



GROUP ASSIGNMENT

CT107-3-3-TXSA

TEXT ANALYTICS AND SENTIMENT ANALYSIS

CSDA__CT107-3-3-TXSA-L-5__2022-12-05

Student ID	Student Name
TP060288	Daniel Candra
TP058396	Kevin Ahmadiputra
TP058466	Kevin Matthew Adyan
TP059076	Kevin Setiawan Miharjo

HAND OUT DATE : 20th DECEMBER 2022

HAND OUT DATE : 7th APRIL 2023

WEIGHTAGE : 25%

INSTRUCTIONS TO CANDIDATES:

1. Submit your assignment via Moodle.
2. Students are advised to underpin their answers with the use of references (cited using the 7th Edition of APA Referencing Style).
3. Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld.
4. Cases of plagiarism will be penalized.
5. You must obtain 50% overall to pass this module.

Table of Contents

1	Unigram	1
1.1	Unigram - Unsmoothed	2
1.2	Unigram - Smoothed	4
2	Bigram	6
2.1	Bigram - Unsmoothed	8
2.2	Bigram - Smoothed	9
3	Sentences Probabilities	11
3.1	Unigram – Manual Compute	11
3.2	Bigram – Manual Compute	12
3.3	Justification of the selected best model	14
3.4	Unigram & Bigram Language Models	14
4	Supervised Text Classification	16
4.1	Sentiment Prediction	16
4.2	Naïve Bayes Classifier	21
5	References	27

1 Unigram

First of all, the unigram refers to a single token or word in a text, which is considered the simplest form in the n-gram series (Nair, 2021). The n in n-gram means the number of words or tokens in sequence. In this section, the Unigram model will be used to calculate the word or token probabilities of the given text or corpus.

UNIGRAM

```
#Import Libraries
import nltk, re
import numpy as np
import pandas as pd
from copy import deepcopy

#Read the Txt File
t1 = open("Text Corpus.txt")
corpus1 = t1.read()
#print(corpus1)

sent = corpus1.split('\n')
print("\n\nThere are ", len(sent), "sentences in the given txt file")
for i in sent:
    print(i)

<s> He read a book </s>
<s> I read a different book </s>
<s> He read a book my Danielle </s>

There are 3 sentences in the given txt file
<s> He read a book </s>
<s> I read a different book </s>
<s> He read a book my Danielle </s>
```

Figure 1: Unigram – Import Libraries & Read txt file

In order to create and use unigram, there are some libraries required which are nltk, re, numpy, pandas, and deepcopy. Detailly, the nltk library provides tools and resources for NLP, numpy support for numerical computations, array, and matrix computations. Meanwhile, the pandas are the library to help in data analysis which provides a set of tools for working with structured data, and the deepcopy it to create deep copies of an object (Desai, 2019).

Besides importing required libraries, Figure 1 also depicts the code to read the given text file and then assign it to a declared variable. Additionally, the split function is used to separate each sentence and shows the three sentences from the given text file.

```

nlk_tk = []
for i in sent:
    nlk_tk = nlk_tk + i.split(' ')

#Display the result
print("There are",len(nlk_tk), "Words in this text file (Dirty)")
print(nlk_tk)

nlk_tk = [nlk_tk for nlk_tk in nlk_tk if nlk_tk != "<s>" and nlk_tk != "</s>"]
print("\n\nThere are",len(nlk_tk), "Words in this text file (Clean without <s> and </s>)")
print(nlk_tk)

There are 21 Words in this text file (Dirty)
['<s>', 'He', 'read', 'a', 'book', '</s>', '<s>', 'I', 'read', 'a', 'different', 'book', '</s>', '<s>', 'He', 'read', 'a', 'book', 'my', 'Danielle', '</s>']

There are 15 Words in this text file (Clean without <s> and </s>)
['He', 'read', 'a', 'book', 'I', 'read', 'a', 'different', 'book', 'He', 'read', 'a', 'book', 'my', 'Danielle']

```

Figure 2: Unigram - Word Tokenization & Pre-processing

Figure 2 above shows the code to perform word tokenization for the given sentences by using the split function which shows there are 21 words. However, <s> and </s> which may be considered noise have to remove. Hence, loop and if-else functions are utilized to remove the noise which results in the sentences being cleaned and there are 15 words after the cleaning process.

1.1 Unigram - Unsmoothed

```

1.1 UNIGRAM - UNSMOOTHED

#Get Unique Words & Sort it
unique = list(set(nlk_tk))
unique.sort()
uniques = [unique for unique in unique if unique != "<s>" and unique != "</s>"]
uniques.sort()

tot_uni = len(uniques) #TOTAL NUMBER OF UNIQUE

print("Unique - There are", tot_uni, "unique words without boundaries in the corpus, which are:")
#print(tot_uni)
for words in uniques:
    print(words)

#Get token without <s> and </s>
tokens = [nlk_tk for nlk_tk in nlk_tk if nlk_tk != "<s>" and nlk_tk != "</s>"]

tot_tokens = len(tokens) #TOTAL NUMBER OF TOKENS (CLEAN)

print("\n\nThere are", tot_tokens, " word tokens in the given txt file exclude <s> and </s>")
print(tot_tokens)
print(tokens)

Unique - There are 8 unique words without boundaries in the corpus, which are:
Danielle
He
I
a
book
different
my
read

There are 15 word tokens in the given txt file exclude <s> and </s>
15
['He', 'read', 'a', 'book', 'I', 'read', 'a', 'different', 'book', 'He', 'read', 'a', 'book', 'my', 'Danielle']

```

Figure 3: Unigram – Get Unique Words & Word Token

The code snippet in Figure 3 tells the step to select and put unique words from the given text into a list variable and then sort it using the sort function from Python. In addition, the <s> and </s> are also removed from the unique words. On the other side, the tokens or words from the given corpus were also collected except the <s> and </s>. As the result, there are 8 unique words and 15-word tokens from the given corpus.

```
#Formula and Makrov theorem
uni_prob = {}

for word in uniques:
    wc = tokens.count(word)
    prob = float(wc)/tot_tokens
    uni_prob[word] = prob

print ("Unsmoothed Unigram Probability Values:\n")
for word, prob in uni_prob.items():
    print("P ({0}) = {1}" .format(word,prob))

Unsmoothed Unigram Probability Values:

P (Danielle) = 0.06666666666666667
P (He) = 0.13333333333333333
P (I) = 0.06666666666666667
P (a) = 0.2
P (book) = 0.2
P (different) = 0.06666666666666667
P (my) = 0.06666666666666667
P (read) = 0.2
```

Figure 4: Unigram – Calculate Probabilities by using the Formula for Unsmoothed Unigram

To calculate the probabilities of each word token in Unsmoothed Unigram, the loop function combined with the formula implemented. By using the $P(w) = c(w)/N$ formula where P is the probability, c represents the number of selected tokens or words, while N is the total amount of text inside the given corpus, the probabilities of each word are calculated and stored inside the dictionary. Finally, each value inside the dictionary is printed by using the loop function, and the probabilities of each word from Unsmoothed Unigram are shown in the table below.

Unigram Probabilities
P (Danielle) = 0.06666666666666667 P (He) = 0.13333333333333333 P (I) = 0.06666666666666667 P (a) = 0.2 P (book) = 0.2 P (different) = 0.06666666666666667 P (my) = 0.06666666666666667 P (read) = 0.2

1.2 Unigram - Smoothed

1.2 UNIGRAM - SMOOTHED

```
#Copy Uniques words
UNK_uniques = deepcopy(uniques)
UNK_uniques.append("<UNK>")
UNK_uniques.sort()

tot_UNK = len(UNK_uniques)

print("There are",tot_UNK, "words in this list:\n")
print(UNK_uniques)

There are 9 words in this list:

['<UNK>', 'Danielle', 'He', 'I', 'a', 'book', 'different', 'my', 'read']
```

Figure 5: Unigram - Copy Method using deepcopy

The deep copy function is used to copy unique words stored in a variable to another variable and then append <UNK> to the newly declared variable which means the total unique word increased by 1 and become 9.

```
#Calculate the Probabilities
uni_prob_smooth = {}

for word in uniques:
    wc1 = tokens.count(word)
    prob_smooth = (float(wc1)+1) / (tot_tokens + tot_uni + 1)
    uni_prob_smooth[word] = prob_smooth

print ("Smoothed Unigram Probability Value:\n")
for word, prob_smooth in uni_prob_smooth.items():
    print("P ({0}) = {1}" .format(word,prob_smooth))

Smoothed Unigram Probability Value:

P (Danielle) = 0.08333333333333333
P (He) = 0.125
P (I) = 0.08333333333333333
P (a) = 0.16666666666666666
P (book) = 0.16666666666666666
P (different) = 0.08333333333333333
P (my) = 0.08333333333333333
P (read) = 0.16666666666666666
```

Figure 6: Unigram – Calculate Probabilities using Formula for Smoothed Unigram

Almost the same as the code to do Unsmoothed Unigram, the code snippet in Figure 6 above shows the way to calculate the probabilities of each word from the given corpus. However, the formula used is the $P(w) = (c(w)+1) / N+V+1$ where the V here indicates the total number of unique words inside the given corpus. The formula is based on the Laplace smoothing by adding 1 to the formula. As a result, the probabilities of each word token from Smoothed Unigram are displayed in the table below.

Unigram Probabilities
P (Danielle) = 0.08333333333333333
P (He) = 0.125
P (I) = 0.08333333333333333
P (a) = 0.16666666666666666
P (book) = 0.16666666666666666
P (different) = 0.08333333333333333
P (my) = 0.08333333333333333
P (read) = 0.16666666666666666

2 Bigram

```
2.0 BIGRAM

In [9]: #step 1 import libraries
import nltk
import numpy as np
import pandas as pd
from copy import deepcopy
from nltk import ngrams
```

Figure 7: Bigram - Libraries for Bigram

To perform the bigram language model, first, the necessary libraries need to be imported. This includes nltk, pandas, numpy, deepcopy, and ngrams.

```
In [10]: #step 2 read dataset
file1 = open("Text Corpus.txt")

#read the file
corpus1 = file1.read()
print(corpus1)

<s> He read a book </s>
<s> I read a different book </s>
<s> He read a book my Danielle </s>
```

Figure 8: Bigram - Reading Input File

The next step is reading the input text file which is named “Text Corpus.txt”

```
In [11]: sentences = corpus1.split('\n')
print("There are ", len(sentences), "Sentences in this text file")
for sent in sentences:
    print(sent)

There are 3 Sentences in this text file
<s> He read a book </s>
<s> I read a different book </s>
<s> He read a book my Danielle </s>
```

Figure 9: Bigram - Printing Output

The figure above shows that there are three sentences in the input text file.


```

In [12]: #Perform pre-cleaning process
tokens = []
for sent in sentences:
    tokens = tokens + sent.split(' ')

print("Words found in text\n")
print(tokens)

print("\n")
uniTokens = list(set(tokens))
print("Unique words found in text\n")
uniTokens

Words found in text

['<s>', 'He', 'read', 'a', 'book', '</s>', '<s>', 'I', 'read', 'a', 'different', 'book', '</s>', '<s>', 'He', 'read', 'a', 'book', 'k', 'my', 'Danielle', '</s>']

Unique words found in text

Out[12]: ['my', 'He', '</s>', 'a', 'Danielle', '<s>', 'book', 'read', 'different', 'I']

```

Figure 10: Bigram - Tokenization Process

The figure above shows the tokenization process. Here, the sentences are split with ' ' as the delimiter. This will return all tokens in the sentences which then will be stored in the tokens list. There are 21 tokens in total, with 10 unique tokens found in the input.

```

In [13]: #Create Bigram model
bigrams = []
for sent in sentences:
    bigram = list(ngrams(sent.split(), 2))
    bigrams = bigrams + bigram

print("A total of", len(bigrams), "bigrams were found\n")
for i in bigrams:
    print(i)

A total of 18 bigrams were found

('<s>', 'He')
('He', 'read')
('read', 'a')
('a', 'book')
('book', '</s>')
('<s>', 'I')
('I', 'read')
('read', 'a')
('a', 'different')
('different', 'book')
('book', '</s>')
('<s>', 'He')
('He', 'read')
('read', 'a')
('a', 'book')
('book', 'my')
('my', 'Danielle')
('Danielle', '</s>')

```

Figure 11: Bigram - Create Bigram Model

The figure above shows the Python code for creating the bigram model. First, tokens from each sentence are split. Then, it is paired into a bigram with the n-grams function with parameter 2 which is then stored in the bigrams list. The output shows that there are 18 bigrams found in the three sentences.

```
In [14]: #Unique bigrams
uniqueBigrams = list(set(bigrams))
uniqueBigrams.sort()
print("A total of ", len(uniqueBigrams), "unique bigrams were found\n")
for i in uniqueBigrams:
    print(i)

A total of 12 unique bigrams were found

('<s>', 'He')
('<s>', 'I')
('Danielle', '</s>')
('He', 'read')
('I', 'read')
('a', 'book')
('a', 'different')
('book', '</s>')
('book', 'my')
('different', 'book')
('my', 'Danielle')
('read', 'a')
```

Figure 12: Bigram - Output

The figure above shows the Python code to sort the unique bigrams. It shows that there are 12 unique bigrams found in the three sentences.

2.1 Bigram - Unsmoothed

2.1 BIGRAM - Unsmoothed

```
In [15]: #compute unsmoothed bigram using probability formula
bigramProbs = {}

for bigram in uniqueBigrams:
    word1 = bigram[0]
    word2 = bigram[1]
    for i in tokens:
        c2 = tokens.count(word1)
        c1 = bigrams.count(bigram)

    prob = float(c1)/float(c2)
    bigramProbs[bigram] = prob

print("Unsmoothed Bigram Probability values\n")
for bigram, prob in bigramProbs.items():
    print(bigram, ":", prob)

Unsmoothed Bigram Probability values

('<s>', 'He') : 0.6666666666666666
('<s>', 'I') : 0.3333333333333333
('Danielle', '</s>') : 1.0
('He', 'read') : 1.0
('I', 'read') : 1.0
('a', 'book') : 0.6666666666666666
('a', 'different') : 0.3333333333333333
('book', '</s>') : 0.6666666666666666
('book', 'my') : 0.3333333333333333
('different', 'book') : 1.0
('my', 'Danielle') : 1.0
('read', 'a') : 1.0
```

Figure 13: Bigram - Unsmoothed Model

The figure above shows the Python code used to compute the unsmoothed bigram probability. The bigram probability is stored in the dictionary called `bigramProbs`. The bigram probability is calculated for each bigram with the formula $probability = C1/C2$ where $C1$ is the number of occurrences of each unique bigram in the `bigrams` list, and $C2$ is the number of

[8]

occurrences of the first element in tokens list. The unsmoothed bigram probability is shown in the following table.

Bigram	Probability Value
('<s>', 'He')	0.66666
('<s>', 'I')	0.33333
('Danielle', '</s>')	1.0
('He', 'read')	1.0
('I', 'read')	1.0
('a', 'book')	0.66666
('a', 'different')	0.33333
('book', '</s>')	0.66666
('book', 'my')	0.33333
('different', 'book')	1.0
('my', 'Danielle')	1.0
('read', 'a')	1.0

2.2 Bigram - Smoothed

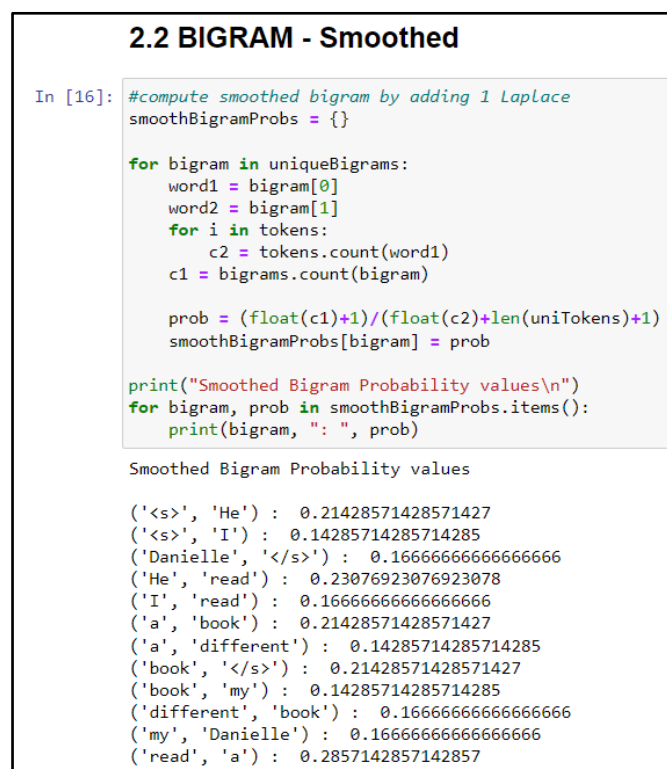


Figure 14: Bigram - Smoothed Model

The figure above shows the Python code used to compute the smoothed bigram probability. The smoothed bigram probability is stored in the dictionary called `smoothBigramProbs`. The smoothed bigram probability is calculated for each bigram by adding 1 to the formula based on the Laplace smoothing. The smoothed probability formula is

$probability = (C1 + 1)/(C2 + length + 1)$ where C1 is the number of occurrences of each unique bigram in the bigrams list, C2 is the number of occurrences of the first element in tokens list, and length is the number of unique tokens. The smoothed bigram probability values are shown in the following table.

Bigram	Probability Value
('<s>', 'He')	0.21428571428571427
('<s>', 'I')	0.14285714285714285
('Danielle', '</s>')	0.16666666666666666
('He', 'read')	0.23076923076923078
('I', 'read')	0.16666666666666666
('a', 'book')	0.21428571428571427
('a', 'different')	0.14285714285714285
('book', '</s>')	0.21428571428571427
('book', 'my')	0.14285714285714285
('different', 'book')	0.16666666666666666
('my', 'Danielle')	0.16666666666666666
('read', 'a')	0.2857142857142857

3 Sentences Probabilities

3.1 Unigram – Manual Compute

Tokens	Frequency
Danielle	1
He	2
I	1
a	3
book	3
different	1
my	1
read	3
Unique tokens (V) = 8	Total number of tokens (N) = 15

$$P(w_i) = \frac{\text{count}(w_i) + 1}{N + V + 1}$$

$$P(\text{Danielle}) = \frac{1 + 1}{15 + 8 + 1} = \frac{2}{24}$$

$$P(\text{He}) = \frac{2 + 1}{15 + 8 + 1} = \frac{3}{24}$$

$$P(I) = \frac{1 + 1}{15 + 8 + 1} = \frac{2}{24}$$

$$P(a) = \frac{3 + 1}{15 + 8 + 1} = \frac{4}{24}$$

$$P(\text{book}) = \frac{3 + 1}{15 + 8 + 1} = \frac{4}{24}$$

$$P(\text{different}) = \frac{1 + 1}{15 + 8 + 1} = \frac{2}{24}$$

$$P(\text{my}) = \frac{1 + 1}{15 + 8 + 1} = \frac{2}{24}$$

$$P(\text{read}) = \frac{3 + 1}{15 + 8 + 1} = \frac{4}{24}$$

$$P(\text{He read a book}) = P(\text{He}) \times P(\text{read}) \times P(a) \times P(\text{book})$$

$$P(\text{He read a book}) = \frac{3}{24} \times \frac{4}{24} \times \frac{4}{24} \times \frac{4}{24} \approx 0.000579$$

$$P(I \text{ read a different book}) = P(I) \times P(\text{read}) \times P(a) \times P(\text{different}) \times P(\text{book})$$

[11]

$$P(I \text{ read a different book}) = \frac{2}{24} \times \frac{4}{24} \times \frac{4}{24} \times \frac{2}{24} \times \frac{4}{24} \approx 0.000032$$

$$P(\text{He read a book my Danielle})$$

$$= P(\text{He}) \times P(\text{read}) \times P(a) \times P(\text{book}) \times P(\text{my}) \times P(\text{Danielle})$$

$$P(\text{He read a book my Danielle}) = \frac{3}{24} \times \frac{4}{24} \times \frac{4}{24} \times \frac{4}{24} \times \frac{2}{24} \times \frac{2}{24} \approx 0.000004$$

3.2 Bigram – Manual Compute

Tokens	Frequency
<s>	3
</s>	3
Danielle	1
He	2
I	1
a	3
book	3
different	1
my	1
read	3
Unique tokens (V) = 10	Total number of tokens (N) = 21

$$P(w_i) = \frac{\text{count}(w_{i-1}w_i) + 1}{\text{count}(w_{i-1}) + V + 1}$$

$$P(< s > \text{ He read a book } </s >)$$

$$= P(\text{He} | < s >) \times P(\text{read} | \text{He}) \times P(a | \text{read}) \times P(\text{book} | a)$$

$$\times P(</s > | \text{book})$$

$$P(\text{He} | < s >) = \frac{2 + 1}{3 + 10 + 1} = \frac{3}{14}$$

$$P(\text{read} | \text{He}) = \frac{2 + 1}{2 + 10 + 1} = \frac{3}{13}$$

$$P(a | \text{read}) = \frac{3 + 1}{3 + 10 + 1} = \frac{4}{14}$$

$$P(\text{book} | a) = \frac{2 + 1}{3 + 10 + 1} = \frac{3}{14}$$

[12]

$$P(</s> | book) = \frac{2+1}{3+10+1} = \frac{3}{14}$$

$$P(< s> He read a book </s>) = \frac{3}{14} \times \frac{3}{13} \times \frac{4}{14} \times \frac{3}{14} \times \frac{3}{14} \approx 0.000649$$

$$\begin{aligned} P(< s> I read a different book </s>) \\ &= P(I | < s>) \times P(read | I) \times P(a | read) \times P(different | a) \\ &\times P(book | different) \times P(</s> | book) \end{aligned}$$

$$P(I | < s>) = \frac{1+1}{3+10+1} = \frac{2}{14}$$

$$P(read | I) = \frac{1+1}{1+10+1} = \frac{2}{12}$$

$$P(a | read) = \frac{3+1}{3+10+1} = \frac{4}{14}$$

$$P(different | a) = \frac{1+1}{3+10+1} = \frac{2}{14}$$

$$P(book | different) = \frac{1+1}{1+10+1} = \frac{2}{12}$$

$$P(</s> | book) = \frac{2+1}{3+10+1} = \frac{3}{14}$$

$$P(< s> I read a different book </s>) = \frac{2}{14} \times \frac{2}{12} \times \frac{4}{14} \times \frac{2}{14} \times \frac{2}{12} \times \frac{3}{14} \approx 0.000035$$

$$\begin{aligned} P(< s> He read a book my Danielle </s>) \\ &= P(He | < s>) \times P(read | He) \times P(a | read) \times P(book | a) \\ &\times P(my | book) \times P(Danielle | my) \times P(</s> | Danielle) \end{aligned}$$

$$P(He | < s>) = \frac{2+1}{3+10+1} = \frac{3}{14}$$

$$P(read | He) = \frac{2+1}{2+10+1} = \frac{3}{13}$$

$$P(a | read) = \frac{3+1}{3+10+1} = \frac{4}{14}$$

$$P(book | a) = \frac{2+1}{3+10+1} = \frac{3}{14}$$

$$P(my | book) = \frac{1+1}{3+10+1} = \frac{2}{14}$$

$$P(Danielle | my) = \frac{1+1}{1+10+1} = \frac{2}{12}$$

$$P(</s> | Danielle) = \frac{1+1}{1+10+1} = \frac{2}{12}$$

$$\begin{aligned} P(< s> He read a book my Danielle </s>) &= \frac{3}{14} \times \frac{3}{13} \times \frac{4}{14} \times \frac{3}{14} \times \frac{2}{14} \times \frac{2}{12} \times \frac{2}{12} \\ &\approx 0.000012 \end{aligned}$$

3.3 Justification of the selected best model

From the performed calculation, the bigram model generally gives a slightly higher probability than unigram models. It means that the model has better performance in predicting whether the sentence has a high chance of existing or not. In addition, bigrams take word orders into account as they consider the probability of the word given the preceding word (Jurafsky & Martin, 2023). By having this feature, it can generate a syntactically more accurate result compared to the unigram model.

3.4 Unigram & Bigram Language Models

3. SENTENCE PROBABILITIES

```

In [17]: #sentence probabilities
dict= {}
col = ['Smoothed unigram sentence probability', 'Smoothed bigram sentence probability']
for sentence in sentences:
    dict[sentence] = {}#initialise dictionary for result
    split_text = sentence.split(' ')#split into tokens
    unigram = [word_tokens for word_tokens in split_text if word_tokens != "<s>" and word_tokens != "</s>"]#omit <s> and </s>
    bigram_var = list(nltk.bigrams(split_text))#bigram pairs

    #calculate sentence probabilities
    for token in unigram:
        dict[sentence]['Smoothed unigram sentence probability'] *= uni_prob_smooth[token]
    for bitoken in bigram_var:
        dict[sentence]['Smoothed bigram sentence probability'] *= smoothBigramProbs[bitoken]

    #display output
    for sentence, probabilities in dict.items():
        print("Sentence: {}".format(sentence))
        for uni_prob, bi_prob in probabilities.items():
            print('{}: {:.6f}'.format(uni_prob, bi_prob))
        print()

```

Sentence: <s> He read a book </s>
 Smoothed unigram sentence probability: 0.000579
 Smoothed bigram sentence probability: 0.000649

Sentence: <s> I read a different book </s>
 Smoothed unigram sentence probability: 0.000032
 Smoothed bigram sentence probability: 0.000035

Sentence: <s> He read a book my Danielle </s>
 Smoothed unigram sentence probability: 0.000004
 Smoothed bigram sentence probability: 0.000012

Figure 15: Sentence Probabilities

Figure 15 shows the Python code to compute the smoothed sentence probabilities using the unigram and bigram models that have been made. Firstly, a dictionary can be initialized to store the resulting unigram and bigram sentence probabilities for each sentence. Then, the sentence can be tokenized into unigram tokens and bigram pairs. For the unigram approach, the <s> and </s> tokens are omitted from the calculation as they do not affect the sentence probability.

After obtaining the tokens, the sentence probability can be calculated by multiplying the probability of every unigram token/bigram pair found in the sentence. The product is stored

in the dictionary, which would then be processed accordingly to display the resulting sentence probabilities, as seen in Figure 15.

4 Supervised Text Classification

4.1 Sentiment Prediction

Valence Aware Dictionary and Sentiment Reasoner (VADER) is a model which is available in the NLTK module used for sentiment analysis. VADER works by using a dictionary to map different features and emotional intensity of a text called polarity score (Calderon, 2017). VADER is chosen for the sentiment prediction section because it doesn't require any pre-processing as well as training data, and it can understand a text with underlying context such as emoticons, capital words, punctuations, slang, etc (Singh, 2020), which will be effective to predict the 'Musical_Instruments_Reviews' dataset which might consist of these features.

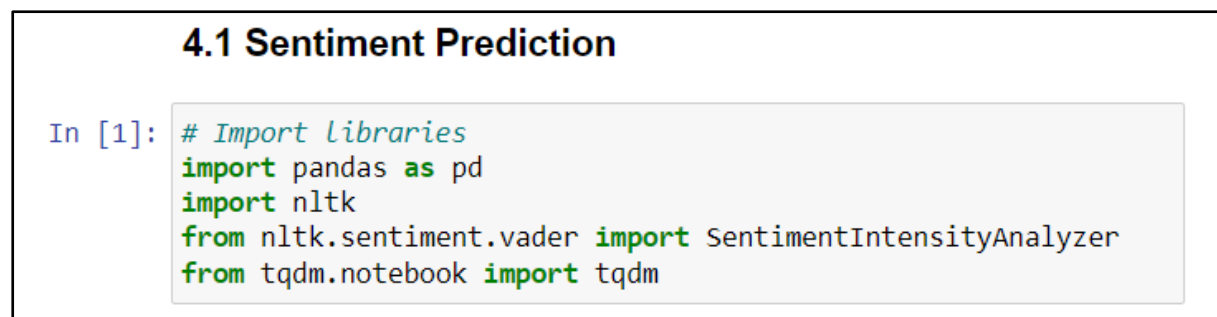


Figure 16 Sentiment Prediction - Import Libraries

First, necessary libraries will be imported which are pandas, nltk, SentimentIntensityAnalyzer, and tqdm.

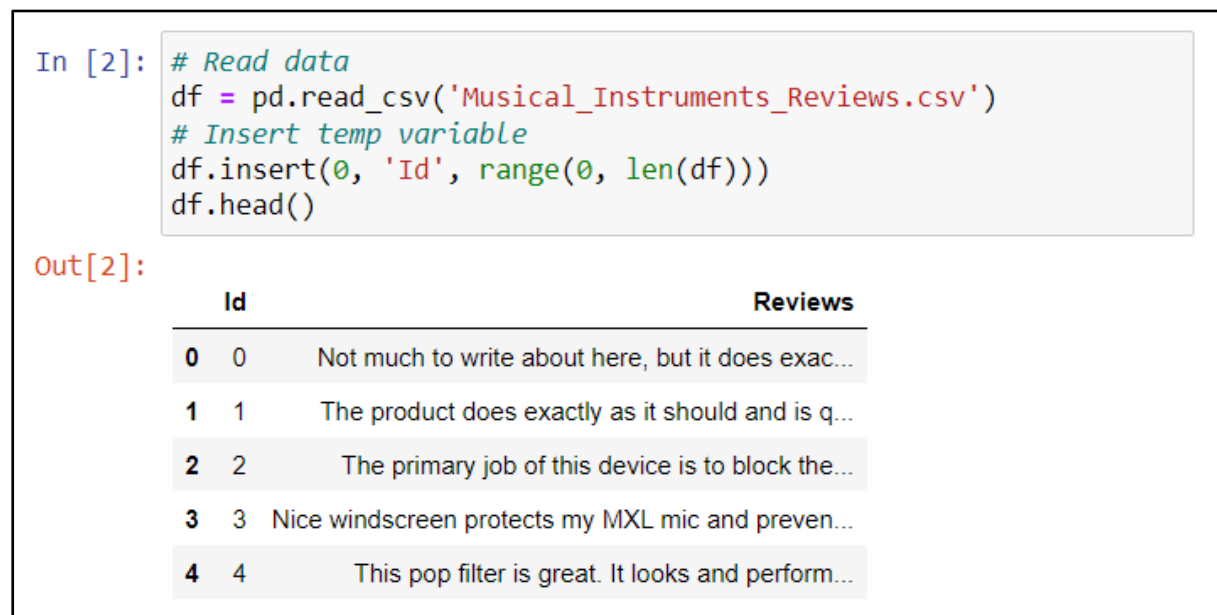



Figure 17 Sentiment Prediction - Read Data & Insert Temp Variable

Using pandas, read the CSV file 'Musical_Instruments_Reviews' and assign it as a data frame. The data frame only consisted of one variable 'Reviews' which consisted of strings. A temporary unique variable 'Id' is inserted to enable some operations later.

```
In [3]: # Declare SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()

In [4]: # Run the polarity score on the dataset
res = {}
for i, row in tqdm(df.iterrows(), total=len(df)):
    myid = row['Id']
    text = row['Reviews']
    res[myid] = sia.polarity_scores(text)

print(res)
```

100%  10254/10254 [00:08<00:00, 960.70it/s]

{0: {'neg': 0.057, 'neu': 0.79, 'pos': 0.153, 'compound': 0.6903}, 1: {'neg': 0.0, 'neu': 0.819, 'pos': 0.181, 'compound': 0.9588}, 2: {'neg': 0.113, 'neu': 0.858, 'pos': 0.029, 'compound': -0.7003}, 3: {'neg': 0.055, 'neu':

Figure 18 Sentiment Prediction - Apply VADER Model to Dataset

An instance of SentimentIntensityAnalyzer 'sia' is declared to enable usage of the VADER model. Then, iterate each row in the data frame. The library tqdm is used to wrap the iterable to visualize the progress of the iteration (PyPI, 2023). Using the polarity_scores function provided by the declared instance enables the prediction of the polarity scores of the review for each row in the data frame. The polarity score result is then stored in a dictionary called 'res'.

```
In [5]: # Create a separate dataset to contain polarity score result
vader = pd.DataFrame(res).T
vader = vader.reset_index().rename(columns={'index': 'Id'})
vader = vader.merge(df, how='left')
vader.head()
```

Out[5]:

	Id	neg	neu	pos	compound	Reviews
0	0	0.057	0.790	0.153	0.6903	Not much to write about here, but it does exac...
1	1	0.000	0.819	0.181	0.9588	The product does exactly as it should and is q...
2	2	0.113	0.858	0.029	-0.7003	The primary job of this device is to block the...
3	3	0.055	0.746	0.199	0.5859	Nice windscreen protects my MXL mic and preven...
4	4	0.000	0.791	0.209	0.7650	This pop filter is great. It looks and perform...

Figure 19 Sentiment Prediction - Polarity Score Result

The polarity score result stored in a dictionary needs to be converted into a data frame using the pandas DataFrame function and formatted by transposing the unique identifier of the dictionary to index and polarity scores to columns using the transpose (T in short) function, this operation will then be assigned to a new data frame called 'vader'. Next, the index column will be renamed to 'Id' and will be merged with the original dataset using 'Id' as the key. Figure 19 shows the merged dataset showing polarity scores and reviews.

VADER polarity scores consist of four scores which are neg, neu, pos, and compound. Neg describes the negativity of the text, neu the neutrality, pos the positivity, and compound reflects the overall sentiment of the text (Malde, 2020). Neg, neu, and pos scores made up for a score of 1. While compound score can range from -1 to 1. For this sentiment prediction, the compound score will be used as the main classifier for whether a review has a positive or negative sentiment.

```
In [8]: # Classify sentiment based on compound score
sentiment = {}
for i in range (0,len(vader)):
    if (vader['compound'][i]) >= 0:
        sentiment[i] = 'Positive'
    else:
        sentiment[i] = 'Negative'
```

Figure 20 Sentiment Prediction - Classify Sentiment based on Compound Score

VADER predicts three scores that represent a text's negativity, neutrality, and positivity. The usual compound score threshold to classify a text's sentiment are (Bajaj, 2021):

- Positive: compound score \geq (greater than equal to) 0.05
- Neutral: compound score $>$ (greater than) -0.05 and $<$ (less than) 0.05
- Negative: compound score \leq (less than equal to) -0.05

However, for this sentiment prediction, since only positive and negative sentiment will be predicted, the score threshold will be modified to:

- Positive: compound score \geq (greater than equal to) 0
- Negative: compound score $<$ (less than) 0

Figure 20 shows the classification operation. Iterate through the 'vader' data frame and classify whether a review is positive or negative, based on the modified threshold score. The sentiment result is stored in a dictionary called 'sentiment'.

```
In [9]: # Create a new dataframe to record the sentiment results
sent_df = pd.DataFrame.from_dict(sentiment, orient = 'index')
sent_df.rename(columns={0: 'Sentiment'}, inplace = True)
sent_df.head()
```

Out[9]:

Sentiment	
0	Positive
1	Positive
2	Negative
3	Positive
4	Positive

Figure 21 Sentiment Prediction - Sentiment Results in Dataframe

A new data frame 'sent_df' will be created from the 'sentiment' dictionary by using pandas DataFrame and from_dict function. Then, the column name of the data frame will be renamed to 'Sentiment'.

```
In [10]: # Join the sentiment result dataframe with original dataframe
df['Sentiment'] = sent_df
# Drop temp variable
df.drop(['Id'], axis = 1, inplace=True)
df.head()
```

Out[10]:

	Reviews	Sentiment
0	Not much to write about here, but it does exac...	Positive
1	The product does exactly as it should and is q...	Positive
2	The primary job of this device is to block the...	Negative
3	Nice windscreen protects my MXL mic and preven...	Positive
4	This pop filter is great. It looks and perform...	Positive

Figure 22 Sentiment Prediction - Join Sentiment Results Dataframe with Original Dataset

The 'Sentiment' column from the 'sent_df' data frame will be joined to the original 'df' data frame. The temporary variable 'Id' will be dropped since it is not used anymore.

```
In [11]: # Export final dataset to csv
df.to_csv('Musical_Instruments_Sentiment.csv', index=False)
```

Figure 23 Sentiment Prediction - Export Final Dataset to csv

Lastly, the updated data frame will be exported to a CSV file named 'Musical_Instrument_Sentiment' using the to_csv function, the index is set to false since the auto-indexing is not needed in the exported CSV file.

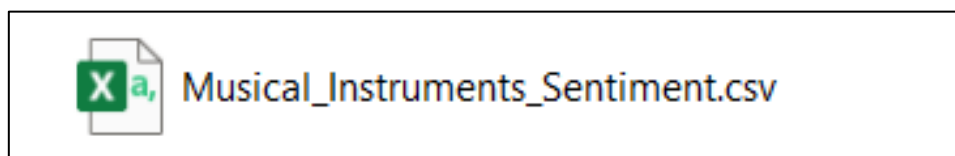


Figure 24 Sentiment Prediction - Exported Dataset

Reviews	Sentiment
Not much to write about here, but it does exactly what it's supposed to. filters out the pop sounds. now my recordings are much more crisp. it is one of the lowest prices pop filters on amazon so m	Positive
The product does exactly as it should and is quite affordable. I did not realize it was double screened until it arrived, so it was even better than I had expected. As an added bonus, one of the screens	Positive
The primary job of this device is to block the breath that would otherwise produce a popping sound, while allowing your voice to pass through with no noticeable reduction of volume or high frequen	Negative
Nice windscreen protects my MXL mic and prevents pops. Only thing is that the gooseneck is only marginally able to hold the screen in position and requires careful positioning of the clamp to avoid	Positive
This pop filter is great. It looks and performs like a studio filter. If you're recording vocals this will eliminate the pops that gets recorded when you sing.	Positive
So good that I bought another one. Love the heavy cord and gold connectors. Bass sounds great. I just learned last night how to coil them up. I guess I should read instructions more carefully. But	Positive
I have used monster cables for years, and with good reason. The lifetime warranty is worth the price alone. Simple fact: cables break, but getting to replace them at no cost is where it's at.	Negative
I now use this cable to run from the output of my pedal chain to the input of my Fender Amp. After I bought Monster Cable to hook up my pedal board I thought I would try another one and update	Positive
Perfect for my Epiphone Sheraton II. Monster cables are well constructed. I have several and never had any problems with any of them over the years. Got this one because I wanted the 90 degree	Positive
Monster makes the best cables and a lifetime warranty doesn't hurt either. This isn't their top of the line series but it works great with my bass guitar rig and has for some time. You can't go wrong wi	Positive
Monster makes a wide array of cables, including some that are very high end. I initially purchased a pair of Monster Rock Instrument Cable - 21 Feet - Angled to Straight 1/4-Inch plug to use with my k	Positive
I got it to have it if I needed it. I have found that I don't really need it that often and rarely use it. If I was really good I can see the need. But this is a keyboard not an organ.	Positive
If you are not used to using a large sustaining pedal while playing the piano, it may appear little awkward.	Positive
I love it, I used this for my Yamaha ypt-230 and it works great, I would recommend it to anyone.	Positive
I bought this to use in my home studio to control my midi keyboard. It does just what I wanted it to do.	Positive

Figure 25 Sentiment Prediction - Exported Dataset Content

Figure 24 and Figure 25 both show that the data frame has been successfully exported into CSV format with the correct output where the sentiment result is separated with a comma beside the review.

4.2 Naïve Bayes Classifier

4.2 Naïve Bayes Classifier

```
In [3]: # Read data
df = pd.read_csv('Musical_Instruments_Sentiment.csv')
df.head()
```

Out[3]:

	Reviews	Sentiment
0	Not much to write about here, but it does exac...	Positive
1	The product does exactly as it should and is q...	Positive
2	The primary job of this device is to block the...	Negative
3	Nice windscreen protects my MXL mic and preven...	Positive
4	This pop filter is great. It looks and perform...	Positive

Figure 26 Naive Bayes Classifier - Read Data

First, read the data from the exported dataset from section 4.1. The dataset should contain two columns which are the reviews and sentiment.

```
In [4]: import matplotlib.pyplot as plt
```

```
In [5]: # Display sentiment distribution count
print("Value counts for Sentiment (pos, neg)")
print(df['Sentiment'].value_counts())
sentiment_count = df.groupby('Sentiment').count()

# Plot sentiment distribution using bar chart
plt.bar(sentiment_count.index.values, sentiment_count['Reviews'],
        color=('r','g'))
plt.xlabel("Review Sentiments")
plt.ylabel('Number of Review')

# Print chart
plt.show()
```

```
Value counts for Sentiment (pos, neg)
Positive    9258
Negative     996
Name: Sentiment, dtype: int64
```

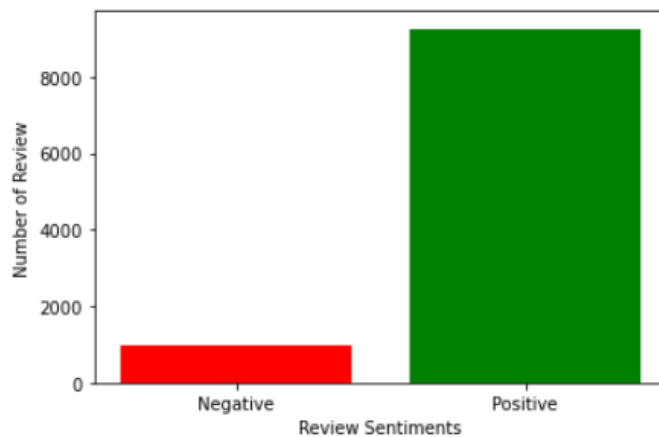


Figure 27 Naive Bayes Classifier - Data Exploration

Data exploration by plotting is done using the matplotlib library. First, the value counts for both negative and positive reviews will be displayed. Then, to visualize the distribution, a bar chart will be used to plot where the x-axis is the review sentiments (positive/negative) and the y-axis is the number of reviews. Figure 27 shows the result of value counts and plotted bar chart, it shows that positive reviews have a higher number of 9258 reviews compared to negative reviews with 996 reviews.


```
In [8]: # Count Vectorization
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import RegexpTokenizer

token = RegexpTokenizer(r'[a-zA-Z0-9]+')
cv = CountVectorizer(lowercase=True, stop_words='english', ngram_range=(1,1),
                    tokenizer = token.tokenize)
text_counts=cv.fit_transform(df['Reviews'])
```

Figure 28 Naive Bayes Classifier - Count Vectorization

Before training, the data needs to be numerically represented using word vectorization. CountVectorizer will generate a sparse matrix that represents all words in the file (Afham, 2019). The CountVectorizer parameters will also act in the pre-processing steps, for instance setting the lowercase to True to convert all words to lowercase, removing English stop words, setting ngram_range to include only individual words, and tokenizing the review text. Finally, the fit_transform function will be used to generate the sparse matrix.

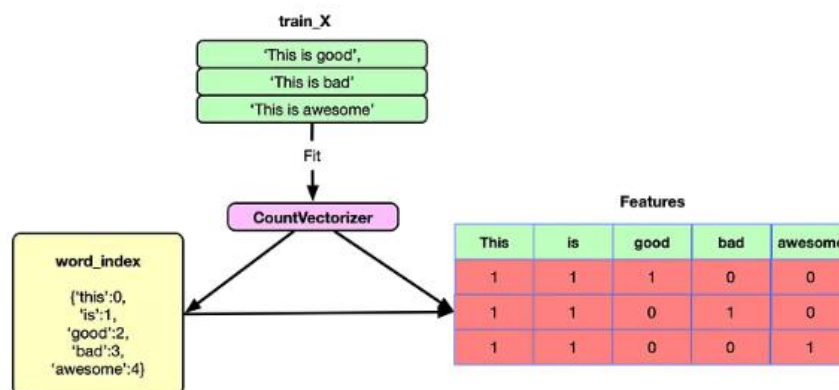


Figure 29 CountVectorizer Sparse Matrix Illustration Example (Afham, 2019)

```
In [9]: from sklearn.model_selection import train_test_split

# Split data to training and test set
X_train, X_test, y_train, y_test = train_test_split(
    text_counts, df['Sentiment'], test_size=0.3, random_state=1)
```

Figure 30 Naive Bayes Classifier - Split training and test dataset

The data is split into training and test dataset using the train_test_split function from sklearn.model_selection library. The dataset will be split between the training and testing dataset with a ratio of 7:3, where the random state is set to 1 to enable randomization during splitting. The training and testing dataset is split into:

[23]

- X_train: Training data that consists of review texts.
- X_test: Testing data that consists of review texts.
- y_train: Training data that consists of review sentiments.
- y_test: Testing data that consists of review sentiments.

```
In [10]: # Build MultinomialNaiveBayes (MNB)
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics

MNB_classifier = MultinomialNB().fit(X_train, y_train)
predicted_MNB = MNB_classifier.predict(X_test)

# Display results
print("Accuracy: ", metrics.accuracy_score(y_test, predicted_MNB))
print("\nClassification Report: \n", metrics.classification_report(y_test, predicted_MNB))

Accuracy: 0.8982775430614235

Classification Report:
              precision    recall  f1-score   support

   Negative      0.33      0.04      0.07       302
   Positive      0.90      0.99      0.95      2775

 accuracy
macro avg      0.62      0.51      0.51      3077
weighted avg    0.85      0.90      0.86      3077
```

Figure 31 Build Multinomial Naive Bayes Classifier

MultinomialNB is imported from `sklearn.naive_bayes` as well as `metrics` from `sklearn`. Using the `fit` function, both `X_train` and `y_train` will be used to train the Multinomial Naïve Bayes classifier. After training has been done, `predict` function will be used to predict the sentiment of reviews in the `X_test` dataset. Then, the model performance is evaluated using `metrics` functions. The model performance aspects that will be measured are accuracy and classification reports.

Figure 31 shows the performance evaluation of the Multinomial Naïve Bayes Classifier. The accuracy of the model is 0.8982%, this means that it managed to correctly predict the test samples with an 89.82% accuracy. The classification report consisted of four metrics which are precision, recall, F1-score, and support.

- Precision is used to measure how many positive predictions (True Positive) are correct.
- Recall is used to measure how many positive cases are predicted correctly over all positive cases in the data.
- F1-score is a combined measure of both precision and recall.
- Support reflects the number of actual occurrences of each class in the dataset.

Based on the classification report shown by precision, recall, F1-score, and support, the Multinomial Naïve Bayes model shows good performance when predicting positive sentiments with a score of 0.90, 0.99, 0.95, and 2775 occurrences respectively. On the other hand, the model is not performing well when predicting reviews with negative sentiments, as seen with a low score of precision, recall, and F1-score of 0.33, 0.04, 0.07, and 302 occurrences respectively. In this case, the model might tend to predict False Positives (FP) which might lead to increased inaccuracy. This inaccuracy is mostly likely caused by the huge gap between the positive and negative training dataset, as seen in Figure 27.

```
In [11]: # Display confusion Matrix
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, predicted_MNB)
print(cm)

[[ 11 291]
 [ 22 2753]]
```

Figure 32 Display Confusion Matrix

A confusion matrix will be visualized to see the distribution of actual vs predicted labels.

		Actual	
		Positive	Negative
Predicted	Positive	11 (True Positive)	291 (False Positive)
	Negative	22 (False Negative)	2753 (True Negative)

Table 1 Naïve Bayes Classifier - Confusion Matrix

The confusion matrix shows that the model correctly predicts 11 negative reviews and 2753 positive reviews, while it incorrectly predicts 291 negative reviews as positive reviews and 22 positive reviews as negative reviews. This supports the statement that the model tends to predict false positives especially incorrectly predicting negative reviews as positive reviews.

```
In [13]: # Cross validation report
print("Cross Validation Report ")
from sklearn.model_selection import cross_validate
MNB_classifier_CV1 = MultinomialNB().fit(X_train, y_train)
cv_score_1 = cross_validate(MNB_classifier_CV1, X_train, y_train, cv=5)
sorted(cv_score_1.keys())
cv_score_1['test_score']
print("Accuracy: %0.2f" % (cv_score_1['test_score'].mean()))

Cross Validation Report
Accuracy: 0.90
```

Figure 33 Naive Bayes Classifier - Cross Validation Report

To validate the model performance, a cross-validation report will be done. Another Multinomial Naïve Bayes model will be trained. Then, the `cross_validate` function will be used to estimate the performance. Then, the accuracy of the cross-validation will be calculated using the mean function to display the overall accuracy. It shows that the cross-validation accuracy shows an accuracy of 90% which is similar to the original accuracy of the model of 89.82%. This means that the model accuracy has been validated properly.

5 References

- Afham, M. (26 September, 2019). *Twitter Sentiment Analysis using NLTK, Python*. Retrieved from Medium: <https://towardsdatascience.com/twitter-sentiment-analysis-classification-using-nltk-python-fa912578614c>
- Bajaj, A. (17 June, 2021). *Can Python understand human feelings through words? – A brief intro to NLP and VADER Sentiment Analysis*. Retrieved from AnalyticsVidhya: <https://www.analyticsvidhya.com/blog/2021/06/vader-for-sentiment-analysis/>
- Calderon, P. (10 April, 2017). *VADER Sentiment Analysis Explained*. Retrieved from Medium: <https://medium.com/@piocalderon/vader-sentiment-analysis-explained-f1c4f9101cd9>
- Desai, R. (2019). *Top 10 Python libraries for data science*. Retrieved from towardsdatascience: <https://towardsdatascience.com/top-10-python-libraries-for-data-science-cd82294ec266>
- Jurafsky, D., & Martin, J. H. (2023). *Speech and language processing*. Retrieved from Stanford: <https://web.stanford.edu/~jurafsky/slp3/>
- Malde, R. (8 June, 2020). *A Short Introduction to VADER*. Retrieved from Medium: <https://towardsdatascience.com/an-short-introduction-to-vader-3f3860208d53#:~:text=Preprocessing.&text=Unlike%20with%20some%20supervised%20methods,stemming%20Flemmatisation%20are%20not%20required.>
- Nair, A. (2021). *Leveraging n-grams to extract context from text*. Retrieved from towardsdatascience: <https://towardsdatascience.com/leveraging-n-grams-to-extract-context-from-text-bdc576b47049>
- PyPI. (4 March, 2023). *tqdm 4.65.0*. Retrieved from PyPI: <https://pypi.org/project/tqdm/>
- Singh, F. (8 December, 2020). *Sentiment Analysis Made Easy Using VADER*. Retrieved from AIM: <https://analyticsindiamag.com/sentiment-analysis-made-easy-using-vader/>