CSE512 Spring 2021 - Machine Learning - Homework 2

Your Name: Sagar Onkar Toshniwal

Solar ID: 113260061

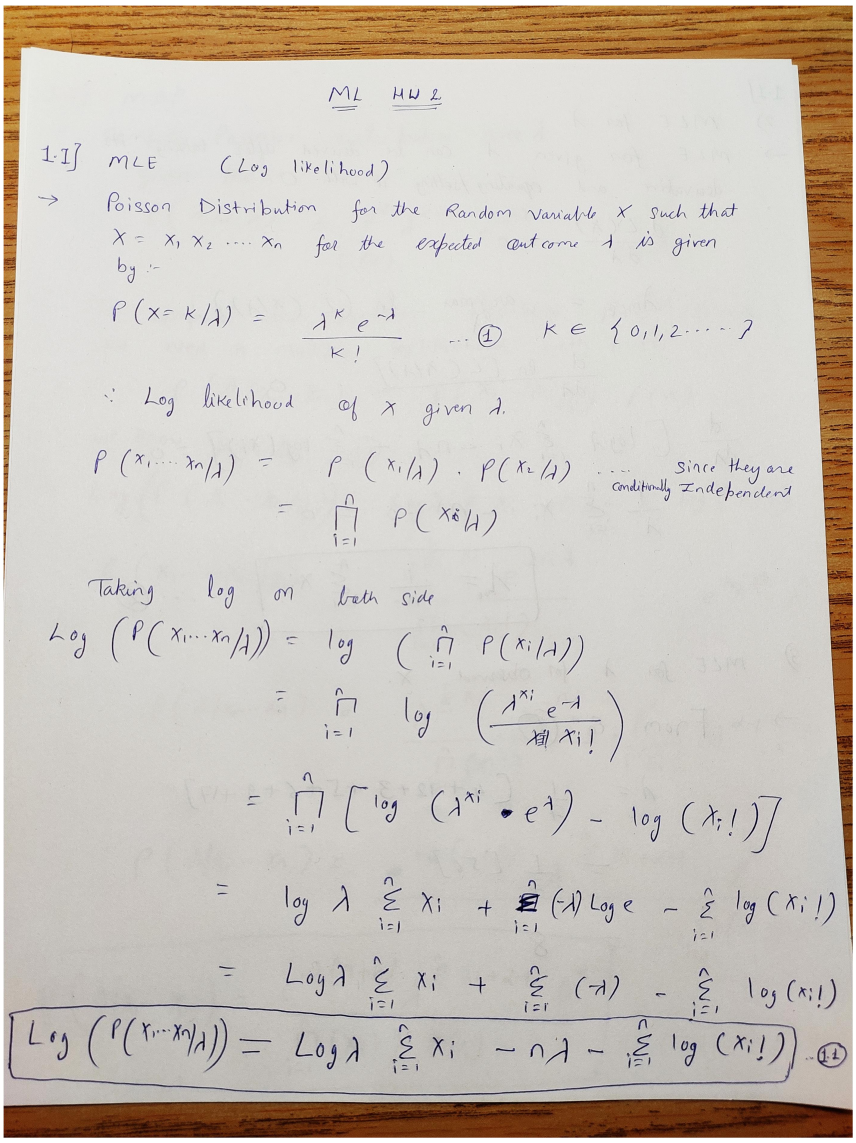
NetID email address: sagaronkar.toshniwal@stonybrook.edu

Names of people whom you discussed the homework with: None but I have taken reference from medium, towardsdatascience, Google, Stackoverflow, etc.

1) Question 1 – Parameter estimation (45 points)

1.1 Question 1.1 – MLE (15 points)

1. (5 points) Give the log-likelihood function of X given _.
2. (5 points) Compute the MLE for _ in the general case.
3. (5 point) Compute the MLE for _ using the observed X.



ML HW 2

1.1] MLE (Log likelihood)

→ Poisson Distribution for the Random Variable X such that $X = x_1 x_2 \cdots x_n$ for the expected outcome λ is given by :-

$$P(X = k/\lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \quad \cdots (1) \quad k \in \{0, 1, 2 \cdots\}$$

∴ Log likelihood of X given λ.

$$P(x_1 \cdots x_n/\lambda) = P(x_1/\lambda) \cdot P(x_2/\lambda) \cdots$$ since they are conditionally Independent

$$= \prod_{i=1}^{n} P(x_i/\lambda)$$

Taking log on both side

$$\text{Log}\left(P(x_1 \cdots x_n/\lambda)\right) = \log\left(\prod_{i=1}^{n} P(x_i/\lambda)\right)$$

$$= \prod_{i=1}^{n} \log\left(\frac{\lambda^{x_i} e^{-\lambda}}{x_i!}\right)$$

$$= \prod_{i=1}^{n} \left[\log\left(\lambda^{x_i} \cdot e^{-\lambda}\right) - \log(x_i!)\right]$$

$$= \log\lambda \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} (-\lambda) \log e - \sum_{i=1}^{n} \log(x_i!)$$

$$= \text{Log}\,\lambda \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} (-\lambda) - \sum_{i=1}^{n} \log(x_i!)$$

$$\boxed{\text{Log}\left(P(x_1 \cdots x_n/\lambda)\right) = \text{Log}\,\lambda \sum_{i=1}^{n} x_i - n\lambda - \sum_{i=1}^{n} \log(x_i!)} \quad \cdots (1)$$

1.1]

2) MLE for λ

→ MLE for given λ can be derived after taking its derivative and equating/setting it with 0.

$$\frac{d\,L(\lambda)}{d\lambda} = 0$$

$$\because \lambda_{MLE} = \arg\max_{\lambda} \ln\left(L\left(x/\lambda\right)\right)$$

$$\therefore \frac{d \ln\left[L(x/\lambda)\right]}{d\lambda} = 0$$

$$\frac{d}{d\lambda}\left[\log\lambda \sum_{i=1}^{n} x_i - n\lambda - \sum_{i=1}^{n} \log(x_i)\right] = 0$$

$$\frac{1}{\lambda}\sum_{i=1}^{n} x_i - n - 0 = 0$$

$$\boxed{\lambda_{MLE} = \frac{1}{n}\sum_{i=1}^{n} x_i} \quad \ldots (2)$$

3) MLE for λ for observed x.

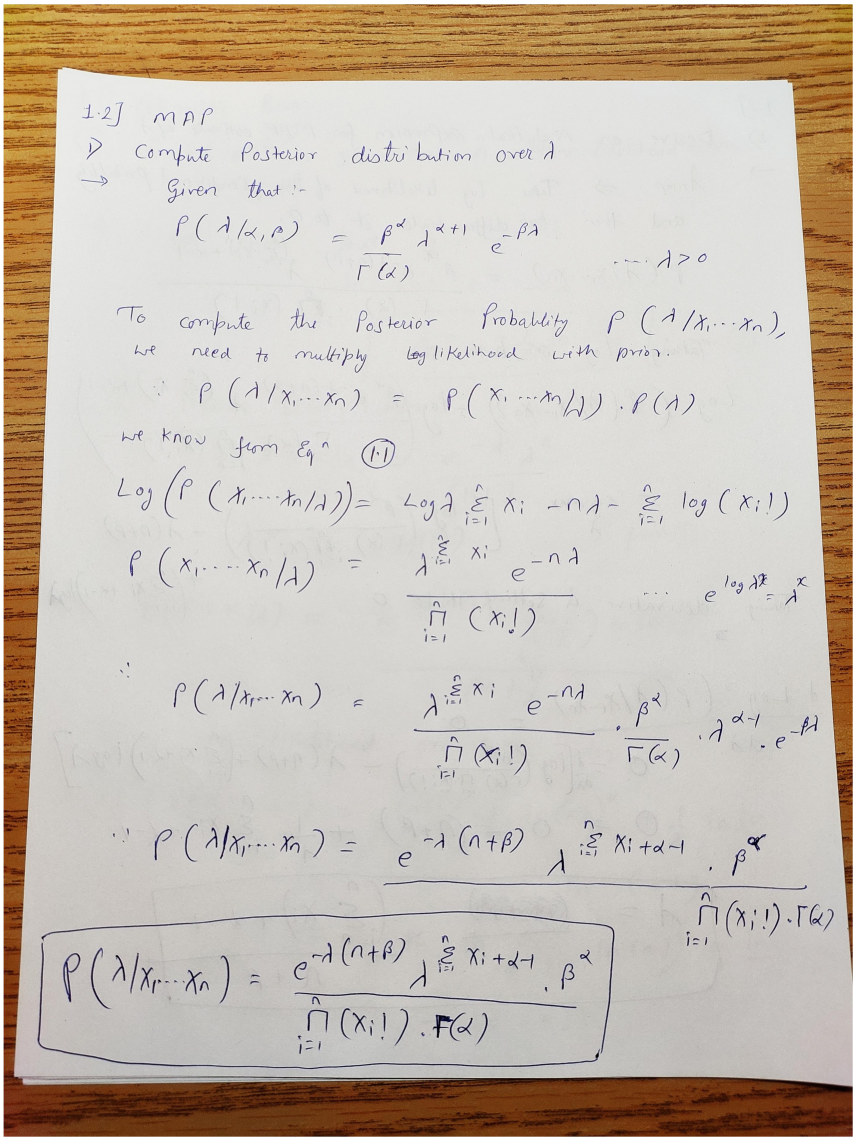
→ From Eqn (2)

$$\lambda = \frac{1}{7}\left[4+12+3+5+6+9+17\right]$$

$$= \frac{1}{7}\left[56\right]$$

$$= 8$$

1.2 Question 1.2 – MAP (15 points)



1.2] MAP

▷ Compute Posterior distribution over λ

→ Given that :-

$$P(\lambda/\alpha,\beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} \lambda^{\alpha+1} e^{-\beta\lambda} \quad \cdots \lambda > 0$$

To compute the Posterior Probability $P(\lambda/x_1 \cdots x_n)$, we need to multiply Log likelihood with prior.

$$\therefore P(\lambda/x_1 \cdots x_n) = P(x_1 \cdots x_n/\lambda) \cdot P(\lambda)$$

we know from Eq^n (1.1)

$$\text{Log}\left(P(x_1 \cdots x_n/\lambda)\right) = \text{Log}\,\lambda \sum_{i=1}^{n} x_i - n\lambda - \sum_{i=1}^{n} \log(x_i!)$$

$$P(x_1 \cdots x_n/\lambda) = \frac{\lambda^{\sum_{i=1}^{n} x_i} e^{-n\lambda}}{\prod_{i=1}^{n} (x_i!)} \quad \cdots e^{\log \lambda^x} = \lambda^x$$

$$\therefore P(\lambda/x_1 \cdots x_n) = \frac{\lambda^{\sum_{i=1}^{n} x_i} e^{-n\lambda}}{\prod_{i=1}^{n} (x_i!)} \cdot \frac{\beta^{\alpha}}{\Gamma(\alpha)} \cdot \lambda^{\alpha-1} \cdot e^{-\beta\lambda}$$

$$\therefore P(\lambda/x_1 \cdots x_n) = \frac{e^{-\lambda(n+\beta)} \lambda^{\sum_{i=1}^{n} x_i + \alpha - 1} \cdot \beta^{\alpha}}{\prod_{i=1}^{n} (x_i!) \cdot \Gamma(\alpha)}$$

$$\boxed{P(\lambda/x_1 \cdots x_n) = \frac{e^{-\lambda(n+\beta)} \lambda^{\sum_{i=1}^{n} x_i + \alpha - 1} \cdot \beta^{\alpha}}{\prod_{i=1}^{n} (x_i!) \cdot \Gamma(\alpha)}}$$

1.2]

2) Derive an Analytical expression for MAP estimate of λ

→ λ_{map} ⇒ Take log likelihood of the Conditional probability and then differentiate it to 0.

$$P(\lambda/x_1 \cdots x_n) = \frac{\beta^{\alpha} e^{-\lambda(n+\beta)} \lambda^{\left(\sum_{i=1}^{n} x_i\right)+\alpha-1}}{\Gamma(\alpha) \cdot \prod_{i=1}^{n} (x_i!)}$$

Taking Log on both side

$$\text{Log}\left(P(\lambda/x_1 \cdots x_n)\right) = \log\left(\frac{\beta^{\alpha} e^{-\lambda(n+\beta)} \lambda^{\left(\sum_{i=1}^{n} x_i\right)+\alpha-1}}{\Gamma(\alpha) \prod_{i=1}^{n} (x_i!)}\right)$$

$$= \arg\max_{\lambda} \left[\log\left(\frac{\beta^{\alpha}}{\Gamma(\alpha) \prod_{i=1}^{n} (x_i!)}\right) - \lambda(n+\beta) + \left(\sum_{i=1}^{n} x_i + \alpha - 1\right) \log\lambda \right]$$

Taking derivative & setting it to 0

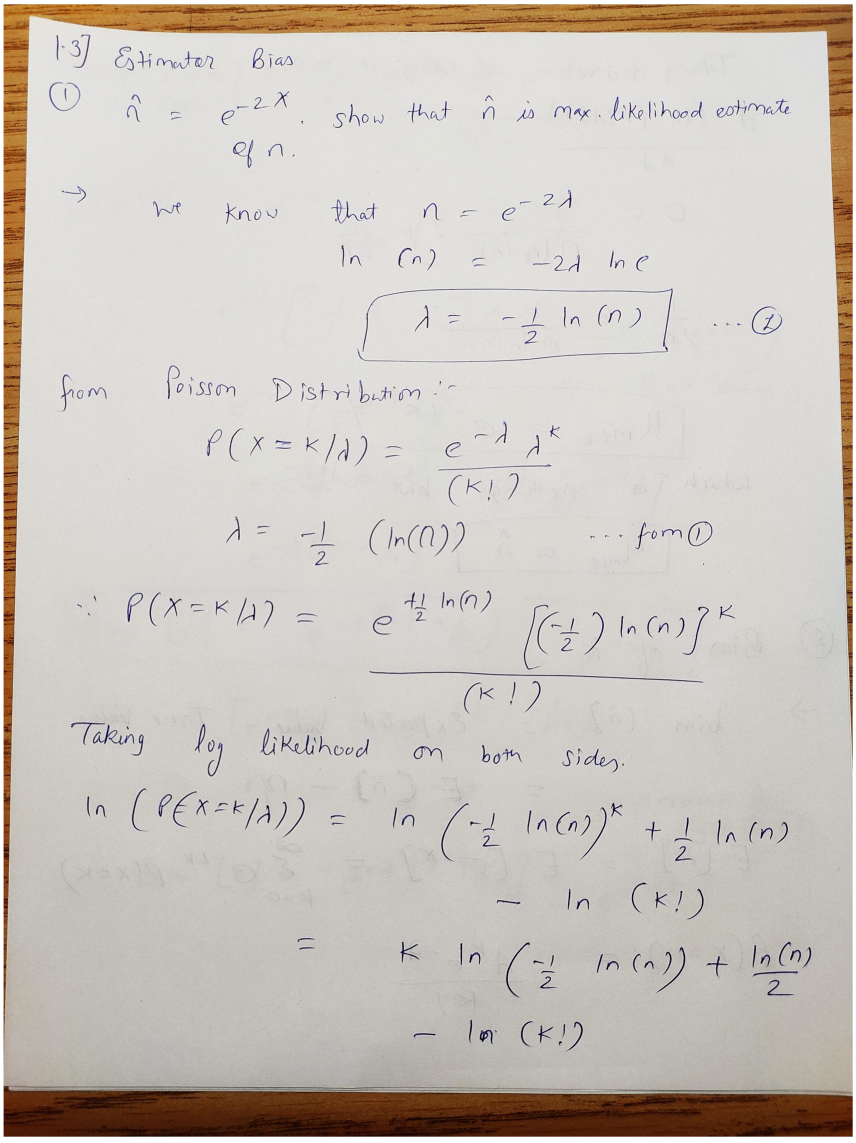
$$\frac{\text{Log}\left(P(\lambda/x_1 \cdots x_n)\right)}{d\lambda} = 0$$

$$0 = \frac{d}{d\lambda}\left[\log\left(\frac{\beta^{\alpha}}{\Gamma(\alpha) \prod_{i=1}^{n} (x_i!)}\right) - \lambda(n+\beta) + \left(\sum_{i=1}^{n} x_i + \alpha - 1\right) \log\lambda\right]$$

$$0 = 0 - (n+\beta) + \frac{1}{\lambda} \sum_{i=1}^{n} x_i + \alpha - 1$$

$$\boxed{\lambda = \frac{\left(\sum_{i=n}^{n} x_i\right) + \alpha - 1}{n+\beta}}$$

1.3 Question 1.3 – Estimator Bias (15 points)



1.3] Estimator Bias

① $\hat{n} = e^{-2\lambda}$. show that $\hat{n}$ is max. likelihood estimate of n.

→ we know that $n = e^{-2\lambda}$

$$\ln(n) = -2\lambda \ln e$$

$$\boxed{\lambda = -\frac{1}{2}\ln(n)} \quad \ldots (1)$$

from Poisson Distribution :-

$$P(X=k/\lambda) = \frac{e^{-\lambda}\lambda^k}{(k!)}$$

$$\lambda = -\frac{1}{2}(\ln(n)) \quad \ldots \text{from } (1)$$

$$\therefore P(X=k/\lambda) = \frac{e^{+\frac{1}{2}\ln(n)}\left[\left(-\frac{1}{2}\right)\ln(n)\right]^k}{(k!)}$$

Taking log likelihood on both sides.

$$\ln\left(P(X=k/\lambda)\right) = \ln\left(-\frac{1}{2}\ln(n)\right)^k + \frac{1}{2}\ln(n) - \ln(k!)$$

$$= k\ln\left(-\frac{1}{2}\ln(n)\right) + \frac{\ln(n)}{2} - \ln(k!)$$

Taking derivative & Setting it to 0.

$$\frac{d\,(\ln(P(\lambda)))}{d\lambda} = 0$$

$$0 = \frac{k}{n \ln(n)} + \frac{1}{2n}$$

$$\frac{1}{2n} = \frac{k}{n \ln(n)}$$

$$\boxed{n_{MLE} = e^{-2k}}$$

which is nothing but

$$\boxed{n_{MLE} \approx \hat{n}}$$

(2) Bias of $\hat{n}$

→ bias $(\hat{n})$ = Expected Value − True Value

$$= E[\hat{n}] - n$$

$$E[\hat{n}] = E[e^{-2x}] = \sum_{k=0}^{\infty} e^{-2k} P(x=k)$$

$$P(x=k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Substitute $P(X=K) = \dfrac{\sum_{K=0}^{\infty} e^{-2K}\,\lambda^K\,e^{-\lambda}}{\sum_{K=0}^{\infty} K!}$ in $E[e^{-2X}]$

$$E[e^{-2\lambda}] = e^{-\lambda} \sum_{K=0}^{\infty} \left[\frac{e^{-2K}\,\lambda^K}{K!}\right]$$

$$= e^{-\lambda}\left[\frac{1}{1} + \frac{e^{-2}\lambda}{1} + \frac{e^{-4}\lambda^2}{2} + \cdots\right] \quad \cdots \text{Taylor series}$$

$$= e^{-\lambda}\left[e^{e^{-2}\lambda}\right]$$

$$= e^{-\lambda}\left[e^{e^{-2}\lambda}\right]$$

$$= e^{-\lambda}\left[-1 + \frac{1}{e^2}\right]$$

$$\boxed{\text{Bias}[\hat{n}] = e^{-\lambda}\left[-1 + \frac{1}{e^2}\right] - e^{-2\lambda}}$$

as $E[e^{-2\lambda}] = e^{-\lambda}\left[-1 + \frac{1}{e^2}\right]$

$n = e^{-2\lambda}$ ... proven from last (1.3.1)

$\therefore$ Bias$[\hat{n}] = e^{-\lambda}\left[-1 + \frac{1}{e^2}\right] - e^{-2\lambda}$

③ $(-1)^x$ is unbiased & prove that it is the only unbiased of n. & briefly explain why it is unbias.

→ Given, $\hat{n} = (-1)^x$

$$E[\hat{n}] = E[(-1)^x] = \sum_{k=0}^{\infty} \frac{(-1)^k \lambda^k e^{-\lambda}}{k!}$$

$$= e^{-\lambda} \cdot e^{-\lambda} = e^{-2\lambda}$$

$$\text{Bias} = E[\hat{n}] - n$$

$$= e^{-2\lambda} - e^{-2\lambda} = 0$$

When Bias is 0, then it is unbiased.
Hence the estimator is unbiased.

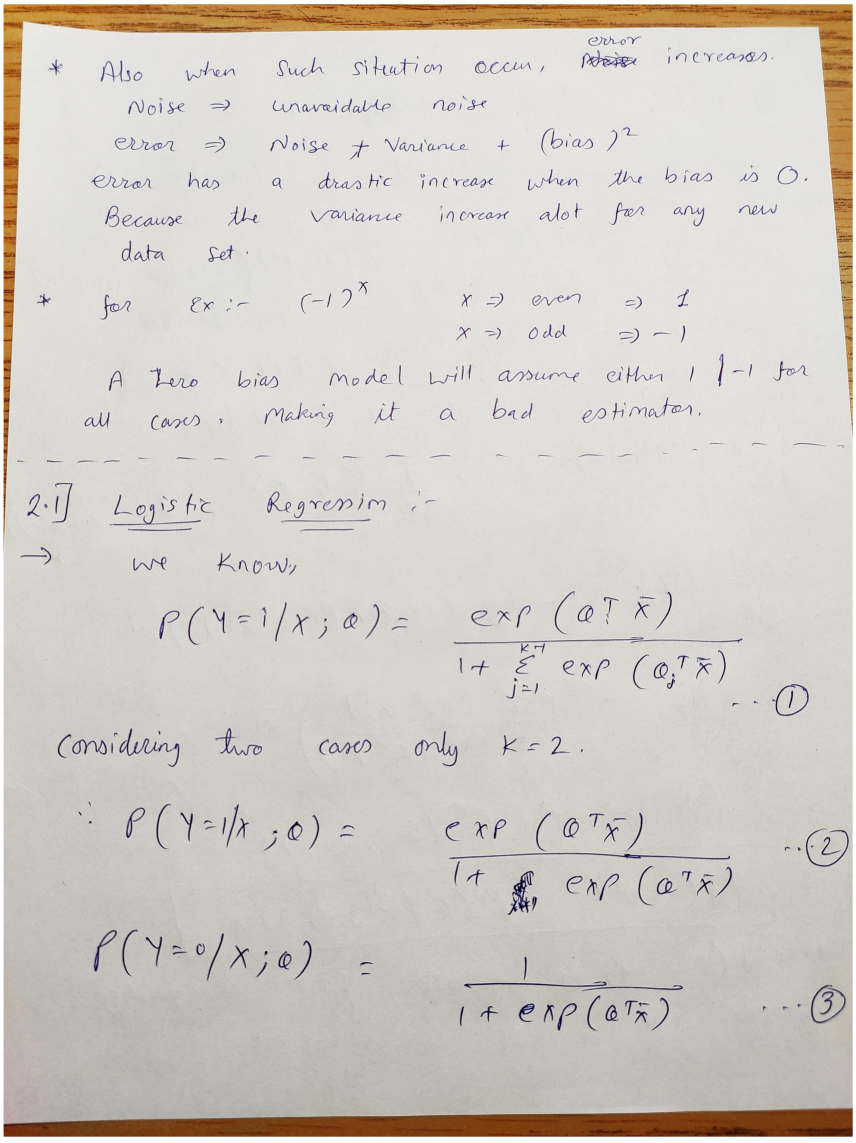
~~To prove only $(-1)^x$ is $(-1)^x$ is the only unbiased~~

Reason why unbiased Estimator is a bad estimator to use :-

* It is a proven fact that Zero bias approach has a poor generalisability to new situation (in test data) and also assumes that the model has a precise knowledge of a true state of world (which definetely isn't the case in Machine Learning)

This results in wrong estimation of the expected value.

2.1 Question 2.1 – Logistic Regression Derivation (10 points)



* Also when such situation occur, error increases.
  Noise ⇒ unavoidable noise
  error ⇒ Noise + Variance + (bias)^2
  error has a drastic increase when the bias is 0.
  Because the variance increase alot for any new data set.

* for Ex :- $(-1)^x$ $x \Rightarrow$ even $\Rightarrow 1$
  $x \Rightarrow$ odd $\Rightarrow -1$
  A zero bias model will assume either 1 / -1 for all cases, making it a bad estimator.

2.1] Logistic Regression :-

→ we know,

$$P(Y=i/x;\theta) = \frac{\exp(\theta_i^T \bar{x})}{1+\sum_{j=1}^{K-1} \exp(\theta_j^T \bar{x})} \quad \cdots (1)$$

considering two cases only $K=2$.

$$\therefore P(Y=1/x;\theta) = \frac{\exp(\theta^T \bar{x})}{1+\exp(\theta^T \bar{x})} \quad \cdots (2)$$

$$P(Y=0/x;\theta) = \frac{1}{1+\exp(\theta^T \bar{x})} \quad \cdots (3)$$

Taking Log likelihood for eq^n (1)

$$\text{Log}\left(P(y^i/x^i;\theta)\right) = y^i \text{Log}\left(P(y=1/\bar{x}^i;\theta)\right) + (1-y^i)\log\left(P(y=0/\bar{x}^i;\theta)\right)$$

$$= y^i \text{Log}\left(\frac{\exp(\theta^T\bar{x})}{1+\exp(\theta^T\bar{x})}\right) + (1-y^i)\left(\text{Log}\left(\frac{1}{1+\exp(\theta^T\bar{x})}\right)\right) \quad \text{... from (2) \& (3)}$$

$$= y^i \theta^T \bar{x}^i - \log\left(1+\exp\left(1+\theta^T\bar{x}^i\right)\right)$$

Taking derivative & setting it on both sides.

$$\frac{d\,\text{Log}\left(P(y\neq x_i;\theta_c)\right)}{d\theta_c} = \frac{d}{d\theta_c}\left(\left(y_c^i\,\theta_c^T - \log\left(1+\exp\left(1+\theta^T\bar{x}^i\right)\right)\right)\bar{x}^i\right)$$

$$= \frac{d}{d\theta_c}\left(\left(y_c^i\,\theta_c^T - \log\left(1+\exp\left(1+\theta_c^T\bar{x}^i\right)\right)\right)\bar{x}^i\right)$$

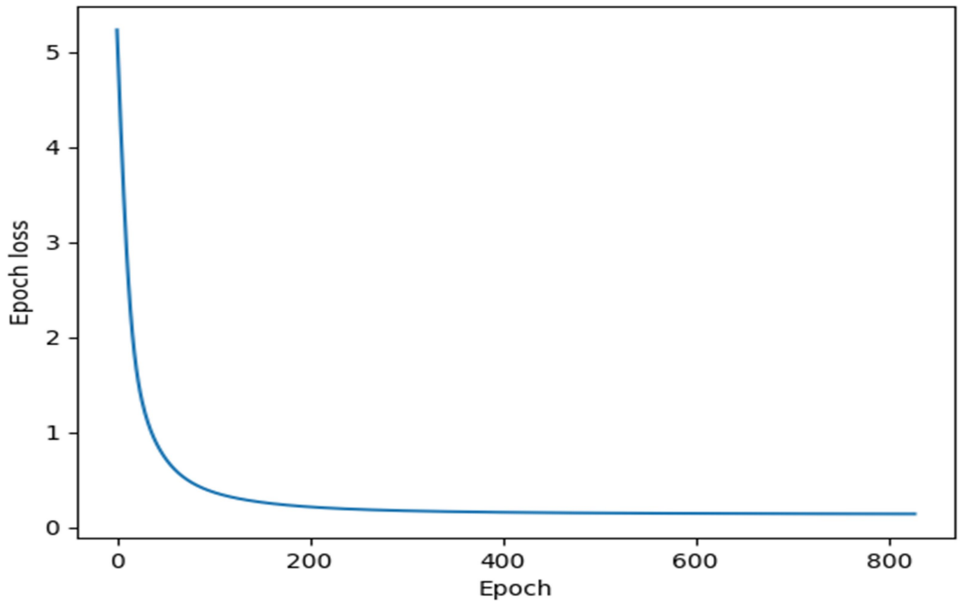
$$= \left(y_i - \frac{\exp(\theta_c^T x^i)}{1+\exp(\theta_c^T\bar{x}^i)}\right)\bar{x}^i$$

$$= \left(y_i - P(y=1/\bar{x};\theta_c)\right)\bar{x}^i$$

$$\boxed{= \left(\delta(c=y^i) - P(c/\bar{x}^i;\theta_c)\right)\bar{x}^i}$$

Where $\delta(c=y^i)$ is indicator funct^n. & equating ground truth.

2.3.1) a) LOSS FUNCTION



Initially the loss is high as the model has considered random weights and its getting updated in the initial 200 epoch. Post 200 epoch the model has learned optimal weights, thus it dosen't update and that's why we see a saturation line

2.3.1) b) Accuracy , Confusion Matrix and Accuracy based on matrix

```
[82]: from sklearn.metrics import confusion_matrix
      conf_mat = confusion_matrix(y_train, y_p)
      conf_mat1 = confusion_matrix(y_test, y_ptest)
      print("Confusion Matrix :")
      print (confusion_matrix(y_test, y_ptest))

      Confusion Matrix :
      [[611  24]
       [ 43  13]]

[79]: from sklearn.metrics import accuracy_score,average_precision_score,precision_recall_curve
      print ("Accuracy on training Data :",accuracy_score(y_train, y_p))
      print ("Accuracy on testing Data :",accuracy_score(y_test, y_ptest))

      Accuracy on training Data : 0.9424746743849494
      Accuracy on testing Data : 0.9030390738060782

[80]: acc_conf = np.trace(conf_mat)/np.sum(conf_mat)
      acc_conf1 = np.trace(conf_mat1)/np.sum(conf_mat1)
      print ()

      print ("Accuracy on training Confusion Matrix :",acc_conf)
      print ("Accuracy on testing Confusion Matrix :",acc_conf1)

      Accuracy on training Confusion Matrix : 0.9424746743849494
      Accuracy on testing Confusion Matrix : 0.9030390738060782
```

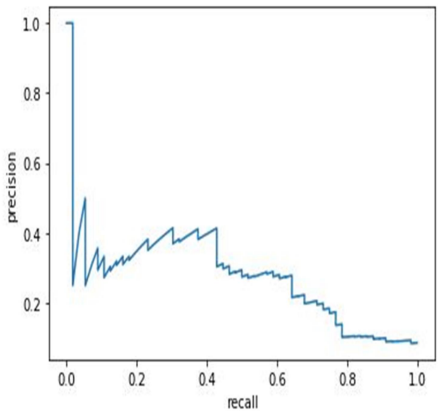
2.3.1) c) Average Precision and Recall

[111]:
```
precision, recall, threshold = precision_recall_curve(y_test, Prob[:,1])
print ("Average Precision Score : ",average_precision_score(y_test, Prob[:,1]))
```

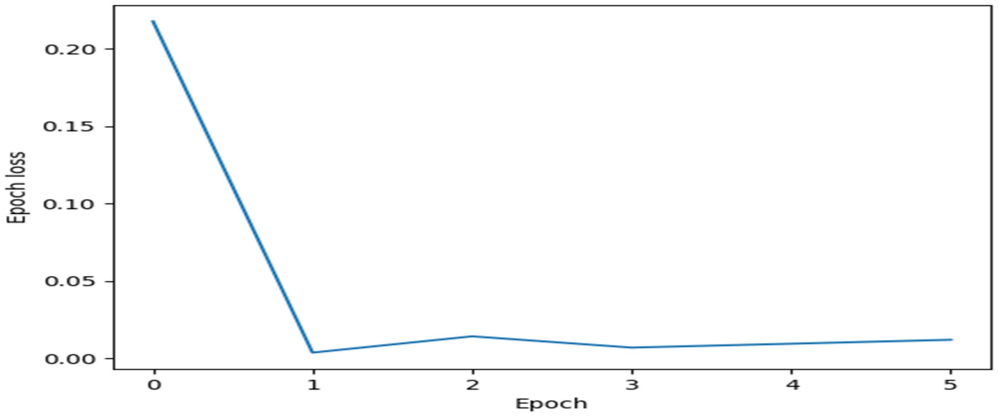
```
Average Precision Score :  0.2825801776059559
```

[110]:
```
plt.plot (recall,precision)
plt.xlabel("recall")
plt.ylabel("precision")
plt.show
```

[110]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



2.3.2) a) Loss Function without Standardization



For non-standardized data, The presence of feature value X in the formula will affect the step size of the gradient descent. The difference in ranges of features will cause different step sizes for each feature. Because of this reason the model learns only for few epoch and then comes out of the loop.

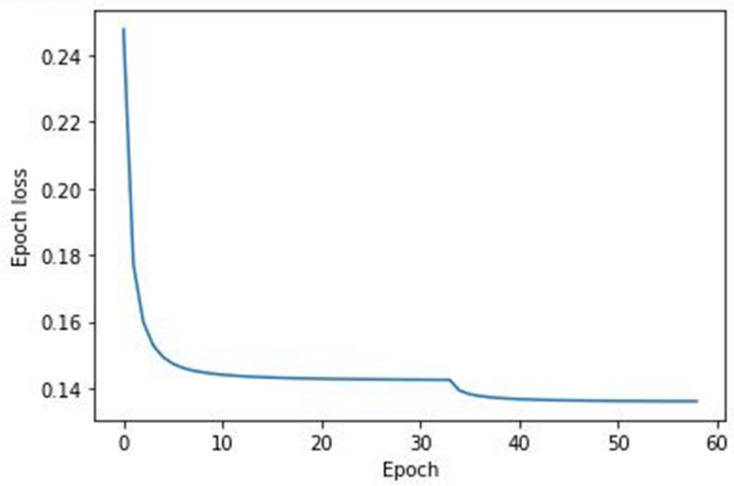
It does not converge, infact it diverges. The number of epoch are very few. In our case just below 10.Since the parameter are not learned well, accuracy is very low. Thus the final performance is very bad as compared to standardized data.

2.3.2) b) Tuning Hyperparameter

- When eta_start increases it converges faster but if it is increased to a lot then it may also diverges. So an optimum value would be between (0.01,0.1). When eta_start decreases it tends to be closer to eta_end and thus it doesn't learn efficiently and may or may not give optimal result.
- When eta_end decreases it converges slow but with optimal value. If eta_end decreases then it dosent provide optimal solution . it approximates early.
- For max_epoch I found 1000 was optimal. Post 1000 it showed the same result with more computational time. When below 1000 accuracy was not as good as it was with 1000 (when other parameters were given in HW2)
- Decreasing batch size take more training time but it converges faster (Also no. of epoch decreases). Increasing batch size will take less training time, more epoch and converge slow.

2.3.2) c) different parameter

- Based on above description(2.3.2.b) and multiple combination, I decreased batch size to 16, increased eta_start to 0.1, decreased epoch to 250. I found that the accuracy increased, precision increased, and computationally it worked faster and converge faster with minimum loss.



[121]:
```
from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(y_train, y_p)
conf_mat1 = confusion_matrix(y_test, y_ptest)
print("Confusion Matrix :")
print(confusion_matrix(y_test, y_ptest))
```

```
Confusion Matrix :
[[609  26]
 [ 36  20]]
```

[122]:
```
from sklearn.metrics import accuracy_score,average_precision_score,precision_recall_curve
print ("Accuracy on training Data :",accuracy_score(y_train, y_p))
print ("Accuracy on testing Data :",accuracy_score(y_test, y_ptest))
```

```
Accuracy on training Data : 0.9424746743849494
Accuracy on testing Data : 0.91027496382055
```

[123]:
```
acc_conf = np.trace(conf_mat)/np.sum(conf_mat)
acc_conf1 = np.trace(conf_mat1)/np.sum(conf_mat1)
print ()

print ("Accuracy on training Confusion Matrix :",acc_conf)
print ("Accuracy on testing Confusion Matrix :",acc_conf1)
```

```
Accuracy on training Confusion Matrix : 0.9424746743849494
Accuracy on testing Confusion Matrix : 0.91027496382055
```