DEVOIR

Fouille de donnés

Auteur :
Mohammed El Amine
SOUADJI

Table des matières

Listings

1 Introduction

Le spam constitue plus de deux tiers du trafic mail et pose un sérieux problème pour tous les responsables de systèmes d'informations. Les nuisances apportées par les spams ne se limitent pas à l'afflux de mails indésirables ou la perte de mails légitimes. Le spam est lié à l'hameçonnage (phishing), aux virus et autres logiciels malveillants, et aux scams. La major parti des travaux qui consiste à lutter contre les spam cherchent à développer une détection de spam la plus précise possible par des techniques d'analyse de textes ou fouille de textes. La fouille de textes ou "l'extraction de connaissances" dans les textes est une spécialisation de la fouille de données et fait partie du domaine de l'intelligence artificielle. Cette technique est souvent désignée sous l'anglicisme text mining. Les outils de text-mining ont pour vocation d'automatiser la structuration des documents peu ou faiblement structurés que sont les textes écrits, comme les fichiers bureautiques de type word, les emails, les documents de présentation de type power-point... . Ainsi, à partir d'un document texte, un outil de text-mining va générer de l'information sur le contenu du document. Cette information n'était pas présente, ou explicite, dans le document sous sa forme initiale, elle va être rajoutée, et donc enrichir le document. Pour extraire du sens de documents non structurés, le text mining s'appuie sur des techniques d'analyse linguistique. Le text mining est utilisé pour classer des documents, réaliser des résumés de synthèse automatique ou encore pour assister la veille stratégique ou technologique selon des pistes de recherches prédéfinies. On peut distinguer deux étapes principales dans la classification mis en place par la fouille de textes, la première étape est le prétraitement des textes pour les rendre utilisables par les algorithmes de fouille de données et la deuxième étapes est la construction du modèle afin de classé des nouveaux textes.

2 Prétraitement

Comme dans la majorité des applications de fouille de données, une étape de prétraitement est nécessaire à l'analyse des données textuelles. Le prétraitement des données textuelles consiste à supprimer les mots vides et à la lemmatisation.

2.1 Suppression des mots vides

Un mot vide est un mot non significatif figurant dans un texte, La signification d'un mot s'évalue à partir de sa distribution (au sens statistique) dans une collection de textes. Un mot dont la distribution est uniforme sur les textes de la collection est dit « vide ». En d'autres termes, un mot qui apparaît avec une fréquence semblable dans chacun des textes de la collection n'est pas discriminant, ne permet pas de distinguer les textes les uns par rapport aux autres. pour cela il faut mieux les supprimer pour accéléré les processus d'extraction des caractéristiques et les processus de classifications.

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect =  CountVectorizer(stop_words='english')
```

Listing 1 – Suppression des mot vides

2.2 Lemmatisation

La lemmatisation est le processus de regroupement des différentes formes d'un mot afin qu'ils puissent être analysés comme un seul élément, pour déterminer le lemme d'un mot le processus peut impliquer des tâches complexes telles que la compréhension du contexte et la détermination de la nature grammaticale d'un mot dans une phrase (exigeant, par exemple, la connaissance de la grammaire d'une langue). Exemple : L'adjectif petit existe sous quatre formes : petit, petite, petits et petites. La forme canonique de tous ces mots est petit.

3 Extraction des caractéristique

Dans l'apprentissage automatique et la reconnaissance de formes, l'extraction de caractéristiques commencent à partir d'un ensemble de données initial et construit un vecteur de caractéristiques destiné à être informatif, non redondante, favorisant l'apprentissage des étapes suivantes, dans certains cas, elle conduise à une meilleure interprétations humaines.

3.1 Sac de mots "Bags of words"

Le modèle de sac de-mots est une représentation simplifier utilisé dans le traitement du langage naturel et la recherche d'informations (IR). Dans ce modèle, un texte est représenté par un ensemble (Sac) de ses mots, sans tenir compte de grammaire et même l'ordre des mots mais en gardant la multiplicité.

3.2 Tokenization des texts

Naturellement, avant de traiter n'importe qu'elle texte réel, il doit être segmenté en unités linguistiques tels que les mots, la ponctuation, des chiffres, des alpha-numériques, etc. Ce processus est appelé tokenization. En anglais, les mots sont souvent séparés les uns des autres par des espaces (espace blanc), mais pas tout les espaces blanc sont égaux. Par exemple «Los Angeles» est considéré comme un seul mot même si il contiens un espace au milieu. Nous pouvons également besoin de séparer les mots simples comme "I'am" en mots séparés "I" et "am".

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 count_vect =  CountVectorizer(stop_words='english')
```

Listing 2 – Tokenization example

3.3 Fréquence du terme TF

TF est le nombre d'occurrence d'un mot dans un document. Il a plusieurs variantes par exemple :

- Boolean : $tf(t,d) = 1$ si le mot t apparaît dans un document d, $tf(t,d) = 0$ sinon.
- Logarithmique : $tf(t,d) = 1+log(f(t,d))$ si le mot t apparaît dans un document d, $tf(t,d) = 0$ sinon.
- Double normalization 0.5 : $tf(t,d) = 0.5 + 0.5 * \frac{f(t,d)}{maxf(t,d)}$.

3.4 Fréquence inverse de document IDF

IDF est une mesure de la quantité d'informations fournit par un mot, selon que le terme est commun ou rare dans tous les documents. le TF souffre d'un problème critique : tous les termes sont considérés comme tout aussi important quand il se agit d'évaluer la pertinence d'une requête. En fait certains termes ont peu ou pas de pouvoir

de discrimination dans la détermination de la pertinence. Par exemple, une collection de documents sur l'industrie automobile est susceptible d'avoir le terme voiture dans presque chaque document.

$$idf(t, D) = log(\frac{N_D}{df(t,D)})$$

- N : nombre total de documents dans le corpus D.
- $df(t, D)$: le nombre de documents dans le corpus D où le terme t apparaît.

3.5 TF-IDF

TF-IDF, est une mesure statistique qui reflète l'importance d'un mot appartenant à un document dans une collection ou dans un corpus. Il est souvent utilisé comme un facteur de pondération en recherche d'information et de text mining. La valeur du TF-IDF augmente proportionnellement au nombre de fois qu'un mot apparaît dans le document, mais elle est compensé par la fréquence du mot dans le corpus, qui aide à tenir compte du fait que certains mots apparaissent plus fréquemment en général.

$$tf - idf(t, d, D) = tf(t, d) * idf(t, D)$$

```
from sklearn.feature_extraction.text import TfidfTransformer
tf_transformer = TfidfTransformer()
```

Listing 3 – TF-IDF example

4 Construction du modèle

La classification est une fonction de dataminig qui prédit avec précision la classe cible pour chaque instance dans les données.

4.1 Arbre de décision

Un arbre de décision est une structure d'organigramme comme dans lequel chaque noeud interne représente un «test» sur un attribut, chaque branche représente le résultat de l'essai et chaque noeud de feuille représente une étiquette de classe (décision prise après le calcul de tous les attributs). Les modèles d'arbre où la variable cible peut prendre un ensemble fini de valeurs sont appelés arbres de classification. Les chemins de la racine à la feuille représente les règles de classification.

```
from sklearn.tree import DecisionTreeClassifier
clf = = DecisionTreeClassifier()
clf.fit(X_train_tfidf, twenty_train.target)
```

Listing 4 – Arbre de décision example

4.2 SVM

la méthode svm construit un hyperplan ou un ensemble d'hyperplans dans un espace de haute ou de dimension infinie, qui peut être utilisé pour la classification ou la régression. Intuitivement, une bonne séparation est réalisée par l'hyperplan qui a la plus grande distance par rapport au point de ne importe quelle classe (on l'appelle la marge) des données d'apprentissage le plus proche, étant donné que, en général, plus grande est la marge plus faible est l'erreur de généralisation du classificateur.

```
from sklearn import svm
clf = svm.SVC(probability=True, kernel='rbf')
clf.fit(X_train_tfidf, twenty_train.target)
```

Listing 5 – Svm example

4.3 Kppv

En intelligence artificielle, la méthode des k plus proches voisins est une méthode d'apprentissage supervisé. En abrégé k-NN ou KNN, de l'anglais k-nearest neighbor. Dans ce cadre, on dispose d'une base de données d'apprentissage constituée de N couples « entrée-sortie ». Pour 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.

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier()
clf.fit(X_train_tfidf, twenty_train.target)
```

Listing 6 – kppv example

4.4 Boosting

Le boosting est un domaine de l'apprentissage automatique. C'est un principe qui regroupe de nombreux algorithmes qui s'appuient sur des ensembles de classifieurs binaires : le boosting optimise leurs performances. Le principe est issu de la combinaison de classifieurs (appelés également hypothèses). Par itérations successives, la connaissance d'un classifieur faible - weak classifier - est ajoutée au classifieur final - strong classifier.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
clf = AdaBoostClassifier(DecisionTreeClassifier())
clf.fit(X_train_tfidf, twenty_train.target)
```

Listing 7 – AdaBoostClassifier example

4.5 Naïve bayésien

Le classificateur bayésien est basé sur le théorème de Bayes avec des hypothèses d'indépendance entre les attributs. Un modèle bayésien est facile à construire, et il n'a pas besoin d'être paramétré ce qui le rend particulièrement utile pour les très grands bases de données. Malgré sa simplicité, le classificateur bayésien fait souvent très bien son travaille et est largement utilisé, car il surpasse souvent des méthodes de classification plus sophistiquées.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

- $P(c|x)$ est appelée la probabilité a posteriori de la classe c sachant l'attribut x.
- $P(c)$ est la probabilité a priori de la classe c.
- $P(x|c)$ est appelé la fonction de vraisemblance de la classe c.
- $P(x)$ est la probabilité a priori de l'attribut x.

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(X_train_tfidf, twenty_train.target)
```

Listing 8 – Naïve bayésien example

5 Validation

La validation est le processus consistant à évaluer les performances des modèles de datamining sur des données réelles. Il est important de valider les modèles de datamining en comprenant leurs qualité et caractéristiques avant de les déployer dans un environnement de production.

5.1 Cross validation

La validation croisée est une technique de validation de modèle pour évaluer la façon dont les résultats d'une analyse statistique se généralisent à un ensemble de données indépendantes. Il est principalement utilisé dans les situations où l'objectif est la classification, et l'on veut estimer la précision d'un modèle prédictif. Dans un problème de classification, un modèle est généralement construit grâce à un ensemble de données avec une classe connue (training dataset), et un ensemble de données avec une classe inconnue pour laquelle le modèle est testé (test dataset). Un tour de validation croisée consiste à diviser un échantillon de données en sous-ensembles complémentaires, effectuer l'analyse sur un sous-ensemble (appelé l'ensemble d'apprentissage), et la validation sur l'autre sous-ensemble (appelé l'ensemble de validation ou l'un ensemble de test). Pour réduire la variabilité, plusieurs cycles de validation croisée sont effectuées en utilisant des partitions différentes, et la moyenne des tours est considéré comme le résultats de validation.

6 Évaluation des performance

Pendant longtemps, le critère retenu pour évaluer les performances des systèmes de classifications a été le taux de bonne classification, c'est-à-dire le nombre d'éléments d'une base de test correctement classés. Le problème d'un tel critère est qu'il suppose que tous les erreurs ont les mêmes conséquences. Dans de nombreuses situations, certaines erreurs ont un coût plus important que d'autres, par exemple, dans notre cas, on peut ce permettre d'avoir quelque spam parmi les messages par contre on a pas le droit de filtré un message ham. Pour cela dans cette partie, je dresse un panorama des critères d'évaluation des systèmes de classification binaire.

6.1 Accuracy

Le taux de bonne classification

$$accuracy = \frac{(TP+TN)}{(TP+FP+TN+FN)}$$

6.2 Courbe Roc

Dans les statistiques, la fonction d'efficacité du récepteur (ROC), ou de la courbe ROC, est une représentation graphique qui illustre les performances d'un système de classification binaire sur la base d'une ou plusieurs de leurs caractéristiques. La courbe est créée en traçant le taux de vrais positifs contre le taux de faux positifs à différents paramètres de seuil. L'aire sous la courbe ROC ou AUC ("Area Under the ROC Curve") et est devenue une meilleure alternative que accuracy pour évaluer des classifieurs. L'AUC d'un classifieur est équivalente à la probabilité qu'un classifieur donne un meilleur rang à un élément positif par rapport à un élément négatif, tous deux choisis aléatoirement dans la base.

6.3 Precision

La précision est la proportion de "vrai" message ham parmi tout les message détecter comme étant "ham".
Nb : Cette quantité ne représente pas un taux d'exemples bien classés par rapport à une classe.

$$precision = \frac{TP}{(TP+FP)}$$

6.4 Recall

Le recall est la proportion de "vrai" message ham parmi tout les messages "ham" disponibles..

$$precision = \frac{TP}{(TP+FN)}$$

6.5 F-measure

F-measure est la moyenne harmonique de la précision et du rappel.

$$F = \frac{2*precision*recall}{(precision+recall)}$$

7 Implementation

Pour la partie implémentation au lieu de mettre chaque réponse à une question dans un programme python à part, j'ai choisi de créer un seul programme python qui peut être paramétré afin que je puisse l'utiliser pour chaque question, j'ai ajouté un script shell *run.sh* au répertoire source qui permet de lancer tous les tests.

```
$ python3 main.py --help
usage: main.py [-h] -d {sms,email} -v VALIDATION -a {nb,svm,knn,boost,
    tree}
                [-k {linear,poly,rbf,sigmoid,precomputed}] [-c {gini,
    entropy}]
                [-m MIN_SAMPLES_SPLIT]

Tp de fouille de donnee

optional arguments:
  -h, --help            show this help message and exit
  -d {sms,email}, --db {sms,email}
                        Le nom de la base de donnee
  -v VALIDATION, --validation VALIDATION
                        Validation type [
                        0: 0.3 train, 0.7 test ,
                        1: cross-validation,
                        2: Tuning hyper-parameters,
                        3: Classification Finale
                        ]
  -a {nb,svm,knn,boost,tree}, --algorithm {nb,svm,knn,boost,tree}
                        Algorithme de classification
  -k {linear,poly,rbf,sigmoid,precomputed}, --kernel {linear,poly,rbf,
    sigmoid,precomputed}
                        Le noyeau du classifieur svm
  -c {gini,entropy}, --criterion {gini,entropy}
                        Critere de l arbre de decision
  -m MIN_SAMPLES_SPLIT, --min_samples_split MIN_SAMPLES_SPLIT
                        Le nombre minimal d exemple pour continuer la
    construction de l arbre

Souadji Mohammed El Amine
```

Listing 9 – Menu d'aide de l'application

Mon Programme se compose de cinq classes :

dataset.py : cette classe représente le type d'une base de données, elle contient les données de l'ensemble d'entraînement avec leur classe et le nom de chaque instance (le nom du fichier) et aussi les données l'ensemble de test et le nom de chaque instance.

loadSmsDb.py : comme son nom l'indique, elle permet le chargement la base de données sms.

loadEmailDb.py : cette classe permet le chargement de la base de données sms.

main.py : c'est la classe principale, elle contient tous les traitements.

DenseTransformer.py : j'ai ajouté cette classe car quand j'utilise les arbres de décision dans le pipeline, il me sort une erreur d'incompatibilité de matrice.

8 Résultat Sms

Pour cette partie d'expérimentation, j'ai suivi le plan suivant :

a. Le chargement de la base de données Sms.

b. La tokenization et suppression des mots vides.

c. Calcule des fréquences TF-idf.

d. La recherche des paramètres optimaux pour les classifieurs.

e. La construction des modèles, en utilisant les quatre classifieurs et adaboost.

f. La validation des modèles, soit en utilisant la cross validation, soit les 70% des sms de la base test dans le cas ou on a construit le modèle avec 30% des sms.

g. Calcule des différentes métriques pour pouvoir comprendre les résultats et comparer les classifieurs.

un système antispam parfait est un système qui détecte tous les spams comme étant spam et tous les messages ham comme étant ham. Bien sûr, un tel système n'existe pas en pratique, donc, on aura toujours quelque erreur. Pour un système antispam, les erreurs n'ont pas les mêmes conséquences, on peut tolérer la réception de quelque spam par contre on ne peut pas accepter le filtrage des messages ham. Pour cela, j'ai choisi le rappel (recall) et l'aire sous la courbe roc comme métrique de comparaison entre les classifieurs.
Malheureusement sklearn ne nous donne pas la matrice de confusion pour la cross validation donc je ne vais pas trop interpréter les résultats de la partie cross validation.

8.1 Naïve bayésien

```
$ python3 main.py -d sms -v 2 -a nb
Tuning hyper-parameters
Best parameters set found on development set:
clf__alpha: 0.01
clf__fit_prior: False
vect__ngram_range: (1, 2)
vect__stop_words: None

$ python3 main.py -d sms -v 0 -a nb
30% train et 70% test
             precision    recall  f1-score   support

        ham       0.99      0.96      0.98      3368
       spam       0.79      0.96      0.87       534

avg / total       0.97      0.96      0.96      3902

accuracy :  0.96
roc_auc : 0.96
average_precision_score : 0.88

Matrice de Confusion
3234  134
  23  511

$ python3 main.py -d sms -v 1 -a nb

Cross validation
accuracy :  0.97
roc_auc : 0.99
average_precision : 0.98
precision : 0.84
recall :  0.95
f1 :  0.89
```

Listing 10 – Naive bayes résultat

Interprétation

On remarque que le $recall_{ham}$ et $recall_{spam}$ = 0.96, ce qui veut dire que 96% des messages ham (resp spam) sont détectés comme étant ham (resp spam). par contre on remarque aussi que la $precision_{spam} = 0.79$, ce qui veut dire que 21% des messages spam sont des messages ham.

8.2 Knn

```
$ python3 main.py -d sms -v 2 -a knn
Tuning hyper-parameters
Best parameters set found on development set:
clf__metric: 'minkowski'
clf__n_neighbors: 7
clf__weights: 'distance'
vect__ngram_range: (1, 2)
vect__stop_words: None

$ python3 main.py -d sms -v 0 -a knn

30% train et 70% test
             precision    recall  f1-score   support

        ham       0.89      1.00      0.94      3397
       spam       1.00      0.20      0.33       505

avg / total       0.91      0.90      0.86      3902

accuracy :  0.90
roc_auc : 0.60
average_precision_score : 0.65

Matrice de Confusion
3397    0
 406   99


$ python3 main.py -d sms -v 1 -a knn

Cross validation
accuracy :  0.93
roc_auc : 0.81
average_precision : 0.83
precision : 1.00
recall :  0.45
f1 :  0.62
```

Listing 11 – Knn résultat

Interprétation

On remarque que le $recall_{ham} = 1$ ce qui veut dire que tous les messages ham sont bien classés et c'est exactement ce que l'on voulait au départ, mais le $recall_{spam} = 0.20$ ce qui signifie que la majorité 80% des messages spam ne sont pas filtrés (détectés comme étant ham), ce qui réduit la $precision_{ham}$ (11% des ham sont en réalité des spam).
Ce classifieur ne peut être utilisé comme étant comme filtre pour les spam vu que l'AUC de sa courbe ROC vaut 0.6 et qu'il stop que 20% des spam.

8.3 SVM

```
$ python3 main.py -d sms -v 2 -a svm
Tuning hyper-parameters
Best parameters set found on development set:
clf__C: 100
clf__gamma: 0.001
clf__kernel: 'linear'
clf__probability: True
vect__ngram_range: (1, 2)
vect__stop_words: None

$ python3 main.py -d sms -v 0 -a svm -k linear

30% train et  70% test
  % self.indptr.dtype.name)
             precision    recall  f1-score   support

        ham       0.97      1.00      0.99      3360
       spam       0.98      0.84      0.90       542

avg / total       0.98      0.98      0.97      3902

accuracy :  0.98
roc_auc : 0.92
average_precision_score : 0.92

Matrice de Confusion
3351    9
  87  455

  $ python3 main.py -d sms -v 1 -a svm -k linear

Cross validation
accuracy :  0.98
roc_auc : 0.99
average_precision : 0.98
precision : 0.98
recall :  0.90
f1 :  0.94
```

Listing 12 – SVM résultat

Interprétation

On remarque que le $recall_{ham} = 1$ ce qui signifie que tous (ou presque tous) les messages ham sont bien détectés, et on a $recall_{spam} = 0.84$ ce qui veut dire il y a que 16% des messages spam sont détectés comme étant ham. et on a un rapport entre la précision et le rappel de $F - mesure = 0.97$ ce que je trouve très bien.
nb : cet algorithme sans un bon paramétrage, il classe tous les messages comme étant ham.

8.4 Arbre de décision

```
$ python3 main.py -d sms -v 2 -a tree
Tuning hyper-parameters
Best parameters set found on development set:
clf__criterion: 'gini'
clf__min_samples_leaf: 1
clf__min_samples_split: 3
vect__ngram_range: (1, 1)
vect__stop_words: 'english'

$ python3 main.py -d sms -v 0 -a tree -c gini -m 3

30% train et 70% test
             precision    recall  f1-score   support

        ham       0.97      0.98      0.98      3385
       spam       0.87      0.81      0.84       517

avg / total       0.96      0.96      0.96      3902

accuracy :  0.96
roc_auc : 0.89
average_precision_score : 0.85

Matrice de Confusion
3325   60
 100  417

$ python3 main.py -d sms -v 1 -a tree -c gini -m 3

Cross validation
accuracy :  0.97
roc_auc : 0.91
average_precision : 0.88
precision : 0.91
recall :  0.83
f1 :  0.86
```

Listing 13 – Arbre décision avec Gini résultat

```
$ python3 main.py -d sms -v 0 -a tree -c entropy -m 3

30% train et 70% test
              precision    recall  f1-score   support

         ham       0.98      0.97      0.97      3401
        spam       0.80      0.84      0.82       501

avg / total       0.95      0.95      0.95      3902

accuracy :  0.95
roc_auc : 0.90
average_precision_score : 0.83

Matrice de Confusion
3293  108
  81  420
$ python3 main.py -d sms -v 1 -a tree -c entropy -m 3

Cross validation
accuracy :  0.96
roc_auc : 0.89
average_precision : 0.87
precision : 0.91
recall :  0.80
f1 :  0.85
```

Listing 14 – Arbre décision avec Entropy résultat

Interprétation

On remarque qu'on a des bonnes valeurs de $recall$ et de $prcision$ avec une légère amélioration en faveur de l'index de gini. par contre on remarque aussi que l'arbre avec l'index de gini classe bien les messages ham par rapport a l'arbre avec l'entropie ($recall_{ham(gini)} = 0.98 > recall_{ham(entropy)} = 0.97$), ce qui le rend performant que l'autre pour le filtrage de spam même si le deuxième arbre filtre bien les spam que le premier($recall_{spam(gini)} = 0.81 < recall_{spam(entropy)} = 0.84$).

8.5 Boosting

```
$ python3 main.py -d sms -v 0 -a boost -c gini -m 3

30% train et 70% test
             precision    recall  f1-score   support

        ham       0.97      0.98      0.97      3377
       spam       0.84      0.81      0.82       525

avg / total       0.95      0.95      0.95      3902

accuracy :  0.95
roc_auc : 0.89
average_precision_score : 0.83

Matrice de Confusion
3294   83
 101  424

$ python3 main.py -d sms -v 1 -a boost -c gini -m 3

Cross validation
accuracy :  0.96
roc_auc : 0.91
average_precision : 0.87
precision : 0.89
recall :  0.83
f1 :  0.86
```

Listing 15 – Adaboost et Arbre décision avec Gini

Interprétation

je ne sais pas comment interpréter ce résultat. Normalement les fonctions de boosting améliorent la classification, mais dans cette expérimentation, on remarque qu'au lieu de l'améliorer, il a dégradé donc il faut mieux utiliser l'arbre tous seul que l'arbre avec l'algorithme adaboost.

8.6 Conclusion

Pour conclure, je propose l'algorithme svm avec le noyau "linear" pour la classification des sms vu que c'est lui qui classe très bien les messages ham et qui offre un bon rapport entre le recall et la précision, par contre dans mon étude je n'ai pas pris en considération le temps, car si l'on ajoute le temps de la construction du modèle l'algorithme naïve bayes sera meilleur que svm, puisqu'il est beaucoup plus rapide que le SVM et offre aussi des bons résultats de classification.

9 Résultat Email

Pour cette partie d'expérimentation, j'ai préféré recommencer toutes les étapes précédentes afin de trouver le bon classifieur au lieu de me basé sur la conclusion précédente, car vu que la base a changé, les comportements des classifieurs changeront aussi. j'ai suivi les étapes suivantes :

a. Le chargement de la base de données Email.

b. La tokenization et suppression des mots vides.

c. Calcule des fréquences TF-idf.

d. La recherche des paramètres optimaux pour les classifieurs.

e. La construction des modèles, en utilisant les quatre classifieurs et adaboost.

f. La validation des modèles, soit en utilisant la cross validation, soit les 30% des Email de la base test dans le cas ou on a construit le modèle avec 70% des Email.

g. Calcule des différentes métriques pour pouvoir comprendre les résultats et comparer les classifieurs.

h. Choix du meilleur classifieur et l'utiliser avec adaboost.

i. Choisir entre le meilleur classifieur et adaboost.

j. Construction du modèle avec toute la base d'entrainement.

k. Classification de la base de test.

Nb : J'ai eu beaucoup de problèmes avec cette basse à cause de la longueur des messages, ce qui nous donnes des instances avec beaucoup d'attributs et cause une erreur de type **memory error**.

9.1 Naïve bayésien

```
Tuning hyper-parameters
Best parameters set found on development set:
clf__alpha: 0.001
clf__fit_prior: False
vect__ngram_range: (1, 2)
vect__stop_words: None

$ python3 main.py -d email -v 0 -a nb

70% train et 30% test
             precision    recall  f1-score   support

        ham       0.98      0.99      0.98       410
       spam       0.94      0.92      0.93       106

avg / total       0.97      0.97      0.97       516

accuracy :  0.97
roc_auc : 0.95
average_precision_score : 0.94

Matrice de Confusion
404   6
  9  97

$ python3 main.py -d email -v 1 -a nb

Cross validation
accuracy :  0.90
roc_auc : 0.94
average_precision : 0.89
precision : 0.86
recall :  0.93
f1 :  0.86
```

Listing 16 – Naive bayes résultat

Interprétation

Comme pour la base Sms, on remarque que l'on a un très bon rappel pour les messages ham et un bon rappel pour les messages spam. donc, on filtre bien les spam et on se trompe rarement dans les messages ham.

9.2 Knn

```
Tuning hyper-parameters
Best parameters set found on development set:
clf__metric: 'minkowski'
clf__n_neighbors: 3
clf__weights: 'uniform'
vect__ngram_range: (1, 1)
vect__stop_words: None
$ python3 main.py -d email -v 0 -a knn

70% train et 30% test
             precision    recall  f1-score   support

        ham       0.94      0.99      0.97       409
       spam       0.94      0.78      0.85       107

avg / total       0.94      0.94      0.94       516

accuracy :  0.94
roc_auc : 0.88
average_precision_score : 0.88

Matrice de Confusion
404   5
 24  83

$ python3 main.py -d email -v 1 -a knn

Cross validation
accuracy :  0.87
roc_auc : 0.89
average_precision : 0.87
precision : 0.81
recall :  0.84
f1 :  0.79
```

Listing 17 – knn résultat

Interprétation

On obtient presque le même résultat qu'avec la base sms toujours un bon rappel pour les ham et un très mauvais rappel pour les spam, ce qui le rend inutilisable.

9.3 SVM

```
Tuning hyper-parameters
Best parameters set found on development set:
clf__C: 1000
clf__gamma: 0.001
clf__kernel: 'rbf'
clf__probability: True
vect__ngram_range: (1, 1)
vect__stop_words: None

$ python3 main.py -d email -v 0 -a svm -k rbf

70% train et 30% test
             precision    recall  f1-score   support

        ham       0.97      0.99      0.98       415
       spam       0.95      0.86      0.90       101

avg / total       0.96      0.96      0.96       516

accuracy :  0.96
roc_auc : 0.92
average_precision_score : 0.92

Matrice de Confusion
410   5
 14  87

python3 main.py -d email -v 1 -a svm -k rbf

Cross validation
accuracy :  0.89
roc_auc : 0.93
average_precision : 0.89
precision : 0.85
recall :  0.94
f1 :  0.86
```

Listing 18 – SVM résultat

Interprétation

On remarque que le $recall_{ham} = 0.99$ ce qui signifie presque tous les messages ham sont bien détectés, et on a $recall_{spam} = 0.86$ ce qui veut dire il y a que 14% des messages spam sont détectés comme étant ham. et on a un rapport entre la precision et le rappel de $F - mesure = 0.96$ ce que je trouve très bien.

9.4 Arbre de décision

```
Tuning hyper-parameters
Best parameters set found on development set:
clf__criterion: 'gini'
clf__min_samples_leaf: 2
clf__min_samples_split: 6
vect__ngram_range: (1, 1)
vect__stop_words: None
$ python3 main.py -d email -v 0 -a tree -c gini -m 6

70% train et 30% test
             precision    recall  f1-score   support

        ham       0.98      0.95      0.97       412
       spam       0.82      0.94      0.87       104

avg / total       0.95      0.95      0.95       516

accuracy :  0.95
roc_auc : 0.94
average_precision_score : 0.89

Matrice de Confusion
390  22
  6  98

$ python3 main.py -d email -v 1 -a tree -c gini -m 6

Cross validation
accuracy :  0.86
roc_auc : 0.88
average_precision : 0.87
precision : 0.79
recall :  0.88
f1 :  0.81
```

Listing 19 – Arbre décision résultat

Interprétation

On obtient des résultats acceptables avec l'arbre de décision mais les algorithmes svm et naive bayes restes les meilleurs.

9.5 Boosting

Cette fois-ci, je n'ai pas utilisé l'arbre de décision, mais j'ai choisi le meilleur algorithme parmi les précédents tests et je l'ai ajouté à AdaBoost, au départ je voulais tester naive bayes et Svm mais, malheureusement, je n'ai pas l'utiliser avec Svm, car j'obtenais une erreur python "MemoryError". Donc, je les testais qu'avec naïve bayes.

```
$ python3 main.py -d email -v 0 -a boost

70% train et  30% test
              precision    recall  f1-score   support

         ham       0.99      0.99      0.99       421
        spam       0.94      0.96      0.95        95

avg / total       0.98      0.98      0.98       516

accuracy :  0.98
roc_auc : 0.97
average_precision_score : 0.95

Matrice de Confusion
415   6
  4  91
$ python3 main.py -d email -v 1 -a boost

Cross validation
accuracy :  0.90
roc_auc : 0.93
average_precision : 0.86
precision : 0.85
recall :  0.95
f1 :  0.87
```

Listing 20 – Adaboost et Naive bayesien

Interprétation

On remarque que cette fois-ci l'algorithme AdaBoost a amélioré le résultat de naïve bayes et nous donne une bonne precision et un bon rappel.

9.6 Conclusion

On remarque que l'algorithme AdaBoost avec naïve bayes offre le meilleur résultat par rapport autres algorithmes, ce qui m'a poussé à le choisir comme classifieur pour classer ma base email de test le résultat est dans le fichier *resultats_Test_Souadji.txt*.