Assignment2_lbasyal

# Part a: Bag of Words

Transforming text to a vector Machine Learning algorithms work with numeric data and we cannot use the provided text data "as is". There are many ways to transform text data to numeric vectors. In this task you will try to use two of them.

Bag of words One of the well-known approaches is a bag-of-words representation. To create this transformation, follow the steps:

Find N most popular words in train corpus and numerate them. Now we have a dictionary of the most popular words. For each title in the corpora create a zero vector with the dimension equals to N. For each text in the corpora iterate over words which are in the dictionary and increase by 1 the corresponding coordinate. Let's try to do it for a toy example. Imagine that we have N = 4 and the list of the most popular words is

['hi', 'you', 'me', 'are'] Then we need to numerate them, for example, like this:

{'hi': 0, 'you': 1, 'me': 2, 'are': 3} And we have the text, which we want to transform to the vector:

'hi how are you' For this text we create a corresponding zero vector

[0, 0, 0, 0] And iterate over all words, and if the word is in the dictionary, we increase the value of the corresponding position in the vector:

'hi': [1, 0, 0, 0] 'how': [1, 0, 0, 0] # word 'how' is not in our dictionary 'are': [1, 0, 0, 1] 'you': [1, 1, 0, 1] The resulting vector will be

[1, 1, 0, 1] Implement the described encoding in the function my_bag_of_words with the size of the dictionary equals to 4.

In [40]:
```
import numpy as np
```

In [41]:
```
def my_bag_of_words(text, words_to_index, dict_size):
  result_vector = np.zeros(dict_size) #creating zeros equal to dict_size (4)
  for word in text.split(): #splitting the text
    if word in words_to_index: #check the dictionary element with the word i
      result_vector[words_to_index[word]] +=1
  return result_vector
```

In [32]:
```
words_to_index = {'hi': 0, 'you': 1, 'me': 2, 'are': 3}
# Represent the word_to_index as per the given question
text = 'hi how are you'
dict_size = 4 # dictionary size
result_vector = my_bag_of_words(text, words_to_index, dict_size) #calling th
print(result_vector) #printing the result
```

```
[1. 1. 0. 1.]
```

# Part b: TF-IDF

## 1. Test the script tfidf_demo.ipynb in the Jupiter note and make sure they work.

In [33]:

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
texts = [
    "good movie", "not a good movie", "did not like",
    "i like it", "good one"
]
# using default tokenizer in TfidfVectorizer
tfidf = TfidfVectorizer(min_df=2, max_df=0.5, ngram_range=(1, 2))
features = tfidf.fit_transform(texts)
pd.DataFrame(
    features.todense(),
    columns=tfidf.get_feature_names()
)
```

Out[33]:

| | good movie | like | movie | not |
|---|---|---|---|---|
| **0** | 0.707107 | 0.000000 | 0.707107 | 0.000000 |
| **1** | 0.577350 | 0.000000 | 0.577350 | 0.577350 |
| **2** | 0.000000 | 0.707107 | 0.000000 | 0.707107 |
| **3** | 0.000000 | 1.000000 | 0.000000 | 0.000000 |
| **4** | 0.000000 | 0.000000 | 0.000000 | 0.000000 |

## 2. Replace the movie review data "texts" in the script file with your own defined document and test it.

In [34]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
texts = [
    "best lecture", "not a best lecture", "did not like",
    "i like it", "best one"
]
# using default tokenizer in TfidfVectorizer
tfidf = TfidfVectorizer(min_df=2, max_df=0.5, ngram_range=(1, 2))
features = tfidf.fit_transform(texts)
pd.DataFrame(
    features.todense(),
    columns=tfidf.get_feature_names()
)
```

Out[34]:

| | best lecture | lecture | like | not |
|---|---|---|---|---|
| 0 | 0.707107 | 0.707107 | 0.000000 | 0.000000 |
| 1 | 0.577350 | 0.577350 | 0.000000 | 0.577350 |
| 2 | 0.000000 | 0.000000 | 0.707107 | 0.707107 |
| 3 | 0.000000 | 0.000000 | 1.000000 | 0.000000 |
| 4 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |

## 3. Given the below documents.

texts = [ "good movie", "not a good movie", "did not like", "i like it", "good one" ]

Given the definition of TF and IDF, what is the sum of TF-IDF values for 1-grams in "good movie" text? Enter a math expression as an answer.

## Solution:

Calculating TF("good") from the abovementioned documents naming D1 to D5 respectively

TF("good", D1) = 1/2

TF("good", D2) = 1/4

TF("good", D3) = 0/3

TF("good", D4) = 0/3

TF("good", D5) = 1/2

Calculating TF("movie") from the abovementioned documents naming D1 to D5 respectively

TF("movie", D1) = 1/2

TF("movie", D2) = 1/4

TF("movie", D3) = 0/3

TF("movie", D4) = 0/3

TF("movie", D5) = 0/2

Calculating IDF

IDF("good") = log(5/3)

IDF("movie") = log(5/2)

In [35]:
```
#Calculating TF using Dict
TF = dict()
TF["good"] = 1/2+1/4+0/3+0/3+1/2
TF["movie"] = 1/2+1/4+0/3+0/3+0/2
TF
```

Out[35]:
```
{'good': 1.25, 'movie': 0.75}
```

In [36]:
```
#Calculating IDF using Dict
IDF = dict()
IDF["good"] = np.log(5/3)
IDF["movie"] = np.log(5/2)
IDF
```

Out[36]:
```
{'good': 0.5108256237659907, 'movie': 0.9162907318741551}
```

In [37]:
```
#Sum of TF-IDF
```

In [38]:
```
TFIDF = TF["good"] * IDF["good"] + TF["movie"] * IDF["movie"]
```

In [39]:
```
#Printing the calculated TFIDF
print(TFIDF)
```

```
1.3257500786131047
```

This is the sum of TF-IDF

In [ ]:

In [ ]:

In [ ]:

In [ ]: