Analysis of Amazon Alexa Reviews

INTRODUCTION

There is abundance of reviews from users available on the internet from social media websites such as Twitter, Facebook or blog posts. Some of the reviews available on sites such as yelp provide a rating which that lets a user know about the general opinion conveyed by the review. But there are so many other websites that only provide the text reviews. The inspiration behind our project is simple, what if it is possible to detect the general sentiment of a review solely based on the text. By using sentiment analysis one can determine how customers feel about a product without having to read thousands of customer reviews. The project talks in detail about the methods and data used to build a machine learning model to predict whether a particular review is positive or negative based on the text.

Adithya – This is my first time working on a full-scale machine learning project and I wanted the topic to be about something that has a practical value. I feel sentiment analysis is more relevant than ever since in the current climate it has become difficult to talk and share opinions face to face and more people have resorted to writing opinions over the internet. As such there is an abundance of data that can be leveraged to understand their sentiments, with the help of machine learning software, we can wade through all that data in minutes, to analyze individual emotions and overall public sentiment on every social platform.

Sandeep - As everything has moved largely online be it shopping, groceries etc. as there is no physical activity because in some places there is lockdown due to the pandemic. So, most of the time that people are spending at home and shopping online has increased which is different from shopping at a physical store as you cannot test the product so we would like to analyze the reviews of the product and try to do analysis on the data by using machine learning so that we can predict the quality of the product from the reviews.

Aakash - Huge amount of data is now being used by the corporate industry to analyze and predict customer behavior. People nowadays take review of brands to the online forum which can either be positive or negative. Companies can use this data to get feedback on what they are doing good and where they need to improve. Analyzing the sentiment of the statement has gained a lot importance and is an essential part of the classification. Learning about the sentiment not only helps the people in corporate but it can also be used to improve city infrastructure like health facilities, area blocks etc. Learning about the model working will help me gain insight in the world of retrieval and classification.

DATASET

The model uses the Amazon Alexa reviews dataset for the analysis. This dataset consists of a nearly 3000 Amazon customer reviews (input text), star ratings, date of review, variant and feedback of various amazon Alexa products like Alexa Echo, Alexa Firesticks etc. We will be focusing mainly text for our models as it is the most important feature as it provides us the sentiment behind the review.

rating	date	variation	verified_reviews	feedback
5	31-Jul-18	Charcoal Fabric	Love my Echo!	1
5	31-Jul-18	Charcoal Fabric	Loved it!	1

PROJECT METHODOLOGY

The first step is to perform pre-processing steps on the data to prepare it for training.

1. Data Cleaning

We check for any spelling errors in the columns and if it's there, then clearing them would be the first task as these column names should be meaningful.

We then check for any null values in our dataset. In the dataset we have there were no null values, so we now free to proceed to the next step.

2. Text Preprocessing

The languages we speak and write are made up of several words that are often derived from other words. For example, the word "Working" is derived from the root word work.

a. Removing Special Characters & Stop Words

The first part of text pre-processing is to remove all the special characters that doesn't add any sentiment to a text. The characters may include commas, emojis, full stops etc.

For the purpose of removing special characters, we import the regular expression library provided by python.

Using the library, we then remove all the additional white spaces, alphanumeric symbols and other special characters. A sample code is show below.

```
#Remove additional white spaces

   review= re.sub('[\s]+', ' ', tweet)

   review = re.sub('[\n]+', ' ', tweet)
```

```
#Remove not alphanumeric symbols white spaces
   review= re.sub(r'[^\w]', ' ', tweet)

#Removes hastag in front of a word """
    review = re.sub(r'#([^\s]+)', r'\1', tweet)
```

b. Word Lemmatization

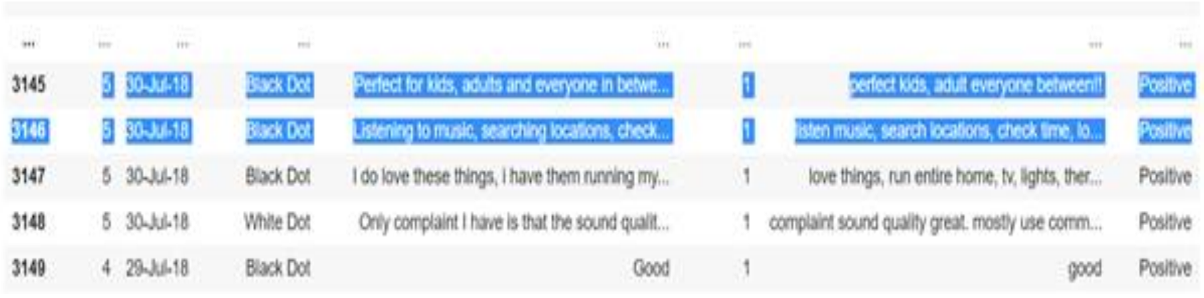
Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization brings context to the words. So, it links words with similar meaning to one word. Lemmatization, unlike Stemming, reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called Lemma and the output of lemmatization is a proper word.

3. Assigning Sentiment

Now that we are only left with the relevant words, our next task is to classify each word as positive, negative or neutral. We assign a sentiment score of +1 for positive, -1 for negative and 0 for neutral.

For the purpose of assigning sentiments, we use the textblob library provided by NLTK.

The below picture shows the data after preprocessing and sentiment assignement.



...	...	...	...	...	...	...	...
3145	5	30-Jul-18	Black Dot	Perfect for kids, adults and everyone in betwe...	1	perfect kids, adult everyone between!!	Positive
3146	5	30-Jul-18	Black Dot	Listening to music, searching locations, check...	1	listen music, search locations, check time, lo...	Positive
3147	5	30-Jul-18	Black Dot	I do love these things, i have them running my...	1	love things, run entire home, tv, lights, ther...	Positive
3148	5	30-Jul-18	White Dot	Only complaint I have is that the sound qualit...	1	complaint sound quality great. mostly use comm...	Positive
3149	4	29-Jul-18	Black Dot	Good	1	good	Positive

4. TF - IDF - Term Frequency Inverse Document Frequency

Basically, to transform text into meaningful representation of numbers which can be used by the models to predict.

It is a measure of originality of a word bby comparing the number of times a word appears in a doc with the number of docs the word appears in

TF-IDF = TF(t,d) * IDF(t)

TF(t,d) -> term frequency -> number of times term t appears in a doc

IDF(t) -> Inverse document frequency

$$\mathrm{idf}(t) = \log \frac{|D|}{1 + |\{d : t \in d\}|}$$

where $|\{d : t \in d\}|$ is the **number of documents** where the term t appears, when the term-frequency function satisfies $\mathrm{tf}(t, d) \neq 0$, we're only adding 1 into the formula to avoid zero-division.

Uses: Document Classification, Topic Modeling, Info retrieval and Stop word filtering

The rationale behind TF-IDF is the following:

- a word that frequently appears in a document has more relevancy for that document, meaning that there is higher probability that the document is about or in relation to that specific word
- a word that frequently appears in more documents may prevent us from finding the right document in a collection; the word is relevant either for all documents or for none. Either way, it will not help us filter out a single document or a small subset of documents from the whole set.

So, then TF-IDF is a score which is applied to every word in every document in our dataset. And for every word, the TF-IDF value increases with every appearance of the word in a document, but is gradually decreased with every appearance in other documents.

TFID example:

Sentence 1 -> echo good

Sentence 2 -> echo useful good

Sentence 3 -> echo bad

Histogram:

Words	**Frequency**
Good	1
Useful	1
Bad	1
Echo	3

TF-IDF

Term Frequency: No of rep of words in a sentence / no of ALL words in a sentence

	S1	**S2**	**S3**
Good	½	½	0
Useful	0	½	0
Bad	0	0	½
Echo	½	½	½

Inverse Document Frequency: log(no of sentences/ no of sentences containing the word)

Words	**IDF**
Good	Log(3/2)= 0.1761
Useful	Log(3/1) = 0.477
Bad	Log(3/1) = 0.477
Echo	Log(3/3) = 0

	F1 Good	**F2 Useful**	**F3 Bad**	**F4 Echo**
S1	½*0.1761 = 0.088	0	0	0
S2	½*0.477 = 0.2385	0.2385	0	0
S3	0	0	0.2385	0

By evaluating TF-IDF or a number of *"the words used in a sentence vs words used in overall document",* we understand -

1. how useful a word is to a sentence (which helps us understand the importance of a word in a sentence).
2. how useful a word is to a document (which helps us understand the important words with more frequencies in a document).

3. helps us ignore words that are misspelled (using n-gram technique) an example of which I am covering below

What we did to improve TF – IDF

sublinear_tf is set to true, which switches tf with 1 + log(tf). The idea is that when a query term occurs 20 times in doc. a and 1 time in doc. b, doc. a should (probably) not be considered 20 times as important but more likely log(20) times as important.

Additionally, stop_words is set to ‘english’, so stop words such as “and”, “the”, “him” will be ignored in this case, and max_df=0.5 means that we are ignoring terms that have a document frequency higher than 0.5 (i.e., the proportion of documents in which the term is found).

Next, we fit and transform the features (terms or words in our case) for both train and test sets. Notice that for the train set we use fit_transform, and for the test set we use just transform.

Latent Semantic Analysis is better than TFID as TFID doesn’t capture correlation between words like BOW

4. Model Training:

We use the following models to predict the sentiment of each review and determine the accuracy of each model.

1. **Naïve Based Model**

It is a classification technique based on Bayes Theorem with an assumption of independence among predictors or features. So, basically features are independent of each other and the sentiment of words together are not considered here. For example, take a fruit Orange, it is round, Orange and about 2 inches of radius. Although these properties in real life depend on each other, the ‘Naïve’ algorithm consider them as independent.

We know Bayes Theorem as

$$p(A|B) = \frac{\mathbf{p(A).p(B|A)}}{\mathbf{p(B)}} \qquad \mathbf{(Bayes' \; Theorem)}$$

In case of Naïve, a supervised learning, algorithm, given bunch of X features, calculates probability of a class.

Multinomial Naive Bayes:

$$\mathbf{p(yi \mid x1, x2, \ldots, xn) = \frac{p(x1, x2, \ldots, xn \mid yi).p(yi)}{p(x1, x2, \ldots, xn)}}$$

P(x1, x2…|yi) means probability of specific combination of feature given a class(label). In ideal case, we need large dataset to calculate on probability distribution of all different combination of features. To overcome this, Naïve Bayes algorithm considers all features independent of each other. So, denominator can be removed.

The basic formula becomes, probability of a data belonging to a class depends on number of observations with class yi divided by total number of observations.

$$\mathbf{p(x1, x2, \ldots, xn \mid \mathit{yi}) = \mathit{p(x1 \mid yi) \,.\, p(x2 \mid yi). \ldots . p(xn \mid yi)}}$$

In case of our example, we need bunch of features, which NB usually calculates using BOW. Using TF-IDF instead of normal vector improves the result as TF-IDF is less susceptible to noise and assign weightage to each word while considering all documents.

Example:

	F1 Good	**F2 Useful**	**F3 Bad**	**F4 Echo**	**Sentiment**
S1	1	0	0	0	Positive
S2	1	1	0	0	Positive
S3	0	0	1	0	Negative

P(c = Positive | x1, x2….xn) and P(c = Negative | x1, x2….xn)

P(c = Positive | x1, x2….xn) = P(positive) * P(x1 | positive) * P(x2 |positive)

P(Positive) -> how many records have sentiment positive / total number of records. 2/3 for above case.

P(x1| positive) -> probability of a feature being positive given the statement is positive.

S1 -> 2/3 * ½ = 0.33

S2 -> 2/3 * 2/2 = 0.66

S3 -> Negative -> 2/3 * 1/3 = 0.2178

P(x1|positive) = word_count_in_class / total_words_in_class

word_count_in_class : sum of(tf-idf_weights of the word for all the documents belonging to that class) //basically replacing the counts with the tfidf weights of the same word calculated for every document within that class.

total_words_in_class : sum of (tf-idf weights of all the words belonging to that class)

Pros:

Easy to predict.

Perform better than others and needs less training data

Works well with discrete values i.e., categorical input variable.

Cons:

Unknown word gets assigned 0 probability so unable to use it. Laplace Smoothing come into the play here.

A bad estimator since it considers each feature separately.

2. Logistic Regression:

It is a classification algorithm, used when the data is categorical in nature I.e. it can be divided into class of two 0 or 1. In reference to our case, either positive or negative sentiment.

The Sigmoid Function

The sigmoid function/logistic function is a function that resembles an “S” shaped curve when plotted on a graph. It takes values between 0 and 1 and “squishes” them towards the margins at the top and bottom, labeling them as 0 or 1.

The equation for the Sigmoid function is this:

$$y = 1/(1 + e^{-x})$$

Since the classification is only done as negative or positive, we are using logistic regression as one of our mode. Here. if x is large negative value, y is closer to 0 and 1 if x is large positive value.

Pros.

Takes Less Time as compared to other models

Since logistic classifier is used for data which can be classified as 1 and 0, it will be able to classifying with higher accuracy.

Cons:

Can be used for discrete values – separating data into yes/no category,

3. Support Vector Machines:

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane.

svm.LinearSVC() from sklearn package can be used for Linear SVM classifier.

4. Polynomial Support Vector Machines:

The function of kernel is to take data as input and transform it into the required form. We add another dimension and maybe the data can be separated.

$$k(x_i, x_j) = (1 + x_i * x_j)^d$$

Here K(xi,xj) is the kernel function and d is the degree of polynomial.

svm.SVC(kernel='poly') from sklearn package is sued for polynomial SVM classifier.

5. **Decision Trees:**

Decision tree is a decision support tool that uses a graphical or tree model of decisions and their possible consequences, including the results of random events, resource costs, and utility.

```
# Decision tree classifier

entropy= DecisionTreeClassifier(criterion='entropy',random_state = 0)
```

The above code is using entropy as a criterion to build a classification tree.

6. Topic Modelling:

Topic Modeling is a process to automatically identify topics present in a text object and to derive hidden patterns exhibited by a text corpus. We are doing this by using the concept of Latent Dirichlet Allocation (LDA).

LDA is a matrix factorization technique. In vector space, any corpus (collection of documents) can be represented as a document-term matrix. The following matrix shows a corpus of N documents D1, D2, D3 … Dn and vocabulary size of M words W1, W2 ... Wn. The value of i,j cell gives the frequency count of word Wj in Document Di.

	W1	W2	W3	W4
D1	0	2	1	3
D2	1	4	0	0
D3	0	2	3	1
D4	1	1	3	0

The below code summarizes the creation of the document words matrix.

```
tokenisedreviews=[d.split() for d in reviewData['cleaned_verified_reviews']]

dictionary = corpora.Dictionary(tokenisedreviews)

doc_term_matrix = [dictionary.doc2bow(rev) for rev in tokenisedreviews]
```

LDA converts this document matrix into two lower dimension matrices document-topics matrix and words topic matrix respectively

	K1	K2	K3	K4
D1	1	0	0	1
D2	1	1	0	0
D3	1	0	0	1
D4	1	0	1	0

Document topic matrix

	W1	W2	W3	W4
K1	0	1	1	1
K2	1	1	1	0
K3	1	0	0	1
K4	1	1	0	0

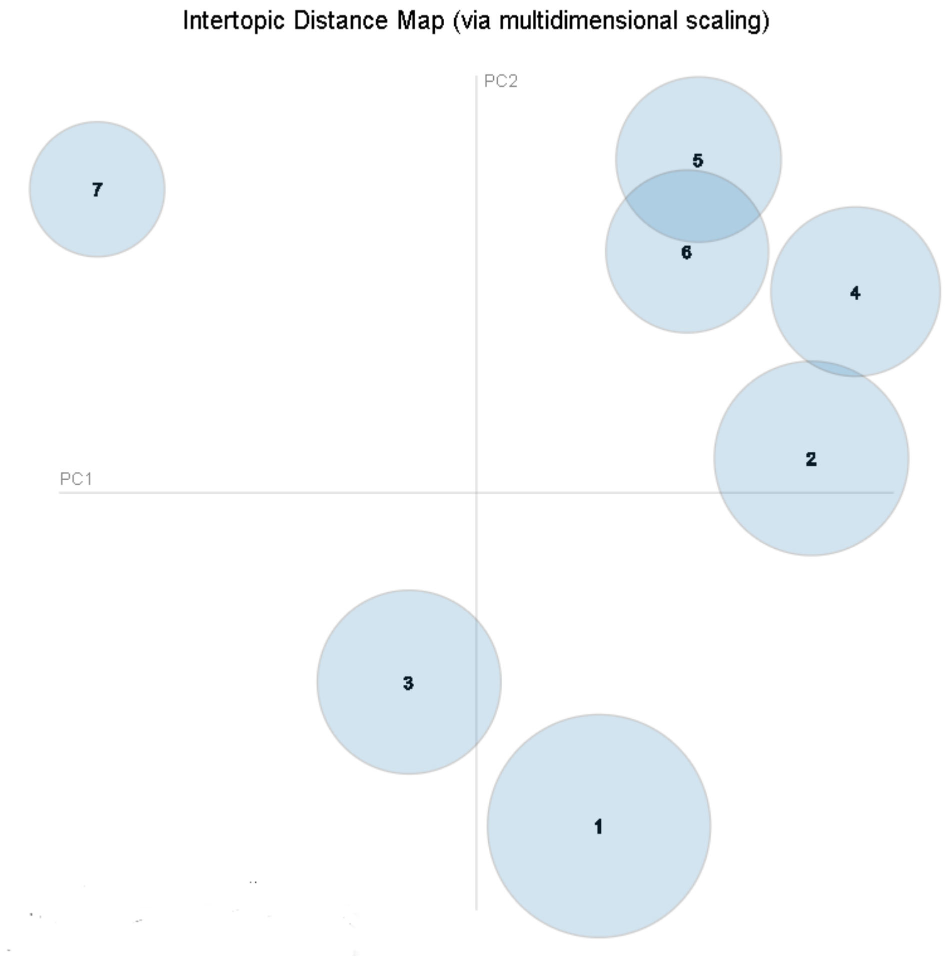
Words Topic matrix

LDA uses sampling techniques to improve these matrices. It Iterates through each word "w" for each document "d" and tries to adjust the current topic – word assignment with a new assignment. A new topic "k" is assigned to word "w" with a probability P which is a product of two probabilities p1 and p2. It does this until a steady state is achieved where the document topic and topic term distributions are fairly good. The below code summarizes the creation of LDA model and of 7 topics which are passes in num_topics parameter

p(word w with topic k) = p(topic k | document d) * p(word w | topic k)

```
# Creating the object for LDA model using gensim library

LDA = gensim.models.ldamodel.LdaModel

# Build LDA model

lda_model = LDA (corpus=doc_term_matrix, id2word=dictionary, num_topics=7, random_state=100,

            chunksize=1000, passes=50)

lda_model.print_topics()
```

```
[(0, '0.063*"easy" + 0.053*"set" + 0.030*"use" + 0.017*"setup" + 0.013*"product" + 0.012*"smart" +
0.011*"tv" + 0.011*"perfect" + 0.010*"connect" + 0.009*"awesome"'), (1, '0.033*"alexa" +
0.028*"music" + 0.021*"light" + 0.017*"love" + 0.016*"weather" + 0.014*"play" + 0.013*"time" +
0.013*"smart" + 0.012*"like" + 0.012*"home"'), (2, '0.047*"echo" + 0.039*"sound" + 0.035*"speaker" +
0.027*"dot" + 0.026*"great" + 0.020*"quality" + 0.020*"music" + 0.018*"better" + 0.017*"good" +
0.017*"work"'), (3, '0.029*"device" + 0.020*"home" + 0.018*"music" + 0.017*"need" + 0.015*"amazon"
+ 0.015*"would" + 0.015*"one" + 0.015*"bought" + 0.013*"play" + 0.012*"google"'), (4, '0.028*"love" +
0.028*"alexa" + 0.019*"thing" + 0.018*"use" + 0.017*"amazon" + 0.016*"clock" + 0.015*"one" +
0.014*"day" + 0.014*"much" + 0.013*"prime"'), (5, '0.043*"like" + 0.025*"alexa" + 0.023*"time" +
0.022*"still" + 0.019*"use" + 0.016*"fun" + 0.015*"learning" + 0.014*"thing" + 0.014*"house" +
0.013*"love"'), (6, '0.087*"love" + 0.073*"great" + 0.050*"work" + 0.037*"echo" + 0.029*"product" +
0.024*"easy" + 0.024*"use" + 0.020*"dot" + 0.018*"set" + 0.014*"music"')]
```



The above image shows the intertopic distance map.

Some of its features are:

- A good topic will have fairly big non overlapping models scattered throughout the chart instead of a single quadrant.
- A model with too many topics will have may overlaps of the bubbles in one region of the chart.

CONCLUSION

In summary, we used our text review as a feature to predict the sentiment of the reviews. We tried many classification algorithms including Naïve Bayes, LDA, Logistics regression, Decision trees and SVM. From the results we can see our accuracy is best when we use SVM and decision tree. One of the reasons for the high accuracy of the models is that the data consist of a much higher number of positive reviews as compared to negative reviews.

Models	**Accuracy**
Naïve Bayes	86
Logistics Regression	90
SVM	94
Decision Trees	93

Using TF-IDF does improves the consideration of weight of a word which helps in improving our accuracy and precision but it does so while considering each feature to be independent of each other. This makes it difficult to detect intent of a statement. It cannot capture the semantics.

One of the main reasons our accuracy is not high enough is because of the data imbalance. The dataset used as a much higher number of positive reviews than negative. A possible solution we have not tried is to find more data points from other sources. We think that might help solve the problem.

Adithya – Working on the project helped me understand new concepts based on Natural Language processing as well as learn more about different classification models such as Naïve bayes and LDA. I believe working on this project has given me a good foundation on machine learning topics and helped me prepare for courses that I plan to take in the future.

Sandeep - This project has helped me understand machine learning algorithms and little bit of understanding of natural language processing. It has also given me a good foundation into NLP topics which will be helpful while taking NLP course in the future.

Aakash – Learning about ML model Algorithm has helped me gain more insight in the field of ML and Information Retrieval. By working on this project, I was exposed to new pre-processing techniques and how different types of models can used to overcome the issue faced in the basic classifiers as improved my existing knowledge of ML and it is providing a good base for the topics I would find in relevant courses.

References :

https://medium.com/analytics-vidhya/tf-idf-term-frequency-technique-easiest-explanation-for-text-classification-in-nlp-with-code-8ca3912e58c3

https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

https://towardsdatascience.com/naive-bayes-classifier-explained-50f9723571ed

THEORY AND APPLICATIONS OF NATURAL LANGUAGE PROCESSING by JI YEON (JAE) IHN

Appendix

Python code

```
!pip install textblob

!python -m textblob.download_corpora

import gensim

from gensim import corpora

!pip install pyLDAvis

import pyLDAvis

import pyLDAvis.gensim

# Cleaning and preprocessing

def processRow(row):

  import re

  import nltk

  from textblob import TextBlob

  from nltk.corpus import stopwords

  from nltk.stem import PorterStemmer
  from textblob import Word

  from nltk.util import ngrams

  import re

  from nltk.tokenize import word_tokenize

  from nltk.stem import WordNetLemmatizer

  review = row



  #Removes unicode strings like "\u002c"  -> ,(comma)

  review = re.sub(r'(\\u[0-9A-Fa-f]+)',r'', review)


  # Removes non-ascii characters. note : \x00 to \x7f is 00 to 255

  # non-ascii characters like copyrigth symbol, trademark symbol

  review = re.sub(r'[^\x00-\x7f]',r'',review)


  #convert any url to URL

  review = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL',review)


  #Convert any @Username to "AT_USER"

  review = re.sub('@[^\s]+','AT_USER',review)


  #Remove additional white spaces

  review = re.sub('[\s]+', ' ', review)

  review = re.sub('[\n]+', ' ', review)


  #Remove not alphanumeric symbols white spaces

  review = re.sub(r'[^\w]', ' ', review)


  #Removes hastag in front of a word """
```

```
  review = re.sub(r'#([^\s]+)', r'\1', review)

  #Replace #word with word
  review = re.sub(r'#([^\s]+)', r'\1', review)

  #Removes all possible emoticons
  review = re.sub(':\)|:\(|:\)|;\)|:-\)|\(-:|:-D|=D|:P|xD|X-p|\^\^|:-*|\^\.\^|\^\-\^|\^\_\^|\,-\)|\)-
:|:\'\(|:\(|:-\(|:\S|T\.T|\.\_\.|:<|:-\S|:-<|\*\-\*|:O|=O|=\-O|O\.o|XO|O\_O|:-\@|=/|:/|X\-
\(|>\.<|>=\(|D:', '', review)

  #remove numbers -> this is optional
  review = ''.join([i for i in review if not i.isdigit()])

  #remove multiple exclamation -> this is optional
  review = re.sub(r"(\!)\1+", ' ', review)

  #remove multiple question marks -> this is optional
  review = re.sub(r"(\?)\1+", ' ', review)

  #remove multistop -> this is optional
  review = re.sub(r"(\.)\1+", ' ', review)

  #trim
  review = review.strip('\'"')
  print(review)


  row = review
  return row
```

```
from google.colab import files

#files.upload()

import pandas as pd


reviewData = pd.read_csv("sample_data/amazon_alexa.csv", sep='\t')


reviewData.columns


reviewData.loc[(reviewData['rating']==3)]


# checking if there any null values in the dataset


reviewData.isnull().any().any()


import numpy as np

# clean your verified_reviews

cleaned_verified_reviews = []


for line in reviewData['verified_reviews'] :

    cleanLine = processRow(line)

    cleaned_verified_reviews.append(cleanLine)




reviewData['cleaned_verified_reviews'] = np.asarray(cleaned_verified_reviews)


reviewData


#pre processing steps like lower case and lemmatization
```

```
import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

from textblob import Word

reviewData['cleaned_verified_reviews'] = reviewData['cleaned_verified_reviews'].apply(lambda x: "
".join(x.lower() for x in x.split()))


stop = stopwords.words('english')

total_stop = stop

reviewData['cleaned_verified_reviews'] = reviewData['cleaned_verified_reviews'].apply(lambda x: "
".join(x for x in x.split() if x not in total_stop))


reviewData['cleaned_verified_reviews'] = reviewData['cleaned_verified_reviews'].apply(lambda x: "
".join([Word(word).lemmatize(pos="v") for word in x.split()]))

#reviewData['cleaned_verified_reviews'] = reviewData['cleaned_verified_reviews'].apply(lambda x:"
".join([Word(word).stem() for word in x.split()]))

reviewData['cleaned_verified_reviews'].head()


reviewData


# Let's define our sentiment analyzer function:

from textblob import TextBlob

def analyze_sentiment(cleaned_verified_reviews):

    analysis = TextBlob(cleaned_verified_reviews)

    if analysis.sentiment.polarity < 0:

        return 'Negative'

    #elif analysis.sentiment.polarity == 0:

        #return 'Neutral'

    else:

        return 'Positive'
```

```
# Lets find the Sentiment by calling the above defn fn.

# create a new column called 'Sentiment'

reviewData['Sentiment'] = reviewData['cleaned_verified_reviews'].apply(lambda x:
analyze_sentiment(x))


# reviewData[(reviewData['Sentiment']=='Negative') & (reviewData['feedback']==1)]

reviewData[['cleaned_verified_reviews','feedback','Sentiment']].sample(3)


#reviewData


# lets add polarity also using textblob

#reviewData['polarity'] = reviewData['cleaned_verified_reviews'].map(lambda text:
TextBlob(text).sentiment.polarity)


#reviewData[['cleaned_verified_reviews', 'Sentiment', 'polarity']]


negative = reviewData.loc[reviewData['Sentiment'] == 'Negative']

positive = reviewData.loc[reviewData['Sentiment'] == 'Positive']


print("Negative Words")

from wordcloud import WordCloud, STOPWORDS

import matplotlib.pyplot as plt

text = negative['verified_reviews'].values

wordcloud = WordCloud(

width=3000,

height=2000,

background_color = 'black',

stopwords = STOPWORDS).generate(str(text))
```

```
fig = plt.figure(
figsize=(10,10),
facecolor = 'k',
edgecolor = 'k')
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()

negative = reviewData.loc[reviewData['Sentiment'] == 'Negative']
positive = reviewData.loc[reviewData['Sentiment'] == 'Positive']

print("Positive Words")
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
text = positive['verified_reviews'].values
wordcloud = WordCloud(
width=3000,
height=2000,
background_color = 'black',
stopwords = STOPWORDS).generate(str(text))
fig = plt.figure(
figsize=(10,10),
facecolor = 'k',
edgecolor = 'k')
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

```
reviewData[(reviewData['Sentiment']=='Positive')]


reviewData.loc[0]


from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer

import sklearn.feature_extraction.text as text

from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, svm

# Splitting data into train and validation

train_x, valid_x, train_y, valid_y =
model_selection.train_test_split(reviewData['cleaned_verified_reviews'], reviewData['Sentiment'],
test_size=0.25, random_state=1)

# TFIDF feature generation for a maximum of around 5000 features

encoder = preprocessing.LabelEncoder()

train_y = encoder.fit_transform(train_y)

valid_y = encoder.fit_transform(valid_y)

#print(encoder.get_feature_names())

print(train_y)

print(valid_y)


tfidf_vect = TfidfVectorizer(sublinear_tf=True, max_df=0.5, stop_words='english', analyzer='word',
token_pattern=r'\w{1,}')


xtrain_tfidf = tfidf_vect.fit_transform(train_x)

xvalid_tfidf = tfidf_vect.transform(valid_x)

print(tfidf_vect.get_feature_names())

features = tfidf_vect.get_feature_names()

print(xtrain_tfidf)

print(xvalid_tfidf)

pd.DataFrame(data = xtrain_tfidf.toarray(), columns = features).iloc[:, 0::2]
```

```
# Model training
# we have defined a generalized function for training any given model:
from sklearn.metrics import precision_score

def train_model(classifier, feature_vector_train, label,feature_vector_valid, is_neural_net=False):
# fit the training dataset on the classifier
classifier.fit(feature_vector_train, label)
# predict the labels on validation dataset
predictions = classifier.predict(feature_vector_valid)
precision=precision_score(valid_y,predictions,average='weighted')
print("Precision score : ", precision)
return metrics.accuracy_score(predictions, valid_y)
# NaiveBayes Classifier
accuracy = train_model(naive_bayes.MultinomialNB(alpha=0.001),xtrain_tfidf, train_y, xvalid_tfidf)
print("Accuracy: ", accuracy)

# Linear Classifier on Word Level TF IDF Vectors
accuracy = train_model(linear_model.LogisticRegression(),xtrain_tfidf, train_y, xvalid_tfidf)
print ("Accuracy: ", accuracy)

# SVM classifier
from sklearn import svm
accuracy = train_model(svm.LinearSVC(),xtrain_tfidf, train_y, xvalid_tfidf)
print ("Accuracy: ", accuracy)

# SVM classifier with Poly filter
accuracy = train_model(svm.SVC(kernel='poly'),xtrain_tfidf, train_y, xvalid_tfidf)
print ("Accuracy: ", accuracy)
```

```
# Decision tree classifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.externals.six import StringIO

from IPython.display import Image

from sklearn.tree import export_graphviz

import pydotplus

entropy = DecisionTreeClassifier(criterion='entropy',random_state = 0)

accuracy = train_model(entropy,xtrain_tfidf, train_y, xvalid_tfidf)

print ("Accuracy: ", accuracy)

dot_data = StringIO()

export_graphviz(entropy, out_file=dot_data,

                filled=True, rounded=True,

                special_characters=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

#Image(graph.create_png())


# Commented out IPython magic to ensure Python compatibility.

#frequency graph

from nltk import FreqDist

import matplotlib.pyplot as plt

import seaborn as sns

# %matplotlib inline

all_words = ' '.join([text for text in reviewData['cleaned_verified_reviews']])

all_words = all_words.split()

fdist = FreqDist(all_words)

words_df = pd.DataFrame({'word':list(fdist.keys()), 'count':list(fdist.values())})

d = words_df.nlargest(columns="count", n = 20)

plt.figure(figsize=(20,5))
```

```
ax = sns.barplot(data=d, x= "word", y = "count")

ax.set(ylabel = 'Count')

plt.show()


tokenisedreviews=[d.split() for d in reviewData['cleaned_verified_reviews']]

dictionary = corpora.Dictionary(tokenisedreviews)

doc_term_matrix = [dictionary.doc2bow(rev) for rev in tokenisedreviews]

# Creating the object for LDA model using gensim library

LDA = gensim.models.ldamodel.LdaModel


# Build LDA model

lda_model = LDA(corpus=doc_term_matrix, id2word=dictionary, num_topics=7, random_state=100,

          chunksize=1000, passes=50)


lda_model.print_topics()
```