Rapport de projet de fin de semestre

Module IARN

USTHB - Master 1 SII

Quadrinôme:

Merrouche Amira

Djeghali Ikram

Moussi Samy

Ait oubelli Syphax

Table des matières

Introduction

L’apprentissage automatique (Machine learning) est un champs d'étude de l’intelligence artificielle qui permet de concevoir des systèmes intelligents capable de resoudre des problemes de la vie de tous les jours dans différents domaines comme la détection de tumeurs, l’analyse des données, la segmentation d’image, les systèmes de recommandation ou toute forme de classification et/ou optimisation.

Il est souvent divisé en deux types majeures soit l’apprentissage supervisé et non supervisé, le premier est très utile pour les problèmes de classification binaire tandis que l'autre excelle aux problèmes de clustering. Afin d’effectuer du machine learning supervise par exemple dans le cas de la classification, on commence par un Dataset qui contient des données étiquetées qu’on divise en deux ensembles d'entraînement et de tests, on entraîne notre modèle sur le premier ensemble puis on l'évalue à partir du deuxième ensemble en utilisant des métriques d'évaluations.

L'apprentissage profond est une technique d’apprentissage automatique qui s’appuie sur les réseaux de neurones pour imiter la manière dont fonctionne le cerveau humain .

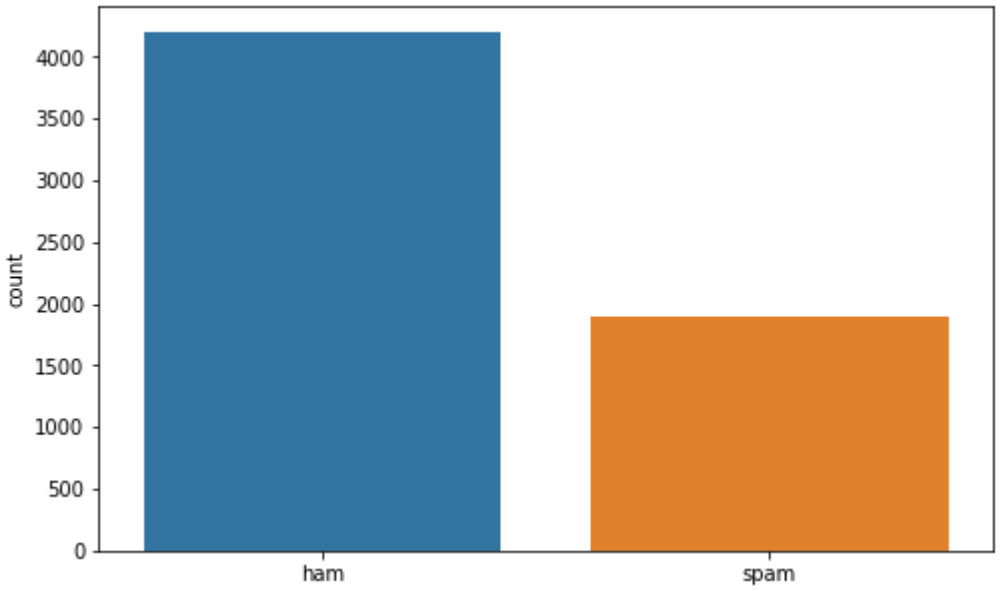
Dans le cadre de notre projet de fin de semestre pour le module d’Apprentissage automatique et réseaux de neurones il sera question d'implémenter des techniques et algorithmes de deep learning et de machine learning afin de résoudre un problème de classification d’email (Spam/Non-Spam) tout en comparant les résultats obtenus à partir des différentes approches.

Importation et Visualisation des données

Base d'apprentissage (Dataset)

Notre base d'apprentissage *SPAMASSASSIN PUBLIC CORPUS* disponible via le lien, contient 6098 mails au total, soit 1897 Spam et 4201 Ham.
On remarque que les classes ne sont pas parfaitement équilibrées, pour cela nous allons explorer deux approches, la première utilisant les données du dataset directement et la deuxième en implémentant une technique de suréchantillonage (Over-sampling) nommée SMOTE (Synthetic Minority Over-sampling Technique).

Voici un aperçu de notre dataset:



Librairies utilisées

Afin de télécharger et d'importer nos données, nous avons choisi de travailler avec les bibliothèques suivantes :

Bibliothèque	Utilisation
os	Pour la manipulation des paths
email	Pour la manipulation des objets emails
urllib	Pour la manipulation des URLs permettants de télécharger la base d'apprentissage
tarfile	Pour l'extraction/décompression des fichiers contenants les emails

numpy	Pour la manipulation des arrays, stockage des données dans des tableaux de type array
matplotlib, seaborn	Pour la visualisation des données

Preprocessing

Étant donné que les modèles de classification ne peuvent pas traiter des emails/textes directement, un prétraitement est nécessaire afin d'obtenir de bons résultats et surtout de sélectionner les caractéristiques (features) sur lesquelles va se baser notre classification.

Librairies utilisées

Bibliothèque	Utilisation
re	Pour les opérations d'expressions régulières
html	Pour le traitement des balises html
nltk	Pour l'extraction des caractéristiques (Vocabulaire)
string	Pour traiter les ponctuations

Fonctions de prétraitement

- def html_to_text(html): Convertir des balises HTML en texte (string).
- def message_to_text(email): Convertir des mails en texte (string).
- def minuscule(texte): Convertir le texte donne en minuscule.
- def replace_html(texte): Remplacer toutes les balises HTML par un espace vide..
- def replace_urls(texte): Remplacer tous les URLs par httpaddr.
- def replace_emails(texte): Replacer toutes les adresses emails par emailaddr.
- def replace_numbers(texte): Remplacer tous les nombres par nombre.
- def replace_dollars(texte): Remplacer les $ par dollar.
- def radicalisation(texte): Remplacer tous les mots par leur radical.
- def remove_non_mots(texte): Suppression des non mots (ponctuation, saut de ligne, tabulation..).
- def Preprocess(X) : Fonction regroupant toutes les fonctions précédentes et qui va appliquer le prétraitement pour l'ensemble de tous nos emails.

Voici donc un exemple complet avec toutes les étapes de prétraitement :

Avant le preprocessing :

```
 Anyone $$$$$ knows how much $ it costs to host a web portal ?
 Well, it depends on how many visitors youre expecting.
 This can be anywhere from less than 10 bucks a month to a couple of
$100.
 You should checkout http://www.rackspace.com/ or perhaps Amazon EC2
if youre running something big..
 To unsubscribe yourself from this mailing list, send an email to:
groupname-unsubscribe@egroups.com
```

Rendre minuscule :

```
 anyone $$$$$ knows how much $ it costs to host a web portal ?
 well, it depends on how many visitors youre expecting.
 this can be anywhere from less than 10 bucks a month to a couple of
$100.
 you should checkout http://www.rackspace.com/ or perhaps amazon ec2
if youre running something big..
 to unsubscribe yourself from this mailing list, send an email to:
groupname-unsubscribe@egroups.com
```

Remplacer les nombres :

```
 anyone $$$$$ knows how much $ it costs to host a web portal ?
 well, it depends on how many visitors youre expecting.
 this can be anywhere from less than nombre bucks a month to a couple
of $nombre.
 you should checkout http://www.rackspace.com/ or perhaps amazon
ecnombre if youre running something big..
 to unsubscribe yourself from this mailing list, send an email to:
groupname-unsubscribe@egroups.com
```

Remplacer les dollars :

```
 anyone dollardollardollardollardollar knows how much dollar it costs
to host a web portal ?
 well, it depends on how many visitors youre expecting.
 this can be anywhere from less than nombre bucks a month to a couple
of dollarnombre.
 you should checkout http://www.rackspace.com/ or perhaps amazon
ecnombre if youre running something big..
 to unsubscribe yourself from this mailing list, send an email to:
groupname-unsubscribe@egroups.com
```

Remplacer les URLs :

```
 anyone dollardollardollardollardollar knows how much dollar it costs
to host a web portal ?
 well, it depends on how many visitors youre expecting.
 this can be anywhere from less than nombre bucks a month to a couple
of dollarnombre.
 you should checkout httpaddr or perhaps amazon ecnombre if youre
running something big..
 to unsubscribe yourself from this mailing list, send an email to:
groupname-unsubscribe@egroups.com
```

Remplacer les adresses mails :

```
 anyone dollardollardollardollardollar knows how much dollar it costs
to host a web portal ? well, it depends on how many visitors youre
expecting.

 this can be anywhere from less than nombre bucks a month to a couple
of dollarnombre.
 you should checkout httpaddr or perhaps amazon ecnombre if youre
running something big..
 to unsubscribe yourself from this mailing list, send an email to:
emailaddr
```

Remplacer les balises HTML :

```
 anyone dollardollardollardollardollar knows how much dollar it costs
to host a web portal ?
 well, it depends on how many visitors youre expecting.
 this can be anywhere from less than nombre bucks a month to a couple
of dollarnombre.
 you should checkout httpaddr or perhaps amazon ecnombre if youre
running something big..
 to unsubscribe yourself from this mailing list, send an email to:
emailaddr
```

Suppression des non mots :

```
 anyone dollardollardollardollardollar knows how much dollar it costs
to host a web portal well it depends on how many visitors youre
expecting this can be anywhere from less than nombre bucks a month to
a couple of dollarnombre you should checkout httpaddr or perhaps
amazon ecnombre if youre running something big to unsubscribe
yourself from this mailing list send an email to emailaddr
```

Radicalisation :

```
 anyon dollardollardollardollardollar know how much dollar it cost to
host a web portal well it depend on how mani visitor your expect this
can be anywher from less than nombr buck a month to a coupl of
dollarnombr you should checkout httpaddr or perhap amazon ecnombr if
your run someth big to unsubscrib yourself from this mail list send
an email to emailaddr
```

Après le pretraitement :

```
 anyon dollardollardollardollardollar know how much dollar it cost to
host a web portal well it depend on how mani visitor your expect this
can be anywher from less than nombr buck a month to a coupl of
dollarnombr you should checkout httpaddr or perhap amazon ecnombr if
your run someth big to unsubscrib yourself from this mail list send an
email to emailaddr
```

Construction du vocabulaire

Le vocabulaire est un ensemble de mots (les plus fréquents) que va utiliser un classifieur pour l'étape de classification, soit k, le nombre de mots se trouvant dans notre vocabulaire. Etant donné que k est un paramètre empirique, il est judicieux de bien choisir sa valeur, nous allons donc pour la partie classification tester plusieurs valeurs de k pour comparer. Notre base se composant de 6000 mails, les valeurs de k a tester sont [1000, 1500, 2000, 2500].

Classification

Après avoir nettoyé nos emails, nous allons pour chacun d'entre eux en extraire les caractéristiques (k mots plus fréquents) pour ensuite débuter la classification.

Extraction de caractéristiques

Dans cette étape, nous avons utilisé la fonction prédéfinie `CountVectorizer` qui convertit une collection de texte (emails) en une matrice binaire de taille égale au vocabulaire (k), ou chaque mot du texte se trouvant dans le vocabulaire représente pour chaque email un 1 si l'email le contient sinon 0.
Note: une autre méthode a été testé en utilisant un fichier contenant les mots les plus fréquent mais a été remplacé vu la meilleur complexité temporelle de CountVectorizer.

Voici donc un exemple de ce que va être représenté un email, avec un vocabulaire de taille 1000 mots:

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0
 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 1 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0
 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 0 0 1 0 0 0 1 0 0 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 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1
 0 0 0 0 0 1 0 1 0 1 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1
 0 1 0 0 0 0 0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1
 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0
 1 1 0 1 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 1
 1 1 0 0 0 0 0 0 1 1 0 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 1 1 0 0 0 0 1 0 0 1 1 0 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0
 1 1 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 1 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
 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 1 1 0 0 0 1 0
 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0
 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 1 0 1 0 0 0 0 1 1
 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 1 0 1 0 0 1
 0 1 0 0 0 1 1 1 0 1 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1
 0]
Taille de la matrice 1000
```

Librairies utilisées

Pour la classification nous avons testé plusieurs modèles de la librairie Sklearn, ou nous allons voir cela en détails dans ce qui suit.
De plus, nous l'avons utilisé pour diviser nos données en train/test avec un pourcentage de 75% pour l'entraînement et 25% pour le test.

Modèles utilisés :

Pour tous ces modèles, nous allons tester 4 valeurs de **K** et comparer entre eux afin de confirmer notre choix précédent lors du prétraitement.

- ## Support Vector Machine:

Algorithme de classification de l'apprentissage supervisé, prédit la classe en séparant les données avec un hyperplan
Utilisé directement à partir de `Pipeline(steps=[('standardscaler', StandardScaler()),('svc', SVC(gamma='auto'))])`
ou `StandardScaler()` est utilisé pour normaliser les données, svc correspondant a l'algorithme choisit qui est SVM et gamma, un paramètre de SVM qui représente le coefficient kernel et qui est a auto (1/nombre de données), voici les résultats obtenus :

k	1000	1500	2000	2500
Accuracy	0.9737	0.9711	0.9704	0.9685
Precision	0.9862	0.9860	0.9837	0.9859
Recall	0.9265	0.9179	0.9179	0.9092
F1-score	0.9554	0.9507	0.9497	0.9460

Nous remarquons que pour **k = 1000** mots, nous obtenons la plus grande valeur d'accuracy égale à **0.973** pour l'algorithme SVM.

- ## Gaussian Naive Bayes:

L'algorithme Naïve Bayes est un algorithme d'apprentissage supervisé, basé sur le théorème de Bayes et utilisé pour résoudre des problèmes de classification.
Il est principalement utilisé dans la classification de texte.
Gaussien : le modèle gaussien suppose que les entités suivent une distribution normale, ce qui n'est pas notre cas.
Utilise directement à partir de `GaussianNB()`, voici les résultats obtenus :

k	1000	1500	2000	2500
Accuracy	0.874	0.826	0.838	0.832
Precision	0.717	0.645	0.663	0.656
Recall	0.969	0.950	0.946	0.939
F1-score	0.824	0.768	0.780	0.772

Nous remarquons que pour **k = 1000** mots, nous obtenons la plus grande valeur d'accuracy égale à **0.874** pour l'algorithme Naive Bayes Gaussien, qui est plutôt bien mais moins bonne que celle de SVM.

- Multinomial Native Bayes:

Contrairement au Naive Bayes Gaussien, Le classificateur Multinomial Naïve Bayes est utilisé lorsque les données sont distribuées multinomiales. Il est principalement utilisé pour les problèmes de classification de documents, cela signifie qu'un document particulier appartient à quelle catégorie.
Le classifieur utilise la fréquence des mots pour la prédiction.
Utilise directement à partir de `MultinomialNB()` et paramètres **fit_prior** = true ,**alpha soothing paramètre** = 1.0 voici les résultats obtenus :

k	1000	1500	2000	2500
Accuracy	0.968	0.971	0.971	0.971
Precision	0.950	0.956	0.954	0.954
Recall	0.946	0.950	0.950	0.950
F1-score	0.948	0.953	0.952	0.952

Nous remarquons que pour **k = 1500** mots, nous obtenons la plus grande valeur d'accuracy égale à **0.971** pour l'algorithme Multinomial Bayes, qui est plutôt bien mais moins bonne que celle de SVM.
Ceci dit, Multinomial Bayes reste mieux que Gaussian, et ce parce que nos données ne suivent pas une distribution normale.

- Régression logistique:

Dans cette partie nous avons utilisé un algorithme d'apprentissage supervisé, il calcule la probabilité d'un événement binaire comme la classification (généralement binaire entre 0 et 1) comme dans notre cas 0 représente Ham et 1 Spam.

Utilisé directement à partir de LogisticRegression() avec comme paramètres un biais de 1, un algorithme d'optimisation de coût =lbfgs et max itération = 100

k	1000	1500	2000	2500
Accuracy	0.979	0.98	0.979	0.98
Precision	0.965	0.969	0.963	0.967
Recall	0.965	0.965	0.967	0.967
F1-score	0.965	0.965	0.965	0.967

Nous remarquons que pour **k = 2500** mots, nous avons une accuracy égale à **0.98** pour l'algorithme Logistic Regression, qui vient prendre place de SVM avec de meilleurs résultats pour toutes les métriques calculées.

- Decision Trees:

Une autre technique d'apprentissage supervisé, pour construire un arbre de décision, nous utilisons l'algorithme CART, qui signifie Classification and Regression Tree algorithm.
Un arbre de décision pose simplement une question et, en fonction de la réponse (Oui/Non), il divise ensuite l'arbre en sous-arbres.
Utilisé directement à partir de `tree.DecisionTreeClassifier()` avec comme paramètres:

max_leaf_nodes = infinite

max_depth= unlimited

criterion = gini (the function to measure the quality of a split.)

splitter = best (The strategy used to choose the split at each node..)

k	1000	1500	2000	2500
Accuracy	0.979	0.980	0.979	0.980
Precision	0.965	0.969	0.963	0.967
Recall	0.965	0.965	0.967	0.967
F1-score	0.965	0.967	0.965	0.967

Pour **k = 2500** mots, nous avons une accuracy égale à **0.98** pour le dernier algorithme Arbre de décision et qui donne exactement les mêmes résultats que la Régression Logistique.

En essayant plusieurs algorithmes d'apprentissage supervisé, nous sommes arrivés à la conclusion que **pour k = 2500** et en utilisant **Decision Tree** et **Logistic Regression**, nous obtenons les meilleurs résultats.

- Neural Network:

Dans cette partie nous avons utilisé un réseau de neurones de Sklearn nomme MLP pour multi-layer perceptron, c'est un algorithme d' apprentissage supervisé qui utilise comme paramètre L-BFGS pour l'entraînement du modèle et la backpropagation, avec une valeur alpha pour la regularization=1e-5 , une activation relu, learning rate = 0.001
deux hidden layers de 5 et 2 respectivement et solver = adam

k	1000	1500	2000	2500
Accuracy	0.978	0.976	0.98	0.977
Precision	0.973	0.959	0.969	0.971
Recall	0.954	0.963	0.965	0.954
F1-score	0.964	0.961	0.967	0.962

Pour un **k = 2000**, nous avons une accuracy de **0.98** qui est meilleure comparée au autres valeurs de k, nous obtenons de meilleurs résultats concernant les autres métriques aussi.

Comparaison Machine Learning-Deep Learning

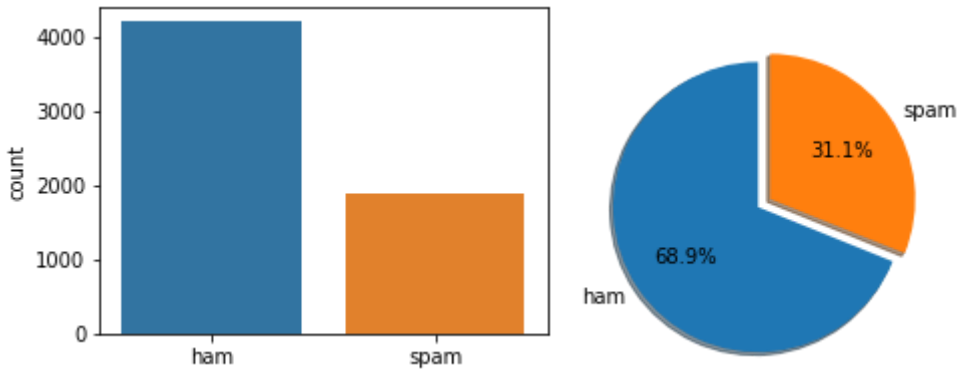
On remarque que la performance du modèle de deep learning est supérieur à tous les autres modèles de deep learning, que ca soit en rapidité d'exécution et de prédictions.
les métriques sont aussi à son avantage avec une précision de 0.96 et accuracy de 0.98, suivi par la régression logistique avec 0.97 et naïve bayes.

Traitement du déséquilibre de classe avec SMOTE

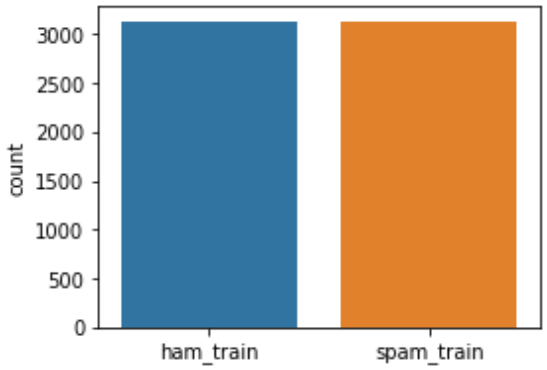
Comme observé au début du prétraitement, il ya bien un déséquilibre dans la distribution des classe, les spam ne représentent que **31%** de notre dataset, notre modèle va alors la majorité du temps rencontrer des exemples de non-spam et aura tendance à mieux les identifier ce qui peut causer des résultats médiocre vis à vis de la détection de la classe spam, pour cela nous avons recours à une technique de sur-échantillonnage appelle **SMOTE: Synthetic minority oversampling technique** qui nous permet de generer des donnees synthetiques dans la classe minoritaire (donc la classe spam) afin d'avoir une distribution équilibrée et une meilleur performance du modèle. il fonctionne de la sorte:

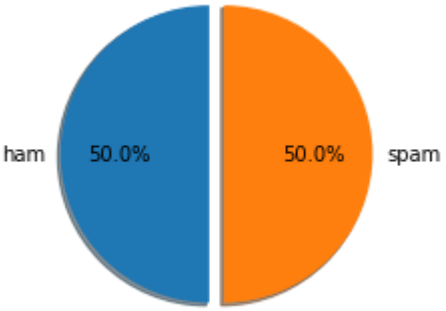
> « Un exemple aléatoire de la classe minoritaire est d'abord choisi. Ensuite, k des voisins les plus proches pour cet exemple sont trouvés (typiquement k = 5). Un voisin sélectionné au hasard est choisi et un exemple synthétique est créé à un point sélectionné au hasard entre les deux exemples dans l'espace des caractéristiques. Page 47, Imbalanced Learning: Foundations, Algorithms, and Applications, 2013.

Il faut noter que ce procédé n'est appliqué que sur l'ensemble d'entraînement après prétraitement et non sur tout le dataset, cela permet de préserver l'intégrité de la prédiction et de ne pas fausser les résultats lors de l'évaluation .



Voici notre ensemble d'entraînement après SMOTE: en utilisant les paramètres `random_state=10,k_neighbors=3 (obtenus par expérimentation)`
Les deux classes ont la même distribution et aucune n'est minoritaire par rapport à l'autre, cette technique est très utilisée dans les problèmes de classification notamment la classification de texte.

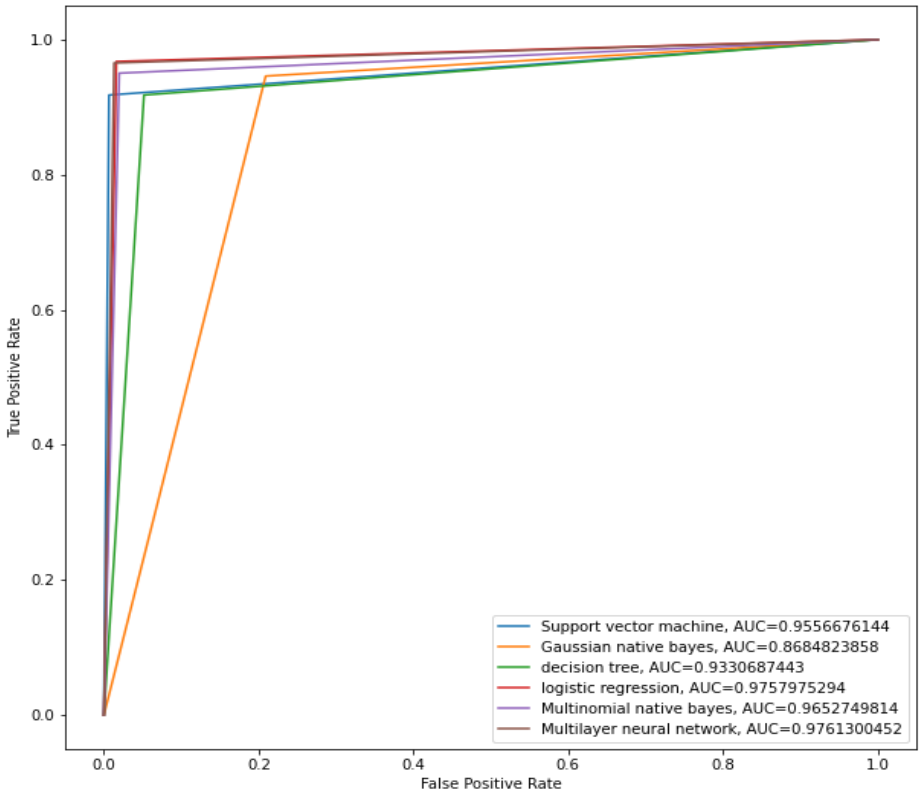




Comparaison SMOTE

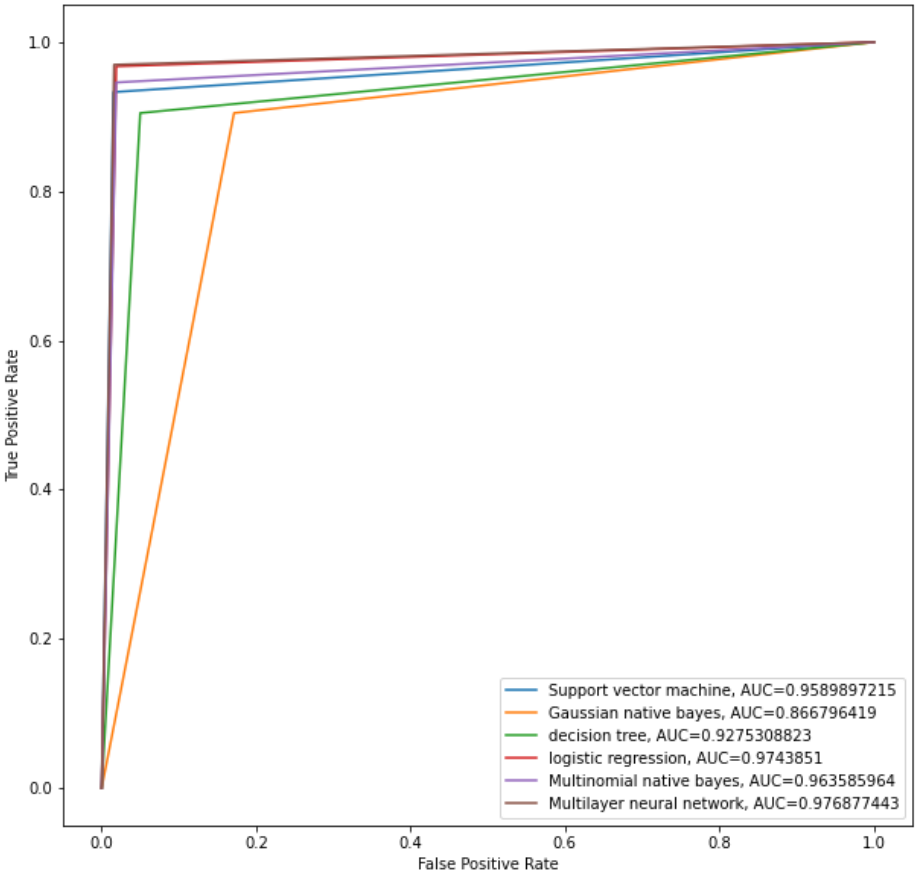
Avant SMOTE

On remarque depuis la courbe ROC que l'algorithme d'apprentissage profond a la meilleure performance avec une AUC =0.97613 suivi par les algorithmes d'apprentissage automatique avec en tête la régression logistique et à la fin gaussian naive bayes.



Après SMOTE

On remarque la même hiérarchie des algorithmes avec une légère amélioration générale des performances avec une max AUC =0.97687.



Pour comparer voici les métriques des deux approches (avec et sans SMOTE):

AVEC SMOTE K=2000	**SANS SMOTE K=2000**
Le classifieurs : Gaussian Accuracy: 0.8518032786885246 Precision: 0.697171381031614 Recall: 0.9049676025917927 F1-score: 0.787593984962406 [[880 182] [44 419]] **Le classifieurs : SVM** Accuracy: 0.9691803278688524 Precision: 0.9642857142857143 Recall: 0.9330453563714903 F1-score: 0.9484083424807903 [[1046 16] [31 432]] **Le classifieurs : Multinomial Naive Bayes** Accuracy: 0.9704918032786886 Precision: 0.9563318777292577 Recall: 0.9460043196544277 F1-score: 0.9511400651465798 [[1042 20] [25 438]]	**Le classifieurs : Gaussian** Accuracy: 0.8380327868852459 Precision: 0.6636363636363637 Recall: 0.9460043196544277 F1-score: 0.7800534283170081 [[840 222] [25 438]] **Le classifieurs : SVM** Accuracy: 0.9704918032786886 Precision: 0.9837962962962963 Recall: 0.91792656587473 F1-score: 0.9497206703910615 [[1055 7] [38 425]] **Le classifieurs : Multinomial Naive Bayes** Accuracy: 0.9711475409836066 Precision: 0.9544468546637744 Recall: 0.9503239740820735 F1-score: 0.9523809523809523 [[1041 21] [23 440]]

Le classifieurs : Logistic Regression Accuracy: 0.9770491803278688 Precision: 0.9572649572649573 Recall: 0.9676025917926566 F1-score: 0.962406015037594 [[1042 20] [15 448]] **Le classifieurs : Decision Tree** Accuracy: 0.9363934426229508 Precision: 0.8877118644067796 Recall: 0.9049676025917927 F1-score: 0.8962566844919785 [[1009 53] [44 419]] **Le classifieurs : Neural Network** Accuracy: 0.979672131147541 Precision: 0.9635193133047211 Recall: 0.9697624190064795 F1-score: 0.9666307857911733 [[1045 17] [14 449]]	**Le classifieurs : Logistic Regression** Accuracy: 0.9790163934426229 Precision: 0.9634408602150538 Recall: 0.9676025917926566 F1-score: 0.9655172413793104 [[1045 17] [15 448]] **Le classifieurs : Decision Tree** Accuracy: 0.9350819672131148 Precision: 0.885593220338983 Recall: 0.9028077753779697 F1-score: 0.8941176470588235 [[1008 54] [45 418]] **Le classifieurs : Neural Network** Accuracy: 0.980327868852459 Precision: 0.9696312364425163 Recall: 0.9654427645788337 F1-score: 0.9675324675324676 [[1048 14] [16 447]]

Conclusion

A travers ce projet, de classification des emails Spam/Non Spam, nous avons été amenés à traiter des données de types emails, en commençant par le **prétraitement** de ces dernières, et cela en plusieurs étapes à savoir, la conversion des emails en textes simple, l'élimination de tout ce qui pouvait provoquer un surapprentissage (nombre, URLs, adresses emails) ainsi que l'élimination des mots/caractères dont l'utilité est minime comme la ponctuation ou les saut de lignes et tabulation. Le prétraitement fait n'est autre que les étapes du **NLP** (Natural Language Processing).

Puis nous avons utilisé et comparé différentes méthodes de deep learning et machine learning; nous avons également géré le traitement du déséquilibre de classe avec les algorithmes de sur-échantillonnage.
Plusieurs de ces méthodes ont donné des résultats assez satisfaisants, notamment **Les Arbres de Décision** ainsi que la **Régression Logistique** , ce qui montre le succès de l'utilisation du machine learning pour le traitement de cette problématique, cependant, l'utilisation de l'apprentissage profond a donné de meilleurs résultats dans notre cas d'étude.

Le machine learning et le deep learning s'est avéré être une méthode efficace pour traiter le problème de la détection de spams parmi les e-mails reçus, qui demeure un problème récurrent dans notre ère digitale et le danger que les e-mails mal intentionnés peuvent causer (vol d'argent, accès à des informations privées...).