# Natural Language Processing Assignment 5 - Bag of Words (BOW) Models

## Delhi Technological University - Dr. Seba Susan

10th September 2020

Anish Sachdeva
DTU/2K16/MC/13
Delhi Technological University

## Overview

## Introduction

In many applications where we use our words as input in Machine Learning Models or in Deep Learning etc. we can't directly use our words as text and character input as machines can't perform numerical and analytical tasks directly on character sequences and perform better when given numerical input.

To rectify this we convert words into vectors and then use the techniques of Linear Algebra and Optimization which are readily available to us to work on our data. We can convert words into vectors using many different methods and there are already many different data sources available online that provide us with pre-computed vectors for words.

A problem with modeling text is that it is messy, and techniques like Machine Learning algorithms prefer well defined fixed-length inputs and outputs.

Machine Learning Algorithms cannot work with raw textual data directory. The text must be converted into numbers of more specifically vectors. This is called feature extraction or feature encoding.

A bag-of-words model or a BoW Model is a way of extracting features from text for use with modelling such as in Machine Learning.

In this project we cover 3 different models namely:

1. The One Hot Vector Model
2. The Term Frequency (TF) Model
3. The Term Frequency Inverse Document Frequency (TF-IDF) Model

## One Hot Vectors

In this method the words are represented as Binary 0 or 1 where 1 means that the word occurs in a particular document and 0 that it doesn't occur in a particular document.

### Implementation [See on GitHub]

```
from collections import Counter

import numpy as np
import pandas
import pprint

# noinspection PyUnresolvedReferences
from utils import tokenize

# importing corpus as resume
resume_file = open('../assets/resume.txt', 'r')
resume = resume_file.read().lower()
resume_file.close()

# tokenizing the resume
tokens = tokenize(resume)

# dividing corpus into 6 documents
k = len(tokens) // 6
documents = []
for i in range(5):
   documents.append(tokens[i * k: (i + 1) * k])
documents.append(tokens[5 * k:])

# calculating most common 5 tokens from each document
```

```
most_common = set()
for document in documents:
    frequencies = Counter(document)
    for word, frequency in frequencies.most_common(5):
        most_common.add(word)

# creating one hot vector for each word in most common
vectors = {}
for word in most_common:
    vector = np.zeros((6), dtype=int)
    for index, document in enumerate(documents):
        vector[index] = word in document
    vectors[word] = vector

pprint.pp(vectors)

# one hot vector representation
table = pandas.DataFrame(data=vectors)

# writing the table in a text file to view output
file = open('../assets/one-hot-vector.txt', 'w')
file.write(table.to_string())
file.close()
```

## Output [See on GitHub]

```
{'applications': [0, 0, 1, 0, 0, 0],
 'students': [1, 0, 0, 1, 0, 0],
 'data': [1, 1, 0, 1, 1, 1],
 'geometry': [0, 1, 0, 0, 0, 0],
 'structures': [1, 0, 0, 0, 1, 1],
 'trinity': [0, 0, 0, 0, 0, 1],
 'also': [1, 0, 1, 1, 0, 0],
 'group': [0, 1, 1, 0, 0, 0],
 'london': [0, 0, 0, 0, 0, 1],
 'computer': [0, 0, 0, 0, 1, 0],
 'algorithms': [1, 1, 0, 0, 1, 1],
 'worked': [0, 1, 1, 1, 0, 0],
 'many': [0, 1, 0, 1, 0, 0],
 'auckland': [0, 1, 0, 0, 0, 1],
 'com': [1, 1, 0, 0, 0, 1],
 'university': [0, 1, 0, 0, 1, 1],
 'guitar': [0, 0, 0, 0, 0, 1],
 'plectrum': [0, 0, 0, 0, 0, 1],
 'mathematics': [0, 1, 0, 0, 1, 1],
 'theory': [0, 1, 1, 0, 0, 0],
 'college': [1, 0, 0, 0, 0, 1],
 'requests': [0, 0, 0, 1, 0, 0],
 'participated': [0, 0, 0, 1, 0, 0],
 'cern': [0, 0, 1, 1, 0, 1],
 'python': [1, 0, 0, 0, 1, 0],
 'java': [1, 1, 1, 1, 1, 1],
 'research': [0, 1, 1, 0, 0, 1]}
```

## Term Frequency (TF) Model

The term frequency model measure the frequency of a given *word* in a document and the vector for word *w* is denoted by *V* such that $V_i = freq(w, doc_i)$

### Implementation [See on GitHub]

```
# implementing the term frequency vectors

import pprint
from collections import Counter

import nltk
import numpy as np
import pandas

# noinspection PyUnresolvedReferences
from utils import tokenize

nltk.download('stopwords')

# importing corpus as resume
resume_file = open('../assets/resume.txt', 'r')
resume = resume_file.read().lower()
resume_file.close()

# tokenizing the resume
tokens = tokenize(resume)

# dividing corpus into 6 documents
k = len(tokens) // 6
documents = []
for i in range(5):
   documents.append(tokens[i * k: (i + 1) * k])
```

```python
documents.append(tokens[5 * k:])

# calculating most common 5 tokens from each document and storing frequency
tables for each document
most_common = set()
document_frequencies = []
for document in documents:
   frequencies = Counter(document)
   document_frequencies.append(frequencies)
   for word, frequency in frequencies.most_common(5):
       most_common.add(word)

# Calculating the term frequency vectors
vectors = {}
for word in most_common:
   vector = np.zeros((6), dtype=int)
   for index, frequencies in enumerate(document_frequencies):
       vector[index] = frequencies[word]
   vectors[word] = vector
pprint.pp(vectors)

# creating the table representation of words & vectors
table = pandas.DataFrame(data=vectors)

# storing the table in text file to view output
output_file = open('../assets/tf.txt', 'w')
output_file.write(table.to_string())
output_file.close()
```

## Output [See on GitHub]

```
{'geometry': array([0, 3, 0, 0, 0, 0]),
 'university': array([0, 2, 0, 0, 2, 1]),
 'plectrum': array([0, 0, 0, 0, 0, 4]),
 'computer': array([0, 0, 0, 0, 3, 0]),
 'trinity': array([0, 0, 0, 0, 0, 4]),
 'theory': array([0, 3, 1, 0, 0, 0]),
 'many': array([0, 1, 0, 2, 0, 0]),
 'applications': array([0, 0, 3, 0, 0, 0]),
 'worked': array([0, 2, 3, 3, 0, 0]),
 'college': array([1, 0, 0, 0, 0, 4]),
 'structures': array([3, 0, 0, 0, 3, 1]),
 'com': array([2, 1, 0, 0, 0, 3]),
 'group': array([0, 1, 2, 0, 0, 0]),
 'python': array([5, 0, 0, 0, 1, 0]),
 'java': array([2, 3, 2, 2, 2, 1]),
 'participated': array([0, 0, 0, 2, 0, 0]),
 'cern': array([0, 0, 4, 1, 0, 2]),
 'algorithms': array([2, 1, 0, 0, 2, 1]),
 'also': array([1, 0, 1, 3, 0, 0]),
 'auckland': array([0, 3, 0, 0, 0, 1]),
 'research': array([0, 1, 2, 0, 0, 1]),
 'requests': array([0, 0, 0, 2, 0, 0]),
 'guitar': array([0, 0, 0, 0, 0, 4]),
 'london': array([0, 0, 0, 0, 0, 4]),
 'data': array([3, 1, 0, 1, 2, 1]),
 'students': array([3, 0, 0, 1, 0, 0]),
 'mathematics': array([0, 3, 0, 0, 2, 1])}
```

## Term Frequency Inverse Document Frequency (TF-IDF) Model

Here the term frequency vector is multiplied with the inverse Document frequency factor such that

$idf(w) = log(N_t/N_w)$ where $N_t$ is the total number of documents, 6 in this case and $N_w$ is the number of documents in which the word *w* is present.

### Implementation [See on GitHub]

```
# Implementing the Term Frequency Inverse Document Frequency (TF-IDF) Vectors


from collections import Counter


import nltk

import numpy as np

import pandas

# noinspection PyUnresolvedReferences

from utils import tokenize


nltk.download('stopwords')


# importing corpus as resume

resume_file = open('../assets/resume.txt', 'r')

resume = resume_file.read().lower()

resume_file.close()


# tokenizing the resume

tokens = tokenize(resume)


# dividing corpus into 6 documents

k = len(tokens) // 6

documents = []

for i in range(5):

    documents.append(tokens[i * k: (i + 1) * k])
documents.append(tokens[5 * k:])

# calculating most common 5 tokens from each document and storing frequency
tables for each document
most_common = set()
document_frequencies = []
for document in documents:
   frequencies = Counter(document)
   document_frequencies.append(frequencies)
   for word, frequency in frequencies.most_common(5):
       most_common.add(word)
print(len(most_common))

# calculating the number of documents each word appears in
N_t = 6
N_w = {}
for word in most_common:
   count = 0
   for frequencies in document_frequencies:
       count = count + (word in frequencies)
   N_w[word] = count

# Computing the TF-IDF Vectors
vectors = {}
for word in most_common:
   vector = [0] * 6
   for index, frequencies in enumerate(document_frequencies):
       vector[index] = frequencies[word] * np.log(N_t / N_w[word])
   vectors[word] = vector

# storing the vectors in tabular format
table = pandas.DataFrame(data=vectors)
```

```
print(table)


# storing the result in text file
output_file = open('../assets/tfidf.txt', 'w')
output_file.write(table.to_string())
output_file.close()
```

## Output [See on GitHub]

```
 requests  geometry  mathematics  university  algorithms  java   college
0  0.000000  0.000000     0.000000    0.000000    0.810930  0.0  1.098612
1  0.000000  5.375278     2.079442    1.386294    0.405465  0.0  0.000000
2  0.000000  0.000000     0.000000    0.000000    0.000000  0.0  0.000000
3  3.583519  0.000000     0.000000    0.000000    0.000000  0.0  0.000000
4  0.000000  0.000000     1.386294    1.386294    0.810930  0.0  0.000000
5  0.000000  0.000000     0.693147    0.693147    0.405465  0.0  4.394449


     group      many       com    theory    python  plectrum  students
0  0.000000  0.000000  1.386294  0.000000  5.493061  0.000000  3.295837
1  1.098612  1.098612  0.693147  3.295837  0.000000  0.000000  0.000000
2  2.197225  0.000000  0.000000  1.098612  0.000000  0.000000  0.000000
3  0.000000  2.197225  0.000000  0.000000  0.000000  0.000000  1.098612
4  0.000000  0.000000  0.000000  0.000000  1.098612  0.000000  0.000000
5  0.000000  0.000000  2.079442  0.000000  0.000000  7.167038  0.000000

    london  research      cern   trinity  participated    guitar
0  0.000000  0.000000  0.000000  0.000000      0.000000  0.000000
1  0.000000  0.693147  0.000000  0.000000      0.000000  0.000000
2  0.000000  1.386294  2.772589  0.000000      0.000000  0.000000
3  0.000000  0.000000  0.693147  0.000000      3.583519  0.000000
4  0.000000  0.000000  0.000000  0.000000      0.000000  0.000000
```

```
5  7.167038  0.693147  1.386294  7.167038      0.000000  7.167038



       data  applications    worked      also  structures  auckland
0  0.546965      0.000000  0.000000  0.693147    2.079442  0.000000
1  0.182322      0.000000  1.386294  0.000000    0.000000  3.295837
2  0.000000      5.375278  2.079442  0.693147    0.000000  0.000000
3  0.182322      0.000000  2.079442  2.079442    0.000000  0.000000
4  0.364643      0.000000  0.000000  0.000000    2.079442  0.000000
5  0.182322      0.000000  0.000000  0.000000    0.693147  1.098612

     computer
0    0.000000
1    0.000000
2    0.000000
3    0.000000
4    5.375278
5    0.000000
```

## Bibliography

1. Speech & Language Processing ~Jurafsky
2. nltk
3. pickle
4. numpy
5. pandas
6. pandas.DataFrame
7. Indexing and Slicing on pandas DataFrame