Stevens Institute of Technology

Department of Electrical Engineering

EE628: Data Acquisition/Proc II

Homework Assignment 1

Submitted by:
Hadia Hameed
CWID: 10440803

Submitted to:
Prof. Rensheng Wang

February 14, 2019

1 Part (a):

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.

1.1 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.

Solution:

In [32]:
```
import nltk
import numpy as np
```

In [31]:
```
#function declaration
def my_bag_of_words(text, words_to_index, dict_size):
    """
    text: a string
    dict_size: size of the dictionary

    return a vector which is a bag-of-words
    representation of 'text'
    """
    result_vector = np.zeros(dict_size, dtype=int)
    tokenizer=nltk.tokenize.WhitespaceTokenizer()
    words = tokenizer.tokenize(text)
    for w in words:
        for key, value in words_to_index.items():
            if w == key:
                result_vector[value] += 1;
    return result_vector
```

In [30]:
```
#calling the function
words_to_index = {'hi': 0, 'you': 1, 'me': 2, 'are': 3}
text = 'hi how are you'
dict_size = 4
my_bag_of_words(text, words_to_index, dict_size)
```

Out[30]:
```
array([1, 1, 0, 1])
```


2 Part (b):

2.1 Test the script tfidf_demo.ipynb in the Jupyter note and make sure they work.

Out[8]:

	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

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

Tf-idf example

In [9]:
```
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"
]
'''
texts = [
    "i hated that movie. it was horrible.","i enjoyed that movie.",
    "i neither enjoyed, nor hated the movie.",
    "the movie was really horrible."
]
# 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[9]:

	enjoyed	hated	horrible	that	that movie	the	the movie	was
0	0.00000	0.447214	0.447214	0.447214	0.447214	0.0	0.0	0.447214
1	0.57735	0.000000	0.000000	0.577350	0.577350	0.0	0.0	0.000000
2	0.50000	0.500000	0.000000	0.000000	0.000000	0.5	0.5	0.000000
3	0.00000	0.000000	0.500000	0.000000	0.000000	0.5	0.5	0.500000

2.3 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.

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

$$\sum_{t \epsilon d} tfidf(t,d,D) = \sum_{t \epsilon d} tf(t,d) \cdot idf(t,D) = \sum_{t \epsilon d} \left(\frac{f_{t,d}}{\sum_{t' \epsilon d} f_{t',d}} \cdot \frac{logN}{|d \epsilon D : t \epsilon d|} \right)$$

Review: "good movie"

- TERM#1: "GOOD"

Calculating TF for "good":

$f_{t,d}$ = frequency count for the term t : "good" in document d : 1 = **1**
$\sum_{t' \epsilon d} f_{t',d}$ = total number of terms in document d : 1 = **2**

$$TF_{good} = \frac{1}{2}$$

Calculating IDF for "good":

N = total number of documents = **5**
$|d \epsilon D : t \epsilon d|$ = total number of documents in which t : "good" appears = **3**

$$IDF_{good} = log\left(\frac{5}{3}\right) = 0.2218$$

- Term#2: "movie"

Calculating TF for "movie":

$f_{t,d}$ = frequency count for the term t : "movie" in document d : 1 = **1**
$\sum_{t' \epsilon d} f_{t',d}$ = total number of terms in document d : 1 = **2**

$$TF_{movie} = \frac{1}{2}$$

Calculating IDF for "movie":

N = total number of documents = **5**
$|d \epsilon D : t \epsilon d|$ = total number of documents in which t : "movie" appears = **2**

$$IDF_{movie} = log\left(\frac{5}{2}\right) = 0.3980$$

- Sum of the TF-IDF values for 1-grams in "good movie" text:

$$TF \cdot IDF\prime = \left(\frac{1}{2} \cdot log\left(\frac{5}{3}\right)\right) + \left(\frac{1}{2} \cdot log\left(\frac{5}{2}\right)\right) = 5 \cdot log\left(\frac{5}{12}\right) = 0.3098$$

