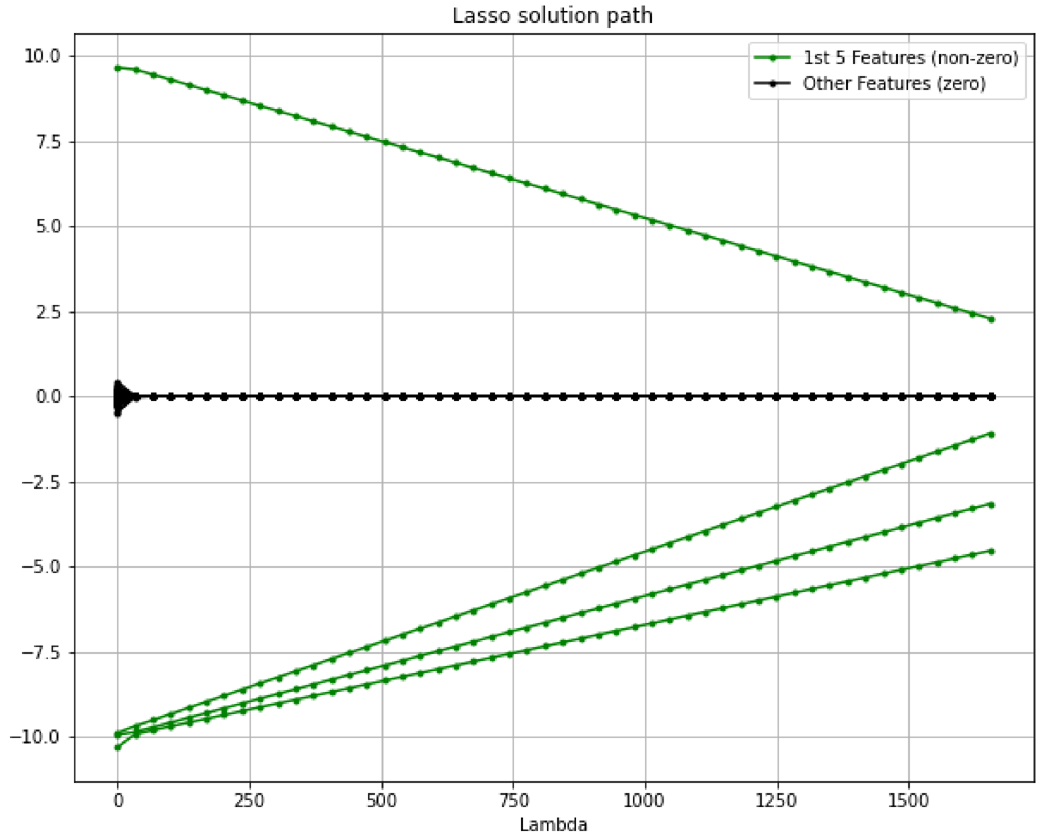


Lasso solution path
1st 5 Features (non-zero)
Other Features (zero)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
200
400
600
800
Lambda

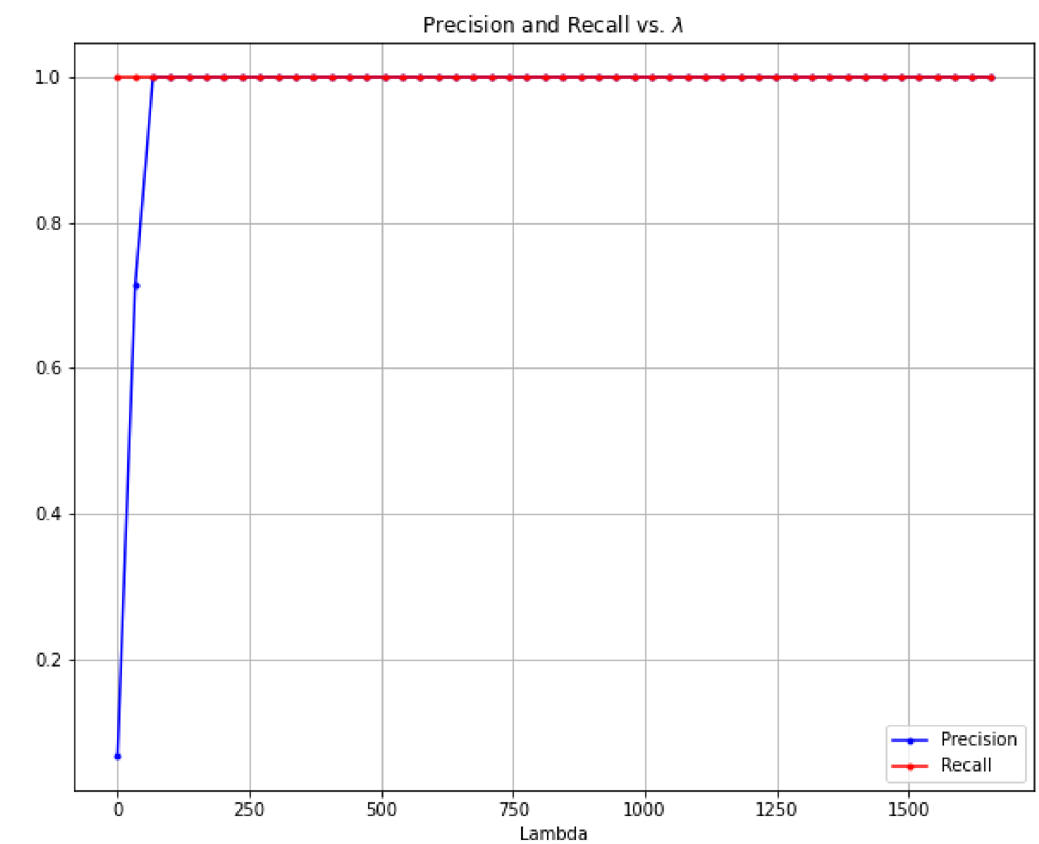


Precision and Recall vs. λ
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
Lambda

For $(n = 100, m = 75)$,

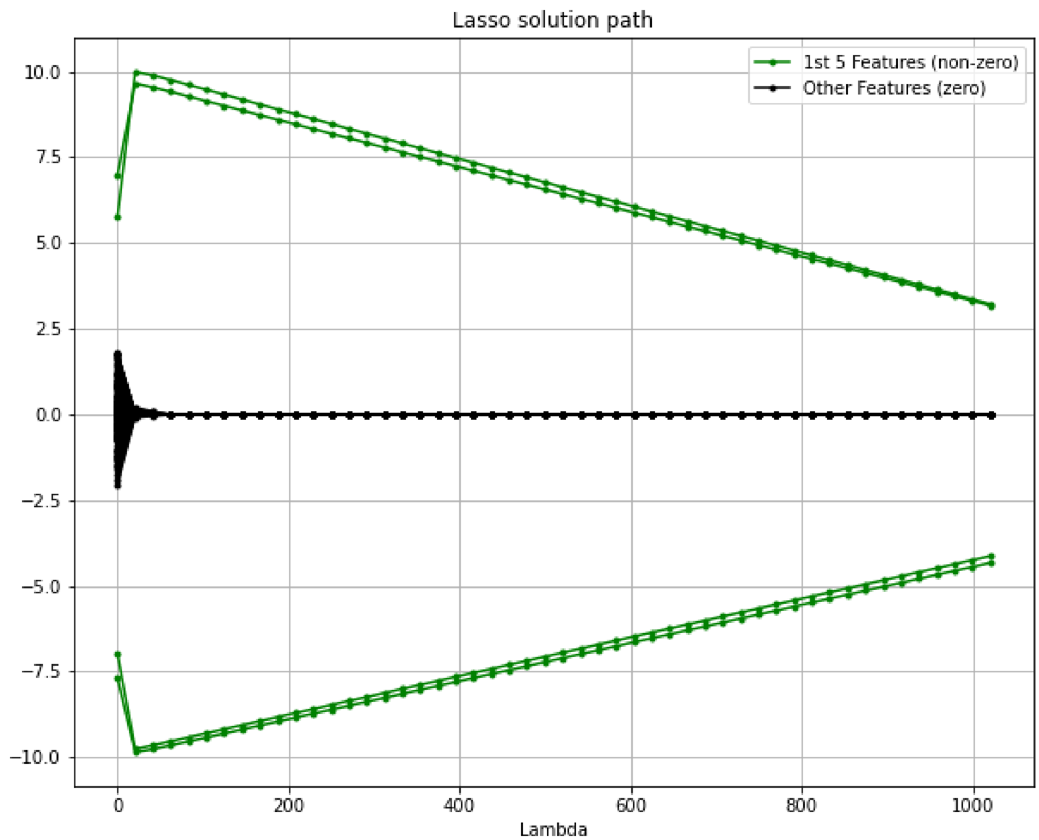


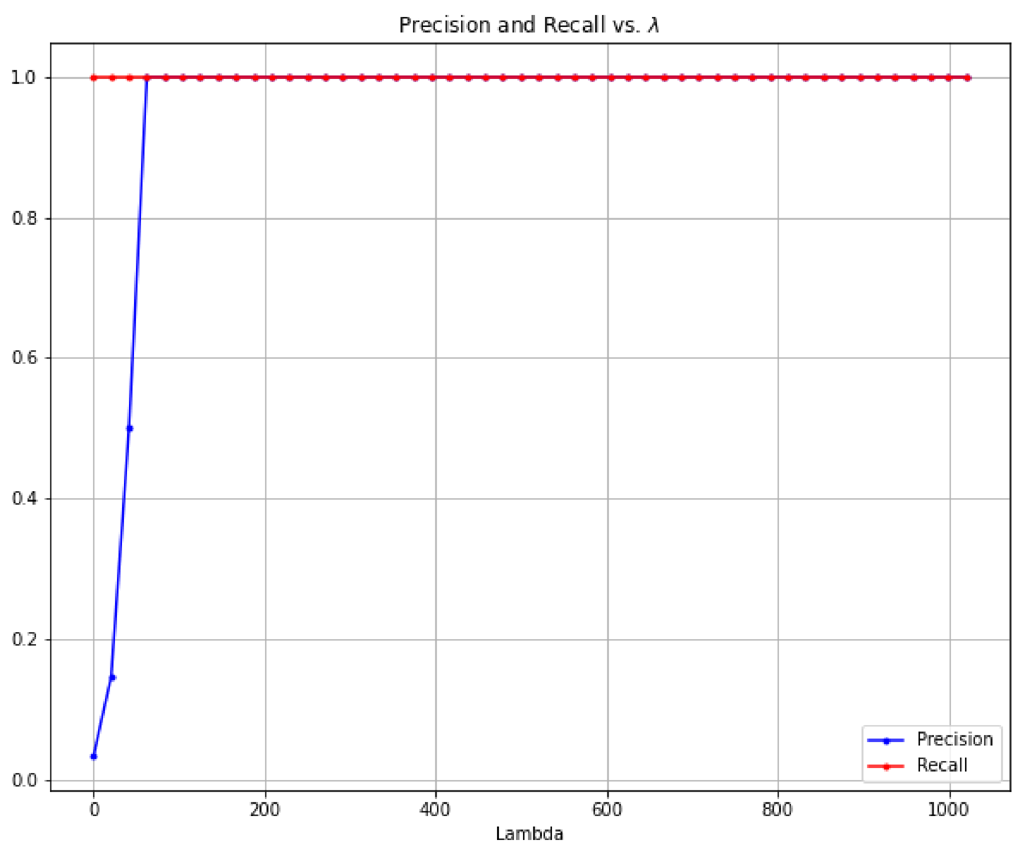
Lasso solution path
1st 5 Features (non-zero)
Other Features (zero)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
250
500
750
1000
1250
1500
Lambda



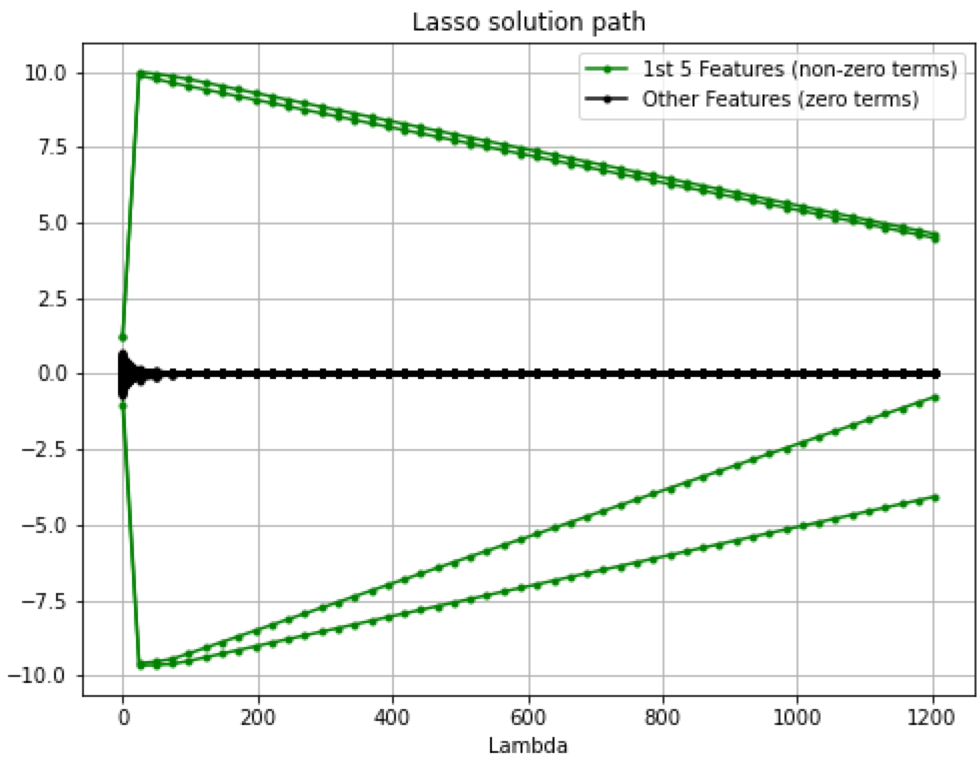
Precision and Recall vs. λ
1.0
0.8
0.6
0.4
0.2
Precision
Recall
0
250
500
750
1000
1250
1500
Lambda

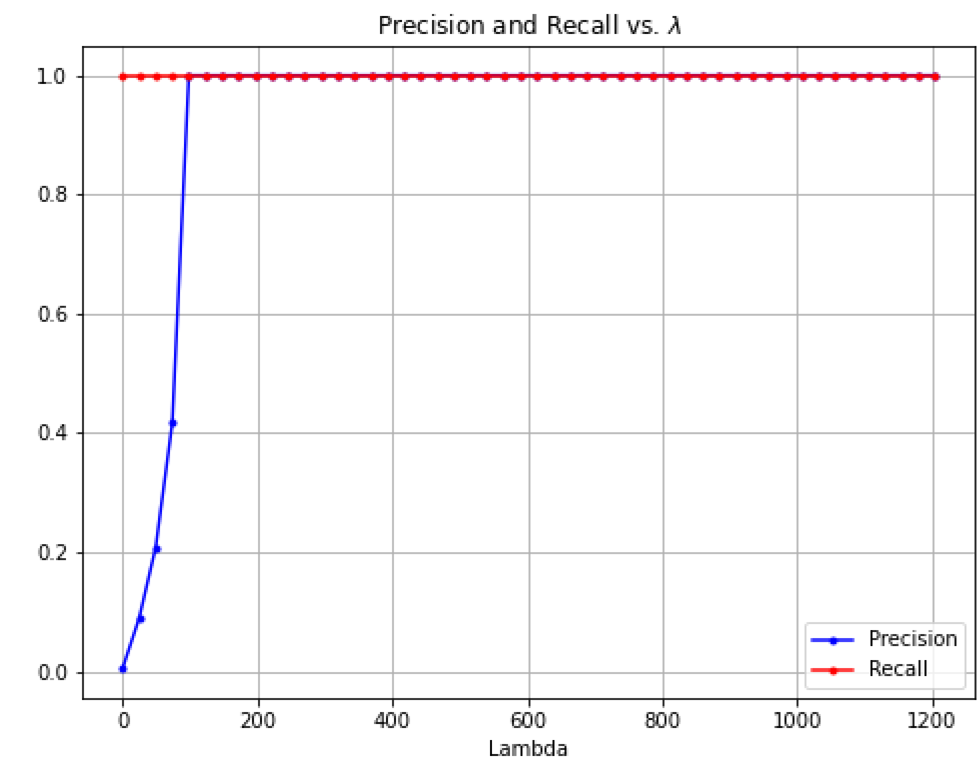
For $(n = 100, m = 150)$,


Lasso solution path
1st 5 Features (non-zero)
Other Features (zero)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
200
400
600
800
1000
Lambda


Precision and Recall vs. λ
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
Lambda

For $(n = 100, m = 1000)$,





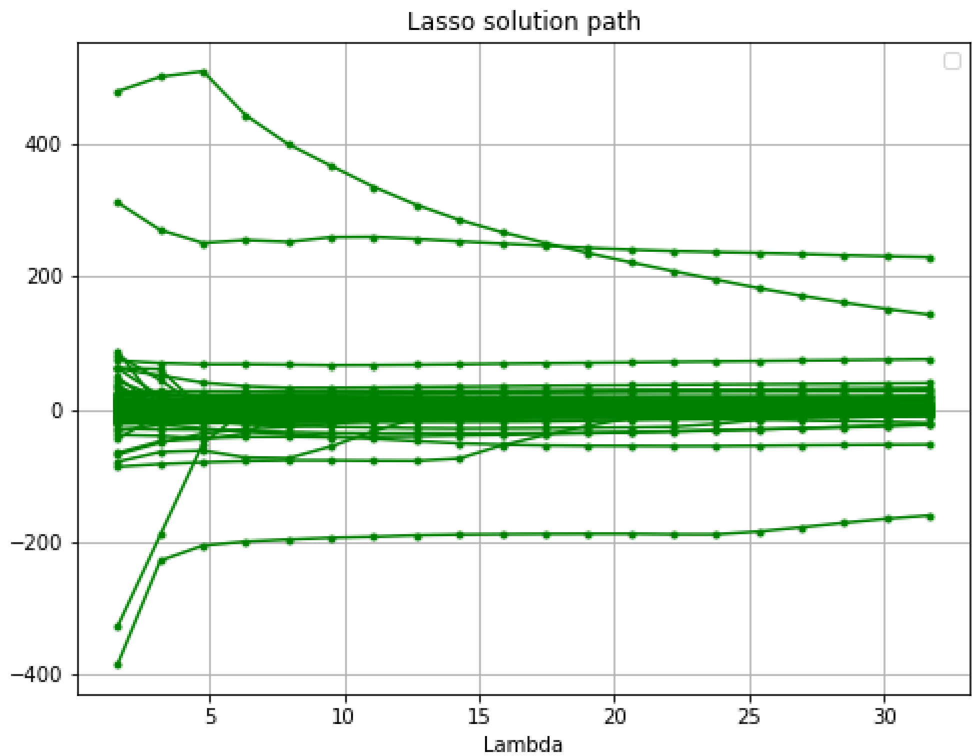
(e)

This portion was completed late and submitted on 2/23.

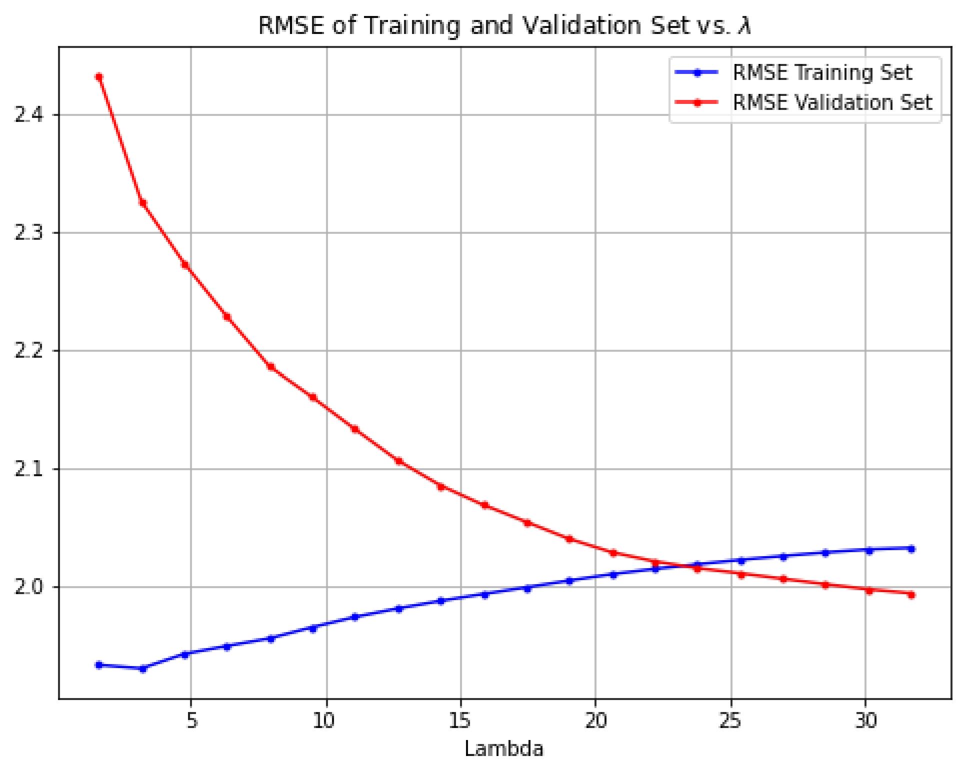
Below is a plot of the lasso solution path and a plot of RMSE (for both the validation and test set). In addition, the top-10 features with the largest magnitude are identified below.

Feature	Magnitude
and	228.92007242281383
were soaked in	-159.55591374549522
the	142.08381416889432
great	75.10155839764127
set	-52.662879026941425
best	39.82050044662048
love	32.49102678604067
amazing	30.24397828436301
delicious	27.33324807664297
of a	-22.706295466396767

Based on the results of the top-10 features, it makes sense that features relating to positive features ("great", "best", "love", "amazing") have a positive correlation to predicting the stars of a yelp review. In contrast, negative ("were soaked in") had a strong negative correlation to the stars yelp review. It is of importance to notice that many of the top features include conjunction terms like "and" and "of a" which can considered potential noise in our solution.



Lasso solution path
400
200
0
−200
−400
5
10
15
20
25
30
Lambda



RMSE of Training and Validation Set vs. λ
RMSE Training Set
RMSE Validation Set
2.4
2.3
2.2
2.1
2.0
5
10
15
20
25
30
Lambda

In [ ]:

In [18]:

```
# 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
import datetime
from datetime import datetime
%matplotlib inline
```

In [19]:

```
# synthetic data generator
# n is number of samples, d is number of dimensions, k is number of nonzeros in w, sigm
# X is a n x d data matrix, y=Xw+w_0+noise is a n-dimensional vector, w is the true wei
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)
```

In [20]:

```
# 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))

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

In [21]:

```
# 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):
    indptr = W.indptr
    last_idx = indptr[0]
    for col_id, idx in enumerate(indptr[1:]):
        if idx == last_idx:
            continue
        else:
            W.data[last_idx:idx] *= x[col_id]
            last_idx = idx
```

In [22]:

```
# Problem 2.1
def lasso(X, y, lmda = 10.0, epsilon = 1.0e-2, max_iter = 100, draw_curve = False):
    # Initialize the weights using provided function
    w, w0 = Initialw(X, y)

    n, m = X.shape
    #print(n, m, y.shape, w.shape, w0.shape)

    # Initialize our iteration, max change, and objective function
    iter = 0
    F = (1/2) * np.sum(X @ w + w0 -  y) + lmda * np.sum(np.absolute(w))

    while True:
        # Update iterator for given loop, reset our max_change, and calculate our new y
        iter += 1
        max_change = 0

        for k in range(m):
            # Solve for r_k
            r_k = y - np.delete(X, [k], axis=1) @ np.delete(w, [k])

            # Solve for a_k and c_k
            X_k = X[:, k]
            a_k = 2 * np.sum(X_k**2)
            c_k = 2 * np.sum(np.multiply(r_k, X_k))

            # Calculate new w_k, cross-compare new weight to old weight to determine if
            w_k = np.sign(c_k) * np.maximum(0, np.absolute(c_k) - lmda) / a_k
            if np.absolute(w_k - w[k]) > max_change:
                max_change = np.absolute(w_k - w[k])

            w[k] = w_k
            #print(w_k)

        # Calculate our new w0
        w0 =  np.mean(y) - np.mean(X, axis=0) @ w

        # Calculate our new objective value F
        F_new = (1/2) * np.sum(X @ w + w0 -  y) + lmda * np.sum(np.absolute(w))
        F = np.append(F, F_new)

        # After updating our weights, check against exit conditions: (1) number of step
        if iter > 100 or max_change <= epsilon:
            break

    # If draw_curve set to true, draw a plot of objective value with respect to coordin
    if draw_curve == True:
```

```
        fig, ax = plt.subplots(figsize=(10, 8))
        plt.plot(F, ls = '-', marker = 'o', label = '')
        plt.grid()
        plt.xlabel('Iteration')
        plt.ylabel('$Objective Value - F$')
        plt.title('Objective value $F$ as a function of coordinate descent step using l
        fig.show()
        plt.savefig('Problem_3_a.png')

    return (w,w0)
```

In [23]:

```
# Problem 2.1: 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)
```

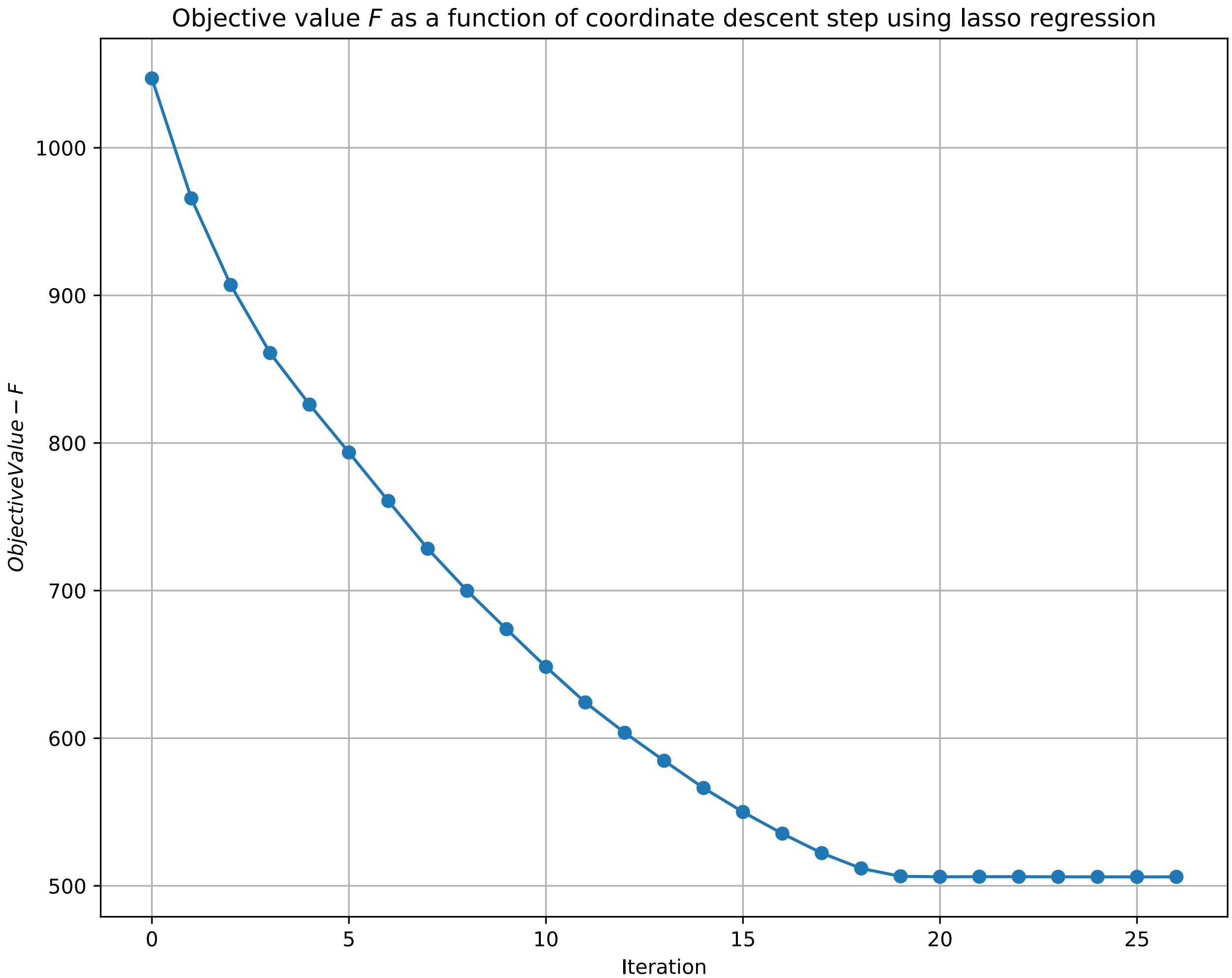
```
[[ 0.10430293 -0.55011253 -0.07271465 ...  0.9858945   0.9762621
   0.66088793]
 [-1.00421694 -0.98028568  1.04231343 ...  0.54423528 -0.12555319
   0.29833038]
 [-0.93920808 -0.88460697 -0.36846914 ...  1.13839265 -0.17706563
  -1.1040073 ]
 ...
 [ 0.22627269 -1.41473902 -1.38744153 ...  0.40629811  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
```

In [24]:

```
# Problem 2.1: run lasso and plot the convergence curve
# TODO: run lasso for one synthetic data
w_lasso, w0_lasso = lasso(X, y, lmda = 10.0, epsilon = 1.0e-2, draw_curve = True, max_i
# have a look at the lasso model you got (sparse? where?)
print(w_lasso)
```

```
[ 9.87701348 -9.72978372 -9.78533191  9.85809008 10.08126976  0.
 -0.          0.04990917 -0.         -0.         -0.         -0.02029786
  0.22101146 -0.          0.10649288 -0.          0.12528557  0.
 -0.         -0.          0.          0.01886022  0.01337888  0.01044934
  0.         -0.02632922 -0.          0.         -0.          0.
```

```
-0.          -0.          0.           0.           0.08118375  0.
-0.06709603 -0.          -0.          -0.           0.          0.
 0.           0.          0.           0.          -0.          0.
 0.04079482 -0.           0.          -0.           0.25100621 -0.
-0.01947527 -0.           0.           0.          -0.          0.
-0.          -0.03668451  0.          -0.13301942  0.          0.
-0.           0.          0.           0.          -0.         -0.05597532
-0.           0.          0.         ]
```



In [25]:

```
# Problem 2.2
def pred_fn(X, theta, theta_0):
    pred = X @ theta + theta_0
    return pred

def root_mean_square_error(pred, y):
    rmse = np.sqrt(np.sum((pred - y)**2) / np.size(y))
    return rmse

def Evaluate(X, y, w, w0, w_true, w0_true):
    # First calculate the precision and recall - find the indices of each non-zero term
    w_nonzero = np.nonzero(w)
    w_true_nonzero = np.nonzero(w_true)

    precision_w = np.size(np.intersect1d(w_nonzero, w_true_nonzero)) / np.size(w_nonzer
    recall_w = np.size(np.intersect1d(w_nonzero, w_true_nonzero)) / np.size(w_true_nonz

    # Calculate RMSE using equation (5) from homework 2
    rmse = root_mean_square_error(pred_fn(X, w, w0), y)
```

```
    # Calculate sparsity
    sparsity_w = np.size(w_nonzero)

    return (rmse, sparsity_w, precision_w, recall_w)
```

In [26]:

```
# Problem 2.2
# TODO: apply your evaluation function to compute precision (of w), recall (of w), spar
Emetric = Evaluate(X, y, w_lasso, w0_lasso, w_true, w0_true)
print(Emetric)
```

(0.6112410261518306, 22, 0.22727272727272727, 1.0)

In [27]:

```
# Problem 2.3
# TODO: compute a lasso solution path, draw the path(s) in a 2D plot
def LassoPath(X, y, filename='temp.png', lmda_start = 0):
    lmda_max = np.amax((y - np.average(y)).T @ X)
    n, m = X.shape

    l_range = 50
    Lmda = np.linspace(1*lmda_start, lmda_max, num=l_range)
    W = np.empty((m, 50))
    W0 = np.empty((1, 50))
    #print(Lmda)

    # Calculate our weights for each lambda and save to our value for W
    for i in range(l_range):
        w_lasso, w0_lasso = lasso(X, y, lmda = Lmda[i], epsilon = 1.0e-2, draw_curve =
        W[:, i] = w_lasso
        W0[:, i] = w0_lasso

    # Generate a 2D plot of our lasso solution path
    fig, ax = plt.subplots(figsize=(8, 6))
    plt.plot(Lmda, W.T[:, 1:5], ls = '-', marker = '.', c = 'green', label = '1st 5 Fea
    plt.plot(Lmda, W.T[:, 5:], ls = '-', marker = '.', c='black', label = 'Other Featur

    # Remove the duplicate labels from our labels to create a succient legend
    handles, labels = plt.gca().get_legend_handles_labels()
    newLabels, newHandles = [], []
    for handle, label in zip(handles, labels):
        if label not in newLabels:
            newLabels.append(label)
            newHandles.append(handle)

    plt.legend(newHandles, newLabels)

    plt.grid()
    #plt.legend()
    plt.xlabel('Lambda')
    plt.ylabel('')
    plt.title('Lasso solution path')
    fig.show()
    plt.savefig(filename) # If saving a file

    return (W, W0, Lmda)
```

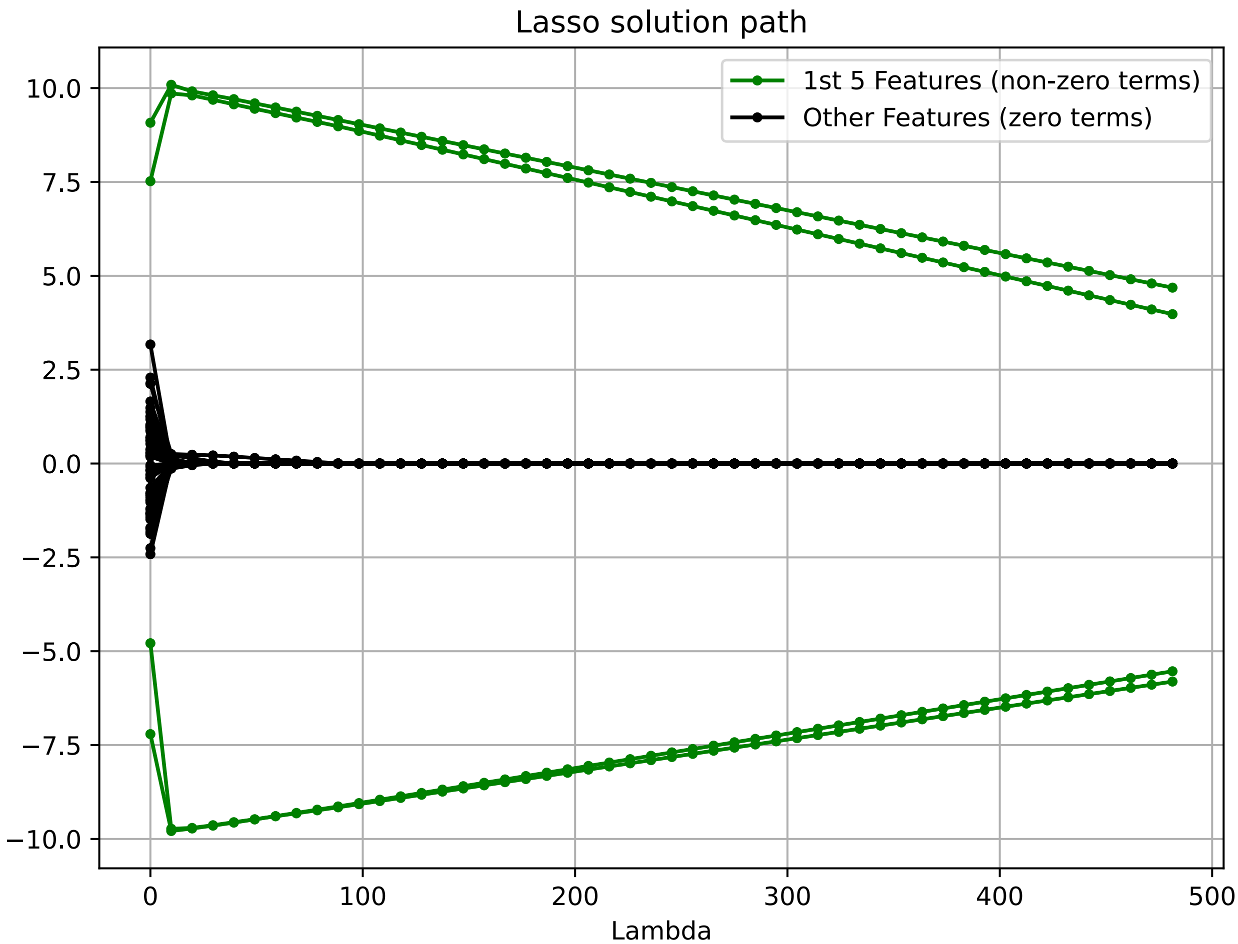
In [28]:

```
# Problem 2.3
# TODO: evaluate a given lasso solution path, draw plot of precision/recall vs. lambda
```

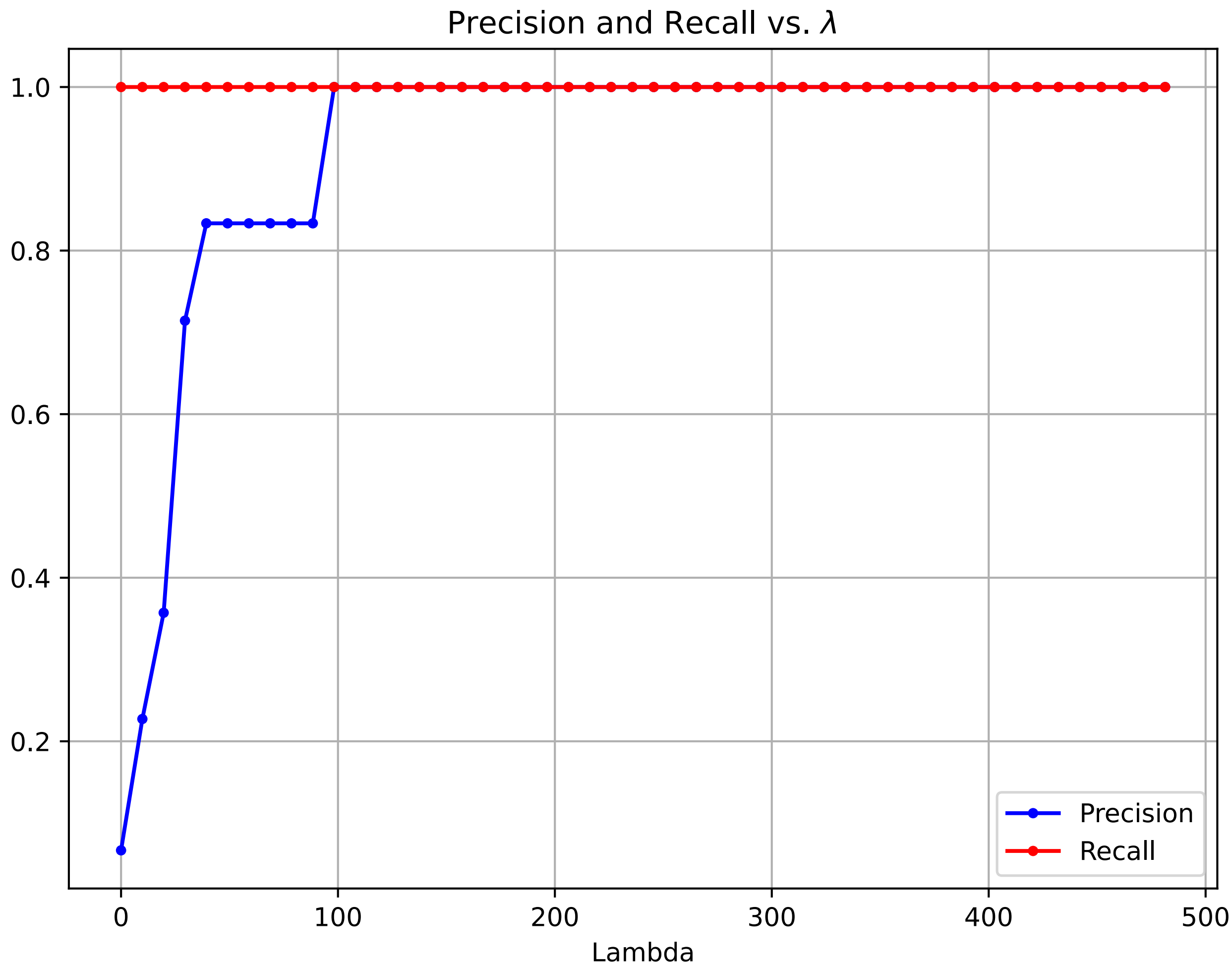
```python
def EvaluatePath(X, y, W, W0, w_true, w0_true, Lmda, filename='temp.png'):

    l_lmda = np.size(Lmda)

    RMSE = np.empty((1, l_lmda))
    Sparsity = np.empty((1, l_lmda))
    Precision = np.empty((1, l_lmda))
    Recall = np.empty((1, l_lmda))

    for i in range(np.size(Lmda)):
        RMSE[:,i], Sparsity[:, i], Precision[:,i], Recall[:,i] = Evaluate(X, y, W[:, i]

    # Generate a 2D plot of precision + recall vs. lmbda
    fig, ax = plt.subplots(figsize=(8, 6))
    plt.plot(Lmda, Precision.T, ls = '-', marker = '.', c = 'blue', label = 'Precision'
    plt.plot(Lmda, Recall.T, ls = '-', marker = '.', c='red', label = 'Recall')
    plt.grid()
    plt.legend()
    plt.xlabel('Lambda')
    plt.ylabel('')
    plt.title('Precision and Recall vs. $\lambda$')
    fig.show()
    plt.savefig(filename) # If saving a file

    return (RMSE, Sparsity, Precision, Recall)
```

In [29]:

```python
# Problem 2.3
# TODO: 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, 'Problem_3_c_1.png')
RMSE, Sparsity, Precision, Recall = EvaluatePath(X, y, W, W0, w_true, w0_true, Lmda, 'P
```

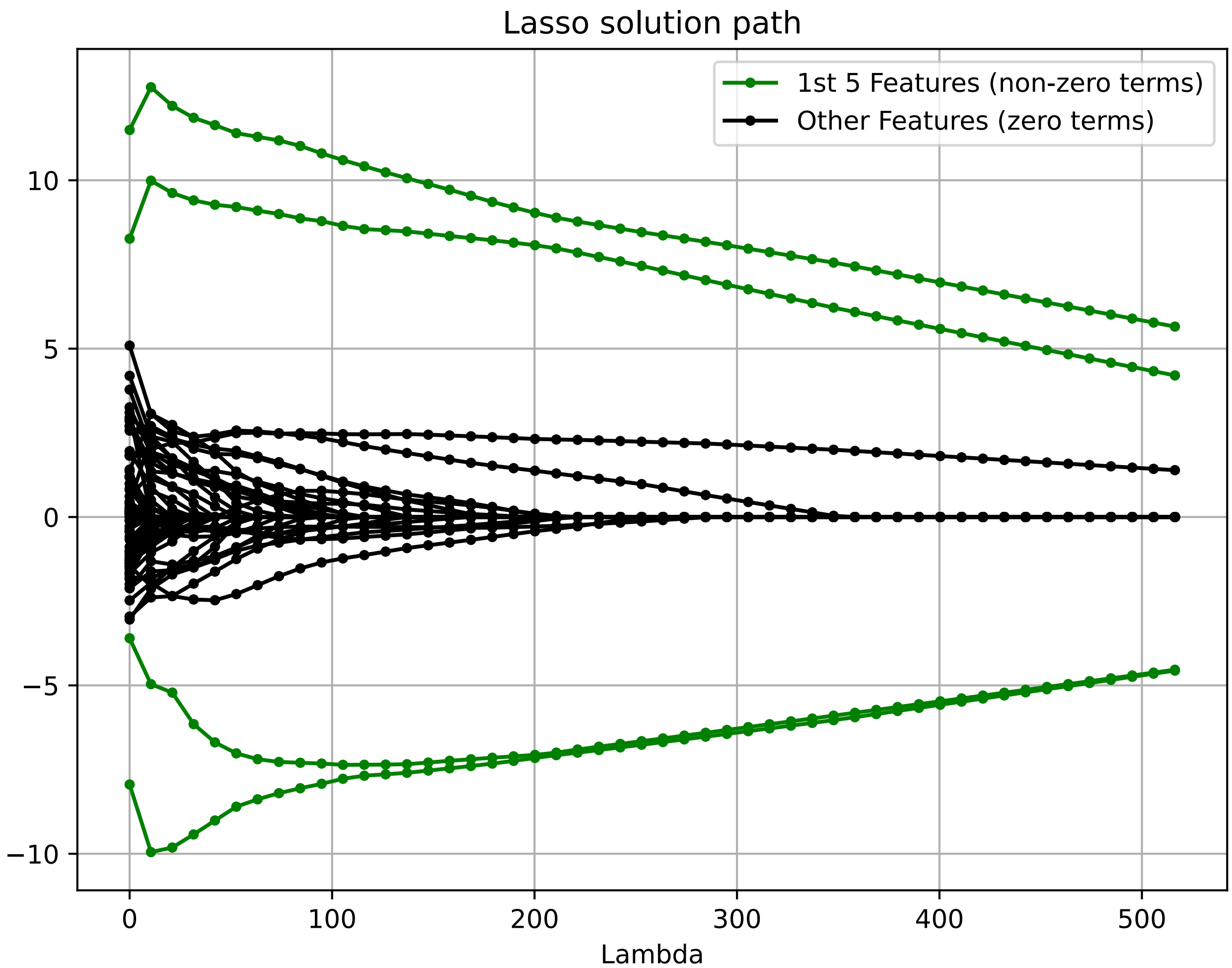


Lasso solution path
1st 5 Features (non-zero terms)
Other Features (zero terms)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
100
200
300
400
500
Lambda



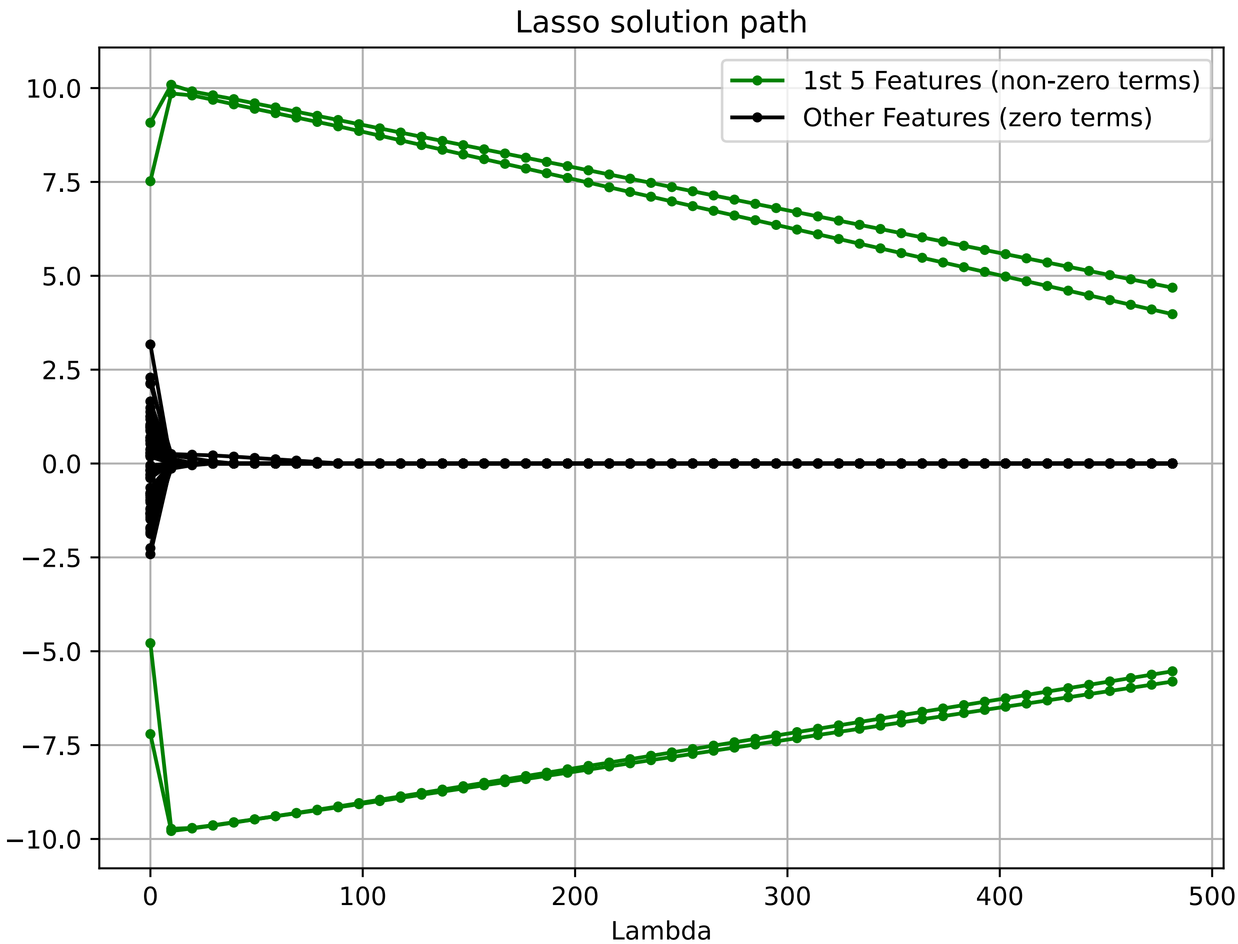
In [30]:

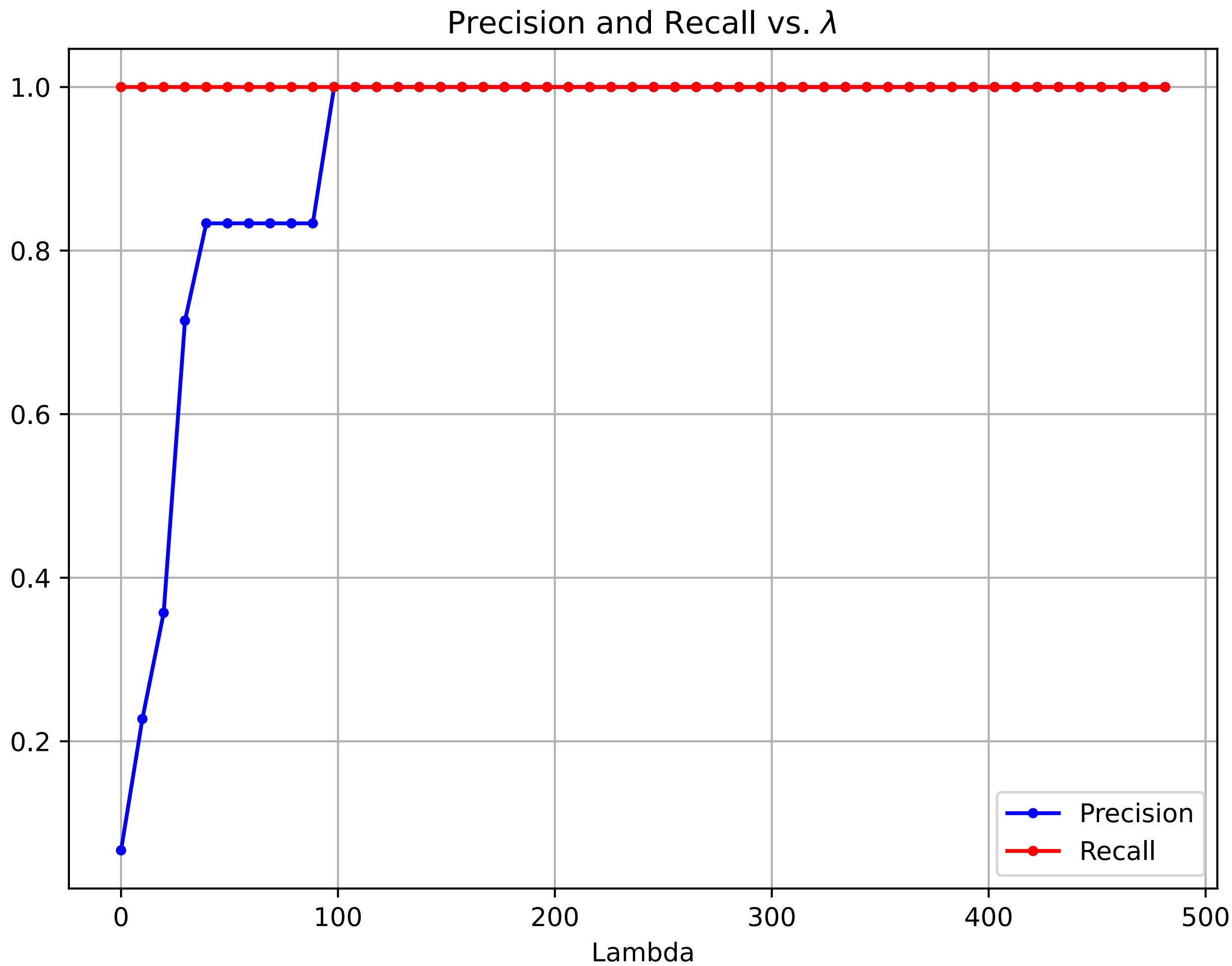
```
# Problem 2.3
# TODO: try a larger std sigma = 10.0
X, y, w_true, w0_true = DataGenerator(n=50, d=75, k=5, sigma=10.0)

W, W0, Lmda = LassoPath(X, y, 'Problem_3_c_3.png')
```



In [31]:

```
# Problem 2.4
# TODO: try another 5 different choices of (n,d)
# draw lasso solution path and precision/recall vs. lambda curves, use them to estimate
n = 50
m = 75

X, y, w_true, w0_true = DataGenerator(n=n, d=m, k=5, sigma=1.0)
W, W0, Lmda = LassoPath(X, y)
RMSE, Sparsity, Precision, Recall = EvaluatePath(X, y, W, W0, w_true, w0_true, Lmda)
```


Lasso solution path
1st 5 Features (non-zero terms)
Other Features (zero terms)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
100
200
300
400
500
Lambda



In [32]:

```
# Problem 2.5: predict reviews' star on Yelp
# data parser reading yelp data
def DataParser(Xfile, yfile, nfile, train_size = 4000, valid_size = 1000):

    # read X, y, feature names from file
    fName = open(nfile).read().splitlines()
    y = np.loadtxt(yfile, dtype=np.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)
```

In [33]:

```
def lasso_sparse(X, y, lmda = 10.0, epsilon = 1.0e-2, max_iter = 100, draw_curve = Fals
    #print(X.shape, y.shape)
```

```
    # Initialize the weights using provided function
    w, w0 = Initialw(X, y)

    n, m = X.shape
    #print(n, m, y.shape, w.shape, w0.shape)

    X = sparse.csc_matrix(X)
    y = sparse.csc_matrix(y).transpose()
    w = sparse.csc_matrix(w).transpose()

    #print(X.shape, y.shape, w.shape, w0.shape)

    w0_temp = sparse.csc_matrix(np.ones(shape=(n,1)) * w0)
    # Initialize our iteration, max change, and objective function
    iter = 0
    #print(X.shape, w.shape, w0_temp.shape, y.shape)
    F = (1/2) * np.sum(X.dot(w) + w0_temp - y) + lmda * abs(w).sum()

    #print('--Start of while loop')

    while True:
        # Update iterator for given loop and reset our max_change
        iter += 1
        #print('Iteration: ', iter)
        max_change = 0

        # Comment out when running real-time
        start_time = datetime.now()
        #print('--Start of for loop')

        for k in range(m):
            start_loop_t = time.time()

            # Solve for r_k
            #X_less_k = X
            #X_less_k[:, k] = 0

            w_less_k = w
            w_less_k[k, 0] = 0

            #print(type(X_less_k), type(w_less_k), X_less_k.shape, w_less_k.shape)

            #X_less_k = np.delete(X.toarray(), [k], axis=1)
            #w_less_k = np.delete(w.toarray(), [k])
            r_k = y - X.dot(w_less_k)

            #print('time to solve for r_k: ', (time.time() - start_loop_t) * 1000)

            # Solve for a_k and c_k
            X_k = X[:, k]
            a_k = 2 * X_k.power(2).sum()
            c_k = 2 * X_k.multiply(r_k).sum()

            #print('time to solve for a_k/c_k: ', (time.time() - start_loop_t) * 1000)

            # Calculate new w_k, cross-compare new weight to old weight to determine if
            w_k = np.sign(c_k) * np.maximum(0, np.absolute(c_k) - lmda) / a_k
            #w_delta[k] = np.absolute(w_k - w[k].toarray())

            if np.absolute(w_k - w[k].toarray()) > max_change:
```

```
                max_change = np.absolute(w_k - w[k].toarray())

            #print('time to solve for w_k + abs: ', (time.time() - start_loop_t) * 1000

            w[k] = w_k
            #print(w_k)

        # Calculate our new w0
        w0 =  np.mean(y) - np.mean(X, axis=0) @ w

        # Calculate our new objective value F
        F_new = (1/2) * np.sum(X.dot(w) + w0_temp - y) + lmda * abs(w).sum()
        F = np.append(F, F_new)

        # After updating our weights, check against exit conditions: (1) number of step
        if iter >= max_iter or max_change <= epsilon:
            break

    return (np.squeeze(w.toarray()), w0)
```

In [34]:

```
# Problem 2.3
# TODO: compute a lasso solution path, draw the path(s) in a 2D plot
def LassoPath_sparse(X, y, filename='temp.png', lmda_start = 0):
    lmda_max = np.amax((y - np.average(y)).T @ X)
    n, m = X.shape

    l_range = 20
    Lmda = np.linspace(lmda_start*lmda_max, lmda_max, num=l_range)
    W = np.empty((m, l_range))
    W0 = np.empty((1, l_range))
    #print(Lmda)

    # Calculate our weights for each lambda and save to our value for W
    start_loop_t = time.time()
    for i in range(l_range):
        print(i, Lmda[i], " Iteration start: ", time.time() - start_loop_t)
        w_lasso, w0_lasso = lasso_sparse(X, y, lmda = Lmda[i], epsilon = 1.0e-2, draw_c
        W[:, i] = w_lasso
        W0[:, i] = w0_lasso
        print(Lmda[i], " Iteration stop: ", time.time() - start_loop_t)

    # Generate a 2D plot of our lasso solution path
    fig, ax = plt.subplots(figsize=(8, 6))
    plt.plot(Lmda, W.T, ls = '-', marker = '.', c = 'green')

    # Remove the duplicate labels from our labels to create a succient legend
    handles, labels = plt.gca().get_legend_handles_labels()
    newLabels, newHandles = [], []
    for handle, label in zip(handles, labels):
        if label not in newLabels:
            newLabels.append(label)
            newHandles.append(handle)

    plt.legend(newHandles, newLabels)

    plt.grid()
    #plt.legend()
    plt.xlabel('Lambda')
    plt.ylabel('')
```

```python
    plt.title('Lasso solution path')
    fig.show()
    plt.savefig(filename) # If saving a file

    return (W, W0, Lmda)
```

In [35]:

```python
def EvaluatePath_sparse(X, y, W, W0, w_true, w0_true, Lmda):

    l_lmda = np.size(Lmda)

    RMSE = np.empty((1, l_lmda))
    Sparsity = np.empty((1, l_lmda))
    Precision = np.empty((1, l_lmda))
    Recall = np.empty((1, l_lmda))

    for i in range(np.size(Lmda)):
        RMSE[:,i], Sparsity[:, i], Precision[:,i], Recall[:,i] = Evaluate(X, y, W[:, i]

    return (RMSE)
```

In [36]:

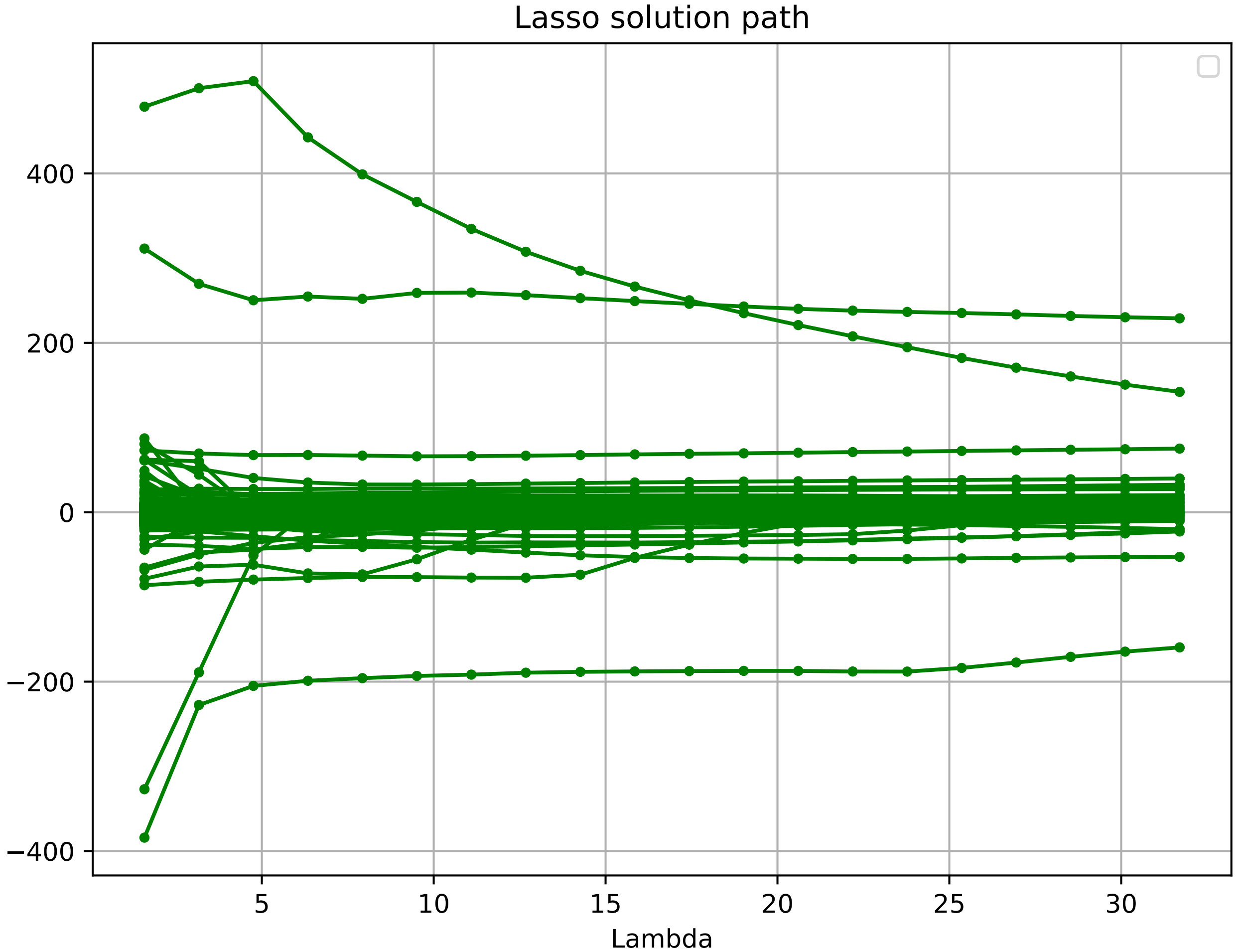
```python
# Problem 2.5: predict reviews' star on Yelp
# TODO: evaluation funtion that computes the lasso path, evaluates the result, and draw
def Validation(X_train, y_train, X_valid, y_valid):
    # Test lasso sparse works
    #w_lasso, w0_lasso = lasso_sparse(X_train, y_train, lmda = 10, epsilon = 1.0e-2, dr

    # Run lasso path
    W, W0, Lmda = LassoPath_sparse(X_train, y_train, 'Problem_3_e_1.png',  lmda_start=0

    RMSE_t = EvaluatePath_sparse(X_train, y_train, W, W0, W, W0, Lmda)

    RMSE_v = EvaluatePath_sparse(X_valid, y_valid, W, W0, W, W0, Lmda)

    # Generate a 2D plot of precision + recall vs. lmbda
    fig, ax = plt.subplots(figsize=(8, 6))
    plt.plot(Lmda, RMSE_t.T, ls = '-', marker = '.', c = 'blue', label = 'RMSE Training
    plt.plot(Lmda, RMSE_v.T, ls = '-', marker = '.', c='red', label = 'RMSE Validation
    plt.grid()
    plt.legend()
    plt.xlabel('Lambda')
    plt.ylabel('')
    plt.title('RMSE of Training and Validation Set vs. $\lambda$')
    fig.show()
    plt.savefig('Problem_3_e_2.png') # If saving a file

    lmda_best_index = np.argmin(RMSE_v)

    lmda_best = Lmda[lmda_best_index]
    w_lasso = W[:, lmda_best_index]
    w0_lasso = W0[:, lmda_best_index]
    return (w_lasso, w0_lasso, lmda_best)
```

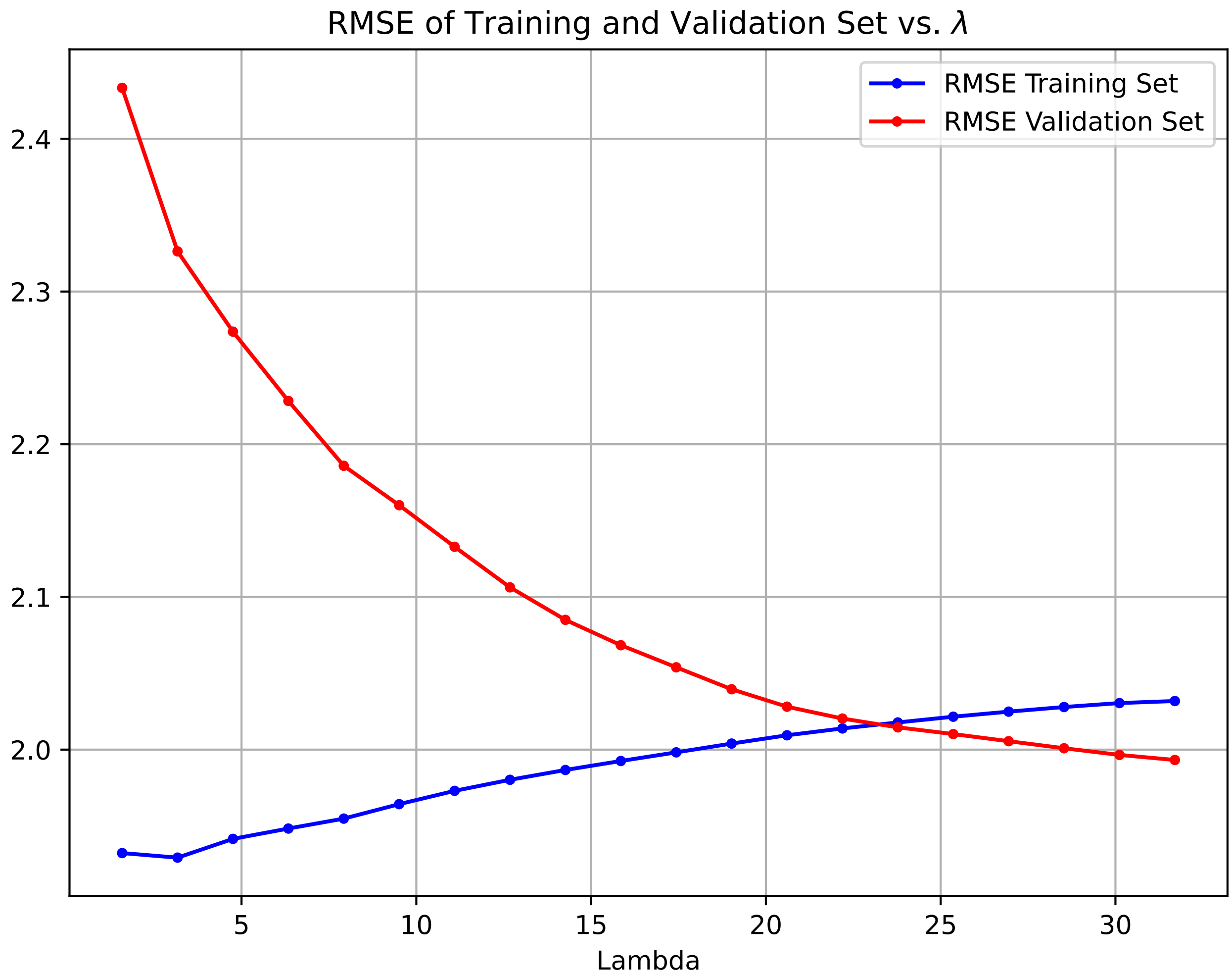
In [37]:

```python
# Problem 2.5: 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 ('.
from scipy.sparse.linalg import lsqr
```

```python
X_train, y_train, X_valid, y_valid, X_test, y_test, fName = DataParser('./data/star_dat

#print(X_train.shape, y_train.shape, X_valid.shape, y_valid.shape, X_test.shape, y_test

# evaluation
w_lasso, w0_lasso, lmda_best = Validation(X_train, y_train, X_valid, y_valid)

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

```
 6643.44523692131
5 9.509979830006111  Iteration start:  6643.44523692131
9.509979830006111  Iteration stop:  7980.131078958511
6 11.094976468340462  Iteration start:  7980.131078958511
11.094976468340462  Iteration stop:  9305.450192451477
7 12.679973106674815  Iteration start:  9305.450192451477
12.679973106674815  Iteration stop:  10621.39349246025
8 14.264969745009168  Iteration start:  10621.39349246025
14.264969745009168  Iteration stop:  11935.407716751099
9 15.84996638334352  Iteration start:  11935.407716751099
15.84996638334352  Iteration stop:  13247.58761548996
10 17.43496302167787  Iteration start:  13247.58761548996
17.43496302167787  Iteration stop:  14561.37648510933
11 19.019959660012223  Iteration start:  14561.37648510933
19.019959660012223  Iteration stop:  15907.903224468231
12 20.604956298346575  Iteration start:  15907.903224468231
20.604956298346575  Iteration stop:  17287.754866600037
13 22.18995293668093  Iteration start:  17287.754866600037
22.18995293668093  Iteration stop:  18663.458851337433
14 23.774949575015278  Iteration start:  18663.458851337433
23.774949575015278  Iteration stop:  20032.85348701477
15 25.35994621334963  Iteration start:  20032.85348701477
25.35994621334963  Iteration stop:  21366.83390212059
16 26.944942851683983  Iteration start:  21366.83390212059
26.944942851683983  Iteration stop:  22688.77535367012
17 28.529939490018336  Iteration start:  22688.77535367012
28.529939490018336  Iteration stop:  24013.767376184464
18 30.11493612835269  Iteration start:  24013.767376184464
30.11493612835269  Iteration stop:  25317.426379680634
19 31.699932766687038  Iteration start:  25317.426379680634
31.699932766687038  Iteration stop:  26625.879186868668
Lasso select features:
and 228.92007242281383
were soaked in -159.55591374549522
the 142.08381416889432
great 75.10155839764127
set -52.662879026941425
best 39.82050044662048
love 32.49102678604067
amazing 30.24397828436301
delicious 27.33324807664297
of a -22.706295466396767
```


Lasso solution path
400
200
0
−200
−400
5
10
15
20
25
30
Lambda


RMSE of Training and Validation Set vs. λ
RMSE Training Set
RMSE Validation Set
2.4
2.3
2.2
2.1
2.0
5
10
15
20
25
30
Lambda

In [ ]: