

SYMBIOSIS INSTITUTE OF BUSINESS MANAGEMENT, BENGALURU  
MBA (BA) Semester III - 2020-22

**TEXT MINING ASSIGNMENT - 1**

**Submitted By: GROUP 3**

Aastha Joshi	20020845001
Aishwarya Singh	20020845004
Harsh Tambe	20020845010
Joel Keith Pais	20020845011
Taksande Sandeep Ravindra	20020845030

---

Q.1. Perform a comprehensive study on the use cases (one use case per method) where text representation methods like one-hot encoding, count vectorizers and tf-idf will be applicable.

Answer:

**a) ONE HOT ENCODING**

Sometimes in data sets, we encounter columns that contain category elements (string values) for example, parameters Gender will have category parameters such as 'Male', and 'Female' performance on such data. These labels have no specific order of preference and also since the data is string labels, the machine learning model cannot work on such data.

Another way to solve this problem would be to put a label where we would give the number of values on these labels, for example a man and a woman marked on the map 0 and 1. as  $1 > 0$  and apparently both labels are equally important in the database. To address this, we will use the One Hot Encoding process.

In this method, we assign one separate column to the labels included in the variable, namely 'Gender'. Here either of the two labels will appear in the data. Therefore, where 'Male' appears, we assign '1' to Male 'columns' and '0' to Female 'columns'. Similarly, we do the opposite of the other side. In this way we make the data better processed for the machine to work with.

One hot code input is applied equally to data sets with more than 2 variations per category. However, it is not recommended to use one hot code for data sets with more than 15 variations.

### Example:

id	color	One Hot Encoding		
1	red	1	0	0
2	blue	0	1	0
3	green	0	0	1
4	blue	0	1	0

### Use-case:

The data set in many Machine Learning or Data Science tasks may comprise text or category values (basically non-numerical values). Color features with values such as red, orange, blue, white, and so on, for example. Breakfast, lunch, snacks, supper, tea, and other values are included in the meal plan. Most algorithms rely on numerical values to produce state-of-the-art outcomes. Consider the following dataset of bridges having column names 'bridge-types' and safety level' with values shown below:

BRIDGE-TYPE	SAFETY-LEVEL
Arch	(TEXT)
Beam	None
Truss	Low
Cantilever	Medium
Tied Arch	High
Suspension	Very-High
Cable	

Each category value is transformed into a new column with this strategy, and the column is given a 1 or 0 (true or false) value.

Considering the above two columns, this will result in binary values for each category in that column. The output after performing one-hot encoding is shown below:

BRIDGE-TYPE (TEXT)	BRIDGE-TYPE (Arch)	BRIDGE-TYPE (Beam)	BRIDGE-TYPE (Truss)	BRIDGE-TYPE (Cantilever)	BRIDGE-TYPE (Tied Arch)	BRIDGE-TYPE (Suspension)	BRIDGE-TYPE (Cable)
Arch	1	0	0	0	0	0	0
Beam	0	1	0	0	0	0	0
Truss	0	0	1	0	0	0	0
Cantilever	0	0	0	1	0	0	0
Tied Arch	0	0	0	0	1	0	0
Suspension	0	0	0	0	0	1	0
Cable	0	0	0	0	0	0	1

SAFETY-LEVEL (TEXT)	SAFETY-LEVEL (None)	SAFETY-LEVEL (Low)	SAFETY-LEVEL (Medium)	SAFETY-LEVEL (High)	SAFETY-LEVEL (Very High)
None	1	0	0	0	0
Low	0	1	0	0	0
Medium	0	0	1	0	0
High	0	0	0	1	0
Very-High	0	0	0	0	1

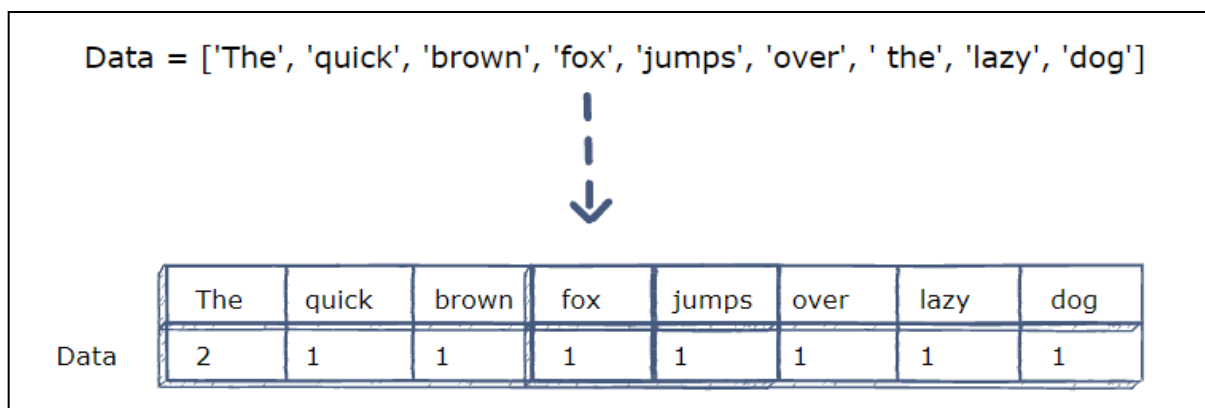
Although this method removes the hierarchy/order concerns, it has the disadvantage of increasing the number of columns in the dataset. If you have numerous unique values in a category column, the number of columns can quickly grow. It was workable in this case, but it will become quite difficult to manage when encoding provides more columns.

## b) COUNT VECTORIZER

Characters and words are incomprehensible to machines. As a result, while dealing with text data, we must express it numerically so that the computer can understand it. The **Count Vectorizer** in Scikit-learn library is used to transform a set of text documents into a vector of term or token counts. It also allows text data to be pre-processed before being converted into a vector representation. Due to this, it makes it a highly flexible feature representation module for text.

We transform raw text to a numerical vector representation of words and n-grams using Count Vectorizer. This makes it simple to utilize this format as signals (features) in Machine Learning tasks like text classification and clustering.

These algorithms only comprehend the idea of numerical features, regardless of the underlying kind of data (text, picture pixels, etc.), allowing us to execute complicated machine learning tasks on a variety of data formats.



### Parameters of Count vectorizer:

- **Lowercase:** Convert all characters to lowercase before tokenizing. Default is set to true and takes Boolean value.

- **Stopwords:** Stopwords are words that contribute little sense to a phrase in any language. They may be easily dismissed without compromising the sentence's meaning. There are three ways of dealing with stopwords - custom stopwords list, sklearn's built-in stopwords list and using `max_df` and `min_df`.
- **Max\_features:** The Count Vectorizer will choose the most commonly occurring words/features/terms. It accepts absolute values so if you set the value as 2, it will select the two most commonly occurring words in the dataset.
- **Binary:** The Count Vectorizer no longer considers the frequency of the term/word when `binary = True` is specified. If it happens, it is set to 1; otherwise, it is set to 0. Binary is set to False by default. When the count of the term/word does not offer relevant information to the machine learning model, this is generally utilized.
- **Vocabulary:** They are the collection of words in the sparse matrix. You may use the function `'get feature names()'` to get the vocabulary without the word's location in the sparse matrix.
- **Token\_pattern:** Regular expression containing a "token" is only used if `analyzer == 'word'`. The default regexp selects tokens of 2 or more alphanumeric characters. If there is a capturing group in `token_pattern` then the captured group content becomes the token. At most one capturing group is permitted.

### Use-case: Feature Extraction


The scikit-learn package in Python has a fantastic feature called Count Vectorizer. It is used to convert a text into a vector based on the frequency (count) of each word that appears throughout the text. This is useful when dealing with a large number of such texts and converting each word into a vector for further text analysis.

Count Vectorizer produces a matrix with each unique word represented by a column and each text sample from the document represented by a row. The count of the term in that particular text sample is the value of each cell.

These words are not saved as strings in Count Vectorizer. Rather, they are assigned a certain index value. The data is represented in the form of a Sparse Matrix.

	about	all	cent	cents	money	new	old	one	two
doc	1	1	3	1	1	1	1	1	1



Index	0	1	2	3	4	5	6	7	8
doc	1	1	3	1	1	1	1	1	1

The above image shows the transformation of the document containing:

```
doc = ["One Cent, Two Cents, Old Cent, New Cent: All About Money"]
```

The frequency of every word in the document is calculated and the transformed image shows the output as a sparse matrix.

### c) TF-IDF

TF-IDF stands for “Term Frequency — Inverse Document Frequency”. This is a technique to quantify words in a set of documents. We generally compute a score for each word to signify its importance in the document and corpus. This method is a widely used technique in Information Retrieval and Text Mining.

For example, let us consider this sentence, “I have a many cloths”. It's easy for us to understand the sentence as we know the semantics of the words and the sentence. But a programming language like python might not understand this. It is easier for any programming language to understand textual data in the form of numerical value. That's why we need to convert all the texts into numbers and vectorize them for the program to understand.

#### Term Frequency (TF)

Term Frequency measures the frequency of a word in the document. However, the frequency of the terms (or words) will depend on the length of the document and thus cannot be directly used to interpret the importance of those words in the document. Therefore, consider the relative frequency of those terms.

So, for now, if we are to compare a document to other sets of documents, then we cannot consider the words that are only present in that document. Instead we will have to consider all the possible words that can appear in the document.

From here onwards, we represent the frequency of a particular word in the document as the number of times the word appears divided by the total number of words. So, for example if a particular word does not appear at all, then the value would be zero, whereas if the entire document is made up of one word, then the value of that word will be one.

But still, we cannot define the importance of those words based on the frequency, as there would be other words such as ‘is’, ‘the’ etcetera that appear for a higher number of times, but have less significance. Such words are called ‘stop words’. Here we use Inverse document frequency.

## Inverse document frequency (IDF)

Before finding out the inverse of the document frequency, we find out the document frequency of a particular word. So, in the corpus of all the documents we have, we check if the word appears at least once, and count the number of such documents.

Now we understand the words like 'is' and 'the' that we considered in the above example are going to appear the greatest number of times. So instead of allocating a higher significance, they should be considered to have less importance. Accordingly, we inverse the document frequency to find the IDF.

The TF and IDF are further operated to find the TF-IDF score and used to interpret the significance of words in the document.

## Use-case: Search Engine

TF IDF is a metric that indicates how significant a word is in a manuscript. The TF-IDF is primarily used for document search and retrieval, with the score indicating the significance of a word in a document. The word becomes rarer as the TFIDF score rises, and vice versa.

One entity who has heavily deployed TF - IDF is Google search engine. Although Google has adopted newer methods for ranking mechanisms. Apart from Google, many other developers too use TF-IDF for search engine optimization. TF IDF has been used for a long time for determining the topical significance of a word or a set of words in a search query.

Google checks the pages in its index against a number of specific attributes it finds relevant to the query to determine how relevant a page is. Because the majority of internet material is text, these characteristics are most likely the presence or lack of specific terms and phrases on the page. Not only are they present, but they are prominent on this page in comparison to other pages on the internet. The TF-IDF method may be useful in this situation. It establishes a standard for stop words to provide even more importance by measuring the average use frequency for this term throughout the entire web.

The TF-IDF is a tool for improving the semantic search relevancy of your pages. It assists in looking beyond particular keywords and into content to ensure that it is related to the search topic. The TF-IDF is used by search engines as part of their ranking algorithms. Keywords are at the heart of all rankings. For one term, a page may rank highly, whereas for another, it may rank poorly or not at all. When ranking sites for a keyword, search engines analyze TF-IDF scores among hundreds of other parameters. A high TF-IDF score shows that the term is related to the page and important. It signifies that the keyword appears several times on the page and is not a popular word seen on millions of other websites. As a result, pages with a high TF-IDF score tend to appear higher in organic search results than pages with a low TF-IDF score. TF-IDF is also used by search

engines for natural language processing. Search engines will analyze the context in which words are used on a website using natural language processing. Search engines can use the TF-IDF to discern between significant and irrelevant words. Even if a term appears dozens of times on a page, search engines will disregard it if the keyword has a high inverse document frequency. Search engines can then more properly process the words on the page to determine what the page is about as a whole.

Q.2. Run any one of the methods in Python on a selected text data frame to show the output from these methods

Answer:

The data set we have considered is a movie\_review dataset taken from Kaggle. It contains 64,720 rows and 3 columns. This is a balanced dataset as the number of positive sentiments (32,937) is almost equal to the number of negative sentiments (31,783). We have taken only the 'review' column from the dataset and performed a count vectorizer on it. The screenshot of the output has been attached below.

```
In [15]: print(df.shape)
df.head(2)

(64720, 3)

Out[15]:
```

	text	tag	text_clean
0	films adapted from comic books have had plenty...	pos	films adapted from comic books have had plenty...
1	for starters , it was created by alan moore ( ...	pos	for starters it was created by alan moore and ...

```
In [16]: df['tag'].value_counts()

Out[16]: pos    32937
neg    31783
Name: tag, dtype: int64
```

Output of Count Vectorizer

```
Count vectoriser

In [9]: from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import KMeans

In [10]: CountVec = CountVectorizer(ngram_range=(1,1),
stop_words='english')

Count_data = CountVec.fit_transform(df.text_clean)
Count_data

Out[10]: <64720x38580 sparse matrix of type '<class 'numpy.int64'>'
with 621681 stored elements in Compressed Sparse Row format>

In [11]: Count_data.toarray()

Out[11]: array([[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0],
...,
[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0]])
```

# APPENDIX

## Loading Libararies

In [1]:

```
import re
import numpy as np
import pandas as pd
```

## Loading Data

In [2]:

```
df = pd.read_csv('movie_review.csv')
```

In [3]:

```
print(df.shape)
df.head(2)
```

(64720, 6)

Out[3]:

	fold_id	cv_tag	html_id	sent_id	text	tag
0	0	cv000	29590	0	films adapted from comic books have had plenty...	pos
1	0	cv000	29590	1	for starters , it was created by alan moore ( ...	pos

In [4]:

```
print(list(df.columns))
```

['fold\_id', 'cv\_tag', 'html\_id', 'sent\_id', 'text', 'tag']

In [5]:

```
df.dtypes
```

Out[5]:

```
fold_id      int64
cv_tag       object
html_id      int64
sent_id      int64
text         object
tag          object
dtype: object
```



In [6]:

```
df.drop(columns = ['fold_id', 'cv_tag', 'html_id', 'sent_id'], inplace = True)
df.head()
```

Out[6]:

	text	tag
0	films adapted from comic books have had plenty...	pos
1	for starters , it was created by alan moore ( ...	pos
2	to say moore and campbell thoroughly researche...	pos
3	the book ( or " graphic novel , " if you will ...	pos
4	in other words , don't dismiss this film becau...	pos

In [7]:

```
df['tag'].value_counts() ## Data Balance check
```

Out[7]:

```
pos    32937
neg    31783
Name: tag, dtype: int64
```

## Cleaning Data

In [8]:

```
df['text_clean'] = df['text'].map(lambda x: re.sub('[^a-zA-Z]', ' ', x))
df['text_clean'] = df['text_clean'].map(lambda x: re.sub(' +', ' ', x))
df['text_clean'] = df['text_clean'].map(lambda x: x.lower())
df.head(2)
```

Out[8]:

	text	tag	text_clean
0	films adapted from comic books have had plenty...	pos	films adapted from comic books have had plenty...
1	for starters , it was created by alan moore ( ...	pos	for starters it was created by alan moore and ...

## Count vectoriser

In [9]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import KMeans
```

In [10]:

```
CountVec = CountVectorizer(ngram_range=(1,1),
                           stop_words='english')

Count_data = CountVec.fit_transform(df.text_clean)
Count_data
```

Out[10]:

```
<64720x38580 sparse matrix of type '<class 'numpy.int64'>'
  with 621681 stored elements in Compressed Sparse Row format>
```

In [11]:

```
Count_data.toarray()
```

Out[11]:

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```