BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

NLP LAB – 4
[INTRODUCTION TO TF-IDF]

DATE: 22/09/2023 – 24/09/2023
TIME: 02 Hours

This Labsheet contains the basics of word embeddings and an introduction to term frequency-inverse document frequency (tf-idf). Word embedding in NLP is an important term that is used for representing words for text analysis in the form of real-valued vectors.

In this approach, words and documents are represented in the form of numeric vectors allowing similar words to have similar vector representations. The extracted features are fed into a machine learning model so as to work with text data and preserve the semantic and syntactic information. This information once received in its converted form is used by NLP algorithms that easily digest these learned representations and process textual information.

Sources:
1. https://web.stanford.edu/~jurafsky/slp3/6.pdf
2. https://www.turing.com/kb/guide-on-word-embeddings-in-nlp

Word Embeddings

Vector semantics instantiates the linguistic vector semantics embeddings hypothesis by learning representations of the meaning of words, called **embeddings**, directly from their distributions in texts. These representations are used in every natural language processing application that makes use of meaning, and the static embeddings we introduce here underlie the more powerful dynamic or contextualized embeddings like **BERT** that we will cover in the later labsheets.

Example

Let's consider an example **problem statement**:
Given a supervised learning task to predict which tweets are about real disasters and which ones are not (classification). Here the independent variable would be the tweets (text) and the target variable would be the binary values (1: Real Disaster, 0: Not real Disaster).
IV: text
TV: 0/1

Now, Machine Learning and Deep Learning algorithms only take numeric input. So, how do we convert tweets to their numeric values? We will dive deep into the techniques to solve such problems, but first let's look at the solution provided by word embedding.

Solution:
Word Embeddings in NLP is a technique where individual words are represented as real-valued vectors in a lower-dimensional space and captures inter-word semantics. Each word is represented by a real-valued vector with tens or hundreds of dimensions.

TF-IDF

You have several libraries and open-source code on Github that provide a decent implementation of TF-IDF. If you don't need a lot of control over how the TF-IDF math is computed then it is also suggested to re-use libraries from known packages such as Spark's MLLib or Python's scikit-learn.

The one problem that I noticed with these libraries is that they are meant as a pre-step for other tasks like clustering, topic modeling and text classification. TF-IDF can actually be used to extract important keywords from a document to get a sense of what characterizes a document. For example, if you are dealing with wikipedia articles, you can use tf-idf to extract words that are unique to a given article. These keywords can be used as a very simple summary of the document, it can be used for text-analytics (when we look at these keywords in aggregate), as candidate labels for a document and more.

Dataset

Since we used some pretty clean user reviews in some of our previous labs, in this example, we will be using a Stackoverflow dataset which is slightly noisier and simulates what you could be dealing with in real life. You can find this dataset attached on Nalanda in the zip. Notice that there are two files, the larger file with (20,000 posts) is used to compute the Inverse Document Frequency (IDF) and the smaller file with 500 posts would be used as a test set for us to extract keywords from. This dataset is based on the publicly available Stackoverflow dump on Google's Big Query.
Let's take a peek at our dataset. The code below reads a one per line json string from `data/stackoverflow-data-idf.json` into a pandas data frame and prints out its schema and total number of posts. Here, `lines=True` simply means we are treating each line in the text file as a separate json string. With this, the json in line 1 is not related to the json in line 2.

```
import pandas as pd

# read json into a dataframe
df_idf=pd.read_json("data/stackoverflow-data-idf.json",lines=True)

# print schema
print("Schema:\n\n",df_idf.dtypes)
print("Number of questions,columns=",df_idf.shape)
```

Take note that this stackoverflow dataset contains 19 fields including post title, body, tags, dates and other metadata which we don't quite need for this tutorial. What we are mostly interested in for this tutorial is the body and title which is our source of text. We will now create a field that combines both body and title so we have it in one field. We will also print the second text entry in our new field just to see what the text looks like.

```
import re
def pre_process(text):

    # lowercase
    text= # TODO

    #remove tags <> using regex
    text= # TODO

    # remove special characters and digits
    text= re.sub("(\\d|\\W)+"," ",text)

    return text

df_idf['text'] = df_idf['title'] + df_idf['body']
df_idf['text'] = df_idf['text'].apply(...) #TODO: Make appropriate call

#show the first 'text'
df_idf['text'][2]
```

Creating IDF

CountVectorizer to create a vocabulary and generate word counts

The next step is to start the counting process. We can use the CountVectorizer to create a vocabulary from all the text in our df_idf['text'] and generate counts for each row in df_idf['text']. The result of the last two lines is a sparse matrix representation of the counts, meaning each column represents a word in the vocabulary and each row represents the document in our dataset where the values are the word counts. Note that with this representation, counts of some words could be 0 if the word did not appear in the corresponding document.

```
from sklearn.feature_extraction.text import CountVectorizer
import re

def get_stop_words(stop_file_path):
    """load stop words """

    with open(stop_file_path, 'r', encoding="utf-8") as f:
        stopwords = # TODO: Read lines from the file f
        stop_set = set(...) # TODO: Strip all stopwords
        return frozenset(stop_set)

#load a set of stop words
stopwords=get_stop_words("resources/stopwords.txt")

#get the text column
docs=df_idf['text'].tolist()

#create a vocabulary of words,
#ignore words that appear in 85% of documents,
#eliminate stop words
cv= # TODO
word_count_vector= # TODO
```

Now let's check the shape of the resulting vector. Notice that the shape below is `(20000,149391)` because we have 20,000 documents in our dataset (the rows) and the vocabulary size is 149391 meaning we have 149391 unique words (the columns) in our dataset minus the stopwords. In some of the text mining applications, such as clustering and text classification we limit the size of the vocabulary. It's really easy to do this by setting `max_features=vocab_size` when instantiating `CountVectorizer`.

```
word_count_vector.shape
(20000, 124901)
```

Let's limit our vocabulary size to 10,000

```
cv= # TODO: Previous definition with the parameter max_features=10000
word_count_vector= # TODO: Previous definition
word_count_vector.shape
```

Now, let's look at 10 words from our vocabulary. Sweet right, these are mostly programming related :)

```
list(cv.vocabulary_.keys())[:10]
['serializing',
 'private',
 'struct',
 'public',
 'class',
 'contains',
 'properties',
 'string',
 'serialize',
 'attempt']
```

We can also get the vocabulary by using `get_feature_names()`

```
list(cv.get_feature_names())[2000:2015]
```

TfidfTransformer to Compute Inverse Document Frequency (IDF)

In the code below, we are essentially taking the sparse matrix from `CountVectorizer` to generate the IDF when you invoke fit. An extremely important point to note here is that the IDF should be based on a large corpora and should be representative of texts you would be using to extract keywords. To understand why IDF should be based on a fairly large collection, please read this page from Standford's IR book.

```
from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer=TfidfTransformer(smooth_idf=True,use_idf=True)
tfidf_transformer.fit(word_count_vector)
```

Let's look at some of the IDF values:

```
tfidf_transformer.idf_
```

Computing TF-IDF and Extracting Keywords

Once we have our IDF computed, we are now ready to compute TF-IDF and extract the top keywords. In this example, we will extract top keywords for the questions in `data/stackoverflow-test.json`. This data file has 500 questions with fields identical to that of `data/stackoverflow-data-idf.json` as we saw above. We will start by reading our test file, extracting the necessary fields (title and body) and get the texts into a list.

```
# read test docs into a dataframe and concatenate title and body
df_test=pd.read_json("data/stackoverflow-test.json",lines=True)
df_test['text'] = df_test['title'] + df_test['body']
df_test['text'] =df_test['text'].apply(...) # TODO: Appropriate call

# get test docs into a list
docs_test=df_test['text'].tolist()
docs_title=df_test['title'].tolist()
docs_body=df_test['body'].tolist()
```

```
def sort_coo(coo_matrix):
    tuples = zip(coo_matrix.col, coo_matrix.data)
    return sorted(tuples, key=lambda x: (x[1], x[0]), reverse=True)

def extract_topn_from_vector(feature_names, sorted_items, topn=10):
    """get the feature names and tf-idf score of top n items"""

    #use only topn items from vector
    sorted_items = sorted_items[...] # TODO: Splice the data

    score_vals = []
    feature_vals = []

    for idx, score in sorted_items:
        fname = feature_names[idx]

        #keep track of feature name and its corresponding score
        # TODO: Append score rounded off to 3 decimal places, in the
score_vals list

        # TODO: Append fname to the feature_vals list


    #create a tuples of feature,score
    #results = zip(feature_vals,score_vals)
    results= {}
    for idx in range(len(feature_vals)):
        results[feature_vals[idx]]=score_vals[idx]

    return results
```

The next step is to compute the tf-idf value for a given document in our test set by invoking `tfidf_transformer.transform(...)`. This generates a vector of tf-idf scores. Next, we sort the words in the vector in descending order of tf-idf values and then iterate over to extract the top-n items with the corresponding feature names, In the example below, we are extracting keywords for the first document in our test set.

The `sort_coo(...)` method essentially sorts the values in the vector while preserving the column index. Once you have the column index then its really easy to look-up the corresponding word value as you would see in `extract_topn_from_vector(...)` where we do `feature_vals.append(feature_names[idx])`.

```
# you only needs to do this once
feature_names= # TODO: get_feature_names using initialized cv

# get the document that we want to extract keywords from
doc=docs_test[0]

#generate tf-idf for the given document using tfidf_transformer
tf_idf_vector= # TODO

#sort the tf-idf vectors by descending order of scores
sorted_items=sort_coo(tf_idf_vector.tocoo())

#extract only the top n; n here is 10
keywords=extract_topn_from_vector(...) # TODO: Pass params

# now print the results
print("\n=====Title=====")
print(docs_title[0])
print("\n=====Body=====")
print(docs_body[0])
print("\n===Keywords===")

for k in keywords:
    print(k,keywords[k])
```

```
=====Title=====
Integrate War-Plugin for m2eclipse into Eclipse Project

=====Body=====
<p>I set up a small web project with JSF and Maven. Now I want to deploy on
a Tomcat server. Is there a possibility to automate that like a button in E
clipse that automatically deploys the project to Tomcat?</p>

<p>I read about a the <a href="http://maven.apache.org/plugins/maven-war-pl
ugin/" rel="nofollow noreferrer">Maven War Plugin</a> but I couldn't find a
tutorial how to integrate that into my process (eclipse/m2eclipse).</p>

<p>Can you link me to help or try to explain it. Thanks.</p>

===Keywords===
eclipse 0.593
war 0.317
integrate 0.281
maven 0.273
tomcat 0.27
project 0.239
```

```
plugin 0.214
automate 0.157
jsf 0.152
possibility 0.146
```

From the keywords above, the top keywords actually make sense, it talks about `eclipse`, `maven`, `integrate`, `war` and `tomcat` which are all unique to this specific question. There are a couple of keywords that could have been eliminated such as possibility and perhaps even project and you can do this by adding more common words to your stop list and you can even create your own set of stop list, very specific to your domain as described here.

```
# put the common code into several methods
def get_keywords(idx):

    #generate tf-idf for the given document using tfidf_transformer
    tf_idf_vector= #TODO

    #sort the tf-idf vectors by descending order of scores
    sorted_items=sort_coo(tf_idf_vector.tocoo())

    #extract only the top n; n here is 10
    keywords=extract_topn_from_vector(...) # TODO: Pass params

    return keywords

def print_results(idx,keywords):
    # now print the results
    print("\n=====Title=====")
    print(docs_title[idx])
    print("\n=====Body=====")
    print(docs_body[idx])
    print("\n===Keywords===")
    for k in keywords:
        print(k,keywords[k])
```

Now let's look at keywords generated for a much longer question:

```
idx=120
keywords=get_keywords(idx)
print_results(idx,keywords)
```

```
=====Title=====
SQL Import Wizard - Error

=====Body=====
<p>I have a CSV file that I'm trying to import into SQL Management Server S
tudio.</p>
```

```
<p>In Excel, the column giving me trouble looks like this:
<a href="https://i.stack.imgur.com/pm0uS.png" rel="nofollow noreferrer"><im
g src="https://i.stack.imgur.com/pm0uS.png" alt="enter image description he
re"></a></p>

<p>Tasks > import data > Flat Source File > select file</p>

<p><a href="https://i.stack.imgur.com/G4b6I.png" rel="nofollow noreferrer">
<img src="https://i.stack.imgur.com/G4b6I.png" alt="enter image description
here"></a></p>

<p>I set the data type for this column to DT_NUMERIC, adjust the DataScale
to 2 in order to get 2 decimal places, but when I click over to Preview, I
see that it's clearly not recognizing the numbers appropriately:</p>

<p><a href="https://i.stack.imgur.com/NZhiQ.png" rel="nofollow noreferrer">
<img src="https://i.stack.imgur.com/NZhiQ.png" alt="enter image description
here"></a></p>

<p>The column mapping for this column is set to type = decimal; precision 1
8; scale 2.</p>

<p>Error message: Data Flow Task 1: Data conversion failed. The data conver
sion for column "Amount" returned status value 2 and status text "The value
could not be converted because of a potential loss of data.".
 (SQL Server Import and Export Wizard)</p>

<p>Can someone identify where I'm going wrong here?  Thanks!</p>

===Keywords===
column 0.365
import 0.286
data 0.283
wizard 0.27
decimal 0.227
conversion 0.224
sql 0.217
status 0.164
file 0.147
appropriately 0.142
```

Generate keywords for a batch of documents

```
#generate tf-idf for all documents in your list. docs_test has 500
documents using tfidf_transformer
tf_idf_vector= # TODO. HINT - Use docs_test for the transform() of cv

results=[]
for i in range(tf_idf_vector.shape[0]):

    # get vector for a single document
    # TODO

    #sort the tf-idf vector by descending order of scores
    # TODO

    #extract only the top n; n here is 10
    # TODO

    results.append(keywords)

df=pd.DataFrame(zip(docs,results),columns=['doc','keywords'])
df
```

	doc	keywords
0	serializing a private struct can it be done i ...	{'eclipse': 0.593, 'war': 0.317, 'integrate': ...
1	how do i prevent floated right content from ov...	{'evaluate': 0.472, 'content': 0.403, 'console...
2	gradle command line i m trying to run a shell ...	{'appdomain': 0.409, 'dynamic': 0.384, 'perfor...
3	loop variable as parameter in asynchronous fun...	{'image': 0.424, 'jpg': 0.412, 'background': 0...
4	canot get the href value hi i need to valid th...	{'uri': 0.371, 'bitmap': 0.318, 'intent': 0.30...
...	...	...
495	how to unbind click and click submit button in...	{'delphi': 0.617, 'compatible': 0.365, 'win': ...
496	swaggerui auth redirect swaggeruiauth of null ...	{'node': 0.547, 'selectsinglenode': 0.304, 'nu...
497	ssrs value display error for ssrs conditional ...	{'logo': 0.549, 'step': 0.33, 'triangle': 0.32...
498	accessing and changing a class instance from a...	{'length': 0.426, 'ev': 0.415, 'introduce': 0....
499	how to print the current time in the format da...	{'oauth': 0.388, 'localhost': 0.383, 'sdk': 0....

500 rows × 2 columns

Exercise

1. Complete all the function definitions and TODOs given
2. Implement the code for Task#2 given below

Task#2:

```
# DATA BLOCK

# Each new line (after '\n') represents a new document
text = '''he really really loves coffee
my sister dislikes coffee
my sister loves tea'''

import math

def main(text):
    # split the text first into lines and then into lists of words
    docs = # TODO

    N = len(docs)

    # create the vocabulary: the list of words that appear at least
once
    vocabulary = # TODO
```

```
    df = {}
    tf = {}
    for word in vocabulary:
        # tf: number of occurrences of word w in document divided by
document length
        # note: tf[word] will be a list containing the tf of each word
for each document
        # for example tf['he'][0] contains the term frequence of the
word 'he' in the first document
        tf[word] = [...] # HINT: Use list comprehension

        # df: number of documents containing word w
        df[word] = # TODO
    # loop through documents to calculate the tf-idf values
    for doc_index, doc in enumerate(docs):
        tfidf = []
        for word in vocabulary:
            # ADD THE CORRECT FORMULA HERE. Remember to use the base 10
logarithm: math.log(x, 10)
            word_tf = # TODO
            word_df = # TODO
            word_tfidf = # TODO (Don’t forget the base for log is 10)
            tfidf.append(word_tfidf)

        print(tfidf)

main(text)
```