Notebook_1_Data_Preparation

June 26, 2022

1 Classifying Unlocked Phone Reviews on Amazon

University of San Diego, 2022 *Nava Roohi, Cole Bailey, Filipp Krasovsky*

1.1 Table of Contents

1.1.1 1. Introduction

The high level goal of this project is to:

Ingest data using an API or Web Scraping tool

Clean, tokenize, and otherwise normalize text data

Linguistically process elements such as POS, NER, etc.

Address class imbalances

Create Feature vectors

Train a model that can categorize reviews by rating

Test the model against unseen data

Dataset Background

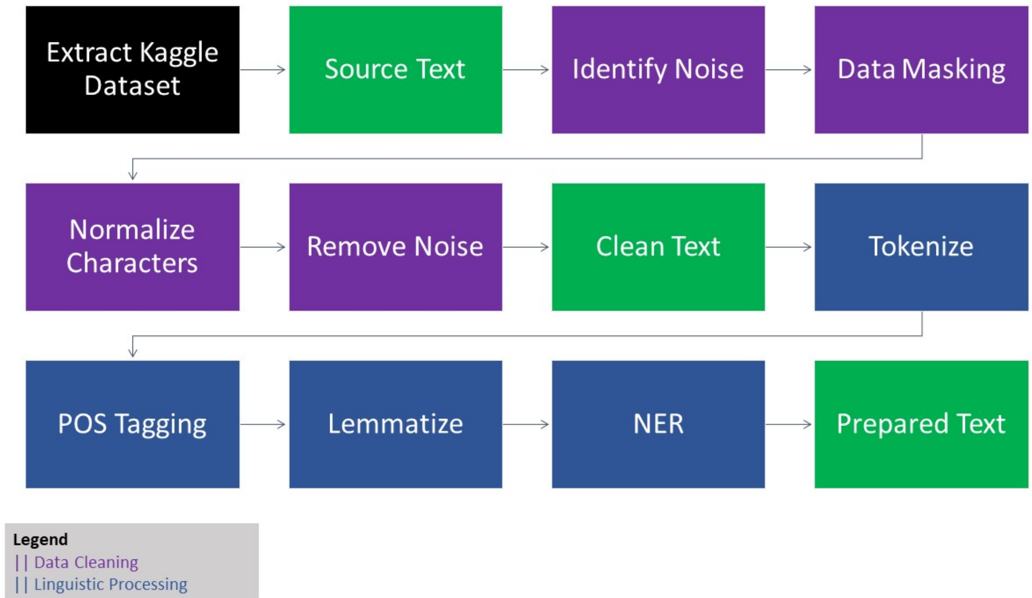
The data used for this exercise is the Amazon Reviews dataset for unlocked mobile phones, which can be downloaded here.

1.1.2 2. Phases

Phase 1: Data Ingestion and Cleaning

```
[13]: from IPython.display import Image
      Image(filename='data_ingestion_pipeline.jpg')
```

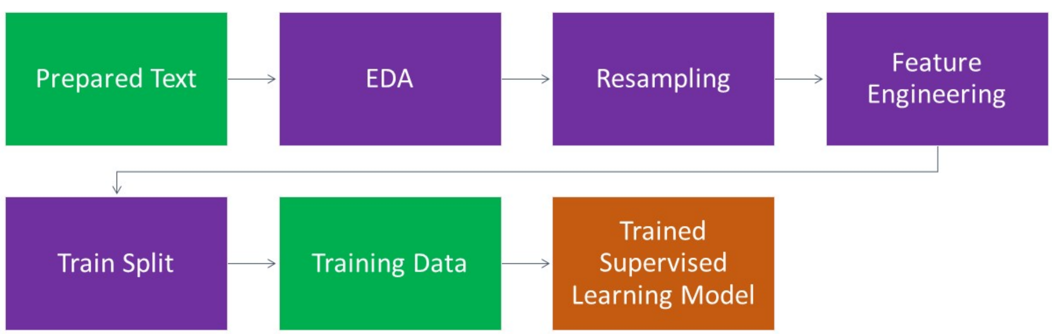
[13]:



Phase 2: Feature Engineering and Training

```
[15]: Image(filename='phase2.jpg')
```

[15]:



1.1.3 3. Data Ingestion

We begin by using the kaggle api to download our amazon reviews dataset to our current directory. We can accomplish this by leaving the download path argument blank when we send our API request.

[2]:
```
import kaggle
import pandas as pd
import matplotlib.pyplot as plt
import pyarrow
import fastparquet
import numpy as np
import os
from collections import Counter, defaultdict
import warnings
import seaborn as sns
warnings.filterwarnings("ignore")
from wordcloud import WordCloud
kaggle.api.authenticate()
import nltk
from string import punctuation
import textacy.preprocessing as tprep
from nltk.corpus import stopwords
from wordcloud import WordCloud, STOPWORDS
import spacy
nlp = spacy.load("en_core_web_sm")
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
```

[3]:
```
kaggle.api.dataset_download_files(
    'PromptCloudHQ/amazon-reviews-unlocked-mobile-phones',
    unzip=True
)
```

[4]:
```
df = pd.read_csv('Amazon_Unlocked_Mobile.csv')
df = df.rename(columns={"Reviews": "text"})
df.head()
```

[4]:
```
                                        Product Name Brand Name   Price  \
0  "CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7…    Samsung  199.99
1  "CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7…    Samsung  199.99
2  "CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7…    Samsung  199.99
3  "CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7…    Samsung  199.99
4  "CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7…    Samsung  199.99

   Rating                                               text  Review Votes
```

```
0        5  I feel so LUCKY to have found this used (phone…           1.0
1        4  nice phone, nice up grade from my pantach revu…           0.0
2        5                                        Very pleased           0.0
3        4  It works good but it goes slow sometimes but i…           0.0
4        4  Great phone to replace my lost phone. The only…           0.0
```

Next, we can do a cursory overview of several data points:

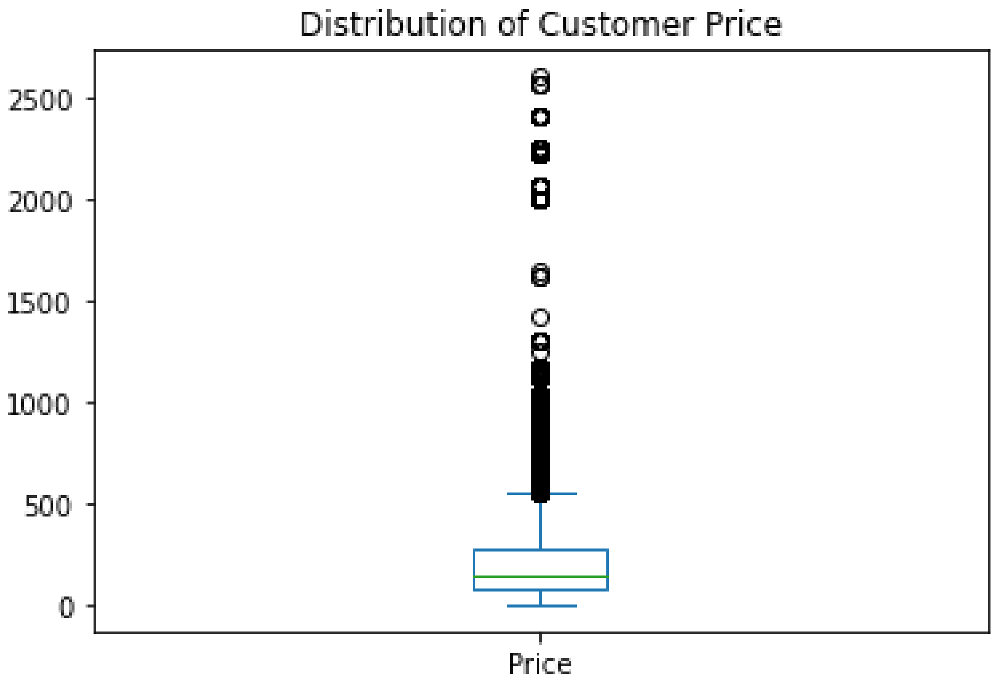
What our distribution of ratings is

What our distribution of prices is

What our distribution of review length is

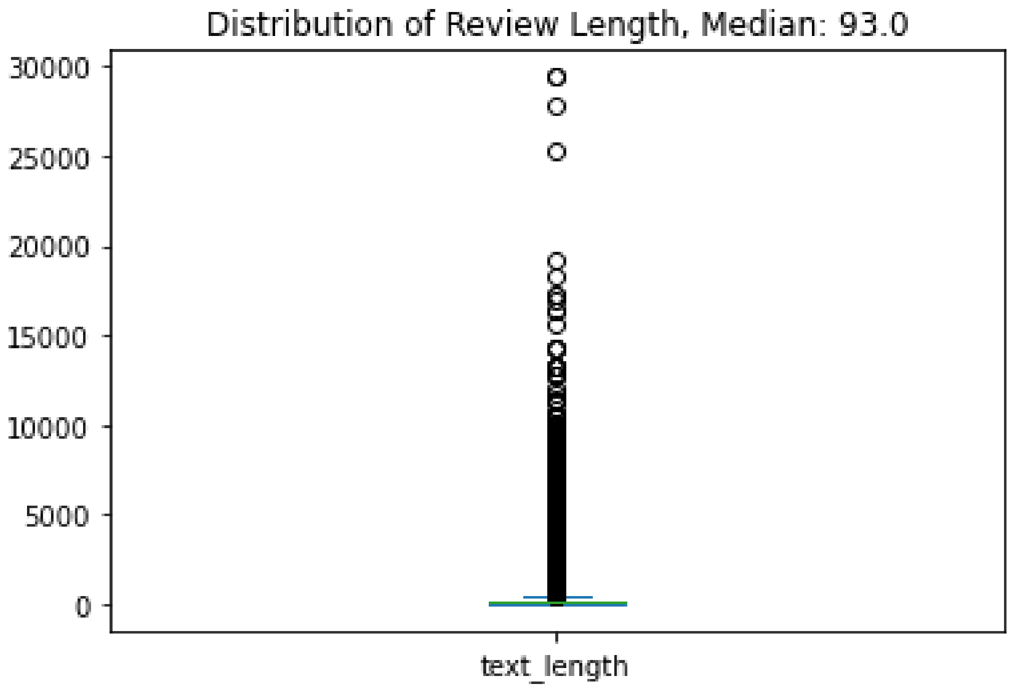
3a. Rating Distribution and Price Distribution

[5]:
```python
df['Rating'].value_counts().sort_index().plot(kind='bar',title="Distribution of␣
↪Customer Reviews")
plt.show()
df['Price'].plot(kind='box',title="Distribution of Customer Price")
plt.show()
```



3b. Review Length Distribution Our findings show that while the majority of reviews are at around 100 characters, some reviews are fairly extensive. We may consider setting an upper bound on the number of characters we're willing to accept to avoid higher dimensionality.

[6]:
```
#create the length feature
df['text'] = df['text'].astype(str)
df['text_length'] = df['text'].map(len)
#get the median character length
median_len = str(np.median(df['text_length']))
sdev = np.std(df['text_length'])
df['text_length'].plot(kind='box',title=f'Distribution of Review Length, Median:
 ↪ {median_len}')
plt.show()
```

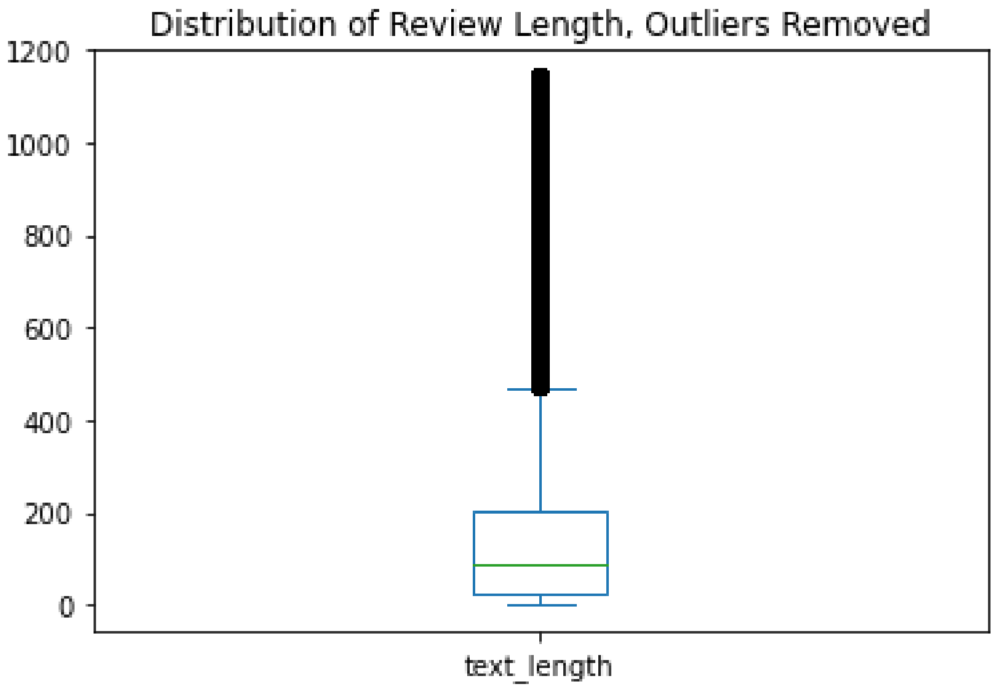


[7]:
```python
#how many outliers do we have?
mean_len= np.mean(df['text_length'])
within_two_sd = df[abs(df['text_length'] - mean_len) <= 2*sdev]
print(len(within_two_sd)/len(df))
```

```
0.9712666731103808
```

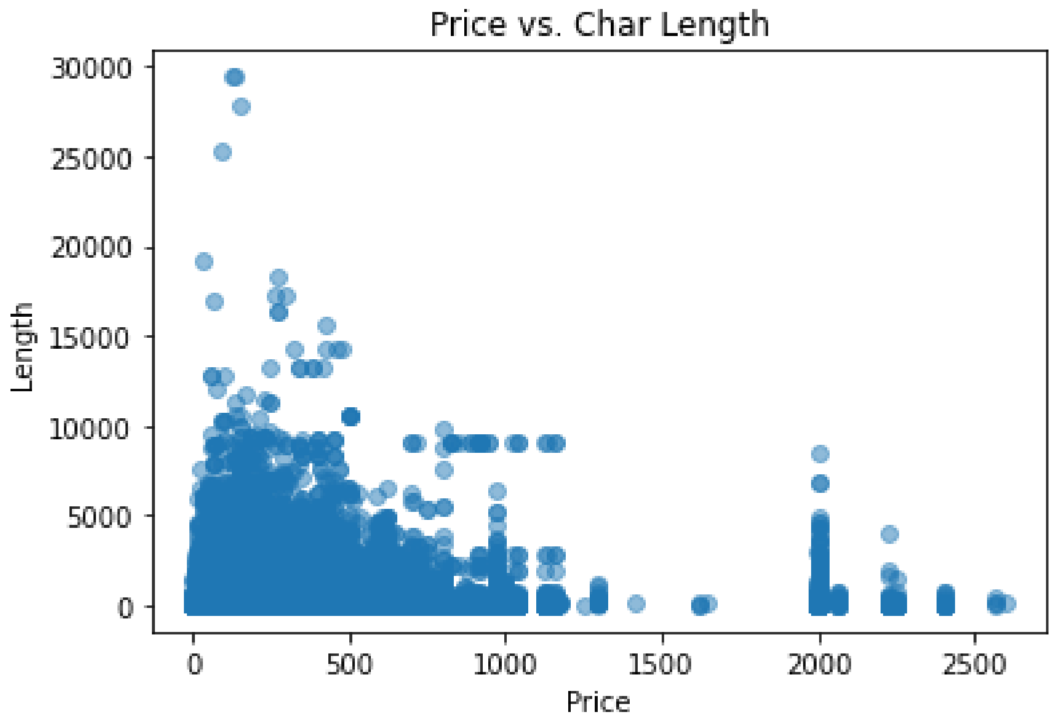
about 3% of our dataset has a text length two standard deviations or more from the mean. Let's examine what our boxplot looks like if we were to remove these. We may also want to consider the fact that we have not cleaned our data yet, so much of this EDA is to be taken with a grain of salt.

[8]:
```python
within_two_sd['text_length'].plot(kind='box',title=f'Distribution of Review␣
↪Length, Outliers Removed')
plt.show()
del within_two_sd #clear kernel space
```



3c. Do Price and Review Length share a connection?

[9]:
```
plt.scatter(df['Price'], df['text_length'], alpha=0.5)
plt.title("Price vs. Char Length")
plt.xlabel("Price")
plt.ylabel("Length")
plt.show()
```



4. Noise Recognition

4a. Text Impurity To begin, we can use a function that identifies suspicious characters and returns an impurity score from 0 to 1 for each review.

[10]:
```
import re
RE_SUSPICIOUS = re.compile(r'[&#<>{}\[\]\\]')

#create a function that keeps track of how impure our datasets are to see how␣
↪much cleaning is needed and
#if our cleaning had any meaningful effect.

def impurity(text, min_len=10):
    """returns the share of suspicious characters in a text"""
    if text == None or len(text) < min_len:
        return 0
    else:
        return len(RE_SUSPICIOUS.findall(text))/len(text)

def impurity_list(text,min_len=10):
    """ returns the list of suspicious characters in a text.
        iterate over the list of tokens and return every word with suspicious␣
↪characters.
```

```
    """
    if text == None or len(text) < min_len:
        return []
    else:
        impure_words = []
        tokens = text.split()
        for t in tokens:
            if len(RE_SUSPICIOUS.findall(t))>0:
                impure_words.append(t)
        return impure_words
```

[11]:
```
#map the function to a new column
df['impurity'] = df['text'].map(impurity)
df['impure_words'] = df['text'].map(impurity_list)
df.head(3)
```

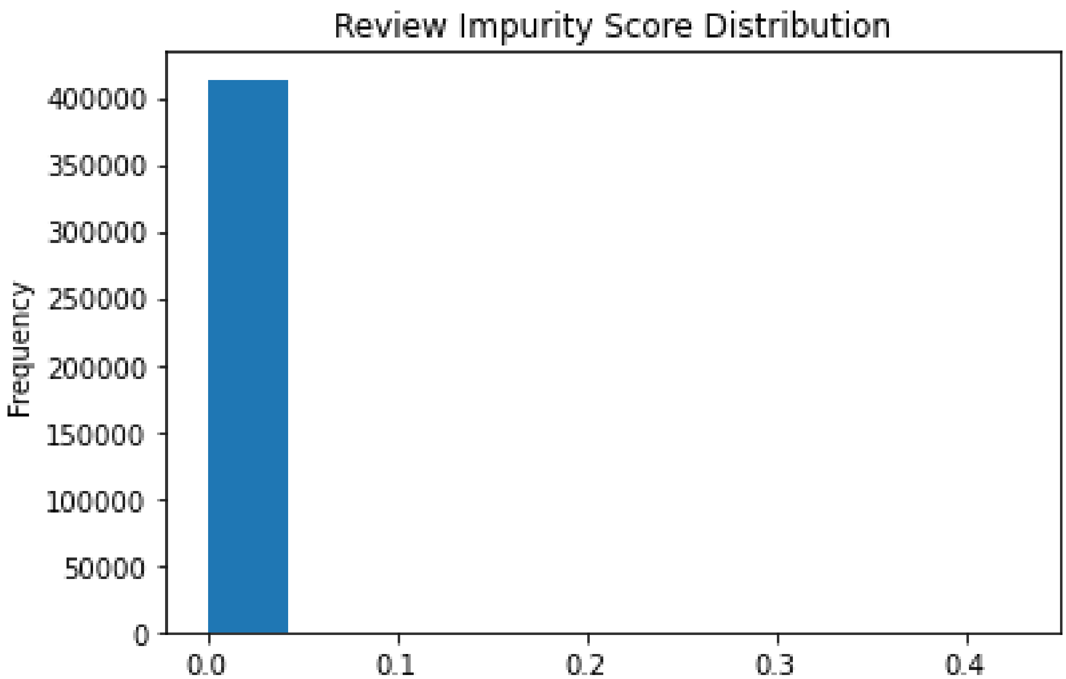
[11]:
```
                                        Product Name Brand Name   Price  \
0  "CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7…    Samsung  199.99
1  "CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7…    Samsung  199.99
2  "CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7…    Samsung  199.99

   Rating                                              text  Review Votes  \
0       5  I feel so LUCKY to have found this used (phone…           1.0
1       4  nice phone, nice up grade from my pantach revu…           0.0
2       5                                      Very pleased           0.0

   text_length  impurity impure_words
0          374  0.008021    [&, &, &]
1          214  0.000000           []
2           12  0.000000           []
```

we can now take steps to determine our worst-case scenario for text purity by getting the top 5 most impure values and generating a histogram. **Findings show that the majority of reviews fall under 5% impurity, with a handful of outliers in a negligible quantity.**

[12]:
```
print(df[['impurity','impure_words']].
 ↪sort_values(by='impurity',ascending=False).head(10))
df['impurity'].plot(kind='hist',bins=10,title='Review Impurity Score␣
 ↪Distribution')
plt.show()
```

```
        impurity                                      impure_words
113456  0.428571                          [Good####, Service#####]
114317  0.428571                          [Good####, Service#####]
261274  0.363918  [helps.>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>…
291904  0.281250                           [MORE>>>>>>>>>SOON!!!!!!]
291934  0.281250                           [MORE>>>>>>>>>SOON!!!!!!]
348810  0.187500                                    [<3, <3, <3]
```

```
142761   0.161290                          [$%@%#'#%'#'#:#:?!!!!!!]
141961   0.161290                          [$%@%#'#%'#'#:#:?!!!!!!]
67467    0.115385                                             [>>>]
120616   0.100000                                       [#it's_okay]
```



1.1.4 5. Data Masking

In this section, we're interested in identifying emails, URLs, and possibly phone numbers. We can use regex to:

validate this information exists in the dataset in non-trivial amounts.

replace or otherwise remove it.

[13]:
```
#first, we create regex that can detect emails and urls.
url = re.compile("(https?:\/\/(?:www\.|(?!
↪www))[a-zA-Z0-9][a-zA-Z0-9-]+[a-zA-Z0-9]\.[^\s]{2,}|www\.
↪[a-zA-Z0-9][a-zA-Z0-9-]+[a-zA-Z0-9]\.[^\s]{2,}|https?:\/\/(?:www\.|(?!
↪www))[a-zA-Z0-9]+\.[^\s]{2,}|www\.[a-zA-Z0-9]+\.[^\s]{2,})", re.IGNORECASE)
email = re.compile('\S+@\S+\.\S+')
phone_numbers=re.compile(r'(\d{3}[-\.\s]??\d{3}[-\.\s]??
↪\d{4}|\(\d{3}\)\s*\d{3}[-\.\s]??\d{4}|\d{3}[-\.\s]??\d{4})')

mask_pipeline = {'urls':url,'emails':email,'phones':phone_numbers}
```

[14]:
```
#iterate over our regex pipeline and store the output in df
for reg in mask_pipeline.keys():
    df[reg] = df['text'].map(mask_pipeline[reg].findall)
```

[15]:
```
df.sort_values(by='urls',ascending=False).head(3)
```

[15]:
```
                                             Product Name          Brand Name  \
394686  SONY XPERIA Z3 COMPACT D5803 16GB (FACTORY UNL…                  Sony
402109  Sudroid Z18 Android 4.2 Mini Water and Dust-pr…  Fosler Corporation
402068  Sudroid Z18 2.45 Inches Unlocked Mini Phone wi…                   NaN

         Price  Rating                                               text  \
394686  799.00       5  I just spent 2 hours with Gloria Estefan in th…
402109   72.99       4  Discovery Z18 - says "Vogue Phone" on backEsse…
402068   69.99       4  Discovery Z18 - says "Vogue Phone" on backEsse…

        Review Votes  text_length  impurity impure_words  \
394686           NaN          118       0.0           []
402109          29.0         1597       0.0           []
402068          29.0         1597       0.0           []


                                                  urls emails phones
394686             [www.youtube.com/watch?v=g8v6cZ21vlc]     []     []
402109  [www.youtube.com/watch?v=XWLBKSAwoiM(Root, htt…     []     []
402068  [www.youtube.com/watch?v=XWLBKSAwoiM(Root, htt…     []     []
```

After we find each kind of information, we want to get a sense of how prevalent it is in the data as well as validate that we actually identified URLs, emails, etc. with our regex mapping. One way to do this is by creating a counter and seeing if the output matches the structure of an email, phone number, or URL.

[16]:
```
#get a broad sense of the presence of each type of information to be masked.
'a function that accepts a data series of tokens and combiens them into one␣
↪list.'
def combine_tokens(tokens):
    out = []
    for token_list in tokens:
        out = out + token_list
    return out

def get_frequency(tokensObj,count_name = 'count'):
    #convert into a list.
    tokens_count = Counter(combine_tokens(tokensObj))
    count_df = pd.DataFrame.from_dict(tokens_count, orient='index').
↪reset_index()
    return(count_df.rename(columns={'index':'token',0:count_name}))
```

[17]:
```
for reg in mask_pipeline.keys():
    this_list = get_frequency(df[reg])
    n = this_list['count'].count()
    s = this_list['count'].sum()
    print(f'there are {n} unique values in {reg} with a total of {s}␣
 ↪occurences')
    print(this_list.head())
    print('--------------\n')
```

```
there are 302 unique values in urls with a total of 675 occurences
                                               token  count
0  https://www.amazon.com/gp/aw/d/B01GYUDMFY/ref=…      2
1  http://www.amazon.com/gp/product/B00EY7SS72/re…      3
2  http://www.amazon.com/gp/product/B00PEJQU9M?re…      2
3  http://www.amazon.com/gp/product/B00ZOER95Q?ps…      1
4                        https://youtu.be/JU-dDjKi4Ig      1
--------------

there are 105 unique values in emails with a total of 227 occurences
                              token  count
0                      Quad-core@1.3      3
1  PLEASE!!!!!carolderenzo@yahoo.com      2
2                  hunalfi@gmail.com      2
3        mohamedjawahiri@hotmail.com      1
4         CarlosGolosina@hotmail.com      1
--------------

there are 249 unique values in phones with a total of 550 occurences
          token  count
0  843-709-3118      1
1    1717308766      1
2  855-368-0829      1
3    2407749011      2
4    2153843082      1
--------------
```

Conclusion: The majority of our data does not contain information that needs to be masked, but we will still replace these instances with blank spaces. initial obvservations suggest that our regex expressions correctly identified the information we were looking for.

[18]:
```
#iterate over our regex pipeline and change our text to not contain masked data.
for k,v in mask_pipeline.items():
    df['text'] = df['text'].apply(lambda x: v.sub("",str(x)))
```

[19]:
```
#iterate over our regex pipeline and recalculate the output in df
for reg in mask_pipeline.keys():
```

```
    df[reg] = df['text'].map(mask_pipeline[reg].findall)

#sanity check one more time after re-running
for reg in mask_pipeline.keys():
    this_list = get_frequency(df[reg])
    print(this_list)
    print('--------------\n')
```

```
Empty DataFrame
Columns: [token]
Index: []
--------------

Empty DataFrame
Columns: [token]
Index: []
--------------

Empty DataFrame
Columns: [token]
Index: []
--------------
```

1.1.5 6. Data Normalization

[20]:
```
#lower case
df['new_reviews'] = df['text'].apply(lambda x: " ".join(x.lower() for x in x.
 ↪split()))
#remove punctuation
df['new_reviews'] = df['new_reviews'].str.replace('[^\w\s]','')
# removing stopwords
stop = stopwords.words('english')
df['new_reviews'] = df['new_reviews'].apply(lambda x: " ".join(x for x in x.
 ↪split() if x not in stop))

df['new_reviews']
```

[20]:
```
0         feel lucky found used phone us used hard phone…
1         nice phone nice grade pantach revue clean set …
2                                                 pleased
3            works good goes slow sometimes good phone love
4         great phone replace lost phone thing volume bu…
                                   …
413835                        another great deal great price
413836                                                      ok
413837            passes every drop test onto porcelain tile
```

```
413838     returned meet needs seemed good selection others
413839    downside apparently verizon longer uses vcast …
Name: new_reviews, Length: 413840, dtype: object
```

1.1.6 7. Tokenization

we can tokenize our data using the spacy library. This initialization also allows us to create a pipeline for POS tagging, lemmatization, and NER as well as tokenization.

[21]:
```python
import spacy
import textacy
nlp = spacy.load('en_core_web_sm')
nlp.pipeline
```

[21]:
```
[('tok2vec', <spacy.pipeline.tok2vec.Tok2Vec at 0x7fd774e309a0>),
 ('tagger', <spacy.pipeline.tagger.Tagger at 0x7fd774a9cb20>),
 ('parser', <spacy.pipeline.dep_parser.DependencyParser at 0x7fd774e356d0>),
 ('attribute_ruler',
  <spacy.pipeline.attributeruler.AttributeRuler at 0x7fd7e59add40>),
 ('lemmatizer',
  <spacy.lang.en.lemmatizer.EnglishLemmatizer at 0x7fd76037eac0>),
 ('ner', <spacy.pipeline.ner.EntityRecognizer at 0x7fd774e35890>)]
```

[22]:
```python
#apply this pipeline to our df to generate tokens:
def extract_nlp(doc):
    return {
        'lemmas' : extract_lemmas(doc,
        exclude_pos = ['PART', 'PUNCT',
        'DET', 'PRON', 'SYM', 'SPACE'],
        filter_stops = False),
        'adjs_verbs' : extract_lemmas(doc, include_pos = ['ADJ', 'VERB']),
        'nouns' : extract_lemmas(doc, include_pos = ['NOUN', 'PROPN']),
        'noun_phrases' : extract_noun_phrases(doc, ['NOUN']),
        'adj_noun_phrases': extract_noun_phrases(doc, ['ADJ']),
        'entities' : extract_entities(doc, ['PERSON', 'ORG', 'GPE', 'LOC'])
    }

def extract_lemmas(doc, **kwargs):
    return [t.lemma_ for t in textacy.extract.words(doc, **kwargs)]

def extract_noun_phrases(doc, preceding_pos=['NOUN'], sep='_'):
    patterns = []
    for pos in preceding_pos:
        patterns.append(f"POS:{pos} POS:NOUN:+")
    spans = textacy.extract.matches.token_matches(doc, patterns=patterns)
    return [sep.join([t.lemma_ for t in s]) for s in spans]
```

```python
def extract_entities(doc, include_types=None, sep='_'):
    ents = textacy.extract.entities(doc,
    include_types=include_types,
    exclude_types=None,
    drop_determiners=True,
    min_freq=1)
    return [sep.join([t.lemma_ for t in e])+'/'+e.label_ for e in ents]
```

[23]:
```python
#initialize empty columns for each linguistic component we will populate to␣
↪avoid errors.
docs = nlp.pipe([''])
for j,doc in enumerate(docs):
    for col,values in extract_nlp(doc).items():
        df[col] = None
```

[24]:
```python
#create a df column for each of our linguistic pipeline steps!

import time
import tqdm.notebook as tq
start = time.localtime()
batch_size = 50

for i in tq.tqdm(range(0, len(df), batch_size),position=0,leave=True):
    docs = nlp.pipe(df['new_reviews'][i:i+batch_size])
    for j, doc in enumerate(docs):
        for col, values in extract_nlp(doc).items():
            df[col].iloc[i+j] = values

end = time.localtime()
print(start,end)
```

[25]:
```python
#remove irrelevant columns to produce a final version for working with models.
df = df[[
   ␣
↪'Rating','new_reviews','lemmas','adjs_verbs','nouns','noun_phrases','adj_noun_phrases','ent
]]
df['tokens'] = df['new_reviews'].map(str.split)
df.head()
```

[25]:
```
   Rating                                       new_reviews lemmas  \
0       5  feel lucky found used phone us used hard phone…   None
1       4  nice phone nice grade pantach revue clean set …   None
2       5                                          pleased   None
3       4     works good goes slow sometimes good phone love   None
4       4  great phone replace lost phone thing volume bu…   None

  adjs_verbs nouns noun_phrases adj_noun_phrases entities  \
```

```
0       None  None          None              None     None
1       None  None          None              None     None
2       None  None          None              None     None
3       None  None          None              None     None
4       None  None          None              None     None

                                             tokens
0  [feel, lucky, found, used, phone, us, used, ha…
1  [nice, phone, nice, grade, pantach, revue, cle…
2                                         [pleased]
3  [works, good, goes, slow, sometimes, good, pho…
4  [great, phone, replace, lost, phone, thing, vo…
```

Finally, with all the needed components for our prepared text, we save to parquet as a checkpoint.

[26]:
```
df.to_parquet('prepared_text.parquet.gzip',
              compression='gzip',
              index=False)
```

Notebook_1B_Lang_Detection

June 26, 2022

1 Ad-Hoc Pipeline Step: Remove Spanish Reviews

This section produces an analysis of how many reviews in our corpus consist of words that are, to any capacity, written in a foreign language. This step was motivated by the fact that several unsupervised learning models kept producing topics where most of the driving words were in Spanish. This notebook removes all spanish reviews and replaces all remaining foreign-language artifacts such as the word excelente in reviews that got identified non-english.

Note: This section step does not remove any reviews unless they have been identifies as Spanish by spacy.

1.1 Table of Contents

1.1.1 Candidates for Removal

Country

ISO

Spanish

es

Russian

ru

1.1.2 Candidates for Replacement

Word

Replacement

Excelente

Excellent

Producto

Product

Recomendado

Recommend

1.1.3 Packages

[189]:
```
import pandas as pd
import textacy
from sklearn.feature_extraction.text import TfidfVectorizer
from spacy.lang.en.stop_words import STOP_WORDS as stopwords
import matplotlib.pyplot as plt
from imblearn.over_sampling import RandomOverSampler
from sklearn.decomposition import NMF
from collections import Counter, defaultdict
import warnings #turn off warnings
warnings.filterwarnings("ignore", category=UserWarning)
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from gensim.models import LdaModel
from gensim.corpora import Dictionary
import time
import spacy
from spacy.language import Language
from spacy_langdetect import LanguageDetector
import re
```

1.1.4 Parquet Ingestion

[2]:
```
df = pd.read_parquet('prepared_text.parquet.gzip')
df = df.rename({"new_reviews":"text"},axis=1)
df.sample(3)
```

[2]:
```
        Rating                                               text  \
400667       3  screen protector side supposed phone screen cu…
298095       1  terrible phones even worst carrier provider ma…
407983       5  quality phone great see used refurbishedits li…
```

```
                                             lemmas  \
400667  [screen, protector, side, suppose, phone, scre…
298095  [terrible, phone, even, bad, carrier, provider…
407983  [quality, phone, great, see, use, refurbishedi…

                                 adjs_verbs  \
400667  [suppose, strangeneed, right, correct]
298095        [terrible, bad, well, recommend]
407983                    [great, think, good]

                                              nouns  \
400667  [screen, protector, phone, screen, cut, incorr…
298095                [phone, carrier, provider, phone]
407983    [quality, phone, refurbishedit, newi, buythank]

                                       noun_phrases  \
400667  [screen_protector, screen_protector_side, prot…
298095                               [carrier_provider]
407983                                  [quality_phone]

                                   adj_noun_phrases entities  \
400667  [right_screen, right_screen_protector, front_c…       []
298095  [terrible_phone, bad_carrier, bad_carrier_prov…       []
407983                                  [good_buythank]       []

                                             tokens
400667  [screen, protector, side, supposed, phone, scr…
298095  [terrible, phones, even, worst, carrier, provi…
407983  [quality, phone, great, see, used, refurbished…
```

1.1.5 Language Detection Functionality and Execution

Because Spacy doesn't treat the `LanguageDetector()` function as a native pipeline step, we have to wrap it in a function. We also need a function that takes in a text argument and returns the detected language.

[7]:
```
def get_lang_detector(nlp, name):
    return LanguageDetector()

def get_text_lang(text):
    doc = nlp(text)
    return(doc._.language['language'])
```

[ ]:
```
nlp = spacy.load("en_core_web_sm")
Language.factory("language_detector", func=get_lang_detector)
nlp.add_pipe('language_detector', last=True)
```

[ ]:
```python
df['language'] = df['text'].map(get_text_lang)
```

1.1.6 Analysis and Remarks

Distribution of Identified Languages Oddly enough, spacy has identified spanish reviews as being only at about 8k, which languages like so, ca, and french have far more members:

[119]:
```python
langs = df.query("language!='en'")['language'].value_counts().
 ↪sort_values(ascending=False)
print(langs)
```

```
so         12541
ca         11002
fr         10364
pt          8723
af          8721
ro          8395
es          8169
sl          4838
cy          3363
sk          3156
it          3139
da          2411
nl          2307
no          2119
pl          1915
et          1580
hr          1137
UNKNOWN     1110
tl           891
sv           801
fi           585
lv           539
sw           493
tr           395
de           367
id           324
cs           298
hu           165
lt           141
sq           129
vi           112
ru             2
Name: language, dtype: int64
```

Observing Spanish Reviews This lines up with expectations - most reviews are clearly in spanish, even if some of them may really be english or mixed-languagd(spanglish):

[59]:
```
df[df['language']=='es'].sample(10)
```

[59]:
```
        Rating                                               text  \
317616       5                                excelente gracias
235210       5                                buenisimo gracias
94088        2  el equipo salio defectuoso el trackpad funcion…
295040       1                                  came broken lol
262982       4  buen telefono bastantes aplicaciones utiles pu…
89247        5                                            bueno
335663       5                                        muy bueno
91010        2  yo soy de venezuela el teléfono llegó casi los…
85620        5  equipo nuevo la batería le dura todo el día fu…
91442        5  excelente equipo lo recomiendo quienes desean …

                                                lemmas  \
317616                            [excelente, gracias]
235210                            [buenisimo, gracias]
94088   [el, equipo, salio, defectuoso, el, trackpad, …
295040                              [come, break, lol]
262982  [buen, telefono, bastante, aplicacione, utile,…
89247                                          [bueno]
335663                                    [muy, bueno]
91010   [yo, soy, de, venezuela, el, teléfono, llegó, …
85620   [equipo, nuevo, la, batería, le, dura, todo, e…
91442   [excelente, equipo, lo, recomiendo, quienes, d…

                  adjs_verbs  \
317616                    []
235210                    []
94088               [siento]
295040         [come, break]
262982  [utile, rede, opcione]
89247                [bueno]
335663                    []
91010                   [do]
85620       [equipo, tiempo]
91442                     []

                                                 nouns  \
317616                            [excelente, gracias]
235210                            [buenisimo, gracias]
94088   [el, equipo, salio, defectuoso, el, trackpad, …
295040                                           [lol]
262982  [buen, telefono, bastante, aplicacione, puede,…
89247                                               []
335663                                    [muy, bueno]
91010   [yo, soy, de, venezuela, el, teléfono, llegó, …
```

```
85620   [nuevo, la, batería, le, dura, todo, el, día, …
91442   [excelente, equipo, lo, recomiendo, quienes, d…

                                             noun_phrases adj_noun_phrases  \
317616                                                 []               []
235210                                                 []               []
94088                                                  []               []
295040                                                 []               []
262982  [bastante_aplicacione, puede_tener, twitter_in…               []
89247                                                  []               []
335663                                                 []               []
91010                                      [tmovile_por]               []
85620                                                  []               []
91442         [por_mucho, por_mucho_tiempo, mucho_tiempo]               []

                                                 entities  \
317616                                                 []
235210                                                 []
94088                    [el_equipo_salio_defectuoso_el/ORG]
295040                                                 []
262982                [bastante/ORG, la_aplicaciones_de/ORG]
89247                                                  []
335663                                            [muy/ORG]
91010   [yo_soy_de_venezuela_el_teléfono_llegó_casi_lo…
85620   [equipo_nuevo_la_batería/PERSON, le_dura_todo/…
91442                          [un/ORG, al_mercado/PERSON]

                                                   tokens language
317616                              [excelente, gracias]       es
235210                              [buenisimo, gracias]       es
94088   [el, equipo, salio, defectuoso, el, trackpad, …       es
295040                               [came, broken, lol]       es
262982  [buen, telefono, bastantes, aplicaciones, util…       es
89247                                           [bueno]       es
335663                                      [muy, bueno]       es
91010   [yo, soy, de, venezuela, el, teléfono, llegó, …       es
85620   [equipo, nuevo, la, batería, le, dura, todo, e…       es
91442   [excelente, equipo, lo, recomiendo, quienes, d…       es
```

Observing Other Foreign Reviews High-level analysis shows that much of these languages don't actually correspond to somali or catalan, but rather, a "type" of review that is generally one word. We can also leverage functionality to see the most common words in each group:

[152]:
```python
def combine_tokens(tokens):
    out = []
    for token_list in tokens:
        for t in token_list:
```

```
            out.append(t)
        #out = out + token_list.shape(0,-1)
    return out

def overview(f):
    print(f'size:{len(f)}')
    print('avg number of tokens per review:')
    print(f['token_count'].mean())
    n = f['tokens'].sample(5,replace=True)
    combined = combine_tokens(f['tokens'])
    print(n)
    print(Counter(combined).most_common(10))
```

[143]:
```
df['token_count'] = df['tokens'].map(len)
df.head(1)
```

[143]:
```
   Rating                                              text  \
0       5   feel lucky found used phone us used hard phone…

                                              lemmas  \
0  [feel, lucky, find, use, phone, use, hard, pho…

                                          adjs_verbs  \
0  [feel, lucky, find, hard, upgrade, sell, like,…

                                               nouns  \
0  [phone, phone, line, son, year, thank, seller,…

                   noun_phrases  \
0  [phone_line, thank_seller]

                                    adj_noun_phrases entities  \
0  [hard_phone, hard_phone_line, old_one, recomme…       []

                                              tokens language  token_count
0  [feel, lucky, found, used, phone, us, used, ha…       en           38
```

[153]:
```
#SOMALI
somali = df[df['language']=='so']
overview(somali)
```

```
size:12541
avg number of tokens per review:
1.1800494378438722
178662    [good]
134232    [good]
85957     [good]
```

```
7329       [good]
27045      [good]
Name: tokens, dtype: object
[('good', 11623), ('bad', 456), ('thanks', 150), ('buy', 148), ('far', 124),
('thank', 113), ('deal', 85), ('job', 72), ('quality', 67), ('x', 43)]
```

[154]:
```
#CATALAN
catalan = df[df['language']=='ca']
overview(catalan)
```

```
size:11002
avg number of tokens per review:
1.4510089074713688
343897      [perfect]
136140    [excellent]
371892      [perfect]
188135    [excellent]
145700    [excellent]
Name: tokens, dtype: object
[('excellent', 6138), ('perfect', 2391), ('exelente', 695), ('great', 364),
('quality', 264), ('good', 247), ('exelent', 232), ('excelent', 184), ('camera',
174), ('value', 167)]
```

[155]:
```
#FRENCH
french = (df[df['language']=='fr'])
overview(french)
```

```
size:10364
avg number of tokens per review:
3.9173099189502123
36718                              [excellent, product]
232149                      [dead, 2, days, use, return]
81105      [apples, samsungs, lgs, favorite, phone, far]
219627        [excelent, phone, came, great, conditions]
61121                                [beautiful, phone]
Name: tokens, dtype: object
[('excellent', 5391), ('phone', 4527), ('product', 2331), ('price', 694),
('love', 663), ('recommend', 603), ('good', 574), ('seller', 563), ('excelent',
456), ('great', 447)]
```

Punjabi is another language group that, while incorrectly tagged as Punjabi, seems to contain many words that are actually spanish:

[156]:
```
#PUNJABI
punjabi = df[df['language']=='pt']
overview(punjabi)
```

```
size:8723
avg number of tokens per review:
```

```
1.2965722801788375
302313      [excelente]
105617     [excellente]
130888      [excelente]
234571      [excelente]
201346       [excelent]
Name: tokens, dtype: object
[('excelente', 5610), ('excelent', 2339), ('recomendado', 262), ('100', 222),
('e', 102), ('item', 94), ('described', 79), ('good', 74), ('sim', 59),
('producto', 45)]
```

it seems that excelente is the most commonly used word in "punjabi" reviews, but analysis seems to show that most of these reviews are literally just the word excelente. If we can modify these reviews to replace excelente with excelent, we can probably avoid removing them.

Iterating over all languages

[157]:
```python
for lang in langs.index:
    if lang != "es":
        print(f"==========| {lang} |============")
        overview(df[df['language']==lang])
```

```
==========| so |============
size:12541
avg number of tokens per review:
1.1800494378438722
98241              [good]
57404         [far, good]
114511             [good]
105827     [good, thanks]
404637             [good]
Name: tokens, dtype: object
[('good', 11623), ('bad', 456), ('thanks', 150), ('buy', 148), ('far', 124),
('thank', 113), ('deal', 85), ('job', 72), ('quality', 67), ('x', 43)]
==========| ca |============
size:11002
avg number of tokens per review:
1.4510089074713688
200082      [xcelente]
292367       [perfect]
175776      [exelente]
393770     [excellent]
234960     [excellent]
Name: tokens, dtype: object
[('excellent', 6138), ('perfect', 2391), ('exelente', 695), ('great', 364),
('quality', 264), ('good', 247), ('exelent', 232), ('excelent', 184), ('camera',
174), ('value', 167)]
==========| fr |============
size:10364
```

```
avg number of tokens per review:
3.9173099189502123
176464                                    [dont, buy]
341354                            [complaints, phone]
29924     [phone, came, perfect, condition, problems]
144150              [plug, phone, sent, charge, good]
318017       [product, seller, describe, double, sim]
Name: tokens, dtype: object
[('excellent', 5391), ('phone', 4527), ('product', 2331), ('price', 694),
('love', 663), ('recommend', 603), ('good', 574), ('seller', 563), ('excelent',
456), ('great', 447)]
==========| pt |============
size:8723
avg number of tokens per review:
1.2965722801788375
403003    [excelente]
128014    [excelente]
108608    [excelente]
155985     [excelent]
134064     [excelent]
Name: tokens, dtype: object
[('excelente', 5610), ('excelent', 2339), ('recomendado', 262), ('100', 222),
('e', 102), ('item', 94), ('described', 79), ('good', 74), ('sim', 59),
('producto', 45)]
==========| af |============
size:8721
avg number of tokens per review:
3.3864235752780645
384334         [workin, great, great, seller, would, vouch]
130486                                            [awesome]
222977    [great, battery, life, like, use, stylus, gps,…
34314     [phone, great, used, looks, works, like, brand…
27007                     [loved, til, lost, week, getting]
Name: tokens, dtype: object
[('works', 2738), ('great', 1673), ('like', 1581), ('work', 1513), ('good',
1405), ('awesome', 920), ('phone', 907), ('working', 842), ('well', 829),
('new', 620)]
==========| ro |============
size:8395
avg number of tokens per review:
1.89053007742704
205971                   [crap]
19881        [experience, great]
158298    [excelente, telefono]
319049    [great, product, far]
28013                   [great]
Name: tokens, dtype: object
[('great', 5101), ('product', 1823), ('excelente', 865), ('producto', 704),
```

```
('nice', 645), ('price', 621), ('excelent', 356), ('perfect', 291), ('love',
245), ('perfecto', 232)]
==========| sl |============
size:4838
avg number of tokens per review:
1.4956593633732949
289674    [good, phone, love]
123476                 [love]
397439                 [love]
41101                  [love]
1292              [im, loven]
Name: tokens, dtype: object
[('love', 4251), ('loved', 286), ('good', 256), ('phone', 227), ('like', 133),
('problems', 129), ('item', 67), ('nice', 58), ('job', 56), ('loves', 45)]
==========| cy |============
size:3363
avg number of tokens per review:
2.232827832292596
207594           [good, gold]
65668             [far, good]
69016                   [gr8]
135845    [good, cell, money]
197116    [good, cell, phone]
Name: tokens, dtype: object
[('good', 2623), ('phone', 1747), ('far', 292), ('new', 153), ('well', 133),
('cell', 118), ('cellphone', 95), ('god', 76), ('awsome', 75), ('one', 72)]
==========| sk |============
size:3156
avg number of tokens per review:
1.3517110266159695
382444    [100, ok]
44637          [ok]
357182         [ok]
1317           [ok]
129384         [ok]
Name: tokens, dtype: object
[('ok', 2491), ('love', 367), ('phone', 343), ('mom', 85), ('price', 76),
('loved', 66), ('nice', 60), ('okey', 57), ('slow', 53), ('loves', 41)]
==========| it |============
size:3139
avg number of tokens per review:
3.375915896782415
271828                          [impress]
270584    [came, damaged, near, camera]
354318               [love, cell, phone]
308538                           [supper]
11984                 [fascinante, nice]
Name: tokens, dtype: object
```

```
[('phone', 590), ('fine', 402), ('love', 278), ('amazing', 268), ('nice', 244),
('cell', 239), ('price', 198), ('cellphone', 198), ('good', 167), ('time', 155)]
==========| da |============
size:2411
avg number of tokens per review:
3.894649523019494
236395                                [son, loved, gift]
338336    [error, making, call, imm, code, error, satisfy]
226403                                [best, nexus, far]
129959                                   [best, android]
43511                            [granddaughter, loves]
Name: tokens, dtype: object
[('phone', 436), ('best', 381), ('love', 292), ('gift', 269), ('great', 268),
('loves', 254), ('ever', 208), ('delivered', 170), ('get', 150), ('like', 145)]
==========| nl |============
size:2307
avg number of tokens per review:
4.743389683571738
210249    [excellent, good, delivery, hope, reach, even,…
161741                              [one, word, excelent]
355567                                        [exselente]
160558    [best, blu, smartphone, looks, feels, amazing,…
339049                        [works, venezuela, 3g, great]
Name: tokens, dtype: object
[('phone', 559), ('work', 535), ('get', 392), ('screen', 294), ('good', 293),
('doesnt', 270), ('venezuela', 257), ('works', 189), ('dont', 171), ('even',
164)]
==========| no |============
size:2119
avg number of tokens per review:
3.8725814063237376
212009                     [lg, g3, better, apple, samsung]
35773                            [8, g, never, big, enough]
280771    [great, love, item, great, seller, got, item, …
226273                      [satisfied, enjoying, nexus, 5]
28606           [goog, value, mony, great, support, seller]
Name: tokens, dtype: object
[('like', 589), ('phone', 544), ('great', 308), ('better', 255), ('love', 220),
('advertised', 182), ('seller', 151), ('far', 124), ('problems', 115), ('ok',
97)]
==========| pl |============
size:1915
avg number of tokens per review:
1.3733681462140992
195889    [nice]
358857    [nice]
180817    [nice]
112274    [nice]
```

```
246594    [nice]
Name: tokens, dtype: object
[('nice', 1675), ('good', 57), ('work', 55), ('ok', 48), ('works', 44),
('watch', 37), ('wow', 33), ('phone', 33), ('piece', 29), ('slow', 28)]
==========| et |============
size:1580
avg number of tokens per review:
1.9753164556962026
99100                                                [like]
314281    [like, updates, take, storage, phone, vs, savi…
167522                                          [looks, ok]
234572                                               [like]
75366                                               [liked]
Name: tokens, dtype: object
[('like', 947), ('looks', 143), ('phone', 138), ('liked', 127), ('good', 107),
('one', 41), ('ok', 31), ('looking', 30), ('great', 29), ('poor', 29)]
==========| hr |============
size:1137
avg number of tokens per review:
2.70712401055409
228957                       [good, price]
153664    [love, camera, take, nice, pic]
259549                     [good, product]
303322                 [ok, good, problem]
40829                      [good, product]
Name: tokens, dtype: object
[('good', 806), ('product', 307), ('price', 282), ('like', 171), ('phone', 137),
('amazing', 129), ('nice', 103), ('ok', 80), ('love', 76), ('service', 45)]
==========| UNKNOWN |============
size:1110
avg number of tokens per review:
0.05945945945945946
283003    [100]
45094        []
285451       []
156651       []
262060       []
Name: tokens, dtype: object
[('100', 32), ('5', 9), ('55', 6), ('3', 4), ('10', 4), ('1', 3), ('100100', 3),
('12345', 3), ('1010', 1), ('000', 1)]
==========| tl |============
size:891
avg number of tokens per review:
2.7037037037037037
158143               [nan]
363547    [amazing, phone]
328935               [nan]
248639    [amazing, phone]
```

```
278635               [nil]
Name: tokens, dtype: object
[('phone', 269), ('amazing', 207), ('samsung', 133), ('okay', 82), ('galaxy',
70), ('good', 62), ('nan', 62), ('big', 61), ('say', 50), ('looking', 45)]
==========| sv |============
size:801
avg number of tokens per review:
2.7952559300873907
30339                      [5, stars]
109077                         [gift]
334390                  [five, stars]
102506    [unlocked, get, lock, phone]
27905                       [5, star]
Name: tokens, dtype: object
[('unlocked', 170), ('5', 87), ('gift', 83), ('stars', 77), ('far', 72),
('star', 64), ('ok', 53), ('great', 43), ('small', 34), ('get', 32)]
==========| fi |============
size:585
avg number of tokens per review:
1.3076923076923077
144720                [okay]
341078               [happy]
172824    [absolutely, junk]
181708          [junk, phone]
104609                [junk]
Name: tokens, dtype: object
[('happy', 180), ('junk', 117), ('small', 54), ('okay', 50), ('sucks', 26),
('luv', 20), ('phone', 17), ('value', 15), ('money', 15), ('ty', 12)]
==========| lv |============
size:539
avg number of tokens per review:
1.1595547309833023
31314     [satisfied]
327886    [satisfied]
188016    [satisfied]
135745         [bien]
234311    [satisfied]
Name: tokens, dtype: object
[('satisfied', 260), ('bien', 211), ('100', 18), ('5', 16), ('size', 15), ('im',
12), ('pies', 12), ('sit', 12), ('plums', 8), ('tiempo', 6)]
==========| sw |============
size:493
avg number of tokens per review:
1.2880324543610548
290415         [Ok]
79281     [amazing]
332176          [k]
19886           [k]
```

```
367318    [amazing]
Name: tokens, dtype: object
[('amazing', 318), ('much', 67), ('like', 60), ('want', 17), ('k', 15), ('fake',
14), ('okay', 8), ('hi', 6), ('watch', 6), ('weak', 6)]
==========| tr |============
size:395
avg number of tokens per review:
2.0151898734177216
43012                 [yea]
224340                [yes]
287635        [little, bulky]
68653     [yes, nice, mobile]
303147                [yes]
Name: tokens, dtype: object
[('yes', 208), ('buy', 48), ('bulky', 27), ('bad', 23), ('nice', 17), ('mobile',
15), ('güzel', 14), ('ürün', 14), ('ama', 14), ('satıcı', 14)]
==========| de |============
size:367
avg number of tokens per review:
1.7438692098092643
209667    [android, 44, sehr, schlecht]
17695                      [glitches]
5125                           [bien]
76379              [much, faster, 5s]
156814                         [bien]
Name: tokens, dtype: object
[('bien', 190), ('item', 28), ('best', 18), ('new', 17), ('glitches', 17),
('im', 14), ('described', 13), ('much', 11), ('daughter', 10), ('satisfied',
10)]
==========| id |============
size:324
avg number of tokens per review:
2.175925925925926
110919             [buenas]
379908             [superb]
16437          [bad, batery]
126070     [beautiful, like]
267722    [stunning, camera]
Name: tokens, dtype: object
[('bad', 52), ('buy', 35), ('garbage', 33), ('buena', 31), ('dont', 22),
('superb', 21), ('camera', 21), ('sim', 19), ('battery', 17), ('didnt', 15)]
==========| cs |============
size:298
avg number of tokens per review:
2.422818791946309
165282                         [nice, photos]
107171              [good, productvery, nice]
511                            [volume, loud]
```

```
140944                    [love, blu, products]
69891     [nice, phone, problem, memory, space]
Name: tokens, dtype: object
[('love', 124), ('nice', 92), ('problem', 73), ('product', 52), ('phone', 39),
('much', 36), ('ok', 19), ('kyou', 17), ('vry', 17), ('memory', 12)]
==========| hu |============
size:165
avg number of tokens per review:
1.981818181818182
409917           [advertized]
302037                   [a1]
24878     [gets, frozen, lot]
55227                    [a1]
58617                    [a1]
Name: tokens, dtype: object
[('a1', 44), ('amazon', 20), ('amazing', 18), ('ok', 16), ('lovely', 14),
('love', 12), ('lot', 11), ('tks', 11), ('frozen', 10), ('gets', 8)]
==========| lt |============
size:141
avg number of tokens per review:
1.375886524822695
332863                                        [buenisimo]
234790                                        [buenisimo]
357757    [love, s5, duos, central, nebraska, viaero, ne…
87816                                         [buenisimo]
171317                                        [buenisimo]
Name: tokens, dtype: object
[('buenisimo', 85), ('audio', 7), ('kargia', 6), ('sturdy', 6), ('bad', 5),
('returning', 5), ('buenismo', 5), ('sirvio', 4), ('optimo', 4), ('gracias', 3)]
==========| sq |============
size:129
avg number of tokens per review:
2.007751937984496
28258           [great, shape]
214540            [like, item]
289325       [dont, like, much]
327401            [like, item]
179167    [fake, quality, poor]
Name: tokens, dtype: object
[('great', 28), ('time', 26), ('like', 25), ('fit', 16), ('shape', 15), ('item',
11), ('trash', 10), ('dont', 9), ('return', 8), ('enjoy', 8)]
==========| vi |============
size:112
avg number of tokens per review:
2.267857142857143
413691                           [thx]
312659                    [4g, phone]
321887                  [thing, huge]
```

```
181510                          [thumb]
174693    [  t,    ,   t ,   , phone]
Name: tokens, dtype: object
[('phone', 69), ('thx', 15), ('4g', 12), ('buy', 11), ('big', 8), ('thí ', 8),
('ph n ', 8), ('  t', 7), ('   ', 7), (' t ', 7)]
==========| ru |============
size:2
avg number of tokens per review:
15.5
277358    [  ,    ,     ,  ,     ,     ,   ,…
197532    [  ,    ,      ,      ,   ,       …
277358    [  ,    ,     ,  ,     ,     ,   ,…
277358    [  ,    ,     ,  ,     ,     ,   ,…
197532    [  ,    ,      ,      ,   ,       …
Name: tokens, dtype: object
[('  ', 2), ('   ', 2), (' ', 2), ('  ', 2), ('     ', 1), ('     ',
1), ('      ', 1), ('      ', 1), (' ', 1), ('htc', 1)]
```

1.1.7 Candidates for Removal

Country

ISO

Spanish

es

Russian

ru

1.1.8 Candidates for Replacement

Word

Replacement

Excelente

Excellent

Producto

Product

Recomendado

Recommend

Replacement and Removal

[174]:
```
#remove spanish reviews
df = df.drop(df[df['language']=='es'].index)
df = df.drop(df[df['language']=='ru'].index)
```

[167]:
```
#remap words
word_remap = {
    'excelente':'excellent',
    'producto':'product',
    'recomendado':'recommend'
}

def word_replace(text):
    out = text
    for k,v in word_remap.items():
        out = re.sub(k,v,out)
    return out
```

[179]:
```
#remove all foreign artifacts
df['new_reviews'] = df['text'].map(word_replace)
df.head(1)
```

[179]:
```
   Rating                                               text  \
0       5  feel lucky found used phone us used hard phone…

                                              lemmas  \
0  [feel, lucky, find, use, phone, use, hard, pho…

                                          adjs_verbs  \
0  [feel, lucky, find, hard, upgrade, sell, like,…

                                               nouns  \
0  [phone, phone, line, son, year, thank, seller,…

                  noun_phrases  \
0  [phone_line, thank_seller]

                                   adj_noun_phrases entities  \
0  [hard_phone, hard_phone_line, old_one, recomme…       []

                                              tokens language  token_count  \
0  [feel, lucky, found, used, phone, us, used, ha…       en           38

                                         new_reviews
0  feel lucky found used phone us used hard phone…
```

[194]:
```
#Sanity check
print(len(df[df['language']=='pt']))
df[df['language']=='pt'].head()
```

```
8723
```

[194]:
```
    Rating       text       lemmas   adjs_verbs nouns noun_phrases  \
40       5  excelente  [excellent]  [excellent]    []           []
41       5  excelente  [excellent]  [excellent]    []           []
58       5  excelente  [excellent]  [excellent]    []           []
60       5  excelente  [excellent]  [excellent]    []           []
65       5  excelente  [excellent]  [excellent]    []           []

   adj_noun_phrases entities       tokens language  token_count new_reviews
40               []       []  [excelente]       pt            1   excellent
41               []       []  [excelente]       pt            1   excellent
58               []       []  [excelente]       pt            1   excellent
60               []       []  [excelente]       pt            1   excellent
65               []       []  [excelente]       pt            1   excellent
```

1.1.9 Retokenization

because of the fact that we replaced three different words in a significant portion of our corpus, we need to rerun the spacy tokenization pipeline for those rows. We can save time by running the pipeline **only** on those chunks:

[186]:
```python
#apply this pipeline to our df to generate tokens:
def extract_nlp(doc):
    return {
        'lemmas' : extract_lemmas(doc,
        exclude_pos = ['PART', 'PUNCT',
        'DET', 'PRON', 'SYM', 'SPACE'],
        filter_stops = False),
        'adjs_verbs' : extract_lemmas(doc, include_pos = ['ADJ', 'VERB']),
        'nouns' : extract_lemmas(doc, include_pos = ['NOUN', 'PROPN']),
        'noun_phrases' : extract_noun_phrases(doc, ['NOUN']),
        'adj_noun_phrases': extract_noun_phrases(doc, ['ADJ']),
        'entities' : extract_entities(doc, ['PERSON', 'ORG', 'GPE', 'LOC'])
    }

def extract_lemmas(doc, **kwargs):
    return [t.lemma_ for t in textacy.extract.words(doc, **kwargs)]

def extract_noun_phrases(doc, preceding_pos=['NOUN'], sep='_'):
    patterns = []
    for pos in preceding_pos:
        patterns.append(f"POS:{pos} POS:NOUN:+")
    spans = textacy.extract.matches.token_matches(doc, patterns=patterns)
    return [sep.join([t.lemma_ for t in s]) for s in spans]

def extract_entities(doc, include_types=None, sep='_'):
    ents = textacy.extract.entities(doc,
    include_types=include_types,
```

```
    exclude_types=None,
    drop_determiners=True,
    min_freq=1)
    return [sep.join([t.lemma_ for t in e])+'/'+e.label_ for e in ents]
```

[193]:
```
import time
import tqdm.notebook as tq
start = time.localtime()
batch_size = 50

for i in tq.tqdm(range(0, len(df), batch_size),position=0,leave=True):
    docs = nlp.pipe(df['new_reviews'][i:i+batch_size])
    for j, doc in enumerate(docs):
        #Only replace the tokens if the review was modified somehow
        if df['text'].iloc[i+j] != df['new_reviews'].iloc[i+j]:
            for col, values in extract_nlp(doc).items():
                df[col].iloc[i+j] = values


end = time.localtime()
print(start,end)
```

```
  0%|          | 0/8114 [00:00<?, ?it/s]
```

```
time.struct_time(tm_year=2022, tm_mon=6, tm_mday=25, tm_hour=15, tm_min=11,
tm_sec=58, tm_wday=5, tm_yday=176, tm_isdst=1) time.struct_time(tm_year=2022,
tm_mon=6, tm_mday=25, tm_hour=15, tm_min=30, tm_sec=28, tm_wday=5, tm_yday=176,
tm_isdst=1)
```

1.1.10 Export

[199]:
```
df.drop(columns=['text','language']).to_parquet(
    'prepared_text.parquet.gzip',
    compression='gzip',
    index=False)
```

Notebook_2_SupervisedModels

June 26, 2022

0.1 Supervised Modeling of Reviews

In this section, we will build a text classication using supervised learning technique which can can predict the rating based on the reviews.

0.1.1 Packages

[1]:
```
import kaggle
import pandas as pd
import matplotlib.pyplot as plt
import pyarrow
import fastparquet
import numpy as np
import os
from collections import Counter, defaultdict
import warnings
import seaborn as sns
warnings.filterwarnings("ignore")
from wordcloud import WordCloud
#kaggle.api.authenticate()
import nltk
from string import punctuation
import textacy.preprocessing as tprep
from nltk.corpus import stopwords
from wordcloud import WordCloud, STOPWORDS
import spacy
nlp = spacy.load("en_core_web_sm")
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.linear_model import LinearRegression
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,␣
↪confusion_matrix,plot_confusion_matrix
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import cross_val_score
```

```python
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
```

0.2 Step 1: Data Preparation

Loading Dataset for Modeling

[2]:
```python
new_df=pd.read_parquet('prepared_text.parquet.gzip', engine='pyarrow')
new_df.head(3)
```

[2]:
```
   Rating                                             lemmas  \
0       5  [feel, lucky, find, use, phone, use, hard, pho…
1       4  [nice, phone, nice, grade, pantach, revue, cle…
2       5                                         [pleased]

                                         adjs_verbs  \
0  [feel, lucky, find, hard, upgrade, sell, like,…
1  [nice, nice, clean, easy, android, fantastic, …
2                                          [pleased]

                                              nouns  \
0  [phone, phone, line, son, year, thank, seller,…
1  [phone, grade, pantach, revue, set, set, phone…
2                                                []

                noun_phrases  \
0  [phone_line, thank_seller]
1             [grade_pantach]
2                          []

                                   adj_noun_phrases       entities  \
0  [hard_phone, hard_phone_line, old_one, recomme…             []
1  [nice_phone, nice_grade, nice_grade_pantach, c…  [android/GPE]
2                                                []             []

                                             tokens  token_count  \
0  [feel, lucky, found, used, phone, us, used, ha…           38
1  [nice, phone, nice, grade, pantach, revue, cle…           24
2                                         [pleased]            1

                                        new_reviews
0  feel lucky found used phone us used hard phone…
1  nice phone nice grade pantach revue clean set …
2                                           pleased
```

Removing unecessary columns prior to modeling

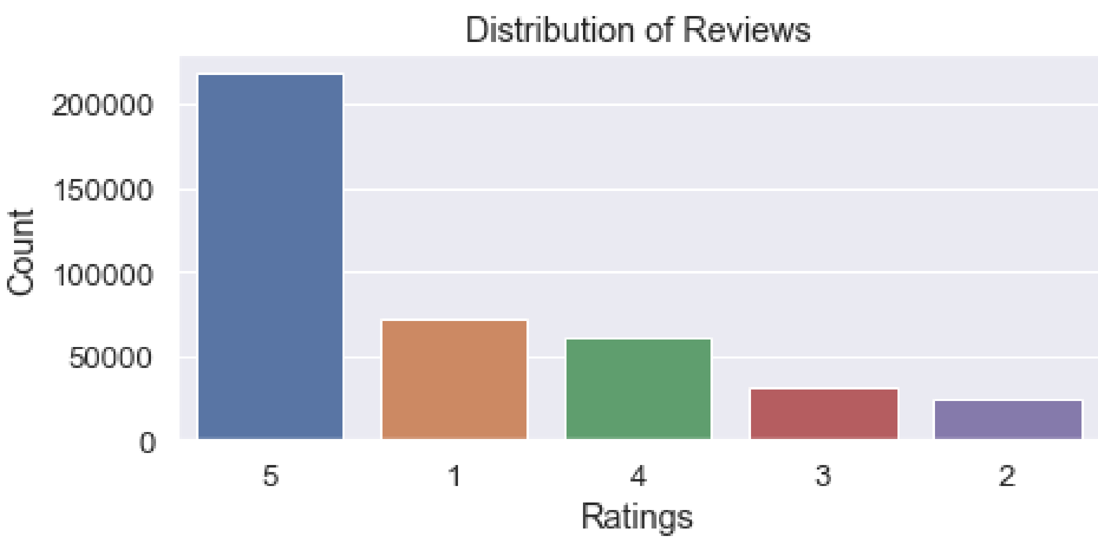
[3]:
```
new_df =new_df[[
    'Rating','new_reviews','tokens'
]]
```

[4]:
```
new_df.info()
print('\n')
print(new_df.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 405669 entries, 0 to 405668
Data columns (total 3 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Rating       405669 non-null  int64
 1   new_reviews  405669 non-null  object
 2   tokens       405669 non-null  object
dtypes: int64(1), object(2)
memory usage: 9.3+ MB


(405669, 3)
```

Checking for Class Imbalance

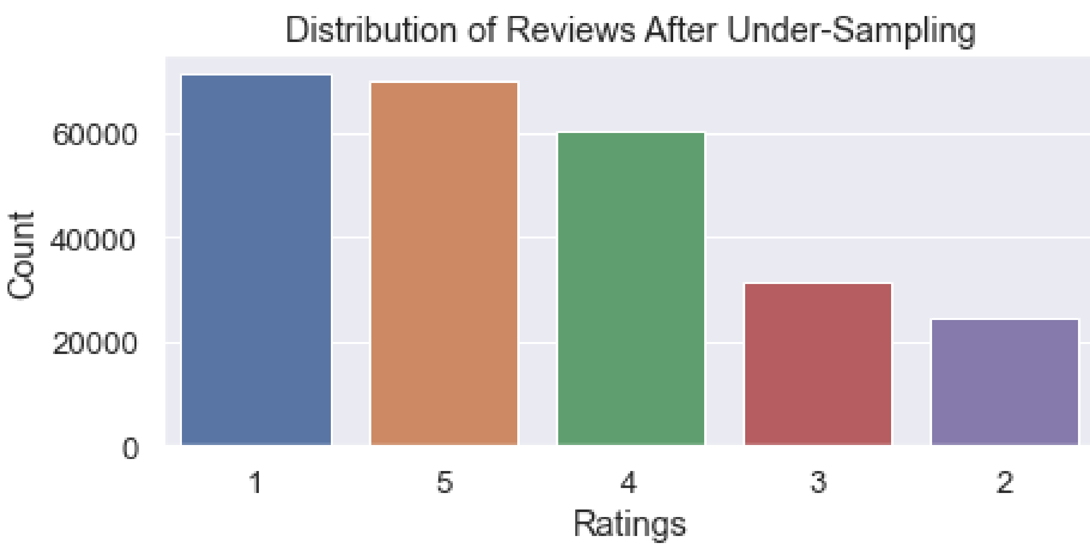
[5]:
```
plt.figure(figsize=(7,3))
sns.set(font_scale = 1.2)
sns.countplot(x="Rating", data=new_df,
order = new_df['Rating'].value_counts().index)
plt.title("Distribution of Reviews")
plt.xlabel("Ratings")
plt.ylabel("Count")
plt.show()
```



Undersampling Majority class

[6]:
```
# undersampling 5-star reviews and oversampling other reviews
five_stars = new_df[new_df['Rating'] == 5].sample(n=70000)
non_five = new_df[new_df['Rating'] != 5 ]
df_bal = pd.concat([five_stars,non_five],axis=0)
```

[7]:
```
plt.figure(figsize=(7,3))
sns.set(font_scale = 1.2)
sns.countplot(x="Rating", data=df_bal,
order = df_bal['Rating'].value_counts().index)
plt.title("Distribution of Reviews After Under-Sampling")
plt.xlabel("Ratings")
plt.ylabel("Count")
plt.show()
```



0.3 Step 2: Train-Test Split on Balanced Dataset

[9]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(df_bal['new_reviews'],
                                                    df_bal['Rating'],
                                                     test_size=0.2,
                                                     random_state=42,
                                                    ␣
  ↪stratify=df_bal['Rating'])
print ('Size of Training Data ', X_train.shape[0])
```

```
print ('Size of Test Data ', X_test.shape[0])
print ('Distribution of classes in Training Data :')
#print ('Positive Sentiment ', str(sum(Y_train == 1)/ len(Y_train) * 100.0))
# print ('Negative Sentiment ', str(sum(Y_train == 0)/ len(Y_train) * 100.0))
#print ('Distribution of classes in Testing Data :')
#print ('Positive Sentiment ', str(sum(Y_test == 1)/ len(Y_test) * 100.0))
#print ('Negative Sentiment ', str(sum(Y_test == 0)/ len(Y_test) * 100.0))
```

```
Size of Training Data  205997
Size of Test Data  51500
Distribution of classes in Training Data :
```

0.4 Step 3: Text Vectorization

Using TF-IDF vectorization to create the vectorized representation:

[10]:
```
def identity_tokenizer(text):
  return text


tfidf = TfidfVectorizer(tokenizer=identity_tokenizer,min_df =␣
 ↪10,ngram_range=(1,2),lowercase=False)
X_train_tf = tfidf.fit_transform(X_train)
X_test_tf = tfidf.transform(X_test)
```

0.5 Step 4: Training the Machine Learning Models

Model : Linear SVC

[11]:
```
model_svc = LinearSVC(random_state=0, tol=1e-5)
model_svc.fit(X_train_tf, Y_train)
```

[11]:
```
LinearSVC(random_state=0, tol=1e-05)
```

Step 5 : Model Evaluation

[12]:
```
Y_pred_svc = model_svc.predict(X_test_tf)
print ('Accuracy Score  - ', accuracy_score(Y_test, Y_pred_svc))
#print ('ROC-AUC Score - ', roc_auc_score(Y_test, Y_pred_svc))
print(classification_report(Y_test, Y_pred_svc))
```

```
Accuracy Score  -  0.5506407766990291
              precision    recall  f1-score   support

           1       0.57      0.87      0.69     14275
           2       0.38      0.02      0.03      4907
           3       0.40      0.08      0.13      6277
           4       0.45      0.44      0.44     12041
           5       0.62      0.72      0.67     14000
```

```
    accuracy                           0.55     51500
   macro avg       0.48      0.42      0.39     51500
weighted avg       0.51      0.55      0.49     51500
```

0.5.1 Baseline Model Evaluation

[13]:
```
clf_baseline = DummyClassifier(strategy='stratified')
clf_baseline.fit(X_train, Y_train)
Y_pred_baseline = clf_baseline.predict(X_test)
print ('Accuracy Score - ', accuracy_score(Y_test, Y_pred_baseline))
```

```
Accuracy Score -  0.23135922330097086
```

0.5.2 Performing Hyperparameter Tuning with Grid Search

[14]:
```
training_pipeline = Pipeline(
        steps=[('tfidf', TfidfVectorizer(stop_words="english")),
                ('model', LinearSVC(random_state=42, tol=1e-5))])
grid_param = [{
        'tfidf__min_df': [5, 10],
        'tfidf__ngram_range': [(1, 3), (1, 6)],
        'model__penalty': ['l2'],
        'model__loss': ['hinge'],
        'model__max_iter': [10000]
},{
'tfidf__min_df': [5, 10], 'tfidf__ngram_range': [(1, 3), (1, 6)], 'model__C':␣
 ↪[1, 10],
'model__tol': [1e-2, 1e-3]
}]
gridSearchProcessor = GridSearchCV(estimator=training_pipeline,
                                   param_grid=grid_param,
                                   cv=5)
gridSearchProcessor.fit(df_bal['new_reviews'], df_bal['Rating'])
best_params = gridSearchProcessor.best_params_

print("Best alpha parameter identified by grid search ", best_params)
best_result = gridSearchProcessor.best_score_
print("Best result identified by grid search ", best_result)
```

```
Best alpha parameter identified by grid search  {'model__loss': 'hinge',
'model__max_iter': 10000, 'model__penalty': 'l2', 'tfidf__min_df': 5,
'tfidf__ngram_range': (1, 6)}
Best result identified by grid search  0.5797621233101519
```

[15]:
```
gridsearch_results = pd.DataFrame(gridSearchProcessor.cv_results_)
gridsearch_results[['rank_test_score', 'mean_test_score',
                        'params']].sort_values(by=['rank_test_score'])[:5]
```

[15]:
```
   rank_test_score  mean_test_score  \
1                1         0.579762
0                2         0.577976
5                3         0.576752
9                4         0.576737
8                5         0.576139

                                              params
1  {'model__loss': 'hinge', 'model__max_iter': 10…
0  {'model__loss': 'hinge', 'model__max_iter': 10…
5  {'model__C': 1, 'model__tol': 0.01, 'tfidf__mi…
9  {'model__C': 1, 'model__tol': 0.001, 'tfidf__m…
8  {'model__C': 1, 'model__tol': 0.001, 'tfidf__m…
```

[20]:
```python
best_chosen_model=gridSearchProcessor.best_estimator_
```

Model Evaluation After Hyper-parameter Tuning

[22]:
```python
Y_pred_bestmodel = best_chosen_model.predict(X_test)
print('Accuracy Score - ', accuracy_score(Y_test, Y_pred_bestmodel))
print(classification_report(Y_test, Y_pred_bestmodel))
```

```
Accuracy Score -  0.8856116504854369
              precision    recall  f1-score   support

           1       0.90      0.98      0.94     14275
           2       0.98      0.81      0.89      4907
           3       0.96      0.82      0.88      6277
           4       0.83      0.85      0.84     12041
           5       0.86      0.88      0.87     14000

    accuracy                           0.89     51500
   macro avg       0.91      0.87      0.88     51500
weighted avg       0.89      0.89      0.89     51500
```

Notebook_3_Unsupervised_Models

June 26, 2022

1 Unsupervised Modeling of Reviews

In this section, we ignore the A Priori classes that our review dataset belongs to and created two unsupervised models - LDA and NMF - and see how the clusters we generated compare to the classes the data belongs to.

1.1 Table of Contents

1.1.1 Packages

[1]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from spacy.lang.en.stop_words import STOP_WORDS as stopwords
import matplotlib.pyplot as plt
from imblearn.over_sampling import RandomOverSampler
from sklearn.decomposition import NMF
from collections import Counter, defaultdict
import warnings #turn off warnings
```

```
warnings.filterwarnings("ignore", category=UserWarning)
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

#GENSIM MODELING - LDA AND NMF
from gensim.models.nmf import Nmf
from gensim.models import LdaModel
from gensim.models import TfidfModel
from gensim.corpora import Dictionary
from IPython.display import Image
```

1.1.2 Parquet Ingestion

[44]:
```
df = pd.read_parquet('prepared_text.parquet.gzip')
```

[45]:
```
df = df.rename({"new_reviews":"text"},axis=1)
df.sample(3)
```

[45]:
```
        Rating                                              lemmas  \
111155       1  [keep, freeze, keep, glitching, suppose, chris…
374738       4  [buy, phone, last, month, honest, first, exper…
229087       3  [buy, husband, week, thus, far, like, shall, s…

                                               adjs_verbs  \
111155                 [keep, freeze, keep, suppose, happy]
374738  [buy, honest, take, familiar, android, operatt…
229087                                    [buy, come, hold]

                                                    nouns  \
111155                  [glitching, christmas, gift, wife]
374738  [phone, month, experience, smartphone, day, op…
229087                 [husband, week, like, day, rereview]

                   noun_phrases  \
111155                [gift_wife]
374738  [experience_smartphone]
229087           [husband_week]

                                        adj_noun_phrases entities  \
111155                                                []       []
374738  [last_month, first_experience, first_experienc…       []
229087                                                []       []

                                                  tokens  token_count  \
111155  [keeps, freezing, keeps, glitching, suppose, c…           10
374738  [bought, phone, last, month, honest, first, ex…           21
229087  [bought, husband, week, thus, far, likes, shal…           12
```

```
                                                       text
111155  keeps freezing keeps glitching suppose christm…
374738  bought phone last month honest first experienc…
229087  bought husband week thus far likes shall see d…
```

as a result of our data cleaning pipeline, much of the groundwork needed for unsupervised analysis has already been laid out for us, primarily in the realm of tokenization. Based on the performance of our LDA and NMF models, we may consider modifying the text we pass to either model by omitting certain parts of speech or entities to improve model performance.

1.1.3 Creating Recyclable Functionality for Modeling

this notebook utilizes the **Gensim** library for modeling LDA and NMF. We can reduce the number of lines of code we write significantly by making this process general-form and creating a function that takes in a data series and prints out topics and their coherence scores. **This also gives us the benefit of having real-time evaluation of model performance.**

[36]:
```
def display_topics_gensim(model):
    for topic in range(0, model.num_topics):
        print("\nTopic %02d" % topic)
        for (word, prob) in model.show_topic(topic, topn=5):
            print("  %s (%2.2f)" % (word, prob))
```

[82]:
```
def get_model_gensim(tokens,model_type,topics=5):

    if model_type not in ['LDA','NMF']:
        print('Not a model - please select from LDA or NMF.')
        return

    #data prep
    dict_gensim = Dictionary(tokens)
    dict_gensim.filter_extremes(no_below=5, no_above=0.7)
    bow_gensim = [dict_gensim.doc2bow(t) for t in tokens]

    #init placeholder for model
    this_model = None

    if model_type == 'LDA':
        this_model = LdaModel(
            corpus=bow_gensim,
            id2word=dict_gensim,
            chunksize=2000,
            alpha='auto',
            eta='auto',
            iterations=400,
            num_topics=topics,
            passes=20,
```

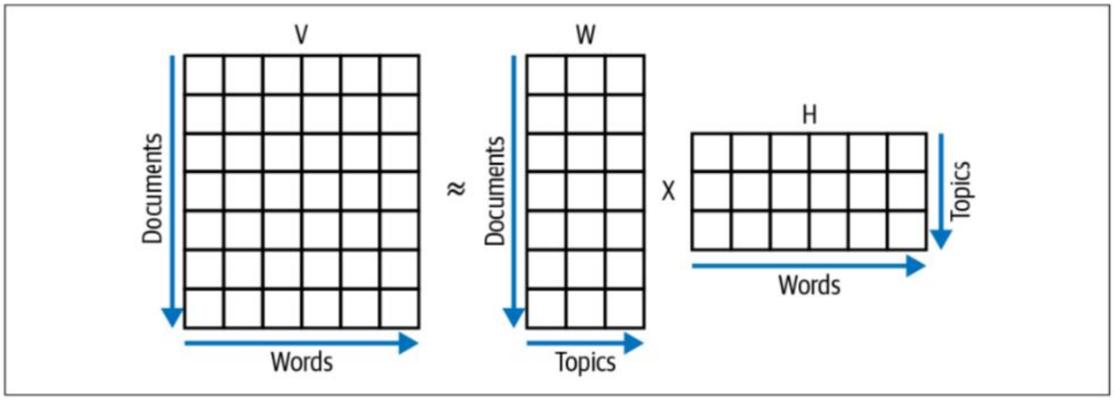
```
            eval_every=None,
            random_state=42
        )

    if model_type == 'NMF':
        #conduct a TF-IDF transformation for NMF
        tfidf_gensim = TfidfModel(bow_gensim)
        vectors_gensim = tfidf_gensim[bow_gensim]

        #run model
        this_model = Nmf(
            vectors_gensim,
            num_topics=topics,
            id2word=dict_gensim,
            kappa=0.1,
            eval_every=5
        )

    #print out topic words
    print(f'Topics for {model_type}')
    display_topics_gensim(this_model)

    #print out coherence score
    score = CoherenceModel(
        model=this_model,
        texts=tokens,
        dictionary=dict_gensim,
        coherence='u_mass'
    )
    this_coh_score = score.get_coherence()
    print(f'Coherence score for {model_type}: {this_coh_score}')

    return (this_model)
```

1.1.4 Nonnegative Matrix Factorization (NMF) with Gensim

The easiest way to find a latent structure in a doc matrix is by factorizing it and seeing what's left over. Because a TFIDF matrix always has positive values, we can represent a document matrix as the product of two smaller matrices: V = W * H, where **W has the same number of rows as V and represent the topic mapping for each document, and H shows how the topics are constituted of features.**

[2]:
```
Image(filename='nmf.jpg')
```

[2]:



we begin by running an NMF analysis for four different types of data - the full review text, only the verb-adjectives, adjective_nouns, and then a combination of the latter two.

1.1.5 NMF With Full Text

[48]:
```
get_model_gensim(df['tokens'],'NMF')
```

```
Topics for NMF

Topic 00
  like (0.03)
  price (0.02)
  work (0.01)
  thanks (0.01)
  awesome (0.01)

Topic 01
  excellent (0.13)
  product (0.08)
  perfect (0.06)
  nice (0.05)
  ok (0.03)

Topic 02
  good (0.71)
  far (0.01)
  condition (0.01)
  phone (0.01)
  quality (0.01)

Topic 03
  great (0.30)
  love (0.14)
```

```
  works (0.11)
  phone (0.05)
  condition (0.03)

Topic 04
  phone (0.01)
  excelente (0.01)
  new (0.01)
  one (0.01)
  battery (0.01)
Coherence score for NMF: -3.955640379099367
```

1.1.6 NMF With Verbs/Adjectives only

[49]:
```
get_model_gensim(df['adjs_verbs'],'NMF')
```

```
Topics for NMF

Topic 00
  good (0.69)
  love (0.08)
  nice (0.02)
  low (0.01)
  fast (0.00)

Topic 01
  good (0.91)
  recommend (0.01)
  thank (0.00)
  slow (0.00)
  overall (0.00)

Topic 02
  new (0.02)
  buy (0.02)
  perfect (0.02)
  love (0.02)
  come (0.01)

Topic 03
  great (0.42)
  work (0.22)
  fast (0.01)
  awesome (0.01)
  easy (0.01)

Topic 04
  excellent (0.61)
```

```
  nice (0.03)
  recommend (0.02)
  thank (0.01)
  fast (0.00)
Coherence score for NMF: -4.730610952079627
```

1.1.7 NMF With Adjectives/Noun Phrases Only

[51]:
```python
get_model_gensim(df['adj_noun_phrases'],'NMF')
```

```
Topics for NMF

Topic 00
  good_condition (0.10)
  great_product (0.07)
  good_product (0.04)
  excellent_product (0.04)
  excellent_condition (0.03)

Topic 01
  good_phone (0.23)
  good_phone_price (0.01)
  easy_use (0.00)
  perfect_condition (0.00)
  excellent_phone (0.00)

Topic 02
  great_phone (0.23)
  great_phone_price (0.01)
  great_phone_love (0.01)
  excellent_phone (0.00)
  easy_use (0.00)

Topic 03
  sim_card (0.09)
  new_phone (0.08)
  easy_use (0.00)
  unlocked_phone (0.00)
  sim_card_phone (0.00)

Topic 04
  nice_phone (0.12)
  great_condition (0.05)
  great_price (0.05)
  excellent_product (0.02)
  excellent_phone (0.01)
Coherence score for NMF: -6.638012922890771
```

1.1.8 NMF With Verbs/Adjs + Adjs/Nouns Phrases

[60]:
```
adj_noun_verbs = df.apply(lambda x: (x.adj_noun_phrases.tolist() + x.adjs_verbs.
 ↪tolist()) , axis=1)
get_model_gensim(adj_noun_verbs,'NMF')
```

```
Topics for NMF

Topic 00
  excellent (0.43)
  excellent_product (0.05)
  excellent_phone (0.02)
  excellent_condition (0.01)
  recommend (0.01)

Topic 01
  love (0.16)
  buy (0.05)
  nice (0.05)
  perfect (0.02)
  nice_phone (0.02)

Topic 02
  great (0.19)
  work (0.18)
  great_phone (0.07)
  great_product (0.02)
  great_price (0.01)

Topic 03
  new (0.02)
  come (0.01)
  get (0.01)
  happy (0.01)
  look (0.01)

Topic 04
  good (0.37)
  good_phone (0.05)
  good_product (0.02)
  good_price (0.01)
  good_condition (0.01)
Coherence score for NMF: -5.281283329510645
```

Final Remarks on NMF: The best performing model was the full-text NMF with a score of -3.9, which is the highest out of all possible variants.

1.1.9 Latent Dirichlet Allocation

LDA views each document as consisting of different topics. In other words, each document is a mix of different topics. In the same way, topics are mixed from words. To keep the number of topics per document low and to have only a few, important words constituting the topics, LDA initially uses a Dirichlet distribution, a so-called Dirichlet prior. This is applied both for assigning topics to documents and for finding words for the topics. The Dirichlet distribution ensures that documents have only a small number of topics and topics are mainly defined by a small number of words.

Note: Because C_V is far too slow as a coherence score generation mechanism, we use u_mass instead. For this type of coherence score, low negative values are associated with success.

1.1.10 Approach: Using LDA with Gensim - Full Text (Best Performance)

[67]:
```
lda_mod = get_model_gensim(df['tokens'],'LDA')
```

```
Topics for LDA

Topic 00
  phone (0.13)
  great (0.07)
  good (0.07)
  works (0.04)
  love (0.03)

Topic 01
  camera (0.02)
  screen (0.01)
  like (0.01)
  samsung (0.01)
  sony (0.01)

Topic 02
  phone (0.08)
  one (0.02)
  would (0.01)
  get (0.01)
  work (0.01)

Topic 03
  sim (0.06)
  card (0.05)
  unlocked (0.03)
  att (0.03)
  verizon (0.02)

Topic 04
```

```
  battery (0.07)
  life (0.02)
  charge (0.02)
  day (0.02)
  sound (0.01)
Coherence score for LDA: -2.2296787912602896
```

1.1.11 Approach: Using LDA with Gensim - Verbs/Adjs

[62]:
```
get_model_gensim(df['adjs_verbs'],'LDA')
```

```
Topics for LDA

Topic 00
  look (0.03)
  want (0.03)
  use (0.02)
  need (0.02)
  find (0.02)

Topic 01
  awesome (0.04)
  arrive (0.04)
  include (0.03)
  lte (0.02)
  original (0.02)

Topic 02
  good (0.30)
  nice (0.08)
  love (0.07)
  recommend (0.06)
  happy (0.05)

Topic 03
  great (0.33)
  work (0.28)
  expect (0.06)
  fast (0.06)
  fine (0.04)

Topic 04
  buy (0.06)
  get (0.05)
  come (0.04)
  new (0.04)
  excellent (0.03)
Coherence score for LDA: -3.494048601109698
```

1.1.12 Approach: Using LDA with Gensim - Nouns/Adjs

[63]:
```
get_model_gensim(df['adj_noun_phrases'],'LDA')
```

```
Topics for LDA

Topic 00
  great_phone (0.08)
  sim_card (0.04)
  nice_phone (0.03)
  unlocked_phone (0.02)
  old_phone (0.02)

Topic 01
  excellent_phone (0.02)
  dual_sim (0.02)
  great_price (0.02)
  good_price (0.01)
  long_time (0.01)

Topic 02
  new_phone (0.07)
  basic_phone (0.02)
  international_version (0.02)
  awesome_phone (0.02)
  good_condition (0.02)

Topic 03
  easy_use (0.03)
  android_phone (0.02)
  great_product (0.02)
  happy_phone (0.02)
  well_phone (0.01)

Topic 04
  good_phone (0.10)
  smart_phone (0.04)
  excellent_product (0.02)
  good_quality (0.01)
  happy_purchase (0.01)
Coherence score for LDA: -6.589716739801671
```

1.1.13 Visualization - LDA with Full text

[68]:
```
from sklearn.feature_extraction.text import CountVectorizer
count_para_vectorizer = CountVectorizer(
    stop_words=stopwords,
    min_df=5,
```

```
    max_df=0.7
)
count_para_vectors = count_para_vectorizer.fit_transform(df["text"])
```

[72]:
```
import pyLDAvis.sklearn
lda_mod = LatentDirichletAllocation(n_components = 5, random_state=42)
W_lda_para_matrix = lda_mod.fit_transform(count_para_vectors)
H_lda_para_matrix = lda_mod.components_

lda_display = pyLDAvis.sklearn.prepare(
    lda_mod,
    count_para_vectors,
    count_para_vectorizer,
    sort_topics=False
)
pyLDAvis.display(lda_display)
```

[72]: <IPython.core.display.HTML object>

Comparrison with Apriori Group Topic generation

[92]:
```
for rating in sorted(df['Rating'].unique()):
    this_rating = df.query(f"Rating=={rating}")
    print(f'TOPIC/RATING: {rating} N = {len(this_rating)}')
    get_model_gensim(this_rating['tokens'],'LDA',topics=1)
    print('-----------------\n\n')
```

```
TOPIC/RATING: 1 N = 71375
Topics for LDA

Topic 00
  phone (0.06)
  work (0.01)
  one (0.01)
  would (0.01)
  screen (0.01)
Coherence score for LDA: -1.8808731546856088
-----------------



TOPIC/RATING: 2 N = 24534
Topics for LDA

Topic 00
  phone (0.05)
  battery (0.01)
  one (0.01)
  screen (0.01)
```

```
  would (0.01)
Coherence score for LDA: -1.6476912049801582
-----------------


TOPIC/RATING: 3 N = 31384
Topics for LDA

Topic 00
  phone (0.05)
  good (0.01)
  like (0.01)
  one (0.01)
  battery (0.01)
Coherence score for LDA: -1.6983812754312848
-----------------


TOPIC/RATING: 4 N = 60204
Topics for LDA

Topic 00
  phone (0.05)
  good (0.02)
  great (0.01)
  like (0.01)
  use (0.01)
Coherence score for LDA: -1.6622381308614727
-----------------


TOPIC/RATING: 5 N = 218172
Topics for LDA

Topic 00
  phone (0.05)
  great (0.02)
  good (0.01)
  love (0.01)
  one (0.01)
Coherence score for LDA: -1.9176512099948588
-----------------
```

1.1.14 Closing Remarks

Overall, our LDA models seem to outperform our NMF models, all else held constant. The LDA that uses the full text has the strongest coherence score out of all candidates. With regards to the visualization from `pyLDAvis`, we observe that topics 1 and 3 have considerable overlap among one another, primarily because they both make strong references to actual features of the phones they review. There is a strong tapering-off for both topics, so they seem well defined.

Topic 2 is largely characterized by refrences to the word 'phone', which isn't particularly meaningful to us since all of these reviews are about phones, but separates itself by possibly containing reviews that mention phones but not their individual features as part of the review.

Topic 5 is poorly defined and small, with no meaningful insights.

Topic 4, however, **is driven by a meaningful combination of sentiment and product descriptions.** This topic is driven by one noun (phone), followed by several adjectives (good, great, excellent), which sends the impression that Topic 4 is comprised largely of positive reviews that don't get specific about technical features.

Remarks on Apriori Groups The general trend in apriori groups is that lower rating values (particular 1 and 2) tend to not include any positive adjectives, but also tend to excluse negative adjectives and simply discuss the product. Insofar as this is the case, Topics 5 and 2 correspond to negative reviews, while 1,3, and 4 correspond to positive reviews, but it's unclear to which respective ranking. We can speculate that Topic 4, which contains the strongest use of positive adjectives, may correspond to a rating of 5.

Notebook_4_Sentiment_analysis

June 26, 2022

0.1 Sentimental Analysis

In this section, we will explore two models to estimate the sentiment from a snippet of text data.

0.1.1 Packages

[1]:
```
import kaggle
import pandas as pd
import matplotlib.pyplot as plt
import pyarrow
import fastparquet
import numpy as np
import os
from collections import Counter, defaultdict
import warnings
import seaborn as sns
warnings.filterwarnings("ignore")
from wordcloud import WordCloud
#kaggle.api.authenticate()
import nltk
from string import punctuation
import textacy.preprocessing as tprep
from nltk.corpus import stopwords
from wordcloud import WordCloud, STOPWORDS
import spacy
nlp = spacy.load("en_core_web_sm")
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.linear_model import LinearRegression
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,␣
↪confusion_matrix,plot_confusion_matrix
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import cross_val_score
```

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
```

0.2 Step 1: Data Preparation

Loading Dataset for Modeling

[2]:
```
new_df=pd.read_parquet('prepared_text.parquet.gzip', engine='pyarrow')
new_df.head(3)
```

[2]:
```
   Rating                                            lemmas  \
0       5  [feel, lucky, find, use, phone, use, hard, pho…
1       4  [nice, phone, nice, grade, pantach, revue, cle…
2       5                                         [pleased]

                                          adjs_verbs  \
0  [feel, lucky, find, hard, upgrade, sell, like,…
1  [nice, nice, clean, easy, android, fantastic, …
2                                          [pleased]

                                               nouns  \
0  [phone, phone, line, son, year, thank, seller,…
1  [phone, grade, pantach, revue, set, set, phone…
2                                                 []

                   noun_phrases  \
0  [phone_line, thank_seller]
1               [grade_pantach]
2                            []

                                    adj_noun_phrases       entities  \
0  [hard_phone, hard_phone_line, old_one, recomme…             []
1  [nice_phone, nice_grade, nice_grade_pantach, c…  [android/GPE]
2                                                 []             []

                                              tokens  token_count  \
0  [feel, lucky, found, used, phone, us, used, ha…           38
1  [nice, phone, nice, grade, pantach, revue, cle…           24
2                                          [pleased]            1

                                         new_reviews
0  feel lucky found used phone us used hard phone…
1  nice phone nice grade pantach revue clean set …
2                                            pleased
```

Removing unecessary columns prior to modeling

[3]:
```python
new_df =new_df[[
    'Rating','new_reviews','tokens'
]]
```

[4]:
```python
new_df.info()
print('\n')
print(new_df.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 405669 entries, 0 to 405668
Data columns (total 3 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Rating       405669 non-null  int64
 1   new_reviews  405669 non-null  object
 2   tokens       405669 non-null  object
dtypes: int64(1), object(2)
memory usage: 9.3+ MB


(405669, 3)
```

Undersampling Majority class

[5]:
```python
# undersampling 5-star reviews and oversampling other reviews
five_stars = new_df[new_df['Rating'] == 5].sample(n=70000)
non_five = new_df[new_df['Rating'] != 5 ]
df_bal = pd.concat([five_stars,non_five],axis=0)
```

We annotated all reviews with a rating of 4 and 5 as positive and with ratings 1 and 2 as negative:

[6]:
```python
# Assigning a new [1,0] target class label based on the product rating
df_bal['sentiment'] = 0
df_bal.loc[df_bal['Rating'] > 3, 'sentiment'] = 1
df_bal.loc[df_bal['Rating'] < 3, 'sentiment'] = 0
```

0.3 Step 2: Train-Test Split on Balanced Dataset

[7]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(df_bal['new_reviews'],
                                                    df_bal['sentiment'],
                                                     test_size=0.2,
                                                     random_state=42,
                                                    ␣
 ↪stratify=df_bal['sentiment'])
print ('Size of Training Data ', X_train.shape[0])
print ('Size of Test Data ', X_test.shape[0])
```

```
print ('Distribution of classes in Training Data :')
print ('Positive Sentiment ', str(sum(Y_train == 1)/ len(Y_train) * 100.0))
print ('Negative Sentiment ', str(sum(Y_train == 0)/ len(Y_train) * 100.0))
print ('Distribution of classes in Testing Data :')
print ('Positive Sentiment ', str(sum(Y_test == 1)/ len(Y_test) * 100.0))
print ('Negative Sentiment ', str(sum(Y_test == 0)/ len(Y_test) * 100.0))
```

```
Size of Training Data  205997
Size of Test Data  51500
Distribution of classes in Training Data :
Positive Sentiment  50.56529949465284
Negative Sentiment  49.434700505347166
Distribution of classes in Testing Data :
Positive Sentiment  50.565048543689315
Negative Sentiment  49.43495145631068
```

0.4 Step 3: Text Vectorization

Using TF-IDF vectorization to create the vectorized representation:

[8]:
```
tfidf = TfidfVectorizer(min_df = 10, ngram_range=(1,1))
X_train_tf = tfidf.fit_transform(X_train)
X_test_tf = tfidf.transform(X_test)
```

0.5 Step 4: Training the Machine Learning Models

Model 1 : Linear SVC

[9]:
```
model_svc = LinearSVC(random_state=42, tol=1e-5)
model_svc.fit(X_train_tf, Y_train)
Y_pred = model_svc.predict(X_test_tf)
print ('Accuracy Score - ', accuracy_score(Y_test, Y_pred))
print ('ROC-AUC Score - ', roc_auc_score(Y_test, Y_pred))
```

```
Accuracy Score -  0.8829708737864078
ROC-AUC Score -  0.8829385881762057
```

As we can see, our model achieves an accuracy of around 88%. Now, let's take look at some of the model predictions and the review text to perform a sense check of the model:

[11]:
```
sample_reviews = df_bal.sample(10)
sample_reviews_tf = tfidf.transform(sample_reviews['new_reviews'])
sentiment_predictions_svc = model_svc.predict(sample_reviews_tf)
sentiment_predictions_svc = pd.DataFrame(data = sentiment_predictions_svc,
                                         index=sample_reviews.index,
                                         columns=['sentiment_prediction'])
sample_reviews_svc = pd.concat([sample_reviews, sentiment_predictions_svc],␣
 ↪axis=1)
print ('Some sample reviews with their sentiment - ')
```

```
sample_reviews_svc[['new_reviews','sentiment_prediction']]
```

```
Some sample reviews with their sentiment -
```

[11]:
```
                                                new_reviews  \
301994  love many great features everything need phone…
113284    ready garbage fault shoulda done research first
297230  like phonei like phone letters small text stil…
145296  100 cant go wrong note couple programs google …
211667                                                 good
122761  ive 4 12 months holding alright freeze reviews…
222442                              gift someone satisfied
126426                               good features quality
123560                                                 good
3726                                            works great

        sentiment_prediction
301994                     1
113284                     0
297230                     1
145296                     1
211667                     1
122761                     0
222442                     1
126426                     1
123560                     1
3726                       1
```

We can see that this model is able to predict the reviews reasonably well. For instance, review 113284 where the customer found the result to be garbage is marked as negative.

Model 2: Deep Neural Multi-layer Perceptron Classifier

[14]:
```
model_clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5,␣
 ↪2), random_state=1)
model_clf.fit(X_train_tf, Y_train)
Y_pred_clf= model_clf.predict(X_test_tf)
print ('Accuracy Score - ', accuracy_score(Y_test, Y_pred_clf))
print ('ROC-AUC Score - ', roc_auc_score(Y_test, Y_pred_clf))
```

```
Accuracy Score -  0.8933009708737865
ROC-AUC Score -  0.8930619524994998
```

[13]:
```
sentiment_predictions_clf = model_clf.predict(sample_reviews_tf)
sentiment_predictions_clf = pd.DataFrame(data = sentiment_predictions_clf,
                                         index=sample_reviews.index,
                                         columns=['sentiment_prediction'])
```

```
sample_reviews_clf = pd.concat([sample_reviews, sentiment_predictions_clf],␣
↪axis=1)
print ('Some sample reviews with their sentiment - ')
sample_reviews_clf[['new_reviews','sentiment_prediction']]
```

```
Some sample reviews with their sentiment -
```

[13]:

```
                                                 new_reviews  \
301994  love many great features everything need phone…
113284    ready garbage fault shoulda done research first
297230  like phonei like phone letters small text stil…
145296  100 cant go wrong note couple programs google …
211667                                                good
122761  ive 4 12 months holding alright freeze reviews…
222442                             gift someone satisfied
126426                              good features quality
123560                                                good
3726                                          works great

        sentiment_prediction
301994                     1
113284                     0
297230                     1
145296                     1
211667                     1
122761                     1
222442                     1
126426                     1
123560                     1
3726                       1
```

This model achieved accuracy around 89% . However, when we look at the same sample reviews, we can see reveiw 122761 where the customer talks about freezing reviews, the model predicted as positive while it looks more negative.

[ ]:

```

```