Final_v2

June 9, 2020

[0]:
```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import os
import time

import datetime
import math
import scipy
from scipy.sparse import hstack
from sklearn.preprocessing import StandardScaler
from nltk.corpus import stopwords
from tqdm import tqdm
import re
import gc
import pickle
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import OneHotEncoder

# Loading Tensorflow libraries
import tensorflow as tf
from tensorflow.keras.layers import Dense,Input
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import load_model
```

[0]:
```
os.chdir("/content/drive/My Drive")
```

[0]:
```
def preprocess(df):
  df['name'] = df['name'].fillna('') + ' ' + df['brand_name'].fillna('')
  df['text'] = (df['item_description'].fillna('') + ' ' + df['name'] + ' '␣
 ↪+df['category_name'].fillna(''))
  return df[['name', 'text', 'shipping', 'item_condition_id']]
```

[0]:
```
def clean_data(train_data):
  # Since Mercari App doesn't allow price to be less than 3 or grater than
  # 2000, we need to remove those kind of data from training data

  train_data = train_data[(train_data.price >= 3) & (train_data.price <=2000)].
↪reset_index(drop=True)

  return train_data
```

[0]:
```
# Defining RMSLE Score

def rmsle_score(y, y_pred):
  assert len(y) == len(y_pred)
  to_sum = [(math.log(y_pred[i] + 1) - math.log(y[i] + 1)) ** 2.0 for i,pred in␣
↪enumerate(y_pred)]
  return (sum(to_sum) * (1.0/len(y))) ** 0.5
```

[0]:
```
# Building the MLP Model

def build_model(train_shape):
  input_layer = Input(shape=(train_shape,), dtype = 'float32',sparse = True)

  layer1 = Dense(256,activation = "relu",
                 kernel_initializer=tf.keras.initializers.he_uniform(seed =␣
↪42))(input_layer)

  layer2 = Dense(64,activation = "relu",
                 kernel_initializer=tf.keras.initializers.he_uniform(seed =␣
↪42))(layer1)

  layer3 = Dense(64,activation = "relu",
                 kernel_initializer=tf.keras.initializers.he_uniform(seed =␣
↪42))(layer2)

  layer4 = Dense(32,activation = "relu",
                 kernel_initializer=tf.keras.initializers.he_uniform(seed =␣
↪42))(layer3)

  output_layer = Dense(1,kernel_initializer=tf.keras.initializers.
↪he_uniform(seed = 42))(layer4)

  model = Model(inputs = input_layer, outputs = output_layer)

  return model
```

[0]:
```
def Vectorize_train_data(train_data):

  # Vectorizing the Name Column and Dumping the Object
  Vectorizer = TfidfVectorizer(max_features=100000,
                               token_pattern='\w+',dtype=np.float32)
  Vectorizer.fit(train_data['name'].values)
  X_train_name = Vectorizer.transform(train_data['name'].values)

  f = open("Name_Vectorizer","wb")
  pickle.dump(Vectorizer,f)
  f.close()

  # Vectorizing the Text Column and Dumping the Object
  Vectorizer = TfidfVectorizer(max_features=100000,ngram_range =(1,2),
                               token_pattern='\w+',dtype=np.float32)
  Vectorizer.fit(train_data['text'].values)
  X_train_text = Vectorizer.transform(train_data['text'].values)

  f = open("Text_Vectorizer","wb")
  pickle.dump(Vectorizer,f)
  f.close()


  # OneHotEncoding the Shipping Column and Dumping the Object
  Vectorizer = OneHotEncoder(dtype=np.float32)
  X_train_ship = Vectorizer.fit_transform(train_data['shipping'].values.
↪reshape(-1,1))

  f = open("Ship_Vectorizer","wb")
  pickle.dump(Vectorizer,f)
  f.close()

  # OneHotEncoding the Item Condition Id and Dumping the Object
  Vectorizer = OneHotEncoder(dtype=np.float32)
  X_train_item = Vectorizer.fit_transform(train_data['item_condition_id'].
↪values.reshape(-1,1))

  f = open("Item_Vectorizer","wb")
  pickle.dump(Vectorizer,f)
  f.close()


  # Stacking up all the Features
  X_train_tfidf = hstack((X_train_name,X_train_text,
                          X_train_ship,X_train_item)).tocsr()
```

```
  # Binarizing the Features

  # X_train_binary = [x.astype(np.bool).astype(np.float32) for x in␣
↪[X_train_tfidf]]
  X_train_binary = X_train_tfidf.astype(np.bool).astype(np.float32)


  print("X_train TFIDF Shape : ",X_train_tfidf.shape)
  print("X_train Binarized Shape : ",X_train_binary.shape)

  return X_train_tfidf,X_train_binary
```

The Training Function trains the Ensemble Model on the Train Data and saves the Model and object using a pickle file.

[0]:
```
def training():

  # Reading the Input Training Data
  train_data = pd.read_csv('train/train.tsv',sep = '\t')

  # Selecting every row except the last five rows
  train_data.drop(train_data.tail(5).index,inplace=True)

  train_data = clean_data(train_data)


  # Log Transformation of the Price Column and Dumping the Object
  scaler = StandardScaler()
  y_train = scaler.fit_transform(np.log1p(train_data['price'].values.
↪reshape(-1, 1)))

  f = open("Standard_Scaler","wb")
  pickle.dump(scaler,f)
  f.close()

  # Pre-processing the Train Data
  train_data = preprocess(train_data)

  X_train_tfidf,X_train_binary = Vectorize_train_data(train_data)

  del train_data
  gc.collect()

  # MLP1

  mlp1 = build_model(X_train_tfidf.shape[1])
  mlp1.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.003),
```

```
              loss= "mean_squared_error")

  for i in range(2):
    mlp1.fit(X_train_tfidf,y_train, batch_size= 2**(9 + i),
             epochs = 1,verbose= 1)

  # Saving the MLP1 Model
  mlp1.save("mlp1.h5")

  print("Saved Model 1")

  # MLP2

  mlp2 = build_model(X_train_binary.shape[1])
  mlp2.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.003),
              loss= "mean_squared_error")

  for i in range(2):
    mlp2.fit(X_train_binary,y_train, batch_size= 2**(9 + i),
             epochs = 1,verbose= 1)


  # Saving the MLP2 Model
  mlp1.save("mlp2.h5")

  print("Saved Model 2")

  print("Training Done.")
```

[0]:
```
def function1(test_data):

  print("Testing on the Test Data..")

  # Pre-processing the Test Data
  test_data = preprocess(test_data)

  # Vectorizing the Name Column in Test Data
  f = open("Name_Vectorizer","rb")
  Vectorizer = pickle.load(f)
  f.close()

  X_test_name = Vectorizer.transform(test_data['name'].values)

  # Vectorizing the Text Column in Test Data
  f = open("Text_Vectorizer","rb")
  Vectorizer = pickle.load(f)
```

```
f.close()

X_test_text = Vectorizer.transform(test_data['text'].values)

# OneHotEncoding the Shipping Column in Test Data
f = open("Ship_Vectorizer","rb")
Vectorizer = pickle.load(f)
f.close()

X_test_ship = Vectorizer.transform(test_data['shipping'].values.reshape(-1,1))

# OneHotEncoding the Item Condition Column in Test Data
f = open("Item_Vectorizer","rb")
Vectorizer = pickle.load(f)
f.close()

X_test_item = Vectorizer.transform(test_data['item_condition_id'].values.
↪reshape(-1,1))

# Stacking up all the Features
X_test_tfidf = hstack((X_test_name,X_test_text,
                       X_test_ship,X_test_item)).tocsr()


# Binarizing the Features
X_test_binary = X_test_tfidf.astype(np.bool).astype(np.float32)


print("X_test TFIDF Shape : ",X_test_tfidf.shape)
print("X_test Binarized Shape : ",X_test_binary.shape)

# Loading the MLP1 and MLP2
mlp1 = load_model("mlp1.h5")

mlp2 = load_model("mlp2.h5")

# Predicting the Value based on the Model Trained

y_pred1 = mlp1.predict(X_test_tfidf)[:,0]

f = open("Standard_Scaler","rb")
scaler = pickle.load(f)
f.close()

y_pred1 = np.expm1(scaler.inverse_transform(y_pred1.reshape(-1, 1))[:, 0])

y_pred2 = mlp2.predict(X_test_binary)[:,0]
```

```
    y_pred2 = np.expm1(scaler.inverse_transform(y_pred2.reshape(-1, 1))[:, 0])

    # Generating Emsemble of the above two MLP's

    y_prediction = 0.55 * y_pred1 + 0.45 * y_pred2

    print("Testing Done...")

    return y_prediction
```

[10]:
```
training()   # Training the Model and Saving Model and Objects accordingly
```

```
X_train TFIDF Shape :  (1481653, 200007)
X_train Binarized Shape :  (1481653, 200007)
2894/2894 [==============================] - 22s 8ms/step - loss: 0.3414
1447/1447 [==============================] - 15s 10ms/step - loss: 0.2025
Saved Model 1
2894/2894 [==============================] - 24s 8ms/step - loss: 0.3496
1447/1447 [==============================] - 15s 10ms/step - loss: 0.2097
Saved Model 2
Training Done.
```

1 Function 1

In Function1 we need to predict the Y(Price) value hence we will be training on the Entire train data except the last 5 rows and will compare the Actual Y(Price) value and the Predicted Y(Price) value.

[11]:
```
df_test = pd.read_csv('train/train.tsv',sep = '\t')
test_data = df_test.tail(5)  # Keeping the last 5 records as test

y_pred = function1(test_data)

i = 0
for index,row in test_data.iterrows():
  print("Input Data : ")
  print(row[['name','item_description','category_name',
                              'brand_name','shipping','item_condition_id']])
  print("Actual Price : ",row['price'])
  print("Predicted Price : ",y_pred[i])
  i = i + 1
  print("*"*80)
```

```
Testing on the Test Data..
X_test TFIDF Shape :  (5, 200007)
```

```
X_test Binarized Shape :  (5, 200007)
Testing Done…
Input Data :
name                              Free People Inspired Dress Free People
item_description     Lace, says size small but fits medium perfectl…
category_name                                    Women/Dresses/Mid-Calf
brand_name                                                  Free People
shipping                                                              1
item_condition_id                                                     2
Name: 1482530, dtype: object
Actual Price :  20.0
Predicted Price :  14.395757
******************************************************************************
Input Data :
name                               Little mermaid handmade dress Disney
item_description     Little mermaid handmade dress never worn size 2t
category_name                                  Kids/Girls 2T-5T/Dresses
brand_name                                                       Disney
shipping                                                              0
item_condition_id                                                     2
Name: 1482531, dtype: object
Actual Price :  14.0
Predicted Price :  8.884997
******************************************************************************
Input Data :
name                           21 day fix containers and eating plan
item_description          Used once or twice, still in great shape.
category_name        Sports & Outdoors/Exercise/Fitness accessories
brand_name                                                      NaN
shipping                                                          0
item_condition_id                                                 2
Name: 1482532, dtype: object
Actual Price :  12.0
Predicted Price :  33.529602
******************************************************************************
Input Data :
name                                          World markets lanterns
item_description     There is 2 of each one that you see! So 2 red …
category_name                       Home/Home Décor/Home Décor Accents
brand_name                                                      NaN
shipping                                                          1
item_condition_id                                                 3
Name: 1482533, dtype: object
Actual Price :  45.0
Predicted Price :  16.089226
******************************************************************************
Input Data :
name                                    Brand new lux de ville wallet
```

```
item_description     New with tag, red with sparkle. Firm price, no…
category_name                          Women/Women's Accessories/Wallets
brand_name                                                           NaN
shipping                                                               0
item_condition_id                                                      1
Name: 1482534, dtype: object
Actual Price :  22.0
Predicted Price :  33.35273
**************************************************************************************
```

[0]:

[0]:

2 Function 2

In Function 2 we need to check the metric(RMSLE) value where we will train the model using the Entire except the last 5 rows of training data and will pass the actual Price value and predicted Price value to check the RMSLE value.

[0]:
```
def function2(X_input,y_value):
  y_pred = function1(X_input)
  y_pred = np.asarray(y_pred)
  rmsle_value = rmsle_score(y_value,y_pred)
  return rmsle_value
```

[13]:
```
df_test = pd.read_csv('train/train.tsv',sep = '\t')

# Reading the only 5 rows of training data
df_input = df_test.tail(5)
actual_price = np.asarray(df_input['price'].values)

rmsle = function2(df_input,actual_price)
print("RMSLE Value : ",rmsle)
```

```
Testing on the Test Data..
X_test TFIDF Shape :  (5, 200007)
X_test Binarized Shape :  (5, 200007)
Testing Done…
RMSLE Value :  0.6878934461452819
```

[0]: