STEAM PLM PQL CLASSIFICATION

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

PATHI PREETHAM REDDY (18BCE0482)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

May, 2022

DECLARATION

I hereby declare that the thesis entitled "**Steam PLM PQL Classification**" submitted by me, for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a record of bonafide work carried out by me under the supervision of Mr. **Jason Selvin**.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 27-05-2022 Preetham Reddy Pathi

INTERNSHIP CERTIFICATE BY THE EXTERNAL GUIDE

This is to certify that the project report entitled "**Steam PLM PQL Classification**" submitted by Preetham Reddy Pathi (18BCE0482) to Vellore Institute of Technology, Vellore in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a record of bonafide work carried out by him under my guidance. The project fulfils the requirements as per the regulations of this Institute and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institutes or universities.

[illegible]

(Signature of the External Supervisor)

Jason Selvin
Sr Director - Data & Analytics
GE Steam Power
Email: jason.selvin@ge.com
Phone: +91 98866 03482

ABSTRACT

General Electric is a multinational conglomerate which is currently in Aviation, Healthcare and Power businesses. GE Steam Power is concerned with producing and servicing steam turbines in various thermal power stations all over the world. I was part of the Data Analytics team in GE Steam Power during my internship period as a Data & Analytics Intern where I worked on a Machine Learning project titled "**Steam PLM PQL Classification**". My role in the project was titled 'Model Tester' which meant I had to develop and test different models and do quite a lot of pre-processing for the project.

The project was to classify the parts into different categories with a given description. For any natural language processing problem, there is quite a lot of pre-processing to be done on the text before we can apply a machine learning model on it. The technologies or libraries that I've used in this project include Scikit-learn, NumPy, Pandas, TF-IDF Vectorizer, Count Vectorizer, Naïve Bayes, Random Forest Classifier, Decision Tree Classifier, Logistic Regression and Matplotlib. Some of the metrics that I used to test the model are accuracy, confusion matrix, R2 Score, probability.

The project is currently in its final stages and the work on the ML model is almost finished. If the project gives the accuracy and confidence that is acceptable by the engineering team it can also be deployed to the real-world where it can have a meaningful impact on the time spent by an engineer to classify these parts manually. And it also adds a level of precision as sometimes human error does happen. We're planning to only use the predictions which it predicts with good confidence since we can get the ones classified with less confidence double checked by an engineer.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to **Jason Selvin,**

Sr Director, Data & Analytics, GE Steam Power, for his/her constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with him / her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Data Analytics.

I would like to express my gratitude to Dr. G. Viswanathan (Chancellor, VIT University, Vellore), Sankar Viswanathan (Vice-President, VIT University, Vellore), Sekar Viswanathan (Vice-President, VIT University, Vellore), GV Selvam (Vice-President, VIT University, Vellore), Rambabu Kodali (Vice-Chancellor, VIT University, Vellore), Dr. S Narayanan (Pro-Vice President, VIT University, Vellore), and Dr Ramesh Babu K (Dean, School of Computer Science and Engineering), for providing with an environment to work in and for his inspiration during the tenure of the course.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. Last, but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Vellore

Date: 27-05-2022 Preetham Reddy Pathi

DISCLAIMER

All the graphs and data that is used in this report are randomly generated to protect the Intellectual Property of GE Steam Power. All the code is re-written for purely demonstration purposes with randomly generated data. None of the GE data is used for demonstration purposes in this report.

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF ABBREVATIONS

Abbreviation	Full Form
PLM	Product Lifecycle Management
PQL	Product Qualification Level
GE	General Electric
TF-IDF	Term Frequency – Inverse Document Frequency
ML	Machine Learning
PaaS	Platform as a Service
MES	Manufacturing Execution System
ERP	Enterprise Resource Planning
eBOM	Engineering Bill of Materials
EDA	Exploratory Data Analysis
NLP	Natural Language Processing
NLTK	Natural Language Tool Kit
POS	Part of Speech
NER	Name Entity Recognition
TF	Term Frequency
DF	Document Frequency
IDF	Inverse Document Frequency
AI	Artificial Intelligence
SVM	Support Vector Machines
MNB	Multinomial Naïve Bayes

CART	Classification and Regression Tree
MSE	Mean Squared Error
D&A	Data and Analytics
BERT	Bidirectional Encoder Representations from Transformers

Chapter 1

1. Introduction

This chapter serves as the beginning of this thesis and constitutes the concepts related to the rest of this thesis focused on text classification. In the later part of the thesis, I will discuss the machine learning models used in the internship, discuss the technologies used in my internship to work on the projects and discuss the problem statements and impact of the project in detail.

1.1 Objective

The objective of this project is to classify every part to a "Part Qualification Level (PQL)". Product qualification is a process done on mass produced products to evaluate it based on the physical testing on the manufactured prototype. The purpose is to verify whether it has met or exceeded its intended quality and reliability requirements. After virtual and product qualification, the products are mass produced. During and after the manufacturing process, the products can be inspected and tested to evaluate their quality and defected parts can be screened out. If the company that is buying the parts specifies the seller with the level of qualification they are expecting, then the seller can price and manufacture accordingly.

For example, if you're mass producing a washing machine. And we need to order screws for making them and let's consider that there are three product qualification levels. Now, for things like the packaging box, you don't need screws with very high qualification level because if some screws are not up to the normal standards there is no risk. Whereas, for putting together the body of the washing machine, we need to use better quality assured screws because maybe if less 20% of the screws are defective its fine. And while making the motor for the washing machine, they have to use very high-quality assured screws since the whole machine may fail with failure of one screw. So here we can see that for a product a company may need multiple levels of product qualification for different parts.

1.2 Motivation

When a product is to be manufactured, the team will decide the set of parts that will be needed to build the product. So, for every project an engineer must sit for multiple days on the parts and manually classify them for their PQL. This can result in several delays. Since most of the parts are previously seen parts and not unseen parts, it is a waste of time for the engineer and the team as it can be done using an ML model. There can be thousands of parts that are needed for a single product and since the engineer must manually classify them, we can expect some human error.

This project can help not only speed up the process but also save the time and effort of an engineer. We can expect significant cost and time savings with the usage of this model. The development cost of this model is minimal when compared with the cost being spent on manual PQL classification.

1.3Background

The PQL classification is a part of the PLM (Product Lifecycle Management) where the product is managed from the planning stage till the product rolls off the production line.

- **Product Lifecycle Management (PLM):**

Product lifecycle management (PLM) refers to the handling of a good as it moves through the typical stages of its product life: development and introduction, growth, maturity/stability, and decline. This handling involves both the manufacturing of the good and the marketing of it. The concept of product life cycle helps inform business decision-making, from pricing and promotion to expansion or cost-cutting.

At the most fundamental level, product lifecycle management (PLM) is the strategic process of managing the complete journey of a product from initial ideation, development, service, and disposal. Put another way, PLM means managing everything involved with a product from cradle to grave.

PLM software is a solution that manages all of the information and processes at every step of a product or service lifecycle across globalized supply chains. This includes the data from items, parts, products, documents, requirements, engineering change orders, and quality workflows.

Today, supply chains have become more global, and businesses are shifting their operating models. For example, many companies are using embedded software services, such as product-as-a-service (PaaS) to sell new products or services. As a result, these organizations are discovering that they need a cloud-based PLM software that is ready to help them be adaptable and responsive.

GE Steam Power uses an in-house developed Product Lifecycle Management system and not a platform-as-a-service (PaaS) based system.

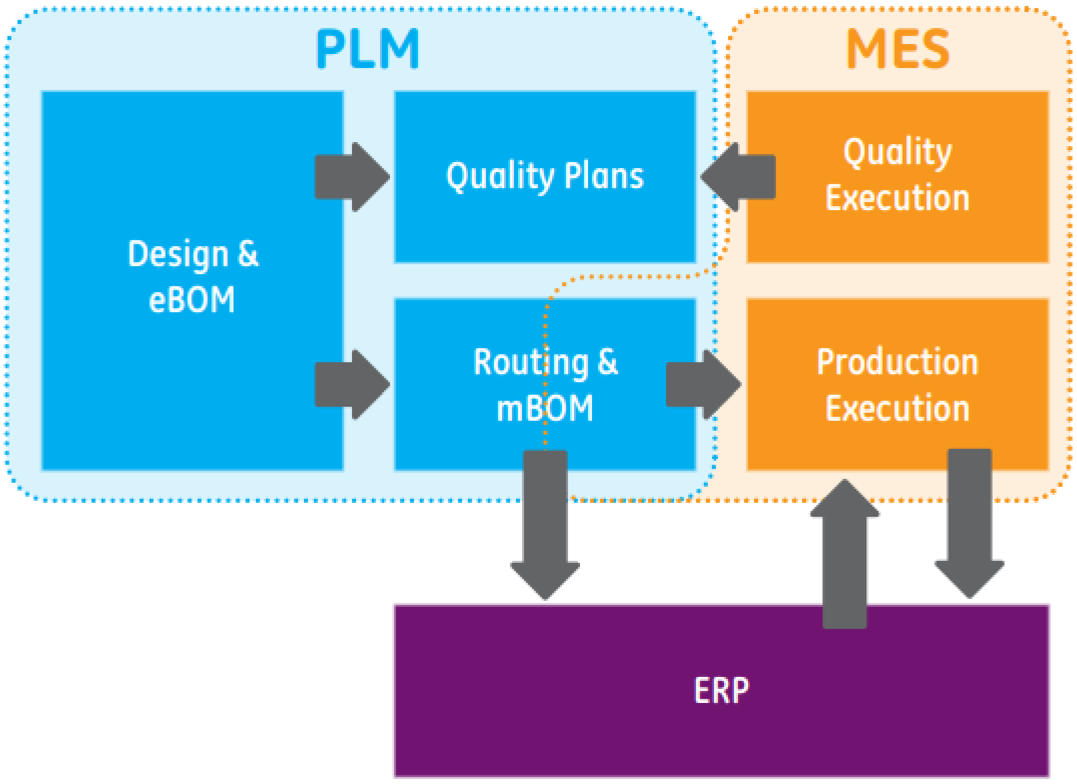


Figure 1 Working of a PLM system with other modules

The PLM system works with the MES (Manufacturing Execution System) and the ERP (Enterprise Resource Planning) systems. The PLM system generates an eBOM (Engineering Bill of Materials) which consists of the list of the parts needed for manufacturing to start. It also contains the PQL (Parts Qualification Level) which decides the level of Qualification the part needs to be assured with. PQL can save costs on a great level since most companies charge the price of a part based on its Qualification level. So, using a low PQL part when high qualification level is not needed saves costs to the business to a great extent.

- **eBOM:**

  An engineering bill of materials (eBOM) is a product recipe structured from the design standpoint, rather than the manufacturing standpoint. It originates in software used by the engineering department, such as computer-aided design or electronic design automation. The engineering bill of materials provides the components and directions to make a given product and includes things like raw materials, items, parts, subassemblies, interrelated data layers, as well as other factors, such as those that contribute to the cost of the product.

The engineering bill of materials focuses on parts as they exist in the design sense, and typically lists items from the engineering perspective, for example, on an assembly drawing. It does not include things like packaging, shipping containers and other components needed for a shippable product or specify how parts should be grouped at each stage of production. Such items are instead typically included in the manufacturing bill of materials.

An accurate engineering bill of materials is critical since the manufacturing bill of materials is based on the EBOM. Inaccuracy or incompleteness can mean incorrect product costs, inventory levels and accounting; production problems and delays; unnecessary revision cycles; and other issues. The right level of detail gives manufacturing information it needs to plan for new tools and testing, enables better part-purchasing decisions and prevents unnecessary changes.

Aligning bills of materials is so important that many in the field recommend that companies work toward a single bill of materials.

The Parts Qualification Level (PQL) is a part of the eBOM (Engineering Bill of Materials).

1.4 Proposed Idea

Our proposed model uses the description of a given part to classify the qualification needed by the part using a Multinomial Naïve Bayes model. The model takes in the numerical vectorized text and predicts ‘C’ or ‘Not C’. Originally, there are three qualification levels ‘A’, ‘B’ or ‘C’ but since almost 70% of the data is ‘C’, 30% of the data is ‘B’ and only 0.32% of the data is ‘A’. So, it is impossible for the model to classify as ‘A’. So, we decided that the model can just predict if the given part is ‘C’ or ‘Not C’. If the data is not equally split, the model may be biased towards the more abundant class. Since we have a 70:30 split I have used a method called Random Under Sampling which is run multiple times to train the model with minimal bias.

Chapter 2

2. Project Description and Goals

2.1 Project Description

The proposed system can be used by the engineer to classify the parts to their Part Qualification Level (PQL). First, we must train the model using the vectorized text data and use different metrics and the test data to evaluate the model. We can save the model and use it later for the engineer to use it.

The engineer can run the parts with this model and look for the results for which the model is confident and use them directly as the chances of a very confident prediction being wrong is rare. For the predictions with lower confidence the engineer can manually classify them. Since most of the predictions will have good confidence score, we can expect this project to help the engineer significantly to improve time and results.

2.2 Goals

The Goal of this project is to save time and money when ordering parts for a project. As most of the predictions given by the model will be accurate, the engineer can just check the parts which had less confidence. So, if most of the parts are being classified automatically, the engineer can save a lot of time and effort and put them in other work. And with the PQL classification process speeding up, we can expect less delays in shipping orders to clients.

Chapter 3

3. Technical Specifications

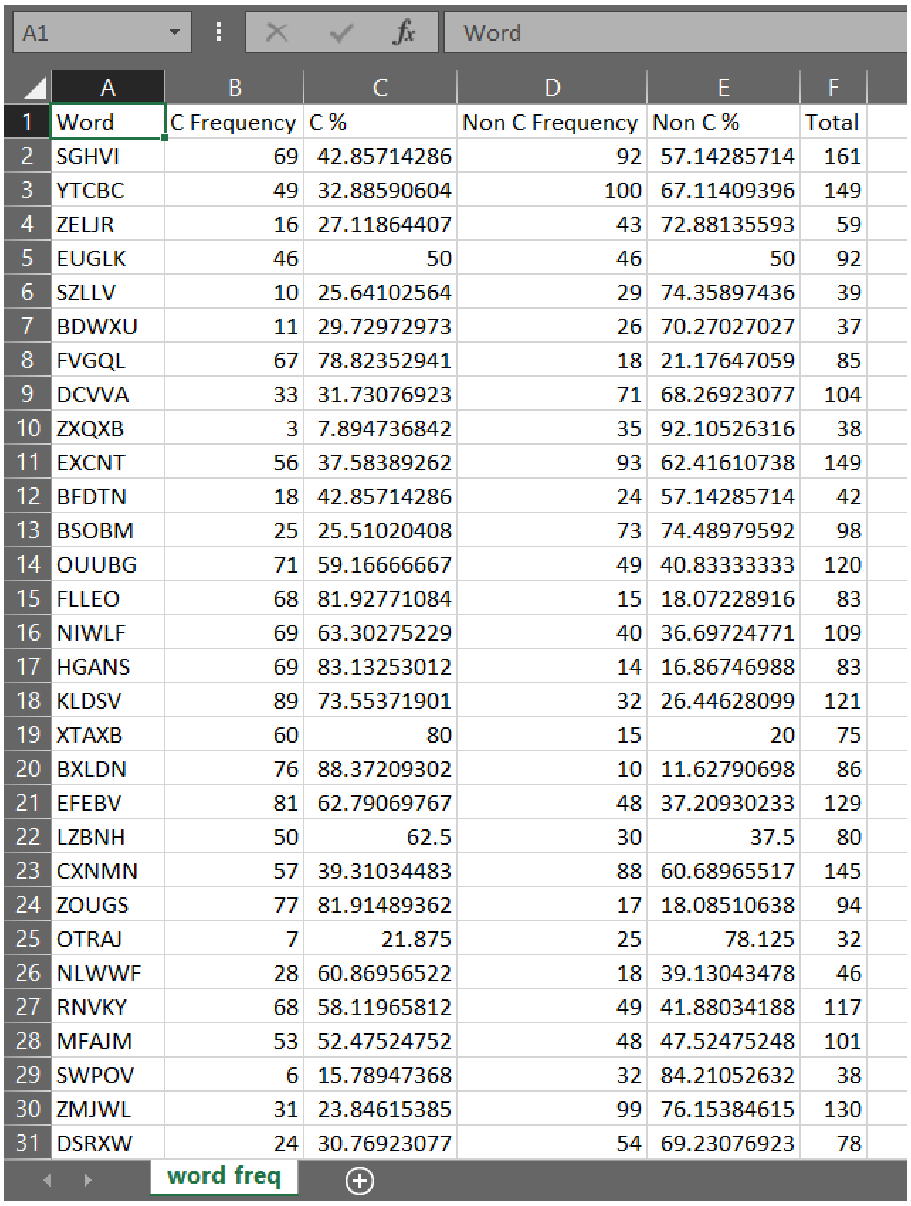
3.1 Exploratory Data Analysis (EDA)

3.1.1 Using Text Statistics

Text statistics visualizations are simple but very insightful techniques.

They include:

- **Word Frequency Analysis:**
  - By Analysing the frequency with which a word is occurring in a document, we can get an idea of what words are making more impact and which aren't.
  - Using this method in addition with the percent of different outcomes containing that word, we can gain a better understanding of the distribution of the data.
  - We can get more analytics about the data using these methods which will help us in making a better and simpler model.
  - We can remove words which occur less than a certain number of times to remove them since they have less impact and may lead to a false classification if occurred under a new class.
  - We can remove the words which occur nearly equally in both the outcomes since they don't make any difference in the predictions made by the model



A1 | Word

	A	B	C	D	E	F
1	Word	C Frequency	C %	Non C Frequency	Non C %	Total
2	SGHVI	69	42.85714286	92	57.14285714	161
3	YTCBC	49	32.88590604	100	67.11409396	149
4	ZELJR	16	27.11864407	43	72.88135593	59
5	EUGLK	46	50	46	50	92
6	SZLLV	10	25.64102564	29	74.35897436	39
7	BDWXU	11	29.72972973	26	70.27027027	37
8	FVGQL	67	78.82352941	18	21.17647059	85
9	DCVVA	33	31.73076923	71	68.26923077	104
10	ZXQXB	3	7.894736842	35	92.10526316	38
11	EXCNT	56	37.58389262	93	62.41610738	149
12	BFDTN	18	42.85714286	24	57.14285714	42
13	BSOBM	25	25.51020408	73	74.48979592	98
14	OUUBG	71	59.16666667	49	40.83333333	120
15	FLLEO	68	81.92771084	15	18.07228916	83
16	NIWLF	69	63.30275229	40	36.69724771	109
17	HGANS	69	83.13253012	14	16.86746988	83
18	KLDSV	89	73.55371901	32	26.44628099	121
19	XTAXB	60	80	15	20	75
20	BXLDN	76	88.37209302	10	11.62790698	86
21	EFEBV	81	62.79069767	48	37.20930233	129
22	LZBNH	50	62.5	30	37.5	80
23	CXNMN	57	39.31034483	88	60.68965517	145
24	ZOUGS	77	81.91489362	17	18.08510638	94
25	OTRAJ	7	21.875	25	78.125	32
26	NLWWF	28	60.86956522	18	39.13043478	46
27	RNVKY	68	58.11965812	49	41.88034188	117
28	MFAJM	53	52.47524752	48	47.52475248	101
29	SWPOV	6	15.78947368	32	84.21052632	38
30	ZMJWL	31	23.84615385	99	76.15384615	130
31	DSRXW	24	30.76923077	54	69.23076923	78

Figure 2 A Demonstration of the Word Analysis done. (Data randomly generated)

- We can see that each word in the vocabulary has been analysed to how many times it has occurred in the document and under which outcome has it occurred.

- With this analysis, we can decide which words are important and which words contribute the least.

- In the project, we have removed the words with frequency less than 5 times.

- We have also removed the words with a C percentage of more than 20% and less than 80%.

- With the removal of these words our vector size has reduced to around 2500 columns all the way down from 4200 columns.
- The model will run faster and will give more confident predictions in the future.

- **Sentence Length Analysis:**
    - It's important to have a look at the length of the text because it's an easy calculation that can give a lot of insights. Maybe, for instance, we are lucky enough to discover that one category is systematically longer than another and the length would simply be the only feature needed to build the model. Unfortunately, this won't be the case as news headlines have similar lengths, but it's worth a try.
    - There are several length measures for text data.
    - Here are some examples:
        - Word count: It counts the number of tokens in the text (separated by a space).
        - Character count: sum the number of characters of each token.
        - Sentence count: count the number of sentences (separated by a period).
        - Average sentence length: sum of sentences length divided by the number of sentences (word count/sentence count).
- **Average Word Length Analysis:**
    - It is the sum of words length divided by the number of words (character count/word count).
    - This gives us an idea of how complex the words in the document are.

- Since NLP mostly works with vectorizers which cannot ‘understand’ the meaning of a sentence, this analysis doesn’t contribute much.

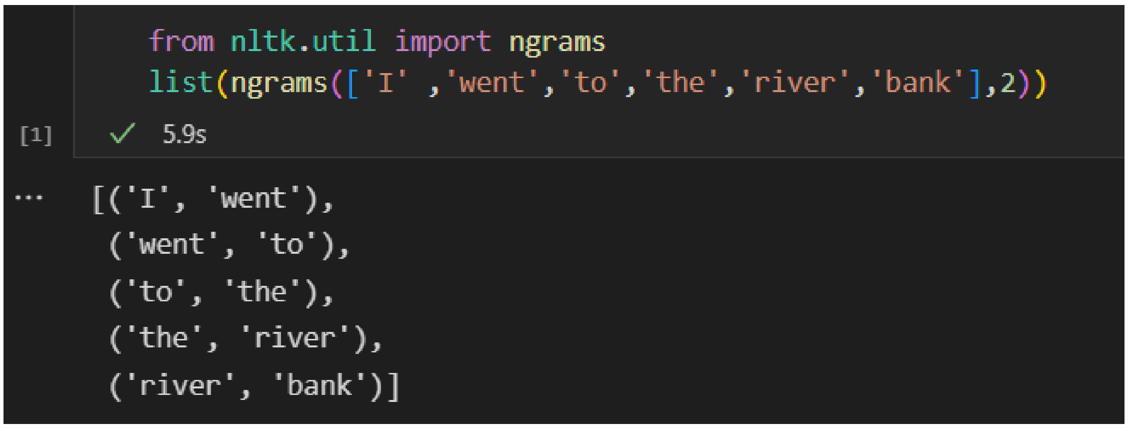
Those really help explore the fundamental characteristics of the text data. To do so, we will be mostly using histograms (continuous data) and bar charts (categorical data).

3.1.2 Ngram exploration

Ngrams are simply contiguous sequences of n words. For example, “riverbank”,” The three musketeers” etc. If the number of words is two, it is called bigram. For 3 words it is called a trigram and so on.

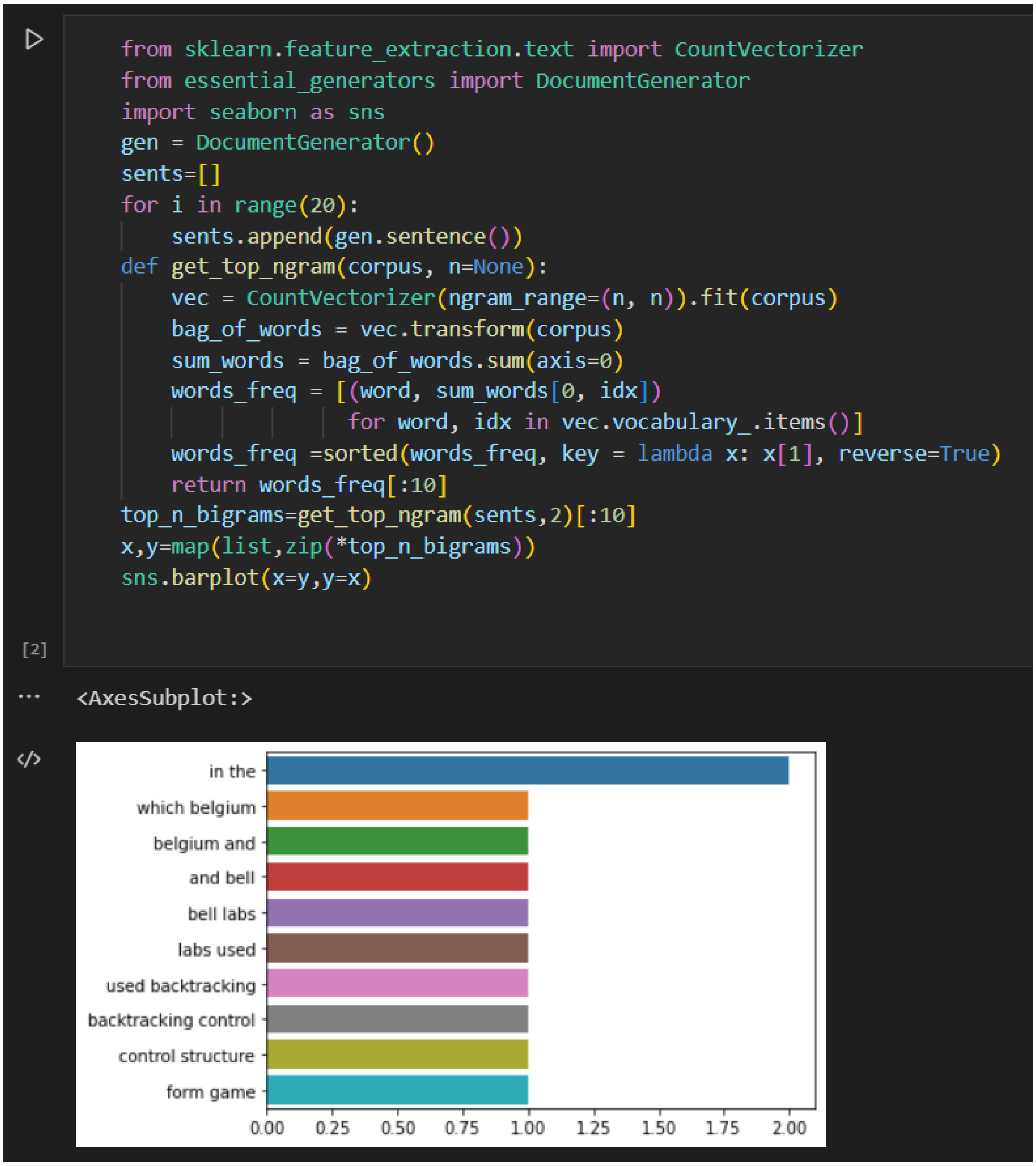
Looking at most frequent n-grams can give you a better understanding of the context in which the word was used.

To implement n-grams we will use ngrams function from nltk.util. For example:



```
from nltk.util import ngrams
list(ngrams(['I' ,'went','to','the','river','bank'],2))
[1]  ✓ 5.9s
...  [('I', 'went'),
      ('went', 'to'),
      ('to', 'the'),
      ('the', 'river'),
      ('river', 'bank')]
```

Figure 3 Demonstration of Ngrams



```
from sklearn.feature_extraction.text import CountVectorizer
from essential_generators import DocumentGenerator
import seaborn as sns
gen = DocumentGenerator()
sents=[]
for i in range(20):
    sents.append(gen.sentence())
def get_top_ngram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(n, n)).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx])
                  for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:10]
top_n_bigrams=get_top_ngram(sents,2)[:10]
x,y=map(list,zip(*top_n_bigrams))
sns.barplot(x=y,y=x)
```

```
[2]
<AxesSubplot:>
```

Figure 4 Demonstration of Ngram Exploration with Random Data

- We should put some effort into data cleaning and see if we were able to combine those synonym terms into one clean token.

3.1.3 Word Cloud

- Word cloud is a great way to represent text data. The size and colour of each word that appears in the word cloud indicate its frequency or importance.

- Word clouds (also known as text clouds or tag clouds) work in a simple way: the more a specific word appears in a source of textual data (such as a speech, blog post, or database), the bigger and bolder it appears in the word cloud.

- A word cloud is a collection, or cluster, of words depicted in different sizes.
- The bigger and bolder the word appears, the more often it's mentioned within a given text and the more important it is.
- Also known as tag clouds or text clouds, these are ideal ways to pull out the most pertinent parts of textual data, from blog posts to databases.
- They can also help business users compare two different pieces of text to find the wording similarities between the two.
- This type of visualization can assist evaluators with exploratory textual analysis by identifying words that frequently appear in a set of interviews, documents, or other text.
- It can also be used for communicating the most salient points or themes in the reporting stage.

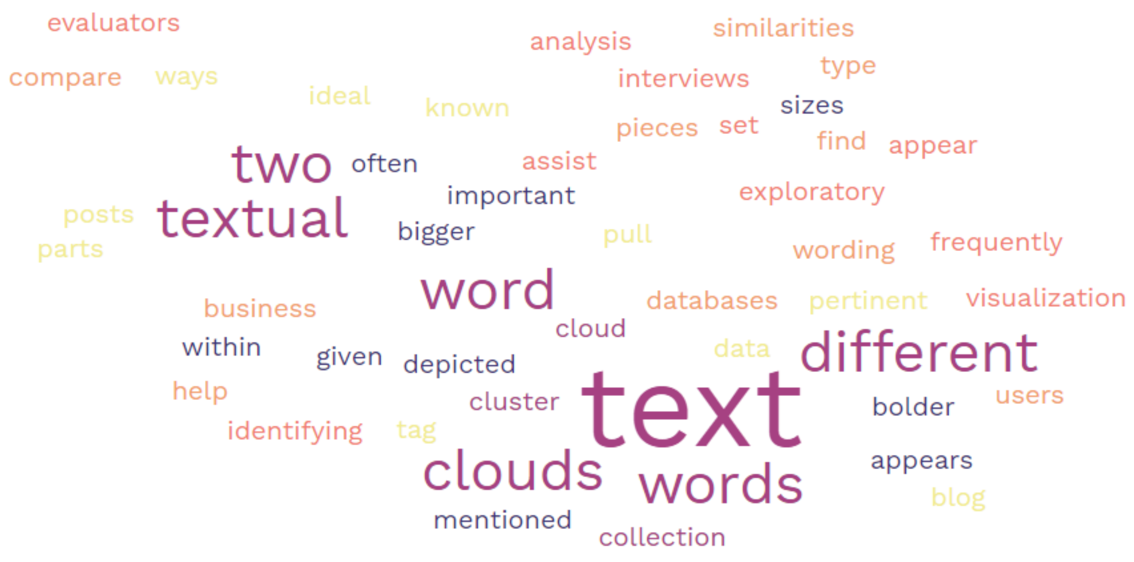


Figure 5 Generated word cloud of the text in this sub-section (for demonstration purposes)

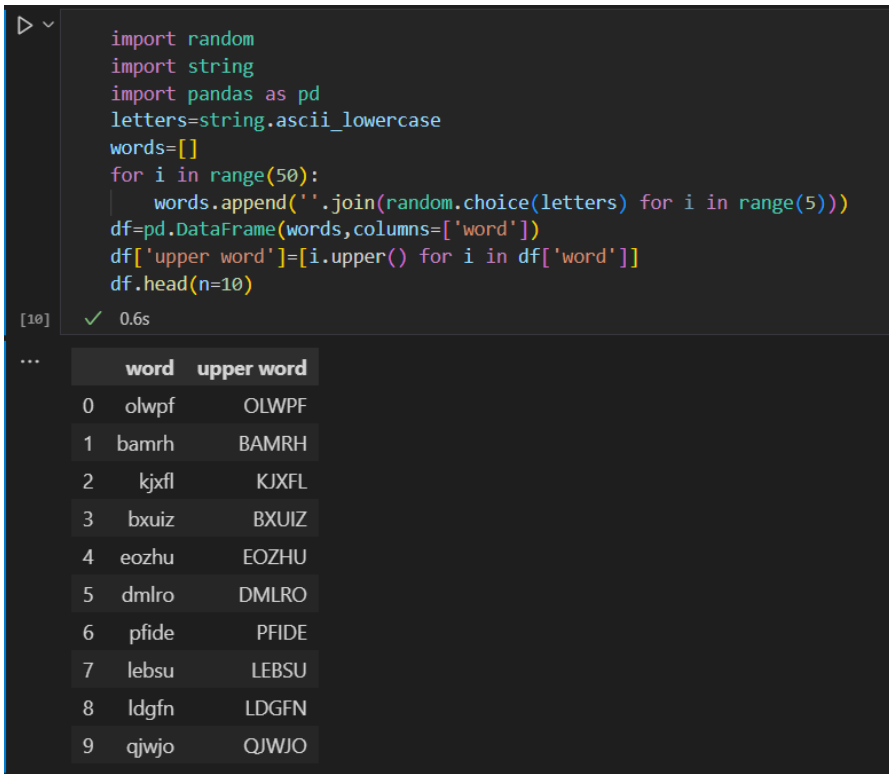
3.2 Text Preprocessing

- Text preprocessing is a method to clean the text data and make it ready to feed data to the model.
- Text data contains noise in various forms like emotions, punctuation, text in a different case.
- When we talk about Human Language then, there are different ways to say the same thing, and this is only the main problem we have to deal with because machines will not understand words, they need numbers, so we need to convert text to numbers in an efficient manner.
- Text processing is a method used under the NLP to clean the text and prepare it for the model building.
- It is versatile and contains noise in various forms like emotions, punctuations, and text written in numerical or special character forms.
- We must deal with these main problems because machines will not understand they ask only for numbers.
- To start with text processing, certain libraries written in Python simplify this process, and their simple, straightforward syntax gives a lot of flexibility.
- The first one is NLTK stands for natural language toolkit useful for all tasks like stemming, POS, tokenization, lemmatizing and many more.
- You might know the contraction; there is not any single sentence that is free from contractions means every time we tend to use words in a manner like didn’t instead of did not, so what happens when we tokenize such words it gets in form like ‘didn’ ‘t’ and there is nothing to do with such words.
- To deal with such words, there is a library called contractions.

3.2.1 Truecasing

- Truecasing is the problem in natural language processing (NLP) of determining the proper capitalization of words where such information is unavailable.
- This commonly comes up due to the standard practice (in English and many other languages) of automatically capitalizing the first word of a sentence.

- It can also arise in badly cased or noncased text (for example, all-lowercase or all-uppercase text messages).
- Truecasing is unnecessary in languages whose scripts do not have a distinction between uppercase and lowercase letters.
- This includes all languages not written in the Latin, Greek, Cyrillic or Armenian alphabets, such as Japanese, Chinese, Thai, Hebrew, Arabic, Hindi, and Georgian.
- If the text is in the same case, it is easy for a machine to interpret the words because the lower case and upper case are treated differently by the machine.
- For example, words like Ball and ball are treated differently by machine.
- So, we need to make the text in the same case and the most preferred case is an upper case to avoid such problems.
- Proper capitalization enables easier detection of proper nouns and helps increasing accuracy in translation.
- In this project, we went with uppercasing since most of the text in the description is small code words.



```
import random
import string
import pandas as pd
letters=string.ascii_lowercase
words=[]
for i in range(50):
    words.append(''.join(random.choice(letters) for i in range(5)))
df=pd.DataFrame(words,columns=['word'])
df['upper word']=[i.upper() for i in df['word']]
df.head(n=10)
```

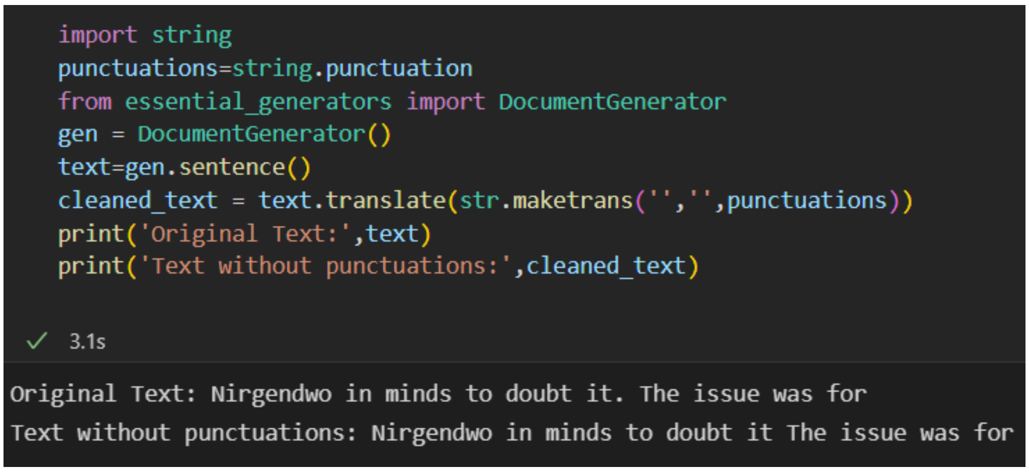
[10] ✓ 0.6s

	word	upper word
0	olwpf	OLWPF
1	bamrh	BAMRH
2	kjxfl	KJXFL
3	bxuiz	BXUIZ
4	eozhu	EOZHU
5	dmlro	DMLRO
6	pfide	PFIDE
7	lebsu	LEBSU
8	ldgfn	LDGFN
9	qjwjo	QJWJO

Figure 6 Demonstration of Upcasing

3.2.2 Removing Punctuations

- The second most common text processing technique is removing punctuations from the textual data. The punctuation removal process will help to treat each text equally.
- For example, the word data and data! are treated equally after the process of removal of punctuations.
- We need to take care of the text while removing the punctuation because the contraction words will not have any meaning after the punctuation removal process. Such as ‘don’t’ will convert to ‘dont’ or ‘don t’ depending upon what you set in the parameter.
- One of the other text processing techniques is removing punctuations. there are a total 32 main punctuations that need to be taken care of.
- We can directly use the string module with a regular expression to replace any punctuation in text with an empty string. 32 punctuations which string module provide us is '!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~'.
- Since the descriptions of the parts in PQL classification are just short text which do not have any full sentences the punctuations can be removed.



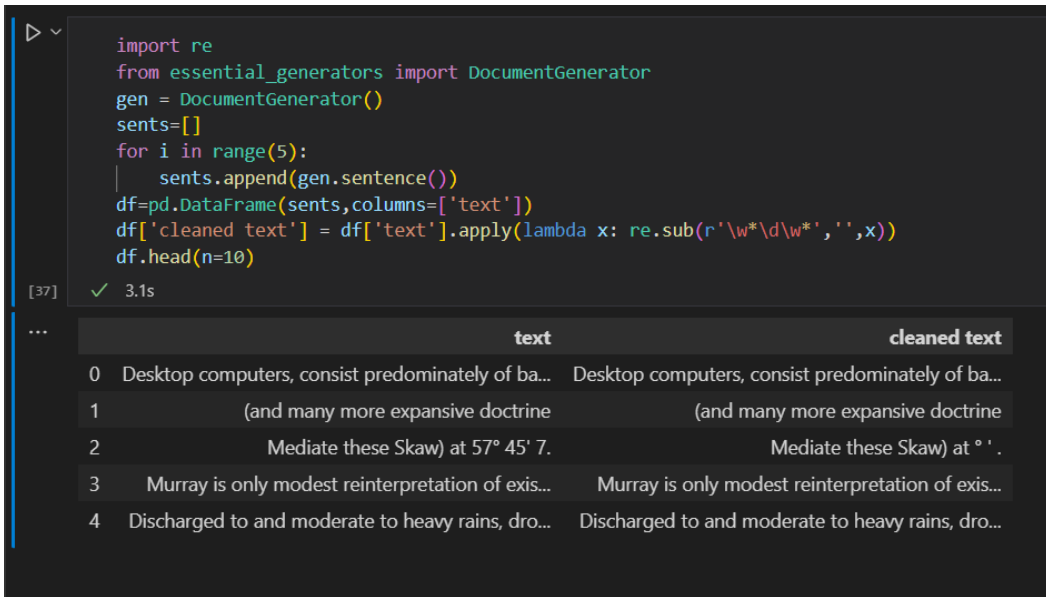
```
import string
punctuations=string.punctuation
from essential_generators import DocumentGenerator
gen = DocumentGenerator()
text=gen.sentence()
cleaned_text = text.translate(str.maketrans('','',punctuations))
print('Original Text:',text)
print('Text without punctuations:',cleaned_text)

✓ 3.1s

Original Text: Nirgendwo in minds to doubt it. The issue was for
Text without punctuations: Nirgendwo in minds to doubt it The issue was for
```

Figure 7 Demonstration of Punctuation removal.

3.2.3 Removing Numbers

- Sometimes number doesn’t hold any vital information in the text depending upon the use cases.
- So, it is better to remove them than to keep them.

- For example, when we are doing sentiment analysis then the number doesn’t hold any specific meaning to the data but if the task is to perform NER (Name Entity Recognition) or POS (Part of Speech tagging) then use the removing of number technique carefully.
- We need to remove the words and digits which are combined like game57 or game5ts7.
- This type of word is difficult to process so better to remove them or replace them with an empty string so, we use regular expressions for this.
- We have applied classification models on the text including and excluding numbers and it didn’t show any difference.
- So, we removed numbers for the sake of simplicity of the model.
- We can use the re (regular expressions) library for this task with which we can remove any unwanted characters with just a single line of code.



```python
import re
from essential_generators import DocumentGenerator
gen = DocumentGenerator()
sents=[]
for i in range(5):
    sents.append(gen.sentence())
df=pd.DataFrame(sents,columns=['text'])
df['cleaned text'] = df['text'].apply(lambda x: re.sub(r'\w*\d\w*','',x))
df.head(n=10)
```

[37] ✓ 3.1s

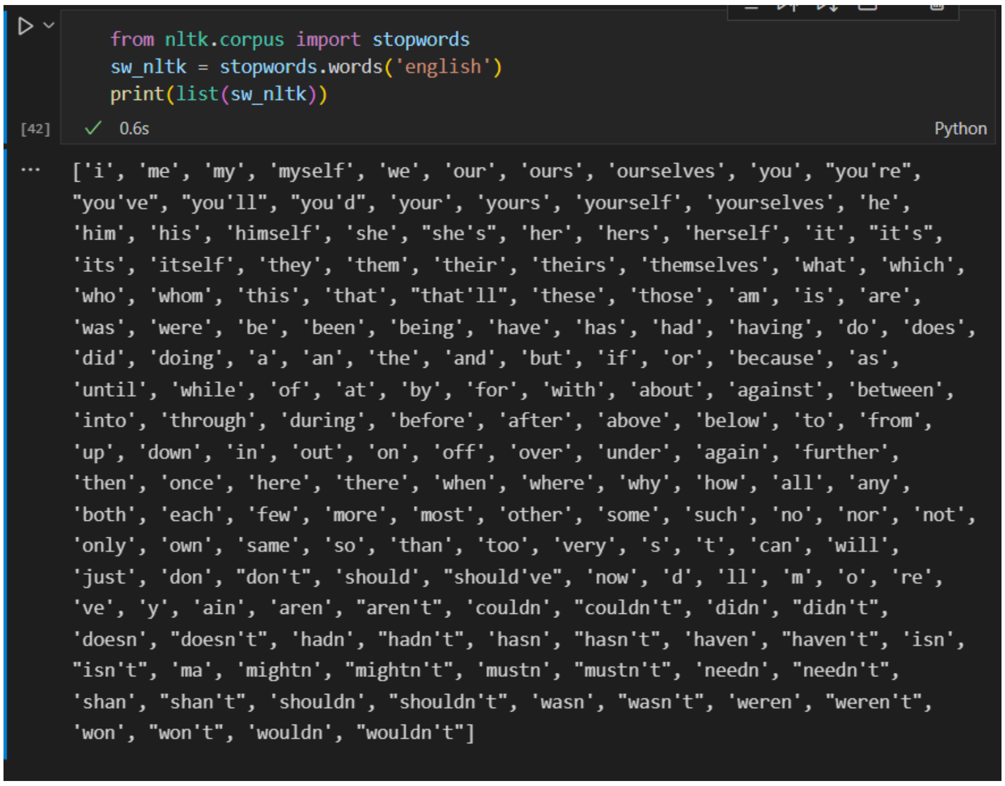
	text	cleaned text
0	Desktop computers, consist predominately of ba...	Desktop computers, consist predominately of ba...
1	(and many more expansive doctrine	(and many more expansive doctrine
2	Mediate these Skaw) at 57° 45' 7.	Mediate these Skaw) at ° ' .
3	Murray is only modest reinterpretation of exis...	Murray is only modest reinterpretation of exis...
4	Discharged to and moderate to heavy rains, dro...	Discharged to and moderate to heavy rains, dro...

Figure 8 Demonstration of removing words containing digits using Regular Expressions

3.4.4 Stopwords Removal

- **Stopword Meaning**

  The words which are generally filtered out before processing a natural language are called stop words. These are the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc) and does not add much information to the text. Examples of a few stop words in English are “the”, “a”, “an”, “so”, “what”.



```
from nltk.corpus import stopwords
sw_nltk = stopwords.words('english')
print(list(sw_nltk))
```

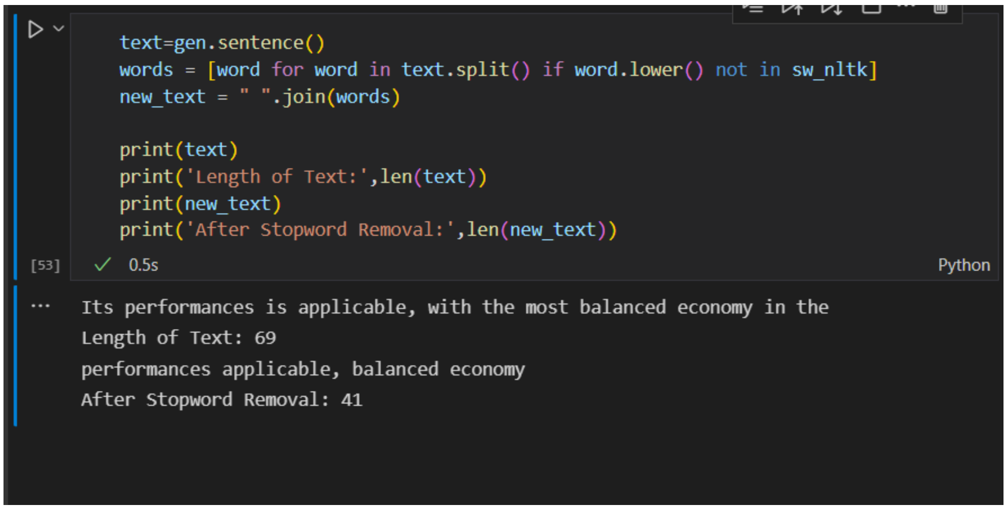
```
[42]  ✓ 0.6s                                                    Python
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
"you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's",
'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which',
'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are',
'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as',
'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from',
'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',
'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',
'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will',
'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're',
've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn',
"isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't",
'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't",
'won', "won't", 'wouldn', "wouldn't"]
```

Figure 9 Stopwords in the NLTK Library

- Stop words are available in abundance in any human language.
- By removing these words, we remove the low-level information from our text to give more focus to the important information.
- In order words, we can say that the removal of such words does not show any negative consequences on the model we train for our task.
- Removal of stop words reduces the dataset size and thus reduces the training time due to the fewer number of tokens involved in the training.
- We do not always remove the stop words. The removal of stop words is highly dependent on the task we are performing and the goal we want to achieve.
- For example, if we are training a model that can perform the sentiment analysis task, we might not remove the stop words as basic things like ‘love it’ and ‘did not love it’ become the same after stopword removal.
- Depending on the use case we can remove or not remove the ‘not’ keyword from stopwords as it can make some important decisions in use cases like sentiment analysis.

- Tasks like text classification do not generally need stop words as the other words present in the dataset are more important and give the general idea of the text.
- So, we generally remove stop words in such tasks.
- In a nutshell, NLP has a lot of tasks that cannot be accomplished properly after the removal of stop words.
- So, think before performing this step.
- The catch here is that no rule is universal, and no stop words list is universal.
- A list not conveying any important information to one task can convey a lot of information to the other task.



```python
text=gen.sentence()
words = [word for word in text.split() if word.lower() not in sw_nltk]
new_text = " ".join(words)

print(text)
print('Length of Text:',len(text))
print(new_text)
print('After Stopword Removal:',len(new_text))
```

[53] ✓ 0.5s Python

```
Its performances is applicable, with the most balanced economy in the
Length of Text: 69
performances applicable, balanced economy
After Stopword Removal: 41
```

Figure 10 Demonstration of Stopword Removal on a randomly generated sentence

3.4.5 Stemming

- Stemming is a technique used to extract the base form of the words by removing affixes from them.
- It is just like cutting down the branches of a tree to its stems. For example, the stem of the words eating, eats, eaten is eat.
- Search engines use stemming for indexing the words.
- That’s why rather than storing all forms of a word, a search engine can store only the stems.

- In this way, stemming reduces the size of the index and increases retrieval accuracy.
- Stemming is the process of producing morphological variants of a root/base word.
- Stemming programs are commonly referred to as stemming algorithms or stemmers.
- A stemming algorithm reduces the words “chocolates”, “chocolatey”, “Choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”.
- Stemming is an important part of the pipelining process in Natural language processing.
- There are mainly two errors in stemming –
    - **Over-stemming**

      It occurs when two words are stemmed from the same root that are of different stems. Over-stemming can also be regarded as false positives.
    - **Under-stemming**

      It occurs when two words are stemmed from the same root that are not of different stems. Under-stemming can be interpreted as false negatives.
- There are multiple stemming algorithms each with its own unique working. Some of them are:
    - **Porter Stemmer**
        - It is one of the most popular stemming methods proposed in 1980. It is based on the idea that the suffixes in the English language are made up of a combination of smaller and simpler suffixes.
        - This stemmer is known for its speed and simplicity. The main applications of Porter Stemmer include data mining and Information retrieval.

    - However, its applications are only limited to English words.
    - Also, the group of stems is mapped on to the same stem and the output stem is not necessarily a meaningful word.
    - The algorithms are fairly lengthy in nature and are known to be the oldest stemmer.
    - Example: EED -> EE means “if the word has at least one vowel and consonant plus EED ending, change the ending to EE” as ‘agreed’ becomes ‘agree’.
    - This class knows several regular word forms and suffixes with the help of which it can transform the input word to a final stem.
    - The resulting stem is often a shorter word having the same root meaning.
- **Snowball Stemmer**
    - When compared to the Porter Stemmer, the Snowball Stemmer can map non-English words too.
    - Since it supports other languages the Snowball Stemmers can be called a multi-lingual stemmer.
    - The Snowball stemmers are also imported from the nltk package.
    - This stemmer is based on a programming language called ‘Snowball’ that processes small strings and is the most widely used stemmer.
    - The Snowball stemmer is way more aggressive than Porter Stemmer and is also referred to as Porter2 Stemmer.
    - Because of the improvements added when compared to the Porter Stemmer, the Snowball stemmer is having greater computational speed.
    - The NLTK Snowball Stemmer supports 15 non-English languages.

- **Lancaster Stemmer**
    - The Lancaster stemmers are more aggressive, and dynamic compared to the other two stemmers.
    - The stemmer is faster, but the algorithm is really confusing when dealing with small words.
    - But they are not as efficient as Snowball Stemmers.
    - The Lancaster stemmers save the rules externally and basically uses an iterative algorithm.
    - Over-stemming renders stems non-linguistic or meaningless.
- **RegExp Stemmer**
    - Regex stemmer identifies morphological affixes using regular expressions. Substrings matching the regular expressions will be discarded.

3.4.6 Lemmatization

- Lemmatization is like stemming, used to stem the words into root word but differs in working.
- Lemmatization is a systematic way to reduce the words into their lemma by matching them with a language dictionary.
- Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Lemmatization is like stemming but it brings context to the words.
- So, it links words with similar meanings to one word.
- Text preprocessing includes both Stemming as well as Lemmatization. Many times, people find these two terms confusing. Some treat these two as the same.
- Lemmatization is preferred over Stemming because lemmatization does morphological analysis of the words.
- One major difference with stemming is that lemmatize takes a part of speech parameter, “pos” If not supplied, the default is “noun”.

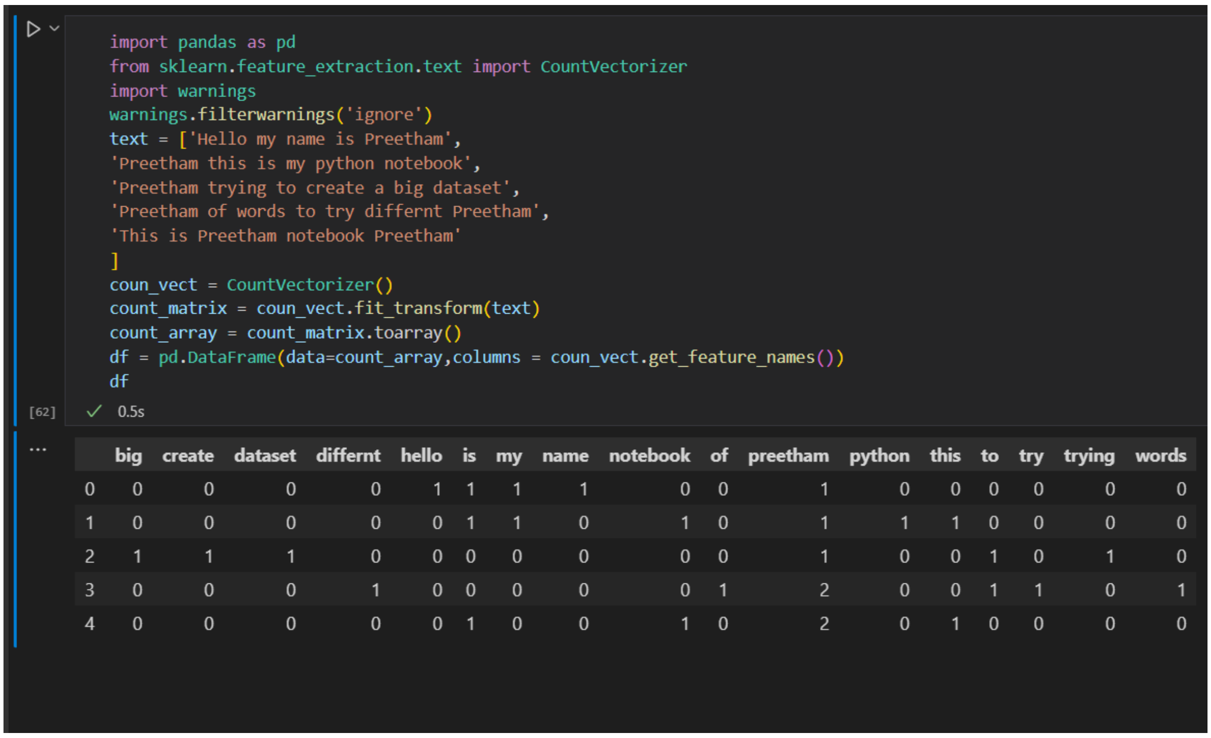
3.3 Vectorization or Word Embeddings

- Processing natural language text and extract useful information from the given word, a sentence using machine learning and deep learning techniques requires the string/text needs to be converted into a set of real numbers (a vector) which are called Word Embeddings.
- Word Embeddings or Word vectorization is a methodology in NLP to map words or phrases from vocabulary to a corresponding vector of real numbers which used to find word predictions, word similarities/semantics.
- The process of converting words into numbers are called Vectorization.
- Word embeddings help in the following use cases.
  - Compute similar words
  - Text classifications
  - Document clustering/grouping
  - Feature extraction for text classifications
  - Natural language processing.
- After the words are converted as vectors, we need to use some techniques such as Euclidean distance, Cosine Similarity to identify similar words.
- There are different types of vectorizers with which we can extract features from text. Some of them are:

3.3.1 Count Vectorizer

- Count Vectorizer is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text.
- This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors (for using in further text analysis).

- To use textual data for predictive modelling, the text must be parsed to remove certain words – this process is called tokenization.
- These words need to then be encoded as integers, or floating-point values, for use as inputs in machine learning algorithms.
- This process is called feature extraction (or vectorization).
- Scikit-learn’s Count Vectorizer is used to convert a collection of text documents to a vector of term/token counts.
- It also enables the pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.
- Count Vectorizer creates a matrix in which each unique word is represented by a column of the matrix, and each text sample from the document is a row in the matrix.
- The value of each cell is nothing but the count of the word in that particular text sample.
- Count Vectorizer saves the vectorized text in a data type known as ‘Spare Matrix’ which saves a lot of storage space and helps in computation time.
- If you do not provide an a-priori dictionary and you do not use an analyser that does feature selection, then the number of features will be equal to the vocabulary size found by analysing the data.



```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
import warnings
warnings.filterwarnings('ignore')
text = ['Hello my name is Preetham',
'Preetham this is my python notebook',
'Preetham trying to create a big dataset',
'Preetham of words to try differnt Preetham',
'This is Preetham notebook Preetham'
]
coun_vect = CountVectorizer()
count_matrix = coun_vect.fit_transform(text)
count_array = count_matrix.toarray()
df = pd.DataFrame(data=count_array,columns = coun_vect.get_feature_names())
df
```

[62] ✓ 0.5s

```
   big  create  dataset  differnt  hello  is  my  name  notebook  of  preetham  python  this  to  try  trying  words
0    0       0        0         0      1   1   1     1         0   0         1       0     0   0    0       0      0
1    0       0        0         0      0   1   1     0         1   0         1       1     1   0    0       0      0
2    1       1        1         0      0   0   0     0         0   0         1       0     0   1    0       1      0
3    0       0        0         1      0   0   0     0         0   1         2       0     0   1    1       0      1
4    0       0        0         0      0   1   0     0         1   0         2       0     1   0    0       0      0
```

Figure 11 Demonstration of Count Vectorizer on Fabricated Text Data

- We can see in the above demonstration that it has taken all the words in the document and made a matrix with each column being a word.
- Then, for each sentence, it puts out a count of each column if the word is present in the sentence.
- With this numerical data, we can train machine learning models.

3.3.2 TF-IDF Vectorizer

- TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency.
- This is very common algorithm to transform text into a meaningful representation of numbers which is used to fit machine algorithm for prediction.
- TFIDF works by proportionally increasing the number of times a word appears in the document but is counterbalanced by the number of documents in which it is present.
- Hence, words like ‘this’, ’are’ etc., that are commonly present in all the documents are not given a very high rank.

- However, a word that is present too many times in a few of the documents will be given a higher rank as it might be indicative of the context of the document.
- Term frequency-inverse document frequency is a text vectorizer that transforms the text into a usable vector. It combines 2 concepts, Term Frequency (TF) and Document Frequency (DF).
- **Term Frequency**
    - The term frequency is the number of occurrences of a specific term in a document.
    - Term frequency indicates how important a specific term in a document.
    - Term frequency represents every text from the data as a matrix whose rows are the number of documents and columns are the number of distinct terms throughout all documents.
    - Term frequency is defined as the number of times a word (i) appears in a document (j) divided by the total number of words in the document.

$$\mathrm{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Equation 1 Term Frequency

- **Document Frequency**

  Document frequency is the number of documents containing a specific term. Document frequency indicates how common the term is.

- **Inverse Document Frequency**

    - Inverse document frequency (IDF) is the weight of a term, it aims to reduce the weight of a term if the term’s occurrences are scattered throughout all the documents.

    - Inverse document frequency refers to the log of the total number of documents divided by the number of documents that contain the word.

    - The logarithm is added to dampen the importance of a very high value of IDF.

$$idf(w) = \log\left(\frac{N}{df_t}\right)$$

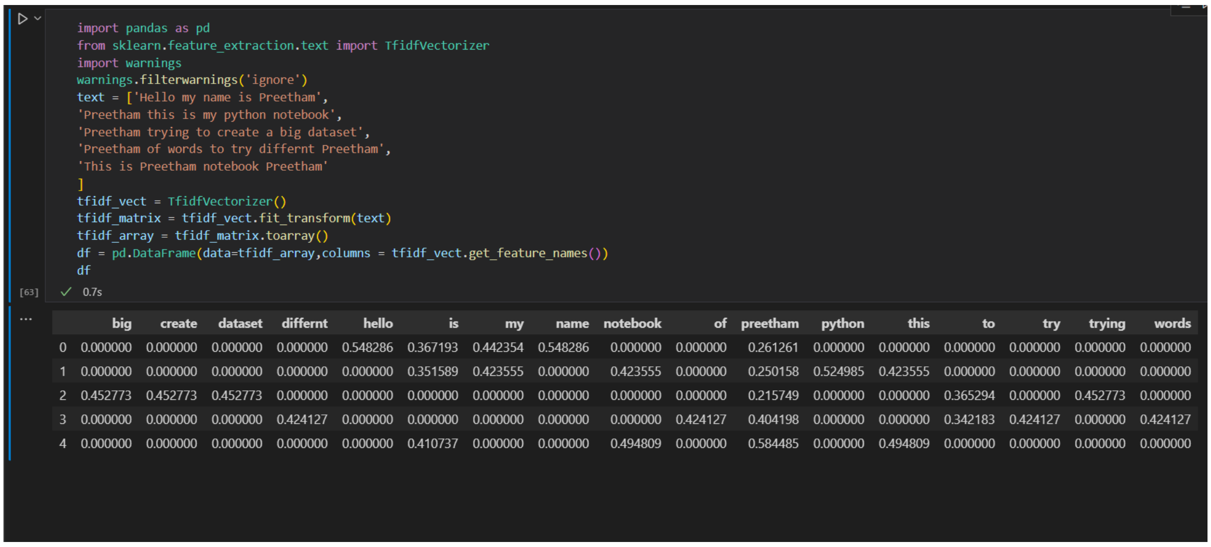
Equation 2 Inverse Document Frequency

- Where idf_i is the IDF score for term i, df_i is the number of documents containing term i, and n is the total number of documents.
- The higher the DF of a term, the lower the IDF for the term.
- When the number of DF is equal to n which means that the term appears in all documents, the IDF will be zero, since log (1) is zero, when in doubt just put this term in the stopword list because it doesn't provide much information.

- TFIDF is computed by multiplying the term frequency with the inverse document frequency.

$$w_{i,j} = tf_{i,j} x \log\left(\frac{N}{df_i}\right)$$

Equation 3 TF-IDF Vectorizer



```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import warnings
warnings.filterwarnings('ignore')
text = ['Hello my name is Preetham',
'Preetham this is my python notebook',
'Preetham trying to create a big dataset',
'Preetham of words to try differnt Preetham',
'This is Preetham notebook Preetham'
]
tfidf_vect = TfidfVectorizer()
tfidf_matrix = tfidf_vect.fit_transform(text)
tfidf_array = tfidf_matrix.toarray()
df = pd.DataFrame(data=tfidf_array,columns = tfidf_vect.get_feature_names())
df
```

[63] ✓ 0.7s

	big	create	dataset	differnt	hello	is	my	name	notebook	of	preetham	python	this	to	try	trying	words
0	0.000000	0.000000	0.000000	0.000000	0.548286	0.367193	0.442354	0.548286	0.000000	0.000000	0.261261	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.351589	0.423555	0.000000	0.423555	0.000000	0.250158	0.524985	0.423555	0.000000	0.000000	0.000000	0.000000
2	0.452773	0.452773	0.452773	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.215749	0.000000	0.000000	0.365294	0.000000	0.452773	0.000000
3	0.000000	0.000000	0.000000	0.424127	0.000000	0.000000	0.000000	0.000000	0.000000	0.424127	0.404198	0.000000	0.000000	0.342183	0.424127	0.000000	0.424127
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.410737	0.000000	0.000000	0.494809	0.000000	0.584485	0.000000	0.494809	0.000000	0.000000	0.000000	0.000000

Figure 12 Demonstration of TF-IDF Vectorizer

3.4 Machine Learning

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.

Instead of relying on manually crafted rules, machine learning text classification learns to make classifications based on past observations. By using pre-labeled examples as training data, machine learning algorithms can learn the different associations between pieces of text, and that a particular output (i.e., tags) is expected for a particular input (i.e., text). A “tag” is the pre-determined classification or category that any given text could fall into.

Some of the most popular text classification algorithms include the Naive Bayes family of algorithms, support vector machines (SVM), and deep learning.

3.4.1 Naïve Bayes Classifier

The Naive Bayes family of statistical algorithms are some of the most used algorithms in text classification and text analysis, overall.

One of the members of that family is Multinomial Naive Bayes (MNB) with a huge advantage, that you can get really good results even when your dataset isn’t very large (~ a couple of thousand tagged samples) and computational resources are scarce.

Naive Bayes is based on Bayes’ Theorem, which helps us compute the conditional probabilities of the occurrence of two events, based on the probabilities of the occurrence of each individual event. So, we’re calculating the probability of each tag for a given text, and then outputting the tag with the highest probability.

$$P(A|B) = \frac{P(B|A)x\,P(A)}{P(B)}$$

Equation 4 Naive Bayes Equation

The probability of A, if B is true, is equal to the probability of B, if A is true, times the probability of A being true, divided by the probability of B being true.

This means that any vector that represents a text will have to contain information about the probabilities of the appearance of certain words within the texts of a given category, so that the algorithm can compute the likelihood of that text belonging to the category.

3.4.2 Support Vector Machines (SVM)

Support Vector Machines (SVM) is another powerful text classification machine learning algorithm, because like Naive Bayes, SVM doesn't need much training data to start providing accurate results. SVM does, however, require more computational resources than Naive Bayes, but the results are even faster and more accurate.

In short, SVM draws a line or "hyperplane" that divides a space into two subspaces. One subspace contains vectors (tags) that belong to a group, and another subspace contains vectors that do not belong to that group.

The optimal hyperplane is the one with the largest distance between each tag. In two dimensions it looks like this:

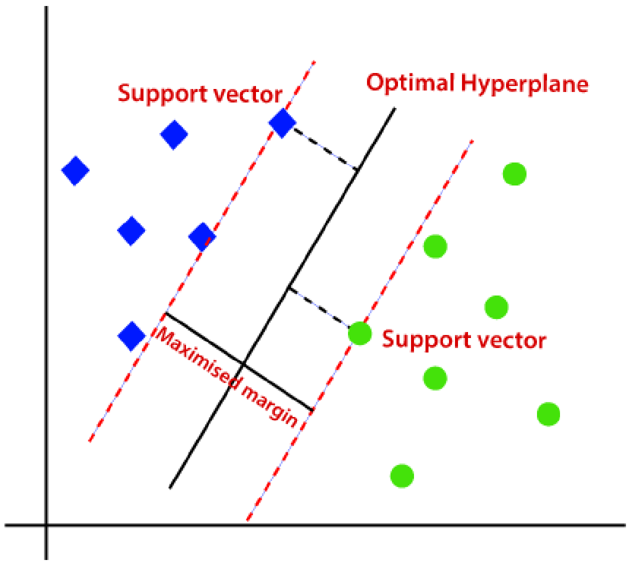


Figure 13 Explanation of SVM Hyperplane

Those vectors are representations of your training texts, and a group is a tag you have tagged your texts with.

When the data is not linear, we can use different kernels. We can use the poly kernel as it generates a custom line which can fit for any distribution of data.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

The support vector machines in scikit-learn support both dense (numpy.ndarray and convertible to that by numpy.asarray) and sparse (any scipy.sparse) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered numpy.ndarray (dense) or scipy.sparse.csr_matrix (sparse) with dtype=float64.

3.4.3 Logistic Regression

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes.

Logistic regression is a calculation used to predict a binary outcome: either something happens or does not. This can be exhibited as Yes/No, Pass/Fail, Alive/Dead, etc.

Independent variables are analysed to determine the binary outcome with the results falling into one of two categories. The independent variables can be categorical or numeric, but the dependent variable is always categorical. Written like this:

$$P(Y = 1|X) \text{ or } P(Y = 0|X)$$

Equation 5 Logistic Regression

It calculates the probability of dependent variable Y, given independent variable X.

This can be used to calculate the probability of a word having a positive or negative connotation (0, 1, or on a scale between). Or it can be used to determine the object contained in a photo (tree, flower, grass, etc.), with each object given a probability between 0 and 1.

Types of Logistic Regression

Generally, logistic regression means binary logistic regression having binary target variables, but there can be two more categories of target variables that can be predicted by it. Based on those number of categories, Logistic regression can be divided into following types –

- Binary or Binomial: In such a kind of classification, a dependent variable will have only two possible types either 1 and 0. For example, these variables may represent success or failure, yes or no, win or loss etc.
- Multinomial: In such a kind of classification, dependent variable can have 3 or more possible unordered types or the types having no quantitative significance. For example, these variables may represent “Type A” or “Type B” or “Type C”.
- Ordinal: In such a kind of classification, dependent variable can have 3 or more possible ordered types or the types having a quantitative significance. For example, these variables may represent “poor” or “good”, “very good”, “Excellent” and each category can have the scores like 0,1,2,3.

3.4.4 Decision Trees

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

The decisions or the test are performed on the basis of features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure. In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm. A decision tree simply asks a question and based on the answer (Yes/No), it further split the tree into subtrees.

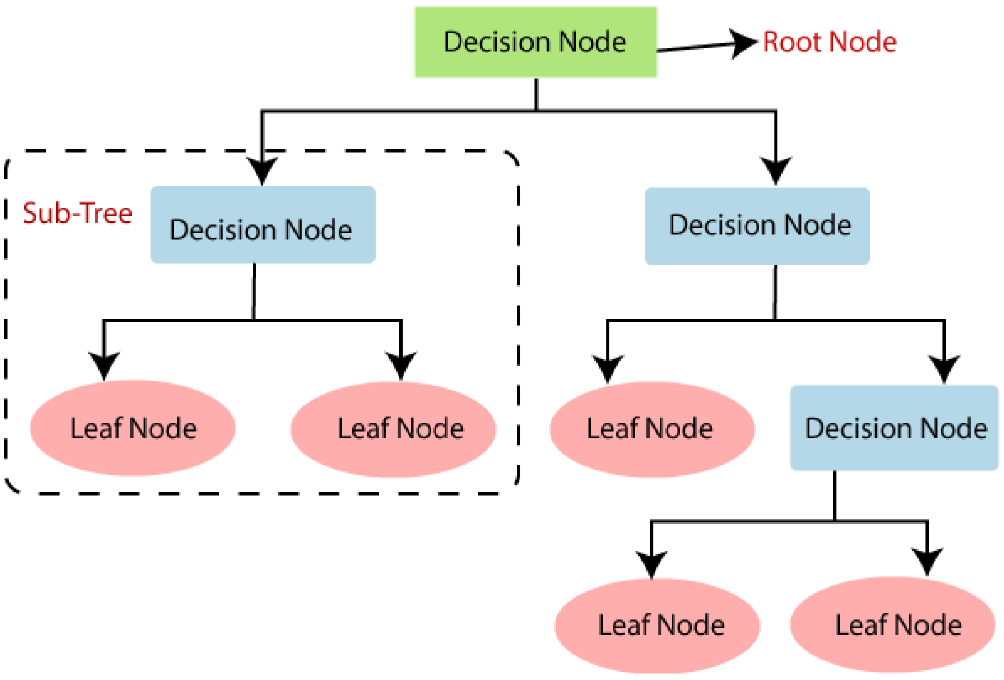


Figure 14 Working of a Decision Tree Model

3.4.5 Random Forest Classifier

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

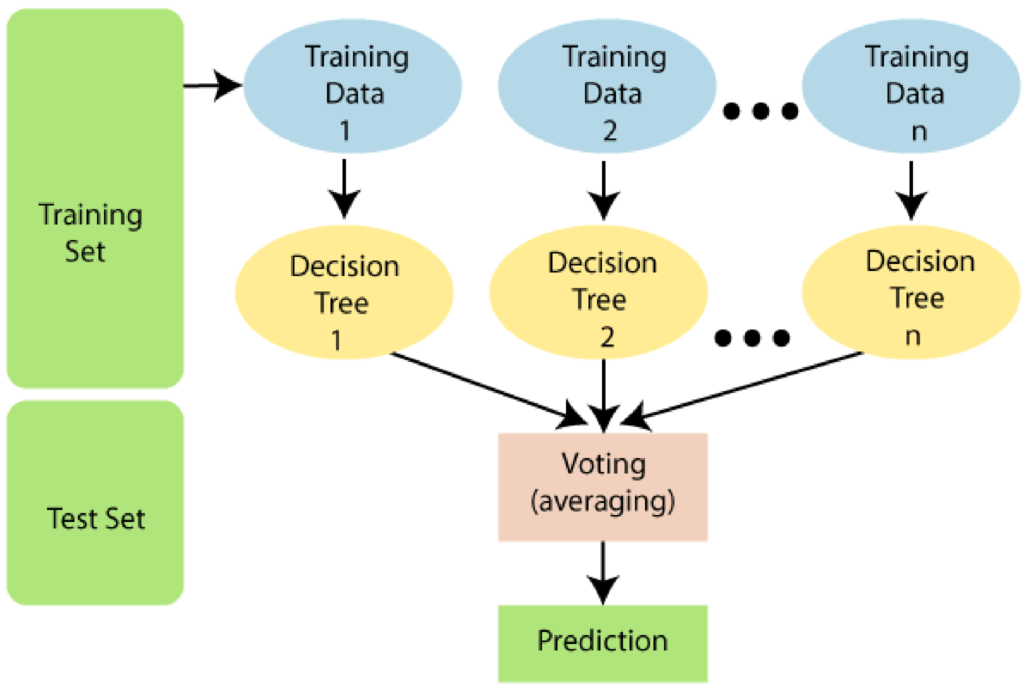


Figure 15 Working of a Random Forest Model

Below are some points that explain why we should use the Random Forest algorithm:

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

Advantages of Random Forest:

- Random Forest can perform both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

Disadvantages of Random Forest:

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

Chapter 4

4. Design Approach

In this chapter we explain the approach used in this project in detail starting from Exploratory Data Analysis, Text Pre-processing, Vectorization, and the machine learning models chosen.

4.1 Block Diagram

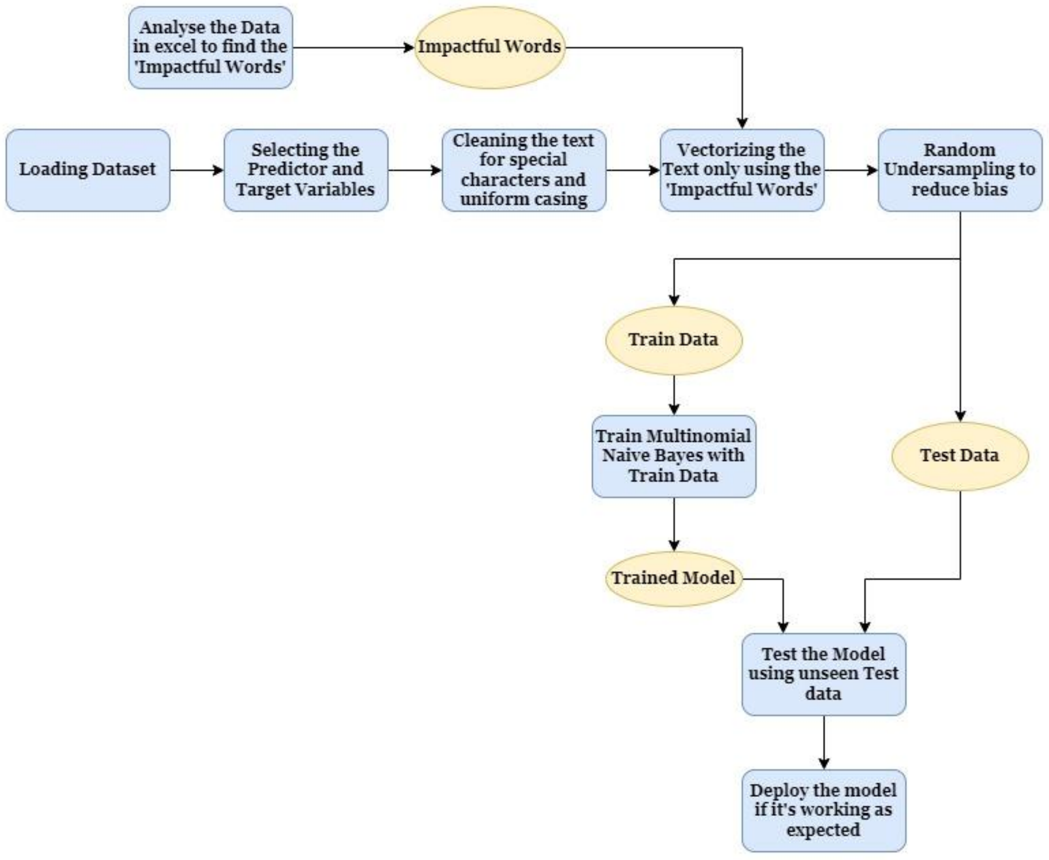

Analyse the Data in excel to find the 'Impactful Words'
Impactful Words
Loading Dataset
Selecting the Predictor and Target Variables
Cleaning the text for special characters and uniform casing
Vectorizing the Text only using the 'Impactful Words'
Random Undersampling to reduce bias
Train Data
Train Multinomial Naive Bayes with Train Data
Test Data
Trained Model
Test the Model using unseen Test data
Deploy the model if it's working as expected

Figure 16. High Level Block Diagram of the proposed system

1. First, we must analyze the data to find the words which make the most impact. If we use all the words that occurred for one or two times, the model may get overfitted to the data and when a new part comes with the word that is from a different class, it gives the wrong prediction. And when a word that is not contributing a lot to the predictions (around 50% of the predictions are 'C' and others are 'Not C') it doesn't make sense to include the word in the vocabulary since it makes the model bigger. So, removing the words that are not making much

impact is very important.

2. Loading the dataset into the environment to work on it.

3. Since the dataset contains many columns, we need to select the columns which are being used as the 'target variable' and 'predictor variable'.

   - **Target variable**: The "target variable" is the variable whose values are to be modelled and predicted by other variables. It is analogous to the dependent variable (i.e., the variable on the left of the equal sign) in linear regression. There must be one and only one target variable in a decision tree analysis.

   - **Predictor variable**: A "predictor variable" is a variable whose values will be used to predict the value of the target variable. It is analogous to the independent variables (i.e., variables on the right side of the equal sign) in linear regression. There must be at least one predictor variable specified for decision tree analysis; there may be many predictor variables.

4. The text data generally contains a lot of special characters and other words which needs to be cleaned before vectorizing since most of these characters don't make a difference. We also need all of the words to be either lowercase or uppercase as if the words are not uniform, the vectorizer will consider them differently which is not desired.

5. Machine Learning models are just mathematical equations which use numbers and create a generalized vector to make predictions to for the new data. So, we need to somehow convert the text in the description to numbers to feed it to the ML model.

   a. **Vectorization**: To convert the text data into numerical data, we need some smart ways which are known as vectorization**,** or in the NLP world, it is known as Word embeddings**.** Therefore, Vectorization or word embedding is the process of converting text data to numerical vectors. Later those vectors are used to build various machine learning models. In this manner, we say this as extracting features with the help of text with an aim to build multiple natural languages, processing models, etc.

6. The training data that was available was not split ideally. It was split in a 70:30 ratio which may make the model have high bias than needed. To reduce the effect of this, we can use under sampling.

   a. **Under Sampling**: Under sampling is a technique to balance uneven datasets by keeping all the data in the minority class and decreasing the size of the majority class. It is one of several techniques data scientists can use to extract more accurate information from originally imbalanced datasets.

7. Once we have the vectorized data which can be used by the model, we can train the model using the vectorized data. There are multiple ML models that can do the classification, but we went with Naïve Bayes. Although Decision Trees give great accuracies for the training dataset, we can see that it gets overfitted to the training data and when the new data with new data comes, it may struggle.

   a. **Naïve Bayes**: Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e., every pair of features being classified is independent of each other.

8. Once we train the model, we can use the testing data that we have kept aside which do have actual values, but we predict them again and check how many of these predictions are matching with actual values. We can use multiple testing metrics to test the model. With different metrics we can get more idea of how the model is working. The metrics that we've used are Accuracy, True Positive, False Positive, True Negative, False Negative and R2 Score.

4.2 Vectorization Comparison

- The two vectorization options seem to be producing very similar results.
- In theory, the TF-IDF vectorizer should give more performance but since the data in this project is just short text, we can understand that the TF-IDF vectorizer is not being utilized to its full potential.
- We can see the advantages of the TF-IDF vectorizer in the computation times since the model training happens faster using the tf-idf vectorizer than the count vectorizer.
- The TF-IDF Vectorizer was finalized to be used in the project.

TF-IDF Vectorizer (Test Acc)	**Count Vectorizer (Test Acc)**
Random Forest: 97.329	Random Forest: 97.256
Decision Tree: 96.488	Decision Tree: 96.561
Logistic Regression: 92.995	Logistic Regression: 93.708
Naïve Bayes: 91.495	Naïve Bayes: 90.983

Table 1 Comparing different vectorization algorithms

4.3 Classification Model Comparison

- The below table shows that the Decision Tree algorithm has the best performance.

- But since decision tree models can get completely over-fitted to the data and not give good results when new data is given.
- We're using the Multinomial Naïve Bayes model for the project which although is not giving great results, but at least it's not over-fitting.

Metric	**Random Forest**	**Decision Tree**	**Logistic Regression**	**Naïve Bayes**
Train Accuracy	99.414	99.414	93.782	92.318
Test Accuracy	97.329	96.488	92.995	91.495
True positive	1632	1638	1464	1477
False Positive	83	135	152	247
False Negative	63	57	231	218
True Negative	3690	3638	3621	3626
R2 Score	0.874	0.828	0.666	0.614
Runtime (s)	19.2	0.9	0.2	1.1

Table 2 Comparing Various Classification Models

Chapter 5

5. Schedule, Tasks and Milestones

5.1 Project Plan

Session	Goal	Key Team Member during this phase
1	Overview of Project and Gather Data Sample	Project Leader
2	Data Exploration	Data Visual Explorer
3	Data Cleaning	Data Gatherer
4	Run Models/Evaluate Results	Model Tester
5	Refine and Choose Model	Model Tester
6	Insights/Future Recommendations	Project Leader

Table 3 Project Schedule

Session 1: (Overview of Project and Gather Data Sample)

Define the problem and create hypothesis. Show how many rows, columns. Need initial data file by now. Define independent and potential dependent variables.

Session 2: (Data Exploration)

What is the mean, median, boxplot, histogram, skewness, pareto, scatter plots, correlations, or other key metrics?

Session 3: (Data Cleaning)

Explain what calculations are needed, what filters required, how to refine problem and transform data. E.g., need to be aggregated? Predicting over what time frame? Ever or in next year? Etc. Would there be other data needed in future or now?

Session 4: (Run Models and Evaluate Results)

Create training/test data, create model, evaluate R-squared, MSE on Regression and Confusion Matrix and Error Rate on Classification

Session 5: (Refine and Choose Model)

Run several models and see which one is best.

Session 6: (Insights and Future Recommendations)

What does the model say in lay person's terms, what are the insights and what recommendations given to business?

5.2 Roles and Responsibilities

Project Leader: Spokesperson, Define the problem and hypothesis, pick the model(s), create project Plan, including start/end dates, scheduling team meetings, Provide final presentation and regular updates.

Data Gatherer: Work with D&A team or others to collect sample data on spreadsheet or from the source. Clean the data, decide how to handle blanks, outliers, filters, duplicates, calculations, etc. How to transform and filter the data after initial exploration.

Data Visual Explorer: Report Key Stats such as Median, Quartiles, Boxplot, Histogram, Skewness, Scatter Plots, Pareto, or other key metrics.

Model Tester: Create training/test data, run the model(s), evaluate the model performance metrics (e.g., R-squared, Mean Squared Error, Confusion Matrix (error rate)), p-values, coefficients, and their positive/negative interpretation.

- The role taken up by me was that of a ‘Model Tester’ where I had to work on a little bit of the pre-processing and all the Machine Learning model development, training, and testing.

- I had discussions with the Data Gatherer and Visual Explorer where I discussed various methods of Data Abstraction among other things.

- Some of these text pre-processing techniques were done by my team mate on excel. I had to do them again on Python.

- I have developed the code for vectorization, machine learning and for testing the model.

Chapter 6

6. Results and Discussion

In this section, we will discuss the accuracies and other metrics that the model has been tested against. We will also discuss the different models used for testing and which of those models was finalized.

6.1 Testing Metrics

Accuracy

Classification Accuracy is what we usually mean, when we use the term accuracy. It is the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ Number\ of\ predictions}$$

Equation 6 Accuracy General Formula

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Equation 7 Accuracy using Confusion Matrix

Where *TP* = True Positives, *TN* = True Negatives, *FP* = False Positives, and *FN* = False Negatives.

- It works well only if there are equal number of samples belonging to each class.

- For example, consider that there are 98% samples of class A and 2% samples of class B in our training set. Then our model can easily get 98% training accuracy by simply predicting every training sample belonging to class A.

- When the same model is tested on a test set with 60% samples of class A and 40% samples of class B, then the test accuracy would drop down to 60%. Classification Accuracy is great but gives us the false sense of achieving high accuracy.

- The real problem arises, when the cost of misclassification of the minor class samples are very high. If we deal with a rare but fatal disease, the cost of failing to diagnose the disease of a sick person is much higher than the cost of sending a healthy person to more tests.

Confusion Matrix

- Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model.

- Let's assume we have a binary classification problem. We have some samples belonging to two classes: YES or NO.

- Also, we have our own classifier which predicts a class for a given input sample. On testing our model on 165 samples, we get the following result.

- There are 4 important terms:

    - True Positives: The cases in which we predicted YES, and the actual output was also YES.

  - True Negatives: The cases in which we predicted NO, and the actual output was NO.
  - False Positives: The cases in which we predicted YES, and the actual output was NO.
  - False Negatives: The cases in which we predicted NO, and the actual output was YES.

R2 Score (Coefficient of Determination):

- The R2 score is a very important metric that is used to evaluate the performance of a regression-based machine learning model.
- It is pronounced as R squared and is also known as the coefficient of determination.
- It works by measuring the amount of variance in the predictions explained by the dataset.
- Simply put, it is the difference between the samples in the dataset and the predictions made by the model.
- If the value of the R2 is 1, it means that the model is perfect and if its value is 0, it means that the model will perform badly on an unseen dataset.
- R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model.
- Whereas correlation explains the strength of the relationship between an independent and dependent variable, R-squared explains to what extent the variance of one variable explains the variance of the second variable.
- So, if the R2 of a model is 0.50, then approximately half of the observed variation can be explained by the model's inputs.

$$R^2 = 1 - \frac{Unexplained\,Variation}{Total\,Variation}$$

Equation 8 R2 Score Formula

- Limitations of R^2 Score:
    - R-squared will give you an estimate of the relationship between movements of a dependent variable based on an independent variable's movements.
    - It doesn't tell you whether your chosen model is good or bad, nor will it tell you whether the data and predictions are biased.
    - A high or low R-square isn't necessarily good or bad, as it doesn't convey the reliability of the model, nor whether you've chosen the right regression.
    - You can get a low R-squared for a good model, or a high R-square for a poorly fitted model, and vice versa.

6.2 Testing the Model with existing data

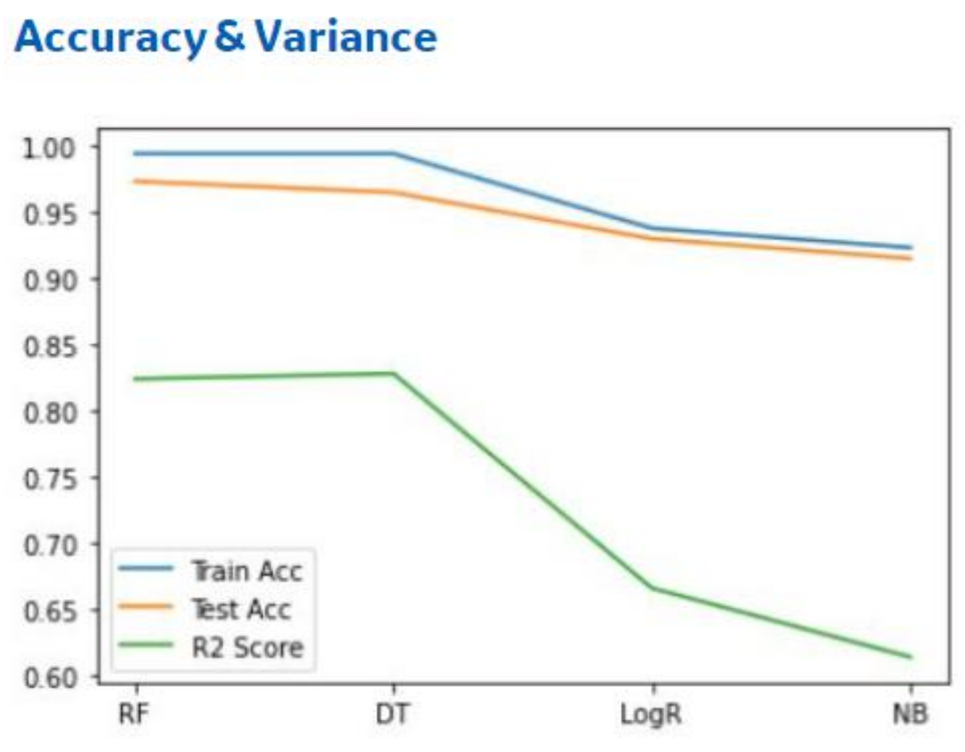


Figure 17 Accuracy and Variance

- It can be seen in the above graph that the decision tree model has the best Train, test accuracies and R2 score.
- But, since decision trees keep on adding sub-trees until it gets completely fitted to the training data, we cannot use it since it makes the model complex and doesn't improve when new data is shown to the model.
- Whereas a Naïve Bayes algorithm performs better when tested with new data.

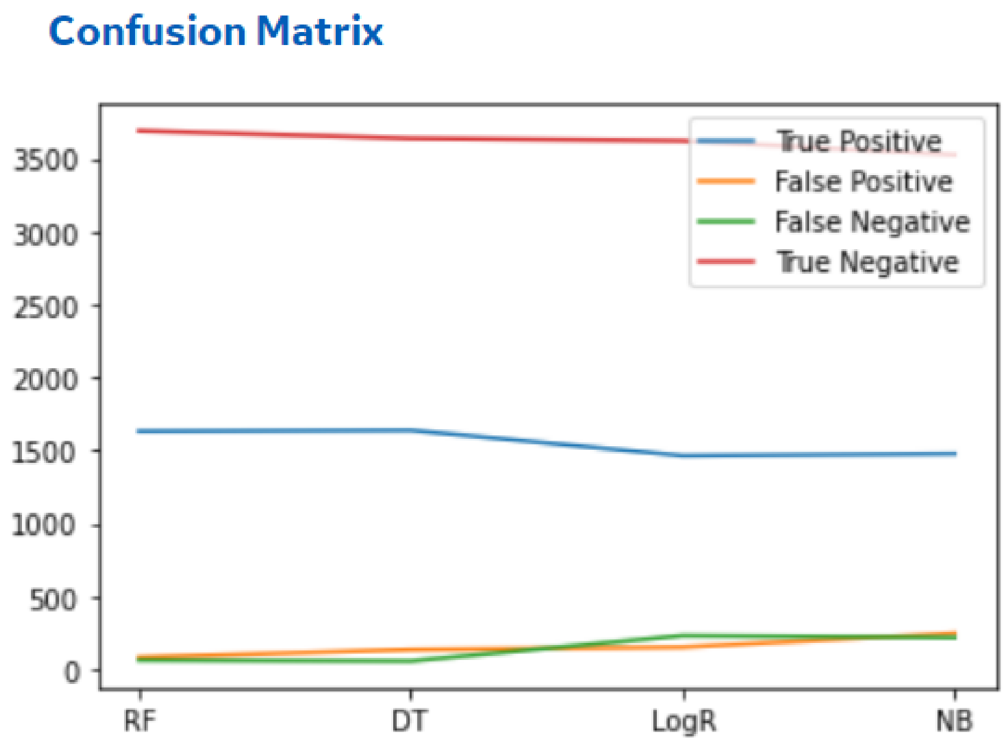


Figure 18 Confusion Matrix achieved

- It can be observed that there are very few negligible numbers of False Negatives and False Positives.
- This shows that the model is relatively free of error.
- The number of True Positives is half of that of True Negatives because the data was a 30:70 split.
- We can expect the model to perform well on new data as the model shows less signs of overfitting in case of Naïve Bayes.

6.3 Testing the model with new data (verified manually)

Confusion Matrix:

	Predicted: Not C	**Predicted: C**
Actual: Not C	608	25
Actual: C	539	807

Table 4 Confusion Matrix of New Data

- From the above confusion matrix, it can be derived that the model is performing very well where the predictions are C as there are only 25 cases where the model predicted C, but it was Not C.
- It can also be seen that the model is struggling with its Not C predictions as almost half of them are wrongly predicted.

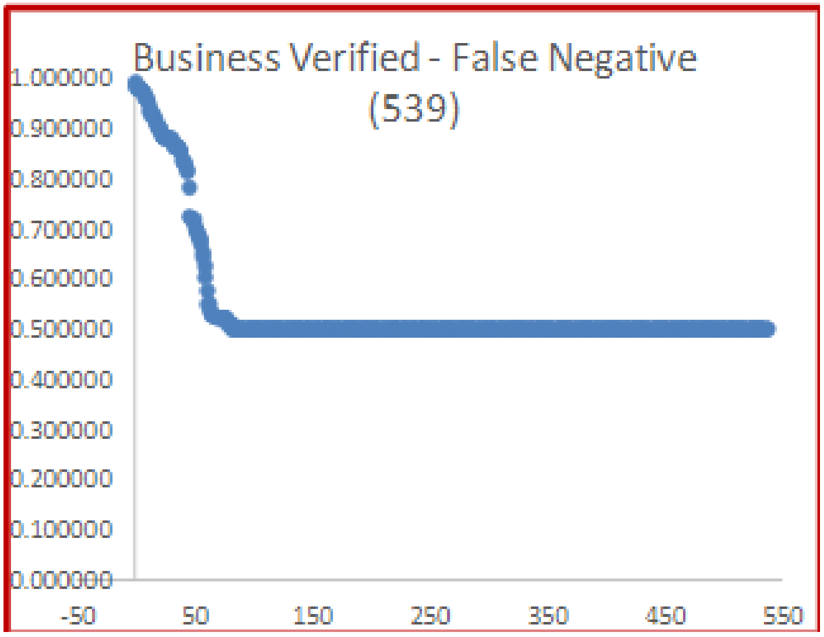


Figure 19 False Negative Model Confidence Graph

- The reason for the high number of False Negatives in the given case can be that most of the new data given to the model have completely new words with which the model is struggling since it does not have any data for those words.
- For a better generalization of the model, we have to fit it against a general corpus in which most of the words needed would be there.
- So, when a new word that hasn't come in the training data comes, there is a high chance that the new word would be in the corpus and can be used to predict.
- Very few of the 539 false negative predictions were predicted with good confidence.
- This shows that when we just use the predictions with more than 0.90 confidence the accuracy is very good and usable in the real world.

Chapter 7

7. Summary

The Automated Parts Qualification Level (PQL) project has product results which are usable for a first iteration project. Although many of the unseen data have been classified with very low confidence, most of the parts that were predicted with high confidence were correct. From the very start, the goal has been to use the predictions with good probability and send the predictions with low probability to the engineer. And this project does deliver on the expectations.

This project is estimated to save the business more than 1 million dollars in a financial year. And if we factor in the time saved on this automated classification, this project makes a lot of sense for the business.

This project was accepted by the USA Boiler Business for more testing and eventually deployment on their PLM system. The team is yet to have advanced discussions with the US boiler business and more testing is to be done on the boiler data. As we will perform training again with the boiler data, we can expect the results to be similar depending on the nature of the data.

7.1 Future Scope

Although the project has been able to satisfy the business to their needs there is a lot that can be done on the machine learning and vectorization part. Currently, the vectorization functions cannot even differentiate between words with same spelling but with different casing. And since the vectorizer doesn't recognize old words with new grammatical changes and related words, it is impossible for the ML model to predict based on the new word.

Present technologies like BERT Transformer can do both vectorization and the classification process by itself. A Transformer is a type of neural network which has two neural networks which work with each other to give results. Here, the first network vectorizes the sentence smartly where it can understand the patterns in the words. And the second transformer will take the vector produced by the first model to predict the PQL. Since both these models will be trained together as one, they achieve a different level of coherence where the first model will be able to provide the second model with the vectors that it can understand the best. Although the back propagation will be very complex to understand, these types of models are already developed and packaged in very usable libraries like PyTorch and TensorFlow.

Chapter 8

8. References

Pandas Documentation:

pandas documentation — pandas 1.4.2 documentation (pydata.org)

Scikit Learn Logistic Regression Documentation:

sklearn.linear_model.LogisticRegression — scikit-learn 1.1.1 documentation

Scikit Learn Decision Trees Documentation:

sklearn.tree.DecisionTreeClassifier — scikit-learn 1.1.1 documentation

Scikit Learn Naïve Bayes Classifier Documentation:

1.9. Naive Bayes — scikit-learn 1.1.1 documentation

Scikit Learn Multinomial Naïve Bayes Documentation:

sklearn.naive_bayes.MultinomialNB — scikit-learn 1.1.1 documentation

Scikit Learn Random Forest Classifier Documentation:

sklearn.ensemble.RandomForestClassifier — scikit-learn 1.1.1 documentation

Scikit Learn Confusion Matrix:

sklearn.metrics.confusion_matrix — scikit-learn 1.1.1 documentation

Scikit Learn R2 Score:

sklearn.metrics.r2_score — scikit-learn 1.1.1 documentation

Scikit Learn Count Vectorizer:

sklearn.feature_extraction.text.CountVectorizer — scikit-learn 1.1.1 documentation

Scikit Learn TF-IDF Vectorizer:

sklearn.feature_extraction.text.TfidfVectorizer — scikit-learn 1.1.1 documentation

NumPy Docs:

NumPy documentation — NumPy v1.22 Manual

MatPlotLib Docs:

Matplotlib documentation — Matplotlib 3.5.2 documentation

Imbalanced Learn Random Under Sampler:

RandomUnderSampler — Version 0.9.1 (imbalanced-learn.org)

NLTK Stop Words:

NLTK :: nltk.corpus