Final_Imputation_v2

June 4, 2020

[0]:
```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import os
from nltk.corpus import stopwords
from tqdm import tqdm
import re
import gc
import time
import math
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder

import pickle
import scipy
import scipy.sparse
from scipy.sparse import hstack
from scipy.stats import uniform
from scipy.stats import randint as sp_randint

# Label Encoding Target Variables
from sklearn import preprocessing

# Regression Models
from sklearn.linear_model import Ridge
from sklearn.svm import SVR
from lightgbm import LGBMRegressor
```

1 Key Points

Since Brand Value already contains Missing Values we will not include it while while filling missing values for Category Name. Also in Production, we won't be having Price column and Price being a Real Valued Column, hence during imputation we need to skip Price Column as well.

[0]:
```
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
```

[4]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[4]: 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 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]:
```
# 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 train_evaluate(train_data):

  X_train,X_test = train_test_split(train_data,test_size = 0.25)

  X_train = X_train[(X_train.price >=3) & (X_train.price <= 2000)]

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

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

  # X_train = preprocess(train)
  # X_test = preprocess(test)

  del train_data
  gc.collect()

  start = time.time()

  Vectorizer = TfidfVectorizer(max_features=50000,ngram_range = (1,3),
                               min_df=25, 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 =(1,3),
                                min_df = 30, dtype=np.float32)
  Vectorizer.fit(X_train['item_description'].values)

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

  X_train['item_condition_id'] = X_train['item_condition_id'].astype('category')
  X_test['item_condition_id'] = X_test['item_condition_id'].astype('category')

  X_train['shipping'] = X_train['shipping'].astype('category')
  X_test['shipping'] = X_test['shipping'].astype('category')
```

```
train_dummies = scipy.sparse.csr_matrix(pd.
↪get_dummies(X_train[["item_condition_id",
                                                        "shipping"]],␣
↪sparse = True).values)

test_dummies = scipy.sparse.csr_matrix(pd.
↪get_dummies(X_test[["item_condition_id",
                                                        "shipping"]],␣
↪sparse = True).values)


unique_value = pd.Series("/".join(X_train["category_name"].unique().
↪astype("str")).split("/")).unique()

Vectorizer = CountVectorizer(vocabulary=unique_value,
                             lowercase = False, binary = True)

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

train_category_name = Vectorizer.transform(X_train['category_name'].values)
test_category_name = Vectorizer.transform(X_test['category_name'].values)

LabelEncoder = LabelBinarizer(sparse_output=True)
LabelEncoder.fit(X_train['brand_name'].values)

train_brand = LabelEncoder.transform(X_train['brand_name'].values)
test_brand = LabelEncoder.transform(X_test['brand_name'].values)


X_train_tfidf = hstack((X_train_name,X_train_text,train_category_name,
                        train_brand,train_dummies)).tocsr()
X_test_tfidf = hstack((X_test_name,X_test_text,test_category_name,
                       test_brand,test_dummies)).tocsr()

print("Time Taken to PreProcess : ", time.time() - start)

del␣
↪X_train_name,X_test_name,X_train_text,X_test_text,train_dummies,test_dummies
gc.collect()

del train_category_name,test_category_name,test_brand,train_brand
gc.collect()

# print("X_train TFIDF Shape : ",X_train_tfidf.shape)
# print("X_test TFIDF Shape : ",X_test_tfidf.shape)
# print("y train Shape : ", y_train.shape)
```

```
# print("y test Shape : ",y_test.shape)

################################### MODEL BUILDING #######################

print("Ridge Solver...")
alpha = [0.001,0.01,0.1,1,10,100]
train_rmsle = []
test_rmsle = []
for i in alpha:
  model = Ridge(solver = "lsqr", fit_intercept = False,alpha = i)
  model.fit(X_train_tfidf, y_train)
  y_pred = np.expm1(model.predict(X_test_tfidf))

  rmsle_train = rmsle_score(np.expm1(y_train),
                            np.expm1(model.predict(X_train_tfidf)))

  train_rmsle.append(rmsle_train)
  rmsle_test = rmsle_score(np.expm1(y_test),y_pred)
  test_rmsle.append(rmsle_test)

  print("Alpha : ", i)
  print("Train RMSLE Score : ", rmsle_train)
  print("Test RMSLE Score : ", rmsle_test)

  del model
  gc.collect()

test_rmsle = np.asarray(test_rmsle)
min_rmsle_index = np.argmin(test_rmsle)


print("Best Alpha Value : ", alpha[min_rmsle_index])
print("Best RMSLE Score : ", test_rmsle[min_rmsle_index])
```

[0]:

```
def impute_train_data(k):

  start = time.time()

  print("Imputing Category Name Values...")
  train_data = pd.read_csv("/content/drive/My Drive/train/train.tsv",sep = '\t')

  # Since Name and Description are Textual Columns Replacing all
  # the Missing/Null Values in them with empty spaces

  train_data['name'] = train_data['name'].replace([np.nan], '')
  train_data['item_description'] = train_data['item_description'].replace([np.
↪nan], '')
```

```python
train_data["brand_name"] = train_data["brand_name"].fillna("missing").
↪astype("category")

train_data['name'] = preprocessing_text(train_data['name'])
train_data['item_description'] =␣
↪preprocessing_text(train_data['item_description'])

# Finding out Entries which are NULL in Category_Name column
mask = train_data['category_name'].isnull()

# Separating them into 2 datasets accordingly
train_without_null = train_data[~mask]
train_with_null = train_data[mask]

del mask
gc.collect()

# As of Now Brand Column also contains NULL Values,Hence we will not use them
# Forming Train and Test Data
X_train =␣
↪train_without_null[['name','item_description','shipping','item_condition_id','brand_name']]
y_train = train_without_null[['category_name']]

X_test =␣
↪train_with_null[['name','item_description','shipping','item_condition_id','brand_name']]

# # Pre-processing Textual Columns in Train and Test Data
# X_train['name'] = preprocessing_text(X_train['name'])
# X_train['item_description'] =␣
↪preprocessing_text(X_train['item_description'])
# X_test['name'] = preprocessing_text(X_test['name'])
# X_test['item_description'] = preprocessing_text(X_test['item_description'])

# Vectorizing Text Data(Name) Using TF-IDF
Vectorizer = TfidfVectorizer(token_pattern='\w+',max_features=10000,dtype =␣
↪np.float32)
Vectorizer.fit(X_train['name'].values)

train_name_tfidf = Vectorizer.transform(X_train['name'].values)
test_name_tfidf = Vectorizer.transform(X_test['name'].values)

# Vectorizing Text Data(Description) using TF-IDF
Vectorizer = TfidfVectorizer(ngram_range =(1,2),token_pattern='\w+',
                             max_features=50000,dtype = np.float32)
Vectorizer.fit(X_train['item_description'].values)
```

```
train_desc_tfidf = Vectorizer.transform(X_train['item_description'].values)
test_desc_tfidf = Vectorizer.transform(X_test['item_description'].values)

# In order to One Hot Encode Shipping and Item Condition ID
# we need to convert it from int64 to category
X_train['item_condition_id'] = X_train['item_condition_id'].astype('category')
X_test['item_condition_id'] = X_test['item_condition_id'].astype('category')

X_train['shipping'] = X_train['shipping'].astype('category')
X_test['shipping'] = X_test['shipping'].astype('category')

train_dummies = scipy.sparse.csr_matrix(pd.
↪get_dummies(X_train[["item_condition_id", "shipping"]],
                                                     sparse = True).values)
test_dummies = scipy.sparse.csr_matrix(pd.
↪get_dummies(X_test[["item_condition_id", "shipping"]],
                                                     sparse = True).values)

LabelEncoder = LabelBinarizer(sparse_output=True)
LabelEncoder.fit(X_train['brand_name'].values)

train_brand = LabelEncoder.transform(X_train['brand_name'].values)
test_brand = LabelEncoder.transform(X_test['brand_name'].values)

# Stacking them all so as to form final dataframe
X_train = hstack((train_name_tfidf,train_desc_tfidf,train_brand,
                  train_dummies)).tocsr().astype('float32')
X_test = hstack((test_name_tfidf,test_desc_tfidf,test_brand,
                 test_dummies)).tocsr().astype('float32')

# Label Encoding the Target Column Category Name
le = preprocessing.LabelEncoder()
le.fit(y_train)
y_train_le = le.transform(y_train)

# print("X_Train Shape : ",X_train.shape)
# print("X_test Shape : ",X_test.shape)
# print("y_train Shape : ", y_train_le.shape)

del␣
↪train_name_tfidf,test_name_tfidf,train_desc_tfidf,test_desc_tfidf,train_dummies,test_dummie
gc.collect()

del train_brand,test_brand
gc.collect()
```

```
  KNN = KNeighborsClassifier(n_neighbors = k ,n_jobs = -1)

  KNN.fit(X_train,y_train_le)

  predictions = KNN.predict(X_test)

  # Inverse Tranform the Output Obtained so as to get true value
  predictions = le.inverse_transform(predictions)

  cat_df = pd.DataFrame(data = predictions.flatten(), columns=['category_name'])

  # Dropping "category_name" from the actual dataframe which contains NULL
  # values so as to replace them with predicted values

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

  train_with_null['category_name'] = cat_df['category_name'].values

  del cat_df,X_train,X_test
  gc.collect()

  # Concatenating Without NULL and Filled NULL Category Dataframes
  # so as to form New Training Data with no missing value in
  # category name

  train_data = pd.concat([train_with_null, train_without_null])

  # Sorting/Rearranging data based on Index
  train_data.sort_index(inplace=True)

  print("Imputing Done....")

  print("Time Required to Impute : ", time.time() - start)

  return train_data
```

[21]:
```
k_values = [5,7,9,11,13,15,17,19,21,23]
ridge_rmsle_values = []
svr_rmsle_values = []
gbm_rmsle_values = []
for k in k_values:
  train_data = impute_train_data(k)
  train_evaluate(train_data)
  print("K Value : ", k)
  print("*"* 50)
```

```
Imputing Category Name Values…
```

```
100%|      | 1482535/1482535 [00:28<00:00, 52384.07it/s]
100%|      | 1482535/1482535 [01:33<00:00, 15933.64it/s]

Imputing Done…
Time Required to Impute :  835.1371712684631
Time Taken to PreProcess :  336.1936089992523
Ridge Solver…
Alpha :  0.001
Train RMSLE Score :  0.6057386508537754
Test RMSLE Score :  0.6340677090363036
Alpha :  0.01
Train RMSLE Score :  0.6057386556355859
Test RMSLE Score :  0.6340640938708002
Alpha :  0.1
Train RMSLE Score :  0.6057391317640158
Test RMSLE Score :  0.6340284286463058
Alpha :  1
Train RMSLE Score :  0.6057847630271568
Test RMSLE Score :  0.6337184086225569
Alpha :  10
Train RMSLE Score :  0.6150108996254071
Test RMSLE Score :  0.6376989476772632
Alpha :  100
Train RMSLE Score :  0.6595073931021646
Test RMSLE Score :  0.6725480670808807
Best Alpha Value :  1
Best RMSLE Score :  0.6337184086225569
K Value :  5
**************************************************
Imputing Category Name Values…

100%|      | 1482535/1482535 [00:28<00:00, 52096.24it/s]
100%|      | 1482535/1482535 [01:33<00:00, 15907.24it/s]

Imputing Done…
Time Required to Impute :  840.9562397003174
Time Taken to PreProcess :  337.9943354129791
Ridge Solver…
Alpha :  0.001
Train RMSLE Score :  0.6055465682894907
Test RMSLE Score :  0.6331572373129212
Alpha :  0.01
Train RMSLE Score :  0.6055465730853514
Test RMSLE Score :  0.633153412336303
Alpha :  0.1
Train RMSLE Score :  0.6055470494906618
Test RMSLE Score :  0.6331156539857388
Alpha :  1
Train RMSLE Score :  0.6055926979455697
```

```
Test RMSLE Score :  0.6327851916117796
Alpha :  10
Train RMSLE Score :  0.6131548524137643
Test RMSLE Score :  0.6355761122058894
Alpha :  100
Train RMSLE Score :  0.6593740776930915
Test RMSLE Score :  0.6708404311867747
Best Alpha Value :  1
Best RMSLE Score :  0.6327851916117796
K Value :  7
**************************************************
Imputing Category Name Values…

100%|      | 1482535/1482535 [00:28<00:00, 52032.46it/s]
100%|      | 1482535/1482535 [01:33<00:00, 15854.75it/s]

Imputing Done…
Time Required to Impute :  848.9661891460419
Time Taken to PreProcess :  343.68534302711487
Ridge Solver…
Alpha :  0.001
Train RMSLE Score :  0.6066238503091517
Test RMSLE Score :  0.634079598136504
Alpha :  0.01
Train RMSLE Score :  0.6066238546644903
Test RMSLE Score :  0.6340762151969178
Alpha :  0.1
Train RMSLE Score :  0.6066242884040317
Test RMSLE Score :  0.6340428293447804
Alpha :  1
Train RMSLE Score :  0.6066659479736352
Test RMSLE Score :  0.633751580490771
Alpha :  10
Train RMSLE Score :  0.612951898304459
Test RMSLE Score :  0.6361219667383642
Alpha :  100
Train RMSLE Score :  0.6579354950341817
Test RMSLE Score :  0.6708324874786286
Best Alpha Value :  1
Best RMSLE Score :  0.633751580490771
K Value :  9
**************************************************
Imputing Category Name Values…

100%|      | 1482535/1482535 [00:28<00:00, 52282.42it/s]
100%|      | 1482535/1482535 [01:33<00:00, 15901.57it/s]

Imputing Done…
Time Required to Impute :  842.7494895458221
Time Taken to PreProcess :  345.1477143764496
```

```
Ridge Solver…
Alpha :  0.001
Train RMSLE Score :  0.6067728992269451
Test RMSLE Score :  0.6359073336831428
Alpha :  0.01
Train RMSLE Score :  0.6067729035968352
Test RMSLE Score :  0.6359038279717496
Alpha :  0.1
Train RMSLE Score :  0.606773338616957
Test RMSLE Score :  0.6358692152766997
Alpha :  1
Train RMSLE Score :  0.606815119095403
Test RMSLE Score :  0.635565786396354
Alpha :  10
Train RMSLE Score :  0.6130652213065153
Test RMSLE Score :  0.6376542658217662
Alpha :  100
Train RMSLE Score :  0.6574405002298676
Test RMSLE Score :  0.6714523378199445
Best Alpha Value :  1
Best RMSLE Score :  0.635565786396354
K Value :  11
**************************************************
Imputing Category Name Values…

100%|      | 1482535/1482535 [00:28<00:00, 52441.62it/s]
100%|      | 1482535/1482535 [01:33<00:00, 15899.87it/s]

Imputing Done…
Time Required to Impute :  839.3259518146515
Time Taken to PreProcess :  336.8516240119934
Ridge Solver…
Alpha :  0.001
Train RMSLE Score :  0.6056718021546633
Test RMSLE Score :  0.6323914169698552
Alpha :  0.01
Train RMSLE Score :  0.6056718069375493
Test RMSLE Score :  0.6323876579803147
Alpha :  0.1
Train RMSLE Score :  0.605672283325267
Test RMSLE Score :  0.6323505636868988
Alpha :  1
Train RMSLE Score :  0.606996412386224
Test RMSLE Score :  0.6325555329410705
Alpha :  10
Train RMSLE Score :  0.6132305716829194
Test RMSLE Score :  0.6347105750951152
Alpha :  100
Train RMSLE Score :  0.6573665257956042
```

```
Test RMSLE Score :  0.6692365242466507
Best Alpha Value :  0.1
Best RMSLE Score :  0.6323505636868988
K Value :  13
**************************************************
Imputing Category Name Values…

100%|      | 1482535/1482535 [00:28<00:00, 52182.44it/s]
100%|      | 1482535/1482535 [01:34<00:00, 15632.87it/s]

Imputing Done…
Time Required to Impute :  846.2030246257782
Time Taken to PreProcess :  339.42184591293335
Ridge Solver…
Alpha :  0.001
Train RMSLE Score :  0.6067161790064477
Test RMSLE Score :  0.6348431127554052
Alpha :  0.01
Train RMSLE Score :  0.6067161833749594
Test RMSLE Score :  0.6348395615067614
Alpha :  0.1
Train RMSLE Score :  0.606716618425943
Test RMSLE Score :  0.6348044959232794
Alpha :  1
Train RMSLE Score :  0.6067584017594798
Test RMSLE Score :  0.6344967750144757
Alpha :  10
Train RMSLE Score :  0.6146931507866946
Test RMSLE Score :  0.6375794316397136
Alpha :  100
Train RMSLE Score :  0.6580255624545699
Test RMSLE Score :  0.6707756290480199
Best Alpha Value :  1
Best RMSLE Score :  0.6344967750144757
K Value :  15
**************************************************
Imputing Category Name Values…

100%|      | 1482535/1482535 [00:28<00:00, 52070.83it/s]
100%|      | 1482535/1482535 [01:34<00:00, 15700.23it/s]

Imputing Done…
Time Required to Impute :  835.5364212989807
Time Taken to PreProcess :  330.6979606151581
Ridge Solver…
Alpha :  0.001
Train RMSLE Score :  0.6036788709160531
Test RMSLE Score :  0.6307092098154068
Alpha :  0.01
Train RMSLE Score :  0.603678876590873
```

```
Test RMSLE Score :  0.6307051062468132
Alpha :  0.1
Train RMSLE Score :  0.603679441467728
Test RMSLE Score :  0.6306646559640957
Alpha :  1
Train RMSLE Score :  0.6071984159821501
Test RMSLE Score :  0.6321786829846144
Alpha :  10
Train RMSLE Score :  0.6134876349047332
Test RMSLE Score :  0.6346264048443583
Alpha :  100
Train RMSLE Score :  0.6575883636614154
Test RMSLE Score :  0.6691118009104146
Best Alpha Value :  0.1
Best RMSLE Score :  0.6306646559640957
K Value :  17
**************************************************
Imputing Category Name Values…

100%|          | 1482535/1482535 [00:27<00:00, 53157.84it/s]
100%|          | 1482535/1482535 [01:31<00:00, 16129.54it/s]

Imputing Done…
Time Required to Impute :  829.8670189380646
Time Taken to PreProcess :  329.5446267127991
Ridge Solver…
Alpha :  0.001
Train RMSLE Score :  0.6055446982251649
Test RMSLE Score :  0.6338445856912512
Alpha :  0.01
Train RMSLE Score :  0.6055447029923758
Test RMSLE Score :  0.6338410432072705
Alpha :  0.1
Train RMSLE Score :  0.6055451770964164
Test RMSLE Score :  0.6338061019600971
Alpha :  1
Train RMSLE Score :  0.6067735257074461
Test RMSLE Score :  0.634065757794985
Alpha :  10
Train RMSLE Score :  0.6130414835190051
Test RMSLE Score :  0.6364142617393814
Alpha :  100
Train RMSLE Score :  0.6578987511769441
Test RMSLE Score :  0.6712429101300971
Best Alpha Value :  0.1
Best RMSLE Score :  0.6338061019600971
K Value :  19
**************************************************
Imputing Category Name Values…
```

```
100%|      | 1482535/1482535 [00:27<00:00, 53312.37it/s]
100%|      | 1482535/1482535 [01:31<00:00, 16180.68it/s]

Imputing Done…
Time Required to Impute :  822.0521638393402
Time Taken to PreProcess :  327.1083126068115
Ridge Solver…
Alpha :  0.001
Train RMSLE Score :  0.601743970109293
Test RMSLE Score :  0.6303083043203002
Alpha :  0.01
Train RMSLE Score :  0.6017439764658437
Test RMSLE Score :  0.6303037746910489
Alpha :  0.1
Train RMSLE Score :  0.6017446074757806
Test RMSLE Score :  0.630259135470246
Alpha :  1
Train RMSLE Score :  0.6018046265242052
Test RMSLE Score :  0.6298753027262303
Alpha :  10
Train RMSLE Score :  0.613042110544989
Test RMSLE Score :  0.6348892174133967
Alpha :  100
Train RMSLE Score :  0.6570946577580428
Test RMSLE Score :  0.6690841910452032
Best Alpha Value :  1
Best RMSLE Score :  0.6298753027262303
K Value :  21
**************************************************
Imputing Category Name Values…

100%|      | 1482535/1482535 [00:27<00:00, 53833.46it/s]
100%|      | 1482535/1482535 [01:30<00:00, 16321.31it/s]

Imputing Done…
Time Required to Impute :  822.0949087142944
Time Taken to PreProcess :  336.8709673881531
Ridge Solver…
Alpha :  0.001
Train RMSLE Score :  0.6061217077645183
Test RMSLE Score :  0.6314360874613002
Alpha :  0.01
Train RMSLE Score :  0.6061217125787131
Test RMSLE Score :  0.6314323969043804
Alpha :  0.1
Train RMSLE Score :  0.6061221917514664
Test RMSLE Score :  0.6313959866523269
Alpha :  1
Train RMSLE Score :  0.607392335741845
```

```
Test RMSLE Score :  0.6317768647231308
Alpha :  10
Train RMSLE Score :  0.6110721958885249
Test RMSLE Score :  0.6323939339189367
Alpha :  100
Train RMSLE Score :  0.6586586700029775
Test RMSLE Score :  0.6691404297755539
Best Alpha Value :  0.1
Best RMSLE Score :  0.6313959866523269
K Value :  23
**************************************************
```

Observation

[28]:

```python
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["K Values","Best Alpha", "Best Test RMSLE Score"]

x.add_row([5,1,0.6334])
x.add_row([7,1,0.633])
x.add_row([9,1,0.633])
x.add_row([11,1,0.635])
x.add_row([13,0.1,0.632])
x.add_row([15,1,0.634])
x.add_row([17,0.1,0.630])
x.add_row([19,0.1,0.633])
x.add_row([21,1,0.629])
x.add_row([23,0.1,0.631])

print(x)
```

```
+----------+------------+-----------------------+
| K Values | Best Alpha | Best Test RMSLE Score |
+----------+------------+-----------------------+
|    5     |     1      |         0.6334        |
|    7     |     1      |         0.633         |
|    9     |     1      |         0.633         |
|    11    |     1      |         0.635         |
|    13    |    0.1     |         0.632         |
|    15    |     1      |         0.634         |
|    17    |    0.1     |          0.63         |
|    19    |    0.1     |         0.633         |
|    21    |     1      |         0.629         |
|    23    |    0.1     |         0.631         |
+----------+------------+-----------------------+
```

[0]:

```

```

2 Without Imputation

[7]:

```
data = pd.read_csv("/content/drive/My Drive/train/train.tsv",sep = '\t')
train,test = train_test_split(data,test_size = 0.25)
print("Train Data Shape : ",train.shape)
print("Test Data Shape : ",test.shape)
```

```
Train Data Shape :  (1111901, 8)
Test Data Shape :  (370634, 8)
```

[8]:

```
train = train[(train.price >=3) & (train.price <= 2000)]
train.shape
```

[8]:

```
(1111222, 8)
```

[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], '')
```

[10]:

```
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%|      | 1111222/1111222 [00:26<00:00, 42575.79it/s]
100%|      | 370634/370634 [00:08<00:00, 42391.92it/s]
100%|      | 1111222/1111222 [01:19<00:00, 13895.03it/s]
100%|      | 370634/370634 [00:26<00:00, 13960.30it/s]
```

[13]:

```
Vectorizer = TfidfVectorizer(max_features=50000,ngram_range = (1,3),
                              min_df=25, dtype=np.float32)
Vectorizer.fit(train['preprocess_name'].values)

train_name = Vectorizer.transform(train['preprocess_name'].values)
test_name = Vectorizer.transform(test['preprocess_name'].values)

Vectorizer = TfidfVectorizer(max_features=100000,ngram_range =(1,3),
                               min_df = 30, dtype=np.float32)
Vectorizer.fit(train['preprocess_desc'].values)

train_desc = Vectorizer.transform(train['preprocess_desc'].values)
test_desc = Vectorizer.transform(test['preprocess_desc'].values)
```

```
print("After Name Vectorization : ")
print(train_name.shape)
print(test_name.shape)

print("After Description Vectorization : ")
print(train_desc.shape)
print(test_desc.shape)
```

```
After Name Vectorization :
(1111222, 30602)
(370634, 30602)
After Description Vectorization :
(1111222, 100000)
(370634, 100000)
```

[0]:
```
# First Fill all the Missing Brand Name with value "missing"
train["brand_name"] = train["brand_name"].fillna("missing").astype("category")
test["brand_name"] = test["brand_name"].fillna("missing").astype("category")
```

[15]:
```
LabelEncoder = LabelBinarizer(sparse_output=True)
LabelEncoder.fit(train['brand_name'].values)

train_brand = LabelEncoder.transform(train['brand_name'].values)
test_brand = LabelEncoder.transform(test['brand_name'].values)

print(train_brand.shape)
print(test_brand.shape)
```

```
(1111222, 4456)
(370634, 4456)
```

[0]:
```
# Fill all the Missing Value with value "missing"
train["category_name"] = train["category_name"].fillna("missing").
 ↪astype("category")
test["category_name"] = test["category_name"].fillna("missing").
 ↪astype("category")
```

[18]:
```
# First We will find all the Unique Values in the Category Name
# Then we will one hot encode it using CountVectorizer
unique_value = pd.Series("/".join(train["category_name"].unique().
 ↪astype("str")).split("/")).unique()
Vectorizer = CountVectorizer(vocabulary=unique_value,
                             lowercase = False, binary= True)

Vectorizer.fit(train['category_name'].values)
train_category_name = Vectorizer.transform(train['category_name'].values)
```

```
test_category_name = Vectorizer.transform(test['category_name'].values)

print(train_category_name.shape)
print(test_category_name.shape)
```

```
(1111222, 936)
(370634, 936)
```

[0]:
```
# In order to One Hot Encode Shipping and Item Condition ID
# we need to convert it from int64 to category
train['item_condition_id'] = train['item_condition_id'].astype('category')
test['item_condition_id'] = test['item_condition_id'].astype('category')

train['shipping'] = train['shipping'].astype('category')
test['shipping'] = test['shipping'].astype('category')
```

[20]:
```
train_dummies = scipy.sparse.csr_matrix(pd.
 ↪get_dummies(train[["item_condition_id", "shipping"]],
                                                       sparse = True).values)
test_dummies = scipy.sparse.csr_matrix(pd.
 ↪get_dummies(test[["item_condition_id", "shipping"]],
                                                      sparse = True).values)

print(train_dummies.shape)
print(test_dummies.shape)
```

```
(1111222, 7)
(370634, 7)
```

[0]:
```
# Transforming price -> log(1 + price)
train['log_price'] = np.log1p(train['price'])
test['log_price'] = np.log1p(test['price'])
```

[24]:
```
# Taking out the Log Price from input Data

y_train = train['log_price'].values
y_test = test['log_price'].values

print(y_train.shape)
print(y_test.shape)
```

```
(1111222,)
(370634,)
```

[25]:
```
X_train = hstack((train_name,train_desc,train_brand,
                  train_category_name,train_dummies)).tocsr().astype('float32')
```

```
X_test = hstack((test_name,test_desc,test_brand,
                 test_category_name,test_dummies)).tocsr().astype('float32')

print("X_train Shape : ",X_train.shape)
print("y_train Shape : ",y_train.shape)
print("X_test Shape : ",X_test.shape)
print("y_test Shape : ",y_test.shape)
```

```
X_train Shape :  (1111222, 136001)
y_train Shape :  (1111222,)
X_test Shape :  (370634, 136001)
y_test Shape :  (370634,)
```

[26]:
```
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
train_rmsle = []
test_rmsle = []
for i in alpha:
  model = Ridge(solver = "lsqr", fit_intercept=False,alpha = i)
  model.fit(X_train, y_train)
  y_pred = np.expm1(model.predict(X_test))
  rmsle_train = rmsle_score(np.expm1(y_train),np.expm1(model.predict(X_train)))
  train_rmsle.append(rmsle_train)
  rmsle_test = rmsle_score(np.expm1(y_test),y_pred)
  test_rmsle.append(rmsle_test)
  print("Alpha : ",i)
  print("Train RMLSE : ",rmsle_train)
  print("Test RMSLE : ",rmsle_test)
  print()
```

```
Alpha :  0.0001
Train RMLSE :  0.4541258865478164
Test RMSLE :  0.4735915316628679

Alpha :  0.001
Train RMLSE :  0.4541258865882285
Test RMSLE :  0.47359131112900277

Alpha :  0.01
Train RMLSE :  0.45412588988334035
Test RMSLE :  0.47358914652593437

Alpha :  0.1
Train RMLSE :  0.45412615385344485
Test RMSLE :  0.4735677576392346

Alpha :  1
Train RMLSE :  0.45415262657291106
Test RMSLE :  0.4733808120158091
```

```
Alpha :  10
Train RMLSE :  0.46544490361520063
Test RMSLE :  0.47989992284415284

Alpha :  100
Train RMLSE :  0.4928966571364844
Test RMSLE :  0.5022832065693176

Alpha :  1000
Train RMLSE :  0.5562787584812299
Test RMSLE :  0.56234218570441

Alpha :  10000
Train RMLSE :  0.6358851191202811
Test RMSLE :  0.6413605189686417
```

Observation: Best Alpha Value : 1 Best Test RMSLE Score : 0.4733

[0]:

```

```