Assignment for Week 5 - Neural Network

Task 1:

Neural Networks

Bike Share data: https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset

Train a regression neural net to predict the amount of bike rentals each hour.
Try at least 5 different architectures, pick the best one and defend your choice

The following sites might be good references for you:

- https://www.springboard.com/blog/beginners-guide-neural-network-in-python-scikit-learn-0-18/
- https://datascience.stackexchange.com/questions/36049/how-to-adjust-the-hyperparameters-of-mlp-classifier-to-get-more-perfect-performa

In [22]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# sklearn packages
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
# plotting
import seaborn as sns

# will show plots without doing plt.show()
%matplotlib inline
```

Task 1

In [4]:
```
# Read dataset to pandas dataframe
bike = pd.read_csv('data_bike/hour.csv')
```

In [5]:
```
bike.head()
```

Out[5]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0	1	1

In [6]:
```
bike.info()
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
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
```

In [7]:
```
bike.describe(include='all')
```

Out[7]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit
count	17379.0000	17379	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000
unique	NaN	731	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	2012-06-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	24	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	8690.0000	NaN	2.501640	0.502561	6.537775	11.546752	0.028770	3.003683	0.682721	1.425283
std	5017.0295	NaN	1.106918	0.500008	3.438776	6.914405	0.167165	2.005771	0.465431	0.639357
min	1.0000	NaN	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	4345.5000	NaN	2.000000	0.000000	4.000000	6.000000	0.000000	1.000000	0.000000	1.000000
50%	8690.0000	NaN	3.000000	1.000000	7.000000	12.000000	0.000000	3.000000	1.000000	1.000000
75%	13034.5000	NaN	3.000000	1.000000	10.000000	18.000000	0.000000	5.000000	1.000000	2.000000
max	17379.0000	NaN	4.000000	1.000000	12.000000	23.000000	1.000000	6.000000	1.000000	4.000000

In [8]:
```
bike.shape
```

Out[8]: (17379, 17)

Dropping instant, year, month. I am dropping them because instant is the row number, and as for year and month I will extract them from the date for more accurate data

In [9]:
```
bike.drop('instant',axis=1, inplace=True)
bike.drop('yr',axis=1, inplace=True)
bike.drop('mnth',axis=1, inplace=True)
```

In [10]:
```
bike.info()
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   dteday      17379 non-null  object
 1   season      17379 non-null  int64
 2   hr          17379 non-null  int64
 3   holiday     17379 non-null  int64
 4   weekday     17379 non-null  int64
 5   workingday  17379 non-null  int64
 6   weathersit  17379 non-null  int64
 7   temp        17379 non-null  float64
 8   atemp       17379 non-null  float64
 9   hum         17379 non-null  float64
 10  windspeed   17379 non-null  float64
 11  casual      17379 non-null  int64
 12  registered  17379 non-null  int64
 13  cnt         17379 non-null  int64
dtypes: float64(4), int64(9), object(1)
memory usage: 1.9+ MB
```

Because date column is not float or integer, model will not be fitted. So, here I am creating 3 columns for year, month, and day. Then dropping Date column

In [11]:
```
# we need to convert dteday to int

new = bike["dteday"].str.split("-", n = 2, expand = True)
bike["year"]= new[0]
bike["month"]= new[1]
bike["day"]= new[2]
```

In [12]:
```
bike['year'] = bike['year'].astype(str).astype(int)
bike['month'] = bike['month'].astype(str).astype(int)
bike['day'] = bike['day'].astype(str).astype(int)
```

In [13]:
```
bike.drop('dteday',axis=1, inplace=True)
```

In [14]:
```
bike.info()
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   season      17379 non-null  int64
 1   hr          17379 non-null  int64
 2   holiday     17379 non-null  int64
 3   weekday     17379 non-null  int64
 4   workingday  17379 non-null  int64
 5   weathersit  17379 non-null  int64
 6   temp        17379 non-null  float64
 7   atemp       17379 non-null  float64
 8   hum         17379 non-null  float64
 9   windspeed   17379 non-null  float64
 10  casual      17379 non-null  int64
 11  registered  17379 non-null  int64
 12  cnt         17379 non-null  int64
 13  year        17379 non-null  int32
 14  month       17379 non-null  int32
 15  day         17379 non-null  int32
dtypes: float64(4), int32(3), int64(9)
memory usage: 1.9 MB
```

In [16]:
```
#gather up names of all the columns
cols = bike.columns

#set the prediction column and the feature columns for KNN
prediction_col = 'cnt'
feature_cols = [c for c in cols if c != prediction_col]

x = bike[feature_cols]
y = bike[prediction_col]

#split the dataset into the train and test data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=41)
```

Scaling the feature on train data.

In [17]:
```
scaler = StandardScaler()
scaler.fit(x_train)

x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

Fitting the model with 2,2,2 hidden layers parameters.

In [32]:
```
mlp = MLPRegressor(hidden_layer_sizes=(2, 2, 2), max_iter=1000)
mlp.fit(x_train, y_train.values.ravel())
```

Out[32]: MLPRegressor(hidden_layer_sizes=(2, 2, 2), max_iter=1000)

In [33]:
```
predictions = mlp.predict(x_test)
```

In [34]:
```
print('mean absolute error:',mean_absolute_error(y_test,predictions))
print('mean squared error:',mean_squared_error(y_test,predictions))
print('R2 score: ',r2_score(y_test,predictions))
```

```
mean absolute error: 0.017015492710373545
mean squared error: 0.0008032808792437198
R2 score:  0.9999999750607013
```

MAE is nearly 0, which means there very small difference between predictions and actuall data
MSE is 7.2
R2 is 0.99

the model is performing very well

Fitting the modell with 22, 22, 22.

In [35]:
```
## model 2

mlp = MLPRegressor(hidden_layer_sizes=(22, 22, 22), max_iter=1000)
mlp.fit(x_train, y_train.values.ravel())
```

Out[35]: MLPRegressor(hidden_layer_sizes=(22, 22, 22), max_iter=1000)

In [36]:
```
predictions = mlp.predict(x_test)
print(predictions)
```

```
[  1.50419527  24.29178038 220.99803113 ...  18.85106158 208.93251813
 180.88469715]
```

In [37]:
```
print('mean absolute error:',mean_absolute_error(y_test,predictions))
print('mean squared error:',mean_squared_error(y_test,predictions))
print('R2 score: ',r2_score(y_test,predictions))
```

```
mean absolute error: 0.12044428044712849
mean squared error: 0.026754377033169765
R2 score:  0.9999991693622744
```

model 2 is better than model 1 but not much because model 1 is nearly perfect.

Fitting the model with 26,26,26 parameters for hidden layer.

In [39]:
```
## model 3

mlp = MLPRegressor(hidden_layer_sizes=(26, 26, 26), max_iter=1000)
mlp.fit(x_train, y_train.values.ravel())
```

Out[39]: MLPRegressor(hidden_layer_sizes=(26, 26, 26), max_iter=1000)

In [40]:
```
predictions = mlp.predict(x_test)
print(predictions)
```

```
[  2.167329    24.07533459 220.73112195 ...  19.03452361 208.72922809
 180.93503591]
```

In [41]:
```
print('mean absolute error:',mean_absolute_error(y_test,predictions))
print('mean squared error:',mean_squared_error(y_test,predictions))
print('R2 score: ',r2_score(y_test,predictions))
```

```
mean absolute error: 0.14414083811118644
mean squared error: 0.033963204389794874
R2 score:  0.9999989455512714
```

For the third model the accuracy reduced by 0.000001.

All models perform very well. I can't chose one model over the other except for the computational power and efficiency.

References

https://www.geeksforgeeks.org/python-pandas-split-strings-into-two-list-columns-using-str-split/

https://stackoverflow.com/questions/39173813/pandas-convert-dtype-object-to-int