# Group Name : Athanasios


## group's members:

**1- Golnoosh Sharifi - 50011414**
**2- Mahdi Rahimianaraki - 50014390**
**3- Siarhei Sheludzko - 3092139**
**4- Aleksei Zhuravlev - 50104961**
**5- Marcel Melchers - 2897058**


**1.1)** Explain what the following text preprocessing concepts are, how they function, and why they are useful.

## 1. Sentence tokenization

Sentence tokenization is the process of splitting text into individual sentences.
It splits a piece of text into sentences based on characters like “.”, “;”, “:”, “!”, “?”.
These sentence tokens help in understanding the context or developing the model for the NLP.

## 2. Word tokenization

This is the most commonly used tokenization technique. It splits a piece of text or sentences into words based on a delimiter. The most commonly used delimiter is space. You can also split your text using more than one delimiter, like space and punctuation marks.
With both sentence tokenizer and word tokenizer, we can count average words per sentence. Such output serves as an important feature for machine training as the answer would be numeric.

## 3. Part-of-speech (POS) tagging

Part-of-Speech (PoS) tagging is defined as the process of assigning one of the parts of speech to the given word. In simple words, we can say that POS tagging is a task of labeling each word in a sentence with its appropriate part of speech. We already know that parts of speech include nouns, verbs, adverbs, adjectives, pronouns, conjunction and their sub-categories.

There are several techniques for POS tagging. One of the oldest techniques of tagging is rule-based POS tagging. Rule-based taggers use a dictionary or lexicon for getting possible tags for tagging each word. Stochastic POS Tagging, Transformation-based Tagging, Hidden Markov Model, Use of HMM for POS Tagging are also other techniques.

POS tags make it possible for automatic text processing tools to take into account which part of speech each word is. This facilitates the use of linguistic criteria in addition to statistics.

## 4. Lemmatization

A lemmatization refers to grouping together words that have the same root or lemma but differ in their inflections or derivatives to allow for easier analysis.By removing inflectional suffixes and prefixes, the dictionary form of the word emerges.
lemmatization, enables computers to extract relevant information from a particular set of text.
Lemmatization brings great value where it is crucial to understand the meaning of a user’s messages, for example in a chatbot.

## 5. Stop word removal

The concept is simply to remove the words that appear in all documents in the corpus. In stop word removal, our input is checked by our criteria (for example the part of speech of a certain word) based on the value of the tokens, and less valuable tokens are removed.
There are also different libraries used for the removal of stop words such as: Natural Language Toolkit (NLTK), spaCy, Gensim, Scikit-Learn,
Taking away these words allows us to focus on the important information by removing low-level information. It helps the model to take into account only key features when these words are removed. There is not much information contained in these words as well. As a result of eliminating them, we can concentrate on the important ones.
Removal of stop words definitely reduces the dataset size and thus reduces the training time due to the fewer number of tokens involved in the training.

## 1-2)

### Our solution link in google colab:

https://colab.research.google.com/drive/1CuG-__MsKCGxjrCjbzO6W-P8RqhwGsk1?usp=sharing

```
from nltk import tokenize
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

# nltk.download("all")

def process_paragraph(paragraph):
   # sentces = paragraph.split(".")
   sentces = tokenize.sent_tokenize(paragraph)
   list_each_sentence = []
   lemmatizer = WordNetLemmatizer()

   for p in range(len(sentces)):
       list_each_sentence.append(word_tokenize(sentces[p]))

   for j in range(len(list_each_sentence)):
       for h in range(len(list_each_sentence[j])):
           list_each_sentence[j][h] =
lemmatizer.lemmatize(list_each_sentence[j][h])

   result_senteces = []

   for j in range(len(list_each_sentence)):
       mkk = []
       for h in range(len(list_each_sentence[j])):
           if list_each_sentence[j][h].lower() not in stopwords.words("english"):
               mkk.append(list_each_sentence[j][h])
       result_senteces.append(mkk)

   return result_senteces


paragraph_input = "Here we will implement the Polynomial Regression using Python.
We will understand it by comparing " \
           "Polynomial Regression model with the Simple Linear Regression model. So
first, let's understand the " \
```

```
            "problem for which we are going to build the model. "

result = process_paragraph(paragraph_input)

for i in range(len(result)):
    print(result[i])
```

## 2.1 Odds Ratio

Let's reduce the table only to 2 categories: „Neither" and „Contacted"

| Spam | Contacted | Neither | Total |
|---|---|---|---|
| True | 59 | 926 | 985 |
| False | 1418 | 1553 | 2871 |
| Total | 1477 | 2479 | 3856 |

odds of spam in „Contacted" = $59/1418 = 0{,}0416$

odds of spam in „Neither" = $926/1553 = 0{,}5962$

odds ratio of spam $= \frac{0{,}0416}{0{,}5962} = 0{,}0697 < 1$

Interpretation:

Getting a spam Email from Contacted-category is about 14 times less probable, then from category Neither.

## 2-4)

**First version of the implementation 2.4.**

**The link of our solution in google colab:**

https://colab.research.google.com/github/mahdimRa/Logistic_Regression_spam_email/blob/main/2_4.ipynb

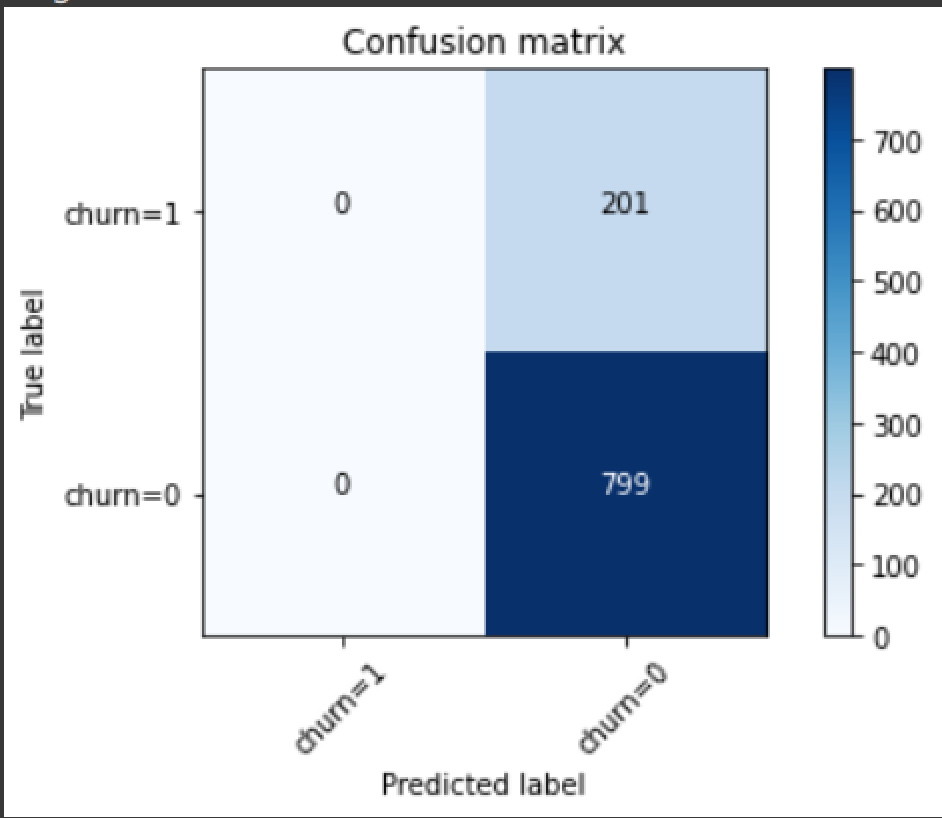
```
import random
import pandas as pd

spam_list = []
not_spam_list = []


def create_line(name_list, count, contact_type, email_type):
    for i in range(count):
        name_list.append([contact_type, email_type])
    return name_list


def list_shuf(list_name, number):
    for i in range(number):
        random.shuffle(list_name)


spam_list = create_line(spam_list, 15, "Colleague", True)
spam_list = create_line(spam_list, 59, "Contacted", True)
spam_list = create_line(spam_list, 926, "Neither", True)

list_shuf(spam_list, 3)


not_spam_list = create_line(not_spam_list, 1029, "Colleague", False)
not_spam_list = create_line(not_spam_list, 1418, "Contacted", False)
not_spam_list = create_line(not_spam_list, 1553, "Neither", False)

list_shuf(not_spam_list, 3)


last_list = spam_list + not_spam_list
list_shuf(last_list, 3)

df = pd.DataFrame(last_list)
df.columns = ['contact_type', 'spam']
df.to_csv('dataSetEmail.csv', index=False)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import jaccard_score
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
import itertools
from sklearn.metrics import log_loss

df = pd.read_csv("dataSetEmail.csv")

X = np.asarray(df[["contact_type"]])
# print(X[0:5])

y = np.asarray(df[["spam"]])
# print(y[0:5])

le_contact = preprocessing.LabelEncoder()
le_contact.fit(["Colleague", "Contacted", "Neither"])
X[:, 0] = le_contact.transform(X[:, 0])
# print(X[0:5])

le_spam = preprocessing.LabelEncoder()
le_spam.fit([True, False])
y[:, 0] = le_spam.transform(y[:, 0])
# print(y[0:5])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=4)
print("Train set:", X_train.shape, y_train.shape)
print("Test set:", X_test.shape, y_test.shape)

LR = LogisticRegression(C=0.08, solver="saga").fit(X_train, np.ravel(y_train))
yhat = LR.predict(X_test)
yhat_prob = LR.predict_proba(X_test)
print("Jacard Score is", jaccard_score(y_test, yhat, pos_label=0))


def plot_confusion_matrix(
   cm, classes, normalize=False, title="Confusion matrix", cmap=plt.cm.Blues
):
   """
   This function prints and plots the confusion matrix.
   Normalization can be applied by setting `normalize=True`.
   """
   if normalize:
       cm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]
```

```python
        print("Normalized confusion matrix")
    else:
        print("Confusion matrix, without normalization")

    print(cm)

    plt.imshow(cm, interpolation="nearest", cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = ".2f" if normalize else "d"
    thresh = cm.max() / 2.0
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(
            j,
            i,
            format(cm[i, j], fmt),
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black",
        )

    plt.tight_layout()
    plt.ylabel("True label")
    plt.xlabel("Predicted label")


print(confusion_matrix(y_test, yhat, labels=[1, 0]))
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1, 0])
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(
    cnf_matrix,
    classes=["churn=1", "churn=0"],
    normalize=False,
    title="Confusion matrix",
)

print("log loss: ",log_loss(y_test, yhat_prob))
```

**Result:**

```
Train set: (4000, 1) (4000, 1)
Test set: (1000, 1) (1000, 1)
Jacard Score is 0.799
[[  0 201]
 [  0 799]]
Confusion matrix, without normalization
[[  0 201]
 [  0 799]]
log loss:  0.38130111912792336
```



## Second version of implementation 2.4:

### The link of our solution in google colab:

https://colab.research.google.com/drive/1P2zWi245w3MtYig4BCumR9knuJvMiu4_?usp=sharing

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import jaccard_score
```

```
def dup_rows(a, indx, num_dups=1):
   return np.insert(a,[indx+1]*num_dups,a[indx],axis=0)

one_hot_template = np.array([[1, 0, 0, 1],
                            [1, 0, 0, 0],
                            [0, 1, 0, 1],
                            [0, 1, 0, 0],
                            [0, 0, 1, 1],
                            [0, 0, 1, 0]])
one_hot_template =  dup_rows(one_hot_template, 5, 1552)
one_hot_template =  dup_rows(one_hot_template, 4, 925)
one_hot_template =  dup_rows(one_hot_template, 3, 1417)
one_hot_template =  dup_rows(one_hot_template, 2, 58)
one_hot_template =  dup_rows(one_hot_template, 1, 1028)
one_hot_template =  dup_rows(one_hot_template, 0, 14)
```

```
one_hot_df = pd.DataFrame(one_hot_template, columns=['colleague', 'contacted',
'neither', 'spam'])
```

```
one_hot_df.head(17)
```

```
# Test train split
X_train, X_test, y_train, y_test = train_test_split(one_hot_df.drop('spam',
axis=1), one_hot_df['spam'], test_size=0.30, random_state=101)
```

```
# Train and fit a logistic regression model on the training set.
LogReg = LogisticRegression(solver='saga')
LogReg.fit(X_train, y_train)
# Predict if the email is spam or not
```

```
LogReg.predict(np.array([[1, 0, 0]]))
```

```
# Score the model
LogReg.score(X_test, y_test)
```

```
jaccard_score(y_test, LogReg.predict(X_test), pos_label=0)
```

**2-5)**

2.5 Trustworthiness

Strength of logistic regression:

- easy to implement and interpret
- fast to train
- very fast in classifying unknown items

Weaknesses of logistic regression:

- number of observations should be higher than the number of the features. Otherwise it will lead to overfitting
- -> Remedy: enough big training set
- Unable to extract complex relationships from data
  -> Remedy: use other approaches ...

Predictions of our model are based only on the category of the emails, do not taking into account the content of the emails

2-6)

Task 2.6

Tf-idf - combination of term frequency (tf) and inverse document frequency (idf)
let's start first with the definitions of this 2 ~~frequencies~~ frequencies.

$Tf_{td}$ - term frequency of term **t** in document **d** is the number of times that **t** occurs in **d**.

Usually is used a „normalized" tf

$$tf_{td} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{if count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

log function is applied to prevent overweighting of long documents

df - document frequency. It's the number of documents in the collection that term occurs in
df is important for ranking
Our goal is to make matching based on more rare terms, as the more frequent terms are less informative than rare. We can do it by setting higher weights for rare terms.
At the same time more frequent terms are not ignored, they get the positive weights too, but not as high as rare terms.
And df gives this intuition into the matching score. But to use it we have to invert it.
Then we get $idf_t$ (inverse document frequency)

$idf_t$ is the measure of term selectivity

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$

N - number of docs in the collection

The **tf-idf** weighted value $w_{t,d}$ for word t in document d can be calculated so:

$$w_{t,d} = tf_{t,d} \times idf$$

(idf only affects queries with multiple terms)

Literature and Resources:


- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambrige University Press. 2008

- Prof. Dr. Elena Demidova - Advanced Methods of Information Retrieval, University Bonn, Lecture Advanced Methods of IR MA-INF4230. Foundations of text Retrieval.


Do you think an approach using tf-idf encoder is superior to the one described above?

Yes. Such encoder is able to analyze the content of the emails. It ~~...~~ uses the "bag of words" approach, and is unable to extract some complicated relationships, but is much better than logistic regression model analyzing email-labels

Example of tf~~-i~~df matrix calculation

Given corpus of sentences:

- My favorite fruit is apple
- I like playing sports
- I don't enjoy sports
- My mother is working

For simplicity lets use only lower case, punktuation removal and tokenization as preprocessing steps

td - matrix

| | D1 | D2 | D3 | D4 | $df_t$ | $idf_t$ |
|---|---|---|---|---|---|---|
| my | 1 | 0 | 0 | 1 | 2 | 0,301 |
| favorite | 1 | 0 | 0 | 0 | 1 | 0,602 |
| fruit | 1 | 0 | 0 | 0 | 1 | 0.602 |
| is | 1 | 0 | 0 | 1 | 2 | 0,301 |
| apple | 1 | 0 | 0 | 0 | 1 | 0,602 |
| i | 0 | 1 | 1 | 0 | 2 | 0,301 |
| like | 0 | 1 | 0 | 0 | 1 | 0,602 |
| playing | 0 | 1 | 0 | 0 | 1 | 0,602 |
| sports | 0 | 1 | 1 | 0 | 2 | 0,301 |
| dont | 0 | 0 | 1 | 0 | 1 | 0,602 |
| enjoy | 0 | 0 | 1 | 0 | 1 | 0,602 |
| mother | 0 | 0 | 0 | 1 | 1 | 0,602 |
| working | 0 | 0 | 0 | 1 | 1 | 0,602 |

td- idx

| | D1 | D2 | D3 | D4 |
|---|---|---|---|---|
| my | 0,301 | 0 | 0 | 0,301 |
| favorite | 0,602 | 0 | 0 | 0 |
| fruit | 0,602 | 0 | 0 | 0 |
| is | 0,301 | 0 | 0 | 0,301 |
| apple | 0,602 | 0 | 0 | 0 |
| i | 0 | 0,301 | 0,301 | 0 |
| like | 0 | 0,602 | 0 | 0 |
| playing | 0 | 0,602 | 0 | 0 |
| sports | 0 | 0,301 | 0,301 | 0 |
| dont | 0 | 0 | 0,602 | 0 |
| enjoy | 0 | 0 | 0,602 | 0 |
| mother | 0 | 0 | 0 | 0,602 |
| working | 0 | 0 | 0 | 0,602 |

Every document can now be represented as a column vector.

## 3)

**The link of our solution in google colab:**

https://colab.research.google.com/drive/1BKoWDPOBNU03eMaD_8XYnfH9V3bAjk-3?usp=sharing

```
# download the dataset
import requests
res =
requests.get('http://archive.ics.uci.edu/ml/machine-learning-databases/00228/smsspa
mcollection.zip')
```

```
# write zip to temporary file (needed to unzip)
with open('temp.zip', 'wb') as f:
 f.write(res.content)
```

```
from zipfile import ZipFile

# unzip the dataset
with ZipFile('temp.zip') as myzip:
 with myzip.open('SMSSpamCollection') as myfile:
   spam_raw = myfile.read().decode("utf-8")
   # print(spam_raw)
 with myzip.open('readme') as myfile:
   readme = myfile.read().decode("ISO-8859-1")
```

```
# get features and labels
X_raw = []
y = []
for line in spam_raw.splitlines():
 label = line.split('\t')[0]
 if label == 'ham':
   y.append(0)
 else:
   y.append(1)
 X_raw.append(line.split('\t')[1])

for i in range(5):
 print(X_raw[i], y[i])
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

# preprocessing and encoding
vectorizer = TfidfVectorizer(strip_accents='unicode', stop_words='english')
X = vectorizer.fit_transform(X_raw)
```

```python
features = vectorizer.get_feature_names_out()

print('shape of transformed dataset:', X.shape)
print('example raw entry:', X_raw[1])
print('spam?', y[1])
print('encoded:')
print(X[1])

print('decoded:')
for i, val in enumerate(X.toarray()[1]):
 if val:
   print(features[i], end=' ')
from sklearn.model_selection import train_test_split

# train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2)
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# training linear regression
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

pred_train = log_reg.predict(X_train)
pred_test = log_reg.predict(X_test)
print('linear regression')
print(f'accuracy score on train set: {accuracy_score(y_train, pred_train):.2f}')
print(f'accuracy score on test set: {accuracy_score(y_test, pred_test):.2f}')
```

```python
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(3)
knn.fit(X_train, y_train)

pred_train = knn.predict(X_train)
pred_test = knn.predict(X_test)
print('k-nearest neighbors, k=3')
print(f'accuracy score on train set: {accuracy_score(y_train, pred_train):.2f}')
print(f'accuracy score on test set: {accuracy_score(y_test, pred_test):.2f}')
```