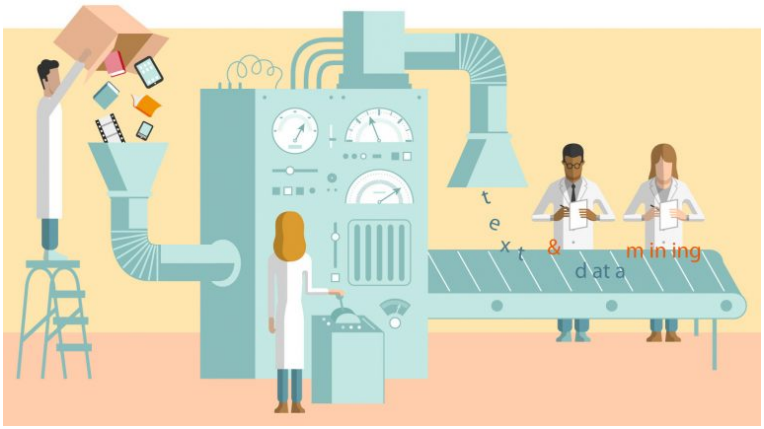
UNIVERSITÉ PARIS 8

Master Informatique Big Data et fouille de données

Data Analysis and Text Mining

Rapport de projet



Presenté par :
Youcef MCIRDI
Mustapha BAAZIZ

Encadré par :
Nistor Grozavu

29 Avril 2020

Table des matières

1 Introduction

L'une des méthodes de traitement du langage naturel les plus utilisées dans les différents problèmes des entreprises est la "classification des textes". Notre objectif pour ce projet est de classer les documents textuels dans des catégories déterminées. L'ensemble de données que nous traitons est "Les 20 données des Newsgroups". Les données des 20 Newsgroups sont la collection d'environ 20 000 documents de newsgroups, divisés en 20 groupes de presse différents. Nous utilisons l'API scikit-learn pour recueillir les données nécessaires à cette tâche

Entrée [2]:

```
categories = ['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

Entrée [3]:

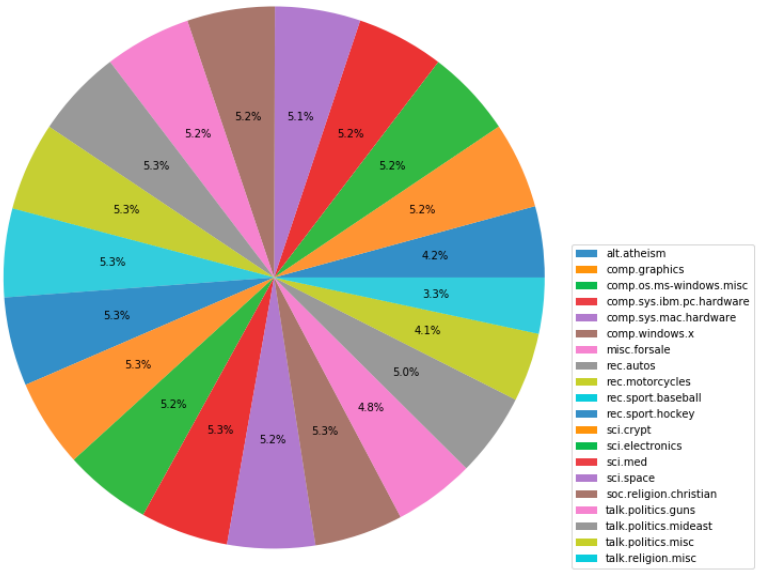
```
 twenty_train = fetch_20newsgroups(subset='train',
...     categories=categories, shuffle=True, random_state=42,remove = ('headers', 'footers', 'quotes'))
pprint(list(twenty_train))

twenty_test = fetch_20newsgroups(subset='test', shuffle=True, remove = ('headers', 'footers', 'quotes'))
pprint(list(twenty_test))
```

```
['data', 'filenames', 'target_names', 'target', 'DESCR']
['data', 'filenames', 'target_names', 'target', 'DESCR']
```

Entrée [4]:

```
print(twenty_train.target_names)
print(twenty_train.target)
print(twenty_train.filenames)
```

Voici une répresentation graphique des catégories ou classes de notre bases de données 20newsgroups



2 Traitement et transformation des textes

Les données textuelles nécessitent une préparation spéciale avant de pouvoir commencer à les utiliser pour la modélisation prédictive.

Le texte doit être analysé pour supprimer des mots, ce qu'on appelle la tokenisation. Ensuite, les mots doivent être encodés en tant qu'entiers ou valeurs à virgule flottante pour être utilisés comme entrée dans un algorithme d'apprentissage machine, appelé extrait de caractéristiques .

Nous devons donc classer les documents, de sorte que chaque document est une "entrée" et qu'une étiquette de classe est la "sortie" de notre algorithme de prédiction. Les algorithmes prennent des vecteurs de nombres en entrée, c'est pourquoi nous devons convertir les documents en vecteurs de nombres de longueur fixe.

La bibliothèque Scikit-learn offre des outils faciles à utiliser pour effectuer à la fois la tokenisation et l'extraction de caractéristiques de vos données textuelles.

Parmi les techniques utilisés :
Bag-of-Words Model : Ce modèle est simple en ce sens qu'il jette toutes les informations d'ordre dans les mots et se concentre sur l'occurrence des mots dans un document
Pour cela, il suffit d'attribuer à chaque mot un numéro unique. Tout document que nous observons peut alors être encodé sous la forme d'un vecteur de longueur fixe avec la longueur du vocabulaire des mots connus. La valeur de chaque position dans le vecteur pourrait être complétée par un compte ou une fréquence de chaque mot dans le document encodé.

C'est le modèle du sac de mots, où nous ne nous intéressons qu'aux schémas d'encodage qui représentent les mots présents ou leur fréquence dans les documents encodés sans aucune information sur l'ordre.
Il existe de nombreuses façons de développer cette méthode simple, à la fois en définissant plus clairement ce qu'est un "mot" et en précisant ce qu'il faut coder pour chaque mot du vecteur.

La bibliothèque Scikit-learn propose trois schémas différents que nous pouvons utiliser, et nous allons brièvement tester deux d'entre elles.

— Le **CountVectorizer** offre un moyen simple à la fois de marquer une collection de documents textuels et de constituer un vocabulaire de mots connus, mais aussi d'encoder de nouveaux documents à l'aide de ce vocabulaire.

1. Créer une instance de la classe CountVectorizer.
2. Appelez la fonction fit() afin d'apprendre un vocabulaire à partir d'un ou plusieurs documents.
3. Appelez la fonction transform() sur un ou plusieurs documents selon les besoins pour encoder chacun sous forme de vecteur.
4. Un vecteur codé est renvoyé avec la longueur de l'ensemble du vocabulaire et un nombre entier correspondant au nombre de fois que chaque mot est apparu dans le document.
   Ci-dessous une capture d'ecran de notre utilisation du CountVectorize.

```
Entrée [17]: from sklearn.feature_extraction.text import CountVectorizer

             count_vect = CountVectorizer(stop_words='english')

             X_train_vec = count_vect.fit_transform(data_train_df.data)
             X_test_vec = count_vect.transform(data_test_df.data)
             print(X_train_vec.shape)
             print(type(X_train_vec))

             (11314, 67822)
             <class 'scipy.sparse.csr.csr_matrix'>
```

```
Entrée [18]: X_train_vec_df = pd.DataFrame(X_train_vec.todense())
             X_train_vec_df.columns = sorted(count_vect.vocabulary_)
             X_train_vec_df.head()
```

Out[18]:

	aa	aaa	aaaaaaaaaaaa	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaauuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuugggggggggggggg	aaah	aaahh	aaahhhh	aaai	aacc	aac
0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	

5 rows × 67822 columns

— **TfidfVectorizer :** le comptage des mots est un bon point de départ, mais il est très basique. Un problème avec les comptages simples est que certains mots comme « the" apparaîtront plusieurs fois et que leur grand nombre ne sera pas très significatif dans les vecteurs codés.
Le TfidfVectorizer va tokeniser les documents, apprendre le vocabulaire et les poids de fréquence inverse des documents, et vous permettre d'encoder de nouveaux documents
. Par ailleurs, à la place du CountVectorizer, il suffit d'utiliser un TfidfTransformer pour calculer les fréquences inverses des documents et commencer à encoder les documents.
Le processus de création, d'ajustement et de transformation est le même qu'avec le CountVectorizer.

```
Entrée [19]: from sklearn.feature_extraction.text import TfidfVectorizer

             tfidfV = TfidfVectorizer(stop_words='english')

             X_train_tfidfV = tfidfV.fit_transform(data_train_df.data)
             X_test_tfidfV = tfidfV.transform(data_test_df.data)
             print(X_train_tfidfV.shape)
             print(type(X_train_tfidfV))

(11314, 67822)
<class 'scipy.sparse.csr.csr_matrix'>
```

```
Entrée [20]: X_train_tfidfV_df = pd.DataFrame(X_train_tfidfV.todense())
             X_train_tfidfV_df.columns = sorted(tfidfV.vocabulary_)
             X_train_tfidfV_df.head()
```

Out[20]:

	aa	aaa	aaaaaaaaaaaa	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaauuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuugggggggggggggg	aaah	aaahh	aaahhhh	aaai	aacc	aa
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	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	
3	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	

5 rows × 67822 columns

3 Analyse de Dataset

Il s'agit d'une approche fondamentale et souvent utilisée qui permet d'établir rapidement une relation avec les nouvelles données, il est toujours préférable d'explorer chaque ensemble de données en utilisant plusieurs techniques exploratoires et de comparer les résultats.
L'objectif de cette étape est de comprendre l'ensemble de données, d'identifier les valeurs manquantes et les points de référence éventuels en utilisant des méthodes visuelles et quantitatives pour avoir une idée de l'histoire qu'il raconte. Elle suggère les prochaines étapes logiques, questions ou domaines de recherche pour nos projet.

Cette étape est nécessaire car les données en générale sont
— Incomplètes : Manquent des valeurs d'attribut, manquent certains attributs d'importance, ou n'ont que des données agrégées
— Bruyant : contenant des erreurs ou des valeurs aberrantes
— Incohérent : Contenant des divergences dans les codes ou les noms
Parmi les étapes d'analyse et de préparation :

— **Importation de dataset :**cette étape consiste à importer la dataset qu'on va analyser, définir les données d'apprentissage et de test et explorer les données en affichant le type,description, le len .....
Voici quelques capture d'écran :

Entrée [9]:
```
for t in twenty_train.target[:10]:
        print(twenty_train.target_names[t])
```
```
rec.autos
comp.sys.mac.hardware
comp.sys.mac.hardware
comp.graphics
sci.space
talk.politics.guns
sci.med
comp.sys.ibm.pc.hardware
comp.os.ms-windows.misc
comp.sys.mac.hardware
```

Entrée [ ]:

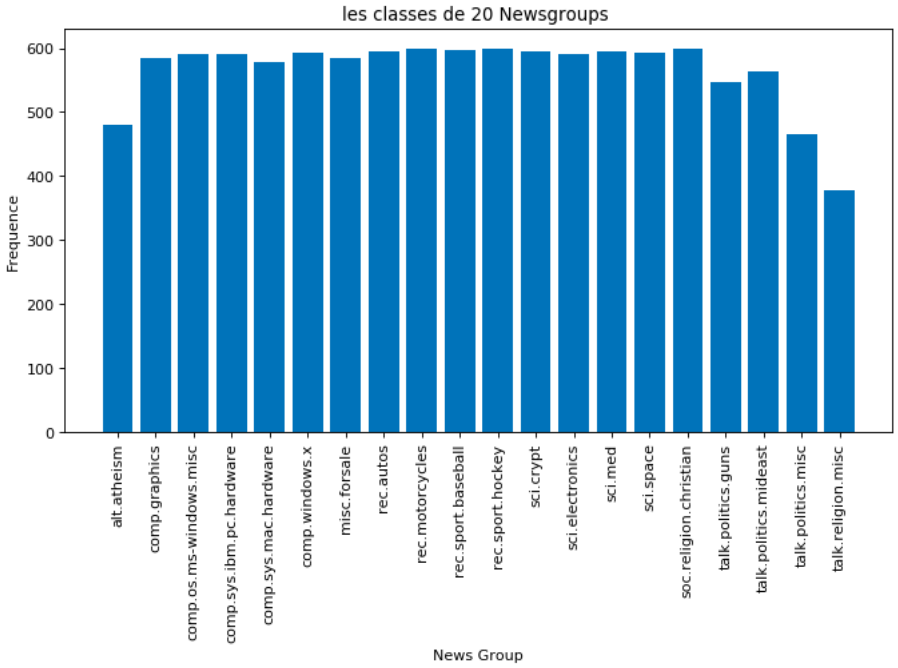
Entrée [10]:
```
frequence_targets = np.unique(twenty_train.target, return_counts=True)
frequence_targets
```
Out[10]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19]),
 array([480, 584, 591, 590, 578, 593, 585, 594, 598, 597, 600, 595, 591,
        594, 593, 599, 546, 564, 465, 377]))
```

Entrée [11]:
```
frequence_target_names = np.unique(twenty_train.target_names, return_counts=True)
list(zip(frequence_target_names))
```
Out[11]:
```
[(array(['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
         'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
         'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles',
         'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt',
         'sci.electronics', 'sci.med', 'sci.space',
         'soc.religion.christian', 'talk.politics.guns',
         'talk.politics.mideast', 'talk.politics.misc',
         'talk.religion.misc'], dtype='<U24'),),
 (array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]),)]
```

Entrée [6]:
```
print(len(twenty_train.data))
print(len(twenty_train.filenames))
print(len(twenty_test.data))
```
```
11314
11314
7532
```

Entrée [7]:
```
for i in range (0,5):
    print("\n".join(twenty_train.data[i].split("\n")[:3]))
    print(twenty_train.target_names[twenty_train.target[i]])
    i+=1
```
```
I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
rec.autos
A fair number of brave souls who upgraded their SI clock oscillator have
shared their experiences for this poll. Please send a brief message detailing
your experiences with the procedure. Top speed attained, CPU rated speed,
comp.sys.mac.hardware
well folks, my mac plus finally gave up the ghost this weekend after
starting life as a 512k way back in 1985.  sooo, i'm in the market for a
new machine a bit sooner than i intended to be...
comp.sys.mac.hardware

Do you have Weitek's address/phone number?  I'd like to get some information
about this chip.
comp.graphics
From article <C5owCB.n3p@world.std.com>, by tombaker@world.std.com (Tom A Baker):


sci.space
```

Entrée [45]:
```
fig=plt.figure(figsize=(10, 5), dpi= 80, facecolor='w', edgecolor='k')
plt.bar(targets_str,frequency)
plt.xticks(rotation=90)
plt.title('les classes de 20 Newsgroups ')
plt.xlabel('News Group')
plt.ylabel('Frequence')
plt.show()
```



Entrée [13]:
```
data_train_df = pd.DataFrame({'data': data_train.data, 'target': data_train.target})
data_train_df.head()
```

Out[13]:

	data	target
0	I was wondering if anyone out there could enli...	7
1	A fair number of brave souls who upgraded thei...	4
2	well folks, my mac plus finally gave up the gh...	4
3	\nDo you have Weitek's address/phone number? ...	1
4	From article <C5owCB.n3p@world.std.com>, by to...	14

Ainsi calculer les fréquences par rapport de classes ainsi trouver l'occurence des mots :

Entrée [10]:
```
frequence_targets = np.unique(twenty_train.target, return_counts=True)
frequence_targets
```

Out[10]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19]),
 array([480, 584, 591, 590, 578, 593, 585, 594, 598, 597, 600, 595, 591,
        594, 593, 599, 546, 564, 465, 377]))
```

Entrée [11]:
```
frequence_target_names = np.unique(twenty_train.target_names, return_counts=True)
list(zip(frequence_target_names))
```

Out[11]:
```
[(array(['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
         'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
         'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles',
         'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt',
         'sci.electronics', 'sci.med', 'sci.space',
         'soc.religion.christian', 'talk.politics.guns',
         'talk.politics.mideast', 'talk.politics.misc',
         'talk.religion.misc'], dtype='<U24'),),
 (array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]),)]
```

Entrée [15]:
```
dst = nlp.FreqDist(splt)
occu = dst['the']
print(dst)
print(len(splt))
print(occu)
print(dst.keys())
```

```
<FreqDist with 68 samples and 86 outcomes>
86
3
dict_keys(['I', 'was', 'wondering', 'if', 'anyone', 'out', 'there', 'could', 'enlighten', 'me', 'on', 'this'
'saw\nthe', 'other', 'day.', 'It', 'a', '2-door', 'sports', 'car,', 'looked', 'to', 'be', 'from', 'the', 'la
/\nearly', '70s.', 'called', 'Bricklin.', 'The', 'doors', 'were', 'really', 'small.', 'In', 'addition,\nthe'
, 'bumper', 'separate', 'rest', 'of', 'body.', 'This', 'is', '\nall', 'know.', 'If', 'can', 'tellme', 'model
', 'engine', 'specs,', 'years\nof', 'production,', 'where', 'made,', 'history,', 'or', 'whatever', 'info', '
', 'funky', 'looking', 'please', 'e-mail.'])
```

— **Conversion en lettres minuscules - Élimination des mots de passage - Suppression des caractères alphanumériques et ponctuations**

Entrée [14]:
```
import re
import string

alphanumeric = lambda x: re.sub(r"""\w*\d\w*""", ' ', x)
punc_lower = lambda x: re.sub('[%s]' % re.escape(string.punctuation), ' ', x.lower())

data_train_df['data'] = data_train_df.data.map(alphanumeric).map(punc_lower)
data_train_df.head()
```

Out[14]:

	data	target
0	i was wondering if anyone out there could enli...	7
1	a fair number of brave souls who upgraded thei...	4
2	well folks my mac plus finally gave up the gh...	4
3	\ndo you have weitek s address phone number ...	1
4	from article world std com by tombaker ...	14

— **Traitement des valeurs nulles :**

Nous devons vérifier si nous avons des valeurs nulles dans notre ensemble de données ou non, ce que nous pouvons faire en utilisant la méthode isnull(). data.isull().sum() qui renvoie le nombre de colonnes avec les valeurs nulls. On traite les valeurs nulls en supprimant les lignes ou les colonnes qui contiennent ces valeurs avec data.dropna().

— **Normalisation :**

La normalisation des données est un processus de traitement des données qui convertit la structure d'ensembles de données disparates en un format de données commun, elle traite de la transformation des

ensembles de données après que les données aient été extraites des systèmes sources et avant qu'elles ne soient chargées dans les systèmes cibles. la normalisation des données peut également être perçue comme un mécanisme de règles de transformation dans les opérations d'échange de données. On transforme nos valeurs de telle sorte que la moyenne des valeurs soit de 0 et l'écart type de 1. **Ce qui étatit fait avec TfIdfVectorizer qui est plus performant comparé àCountVectorizer.**

— **Traitement des variables catégorielles :**

Les données catégorielles sont les données qui prennent généralement un nombre limité de valeurs possibles, elles n'ont pas besoin d'être numériques, elles peuvent être de nature textuelle, on trois manières pour les traiter.

La suppression cette manière marche quand les colonnes ne contiennent pas des valeurs utiles.

Encodage des etiquettes en attribuant chaque valeur unique a un nombre entier différent.

Et aussi on a le codage à une seule valeur crée de nouvelles colonnes indiquant la présence (ou l'absence) de chaque valeur possible dans les données initiales

— **Multicollinéarité :**

La multicollinéarité existe chaque fois qu'une variable indépendante est fortement corrélée avec une ou plusieurs des autres variables indépendantes dans une relation de régression multiple. La multicollinéarité est un problème car elle nuit à la pertinence statistique d'une variable indépendante.

4 Machine Learning

Models de classification :

1. Linear Regression

```
Entrée [21]: from sklearn.model_selection import train_test_split
             from sklearn.linear_model import LinearRegression


             X_train, X_test, y_train, y_test = np.asarray(train_test_split(X_train_tfidfV, data_train_df.target, t

             reg = LinearRegression()
             reg.fit(X_train, y_train)
             y_pred = reg.predict(X_test_tfidfV)
             print('Score: ', reg.score(X_test, y_test))

          Score:  0.38239738090587294
```

2. KNN

```
Entrée [37]: text_clf = Pipeline([
                    ('vect', TfidfVectorizer(stop_words='english', sublinear_tf=True)),
                    ('clf', KNeighborsClassifier(n_neighbors=100))])  [[]]

             text_clf.fit(data_train_df.data, data_train_df.target)
             print(text_clf.score(data_test_df.data, data_test_df.target))
             print(classification_report(data_test_df.target, text_clf.predict(data_test_df.data)))

          0.05284121083377589
                        precision    recall  f1-score   support

                     0       0.22      0.02      0.03       319
                     1       0.10      0.02      0.03       389
                     2       0.06      0.26      0.09       394
                     3       0.00      0.00      0.00       392
                     4       0.04      0.04      0.04       385
                     5       0.00      0.00      0.00       395
                     6       0.00      0.00      0.00       390
                     7       0.05      0.49      0.09       396
                     8       0.15      0.01      0.01       398
                     9       0.05      0.13      0.07       397
                    10       0.04      0.01      0.01       399
                    11       0.04      0.00      0.00       396
                    12       0.03      0.00      0.00       393
                    13       0.04      0.01      0.01       396
                    14       0.04      0.01      0.01       394
                    15       0.00      0.00      0.00       398
                    16       0.12      0.00      0.01       364
                    17       0.06      0.01      0.02       376
                    18       0.14      0.00      0.01       310
                    19       0.14      0.01      0.02       251

             accuracy                           0.05      7532
            macro avg       0.07      0.05      0.02      7532
         weighted avg       0.06      0.05      0.02      7532

          CPU times: user 10.6 s, sys: 1.47 s, total: 12.1 s
          Wall time: 12.1 s
```

3. SVM

```
Entrée [5]: # Using SVM Classfier
            text_clf_svm = Pipeline([('vect', CountVectorizer(stop_words='english',tokenizer = my_tokenizer,ngram_range=(1, 2))),
                                     ('tfidf', TfidfTransformer(use_idf=True)),
                                     ('clf', svm.LinearSVC(C=10))
            ])
            text_clf_svm.fit(data_train.data, data_train.target)

            svm_predicted = text_clf_svm.predict(data_test.data)
            svm_accuracy = text_clf_svm.score(data_test.data,twenty_test.target)
            svm_f1_score = f1_score(data.target, svm_predicted,average='weighted')
            print(svm_accuracy)
            print(svm_f1_score)

         0.861789697292
         0.856518115916
```

en utilisant gridsearch qui permet d'obtenir les meilleurs paramètres pour optimiser la précision

```
Entrée [38]: %%time
pipeline = Pipeline([
        ('vect', TfidfVectorizer(stop_words='english', sublinear_tf=True)),
        ('clf', KNeighborsClassifier())])
parameters = {
       'clf__n_neighbors': (5, 10, 100, 200),
       'clf__weights': ('uniform', 'distance')
}

grid_search = GridSearchCV(pipeline, parameters)
grid_search.fit(data_train_df.data, data_train_df.target)
```

```
CPU times: user 1min 49s, sys: 9.28 s, total: 1min 58s
Wall time: 2min
```

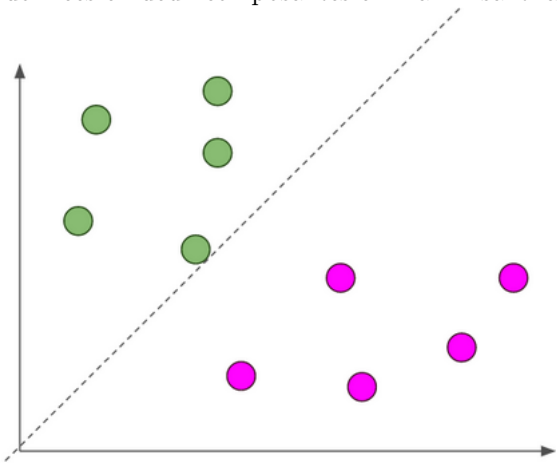
```
Out[38]: GridSearchCV(cv=None, error_score=nan,
                      estimator=Pipeline(memory=None,
                                         steps=[('vect',
                                                 TfidfVectorizer(analyzer='word',
                                                                 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='english',
                                                                 st...
                                                 KNeighborsClassifier(algorithm='auto',
                                                                      leaf_size=30,
                                                                      metric='minkowski',
                                                                      metric_params=None,
                                                                      n_jobs=None,
                                                                      n_neighbors=5, p=2,
                                                                      weights='uniform'))],
                                         verbose=False),
                      iid='deprecated', n_jobs=None,
                      param_grid={'clf__n_neighbors': (5, 10, 100, 200),
                                  'clf__weights': ('uniform', 'distance')},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                      scoring=None, verbose=0)
```

4. **Comparaison entre plusieurs modéles**

Entrée [106]:
```
models = [KNeighborsClassifier(),
          LogisticRegression(solver='lbfgs', multi_class='ovr'),
          DecisionTreeClassifier(),
          SVC(gamma='scale'),
          RandomForestClassifier(n_estimators=100),
          ExtraTreesClassifier(n_estimators=100)]

tvec = TfidfVectorizer(stop_words='english',
                       #sublinear_tf=True,
                       max_df=0.5,
                       max_features=1000)

tvec.fit(data_train['data'])
X_train = tvec.transform(data_train['data'])
X_test = tvec.transform(data_test['data'])

res = []

for model in models:
    print(model)
    print()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    print(score)
    print()
    cm = docm(y_test, y_pred, data_train.target_names)
    print(cm)
    res.append([model, score])
    print()
    print('-'*60)
    print()
```

5 Partie théorique

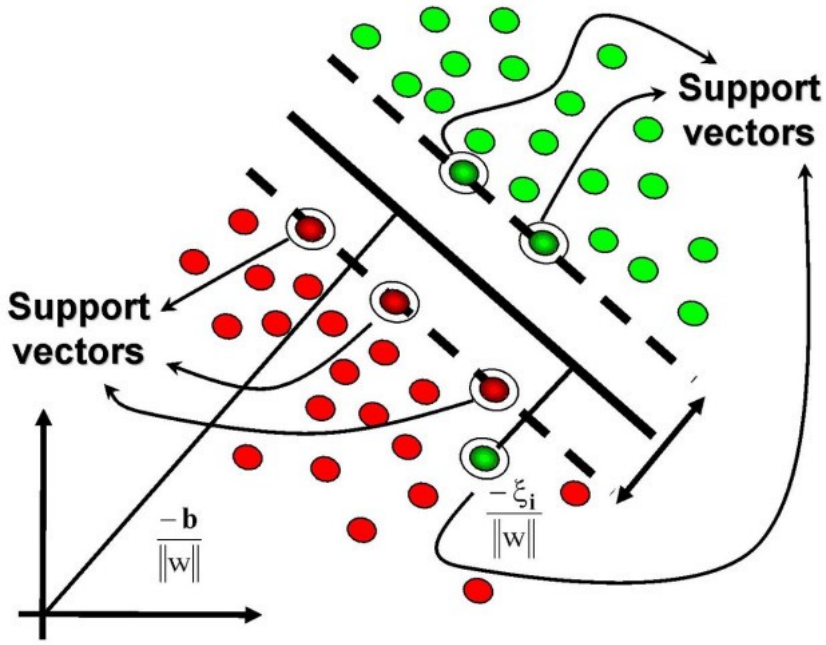
SVM : L'objectif principal du SVM est de trouver l'hyperplan optimal qui sépare linéairement les points de données en deux composantes en maximisant la marge .



L'hyperplan (ligne) est trouvé à travers la marge maximale, c'est-à-dire la distance maximale entre les points de données des deux classes.
Les SVM maximisent la marge autour de l'hyperplan de séparation leur objectif est de Trouver le meilleur hyperplan en examinant les points de données et l'hyperplan qui en résulte.
Les hyperplans proches des points de données ont des marges plus petites.
Plus un hyperplan est éloigné d'un point de données, plus sa marge sera grande
. Cela signifie que l'hyperplan optimal sera celui qui aura la plus grande marge, car une marge plus grande garantit que de légères déviations dans les points de données ne devraient pas affecter le résultat du modèle.

Comme la marge est calculée en tenant compte uniquement de points de données spécifiques, les vecteurs

de support sont des points de données qui sont plus proches de l'hyperplan et qui influencent la position et l'orientation de l'hyperplan.
En utilisant ces vecteurs de support, nous maximisons la marge du classificateur. La suppression des vecteurs de support modifiera la position de l'hyperplan. Ce sont les points qui nous aident à construire notre SVM.



L'hypothèse h est définie comme suit :

$$h(x_i) = \begin{cases} +1 & if\ w \cdot x + b \geq 0 \\ -1 & if\ w \cdot x + b < 0 \end{cases}$$

Le point situé au-dessus ou sur l'hyperplan sera classé dans la classe +1, et le point situé en dessous de l'hyperplan sera classé dans la classe -1.

Le calcul du classificateur SVM revient à minimiser une expression de la forme

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(w \cdot x_i - b)\right)\right] + \lambda\|w\|^2.$$

En effet, en choisissant une valeur suffisamment petite pour le lambda, on obtient le classificateur à marge dure pour les données d'entrée classifiables de manière linéaire.

6 Conclusion

Dans l'ensemble, nous avons vu que les classificateurs les plus performants étaient le classificateur SVM ainsi le LogisticRegression qui donne des bonnes performances, nous avons également remarqué que l'utilisation de TF-IDF Vectorizer donnait de meilleurs résultats que Count Vectorizer.
On conclu aussi que Support Vector Machine (SVM) est l'un des algorithmes d'apprentissage machine supervisé les plus puissants du monde. Contrairement à de nombreux autres algorithmes d'apprentissage machine tels que les réseaux de neurones, il n'est pas nécessaire de procéder à de nombreux réglages pour obtenir de bons résultats avec SVM