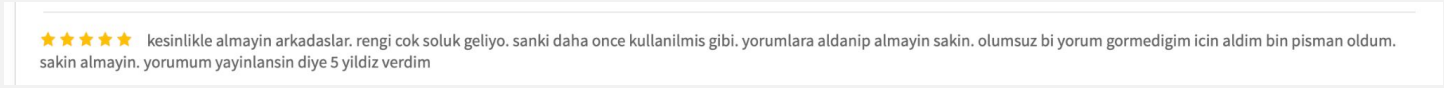
CAPSTONE PROJECT

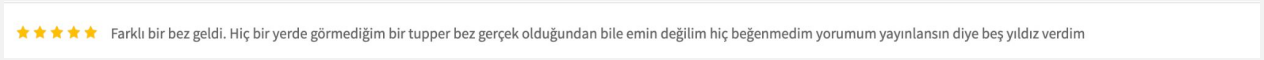
Ece Simsek

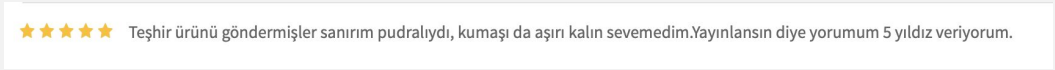
13 Jul, 2021

Text Classification on Amazon Customer Reviews Dataset

TABLE OF CONTENTS



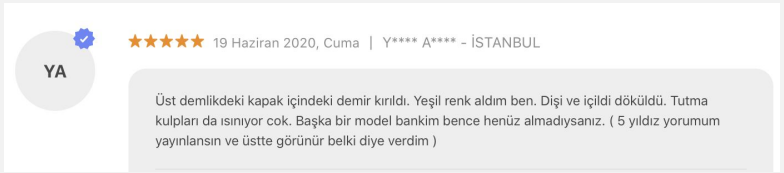


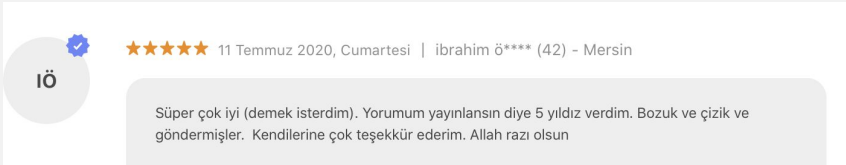


MOTIVATION

The spread of e-commerce and the increase in product diversity have created a need for customers to choose the right and most suitable product. Therefore, when customers want to choose the most suitable product for themselves and their budget, the reviews of other customers about their experiences with that product have become more valuable. So, it is possible to say that customer reviews provide an objective feedback to the customer who will buy the product in his purchasing process.

The main thing that reflects the feedback is actually the customer review **text itself**.





GOALS

The **main purposes** of the project are,

To analyze the reviews made by customers using NLP techniques and try to predict the most accurate rating based on the meaning of the text.

Establish a rate prediction model based on customer reviews using NLP techniques in order to show more consistent and helpful comments to customers in most appropriate order.

BACKGROUND

- All Beauty Category - Amazon Reviews Dataset
- Every Amazon product review is summarized by a numerical rating which is an integer from **one to five stars**
- The main thing that reflects the feedback is the **text itself, not rating**
- Variable of “Overall” (rating) is multi-class label for supervised text classification
- Customer reviews texts are the core predictor for ML-DL Models.

BACKGROUND

01

NLP Preprocessing

Tokenization, Lemmatization, removing stop words, punctuations, extra spaces etc.

02

NLP Techniques

Bag of Words, TF-IDF, Word Embedding, LDA, Topic Modelling

03

Libraries

NLTK, Gensim, Tensorflow, Pandas, Numpy, Matplotlib, Seaborn, Sklearn

04

ML - DL Algorithms

Random Forest, Decision Tree, Extra Tree Classifier, XGBoost, LSTM, LSTM + CNN

AMAZON DATASET

ALL BEAUTY

The category of “All Beauty” from 29 different categories

371,345 Reviews

371,345 Customer Reviews

12 FEATURES

- **reviewerID** - ID of the reviewer
- **asin** - ID of the product
- **reviewerName** - name of the reviewer
- **vote** - helpful votes of the review
- **style** - a dictionary of the product metadata
- **reviewText** - text of the review
- **overall** - rating of the product
- **summary** - summary of the review
- **unixReviewTime** - time of the review (unix time)
- **reviewTime** - time of the review (raw)
- **image** - images that users post after they have received the product
- **verified** - information whether the customer who made the review purchased the product or not

MAY 1996 - OCT 2018

Reviews spanning May 1996 - Oct 2018

184 MB

184 MB Total Size

JSON

The dataset in JSON Format

AMAZON DATASET

	overall	verified	reviewTime	reviewerID	asin	reviewerName	reviewText	summary	unixReviewTime	vote	style	image
0	1.0	True	02 19, 2015	A1V6B6TNIC10QE	0143026860	theodore j bigham	great	One Star	1424304000	NaN	NaN	NaN
1	4.0	True	12 18, 2014	A2F5GHSXFQ0W6J	0143026860	Mary K. Byke	My husband wanted to reading about the Negro ...	... to reading about the Negro Baseball and th...	1418860800	NaN	NaN	NaN
2	4.0	True	08 10, 2014	A1572GUYS7DGSR	0143026860	David G	This book was very informative, covering all a...	Worth the Read	1407628800	NaN	NaN	NaN
3	5.0	True	03 11, 2013	A1PSGLFK1NSVO	0143026860	TamB	I am already a baseball fan and knew a bit abo...	Good Read	1362960000	NaN	NaN	NaN
4	5.0	True	12 25, 2011	A6IKXKZMTKGSC	0143026860	shoecanary	This was a good story of the Black leagues. I ...	More than facts, a good story read!	1324771200	5	NaN	NaN

Customer reviews for “Beauty” products from May 1996 up to Oct 2018

371,345 reviews by 324,038 customers on 32,586 unique products

NLP PREPROCESSING

- The final dataframe for the model will be drawn from the "reviewText" column
- The "overall" column will serve as the labels for the classification

Sample reviewText:

```
df["reviewText"][300102]
```

```
"The product was supposed to be new. The package had been opened before. Now that I have tried it, the poor staying
quality tells me that this is not the real Dermacol foundation. Other differences are fake has no color number on t
ube, smaller lid circumferance & longer lid on the fake, real lid has gold fleck appearance, liquid pours out of fa
ke tube upon opening, fake tube's length is longer than real tube. The shade color is not the same as the real Derm
acol 221 I purchased from different seller."
```

NLP PREPROCESSING - LEMMATIZATION

WordNetLemmatizer from NLTK library is used.

```
import nltk
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
```

```
df["reviewText"][300102]
```

```
"The product was supposed to be new. The package had been opened before. Now that I have tried it, the poor staying quality tells me that this is not the real Dermacol foundation. Other differences are fake has no color number on t
ube, smaller lid circumferance & longer lid on the fake, real lid has gold fleck appearance, liquid pours out of fa
ke tube upon opening, fake tube's length is longer than real tube. The shade color is not the same as the real Derm
acol 221 I purchased from different seller."
```

After Lemmatization:

```
"The product be suppose to be new . The package have be open before . Now that I have try it , the poor stay qualit
y tell me that this be not the real Dermacol foundation . Other difference be fake have no color number on tube , s
mall lid circumferance & long lid on the fake , real lid have gold fleck appearance , liquid pour out of fake tube
upon opening , fake tube 's length be long than real tube . The shade color be not the same as the real Dermacol 22
1 I purchase from different seller ."
```

```
CPU times: user 12min 15s, sys: 22.6 s, total: 12min 38s
Wall time: 12min 39s
```

NLP PREPROCESSING - NORMALIZATION

Removing digits, punctuations, stop words, extra spaces characters and converting it the lower case:

Converting to Lower Case

```
df_lemmatized["reviewText"] = df_lemmatized["reviewText"].str.lower()
```

Removing the Stop Words

```
stop_words = stopwords.words("english")

df_lemmatized["reviewText"] = df_lemmatized["reviewText"].apply(str)


remove_stop_words = lambda row: " ".join([token for token in row.split(" ") \
                                 if token not in stop_words])
df_lemmatized["reviewText"] = df_lemmatized["reviewText"].apply(remove_stop_words)
```

Keeping only Alphanumeric Chars and White Spaces and Removing Extra Spaces

```
df_lemmatized["reviewText"] = df_lemmatized["reviewText"].str.replace(pat=r"[^\w\s]", repl=" ", regex=True)
df_lemmatized["reviewText"] = df_lemmatized["reviewText"].str.replace(pat=r"[\d]", repl=" ", regex=True)
df_lemmatized["reviewText"] = df_lemmatized["reviewText"].str.replace(pat=r"[\s]+", repl=" ", regex=True)
df_lemmatized["reviewText"] = df_lemmatized["reviewText"].str.replace(pat=r"[!\"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~]",
```

```
"The product be suppose to be new . The package have be open before . Now that I have try it , the poor stay qualit
y tell me that this be not the real Dermacol foundation . Other difference be fake have no color number on tube , s
mall lid circumferance & long lid on the fake , real lid have gold fleck appearance , liquid pour out of fake tube
upon opening , fake tube 's length be long than real tube . The shade color be not the same as the real Dermacol 22
1 I purchase from different seller ."
```

After Normalization:

```
'product suppose new package open try poor stay quality tell real dermacol foundation difference fake color number
tube small lid circumferance long lid fake real lid gold fleck appearance liquid pour fake tube upon opening fake t
ube s length long real tube shade color real dermacol purchase different seller '
```

NLP PREPROCESSING - TOKENIZATION

Tokenization

```
df['reviewText'][300102]
```

```
"The product was supposed to be new. The package had been opened before. Now that I have tried it, the poor staying
quality tells me that this is not the real Dermacol foundation. Other differences are fake has no color number on t
ube, smaller lid circumferance & longer lid on the fake, real lid has gold fleck appearance, liquid pours out of fa
ke tube upon opening, fake tube's length is longer than real tube. The shade color is not the same as the real Derm
acol 221 I purchased from different seller."
```

```
df_lemmatized['reviewText'][256547]
```

```
'product suppose new package open try poor stay quality tell real dermacol foundation difference fake color number
tube small lid circumferance long lid fake real lid gold fleck appearance liquid pour fake tube upon opening fake t
ube s length long real tube shade color real dermacol purchase different seller '
```

```
corpora = df_lemmatized["reviewText"].values
tokenized = [corpus.split(" ") for corpus in corpora]

print(tokenized[256547])
```

```
['product', 'suppose', 'new', 'package', 'open', 'try', 'poor', 'stay', 'quality', 'tell', 'real', 'dermacol', 'fou
ndation', 'difference', 'fake', 'color', 'number', 'tube', 'small', 'lid', 'circumferance', 'long', 'lid', 'fake',
'real', 'lid', 'gold', 'fleck', 'appearance', 'liquid', 'pour', 'fake', 'tube', 'upon', 'opening', 'fake', 'tube',
's', 'length', 'long', 'real', 'tube', 'shade', 'color', 'real', 'dermacol', 'purchase', 'different', 'seller', '']
```

COUNT-BASED FEATURE ENGINEERING (BOW)

- For a machine learning model to work with text input, documents must first be vectorized or converted into containers of numerical values
- Bag of Words is the classical approach of getting token frequency per document

Vocabulary Creation for BOW:

```
from gensim.corpora.dictionary import Dictionary
```

Vocabulary Creation

```
vocabulary = Dictionary(tokenized)
```

```
vocabulary.token2id
```

```
 'anyone': 29,
 'audience': 30,
 'black': 31,
 'buy': 32,
 'class': 33,
 'continue': 34,
 'enjoy': 35,
 'exciting': 36,
 'fact': 37,
 'find': 38,
 'good': 39,
 'high': 40,
 'history': 41,
 'interested': 42,
 'mckissack': 43,
 'recommend': 44,
 's': 45,
 'school': 46,
 'story': 47,
 'teach': 48,
```

Bag of Words (BOW)

```
bow = [vocabulary.doc2bow(doc) for doc in tokenized]
```

Sample Bag of Words model:

```
for idx, freq in bow[256547]:
    print(f"Word: {vocabulary.get(idx)}, Freq: {freq}")
```

```
Word: , Freq: 1
Word: s, Freq: 1
Word: color, Freq: 2
Word: number, Freq: 1
Word: tell, Freq: 1
Word: product, Freq: 1
Word: small, Freq: 1
Word: seller, Freq: 1
Word: purchase, Freq: 1
Word: quality, Freq: 1
Word: appearance, Freq: 1
Word: long, Freq: 2
Word: open, Freq: 1
Word: foundation, Freq: 1
Word: stay, Freq: 1
Word: try, Freq: 1
Word: package, Freq: 1
Word: opening, Freq: 1
Word: different, Freq: 1
Word: new, Freq: 1
Word: poor, Freq: 1
Word: suppose, Freq: 1
Word: real, Freq: 4
Word: difference, Freq: 1
Word: lid, Freq: 3
Word: length, Freq: 1
Word: gold, Freq: 1
Word: liquid, Freq: 1
Word: upon, Freq: 1
Word: shade, Freq: 1
Word: pour, Freq: 1
Word: tube, Freq: 4
Word: fake, Freq: 4
Word: fleck, Freq: 1
Word: dermacol, Freq: 2
Word: circumferance, Freq: 1
```

COUNT-BASED FEATURE ENGINEERING (TF - IDF)

- Term Frequency-Inverse Document Frequency (TF-IDF) is another approach where continuous values are assigned to tokens.
- The **term frequency** is a raw count of instances a word appears in a document. The **inverse document frequency** means, how common or rare a word is in the entire document set. The closer it is to 0, the more common a word is. So, if the word is very common and appears in many documents, this number will approach 0. Otherwise, it will approach 1.
- Multiplying these two numbers results in the TF-IDF score of a word in a document. The higher the score, the more relevant that word is in that particular document.

```
from gensim.models.tfidfmodel import TfidfModel
```

TF-IDF

```
tfidf = TfidfModel(bow)

for idx, weight in tfidf[bow[256547]]:
    print(f"Word: {vocabulary.get(idx)}, Weight: {weight:.3f}")
```

```
Word: , Weight: 0.004
Word: s, Weight: 0.037
Word: color, Weight: 0.105
Word: number, Weight: 0.107
Word: tell, Weight: 0.077
Word: product, Weight: 0.028
Word: small, Weight: 0.059
Word: seller, Weight: 0.088
Word: purchase, Weight: 0.056
Word: quality, Weight: 0.059
Word: appearance, Weight: 0.113
Word: long, Weight: 0.104
Word: open, Weight: 0.082
Word: foundation, Weight: 0.098
Word: stay, Weight: 0.071
Word: try, Weight: 0.048
Word: package, Weight: 0.081
Word: opening, Weight: 0.130
Word: different, Weight: 0.069
Word: new, Weight: 0.068
Word: poor, Weight: 0.105
Word: suppose, Weight: 0.091
Word: real, Weight: 0.343
Word: difference, Weight: 0.075
Word: lid, Weight: 0.314
Word: length, Weight: 0.096
Word: gold, Weight: 0.111
Word: liquid, Weight: 0.097
Word: upon, Weight: 0.113
Word: shade, Weight: 0.099
Word: pour, Weight: 0.126
Word: tube, Weight: 0.383
Word: fake, Weight: 0.438
Word: fleck, Weight: 0.172
Word: dermacol, Weight: 0.364
Word: circumferance, Weight: 0.245
```

WORD EMBEDDING FOR FEATURE ENGINEERING (word2vec)

- Count-based techniques, however, do not give regard to word sequence and sentence structure, and thus lose semantics
- The *Word2Vec* technique from Gensim lib actually embeds meaning in vectors by quantifying how often a word appears within the vicinity of a given set of other words

```
class gensim.models.word2vec.Word2Vec(sentences=None, corpus_file=None, vector_size=100,
alpha=0.025, window=5, min_count=5, max_vocab_size=None, sample=0.001, seed=1, workers=3,
min_alpha=0.0001, sg=0, hs=0, negative=5, ns_exponent=0.75, cbow_mean=1, hashfxn=<built-in
function hash>, epochs=5, null_word=0, trim_rule=None, sorted_vocab=1, batch_words=10000,
compute_loss=False, callbacks=(), comment=None, max_final_vocab=None)
```

	0	1	2	3	4	5	6	7	8	9	...	90	91	92
great	-1.201949	0.144700	0.574869	2.752909	-0.388276	-1.574940	0.477391	0.049548	-1.164826	-0.106699	...	1.994949	2.316634	1.655789
husband	0.044235	-0.624490	0.766864	-0.096014	-1.737497	-1.962552	1.309048	2.582413	-0.380935	0.499077	...	-0.424558	0.666695	0.228896
want	-1.975531	1.179926	-1.058031	0.623056	-0.554303	1.691741	-0.853700	0.850673	0.688437	-1.651572	...	-0.635158	0.537367	-0.549163
read	-0.597246	-2.358415	0.198330	-0.574358	-0.256990	0.211905	0.574853	1.306706	-0.310360	-1.215431	...	0.877756	2.454601	-0.734411
negro	0.112753	0.003688	-0.110379	0.157676	0.004753	0.150588	0.081015	0.107179	0.054396	0.117308	...	-0.184214	-0.046896	-0.078399
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
soonpure	0.043576	0.014503	-0.046892	-0.001987	-0.002582	-0.033010	0.037386	0.041540	0.074177	-0.017852	...	-0.123165	0.062622	0.004487
zed	0.012950	-0.010936	0.014227	0.038398	0.038637	0.059376	-0.009664	0.135040	0.070107	0.189067	...	-0.063432	-0.011397	-0.060631
karangahake	0.045661	-0.017402	0.007784	0.067614	0.047250	0.041291	0.152289	0.035717	-0.005302	0.141260	...	-0.014565	0.097467	-0.009068
mwh	0.047629	0.009111	-0.088758	0.167211	0.078874	0.141506	0.068036	-0.005600	0.014016	0.209020	...	-0.146891	-0.022248	0.106202
nrr	-0.024798	0.088870	-0.171527	0.025644	0.095790	0.060829	0.018176	0.204136	0.060063	0.093949	...	-0.193412	-0.084400	0.021993

18691 rows × 100 columns

FINAL DATAFRAME for MODELLING

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	
great	-1.201949	0.144700	0.574869	2.752909	-0.388276	-1.574940	0.477391	0.049548	-1.164826	-0.106699	...	1.994949	2.316634	1.655789	1.26
husband	0.044235	-0.624490	0.766864	-0.096014	-1.737497	-1.962552	1.309048	2.582413	-0.380935	0.499077	...	-0.424558	0.666695	0.228896	-0.99
want	-1.975531	1.179926	-1.058031	0.623056	-0.554303	1.691741	-0.853700	0.850673	0.688437	-1.651572	...	-0.635158	0.537367	-0.549163	-1.61
read	-0.597246	-2.358415	0.198330	-0.574358	-0.256990	0.211905	0.574853	1.306706	-0.310360	-1.215431	...	0.877756	2.454601	-0.734411	-2.04
negro	0.112753	0.003688	-0.110379	0.157676	0.004753	0.150588	0.081015	0.107179	0.054396	0.117308	...	-0.184214	-0.046896	-0.078399	0.09

MEAN

	0	1	2	3	4	5	6	7	8	9	...	91	92	93	94
0	-1.201949	0.144700	0.574869	2.752909	-0.388276	-1.574940	0.477391	0.049548	-1.164826	-0.106699	...	2.316634	1.655789	1.260824	-0.539082
1	-0.122476	-0.274320	-0.235948	0.239716	-0.083765	-0.369554	0.212815	0.528531	0.055425	-0.435444	...	0.435009	-0.108200	-0.289259	-0.343994
2	-0.134644	-1.206929	-0.519902	0.246288	0.966006	0.071862	-0.058469	0.253898	-0.057673	0.014242	...	-0.303036	0.419713	-0.115435	0.591335
3	-0.163815	-0.474026	-0.003632	0.119767	0.116645	-0.097260	-0.000558	-0.022949	-0.014875	-0.603142	...	0.135594	-0.182681	-0.557722	0.470839
4	-0.157024	-0.151582	-0.406453	-0.000831	-0.129803	-0.177306	-0.096088	0.430276	0.012211	-0.167220	...	0.243920	0.034709	-0.423191	-0.105042

Review ID

MODELLING

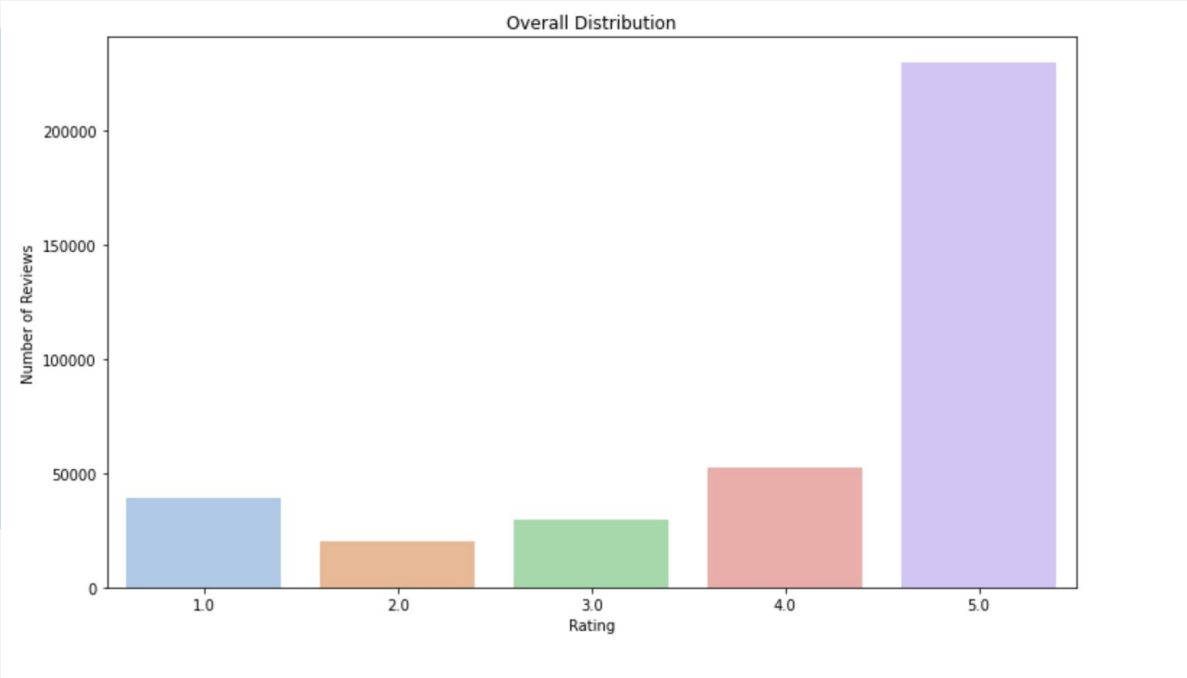
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

	Random Forest	Decision Tree	ExtraTree Classifier	XGBoost
Train Accuracy	98.8%	98.6%	99.07%	74.15%
Test Accuracy	64.04%	50.6%	64.47%	67.4%

Let's see the precision and recall values in notebook..

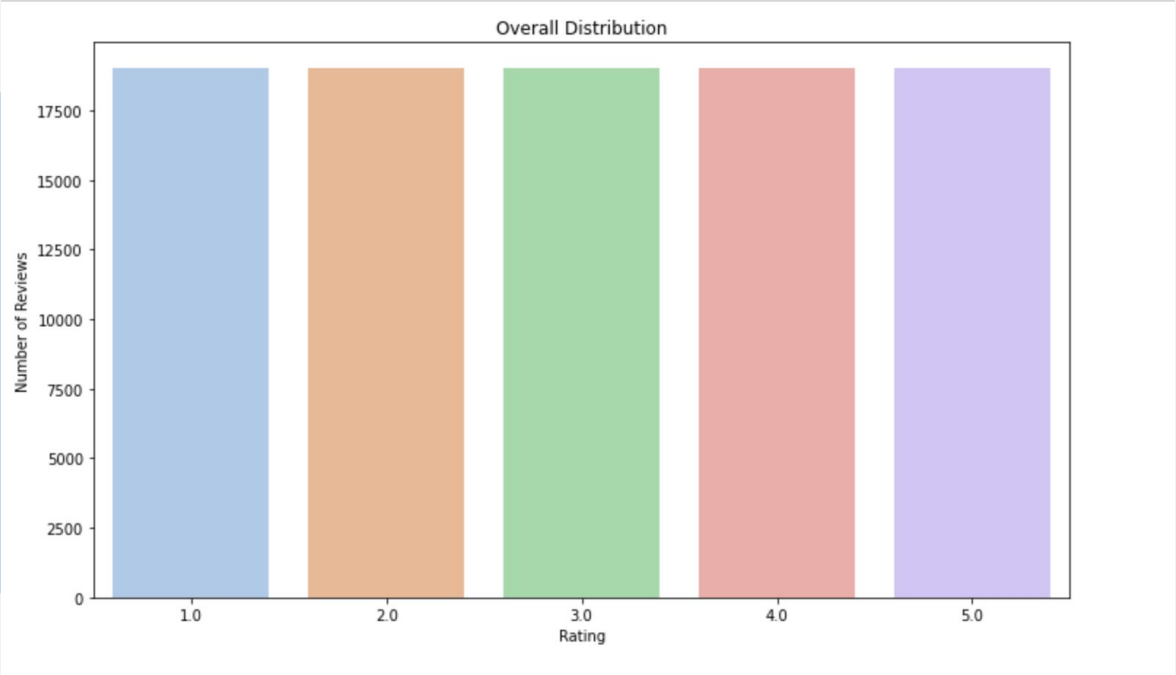
DEALING WITH UNBALANCED DATASET

The distribution of ratings shows that customers tend to give 5 stars generally. This leads to the imbalanced problem in the dataset.



DEALING WITH UNBALANCED DATASET

Undersampling method:



The number of reviews decreased from 371k to 95k after undersampling method.

MODELLING WITH BALANCED DATASET

```
xgB = XGBClassifier()
xgB.fit(X_train, y_train)
xgb_train_predict = pd.DataFrame(data=xgB.predict(X_train))
classification_metrics(y_train, xgb_train_predict)
```

Train:

```
[[12791   900   741   438   330]
 [ 1596 10821  1180  1062   541]
 [ 1008  1275 10474  1603   840]
 [  463   753  1042 11240  1702]
 [  286   380   469  1132 12933]]

Accuracy:  0.7665657894736843

Classification report:

              precision    recall  f1-score   support

         1.0       0.79      0.84      0.82     15200
         2.0       0.77      0.71      0.74     15200
         3.0       0.75      0.69      0.72     15200
         4.0       0.73      0.74      0.73     15200
         5.0       0.79      0.85      0.82     15200

    accuracy                           0.77     76000
   macro avg       0.77      0.77      0.77     76000
weighted avg       0.77      0.77      0.77     76000
```

Test:

```
xgb_test_predict = pd.DataFrame(data=xgB.predict(X_test))
classification_metrics(y_test, xgb_test_predict)
```

```
Confusion Matrix:

[[2197  885  357  174  187]
 [1070 1200  829  453  248]
 [ 518  911 1135  857  379]
 [ 198  396  666 1464 1076]
 [ 173  207  230  784 2406]]

Accuracy:  0.4422105263157895

Classification report:

              precision    recall  f1-score   support

         1.0       0.53      0.58      0.55      3800
         2.0       0.33      0.32      0.32      3800
         3.0       0.35      0.30      0.32      3800
         4.0       0.39      0.39      0.39      3800
         5.0       0.56      0.63      0.59      3800

    accuracy                           0.44     19000
   macro avg       0.43      0.44      0.44     19000
weighted avg       0.43      0.44      0.44     19000
```

MODELLING - DEEP LEARNING (LSTM)

```
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(LSTM(16))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(Dense(5, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 5
batch_size = 64

history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,validation_split=0.1,callbacks=[EarlyStop
```

```
accr = model.evaluate(X_test,Y_test)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

```
999/999 [==============================] - 9s 9ms/step - loss: 0.8258 - accuracy: 0.6963
Test set
  Loss: 0.826
  Accuracy: 0.696
```

```
Confusion Matrix:

[[ 2520    50   276   221   644]
 [  162   941   467    42  2947]
 [  412   378   764   169   919]
 [  737    94   386   209   496]
 [  294   685   293    35 17824]]

Accuracy:  0.6963241044892852

Classification report:

              precision    recall  f1-score   support

           0       0.61      0.68      0.64      3711
           1       0.44      0.21      0.28      4559
           2       0.35      0.29      0.32      2642
           3       0.31      0.11      0.16      1922
           4       0.78      0.93      0.85     19131

    accuracy                           0.70     31965
   macro avg       0.50      0.44      0.45     31965
weighted avg       0.65      0.70      0.66     31965
```

Let's compare the precision and recall values of XGBoost classifier..

MODELLING - DEEP LEARNING (LSTM + CNN)

```
from tensorflow.keras.layers import Conv1D, MaxPooling1D
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(50000, 16, input_length=250))
model.add(Conv1D(filters=64, kernel_size = 3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(16))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(Dense(5, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
Confusion Matrix:

[[ 2469   102   361   273   506]
 [  169  1243   615    75  2457]
 [  409   517   844   185   687]
 [  739   152   439   219   373]
 [  386  1553   537   102 16553]]

Accuracy:  0.6672297825746911

Classification report:

              precision    recall  f1-score   support

           0       0.59      0.67      0.63      3711
           1       0.35      0.27      0.31      4559
           2       0.30      0.32      0.31      2642
           3       0.26      0.11      0.16      1922
           4       0.80      0.87      0.83     19131

    accuracy                           0.67     31965
   macro avg       0.46      0.45      0.45     31965
weighted avg       0.64      0.67      0.65     31965
```

Although the overall accuracy compared to LSTM was lower, the probability of predicting other classes increased.

EXPLORATORY DATA ANALYSIS (EDA)

Topic Modelling with LDA:

```
Topic 1:
 hair 0.05713774263858795
 use 0.036590639501810074
 product 0.022688036784529686
 work 0.01836898922920227
 t 0.016205700114369392

Topic 2:
 t 0.049344129860401154
 n 0.04901977628469467
 product 0.03307762369513512
 get 0.014365723356604576
 buy 0.013248815201222897

Topic 3:
 hair 0.04396681860089302
 t 0.022642631083726883
 n 0.022480880841612816
 get 0.015244336798787117
 use 0.015058043412864208

Topic 4:
 brush 0.025476820766925812
 razor 0.024815315380692482
 shave 0.024490581825375557
 use 0.014448480680584908
 shaver 0.013407531194388866

Topic 5:
 use 0.02587071806192398
 n 0.017022639513015747
 t 0.017021844163537025
 one 0.012683774344623089
 teeth 0.011737131513655186

Topic 5:
 use 0.02587071806192398
 n 0.017022639513015747
 t 0.017021844163537025
 one 0.012683774344623089
 teeth 0.011737131513655186

Topic 6:
 look 0.015447891317307949
 make 0.014532185159623623
 get 0.013522359542548656
 use 0.012344281189143658
 like 0.011958295479416847

Topic 7:
 love 0.05064253881573677
 use 0.02666466124355793
 great 0.022818630561232567
 one 0.017748015001416206
 year 0.017409350723028183

Topic 8:
 use 0.020839886739850044
 product 0.019531290978193283
 smell 0.01601346582174301
 t 0.014349640347063541
 n 0.014276927337050438

Topic 9:
 skin 0.037778306752443314
 use 0.031199753284454346
 product 0.023964889347553253
 face 0.012688414193689823
 feel 0.011353571899235249

Topic 10:
 great 0.02860287018120289
 color 0.027367671951651573
 good 0.026333030313253403
 love 0.023227887228131294
 nice 0.01951318047940731
```

Word2vec similarities:

```
word_vec.wv.most_similar('husband', topn=5)
```

```
[('wife', 0.8936811685562134),
 ('boyfriend', 0.8117631077766418),
 ('son', 0.8020001649856567),
 ('girlfriend', 0.7422683238983154),
 ('hubby', 0.7285659313201904)]
```

```
word_vec.wv.similarity('husband', 'girlfriend')
```

```
0.7422683
```

```
word_vec.wv.similarity('boy', 'girl')
```

```
0.60577476
```

```
word_vec.wv.most_similar('shave', topn=5)
```

```
[('shaving', 0.8168691992759705),
 ('shaves', 0.735748291015625),
 ('razor', 0.6707531213760376),
 ('shaver', 0.5992351770401001),
 ('blade', 0.5751073360443115)]
```

EXPLORATORY DATA ANALYSIS (EDA)

Word2vec similarities:

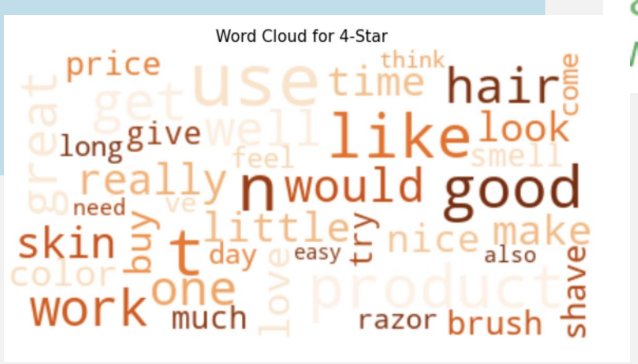
```
word_bank = ["blade", "skin", "hair", "shampoo", "eye", "parfume", "lip", "fruit", "pencil", "wave", "teeth"]

for word in word_bank[:]:
    related_vec = word_vec.wv.most_similar(word, topn=5)
    related_words = np.array(related_vec)[:,0]
    word_bank.extend(related_words)
    print(f"{word}: {related_words}")
```

```
blade: ['blades' 'razor' 'feathers' 'cartridge' 'mach']
skin: ['face' 'complexion' 'flaky' 'redness' 'rosacea']
hair: ['wavy' 'frizzy' 'curly' 'curl' 'strand']
shampoo: ['conditioner' 'detangler' 'poo' 'pantene' 'dandruff']
eye: ['eyes' 'eyelid' 'undereye' 'crease' 'wrinkle']
parfume: ['perry' 'prada' 'fragrence' 'flavors' 'amor']
lip: ['lips' 'chapstick' 'lipstick' 'chap' 'lipgloss']
fruit: ['cinnamon' 'rosemary' 'grape' 'chamomile' 'melon']
pencil: ['liner' 'sharpener' 'eyeliner' 'eyeshadow' 'palette']
wave: ['curl' 'curls' 'bouncy' 'wavy' 'frizz']
teeth: ['tooth' 'gum' 'molar' 'tray' 'mouth']
```

EXPLORATORY DATA ANALYSIS (EDA)

Word Clouds:



THANKS

Do you have any questions?