

20 Newsgroup

Text Mining

Auteur
KHEMIRI Rami
GHOUIBI Ghassen

Encadrée
Mr. Nistor Grozavu

Contents

I Introduction	2
II Text Processing and Transformation	2
II.1. Extracting features from text files	2
II.1.1 Bags of words	2
II.1.2 Tokenizing text	3
II.2. From occurrences to frequencies	3
III Analysis of the dataset	4
III.1. Dataset Shape	4
III.2. Outliers	5
III.3. Correlation between variables	6
IV Machine learning	6
IV.1. Regression Line	7
IV.2. K-NN	7
IV.3. Naive base	8
IV.4. Linear SVC	9
IV.5. Analyze result	10
V Visualization	10
VI Theoretical details	11

I Introduction

Ce rapport correspond au projet '20newsgroup', qui composé d'un ensemble de données texte on va explorer ces données en utilisant l'apprentissage automatique et en tirer des informations.

Les sections dans ce rapport seront de cet ordre :

- Text Processing and Transformation
- Analysis of the dataset
- Machine Learning
- Visualization
- Theoretical details
- Conclusion

Notre choix se porte sur le langage Python pour la réalisation de ce projet.

II Text Processing and Transformation

II.1 Extracting features from text files

Pour transformer le texte en vecteurs on aura besoin de étape pour y arrivé, en utilisant **Bag of words** et **Tokenizing text**.

Dans cette section on à suivi le tutoriel suivant.

II.1.1 Bags of words

Bag of words, est une représentation simplificatrice utilisé dans le traitement du langage naturel et la recherche d'informations.

Dans ce modèle un texte est représenté comme le sac de ses mots sans tenir compte de la grammaire et même l'ordre des mots mais en gardant la multiplicité.

On va tout simplement attribuer un entier à chaque mot dans un texte on prend un exemple¹:

John likes to watch movies. Mary likes movies too.

$BoW1 = \{ "John":1, "likes":2, "to":1, "watch":1, "movies":2, "Mary":1, "too":1 \};$

¹Cette exemple est pris de wikipedia

https://en.wikipedia.org/wiki/Bag-of-words_model

II.1.2 Tokenizing text

Dans cette partie on va essayer de **tokeniser** le texte, c'est le processus de conversion d'une séquence de caractères en une séquence de jetons (chaînes avec une signification assignée et donc identifiée).

Notre but principal c'est transformer ce texte en vecteur, le prétraitement du texte, tokenisation et filtrage des mots sont inclus dans **CountVectorizer**.

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(twenty_train.data)
```

CountVectorizer prend en charge le nombre de N-grammes de mots ou de caractères consécutifs. Une fois installé, le vectoriseur a construit un dictionnaire d'indices.

En effet, on va essayer d'appréhender la variable **X_train_counts** est une **sparse** matrice qu'on va utiliser parce qu'elle supporte l'addition, multiplication, division et matrix power.

Avec une dimension égale à 2 notre matrice est composée d'un tuple et un indice (nombre d'occurrences) d'où on voulait savoir leur attribut et ce qu'on peut extraire comme informations de cette matrice voici les attributs:

- dtype : type de matrice
- shape : shape de matrice
- ndim : dimension de matrice
- nnz : nombre de valeur stockée (incluant les zéro)

II.2 From occurrences to frequencies

Extraire le nombre d'occurrences est un bon début mais il y a un problème pour les textes plus longs auront des moyennes plus élevées que les documents les plus courts, même s'ils parlent du même sujet.

Pour éviter ces divergences il suffit de diviser le nombre d'occurrences de chaque mot dans un texte par le nombre total de mots dans le texte, ces nouvelles **features** sont appelées **tf Term Frequency**.

Une autre opération est recommander c'est réduire l'échelle des poids pour les mots qui apparaissent dans des nombreux textes du corpus et sont donc moins informatifs que ceux qui n'apparaissent que dans une petite partie du corpus.

```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

Dans le code ci-dessus, nous utilisons la méthode `fit_tranform` pour transformer notre matrice de comptage en une représentation `tf-idf` en ignorant le traitement redondant. En accédant à la variable `X_train_tfidf.data` on remarque notre matrice est normaliser maintenant au plutôt représente en `tfidf`.

TF-IDF² est une méthode de pondération souvent utilisée en recherche d'information et fouilles de textes. Cette mesure statistique permet d'évaluer l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus.

Le poids augmente proportionnellement au nombre d'occurrences du mot dans le document. Il varie également en fonction de la fréquence du mot dans le corpus.

Des variantes de la formule originale sont souvent utilisées dans des moteurs de recherche pour apprécier la pertinence d'un document en fonction des critères de recherche de l'utilisateur.

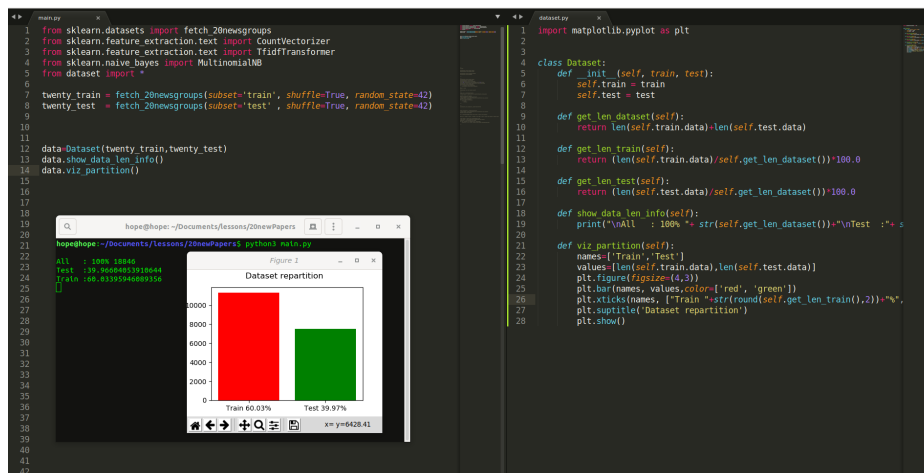
III Analysis of the dataset

Dans cette section on va essayer d'analyser notre dataset, nous allons commencer par extraire quelques informations statistiques, vérifier les valeurs manquantes, les outliers, la corrélation entre les variables et finir par présenté le but de cette analyse.

III.1 Dataset Shape

Avant de regarder dans notre dataset et visualiser sa forme il faut comprendre notre data, dans le cas on a choisi de créer un classe Dataset qui va nous permettre de faire des opérations dessus sans polluer notre code.

²Cette définition est prise à partir de wikipedia <https://fr.wikipedia.org/wiki/TF-IDF>



On va continuer sur la même classe concernant notre analyse du dataset, en effet avec un simple appel à une méthode on obtient le shape de notre dataset de training.

III.2 Outliers

Un outlier peut être :

- une valeur *aberrante* : c'est une valeur qui est manifestement fautive
- une valeur *atypique* : c'est une valeur qui "sort du lot", mais pas forcément fautive.

Dans cette partie on choisit de faire une fonction qui va nous permettre de détecter les outliers dans notre dataset on a réussi à déduire quelques mots qui seront communs dans notre dataset on trouve quasiment toujours le nombre de lignes ainsi le sujet du mail passé par l'organisation et le sujet d'où notre fonction était mise en place pour pouvoir savoir le nombre des valeurs manquantes et leur impacte sur notre dataset si jamais le nombre est très grand il faudra trouver une solution voici un exemple du résultat obtenu :

```

main.py
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from dataset import *

twenty_train = fetch_20newsgroups(subset='train', shuffle=True, random_state=42)
twenty_test = fetch_20newsgroups(subset='test', shuffle=True, random_state=42)

print(twenty_train.filesnames.shape)

data=Dataset(twenty_train, twenty_test)
data.show_data_len_info()
print("\n\nOutliers\n")
data.outlier_detect()

hope@hope: ~/Documents/lessons/20newpapers$ python3 main.py
(11314,)
All : 1000 10000
Test : 100 10000
Train : 100 10000
Total outlier found : 496
Lines not found in : 38
Subject not found in : 0
Organization not found in : 458
From not found in : 0
hope@hope: ~/Documents/lessons/20newpapers$ python3 main.py
(11314,)
All : 1000 10000
Test : 100 10000
Train : 100 10000
Outliers
Total outlier found : 496
Lines not found in : 38
Subject not found in : 0
Organization not found in : 458
From not found in : 0
hope@hope: ~/Documents/lessons/20newpapers$

dataset.py
def test_lines(self):
    numberOutlier=0
    for x in self.train.data:
        if(x.find('Lines'))==1:
            numberOutlier+=1
    return numberOutlier

def test_subject(self):
    numberOutlier=0
    for x in self.train.data:
        if(x.find('Subject'))==1:
            numberOutlier+=1
    return numberOutlier

def test_organization(self):
    numberOutlier=0
    for x in self.train.data:
        if(x.find('Organization'))==1:
            numberOutlier+=1
    return numberOutlier

def test_from(self):
    numberOutlier=0
    for x in self.train.data:
        if(x.find('From'))==1:
            numberOutlier+=1
    return numberOutlier

def outlier_detect(self):
    total=self.test_lines()+self.test_subject()+self.test_organization()
    print("Total Outlier found : ",total, "\nLines not found in : ",self.te

```

On remarque dans la figure ci-dessus que la totalité de nos outliers sont 496 contre 11314 initial qui ne représente que 4.38% de notre dataset

III.3 Correlation between variables

Les variables d'une dataset peuvent être liées pour plusieurs raisons par exemple une variable peut dépendre d'une autre ou il existe une association même deux variables dépend d'un troisième inconnu.

Une corrélation positive c'est à dire les deux variables bouge dans la même direction, quand c'est négative c'est à dire dans la direction inverse ou peut aussi être neutre dans ce cas il y a de relation entre ces deux variables.

```

from sklearn.metrics import matthews_corrcoef
def correlation(self, v1, v2):
    print(matthews_corrcoef(v1, v2))

```

On a trouve une corrélation positive entre le texte et la catégorie.

IV Machine learning

Dans cette partie on va étudier de plus près quelques modèles de machine learning, mais avant de tester nos modèles il faudra détaillé notre but.

En effet on remarque bien que notre dataset est composé de texte, et chaque texte est représenté par un sujets dans notre cas c'est des catégories:

```

def get_categories(self):
    print(self.train.target_names)

```

On remarque qu'on 20 catégories, on va essayer d'appliquer des modèles sur le texte sans prendre en compte quelque paramètres (en éliminant quelques informations) pour pouvoir prédire notre catégorie donc notre target sera le sujet du texte.

On va voir plus en détails avec les modèles choisis qu'on aura la chance de pouvoir visualiser quelques résultats.

IV.1 Regression Line

On parle de modèles linéaire ou modèle de régression linéaire quand on cherche à établir une relation variable dite expliquée et une ou plusieurs variables dites explicatives.

Le modèle³ de régression linéaire est souvent estimé par la méthode des moindres carrés mais il existe aussi de nombreuses autres méthodes pour estimer ce modèle. On peut par exemple estimer le modèle par maximum de vraisemblance ou encore par inférence bayésienne.

Dans cette sous section on va essayer de prédire quelques variables, on a choisi de travailler sur la target 'Lines' qui est représentée dans nos outliers au par avant.

IV.2 K-NN

Notre choix se porte sur K-NN, la méthode des k plus proches voisins est une méthode d'apprentissage supervisé.

Pour⁴ estimer la sortie associée à une nouvelle entrée x, la méthode des k plus proches voisins consiste à prendre en compte (de façon identique) les k échantillons d'apprentissage dont l'entrée est la plus proche de la nouvelle entrée x, selon une distance à définir.

³prise de wikipedia https://fr.wikipedia.org/wiki/R%C3%A9gression_lin%C3%A9aire

⁴Prise de wikipedia https://fr.wikipedia.org/wiki/M%C3%A9thode_des_k_plus_proches_voisins


```
def knn(self, test, data, target, categories):
    print("-----KNN-----")
    count_vect = CountVectorizer()
    X_train_counts = count_vect.fit_transform(data)
    tfidf_transformer = TfidfTransformer()
    X_train_tfidf = tfidf_transformer.fit_transform(
        X_train_counts)

    clf = KNeighborsClassifier(n_neighbors=7).fit(X_train_tfidf,
        target)
    X_new_counts = count_vect.transform(test)
    X_new_tfidf = tfidf_transformer.transform(X_new_counts)
    predicted = clf.predict(X_new_tfidf)
    print("Predicted categories is :", categories[predicted[0]], "
        with score  \%", round(clf.score
        (X_train_tfidf, target)*100.0, 2
        ))
```

Une représentation graphique sera attribuer dans la section Visualiza-
tion.

IV.3 Naive base

notre choix se porte sur Naive base vu que ce dernier est présent dans notre
tutoriel, c'est un type de classification probabilist basé sur le théorème de
Bayes avec une forte indépendance des phypothèses.

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)} \quad (1)$$

où $P(A/B)$ désigne la probabilité conditionnelle de A sachant B.

```
def naive_bayes(self, test, data, target, categories):

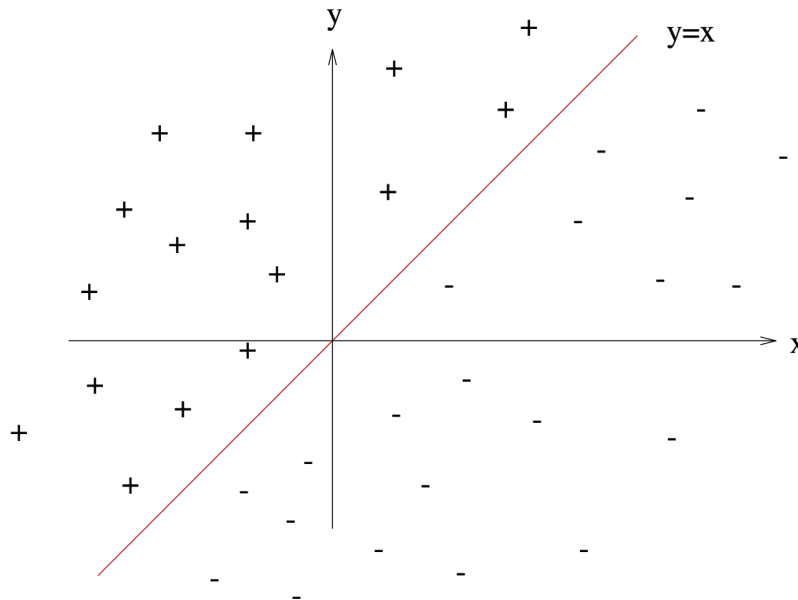
    print("-----Naive-Bayes-----")
    count_vect = CountVectorizer()
    X_train_counts = count_vect.fit_transform(data)
    tfidf_transformer = TfidfTransformer()
    X_train_tfidf = tfidf_transformer.fit_transform(
        X_train_counts)

    clf = MultinomialNB().fit(X_train_tfidf, target)
    X_new_counts = count_vect.transform(test)
    X_new_tfidf = tfidf_transformer.transform(X_new_counts)
    predicted = clf.predict(X_new_tfidf)
    print("Predicted categories is :", categories[predicted[0]],
          with score  \%", round(clf.score
                                (X_train_tfidf, target)*100.0, 2
                                ))
```

Une représentation graphique sera attribuer dans la section Visualiza-tion.

IV.4 Linear SVC

SVM⁵ sont un ensemble de technique d'apprentissage supervisé destinées à résoudre des problèmes de discrimination et de régression. Les SVM sont une généralisation des classifieurs linéaires.



⁵prise de wikipedia https://fr.wikipedia.org/wiki/Machine_%C3%A0_vecteurs_de_support

Exemple d'un problème de discrimination à deux classes, avec une séparatrice linéaire : la droite d'équation $y = x$. Le problème est linéairement séparable.

IV.5 Analyze result

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from dataset import *
from textprocessing import *
from model import *
from sklearn.pipeline import Pipeline

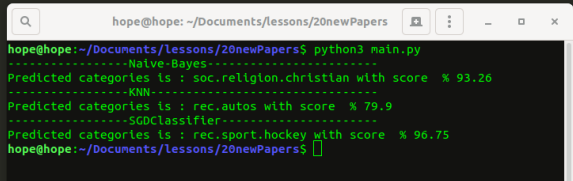
twenty_train = fetch_20newsgroups(subset='train', shuffle=True, random_state=42)
twenty_test = fetch_20newsgroups(subset='test', shuffle=True, random_state=42)

data=Dataset(twenty_train, twenty_test)

tokenizer=TokenizeGroup((data.get_data_train()))

model=Models()

model.naive_bayes(["God is love"], twenty_train.data, twenty_train.target, twenty_train.target_names)
model.knn(["I want to buy a new car"], twenty_train.data, twenty_train.target, twenty_train.target_names)
model.sgdc(["We always win in games"], twenty_train.data, twenty_train.target, twenty_train.target_names)
```



```
hope@hope: ~/Documents/lessons/20newPapers$ python3 main.py
-----Naive-Bayes-----
Predicted categories is : soc.religion.christian with score % 93.26
-----KNN-----
Predicted categories is : rec.autos with score % 79.9
-----SGDClassifier-----
Predicted categories is : rec.sport.hockey with score % 96.75
hope@hope:~/Documents/lessons/20newPapers$
```

La figure ci-dessus montre l'utilisation des modèles sur une phrase pour prédire la catégories avec un score pour chaque modèle.

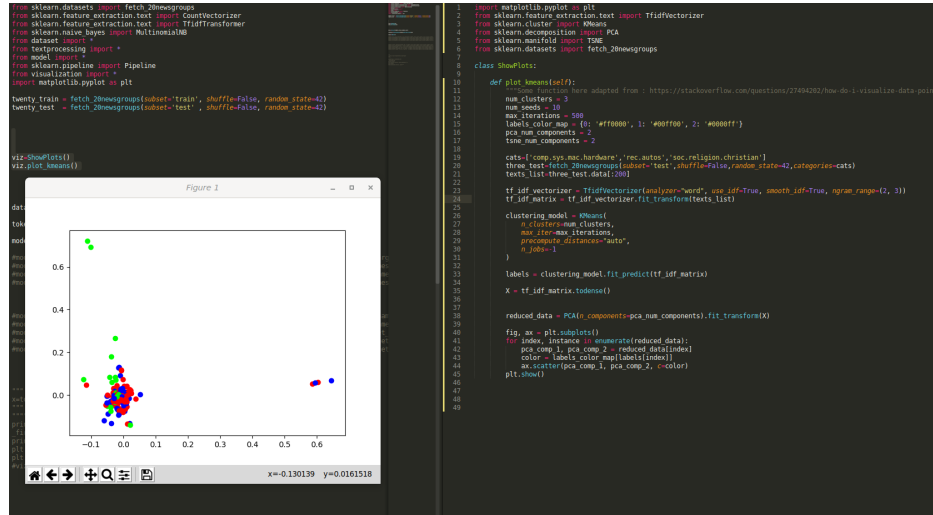
V Visualization

Dans cette partie on a choisi de visualiser les résultats de KMeans, tout d'abord il faudrait faire quelques manipulation sur notre dataset, on commence par choisir 3 catégories :

- 'comp.sys.mac.hardware'
- 'rec.autos'
- 'soc.religion.christian'

Pour des raisons de ressources on va prendre 200 échantillons, le nombre de nos clusters sera trois vu qu'on a trois catégories, ensuite on va transformer ce texte en `tfidf` on appliquer notre algorithme Kmeans voici les paramètres choisis:

```
clustering_model = KMeans(
    n_clusters=3,
    max_iter=500,
    precompute_distances="auto",
    n_jobs=-1
)
```



Voici un visualisation du résultat obtenu.

VI Theoretical details

Dans cette partie on va mettre en compétition nos modèles utilisés durant ce rapport ou inclus dans le code.

Tout d'abord on va découper notre data pour travailler sur une partie on va prendre 100 texte pour tester et voici le résultat obtenu :

Model	Score
Naive Bayes	0.932
K-NN	0.799
SGDClassifier	0.967
SVM	0.999

On remarque dans le tableau ci-dessus que notre modèle SVM obtient le meilleur score du coup on va regarder de plus prêt les paramètres et sa particularité.

```

clf = LinearSVC(C=1,
                class_weight=None,
                dual=True,
                fit_intercept=True,
                intercept_scaling=1,
                loss='squared_hinge',
                max_iter=1000,
                multi_class='ovr',
                penalty='l2',
                random_state=None,
                tol=0.01,
                verbose=0)

```

- C: Paramètre de régularisation. La force de la régularisation est inversement proportionnelle à C. Doit être strictement positive.
- class_weight :
- dual : Contrôle la génération de nombres pseudo aléatoires pour mélanger les données à double coordonnée.
- fit_intercept : Indique s'il faut calculer l'ordonnée à l'origine pour ce modèle.
- loss : Spécifie la fonction de perte. «hinge» est la perte SVM standard (utilisée par exemple par la classe SVC) tandis que «squared_hinge» est le carré de la perte.
- max_iter : The maximum number of iterations to be run.
- multi_class : Détermine la stratégie multi-classes si y contient plus de deux classes. "ovr" forme n-classes classificateurs, tandis que "crammer_singer" optimise un objectif commun sur toutes les classes.
- penalty : Spécifie la norme utilisée dans la pénalisation. La pénalité «l2» est la norme utilisée dans SVC. Le «l1» conduit à des vecteurs coef_.
- random_state : Contrôle la génération de nombres pseudo aléatoires pour mélanger les données.
- tol : Tolérance pour les critères d'arrêt.
- verbose : Activer la sortie détaillée.