DL

Boston: linear regression

```
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")

from sklearn.datasets import load_boston
boston = load_boston()

data = pd.DataFrame(boston.data)

data.head()

#Adding the feature names to the dataframe
data.columns = boston.feature_names
data.head()

#Adding target variable to dataframe
data['PRICE'] = boston.target

#Check the shape of dataframe
data.shape
data.columns
data.dtypes
# Identifying the unique number of values in the dataset
data.nunique()
# Check for missing values
data.isnull().sum()

# See rows with missing values
data[data.isnull().any(axis=1)]

# Viewing the data statistics
data.describe()

# Finding out the correlation between the features
corr = data.corr()
corr.shape
```

```python
# Plotting the heatmap of correlation between features
plt.figure(figsize=(20,20))
sns.heatmap(corr, cbar=True, square= True, fmt='.1f',
annot=True,annot_kws={'size':15}, cmap='gray')

# Spliting target variable and independent variables
X = data.drop(['PRICE'], axis = 1)
y = data['PRICE']

# Splitting to training and testing data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size =
0.3,random_state = 4)

# Import library for Linear Regression
from sklearn.linear_model import LinearRegression

# Create a Linear regressor
lm = LinearRegression()
# Train the model using the training sets
lm.fit(X_train, y_train)

# Value of y intercept
lm.intercept_

#Converting the coefficient values to a dataframe
coeffcients = pd.DataFrame([X_train.columns,lm.coef_]).T
coeffcients = coeffcients.rename(columns={0: 'Attribute', 1:
'Coefficients'})
coeffcients

# Model prediction on train data
y_pred = lm.predict(X_train)

# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-
metrics.r2_score(y_train,y_pred))*(len(y_train)-1)/(len(y_train)-
X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
# Visualizing the differences between actual prices and predicted
values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()

# Checking residuals
plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()

# Checking Normality of errors
sns.distplot(y_train-y_pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()

# Predicting Test data with the model
y_test_pred = lm.predict(X_test)

# Model Evaluation
acc_linreg = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_linreg)
print('Adjusted R^2:',1 - (1-
metrics.r2_score(y_test,y_test_pred))*(len(y_test)-1)/(len(y_test)-
X_test.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

mnist fashion : CNN

```
#Import necessary libraries to carry out this classification
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import keras

(x_train, y_train), (x_test, y_test) =
tf.keras.datasets.fashion_mnist.load_data()
```

```
class_names = ['T_shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
'Sandal','Shirt', 'Sneaker', 'Bag', 'Ankle boot']

print("Number of images in training set {}".format(x_train.shape))
print("Number of labels in training set {}".format(y_train.shape))
print("Number of images in test set {}".format(x_test.shape))
print("Number of labels in test set {}".format(y_train.shape))

plt.figure()
plt.imshow(np.squeeze(x_train[220]))

y_train[220]

# Let us plot some training images to see how they look
plt.figure(figsize=(10,10))
for i in range(15):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_train[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[y_train[i]])
plt.show()

x_train=x_train/255
x_test=x_test/255

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import
Conv2D,MaxPooling2D,Dense,Flatten,Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard

cnn_model = Sequential()
cnn_model.add(Conv2D(32,3,3,input_shape = (28,28,1),activation =
'relu'))
cnn_model.add(MaxPooling2D(pool_size= (2,2)))
cnn_model.add(Flatten())
cnn_model.add(Dense(32,activation = 'relu'))
cnn_model.add(Dense(10,activation = 'sigmoid'))
cnn_model.summary()

##Once the model architecture is defined, we will compile and build the
model.
cnn_model.compile(loss ='sparse_categorical_crossentropy',optimizer =
Adam(learning_rate=0.001),metrics= ['accuracy'])
```

```
# history=cnn_model.fit(x_train, y_train, batch_size=32, epochs=10)
history=cnn_model.fit(x_train,y_train,batch_size =512,epochs =
5,verbose = 1,validation_data = (x_test,y_test) )

cnn_model.evaluate(x_test, y_test)

probability_model = tf.keras.Sequential([cnn_model,
tf.keras.layers.Softmax()])
predictions = probability_model.predict(x_test)

img = x_test[6]
plt.imshow(img)

y_predict = class_names[np.argmax(predictions[6])]
y_predict

y_actual = class_names[y_test[6]]
y_actual

img = x_test[0]
plt.imshow(img)

y_predict = class_names[np.argmax(predictions[0])]
y_predict

y_actual = class_names[y_test[0]]
y_actual

# we are using the normalized input data
test_loss, test_accuracy = cnn_model.evaluate(x_train, y_train)

print(test_accuracy)

print(test_loss)

history.history??
history.history.keys()
plt.plot(history.history['accuracy'],label='Accuracy')
plt.plot(history.history['val_accuracy'])
plt.title("Accuracy vs Validation Accuracy")
plt.xlabel("No. of epoch")
plt.ylabel("Accuracy")
plt.legend(['Train_acc', 'Val_acc'], loc='lower right')
plt.show()
```

```
plt.plot(history.history['loss'],label='Accuracy')
plt.plot(history.history['val_loss'])
plt.title("Loss vs Validation Accuracy")
plt.xlabel("No. of epoch")
plt.ylabel("Accuracy")
plt.legend(['Train_acc', 'Val_acc'], loc='lower right')
plt.show()
```

RNN stock price:

```
# Importing the libraries
import numpy as np #allow to make arrays
import matplotlib.pyplot as plt #visualize results on charts
import pandas as pd #import dataset and manage easily

# Importing the training set - only importing training set, test set
later on
#rnn has no idea of the test set's data, then after training is done,
test set will eb important
dataset_train = pd.read_csv(r'Google_Stock_Price_Train.csv')
#need to make into numpy arrays because only nump arrays can be input
values in keras
training_set = dataset_train.iloc[:, 1:2].values
#getting everything from the columns (.values makes the numpy array)

# Feature Scaling
# Normalizing the Dataset
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)
#fit (gets min and max on data to apply formula) tranform(compute scale
stock prices to each formula)

X_train = []
y_train = []
for i in range(60, 1257): # upper bound is number of values
    X_train.append(training_set_scaled[i-60:i, 0]) #takes 60 previous
stock prices from 60 past stock prices
    y_train.append(training_set_scaled[i, 0]) #contains stock price
learned to predict
X_train, y_train = np.array(X_train), np.array(y_train) # make into
numpy arrays
```

```
# Reshaping- add dimension in numpy array
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

X_train.shape

#Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

# Initialising the RNN
regressor = Sequential()

# Adding the first LSTM layer and some Dropout regularisation
#dropout to prevent overfitting
regressor.add(LSTM(units = 50, return_sequences = True, input_shape =
(X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

#automatically recognised through input shape
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# same as second layer
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

#(so it is removed becasue default is false)
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

#add fully connected layer through dense class- dimesion/units/neurons
is 1
regressor.add(Dense(units = 1))

#regressior because predicting continuous value,
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

regressor.fit(X_train, y_train, epochs = 200, batch_size = 32)

# Getting the Test Set
dataset_test = pd.read_csv('Google_Stock_Price_Test.csv')
real_stock_price = dataset_test.iloc[:, 1:2].values
```

```
# Getting the predicted stock price
dataset_total = pd.concat((dataset_train['Open'],
dataset_test['Open']), axis =0)
inputs = dataset_total[len(dataset_total) - len(dataset_test) -
60:].values
#getting input of each previous financial days
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []


inputs.shape

for i in range(60, 80):
    X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)

# Visualising the results
plt.plot(real_stock_price, color = 'red',label = 'Real Google Stock
Price')
plt.plot(predicted_stock_price, color = 'blue',label='Predicted Google
StockPrice')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Google Stock Price')
plt.legend()
plt.show()
```