

Natural Language processing

TP 3 – Term Frequency - Inverse Document Frequency (TF – IDF)

Dans ce TP, vous allez mettre en œuvre l'algorithme de TF-IDF, qui est une technique couramment utilisée en NLP et en recherche d'informations. L'objectif de TF-IDF est de quantifier l'importance d'un terme particulier dans un document, par rapport à une collection de documents. Cela peut être utile pour des tâches telles que la détection de spam, où nous voulons identifier les mots les plus indicatifs pour les courriels indésirables.

Vous utiliserez une base de données de spam pour tester votre implémentation de TF-IDF. La base de données consiste en une collection de messages électroniques, certains étant étiquetés comme spam et d'autres comme non spam (également appelés "ham"). Votre tâche consiste à construire un classificateur qui peut prédire si un message électronique donné est du spam ou du ham, en fonction des mots qui apparaissent dans le message.

Pour ce faire, vous commencerez par prétraiter les messages électroniques en les tokenisant et enlevant les mots vides. Ensuite, vous allez implémenter l'algorithme TF-IDF pour calculer le poids de chaque terme dans chaque document. Enfin, vous utiliserez ces poids pour entraîner un classificateur en utilisant un algorithme d'apprentissage supervisé, comme la régression logistique.

Tout au long du laboratoire, vous utiliserez le langage de programmation Python et la bibliothèque scikit-learn. Si vous n'êtes pas familier avec ces outils, ne vous inquiétez pas ! Nous vous fournirons le contexte et les ressources nécessaires pour compléter le TP

Exercice 1 – BOW

1. Importer les dépendances

```
## import statements ##
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import re

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, roc_curve, auc, f1_score

sns.set_context('talk')
sns.set_color_codes()
plot_kwds = {'alpha': 0.25, 's': 80, 'linewidths':0}

import warnings; warnings.simplefilter('ignore')
```

2. Importer les données et déclarer quelques variables

```
data_files = './data/Comment_Spam.xls'

data = pd.read_excel(data_files)
data = data[['Comment', 'Class']]
train_data = data
train_data.head()
```

3. Prétraitemet des données

```
def process_content(content):
    return " ".join(re.findall("[A-Za-z]+",content.lower()))

train_data[ 'processed_comment' ] = train_data[ 'Comment' ].apply(process_content)
```

4. Observer l'avant et l'après prétraitemet

```
train_data.head(20)
```

5. Train, test split

```
X_train, X_test, y_train, y_test = train_test_split(train_data[ 'processed_comment' ],
                                                    train_data[ 'Class' ],test_size=0.2,random_state=57)
```

6. Définir une data pipeline, afin d'implémenter les étapes nécessaires a la construction du modèle de spam

- Nettoyage des données à l'aide de **CountVectorizer** avec *stop_word='english'* pour supprimer les mot vides
- Transformation de données à l'aide de **TF-IDF**
- Entraînemet de modèle à l'aide de **LogisticRegression**

```
model = Pipeline([('vect', CountVectorizer(stop_words='english')),
                  ('tfidf', TfidfTransformer()),
                  ('clf', LogisticRegression()),])
```

7. Exécuter le pipeline

```
#Pipeline execution
model.fit(X_train, y_train)

#Prediction with test set
predicted = model.predict(X_test)

#Confusion matrix calculation
confusion_matrix(y_test,predicted)
```

8. Imprimer le rapport de performance du modele

```
print('accuracy_score',accuracy_score(y_test,predicted))
print('Reporting...')
print(classification_report(y_test,predicted))
```

9. Imprimer les résultats de la validation croisée

```
#cross validation on training
print(cross_val_score(model, X_train, y_train, cv=5))

#cross validation on test
print(cross_val_score(model, X_test, y_test, cv=5))
```

10. Essayer quelques phrases afin de les classifier comme spam ou ham

C1 = 'Subscribe to my Youtube Channel!! :)' (spam)

C1 = 'Hi veronica, hope you are doing good' (ham)

C1 = 'Earn money for being online with 0 efforts! ...' (spam)

```
#testing a comment and doing the classification
c1 = ['Im super happy']
content = pd.DataFrame(c1, columns=['Comment'])
```

11. Faire la prédiction sur les différent exemples

```
#test data pre tretment
test_data = pd.DataFrame(c1, columns=['Comment'])
test_data['processed_comment'] = test_data['Comment'].apply(process_content)
x_test_new = test_data['processed_comment']

#prediction of the class: 0= ham.  1=spam
model.predict(x_test_new)
```