Mercari-III

May 22, 2020

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

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import KFold

from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge

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
```

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

[0]:
```python
# Reference : https://www.kaggle.com/lopuhin/
 ↪mercari-golf-0-3875-cv-in-75-loc-1900-s
# The central Idea of pre-processing and MLP architecture has been adopted from␣
 ↪the
```

```
# above reference.
# This Function concatenates text features from␣
↪item-name,brand,category,description to create
# the concatenated text feature

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]:
```
# Reference : https://www.kaggle.com/lopuhin/
↪mercari-golf-0-3875-cv-in-75-loc-1900-s
@contextmanager
def timer(name):
    t0 = time.time()
    yield
    print(f'[{name}] done in {time.time() - t0:.0f} s')
```

[0]:
```
# Reference : https://www.kaggle.com/valkling/
↪mercari-rnn-2ridge-models-with-notes-0-42755
data = pd.read_csv('train.tsv',sep='\t')
data = data[(data.price >= 3) & (data.price <= 2000)].reset_index(drop=True)

cv = KFold(n_splits=20, shuffle=True, random_state=42)
train_ids, test_ids = next(cv.split(data))
# The above two line of code does K Fold Train Test Splitting where the Entire␣
↪data
# is divided into 20 folds each of size = len(data) // fold_size.
# Here data is roughly 1.4M and fold is 20 so each fold would be approx 74K

train, test = data.iloc[train_ids], data.iloc[test_ids]
# The Id's are saved to train and test respectively

# As we have seen that taking log values of the Price Column and
# standardizing them gives good result we will do the same

scaler = StandardScaler()
train_price = train['price'].values.reshape(-1,1)
test_price = test['price'].values.reshape(-1,1)

y_train = scaler.fit_transform(np.log1p(train_price))
y_test = scaler.transform(np.log1p(test_price))

print("X_Train Data Shape : ",train.shape)
```

```
print("y_train Shape : ",y_train.shape)
print("X_Test Data Shape : ",test.shape)
print("y_test Shape : ",y_test.shape)
```

```
X_Train Data Shape :  (1407575, 8)
y_train Shape :  (1407575, 1)
X_Test Data Shape :  (74083, 8)
y_test Shape :  (74083, 1)
```

1 Part I:

Here we will be using ensemble of 2 MLP's

[0]:
```
X_train = preprocess(train)
X_test = preprocess(test)
print(X_train.shape)
print(X_test.shape)
```

```
(1407575, 4)
(74083, 4)
```

[0]:
```
# Here we will create two Vectorized Copies of Train and Test
# One would be a normal TF-IDF Vectorized Copy
# Other being a Binarized Copy

Vectorizer = TfidfVectorizer(max_features=100000,token_pattern='\w+', dtype=np.
 ↪float32)
Vectorizer.fit(X_train['name'].values)

X_train_name = Vectorizer.transform(X_train['name'].values)
X_test_name = Vectorizer.transform(X_test['name'].values)

Vectorizer = TfidfVectorizer(max_features=100000,ngram_range =␣
 ↪(1,2),token_pattern='\w+', dtype=np.float32)

Vectorizer.fit(X_train['text'].values)

X_train_text = Vectorizer.transform(X_train['text'].values)
X_test_text = Vectorizer.transform(X_test['text'].values)


Vectorizer = OneHotEncoder(dtype=np.float32)

X_train_ship = Vectorizer.fit_transform(X_train['shipping'].values.
 ↪reshape(-1,1))
X_test_ship = Vectorizer.transform(X_test['shipping'].values.reshape(-1,1))
```

```
Vectorizer = OneHotEncoder(dtype=np.float32)

X_train_item = Vectorizer.fit_transform(X_train['item_condition_id'].values.
 ↪reshape(-1,1))
X_test_item = Vectorizer.transform(X_test['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()

# Creating binary version of the Dataset, it means after we get a sparse␣
 ↪matrix,
# we will clip all non-zero values to 1. This is almost the same as using a
# CountVectorizer with binary=True but works much faster than that as
# we don't need to re-process the data.

X_train_binary, X_test_binary = [x.astype(np.bool).astype(np.float32)
                                    for x in [X_train_tfidf, X_test_tfidf]]
```

[0]:
```
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)
```

```
X_train TFIDF Shape :  (1407575, 200007)
X_train Binarized Shape :  (1407575, 200007)
X_test TFIDF Shape :  (74083, 200007)
X_test Binarized Shape :  (74083, 200007)
```

[0]:
```
# Reference : https://www.kaggle.com/c/ashrae-energy-prediction/discussion/
 ↪113064

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
```

MLP Model 1

[0]:
```
input_layer = Input(shape=(X_train_tfidf.shape[1],), 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)

model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 200007)]          0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               51202048
_________________________________________________________________
dense_6 (Dense)              (None, 64)                16448
_________________________________________________________________
dense_7 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_8 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 33
=================================================================
Total params: 51,224,769
Trainable params: 51,224,769
Non-trainable params: 0
_________________________________________________________________
```

[0]:
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.003), loss =␣
↪"mean_squared_error")

for i in range(2):
  with timer (f'epoch {i + 1}'):
    model.fit(X_train_tfidf,y_train, batch_size= 2**(9 + i), epochs = 1,␣
↪verbose = 1,
                validation_data = (X_test_tfidf,y_test))
```

```
2750/2750 [==============================] - 511s 186ms/step - loss: 0.3430 -
val_loss: 0.3004
[epoch 1] done in 519 s
1375/1375 [==============================] - 272s 198ms/step - loss: 0.2016 -
val_loss: 0.2884
[epoch 2] done in 278 s
```

[0]:

```
model.save('model_part_1.h5')
y_pred = model.predict(X_test_tfidf)[:,0]
y_pred = np.expm1(scaler.inverse_transform(y_pred.reshape(-1, 1))[:, 0])
print("RMSLE from 1st MLP : ", rmsle_score(test_price,y_pred))
```

```
RMSLE from 1st MLP :  0.4005131897701359
```

MLP Model 2

[0]:

```
input_layer = Input(shape=(X_train_binary.shape[1],), 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)

model1 = Model(inputs = input_layer, outputs = output_layer)

model1.summary()
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 200007)]          0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               51202048
_________________________________________________________________
dense_11 (Dense)             (None, 64)                16448
_________________________________________________________________
```

```
dense_12 (Dense)             (None, 64)               4160
_________________________________________________________________
dense_13 (Dense)             (None, 32)               2080
_________________________________________________________________
dense_14 (Dense)             (None, 1)                33
=================================================================
Total params: 51,224,769
Trainable params: 51,224,769
Non-trainable params: 0
_________________________________________________________________
```

[0]:
```
model1.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.003), loss =␣
 ↪"mean_squared_error")

for i in range(3):
  with timer (f'epoch {i + 1}'):
    model1.fit(X_train_binary,y_train, batch_size= 2**(9 + i), epochs = 1,␣
 ↪verbose = 1,
              validation_data = (X_test_binary,y_test))
```

```
2750/2750 [==============================] - 504s 183ms/step - loss: 0.3522 -
val_loss: 0.3063
[epoch 1] done in 511 s
1375/1375 [==============================] - 264s 192ms/step - loss: 0.2087 -
val_loss: 0.2966
[epoch 2] done in 271 s
688/688 [==============================] - 149s 217ms/step - loss: 0.1194 -
val_loss: 0.2977
[epoch 3] done in 155 s
```

After 2 Epochs the Model starts overfitting.

[0]:
```
model1.save('model_part_2.h5')
y_pred_1 = model1.predict(X_test_binary)[:,0]
y_pred_1 = np.expm1(scaler.inverse_transform(y_pred_1.reshape(-1, 1))[:, 0])
print("RMSLE from 2nd MLP : ", rmsle_score(test_price,y_pred_1))
```

```
RMSLE from 2nd MLP :  0.4069041817341567
```

Ensemble the above two models

[0]:
```
print('Ensemble (weighted average of predictions from 2 models/runs')
y_prediction = np.average([y_pred, y_pred_1], weights=[0.66, 0.34], axis=0)
print("RMSLE from the Ensemble : ", rmsle_score(test_price,y_prediction))
```

```
Ensemble (weighted average of predictions from 2 models/runs
RMSLE from the Ensemble :  0.3905347322260435
```

Hence Using a Weighted Emsemble of two MLP's result in a Test RMSLE of 0.3905

2 Part II

Here we will create an emsemble of 2 MLP's with 2 Ridge Model

[0]:
```
print("X_Train Data Shape : ",train.shape)
print("y_train Shape : ",y_train.shape)
print("X_Test Data Shape : ",test.shape)
print("y_test Shape : ",y_test.shape)
```

```
X_Train Data Shape :  (1407575, 8)
y_train Shape :  (1407575, 1)
X_Test Data Shape :  (74083, 8)
y_test Shape :  (74083, 1)
```

[0]:
```
# Reference : Applied AI Course
def decontracted(phrase):
  # specific
  phrase = re.sub(r"won't", "will not", phrase)
  phrase = re.sub(r"can\'t", "can not", phrase)
  # general
  phrase = re.sub(r"n\'t", " not", phrase)
  phrase = re.sub(r"\'re", " are", phrase)
  phrase = re.sub(r"\'s", " is", phrase)
  phrase = re.sub(r"\'d", " would", phrase)
  phrase = re.sub(r"\'ll", " will", phrase)
  phrase = re.sub(r"\'t", " not", phrase)
  phrase = re.sub(r"\'ve", " have", phrase)
  phrase = re.sub(r"\'m", " am", phrase)

  return phrase
```

[0]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[0]: True

[0]:
```
stop_words = stopwords.words('english')
def preprocessing_text(text):
  preprocessed_text = []
  for sentence in tqdm(text.values):
    sentence = decontracted(sentence)
    sent = sentence.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
```

```
    sent = ' '.join(e for e in sent.split() if e not in stop_words)
    preprocessed_text.append(sent.lower().strip())
  return preprocessed_text
```

[0]:
```
def new_preprocess(df):
  df['name'] = df['preprocess_name'].fillna('') + ' ' + df['brand_name'].
 ↪fillna('')
  df['text'] = (df['preprocess_desc'].fillna('') + ' ' + df['preprocess_name']␣
 ↪+ ' ' + df['category_name'].fillna(''))

  return df[['name', 'text', 'shipping', 'item_condition_id']]
```

[0]:
```
train['name'] = train['name'].replace([np.nan], '')
test['name'] = test['name'].replace([np.nan], '')

train['item_description'] = train['item_description'].replace([np.nan], '')
test['item_description'] = test['item_description'].replace([np.nan], '')
```

[0]:
```
train['preprocess_name'] = preprocessing_text(train['name'])
test['preprocess_name'] = preprocessing_text(test['name'])

train['preprocess_desc'] = preprocessing_text(train['item_description'])
test['preprocess_desc'] = preprocessing_text(test['item_description'])
```

```
100%|      | 1407575/1407575 [00:33<00:00, 42552.76it/s]
100%|      | 74083/74083 [00:01<00:00, 42599.08it/s]
100%|      | 1407575/1407575 [01:47<00:00, 13149.41it/s]
100%|      | 74083/74083 [00:05<00:00, 12931.87it/s]
```

[0]:
```
X_train = new_preprocess(train)
X_test = new_preprocess(test)
print(X_train.shape)
print(X_test.shape)
```

```
(1407575, 4)
(74083, 4)
```

[0]:
```
Vectorizer = TfidfVectorizer(max_features=100000,token_pattern='\w+', dtype=np.
 ↪float32)
Vectorizer.fit(X_train['name'].values)

X_train_name = Vectorizer.transform(X_train['name'].values)
X_test_name = Vectorizer.transform(X_test['name'].values)

Vectorizer = TfidfVectorizer(max_features=100000,ngram_range =␣
 ↪(1,2),token_pattern='\w+', dtype=np.float32)
```

```
Vectorizer.fit(X_train['text'].values)

X_train_text = Vectorizer.transform(X_train['text'].values)
X_test_text = Vectorizer.transform(X_test['text'].values)

Vectorizer = CountVectorizer(vocabulary= list(X_train['shipping'].
 ↪unique()),lowercase=False, binary = True)

X_train_ship = Vectorizer.fit_transform(X_train['shipping'].values.astype(str))
X_test_ship = Vectorizer.transform(X_test['shipping'].values.astype(str))

Vectorizer = CountVectorizer(vocabulary= list(X_train['item_condition_id'].
 ↪unique()),lowercase=False, binary = True)

X_train_item = Vectorizer.fit_transform(X_train['item_condition_id'].values.
 ↪astype(str))
X_test_item = Vectorizer.transform(X_test['item_condition_id'].values.
 ↪astype(str))

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()
```

[0]:
```
print("X_train TFIDF Shape : ",X_train_tfidf.shape)
print("X_test TFIDF Shape : ",X_test_tfidf.shape)
```

```
X_train TFIDF Shape :  (1407575, 200007)
X_test TFIDF Shape :  (74083, 200007)
```

[0]:
```
ridge_model = Ridge(solver = "lsqr", fit_intercept=False, alpha=10)
ridge_model.fit(X_train_tfidf, y_train)

ridge_pred = ridge_model.predict(X_test_tfidf)[:,0]
ridge_pred = np.expm1(scaler.inverse_transform(ridge_pred.reshape(-1, 1))[:, 0])
```

[0]:
```
input_layer = Input(shape=(X_train_tfidf.shape[1],), 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)

model3 = Model(inputs = input_layer, outputs = output_layer)

model3.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 200007)]          0
_________________________________________________________________
dense (Dense)                (None, 256)               51202048
_________________________________________________________________
dense_1 (Dense)              (None, 64)                16448
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 51,224,769
Trainable params: 51,224,769
Non-trainable params: 0
_________________________________________________________________
```

[0]:
```
model3.compile(optimizer="adam", loss = "mean_squared_error")

for i in range(2):
  with timer (f'epoch {i + 1}'):
    model3.fit(X_train_tfidf,y_train, batch_size= 2**(9 + i), epochs = 1,␣
↪verbose = 1,
              validation_data = (X_test_tfidf,y_test))
```

```
2750/2750 [==============================] - 437s 159ms/step - loss: 0.3677 -
val_loss: 0.3227
[epoch 1] done in 442 s
1375/1375 [==============================] - 231s 168ms/step - loss: 0.2174 -
val_loss: 0.3124
[epoch 2] done in 235 s
```

[0]:
```
mlp1_pred = model3.predict(X_test_tfidf)[:,0]
mlp1_pred = np.expm1(scaler.inverse_transform(mlp1_pred.reshape(-1, 1))[:, 0])
```

[0]:
```
input_layer = Input(shape=(X_train_tfidf.shape[1],), dtype='float32',␣
↪sparse=True)

layer1 = Dense(512, activation = "relu",kernel_initializer=tf.keras.
↪initializers.he_uniform(seed = 42))(input_layer)

layer2 = Dense(256, 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(64, activation = "relu",kernel_initializer=tf.keras.initializers.
↪he_uniform(seed = 42))(layer3)

layer5 = Dense(32, activation = "relu",kernel_initializer=tf.keras.initializers.
↪he_uniform(seed = 42))(layer4)

output_layer = Dense(1,kernel_initializer=tf.keras.initializers.he_uniform(seed␣
↪= 42))(layer5)

model4 = Model(inputs = input_layer, outputs = output_layer)

model4.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 200007)]          0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               102404096
_________________________________________________________________
dense_6 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_7 (Dense)              (None, 64)                16448
_________________________________________________________________
dense_8 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_9 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 33
=================================================================
Total params: 102,558,145
```

```
Trainable params: 102,558,145
Non-trainable params: 0
_________________________________________________________________
```

[0]:
```
model4.compile(optimizer="adam", loss = "mean_squared_error")

for i in range(2):
  with timer (f'epoch {i + 1}'):
    model4.fit(X_train_tfidf,y_train, batch_size= 2**(9 + i), epochs = 1,␣
 ↪verbose = 1,
               validation_data = (X_test_tfidf,y_test))
```

```
2750/2750 [==============================] - 857s 311ms/step - loss: 0.3664 -
val_loss: 0.3240
[epoch 1] done in 861 s
1375/1375 [==============================] - 454s 330ms/step - loss: 0.2074 -
val_loss: 0.3075
[epoch 2] done in 459 s
```

[0]:
```
mlp2_pred = model4.predict(X_test_tfidf)[:,0]
mlp2_pred = np.expm1(scaler.inverse_transform(mlp2_pred.reshape(-1, 1))[:, 0])
```

[0]:
```
model3.save('model_part_3.h5')
model4.save('model_part_4.h5')
```

[0]:
```
print('Ensemble (weighted average of predictions from 3 models/runs')
y_prediction = np.average([ridge_pred, mlp1_pred,mlp2_pred], weights=[0.1, 0.
 ↪405,0.495], axis=0)
print("RMSLE from the Ensemble : ", rmsle_score(test_price,y_prediction))
```

```
Ensemble (weighted average of predictions from 3 models/runs
RMSLE from the Ensemble :  0.40639334528962295
```

Ensemble of 2 MLP's and Ridge Model leads to Test RMSLE 0.40639334528962295

One could defintely try other things like BatchNormalization, Dropouts,changing Activation Functions, but cannot increase the model's parameters as it would start overfitting and will also increase the time to train and test the Model.

[1]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "Test RMSLE"]

x.add_row(["Weighted Ensemble of 2 Sparse MLP's",0.3905])

x.add_row(["Weighted Ensemble of 2 Sparse MLP's with Ridge Regressor ",0.4063])


print(x)
```

Model	Test RMSLE
Weighted Ensemble of 2 Sparse MLP's	0.3905
Weighted Ensemble of 2 Sparse MLP's with Ridge Regressor	0.4063

Both the Models performs better than other Models that we have tried so far and the advantage of them over others is they require lesser time to train and test as compared to other models.

[0]: