Trip Advisor Hotel Reviews - Case Study

Explore Hotel aspects and Predict the rating of each review.

Hotels play a crucial role in traveling and with the increased access to information new pathways of selecting the best ones emerged. With this dataset, consisting of 20k reviews crawled from Tripadvisor, you can explore what makes a great hotel and maybe even use this model in your travels!

This Case Study is taken from : https://www.kaggle.com/andrewmvd/trip-advisor-hotel-reviews

Acknowledgements

Citation Alam, M. H., Ryu, W.-J., Lee, S., 2016. Joint multi-grain topic sentiment: modeling semantic aspects for online reviews. Information Sciences 339, 206–223. DOI https://zenodo.org/record/1219899#.X64_imgzbIU

Dataset link: https://www.kaggle.com/andrewmvd/trip-advisor-hotel-reviews

In [ ]:

```

```

In [ ]:

```

```

Summary of my Idea and what I did.

**The problem given is to predict whether the "review" given is either good or bad by using "rating" as class label.
How good the model predicts is to be found based on the metrics given i.e MAE and RMSE.
We can generally understand that if the rating is 3 and above, it is good or else it is bad.
In this task, we have to predict the rating based on the review which should be automaticlly done by our model and that rating which inturn decides the review is either good or bad. As, we have 1-5 rating range for every review, we have to predict the rating in that range only by classification.
This means, the problem is a "Muti-Class Classification" problem.
We can use classification ML techniques like SVM, Decision Trees, Random Forest, Boosting techniques etc., and Deep Learning models too using certain type of loss and activation function.
Finally based on the metrics we have to decide which model to use. The metric we use is F1-Score and it's given to use RMSE and MAE.
I've covered 3 models which are Linear SVM, LGBMClassifer and a Deep Learning Model and analysed results.**

In [ ]:

```
# Importing the packages

%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
```

```
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

In [ ]:

```
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from plotly import graph_objs as go
import plotly.figure_factory as ff
import seaborn as sns
from bs4 import BeautifulSoup
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection  import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
```

Reading the file

In [ ]:

```
df = pd.read_csv("tripadvisor_hotel_reviews.csv")
df.head()
```

Out[ ]:

	Review	Rating
0	nice hotel expensive parking got good deal sta...	4
1	ok nothing special charge diamond member hilto...	2
2	nice rooms not 4* experience hotel monaco seat...	3
3	unique, great stay, wonderful time hotel monac...	5
4	great stay great stay, went seahawk game aweso...	5

Exploratory Data Analysis

Shape of the file

In [ ]:

```
df.shape
```

Out[ ]:

```
(20491, 2)
```

There are 20491 rows and 2 columns.

In [ ]:

```

```

Checking the type of columns

In [ ]:

```
df.dtypes
```

Out[ ]:

```
Review     object
Rating      int64
dtype: object
```

Checking for null values

In [ ]:

```
df[df.isnull().any(axis=1)]
```

Out[ ]:

Review	Rating

In [ ]:

```
df.isnull().sum()
```

Out[ ]:

```
Review     0
Rating     0
dtype: int64
```

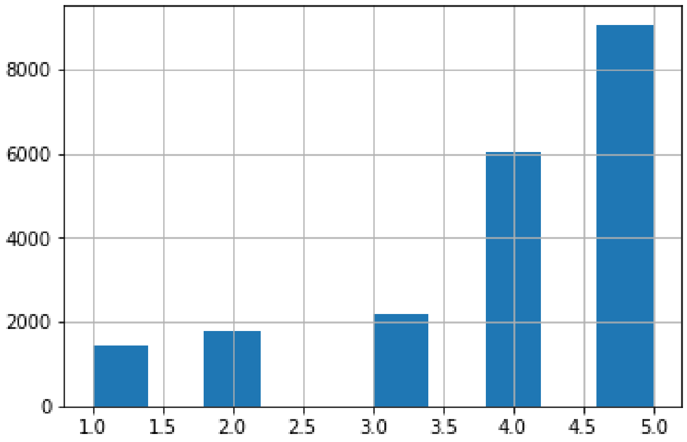
Histogram of frequency of data points according to the rating

In [ ]:

```
df['Rating'].hist()
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fea304e668>
```



In [ ]:

```

```

Getting the count of data points of individual rating

In [ ]:

```
df['Rating'].value_counts()
```

```
df['Rating'].value_counts()
```

Out[ ]:

```
5    9054
4    6039
3    2184
2    1793
1    1421
Name: Rating, dtype: int64
```

In [ ]:

```

```

The distribution of the above values

In [ ]:

```
class_dist = df['Rating'].value_counts()

def ditribution_plot(x,y,name):
    fig = go.Figure([
        go.Bar(x=x, y=y)
    ])

    fig.update_layout(title_text=name)
    fig.show()
```

In [ ]:

```
ditribution_plot(x= class_dist.index, y= class_dist.values, name= 'Class Distribution')
```

In [ ]:

```

```

Top words in the Reviews corpus

In [ ]:

```
from wordcloud import WordCloud
```

In [ ]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [ ]:

```

```

In [ ]:

```
vectorizer = TfidfVectorizer()
vectorizer.fit_transform(df['Review'])
```

Out[ ]:

```
<20491x52923 sparse matrix of type '<class 'numpy.float64'>'
 with 1705964 stored elements in Compressed Sparse Row format>
```

In [ ]:

```
word_list = vectorizer.get_feature_names()
```

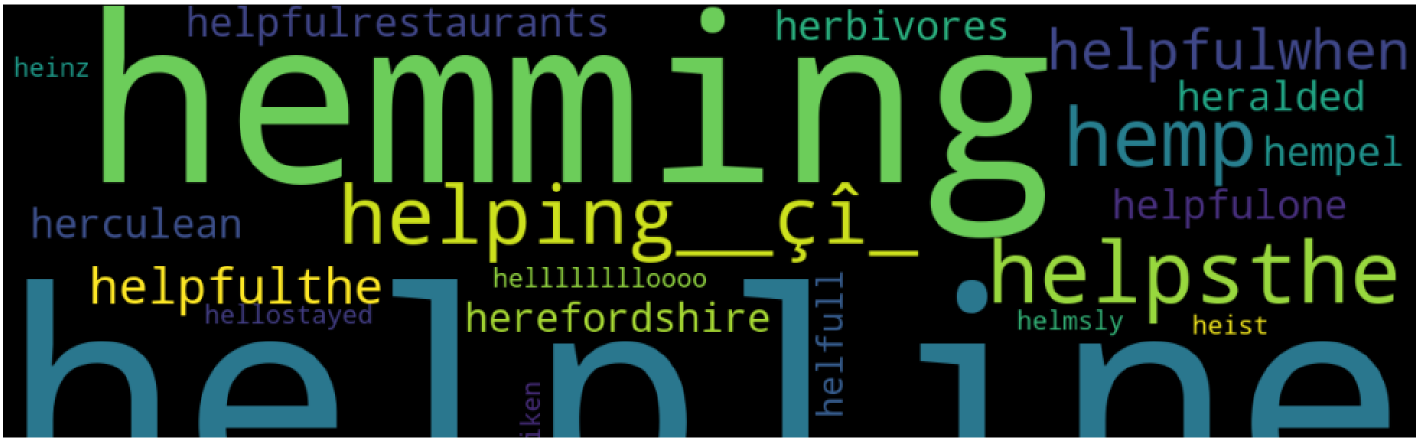
In [ ]:

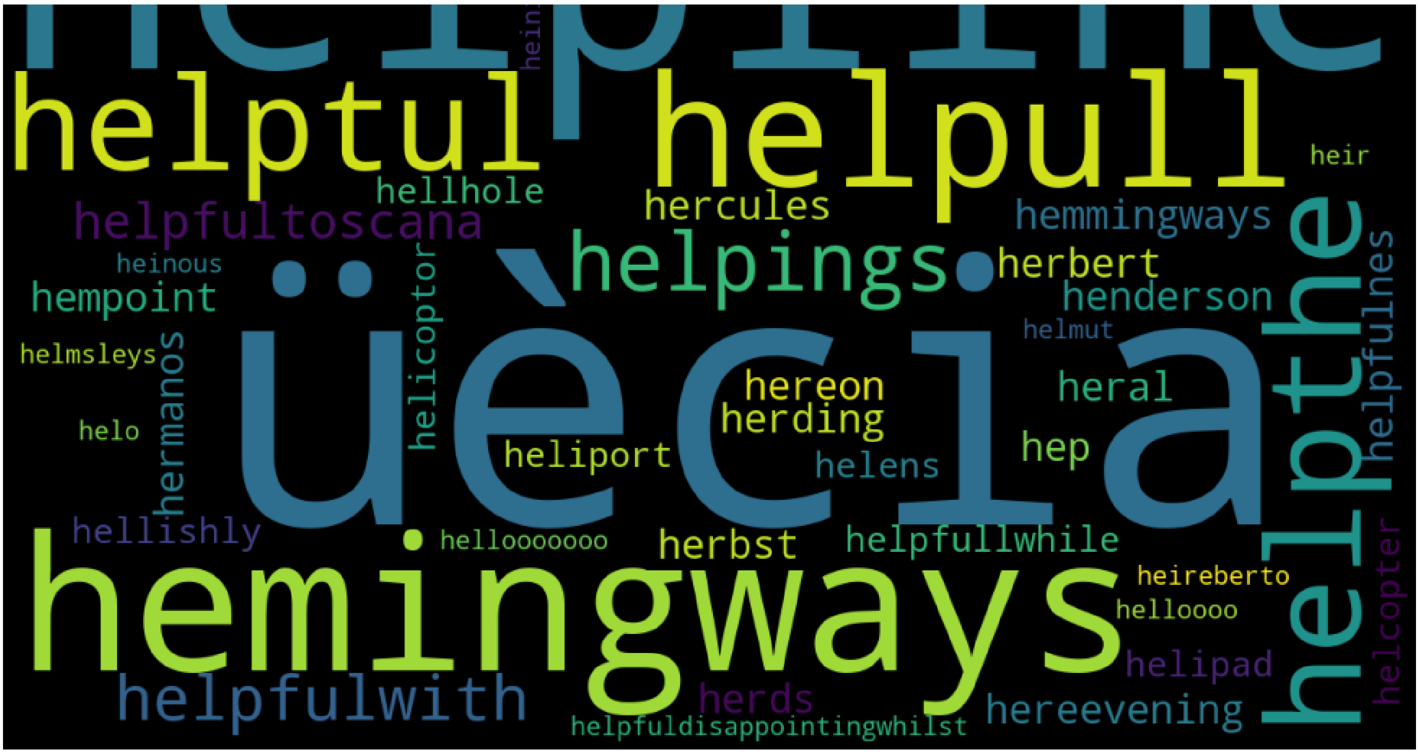
```
words2=[]
idf2=vectorizer.idf_
features=np.argsort(idf2)[::-1]
for i in features[0:60]:
    words2.append(word_list[i])
print(words2)
```

```
['üècia', 'helpline', 'hemming', 'hemingways', 'helpull', 'helptul', 'helpthe', 'helpsthe
', 'helpings', 'hemp', 'helping__çî_', 'helpfulwith', 'helpfulwhen', 'helpfultoscana', 'h
elpfulthe', 'helpfulrestaurants', 'hemmingways', 'hempel', 'hermanos', 'herculean', 'here
on', 'herefordshire', 'hereevening', 'herds', 'herding', 'hercules', 'herbst', 'hempoint'
, 'herbivores', 'herbert', 'heralded', 'heral', 'hep', 'henderson', 'helpfulone', 'helpfu
lnes', 'helpfullwhile', 'helens', 'hellishly', 'hellhole', 'heliport', 'helipad', 'helico
ptor', 'helfull', 'helcopter', 'helpfuldisappointingwhilst', 'heist', 'heireberto', 'heir
', 'heinz', 'heinous', 'heiniken', 'helllllllloooo', 'helloooo', 'hellooooooo', 'hellosta
yed', 'helmsleys', 'helmsly', 'helmut', 'helo']
```

In [ ]:

```
from wordcloud import WordCloud
from wordcloud import WordCloud
wordcloud = WordCloud(width = 1200, height = 1000).generate(" ".join(words2))
plt.figure(figsize = (20, 15))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```





In [ ]:

In [ ]:

Data Preprocessing

In [ ]:

```
import re

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
```

In [ ]:

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/cloudportal456/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[ ]:

```
True
```

In [ ]:

```
from nltk.corpus import stopwords
stopwords = set(stopwords.words('english'))
```

In [ ]:

```
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(df['Review'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopword
s)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 20491/20491 [00:08<00:00, 2393.44it/s]
```

In [ ]:

In [ ]:

```
preprocessed_reviews[232]
```

Out[ ]:

```
'friendly staff comfortable beds stayed hotel tour included hotel coast california enjoye
d hotel great location tourists close market place attractions desk staff accommodating t
hanks christopher rest hotel staff friendly loved beds comfortable breakfast included tou
r enjoyed want included buffet fruit fresh hi morgan wait staff hotel bright clean lookin
g fresh flowers lobby area seattle try make short trip boeing aircraft factory loved tour
people seattle portland willing help tourists directions suggestions'
```

In [ ]:

Perform Modelling

In [ ]:

```
le = LabelEncoder()
Y = le.fit_transform(df[r'Rating'])
```

In [ ]:

```
X = np.array(preprocessed_reviews)
```

Performing Train_Test Split

In [ ]:

```
X_train,X_cv,y_train,y_cv = train_test_split(X,Y,test_size=0.2)
X_train,X_test,y_train,y_test = train_test_split(X_train,y_train,test_size=0.2)
```

In [ ]:

```
print(X_train.shape)
print(X_cv.shape)
print(X_test.shape)
```

```
(13113,)
(4099,)
(3279,)
```

Using Count Vectozier with Bi-Grams

In [ ]:

```
count_vect = CountVectorizer(lowercase=True,ngram_range=(1, 2))
X_train=count_vect.fit_transform(X_train)
X_cv=count_vect.transform(X_cv)
X_test=count_vect.transform(X_test)

scalar = StandardScaler(with_mean = False)
X_train_bow = scalar.fit_transform(X_train)
X_test_bow= scalar.transform(X_test)
X_cv_bow=scalar.transform(X_cv)
```

In [ ]:

```

```

In [ ]:

```
'''alphas = [10**-3,10**-2, 10**-1,1,10,10**2,10**3,10**4,10**6]
scores = []
for i in alphas:
    model = SGDClassifier(alpha = i,loss="hinge")
    model.fit(X_train_bow, y_train)
    scores.append(model.score(X_cv, y_cv))'''
```

In [ ]:

```
#optimal_alpha= alphas[scores.index(max(scores))]
#optimal_alpha
```

Out[ ]:

```
0.01
```

Fine tuning Linear SVM Model

In [ ]:

```
#https://stackoverflow.com/questions/55893734/how-can-i-use-sgdclassifier-hinge-loss-with
-gridsearchcv-using-log-loss-metric

#%%time
grid_params =  {'base_estimator__alpha': [ 10**-3,10**-2,10**-1,1,10,10**2,10**3]}
clf = SGDClassifier(loss='hinge')
calibrated_clf = CalibratedClassifierCV(base_estimator=clf, method='sigmoid', cv=3)
svm_model = GridSearchCV(calibrated_clf, param_grid=grid_params, cv=5)
svm_model.fit(X_train_bow, y_train)
```

Out[ ]:

```
GridSearchCV(cv=5,
             estimator=CalibratedClassifierCV(base_estimator=SGDClassifier(),
                                              cv=3),
             param_grid={'base_estimator__alpha': [0.001, 0.01, 0.1, 1, 10, 100,
                                                   1000]})
```

In [ ]:

```

```

In [ ]:

```
print(svm_model.best_params_)
```

```
{'base_estimator__alpha': 1}
```

Predicting the results after Fine Tuning

In [ ]:

```
clf = SGDClassifier(alpha=1, fit_intercept=True,
       learning_rate='optimal', loss='hinge',
        verbose=0, warm_start=False)
clf.fit(X_train_bow,y_train)
tr_pred = clf.predict(X_train_bow)
cv_pred = clf.predict(X_cv_bow)
```

In [ ]:

```
te_pred = clf.predict(X_test_bow)
```

In [ ]:

```
print(confusion_matrix(y_cv,cv_pred))
```

```
[[ 127   42   11   29   70]
 [  50   49   38   84  130]
 [  23   19   34  159  220]
 [   4    9    6  264  928]
 [   2    2    2  130 1667]]
```

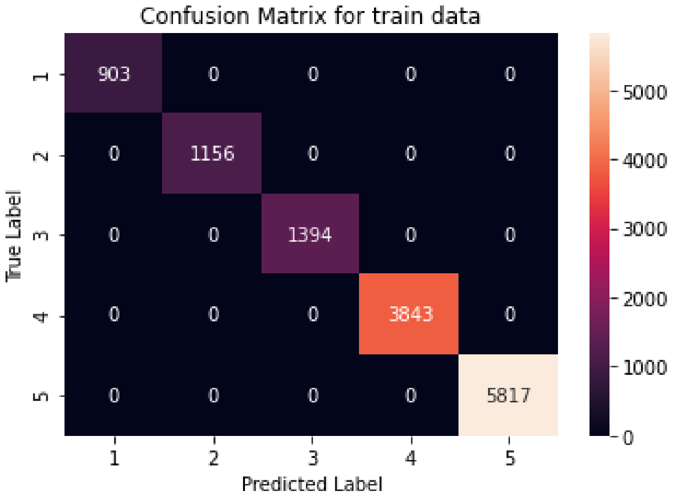
In [ ]:

```

```

Confusion Matrix of Train data

In [ ]:

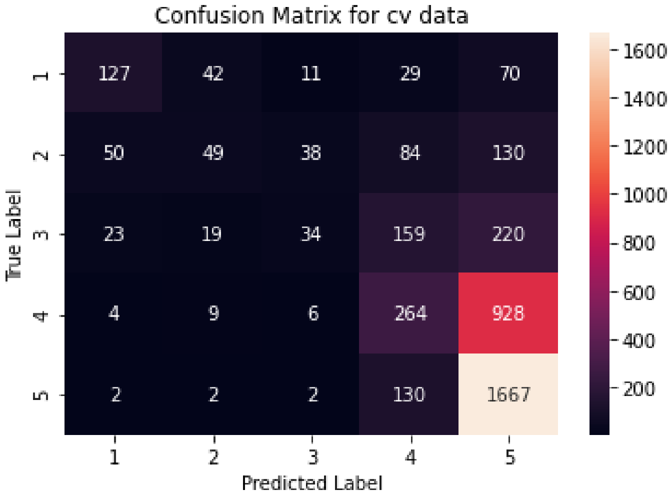
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

conf_mat = confusion_matrix(y_train, tr_pred)
class_label = ["1", "2", "3", "4", "5"]
df = pd.DataFrame(conf_mat, index = class_label, columns = class_label)
sns.heatmap(df, annot = True,fmt="d")
plt.title("Confusion Matrix for train data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



Confusion Matrix of CV data

In [ ]:

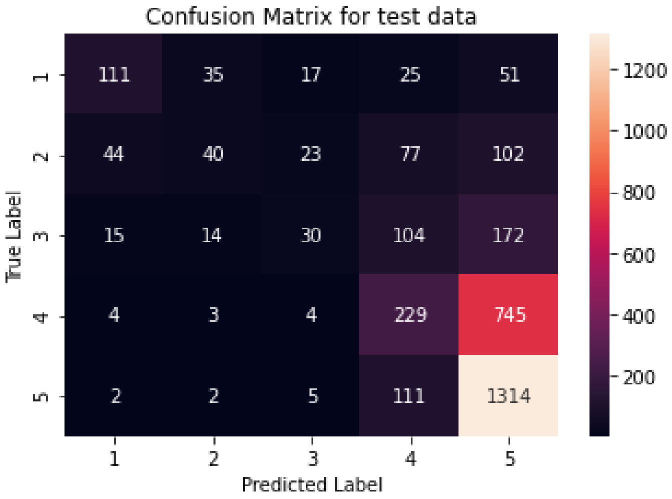
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns

conf_mat = confusion_matrix(y_cv, cv_pred)
class_label = ["1", "2", "3", "4", "5"]
df = pd.DataFrame(conf_mat, index = class_label, columns = class_label)
sns.heatmap(df, annot = True,fmt="d")
plt.title("Confusion Matrix for cv data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



Confusion Matrix of Test data

In [ ]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

conf_mat = confusion_matrix(y_test, te_pred)
class_label = ["1", "2", "3", "4", "5"]
df = pd.DataFrame(conf_mat, index = class_label, columns = class_label)
sns.heatmap(df, annot = True,fmt="d")
plt.title("Confusion Matrix for test data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



In [ ]:

F1-Score

In [ ]:

```
from sklearn.metrics import precision_score,recall_score,f1_score
```

In [ ]:

```
print("Train ",f1_score(y_train,tr_pred,average='micro'))
print("cv ",f1_score(y_cv,cv_pred,average='micro'))
print("Test ",f1_score(y_test,te_pred,average='micro'))
```

```
Train  1.0
cv  0.5223225176872408
Test  0.5257700518450747
```

RMSE

In [ ]:

```
rms = sqrt(mean_squared_error(y_test, te_pred))
print(rms)
```

```
1.142620221161812
```

Mean Abolute Deviation Error

In [ ]:

```
from sklearn.metrics import mean_absolute_error
mae=mean_absolute_error(y_test, te_pred)
print(mae)
```

```
0.6919792619701128
```

In [ ]:

In [ ]:

In [ ]:

Fine Tuning LGBMClassifier

In [ ]:

```
from lightgbm import LGBMRegressor,LGBMClassifier
from sklearn.model_selection import GridSearchCV
```

In [ ]:

```
grid={
 'max_depth': [30,40,45],
 'n_estimators': [4000,4500],
 'learning_rate':[0.1,0.2]}
```

In [ ]:

```
%%time
clf= LGBMClassifier(colsample_bytree=0.8,subsample=0.9,min_child_samples=50,num_leaves=20
)
rf_random = GridSearchCV(estimator = clf, param_grid = grid,
                               cv=3, verbose=1)
rf_random.fit(X_train_bow,y_train,verbose=True)
```

In [ ]:

```
bestpar=rf_random.best_params_
bestpar
```

Out[ ]:

```
{'learning_rate': 0.1, 'max_depth': 30, 'n_estimators': 4000}
```

Predicting the results after Fine Tuning

In [ ]:

```
clf=LGBMClassifier(num_leaves=10,colsample_bytree=0.8,subsample=0.9,min_child_samples=50,
                   learning_rate= 0.1, max_depth= 20, n_estimators= 4000)
clf.fit(X_train_bow,y_train)
predt=clf.predict(X_train_bow)
```

In [ ]:

```
predcv=clf.predict(X_cv_bow)
```

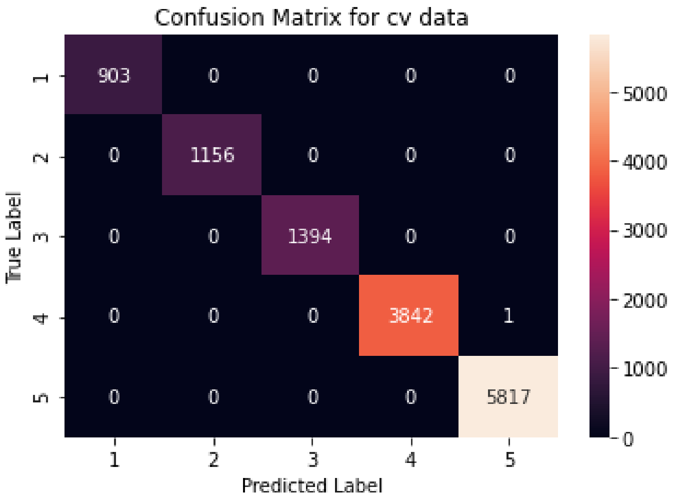
In [ ]:

```
predte = clf.predict(X_test_bow)
```

Confusion Matrix of Train data

In [ ]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

conf_mat = confusion_matrix(y_train, predt)
class_label = ["1", "2", "3", "4", "5"]
df = pd.DataFrame(conf_mat, index = class_label, columns = class_label)
sns.heatmap(df, annot = True,fmt="d")
plt.title("Confusion Matrix for cv data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

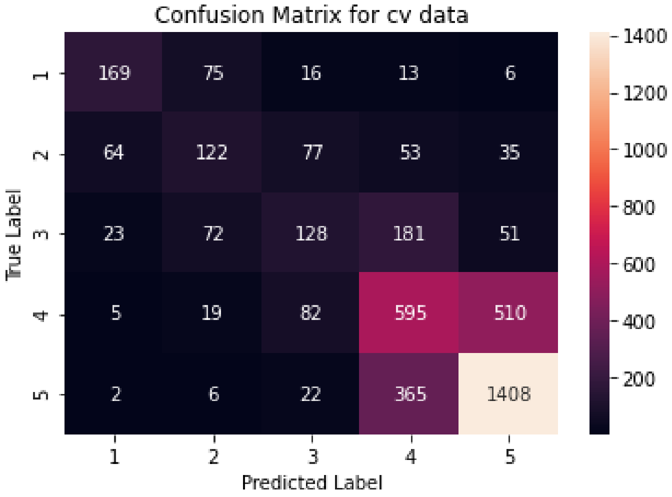


Confusion Matrix of CV data

In [ ]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

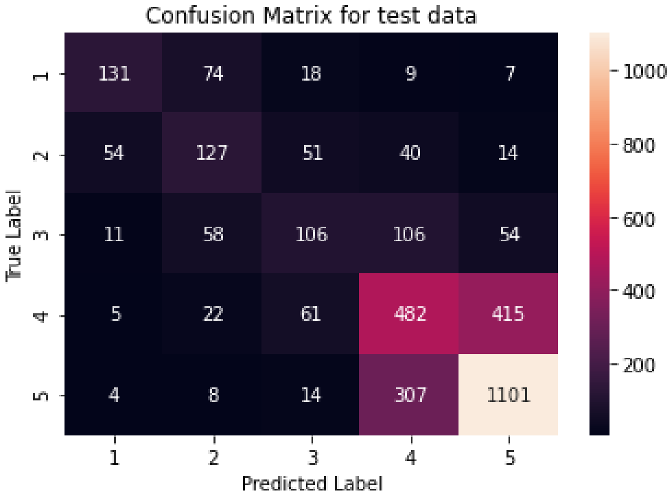
```
conf_mat = confusion_matrix(y_cv, predcv)
class_label = ["1", "2", "3", "4", "5"]
df = pd.DataFrame(conf_mat, index = class_label, columns = class_label)
sns.heatmap(df, annot = True,fmt="d")
plt.title("Confusion Matrix for cv data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



Confusion Matrix of Test data

In [ ]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

conf_mat = confusion_matrix(y_test, predte)
class_label = ["1", "2", "3", "4", "5"]
df = pd.DataFrame(conf_mat, index = class_label, columns = class_label)
sns.heatmap(df, annot = True,fmt="d")
plt.title("Confusion Matrix for test data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



In [ ]:

In [ ]:

```
predte
```

Out[ ]:

```
array([1, 4, 4, ..., 0, 4, 4])
```

F1-Scores of the data

In [ ]:

```
print("The train f1-score is", f1_score(y_train,predt,average='micro'))
print("The cv f1-score is", f1_score(y_cv,predcv,average='micro'))
print("The test f1-score is", f1_score(y_test,predte,average='micro'))
```

```
The train f1-score is 0.9999237398001983
The cv f1-score is 0.5908758233715541
The test f1-score is 0.5937785910338518
```

RMSE

In [ ]:

```
rms = sqrt(mean_squared_error(y_test, predte))
print(rms)
```

```
0.830568759812122
```

Mean Absolute Error

In [ ]:

```
from sklearn.metrics import mean_absolute_error
mae=mean_absolute_error(y_test, predte)
print(mae)
```

```
0.4867337602927722
```

In [ ]:

In [ ]:

In [ ]:

Training Deep Learning Model

In [ ]:

```
import plotly.figure_factory as ff
import gc

from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
import json
from tensorflow.keras.preprocessing import text, sequence
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
from tensorflow.keras import layers
from keras.layers import Reshape,Concatenate
from tensorflow.keras.layers import Reshape
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Embedding
from tensorflow.keras import regularizers
from tensorflow.keras.layers import LeakyReLU
```

In [ ]:

```

```

Tokenizing the data

In [ ]:

```
encoding = {1: 0,
            2: 1,
            3: 2,
            4: 3,
            5: 4
           }

#labels = ['1', '2', '3', '4', '5']

y = df['Rating'].copy()
y.replace(encoding, inplace=True)
```

In [ ]:

```
X_train,X_cv,y_train,y_cv = train_test_split(X,y,test_size=0.2, random_state=42, stratif
y=y)
X_train,X_test,y_train,y_test = train_test_split(X_train,y_train,test_size=0.2)
```

In [ ]:

```
tokenizer = Tokenizer(lower=False, num_words=80000)

for text in tqdm(X_train):
    tokenizer.fit_on_texts(text.split(" "))
```

In [ ]:

```
pickle.dump(tokenizer, open("tokenizertripadv.pickel", "wb"))
```

In [ ]:

```
tokenizer = pickle.load(open("tokenizertripadv.pickel","rb"))
```

In [ ]:

```
max_length = max([len(x) for x in X])
vocab_size = len(tokenizer.word_index)+1
exp_sen = 1
```

In [ ]:

```
max_length
```

Out[ ]:

```
12610
```

In [ ]:

```

```

In [ ]:

In [ ]:

```

```

Converting text to sequences

In [ ]:

```
def compute_text(X_train,X_cv,X_test,tokenizer):

    #train_text = tokenizer.texts_to_sequences(X_train.text.values)
    train = tokenizer.texts_to_sequences(X_train)
    cv = tokenizer.texts_to_sequences(X_cv)
    test = tokenizer.texts_to_sequences(X_test)

    #train_text = sequence.pad_sequences(train_text, maxlen=300)
    train = sequence.pad_sequences(train,maxlen=max_length)
    cv = sequence.pad_sequences(cv,maxlen=max_length)
    test = sequence.pad_sequences(test,maxlen=max_length)

    return train,cv,test
```

In [ ]:

```
train,cv,test = compute_text(X_train,X_cv,X_test,tokenizer)
```

Using Glove Vectors as Pre-trianed word vectors embedding

In [ ]:

```
!wget --header="Host: doc-0c-as-docs.googleusercontent.com" --header="User-Agent: Mozill
a/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.42
40.198 Safari/537.36" --header="Accept: text/html,application/xhtml+xml,application/xml;q
=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9" -
-header="Accept-Language: en-US,en;q=0.9" --header="Cookie: AUTH_cl0pg8h2f8j20p62kq5685pg
au6hs4qu_nonce=c4su9akftthqm" --header="Connection: keep-alive" "https://doc-0c-as-docs.g
oogleusercontent.com/docs/securesc/nodkv35vnf4s6shndpmb1esh0jpiumep/mkosuet6vnn1h9niuhjp3
0t6k674gcm1/1605784575000/00484516897554883881/09523152760876890323/1pGd5tLwA30M7wkbJKdXH
aae9tYVDICJ_?e=download&authuser=0&nonce=c4su9akftthqm&user=09523152760876890323&hash=0j1
tfi7ip7l2p9t6cp61tl28o6rcec1j" -c -O 'glove_vectors'
```

```
--2020-11-19 11:17:25--  https://doc-0c-as-docs.googleusercontent.com/docs/securesc/nodkv
35vnf4s6shndpmb1esh0jpiumep/mkosuet6vnn1h9niuhjp30t6k674gcm1/1605784575000/00484516897554
883881/09523152760876890323/1pGd5tLwA30M7wkbJKdXHaae9tYVDICJ_?e=download&authuser=0&nonce
=c4su9akftthqm&user=09523152760876890323&hash=0j1tfi7ip7l2p9t6cp61tl28o6rcec1j
Resolving doc-0c-as-docs.googleusercontent.com (doc-0c-as-docs.googleusercontent.com)...
173.194.217.132, 2607:f8b0:400c:c13::84
Connecting to doc-0c-as-docs.googleusercontent.com (doc-0c-as-docs.googleusercontent.com)
|173.194.217.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/octet-stream]
Saving to: ‘glove_vectors’

glove_vectors           [       <=>          ] 121.60M  75.2MB/s    in 1.6s

2020-11-19 11:17:27 (75.2 MB/s) - ‘glove_vectors’ saved [127506004]
```

In [ ]:

```
with open('./glove_vectors', 'rb') as f:
 glove=pickle.load(f)
 glove_words=set(glove.keys())

embedd_matrix= np.zeros((len(tokenizer.word_index)+1,300))
for i,j in tokenizer.word_index.items():
  if i in glove_words:
   embed_vec=glove[i]
   embedd_matrix[j]=embed_vec
print(embed_vec.shape,embedd_matrix.shape)
```

```
(300,) (38956, 300)
```

In [ ]:

```
cv.shape
```

Out[ ]:

```
(4099, 12610)
```

In [ ]:

```

```

In [ ]:

```
from tensorflow.keras.layers import Input, Dense, Embedding, SpatialDropout1D, concatenate, Masking
from tensorflow.keras.layers import LSTM, Bidirectional, GlobalMaxPooling1D, Dropout
from tensorflow.keras.preprocessing import text, sequence
from tqdm import tqdm_notebook as tqdm
import tensorflow as tf
import tensorflow.keras
import pickle
import tensorflow.keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Model
```

In [ ]:

```
k = tf.keras.initializers.glorot_normal(seed=None)
```

In [ ]:

```
train.shape[1:]
```

Out[ ]:

```
(12610,)
```

Model

In [ ]:

```
text_in = Input(shape=train.shape[1:],name='input1')
t= Embedding(*embedd_matrix.shape, weights=[embedd_matrix])(text_in)
#t = Embedding(vocab_size, embedding_dim)(text_in)

t= layers.Bidirectional(tf.keras.layers.LSTM(64,use_bias=True,return_sequences=True))(t)
#t =   tf.keras.layers.Bidirectional(tf.keras.layers.GRU(64,use_bias=True,
#                 recurrent_dropout=0.2, return_sequences=True))(t)
t=tf.keras.layers.LeakyReLU(alpha=0.4)(t)
t= tf.keras.layers.GlobalMaxPooling1D()(t)
#t= tf.keras.layers.MaxPool1D()(t)
#t= tensorflow.keras.layers.Flatten()(t)

hidden = Dense(256, activation='relu',kernel_initializer=k)(t)
hidden=tf.keras.layers.LeakyReLU(alpha=0.4)(hidden)
hidden = Dropout(0.4)(hidden)
hidden = Dense(196, activation='relu',kernel_initializer=k)(hidden)
hidden=tf.keras.layers.LeakyReLU(alpha=0.4)(hidden)
hidden = Dropout(0.5)(hidden)
hidden = Dense(250, activation='relu',kernel_initializer=k)(hidden)
hidden = Dropout(0.4)(hidden)
hidden = Dense(196, activation='relu',kernel_initializer=k)(hidden)



out1 = Dense(5, activation='softmax',name='out1')(hidden)
model = Model(inputs=[text_in], outputs=[out1])

model.compile(loss="sparse_categorical_crossentropy",
              optimizer= tf.keras.optimizers.Adamax(learning_rate=0.01),
              metrics=['accuracy'])
model.summary()
```

```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input1 (InputLayer)          [(None, 12610)]           0         
_________________________________________________________________
embedding_1 (Embedding)      (None, 12610, 300)        11686800  
_________________________________________________________________
bidirectional_1 (Bidirection (None, 12610, 128)        186880    
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 12610, 128)        0         
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 256)               33024     
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 256)               0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 196)               50372     
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 196)               0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 196)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 250)               49250     
_________________________________________________________________
dropout_5 (Dropout)          (None, 250)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 196)               49196     
_________________________________________________________________
out1 (Dense)                 (None, 5)                 985       
=================================================================
Total params: 12,056,507
Trainable params: 12,056,507
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
print(5)
```

```
5
```

In [ ]:

```
EPOCHS = 5
BATCH_SIZE = 50
```

Training the model

In [ ]:

```
history = model.fit(train, y_train, epochs=EPOCHS, validation_split=0.3, batch_size=BATC
H_SIZE,
                    verbose=1)
```

```
Epoch 1/5
Epoch 1/5
184/184 [==============================] - 284s 2s/step - loss: 1.1611 - accuracy: 0.4736
- val_loss: 0.9718 - val_accuracy: 0.5676
Epoch 2/5
184/184 [==============================] - 283s 2s/step - loss: 0.8947 - accuracy: 0.5904
- val_loss: 0.8608 - val_accuracy: 0.6057
Epoch 3/5
184/184 [==============================] - 284s 2s/step - loss: 0.7617 - accuracy: 0.6614
- val_loss: 0.8799 - val_accuracy: 0.6202
Epoch 4/5
184/184 [==============================] - 282s 2s/step - loss: 0.6508 - accuracy: 0.7153
- val_loss: 0.8754 - val_accuracy: 0.6317
Epoch 5/5
184/184 [==============================] - 283s 2s/step - loss: 0.5396 - accuracy: 0.7743
- val_loss: 0.9957 - val_accuracy: 0.6182
```

The training is taking more time than expected. So, I'm skipping for now.

In [ ]:

```
history.history['accuracy']
```

Out[ ]:

```
[0.4837128221988678,
 0.6069288849830627,
 0.6785053014755249,
 0.7350473999977112,
 0.8007408380508423]
```

In [ ]:

```

```

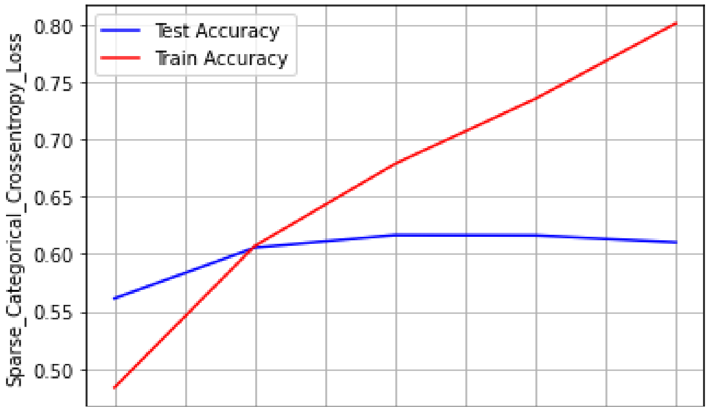
Accuracy plot

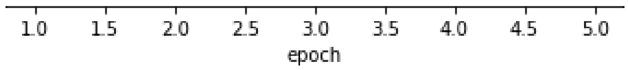
In [ ]:

```
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Test Accuracy")
    ax.plot(x, ty, 'r', label="Train Accuracy")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [ ]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Sparse_Categorical_Crossentropy_Loss')

# list of epoch numbers
x = list(range(1,5+1))

vy = history.history['val_accuracy']
ty = history.history['accuracy']
plt_dynamic(x, vy, ty, ax)
```





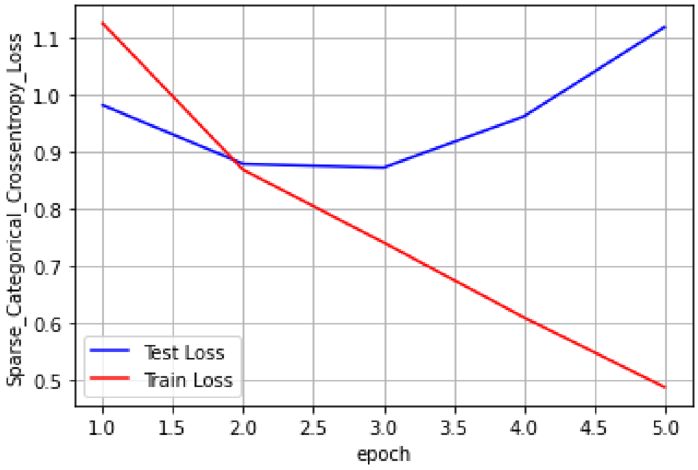
Loss Plot

In [ ]:

```
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Test Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [ ]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Sparse_Categorical_Crossentropy_Loss')

# list of epoch numbers
x = list(range(1,5+1))

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```



In [ ]:

In [ ]:

In [ ]:

```
result = model.evaluate(test,y_test,batch_size=50)
```

```
66/66 [==============================] - 35s 527ms/step - loss: 0.9636 - accuracy: 0.6127
```

In [ ]:

```
result[0]
```

Out[ ]:

```
0.9636454582214355
```

In [ ]:

```
pred = model.predict(test)
```

In [ ]:

```
len(pred)
```

Out[ ]:

```
3279
```

In [ ]:

```
labels = ['1', '2', '3', '4', '5']
```

In [ ]:

```
model.save("tripadv")
```

Appending all the values to list

In [ ]:

```
l=[]
for i in pred:
    l.append(list(i))
```

In [ ]:

```
l2=[]
for i in l:
    l2.append(i.index(max(i)))
print(len(l2))
```

```
3279
```

In [ ]:

```

```

F1-Score

In [ ]:

```
print("The test f1-score is", f1_score(y_test,l2,average='micro'))
```

```
The test f1-score is 0.6126867947544983
```

In [ ]:

```

```

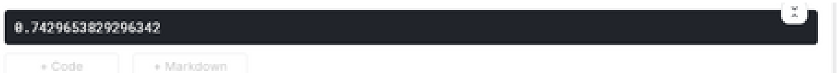
RMSE

In [ ]:

```
from sklearn.metrics import mean_squared_error
```

In [ ]:

```
rms = np.sqrt(mean_squared_error(y_test, l2))
print(rms)
```



```
0.7429653829296342
```

MAE

In [ ]:

```
from sklearn.metrics import mean_absolute_error
mae=mean_absolute_error(y_test, l2)
print(mae)
```

```
0.43671851174138454
```

In [ ]:

```

```

Conclusion

In [3]:

```
from prettytable import PrettyTable

x=PrettyTable()

x.field_names=(['Model','Test F1-Score','MAE','RMSE'])
x.add_row(['Linear SVM','0.525','0.691','1.142'])
x.add_row(['LGBMClassifier','0.593','0.486','0.830'])
x.add_row(['LSTM Model','0.612','0.436','0.742'])
print(x)
```

```
+----------------+---------------+-------+-------+
|     Model      | Test F1-Score |  MAE  |  RMSE |
+----------------+---------------+-------+-------+
|   Linear SVM   |     0.525     | 0.691 | 1.142 |
| LGBMClassifier |     0.593     | 0.486 | 0.830 |
|   LSTM Model   |     0.612     | 0.436 | 0.742 |
+----------------+---------------+-------+-------+
```

By the above results we can say that LSTM Model gave good results though the results can improved further by other techniques. Next, LGBMClassifier performed better compared to Linear SVM.

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```