**Bike Rental Prediction**

**Problem Description**: Bike sharing system is an important part of many tourist cities, where customers, mainly tourists rent bike from one station and can return it to other station, making the system fully automatic.

The inflow of tourists as well as other potential customer depends on many factors like weather, is it weekend or holiday etc. So does the bike rents on a particular day depends on these factors.

**Aim**: Our Aim is to create a model, which when fed with weather, holiday and other information can predict number of bikes that would be rented out.

**Data Set**:

Two-year historical log corresponding to years 2011 and 2012 from Capital Bikeshare system, Washington D.C., USA which is publicly available in <http://capitalbikeshare.com/system-data>. We aggregated the data on two hourly and daily basis and then extracted and added the corresponding weather and seasonal information. Weather information are extracted from [http://www.freemeteo.com](http://www.freemeteo.com/).



**Dataset Columns:**

Both hour.csv and day.csv have the following fields, except hr, which is not available in day.csv

**Instant**: record index

**dteday**: date

**Season**: season (1: spring, 2: summer, 3: fall, 4: winter)

**yr**: year (0: 2011, 1:2012)

**mnth** : month (1 to 12)

**hr**: hour (0 to 23)

**holiday** : weather day is holiday or not (extracted from <http://dchr.dc.gov/page/holiday-schedule>) weekday

**workingday** : if day is neither weekend nor holiday is 1, otherwise is 0.

**Weathersit** : 1: Clear, Few clouds, Partly cloudy, Partly cloudy

2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist

3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

**temp** : Normalized temperature in Celsius. The values are divided to 41 (max)

**atemp**: Normalized feeling temperature in Celsius. The values are divided to 50 (max)

**hum**: Normalized humidity. The values are divided to 100 (max)

**windspeed**: Normalized wind speed. The values are divided to 67 (max)

**casual**: count of casual users

**registered**: count of registered users

**cnt**: count of total rental bikes including both casual and registered

**Solution**:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

We are going to use above libraries for the solution. **Numpy** is used for the calculations and **pandas** is used for the creation of dummy variables in our algorithm. **Matplotlib** is used for the plotting of graphs.

data\_path = 'Bike-Sharing-Dataset/hour.csv'

data= pd.read\_csv(data\_path)

data.head()

The above code is to load the data and read it correctly. We are checking with 5 columns of the data either from hourly data or day data.

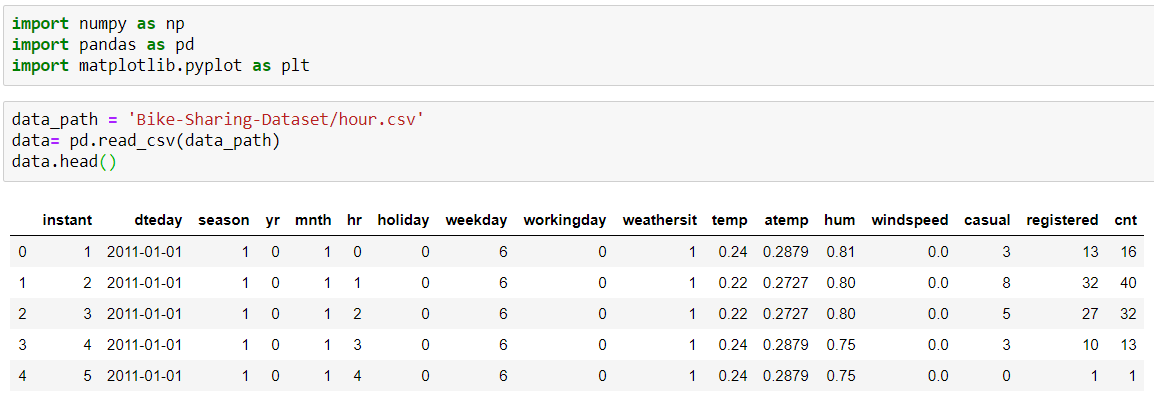


Fig (1): Screenshot from Jupyter Notebook

We can check the data by plotting to find the average no of bikes rented over a period. The bikes rented are of two categories. They are casual and registered. The total count is represented by the column cnt.

fields= ['season', 'weathersit', 'mnth', 'hr', 'weekday']

for x in fields:

dummies = pd.get\_dummies(data[x], prefix=x, drop\_first=False)

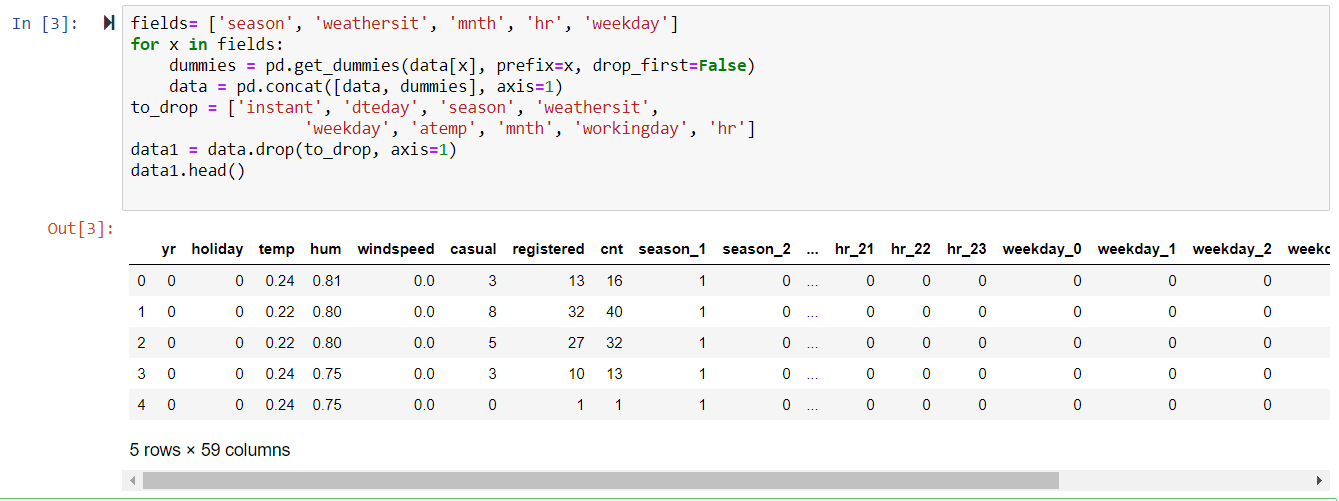
data = pd.concat([data, dummies], axis=1)

to\_drop = ['instant', 'dteday', 'season', 'weathersit',

'weekday', 'atemp', 'mnth', 'workingday', 'hr']

data1 = data.drop(to\_drop, axis=1)

data1.head()

We have some categorical data for the variables 'season', 'weathersit', 'mnth', 'hr', 'weekday', which is not good for training the network so we are going to convert into the binary data. This is done by the pandas library which consists of method get dummies. After this the no of columns is increased to 59 and u can see the results in the below screenshot. . Fig (2): Screenshot from Jupyter Notebook

quant\_features = ['casual', 'registered', 'cnt', 'temp', 'hum', 'windspeed']

scaled= {}

for x in quant\_features:

mean, standard = data1[x].mean(), data1[x].std()

scaled[x] = [mean, standard]

data1.loc[:, x] = (data1[x] - mean)/standard

We are scaling the data variables in such a way that the variables data has mean of zero and standard deviation of 1 as it is continuous data it is tough to train and to standardize the scaling of data has been done.

data\_t= data[-21\*24:]

data1 = data1[:-21\*24]

target\_fields = ['cnt', 'casual', 'registered']

features, targets = data1.drop(target\_fields, axis=1), data1[target\_fields]

test\_features, test\_targets = data\_t.drop(target\_fields, axis=1), data\_t[target\_fields]

train\_features, train\_targets = features[:-60\*24], targets[:-60\*24]

val\_features, val\_targets = features[-60\*24:], targets[-60\*24:]



Fig (3): Screenshot from Jupyter Notebook

We are splitting the data into three sets. One is training set, validation set and test set. As our aim is to predict the number of rental bikes in an hour or day we are setting aside ‘cnt', 'casual', 'registered’ output variables and rest of the variables as inputs. As the data is continuous time series data, we need to make sure that the data we have trained which is historical data needs to predict future data. So here historical data is train \_features and train \_targets and future data (val\_features, val\_targets) is for validating the trained data set. At the end after training and validating the neural network once stabilization is achieved, we are going to test the neural network with the test data set. The test data set is last 21 days from the data set and validation data is the last 60 days after removal of test data and rest of the data set is the training data set.

from neuralnetwork import NNBike

def MSE(y, Y):

return np.mean((y-Y)\*\*2)

We have implemented the neural network and we have imported it. In this Neural network we have defined functions train , forwardpass , backpropagation, Weightupdation, run, and also declared some of the constants as iterations , learning rate,hidden nodes,output nodes. This is MLP Model with back propagation.

import numpy as np

iterations = 5000

lr = 0.5

HiddenNodes = 20

OutputNodes = 1

class NNBike(object):

def \_\_init\_\_(self, InputNodes, HiddenNodes, OutputNodes, lr):

# Number of Input,Hidden and Output nodes

self.InputNodes = InputNodes

self.HiddenNodes = HiddenNodes

self.OutputNodes = OutputNodes

#Randomly intializing weights

self.Input\_Weights=np.random.normal(0.0,self.InputNodes\*\*-0.5,(self.InputNodes, self.HiddenNodes))

self.Hidden\_Weights=np.random.normal(0.0,self.HiddenNodes\*\*-0.5,(self.HiddenNodes,

self.OutputNodes))

self.lr = lr

#Sigmoid function activation

self.activation\_function = lambda x : 1/(1 + np.exp(-x))

def train(self, features, targets):

#Train network to implement forwardpass and backpropagation

noofrecords = features.shape[0]

delta\_weights\_i\_h = np.zeros(self.Input\_Weights.shape)

delta\_weights\_h\_o = np.zeros(self.Hidden\_Weights.shape)

for X, y in zip(features, targets):

final\_outputs, hidden\_outputs = self.forwardpass(X)

delta\_weights\_i\_h, delta\_weights\_h\_o = self.backpropagation(final\_outputs, hidden\_outputs, X,

y,delta\_weights\_i\_h, delta\_weights\_h\_o)

self.Weightupdation(delta\_weights\_i\_h, delta\_weights\_h\_o, noofrecords )

def forwardpass(self, X):

hidden\_inputs = np.dot(X, self.Input\_Weights)

hidden\_outputs = self.activation\_function(hidden\_inputs)

final\_inputs = np.dot(hidden\_outputs, self.Hidden\_Weights)

final\_outputs = final\_inputs

return final\_outputs, hidden\_outputs

def backpropagation(self, final\_outputs, hidden\_outputs, X, y, delta\_weights\_i\_h, delta\_weights\_h\_o):

# Based on error value of weights is changed here

error = y - final\_outputs

hidden\_error = np.dot(self.Hidden\_Weights, error)

output\_error\_term = error

hidden\_error\_term = hidden\_error \* hidden\_outputs \* (1 - hidden\_outputs)

delta\_weights\_i\_h += hidden\_error\_term \* X[:,None]

delta\_weights\_h\_o += output\_error\_term \* hidden\_outputs[:,None]

return delta\_weights\_i\_h, delta\_weights\_h\_o

def Weightupdation(self, delta\_weights\_i\_h, delta\_weights\_h\_o, noofrecords ):

self.Hidden\_Weights += self.lr\*delta\_weights\_h\_o/noofrecords

self.Input\_Weights += self.lr\*delta\_weights\_i\_h/noofrecords

def run(self, features):

hidden\_inputs = np.dot(features, self.Input\_Weights)

hidden\_outputs = self.activation\_function(hidden\_inputs)

final\_inputs = np.dot(hidden\_outputs, self.Hidden\_Weights)

final\_outputs = final\_inputs

return final\_outputs

**Neural Network Diagram**:

A close up of a logo

Description automatically generated

Fig (4): MPL model for bike rental

We have implemented the above neural network

import sys

from neuralnetwork import iterations, lr, HiddenNodes, OutputNodes

N\_i = train\_features.shape[1]

network = NNBike(N\_i, HiddenNodes, OutputNodes, lr)

losses = {'train':[], 'validation':[]}

for ii in range(iterations):

batch = np.random.choice(train\_features.index, size=128)

X, y = train\_features.ix[batch].values, train\_targets.ix[batch]['cnt']

network.train(X, y)

train\_loss = MSE(network.run(train\_features).T, train\_targets['cnt'].values)

val\_loss = MSE(network.run(val\_features).T, val\_targets['cnt'].values)

sys.stdout.write("\rProgress: {:2.1f}".format(100 \* ii/float(iterations)) \

+ "% ... Training loss: " + str(train\_loss)[:5] \

+ " ... Validation loss: " + str(val\_loss)[:5])

sys.stdout.flush()

losses['train'].append(train\_loss)

losses['validation'].append(val\_loss)

As we have splitted the data into three sets, we are using the training set and validation set for training and achieving Mean square error very less in both the cases and updating the weights through gradient descent.As there is lot of data we are training with the batch of values in training to make the network to train quickly and reduce the resource utilization and to increase the performance.

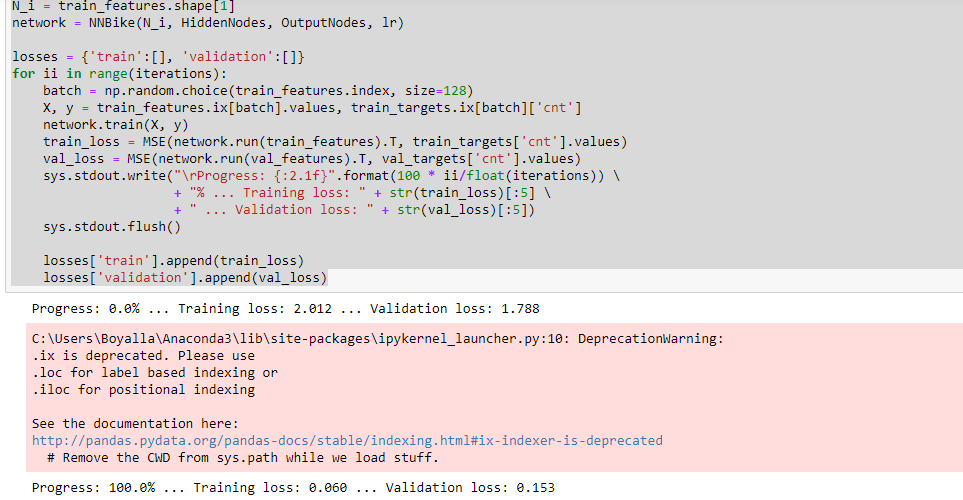


Fig (5): Training of the network.

We have plotted the graphs for both the training loss and validation loss ,which are in the range of 0.1



Fig (6): Graph representing training and validation loss.

fig, ax = plt.subplots(figsize=(8,4))

mean, std = scaled['cnt']

predictions = network.run(test\_features).T\*std + mean

ax.plot(predictions[0], label='Prediction')

ax.plot((test\_targets['cnt']\*std + mean).values, label='Data')

ax.set\_xlim(right=len(predictions))

ax.legend()

dates = pd.to\_datetime(data.ix[data\_t.index]['dteday'])

dates = dates.apply(lambda d: d.strftime('%b %d'))

ax.set\_xticks(np.arange(len(dates))[12::24])

\_ = ax.set\_xticklabels(dates[12::24], rotation=45)

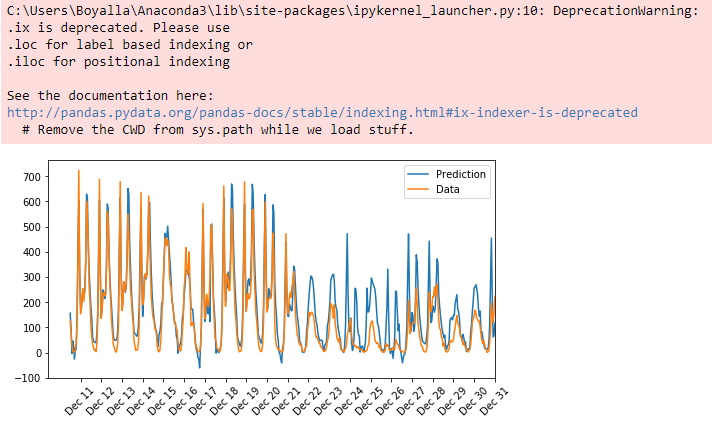


Fig (7): Graph representing predicted versus actual data.

We are testing the trained network with the test data kept aside and we have plotted the test predictions with the actual value.