Final

June 7, 2020

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

import shutil
import datetime
import math
from contextlib import contextmanager
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]:
```
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,test_data):
  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)
  X_test_name = Vectorizer.transform(test_data['name'].values)

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

  Vectorizer = OneHotEncoder(dtype=np.float32)
  X_train_ship = Vectorizer.fit_transform(train_data['shipping'].values.
↪reshape(-1,1))
  X_test_ship = Vectorizer.transform(test_data['shipping'].values.reshape(-1,1))

  Vectorizer = OneHotEncoder(dtype=np.float32)
  X_train_item = Vectorizer.fit_transform(train_data['item_condition_id'].
↪values.reshape(-1,1))
  X_test_item = Vectorizer.transform(test_data['item_condition_id'].values.
↪reshape(-1,1))

  X_train_tfidf =hstack((X_train_name,X_train_text,
                         X_train_ship,X_train_item)).tocsr()
  X_test_tfidf = hstack((X_test_name,X_test_text,
                         X_test_ship,X_test_item)).tocsr()

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

↪  X_test_tfidf]]

  # print("X_train TFIDF Shape : ",X_train_tfidf.shape)
  # print("X_train Binarized Shape : ",X_train_binary.shape)
  # print("X_test TFIDF Shape : ",X_test_tfidf.shape)
  # print("X_test Binarized Shape : ",X_test_binary.shape)

  return X_train_tfidf,X_train_binary,X_test_tfidf,X_test_binary
```

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

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

```
# Selecting every row except the last one
train_data.drop(train_data.tail(1).index,inplace=True)

train_data = clean_data(train_data)

# Reading the Single Test Data
test_data = X_input

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

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

X_train_tfidf,X_train_binary,X_test_tfidf,X_test_binary =␣
↪Vectorize(train_data,
                                                        test_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)


# 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)
```

```
    # Predicting the Value based on the Model Trained

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

    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

    return y_prediction
```

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 one and will compare the Actual Y(Price) value and the Predicted Y(Price) value.

[9]:
```
df_test = pd.read_csv('train/train.tsv',sep = '\t')

# Reading only the last training data
df_input = df_test.tail(1)
y_pred = function1(df_input)

print("Input Value : ", df_input.values[0])
print("Actual Price : ", df_input['price'].values[0])
print("Predicted Price : ", y_pred)
```

```
2894/2894 [==============================] - 26s 9ms/step - loss: 0.3417
1447/1447 [==============================] - 17s 12ms/step - loss: 0.2035
2894/2894 [==============================] - 27s 9ms/step - loss: 0.3490
1447/1447 [==============================] - 17s 12ms/step - loss: 0.2099
Input Value :  [1482534 'Brand new lux de ville wallet ' 1
 "Women/Women's Accessories/Wallets" nan 22.0 0
 'New with tag, red with sparkle. Firm price, no free shipping.'
 "New with tag, red with sparkle. Firm price, no free shipping. Brand new lux de
ville wallet  Women/Women's Accessories/Wallets"]
Actual Price :  22.0
Predicted Price :  [33.144867]
```

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 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
```

[11]:
```
df_test = pd.read_csv('train/train.tsv',sep = '\t')

# Reading the only last training data
df_input = df_test.tail(1)
actual_price = np.asarray([df_input['price'].values[0]])

rmsle = function2(df_input,actual_price)
print("Input Value : ", df_input.values[0])
print("RMSLE Value : ",rmsle)
```

```
2894/2894 [==============================] - 27s 9ms/step - loss: 0.3413
1447/1447 [==============================] - 17s 12ms/step - loss: 0.2019
2894/2894 [==============================] - 27s 9ms/step - loss: 0.3494
1447/1447 [==============================] - 17s 12ms/step - loss: 0.2088
Input Value :  [1482534 'Brand new lux de ville wallet ' 1
 "Women/Women's Accessories/Wallets" nan 22.0 0
 'New with tag, red with sparkle. Firm price, no free shipping.'
 "New with tag, red with sparkle. Firm price, no free shipping. Brand new lux de
ville wallet  Women/Women's Accessories/Wallets"]
RMSLE Value :  0.24787538615201443
```

[0]:
```

```