Review Text Analysis on Steam Online Gaming Platform Data using Naive-Bayes, SVM and kNN

Student ID: 20201127

Kaushik Srinivas

Contents

Chapter 1

Introduction

These days every organization is keen on getting feedback from customers. Later, this data can be used for extracting some useful insights about the product developed, business plan, and so on. Manual extraction of such information is almost infeasible considering the amount of data extracted. Therefore, the review text analysis process uses machine learning algorithms to classify the feedback retrieved from each customer is either positive/1 or negative/0 based on some contextual parameters. The principle goal here is to rightly classify the review text as much as possible. The report consists of three main parts. The first one was text data modeling where the encoding issue was addressed, the second one was text data preprocessing where the review text was preprocessed to better suit the final implementation part [i.e. machine learning model implementation]. Various ML models like Naïve-Bayes, Support Vector Machine [SGD], and kNearestNeighbors were implemented. Later, I compared all three techniques with respect to an accuracy level of detecting the classification. I found that Naïve-Bayes performs better than SVM and kNearestNeighbors for the parameter whether or not to recommend the game and all three performs similarly for determining whether the review was for beta version or not.

Objective

The main objective of this project is to determine whether machine learning models can be used to determine whether the review posted on to Steam Gaming platform website suggest

- The game developed by this organization can be recommended or not.
- The review posted was for the beta version or the original release version of the game.

Related Work

This section gives a brief description of various concepts considered in solving the problem. First, ongoing through the raw web scrapped given data content, I noticed some strange characters which are impossible to read. On further analysis, I understood that these characters are related to language/symbols that cannot be represented in traditional ASCII format. For example, the Chinese language contains thousands of characters that are impossible to encode within 8 bits. To solve this, I made use of encode/decode function in python. Another major hurdle in-text modeling part was the presence of multiple languages. Two methods are considered to solve this issue. One was by making use of the 'polyglot' package. This package detects the type of language present and implements the corresponding language-specific text preprocessing like stop word removal, tokenization, and so on. The other one was the 'google_trans_new' API which converts text in another language to English. I implemented google translator API to remove language ambiguity.

In-text preprocessing part, the customization of text can be done using either the nltk library or textblob package. Both have their advantage and disadvantage. Therefore, I made use of both these package based on the task. In the implementation part, I did some research to figure out algorithms based on a probabilistic approach. I made use of one such algorithm named Multinomial Naïve-

Bayes. And other two are based on the geometric/kernel approach, namely SVM based on SGD and kNearestNeighbors.

Readers Guide

Chapter 2 gives in a detailed explanation of how raw data extracted from web was cleaned and converted to text containing only English words. Chapter 3 talks about various text preprocessing approaches and feature selection criteria followed. Chapter 4, the methods followed to select various hyper-parameters for each machine learning model implemented. Chapter 5 provides a graphical representation of the results obtained. Chapters 6 and 7 give information about the conclusion made and answers to the question asked in the assignment.

Chapter 2

Approach to Text Data Modelling

Overview

The given raw dataset consist of certain features that have to be removed/converted before sending these data to text preprocessing. On close observation, I found the review text field consist of following error features,

1. Poorly/wrongly encoded text format.
2. Certain text are in the binary format.
3. Review text also consist of smiley or other pictorial representations.
4. Reviews are also written in multiple languages.

This section gives in-depth explanation of how to solve the above issues. Before moving on to the next part consider the below table containing packages used in this chapter.

Package	**Why it is used?**
1. chardet	Used to determine the type of text encoding
2. json_lines	To read raw data in the json format.
3. pandas	Mainly used for data frame creation.
4. numpy	Used for array creation. [Mainly]
5. emoji	This package contains the Unicode of all the emoji created till now.
6. re	Regular expression [Regex].
7. google_trans_new	Execute Google translator API calls.

Import the Dataset

Code:

```
x = []
y = []
z = []
with open(r'C:\Users\gowtham\finale_Exam.php', 'rb') as f:
    for items in json_lines.reader(f):
        x.append(items['text'])  # Review text
        y.append(items['voted_up']) # True when game recommended by the user
        z.append(items['early_access']) #True when the review is for beta version.
```

Explanation:

Here, open function was used to open the finale_Exam.php file containing raw data. The file was open in raw bytes format and json_lines.reader() was used to extract text, voted_up and early_access values into three arrays x, y and z respectively.

Encoding/Decoding

First, it is necessary to determine the type of encoding done on data while scrapping from the website. For that we make use of chardet function.

Code:

```
with open(u'C:\\Users\\gowtham\\finale_Exam.php', 'rb') as detect_file_encoding:
    detection = chardet.detect(detect_file_encoding.read())# Find the encoding type of php file.
    print('Chardet:', detection)
```

```
Chardet: {'encoding': 'utf-8', 'confidence': 0.99, 'language': ''}
```

On executing above code we got the type of encoding, which was **utf-8.**

But, it is necessary to determine the encoding type of text which are impossible to read. For that,

Code:

```
#Within the utf-8 file sometext are not enocoded properly. So we are determing the type of encode used in those sentences.
print(x[4])
text_file = open("Encode_test.txt", "w")
text_file.write("Line to be tested: {}".format(x[4]))
text_file.close()
with open(u'C:\\Users\\gowtham\\Encode_test.txt', 'rb') as detect_line_encoding:
    detection_line = chardet.detect(detect_line_encoding.read())# Find the encoding type of php file.
    print('Chardet:', detection_line)
```

```
âœ“
Chardet: {'encoding': 'Windows-1252', 'confidence': 0.73, 'language': ''}
```

Explanation:

The content present in x[4] cannot be decoded. Therefore a text file was opened and it was written into it. On performing chardet function again we got the encoding format of that particular sentence. This is because while scraping words/emoji present in certain languages like Chinese, Korean, Cyrillic etc are encode into **Windows-1252 format. [Default one]**

How to rectify it?

This can be done by encoding the text again in Windows-1252/cp-1252 format and decode it back with utf-8.

Code:

```
print (x[1])
a = []
for i in range(len(x)):
    badtext = x[i]
    encoded_text = badtext.encode('cp1252','ignore')
    goodtext = encoded_text.decode('utf-8','ignore')
    a.append(goodtext)
    badtext = ''
    goodtext = ''
print (a[1])
```

```
Ð˜Ð³Ñ€Ð° Ð¿Ð¾ Ñ□ÐµÐ±Ðµ Ñ…Ð¾Ñ€Ð¾ÑˆÐ°Ñ□, Ð½Ð¾ Ñ‡Ð¸Ñ‚Ð°ÐºÐ¾Ð² Ð²Ñ□Ñ' Ð±Ð¾Ð»ÑŒÑˆÐµ Ð¸ Ð±Ð¾Ð»ÑŒÑˆÐµ. Ð›ÑƒÑ‡ÑˆÐµ Ð±Ñ‹ Ð²Ð°Ð»Ð² Ñ□Ð´Ðµ
Ð»Ð°Ð»Ð¸ ÐµÑ' Ð¿Ð»Ð°Ñ‚Ð½Ð¾Ð¹!
Игра по ебе хороша, но читаков вё больше и больше. Лучше бы валв делали её платной!
```

The unreadable characters present in x[1] is nothing but Russian text poorly encoded by cp1252. After correcting the poorly encoded text and stored it into array a[], a[1] gives the corresponding rightly encoded text for x[1]. This conversion was done for entire observation in x[].

Converting Binary Text:

After successfully encoded the text, the next type of text present was in the form of binary.

Code:

```
tmp_index = []
count = 0;
for i in range(len(a)):
    test_str = set(a[i])
    bits = {'0','1'}
    if bits == test_str or test_str == {'0'} or test_str == {'1'}:
        print ("The index number in which the binary string was present:{}".format(i))
        tmp_index.append(i)
        count = count+1
print ("Total binary string in the dataset:{}".format(count))
```

```
The index number in which the binary string was present:284
The index number in which the binary string was present:3350
Total binary string in the dataset:2
```

```
print (a[284])
print ("\n",a[3350])
```

```
0101010001001000010010010101001100100000010010010101001100100000010011100100111101010100001000000100000010010000000100011101000000
10100110101000101001000000010010010101010001010011001000000100000100100000001010111010000001010110010010000000100111101000110001000
0001001100010010010100011001000101001000000010000010100111001000100001000000010011010101100100100000001010010010001010100110001001
00101000111010010010100111101001110

 1
```

Explanation:

The above code test for any bits present in the array 'a'. tmp_index stores the index value where that binary text occurs and count stores the total occurrences. It is clearly visible at index 284 and 3350 the array contain only binary representation of text. It has to be converted into English.

Code:

```
def binaryToDecimal(temp_data):
    decimal_String = int(temp_data,2)
    return(decimal_String)
for i in tmp_index:
    binary_data = a[i]
    str_data =' '
    for j in range(0, len(binary_data), 8):
        temp_data = binary_data[j:j + 8]
        decimal_value = binaryToDecimal(temp_data)
        str_data = str_data + chr(decimal_value)
    a[i]=str_data
print("The binary converted text:{}".format(a[284]))
```

```
The binary converted text: THIS IS NOT A GAME ITS A WAY OF LIFE AND MY RELIGION
```

Explanation:

Here, bits are sliced into 8 bits[utf-8 standard] and later I made use of two function int() - converts specified value to integer number and chr() – converts that integer to character based on Unicode. As we can see the array a[284] is now got converted to a sentence.

Removing Smileys/Pictorial Text

Since I'm going to implement machine learning algorithm based on words present in the sentence, it is wise to remove emoji and all other pictorial representation from the text array.

Code:

```
print(a[5])
```

```
♥♥♥ 1uy1
```

```
b =[]
for i in range(len(a)):
    emoji_free_text = ''
    emoji_free_text = emoji.get_emoji_regexp().sub('', a[i])
    b.append(emoji_free_text)
```

```
print(b[5])
```

```
 1uy1
```

Explanation:

For this we make use of regex [replaces all emoji with empty string] and emoji package [To detect emoji present in a text] . The resultant output was stored in array b[]. In the above code the heart symbol was removed.

Language Conversion

To remove the complexity arise from different language representation, it is good to translate all those representation to English.

Code:

```
print (b[0])
```

```
owo
```

```
from google_trans_new import google_translator
c = []
for i in range(len(b)):
    try:
        translator = google_translator()
        translated = translator.translate(b[i], lang_tgt='en')
        c.append(translated)
    except:
        print (b[i])
        c.append(b[i])
print [c[0]]
```

```
Nice
```

Explanation:

From google_trans_new I made use of google_translator function which makes API call to google servers. By this function I converted all non-English text to English and stored it in array 'c[]'. For example, b[0] is 'owo' Russian text converted into 'Nice'.

Note:

This translation API can perform at most 1400 lines from single IP address. But the given dataset contains 5000 lines. I overcome this hurdle by making use of Nord VPN and implemented cmd line code to switch VPN address to execute translation for all the 5000 lines. Other method, manually one can change IP address after every 1400 lines.

Export the dataset

Now the given raw data set was cleaned and converted to English. It is now ready to undergo text preprocessing (NLP). Therefore convert the array into a data frame and export it has CSV file.

Code:

```
df_final = pd.DataFrame({"translated_text": c,"voted_up": y,"early_access" : z})
df_final.to_csv("final_dataset1.csv")
```

Chapter 3

Approach to Text Data

Preprocessing 3.1 Overview

This section explains in detail the various text preprocessing concepts that can be applied to our dataset containing English words only. Let us go through all the basic packages that are imported and used.

Packages	Explanation
1. nltk	Parent package to import nltk based packages.
2. pandas	To create mainly[data frame]
3. numpy	To create mainly[numpy array]
4. Textblob	Used effectively for text preprocessing[like tokenize, spell correct]
5. stopwords	Contains corpus of stop words present in English language. [nltk]
6. SnowballStemmer	Used for word stemming.[nltk]
7. word_tokenize	Used for splitting words. [nltk]
8. ngram	Used for clubbing words[nltk]
9. matplotlib	Used for graph plotting
10. seaborn	Similar to matplotlib but with advance features and styles.
11. re	Regular expression[Regex]
12. warning	To remove any warning message.
13. string	Used for string modulation
14. WordnetLemmetizer	Used for performing lemmatization[nltk]

Note:

In the above table [nltk] signifies that the package is a child of parent nltk. In all the upcoming function or concepts, I made use of lambda method to define a function.

3.2 Import the Modified Dataset

Code:

```
train_set = pd.read_csv('final_dataset.csv')
train_set.columns
```

```
Index(['Unnamed: 0', 'translated_text', 'voted_up', 'early_access'], dtype='object')
```

Explanation:

Reading the final_dataset.csv file and storing it as a data frame named train_set. In addition printing all the column names in that data frame.

3.3 Basic Text Modelling and Cleaning Activities

1. Dropping unwanted column values [Unnamed: 0] and filling empty values present in 'translated_text' with empty string

Code:

```
train_set.drop(columns = ['Unnamed: 0'], axis = 1, inplace = True)
train_set['translated_text'] = train_set['translated_text'].fillna(" ")
train_set.head()
```

	translated_text	voted_up	early_access
0	that	True	False
1	The game is good, but there are more and more ...	True	False
2	Very unique experience for sure, we need more ...	True	False
3	It needs work in areas, namely graphics, stabi...	True	True
4	✓	True	False

Explanation:

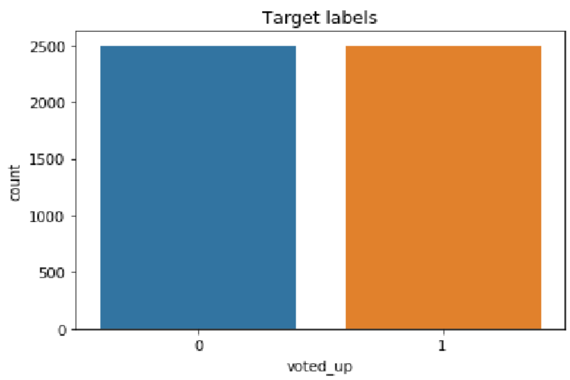
'Unnamed: 0' column was dropped because it was just additional index column. The above code axis 1 specify column and inplace = true reflects the changes back to original dataset. Next fillna() function was used to perform empty string addition. This was done because text preprocessing can't be done on float data type. Therefore we are converting all float type [empty set] to string.

2. Converting Boolean characters in 'voted_up' and 'early_access' to its numeric equivalent and finding their distribution.

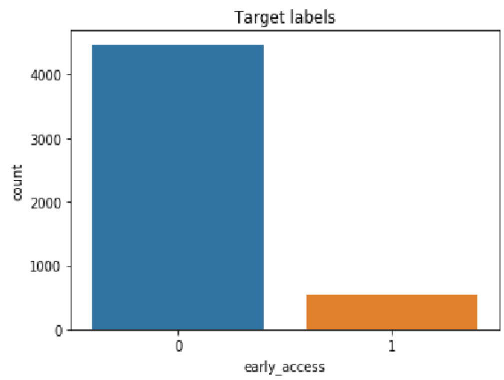
Code:

```
train_set["voted_up"] = train_set["voted_up"].astype(int)
train_set["early_access"] = train_set["early_access"].astype(int)
train_set.head()
warnings.filterwarnings("ignore")

target_Arr = np.array([int(i) for i in train_set["voted_up"]])
print("\n Target_Arr[voted_up]",target_Arr)
print("\n Length of Target_ARR[voted_up]",len(target_Arr))
count = 0
for i in target_Arr:
    if i == 0:
        count+=1
print("\n0 : ",count)
print("\n1 : ",len(target_Arr)-count)
sns.countplot(train_set.voted_up)
plt.title("Target labels")
plt.show()
target_Array = np.array([int(i) for i in train_set["early_access"]])
print("\n Target_Arr[early_access]",target_Array)
print("\n Length of Target_ARR[early_access]",len(target_Array))
count = 0
for i in target_Array:
    if i == 0:
        count+=1
print("\n0 : ",count)
print("\n1 : ",len(target_Array)-count)
sns.countplot(train_set.early_access)
plt.title("Target labels")
plt.show()
```

```
Target_Arr[voted_up] [1 1 1 ... 0 0 0]

Length of Target_ARR[voted_up] 5000

0 :  2500

1 :  2500
```




```
Target_Arr[early_access] [0 0 0 ... 0 1 0]

Length of Target_ARR[early_access] 5000

0 :  4468

1 :  532
```



Explanation:

First, the categorical representation on True and False present in 'voted_up' and 'early_access' converted to their numerical equivalent 1 and 0 using **int** function. This was done because ML models can work only on numerical data. Later their distribution count was calculated, printed and plotted with the help of seaborn function. From the histogram it is clear that 'voted_up' contains equally distributed data which is good for training and prediction. But on the other hand the 'early_access' distribution was of the form 9:1. So it might be not good for predicting unknown values. But all these are assumptions, until we finalize and implement the model to verify these assumptions.

3. Dropping all rows from data frame containing non-English character in 'translated_text' column

Code:

```
from nltk.corpus import words
Word = list(set(words.words()))
train_set = train_set[train_set['translated_text'].str.contains('|'.join(Word))]
train_set = train_set.reset_index(drop=True)
train_set.shape
```

```
(4885, 3)
```

Explanation:

Here, we are dropping all rows containing non English words. This was done by making use of words package from nltk and join function. Later I perform reset_index() to reorder index. After this step, I was left with 4885 rows and 3 columns which signifies 115 rows was removed. This was done because non-English characters caries no value.

4. **Converting all characters present in the 'translated_text' column to lower case**

This can be done by lower() function. This was done because the word 'BAD' and 'bad' will be treated as different words while performing bag of words or tf-idf vectorization. Therefore, I converted all the words to lower case.

Code:

```
train_set['translated_text'] = train_set['translated_text'].apply(lambda x: " ".join(x.lower() for x in x.split()))
train_set['translated_text'].head()
```

```
0                                                 that
1    the game is good, but there are more and more ...
2    very unique experience for sure, we need more ...
3    it needs work in areas, namely graphics, stabi...
4                                                 good
Name: translated_text, dtype: object
```

3.4 Feature Selection based on certain Characteristics

1. **Replacing all the special characters present in the 'translated_text' column with '' ''.**

Code:

```
train_set['translated_text'] = train_set['translated_text'].str.replace('[^\w\s]','')
train_set['translated_text'].head()
```

```
0                                                 that
1    the game is good but there are more and more c...
2    very unique experience for sure we need more o...
3    it needs work in areas namely graphics stabili...
4                                                 good
Name: translated_text, dtype: object
```

Explanation:

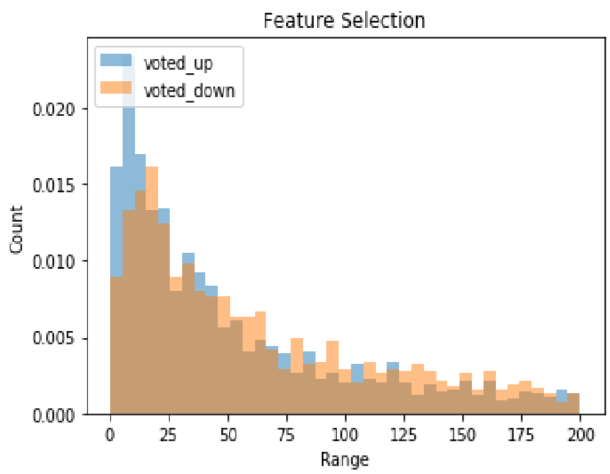
It was already decided that I'm going to make use of words in the review text to build the classifier model. But, what about the punctuation or special characters present in text. Whether we need to keep or remove it. For example, to classify mail has ham or spam, punctuation have equal weightage like words. In our case, since we are finding whether the review is good or bad, there was no need of punctuation mark. Therefore, I removed it using string replace function. '\w' specify numbers and characters in English. '\s' specify white space.

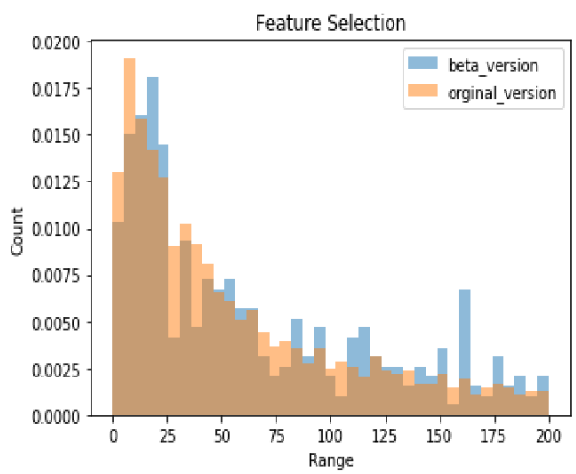
2. **Based on word count as a parameter**

```
#To caluclate length of reviews excluding spaces
train_set['txt_length'] = train_set['translated_text'].apply(lambda x: len(x) - x.count(" "))
print(train_set.head())
#Plotting the above findings in terms of histogram to evaluate whether words can be used to classify text.
bins = np.linspace(0,200,40)
plt.hist(train_set[train_set['voted_up'] == 1]['txt_length'],bins,alpha=0.5,normed=True,label='voted_up')
plt.hist(train_set[train_set['voted_up'] == 0]['txt_length'],bins,alpha=0.5,normed=True,label='voted_down')
plt.legend(loc = 'upper left')
plt.show()
#Plotting the above findings in terms of histogram to evaluate whether words can be used to classify text.
bins = np.linspace(0,200,40)
plt.hist(train_set[train_set['early_access'] == 1]['txt_length'],bins,alpha=0.5,normed=True,label='beta_version')
plt.hist(train_set[train_set['early_access'] == 0]['txt_length'],bins,alpha=0.5,normed=True,label='orginal_version')
plt.legend(loc = 'upper left')
plt.show()
```

Output:

```
                                    translated_text  voted_up  early_access  \
0                                              that         1             0
1  the game is good, but there are more and more ...         1             0
2  very unique experience for sure, we need more ...         1             0
3  it needs work in areas, namely graphics, stabi...         1             1
4                                              good         1             0

   txt_length
0           4
1          80
2          48
3        1313
4           4
```





Explanation:

I fixed words in the text as my building block for classification model. Therefore, in order to prove my assumption, I plotted histogram of length of character distribution based on '0' and '1' value for column 'early_access' and 'voted_up'. We clearly see that there is word difference between 'voted_up' and 'voted_down' as well as for 'beta_version' and 'orginal_version'. So it's a good feature to distinguish. I used **matplotlib** function to plot the histogram.

3.5 Text Preprocessing Steps

In all the upcoming function or concepts, I made use of lambda method to define a function and split() to split words in a text.

1. Removing all the stop words present in the 'translated_text' column.

Code:

```
stop = stopwords.words('english')
train_set['translated_text'] = train_set['translated_text'].apply(lambda x: " ".join(x for x in x.split() if x not in stop))
train_set['translated_text'].head()
```

```
0
1      game good cheats would better valves made paid
2                        unique experience sure need
3   needs work areas namely graphics stability qui...
4                                               good
Name: translated_text, dtype: object
```

Explanation:

This was done because, stop words like ‘the’, ‘is’, ‘was’ etc can occur numerous time in our dataset but carries no additional information or meaning. For this stopwords package from nltk was used. The code basically club all English words which are not present in that stop word corpus.

2. Dropping rows containing empty strings within the data frame.

Code:

```
train_set.replace('', np.nan, inplace=True)
train = train_set.dropna()
train = train.reset_index(drop=True)
print (train.head())
train.shape
```

```
                                     translated_text  voted_up  early_access  \
0     game good cheats would better valves made paid         1             0
1                        unique experience sure need         1             0
2  needs work areas namely graphics stability qui...         1             1
3                                               good         1             0
4                                     may ________ 00        1             0

   txt_length
0          80
1          48
2        1313
3           4
4          31
(4844, 4)
```

Explanation:

It was removed because it has no meaning. Empty string was first assigned a ‘nan’ value and by making use of dropna() all rows containing ‘nan’ was dropped. The resultant data frame was assigned to a new df named ‘train’. At the end, the new data frame contains only 4844 rows.

3. Correcting spelling mistakes in the data frame.

```
x = "badd"
TextBlob(x).correct()
```

```
TextBlob("bad")
```

```
train['translated_text'].apply(lambda x: str(TextBlob(x).correct()))
train['translated_text'].head()
```

```
0      game good cheats would better valves made paid
1                        unique experience sure need
2   needs work areas namely graphics stability qui...
3                                               good
4                                    may ________ 00
Name: translated_text, dtype: object
```

Explanation:

Since we made use of google translator some poorly translated text may contain wrong spelling. Therefore, I made use of textblob to correct the spelling. Because it is more efficient way to do. Consider 'badd' wrongly written text converted to 'bad'

Note:

Since our dataset contains 4844 rows of text, this action takes some time for completion

4. Performing Tokenization of words in 'translated_text' column.

Code:

```
from nltk.tokenize import word_tokenize
train['translated_text'].apply(lambda x:word_tokenize(x.lower()))
train['translated_text'].head()
```

```
0       game good cheats would better valves made paid
1                             unique experience sure need
2    needs work areas namely graphics stability qui...
3                                                  good
4                                       may ________ 00
Name: translated_text, dtype: object
```

Explanation:

I made use of word_tokenize because there may be text like "the game isn't good' than it becomes difficult to catch the context of the text. So it is necessary to split text isn't to is n' t. Therefore tokenization of words performed.

5. Performing Stemming of words in 'translated_text' column.

Code:

```
stemmer = SnowballStemmer("english", ignore_stopwords=True)
train['translated_text'].apply(lambda x: " ".join([stemmer.stem(word) for word in x.split()]))
train['translated_text'].head()
```

```
0       game good cheats would better valves made paid
1                             unique experience sure need
2    needs work areas namely graphics stability qui...
3                                                  good
4                                       may ________ 00
Name: translated_text, dtype: object
```

Explanation:

Here, I made use of SnowballStemmer which performs quickly and more effectively than PorterStemmer. But, what is the need for stemming, for ex. Consider words like, liked, likeable all carry same meaning but three different word. To solve this stemming was performed.

6. Performing lemmatization of words in 'translated_text' column.

Code:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("worse", pos="a")) #specifying that worse represent here is adjective
```

```
bad
```

```
train['translated_text'] = train['translated_text'].apply(lambda x: " ".join([lemmatizer.lemmatize(x) for word in x.split()]))
train['translated_text'].head()
```

```
0    game good cheats would better valves made paid...
1    unique experience sure need unique experience ...
2    needs work areas namely graphics stability qui...
3                                                good
4      may ________ 00 may ________ 00 may ________ 00
Name: translated_text, dtype: object
```

Explanation:

In the above code, the word ‘worse’ can be converted to word ‘bad’ because both carry same meaning. To achieve this I made use of powerful stemming package WordNetLemmatizer.

Note:

All the above preprocessing activities were performed because to reduce the number of words, so the complexity of word representation in terms of bag of words/tf-idf can be reduced.

3.6 Data Splitting [Test/Train]

Code:

```
#Assign each column Values:
train_text=train['translated_text']
target1=train.iloc[:,1]
target2=train.iloc[:,2]
from sklearn.model_selection import train_test_split
train_x1,test_x1,train_y1,test_y1 = train_test_split(train_text,target1,test_size = 0.2, random_state = 1)
train_x2,test_x2,train_y2,test_y2 = train_test_split(train_text,target2,test_size = 0.2, random_state = 1)
```

Explanation:

Consider the dataset from our project containing ‘voted_up’ column. The first 2500 records contain ‘true’ and the next 2500 records with ‘false’ value. Here if we use k-fold of ‘5’ than there are five experiment each with 1000 records, here the two experiments will contain only class ‘true’ which is not good. It can leads to under fitting problem. Therefore, the data can be sliced and trained using **train test split** or by using **stratified cross** validation. I made use of train_test_split package from sklearn to split the dataset 80:20. 80 for training and 20 for test testing. And I gave random_state ‘1’ means the data got sliced in a random manner. **I’m planning to implement stratified cross validation to split dataset has future work.**

Chapter 4

Implementation [Machine Learning Algorithm]

4.1 Overview

Choice of Execution method:

Before moving on to the implementation of various machine learning algorithms, I faced one issue in selecting the hyper parameters to be considered. For example, for the creation of a bag of words using the countvectorizer function, we can pass several parameters like max_df, ngram,min_df, etc. Similarly, some Machine Learning model performs well without tf-idf, and these models can take different values for l2 penalty. So it won't be a wise option to implement cross-validation separately for each parameter selection. For example, cross-validation to choose ngram -> (1,1) or (2,2) than performing another cross-validation for selecting l2 penalty value. Therefore, I tried to make use of the '**Pipeline'** function through which I can define countvectorizer, tf-idf, and classifier function. All the function specified within '**Pipeline'** are executed in a **series manner**. In order to choose the right hyper parameters, I made use of the **'GridSearchCV'** function. This function basically tests all combinations of input parameters and output the parameters that suit best for the model based upon model score. It makes use of a cross-validation value of '5'. **GridSearchCV** takes pipeline object, parameters and other tuning parameters as input.

Drawback of GridSearchCV:

Suppose consider, if user want to choose best 'alpha' value from set {0.0001,0.01,1,5,10} for penalty 'l2' and 'ngram' value from {(1,1), (2,2)}. Than we got 5C2 = 10 combinations which means the classifier model have to run for 10 different times to fetch the optimal combination solution. This takes quite some time based on your system capacity.

Choice of Algorithm:

Since I need to develop a classifier model that should best satisfy our two main objectives which carry different meanings. For example, for the 'voted_up' column the feature words that carry positive response can be words like 'good', 'awesome', and so on. On the other hand, for 'early_ access' words like 'new version', 'beta' plays a major role in classifying the reviews. From this, it's clear that each word present within the sentence carries some added value to it in predicting the target value. Therefore, I choose Multinomial Naïve-Bayes Bayes which assign a probable value to all feature word present in a sentence to calculate the probabilistic value of target variable either it's positive or negative. Second, I choose to implement the Stochastic Gradient Descent method for executing Linear SVM, here the derivative of loss will be calculated by taking 'n' points = '1' at a time. At last, I choose the normal K-nearest neighbors classifier which does not fall in the above two categories.

To conclude:

The dataset contains minimal rows and the number of words present in each sentence is minimal, I decided to implement probabilistic based Machine learning model along with geometric based ones. In order to create a baseline model, I made use of dummy classifier which predicts any one class all the time.

Machine Learning models:

- Baseline model[Dummy classifier]
- Multinomial Naïve-Bayes[Probabilistic]
- Linear SVM based upon Stochastic Gradient decent [Kernel]
- kNearestNeighbors [Geometric]

Note:

Probabilistic means the output will be predicted based upon the probability value.

Geometric means the output calculated based on calculating the distance between two points.

The below table consist of general packages used in this section.

Package	**Explanation**
Pipeline	To create pipeline function
GridSearchCV	To create cross validation function to implement various combination of parameters over the defined model.
CountVectorizer	To implement bag of words through CountVectorizer function.
TfidfTransformer	To create Tf-Idf representation of words. [TfidfTransformer function]
Dummyclassifier	To implement classification function based on certain parameters.
Confusion matrix	To build confusion matrix.
roc_auc_score	To calculate area under curve value. [here curve is ROC]
roc_curve	To draw roc curve for each model output.

Let us consider short description and reason to use certain concepts,

1. **Bag of words**

Since machine process everything in the form integers, it is mandatory to convert words in a sentence to some integer format. So I made use of bag of words.

Example, Consider a sentence

S1: He hits the ball with the bat. -> He hits the ball with bat

1 1 2 1 1 1 -> Integer conversion

2. **Tf-Idf**

We make use of Tf-Idf approach to address drawback present bag of words representation. In above example the carries value ‘2’ while calculating distance between ‘the’ and ‘ball’ was one, but between ‘the’ and ‘hits’ was zero. In order to **nullify or normalize** this drawback Tf-Idf approach was performed.

3. **ngram:** It was used to club words in a sentence. For example ‘new version’ as a whole carry different meaning when compared individually. (1,1) and (1,2)/(2,2) are used.

4. **df_max = 0.2** specify remove words that occur more than 20% of the document. In this report all models are tested for 0.1, 0.2 and 0.3 values. Done to avoid **over fitting** of data.

Justification of certain parameters used:

5. **n-jobs** = -1, signifies to make use of entire processor power [No parallel jobs]
6. **cv** = 5, cross validation of 5 splits.
7. **Idf** – Boolean, specifies whether to perform inverse document frequency.

4.2 Baseline Model using dummy classifier.

Here, I implemented a baseline model with the help of dummy classifier that predicts the class with most frequency all the time.

1. Voted_up

I got an accuracy score of 0.5025 and predicted 'NO' for all the times.

Code:

```
#Dummy classifier for [Voted_up]
from sklearn.dummy import DummyClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import classification_report
dummy = DummyClassifier(strategy='most_frequent').fit(train_x1,train_y1)
y1dummy = dummy.predict(test_x1)
cmtx_dummy1 = pd.DataFrame(
    confusion_matrix(test_y1, y1dummy),
    index=['True:NO', 'True:YES'],
    columns=['Pred:NO', 'Pred:YES']
)
print (cmtx_dummy1)
print(classification_report(test_y1, y1dummy, zero_division = 0))
accuracy_baseline = metrics.accuracy_score(test_y1,y1dummy)
print("BASELINE MODEL ACCURACY : ",accuracy_baseline)
```

Output:

```
          Pred:NO  Pred:YES
True:NO       487         0
True:YES      482         0
              precision    recall  f1-score   support

           0       0.50      1.00      0.67       487
           1       0.00      0.00      0.00       482

    accuracy                           0.50       969
   macro avg       0.25      0.50      0.33       969
weighted avg       0.25      0.50      0.34       969

BASELINE MODEL ACCURACY :  0.5025799793601651
```

2. Early access:

Code:

```
#Dummy classifier for [early_access]
dummy_ea = DummyClassifier(strategy='most_frequent').fit(train_x2,train_y2)
y2dummy = dummy_ea.predict(test_x2)
cmtx_dummy2 = pd.DataFrame(
    confusion_matrix(test_y2, y2dummy),
    index=['True:NO', 'True:YES'],
    columns=['Pred:NO', 'Pred:YES']
)
print (cmtx_dummy2)
print(classification_report(test_y2, y2dummy, zero_division = 0))
accuracy_baseline = metrics.accuracy_score(test_y1,y2dummy)

print("BASELINE MODEL ACCURACY : ",accuracy_baseline)
```

Output:

```
          Pred:NO  Pred:YES
True:NO       877         0
True:YES       92         0
              precision    recall  f1-score   support

           0       0.91      1.00      0.95       877
           1       0.00      0.00      0.00        92

    accuracy                           0.91       969
   macro avg       0.45      0.50      0.48       969
weighted avg       0.82      0.91      0.86       969

BASELINE MODEL ACCURACY :  0.5025799793601651
```

I got same accuracy of 0.525 and predicted 'NO' for most time.

4.3 Multinomial Naïve-Bayes model

As discussed early, it is a probabilistic based model that make use of Bayes probabilistic formula

P(A/B) = (P(B/A) * P(A))/ P(B) here, A is target variable and B are input features ranging x1, x2, …………., xn. [In our case, the words in each sentence]

```
#Define the structure of the kNN-Classifier algorithm.
from sklearn.naive_bayes import MultinomialNB
clf_nb = Pipeline([('vect', CountVectorizer()),
                   ('tfidf', TfidfTransformer()),
                   ('clf', MultinomialNB())])
parameters = {'vect__ngram_range': [(1, 1), (2, 2)],
              'vect__max_df':[0.1,0.2,0.3],
              'tfidf__use_idf': (True, False),
              'clf__alpha': [0.1,0.001, 1, 10, 100]}
gs_clf_nb = GridSearchCV(clf_nb, parameters, n_jobs=-1, cv = 5)
```

Code: voted_up

```
#[voted_up]Naive-Bayes model involving object creation and trainning.
gs_clf_nb = gs_clf_nb.fit(train_x1, train_y1)

#Outputting the best combination parameters along with accuracy score.
print("The right combination values to get best score[voted_up]:",gs_clf_nb.best_params_)
print("The best score value[voted_up]:",gs_clf_nb.best_score_)

#Predicting the target[voted_up value]
y1_predicted_nb = gs_clf_nb.predict(test_x1)
print("The accuracy score obtained for test data[voted_up]:")
print(np.mean(y1_predicted_nb == test_y1))

#Confusion matrics for naive_bayes ml model.
cmtx_nb = pd.DataFrame(
    confusion_matrix(test_y1, y1_predicted_nb),
    index=['True:NO', 'True:YES'],
    columns=['Pred:NO', 'Pred:YES']
)
print("\nCONFUSION MATRIX")
print (cmtx_nb)
print("\nCLASSIFICATION REPORT")
print(classification_report(test_y1, y1_predicted_nb, zero_division = 0))
fpr_nb, tpr_nb, threshold = roc_curve(test_y1, y1_predicted_nb)
```

Output:

```
The right combination values to get best score[voted_up]: {'clf__alpha': 1, 'tfidf__use_idf': True, 'vect__max_df': 0.2, 'vect_
_ngram_range': (1, 1)}
The best score value[voted_up]: 0.760516129032258
The accuracy score obtained for test data[voted_up]:
0.7750257997936016

CONFUSION MATRIX
          Pred:NO  Pred:YES
True:NO       421        66
True:YES      152       330

CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.73      0.86      0.79       487
           1       0.83      0.68      0.75       482

    accuracy                           0.78       969
   macro avg       0.78      0.77      0.77       969
weighted avg       0.78      0.78      0.77       969
```

Explanation:

alpha = In Naïve-bayes it doesn’t signify ‘l2’ penalty, rather it specified for smoothening purposes.

So the model return the best combination of value as alpha = 1, ngram = (1,1), df_max = 0.2, idf =true. The model accuracy score for train and test data was 0.76 and 0.77 respectively. This signifies our model doesn’t have any over or under fitting issues.

Code: early_access

```
#[early_access]Naive-Bayes model involving object creation and trainning.
gs_clf_nb_ea = gs_clf_nb.fit(train_x2, train_y2)

#Outputting the best combination parameters along with accuracy score.
print("The right combination values to get best score[early_access]:",gs_clf_nb_ea.best_params_)
print("The best score value[early_access]:",gs_clf_nb_ea.best_score_)

#Predicting the target[early access] value.
y2_predicted_nb = gs_clf_nb_ea.predict(test_x2)
print("The accuracy score obtained for test data[early_access]:")
print(np.mean(y2_predicted_nb == test_y2))

#Confusion matrics for naive_bayes ml model.
cmtx_nb_ea = pd.DataFrame(
    confusion_matrix(test_y2, y2_predicted_nb),
    index=['True:NO', 'True:YES'],
    columns=['Pred:NO', 'Pred:YES']
)
print("\nCONFUSION MATRIX")
print (cmtx_nb_ea)
print("\nCLASSIFICATION REPORT")
print(classification_report(test_y2, y2_predicted_nb, zero_division = 0))
fpr_nb_ea, tpr_nb_ea, threshold = roc_curve(test_y2, y2_predicted_nb)
```

Output:

```
The right combination values to get best score[early_access]: {'clf__alpha': 1, 'tfidf__use_idf': True, 'vect__max_df': 0.1, 'v
ect__ngram_range': (1, 1)}
The best score value[early_access]: 0.8895483870967743
The accuracy score obtained for test data[early_access]:
0.9050567595459237

CONFUSION MATRIX
          Pred:NO  Pred:YES
True:NO       877         0
True:YES       92         0

CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.91      1.00      0.95       877
           1       0.00      0.00      0.00        92

    accuracy                           0.91       969
   macro avg       0.45      0.50      0.48       969
weighted avg       0.82      0.91      0.86       969
```

Explanation:

Here, the max_df value I got was 0.1, ngram = (1,1), idf =true, alpha =1. And the point to be noted was even though we got high accuracy score but the model failed to predict class ‘1’. This is because the train model contains 9:1 split ratio for 0 and 1 correspondingly. Therefore it perform same like baseline model.

4.4 Linear SVM [Stochastic Gradient Decent classifier] model

As discussed earlier, the derivative of loss take one sample at a time and updates the weights for next iteration. This will be more helpful to capture in depth characteristics of each features while training.

Code:

```
#Define the structure of the SVMClassifier through SGD algorithm.
from sklearn.linear_model import SGDClassifier
clf_sgd_svm = Pipeline([('vect', CountVectorizer()),
                        ('tfidf', TfidfTransformer()),
                        ('clf-sgd-svm', SGDClassifier(loss='hinge', penalty='l2',n_iter_no_change = 5, random_state=42,
                                                      learning_rate = 'optimal')),
])
parameters_svm = {'vect__ngram_range': [(1, 1), (2, 2)],
                  'vect__max_df':[0.1,0.2,0.3],
                  'tfidf__use_idf': (True, False),
                  'clf-sgd-svm__alpha': (0.1,0.01,0.001, 1, 10)
}
gs_clf_sgd_svm = GridSearchCV(clf_sgd_svm, parameters_svm, n_jobs=-1, cv = 5)
```

Explanation:

Loss = ‘hinge’ signifies that the SGD classifier was for Linear SVM.

Penalty = ‘l2’ was used.

N_iter_no_change = 5 signifies, if the weight calculated doesn’t change for more than 5 epochs, this signifies we attained the local minima.

Random_state = 42 signifies the shuffling pattern to be followed.

Learning rate = ‘optimal’ learning rate is calculated based on alpha value added. So we cannot customize learning rate as we did in our week 1 assignment.

Alpha = ‘C’ value that need to be added to the loss function.

All other parameters are general common to all models and explanation was given earlier.

1. **Voted up**

```
#[voted_up]SVM model object creation and trainning.
gs_clf_sgd_svm = gs_clf_sgd_svm.fit(train_x1,train_y1)

#Outputting the best combination parameters along with accuracy score.
print("The right combination values to get best score[voted_up]:",gs_clf_sgd_svm.best_params_)
print("The best score value[voted_up]:",gs_clf_sgd_svm.best_score_)

#Predicting the target[voted_up value]
y1_predicted_svm = gs_clf_sgd_svm.predict(test_x1)
print("The accuracy score obtained for test data[voted_up]:")
print(np.mean(y1_predicted_svm == test_y1))

#Plotting confusion matrix.
cmtx_svm = pd.DataFrame(
    confusion_matrix(test_y1, y1_predicted_svm),
    index=['True:NO', 'True:YES'],
    columns=['Pred:NO', 'Pred:YES']
)
print("\nCONFUSION MATRIX")
print (cmtx_svm)
print("\nCLASSIFICATION REPORT")
print(classification_report(test_y1, y1_predicted_svm, zero_division = 0))
fpr_svm, tpr_svm, threshold = roc_curve(test_y1, y1_predicted_svm)
```

Output:

```
The right combination values to get best score[voted_up]: {'clf-sgd-svm__alpha': 0.001, 'tfidf__use_idf': True, 'vect__max_df':
0.1, 'vect__ngram_range': (1, 1)}
The best score value[voted_up]: 0.751741935483871
The accuracy score obtained for test data[voted_up]:
0.7523219814241486

CONFUSION MATRIX
         Pred:NO  Pred:YES
True:NO      381       106
True:YES     134       348

CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.74      0.78      0.76       487
           1       0.77      0.72      0.74       482

    accuracy                           0.75       969
   macro avg       0.75      0.75      0.75       969
weighted avg       0.75      0.75      0.75       969
```

Explanation:

The **best alpha value is = 0.001,** ngram = (1,1), df_max = 0.1, idf =true.. The accuracy score obtained for train and test data was 0.7517 and 0.7523 respectively. [Signifies perfect model]

2. **early_access**

```
#[early_access]SVM model object creation and trainning.
gs_clf_sgd_svm_ea = gs_clf_sgd_svm.fit(train_x2,train_y2)

#Outputting the best combination parameters along with accuracy score.
print("The right combination values to get best score[early_access]:",gs_clf_sgd_svm_ea.best_params_)
print("The best score value[early_access]:",gs_clf_sgd_svm_ea.best_score_)

#Predicting the target[early_access value]
y2_predicted_svm = gs_clf_sgd_svm_ea.predict(test_x2)
print("The accuracy score obtained for test data[early_access]:")
print(np.mean(y2_predicted_svm == test_y2))

#Confusion matrix[early_access]
cmtx_svm_ea = pd.DataFrame(
    confusion_matrix(test_y2, y2_predicted_svm),
    index=['True:NO', 'True:YES'],
    columns=['Pred:NO', 'Pred:YES']
)
print("\nCONFUSION MATRIX")
print (cmtx_svm_ea)
print("\nCLASSIFICATION REPORT")
print(classification_report(test_y2, y2_predicted_svm, zero_division = 0))
fpr_svm_ea, tpr_svm_ea, threshold = roc_curve(test_y2, y2_predicted_svm)
```

Output:

```
The right combination values to get best score[early_access]: {'clf-sgd-svm__alpha': 0.1, 'tfidf__use_idf': True, 'vect__max_d
f': 0.1, 'vect__ngram_range': (1, 1)}
The best score value[early_access]: 0.8895483870967743
The accuracy score obtained for test data[early_access]:
0.9050567595459237

CONFUSION MATRIX
         Pred:NO  Pred:YES
True:NO      877         0
True:YES      92         0

CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.91      1.00      0.95       877
           1       0.00      0.00      0.00        92

    accuracy                           0.91       969
   macro avg       0.45      0.50      0.48       969
weighted avg       0.82      0.91      0.86       969
```

Explanation:

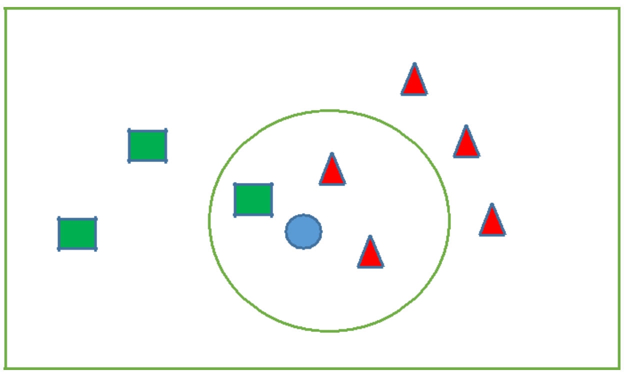
Here, the alpha value I got was 0.1, ngram = (1,1), df_max = 0.1, idf =true.. And the point to be noted was even though we got high accuracy score but the model failed to predict class ‘1’. This is because the train model contains 9:1 split ratio for 0 and 1 correspondingly. Therefore it perform same like baseline model.

4.5 K-NearestNeighbors classifier model.

Principle concept:

In kNN classifier algorithm tries to map an object with a class of object based upon the plurality concept. i.e an object is mapped to the class most common among its neighbors. For example if k=2, then the object will be assigned to the class containing two neighbors.

Implementation diagram:



Consider the above graph where the new point in blue needed to be mapped with a class when the given k value is 2 then it will be mapped to the class of red since it is closer to two objects belonging to red class when compared to class green.

Code:

```
#Define the structure of the kNN-Classifier algorithm.
from sklearn.neighbors import KNeighborsClassifier
clf_knn = Pipeline([('vect', CountVectorizer()),
                    ('tfidf', TfidfTransformer()),
                   ('clf-knn', KNeighborsClassifier(weights='uniform',metric = 'minkowski'))])
parameters_knn = {'vect__ngram_range': [(1, 1), (1, 2)],
              'vect__max_df':[0.1,0.2,0.3],
              'tfidf__use_idf': (True, False),
              'clf-knn__n_neighbors': [3,5,7,9]}
gs_clf_knn = GridSearchCV(clf_knn, parameters_knn, n_jobs=-1, cv = 5)
```

Explanation:

n_neighbors: used to specify the closest number of observation acceptable.

weights: specify that the weights are of uniform distribution.

metric: ‘minkowski’ default metric to calculate the distance.

1. **voted_up**

```
#[voted_up]kNN_classifier model object creation and trainning.
gs_clf_knn = gs_clf_knn.fit(train_x1, train_y1)

#Outputting the best combination parameters along with accuracy score.
print("The right combination values to get best score[voted_up]:",gs_clf_knn.best_params_)
print("The best score value[voted_up]:",gs_clf_knn.best_score_)

#Predicting the target[voted_up value]
y1_predicted_knn = gs_clf_knn.predict(test_x1)
print("The accuracy score obtained for test data[voted_up]:")
print(np.mean(y1_predicted_knn == test_y1))

#Confusion matrix[voted_up]
cmtx_knn = pd.DataFrame(
    confusion_matrix(test_y1, y1_predicted_knn),
    index=['True:NO', 'True:YES'],
    columns=['Pred:NO', 'Pred:YES']
)
print("\nCONFUSION MATRIX")
print (cmtx_knn)
print("\nCLASSIFICATION REPORT")
print(classification_report(test_y1, y1_predicted_knn, zero_division = 0))
fpr_knn, tpr_knn, threshold = roc_curve(test_y1, y1_predicted_knn)
```

Output:

```
The right combination values to get best score[voted_up]: {'clf-knn__n_neighbors': 5, 'tfidf__use_idf': False, 'vect__max_df':
0.2, 'vect__ngram_range': (1, 1)}
The best score value[voted_up]: 0.6095483870967742
The accuracy score obtained for test data[voted_up]:
0.5985552115583075

CONFUSION MATRIX
          Pred:NO  Pred:YES
True:NO       451        36
True:YES      353       129

CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.56      0.93      0.70       487
           1       0.78      0.27      0.40       482

    accuracy                           0.60       969
   macro avg       0.67      0.60      0.55       969
weighted avg       0.67      0.60      0.55       969
```

Explanation:

I got optimal solution **for n_neighnors: 5** use-idf : false, max_df = 0.2 and ngram = (1,1). The accuracy score for train and test was 0.60 and 0.59 respectively. The reason for low score observed here because unlike other two models **it just calculate the distance of feature which are closer and output the results.**

2. **early_access**

```
#[early_access]kNN_classifier model object creation and trainning.
gs_clf_knn_ea = gs_clf_knn.fit(train_x2, train_y2)

#Outputting the best combination parameters along with accuracy score.
print("The right combination values to get best score[early_access]:",gs_clf_knn_ea.best_params_)
print("The best score value[early_access]:",gs_clf_knn_ea.best_score_)

#Predicting the target[early_access value]
y2_predicted_knn = gs_clf_knn_ea.predict(test_x2)
print("The accuracy score obtained for test data[early_access]:")
print(np.mean(y2_predicted_knn == test_y2))

#early access [Confusion matrix]
cmtx_knn_ea = pd.DataFrame(
    confusion_matrix(test_y2, y2_predicted_knn),
    index=['True:NO', 'True:YES'],
    columns=['Pred:NO', 'Pred:YES']
)
print("\nCONFUSION MATRIX")
print (cmtx_knn_ea)
print("\nCLASSIFICATION REPORT")
print(classification_report(test_y2, y2_predicted_knn, zero_division = 0))
fpr_knn_ea, tpr_knn_ea, threshold = roc_curve(test_y2, y2_predicted_knn)
```

Output:

```
The right combination values to get best score[early_access]: {'clf-knn__n_neighbors': 7, 'tfidf__use_idf': True, 'vect__max_d
f': 0.2, 'vect__ngram_range': (1, 2)}
The best score value[early_access]: 0.8898064516129033
The accuracy score obtained for test data[early_access]:
0.9050567595459237

CONFUSION MATRIX
          Pred:NO  Pred:YES
True:NO       877         0
True:YES       92         0

CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.91      1.00      0.95       877
           1       0.00      0.00      0.00        92

    accuracy                           0.91       969
   macro avg       0.45      0.50      0.48       969
weighted avg       0.82      0.91      0.86       969
```

Explanation:

I got optimal solution **for n_neighnors: 7** use-idf : True, max_df = 0.2 and ngram = (1,2). Again no true positive was predicted this is because very minimal availability of train data for true positive. [9:1] ratio.

Chapter 5

Performance Evaluation

Precision (also called positive predictive value) is the fraction of retrieved instances that are relevant while recall (also known as sensitivity) is the fraction of relevant instances that are retrieved. Both precision and recall are therefore based on an understanding and measure of relevance. F-score is calculated from precision and recall.

Consider the following figure,

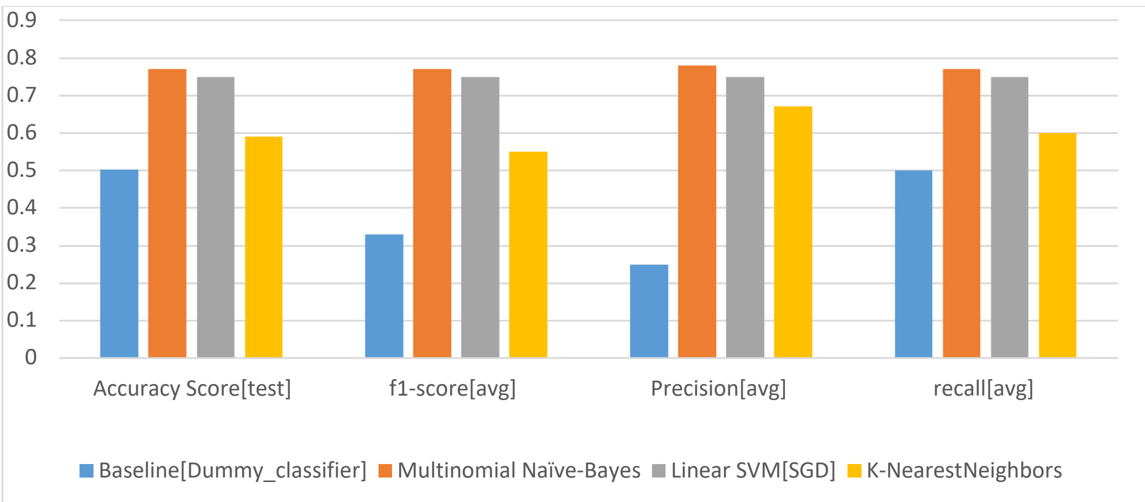


Fig1: voted_up

From fig 1, it was clear that Naïve-Bayes model outperforms all other model in terms of all metrics. Second comes the Linear SVM [using SGD] followed by kNN and baseline model.

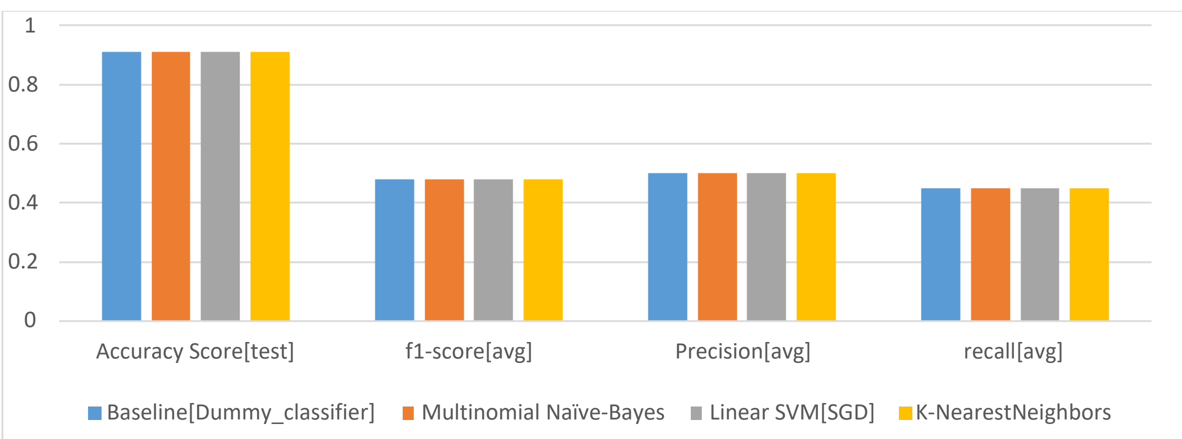


Fig2: early_access

From fig 2. It was clear that all our model perform similar to that of baseline model. Therefore it is not possible to determine early_access value using ml models trained.

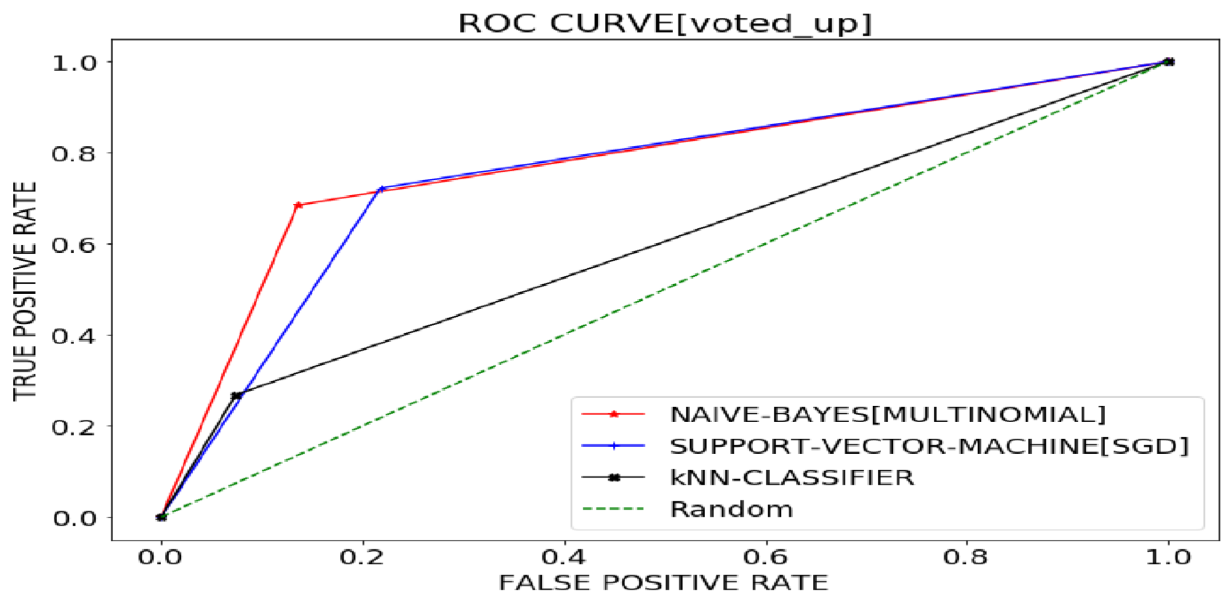
Further evaluation can be done by plotting roc curve and calculating Area under curve value.

ROC is a curve plotted based on True Positive and False Positive Rate and AUC is simply the area of region under ROC. It signifies larger the AUC value the better the model performs.

Code: [voted_up]

```
#Implementing ROC curve and AUC value[voted_up]
from sklearn.metrics import roc_curve
plt.rcParams['figure.figsize'] = (10.0,7.0)
plt.rcParams['figure.constrained_layout.use']= True
plt.rc('font',size=18)
plt.plot(fpr_nb, tpr_nb, marker='*',color='red', label='NAIVE-BAYES[MULTINOMIAL]')
plt.plot(fpr_svm, tpr_svm, marker='+',color='blue', label='SUPPORT-VECTOR-MACHINE[SGD]')
plt.plot(fpr_knn, tpr_knn, marker='X',color='black', label='kNN-CLASSIFIER')
plt.plot([0, 1], [0, 1], linestyle='--', label='Random',color ='green')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend()
plt.title('ROC CURVE[voted_up]')
plt.show()
auc_nb = roc_auc_score(test_y1,y1_predicted_nb)
print('\nAUC Naive-Bayes[voted_up]: %.3f' % auc_nb)
auc_svm = roc_auc_score(test_y1,y1_predicted_svm)
print('\nAUC Support_Vector_Machine[voted_up]: %.3f' % auc_svm)
auc_kNN = roc_auc_score(test_y1,y1_predicted_knn)
print('\nAUC kNearestNeighboursClassifier[voted_up]: %.3f' % auc_kNN)
```

Output:



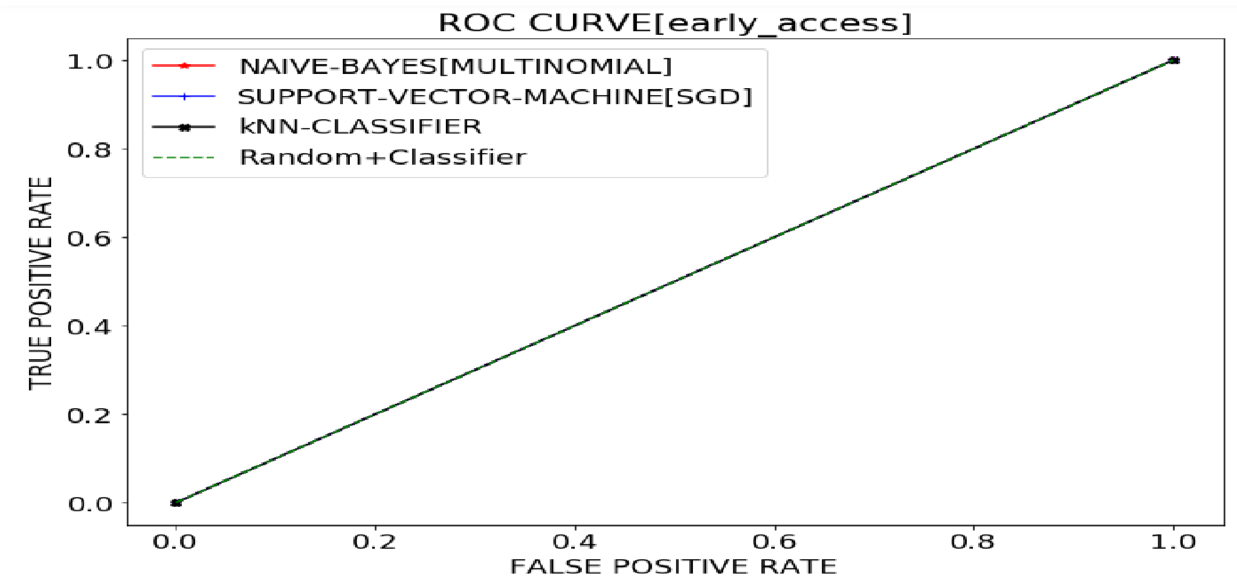

```
AUC Naive-Bayes[voted_up]: 0.775

AUC Support_Vector_Machine[voted_up]: 0.752

AUC kNearestNeighboursClassifier[voted_up]: 0.597
```

Explanation: Based on AUC and roc curve it's proven that Naïve-Bayes performs better than other modes for [voted_up] with AUC value of 0.775.

Code: [early_access]

```
#Implementing ROC curve and AUC value[early_access]
plt.plot(fpr_nb_ea, tpr_nb_ea, marker='*',color='red', label='NAIVE-BAYES[MULTINOMIAL]')
plt.plot(fpr_svm_ea, tpr_svm_ea, marker='+',color='blue', label='SUPPORT-VECTOR-MACHINE[SGD]')
plt.plot(fpr_knn_ea, tpr_knn_ea, marker='X',color='black', label='kNN-CLASSIFIER')
plt.plot([0, 1], [0, 1], linestyle='--', label='Random+Classifier',color ='green')
plt.xlabel('FALSE POSITIVE RATE')
plt.ylabel('TRUE POSITIVE RATE')
plt.legend()
plt.title('ROC CURVE[early_access]')
plt.show()
auc_nb_ea = metrics.roc_auc_score(test_y2,y2_predicted_nb)
print('\nAUC Naive-Bayes[early_acess]: %.3f' % auc_nb_ea)
auc_svm_ea = metrics.roc_auc_score(test_y2,y2_predicted_svm)
print('\nAUC Support_Vector_Machine[early_acess]: %.3f' % auc_svm_ea)
auc_kNN_ea = metrics.roc_auc_score(test_y2,y2_predicted_knn)
print('\nAUC kNearestNeighboursClassifier[early_acess]: %.3f' % auc_kNN_ea)
```

Output:




```
AUC Naive-Bayes[early_acess]: 0.500

AUC Support_Vector_Machine[early_acess]: 0.500

AUC kNearestNeighboursClassifier[early_acess]: 0.500
```

Explanation:

From above results it's clear that our model performs same like baseline model, because the number of sample available for training positive rate is very minimal. One reason for this behavior was poorly distributed train/ test split ratio of 9:1. Another one might be the issue in train –test split of data where very minimal training data points for '1' was available. As mentioned earlier, the prediction accuracy for label '1' can be slightly increased by making use of **'stratified cv'** approach.

Chapter 6

Conclusion [Appendix]

In this project, I successfully downloaded the raw dataset containing review text for Steam gaming platform and performed data cleaning activities, data preprocessing, implementation of machine learning algorithms like Multinomial Naïve-Bayes classifier, Linear Support vector Machine using SGD, K-NearestNeighbors and evaluated the performance using charts and graphs based on classification report, 'ROC' curve and 'AUC' value against baseline classifier.

Note:

If you notice, for 'voted_up' the accuracy score of both train and test data almost similar signifies that the model built was free from over and under fitting issues. On the other hand, even tough, the accuracy score of train and test are high for early_access, the model fail to predict positive label class. This behavior will cause problem while testing other future test datasets.

To conclude, based on the results obtained it is clear that proposed machine learning model can be used to predict whether the game was recommended by the user or not, but it cannot be used to predict whether the review was for beta version or not.

Codes are attached in a Final_exam.zip file along with this report.

finale_Exam.php: Contains downloaded raw json file.

Final_dataset.csv: Contains English converted text, index, voted_up and early_access column.

NLP_ml(1).py: Contains code for executing chapter 2 [Cleaning raw data]

NLP_ml(2).py: Contains code for Chapter 3 , 4 and 5 [Text preprocessing, ML implementation and evaluation] **Note: High computational power computer is required to run all these codes.** **Encode_text.txt:** Contains wrongly encoded text.