Gaussian Mixture Models

Mathurin, Fanta, Fatoumata, Fonteh

Outline

- Model Overview
- Methodology
- Conclusion

Model Overview

Problem Statement

In quality control, products leaving productions chains must have specific features to be acceptable by the customers. Failure to ensure this, will lead to customer dissatisfaction and hence low profits. After performing this task for long, it is realised that the products of good quality can be classified into 2 main groups (K=2).

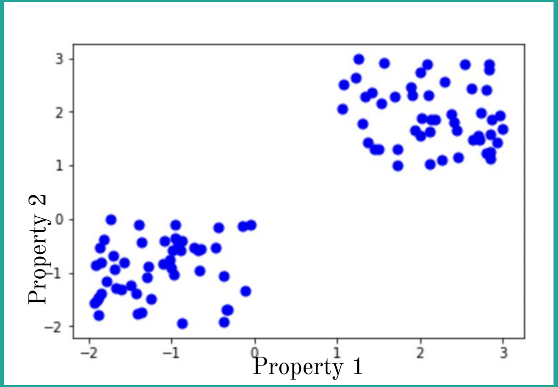


Figure 1: Plot of products based on some properties

Problem Statement (ctd)

With these observations, we want to learn the probability density function from which these observations are sampled. With this density function, we can only accept products with probability ($\mathbb{p}$(product) $>\varepsilon$), ε chosen by the quality control team. Hence, our goal is to model this probability distribution.

Mathematical Modeling

We suppose there is a latent(hidden/unobserved) random variable z and $z^{(i)}$, $x^{(i)}$ are distributed

$$\mathbb{p}(x^{(i)},z^{(i)}) = \mathbb{p}(x^{(i)}/z^{(i)})\mathbb{p}(z^{(i)}) \qquad (1)$$

Assumptions

- $(x^{(i)},z^{(i)}) \sim N(\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)$
- $z^{(i)} \sim \mathrm{Mult}(\pi)$, $z^{(i)} \in \{1, \dots , k\}$

Mathematical Modeling (MLE)

With the above assumptions, it follows that;

$$p(x) = \sum^{K}_{k=1} p(x^{(i)}/z^{(i)})p(z^{(i)}) = \sum^{K}_{k=1} \pi_k N(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \qquad (2)$$

Knowing that x(i) $i \in \{1, \ldots, n\}$ are iid and applying MLE, we have

$$\begin{aligned} \log(p(x; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})) &= \sum^{n}_{i=1} \log(\sum^{K}_{k=1} p(x^{(i)}/z^{(i)})p(z^{(i)})) \\ &= \sum^{n}_{i=1} \log(\sum^{K}_{k=1} \pi_k N(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)) \qquad (3) \end{aligned}$$

Challenges

- The values of $z^{(i)}$ are not known. So, we can not estimate the optimum parameters in a straightforward manner
- Also, the log of the sum could have some singularities.

Mathematical Modeling (MLE)

With the above assumptions, it follows that;

$$p(x) = \sum^{K}_{k=1} p(x^{(i)}/z^{(i)})p(z^{(i)}) = \sum^{K}_{k=1} \pi_k N(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \qquad (2)$$

Knowing that x(i) $i \in \{1, \dots, n\}$ are iid and applying MLE, we have

$$\log(p(x;\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma})) = \sum^{n}_{i=1} \log(\sum^{K}_{k=1} p(x^{(i)}/z^{(i)})p(z^{(i)}))$$

$$= \sum^{n}_{i=1} \log(\sum^{K}_{k=1} \pi_k N(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)) \qquad (3)$$

Challenges

- The values of $z^{(i)}$ are not known. So, we can not estimate the optimum parameters in a straightforward manner
- Also, the log of the sum could have some singularities.

Parameters Estimation (Expectation Maximization Algorithm)

Since we cannot estimate the parameters of the model in a using the MLE method, we resort to the expectation maximization algorithm which works in the following way.

- **E-step**

  Set $\mathbf{r}_{ij} = \boldsymbol{\pi}_j N(\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j) / \sum^{K}_{k=1} \pi_k N(\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)$

  $\mathbf{r}_{ij}$ **represents how much x(i) is assigned to the** $\boldsymbol{\mu}_j$ Gaussian (The strength of the assignment to the Gaussian).

Expectation Maximization Algorithm (ctd)

Since we cannot estimate the parameters of the model in a using the MLE method, we resort to the expectation maximization algorithm which works in the following way.

- **M-step**

$\boldsymbol{\pi}_j = 1/n \sum^n_{i=1} r_{ij}$

$\boldsymbol{\mu}_j = \sum^n_{i=1} r_{ij}\, x^{(i)} \,/\, \sum^n_{i=1} r_{ij}$

$\boldsymbol{\Sigma}_j = \sum^n_{i=1} r_{ij}\, (x^{(i)} - \boldsymbol{\mu}_j)(x^{(i)} - \boldsymbol{\mu}_j)^T / \sum^n_{i=1} r_{ij}$

Step E and M are repeated until convergence.

Methodology

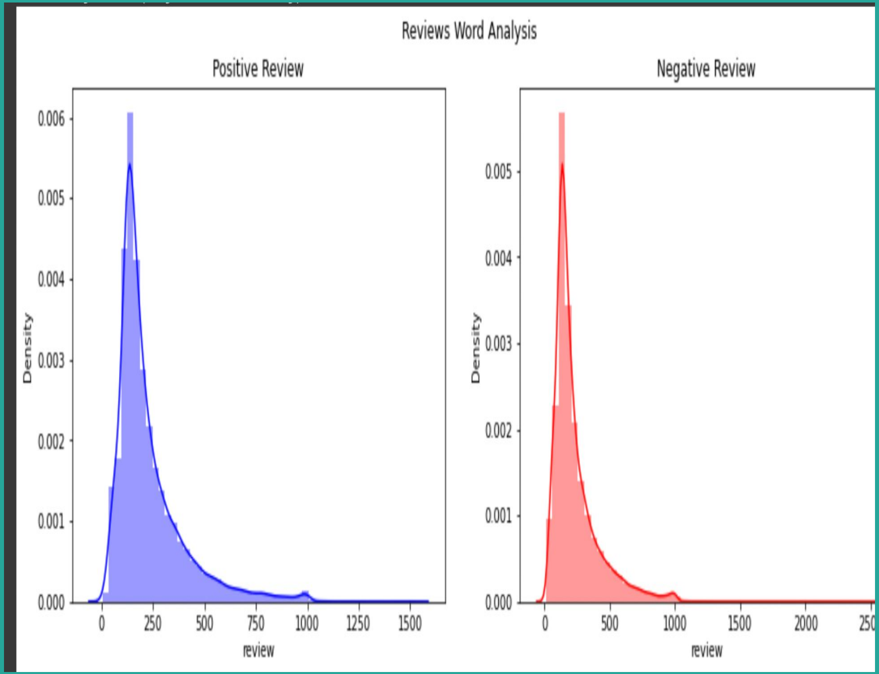
Data Preprocessing

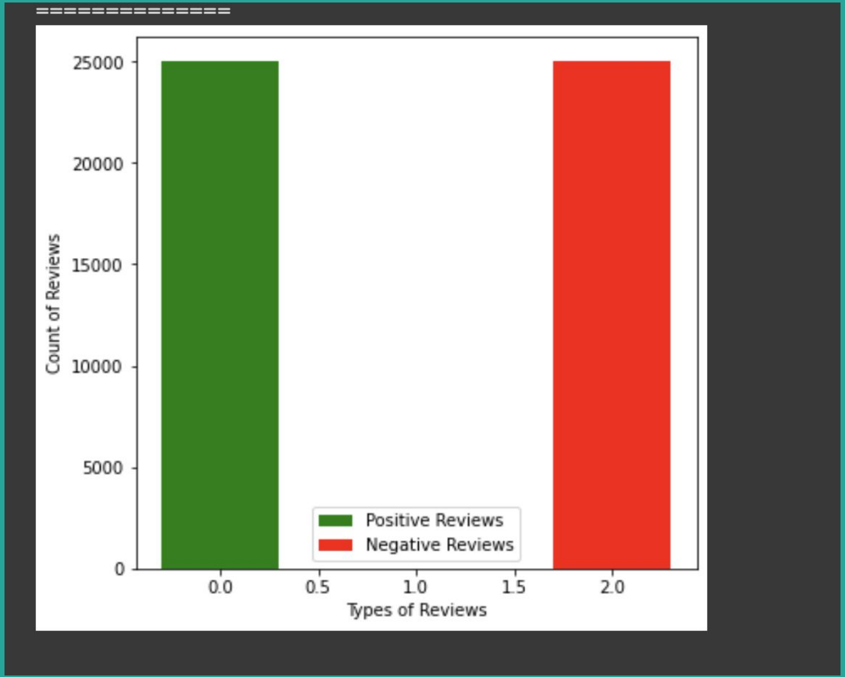
Dataset Description

Our database contains a "review" column containing textual information (input features) and the "sentiment" column containing output labels. The task of our classifier is to correctly predict the "sentiment" from any review or textual column. We must therefore apply our data cleaning and transformation steps to the "review" column.

Statistical Analysis

This is the beginning of the analysis phase where we will first check the amount of data present. We will continue with pictorial representations related to the words and frequency correspondences.


Reviews Word Analysis
Positive Review
Negative Review
Density
review
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0
250
500
750
1000
1250
1500
2000

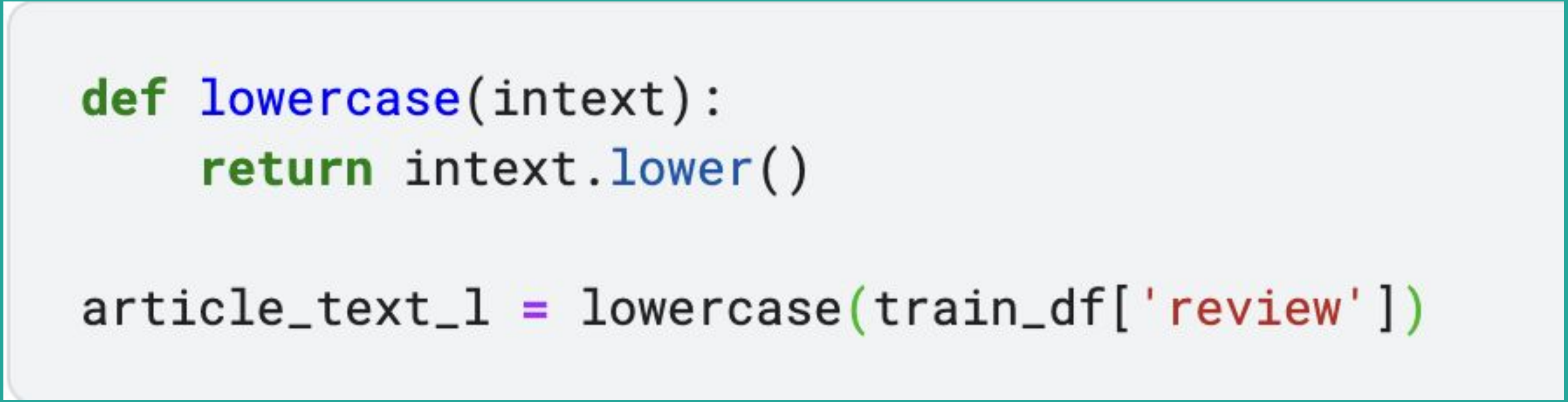

Count of Reviews
Types of Reviews
Positive Reviews
Negative Reviews
0
5000
10000
15000
20000
25000
0.0
0.5
1.0
1.5
2.0

Word cloud

A word cloud is a visual representation of words. Cloud creators are used to highlight popular words and phrases based on frequency and relevance. They provide us with quick and simple visual insights that can lead to more in-depth analyses.

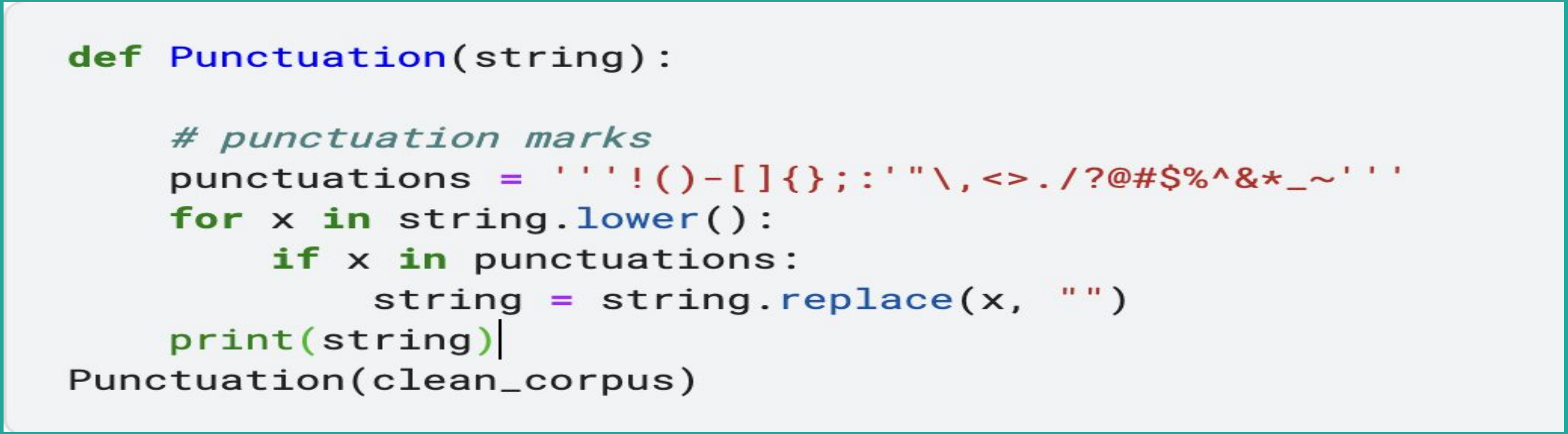
Changing Text to Lowercase

We transform the input text to lowercase using the script snippet below;



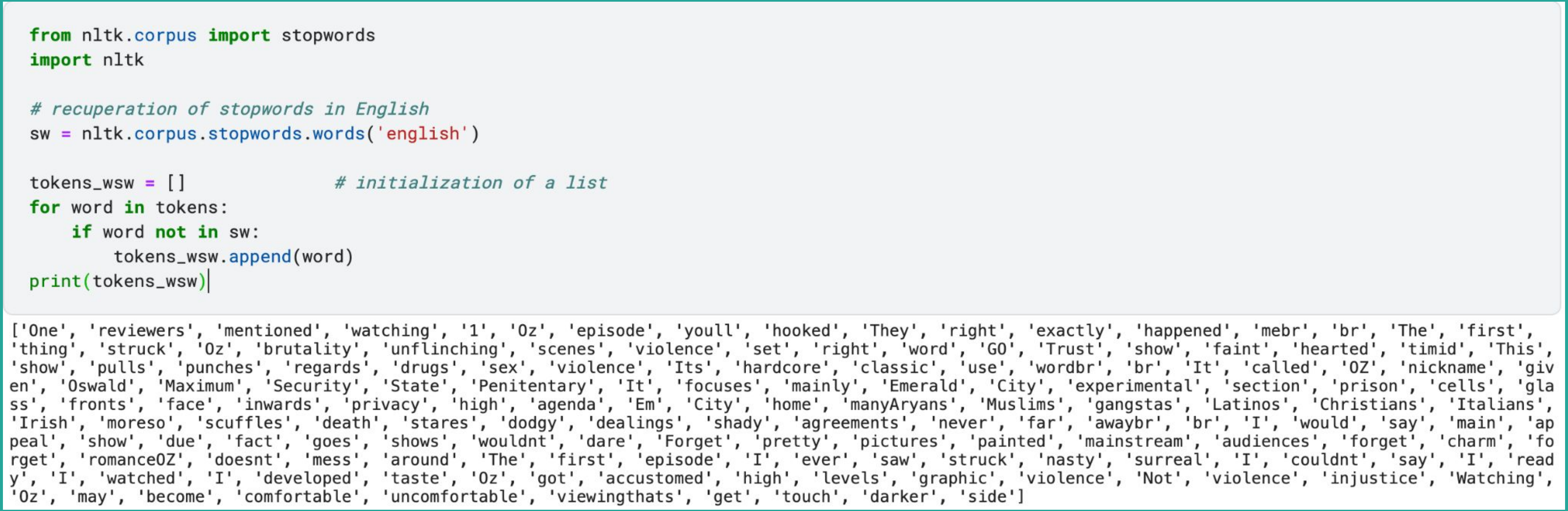
```
def lowercase(intext):
    return intext.lower()

article_text_l = lowercase(train_df['review'])
```

Removal of all punctuation in our database.



```
def Punctuation(string):

    # punctuation marks
    punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
    for x in string.lower():
        if x in punctuations:
            string = string.replace(x, "")
    print(string)
Punctuation(clean_corpus)
```

Stopwords are the most common words in any natural language. For the purpose of analyzing text data and building NLP models, these stopwords might not add much value to the meaning of the document. Generally, the most common words used in a text are “the”, “is”, “in”, “for”, “where”, “when”, “to”, “at” etc.

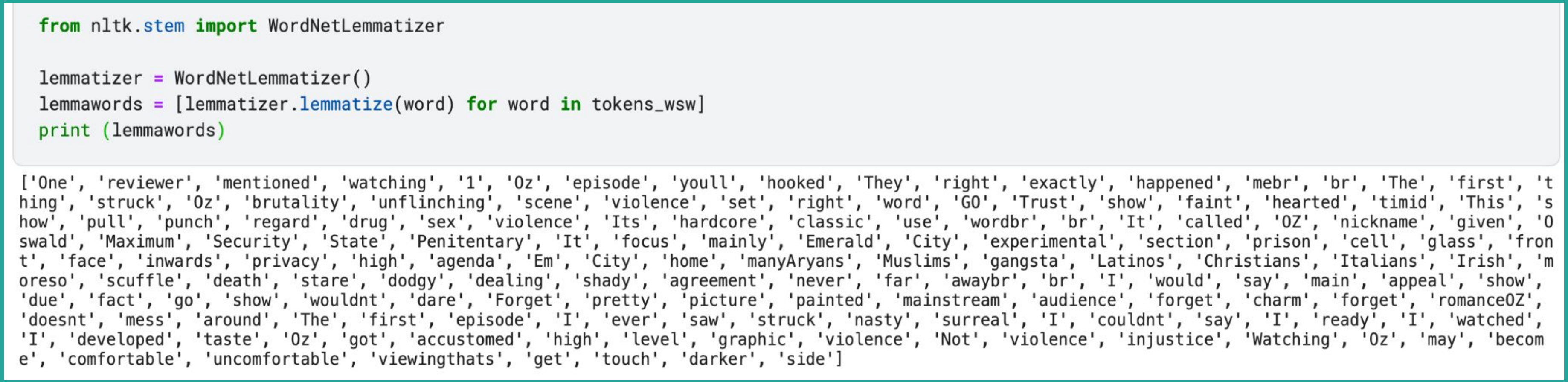


```python
from nltk.corpus import stopwords
import nltk

# recuperation of stopwords in English
sw = nltk.corpus.stopwords.words('english')

tokens_wsw = []             # initialization of a list
for word in tokens:
    if word not in sw:
        tokens_wsw.append(word)
print(tokens_wsw)
```

```
['One', 'reviewers', 'mentioned', 'watching', '1', 'Oz', 'episode', 'youll', 'hooked', 'They', 'right', 'exactly', 'happened', 'mebr', 'br', 'The', 'first',
'thing', 'struck', 'Oz', 'brutality', 'unflinching', 'scenes', 'violence', 'set', 'right', 'word', 'GO', 'Trust', 'show', 'faint', 'hearted', 'timid', 'This',
'show', 'pulls', 'punches', 'regards', 'drugs', 'sex', 'violence', 'Its', 'hardcore', 'classic', 'use', 'wordbr', 'br', 'It', 'called', 'OZ', 'nickname', 'giv
en', 'Oswald', 'Maximum', 'Security', 'State', 'Penitentary', 'It', 'focuses', 'mainly', 'Emerald', 'City', 'experimental', 'section', 'prison', 'cells', 'gla
ss', 'fronts', 'face', 'inwards', 'privacy', 'high', 'agenda', 'Em', 'City', 'home', 'manyAryans', 'Muslims', 'gangstas', 'Latinos', 'Christians', 'Italians',
'Irish', 'moreso', 'scuffles', 'death', 'stares', 'dodgy', 'dealings', 'shady', 'agreements', 'never', 'far', 'awaybr', 'br', 'I', 'would', 'say', 'main', 'ap
peal', 'show', 'due', 'fact', 'goes', 'shows', 'wouldnt', 'dare', 'Forget', 'pretty', 'pictures', 'painted', 'mainstream', 'audiences', 'forget', 'charm', 'fo
rget', 'romanceOZ', 'doesnt', 'mess', 'around', 'The', 'first', 'episode', 'I', 'ever', 'saw', 'struck', 'nasty', 'surreal', 'I', 'couldnt', 'say', 'I', 'read
y', 'I', 'watched', 'I', 'developed', 'taste', 'Oz', 'got', 'accustomed', 'high', 'levels', 'graphic', 'violence', 'Not', 'violence', 'injustice', 'Watching',
'Oz', 'may', 'become', 'comfortable', 'uncomfortable', 'viewingthats', 'get', 'touch', 'darker', 'side']
```

Lemmatization

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma



```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
lemmawords = [lemmatizer.lemmatize(word) for word in tokens_wsw]
print (lemmawords)
```

```
['One', 'reviewer', 'mentioned', 'watching', '1', 'Oz', 'episode', 'youll', 'hooked', 'They', 'right', 'exactly', 'happened', 'mebr', 'br', 'The', 'first', 't
hing', 'struck', 'Oz', 'brutality', 'unflinching', 'scene', 'violence', 'set', 'right', 'word', 'GO', 'Trust', 'show', 'faint', 'hearted', 'timid', 'This', 's
how', 'pull', 'punch', 'regard', 'drug', 'sex', 'violence', 'Its', 'hardcore', 'classic', 'use', 'wordbr', 'br', 'It', 'called', 'OZ', 'nickname', 'given', 'O
swald', 'Maximum', 'Security', 'State', 'Penitentary', 'It', 'focus', 'mainly', 'Emerald', 'City', 'experimental', 'section', 'prison', 'cell', 'glass', 'fron
t', 'face', 'inwards', 'privacy', 'high', 'agenda', 'Em', 'City', 'home', 'manyAryans', 'Muslims', 'gangsta', 'Latinos', 'Christians', 'Italians', 'Irish', 'm
oreso', 'scuffle', 'death', 'stare', 'dodgy', 'dealing', 'shady', 'agreement', 'never', 'far', 'awaybr', 'br', 'I', 'would', 'say', 'main', 'appeal', 'show',
'due', 'fact', 'go', 'show', 'wouldnt', 'dare', 'Forget', 'pretty', 'picture', 'painted', 'mainstream', 'audience', 'forget', 'charm', 'forget', 'romanceOZ',
'doesnt', 'mess', 'around', 'The', 'first', 'episode', 'I', 'ever', 'saw', 'struck', 'nasty', 'surreal', 'I', 'couldnt', 'say', 'I', 'ready', 'I', 'watched',
'I', 'developed', 'taste', 'Oz', 'got', 'accustomed', 'high', 'level', 'graphic', 'violence', 'Not', 'violence', 'injustice', 'Watching', 'Oz', 'may', 'becom
e', 'comfortable', 'uncomfortable', 'viewingthats', 'get', 'touch', 'darker', 'side']
```

Term Frequency — Inverse Document Frequency (tf-idf)

Tf-idf 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. Tf-idf is the product of two terms (tf and idf)

Tf-idf = term frequency (**tf**) * inverse document frequency(**idf**)

Term Frequency — Inverse Document Frequency (tf-idf)

Term frequency (tf) measures the frequency of a word in a document. We need the word counts of all the vocab words and the length of the document to compute tf.

$$\mathbf{tf(t,d) = \log(1 + f(t,d))} \qquad \textbf{(4)}$$

where f(t,d) =count of t in d / number of words in d

Document frequency (df) is the count of occurrences of term t in the document set N. In other words, DF is the number of documents in which the word is present.

$$\mathbf{df(t) = \text{occurrence of t in N documents}} \qquad \textbf{(5)}$$

Inverse Document Frequency (idf)

idf is the inverse of the document frequency which measures the informativeness of term t.

$$\mathrm{idf} = N/\mathrm{df}(t) \qquad (6)$$

Because of the large number of documents in many collections, this measure too is usually squashed with a log function.

$$\mathrm{idf} = \log(N/\mathrm{df}(t)) \qquad (7)$$

Count Vectorizer

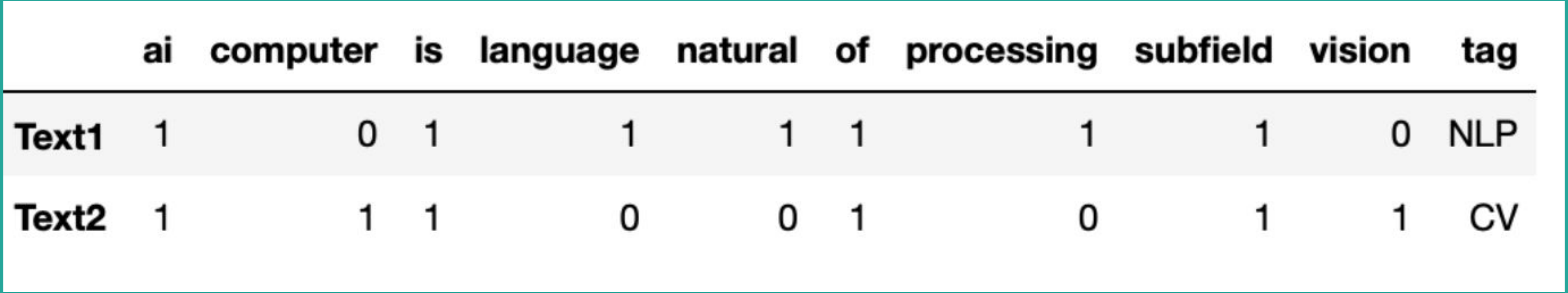
A countvectorizer or count frequency is based on term frequencies and is capable of squeezing an entire sentence into a single vector. Each position of a count vector is assigned to a particular token as before and its value represents the number of appearances that token has in the sentence.

Step 1: Convert each document into a sequence of words containing that document.

Step 2: From the set of all the words in the corpus, count how often the word occurs in the document.

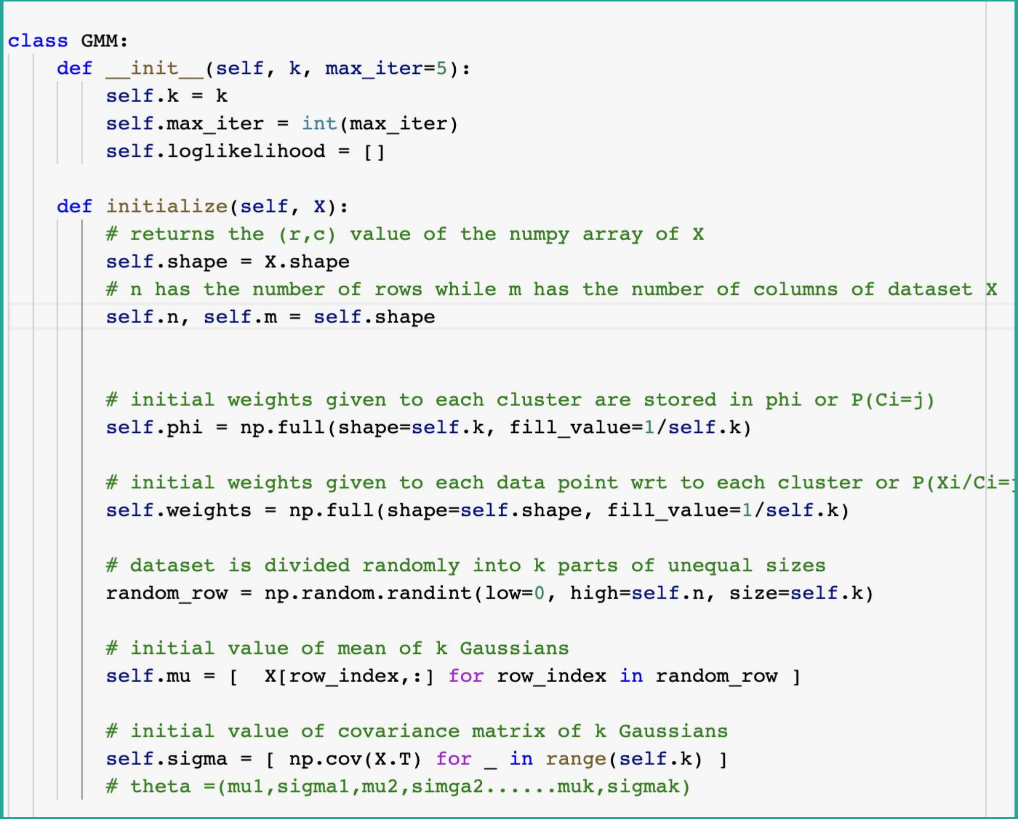
Let’s take this example

```
Text1 = “Natural Language Processing is a subfield of AI”
tag1 = "NLP"

Text2 = “Computer Vision is a subfield of AI”
tag2 = "CV"
```



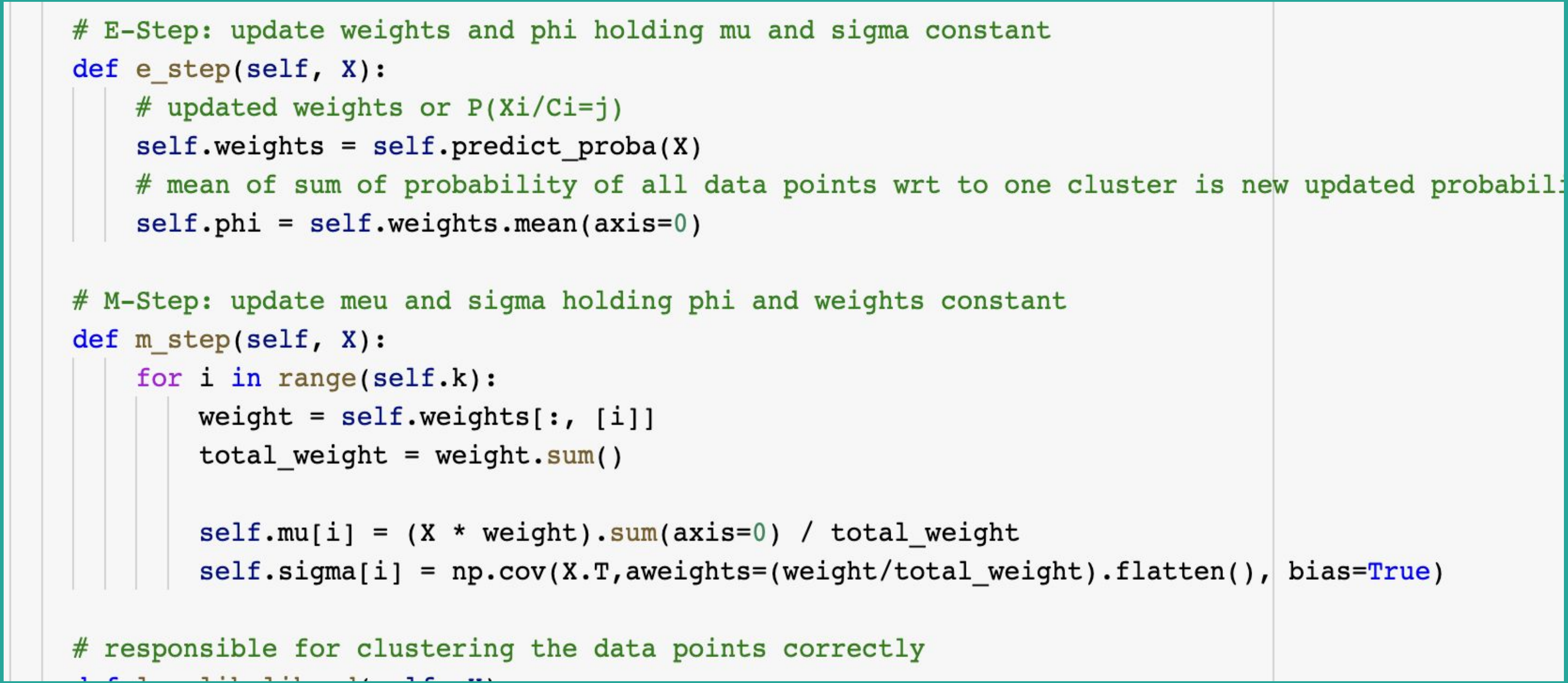
	ai	computer	is	language	natural	of	processing	subfield	vision	tag
Text1	1	0	1	1	1	1	1	1	0	NLP
Text2	1	1	1	0	0	1	0	1	1	CV

Class Implementation



```
class GMM:
    def __init__(self, k, max_iter=5):
        self.k = k
        self.max_iter = int(max_iter)
        self.loglikelihood = []

    def initialize(self, X):
        # returns the (r,c) value of the numpy array of X
        self.shape = X.shape
        # n has the number of rows while m has the number of columns of dataset X
        self.n, self.m = self.shape


        # initial weights given to each cluster are stored in phi or P(Ci=j)
        self.phi = np.full(shape=self.k, fill_value=1/self.k)

        # initial weights given to each data point wrt to each cluster or P(Xi/Ci=
        self.weights = np.full(shape=self.shape, fill_value=1/self.k)

        # dataset is divided randomly into k parts of unequal sizes
        random_row = np.random.randint(low=0, high=self.n, size=self.k)

        # initial value of mean of k Gaussians
        self.mu = [  X[row_index,:] for row_index in random_row ]

        # initial value of covariance matrix of k Gaussians
        self.sigma = [ np.cov(X.T) for _ in range(self.k) ]
        # theta =(mu1,sigma1,mu2,simga2......muk,sigmak)
```

Class Implementation



```
# E-Step: update weights and phi holding mu and sigma constant
def e_step(self, X):
    # updated weights or P(Xi/Ci=j)
    self.weights = self.predict_proba(X)
    # mean of sum of probability of all data points wrt to one cluster is new updated probabili
    self.phi = self.weights.mean(axis=0)

# M-Step: update meu and sigma holding phi and weights constant
def m_step(self, X):
    for i in range(self.k):
        weight = self.weights[:, [i]]
        total_weight = weight.sum()

        self.mu[i] = (X * weight).sum(axis=0) / total_weight
        self.sigma[i] = np.cov(X.T,aweights=(weight/total_weight).flatten(), bias=True)

# responsible for clustering the data points correctly
```

Thanks for Your Keen Attention!

References

[1] Unsupervised Learning, Lester Mackey, Standford University

[2] Pattern Recognition and Machine learning, Christopher M. Bishop page 428 -436

[3] Categorical Variable Encoding Techniques

[4] An Overview for Text Representations in NLP

[5] Text data representation with one-hot encoding, Tf-Idf, Count Vectors, Co-occurrence Vectors and Word2Vec