Rapport de Projet

Komlan Jean-Marie DANTODJI

Data Analysis et Text Mining

1- Text Processing and Transformation
Supposons on a la liste de document suivant à transformer :

```
In [45]: data = ['This movie is very scary and long','This movie is not scary and is slow',
         'This movie is spooky and good']
```

Il s'agit de trouver un vocabulaire et de trouver une fréquence d'apparition de chaque mot du vocabulaire dans chaque document.

```
Vocabulaire:
 ['and', 'good', 'is', 'long', 'movie', 'not', 'scary', 'slow', 'spooky', 'this', 'very']
```

Etape 1:
Dans chaque document déterminer le nombre d'occurrence de chaque élément du vocabulaire.
Ainsi on se retrouve avec la matrice après passage de la fonction :
def CountVectorizerOwn(all_documents)

```
In [46]: data = CountVectorizerOwn(data)
         data
```

Out[58]:

	and	good	is	long	movie	not	scary	slow	spooky	this	very
0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0
1	1.0	0.0	2.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0
2	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0

Etape 2 : Réduction de discrimination entre les mots d'un document
Les mots fréquemment employé dans un document aura plus de chance d'être représenté que les autres. Pour cela il faut diviser chaque occurrence par le nombre total de mots dans le dossier.

```
In [47]: data = TfidfTransformerOwn(data)
         data
```

Réduction de discrimination au sein du même document :

$$Tf\_d\_un\_mot = \frac{nombre\_occurence\_d\_un\_mot\_dans\_un\_document}{nombre\_total\_de\_mots\_dans\_document}$$

Out[135]:

	and	good	is	long	movie	not	scary	slow	spooky	this	very
0	0.142857	0.000000	0.142857	0.142857	0.142857	0.000	0.142857	0.000	0.000000	0.142857	0.142857
1	0.125000	0.000000	0.250000	0.000000	0.125000	0.125	0.125000	0.125	0.000000	0.125000	0.000000
2	0.166667	0.166667	0.166667	0.000000	0.166667	0.000	0.000000	0.000	0.166667	0.166667	0.000000

Réduction de discrimination dans l'ensemble des documents :
Ceci permet de réduire la fréquence d'apparition des mots les plus courent comme « is », « the ».

$$Idf\_d\_un\_mot = Tf\_d\_un\_mot * \log \left(\frac{nombre\_de\_document\_total}{nombre\_de\_doc\_qui\_contiennent\_\_le\_mot}\right)$$

Out[137]:

	and	good	is	long	movie	not	scary	slow	spooky	this	very
0	0.0	0.00000	0.0	0.06816	0.0	0.00000	0.025156	0.00000	0.00000	0.0	0.06816
1	0.0	0.00000	0.0	0.00000	0.0	0.05964	0.022011	0.05964	0.00000	0.0	0.00000
2	0.0	0.07952	0.0	0.00000	0.0	0.00000	0.000000	0.00000	0.07952	0.0	0.00000

Dans le cas de la donnée de 20 Newsgroup dataset,
J'ai choisie les documents concernant quatre thèmes notamment :
["talk.politics.misc", "soc.religion.christian", "comp.os.ms-windows.misc", "rec.sport.baseball"]

- Data présentant les features après traitement avec mon algorithme :

J'ai choisie les 500 premiers documents à cause de la capacité de ram et de calcule de mon système.

Dataset with Own algorithm

In [7]:
```
data_own = TextProcessing(raw_data.data[:500], "my_own")
```

In [8]:
```
data_own = pd.DataFrame(data_own)
data_own
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	...	23973	23974	23975	23976	23977	23978	23979	23980	23981	23982
0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0
1	0.007043	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0
2	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0
3	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0
4	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.006206	0.0	0.0	0.006698	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
495	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0
496	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0
497	0.006424	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.016202	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0
498	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0
499	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0

500 rows × 23983 columns

- Data présentant les features après traitement de Scikit-Learn

Dataset Scikit-learn algorithm

In [138]:
```
data_scikit = TextProcessing(raw_data.data, "scikit-learn")
data_scikit = pd.DataFrame(data_scikit.toarray())
data_scikit
```

Out[138]:

	0	1	2	3	4	5	6	7	8	9	...	62651	62652	62653	62654	62655	62656	62657	62658	62659	62660
0	0.000000	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	0.000000	0.0	0.0
1	0.000000	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	0.000000	0.0	0.0
2	0.000000	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	0.000000	0.0	0.0
3	0.000000	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	0.000000	0.0	0.0
4	0.000000	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	0.039778	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2247	0.000000	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	0.000000	0.0	0.0
2248	0.000000	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	0.000000	0.0	0.0
2249	0.000000	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	0.000000	0.0	0.0
2250	0.125988	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	0.000000	0.0	0.0
2251	0.000000	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	0.000000	0.0	0.0

2252 rows × 62661 columns

2- Analyse de données

Caractéristiques des données :

- Données null ou non NaN

2- Analysis of the dataset

In [10]:
```
data_scikit.isnull().sum()
```

Out[10]:
```
0        0
1        0
2        0
3        0
4        0
        ..
23940    0
23941    0
23942    0
23943    0
23944    0
Length: 23945, dtype: int64
```

In [11]:
```
data_scikit.isna().sum()
```

Out[11]:
```
0        0
1        0
2        0
3        0
4        0
        ..
23940    0
23941    0
23942    0
23943    0
23944    0
Length: 23945, dtype: int64
```

- Informations statistiques :

. Show some statistics on the Data

In [12]:
```
data_scikit.describe()
```

Out[12]:

	0	1	2	3	4	5	6	7	8	9	...	23935	23936
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	...	500.000000	5.000000e+02
mean	0.003981	0.001431	0.000017	0.000056	0.000079	0.000056	0.000180	0.000108	0.000255	0.000255	...	0.000014	2.592540e-07
std	0.043405	0.011500	0.000382	0.001255	0.001761	0.001255	0.002878	0.002404	0.004038	0.004038	...	0.000320	5.797097e-06
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000e+00
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000e+00
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000e+00
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000e+00
max	0.928060	0.196861	0.008539	0.028061	0.039375	0.028061	0.052342	0.053760	0.066519	0.066519	...	0.007151	1.296270e-04

8 rows × 23945 columns

- Matrice de corrélation :
  Pour la matrice de corrélation mon PC mets beaucoup de temps à faire les calculs de la matrice 23936x23936 éléments.

3- Machine Learning

Pour l'apprentissage, j'ai testé quatre modeles :

- Linear regression
- K-Near Neaghbor
- Support Vector Machine
- Decision Tree

Modeles	Linear Regression	SVM	KNN	Decision Tree
Score(scikit-Learn data)	0	0.78	0.78	0.57
Score(Data on my Algorithm)	0	0.47	0.47	0.56

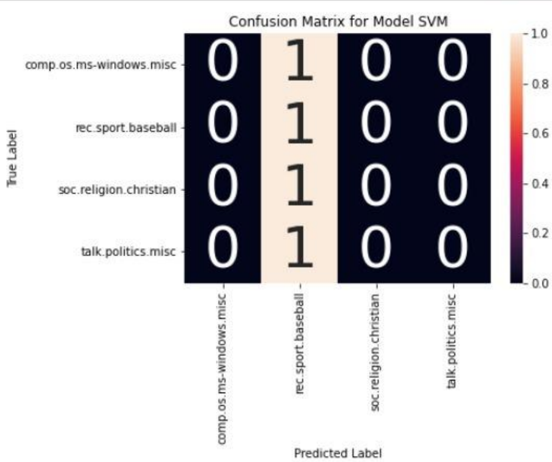
On remarque pour la prédiction de Linear regression, que un seul feature ne peut pas prédire l'autre à cause du nombre de feature très élevé (23938 features)

-Les autres scores sont aussi mauvais à cause de l'échantillon de donnée non représentatif. (500 considéré), par contre la donnée de scikit-learn étant plus représentatif, on note une amélioration de score du coté de SVM et KNN.

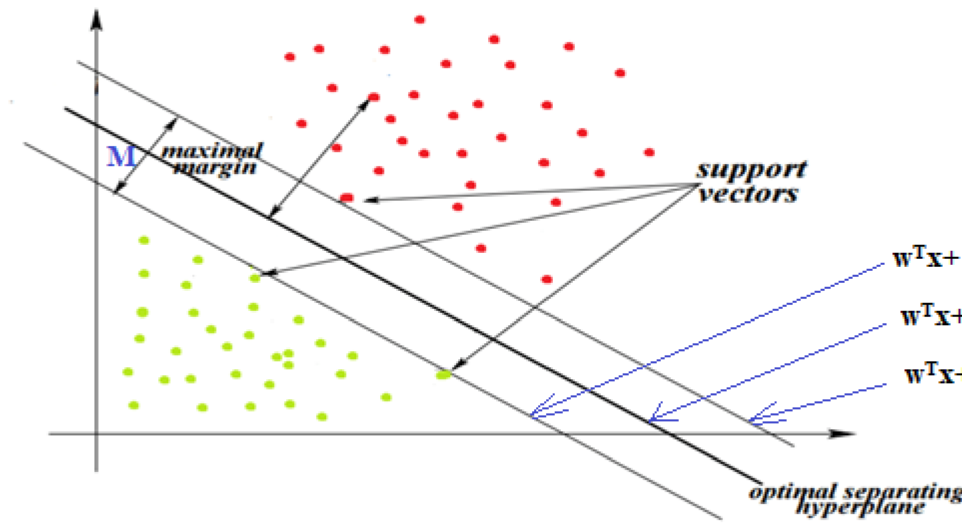
```
In [149]: X = data_scikit
          y = raw_data.target
          X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2, random_state=42)

In [150]: #KNN
          knn_clf = KNeighborsClassifier(n_neighbors=5)
          knn_clf.fit(X_train, y_train)
          predicted = knn_clf.predict(X_test)
          score = np.sum(predicted == y_test) / len(y_test)
          print("Score ", score)

          Score  0.7849223946784922

In [151]: #SVM
          from sklearn.svm import SVC
          svm_clf = SVC(gamma='auto')
          svm_clf.fit(X_train, y_train)
          predictions = svm_clf.predict(X_test)
          score = np.sum(predicted == y_test) / len(y_test)
          print("Score ", score)

          Score  0.7849223946784922

In [152]: #Decision Tree
          from sklearn.tree import DecisionTreeClassifier
          dt_clf = DecisionTreeClassifier(max_depth=2)
          dt_clf.fit(X_train, y_train)
          predicted = dt_clf.predict(X_test)
          score = np.sum(predicted == y_test) / len(y_test)
          print("Score ", score)

          Score  0.5764966740576497
```

4- Visualisation

```
          predict_train_svm = svm_clf.predict(X_train)

In [157]: def plot_confusion_matrix(cm, classes=None, title=None):
              if classes is not None:
                  sns.heatmap(cm, xticklabels=classes, yticklabels=classes, vmin=0, vmax=1, annot=True, annot_kws={'size':50})
              else:
                  sns.heatmap(cm, vmin=0, vmax=1)
              plt.title(title)
              plt.xlabel("Predicted Label")
              plt.ylabel("True Label")

In [158]: from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, r2_score
          cm_svm = confusion_matrix(y_train, predict_train_svm)
          cm_svm_norm = cm_svm/cm_svm.sum(axis=1)[:, np.newaxis]
          classes = raw_data.target_names
          plot_confusion_matrix(cm_svm_norm, classes, title="Confusion Matrix for Model SVM")
```



5- Détails théoriques : Support Vector Machine



L'objectif du SVM est de déterminer l'hyperplan qui sépare une classe des autres avec une marge maximale M.

Soit x_0 et x_1 deux vecteurs supports aux deux extrémités et

$$Soit\ l'hyperplant\ (P): w^T x + b = 0, \qquad w \in R^p, \qquad b \in R$$

$$Soit\ f(x) = w^T x + b$$

$$y_i \in \{-1,1\}, \qquad i \in \{0, n\}$$

$$\begin{cases} if\ y_i = 1 => w^T x + b \geq 1\ (top\ side) \\ if\ y_i = -1 => w^T x + b \leq -1\ (bottom\ side) \end{cases} => y_i(w^T x + b) \geq 1$$

$$M = d(x_0, P) + d(x_1, P) = \frac{|w^T x_0 + b|}{\|w\|} + \frac{|w^T x_1 + b|}{\|w\|} = \frac{|1|}{\|w\|} + \frac{|-1|}{\|w\|} = \frac{2}{\|w\|}$$

$$\begin{cases} Max\ (M) = Max(\frac{2}{\|w\|}) \\ on\ y_i(w^T x + b) \geq 1 \end{cases} => \begin{cases} Min\ (\|w\|) \\ on\ y_i f(x) \geq 1 \end{cases}$$

Minimum Cost of function

$$J = \gamma \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} Max(0, 1 - y_i(w^T x + b)) = \gamma \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} Max(0, 1 - y_i f(x))$$

$$Max(0, 1 - y_i f(x)) = \begin{cases} 0 & if\ y_i f(x) \geq 1 \\ 1 - y_i f(x) & else \end{cases}$$

Gradients:

if $y_i f(x) \geq 1$:

$$\frac{dJ_i}{dw_k} = 2\gamma w_k\ ; \qquad \frac{dJ_i}{db} = 0$$

Else:

$$\frac{dJ_i}{dw_k} = 2\gamma w_k - y_i . x_i; \qquad \frac{dJ_i}{db} = y_i$$

Update :

$$\begin{cases} w = w - \gamma \dfrac{dJ_i}{dw_k} \\ b = b - \gamma \dfrac{dJ_i}{db} \end{cases}$$