SM5000 : Foundation of Machine Learning

Assignment – 1

Submitted By: –

Shubham Jain(SM20MTECH12007)

Rahul Bidla(SM20MTECH12014)

Solution 1 : Linear Regression [Theory]

It is the multivariate regression case where $(x_0, x_1, \ldots., x_n)$ are independent variables and $(y_0, y_1, \ldots., y_n)$ are dependent variables.

Dependent variables are depending on some set of independent variables so independent noise is added to each dimension of the independent variable set.

Our Model represented as:

$$y(x, \omega) = \omega_o + \sum_{i=1}^{D} \omega_i x_i$$

if error(noise) is added to each observed x value, our function of y will change as given

$$y'(x, \omega) = \omega_o + \sum_{i=1}^{D} \omega_i (x_i + \varepsilon_i)$$

$$y'(x, \omega) = \omega_o + \sum_{i=1}^{D} \omega_i\, x_i + x_i \varepsilon_i$$

$$y'(x, \omega) = y(x, \omega) + \sum_{i=1}^{D} \omega_i \varepsilon_i$$

Our New error function will be represented as:

$$E'_D(x, \omega) = \frac{1}{2} \sum_{n=1}^{N} \{ y'(x_n , \omega) - t_n \}^2$$

$$E'_D(x, \omega) = \frac{1}{2} \sum_{n=1}^{N} \{ y'(x_n, \omega) - t_n \}^2$$

$$E'_D(x, \omega) = \frac{1}{2} \sum_{n=1}^{N} \left\{ y(x_n , \omega) + \sum_{i=1}^{D} \omega_i \varepsilon_{ni} - t_n \right\}^2 \qquad \ldots \ldots \ldots .1$$

Expending equation 1:

$$E_D'(x,\omega) = \frac{1}{2}\sum_{n=1}^{N}\left\{ (y(x_n\,,\omega) - t_n)^2 + \left(\sum_{i=1}^{D}\omega_i\varepsilon_{ni}\right)^2 + 2(y(x_n\,,\omega) - t_n)\left(\sum_{i=1}^{D}\omega_i\varepsilon_{ni}\right)\right\} \ldots 2$$

Taking expectation of equation 2:

$$E\{E_D'(x,\omega)\} = \frac{1}{2}\; E\left\{\sum_{n=1}^{N}\left\{ (y(x_n\,,\omega) - t_n)^2 + \left(\sum_{i=1}^{D}\omega_i\varepsilon_{ni}\right)^2 + 2(y(x_n\,,\omega) - t_n)\left(\sum_{i=1}^{D}\omega_i\varepsilon_{ni}\right)\right\}\right\}$$

$$E\{E_D'(x,\omega)\} = \frac{1}{2}\sum_{n=1}^{N}\left\{E\{(y(x_n\,,\omega) - t_n)^2\} + E\left\{\left(\sum_{i=1}^{D}\omega_i\varepsilon_{ni}\right)^2\right\} + 2(y(x_n\,,\omega) - t_n)\left\{E\left(\sum_{i=1}^{D}\omega_i\varepsilon_{ni}\right)\right\}\right\}$$

I II III

I. $E\{(y(x_n\,,\omega) - t_n)^2\} = (y(x_n\,,\omega) - t_n)^2$ **as** $(y(x_n\,,\omega) - t_n)^2$ is free of noise

II. $2(y(x_n\,,\omega) - t_n)\{E(\sum_{i=1}^{D}\omega_i\varepsilon_{ni})\} = 2(y(x_n\,,\omega) - t_n)[(\sum_{i=1}^{D}\omega_i E\{\varepsilon_{ni}\})] = 0$

as $E\{\varepsilon_{ni}\} = 0$ is given to us.

III. Solving $E\{(\sum_{i=1}^{D}\omega_i\varepsilon_{ni})^2\}$ below:

$$\begin{aligned}
E\left\{\left(\sum_{i=1}^{D}\omega_i\varepsilon_{ni}\right)^2\right\} &= E\left\{\sum_{i=1}^{D}\sum_{j=1}^{D}\omega_i\varepsilon_{ni}\,\omega_j\varepsilon_{nj}\right\} \\
&= E\left\{\sum_{i=1}^{D}\sum_{j=1}^{D}\omega_i\,\omega_j\varepsilon_{ni}\varepsilon_{nj}\right\} \\
&= \sum_{i=1}^{D}\sum_{j=1}^{D}\omega_i\,\omega_j E\{\varepsilon_{ni}\varepsilon_{nj}\} \\
&= \sum_{i=1}^{D}\sum_{j=1}^{D}\omega_i\,\omega_j\delta_{ij}\sigma^2 \\
&= \sum_{i=1}^{D}{\omega_i}^2\sigma^2
\end{aligned}$$

Substituting the above results in equation 3, we obtain:

$$E\{E_D'(x,\omega)\} = \frac{1}{2}\sum_{n=1}^{N}\left\{(y(x_n\,,\omega) - t_n)^2 + \sum_{i=1}^{D}\omega_i{}^2\sigma^2\right\}$$

final equation will be:

$$\boldsymbol{E\{E_D'(x,\omega)\} = \; E_D(\omega) + \; \frac{N}{2}\sum_{i=1}^{D}\omega_i^2\,\sigma^2}$$

$\therefore\ Here, we\ get\ a\ weight\ decay\ (L2\ regularization\ term\ in\ which\ bias\ term\ (\omega o)\ is\ absent).$

Solution 2 : $\mathbf{Multi - Output\ Regression}$ [Theory]

Solution 2 (Part-1): Provide the expression for the likelihood, and derive ML and MAP estimates of W in the multi output regression case.

1. **a) For MLE:**

Multiple outputs: In some applications, we may wish to predict K > 1 target variables, which we denote collectively by the target vector t. This could be done by introducing a different set of basis functions for each component of t, leading to multiple, independent regression problems. However, a more interesting, and more common, approach is to use the same set of basis functions to model all of the components of the target vector so that

$$\mathrm{y(x, w)} = \mathbf{W^T}\emptyset(\mathrm{x})$$

where y is a K-dimensional column vector, W is an M × K matrix of parameters, and φ(x) is an M-dimensional column vector with elements φj (x), with φ0(x)=1 as before. Suppose we take the conditional distribution of the target vector to be an isotropic Gaussian of the form

$$p(\mathrm{t}|\mathrm{x}, \mathbf{W}, \beta) = \mathcal{N}(\mathrm{t}|\mathbf{W^T}\emptyset(\mathrm{x}), \beta^{-1}\mathbf{I})$$

Taking Log we get:-

$$\begin{aligned}\ln p(\mathbf{T}|\mathbf{X}, \mathbf{W}, \beta) &= \sum_{n=1}^{N} \ln\ \mathcal{N}(t_n|\mathbf{W^T}\emptyset(\mathrm{x_n}), \beta^{-1}\mathbf{I})\\ &= \frac{NK}{2}\ln\left(\frac{\beta}{2\pi}\right) - \frac{\beta}{2}\sum_{n=1}^{N}\|t_n - \mathbf{W^T}\emptyset(\mathrm{x_n})\|^{\mathbf{2}}\end{aligned}$$

equating $\mathrm{argmax_W}\ \ln \mathrm{p}(\mathbf{T}|\mathbf{X}, \mathbf{W}, \beta) = 0$ and solving for W will give us $\mathrm{W_{ML}}$(estimated)

$$argmax_W \ln p(\mathbf{T}|\mathbf{X}, \mathbf{W}, \beta)\ = argmax_W\ \left(\frac{NK}{2}\ln\left(\frac{\beta}{2\pi}\right) - \frac{\beta}{2}\sum_{n=1}^{N}\|t_n - \mathbf{W^T}\emptyset(\mathrm{x_n})\|^{\mathbf{2}}\right)$$

$$argmax_W \ln p(\mathbf{T}|\mathbf{X}, \mathbf{W}, \beta)$$

$$=\ argmax_W\ \left(\frac{NK}{2}\ln\left(\frac{\beta}{2\pi}\right)\right) + argmax_W\left(-\frac{\beta}{2}\sum_{n=1}^{N}\|t_n - \mathbf{W^T}\emptyset(\mathrm{x_n})\|^{\mathbf{2}}\right)$$

We know $argmax_W\ (X)\ =\ argmin_W\ (-X)$

$$argmax_W\ \ \ln p(\mathbf{T}|\mathbf{X},\mathbf{W},\beta)\ =\ 0 + argmin_W\left(\frac{\beta}{2}\sum_{n=1}^{N}\|t_n - \mathbf{W^T}\emptyset(\mathrm{x_n})\|^{\mathbf{2}}\right)$$

$$argmax_W\ \ \ln p(\mathbf{T}|\mathbf{X},\mathbf{W},\beta)\ = \frac{\beta}{2}.2\sum_{n=1}^{N}\Big(t_n - \mathbf{W^T}\emptyset(\mathrm{x_n})\Big)\Big(0 - \emptyset^{\mathbf{T}}(\mathrm{x_n})\Big) = 0$$

$$= \sum_{n=1}^{N}\Big(-t_n\emptyset^{\mathbf{T}}(\mathrm{x_n}) + \mathbf{W^T}\emptyset(\mathrm{x_n})\emptyset^{\mathbf{T}}(\mathrm{x_n})\Big) = 0$$

Taking $-t_n\emptyset^{\mathbf{T}}(\mathrm{x_n})$ to right side of equal sign and taking transpose, we get:

$$\sum_{n=1}^{N}\Big(\mathbf{W^T}\emptyset(\mathrm{x_n})\emptyset^{\mathbf{T}}(\mathrm{x_n})\Big)^T = \sum_{n=1}^{N}\Big(t_n\emptyset^{\mathbf{T}}(\mathrm{x_n})\Big)^T$$

Solving transpose we get:-

$$\sum_{n=1}^{N}\emptyset(\mathrm{x_n})\emptyset^{\mathbf{T}}(\mathrm{x_n}).W = \sum_{n=1}^{N}t_n{}^T.\emptyset(\mathrm{x_n})$$

Finally :-

$$W = \sum_{n=1}^{N}\Big(\emptyset(\mathrm{x_n})\emptyset^{\mathbf{T}}(\mathrm{x_n})\Big)^{-1}\sum_{n=1}^{N}t_n{}^T.\emptyset(\mathrm{x_n})$$

This can be written as:-

$$\mathrm{W_{ML}} = (\boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{\Phi})^{-\mathbf{1}}.(\boldsymbol{T}^{\boldsymbol{T}}\boldsymbol{\Phi})$$

Hence, we get answer in closed form

1. b) For MAP:

- Prior

  Suppose we take the prior distribution to be an isotropic Gaussian of the form

$$P(W|\alpha) = \mathcal{N}(W|0, \alpha^{-1}\mathbf{I})$$
$$= \frac{1}{\left(\sqrt{2\pi\alpha^{-1}}\right)^k} \exp\left(-\frac{(W-0)^2}{2(\alpha^{-1})}\right)$$
$$= \left(\frac{\alpha}{2\pi}\right)^{\frac{k}{2}} \exp\left(\frac{-\alpha\|w\|^2}{2}\right)$$

- Posterior

  Posterior can be written as

$$P(W|X,t,\alpha,\beta) = \frac{P(t \,\big|\, X,t,W,\beta)\;\; P(W \,\big|\, \alpha)}{P(t|X,\alpha,\beta)}$$

$$P(W|X,t,\alpha,\beta)\ \alpha\ .\ P(t|X,W,\beta).P(W|\alpha)$$

$$P(W|X,t,\alpha,\beta) = \left(\frac{\beta}{2\pi}\right)^{\frac{k}{2}} exp\left(\frac{-\beta}{2}\left(t - \emptyset(\mathrm{x_n})\right)^2\right).\left(\frac{\alpha}{2\pi}\right)^{\frac{k}{2}} \exp\left(\frac{-\alpha}{2}W^TW\right)$$

Taking log both sides we get-

$$ln\ (P(W|X,t,\alpha,\beta)) = ln\sum_{n=1}^{N}\left(\frac{\beta}{2\pi}\right)^{\frac{k}{2}}.exp\left[\frac{-\beta}{2}\left(t-\emptyset(\mathrm{x_n})\right)^2\right] + n\sum_{n=1}^{N}\left(\frac{\alpha}{2\pi}\right)^{\frac{k}{2}} \exp\left(\frac{-\alpha}{2}W^TW\right)$$

$$= \frac{NK}{2}\ ln\left(\frac{\beta}{2\pi}\right) - \frac{\beta}{2}\sum_{n=1}^{N}\|t_n - \mathbf{W^T}\emptyset(\mathrm{x_n})\|^{\mathbf{2}} + \frac{NK}{2}\ ln\left(\frac{\alpha}{2\pi}\right) - \frac{\alpha}{2}\sum_{n=1}^{N}(W^TW)$$

Equating $argmax_W\ (\ln P(W|X,t,\alpha,\beta)) = 0$ and solving for W will give us W(MAP)

$$argmax_W\left(\frac{NK}{2}\ ln\left(\frac{\beta}{2\pi}\right)-\frac{\beta}{2}\sum_{n=1}^{N}\|t_n-\mathbf{W^T}\emptyset(\mathrm{x_n})\|^{\mathbf{2}}+\frac{NK}{2}\ ln\left(\frac{\alpha}{2\pi}\right)-\frac{\alpha}{2}\sum_{n=1}^{N}\|W\|^2\right) = 0 \qquad \mathbf{-1}$$

- $argmax_W\left(\frac{NK}{2}\ ln\left(\frac{\beta}{2\pi}\right)\right) = 0 \qquad \mathbf{-2}$
- $argmax_W\left(\frac{NK}{2}\ ln\left(\frac{\alpha}{2\pi}\right)\right) = 0 \qquad \mathbf{-3}$

Put equation 2 , and equation 3 in equation we get

$$argmax_W\ \left(-\frac{\beta}{2}\sum_{n=1}^{N}\|\mathrm{t_n}-\mathbf{W^T}\emptyset(\mathrm{x_n})\|^{\mathbf{2}}-\sum_{n=1}^{N}\frac{\alpha}{2}\|\mathrm{W}\|^2\right) = 0$$

We know $argmax_W\ (X) = argmin_W\ (-X)$

$$argmin_W\ \left(\frac{\beta}{2}\sum_{n=1}^{N}\|t_n-\mathbf{W^T}\emptyset(\mathrm{x_n})\|^{\mathbf{2}}+\frac{\alpha}{2}\sum_{n=1}^{N}\|W\|^2\right) = 0$$

$$\frac{\beta}{2}.2\sum_{n=1}^{N}\left(t_n-\mathbf{W^T}\emptyset(\mathrm{x_n})\right)\left(0-\emptyset^{\mathbf{T}}(\mathrm{x_n})\right)+\frac{\alpha}{2}.2\sum(W^T) = 0$$

$$\sum_{n=1}^{N}\left(\beta\left(-t_n\emptyset^{\mathbf{T}}(\mathrm{x_n})+\mathbf{W^T}\emptyset(\mathrm{x_n})\emptyset^{\mathbf{T}}(\mathrm{x_n})\right)+\alpha.W^T\right) = 0$$

$$\sum_{n=1}^{N}(-t_n\emptyset^{\mathbf{T}}(\mathrm{x_n})+\mathbf{W^T}\emptyset(\mathrm{x_n})\emptyset^{\mathbf{T}}(\mathrm{x_n})+\lambda.W^T) = 0$$

$$where\ \lambda = \frac{\alpha}{\beta}$$

Taking $\sum_{n=1}^{N} t_n \emptyset^{\mathbf{T}}(\mathrm{x_n})$ term to right side of equal sign

$$\sum_{n=1}^{N} \mathbf{W^T} \emptyset(\mathrm{x_n}) \emptyset^{\mathbf{T}}(\mathrm{x_n}) + \lambda . W^T = \sum_{n=1}^{N} t_n \emptyset^{\mathbf{T}}(\mathrm{x_n})$$

$$\sum_{n=1}^{N} (W^T)(\emptyset(\mathrm{x_n}) \emptyset^{\mathbf{T}}(\mathrm{x_n}) + \lambda) = \sum_{n=1}^{N} t_n \emptyset^{\mathbf{T}}(\mathrm{x_n})$$

Taking Transpose, and solving for W.

$$\sum_{n=1}^{N} (\emptyset^{\mathbf{T}}(\mathrm{x_n}) \emptyset(\mathrm{x_n}) + \lambda)(W) = \sum_{n=1}^{N} \emptyset(\mathrm{x_n})\, {t_n}^T$$

Finally we get-

$$W = \sum_{n=1}^{N} (\emptyset^{\mathbf{T}}(\mathrm{x_n}) \emptyset(\mathrm{x_n}) + \lambda)^{-1} \emptyset(\mathrm{x_n}) {t_n}^T$$

Which can be written as closed form

$$\boldsymbol{W_{MAP}} = (\boldsymbol{\Phi^T \Phi} + \boldsymbol{\lambda}\mathbf{I})^{-\mathbf{1}} \boldsymbol{\Phi}$$

Where $\lambda = \frac{\alpha}{\beta}$

We see that this is somewhat similar to WMLE, the only difference is the $\boldsymbol{\lambda}\mathbf{I}$ is added to $\boldsymbol{\Phi^T \Phi}$.

Solution 2 (Part-2): Find the MLE for w1 and w2

- $X = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$

- $X^T = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$

- $Y = \begin{bmatrix} -1 & -1 \\ -1 & -2 \\ -2 & -1 \\ 1 & 1 \\ 1 & 2 \\ 2 & 1 \end{bmatrix}$

We Derived above the closed form solution for w in case of multi output regression as-

$$\boldsymbol{W} = (\boldsymbol{\varphi}^{\boldsymbol{T}}\boldsymbol{\varphi})\ \boldsymbol{\varphi}^{\boldsymbol{T}}\boldsymbol{Y}$$

Here W will be 2X2 matrix

Finding values according to closed form solution-

- $\varphi^T\varphi = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$

- $(\varphi^T\varphi)^{-1} = \frac{Adj\ (\varphi^T\varphi)}{|\varphi^T\varphi|} = \frac{\begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}}{9}$

- $\varphi^T Y = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -1 & -1 \\ -1 & -2 \\ -2 & -1 \\ 1 & 1 \\ 1 & 2 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} -4 & -4 \\ 4 & 4 \end{bmatrix}$

Substituting above Matrix forms in our equation $\boldsymbol{W} = \ (\boldsymbol{\varphi^T \varphi})\ \boldsymbol{\varphi^T Y}$ we get

$$W_{2x2} = \frac{1}{9} \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} -4 & -4 \\ 4 & 4 \end{bmatrix}$$

Further Solving this we get-

$$\begin{bmatrix} \omega_1 & \omega_2 \\ \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \frac{-4}{3} & \frac{-4}{3} \\ \frac{4}{3} & \frac{4}{3} \end{bmatrix}$$

w_1 and w_2 are

$$\omega_1 = \begin{bmatrix} \frac{-4}{3} \\ \frac{4}{3} \end{bmatrix}$$

$$\omega_2 = \begin{bmatrix} \frac{-4}{3} \\ \frac{4}{3} \end{bmatrix}$$

Answer 3 [Programming Question] : MAP and ML estimation of Poisson Distribution (Modeling the horse kick deaths)

Importing Libraries

In [1]:
```
import pandas as pd
from scipy.special import factorial
import scipy as scy
import scipy.stats as stats
from scipy.stats import poisson
import numpy as np
import matplotlib.pyplot as plt
```

Given data is divided in training data(13 years) and test data(7 years)

Training Data : 1875 to 1887 to model the Poisson distribtion. Test Data : 1888 to 1894 to test the model.

Importing Training data

In [2]:
```
training = pd.read_csv('training_set.csv', index_col = 'YEAR')
training
```

Out[2]:

YEAR	G	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XIV	XV
1875	0	0	0	0	0	0	0	1	1	0	0	0	1	0
1876	2	0	0	0	1	0	0	0	0	0	0	0	1	1
1877	2	0	0	0	0	0	1	1	0	0	1	0	2	0
1878	1	2	2	1	1	0	0	0	0	0	1	0	1	0
1879	0	0	0	1	1	2	2	0	1	0	0	2	1	0
1880	0	3	2	1	1	1	0	0	0	2	1	4	3	0
1881	1	0	0	2	1	0	0	1	0	1	0	0	0	0
1882	1	2	0	0	0	0	1	0	1	1	2	1	4	1
1883	0	0	1	2	0	1	2	1	0	1	0	3	0	0
1884	3	0	1	0	0	0	0	1	0	0	2	0	1	1
1885	0	0	0	0	0	0	1	0	0	2	0	1	0	1
1886	2	1	0	0	1	1	1	0	0	1	0	1	3	0
1887	1	1	2	1	0	0	3	2	1	1	0	1	2	0

Importing Test data

In [3]:
```
test_data = pd.read_csv('test_set.csv' ,index_col = 'YEAR')
test_data
```

Out[3]:

YEAR	G	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XIV	XV
1888	0	1	1	0	0	1	1	0	0	0	0	1	1	0
1889	0	0	1	1	0	1	1	0	0	1	2	2	0	2
1890	1	2	0	2	0	1	1	2	0	2	1	1	2	2
1891	0	0	0	1	1	1	0	1	1	0	3	3	1	0
1892	1	3	2	0	1	1	3	0	1	1	0	1	1	0
1893	0	1	0	0	0	1	0	2	0	0	1	3	0	0
1894	1	0	0	0	0	0	0	0	1	0	1	1	0	0

In [4]:
```
headings = list(test_data)
```

```
print(headings)
```

['G', 'I', 'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX', 'X', 'XI', 'XIV', 'XV']

important stats of training data

In [5]:
```
training.describe()
```

Out[5]:

	G	I	II	III	IV	V	VI	VII	VIII	IX	X	XI
count	13.0	13.000000	13.000000	13.000000	13.000000	13.000000	13.000000	13.000000	13.000000	13.000000	13.000000	13.000000
mean	1.0	0.692308	0.615385	0.615385	0.461538	0.384615	0.846154	0.538462	0.307692	0.692308	0.538462	1.000000
std	1.0	1.031553	0.869718	0.767948	0.518875	0.650444	0.987096	0.660225	0.480384	0.751068	0.776250	1.290994
min	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	1.0	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000	0.000000	1.000000
75%	2.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
max	3.0	3.000000	2.000000	2.000000	1.000000	2.000000	3.000000	2.000000	1.000000	2.000000	2.000000	4.000000

The given data has been modeled using Poisson distribution.

Poisson Distribution:

PDF for poissons is given as

$$p(\text{k events in interval}) = \frac{e^{-\lambda}\lambda^{k}}{k!} \tag{1}$$

where λ is the average (mean) no of events per interval and k is no of events occured independently.

Part 1: MLE to learn the parameters

Let $X_1, X_2, \ldots\ldots, X_n$ are independent Poisson random variables each having mean λ, then maximum likelihood estimator(MLE) of λ can be given as

$$\hat{\lambda}_{ML} = \frac{\sum_1^n x_i}{n} \tag{2}$$

Example:

For G-Corps $\hat{\lambda}_{ML}$ estimate will be

$$\hat{\lambda}_{ML} = \frac{0+2+2+1+0+0+1+1+0+3+0+2+1}{13} = 1 \tag{3}$$

In [6]:
```
list1 = list(training.mean())
lambdaForEachCorps= pd.DataFrame([training.mean()],index = ['MLE (λ)'] ,columns = headings)
display(lambdaForEachCorps)
```

	G	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XIV	XV
MLE (λ)	1.0	0.692308	0.615385	0.615385	0.461538	0.384615	0.846154	0.538462	0.307692	0.692308	0.538462	1.0	1.461538	0.307692

we use MLE ($\hat{\lambda}_{ML}$), to predict no of deaths for remaining 7 years.

Example:

For G-Corps

$$p(\text{0 deaths in 7 years}) = \frac{e^{-1}1^{0}}{0!} * 7 = 3 \tag{4}$$

In [7]:
```
summed1 = 0
for idx, i in enumerate(list1):
```

```
    t = np.arange(0, 5, 1)
    d = np.exp(-i)*np.power(i, t)/factorial(t)
    pred = d*7

    actual_for_thisYear=(list(test_data[headings[idx]]))

    sum1=0

    predictedList = []

    actualList = []

    for idx2, num in enumerate(pred):
        a=round(num)
        predictedList.append(a)

        b=actual_for_thisYear.count(idx2)
        actualList.append(b)

        mse= round(np.power((a-b),2)/7,2)

        sum1 = sum1 + mse


    RMSE = round(np.power(sum1,0.5),2)
    sum1 = pd.DataFrame([[sum1 , RMSE]], index = [f'For {headings[idx]} corp'], columns = ['MSE','RMSE'])
    summed1 = summed1 +RMSE
    prediction= pd.Series(predictedList)
    actual= pd.Series(actualList)
    df = pd.DataFrame([predictedList,actualList], index =[ f'PREDICTION for {headings[idx]} corp',
                                                            f'ACTUAL for {headings[idx]} corp'],
                                                 columns= ['yearsWithZeroDeath','yearsWithOneDeath',
                                                           'yearsWithTwoDeath','yearsWithThreeDeath',
                                                          'yearsWithFourDeath'])


    display(df)
    display(sum1)
    print('--------------------------------------------------------------------------------------')
print(summed1)
```

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for G corp	3	3	1	0	0
ACTUAL for G corp	4	3	0	0	0

	MSE	RMSE
For G corp	0.28	0.53

----

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for I corp	4	2	1	0	0
ACTUAL for I corp	3	2	1	1	0

	MSE	RMSE
For I corp	0.28	0.53

----

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for II corp	4	2	1	0	0
ACTUAL for II corp	4	2	1	0	0

	MSE	RMSE
For II corp	0.0	0.0

----

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for III corp	4	2	1	0	0
ACTUAL for III corp	4	2	1	0	0

	MSE	RMSE
For III corp	0.0	0.0

---

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for IV corp	4	2	0	0	0
ACTUAL for IV corp	5	2	0	0	0

	MSE	RMSE
For IV corp	0.14	0.37

---

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for V corp	5	2	0	0	0
ACTUAL for V corp	1	6	0	0	0

	MSE	RMSE
For V corp	4.58	2.14

---

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for VI corp	3	3	1	0	0
ACTUAL for VI corp	3	3	0	1	0

	MSE	RMSE
For VI corp	0.28	0.53

---

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for VII corp	4	2	1	0	0
ACTUAL for VII corp	4	1	2	0	0

	MSE	RMSE
For VII corp	0.28	0.53

---

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for VIII corp	5	2	0	0	0
ACTUAL for VIII corp	4	3	0	0	0

	MSE	RMSE
For VIII corp	0.28	0.53

---

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for IX corp	4	2	1	0	0
ACTUAL for IX corp	4	2	1	0	0

	MSE	RMSE
For IX corp	0.0	0.0

---

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for X corp	4	2	1	0	0
ACTUAL for X corp	2	3	1	1	0

	MSE	RMSE
For X corp	0.85	0.92

```
--------------------------------------------------------------------------------
```

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for XI corp	3	3	1	0	0
ACTUAL for XI corp	0	4	1	2	0

	MSE	RMSE
For XI corp	2.0	1.41

```
--------------------------------------------------------------------------------
```

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for XIV corp	2	2	2	1	0
ACTUAL for XIV corp	3	3	1	0	0

	MSE	RMSE
For XIV corp	0.56	0.75

```
--------------------------------------------------------------------------------
```

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for XV corp	5	2	0	0	0
ACTUAL for XV corp	5	0	2	0	0

	MSE	RMSE
For XV corp	1.14	1.07

```
--------------------------------------------------------------------------------
9.310000000000002
```

2) Using maximum aposteriori estimation to learn parameters

a) Prior Assumption: Gamma Distribution with parameters shape(α) = 2.5 and rate(β) = 1

$$\lambda \sim \Gamma(\alpha, \beta)$$

- Justification: given distribution is Poisson distribution which supports interval from 0 to ∞ and the rate of deaths per year $\approx$ 1.

Hence we have choosen gamma distribution as the prior distribution, which supports from interval 0 to ∞ and the rate(β) parameter =1 and shape(α) =2

The pdf of gamma distribution can be given as

$$f(\lambda) = \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta\lambda}$$

Maximum A-posteriori (MAP) Estimation:

$$prob(\lambda|y) = \frac{prob(y|\lambda).prob(\lambda)}{prob(y)}$$

$$\hat{\lambda}_{MAP} = \underset{\lambda}{argmax}(\prod prob(y|\lambda).prob(\lambda))$$

Using Loglokelihood

$$\hat{\lambda}_{MAP} = \underset{\lambda}{argmax}(\sum \log prob(y|\lambda) + \log prob(\lambda))$$

by substituting likelihood and prior distributions

$$\hat{\lambda}_{MAP} = \underset{\lambda}{argmax}(\sum \log(\lambda^{\sum y_i} e^{-n\lambda}) + \log(\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda})$$

$$\hat{\lambda}_{MAP} = \underset{\lambda}{argmax}(\log(\lambda^{\sum y_i} e^{-n\lambda}) + \log(\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda})$$

MAP estimate will be obtained by equating the gradient of above eq to 0.

$$\frac{\alpha-1}{\lambda} - \beta + \frac{\sum y_i}{\lambda} - n = 0$$

$$\therefore \hat{\lambda}_{MAP} = \frac{\alpha + \sum y_i - 1}{\beta + n}$$

Example:

For G-Corps

$$\hat{\lambda}_{MAP} = \frac{2.5 + (0+2+2+1+0+0+1+1+0+3+0+2+1) - 1}{1+13} = 1.035 \tag{5}$$

we use ($\hat{\lambda}_{MAP}$), to predict no of deaths for remaining 7 years.

$$p(\text{k deaths in 7 years}) = \frac{e^{-\hat{\lambda}_{MAP}} \hat{\lambda}_{MAP}^{\,k}}{k!} * 7 \tag{6}$$

In [8]:

```
α = 2.5
β = 1
lambdaForMAP = []
for i in training:
    sum1 = np.sum(training[i])
    n = len(training)
    lambdaForMAP.append((α + sum1 -1)/(β+n))
lambdaMAPForEachCorps= pd.DataFrame([lambdaForMAP], index = ['MAP (λ)'] ,columns = headings)
display(lambdaMAPForEachCorps)
```

	G	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XIV	XV
MAP (λ)	1.035714	0.75	0.678571	0.678571	0.535714	0.464286	0.892857	0.607143	0.392857	0.75	0.607143	1.035714	1.464286	0.392857

In [9]:

```
summed =0
for idx, i in enumerate(lambdaForMAP):
    t = np.arange(0, 5, 1)
    d = np.exp(-i)*np.power(i, t)/factorial(t)
    pred1 = d*7

    actual_for_thisYear=(list(test_data[headings[idx]]))

    sum11=0

    predictedListMAP = []

    actualList = []

    for idx2, num in enumerate(pred1):
        a=round(num)
        predictedListMAP.append(a)

        b=actual_for_thisYear.count(idx2)
        actualList.append(b)

        mse= round(np.power((a-b),2)/7,2)

        sum11 = sum11 + mse


    RMSE = round(np.power(sum11,0.5),2)
    summed = summed +RMSE
    sum11 = pd.DataFrame([[sum11 , RMSE]], index = [f'For {headings[idx]} corp'], columns = ['MSE','RMSE'])

    prediction= pd.Series(predictedListMAP)
    actual= pd.Series(actualList)
    df1 = pd.DataFrame([predictedListMAP,actualList], index =[ f'PREDICTION for {headings[idx]} corp',
                                                              f'ACTUAL for {headings[idx]} corp'],
                                                  columns= ['yearsWithZeroDeath','yearsWithOneDeath',
```

```
                                                    'yearsWithTwoDeath','yearsWithThreeDeath',
                                                    'yearsWithFourDeath'])

    display(df1)
    display(sum11)
print(summed)
```

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for G corp	2	3	1	0	0
ACTUAL for G corp	4	3	0	0	0

	MSE	RMSE
For G corp	0.71	0.84

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for I corp	3	2	1	0	0
ACTUAL for I corp	3	2	1	1	0

	MSE	RMSE
For I corp	0.14	0.37

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for II corp	4	2	1	0	0
ACTUAL for II corp	4	2	1	0	0

	MSE	RMSE
For II corp	0.0	0.0

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for III corp	4	2	1	0	0
ACTUAL for III corp	4	2	1	0	0

	MSE	RMSE
For III corp	0.0	0.0

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for IV corp	4	2	1	0	0
ACTUAL for IV corp	5	2	0	0	0

	MSE	RMSE
For IV corp	0.28	0.53

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for V corp	4	2	0	0	0
ACTUAL for V corp	1	6	0	0	0

	MSE	RMSE
For V corp	3.58	1.89

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for VI corp	3	3	1	0	0
ACTUAL for VI corp	3	3	0	1	0

	MSE	RMSE
For VI corp	0.28	0.53

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for VII corp	4	2	1	0	0
ACTUAL for VII corp	4	1	2	0	0

	MSE	RMSE
For VII corp	0.28	0.53

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for VIII corp	5	2	0	0	0
ACTUAL for VIII corp	4	3	0	0	0

	MSE	RMSE
For VIII corp	0.28	0.53

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for IX corp	3	2	1	0	0
ACTUAL for IX corp	4	2	1	0	0

	MSE	RMSE
For IX corp	0.14	0.37

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for X corp	4	2	1	0	0
ACTUAL for X corp	2	3	1	1	0

	MSE	RMSE
For X corp	0.85	0.92

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for XI corp	2	3	1	0	0
ACTUAL for XI corp	0	4	1	2	0

	MSE	RMSE
For XI corp	1.28	1.13

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for XIV corp	2	2	2	1	0
ACTUAL for XIV corp	3	3	1	0	0

	MSE	RMSE
For XIV corp	0.56	0.75

	yearsWithZeroDeath	yearsWithOneDeath	yearsWithTwoDeath	yearsWithThreeDeath	yearsWithFourDeath
PREDICTION for XV corp	5	2	0	0	0
ACTUAL for XV corp	5	0	2	0	0

	MSE	RMSE
For XV corp	1.14	1.07

```
9.46
```

Since the Posteriori will be a Gamma distribution

$$\lambda \ \sim \ \ \Gamma(\alpha + \sum y_i, \beta + n)$$

The values of shape and rate parameters calculated each Crop wise.

Mode for MLE,Mode for Prior and Mode for MAP are calculated below

In [10]:
```
modeForMLE = np.floor(list1)
#print(modeForMLE)
```

mode = For non-integer λ, it is the largest integer less than λ. For integer λ, x = λ and x = λ - 1 are both the mode.

In [11]:
```
modeForPrior = []
for i in training:
    modePrior = round((α-1) / β , 2)
    modeForPrior.append(modePrior)
#print(modeForPrior)
```

In [12]:
```
modeForMAP = [ round(elem, 2) for elem in lambdaForMAP]
#print(modeForMAP)
```

In [13]:
```
mode = pd.DataFrame([modeForMLE,modeForPrior,modeForMAP], index = ['Mode For MLE','Mode For Prior','Mode For MAP']
                                                    columns = headings)
display(mode)
```

	G	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XIV	XV
Mode For MLE	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	0.00
Mode For Prior	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50
Mode For MAP	1.04	0.75	0.68	0.68	0.54	0.46	0.89	0.61	0.39	0.75	0.61	1.04	1.46	0.39

Part 2: Using maximum aposteriori estimation to learn parameters

---

- **For II - Corps:**

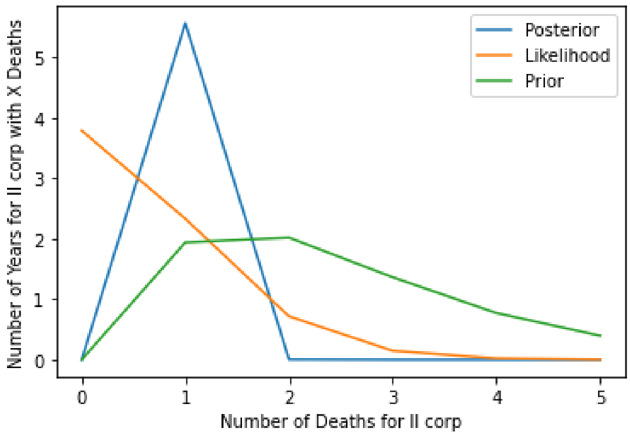
Likelihood mode = 0.00 prior mode = 1.5 Posteriori mode = 0.68

mode of Posteriori distribution is in between likelihood and prior distributions

Posterior,Likelihood and Prior for Number of Deaths for II corp

In [14]:
```
x= np.arange(0.0,6.0,1)

#a=plt.plot(x, poisson.pmf(x,lambdaForMAP[2]),label='Posterior')
a=plt.plot(x, 7*stats.gamma.pdf(x,a= α + np.sum(training["II"]) , scale=1/(β+13) ),label='Posterior')
b=plt.plot(x, 7*poisson.pmf(x,list1[2]),label='Likelihood')
c=plt.plot(x, 7*stats.gamma.pdf(x, a=α, scale=1/β),label='Prior')
plt.legend()
plt.xlabel('Number of Deaths for II corp')
plt.ylabel('Number of Years for II corp with X Deaths')
plt.show()
```



- **For IV - Corps:**

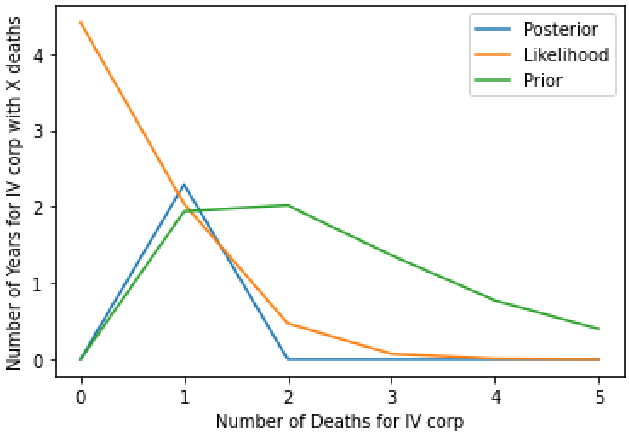
Likelihood mode = 0.00 prior mode = 1.5 Posteriori mode = 0.54

mode of Posteriori distribution is in between likelihood and prior distributions

Posterior,Likelihood and Prior for Number of Deaths for IV corp

In [15]:
```
x= np.arange(0.0,6.0,1)

#a=plt.plot(x, poisson.pmf(x,lambdaForMAP[4]),label='Posterior')
a=plt.plot(x, 7*stats.gamma.pdf(x,a= α + np.sum(training["IV"]) , scale=1/(β+13) ),label='Posterior')
b=plt.plot(x, 7*poisson.pmf(x,list1[4]),label='Likelihood')
c=plt.plot(x, 7*stats.gamma.pdf(x, a=α, scale=1/β),label='Prior')
plt.legend()
plt.xlabel('Number of Deaths for IV corp')
plt.ylabel('Number of Years for IV corp with X deaths')
plt.show()
```



- **For VI - Corps:**

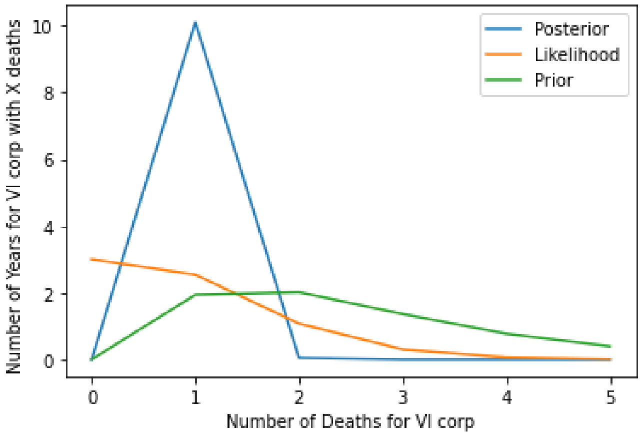
Likelihood mode = 0.00 prior mode = 1.5 Posteriori mode = 0.89

mode of Posteriori distribution is in between likelihood and prior distributions

Posterior,Likelihood and Prior for Number of Deaths for VI corp

In [16]:
```
x= np.arange(0.0,6.0,1)

#a=plt.plot(x, poisson.pmf(x,lambdaForMAP[6]),label='Posterior')
a=plt.plot(x, 7*stats.gamma.pdf(x,a= α + np.sum(training["VI"]) , scale=1/(β+13) ),label='Posterior')
b=plt.plot(x, 7*poisson.pmf(x,list1[6]),label='Likelihood')
c=plt.plot(x, 7*stats.gamma.pdf(x, a=α, scale=1/β),label='Prior')
plt.legend()
plt.xlabel('Number of Deaths for VI corp')
plt.ylabel('Number of Years for VI corp with X deaths ')
plt.show()
```



.

---

completed by:- Shubham jain(SM20MTECH12007) and Rahul bidla(SM20MTECH12014)

Question 4: Bike Sharing Demand

Part 1: Explain maximum likelihood estimation in poisson regression and derive the loss function which is used to estimate the parameteres.

mean of poisson's equation:

$$\lambda = \mathbb{E}[Y|x] = e^{\theta^T x} \tag{1}$$

Pdf for poissons eqn is given as

$$p(y|x,\theta) = \frac{\lambda^y e^{-\lambda}}{y!} = \frac{e^{y\theta^T x} e^{-e^{\theta^T x}}}{y!} \tag{2}$$

the probability for observing y1.....yn is given as

$$p(y_1,\ldots,y_m \mid x_1,\ldots,x_m;\theta) = \prod_{i=1}^{m} \frac{e^{y_i\theta' x_i} e^{-e^{\theta' x_i}}}{y_i!}. \tag{3}$$

the above equation can be represented as likelihood function

$$L(\theta \mid X,Y) = \prod_{i=1}^{m} \frac{e^{y_i\theta' x_i} e^{-e^{\theta' x_i}}}{y_i!}. \tag{4}$$

taking log ov above likelihood function gives us log likelihood function

$$\ell(\theta \mid X,Y) = \log L(\theta \mid X,Y) = \sum_{i=1}^{m} \left(y_i\theta' x_i - e^{\theta' x_i} - \log(y_i!) \right). \tag{5}$$

$$\ell(\theta \mid X,Y) = \sum_{i=1}^{m} \left(y_i\theta' x_i - e^{\theta' x_i} \right). \tag{6}$$

minimising log likelihood and solving for theta

$$\underset{\theta}{minimize}\ \ell(\theta \mid X,Y) = \sum_{i=1}^{m} \left(e^{\theta' x_i} - y_i\theta' x_i \right). \tag{7}$$

$$\nabla\ell(\theta \mid X,Y) = \frac{\partial\ell(\theta \mid X,Y)}{\partial\theta} = \sum_{i=1}^{m} \left(e^{\theta' x_i} - y_i \right) x_i \tag{8}$$

we get theta as:-

$$\theta := \theta - \alpha\nabla\ell(\theta \mid X,Y) \tag{9}$$

Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib notebook
```

Importing Data files

In [2]:
```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

In [3]:
```
train.head(2)
display(test.head(2))
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.0027
1	2011-01-20 01:00:00	1	0	1	1	10.66	13.635	56	0.0000

Processing Given Data

In [4]:
```
train['year'] = pd.DatetimeIndex(train['datetime']).year
train['month'] = pd.DatetimeIndex(train['datetime']).month
train['day'] = pd.DatetimeIndex(train['datetime']).day
train['dayOfWeek'] = pd.DatetimeIndex(train['datetime']).weekday
train['hour'] = pd.DatetimeIndex(train['datetime']).hour
```

In [5]:
```
test['year'] = pd.DatetimeIndex(test['datetime']).year
test['month'] = pd.DatetimeIndex(test['datetime']).month
test['day'] = pd.DatetimeIndex(test['datetime']).day
test['dayOfWeek'] = pd.DatetimeIndex(test['datetime']).weekday
test['hour'] = pd.DatetimeIndex(test['datetime']).hour
```

In [6]:
```
train = train[['year','season','month','day','dayOfWeek','hour','holiday','workingday',
               'weather','temp','atemp','humidity','windspeed','count']]
train.head(2)
```

Out[6]:

	year	season	month	day	dayOfWeek	hour	holiday	workingday	weather	temp	atemp	humidity	winds
0	2011	1	1	1	5	0	0	0	1	9.84	14.395	81	
1	2011	1	1	1	5	1	0	0	1	9.02	13.635	80	

In [7]:
```
test = test[['year','season','month','day','dayOfWeek','hour','holiday','workingday',
             'weather','temp','atemp','humidity','windspeed']]
```

In [8]:
```
test.isnull().sum()
train.isnull().sum()
```

Out[8]:
```
year          0
season        0
month         0
day           0
dayOfWeek     0
hour          0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
count         0
dtype: int64
```

Part 2: finding mean count per year, month, day, working day etc

In [9]:
```
meanY = train.groupby('year', axis = 0).mean()
print(meanY['count'])
```

```
year
2011    144.223349
2012    238.560944
Name: count, dtype: float64
```

In [10]:
```
meanS = train.groupby('season', axis = 0).mean()
print(meanS['count'])
```

```
season
1    116.343261
2    215.251372
3    234.417124
4    198.988296
Name: count, dtype: float64
```

In [11]:
```
meanM = train.groupby('month', axis = 0).mean()
print(meanM['count'])
```

```
month
1      90.366516
2     110.003330
3     148.169811
4     184.160616
5     219.459430
6     242.031798
7     235.325658
8     234.118421
9     233.805281
10    227.699232
11    193.677278
12    175.614035
Name: count, dtype: float64
```

In [12]:
```
meanD = train.groupby('day', axis = 0)['count'].mean()
print(meanD)
```

```
day
1     180.333913
2     183.910995
3     194.696335
4     195.705575
5     189.765217
6     189.860140
7     183.773519
8     179.041812
9     187.897391
10    195.183566
11    195.679577
12    190.675393
13    194.160279
14    195.829268
15    201.527875
16    191.353659
17    205.660870
18    192.605684
19    192.311847
Name: count, dtype: float64
```

In [13]:
```
meanDW = train.groupby('dayOfWeek', axis = 0).mean()
print(meanDW['count'])
```

```
dayOfWeek
0    190.390716
1    189.723847
2    188.411348
3    197.296201
4    197.844343
5    196.665404
6    180.839772
Name: count, dtype: float64
```

In [14]:
```
meanH = train.groupby('hour', axis = 0).mean()
print(meanH['count'])
```

```
hour
0      55.138462
1      33.859031
2      22.899554
3      11.757506
4       6.407240
5      19.767699
6      76.259341
```

```
7     213.116484
8     362.769231
9     221.780220
10    175.092308
11    210.674725
12    256.508772
13    257.787281
14    243.442982
15    254.298246
16    316.372807
17    468.765351
18    430.859649
19    315.278509
20    228.517544
21    173.370614
22    133.576754
23     89.508772
Name: count, dtype: float64
```

In [15]:
```
meanHol = train.groupby('holiday', axis = 0).mean()
print(meanHol['count'])
```

```
holiday
0    191.741655
1    185.877814
Name: count, dtype: float64
```

In [16]:
```
meanWD = train.groupby('workingday', axis = 0).mean()
print(meanWD['count'])
```

```
workingday
0    188.506621
1    193.011873
Name: count, dtype: float64
```

In [17]:
```
meanW = train.groupby('weather', axis = 0).mean()
print(meanW['count'])
```

```
weather
1    205.236791
2    178.955540
3    118.846333
4    164.000000
Name: count, dtype: float64
```

Part 3: Plot count against any 5 features

a) Hypothesis Generation

Before exploring the data to understand the relationship between variables, it is recommended that we focus on hypothesis generation first. Here are some of the hypothesis which could influence the demand of bikes:

Hourly trend:

There must be high demand during office timings. Early morning and late evening can have different trend (cyclist) and low demand during 10:00 pm to 4:00 am.

Daily Trend:

demand of bike increases on weekdays than on weekends and holidays.

Season Trend:

Using of bike can be related with seasons, as people may tend not to use bike in rainy season and winter, but may prefer using bike in summer and spring.

Temperature:

People tends to use more bike when temprature rises.

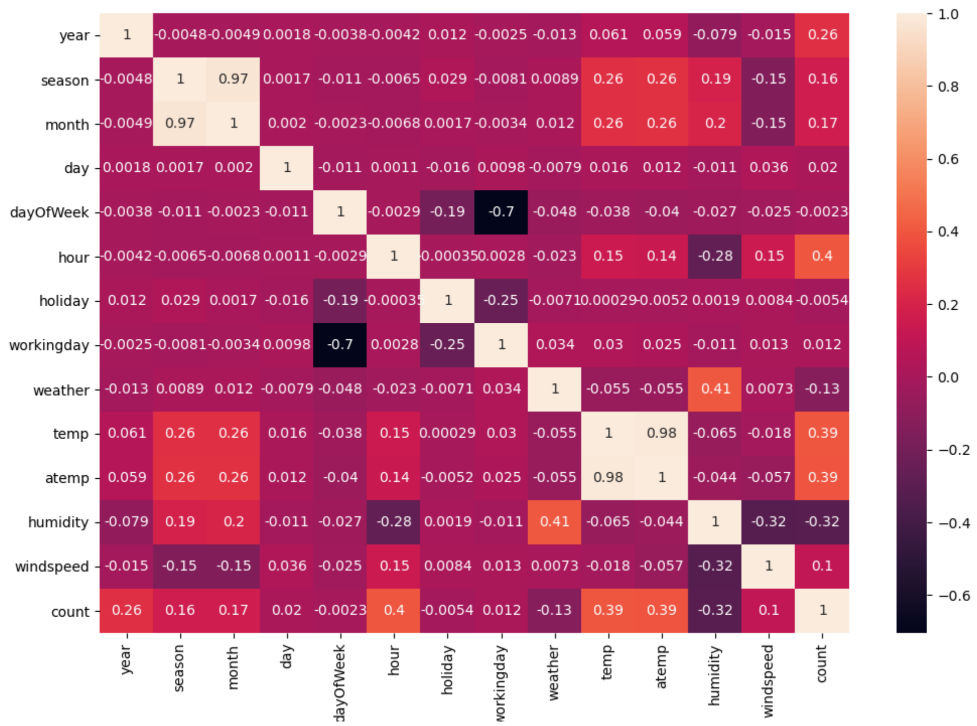
Weather:

in good weather people are more likely to use bike compared to bad weather.

b) Testing our Hypothesis with plots

In [18]:
```
#pd.options.display.float_format = '{3,.2f}'.format
```

In [19]:
```
correlation = train.corr()
plt.figure(figsize = (12, 8))
sns.heatmap(correlation, annot = True)
plt.show()
```

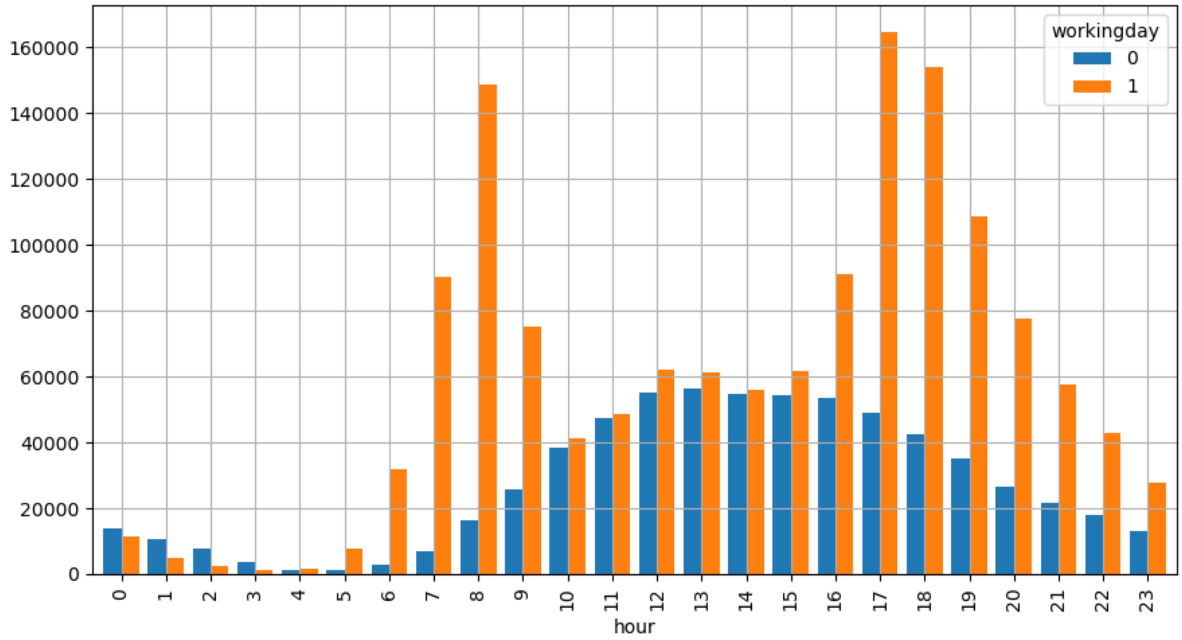


In [20]:
```
by_hour = train.groupby(['hour', 'workingday'])['count'].agg('sum').unstack()
by_hour.head(3)
```

Out[20]:

workingday	0	1
hour		
0	13701	11387
1	10427	4945
2	7686	2573

In [21]:
```
by_hour.plot(kind='bar', figsize=(9,5), width=0.8);
plt.grid(True)
plt.tight_layout()
```



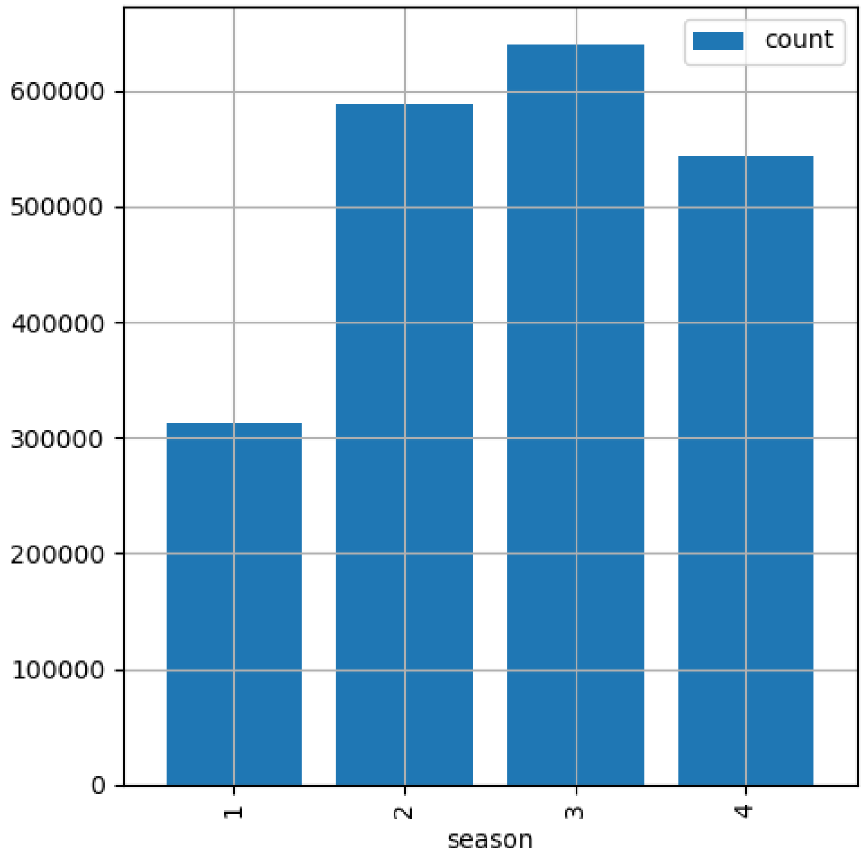
we can clearly see

- on working days peak is at 7-9 in morning and 4 to 8 in evening.
- on non-working days(Holidays, Weekends) peak is very less and occurs 12-4 in noon

In [22]:
```
by_season = train.groupby(['season'])[['count']].agg(sum)
display(by_season)
```

	count
season	
1	312498
2	588282
3	640662
4	544034

In [23]:
```
by_season.plot(kind='bar', figsize=(5,5), width=0.8);
plt.grid(True)
plt.tight_layout()
```

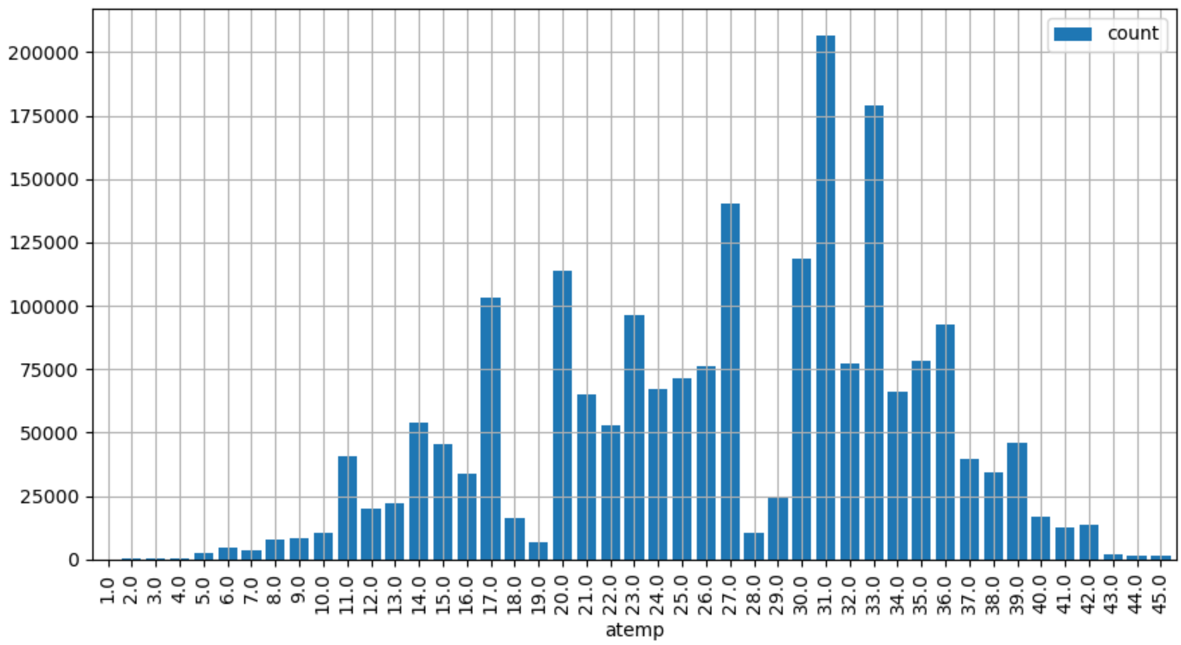


we can clearly see

- according to hypothesis season 1 must be rainy season and the bike demand is fairly less for that season.
- season 2, and season 3 will be spring and summer respectively
- season 4 must be winter

In [24]:
```
x = train.round()
by_temp = x.groupby(['atemp'])[['count']].agg(sum)
```

In [25]:
```
by_temp.plot(kind='bar', figsize=(9,5), width=0.8);
plt.grid(True)
plt.tight_layout()
```

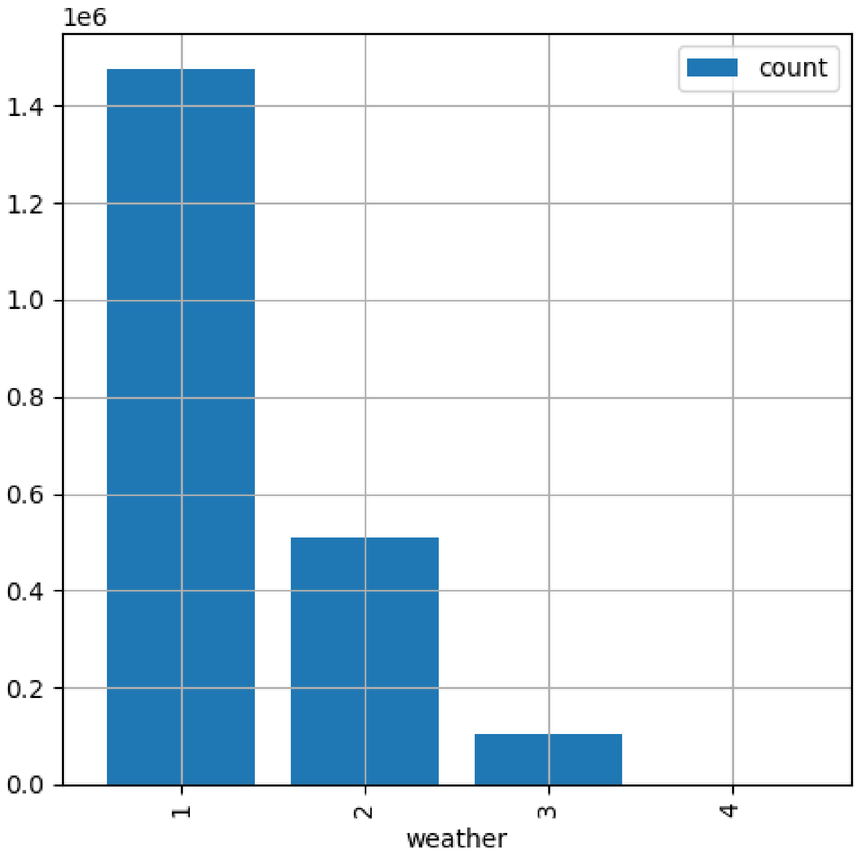


we see here the vehicle rented count increase with increase in temprature till 35 C and than start to decrease.

In [26]:
```
by_weather = train.groupby(['weather'])[['count']].agg(sum)
display(by_season)
```

	count
season	
1	312498
2	588282
3	640662
4	544034

In [27]:
```
by_weather.plot(kind='bar', figsize=(5,5), width=0.8);
plt.grid(True)
plt.tight_layout()
```



we can assume :-

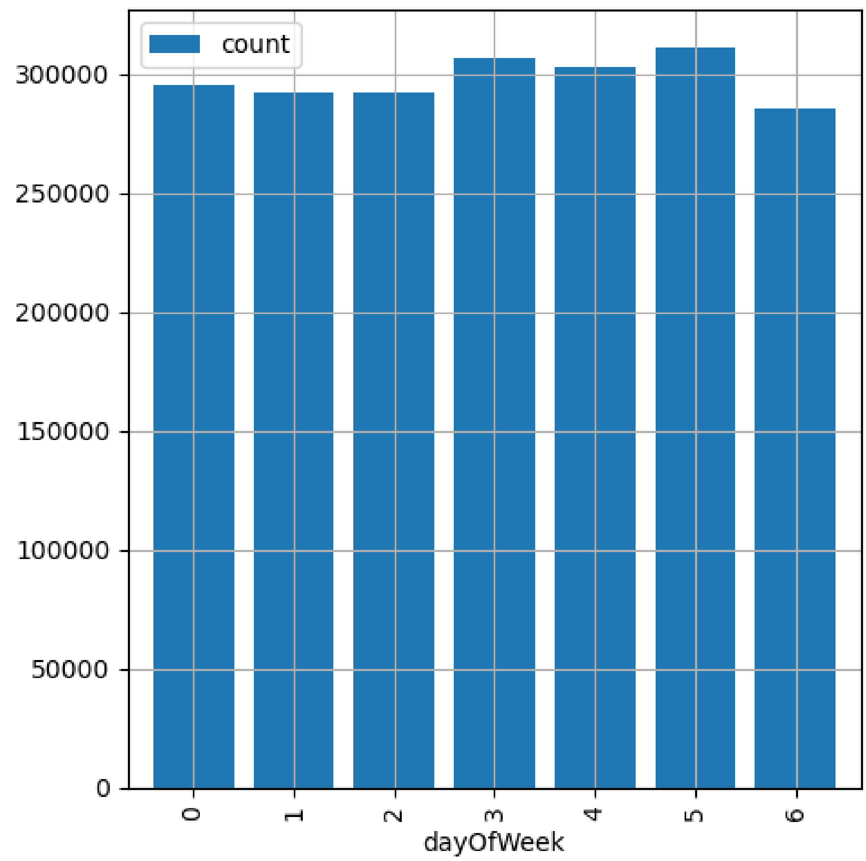
- weather 1 corresponds to: great weather
- weather 2 corresponds to: good weather
- weather 3 corresponds to: bad weather
- weather 4 corresponds to: worse weather

In [28]:
```
by_dayOfWeek = train.groupby(['dayOfWeek'])[['count']].agg(sum)
display(by_dayOfWeek)
```

	count
dayOfWeek	
0	295296

	count
dayOfWeek	
1	291985
2	292226
3	306401
4	302504
5	311518
6	285546

In [29]:
```
by_dayOfWeek.plot(kind='bar', figsize=(5,5), width=0.8);
plt.grid(True)
plt.tight_layout()
```



Clearly we could not stablish any clear relationship between day of week and bike rent count

Part 4: Apply L1 and L2 norm regularization over weight vectors, and find the best hyper-parameter settings for the mentioned problem using validation data and report the accuracy on test data for no regularization, L1 norm regularization and L2 norm regularization.

In [30]:
```
train.head(2)
```

Out[30]:

	year	season	month	day	dayOfWeek	hour	holiday	workingday	weather	temp	atemp	humidity	winds
0	2011	1	1	1	5	0	0	0	1	9.84	14.395	81	
1	2011	1	1	1	5	1	0	0	1	9.02	13.635	80	

In [31]:
```
# Separating the input features and output parameter from the given training data
X_data1 = train[['year','hour','season','weather','temp',
                 'atemp','humidity','windspeed']]
y_data1 = train[['count']]
day = train[['day']]
X_data1.head()
```

Out[31]:

	year	hour	season	weather	temp	atemp	humidity	windspeed
0	2011	0	1	1	9.84	14.395	81	0.0
1	2011	1	1	1	9.02	13.635	80	0.0
2	2011	2	1	1	9.02	13.635	80	0.0
3	2011	3	1	1	9.84	14.395	75	0.0
4	2011	4	1	1	9.84	14.395	75	0.0

In [32]:
```
def isCategory(df):
    NonCategorical = df.loc[:,'temp':'windspeed']
    Catogerical = df.loc[:,'year':'weather']
    return NonCategorical, Catogerical
```

In [33]:
```
def normalize(df):
    result = df.copy()
    for feature_name in df.columns:
        max_value = df[feature_name].max()
        min_value = df[feature_name].min()
        result[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return result
```

In [34]:
```
Xdf = normalize (X_data1)
ydf = normalize(y_data1)
Xdf.head()
```

Out[34]:

	year	hour	season	weather	temp	atemp	humidity	windspeed
0	0.0	0.000000	0.0	0.0	0.224490	0.305068	0.81	0.0
1	0.0	0.043478	0.0	0.0	0.204082	0.288064	0.80	0.0
2	0.0	0.086957	0.0	0.0	0.204082	0.288064	0.80	0.0
3	0.0	0.130435	0.0	0.0	0.224490	0.305068	0.75	0.0
4	0.0	0.173913	0.0	0.0	0.224490	0.305068	0.75	0.0

In [35]:
```
Xdf = pd.concat([Xdf , day ], axis=1)
Xdf.head(2)
```

Out[35]:

	year	hour	season	weather	temp	atemp	humidity	windspeed	day
0	0.0	0.000000	0.0	0.0	0.224490	0.305068	0.81	0.0	1
1	0.0	0.043478	0.0	0.0	0.204082	0.288064	0.80	0.0	1

In [36]:
```
ydf = pd.concat([ydf , day ], axis=1)
ydf.head(2)
```

Out[36]:

	count	day
0	0.015369	1
1	0.039959	1

In [37]:
```
def train_valid_split(data, cutoff_day=15):
    train = data[data['day'] <= cutoff_day]
```

```
    valid = data[data['day'] > cutoff_day]

    return train, valid
```

In [38]:
```
trainX,valX = train_valid_split(Xdf)
trainY,valY = train_valid_split(ydf)
```

In [39]:
```
trainX.pop('day') ;valX.pop('day') ;  trainY.pop('day'); valY.pop('day')
```

Out[39]:
```
348      16
349      16
350      16
351      16
352      16
         ..
10881    19
10882    19
10883    19
10884    19
10885    19
Name: day, Length: 2286, dtype: int64
```

In [40]:
```
print(trainX.shape)
print(valX.shape)
print(trainY.shape)
print(valY.shape)
```

```
(8600, 8)
(2286, 8)
(8600, 1)
(2286, 1)
```

In [41]:
```
# Test data set
testX = test[['year','hour','season','weather',
              'temp','atemp','humidity','windspeed']]
```

In [42]:
```
testX = normalize (testX)
testX.head()
```

Out[42]:

	year	hour	season	weather	temp	atemp	humidity	windspeed
0	0.0	0.000000	0.0	0.0	0.25	0.2273	0.47619	0.464346
1	0.0	0.043478	0.0	0.0	0.25	0.2727	0.47619	0.000000
2	0.0	0.086957	0.0	0.0	0.25	0.2727	0.47619	0.000000
3	0.0	0.130435	0.0	0.0	0.25	0.2576	0.47619	0.196458
4	0.0	0.173913	0.0	0.0	0.25	0.2576	0.47619	0.196458

In [43]:
```
def prediction(w, X):
    yCap = np.exp(np.matmul(X, w))
    return yCap
```

In [44]:
```
def gradient(X, y, yCap):
    gradient = np.divide(np.matmul(np.transpose(X), np.subtract(yCap, y)), len(y))
    return gradient

def G_l1(X, y, yCap, w, reg_const):
    G_l1 = gradient(X, y, yCap)
    n = len(w)
    for i in range(len(w)):
        if w[i,0] > 0:
            G_l1[i,0] += (reg_const / n)
        else:
            G_l1[i,0] -= (reg_const / n)
    return G_l1

def G_l2(X, y, yCap, w, reg_const):
    gradient_values_l2 = gradient(X, y, yCap) + reg_const * w
    return gradient_values_l2 # L2 norm regularized weight parameters
```

In [45]:
```
def gradient_descent(X,y,alpha=0.1,iterations=50000,norm=0,reg_const=0):

    w = np.zeros((len(X[1,:]), 1)) #np.zeros((X.shape[1], 1))

    Loss = []

    for i in range(iterations):
        yCap = prediction(w, X)
        if norm == 1: # L1 Regularization
            gradient_value = G_l1(X, y, yCap, w, reg_const)
            L = float(np.sum(np.subtract(yCap,np.multiply(y,np.matmul(X, w))))+
                      ((reg_const/2)*np.sum(gradient_value)))
        elif norm == 2: # L2 Regularization
            gradient_value = G_l2(X, y, yCap, w, reg_const)
            L = float(np.sum(np.subtract(yCap,np.multiply(y,np.matmul(X, w))))+
                      ((reg_const/2)*np.matmul(gradient_value.T,gradient_value)))
        else:
            gradient_value = gradient(X, y, yCap)
            L = float(np.sum(np.subtract(yCap,np.multiply(y,np.matmul(X, w)))))
        Loss.append(L)

        w = np.subtract(w, alpha * gradient_value)

    return w, Loss
```

In [46]:
```
trainX = trainX.to_numpy()
trainY = trainY.to_numpy()
valX = valX.to_numpy()
valY = valY.to_numpy()
testX = testX.to_numpy()
```

In [47]:
```
w,Loss = gradient_descent(trainX, trainY, iterations = 50000, norm=0)
print('Estimated weight paramters with out regularization are:')
print(w)
```

```
Estimated weight paramters with out regularization are:
[[ 0.21404355]
 [ 0.24316723]
 [ 0.42599567]
 [ 0.22431102]
 [ 1.01124393]
 [-0.85963286]
 [-3.1850456 ]
 [-1.35657141]]
```

In [48]:
```
valY_cap = prediction(w, valX)
valY_cap=pd.DataFrame(valY_cap, columns=['count'])
valY_cap.head(2)
```

Out[48]:

	count
0	0.163206
1	0.137080

In [49]:
```
valY=pd.DataFrame(valY, columns=['count'])
valY.head(2)
```

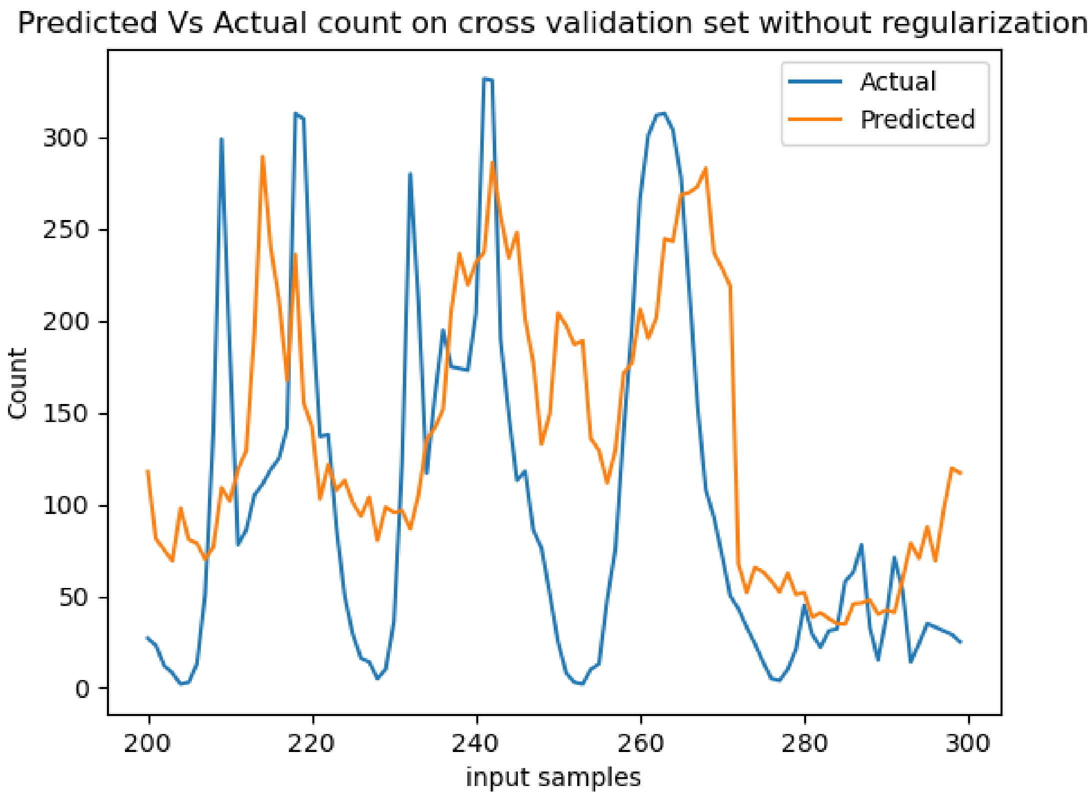
Out[49]:

	count
0	0.038934
1	0.022541

In [50]:
```
df_ds = train.describe()
```

In [51]:
```
valY_act = valY*(df_ds.iloc[7]['count']-df_ds.iloc[3]['count'])+df_ds.iloc[3]['count']
valY_cap_act = valY_cap*(df_ds.iloc[7]['count']-df_ds.iloc[3]['count'])+df_ds.iloc[3]['count']
plt.figure()
```

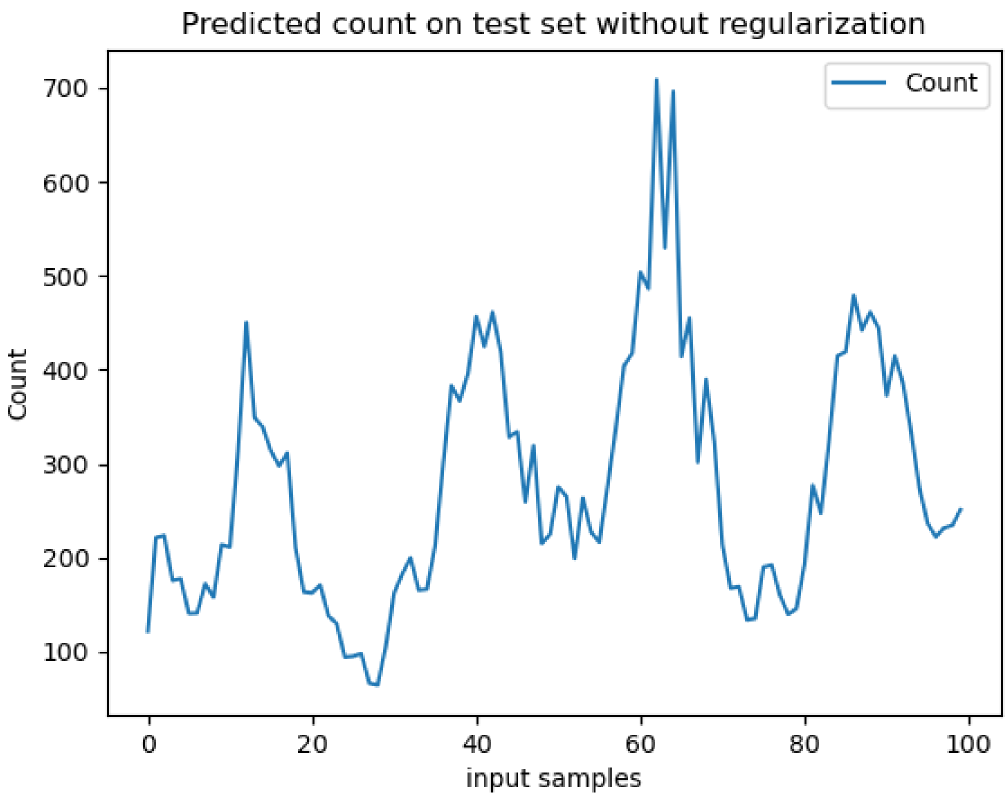
```
plt.plot(valY_act[200:300],label='Actual')
plt.plot(valY_cap_act[200:300],label='Predicted')
plt.title('Predicted Vs Actual count on cross validation set without regularization')
plt.xlabel('input samples')
plt.ylabel('Count')
plt.legend(loc='best')
plt.show()
```



In [52]:
```
rmse_no_reg = np.sqrt(np.mean(np.square(valY_act - valY_cap_act)))
print('RMSE for the cross validation data is: {}'.format(rmse_no_reg))
```

```
RMSE for the cross validation data is: count    167.237206
dtype: float64
```

In [53]:
```
testY_cap = prediction(w, testX)
testY_cap_act = testY_cap*(df_ds.iloc[7]['count']-df_ds.iloc[3]['count'])+df_ds.iloc[3]['count'
plt.figure()
plt.plot(testY_cap_act[:100],label='Count')
plt.title('Predicted count on test set without regularization')
plt.xlabel('input samples')
plt.ylabel('Count')
plt.legend(loc='best')
plt.show()
```



In [54]:
```
y_test_cap = prediction(w, testX)
y_test_cap=pd.DataFrame(valY_cap, columns=['count'])
y_test_cap.head(3)
```

Out[54]:

	count
0	0.163206
1	0.137080
2	0.146613

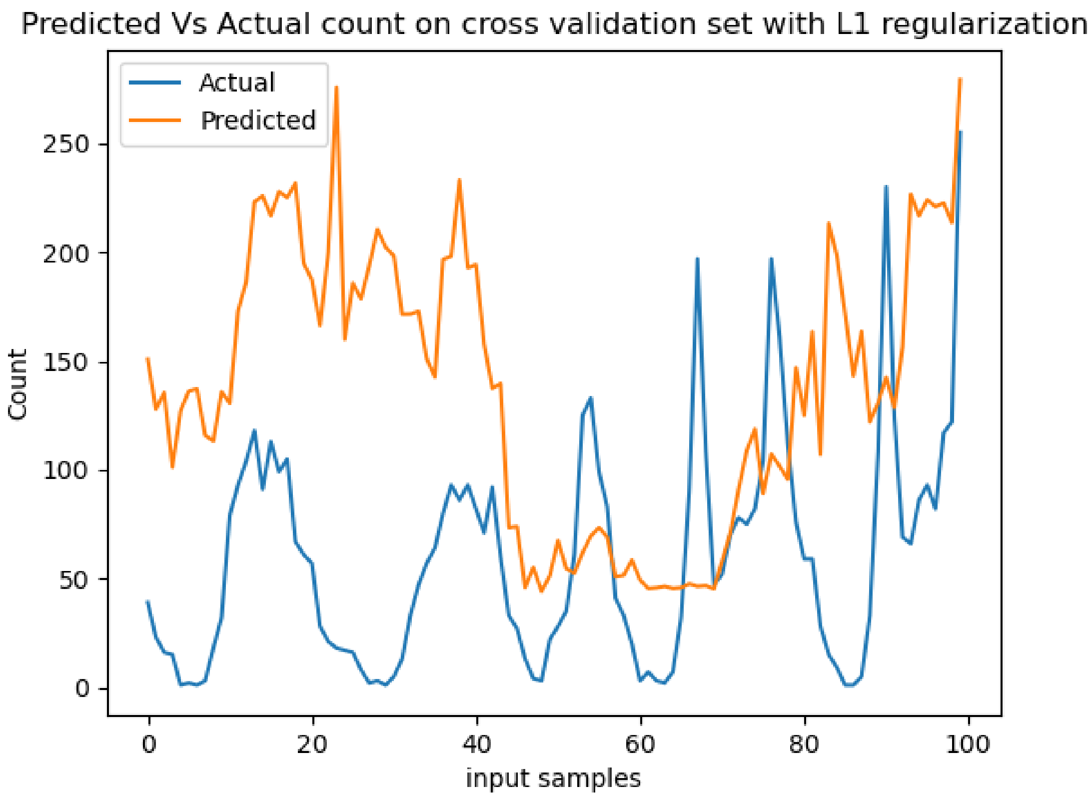
In [55]:
```
reg_const = [ 0.001,0.003,0.01,0.03,0.1]
l1_w = []
for i in reg_const:
    w,Loss = gradient_descent(trainX, trainY ,alpha=5, iterations = 10000, norm=1,reg_const=i)
    print('weight parameters with L1 norm for reg_const {0} are:'.format(i))
    print(w)
    l1_w.append(w)
    valY_cap_l1 = prediction(w, valX)
    rmse_l1_reg = np.sqrt(np.mean(np.square(valY - valY_cap_l1)))
    print('RMSE for reg_const {0} is: {1}'.format(i,rmse_l1_reg))
```

```
weight parameters with L1 norm for reg_const 0.001 are:
[[ 0.13756084]
 [ 0.17326245]
 [ 0.35380469]
 [ 0.18502593]
 [ 1.17136302]
 [-1.14567907]
 [-3.20065862]
 [-1.34837289]]
RMSE for reg_const 0.001 is: count    0.175745
dtype: float64
weight parameters with L1 norm for reg_const 0.003 are:
[[ 0.12967499]
 [ 0.14056389]
 [ 0.34428037]
 [ 0.13566448]
 [ 0.14575541]
 [-0.10610593]
 [-3.21282666]
```

```
 [-1.27404936]]
RMSE for reg_const 0.003 is: count    0.176241
dtype: float64
weight parameters with L1 norm for reg_const 0.01 are:
[[ 1.13724981e-01]
 [ 8.16261962e-02]
 [ 3.08978550e-01]
 [ 7.20648982e-03]
 [-1.86425156e-03]
 [ 1.56404057e-03]
 [-3.08147557e+00]
 [-1.06623545e+00]]
RMSE for reg_const 0.01 is: count    0.17687
dtype: float64
weight parameters with L1 norm for reg_const 0.03 are:
[[-0.0178845 ]
 [-0.06247101]
 [ 0.07311545]
 [-0.00929195]
 [-0.00760109]
 [-0.21243629]
 [-2.96979858]
 [-0.73719493]]
RMSE for reg_const 0.03 is: count    0.194238
dtype: float64
weight parameters with L1 norm for reg_const 0.1 are:
[[ 1.59828800e-01]
 [-4.45385697e-03]
 [ 2.61974792e-01]
 [ 8.06420203e-04]
 [ 9.34871926e-02]
 [ 3.09488507e-02]
 [-2.31354227e+00]
 [ 1.46025150e-02]]
RMSE for reg_const 0.1 is: count    0.231962
dtype: float64
```

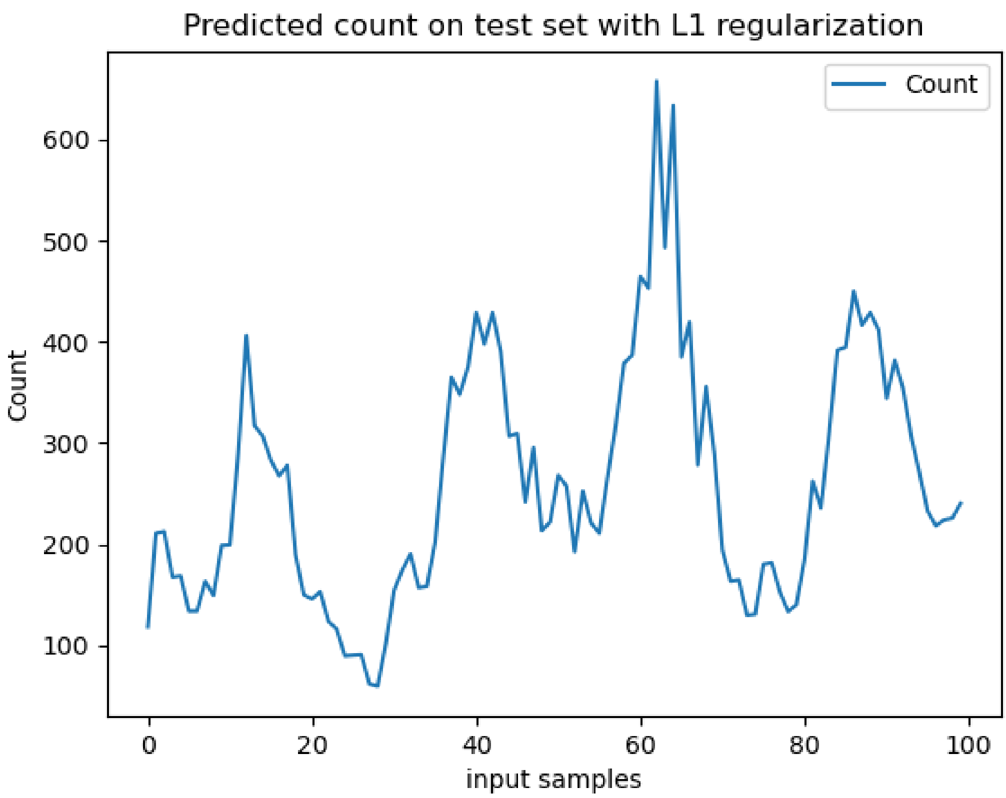
In [56]:

```
valY_cap_l1 = prediction(l1_w[0], valX)
valY_act_l1 = valY*(df_ds.iloc[7]['count']-df_ds.iloc[3]['count'])+df_ds.iloc[3]['count']
valY_cap_act_l1 = valY_cap_l1*(df_ds.iloc[7]['count']-df_ds.iloc[3]['count'])+df_ds.iloc[3]['co
plt.figure()
plt.plot(valY_act_l1[:100],label='Actual')
plt.plot(valY_cap_act_l1[:100],label='Predicted')
plt.title('Predicted Vs Actual count on cross validation set with L1 regularization')
plt.xlabel('input samples')
plt.ylabel('Count')
plt.legend(loc='best')
plt.show()
```



In [68]:

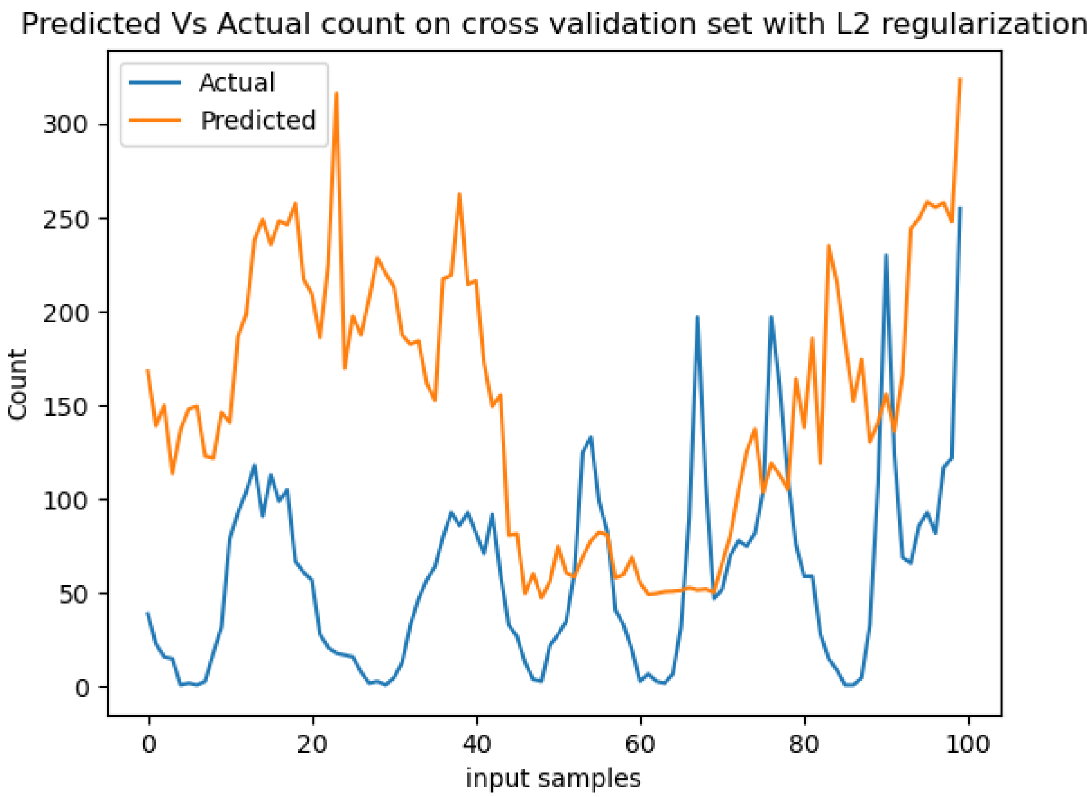
```
testY_cap_l1 = prediction(l1_w[0], testX)
testY_cap_act_l1 = testY_cap_l1*(df_ds.iloc[7]['count']-df_ds.iloc[3]['count'])+df_ds.iloc[3]['
plt.figure()
plt.plot(testY_cap_act_l1[:100],label='Count')
plt.title('Predicted count on test set with L1 regularization')
plt.xlabel('input samples')
plt.ylabel('Count')
plt.legend(loc='best')
plt.show()
```



In [58]:
```
reg_const = [0.0001,0.001,0.003,0.01]
l2_w = []
for i in reg_const:
    w,Loss = gradient_descent(trainX, trainY, iterations = 10000, norm=2,reg_const=i)
    print('weight parameters with L1 norm for reg_const {0} are:'.format(i))
    print(w)
    l2_w.append(w)
    plt.plot(Loss,label=i)
    valY_cap = prediction(w, valX)
    rmse_l2_reg = np.sqrt(np.mean(np.square(valY - valY_cap)))
    print('RMSE for reg_const {0} is {1}'.format(i,rmse_l2_reg))
```

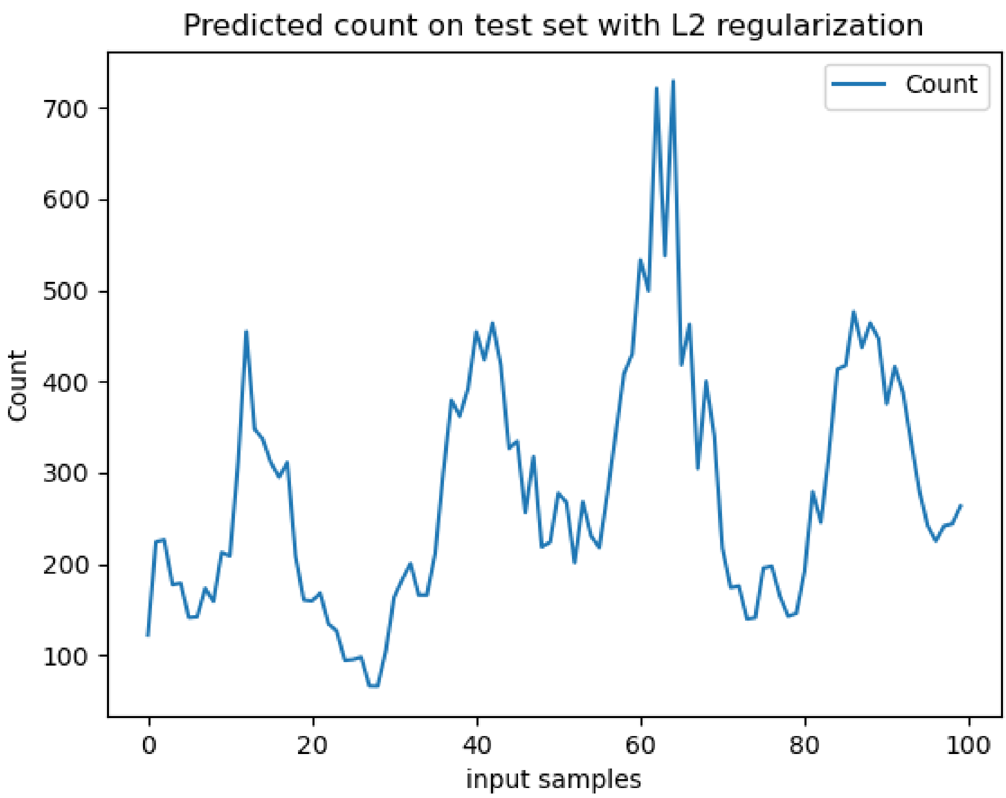
```
weight parameters with L1 norm for reg_const 0.0001 are:
[[ 0.20657005]
 [ 0.22163597]
 [ 0.41675576]
 [ 0.20410437]
 [ 0.33957369]
 [-0.1831982 ]
 [-3.1917347 ]
 [-1.31855227]]
RMSE for reg_const 0.0001 is count    0.171402
dtype: float64
weight parameters with L1 norm for reg_const 0.001 are:
[[ 0.17920981]
 [ 0.16270002]
 [ 0.34999295]
 [ 0.08596776]
 [ 0.22775037]
 [-0.16433602]
 [-2.96201196]
 [-1.15797432]]
RMSE for reg_const 0.001 is count    0.171923
dtype: float64
weight parameters with L1 norm for reg_const 0.003 are:
[[ 0.13277592]
 [ 0.07143345]
 [ 0.24156871]
 [-0.06464141]
 [ 0.08056864]
 [-0.17058134]
 [-2.58996833]
 [-0.93241396]]
RMSE for reg_const 0.003 is count    0.174865
dtype: float64
weight parameters with L1 norm for reg_const 0.01 are:
[[ 0.03567657]
 [-0.08657984]
 [ 0.04097541]
 [-0.21003999]
 [-0.11706019]
 [-0.24646499]
 [-1.91250394]
 [-0.61252694]]
RMSE for reg_const 0.01 is count    0.186601
dtype: float64
```

In [59]:
```
valY_cap_l2 = prediction(l2_w[0], valX)
valY_act_l2 = valY*(df_ds.iloc[7]['count']-df_ds.iloc[3]['count'])+df_ds.iloc[3]['count']
valY_cap_act_l2 = valY_cap_l2*(df_ds.iloc[7]['count']-df_ds.iloc[3]['count'])+df_ds.iloc[3]['co
plt.figure()
plt.plot(valY_act_l2[:100],label='Actual')
plt.plot(valY_cap_act_l2[:100],label='Predicted')
plt.title('Predicted Vs Actual count on cross validation set with L2 regularization')
plt.xlabel('input samples')
plt.ylabel('Count')
plt.legend(loc='best')
plt.show()
```



In [60]:

```
testY_cap = prediction(l2_w[0], testX)
testY_cap_act = testY_cap*(df_ds.iloc[7]['count']-df_ds.iloc[3]['count'])+df_ds.iloc[3]['count'
plt.figure()
plt.plot(testY_cap_act[:100],label='Count')
plt.title('Predicted count on test set with L2 regularization')
plt.xlabel('input samples')
plt.ylabel('Count')
plt.legend(loc='best')
plt.show()
```



Part 5: most important features determining count of bikes rented

we can see this by plotting correlation of predicted values for rented bikes with other features.

In [61]:
```
testX_pd = pd.DataFrame(testX, columns=['year','hour','season','weather','temp','atemp','humidi
testX_pd.head(3)
```

Out[61]:

	year	hour	season	weather	temp	atemp	humidity	windspeed
0	0.0	0.000000	0.0	0.0	0.25	0.2273	0.47619	0.464346
1	0.0	0.043478	0.0	0.0	0.25	0.2727	0.47619	0.000000
2	0.0	0.086957	0.0	0.0	0.25	0.2727	0.47619	0.000000

In [62]:
```
test_pred = pd.DataFrame(testY_cap_act, columns=['count'])
test_pred.head(3)
```

Out[62]:

	count
0	121.853314
1	224.220604
2	226.382033

In [63]:
```
test_pred.shape
```

Out[63]:
```
(6493, 1)
```

In [64]:
```
testX_pd.shape
```

Out[64]:
```
(6493, 8)
```

In [65]:
```
test_pred = pd.concat([test_pred ,testX_pd], axis=1)
test_pred.head(2)
```

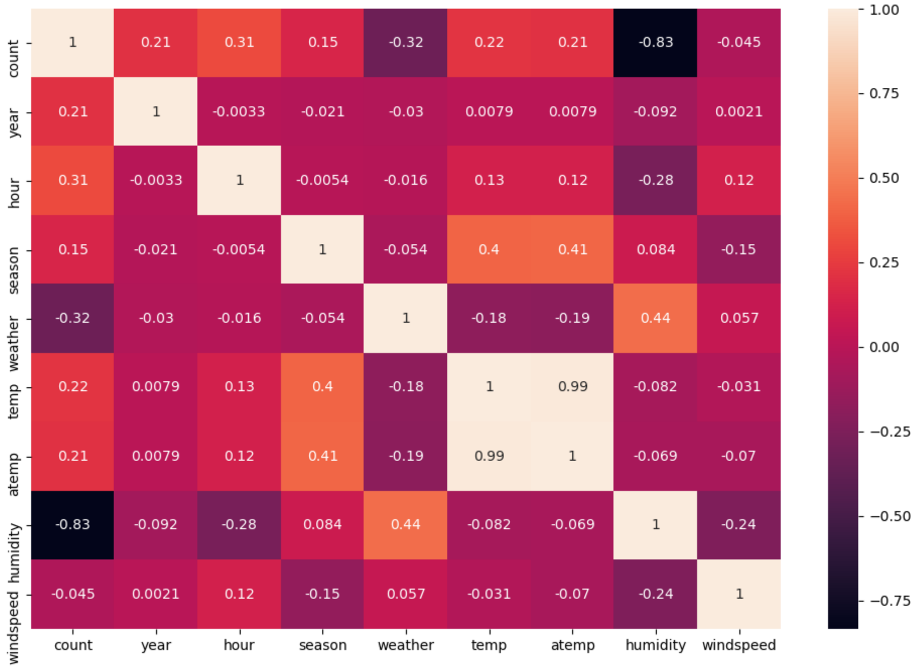
Out[65]:

	count	year	hour	season	weather	temp	atemp	humidity	windspeed
0	121.853314	0.0	0.000000	0.0	0.0	0.25	0.2273	0.47619	0.464346
1	224.220604	0.0	0.043478	0.0	0.0	0.25	0.2727	0.47619	0.000000

In [66]:
```
test_pred.isnull().sum()
```

Out[66]:
```
count        0
year         0
hour         0
season       0
weather      0
temp         0
atemp        0
humidity     0
windspeed    0
dtype: int64
```

In [67]:
```
corr = test_pred.corr()
plt.figure(figsize = (12, 8))
sns.heatmap(corr, annot = True)
plt.show()
```



here we clearly see that features that are most usefull in determining bike rented count are humidity, weather, hour

---

Completed by:- Shubham jain(SM20MTECH12007) and Rahul bidla(SM20MTECH12014)