Project Omega

Project's Goal

Our main research goal is to set a program that will predict the gender of the author from a tweet. Down the road, we have the ability to extend the model to any text.

To do so, we will use the following technics:

- Tokenization
- BOW (Bag Of Word)
- TF-IDF
- Cosine Similarity
- Logistic Regression
- Decision tree and random forest
- K-Nearest-Neighbors

All of the technics mentionned above, have been studied during class hours.

In term of business, with this idea we aim to enhence marketing targeting. Third party advertisement companies can have better knowledge on Twitter's gender distribution. Thus, they can advise companies that want to make advertisement on social network like Twitter.

Here you can find a small video that explains this notebook.

```
%%HTML
<iframe width="560" height="340" src="https://www.youtube.com/
embed/MEuSZgx0HEc"></iframe>
```

Importation and Installation of methods

For this project, we will use different packages. The main part will be about text mining. For this, we will use packages presented in class, which are *spacy*, *nltk* and *enchant*.

```
import nltk
!pip install spacy
!apt install -qq enchant
!pip install pyenchant
!pip install nltk
!python -m spacy download en
nltk.download('punkt')
nltk.download('wordnet')
!pip install chart_studio
```

```
Requirement already satisfied: spacy in /usr/local/lib/python3.6/dist-packages (2.1.9)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.6/dist-packages (from spacy) (1.0.2)
Requirement already satisfied: thinc<7.1.0,>=7.0.8 in /usr/local/lib/python3.6/dist-packages (from spacy) (7.0.8)
Requirement already satisfied: srsly<1.1.0,>=0.0.6 in /usr/local/lib/python3.6/dist-packages (from spacy) (0.2.0)
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.6/dist-packages (from spacy) (1.17.4)
Requirement already satisfied: plac<1.0.0,>=0.9.6 in /usr/local/lib/python3.6/dist-packages (from spacy) (0.9.6)
Requirement already satisfied: blis<0.3.0,>=0.2.2 in /usr/local/lib/python3.6/dist-packages (from spacy) (0.2.4)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.6/dist-packages (from spacy) (2.0.3)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.6/dist-packages (from spacy) (2.21.0)
Requirement already satisfied: preshed<2.1.0,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from spacy) (2.0.1)
Requirement already satisfied: wasabi<1.1.0,>=0.2.0 in /usr/local/lib/python3.6/dist-packages (from spacy) (0.4.2)
Requirement already satisfied: tqdm<5.0.0,>=4.10.0 in /usr/local/lib/python3.6/dist-packages (from thinc<7.1.0,>=7.0.8->spacy) (4
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests<3.0.0,>=2.13.0->spa
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (2.
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests<3.0.0,>=2.13.0->spa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests<3.0.0,>=2.13.0->spacy)
The following package was automatically installed and is no longer required:
  libnvidia-common-430
Use 'apt autoremove' to remove it.
The following additional packages will be installed:
  aspell aspell-en dictionaries-common emacsen-common hunspell-en-us
  libaspell15 libenchant1c2a libhunspell-1.6-0 libtext-iconv-perl
Suggested packages:
  aspell-doc spellutils wordlist hunspell openoffice.org-hunspell
  | openoffice.org-core libenchant-voikko
The following NEW packages will be installed:
  aspell aspell-en dictionaries-common emacsen-common enchant hunspell-en-us
  libaspell15 libenchant1c2a libhunspell-1.6-0 libtext-iconv-perl
0 upgraded, 10 newly installed, 0 to remove and 7 not upgraded.
Need to get 1,310 kB of archives.
After this operation, 5,353 kB of additional disk space will be used.
Preconfiguring packages ...
Selecting previously unselected package libtext-iconv-perl.
(Reading database ... 134985 files and directories currently installed.)
Preparing to unpack .../0-libtext-iconv-perl_1.7-5build6_amd64.deb ...
Unpacking libtext-iconv-perl (1.7-5build6) ...
Selecting previously unselected package libaspell15:amd64.
Preparing to unpack .../1-libaspell15_0.60.7~20110707-4ubuntu0.1_amd64.deb ...
Unpacking libaspell15:amd64 (0.60.7~20110707-4ubuntu0.1) ...
Selecting previously unselected package emacsen-common.
Preparing to unpack .../2-emacsen-common_2.0.8_all.deb ...
Unpacking emacsen-common (2.0.8) ...
Selecting previously unselected package dictionaries-common.
Preparing to unpack .../3-dictionaries-common_1.27.2_all.deb ...
Adding 'diversion of /usr/share/dict/words to /usr/share/dict/words.pre-dictionaries-common by dictionaries-common'
Unpacking dictionaries-common (1.27.2) ...
Selecting previously unselected package aspell.
Preparing to unpack .../4-aspell_0.60.7~20110707-4ubuntu0.1_amd64.deb ...
Unpacking aspell (0.60.7~20110707-4ubuntu0.1) ...
Selecting previously unselected package aspell-en.
Preparing to unpack .../5-aspell-en_2017.08.24-0-0.1_all.deb ...
Unpacking aspell-en (2017.08.24-0-0.1) ...
Selecting previously unselected package hunspell-en-us.
Preparing to unpack .../6-hunspell-en-us_1%3a2017.08.24_all.deb ...
Unpacking hunspell-en-us (1:2017.08.24) ...
Selecting previously unselected package libhunspell-1.6-0:amd64.
Preparing to unpack .../7-libhunspell-1.6-0_1.6.2-1_amd64.deb ...
Unpacking libhunspell-1.6-0:amd64 (1.6.2-1) ...
Selecting previously unselected package libenchant1c2a:amd64.
Preparing to unpack .../8-libenchant1c2a_1.6.0-11.1_amd64.deb ...
Unpacking libenchant1c2a:amd64 (1.6.0-11.1) ...
Selecting previously unselected package enchant.
Preparing to unpack .../9-enchant_1.6.0-11.1_amd64.deb ...
Unpacking enchant (1.6.0-11.1) ...
Setting up libhunspell-1.6-0:amd64 (1.6.2-1) ...
Setting up libaspell15:amd64 (0.60.7~20110707-4ubuntu0.1) ...
Setting up emacsen-common (2.0.8) ...
Setting up libtext-iconv-perl (1.7-5build6) ...
Setting up dictionaries-common (1.27.2) ...
Setting up aspell (0.60.7~20110707-4ubuntu0.1) ...
Setting up hunspell-en-us (1:2017.08.24) ...
Setting up libenchant1c2a:amd64 (1.6.0-11.1) ...
Setting up aspell-en (2017.08.24-0-0.1) ...
Setting up enchant (1.6.0-11.1) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for dictionaries-common (1.27.2) ...
aspell-autobuildhash: processing: en [en-common].
aspell-autobuildhash: processing: en [en-variant_0].
aspell-autobuildhash: processing: en [en-variant_1].
aspell-autobuildhash: processing: en [en-variant_2].
aspell-autobuildhash: processing: en [en-w_accents-only].
aspell-autobuildhash: processing: en [en-wo_accents-only].
aspell-autobuildhash: processing: en [en_AU-variant_0].
aspell-autobuildhash: processing: en [en_AU-variant_1].
aspell-autobuildhash: processing: en [en_AU-w_accents-only].
aspell-autobuildhash: processing: en [en_AU-wo_accents-only].
aspell-autobuildhash: processing: en [en_CA-variant_0].
aspell-autobuildhash: processing: en [en_CA-variant_1].
aspell-autobuildhash: processing: en [en_CA-w_accents-only].
```

```
aspell-autobuildhash: processing: en [en_CA-wo_accents-only].
aspell-autobuildhash: processing: en [en_GB-ise-w_accents-only].
aspell-autobuildhash: processing: en [en_GB-ise-wo_accents-only].
aspell-autobuildhash: processing: en [en_GB-ize-w_accents-only].
aspell-autobuildhash: processing: en [en_GB-ize-wo_accents-only].
aspell-autobuildhash: processing: en [en_GB-variant_0].
aspell-autobuildhash: processing: en [en_GB-variant_1].
aspell-autobuildhash: processing: en [en_US-w_accents-only].
aspell-autobuildhash: processing: en [en_US-wo_accents-only].
Collecting pyenchant
  Downloading https://files.pythonhosted.org/packages/9e/54/04d88a59efa33fefb88133ceb638cdf754319030c28aadc5a379d82140ed/pyenchan
     |████████████████████████████████| 71kB 4.4MB/s
Building wheels for collected packages: pyenchant
  Building wheel for pyenchant (setup.py) ... done
  Created wheel for pyenchant: filename=pyenchant-2.0.0-cp36-none-any.whl size=71113 sha256=d8abaf46b2b22452ab1ceeaeeeef55f030d34
  Stored in directory: /root/.cache/pip/wheels/ee/8e/01/f427e9c6c0ae5e22095f3d2aac8997abe7b317307a9de497f4
Successfully built pyenchant
Installing collected packages: pyenchant
Successfully installed pyenchant-2.0.0
Requirement already satisfied: nltk in /usr/local/lib/python3.6/dist-packages (3.2.5)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from nltk) (1.12.0)
Requirement already satisfied: en_core_web_sm==2.1.0 from https://github.com/explosion/spacy-models/releases/download/en_core_web
✓ Download and installation successful
You can now load the model via spacy.load('en_core_web_sm')
✓ Linking successful
/usr/local/lib/python3.6/dist-packages/en_core_web_sm -->
/usr/local/lib/python3.6/dist-packages/spacy/data/en
You can now load the model via spacy.load('en')
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
Requirement already satisfied: chart_studio in /usr/local/lib/python3.6/dist-packages (1.0.0)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from chart_studio) (1.12.0)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from chart_studio) (2.21.0)
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from chart_studio) (4.1.1)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packages (from chart_studio) (1.3.3)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests->chart_studio) (2.8)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests->chart_studio) (2019.1
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests->chart_studio) (1.2
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests->chart_studio) (3.0
```

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
import spacy
from spacy.lang.en import English
import enchant
from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import accuracy_score
import collections
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
from sklearn.base import TransformerMixin
from sklearn.pipeline import Pipeline
import chart_studio.plotly as py
import plotly.graph_objs as go
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style("darkgrid")
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
import operator
```

The Dataset

```
#Import data
data = pd.read_csv('https://raw.githubusercontent.com/XaviJunior/omega/Data/Data/gender-classifier-DFE-791531.csv',encoding="latin-1")
textgender=data[["gender",'text',"description"]]
textgender=textgender.dropna()
X=textgender[['text','description']]
```

```
y=textgender["gender"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=14)
train= pd.concat([X_train.reset_index(drop='True'),y_train.reset_index(drop='True')],axis=1)
test=pd.concat([X_test.reset_index(drop='True'),y_test.reset_index(drop='True')],axis=1)
data.head(5)
```

	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	gender	gender:confidence	profile_yn	profile_yn:conf
0	815719226	False	finalized	3	10/26/15 23:24	male	1.0000	yes	
1	815719227	False	finalized	3	10/26/15 23:30	male	1.0000	yes	
2	815719228	False	finalized	3	10/26/15 23:33	male	0.6625	yes	
3	815719229	False	finalized	3	10/26/15 23:10	male	1.0000	yes	
4	815719230	False	finalized	3	10/27/15 1:15	female	1.0000	yes	

Cosine Similarity

The first method we will apply, is the cosine-similarity. The later is frequently used for text-similarity.
Our first idea, was to apply the cosine similarity between all train tweets and test tweets. Then classify the tweet according to the most similar one. However, we realised that the data set was too big and it would take too much time to go through all the test tweets. (Our code ran an entire night and did not end the task)
So our second idea was to merge all train tweets and the description of each account (to gain text) in two documents: one with text written by male and another by female. With that, we will be able to gain more information such as most common words used by each gender.

Data Cleaning

We began by separating tweet by gender. We made two new tables, one for male and one for female.
We are not interested by all the feautres (columns), we need to keep only "gender", "text" and "name" colums for the brand.
We chose to split the data by gender. Thus, we are able to construct a general BOW for each gender. As a tweet is necessarily written by a male or a female (only taking into account biological gender), we decided to ingonre tweets written by brand or tweet labeled as unknown because they won't help us to reach the goal we set at the beggining of this project

```
#New Table with Male Only and Split in Test/Train Data
Male = train[train['gender'] == 'male']
Maletext=Male['text']
MaleDes=Male['description']
ym=Male["gender"]
ListMale=Maletext.values.tolist()+MaleDes.values.tolist()

#New Table with Female Only and split in Test/Train Data
Female = train[train['gender'] == 'female']
Femaletext=Female['text']
FemaleDes=Female['description']
yf=Female["gender"]
ListFemale=Femaletext.values.tolist()+FemaleDes.values.tolist()
```

Merging

Since we created different tables for both gender. We will merge all the tweets in two documents, for one male and one for female. We do that because we are interested to know which words are the most used by male and female.

```
#Merge all Male Text in a single string
TextMale=" "
for i in range(0,len(ListMale)):
  TextMale=TextMale+ListMale[i]
textmale=TextMale.lower()
```

```
#Merge all Female Text in a single string
TextFemale=" "
for i in range(0,len(ListFemale)):
  TextFemale=TextFemale+ListFemale[i]
textfemale=TextFemale.lower()
```

Tokenization, Stop Words, Lemmatization

Here, we created a function that will proceed text cleaning.
First, the tokenization will allow us to separate each word and to be able to calulate the frequency of every single word. The tokenization's technique chosen is the white-space one.
Now that we have tokens, we want to clean the text by removing stop words and other useless tokens such as "\x89\x9d_\x95ü\x8f\x89\x9d_\x95ü\x8f\x89\x9d_\x95ü\x8f". To do so, we will use a stop words list available in spacy and filter words that are not in the english dictionnary available on pyenchant. We are aware that by using an english dictionnary we may lose some expression that are used on twitter. But for us, it was the only sustainable solution to filter non-sense tokens. We will also test this function without the dictionnary to see which one gives use the best accuracy.
Finally we did some lemmatization.

```
#Cleaning function
def cleaning(input):
  nlp = English()
  Doc=nlp(input)
  Token = []
#Check if words in english dictionnary
  for token in Doc:
    Token.append(token.text)
  Words=[]
  d = enchant.Dict("en_US")
  Text=''
  for i in Token:
    if d.check(i)==True and i != ' ': #comment this line to try without dictionnary
      Text+=' '+i         #remove an indenatation to try without dictionnary
  doc=nlp(Text)
#Remove stop words and punctuation
  for word in doc:
    if word.is_stop==False and word.is_punct==False:
      Words.append(word)
  words=''
#Lemmatization
  for j in Words:
    lemm=lemmatizer.lemmatize(str(j))
    words+=lemm+' '
  words=words.lower()
  return(words)


#Tokenizer found in the DMML class lab 6.1
import string
from spacy.lang.en.stop_words import STOP_WORDS
from spacy.lang.en import English

# Create our list of punctuation marks
punctuations = string.punctuation

# Create our list of stopwords
nlp = spacy.load('en')
stop_words = spacy.lang.en.stop_words.STOP_WORDS

# Load English tokenizer, tagger, parser, NER and word vectors
parser = English()

# Creating our tokenizer function
def spacy_tokenizer(sentence):
    # Creating our token object, which is used to create documents with linguistic annotations.
    mytokens = parser(sentence)

    # Lemmatizing each token and converting each token into lowercase
    mytokens = [ word.lemma_.lower().strip() if word.lemma_ != "-PRON-" else word.lower_ for word in mytokens ]

    # Removing stop words
    mytokens = [ word for word in mytokens if word not in stop_words and word not in punctuations ]

    # return preprocessed list of tokens
    return mytokens
```

Now, we will apply this function to both documents.

```
Fem=cleaning(textfemale)
Mal=cleaning(textmale)
```

Bag Of Words (BOW)

Here we are going to create a table with a line per gender. The columns are all the words used on the train tweet of our dataset. As we merged all the tweet, it would have made no sense to set bag of words of n-grams other than n=1. Because at some point some BOW would have been wirtten by two different people. We create two functions, the first is to plot the BOW for our female and male documents, the second one will plot the BOW with these documents and the tweet for which we want to predict the gender.
As we only have 2 documents, doing some TF-IDF would not make a lot of sence.

```
#Creation and ploting of bag of words for two documents
def bagofwords(Fem,Mal):
  text=[Fem,Mal]
  count = CountVectorizer()
  bow = count.fit_transform(text)
  feature_names = count.get_feature_names()
  BagOfWords=pd.DataFrame(bow.todense(),columns=feature_names, index=['Female','Male'])
  return BagOfWords.head()
```

```
#Creation and ploting of bag of words for three documents
def BOW(clean):
  clean=cleaning(clean)
  text=[Fem,Mal,clean]
  count = CountVectorizer()
  bow = count.fit_transform(text)
  feature_names = count.get_feature_names()
  BagOfWords=pd.DataFrame(bow.todense(),columns=feature_names, index=['Female','Male','Test'])
  return BagOfWords.head()
```

```
bagofwords(Fem,Mal)
```

	00	000	00000000000000002344	01	01234	017	01912843723	0305	031012	04	044	05	06	07	07840458711	07854	07989
Female	2	4	1	3	0	0	0	0	0	2	1	2	1	1	1	0	1
Male	0	16	0	2	1	1	1	1	1	0	0	1	2	2	0	1	0

2 rows × 10963 columns

Cleaning of the Test Data

This part is a little bit more technical than the first cleaning we did. In the first case, we merged all tweet so we had only two documents.
Now, we want to clean tweet by tweet and have several clean documents. As we cleaned the *train* set, we need to apply the same method to the *test* set. The goal of this section is to build a list of *cleaned tweets* by gender.
This time, knowing that our goal is to predict gender by tweet only, we won't take into account the account's description

```
#Here, we will set the test set as random mix of tweet.
TestSet = test[(test.gender=='male') | (test.gender=='female')]
X_test=TestSet["text"]
y_test=TestSet["gender"]
Test=X_test.values.tolist()
```

```
Clean_Test=[]
for i in Test:
  Clean_Test.append(spacy_tokenizer(i))
Test_Clean=[]
for i in range(0, len(Clean_Test)):
  Test_Clean.append(' '.join(Clean_Test[i]))
```

Classifying

Now, the funny part begins! As explained before, the first method chosen is the cosine-similarity. To do so, we created a function that will compute the cosine-similarity of each test tweet with our two documents (female's one and male's one). The function will classify the tweet accordingly to the highest cosine-similarity. In our case, the base rate is female, but in case of an equality situation between the cosine-similarities, we will classify the tweet as a female. Anyway, we will check the baserate for the overall twitter base user and not for our dataset

```
#Classification with Cosine Similarity
def classifier(clean):
```

```
  clean=cleaning(clean)
  y_pred=[]
  text=[Fem,Mal,clean]
  count = CountVectorizer()
  bow = count.fit_transform(text)
  cosine =cosine_similarity(bow[2],bow)
  if cosine[0][0]>=cosine[0][1]:
    y_pred.append("female")
  else:
    y_pred.append("male")
  return y_pred


#might take some time as we have a lot of data, model not optimized...
y_pred=[]
for i in Test_Clean:
  y_pred.append(classifier(i))
```

Accuracy

```
accuracy_score(y_pred,y_test)
```

```
0.5831111111111111
```

We see that our accuracy is higher than the BaseRate of the model. It means that the accuracy will be higher by using our model than classifying every tweet as written by female. But the difference is not huge. Therefor, we will try with another model.

Predict Tweet's gender

Here, you can input a tweet of your choice and our model will try to predict the author's gender.

```
#Congratulations to Boris Johnson on his great WIN! Britain and the United States will now be free to strike a massive new Trade Deal
Tweet=input("Enter a Tweet: ")
print('This tweet was probably written by a:',classifier(Tweet))
BOW(Tweet)
```

```
Enter a Tweet: Congratulations to Boris Johnson on his great WIN! Britain and the United States will now be free to strike a mass
This tweet was probably written by a: ['male']
```

	00	000	00000000000000002344	01	01234	017	01912843723	0305	031012	04	044	05	06	07	07840458711	07854	07989
Female	2	4	1	3	0	0	0	0	0	2	1	2	1	1	1	0	1
Male	0	16	0	2	1	1	1	1	1	0	0	1	2	2	0	1	0
Test	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3 rows × 10970 columns

Logistic Regression - Sentiment Analysis Model

In this part, we will predict the gender with a logistic Regression. To do so, we will try to adapt a *Sentiment Analysis* model. The equivalent of the sentiment will be the gender. To do so, we won't use the merge documents for the training part. We will keep all tweets as an input.

Data Cleaning

For this model, we need to clean all our tweets one by one. This time, we won't use the description.

```
TrainSet = train[(train.gender=='male') | (train.gender=='female')]
X_train=TrainSet["text"]
y_train=TrainSet["gender"]
Train=X_train.values.tolist()


Clean_Train=[]
for i in Train:
  Clean_Train.append(spacy_tokenizer(i))
Train_Clean=[]
for i in range(0, len(Clean_Train)):
  Train_Clean.append(' '.join(Clean_Train[i]))
```

Classifying

For this model, we will use a cleaning function that was given during our DMML class (see lab 6.1). The cleaning methods works better with this model than the function we wrote ourself. However, our function is better with the cosine-similarity than that cleaning's function.

```
tfidf_vector = TfidfVectorizer(tokenizer = spacy_tokenizer)
from sklearn.linear_model import LogisticRegression
classifier2 = LogisticRegression(solver="lbfgs")

# Create pipeline using Bag of Words
pipe = Pipeline([('vectorizer', tfidf_vector),
                 ('classifier2', classifier2)])

# model generation
y_train=TrainSet["gender"]
pipe.fit(Train,y_train)
```

```
Pipeline(memory=None,
             steps=[('vectorizer',
                     TfidfVectorizer(analyzer='word', binary=False,
                                     decode_error='strict',
                                     dtype=<class 'numpy.float64'>,
                                     encoding='utf-8', input='content',
                                     lowercase=True, max_df=1.0, max_features=None,
                                     min_df=1, ngram_range=(1, 1), norm='l2',
                                     preprocessor=None, smooth_idf=True,
                                     stop_words=None, strip_accents=None,
                                     sublinear_tf=False,
                                     token_patt...
                                     tokenizer=<function spacy_tokenizer at 0x7fdd55ea0ae8>,
                                     use_idf=True, vocabulary=None)),
                    ('classifier2',
                     LogisticRegression(C=1.0, class_weight=None, dual=False,
                                        fit_intercept=True, intercept_scaling=1,
                                        l1_ratio=None, max_iter=100,
                                        multi_class='warn', n_jobs=None,
                                        penalty='l2', random_state=None,
                                        solver='lbfgs', tol=0.0001, verbose=0,
                                        warm_start=False))],
             verbose=False)
```

Accuracy

```
from sklearn import metrics
# Predicting with a test dataset
predicted = pipe.predict(Test)

# Model Accuracy
print(" test Accuracy:",metrics.accuracy_score(y_test, predicted))
```

```
test Accuracy: 0.5897777777777777
```

We conclude that the accuracy in this case is quite similar to the accuracy obtained with the cosine similarity.

Predict Tweet's gender

```
Tweet=input("Enter a Tweet: ")
Tweet=[Tweet]
print('This tweet was probably written by a:',pipe.predict(Tweet))
```

```
Enter a Tweet: Congratulations to Boris Johnson on his great WIN! Britain and the United States will now be free to strike a mass
This tweet was probably written by a: ['male']
```

SK-Learn techniques

As we are not satisfied by our accuracy, we will try other methods. Here we will use the differents methods available on the Scikit-Learn library. For this method, the input of the model cannot be a list of words. We will input array that can be represent graphically as bag of words. To construct this method, we took example of the Lab Session of the DMML class. We will use the data that we cleaned previously and transform them in arrays.

Data transformation

In this part, we will transform ours lsits of words into array.

```
Train_texts = Train_Clean
#transform the list of words into array
count = CountVectorizer()
count2=CountVectorizer(ngram_range=(1, 2))
bow_train = count.fit_transform(Train_texts)
bow_train2=count2.fit_transform(Train_texts)
bow_train.toarray()
bow_train2.toarray()
feature_names = count.get_feature_names()


#Add a column with the gender and transfrom 'male' as 0 and 'female' as 1
y_train=TrainSet["gender"]
y_train= y_train.replace({'male':0})
y_train = y_train.replace({'female':1})
y_train=y_train.values.tolist()


#transform the list of words into array
y_test=TestSet['gender']
Texts_test = Test_Clean
bow_test = count.transform(Texts_test)
bow_test2 = count2.transform(Texts_test)
bow_test.toarray()
bow_test2.toarray()
#Change the value of the test set as it matches the train one
y_test=y_test.replace({'male':0})
y_test=y_test.replace({'female':1})
```

Bag Of Words DataFrame

The columns of this dataframe represent all words available in our dataset. To win some time, we didn't use the english dictionnary in the cleaning process. We have some non-sense token but on the other hand, we can capture some words that are only use on social-networks.

```
#Create a bag of words DataFrame
BoW=pd.DataFrame(
    bow_train.todense(),
    columns=feature_names
    )


#with n-gram 1 and 2, it might crash
BoW['Gender']=y_train
cols = list(BoW.columns)
cols = [cols[-1]] + cols[:-1]
BoW = BoW[cols]


#Display the BagOfWords
BoW.head()
```

	Gender	00	000	00000000000000002344	007	00am	00irccrrhb	00sev	00zaue0wbk	01	017	01iubj2hqu	029fj3orkc	02glrddxzw	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

5 rows × 21090 columns

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(ngram_range=(1, 1))
tfidf_train = tfidf.fit_transform(Train_texts)
tfidf_train.toarray()
TFIDF=pd.DataFrame(tfidf_train.todense(),columns=tfidf.get_feature_names())
TFIDF['Gender']=y_train
cols = list(TFIDF.columns)
cols = [cols[-1]] + cols[:-1]
TFIDF = TFIDF[cols]
TFIDF.head()
```

	Gender	00	000	0000000000000002344	007	00am	00irccrrhb	00sev	00zaue0wbk	01	017	01iubj2hqu	029fj3orkc	02glrddxzw
0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 21090 columns

```
tfidf_test=tfidf.transform(Texts_test)
tfidf_test.toarray()
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

Dimensionality-Reduction

As we saw in the last lecture, we can gain in accuracy and in computing time by applying dimensionality-reduction.

```
from sklearn.decomposition import TruncatedSVD
from scipy import sparse as sp
clf = TruncatedSVD()
Xpca_Test = clf.fit_transform(bow_test)
Xpca_Train = clf.fit_transform(bow_train)
```

Now, from this part, we have clean all the data we need. So the end of the notebook will consist of training different models and computre their accuracy. That's why it will be less commented.

Decision Tree

```
#define the classifier, the max_depth parameter is defined thanks to a "test" done later
clf = DecisionTreeClassifier(criterion='entropy', max_depth=781,random_state=14)
```

Accuracy

```
#Train the model, first we no dimensionality reduction
clf.fit(bow_train, y_train)
clf.score(bow_test,y_test)
```

```
0.5822222222222222
```

```
#try with n-grams=1,2
clf.fit(bow_train2,y_train)
clf.score(bow_test2,y_test)
```

```
0.5697777777777778
```

```
#try with TF-IDF
clf.fit(tfidf_train,y_train)
clf.score(tfidf_test, y_test)
```

```
0.5444444444444444
```

```
#train and test the model after the dimensionality reduction --> result is worst than the base-rate...
clf.fit(Xpca_Train, y_train)
clf.score(Xpca_Test,y_test)
```

```
0.48488888888888887
```

Best accuracy with n-gram n=1.
The accuracy of this model is quite good. But to try to improve it, we will use another model based on decision tree: *Random Forest Classifier*.

This model is descibed later.
We could have represent the model graphically, but as we set an optimal depth at 781, it would not have been readable.

Test the Best Depth

Here, we will try to find the optimum depth of our decision tree. The result for the depth are pretty sursprinsing. One can think that small depth would be better and other can think the opposite. But in fact, we cannot tell what is the best. The accuracy can increase with the depth but can also decrease. It looks like it is a little bit random...

```
#if you don't want to run it, optimum : Depth =781   Accuracy=58.2 % --> we may increase the range... or reduce the steps..
max_acc=0
depth=0
for i in range(1,1000,5):
  clf = DecisionTreeClassifier(criterion='entropy',max_depth=i,random_state=14)
  clf.fit(bow_train, y_train)
  if clf.score(bow_test,y_test) > max_acc:
    max_acc= clf.score(bow_test,y_test)
    depth=i
    print('depth:', i,' acc =', max_acc )
```

Predict Tweet's gender

```
clf.fit(bow_train, y_train)
Tweet=input('Enter a Tweet: ')
Tweet=[Tweet]
bow_tweet= count.transform(Tweet)
bow_tweet=bow_tweet.toarray()
Gender=clf.predict(bow_tweet)
if Gender ==0:
  Gender='male'
else:
  Gender='female'
print('This tweet was probably written by a:',Gender)
```

```
Enter a Tweet: Congratulations to Boris Johnson on his great WIN! Britain and the United States will now be free to strike a mass
This tweet was probably written by a: male
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rclf=RandomForestClassifier(criterion='entropy',max_depth=371, random_state=14, n_estimators=76)
```

```
rclf.fit(bow_train,y_train)
rclf.score(bow_test,y_test)
```

```
0.6173333333333333
```

```
#Try with n-grams n=1,2
rclf.fit(bow_train2,y_train)
rclf.score(bow_test2,y_test)
```

```
0.5982222222222222
```

```
#try with TF-IDF
rclf.fit(tfidf_train,y_train)
rclf.score(tfidf_test,y_test)
```

```
0.592
```

```
#train and test the model after the dimensionality reduction --> result is worst than the base-rate...
rclf.fit(Xpca_Train, y_train)
rclf.score(Xpca_Test,y_test)
```

```
0.4928888888888889
```

Test of the "best" parameters

We will focus on two distinct parameters: depth and n_estimators. The optimal way to find the best parameters is to do iteration on depth and n_estimators on the same time. For example, for depth = 1 try all n_estimators between 1 and 1000. Then do the same with depth =2. It would

have taken to much time. We decided to find the best depth with the default n_estimators (=10). Then having the best depth, we iterated the number of estimators according this depth.
Like before, we cannot say if small number or big number are better or not.

```
#depth: 371  acc = 0.5964444444444444
max_acc=0
depth=0
for i in range(1,1000,5):
  rclf=RandomForestClassifier(criterion='entropy',max_depth=i, random_state=14,n_estimators=10)
  rclf.fit(bow_train, y_train)
  if rclf.score(bow_test,y_test) > max_acc:
    max_acc= rclf.score(bow_test,y_test)
    depth=i
    print('depth:', i,' acc =', max_acc )
```

```
#Nothing was higher than the combination depth=371 and n_estimators=76 --> to find the best combination, should have done iterations o
max_acc=0
n_estim=0
for i in range(1,1000,5):
  rclf=RandomForestClassifier(criterion='entropy',max_depth=371, random_state=14,n_estimators=i)
  rclf.fit(bow_train, y_train)
  if rclf.score(bow_test,y_test) > max_acc:
    max_acc= rclf.score(bow_test,y_test)
    n_estim=i
    print('n_estim:', i,' acc =', max_acc )
```

Best accuracy with n-gram n=1, max_depth= 371 and n_estimators = 76. Not all the combinations were tested.
This is the model that gives the best accuracy of this notebook. For us, this result is possible thanks to the way a random forest works. Having multiple random decision trees increases our chance of having better predictions than using only one decision tree. Because a tweet is not a long text, it is useful to try different variant of classification on it. It is exactly how works a random forest, a tweet may not be classified the same way for every tree of the forest.

Predict Tweet's gender

```
rclf.fit(bow_train, y_train)
Tweet=input('Enter a Tweet: ')
Tweet=[Tweet]
bow_tweet= count.transform(Tweet)
bow_tweet=bow_tweet.toarray()
Gender=rclf.predict(bow_tweet)
if Gender ==0:
  Gender='male'
else:
  Gender='female'
print('This tweet was probably written by a:',Gender)
```

```
Enter a Tweet: Congratulations to Boris Johnson on his great WIN! Britain and the United States will now be free to strike a mass
This tweet was probably written by a: male
```

Basic Logistic Regression

Let's try a more 'basic' regression. This one should not give weight to words as the one working as a sentiment analysis does.

```
from sklearn.linear_model import LogisticRegressionCV
LR = LogisticRegressionCV(solver='lbfgs', cv=5,max_iter=1000, random_state=14)
```

```
LR.fit(bow_train,y_train)
LR.score(bow_test,y_test)
```

```
0.6004444444444444
```

```
#Try with n-grams n=1,2
LR.fit(bow_train2,y_train)
LR.score(bow_test2,y_test)
```

```
0.5924444444444444
```

```
#try with TF-IDF
LR.fit(tfidf_train,y_train)
LR.score(tfidf_test, y_test)
```

```
0.5086666666666667
```

```
#With dimensionality reduction
LR.fit(Xpca_Train, y_train)
LR.score(Xpca_Test,y_test)
```

```
0.5168888888888888
```

Best accuracy n-gram n=1.
This model gives one of the best result with an accuracy of 60%. But a downside is that it takes more time than the other to do the computation.

Predict Tweet's gender

```
LR.fit(bow_train, y_train)
Tweet=input('Enter a Tweet: ')
Tweet=[Tweet]
bow_tweet= count.transform(Tweet)
bow_tweet=bow_tweet.toarray()
Gender=LR.predict(bow_tweet)
if Gender ==0:
  Gender='male'
else:
  Gender='female'
print('This tweet was probably written by a:',Gender)
```

```
Enter a Tweet: Congratulations to Boris Johnson on his great WIN! Britain and the United States will now be free to strike a mass
This tweet was probably written by a: male
```

KNN Classification

The last method we will try is a the k-nearest-neighbor.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10, weights='distance')
```

```
knn.fit(bow_train,y_train)
knn.score(bow_test,y_test)
```

```
0.5191111111111111
```

```
#Try with n-grams n=1,2
knn.fit(bow_train2,y_train)
knn.score(bow_test2,y_test)
```

```
0.5262222222222223
```

```
#try with TF-IDF
knn.fit(tfidf_train,y_train)
knn.score(tfidf_test, y_test)
```

```
0.5648888888888889
```

```
#With dimensionality reduction
knn.fit(Xpca_Train,y_train)
knn.score(Xpca_Test,y_test)
```

```
0.5088888888888888
```

Best accuracy with TF-IDF and n-gram n=1.
This is the only model that works better with an input different of a bag of words with n-gram. The best accuracy of this model is the one using the TF-IDF approach.
Something intersting with the KNN method is that we can set weight or not. In this case, it makes sense to give weight to the closest points. The nearest it will be, the most impact it will have on the prediction.

Predict Tweet's gender

```
knn.fit(bow_train,y_train)
Tweet=input('Enter a Tweet: ')
Tweet=[Tweet]
bow_tweet= count.transform(Tweet)
bow_tweet=bow_tweet.toarray()
```

```
Gender=knn.predict(bow_tweet)
if Gender ==0:
  Gender='male'
else:
  Gender='female'
print('This tweet was probably written by a:',Gender)
```

```
Enter a Tweet: Congratulations to Boris Johnson on his great WIN! Britain and the United States will now be free to strike a mass
This tweet was probably written by a: female
```

Gender Prediction

In this part, we will use all the methods tested before. The goal is to take all the predictions and to see which vote (male or female) is the most frequent. To do so, we will use the basic version of each method. (i.e. the one without dimensionality reduction). For the fit, we chose the one with the best accuracy for each model.

```
#re-fit the algo with the input data
knn.fit(tfidf_train,y_train)
LR.fit(bow_train,y_train)
clf.fit(bow_train,y_train)
rclf.fit(bow_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
                       max_depth=371, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=76,
                       n_jobs=None, oob_score=False, random_state=14, verbose=0,
                       warm_start=False)
```

```
def GenderPrediction():
  Results=[]
  Tweet=input('Enter a Tweet or a Text: ')
  Results.append(classifier(Tweet))
  Tweet=[Tweet]
  Results.append(pipe.predict(Tweet))
  bow_tweet= count.transform(Tweet)
  tfidf_tweet=tfidf.transform(Tweet)
  tfidf_tweet=tfidf_tweet.toarray()
  bow_tweet=bow_tweet.toarray()
  Results.append(knn.predict(tfidf_tweet))
  Results.append(LR.predict(bow_tweet))
  Results.append(clf.predict(bow_tweet))
  Results.append(rclf.predict(bow_tweet))
  Pred_F=0
  Pred_M=0
  for i in Results:
    if i[0] == 0 or i[0] =='male':
      Pred_M +=1
    else:
      Pred_F+=1
  if Pred_M>Pred_F:
    print('This tweet was probably written by a male.')
  else:
    print('This tweet was probably written by a female.')
```

```
GenderPrediction()
#Congrations to Boris Johnson on his great WIN! Britain and the United States will now be free to strike a massive new Trade Deal afte

#No matter where we or our ancestors are from, millions of Americans will sit down together to break bread today. It's a wonderful rem
```

```
Enter a Tweet or a Text: Congratulations to Boris Johnson on his great WIN! Britain and the United States will now be free to str
This tweet was probably written by a male.
```

Conclusion

To sum up, we found that predicting the gender of someone by analysing his/her tweet is not an easy task. It might be due to the fact that a tweet is not long enough to collect enough data about the author. It may be interesting to adapt the model to longer text such as university thesis or books.

((We could also extend the model to famous author and try to indentify their own writting style. And why not try to write a story according their style. (Something similar to what was done with Rembrandt's painting).))
However, the majortiy of our techniques led to an accuracy that was higher than the base rate. It means that our model is not useless.

We also tried to apply some dimensionality reduction. The result of that process was disappointing. It may be not appropriate to our model or we were maybe not able to use it properly.

We applied TF-IDF and n-grams n=1,2 for all the model. It only increased the accuracy for the knn model. Our best accuracy was with a random forest model. An accuracy of 61%, which is 10% more than the base rate. Finally, the business solution will use all the techniques tried. It will predict the gender according to the one receiving the most of "votes" from the different methods.