



GROUP ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT107-3-3-TXSA

TEXT ANALYTICS AND SENTIMENT ANALYSIS

UC3F2011CS(DA)_IS

HAND OUT DATE: 15 DECEMBER 2020

HAND IN DATE: 26 FEBRUARY 2021

WEIGHTAGE: 25%

INSTRUCTIONS TO CANDIDATES:

- 1 Submit your assignment at the administrative counter.**
- 2 Students are advised to underpin their answers with the use of references (cited using the Harvard Name System of Referencing).**
- 3 Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld.**
- 4 Cases of plagiarism will be penalized.**
- 5 The assignment should be bound in an appropriate style (comb bound or stapled).**
- 6 Where the assignment should be submitted in both hardcopy and softcopy, the softcopy of the written assignment and source code (where appropriate) should be on a CD in an envelope / CD cover and attached to the hardcopy.**
- 7 You must obtain 50% overall to pass this module.**

Table of Contents

1.0 Unigram Language Model	4
1.1 Compute the unsmoothed unigram probabilities manually and tabulate the respective values for all the given tokens.	5
1.2 Compute the smoothed unigram probabilities manually using the Laplace smoothing technique and tabulate the respective values for all the given tokens.....	5
1.3 Implement unsmoothed and smoothed unigram language models in python and report the respective probability values	6
2.0 Bigram Language Model.....	8
2.1 Compute the unsmoothed bigram probabilities manually and tabulate the respective values for all the given tokens.	8
2.2 Compute the smoothed unigram probabilities manually using the Laplace smoothing technique and tabulate the respective values for all the given tokens.....	11
2.3 Implement unsmoothed and smoothed bigram language models in python and report the respective probability values.	14
3.0 Sentence Probability	16
3.1 Compute manually the unigram probabilities for all the given sentences separately.	16
3.2 Compute manually the bigram probabilities for all the given sentences separately.	17
3.3 Justify which language model is more suitable to calculate the sentence probabilities.....	18
3.4 Implement and report the respective sentence probabilities in python using both unigram and bigram language models.....	19
4.0 Supervised Text Classification.....	21
4.1 Predict the sentiments (consider 3 levels) for each review found in the data set using the Rule-Based Unsupervised Technique (a library named text blob). Implement using suitable python codes and report the portion of the results.	21
4.2 Export the resulting data set along with the predicted sentiments to a .csv format using python and report the relevant code used for this operation. (<i>Note: The resulting data set must be submitted via the given submission link in MOODLE</i>).	22
4.3 Build a supervised sentiment classification model using <i>Naïve Bayes Classifier</i> from NLTK and report the following model performance measures:	23

4.4 Use the data set exported in the above task 2, and perform the supervised text classification on the sentiment using *SAS Text Miner* and print the following model performance measures: ... 25

5.0 References.....30


1.0 Unigram Language Model

	He	Read	A	Book	I	Different	My	Mulan
Count	2	3	3	3	1	1	1	1
Unsmoothed	$\frac{2}{15}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{15}$	$\frac{1}{15}$	$\frac{1}{15}$	$\frac{1}{15}$
Smoothed - Laplace	$\frac{1}{8}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{12}$

Table 1 Count, Unsmoothed Unigram probability, Smoothed Unigram Probability

The n-gram stands for the number of lengths that is identified for the language model, Unigram represents only a single word, for instance “He” is considered to be unigram (Das, 2020). The unigram language model are consisted of independent word probability and it is multiplied to calculate the whole sentence probability (Nguyen, 2020). The formula for calculating probability for individual tokens are as shown below.

$$P_{\text{train}}(\text{dream} | \text{i have a}) = P_{\text{train}}(\text{dream}) = \frac{n_{\text{train}}(\text{dream})}{N_{\text{train}}}$$



total number of words in training text

Figure 1.1 Formula for Probability of Words

Count of Words

C(He): 2 C(read): 3 C(a): 3 C(book):3

C(I) : 1 C(different):1 C(my):1 C(mulan):1

Total count of words in the training text (N): 2 + 3 + 3 + 3 + 1 + 1 + 1 + 1 = 15

1.1 Compute the unsmoothed unigram probabilities manually and tabulate the respective values for all the given tokens.

Probability of Each Words

$$P(\text{He}): \frac{2}{15} \quad P(\text{read}): \frac{3}{15} = \frac{1}{5} \quad P(\text{a}): \frac{3}{15} = \frac{1}{5} \quad P(\text{book}): \frac{3}{15} = \frac{1}{5}$$

$$P(\text{I}): \frac{1}{15} \quad P(\text{different}): \frac{1}{15} \quad P(\text{my}): \frac{1}{15} \quad P(\text{mulan}): \frac{1}{15}$$

	He	Read	A	Book	I	Different	My	Mulan
Count	2	3	3	3	1	1	1	1
Unsmoothed	$\frac{2}{15}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{15}$	$\frac{1}{15}$	$\frac{1}{15}$	$\frac{1}{15}$

Table 1.1.1 Table of probability for unsmoothed language model

1.2 Compute the smoothed unigram probabilities manually using the Laplace smoothing technique and tabulate the respective values for all the given tokens.

$$\frac{C(w)+1}{\sum C(W)+V+1}$$

Figure 1.2.1 Formula for Laplace Smoothing

Laplace smoothing is a simple smoothing method that ensures that probabilities are never zero (Krishan, 2017).

V = Total number of unique words = 8

Probability of Each Words

$$P(\text{He}): \frac{2+1}{15+8+1} = \frac{3}{24} = \frac{1}{8} = 0.125$$

$$P(\text{read}): \frac{3+1}{15+8+1} = \frac{4}{24} = \frac{1}{6} = 0.167$$

$$P(\text{a}): \frac{3+1}{15+8+1} = \frac{4}{24} = \frac{1}{6} = 0.167$$

$$P(\text{book}): \frac{3+1}{15+8+1} = \frac{4}{24} = \frac{1}{6} = 0.167$$

$$P(\text{I}): \frac{1+1}{15+8+1} = \frac{2}{24} = \frac{1}{12} = 0.083$$

$$P(\text{different}): \frac{1+1}{15+8+1} = \frac{2}{24} = \frac{1}{12} = 0.083$$

$$P(\text{my}): \frac{1+1}{15+8+1} = \frac{2}{24} = \frac{1}{12} = 0.083$$

$$P(\text{mulan}): \frac{1+1}{15+8+1} = \frac{2}{24} = \frac{1}{12} = 0.083$$

	He	Read	A	Book	I	Different	My	Mulan
Count	2	3	3	3	1	1	1	1
Smoothed - Laplace	$\frac{1}{8}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{12}$

Table 1.2.2 Table of probability for smoothed language model

1.3 Implement unsmoothed and smoothed unigram language models in python and report the respective probability values

```
#read data form file
group_data = open("C:/Users/jared/Desktop/Text Corpus.txt", "r")
txt_data = group_data.read()

stripped1_text = ""

#remove <s> and </s>
txt_data = txt_data.replace("<s>", "")
txt_data = txt_data.replace("</s>", "")
for i in txt_data:
    stripped1_text += i.strip("\n")

tokens = nltk.tokenize.word_tokenize(stripped1_text.lower())
tokens = list(everygrams(tokens, max_len=1))
n = 1
train_data, padded_sents = padded_everygram_pipeline(n, tokens)
model = MLE(n)
model.fit(train_data, padded_sents)

print("Unigram")
print("P(he)      : ", model.score('he'))
print("P(read)     : ", model.score('read'))
print("P(a)         : ", model.score('a'))
print("P(book)      : ", model.score('book'))
print("P(i)         : ", model.score('i'))
print("P(different) : ", model.score('different'))
print("P(my)        : ", model.score('my'))
print("P(mulan)     : ", model.score('mulan'))
```

Figure 1.3.1 Source Code for Unsmoothed Unigram Language Model

```

#read data form file
group_data = open("C:/Users/jared/Desktop/Text Corpus.txt","r")
txt_data = group_data.read()

stripped1_text = ""

#remove <s> and </s>
txt_data = txt_data.replace("<s>","")
txt_data = txt_data.replace("</s>","")
for i in txt_data:
    stripped1_text += i.strip("\n")

tokens = nltk.tokenize.word_tokenize(stripped1_text.lower())
tokens = list(everygrams(tokens, max_len=1))
n = 1
train_data, padded_sents = padded_everygram_pipeline(n, tokens)
smoothed_model = Laplace(n)
smoothed_model.fit(train_data, padded_sents)

print("Laplace Smoothing")
print("P(he)      : ",smoothed_model.score('he'))
print("P(read)    : ",smoothed_model.score('read'))
print("P(a)        : ",smoothed_model.score('a'))
print("P(book)     : ",smoothed_model.score('book'))
print("P(i)         : ",smoothed_model.score('i'))
print("P(different) : ",smoothed_model.score('different'))
print("P(my)        : ",smoothed_model.score('my'))
print("P(mulan)     : ",smoothed_model.score('mulan'))

```

Figure 1.3.2 Source Code for Smoothed Unigram Language Model

Unsmoothed		Smoothed	
P(he)	: 0.1333333333333333	P(he)	: 0.125
P(read)	: 0.2	P(read)	: 0.1666666666666666
P(a)	: 0.2	P(a)	: 0.1666666666666666
P(book)	: 0.2	P(book)	: 0.1666666666666666
P(i)	: 0.0666666666666667	P(i)	: 0.0833333333333333
P(different)	: 0.0666666666666667	P(different)	: 0.0833333333333333
P(my)	: 0.0666666666666667	P(my)	: 0.0833333333333333
P(mulan)	: 0.0666666666666667	P(mulan)	: 0.0833333333333333

Figure 1.3.3 Probability of Each Token for Unsmoothed & Smoothed

Figure 1.3.1 and Figure 1.3.2 shows the source code for the construction of the unigram probability model for unsmoothed and smoothed. The program will first read the content from the text data and then remove the padding along with the line break “\n”. The padding <s> and </s> is not to be considered as token, hence it is removed. The program will then perform tokenization and it will then be trained into a unigram model. The n will be set to 1 for the training of the unigram model. For the training of unsmoothed model, MLE model will be used for the probability calculation and the Laplace model will be used for calculation of smoothed unigram model. The result of probability for each token is as shown in figure 1.3.3.

2.0 Bigram Language Model

2.1 Compute the unsmoothed bigram probabilities manually and tabulate the respective values for all the given tokens.

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

Figure 2.1.1: Unsmoothed Bigram probability formula

Figure above shows the unsmoothed bigram probability formula used to calculate the bigram probabilities. There are 12 conditions that are present in the sentences provided in text file. The conditions are manually computed as below:

$$\begin{aligned} P(He | < s >) &= \frac{C(< s >, He)}{C(< s >)} \\ &= \frac{2}{3} \end{aligned}$$

$$\begin{aligned} P(I | < s >) &= \frac{C(< s >, I)}{C(< s >)} \\ &= \frac{1}{3} \end{aligned}$$

$$\begin{aligned} P(read | He) &= \frac{C(He, read)}{C(He)} \\ &= \frac{2}{2} \end{aligned}$$

$$\begin{aligned} P(read | I) &= \frac{C(I, read)}{C(I)} \\ &= \frac{1}{1} \end{aligned}$$

$$\begin{aligned} P(a | read) &= \frac{C(read, a)}{C(read)} \\ &= \frac{3}{3} \end{aligned}$$

$$\begin{aligned} P(book | a) &= \frac{C(a, book)}{C(a)} \\ &= \frac{2}{3} \end{aligned}$$

$$P(different | a) = \frac{C(a, different)}{C(a)}$$

$$= \frac{1}{3}$$

$$P(\text{book}|\text{different}) = \frac{C(\text{different}, \text{book})}{C(\text{different})}$$

$$= \frac{1}{1}$$

$$P(</s> | \text{book}) = \frac{C(\text{book}, </s>)}{C(\text{book})}$$

$$= \frac{2}{3}$$

$$P(\text{my}|\text{book}) = \frac{C(\text{book}, \text{my})}{C(\text{book})}$$

$$= \frac{1}{3}$$

$$P(\text{Mulan}|\text{my}) = \frac{C(\text{my}, \text{Mulan})}{C(\text{my})}$$

$$= \frac{1}{1}$$

$$P(</s> | \text{Mulan}) = \frac{C(\text{Mulan}, </s>)}{C(\text{Mulan})}$$

$$= \frac{1}{1}$$

Table 2.1.2: Tabulation of bigram probability results

	<s>	He	I	read	a	different	book	my	Mulan	</s>
He	0.67	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
I	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
read	0.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
a	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
different	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
book	0.00	0.00	0.00	0.00	0.67	1.00	0.00	0.00	0.00	0.00
my	0.00	0.00	0.00	0.00	0.00	0.00	0.33	0.00	0.00	0.00
Mulan	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
</s>	0.00	0.00	0.00	0.00	0.00	0.00	0.67	0.00	1.00	0.00
<s>	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 1 shows all the results of bigram probability. In the yellow box highlighted, the probability of 'He' token given start of sentence ('<s>') is 0.67.

2.2 Compute the smoothed unigram probabilities manually using the Laplace smoothing technique and tabulate the respective values for all the given tokens.

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

Figure 2.2.1: Laplace Smoothing Formula

Figure above shows the Laplace Smoothing formula used to smoothen the bigram model. Number of unique words, $V = 10$, including $\langle s \rangle$ and $\langle /s \rangle$. Following are examples of implementing the Laplace smoothing technique on conditions present above.

$$\begin{aligned} P(He | \langle s \rangle) &= \frac{C(\langle s \rangle, He) + 1}{C(\langle s \rangle) + V + 1} \\ &= \frac{2 + 1}{3 + 10 + 1} \\ &= \frac{3}{14} \end{aligned}$$

$$\begin{aligned} P(I | \langle s \rangle) &= \frac{C(\langle s \rangle, I) + 1}{C(\langle s \rangle) + V + 1} \\ &= \frac{1 + 1}{3 + 10 + 1} \\ &= \frac{1}{7} \end{aligned}$$

$$\begin{aligned} P(read | He) &= \frac{C(He, read) + 1}{C(He) + V + 1} \\ &= \frac{2 + 1}{2 + 10 + 1} \\ &= \frac{3}{13} \end{aligned}$$

$$\begin{aligned} P(read | I) &= \frac{C(I, read) + 1}{C(I) + V + 1} \\ &= \frac{1 + 1}{1 + 10 + 1} \\ &= \frac{1}{6} \end{aligned}$$

$$\begin{aligned}
 P(a|read) &= \frac{C(read, a) + 1}{C(read) + V + 1} \\
 &= \frac{3 + 1}{3 + 10 + 1} \\
 &= \frac{2}{7}
 \end{aligned}$$

$$\begin{aligned}
 P(book|a) &= \frac{C(a, book) + 1}{C(a) + V + 1} \\
 &= \frac{2 + 1}{3 + 10 + 1} \\
 &= \frac{3}{14}
 \end{aligned}$$

$$\begin{aligned}
 P(different|a) &= \frac{C(a, different) + 1}{C(a) + V + 1} \\
 &= \frac{1 + 1}{3 + 10 + 1} \\
 &= \frac{1}{7}
 \end{aligned}$$

$$\begin{aligned}
 P(book|different) &= \frac{C(different, book) + 1}{C(different) + V + 1} \\
 &= \frac{1 + 1}{1 + 10 + 1} \\
 &= \frac{1}{6}
 \end{aligned}$$

$$\begin{aligned}
 P(</s> | book) &= \frac{C(book, </s>) + 1}{C(book) + V + 1} \\
 &= \frac{2 + 1}{3 + 10 + 1} \\
 &= \frac{3}{14}
 \end{aligned}$$

$$\begin{aligned}
 P(my|book) &= \frac{C(book, my) + 1}{C(book) + V + 1} \\
 &= \frac{1 + 1}{3 + 10 + 1} \\
 &= \frac{1}{7}
 \end{aligned}$$

$$\begin{aligned}
 P(Mulan|my) &= \frac{C(my, Mulan) + 1}{C(my) + V + 1} \\
 &= \frac{1 + 1}{1 + 10 + 1} \\
 &= \frac{1}{6}
 \end{aligned}$$

$$\begin{aligned}
 P(</s> |Mulan) &= \frac{C(Mulan, </s>) + 1}{C(Mulan) + V + 1} \\
 &= \frac{1 + 1}{1 + 10 + 1} \\
 &= \frac{1}{6}
 \end{aligned}$$

Table 2.2.1: Tabulation of bigram probability results

	<s>	He	I	read	a	different	book	my	Mulan	</s>
He	0.21	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08
I	0.14	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08
read	0.07	0.23	0.17	0.07	0.07	0.07	0.07	0.07	0.07	0.07
a	0.07	0.07	0.07	0.29	0.07	0.07	0.07	0.07	0.07	0.07
different	0.08	0.08	0.08	0.08	0.14	0.08	0.08	0.08	0.08	0.08
book	0.07	0.07	0.07	0.07	0.21	0.17	0.07	0.07	0.07	0.07
my	0.08	0.08	0.08	0.08	0.08	0.08	0.14	0.08	0.08	0.08
Mulan	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.17	0.08	0.08
</s>	0.07	0.07	0.07	0.07	0.07	0.07	0.21	0.07	0.17	0.07
<s>	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07

Table 1 above shows the results of all conditions after implementing the Laplace smoothing technique. The yellow cell selected is interpreted as the probability of ‘a’ token given ‘read’ is 0.29.

2.3 Implement unsmoothed and smoothed bigram language models in python and report the respective probability values.

```
#read data form file
group_data = open("Text Corpus.txt","r")
txt_data = group_data.read()

stripped1_text = ""

#remove <s> and </s>
txt_data = txt_data.replace("<s>","")
txt_data = txt_data.replace("</s>","")
for i in txt_data:
    stripped1_text += i.strip("\n")

tokens = nltk.tokenize.word_tokenize(stripped1_text.lower())
tokens = list(everygrams(tokens, max_len = 2))
n = 2
train_data, padded_sents = padded_everygram_pipeline(n,tokens)
model = MLE(n)
model.fit(train_data,padded_sents)

print("Unsmoothed Bigram")
print("P(He|<s>)      :",model.score('he','<s>'.split()))
print("P(I|<s>)       :",model.score('i','<s>'.split()))
print("P(read|He)      :",model.score('read','he'.split()))
print("P(read|I)       :",model.score('read','i'.split()))
print("P(a|read)        :",model.score('a','read'.split()))
print("P(book|a)        :",model.score('book','a'.split()))
print("P(different|a)    :",model.score('different','a'.split()))
print("P(book|different):",model.score('book','different'.split()))
print("P(</s>|book)     :",model.score('</s>','book'.split()))
print("P(my|book)        :",model.score('my','book'.split()))
print("P(Mulan|my)       :",model.score('mulan','my'.split()))
print("P(</s>|Mulan)     :",model.score('</s>','mulan'.split()))
```

Figure 2.3.1: Unsmoothed Bigram source code

```
#read data form file
group_data = open("Text Corpus.txt","r")
txt_data = group_data.read()

stripped1_text = ""

#remove <s> and </s>
txt_data = txt_data.replace("<s>","")
txt_data = txt_data.replace("</s>","")
for i in txt_data:
    stripped1_text += i.strip("\n")

tokens = nltk.tokenize.word_tokenize(stripped1_text.lower())
tokens = list(everygrams(tokens, max_len = 2))
n = 2
train_data, padded_sents = padded_everygram_pipeline(n,tokens)
smoothed_model = Laplace(n)
smoothed_model.fit(train_data,padded_sents)

print("Laplace Smoothed Bigram")
print("P(He|<s>)      :",smoothed_model.score('he','<s>'.split()))
print("P(I|<s>)       :",smoothed_model.score('i','<s>'.split()))
print("P(read|He)      :",smoothed_model.score('read','he'.split()))
print("P(read|I)       :",smoothed_model.score('read','i'.split()))
print("P(a|read)        :",smoothed_model.score('a','read'.split()))
print("P(book|a)        :",smoothed_model.score('book','a'.split()))
print("P(different|a)    :",smoothed_model.score('different','a'.split()))
print("P(book|different):",smoothed_model.score('book','different'.split()))
print("P(</s>|book)     :",smoothed_model.score('</s>','book'.split()))
print("P(my|book)        :",smoothed_model.score('my','book'.split()))
print("P(Mulan|my)       :",smoothed_model.score('mulan','my'.split()))
print("P(</s>|Mulan)     :",smoothed_model.score('</s>','mulan'.split()))
```

Figure 2.3.2: Laplace Smoothed Bigram source code

Unsmoothed Bigram		Lapalce Smoothed Bigram	
P(He <s>)	: 0.13793103448275862	P(He <s>)	: 0.125
P(I <s>)	: 0.06896551724137931	P(I <s>)	: 0.075
P(read He)	: 0.4	P(read He)	: 0.1875
P(read I)	: 0.3333333333333333	P(read I)	: 0.14285714285714285
P(a read)	: 0.3333333333333333	P(a read)	: 0.2
P(book a)	: 0.2222222222222222	P(book a)	: 0.15
P(different a)	: 0.1111111111111111	P(different a)	: 0.1
P(book different)	: 0.3333333333333333	P(book different)	: 0.14285714285714285
P(</s> book)	: 0.6666666666666666	P(</s> book)	: 0.35
P(my book)	: 0.1111111111111111	P(my book)	: 0.1
P(Mulan my)	: 0.3333333333333333	P(Mulan my)	: 0.14285714285714285
P(</s> Mulan)	: 1.0	P(</s> Mulan)	: 0.23076923076923078

Figure 2.3.3: Results of Bigrams probabilities

Figure 2.3.1 and 2.3.2 above shows the source code for computing unsmoothed and smoothed bigram in Python. Figure 2.3.3 above shows the probability value results obtained.

3.0 Sentence Probability

3.1 Compute manually the unigram probabilities for all the given sentences separately.

The unigram probabilities for all given sentences could be calculated using the formula as shown below:

$$P(w_1, w_2, w_3, \dots, w_{n-2}, w_{n-1}, w_n) = P(w_1) * P(w_2) * P(w_3) \dots * P(w_{n-2}) * P(w_{n-1}) * P(w_n)$$

$$P(w_i) = \frac{c(w_i) + 1}{\sum_{\tilde{w}} c(\tilde{w}) + V + 1}$$

$c(w_i)$ – count of word concerned

$c(w)$ – total number of words

V – total number of vocabulary/unique words

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

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

$$= \frac{2+1}{15+8+1} * \frac{3+1}{15+8+1} * \frac{3+1}{15+8+1} * \frac{3+1}{15+8+1}$$

$$= 0.125 * 0.1667 * 0.1667 * 0.1667$$

$$= 0.000579$$

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

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

$$= \frac{1+1}{15+8+1} * \frac{3+1}{15+8+1} * \frac{3+1}{15+8+1} * \frac{1+1}{15+8+1} * \frac{3+1}{15+8+1}$$

$$= 0.0833 * 0.1667 * 0.1667 * 0.0833 * 0.1667$$

$$= 0.00003214$$

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

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

$$= \frac{2+1}{15+8+1} * \frac{3+1}{15+8+1} * \frac{3+1}{15+8+1} * \frac{3+1}{15+8+1} * \frac{1+1}{15+8+1} * \frac{1+1}{15+8+1}$$

$$= 0.125 * 0.1667 * 0.1667 * 0.1667 * 0.0833 * 0.0833$$

$$= 0.000004018$$

3.2 Compute manually the bigram probabilities for all the given sentences separately.

The bigram probabilities for all given sentences could be calculated using the formula as shown below:

$$P(w_1, w_2, w_3, \dots, w_{n-2}, w_{n-1}, w_n) = P(w_2|w_1) * P(w_3|w_2) * P(w_4|w_3) \dots * P(w_{n-1}|w_{n-2}) * P(w_n |w_{n-1})$$

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

$c(w_i)$ – count of word concerned

$c(w)$ – total number of words

V – total number of vocabulary/unique words

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

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

$$= \frac{2+1}{3+10+1} * \frac{2+1}{2+10+1} * \frac{3+1}{3+10+1} * \frac{2+1}{3+10+1} * \frac{2+1}{3+10+1}$$

$$= 0.2143 * 0.2308 * 0.2857 * 0.2143 * 0.2143$$

$$= 0.0006487$$

$P(<s>, \text{I, read, a, different, book}, </s>)$

$$= P(\text{I} | <s>) * P(\text{read} | \text{I}) * P(\text{a} | \text{read}) * P(\text{different} | \text{a}) * P(\text{book} | \text{different}) * P(</s> | \text{book})$$

$$= \frac{1+1}{3+10+1} * \frac{1+1}{1+10+1} * \frac{3+1}{3+10+1} * \frac{1+1}{3+10+1} * \frac{1+1}{1+10+1} * \frac{2+1}{3+10+1}$$

$$= 0.1429 * 0.1667 * 0.2857 * 0.1429 * 0.1667 * 0.2143$$

$$= 0.00003474$$

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

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

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

$$= \frac{2+1}{3+10+1} * \frac{2+1}{2+10+1} * \frac{3+1}{3+10+1} * \frac{2+1}{3+10+1} * \frac{1+1}{3+10+1} * \frac{1+1}{1+10+1} * \frac{1+1}{1+10+1}$$

$$= 0.2143 * 0.2308 * 0.2857 * 0.2143 * 0.1429 * 0.1667 * 0.1667$$

$$= 0.00001203$$

3.3 Justify which language model is more suitable to calculate the sentence probabilities.

Each word in a sentence is counted as a token. The sentence probabilities of any sentence are calculated by multiplying the probability of occurrence of each token. There are two language model that could be used to calculate the sentence probabilities, which are the unigram model and bigram model. The main difference between the two models is that unigram model considers all token are independent of each other. The probability of the previous token will not affect the probability of the following token. For bigram model, the sentence probability is calculated by using conditional probability of the token based on the probability of the previous token. (Srinidhi, 2019) By using the bigram model, it could explain the relationship between two words and why is the word coming after the previous. This calculation is more accurate as it considers the probability of current token and the previous token together. Therefore, bigram model is appropriate and suitable to calculate the sentence probabilities of any given sentence.

3.4 Implement and report the respective sentence probabilities in python using both unigram and bigram language models.

The Python code to calculate the sentence probabilities using unigram language model is shown in figure 3.4.1 below:

```
pHe = smoothed_model.score('he')
pRead = smoothed_model.score('read')
pA = smoothed_model.score('a')
pBook = smoothed_model.score('book')
pI = smoothed_model.score('i')
pDifferent = smoothed_model.score('different')
pMy = smoothed_model.score('my')
pMulan = smoothed_model.score('mulan')

pHeReadABook = pHe * pRead * pA * pBook
pIReadADifferentBook = pI * pRead * pA * pDifferent * pBook
pHeReadABookMyMulan = pHe * pRead * pA * pBook * pMy * pMulan

print("-- Sentence Probabilities using Smoothed Unigram Language Model")
print("The sentence probability of 'He read a book' is", format(float(pHeReadABook), '.20f'))
print("The sentence probability of 'He read a book' is", format(float(pIReadADifferentBook), '.20f'))
print("The sentence probability of 'He read a book' is", format(float(pHeReadABookMyMulan), '.20f'))
```

Figure 3.4.1 Python Source Code for Unigram Language Model

The output obtained are shown in figure 3.4.2 below:

```
-- Sentence Probabilities using Smoothed Bigram Language Model --
The sentence probability of '<s>He read a book</s>' is 0.00064876814147951168
The sentence probability of '<s>I read a different book</s>' is 0.00003470776065528251
The sentence probability of '<s> He read a book my Mulan </s>' is 0.00001201422484221318
```

Figure 3.4.2 Sentence Probabilities of Unigram Language Model

The result obtained by calculating manually above is :

$$P(\text{He, read, a, book}) = 0.0006487$$

$$P(\text{I, read, a, different, book}) = 0.00003214$$

$$P(\text{He, read, a, book, my, mulan}) = 0.000004018$$

The calculation by Python and manually is similar up to 3 significant figures.

The Python code to calculate the sentence probabilities using bigram language model is as figure 3.4.3 below:

```
pHeS = smoothed_model.score('he' '<s>'.split())
pReadHe = smoothed_model.score('read' 'he'.split())
pARead = smoothed_model.score('a' 'read'.split())
pBookA = smoothed_model.score('book' 'a'.split())
pSBook = smoothed_model.score('</s>' 'book'.split())
pIS = smoothed_model.score('i' '<s>'.split())
pReadI = smoothed_model.score('read' 'i'.split())
pDifferentA = smoothed_model.score('different' 'a'.split())
pBookDifferent = smoothed_model.score('book' 'different'.split())
pMyBook = smoothed_model.score('my' 'book'.split())
pMulanMy = smoothed_model.score('mulan' 'my'.split())
pSMulan = smoothed_model.score('</s>' 'mulan'.split())

pSHeReadABooks = pHeS * pReadHe * pARead * pBookA * pSBook
pSIReadADifferentBooks = pIS * pReadI * pARead * pDifferentA * pBookDifferent * pSBook
pSHeReadABookMyMulanS = pHeS * pReadHe * pARead * pBookA * pMyBook * pMulanMy * pSMulan

print("-- Sentence Probabilities using Smoothed Unigram Language Model")
print("The sentence probability of 'He read a book' is", format(float(pSHeReadABooks), '.20f'))
print("The sentence probability of 'He read a book' is", format(float(pSIReadADifferentBooks), '.20f'))
print("The sentence probability of 'He read a book' is", format(float(pSHeReadABookMyMulanS), '.20f'))
```

Figure 3.4.3 Python Source Code for Bigram Language Model

The output obtained are shown in figure 3.4.4 below.

```
-- Sentence Probabilities using Smoothed Bigram Language Model --
The sentence probability of '<s>He read a book</s>' is 0.00064876814147951168
The sentence probability of '<s>I read a different book</s>' is 0.00003470776065528251
The sentence probability of '<s> He read a book my Mulan </s>' is 0.00001201422484221318
```

Figure 3.4.4 Sentence Probabilities of Bigram Language Model

The result obtained by calculating manually above is :

$P(<s>, \text{He}, \text{read}, \text{a}, \text{book}, </s>) = 0.0006487$

$P(<s>, \text{I}, \text{read}, \text{a}, \text{different}, \text{book}, </s>) = 0.00003474$

$P(<s>, \text{He}, \text{read}, \text{a}, \text{book}, \text{my}, \text{mulan}, </s>) = 0.00001203$

The calculation by Python and manually is similar up to 3 significant figures.

4.0 Supervised Text Classification

4.1 Predict the sentiments (consider 3 levels) for each review found in the data set using the Rule-Based Unsupervised Technique (a library named text blob). Implement using suitable python codes and report the portion of the results.

Pandas library will be imported to read data from a CSV file. The 3 levels of sentiments are negative, neutral, and positive. Any sentence with negative polarity value will be defined as negative sentiment, whereas positive polarity value will be defined as positive sentiment. Neutral sentence has polarity value equal to zero. The Python code are shown in figure 4.1.1 below.

```
from textblob import TextBlob
import pandas as pd

data = pd.read_csv("C:/Users/benhu/OneDrive/Desktop/TXSA/Assignment-20201215/Group Assignment Data/Musical Instruments_Reviews.csv")
data.head()
reviews = data['Reviews']

for line in reviews:
    print(line)
    sent = TextBlob(line)
    print("The polarity is: ", sent.polarity)
    if (sent.polarity == 0):
        print("The sentiment of the sentence is neutral")
    else:
        if (sent.polarity > 0):
            print("The sentiment of the sentence is positive")
        else:
            print("The sentiment of the sentence is negative")
    print()
```

Figure 4.1.1 Python Source Code for Question 4.1

The results obtained are shown in figure 4.1.2 below.

```
C:\Users\benhu\anaconda3\python.exe C:/Users/benhu/Downloads/Assignment_Group_TXSA.py
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 might as well buy it, they honestly work the same despite their pricing,
The polarity is: 0.25
The sentiment of the sentence is positive

The product does exactly as it should and is quite affordable.I did not realized it was double screened until it arrived, so it was even
better than I had expected.As an added bonus, one of the screens carries a small hint of the smell of an old grape candy I used to buy,
so for reminiscent's sake, I cannot stop putting the pop filter next to my nose and smelling it after recording. :DIf you needed a pop
filter, this will work just as well as the expensive ones, and it may even come with a pleasing aroma like mine did!Buy this product! :)
The polarity is: 0.05277777777777778
The sentiment of the sentence is 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 frequencies. The double cloth filter blocks the pops and lets the voice through
with no coloration. The metal clamp mount attaches to the mike stand secure enough to keep it attached. The goose neck needs a little
coaxing to stay where you put it.
The polarity is: 0.1675
The sentiment of the sentence is positive

Nice windscreens 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 sagging.
The polarity is: 0.2
The sentiment of the sentence is 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.
The polarity is: 0.8
The sentiment of the sentence is 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 no harm done, still works great!
The polarity is: 0.33888888888888885
The sentiment of the sentence is positive
```

Figure 4.1.2 Results of Question 4.1

4.2 Export the resulting data set along with the predicted sentiments to a .csv format using python and report the relevant code used for this operation. (Note: The resulting data set must be submitted via the given submission link in MOODLE).

Pandas library will be imported to read data from a CSV file. A new list will be created to store the predicted polarity and sentiment. Once the sentiment and polarity are predicted, both the value will be added to the list created. Two new columns in the CSV will also be created to store both the polarity and sentiment value. Once all the values are stored, the CSV will be saved and closed. The Python code is shown in figure 4.2.1 below.

```
from textblob import TextBlob
import pandas as pd

csv_file = ("C:/Users/benhu/OneDrive/Desktop/TXSA/Assignment-20201215/Group_Assignment_Data/Musical_Instruments_Reviews.csv")
data = pd.read_csv(csv_file)
reviews = data['Reviews']

#Create list to store polarity and sentiment
polarityRow = []
sentimentRow = []

for line in reviews:
    print(line)
    sent = TextBlob(line)
    sentiment = ""
    print("The polarity is: ", sent.polarity)
    if (sent.polarity == 0):
        print("The sentiment of the sentence is neutral")
        sentiment = "Neutral"
    else:
        if (sent.polarity > -0):
            print("The sentiment of the sentence is positive")
            sentiment = "Positive"
        else:
            print("The sentiment of the sentence is negative")
            sentiment = "Negative"
    print()

    # Add polarity and sentiment into the list created
    polarityRow.append(sent.polarity)
    sentimentRow.append(sentiment)

#Create new column to store polarity and sentiment
data["Polarity"] = polarityRow
data["Sentiment"] = sentimentRow

#Save the changes in CSV file
data.to_csv(csv_file, index=False)
```

Figure 4.2.1 Python Source Code for Question 4.2

Figure 4.2.2 below shows the new column “Polarity” and “Sentiment” created in the Musical_Instruments_Reviews CSV file.

Reviews	Polarity	Sentiment
Not much to write about here, but it does exactly what it's supposed to. filters out the pop sounds. now my	0.25	Positive
The product does exactly as it should and is quite affordable.I did not realized it was double screened until	0.052777778	Positive
The primary job of this device is to block the breath that would otherwise produce a popping sound, while i	0.1675	Positive
Nice windscreen protects my MXL mic and prevents pops. Only thing is that the gooseneck is only marginal	0.2	Positive
This pop filter is great. It looks and performs like a studio filter. If you're recording vocals this will eliminat	0.8	Positive
So good that I bought another one. Love the heavy cord and gold connectors. Bass sounds great. I just lear	0.338888889	Positive
I have used monster cables for years, and with good reason. The lifetime warranty is worth the price alone.	0.333333333	Positive
I now use this cable to run from the output of my pedal chain to the input of my Fender Amp. After I bough	0.006540404	Positive
Perfect for my Epiphone Sheraton II. Monster cables are well constructed. I have several and never had an	0.5	Positive
Monster makes the best cables and a lifetime warranty doesnt hurt either. This isnt their top of the line ser	0.33	Positive
Monster makes a wide array of cables, including some that are very high end. I initially purchased a pair of M	0.107653333	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 rec	0.4	Positive
If you are not use to using a large sustaining pedal while playing the piano, it may appear little awkward.	-0.191071429	Negative
I love it, I used this for my Yamaha ypt-230 and it works great, I would recommend it to anyone	0.65	Positive
I bought this to use in my home studio to control my midi keyboard. It does just what I wanted it to do.	0	Neutral
I bought this to use with my keyboard. I wasn't really aware that there were other options for keyboard pec	0.15	Positive
This Fender cable is the perfect length for me! Sometimes I find it a bit too long but I don't mind. The build	0.316666667	Positive
wanted it just on looks alone...It is a nice looking cord... I know it will perform...as for Sam Ash ...this cord w	0.525	Positive
I've been using these cables for more than 4 months and they are holding up pretty well. For years I used s	0.132730159	Positive

Figure 4.2.2 Result of Question 4.2

4.3 Build a supervised sentiment classification model using *Naïve Bayes Classifier* from NLTK and report the following model performance measures:

- Accuracy
- Precision
- Recall
- F1 Score

The python codes must be neat with clear output. Provide relevant and necessary comments in the code to explain the purpose of the code snippet.

Deliverable:

- Complete & running Python code.
- Output stating the above FOUR (04) performance measures.

```

#Feature extractor function
def wordFeatures(word):
    stopset = list(set(cp.stopwords.words('english')))
    return dict([(word,True) for word in word_tokenize(word) if word not in stop

#Read file
file = pd.read_csv("Musical_Instruments_Reviews.csv")
#Process data so that 'Review' column and 'Sentiment' column is taken
data = [(wordFeatures(row['Reviews']),row['Sentiment'])for n, row in file.iterro

#Shuffling the data
random.shuffle(data)

#Splitting data into train and test datasets
split = round(len(data)*0.7)
train_set,test_set = data[:split], data[split:]

#Creating the NaiveBayes Classifier
classifier = nltk.NaiveBayesClassifier.train(train_set)

#two testsets, one is the original test set
#the other is the one to be predicted by classifier
refsets = collections.defaultdict(set)
testsets = collections.defaultdict(set)

#adding the predictions of model into testset
for i, (feats, label) in enumerate(test_set):
    refsets[label].add(i)
    observed = classifier.classify(feats)
    testsets[observed].add(i)

# Show the accuracy of the Naive Bayes Classifier
print("Accuracy of Naive Bayes Classifier:", nltk.classify.accuracy(classifier,
# Show average performance measures of the model
print("\nAverage of Performance measures:\n")
print("Precision : ", (nltk.scores.precision(refsets['Negative'], testsets['Nega
print("Recall : ", (nltk.scores.recall(refsets['Negative'], testsets['Negativ
print("F1-score : ", (nltk.scores.f_measure(refsets['Negative'], testsets['Nega

# Show performance measures for each sentiment
print("\nPerformance measures for each sentiment:\n")
print("\t\tprecision\trecall\tF1-score")
print("Negative\t", "%.2f" % nltk.scores.precision(refsets['Negative'], testset
print("Neutral\t\t", "%.2f" % nltk.scores.precision(refsets['Neutral'], testset
print("Positive\t", "%.2f" % nltk.scores.precision(refsets['Positive'], testset

```

Figure 4.3.1: Source code for building sentiment classification

```

Accuracy of Naive Bayes Classifier: 0.9070221066319896

Performance measures for each sentiment:

                precision      recall      F1-score
Negative         0.44          0.05          0.09
Neutral          0.07          0.04          0.05
Positive         0.92          0.99          0.95

Average of Performance measures:

Precision : 0.47529956967891424
Recall : 0.3590088379522598
F1-score : 0.36484551570492446

```

Figure 4.3.2: Results of performance measure of model

Figure 4.3.1 above shows the source code for building a sentiment classification model using Naïve Bayes Classifier from NLTK. Figure 4.3.2 above shows the accuracy of the model and the performance measures of precision, recall and f1-score according to each sentiment. The high accuracy of the model shows that the model is accurate 90.7 % of the time. The low precision, recall, and f1-score for negative and neutral sentiments may indicate a dataset that is biased to positive sentiments. More data that includes neutral and negative sentiments should be added in. The performance measures show that the model is good at indicating positive sentiments but may not perform equally well with negative and neutral sentiments. The average precision, recall and f1-score of the sentiment classification shows that the model is not doing as well to predict sentiments even though its accuracy is at 90.7%.

4.4 Use the data set exported in the above task 2, and perform the supervised text classification on the sentiment using SAS Text Miner and print the following model performance measures:

- a. Accuracy
- b. Precision
- c. Recall
- d. F1 Score

The process flow diagram must be neat and given in the report. Provide relevant and necessary explanations with the suitable output (screenshots) to support the answers.

Deliverable:

- a. Process flow diagram (.xml)
- b. Output stating the above FOUR (04) performance measures

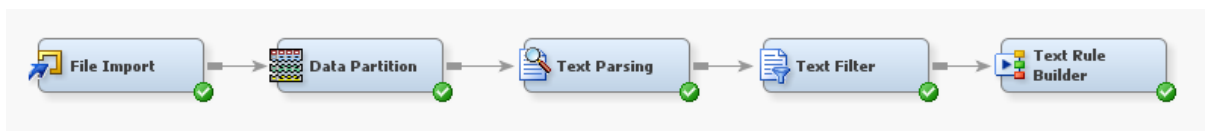


Figure 4.4.1 Process Flow Diagram

Figure 4.4.1 shows the process flow diagram that is generated for performing the supervised text classification. The process is consisted of File Import, Data Partition, Text Parsing, Text Filter and Text Role Builder. The File Import node is used for loading in data that is exported in task 2 to SAS Enterprise Miner. Furthermore, target variable “sentiment” has also been selected for the project. Upon completion of those steps, the data is then partition into training and validation data using the Data Partition node, the data is partitioned into 70% training and 30% validation. In addition to that, Text Parsing is used, and no modification has been conducted. Afterwards, Text Filter node is used for speeling check to maximize the accuracy of the model. Text Rule Builder will then provide the results of the model.

Accuracy

Fit Statistics	Statistics Label	Train	Validation
ASE_	Average Squar...	0.053512	0.05815
DIV_	Divisor for ASE	21528	9234
MAX_	Maximum Abs...	0.819218	0.859037
NOBS_	Sum of Freque...	7176	3078
RASE_	Root Average ...	0.231326	0.241144
SSE_	Sum of Squar...	1151.997	536.9609
DISF_	Frequency of ...	7176	3078
MISC_	Misclassificati...	0.159002	0.181287
WRONG	Number of Wr...	1141	558

Figure 4.4.2 Accuracy of the Model

Figure 4.4.2 shows the overall accuracy of the model. Misclassification rate is served as the inaccurate results of the model, it can be used for calculating the overall accuracy of the model. In this project, the misclassification rate is 0.181287 which indicates that the inaccurate results are 18.13% and the accuracy can be calculated as $100\% - 18.13\% = 81.87\%$.

Precision

Target Value	Rule #	Rule	Precision ▼
POSITIVE		1 excellent	100.0%
POSITIVE		2 sound & year	100.0%
POSITIVE		3 sound & easy	100.0%
POSITIVE		4 price & ~hate & rec...	100.0%
POSITIVE		5 impress	100.0%
POSITIVE		6 sound & good	100.0%
POSITIVE		7 smooth	100.0%
POSITIVE		12 good & ~wrong & ~...	99.75%
POSITIVE		11 good quality	99.73%
POSITIVE		10 quick	99.72%
POSITIVE		9 quickly	99.71%
POSITIVE		8 great & ~bad	99.70%
POSITIVE		13 sound	99.69%
POSITIVE		15 sound & ~terrible ...	99.60%
POSITIVE		14 great	99.59%
POSITIVE		16 nice & ~useless & ...	99.47%
POSITIVE		17 great	99.35%
POSITIVE		18 perfect	99.29%
POSITIVE		19 best & ~crap	99.27%
POSITIVE		20 perfectly	99.21%
POSITIVE		21 better	99.16%
POSITIVE		22 able	99.15%
POSITIVE		23 nice	99.11%
POSITIVE		26 cool	99.11%
POSITIVE		25 pretty	99.11%
POSITIVE		24 easy	99.11%

Figure 4.4.3 Precision of the Model

POSITIVE	24easy	99.11%
POSITIVE	27awesome	99.08%
POSITIVE	28value	99.07%
POSITIVE	29protect	99.05%
POSITIVE	31electric	98.98%
POSITIVE	30love & ~bad & ~hard	98.97%
POSITIVE	32pretty	98.95%
POSITIVE	33exactly	98.92%
POSITIVE	34good & ~hard & ~c...	98.56%
POSITIVE	35perfect	98.53%
POSITIVE	36happy	98.46%
POSITIVE	38cheap	98.40%
POSITIVE	37good & ~bad	98.39%
POSITIVE	39easily	98.33%
POSITIVE	40easy	98.07%
NEGATIVE	42chew	84.62%
NEGATIVE	41hate & ~good	81.82%
NEGATIVE	44guardian	80.00%
NEGATIVE	43wrong & ~sound & ...	78.57%
NEGATIVE	45waste	78.43%
NEGATIVE	46wrong & ~good & ~...	67.47%
NEGATIVE	50intermittent	58.72%
NEGATIVE	52terrible	58.38%
NEGATIVE	49standard guitar	58.33%
NEGATIVE	48nifty	57.93%
NEGATIVE	51useless	57.67%
NEGATIVE	54bad	57.56%
NEGATIVE	47hard & ~nice & ~pri...	57.50%
NEGATIVE	55base	57.31%
NEGATIVE	53bad & ~good	57.14%
NEUTRAL	56tester	40.00%

Figure 4.4.4 Precision of the Model

Figure 4.4.3 and Figure 4.4.4 shows the precision of the model. The results show higher precision of the positive results and neutral have the lowest in terms of precision and the rest covered by negative.

Recall

Target Value	Rule #	Rule	Precision ▼	Recall
POSITIVE		1excellent	100.0%	4.42%
POSITIVE		2sound & year	100.0%	7.22%
POSITIVE		3sound & easy	100.0%	9.24%
POSITIVE		4price & ~hate & rec...	100.0%	11.15%
POSITIVE		5impress	100.0%	12.66%
POSITIVE		6sound & good	100.0%	14.18%
POSITIVE		7smooth	100.0%	15.50%
POSITIVE		12good & ~wrong & ~...	99.75%	36.04%
POSITIVE		11good quality	99.73%	33.69%
POSITIVE		10quick	99.72%	32.90%
POSITIVE		9quickly	99.71%	31.88%
POSITIVE		8great & ~bad	99.70%	30.22%
POSITIVE		13sound	99.69%	38.98%
POSITIVE		15sound & ~terrible ...	99.60%	44.98%
POSITIVE		14great	99.59%	43.96%
POSITIVE		16nice & ~useless & ...	99.47%	51.45%
POSITIVE		17great	99.35%	55.69%
POSITIVE		18perfect	99.29%	57.77%
POSITIVE		19best & ~crap	99.27%	59.61%
POSITIVE		20perfectly	99.21%	61.22%
POSITIVE		21better	99.16%	62.71%
POSITIVE		22able	99.15%	63.61%
POSITIVE		23nice	99.11%	64.66%
POSITIVE		26cool	99.11%	66.30%
POSITIVE		25pretty	99.11%	65.99%
POSITIVE		24easy	99.11%	65.66%
POSITIVE		27awesome	99.08%	67.13%
POSITIVE		28value	99.07%	67.68%
POSITIVE		29protect	99.05%	68.20%
POSITIVE		31electric	98.98%	70.89%
POSITIVE		30love & ~bad & ~hard	98.97%	70.38%
POSITIVE		32pretty	98.95%	71.80%
POSITIVE		33exactly	98.92%	72.48%
POSITIVE		34good & ~hard & ~c...	98.56%	79.82%
POSITIVE		35perfect	98.53%	80.41%
POSITIVE		36happy	98.46%	81.46%
POSITIVE		38cheap	98.40%	83.24%
POSITIVE		37good & ~bad	98.39%	82.83%
POSITIVE		39easily	98.33%	84.00%
POSITIVE		40easy	98.07%	85.88%
NEGATIVE		42chew	84.62%	4.67%
NEGATIVE		41hate & ~good	81.82%	3.82%
NEGATIVE		44guardian	80.00%	7.64%
NEGATIVE		43wrong & ~sound & ...	78.57%	7.01%
NEGATIVE		45waste	78.43%	8.49%
NEGATIVE		46wrong & ~good & ~...	67.47%	11.89%
NEGATIVE		50intermittent	58.72%	21.44%
NEGATIVE		52terrible	58.38%	24.42%
NEGATIVE		49standard guitar	58.33%	20.81%
NEGATIVE		48nifty	57.93%	20.17%
NEGATIVE		51useless	57.67%	23.14%
NEGATIVE		54bad	57.56%	29.09%
NEGATIVE		47hard & ~nice & ~pri...	57.50%	19.53%
NEGATIVE		55base	57.31%	30.79%
NEGATIVE		53bad & ~good	57.14%	28.03%
NEUTRAL		56tester	40.00%	1.61%

Figure 4.4.5 Recall of the model

Figure 4.4.5 shows all the result of recall for the model and the use for recall is to identify the true positive numbers. Thus, it can also be used to indicate the accuracy of the model. Few data will reflect higher precision with low recall, this indicates the positive results as predicted result is the same as the training data.

F1 Score

Target Value	Rule #	Rule	Precision ▼	Recall	F1 score
POSITIVE		1excellent	100.0%	4.42%	8.47%
POSITIVE		2sound & year	100.0%	7.22%	13.46%
POSITIVE		3sound & easy	100.0%	9.24%	16.91%
POSITIVE		4price & ~hate & rec...	100.0%	11.15%	20.07%
POSITIVE		5impress	100.0%	12.66%	22.47%
POSITIVE		6sound & good	100.0%	14.18%	24.83%
POSITIVE		7smooth	100.0%	15.50%	26.84%
POSITIVE		12good & ~wrong & ~...	99.75%	36.04%	52.95%
POSITIVE		11good quality	99.73%	33.69%	50.36%
POSITIVE		10quick	99.72%	32.90%	49.47%
POSITIVE		9quickly	99.71%	31.88%	48.31%
POSITIVE		8great & ~bad	99.70%	30.22%	46.39%
POSITIVE		13sound	99.69%	38.98%	56.04%
POSITIVE		15sound & ~terrible ...	99.60%	44.98%	61.97%
POSITIVE		14great	99.59%	43.96%	61.00%
POSITIVE		16nice & ~useless & ...	99.47%	51.45%	67.82%
POSITIVE		17great	99.35%	55.69%	71.37%
POSITIVE		18perfect	99.29%	57.77%	73.05%
POSITIVE		19best & ~crap	99.27%	59.61%	74.49%
POSITIVE		20perfectly	99.21%	61.22%	75.72%
POSITIVE		21better	99.16%	62.71%	76.83%
POSITIVE		22able	99.15%	63.61%	77.50%
POSITIVE		23nice	99.11%	64.66%	78.26%
POSITIVE		26cool	99.11%	66.30%	79.45%
POSITIVE		25pretty	99.11%	65.99%	79.23%
POSITIVE		24easy	99.11%	65.66%	78.99%
POSITIVE		27awesome	99.08%	67.13%	80.04%
POSITIVE		28value	99.07%	67.68%	80.42%
POSITIVE		29protect	99.05%	68.20%	80.78%
POSITIVE		31electric	98.98%	70.89%	82.61%
POSITIVE		30love & ~bad & ~hard	98.97%	70.38%	82.27%
POSITIVE		32pretty	98.95%	71.80%	83.22%
POSITIVE		33exactly	98.92%	72.48%	83.66%
POSITIVE		34good & ~hard & ~c...	98.56%	79.82%	88.20%
POSITIVE		35perfect	98.53%	80.41%	88.55%
POSITIVE		36happy	98.46%	81.46%	89.16%
POSITIVE		38cheap	98.40%	83.24%	90.19%
POSITIVE		37good & ~bad	98.39%	82.83%	89.94%
POSITIVE		39easily	98.33%	84.00%	90.60%
POSITIVE		40easy	98.07%	85.88%	91.57%
NEGATIVE		42chew	84.62%	4.67%	8.85%
NEGATIVE		41hate & ~good	81.82%	3.82%	7.30%
NEGATIVE		44guardian	80.00%	7.64%	13.95%
NEGATIVE		43wrong & ~sound & ...	78.57%	7.01%	12.87%
NEGATIVE		45waste	78.43%	8.49%	15.33%
NEGATIVE		46wrong & ~good & ~...	67.47%	11.89%	20.22%
NEGATIVE		50intermittent	58.72%	21.44%	31.42%
NEGATIVE		52terrible	58.38%	24.42%	34.43%
NEGATIVE		49standard guitar	58.33%	20.81%	30.67%
NEGATIVE		48nifty	57.93%	20.17%	29.92%
NEGATIVE		51useless	57.67%	23.14%	33.03%
NEGATIVE		54bad	57.56%	29.09%	38.65%
NEGATIVE		47hard & ~nice & ~pri...	57.50%	19.53%	29.16%
NEGATIVE		55base	57.31%	30.79%	40.06%
NEGATIVE		53bad & ~good	57.14%	28.03%	37.61%
NEUTRAL		56tester	40.00%	1.61%	3.10%

Figure 4.4.6 F1 Score of the model

F1 Score is the weighted average of both precision and recall values, it is often used to find either there is uneven class distribution or to find the balance between precision and the recall of the model (Shuang, 2018).

5.0 References

Das, A., 2020. *Complete Guide on Language Modelling: Unigram Using Python*. [Online]
Available at: <https://analyticsindiamag.com/complete-guide-on-language-modelling-unigram-using-python/>

[Accessed 2 March 2021].

Krishan, 2017. *Tag: Laplace smoothing*. [Online]
Available at: <https://iksinc.online/tag/laplace-smoothing/>

[Accessed 2 March 2021].

Nguyen, K., 2020. *N-gram language models*. [Online]
Available at: <https://medium.com/mti-technology/n-gram-language-model-b7c2fc322799>

[Accessed 2 March 2021].

Srinidhi, S., 2019. *Understanding Word N-grams and N-gram Probability in Natural Language Processing*. [Online]
Available at: <https://towardsdatascience.com/understanding-word-n-grams-and-n-gram-probability-in-natural-language-processing-9d9eef0fa058>

[Accessed 6 March 2021].