Data-Driven Marketing 021 1222

The P-values Team

PDF file for the Final Project

Before we start:

- Unfortunately, this file is not converted directly from the .rmd file. we met many issues when copying and pasting each python line into the .rmd file. In addition, r studio was unstable, and many codes which can run in other IDE couldn't run smoothly in the .rmd file. Thus, this file was converted into PDF from multiple VS code files.
- Purpose of the ML, in this case, we are trying to understand the Dynamic aspects of the price today by exploring the Mercari dataset.
- In this project, as mentioned above, the top objective is to understand dynamic pricing. ML here is more about helping us to enhance our understanding rather than predicting the price accurately.
- I intended to do the EDA part in Python. However, I found Tableau is surprisingly helpful in this case. Thus, all the charts in the final presentation are done by Tableau.
- Models Considered: Catboost | LGBM | XGB | RNN | CNN
- Models used: LGBM | XGB | RNN | CNN (Catboost was dropped since it requires more than 2 hours to run)
- For RNN model, I got energy from this link: a-simple-nn-solution-with-keras
- For LGBM model, energy: ridge-lgbm, ftrl-fm-lgb, and price-recommendation-tds
- Got some EDA inspiration from: mercari-price-prediction
- I only have limited time to develop these models. There's still a lot of space for improvements, especially in the optimization part.
- Closing thoughts can be found at the very end of this file.
- I hope you will enjoy it, cheers.

Opportunistic Pricing Decisions with Machine Learning

Base Model: XGBoost

Content (XGB)

Why XGB

The list of benefits and attributes of XGBoost is extensive, and includes the following:

- A large and growing list of data scientists globally that are actively contributing to XGBoost open source development
- Usage on a wide range of applications, including solving problems in regression, classification, ranking, and user-defined prediction challenges
- A library that's highly portable and currently runs on OS X, Windows, and Linux platforms
- Cloud integration that supports AWS, Azure, Yarn clusters, and other ecosystems
- Active production use in multiple organizations across various vertical market areas

How XGB Work

XGBoost is a scalable and highly accurate implementation of gradient boosting that pushes the limits of computing power for boosted tree algorithms, being built largely for energizing machine learning model performance and computational speed. With XGBoost, trees are built in parallel, instead of sequentially like GBDT. It follows a level-wise strategy, scanning across gradient values and using these partial sums to evaluate the quality of splits at every possible split in the training set.

Dataset Prep

PS: Only Train Dataset from Mercari Train Dataset (Kaggle)

In [2]:
```
# All libraries required for the XGB are listed below:
#!pip install numpy
#!pip install pandas
#!pip install seaborn
#!pip install matplotlib
#!pip install sklearn
#!pip install xgboost

import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from multiprocessing import Pool
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import math

from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.pipeline import FeatureUnion
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import mean_squared_log_error
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
import xgboost as xgb
```

Read original Mercari Train file

In [3]:
```
df = pd.read_csv(f'train.tsv', sep='\t')
df = df.drop('train_id', axis = 1)
```

In [4]:
```
df.head(3)
#df.shape
```

Out[4]:

	name	item_condition_id	category_name	brand_name	price	shipping	item_description
0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN	10.0	1	No description yet
1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer	52.0	0	This keyboard is in great condition and works ...
2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target	10.0	1	Adorable top with a hint of lace and a key hol...

Identify and change the missing or blank records with 'missing'

In [5]:
```
for i in df:
    if df[i].isnull().sum()>0:
        print(i, df[i].isnull().sum())
```

```
category_name 6327
brand_name 632682
item_description 4
```

Split the category_name since in this col there're multiple sub categories

In [6]:
```
def split_cat(text):
    try: return text.split("/")
    except: return ("No Label", "No Label", "No Label")

df['general_cat'], df['subcat_1'], df['subcat_2'] = \
zip(*df['category_name'].apply(lambda x: split_cat(x)))

df.drop('category_name', axis = 1, inplace = True)
```

In [7]:
```
df['brand_name'].fillna(value = 'missing', inplace = True)
df['item_description'].fillna(value = 'missing', inplace = True)
```

In [8]:
```
df.head(3)
```

Out[8]:

	name	item_condition_id	brand_name	price	shipping	item_description	general_cat	subcat_1	subcat_2
0	MLB Cincinnati Reds T Shirt Size XL	3	missing	10.0	1	No description yet	Men	Tops	T-shirts
1	Razer BlackWidow Chroma Keyboard	3	Razer	52.0	0	This keyboard is in great condition and works ...	Electronics	Computers & Tablets	Components & Parts
2	AVA-VIV Blouse	1	Target	10.0	1	Adorable top with a hint of lace and a key hol...	Women	Tops & Blouses	Blouse

Evaluate item_description positive|Neutral|Negative levels

In [9]:
```
n_threads = 4

def flatten(l): return [item for sublist in l for item in sublist]

def _get_lemma_desc(args):
    data, index = args
    lmtzr = WordNetLemmatizer()
    lemmas = []
    for s in data:
        words = word_tokenize(s)
        lemmas.append(' '.join([lmtzr.lemmatize(w).lower() for w in words if w.isalpha()]))
    return pd.Series(lemmas, index=index)

def get_lemma_desc(data, index):
    p = Pool(processes=n_threads)
    n = math.ceil(len(data) / n_threads)
    lemmas = p.map(_get_lemma_desc, [(data[i:i + n], index[i:i + n]) for i in range(0, len(data), n)])
    return np.array(flatten(lemmas))

def _get_sentiment(data):
    sia = SentimentIntensityAnalyzer()
    scores = []
    for s in data:
        scores.append(sia.polarity_scores(s)['compound'])
    return np.array(scores)

def get_sentiment(data):
    p = Pool(processes=n_threads)
    n = math.ceil(len(data) / n_threads)
    scores = p.map(_get_sentiment, [data[i:i + n] for i in range(0, len(data), n)])
    return np.array(flatten(scores))
```

In [10]:
```
import nltk
from tqdm import tqdm
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\Abram\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[10]: True

Sentiment Analysis

In [11]:
```
sid = SentimentIntensityAnalyzer()
def sentiment_analyzer(df,preprocess_text):
    temp = []
    for sentence in tqdm(preprocess_text):
        for_sentiment = sentence
        ss = sid.polarity_scores(for_sentiment)
        temp.append(ss)
    negative=[]
    neutral=[]
    positive=[]
    compounding=[]
    for i in temp:
        for polarity,score in i.items():
            if(polarity=='neg'):
                negative.append(score)
            if(polarity=='neu'):
                neutral.append(score)
            if(polarity=='pos'):
                positive.append(score)
            if(polarity=='compound'):
                compounding.append(score)
    df['negative']=negative
    df['neutral']=neutral
    df['positive']=positive
    df['compound']=compounding
    return(df)

df_1 = sentiment_analyzer(df,df['item_description'].values)
```

```
100%|██████████████████████████████████████████████████████████████████████| 1482535/1482535 [05:24<00:00, 4574.44it/s]
```

In [12]:
```
df_1.head()
```

Out[12]:

	name	item_condition_id	brand_name	price	shipping	item_description	general_cat	subcat_1	subcat_2	negative	neutral	positive	compound
0	MLB Cincinnati Reds T Shirt Size XL	3	missing	10.0	1	No description yet	Men	Tops	T-shirts	0.524	0.476	0.000	-0.2960
1	Razer BlackWidow Chroma Keyboard	3	Razer	52.0	0	This keyboard is in great condition and works ...	Electronics	Computers & Tablets	Components & Parts	0.000	0.753	0.247	0.8957
2	AVA-VIV Blouse	1	Target	10.0	1	Adorable top with a hint of lace and a key hol...	Women	Tops & Blouses	Blouse	0.000	0.798	0.202	0.6792
3	Leather Horse Statues	1	missing	35.0	1	New with tags. Leather horses. Retail for [rm]...	Home	Home Décor	Home Décor Accents	0.000	0.833	0.167	0.6808
4	24K GOLD plated rose	1	missing	44.0	0	Complete with certificate of authenticity	Women	Jewelry	Necklaces	0.000	1.000	0.000	0.0000

Assigning sentiment analysis results values(-1:Negative, 1:Positive, 0:Neutral)

Be advised: Accuracy of this assignment still needed to be test

In [13]:
```
df_1.loc[df['compound'] <= -0.5, 'Description_level'] = -1
df_1.loc[df['compound'] >= 0.5, 'Description_level'] = 1
df_1['Description_level'].fillna(value = '0', inplace = True)
```

In [14]:
```
df_1 = df_1.drop(['negative', 'neutral', 'positive', 'compound'], axis=1)
```

In [15]:
```
df_1.head()
```

Out[15]:

	name	item_condition_id	brand_name	price	shipping	item_description	general_cat	subcat_1	subcat_2	Description_level
0	MLB Cincinnati Reds T Shirt Size XL	3	missing	10.0	1	No description yet	Men	Tops	T-shirts	0
1	Razer BlackWidow Chroma Keyboard	3	Razer	52.0	0	This keyboard is in great condition and works ...	Electronics	Computers & Tablets	Components & Parts	1.0
2	AVA-VIV Blouse	1	Target	10.0	1	Adorable top with a hint of lace and a key hol...	Women	Tops & Blouses	Blouse	1.0
3	Leather Horse Statues	1	missing	35.0	1	New with tags. Leather horses. Retail for [rm]...	Home	Home Décor	Home Décor Accents	1.0
4	24K GOLD plated rose	1	missing	44.0	0	Complete with certificate of authenticity	Women	Jewelry	Necklaces	0

In [16]:
```
#df_1.to_csv (r'C:\Users\abram2021\Desktop\Mercari_v2.csv', index = False, header=True)
```

Transform the price since its heavily right-skewed

In [17]:
```
df_1['price_log'] = df_1.apply(lambda row: np.log(row['price']+1), axis=1)
```

In [18]:
```
df = df_1
```

LabelEncoder to transform txt in certain columns into nums

In [19]:
```
le = LabelEncoder()

labels = ["brand_name","general_cat","subcat_1","subcat_2"]
for label in labels:
    df[label] = df[label].astype(str)
    le.fit(np.hstack(df[label]))
    df[label] = le.transform(df[label])
```

In [20]:
```
df.head(3)
```

Out[20]:

	name	item_condition_id	brand_name	price	shipping	item_description	general_cat	subcat_1	subcat_2	Description_level	price_log
0	MLB Cincinnati Reds T Shirt Size XL	3	4786	10.0	1	No description yet	5	103	763	0	2.397895
1	Razer BlackWidow Chroma Keyboard	3	3557	52.0	0	This keyboard is in great condition and works ...	1	30	212	1.0	3.970292
2	AVA-VIV Blouse	1	4180	10.0	1	Adorable top with a hint of lace and a key hol...	10	104	94	1.0	2.397895

In [21]:
```
df.general_cat = df.general_cat.astype(str)
vectorizer = CountVectorizer(token_pattern='\d+')
x = vectorizer.fit_transform(df.general_cat)
```

In [22]:
```
df.item_condition_id = df.item_condition_id.astype(str)
df.shipping = df.shipping.astype(str)
df.Description_level = df.Description_level.astype(str)
df.general_cat = df.general_cat.astype(str)
df.subcat_1 = df.subcat_1.astype(str)
df.subcat_2 = df.subcat_2.astype(str)
df.brand_name = df.brand_name.astype(str)

df.dtypes
```

Out[22]:
```
name                  object
item_condition_id     object
brand_name            object
price                float64
shipping              object
item_description      object
general_cat           object
subcat_1              object
subcat_2              object
Description_level     object
price_log            float64
dtype: object
```

In [23]:
```
default_preprocessor = CountVectorizer().build_preprocessor()
def preprocessor(field):
    field_idx = list(df.columns).index(field)
    return lambda x: default_preprocessor(x[field_idx])


vectorizer = FeatureUnion([
    ('name', CountVectorizer(
        ngram_range=(1, 2),
        max_features=50000,
        preprocessor = preprocessor('name'))),
    ('general_cat', CountVectorizer(
        token_pattern='\d+',
        preprocessor=preprocessor('general_cat'))),
    ('subcat_1', CountVectorizer(
        token_pattern='\d+',
        preprocessor=preprocessor('subcat_1'))),
    ('subcat_2', CountVectorizer(
        token_pattern='\d+',
        preprocessor=preprocessor('subcat_2'))),
    ('brand_name', CountVectorizer(
        token_pattern='\d+',
        preprocessor=preprocessor('brand_name'))),
    ('shipping', CountVectorizer(
        token_pattern='\d+',
        preprocessor=preprocessor('shipping'))),
    ('Description_level', CountVectorizer(
        token_pattern='\d+',
        preprocessor=preprocessor('Description_level'))),
    ('item_condition_id', CountVectorizer(
        token_pattern='\d+',
        preprocessor=preprocessor('item_condition_id'))),
    ('item_description', TfidfVectorizer(
        ngram_range=(1, 3),
        max_features=100000,
        preprocessor=preprocessor('item_description'))),
])
```

In [24]:
```
X = vectorizer.fit_transform(df.values)
```

In [25]:
```
train = df
```

In [26]:
```
trainData = X[:train.shape[0]]
target = np.log1p(train.price)

testData = X[train.shape[0]:]
```

In [27]:
```
X_train, X_valid, y_train, y_valid = train_test_split(trainData, target, test_size=0.2, random_state=0)
```

Set parameters(Optimized)

In [28]:
```
trainmat=xgb.DMatrix(X_train,y_train)
our_params={
    'eta':0.3,
    'seed':123,
    'subsample':0.9,
    'colsample_bytree':0.7,
    'objective':'reg:linear',
    'max_depth':9,
    'gamma':0.2,
    'min_child_weight':1}
```

In [29]:
```
final_gb=xgb.train(our_params,trainmat)
```

```
[20:59:40] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.6.0/src/objective/regression_obj.cu:203: reg:linear is now deprecated in favor of reg:squarederror.
```

In [30]:
```
validmat=xgb.DMatrix(X_valid)
```

In [31]:
```
y_pred=final_gb.predict(validmat)
```

In [32]:
```
print('Score:', mean_squared_error(y_valid, np.array(y_pred)))
```

```
Score: 0.35451420066897144
```

RMSE

In [34]:
```
print('RMSE:', mean_squared_error(y_valid, y_pred) ** 0.5)
```

```
RMSE: 0.5954109510825035
```

In [35]:
```
#testmat=xgb.DMatrix(testData)
#preds = final_gb.predict(testmat)
#For new test purposes
```

```
[21:00:29] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.6.0/src/learner.cc:1350: Empty dataset at worker: 0
```

In [36]:
```
predmat = xgb.DMatrix(trainData)
```

In [37]:
```
pred_comp = final_gb.predict(predmat)
```

In [38]:
```
pred_comp
```

Out[38]:
```
array([2.4284809, 3.3430777, 2.5822804, ..., 2.798585 , 2.509044 ,
       2.8560402], dtype=float32)
```

In [39]:
```
np.exp(pred_comp[0])-1
```

Out[39]: 10.341639518737793

In [40]:
```
df_pred = pd.DataFrame(pred_comp, columns = ['pred'])
```

In [41]:
```
df_final = pd.concat([df_1, df_pred], axis = 1)
```

In [42]:
```
df_final.head()
```

Out[42]:

	name	item_condition_id	brand_name	price	shipping	item_description	general_cat	subcat_1	subcat_2	Description_level	price_log	pred
0	MLB Cincinnati Reds T Shirt Size XL	3	4786	10.0	1	No description yet	5	103	763	0	2.397895	2.428481
1	Razer BlackWidow Chroma Keyboard	3	3557	52.0	0	This keyboard is in great condition and works ...	1	30	212	1.0	3.970292	3.343078
2	AVA-VIV Blouse	1	4180	10.0	1	Adorable top with a hint of lace and a key hol...	10	104	94	1.0	2.397895	2.582280
3	Leather Horse Statues	1	4786	35.0	1	New with tags. Leather horses. Retail for [rm]...	3	55	404	1.0	3.583519	2.736905
4	24K GOLD plated rose	1	4786	44.0	0	Complete with certificate of authenticity	10	58	536	0	3.806662	2.840543

Content (LGBM)

Why LightGBM

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency
- Lower memory usage
- Better accuracy
- Parallel and GPU learning supported
- Capable of handling large-scale data

Therefore, we are going to give it a try.

How LightGBM Work

LightGBM is a supported decision tree algorithm, it splits the tree leaf wise with the simplest fit whereas other boosting algorithms split the tree depth wise or level wise instead of leaf-wise.

When growing on an equivalent leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence leads to far better accuracy which may rarely be achieved by any of the prevailing boosting algorithms.

It produces far more complex trees by following leaf wise split approach instead of a level-wise approach which is that the main think about achieving higher accuracy. However, it can sometimes cause overfitting which may be avoided by setting the max_depth parameter.

Reference: LightGBM Website

Dataset Prep

PS: Only Train Dataset from Mercari Train Dataset (Kaggle)

In [1]:

```python
# All libraries required for the Base Model: LGBM are listed below:
#!pip install numpy
#!pip install pandas
#!pip install matplotlib
#!pip install scipy
#!pip install sklearn
#!pip install lightgbm

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.sparse import csr_matrix, hstack
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn import metrics
from sklearn.metrics import mean_squared_error
import lightgbm as lgb
```

In [2]:

```python
df = pd.read_csv(f'train.tsv', sep='\t')
df.head()
#df.shape
```

Out[2]:

	train_id	name	item_condition_id	category_name	brand_name	price	shipping	item_descriptio
0	0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN	10.0	1	No description ye
1	1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer	52.0	0	This keyboard is in great conditio and works ..
2	2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target	10.0	1	Adorable top witl a hint of lace anc a key hol..
3	3	Leather Horse Statues	1	Home/Home Décor/Home Décor Accents	NaN	35.0	1	New with tags Leather horses Retail for [rm]..
4	4	24K GOLD plated rose	1	Women/Jewelry/Necklaces	NaN	44.0	0	Complete witl certificate o authenticity

From EDA we know there're missing values in many of the cols

In [3]:

```python
print('Missing value counts:')
for i in df:
    if df[i].isnull().sum()>0:
        print(i, df[i].isnull().sum())
```

```
Missing value counts:
category_name 6327
brand_name 632682
item_description 4
```

General Setting

In [4]:

```python
NUM_BRANDS = 4000
```

```
NUM_CATEGORIES = 1000
NAME_MIN_DF = 10
MAX_FEATURES_ITEM_DESCRIPTION = 50000
```

Functions: Replace the null with 'missing'/Cut dataset/convert cat

In [5]:

```
def handle_missing_inplace(df):
    df['category_name'].fillna(value='missing', inplace=True)
    df['brand_name'].fillna(value='missing', inplace=True)
    df['item_description'].replace('No description yet,''missing', inplace=True)
    df['item_description'].fillna(value='missing', inplace=True)
def cutting(df):
    pop_brand = df['brand_name'].value_counts().loc[lambda x: x.index != 'missing'].index[
    df.loc[~df['brand_name'].isin(pop_brand), 'brand_name'] = 'missing'
    pop_category = df['category_name'].value_counts().loc[lambda x: x.index != 'missing'].
def to_categorical(df):
    df['category_name'] = df['category_name'].astype('category')
    df['brand_name'] = df['brand_name'].astype('category')
    df['item_condition_id'] = df['item_condition_id'].astype('category')
```

Split dataset into train and test and remove price of 0

In [6]:

```
msk = np.random.rand(len(df)) < 0.8
train = df[msk]
test = df[~msk]
test_new = test.drop('price', axis=1)
y_test = np.log1p(test["price"])
train = train[train.price != 0].reset_index(drop=True)
train.shape
```

Out[6]: (1185898, 8)

Merge train and new test data

In [7]:

```
nrow_train = train.shape[0]
y = np.log1p(train["price"])
merge: pd.DataFrame = pd.concat([train, test_new])
```

Implement previous functions to dataset

In [8]:

```
handle_missing_inplace(merge)
cutting(merge)
to_categorical(merge)
```

Model

In [9]:

```
cv = CountVectorizer(min_df=NAME_MIN_DF)
X_name = cv.fit_transform(merge['name'])
cv = CountVectorizer()
X_category = cv.fit_transform(merge['category_name'])
```

In [10]:

```
tv = TfidfVectorizer(max_features=MAX_FEATURES_ITEM_DESCRIPTION, ngram_range=(1, 3), stop_
X_description = tv.fit_transform(merge['item_description'])
```

In [11]:

```
lb = LabelBinarizer(sparse_output=True)
```

Loading [MathJax]/extensions/Safe.js

```
X_brand = lb.fit_transform(merge['brand_name'])
```

In [12]:

```
X_dummies = csr_matrix(pd.get_dummies(merge[['item_condition_id', 'shipping']], sparse=Tru
```

In [13]:

```
sparse_merge = hstack((X_dummies, X_description, X_brand, X_category, X_name)).tocsr()
```

In [14]:

```
mask = np.array(np.clip(sparse_merge.getnnz(axis=0) - 1, 0, 1), dtype=bool)
sparse_merge = sparse_merge[:, mask]
```

In [15]:

```
X = sparse_merge[:nrow_train]
X_test = sparse_merge[nrow_train:]
```

In [16]:

```
train_X = lgb.Dataset(X, label=y)
```

Set parameters

In [17]:

```
params = {
        'learning_rate': 0.75,
        'application': 'regression',
        'max_depth': 3,
        'num_leaves': 100,
        'verbosity': -1,
        'metric': 'RMSE',
    }
```

Training

In [18]:

```
gbm = lgb.train(params, train_set=train_X, num_boost_round=3200, verbose_eval=100)
```

```
C:\Users\Abram\Anaconda3\lib\site-packages\lightgbm\engine.py:239: UserWarning: 'verbose_e
val' argument is deprecated and will be removed in a future release of LightGBM. Pass 'log
_evaluation()' callback via 'callbacks' argument instead.
  _log_warning("'verbose_eval' argument is deprecated and will be removed in a future rele
ase of LightGBM. "
```

Prediction

In [19]:

```
y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration)
```

RMSE

In [21]:

```
from sklearn.metrics import mean_squared_error
print('RMSE:', mean_squared_error(y_test, y_pred) ** 0.5)
```

```
RMSE: 0.458683501078225
```

Content (NN Solutions)

Why NN

NN solutions have some advantages listed below: It helps to model the nonlinear and complex relationships of the real world. They are used in pattern recognition because they can generalize. They have many applications like text summarization, signature identification, handwriting recognition, and many more. It can model data with high volatility.

Therefore, we are going to give it a try.

Model (RNN)

In [1]:

```
# All libraries required for the Base Model: nn are listed below:
#!pip install numpy
#!pip install pandas
#!pip install matplotlib
#!pip install scipy
#!pip install sklearn
#!pip install keras

import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_log_error
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Input, Dropout, Dense, BatchNormalization, Activation, concatenate, GRU, Embedding, Flatten, BatchNormalization
from keras.models import Model
from keras.callbacks import ModelCheckpoint, Callback, EarlyStopping
from keras import backend as K
```

In [2]:

```
train = pd.read_table("train.tsv")
print(train.shape)
```

```
(1482535, 8)
```

In [3]:

```
def handle_missing(df):
    df.category_name.fillna(value="missing", inplace=True)
    df.brand_name.fillna(value="missing", inplace=True)
    df.item_description.fillna(value="missing", inplace=True)
    return (df)

train = handle_missing(train)
print(train.shape)
```

```
(1482535, 8)
```

In [4]:

```
le = LabelEncoder()

le.fit(np.hstack(train.category_name))
train.category_name = le.transform(train.category_name)
le.fit(np.hstack(train.brand_name))
train.brand_name = le.transform(train.brand_name)
del le

train.head()
```

Out[4]:

	train_id	name	item_condition_id	category_name	brand_name	price	shipping	item_description
0	0	MLB Cincinnati Reds T Shirt Size XL	3	808	4786	10.0	1	No description yet
1	1	Razer BlackWidow Chroma Keyboard	3	86	3557	52.0	0	This keyboard is in great condition and works ...
2	2	AVA-VIV Blouse	1	1254	4180	10.0	1	Adorable top with a hint of lace and a key hol...
3	3	Leather Horse Statues	1	485	4786	35.0	1	New with tags. Leather horses. Retail for [rm]...
4	4	24K GOLD plated rose	1	1181	4786	44.0	0	Complete with certificate of authenticity

In [5]:

```
#Text to seq
raw_text = np.hstack([train.item_description.str.lower(), train.name.str.lower()])

tok_raw = Tokenizer()
tok_raw.fit_on_texts(raw_text)

train["seq_item_description"] = tok_raw.texts_to_sequences(train.item_description.str.lower())
train["seq_name"] = tok_raw.texts_to_sequences(train.name.str.lower())
train.head()
```

Out[5]:

	train_id	name	item_condition_id	category_name	brand_name	price	shipping	item_description	seq_item_description	seq_name
0	0	MLB Cincinnati Reds T Shirt Size XL	3	808	4786	10.0	1	No description yet	[12, 68, 79]	[3852, 8823, 6896, 208, 84, 6, 155]
1	1	Razer BlackWidow Chroma Keyboard	3	86	3557	52.0	0	This keyboard is in great condition and works ...	[29, 2627, 10, 7, 39, 17, 1, 207, 51, 19, 1113...	[10760, 25565, 16369, 2627]
2	2	AVA-VIV Blouse	1	1254	4180	10.0	1	Adorable top with a hint of lace and a key hol...	[604, 60, 9, 4, 5347, 11, 192, 1, 4, 886, 1290...	[7634, 10563, 666]
3	3	Leather Horse Statues	1	485	4786	35.0	1	New with tags. Leather horses. Retail for [rm]...	[5, 9, 61, 178, 6528, 230, 3, 21, 166, 1085, 2...	[178, 2610, 14248]
4	4	24K GOLD plated rose	1	1181	4786	44.0	0	Complete with certificate of authenticity	[807, 9, 6888, 11, 1997]	[4884, 104, 1032, 280]

In [6]:

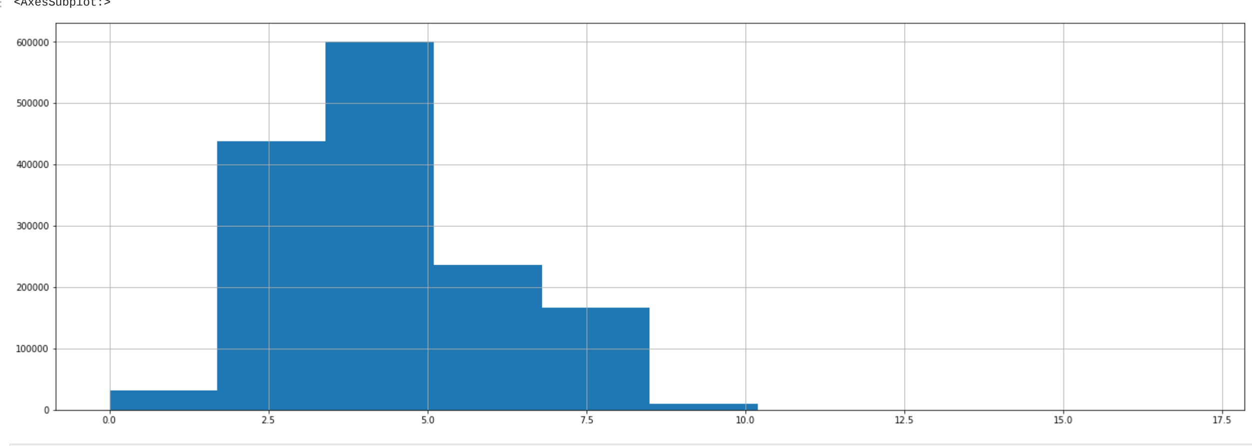
```
max_name_seq = np.max([np.max(train.seq_name.apply(lambda x: len(x)))])
max_seq_item_description = np.max([np.max(train.seq_item_description.apply(lambda x: len(x)))
                                   ])
print("max name seq "+str(max_name_seq))
print("max item desc seq "+str(max_seq_item_description))
```

```
max name seq 17
max item desc seq 269
```

In [7]:

```
fig, ax = plt.subplots(figsize=(24,8))
train.seq_name.apply(lambda x: len(x)).hist(ax = ax)
```

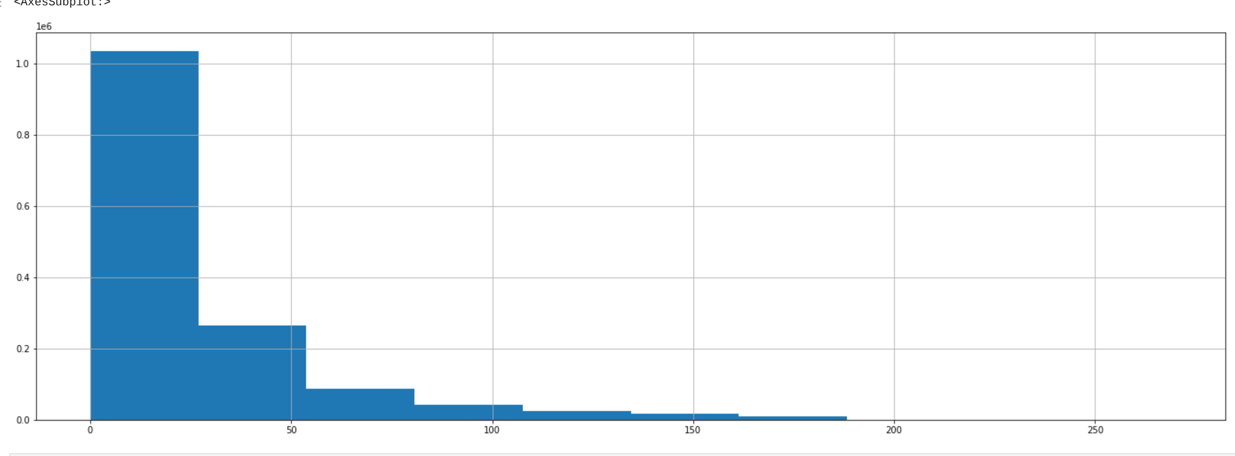
Out[7]: <AxesSubplot:>



In [8]:

```
fig, ax = plt.subplots(figsize=(24,8))
train.seq_item_description.apply(lambda x: len(x)).hist(ax = ax)
```

Out[8]: <AxesSubplot:>



In [9]:

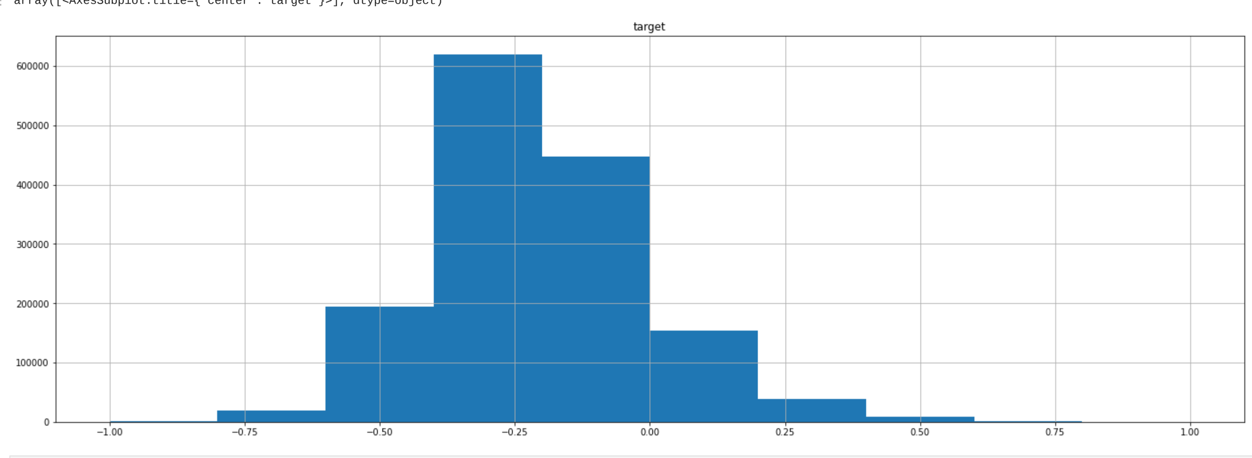
```
MAX_NAME_SEQ = 10
MAX_ITEM_DESC_SEQ = 75
MAX_TEXT = np.max([np.max(train.seq_name.max())
                   , np.max(train.seq_item_description.max())])+2
MAX_CATEGORY = np.max([train.category_name.max()])+1
MAX_BRAND = np.max([train.brand_name.max()])+1
MAX_CONDITION = np.max([train.item_condition_id.max()])+1
```

In [10]:

```
train["target"] = np.log(train.price+1)
target_scaler = MinMaxScaler(feature_range=(-1, 1))
train["target"] = target_scaler.fit_transform(train.target.values.reshape(-1,1))

fig, ax = plt.subplots(figsize=(24,8))
pd.DataFrame(train.target).hist(ax = ax)
```

Out[10]: array([<AxesSubplot:title={'center':'target'}>], dtype=object)



In [11]:

```
dtrain, dvalid = train_test_split(train, random_state=123, train_size=0.8)
print(dtrain.shape)
print(dvalid.shape)
```

```
(1186028, 11)
(296507, 11)
```

In [12]:

```
dtrain, test = train_test_split(dtrain, random_state=123, train_size=0.8)
test.shape
```

Out[12]: (237206, 11)

In [13]:

```
def get_keras_data(df):
    X = {
        'name': pad_sequences(df.seq_name, maxlen=MAX_NAME_SEQ)
        ,'item_desc': pad_sequences(df.seq_item_description, maxlen=MAX_ITEM_DESC_SEQ)
        ,'brand_name': np.array(df.brand_name)
        ,'category_name': np.array(df.category_name)
        ,'item_condition': np.array(df.item_condition_id)
        ,'num_vars': np.array(df[["shipping"]])
    }
    return X

X_train = get_keras_data(dtrain)
X_valid = get_keras_data(dvalid)
X_test = get_keras_data(test)
```

In [14]:

```
def get_callbacks(filepath, patience=2):
    es = EarlyStopping('val_loss', patience=patience, mode="min")
    msave = ModelCheckpoint(filepath, save_best_only=True)
    return [es, msave]

def rmsle_cust(y_true, y_pred):
    first_log = K.log(K.clip(y_pred, K.epsilon(), None) + 1.)
    second_log = K.log(K.clip(y_true, K.epsilon(), None) + 1.)
    return K.sqrt(K.mean(K.square(first_log - second_log), axis=-1))

def get_model():
    #params
    dr_r = 0.1

    #Inputs
    name = Input(shape=[X_train["name"].shape[1]], name="name")
    item_desc = Input(shape=[X_train["item_desc"].shape[1]], name="item_desc")
    brand_name = Input(shape=[1], name="brand_name")
    category_name = Input(shape=[1], name="category_name")
    item_condition = Input(shape=[1], name="item_condition")
    num_vars = Input(shape=[X_train["num_vars"].shape[1]], name="num_vars")

    #Embeddings layers
    emb_name = Embedding(MAX_TEXT, 50)(name)
    emb_item_desc = Embedding(MAX_TEXT, 50)(item_desc)
    emb_brand_name = Embedding(MAX_BRAND, 10)(brand_name)
    emb_category_name = Embedding(MAX_CATEGORY, 10)(category_name)
    emb_item_condition = Embedding(MAX_CONDITION, 5)(item_condition)

    #rnn layer
    rnn_layer1 = GRU(16) (emb_item_desc)
    rnn_layer2 = GRU(8) (emb_name)

    #main layer
    main_l = concatenate([
        Flatten() (emb_brand_name)
        , Flatten() (emb_category_name)
        , Flatten() (emb_item_condition)
        , rnn_layer1
        , rnn_layer2
        , num_vars
    ])
    main_l = Dropout(dr_r) (Dense(128) (main_l))
    main_l = Dropout(dr_r) (Dense(64) (main_l))

    #output
    output = Dense(1, activation="linear") (main_l)

    #model
    model = Model([name, item_desc, brand_name
                   , category_name, item_condition, num_vars], output)
    model.compile(loss="mse", optimizer="adam", metrics=["mae", rmsle_cust])

    return model


model = get_model()
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 brand_name (InputLayer)        [(None, 1)]          0           []

 category_name (InputLayer)     [(None, 1)]          0           []

 item_condition (InputLayer)    [(None, 1)]          0           []

 item_desc (InputLayer)         [(None, 75)]         0           []

 name (InputLayer)              [(None, 10)]         0           []

 embedding_2 (Embedding)        (None, 1, 10)        48100       ['brand_name[0][0]']

 embedding_3 (Embedding)        (None, 1, 10)        12880       ['category_name[0][0]']

 embedding_4 (Embedding)        (None, 1, 5)         30          ['item_condition[0][0]']

 embedding_1 (Embedding)        (None, 75, 50)       12954400    ['item_desc[0][0]']

 embedding (Embedding)          (None, 10, 50)       12954400    ['name[0][0]']

 flatten (Flatten)              (None, 10)           0           ['embedding_2[0][0]']

 flatten_1 (Flatten)            (None, 10)           0           ['embedding_3[0][0]']

 flatten_2 (Flatten)            (None, 5)            0           ['embedding_4[0][0]']

 gru (GRU)                      (None, 16)           3264        ['embedding_1[0][0]']

 gru_1 (GRU)                    (None, 8)            1440        ['embedding[0][0]']

 num_vars (InputLayer)          [(None, 1)]          0           []

 concatenate (Concatenate)      (None, 50)           0           ['flatten[0][0]',
                                                                  'flatten_1[0][0]',
                                                                  'flatten_2[0][0]',
                                                                  'gru[0][0]',
                                                                  'gru_1[0][0]',
                                                                  'num_vars[0][0]']

 dense (Dense)                  (None, 128)          6528        ['concatenate[0][0]']

 dropout (Dropout)              (None, 128)          0           ['dense[0][0]']

 dense_1 (Dense)                (None, 64)           8256        ['dropout[0][0]']

 dropout_1 (Dropout)            (None, 64)           0           ['dense_1[0][0]']

 dense_2 (Dense)                (None, 1)            65          ['dropout_1[0][0]']

==================================================================================================
Total params: 25,989,363
Trainable params: 25,989,363
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [15]:

```
BATCH_SIZE = 20000
epochs = 5

model = get_model()
model.fit(X_train, dtrain.target, epochs=epochs, batch_size=BATCH_SIZE
          , validation_data=(X_valid, dvalid.target)
          , verbose=1)
```

```
Epoch 1/5
48/48 [==============================] - 216s 4s/step - loss: 0.0310 - mae: 0.1348 - rmsle_cust: 0.0160 - val_loss: 0.0198 - val_mae: 0.1066 - val_rmsle_cust: 0.0145
Epoch 2/5
48/48 [==============================] - 212s 4s/step - loss: 0.0196 - mae: 0.1062 - rmsle_cust: 0.0138 - val_loss: 0.0174 - val_mae: 0.0991 - val_rmsle_cust: 0.0133
Epoch 3/5
48/48 [==============================] - 207s 4s/step - loss: 0.0171 - mae: 0.0990 - rmsle_cust: 0.0130 - val_loss: 0.0162 - val_mae: 0.0960 - val_rmsle_cust: 0.0128
Epoch 4/5
48/48 [==============================] - 197s 4s/step - loss: 0.0156 - mae: 0.0945 - rmsle_cust: 0.0124 - val_loss: 0.0157 - val_mae: 0.0942 - val_rmsle_cust: 0.0126
Epoch 5/5
48/48 [==============================] - 214s 4s/step - loss: 0.0146 - mae: 0.0914 - rmsle_cust: 0.0120 - val_loss: 0.0156 - val_mae: 0.0939 - val_rmsle_cust: 0.0125
```

Out[15]: <keras.callbacks.History at 0x1de8868d810>

Model Evaluation (RNN)

RMSE

In [18]:

```
print('RMSE:', mean_squared_error(y_true, y_pred) ** 0.5)
```

```
RMSE: 28.3813718261156
```

Model (CNN)

In [19]:

```
from keras.layers import Conv1D,GlobalMaxPooling1D
def cnn_model(lr=0.001, decay=0.0):
    # Inputs
    name = Input(shape=[X_train["name"].shape[1]], name="name")
    item_desc = Input(shape=[X_train["item_desc"].shape[1]], name="item_desc")
    brand_name = Input(shape=[1], name="brand_name")
    category_name = Input(shape=[1], name="category_name")
    item_condition = Input(shape=[1], name="item_condition")
    num_vars = Input(shape=[X_train["num_vars"].shape[1]], name="num_vars")

    # Embeddings layers
    emb_name = Embedding(MAX_TEXT, 20)(name)
    emb_item_desc = Embedding(MAX_TEXT, 60)(item_desc)
    emb_brand_name = Embedding(MAX_BRAND, 10)(brand_name)
    emb_category_name = Embedding(MAX_CATEGORY, 10)(category_name)

    convs1 = []
    convs2 = []

    for filter_length in [1,2,3]:
        cnn_layer1 = Conv1D(filters=50, kernel_size=filter_length, padding='same', activation='relu', strides=1) (emb_name)
        cnn_layer2 = Conv1D(filters=50, kernel_size=filter_length, padding='same', activation='relu', strides=1) (emb_item_desc)

        maxpool1 = GlobalMaxPooling1D() (cnn_layer1)
        maxpool2 = GlobalMaxPooling1D() (cnn_layer2)

        convs1.append(maxpool1)
        convs2.append(maxpool2)

    convs1 = concatenate(convs1)
    convs2 = concatenate(convs2)

    # main layers
    main_l = concatenate([
        Flatten() (emb_brand_name),
        Flatten() (emb_category_name),
        item_condition,
        convs1,
        convs2,
        num_vars,
    ])

    main_l = Dense(256)(main_l)
    main_l = Activation('elu')(main_l)

    main_l = Dense(128)(main_l)
    main_l = Activation('elu')(main_l)

    main_l = Dense(64)(main_l)
    main_l = Activation('elu')(main_l)

    # the output layer.
    output = Dense(1, activation="linear") (main_l)

    model = Model([name, item_desc, brand_name , category_name, item_condition, num_vars], output)

    optimizer = 'adam'
    model.compile(loss="mse", optimizer=optimizer)

    return model

model = cnn_model()
model.summary()
del model
```

```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 name (InputLayer)              [(None, 10)]         0           []

 item_desc (InputLayer)         [(None, 75)]         0           []

 embedding_10 (Embedding)       (None, 10, 20)       5181760     ['name[0][0]']

 embedding_11 (Embedding)       (None, 75, 60)       15545280    ['item_desc[0][0]']

 brand_name (InputLayer)        [(None, 1)]          0           []

 category_name (InputLayer)     [(None, 1)]          0           []

 conv1d (Conv1D)                (None, 10, 50)       1050        ['embedding_10[0][0]']

 conv1d_2 (Conv1D)              (None, 10, 50)       2050        ['embedding_10[0][0]']

 conv1d_4 (Conv1D)              (None, 10, 50)       3050        ['embedding_10[0][0]']

 conv1d_1 (Conv1D)              (None, 75, 50)       3050        ['embedding_11[0][0]']

 conv1d_3 (Conv1D)              (None, 75, 50)       6050        ['embedding_11[0][0]']

 conv1d_5 (Conv1D)              (None, 75, 50)       9050        ['embedding_11[0][0]']

 embedding_12 (Embedding)       (None, 1, 10)        48100       ['brand_name[0][0]']

 embedding_13 (Embedding)       (None, 1, 10)        12880       ['category_name[0][0]']

 global_max_pooling1d (GlobalMa  (None, 50)          0           ['conv1d[0][0]']
 xPooling1D)

 global_max_pooling1d_2 (Global  (None, 50)          0           ['conv1d_2[0][0]']
 MaxPooling1D)

 global_max_pooling1d_4 (Global  (None, 50)          0           ['conv1d_4[0][0]']
 MaxPooling1D)

 global_max_pooling1d_1 (Global  (None, 50)          0           ['conv1d_1[0][0]']
 MaxPooling1D)

 global_max_pooling1d_3 (Global  (None, 50)          0           ['conv1d_3[0][0]']
 MaxPooling1D)

 global_max_pooling1d_5 (Global  (None, 50)          0           ['conv1d_5[0][0]']
 MaxPooling1D)

 flatten_6 (Flatten)            (None, 10)           0           ['embedding_12[0][0]']

 flatten_7 (Flatten)            (None, 10)           0           ['embedding_13[0][0]']

 item_condition (InputLayer)    [(None, 1)]          0           []

 concatenate_2 (Concatenate)    (None, 150)          0           ['global_max_pooling1d[0][0]',
                                                                  'global_max_pooling1d_2[0][0]',
                                                                  'global_max_pooling1d_4[0][0]']

 concatenate_3 (Concatenate)    (None, 150)          0           ['global_max_pooling1d_1[0][0]',
                                                                  'global_max_pooling1d_3[0][0]',
                                                                  'global_max_pooling1d_5[0][0]']

 num_vars (InputLayer)          [(None, 1)]          0           []

 concatenate_4 (Concatenate)    (None, 322)          0           ['flatten_6[0][0]',
                                                                  'flatten_7[0][0]',
                                                                  'item_condition[0][0]',
                                                                  'concatenate_2[0][0]',
                                                                  'concatenate_3[0][0]',
                                                                  'num_vars[0][0]']

 dense_6 (Dense)                (None, 256)          82688       ['concatenate_4[0][0]']

 activation (Activation)        (None, 256)          0           ['dense_6[0][0]']

 dense_7 (Dense)                (None, 128)          32896       ['activation[0][0]']

 activation_1 (Activation)      (None, 128)          0           ['dense_7[0][0]']

 dense_8 (Dense)                (None, 64)           8256        ['activation_1[0][0]']

 activation_2 (Activation)      (None, 64)           0           ['dense_8[0][0]']

 dense_9 (Dense)                (None, 1)            65          ['activation_2[0][0]']

==================================================================================================
Total params: 20,936,225
Trainable params: 20,936,225
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [20]:

```
train['price'] = np.log1p(train["price"])
```

In [21]:

```
train_df, test_df = train_test_split(train, random_state=347, train_size=0.80)
train_df, dev_df = train_test_split(train_df, random_state=347, train_size=0.80)

Y_train = train_df.target.values.reshape(-11, 1)
Y_dev = dev_df.target.values.reshape(-1, 1)
```

In [22]:

```
n_trains = train_df.shape[0]
n_devs = dev_df.shape[0]
n_tests = test_df.shape[0]
print("Training:", n_trains, "examples")
print("Validating:", n_devs, "examples")
print("Testing:", n_tests, "examples")
```

```
Training: 948822 examples
Validating: 237206 examples
Testing: 296507 examples
```

In [23]:

```
full_df = pd.concat([train_df, dev_df, test_df])
```

In [24]:

```
le = LabelEncoder()

le.fit(full_df.category_name)
full_df.category_name = le.transform(full_df.category_name)

le.fit(full_df.brand_name)
full_df.brand_name = le.transform(full_df.brand_name)

del le
```

In [25]:

```
raw_text = np.hstack([full_df.item_description.str.lower(), full_df.name.str.lower()])

tok_raw = Tokenizer()
tok_raw.fit_on_texts(raw_text)

full_df['seq_item_description'] = tok_raw.texts_to_sequences(full_df.item_description.str.lower())
full_df['seq_name'] = tok_raw.texts_to_sequences(full_df.name.str.lower())

del tok_raw
```

In [26]:

```
MAX_NAME_SEQ = 10
MAX_ITEM_DESC_SEQ = 75
MAX_TEXT = np.max([
    np.max(full_df.seq_name.max()),
    np.max(full_df.seq_item_description.max()),
]) + 4
MAX_CATEGORY = np.max(full_df.category_name.max()) + 1
MAX_BRAND = np.max(full_df.brand_name.max()) + 1
MAX_CONDITION = np.max(full_df.item_condition_id.max()) + 1
```

In [27]:

```
def get_keras_data(df):
    X = {
        'name': pad_sequences(df.seq_name, maxlen=MAX_NAME_SEQ),
        'item_desc': pad_sequences(df.seq_item_description, maxlen=MAX_ITEM_DESC_SEQ),
        'brand_name': np.array(df.brand_name),
        'category_name': np.array(df.category_name),
        'item_condition': np.array(df.item_condition_id),
        'num_vars': np.array(df[["shipping"]]),
    }
    return X

train = full_df[:n_trains]
dev = full_df[n_trains:n_trains+n_devs]
test = full_df[n_trains+n_devs:]

X_train = get_keras_data(train)
X_dev = get_keras_data(dev)
X_test = get_keras_data(test)
```

In [28]:

```
BATCH_SIZE = 1024
epochs = 2

# Calculate learning rate decay.
exp_decay = lambda init, fin, steps: (init/fin)**(1/(steps-1)) - 1
steps = int(n_trains / BATCH_SIZE) * epochs
lr_init, lr_fin = 0.007, 0.0005
lr_decay = exp_decay(lr_init, lr_fin, steps)

model = cnn_model(lr=lr_init, decay=lr_decay)

print("Fitting CNN model to training examples...")
model.fit(
        X_train, Y_train, epochs=epochs, batch_size=BATCH_SIZE,
        validation_data=(X_dev, Y_dev), verbose=2,
)
```

```
Fitting CNN model to training examples...
Epoch 1/2
927/927 - 540s - loss: 0.0191 - val_loss: 0.0145 - 540s/epoch - 582ms/step
Epoch 2/2
927/927 - 600s - loss: 0.0123 - val_loss: 0.0142 - 600s/epoch - 647ms/step
```

Out[28]: <keras.callbacks.History at 0x1dead118370>

In [29]:

```
Y_dev_preds_cnn = model.predict(X_dev, batch_size=BATCH_SIZE)
```

RMSE

In [32]:

```
print('RMSE:', mean_squared_error(Y_dev, Y_dev_preds_cnn) ** 0.5)
```

```
RMSE: 0.11936483918975663
```

To Be Finished: user input product info and get prediction PS: have to go through preprocessing first

In [ ]:

```
userInput = [
    {'name': 'iPad',
    'item_condition_id': '1',
    'brand_name':'Apple',
    'price': 100,
    'shipping': 0,
    'item_description': 'No Des',
    'general_cat': 'Electronics',
    'subcat_1': 'Computers & Tablets',
    'subcat_2':'iPad'}
]

input_item = pd.DataFrame(userInput)
input_item
```

	name	item_condition_id	brand_name	price	shipping	item_description	general_cat	subcat_1	subcat_2
0	iPad	1	Apple	180	0	No Des	Electronics	Computers & Tablets	iPad

Closing Thoughts:

- Many pages of model files could cause redundancy in this report.
- I intended to build a content-based recommendation engine for this project. However, this may not be related to our objective, plus not sufficient time available.
- Optimization of different models could be the crucial part of this project because we can further simulate price changes in different scenarios, for example, price changes in B2B/B2C, Season Promotion, Inflation, supply and demand situations, and so on. Maybe I will search for more datasets to let this feature happen in the future.
- We chose XGBoost because it took less time to generate results, even though LGBM can offer us better evaluation scores, most optimization service providers (like Intel Optimization Environment) now do not support LGBM. Thus, if this is a real company case, for better future development considerations, XGB is a better choice.
- The library used in the XGBoost model part is xgboost. Unfortunately, this is not ideal since sklearn's xgboost do compatible with more services. For example, SHAP analysis, some evaluation dashboards and interactive web libraries do not support xgboost but sklearn's xgboost. This is one of my regrets after all this finished.
- There's a lot to improve in both the EDA and model parts. Interactive dashboards were built but due to the limitations of the presentation that we can't directly show in class, but hopefully we can be better in the future.

Cheers