

Experiment No. 1: Linear regression using deep neural network on Boston housing dataset.

```
from keras.callbacks import ModelCheckpoint

from keras.models import Sequential

from keras.layers import Dense, Activation, Flatten

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_absolute_error

from matplotlib import pyplot as plt

import seaborn as sb

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import warnings

warnings.filterwarnings('ignore')

warnings.filterwarnings('ignore', category=DeprecationWarning)

from xgboost import XGBRegressor


#processing of Dataset


def get_data():

    #get train data

    train_data_path = 'train.csv'

    train = pd.read_csv(train_data_path)


    #get test data
```

```
test_data_path ='test.csv'
```

```
test = pd.read_csv(test_data_path)
```

```
return train, test
```

```
def get_combined_data ():
```

```
#reading train data
```

```
train, test = get_data()
```

```
target = train.SalePrice
```

```
train.drop (['SalePrice'],axis = 1, inplace = True)
```

```
combined=train.append(test)
```

```
combined.reset_index(inplace=True)
```

```
combined.drop(['index', 'Id'], inplace=True, axis=1)
```

```
return combined, target
```

```
#Load train and test data into pandas DataFrames
```

```
train_data, test_data=getdata()
```

```
#Combine train and test data to process them together
```

```
combined, target = get_combined_data()
```

```
# define a function to get the columns that don't have any missing values
```

```
def get_cols_with_no_nans(df,col_type):
```

“

Arguments :

df: The dataframe to process

col_type:

num : to only get numerical columns with no nans

no_num : to only get non-numerical columns with no nans

all : to get any columns with no nans

”

```
if (col_type == 'num'):
```

```
    predictors = df.select_dtypes(exclude=['object'])
```

```
elif (col_type == 'no_num'):
```

```
    predictors = df.select_dtypes(include=['object'])
```

```
elif (col_type == 'all'):
```

```
    predictors = df
```

```
else
```

```
    print('Error : choose a type (num, no_num, all)')
```

```
    return 0
```

```
    cols_with_no_nans = []
```

```
    for col in predictors.columns:
```

```
        if not df[col].isnull().any():
```

```
            cols_with_no_nans.append(col)
```

```
    return cols_with_no_nans
```

```
# Get the columns that do not have any missing values.
```

```
num_cols = get_cols_with_no_nans(combined, 'num')
```

```
cat_cols = get_cols_with_no_nans(combined, 'no_num')
```

```
# Let's see how many columns we got

print (Number of numerical columns with no nan values:, len(num_cols))

print ('Number of nun-numerical columns with no nan values:', len(cat_cols))

[out]:

Number of numerical columns with no nan values : 25

Number of nun-numerical columns with no nan values : 20
```

```
#One Hot Encode The Categorical Features
```

```
def oneHotEncode(df,colNames):

for col in colNames:

if( df[col].dtype == np.dtype('object')):

dummies = pd.get_dummies(df[col],prefix=col)

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

```
#drop the encoded column

dfdrop([col],axis = 1, inplace=True)

return df
```

```
print("There were {} columns before encoding categorical
features'.format(combined.shape[1]))
```

```
combined = oneHotEncode(combined, cat_cols)

print('There are {} columns after encoding categorical features'.format(combined.shape[1]))
```

```
[out]:

There were 45 columns before encoding categorical features

There are 149 columns after encoding categorical features
```

```
#split back combined dataframe to training data and test data
```

```
def split_combined ():
```

```
    global combined
```

```
    train = combined[:1460]
```

```
    test = combined [1460:]
```

```
    return train, test
```

```
    train, test = split_combined()
```

```
#Making the Deep Neural Network
```

```
NN_model = Sequential()
```

```
# The Input Layer :
```

```
NN_model.add(Dense(128, kernel_initializer='normal', input_dim = train.shape [1],  
activation='relu'))
```

```
# The Hidden Layers:
```

```
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
```

```
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
```

```
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
```

```
# The Output Layer:
```

```
NN_model.add (Dense(1, kernel_initializer='normal',activation='linear'))
```

```
# Compile the network:
```

```
NN_model.compile (loss='mean_absolute_error', optimizer='adam', metrics=['mean  
_absolute_error'])
```

```
NN_model.summary()
```

[Out]:

```
-----Layer (type) Output Shape Param #  
=====
```

-----	dense_1 (Dense)	(None, 128)	19200
-----	dense_2 (Dense)	(None, 256)	33024
-----	dense_3 (Derrse)	(None, 256)	65792
-----	dense_4 (Dense)	(None, 256)	65792
-----	dense_5 (Dense)	(None, 1)	257

```
=====
```

Total params: 184,065 Trainable params: 184,065 Non-trainable params: 0

```
#Define a checkpoint call back:
```

```
checkpoint_name = "Weights-{epoch:03d}--(val_loss:.5f).hdf5"
```

```
checkpoint = ModelCheckpoint (checkpoint_name, monitor='val _loss', verbose = 1,  
save_best_only= True, mode ='auto')
```

```
callbacks_list = [checkpoint]
```

```
#Train the model:
```

```
NN_model.fit(train, target, epochs=500, batch_size=32, validation_split = 0.2,  
callbacks=callbacks_list)
```

[out]:

Train on 1168 samples, validate on 292 samples

Epoch 1/500

1168/1168 [=====]-0s 266us/step - loss: 19251.8903 - mean_absolute_error:

19251.8903 - val_loss: 23041.8968 - val_mean_absolute_error: 23041.8968

Epoch 00001: val_loss did not improve from 21730.93555

Epoch 2/500

1168/1168 [=====]- 0s 268us/step- loss: 18180.4985- mean _absolute_error:

18180,4985- val_loss: 22197.7991 - val _mean_absolute_error: 22197.7991

Epoch 00002: val_loss did not improve from 21730.93555

Epoch 00500: val_loss did not improve from 18738.1983 1

Load wights file of the best model :

wights_file = "Weights-478--18738.19831. hdf5" # choose the best checkpoint

NN_model.load_weights (wights_file) # load it

NN_model.compile(loss='mean absolute error'.

optimizer='adam',metrics=['mean_absolute_error'])

#Test the model

def make_submission (prediction, sub_name):

my_submission = pd.DataFrame({'Id':pd.read_csv('test.csv'). Id, 'SalePrice':prediction})

my_submission.to_csv('{} .csv'.format(sub_name), index=False)

print('A submission file has been made')

```
predictions = NN_model.predict(test)
```

OUT:

0.14605