# TP2RI

October 7, 2022

## 0.1 Import necessary libs

[ ]:
```
import re
import unicodedata as ud
import numpy as np
import math
import collections
from string import punctuation
from nltk import ngrams
from nltk import word_tokenize, RegexpTokenizer
from nltk.corpus import stopwords
import nltk
from nltk.tokenize import word_tokenize
import matplotlib.pyplot as plt
from nltk.stem import PorterStemmer, ISRIStemmer
import json
import os
from nltk.corpus.reader.plaintext import PlaintextCorpusReader
from collections import Counter
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

## 0.2 Read parameters's input

[ ]:
```
typeofclass = 2
lang = "ar"
n = 2
c_path = "./corpora"
c_path = os.path.abspath(c_path)
# typeofclass = int(
#     input("to use word based classification use 1, for character based one␣
 ↪use 2: "))
# lang = input("enter en for english corpus, ar for the arabic one: ")
# n = int(input("Enter the number of ngrams: "))
```

### 0.2.1 configure tokenizers, stemmers, stopwords based on parameters's values

[ ]:
```
if lang == "en":
    c_path += "/english"
    stopwords = stopwords.words("english")
    stemer = PorterStemmer()
else:
    c_path += "/arabic"
    stopwords = stopwords.words("arabic")
    stemer = ISRIStemmer()


stoplist = set(stopwords + list(punctuation))

retoken = RegexpTokenizer(r'\w+|\$[\d\.]+|\S+')

pattern = re.compile(r'^([0-9]+\w*)')
if lang == "ar":
    pattern = re.compile(r'^([0-9]+\w*)|[a-zA-Z0-9]+')
```

### 0.2.2 preprocessing function

[ ]:
```
def preprocess_text(text: str):
    text = ''.join(c for c in text if not ud.category(c).startswith('P'))
    text = ' '.join(retoken.tokenize(text))
    tokens = [
        token for token in nltk.word_tokenize(
            text)
        if token.lower() not in stoplist
        and
        not token.lower().isdigit()
        and
        pattern.match(token) == None
    ]
    if typeofclass == 2:
        tokens = [c for w in tokens for c in w]
    else:
        tokens = [stemer.stem(w) for w in tokens]
    return tokens
```

### 0.2.3 Reading corpus's files

[ ]:
```
corpus = PlaintextCorpusReader(c_path, ".*")
```

### 0.2.4 getting ngrams -> frequency dictionary for each corpus file

[ ]:
```
corpus_ngrams = []
filesids = corpus._fileids
dictNgrams = {}
for f in filesids:
    file_sentences = corpus.sents(f)
    corpus_ngrams_perfile = []
    for sent in file_sentences:
        sent_words = preprocess_text(" ".join(sent))
        sent_n_grams = ngrams(sent_words, n)

        corpus_ngrams.append(list(sent_n_grams))
        corpus_ngrams_perfile += [w for w in corpus_ngrams[-1]]

    frq_dist_f = nltk.FreqDist(corpus_ngrams_perfile)
    temp_dict = {" ".join(k): v for k, v in dict(frq_dist_f).items()}
    temp_dict = collections.OrderedDict(sorted(temp_dict.items()))
    dictNgrams[f] = temp_dict
```

### 0.2.5 Transform ngrams's dictionary into a pandas dataFrame

[ ]:
```
tf_idf = pd.DataFrame(dictNgrams)
tf_idf = tf_idf.sort_index()
tf_idf = tf_idf.fillna(0)
```

### 0.2.6 Calculating idf and Tf.idf score for each corpus file

[ ]:
```
tf_idf["idf"] = tf_idf.apply(lambda row: math.log10(
    len(row)/(len(row)-len([w for w in row if w == 0]))), axis=1)
for f in filesids:
    tf_idf["tf.idf"+f] = (1 + np.log10(tf_idf[f]))*tf_idf["idf"]
tf_idf = tf_idf.replace([np.inf, -np.inf], 0)
```

### 0.2.7 Save dataframe into a csv

[ ]:
```
tf_idf.to_csv("ngrams.csv")
```