# Question 4

November 4, 2020

# 1 CS5590: Foundations of Machine Learning

## 1.1 Assignment 1

## 1.2 Question 4

## 1.3 Authors



| Name | Roll number |
|---|---|
| Raj Patil | CS18BTECH11039 |
| Karan Bhukar | CS18BTECH11021 |



[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random
```

[2]:
```python
d = pd.read_csv('hour.csv');d.info()
# no null vals
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   instant     17379 non-null  int64
 1   dteday      17379 non-null  object
 2   season      17379 non-null  int64
 3   yr          17379 non-null  int64
 4   mnth        17379 non-null  int64
 5   hr          17379 non-null  int64
 6   holiday     17379 non-null  int64
 7   weekday     17379 non-null  int64
 8   workingday  17379 non-null  int64
 9   weathersit  17379 non-null  int64
 10  temp        17379 non-null  float64
 11  atemp       17379 non-null  float64
 12  hum         17379 non-null  float64
```

```
 13  windspeed   17379 non-null  float64
 14  casual      17379 non-null  int64
 15  registered  17379 non-null  int64
 16  cnt         17379 non-null  int64
dtypes: float64(4), int64(12), object(1)
memory usage: 2.3+ MB
```

[3]:
```
d.iloc[0]
```

[3]:
```
instant                1
dteday        2011-01-01
season                 1
yr                     0
mnth                   1
hr                     0
holiday                0
weekday                6
workingday             0
weathersit             1
temp                0.24
atemp             0.2879
hum                 0.81
windspeed              0
casual                 3
registered            13
cnt                   16
Name: 0, dtype: object
```

### 1.3.1 subquestion 2 and 3

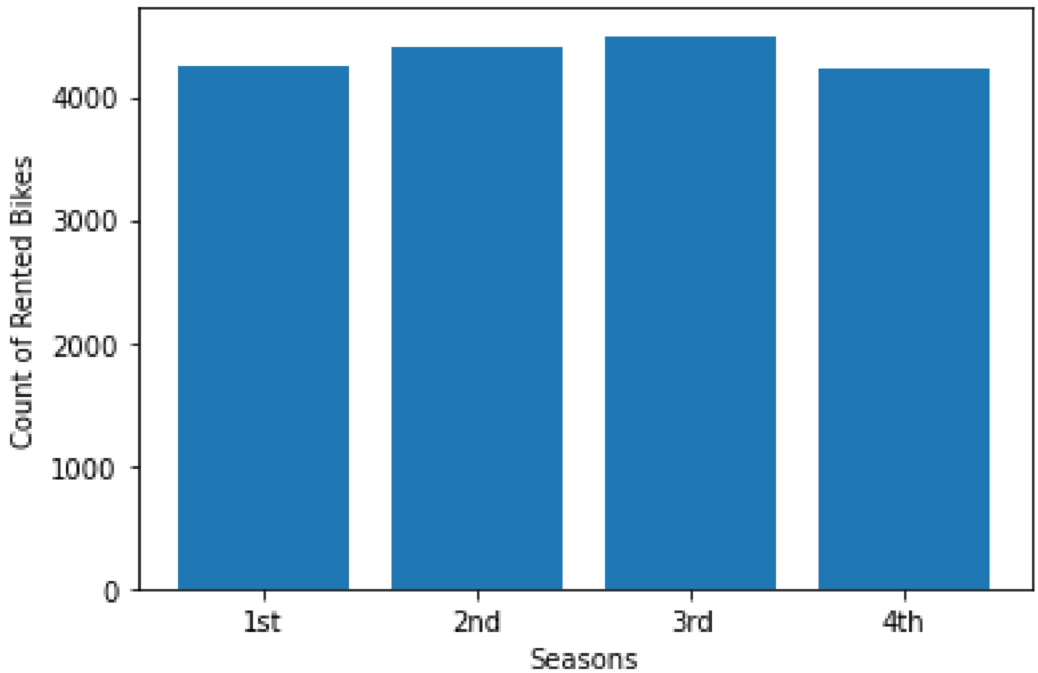
[46]:
```
def plot(feature,d):
  if feature == 'season':
    X=["1st","2nd","3rd","4th"]
    Y=[0,0,0,0]
    for i in range(d.iloc[:,2].shape[0]):
       Y[d.iloc[i,2]-1] += 1
    mean = [i/d.iloc[:,2].shape[0] for i in Y]
    for i,me in enumerate(mean):
      print("mean of count for season {} is = {}".format(i+1,me))
    plt.xlabel("Seasons")
    plt.ylabel("Count of Rented Bikes")
    plt.bar(X,Y)

  elif feature == 'yr':
    X=["1st","2nd"]
    Y=[0,0]
    for i in range(d.iloc[:,3].shape[0]):
```

```
        Y[d.iloc[i,3]] += 1
    mean = [i/d.iloc[:,3].shape[0] for i in Y]
    for i,me in enumerate(mean):
      print("mean of count for year {} is = {}".format(i,me))
    plt.xlabel("year")
    plt.ylabel("Count of Rented Bikes")
    plt.bar(X,Y)

  elif feature == 'month':
    X=[1,2,3,4,5,6,7,8,9,10,11,12]
    Y=[0,0,0,0,0,0,0,0,0,0,0,0]
    month = ['January', 'February', 'March', 'April', 'May', 'June', 'July',␣
↪'August', 'September', 'October', 'November', 'December']
    for i in range(d.iloc[:,4].shape[0]):
        Y[d.iloc[i,4]-1] += 1
    mean = [i/d.iloc[:,4].shape[0] for i in Y]
    for i,me in enumerate(mean):
      print("mean of count for month {} is = {}".format(i+1,me))
    plt.xlabel("Month of the year")
    plt.ylabel("Count of Rented Bikes")
    plt.xticks(X, month, rotation ='vertical')
    plt.bar(X,Y)

  elif feature == 'holiday':
    X=["No","Yes"]
    Y=[0,0]
    for i in range(d.iloc[:,6].shape[0]):
        Y[d.iloc[i,6]] += 1
    mean = [i/d.iloc[:,6].shape[0] for i in Y]
    for i,me in enumerate(mean):
      print("mean of count for holiday {} is = {}".format(i,me))
    plt.xlabel("Holiday")
    plt.ylabel("Count of Rented Bikes")
    plt.bar(X,Y)

  elif feature == 'weekday':
    X=["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"]
    Y=[0,0,0,0,0,0]
    for i in range(d.iloc[:,7].shape[0]):
        Y[d.iloc[i,7]-1] += 1
    mean = [i/d.iloc[:,7].shape[0] for i in Y]
    for i,me in enumerate(mean):
      print("mean of count for weekday {} is = {}".format(i+1,me))
    plt.xlabel("Day of the week",labelpad=20)
    plt.ylabel("Count of Rented Bikes")
    plt.bar(X,Y)
```

```
    plt.show()
```

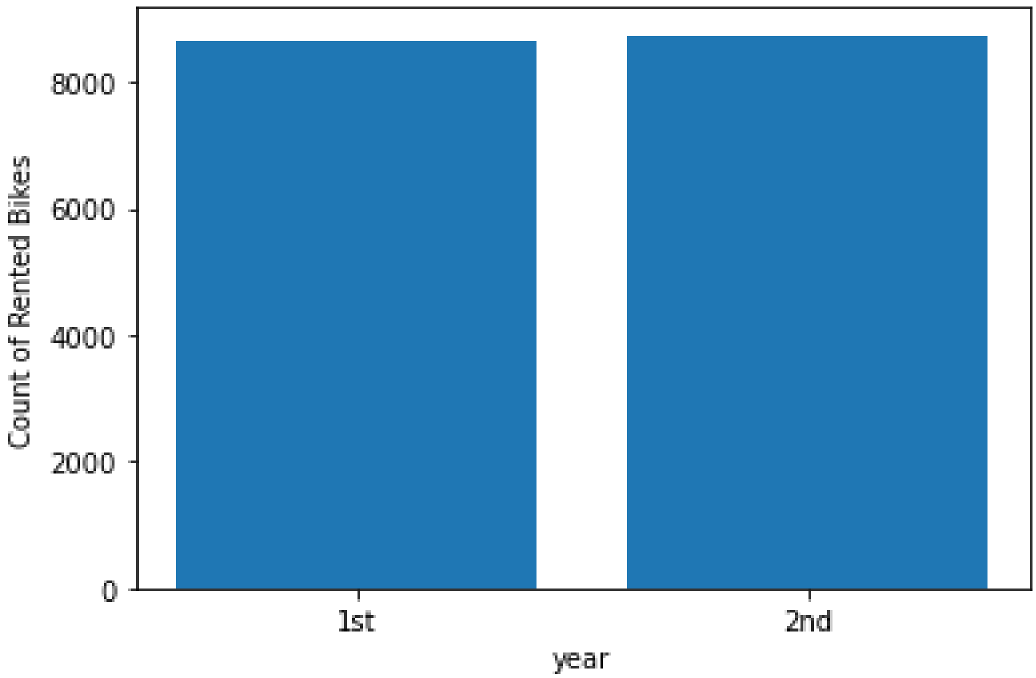
[47]:
```
plot("season",d)
```

```
mean of count for season 1 is = 0.2440876920421198
mean of count for season 2 is = 0.25369699062086426
mean of count for season 3 is = 0.25870303239541975
mean of count for season 4 is = 0.24351228494159619
```



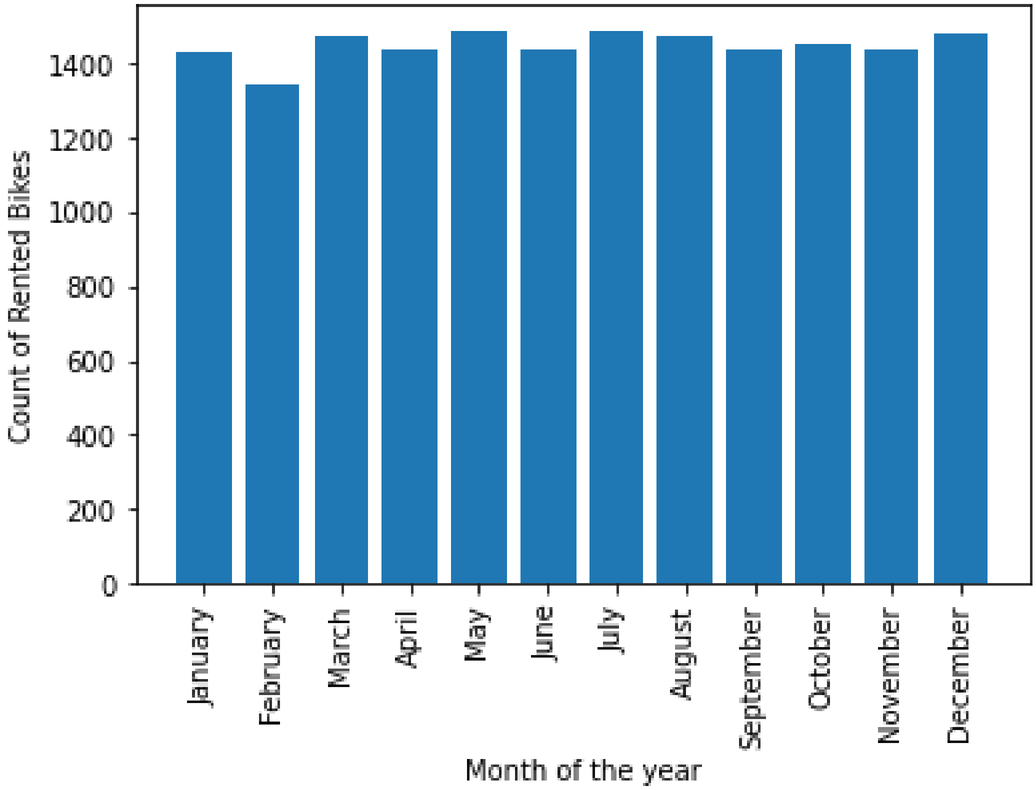

[48]:
```
plot("yr",d)
```

```
mean of count for year 0 is = 0.4974394384026699
mean of count for year 1 is = 0.5025605615973301
```



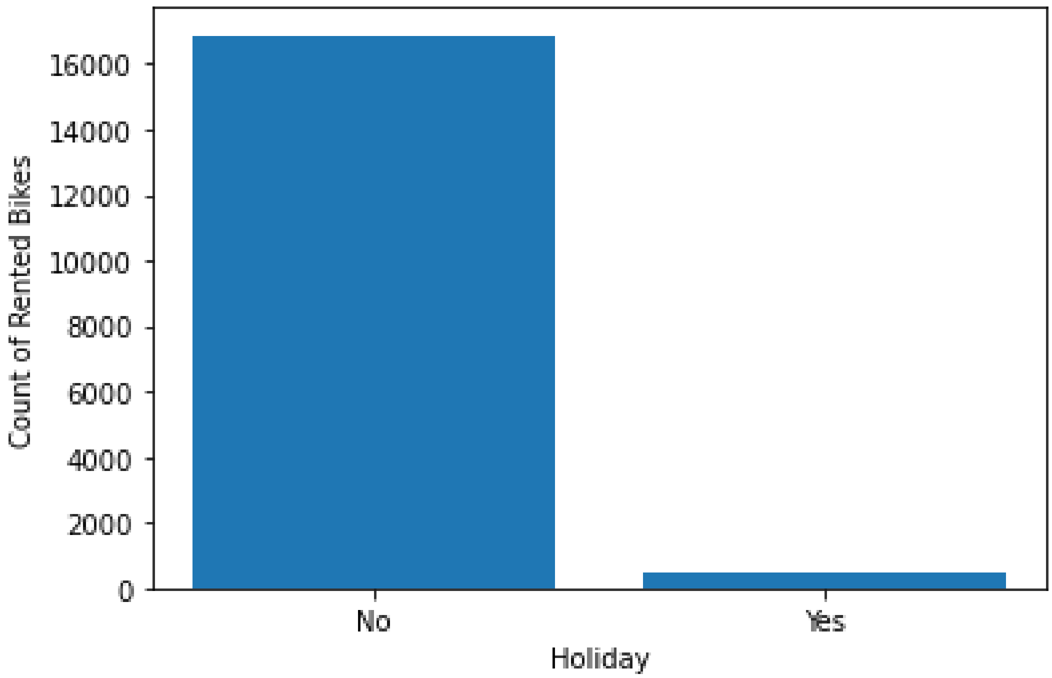

[49]:
```
plot("month",d)
```

```
mean of count for month 1 is = 0.08222567466482536
mean of count for month 2 is = 0.07716209218021751
mean of count for month 3 is = 0.08475746590712929
mean of count for month 4 is = 0.08268600034524426
mean of count for month 5 is = 0.08562057655791472
mean of count for month 6 is = 0.08285862247540135
mean of count for month 7 is = 0.08562057655791472
mean of count for month 8 is = 0.08487254732723402
mean of count for month 9 is = 0.08268600034524426
mean of count for month 10 is = 0.08349157028597733
mean of count for month 11 is = 0.08268600034524426
mean of count for month 12 is = 0.08533287300765291
```



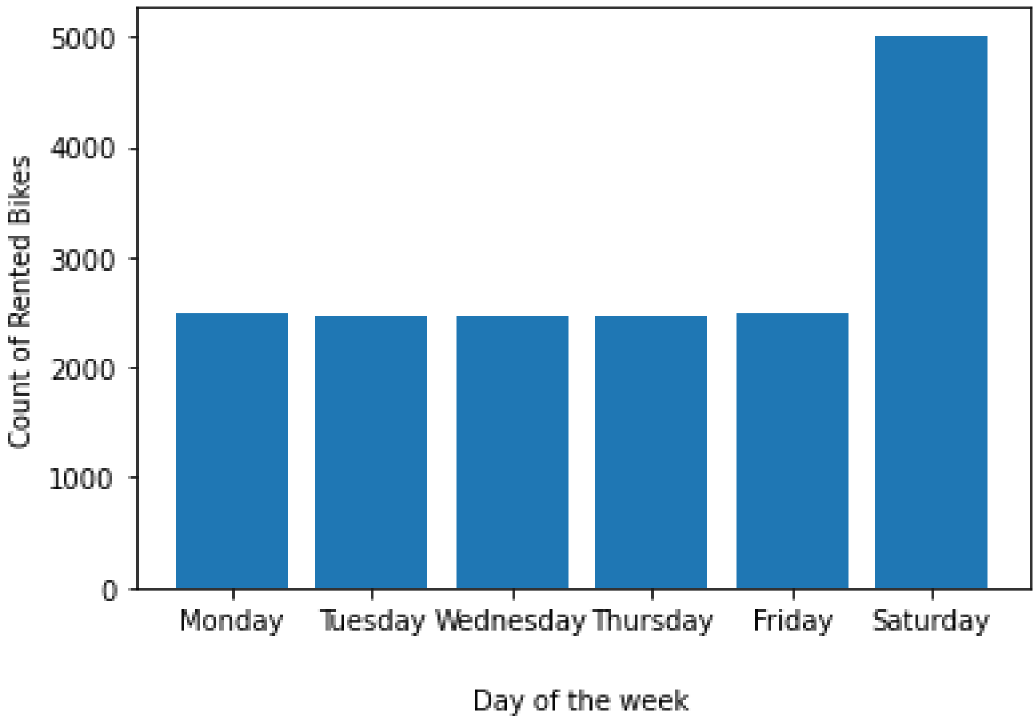

[50]:
```
plot("holiday",d)
```

```
mean of count for holiday 0 is = 0.971229644973819
mean of count for holiday 1 is = 0.028770355026181024
```




[51]:
```
plot("weekday",d)
```

```
mean of count for weekday 1 is = 0.1426434202198055
mean of count for weekday 2 is = 0.1411473617584441
mean of count for weekday 3 is = 0.14241325737959606
mean of count for weekday 4 is = 0.14218309453938663
mean of count for weekday 5 is = 0.1431037459002244
mean of count for weekday 6 is = 0.2885091202025433
```




### 1.4 cleaning up data

[4]:
```
data = d.drop(['instant','casual','registered'],axis=1)
# casual and registered are not features : they are certain predictors of count
# probably meant to be used in the anomaly detection challenge
# sticking to zero-indexed data
```

### 1.5 splitting data:

using a phase indicator: train as 0, val as 1 and test as 2

[5]:
```
data['split'] = 0
```

[6]:
```
# data.loc[data['dteday'].str[-2:].astype(int) < 20,'split'] = 0  #train split
data.loc[data['dteday'].str[-2:].astype(int) >= 20,'split'] = 2 # test split
assert len(data) == data[data['split']==0]['split'].
 ↪count()+data[data['split']==2]['split'].count()
```

[7]:
```
# ## validation split
# choosing 20 percent of the train data to be validation
# starting after the first 100 records as validating right away makes no sense
val_candidates = [i for i in range(100,len(data)) if data.iloc[i]['split']==0 ]
val_chosen = random.sample(val_candidates,int(0.2*(len(val_candidates) +100)))
```

```
data.iloc[val_chosen,-1] = 1
```

[8]:
```
# ## some final processing
data = data.drop('dteday',axis=1) # not needed anymore
```

[9]:
```
norm  = ['weathersit','season','mnth','hr','weekday']
# ones to be normalized
# could check a one hot-encoded version of season as well
# but they follow an order so going with this for now
for n in norm:
    data.loc[:,n] = data[n]/max(data[n])
```

main routines have been well documented in the model.py module in the same directory proceeding with analysis in the notebook from now on

### 1.6 this is the report with experiments

### 1.7 check model.py for code : it is self-explanatory(well-commented)

# 2 Poisson regression

- modelling the rate per hour to the be parameter of a poisson model
- as the count is always positive, modelling the parameter as a log linear model

hence for the parameters $W_{d-vect}$ and features $X_{d-vect}$, the probability density is given as:

$$f(y) = \frac{\lambda^y \cdot e^{-\lambda}}{y!}$$

where

$$\lambda = e^{W^T X}$$

## 2.1 some nuances

- we will have to use an iterative method for optimization as a closed form solution won't be possible, due to the exponential term in the log-likelihood
- using the negative-log likelihood as the loss function
- using an lr $\eta$
- for L1 and L2 regression, introducing hyper-parameters $\alpha$ and $\beta$ respectively and these will be altered in the validation phase via grid search
- only the previous data can be used for prediction hence we will have to update the weight vector intermittently
- proceeding with a batch size of 1
- will have modes for processing : train,val and test
- will differentiate w.r.t $W$ in the train mode
- will differentiate w.r.t hyper-parameter in val mode
- will only output prediction in test mode
- normalizing some features to the train set and fitting the test to it while prediction, for better performance and also for the last sub-question

# 3 subquestion 1

# 4 Likelihood and Loss function

for an $X, y$ pair: the likelihood will be

$$\mathcal{L}(W) = \frac{e^{-\lambda}\lambda^y}{y!}$$

$$\therefore ln(\mathcal{L}(W)) = yln(\lambda) - \lambda - ln(y!)$$

ignoring the constant factorial from here onwards

$$ln(\mathcal{L}(W)) = yW^TX - e^{W^TX}$$

employing the loss function to be the negative log likelihood

$$\therefore L(loss) = e^{W^TX} - yW^TX$$

also note the derivative:

$$\underset{W}{\nabla} L = e^{W^TX}X - yX = X(e^{W^TX} - y)$$

## 4.1 L1 regularization

for $\alpha$ as hyper-parameter
we have an additional term of $\alpha|W|$ with the loss function

$$\underset{W}{\nabla} L_{L1} = (existing\ term) \cdots + \alpha\nabla\|W\|$$

$$\|W\| = (\|W\|^2)^{\frac{1}{2}}$$

$$\therefore \nabla\|W\| = \frac{1}{2}\|W\|^{2^{\frac{-1}{2}}} \cdot 2W = \frac{W}{\|W\|}$$

overall(for training phase)

$$\therefore \underset{W}{\nabla} L_{L1} = X(e^{W^TX} - y) + \alpha\frac{W}{\|W\|}$$

## 4.2 L2 regularization

for $\beta$ as hyper-parameter
we have an additional term of $\beta\|W\|^2$ with the loss function

$$\underset{W}{\nabla} L_{L2} = (existing\ term) \cdots + \beta\nabla\|W\|^2$$

overall(for training phase)

$$\therefore \underset{W}{\nabla} L_{L2} = X(e^{W^TX} - y) + 2\beta W$$

## 4.3 Experiments

[60]:
```
import importlib
import model
```

[61]:
```
importlib.reload(model)
# for quick retesting
```

[61]:
```
<module 'model' from '/mnt/c/leisure and imp docs/BTECH CSE 4 yrs/3rd
year/sem5/Foundations of Machine Learning/fml-assignment-1/Question 4/model.py'>
```

[103]:
```
base = model.BaseModel(0.0001,13,data,-1,-2);
L1 = model.L1Model(0.0001,13,data,-1,-2,0.0001);
L2 = model.L2Model(0.0001,13,data,-1,-2,0.0001);
```

Notes: - note that the question demands that we use only the previous data to predict what comes next - for this we predict whenever we encounter a test split index right away - also, we are processing the index one by one ( batch-size=1 ) - validating at random times from the train set (except the first 100 train split indices)

check model.py : it has been completely documented

### 4.4 full pass on dataset

base model

[104]:
```
base.full_pass()
```

```
100%|          | 17379/17379 [00:13<00:00, 1327.98it/s]
```

L1 model

[105]:
```
L1.full_pass()
```

```
100%|          | 17379/17379 [00:13<00:00, 1298.18it/s]
```

L2 model

[106]:
```
L2.full_pass()
```

```
100%|          | 17379/17379 [00:13<00:00, 1327.64it/s]
```

### 4.5 errors

[107]:
```
print('base error:', base.test_rms())
print('L1 error  :', L1.test_rms())
print('L2 error  :', L2.test_rms())
```

```
base error: 214.5731040277971
L1 error  : 155.5828397399973
L2 error  : 159.57408948642873
```

## 4.6 note the performance improvement with the same lr with regularization

checking final W:

[108]:
```
base.W
```

[108]:
```
array([ 3.24786636,  3.04850391, -0.99508023, -0.70369008,  0.52856336,
       -0.21777155,  0.14183319,  0.03915093, -0.20839004,  1.41613996,
        0.88802293, -1.11314864,  0.92164374])
```

[109]:
```
L1.W
```

[109]:
```
array([ 3.05861708,  0.19296796,  0.16451219,  0.97007019,  0.53058912,
       -0.18821904,  0.18368329,  0.08141973, -0.09701366,  1.11878637,
        1.41180134, -1.13397959,  1.11854914])
```

[110]:
```
L2.W
```

[110]:
```
array([ 3.69706063,  0.33118782,  0.0797629 ,  0.10201965,  0.54241072,
       -0.08303447,  0.24122738,  0.08712386, -0.07443048,  2.16669895,
        0.40780129, -1.00287435,  1.2805619 ])
```

### 4.6.1 full disclosure:

- the grid search in our model.py is not completely functional and so has been commented out for now

## 4.7 subquestion 5

### 4.7.1 for checking which features are the most important: looking at the absolute value of the weights

[142]:
```
features = list(data.columns[:12])
features
```

[142]:
```
['season',
 'yr',
 'mnth',
 'hr',
 'holiday',
 'weekday',
 'workingday',
 'weathersit',
 'temp',
 'atemp',
 'hum',
 'windspeed']
```

using the weights from the selection operator for now:

[129]:
```python
weights = L1.W
```

[131]:
```python
weights
# 0 is the bias
```

[131]:
```
array([ 3.05861708,  0.19296796,  0.16451219,  0.97007019,  0.53058912,
       -0.18821904,  0.18368329,  0.08141973, -0.09701366,  1.11878637,
        1.41180134, -1.13397959,  1.11854914])
```

note the large values for index number 3,4,9,10,11,12

these can be used to infer the most prominent features

[144]:
```python
prominent = [features[i-1] for i in [3,4,9,10,11,12]]
prominent
```

[144]:
```
['mnth', 'hr', 'temp', 'atemp', 'hum', 'windspeed']
```

Note that these make sense at the first look: - more bikers later in the year than before(for month) - temperature: people get out on a sunny day - feeling temperature: as expected - this is a stronger predictor than temperature - hum : less people get out on humid days - windspeed : who doesn't like to ride on windy days…

[ ]:
```python

```