# Lab 7 – Natural language processing (NLP)

Table of Contents


I. Feature Engineering ..... 2
1. Text Normalization ..... 3
2. Implement TF-IDF ..... 3
3. Compare the results with the reference implementation of scikit-learn library. ..... 4
4. Apply TF-IDF for information retrieval ..... 4

II. Text Processing ..... 5
1. Preprocessing ..... 5
2. Bag-of-words ..... 5

III. Text Similarity ..... 5
1. Similarity metrics ..... 5
2. TF-IDF ..... 5
3. Plagiarism checker ..... 5

III. Text Classification ..... 6

IV. Topic Modelling ..... 6

V. Named Entity Recognition ..... 6

VI. Exercise ..... 6


# I. Feature Engineering

Complete the code with TODO tag in the Jupyter notebooks.

In this exercise we will understand the functioning of TF/IDF ranking. Implement the feature engineering and its application, based on the code framework provided below. Please read the file feature_engineering.ipynb

First, we use textual data from Twitter.

| | id | created_at | text |
|---|---|---|---|
| **0** | 849636868052275200 | 2017-04-05 14:56:29 | b'And so the robots spared humanity ... https:... |
| **1** | 848988730585096192 | 2017-04-03 20:01:01 | b"@ForIn2020 @waltmossberg @mims @defcon_5 Exa... |
| **2** | 848943072423497728 | 2017-04-03 16:59:35 | b'@waltmossberg @mims @defcon_5 Et tu, Walt?' |
| **3** | 848935705057280001 | 2017-04-03 16:30:19 | b'Stormy weather in Shortville ...' |
| **4** | 848416049573658624 | 2017-04-02 06:05:23 | b"@DaveLeeBBC @verge Coal is dying due to nat ... |

## 1. Text Normalization

Now we need to normalize text by stemming, tokenizing, and removing stopwords.

In [3]:
```
def normalize(document):
    # TODO: remove punctuation
    text = "".join([ch for ch in document if ch not in string.punctuation])

    # TODO: tokenize text
    tokens = nltk.word_tokenize(text)

    # TODO: Stemming
    stemmer = PorterStemmer()
    ret = " ".join([stemmer.stem(word.lower()) for word in tokens])
    return ret

original_documents = [x.strip() for x in data['text']]
documents = [normalize(d).split() for d in original_documents]
documents[0]
```

As you can see that the normalization is still not perfect. Please feel free to improve upon ,e.g. https://marcobonzanini.com/2015/03/09/mining-twitter-data-with-python-part-2/

## 2. Implement TF-IDF

Now you need to implement TF-IDF, including creating the vocabulary, computing term frequency, and normalizing by tf-idf weights.

In [4]:
```
# Flatten all the documents
flat_list = [word for doc in documents for word in doc]

# TODO: remove stop words from the vocabulary
words = [word for word in flat_list if word not in stopwords.words('english')]

# TODO: we take the 500 most common words only
counts = Counter(words)
vocabulary = counts.most_common(500)
print([x for x in vocabulary if x[0] == 'tesla'])
vocabulary = [x[0] for x in vocabulary]
assert len(vocabulary) == 500

# vocabulary.sort()
vocabulary[:5]
```

```
[('tesla', 287)]
```

In [6]:
```
def idf(vocabulary, documents):
    """TODO: compute IDF, storing values in a dictionary"""
    idf = {}
    num_documents = len(documents)
    for i, term in enumerate(vocabulary):
        idf[term] = math.log(num_documents / sum(term in document for document in documents), 2)
    return idf

idf = idf(vocabulary, documents)
[idf[key] for key in vocabulary[:5]]
```

## 3. Compare the results with the reference implementation of scikit-learn library.

Now we use the scikit-learn library. As you can see that, the way we do text normalization affects the result. Feel free to further improve upon e.g. https://stackoverflow.com/questions/36182502/add-stemming-support-to-countvectorizer-sklearn

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

tfidf = TfidfVectorizer(analyzer='word', ngram_range=(1,1), min_df = 1, stop_words = 'english', max_features=500)

features = tfidf.fit(original_documents)
corpus_tf_idf = tfidf.transform(original_documents)

sum_words = corpus_tf_idf.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in tfidf.vocabulary_.items()]
print(sorted(words_freq, key = lambda x: x[1], reverse=True)[:5])
print('testla', corpus_tf_idf[1, features.vocabulary_['tesla']])
```

```
[('http', 163.54366542841234), ('https', 151.85039944652075), ('rt', 112.61998731390989), ('tesla', 95.96401470715628
1), ('xe2', 88.209444863464768)]
testla 0.349524310066
```

## 4. Apply TF-IDF for information retrieval

We can use the vector representation of documents to implement an information retrieval system. We test with the query Q = "tesla nasa"

In [9]:

```
def cosine_similarity(v1,v2):
    """TODO: compute cosine similarity"""
    sumxx, sumxy, sumyy = 0, 0, 0
    for i in range(len(v1)):
        x = v1[i]; y = v2[i]
        sumxx += x*x
        sumyy += y*y
        sumxy += x*y
    if sumxy == 0:
            result = 0
    else:
            result = sumxy/math.sqrt(sumxx*sumyy)
    return result

def search_vec(query, k, vocabulary, stemmer, document_vectors, original_documents):
    q = query.split()
    q = [stemmer.stem(w) for w in q]
    query_vector = vectorize(q, vocabulary, idf)

    # TODO: rank the documents by cosine similarity
    scores = [[cosine_similarity(query_vector, document_vectors[d]), d] for d in range(len(document_vectors))]
    scores.sort(key=lambda x: -x[0])

    print('Top-{0} documents'.format(k))
    for i in range(k):
        print(i, original_documents[scores[i][1]])

query = "tesla nasa"
stemmer = PorterStemmer()
search_vec(query, 5, vocabulary, stemmer, document_vectors, original_documents)
```

```
Top-5 documents
0 b'@ashwin7002 @NASA @faa @AFPAA We have not ruled that out.'
1 b'RT @NASA: Updated @SpaceX #Dragon #ISS rendezvous times: NASA TV coverage begins Sunday at 3:30amET: http://t.co/
qrm0Dz4jPE. Grapple at  ...'
2 b"Deeply appreciate @NASA's faith in @SpaceX. We will do whatever it takes to make NASA and the American people pro
ud."
3 b'Would also like to congratulate @Boeing, fellow winner of the @NASA commercial crew program'
4 b"@astrostephenson We're aiming for late 2015, but NASA needs to have overlapping capability to be safe. Would do t
he same"
```

We can also use the scikit-learn library to do the retrieval.

```
new_features = tfidf.transform([query])

cosine_similarities = linear_kernel(new_features, corpus_tf_idf).flatten()
related_docs_indices = cosine_similarities.argsort()[::-1]

topk = 5
print('Top-{0} documents'.format(topk))
for i in range(topk):
    print(i, original_documents[related_docs_indices[i]])
```

```
Top-5 documents
0 b'RT @NASA: Updated @SpaceX #Dragon #ISS rendezvous times: NASA TV coverage begins Sunday at 3:30amET: http://t.co/
qrm0Dz4jPE. Grapple at  ...'
1 b"Deeply appreciate @NASA's faith in @SpaceX. We will do whatever it takes to make NASA and the American people pro
ud."
2 b'@NASA Best of luck to the Cygnus launch'
3 b'RT @SpaceX: Success! Congrats @NASA on @MarsCuriosity!'
4 b'@ashwin7002 @NASA @faa @AFPAA We have not ruled that out.'
```

# II. Text Processing

## 1. Preprocessing

The first NLP exercise is about preprocessing.

You will practice preprocessing using NLTK on raw data.
This is the first step in most of the NLP projects, so you have to master it.

Open the Preprocessing.ipynb notebook and follow the instructions.

## 2. Bag-of-words

Now that we master the preprocessing, let's make our first Bag Of Words (BOW).
We will reuse our dataset of Coldplay songs to make a BOW.

Open the BOW.ipynb notebook and follow the instructions.

# III. Text Similarity

## 1. Similarity metrics

We will work on applying similarity: Jaccard and Cosine similarity. This exercise is a simple application.

Open Similarity.ipynb notebook and follow the instructions.

## 2. TF-IDF

We will compute the TF-IDF on a corpus of newspaper headlines.
Open TF-IDF.ipynb notebook and follow the instructions.

## 3. Plagiarism checker

In the folder, you will find source texts (Asource.txt, Bsource.txt, Csource.txt and Dsource.txt) from which some texts were inspired (A1.txt was inspired from Asource.txt and so on). Some are plagiarism, some are regular inspiration.

Your job is to use text similarity to define a plagiarism detection algorithm based on those short examples.

In the examples, there are sources, direct plagiarism (A1, B1, C1, D1) and sometimes examples of so called mosaic plagiarism (D2). Can you make an algorithm that detects all kind of plagiarism without false positive?

Open Plagiarism.ipynb notebook and follow the instructions.

## III. Text Classification

It's time to make our first real Machine Learning application of NLP: a spam classifier!

A spam classifier is a Machine Learning model that classifier texts (email or SMS) into two categories: Spam (1) or legitimate (0).
To do that, we will reuse our knowledge: we will apply preprocessing and BOW (Bag Of Words) on a dataset of texts.

Then we will use a classifier to predict to which class belong a new email/SMS, based on the BOW.

Open Spam-Classifier.ipynb notebook and follow the instructions.

## IV. Topic Modelling

This exercise is about modelling the main topics of a database of News headlines.

The tasks include:

- Text pre-processing
- Topic modelling using LSA
- Topic modelling using LDA
- Topic visualization

Open the News-Topic-Modelling.ipynb notebook and follows the instructions.

## V. Named Entity Recognition

This exercise is about named entity recognition and its application in hiding personal information.

The tasks include:

- Text pre-processing (if needed)
- Named Entity Recognition
- Censor any person names

Open the GDPR-Compliance.ipynb notebook and follows the instructions.

## VI. Exercise

1/ By using the job market data, finish the following task to analyze the top important keywords for IT sector.

- Filter the jobs for IT sector only.
- Put the description of all jobs into a list.

- Use scikit-learn to get top 20 important keywords.
- Choose one favourite keyword and perform information retrieval with scikit-learn.

2/ Implement a method capable of extracting n-grams from a given sequence object (e.g., string and list). Utilize this feature to produce word tri-grams, letter tri-grams based on the sentence "I like deadline and want to immerse myself in deadline."

3/ Construct a program that satisfies these criteria in line with them:

You will be provided with a string of words that are separated by a certain amount of space. Every time one of the following words occurs in this order:
If the phrase can be written with no more than four letters, then its original form should be maintained. In any other circumstance, the opening letter and the final letter should both be kept the same. Switch out some of the letters for others and rearrange them in a chaotic sequence (in the middle of the word)
Using a statement such as "I couldn't believe that I could completely understand what I was reading: the astounding power of the human mind" as an example, describe the results by providing a statement such as "I couldn't believe that I could truly comprehend what I was reading."

4/ The zip file alice.zip includes the text file alice.txt containing Lewis Carroll's book Alice's Adventures in Wonderland, which is available on Project Gutenberg. Apply a part-of-speech (POS) tagger to the text file, then save the output to a separate file. Implement programmes that read the results of POS tagging and carry out the tasks.

## Guidelines

- Install program capable of reading the results of the tagging performed on the different portions of speech. In this situation, a phrase should be represented as a list of mapping objects, with each object linking a surface form, lemma (base form), and part-of-speech tag with the respective keys text, lemma, and pos. Use this illustration for the remaining worksheet questions.
- Find all of the surface forms of the verbs that occur in the text and extract them.
- Extract the lemmas of every verb in the text.
- Extract sentences of the pattern "A of B", where A and B are nouns.
- Find and eliminate the longest noun phrase consisting of consecutive nouns.
- Obtain a list of the terms and the percentages of how often they occur, with the percentages decreasing as the frequency of the phrases increases.
- Create a chart showing the frequency with which the ten most popular terms are used (e.g., bar chart).
- Compile a list of words that are often seen in the same context as "Alice." Create a chart (e.g., a bar chart) to illustrate the top 10 words that are often seen in the same context as the word "Alice," as well as the frequency with which they occur.
- Please create a word frequency histogram (x-axis is a scalar range representing a frequency ranging from 1 to the largest frequency of a given word in the entire corpus, and the y-axis is the count of unique words that fall into the count of the x value).
- Using a log-log scale, plot the rank order along the x-axis and the frequency along the y-axis.