Mémoire sur le data chalenge kaggle suivant :

Natural Language Processing with Disaster Tweets

(Traitement automatique du langage naturel avec des tweets de catastrophe)

Rédigé par Nasseme Ahamed

Pour le compte de l'UE mémoire/stage ISDS Master 1

2022-2023

TABLES DES MATIERES

I-INTRODUCTION

« L'intelligence artificielle va changer notre monde plus que tout ce que nous avons vu jusqu'à présent." » Cette phrase de Warren Buffet est aujourd'hui plus réelle que jamais. Les avancées réalisées dans le domaine de l'intelligence artificielle sont considérables, malgré les difficultés rencontrées. Actuellement, l'intelligence artificielle n'est plus seulement un rêve, mais une réalité tangible. Cette réalité est observée à travers les chat bots, tels que ChatGPT, ainsi que les algorithmes de recherche ou de recommandation, comme ceux de YouTube ou Google, entre autres.

C'est sur ce sujet que nous allons nous pencher, en nous intéressant plus particulièrement aux algorithmes NLP (Natural Language Processing), en français les algorithmes de traitement du langage naturel. Ils trouvent des applications dans de nombreux domaines incluant le langage naturel, tels que la compréhension du langage naturel, la traduction automatique, l'analyse des sentiments, le résumé automatique, les chats bots et les assistants virtuels, etc.

Dans ce mémoire, nous nous consacrerons à un défi Kaggle lié aux NLPs. Les données sont constituées de plusieurs tweets classés en 0 (pour les tweets normaux) et 1 (pour les tweets urgents). L'objectif est de développer un algorithme entraîné sur ces données pour prédire si un tweet est urgent ou non. Ce type d'algorithme suscite l'intérêt de diverses institutions, notamment des médias, des services de secours tels que les pompiers, les forces de l'ordre et les ONG. En effet, ces algorithmes permettent de repérer rapidement les situations dangereuses grâce à la diffusion des informations sur les réseaux sociaux.

Les données se composent de trois parties : une partie d'entraînement (Train) pour la construction du modèle, une partie de test pour l'évaluation finale (Test), et une partie d'échantillon (Sample) contenant les cibles des données de test.

Nous commencerons par importer et analyser les données, puis diviserons la partie d'entraînement en une section pour la construction du modèle et une autre pour sa validation. Nous utiliserons initialement des méthodes classiques. À partir d'une matrice de jetons (mots) créée à partir des données d'entraînement, nous appliquerons plusieurs algorithmes de classification, notamment RandomForestClassifier, MultinomialNB et SVM.

Enfin, nous aborderons des algorithmes plus avancés, tels que BERT, un modèle de traitement du langage naturel développé par Google en 2018. BERT capture le contexte global d'un mot en considérant à la fois les mots précédents et suivants dans une phrase. Nous explorerons également TensorFlow, une bibliothèque open source créée par Google, spécialisée dans le deep learning. Elle est largement utilisée dans le domaine de l'apprentissage automatique et du traitement du langage naturel (NLP).

II-PRÉ-REQUIS

Durant cette section, nous aurons besoin du langage Python, en particulier de divers modules tels que pandas, numpy, seaborn, matplotlib, sklearn, et même TensorFlow pour la seconde partie. Il convient de noter que TensorFlow, comme nous l'examinerons par la suite, offre des algorithmes plus sophistiqués adaptés à notre problématique.

III-OBSERVATIONS DES DONNÉES

On va commencer l'étude de nos données avec la lecture et le chargement du fichier CSV ("train.csv") un en utilisant la bibliothèque pandas de Python.

Les données sont composée d'identifiant(id), keyword, location et text comme on le voit dans le tableau suivant :

```
df[45:50]
```

	id	keyword	location	text	target
45	65	ablaze	NaN	I gained 3 followers in the last week. You? Kn...	0
46	66	ablaze	GREENSBORO,NORTH CAROLINA	How the West was burned: Thousands of wildfire...	1
47	67	ablaze	NaN	Building the perfect tracklist to life leave t...	0
48	68	ablaze	Live On Webcam	Check these out: http://t.co/rOI2NSmEJJ http:/...	0
49	71	ablaze	England.	First night with retainers in. It's quite weir...	0

C'est une représentation du dataframe, de la ligne 45 à la ligne 50(50 non incluse).

D'autre part, nos données présentent un équilibre satisfaisant (autant de textes urgents que de textes normaux), ce qui permet d'éviter les problèmes de biais, comme illustré ci-dessous :

```
df.target.value_counts()
```

```
0    4342
1    3271
Name: target, dtype: int64
```

```
df.target.value_counts(normalize=True)
```

```
0    0.57034
1    0.42966
Name: target, dtype: float64
```

Clairement on a environ 57% de tweets normaux et 43% de tweets urgents

Featuring engineering

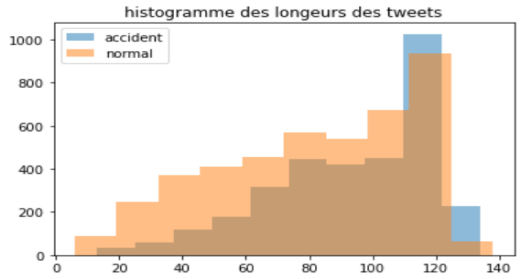
- La longueur des tweets peut servir de bon indicateur pour la caractérisation des tweets normaux et urgents. Pour cela, nous allons ajouter une colonne 'length_texte', qui fera correspondre, pour chaque ligne, la taille du tweet correspondant. Ainsi, nous pourrons examiner l'impact de cette variable sur nos données. Voici le nouveau tableau:

```
df[45:50]
```

	id	keyword	location	text	target	text_length
45	65	ablaze	NaN	I gained 3 followers in the last week. You? Kn...	0	81
46	66	ablaze	GREENSBORO,NORTH CAROLINA	How the West was burned: Thousands of wildfire...	1	85
47	67	ablaze	NaN	Building the perfect tracklist to life leave t...	0	54
48	68	ablaze	Live On Webcam	Check these out: http://t.co/rOI2NSmEJJ http:/...	0	107
49	71	ablaze	England.	First night with retainers in. It's quite weir...	0	112

Graphique illustratif :

```
plt.hist(df[df["target"]==1]["text_length"], alpha=0.5, label="accident")
plt.hist(df[df["target"]==0]["text_length"], alpha=0.5, label="normal")
plt.legend(loc="upper left")
plt.title("histogramme des longeurs des tweets")
plt.show()
```



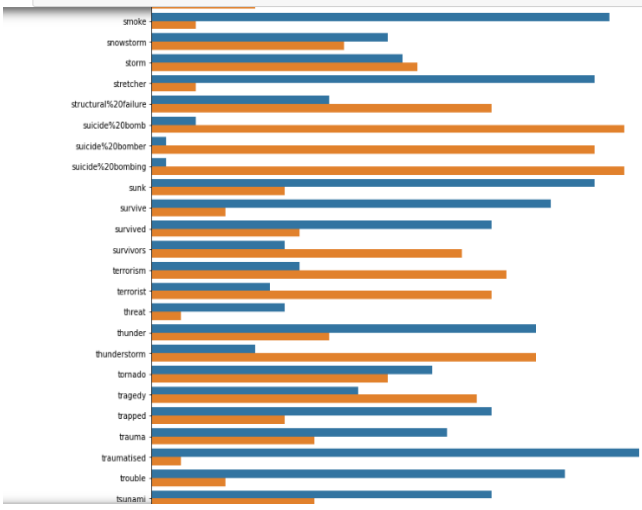
Il semble que plus les tweets sont long, plus ils sont accidentels(urgent) et plus c'est court plus c'est normal

Dans les modèles qu'on va utiliser, on ne va pas prendre en compte cette remarque, car même si, elle semblait vrai ; les chargements techniques seront assez longs, et on aura pas beaucoup de changements considérable.

-L'analyse des keywords permet de voir s'il y'a une forte corrélation entre les Targets et les keywords. Intuitivement, il est clair que les keywords apportent une information de plus sur l'urgence du tweet, mais comme précédemment, pour des raisons de simplicité de notre algorithme, on ne va pas tenir compte de ce featuring engineering. Par contre les evoqué nous permet de voir d'autres horizons, et des possibilité d'amélioration de nos modèles.

Voici les résultats :

```
plt.figure(figsize=(15,100))
sns.countplot(data=df,y="keyword",hue="target")
```



L'intuition qu'on avait est vérifiée, des mots comme suicid, terrorism etc. sont plus fréquent comme keyword pour des tweet urgent, toute fois on remarque quelque inattendu, comme traumatised qui est plutôt récurent chez des tweets normaux. Pourtant le mot peu bien être classé comme keyword de tweet urgent.

IV-PARTITION DES DONNÉES

Comme évoqué plus haut, on divisera les données train en 2 parties : train (pour l'entrainement du modèle) et test (pour l'évaluation du modèle).

D'autre part, pour chacune de ces deux parties, nous disposerons des variables X pour les caractéristiques et Y pour les étiquettes. Voici comment se présente la division:

```
[12]: const=200
      X_train,X_test,Y_train,Y_test=train_test_split(
          df["text"],
          df["target"],
          test_size=0.15,
          random_state=const)
```

Le test_size est à 0.15 ce qui veut dire que 85% des données sont allouées aux données d'entrainement et 15% aux données de validation

REMARQUE 1

Il est nécessaire de noter que la plupart des documents et vidéos traitant de ce sujet ont tendance à recommander un test_size entre 20% et 30%, cela afin d'éviter le surapprentissage. Par conséquent, le choix d'un test_size de 15% que j'ai fait pourrait probablement conduire à des résultats assez bons lors de la validation du modèle. Cependant, il y a un risque élevé que le modèle soit surestimé. Par conséquent, nous allons relativiser nos résultats de validation."

V-preprocessing

Dans cette partie on va utiliser le CountVectorizer. CountVectorizer est une classe de la bibliothèque Python scikit-learn, qui est utilisée pour transformer du texte en vecteurs de compte. C'est l'une des techniques les plus simples de représentation de texte en apprentissage automatique, elle est largement utilisée dans le traitement du langage naturel (NLP), pour préparer des données textuelles avant de les utiliser dans des modèles d'apprentissage automatique.

Le processus de CountVectorizer consiste à convertir un ensemble de documents textuels en une matrice de fréquence de termes, où chaque ligne représente un texte et chaque colonne représente un mot du vocabulaire. La valeur dans chaque cellule de la matrice indique le nombre d'occurrences du mot correspondant dans le document correspondant.

Ainsi on pourra utiliser les matrices obtenue pour appliquer les méthodes de classification, car c'est des variables numériques, donc manipulable par les algorithmes

Voici la création de la matrice :

```
n [17]: Vectorizer=CountVectorizer()
        X_train_vec_pre=Vectorizer.fit_transform(X_train)
        X_test_vec_pre=Vectorizer.transform(X_test)
```

X_train_vec et X_test_vec représentent respectivement les matrices numériques qui correspondent aux vecteurs des tweets de X_train et X_test. C'est à partir de ces matrices que nous effectuerons l'entraînement en utilisant un algorithme de classification

REMARQUE 2

On aura besoin d'analyser les performances de chacun de nos modèles et d'avoir une vue d'ensemble sur ces performances, ainsi on va utiliser l'objet classification_report(), qui va nous donner, pour chaque modèle, les valeurs suivantes :

Precision : La précision est le rapport entre le nombre de vrais positifs et le nombre total de prédictions positives. C'est une mesure de l'exactitude des prédictions positives du modèle. Mathématiquement, cela se calcule comme : **Précision = Vrais Positifs / (Vrais Positifs + Faux Positifs)**

Recall : Le rappel est le rapport entre le nombre de vrais positifs et le nombre total d'exemples réels positifs. Il mesure la capacité du modèle à retrouver tous les exemples positifs. Mathématiquement, cela se calcule comme : **Rappel = Vrais Positifs / (Vrais Positifs + Faux Négatifs)**

F1-score : Le score F1 est la moyenne harmonique de la précision et du rappel. Il est utilisé lorsque vous voulez trouver un équilibre entre la précision et le rappel. Le F1-score est d'autant plus élevé que la précision et le rappel sont équilibrés. Mathématiquement, cela se calcule comme : **F1-score = 2 * (Précision * Rappel) / (Précision + Rappel)**

Support : Le support représente le nombre d'occurrences de chaque classe dans les données de test. Il indique combien d'exemples réels appartiennent à chaque classe.

Accuracy : L'exactitude est le rapport entre le nombre d'exemples correctement classés et le nombre total d'exemples. C'est une mesure globale de la précision du modèle. Mathématiquement, cela se calcule comme : **Exactitude = (Vrais Positifs + Vrais Négatifs) / (Vrais Positifs + Faux Positifs + Vrais Négatifs + Faux Négatifs)**

Macro avg : C'est la moyenne non pondérée des métriques de précision, de rappel et de score F1 pour toutes les classes. Chaque classe contribue de manière égale au calcul de cette moyenne.

Weighted avg : C'est la moyenne pondérée des métriques de précision, de rappel et de score F1 pour toutes les classes. Cette moyenne est pondérée par le nombre d'exemples réels de chaque classe.

VI-Classificateurs Traditionnels

Pour cette catégorie de classificateurs, on va étudier 3 différents algorithme, en premier RandomForestClassifier puis MultinomialNB et enfin SVM

Les modèles SVM, RandomForestClassifier et MultinomialNB ont des bases mathématiques et statistiques différentes, adaptées à divers types de problèmes et de données. Chacun de ces modèles a ses propres avantages et inconvénients, et leur choix dépendra du problème spécifique que vous cherchez à résoudre et des caractéristiques de vos données.

VI-a-RandomForestClassifier

RandomForestClassifier est un algorithme d'apprentissage automatique de type "Ensemble Learning" (apprentissage par ensemble) utilisé pour des tâches de classification ou de régression.

Il est basé sur l'idée de construire plusieurs arbres de décision indépendants pendant le processus d'entraînement et de combiner leurs prédictions pour prendre une décision finale.

Cet algorithme est souvent utilisé pour des tâches de classification, telles que la catégorisation d'images, la détection de spam, la prédiction de maladies, etc. Il est apprécié pour sa robustesse, sa capacité à gérer des données avec des caractéristiques manquantes ou du bruit, ainsi que pour sa réduction du sur apprentissage (overfitting) par rapport à un arbre de décision unique.

Mathématiques : Random Forest (Forêt aléatoire) est une méthode d'ensemble qui combine plusieurs arbres de décision. Chaque arbre est construit en utilisant une sous-section aléatoire des données et des caractéristiques. Les prédictions sont ensuite agrégées pour produire un résultat final.

Random Forest est basé sur des statistiques d'échantillonnage aléatoire et d'agrégation de prédictions. Il est robuste contre le sur apprentissage (overfitting) grâce à la combinaison d'arbres multiples et à la réduction de la variance.

```
In [20]: rfc=RandomForestClassifier(random_state=const)
         rfc_fit=rfc.fit(X_train_vec_pre,Y_train)
```

rfc_fit est le modèle RandomForestClassifier entraîné avec nos données d'entraînement

```
In [21]: rfc_fit.score(X_test_vec_pre,Y_test)
Out[21]: 0.7968476357267951
```

Au-dessus, nous avons calculé l'efficacité de notre modèle avec les données de test, et le résultat est légèrement supérieur à 79%. Cependant, en observant les résultats ci-dessous, nous constatons qu'en approfondissant nos tests avec trois essais simultanés, notre modèle affiche une performance moyenne d'environ 78%.

```
In [23]: resultat=cross_val_score(rfc, X_train_vec_pre,Y_train, cv=3)
         resultat
Out[23]: array([0.77839592, 0.77561428, 0.78535002])
```

en moyenne, on a un taux de réussite de 78% environ

Voici les résultats de classification _report() :

```
In [22]: print(classification_report(Y_test, rfc_fit.predict(X_test_vec_pre)))
```

	precision	recall	f1-score	support
0	0.77	0.94	0.84	668
1	0.88	0.59	0.71	474
accuracy			0.80	1142
macro avg	0.82	0.77	0.78	1142
weighted avg	0.81	0.80	0.79	1142

VI-b-MultinomialNB

MultinomialNB (Naive Bayes multinomial) est un algorithme d'apprentissage automatique utilisé pour la classification de textes et d'autres types de données discrètes. Il fait partie de la famille des classificateurs Naive Bayes, qui sont basés sur le théorème de Bayes et supposent que les caractéristiques sont indépendantes les unes des autres, d'où le terme "naïf".

Les avantages de MultinomialNB sont sa simplicité, son efficacité en termes de calcul, et il est souvent performant même avec des ensembles de données de petite taille. Cependant, comme tous les classificateurs Naive Bayes, il suppose l'indépendance conditionnelle des caractéristiques, ce qui peut ne pas être réaliste pour certaines tâches. Malgré cela, il s'avère être un choix populaire et solide pour de nombreuses applications de traitement du langage naturel.

Mathématiques : Le modèle Naïve Bayes multinomial est basé sur le théorème de Bayes et est principalement utilisé pour la classification de texte. Il suppose que les caractéristiques sont conditionnellement indépendantes, d'où le terme "Naïve". Il est particulièrement adapté aux données discrètes, telles que les fréquences de mots dans un texte. Ce modèle est étroitement lié aux concepts de probabilité et d'indépendance conditionnelle. Il est souvent utilisé dans le traitement automatique du langage naturel et d'autres domaines où les données sont exprimées sous forme de comptages.

```
nb = MultinomialNB()
nb_fit=nb.fit(X_train_vec_pre,Y_train)
```

Nb est le modèle MultinomialNB entraîné avec nos données d'entraînement

```
[26]: nb_fit.score(X_test_vec_pre,Y_test)
Out[26]: 0.8099824868651488
```

Le résultat de ce dernier modèle à 80%, semble être plus intéressant que le 1er modèle, une tendance qui va être confirmé par la cross-validation suivant :

```
resultat2=cross_val_score(nb, X_train_vec_pre,Y_train, cv=3)
resultat2
```

```
: array([0.78535002, 0.78488642, 0.80296708])
```

Résultat de la classification_report() :

```
]: print(classification_report(Y_test, nb.predict(X_test_vec_pre)))
              precision    recall  f1-score   support

           0       0.81      0.89      0.85       668
           1       0.82      0.70      0.75       474

    accuracy                           0.81      1142
   macro avg       0.81      0.79      0.80      1142
weighted avg       0.81      0.81      0.81      1142
```

La tendance semble montrer que ce dernier est meilleur, mais encore une fois il faut faire attention car la différence n'est pas assez importante, et il faut qu'on vérifie s'il n'y a effectivement pas de biais en cours.

VI-b-SVM

SVM (Support Vector Machine) est un algorithme d'apprentissage supervisé utilisé pour la classification et la régression. L'objectif principal de SVM dans le contexte de la classification est de trouver un hyperplan (une frontière de décision) qui sépare les exemples de différentes classes de manière optimale dans l'espace des caractéristiques

Mathématique : Le SVM vise à trouver un hyperplan dans un espace de dimension élevée qui sépare au mieux les différentes classes de données. Mathématiquement, il s'agit de trouver l'hyperplan qui maximise la marge entre les classes tout en minimisant les erreurs de classification. Il est efficace dans des espaces de grande dimension et est largement utilisé pour la classification binaire. Il repose sur des concepts statistiques tels que la marge maximale et la projection des données dans un espace de caractéristiques

```
In [30]: svm_=svm.SVC(kernel="linear",probability=True)
         svm_fit=svm_.fit(X_train_vec_pre,Y_train)
```

```
In [31]: svm_fit.score(X_test_vec_pre,Y_test)
Out[31]: 0.8056042031523643
```

```
In [32]: print(classification_report(Y_test, svm_fit.predict(X_test_vec_pre)))
```

	precision	recall	f1-score	support
0	0.82	0.86	0.84	668
1	0.78	0.73	0.76	474
accuracy			0.81	1142
macro avg	0.80	0.80	0.80	1142
weighted avg	0.80	0.81	0.80	1142

À peu près comme le modèle MultinomialNB, le modèle SVM est légèrement meilleur que le RandomForestClassifier

VII-PIPELINE

Pour pouvoir faire une prédiction, il faut d'abord convertir les donner en matrice numérique, par exemple à l'aide de CountVectorizer, dans les cas précèdent. C'est pour ça qu'on utilise les pipelines.

Les pipelines (ou workflows) sont des outils puissants qui permettent d'organiser et d'automatiser les différentes étapes du processus de prétraitement des données et de modélisation.

Les pipelines permettent de structurer le flux de travail de manière ordonnée, ce qui facilite la répétabilité, la maintenance et l'optimisation des modèles

En général, un pipeline dans le contexte de la data science comprend deux grandes parties :

-Le prétraitement des données et La modélisation Sur ce, on va faire trois pipeline pour les trois modèles précèdent : Voici un exemple avec MultinomialNB :

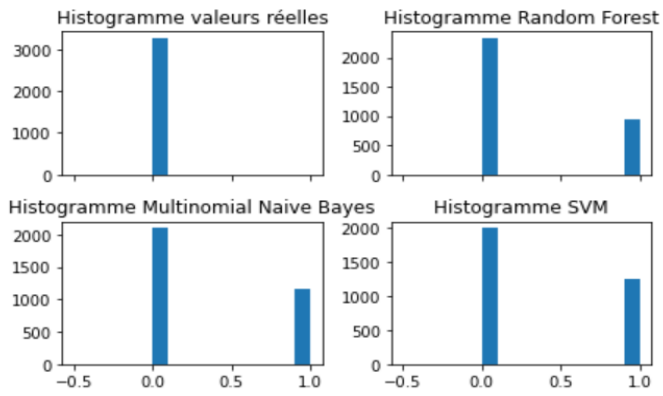
```
ppln_MNB =make_pipeline(
    CountVectorizer(),
    MultinomialNB()
)
```

```
ppln_MNB_fit=ppln_MNB.fit(X_train,Y_train)
```

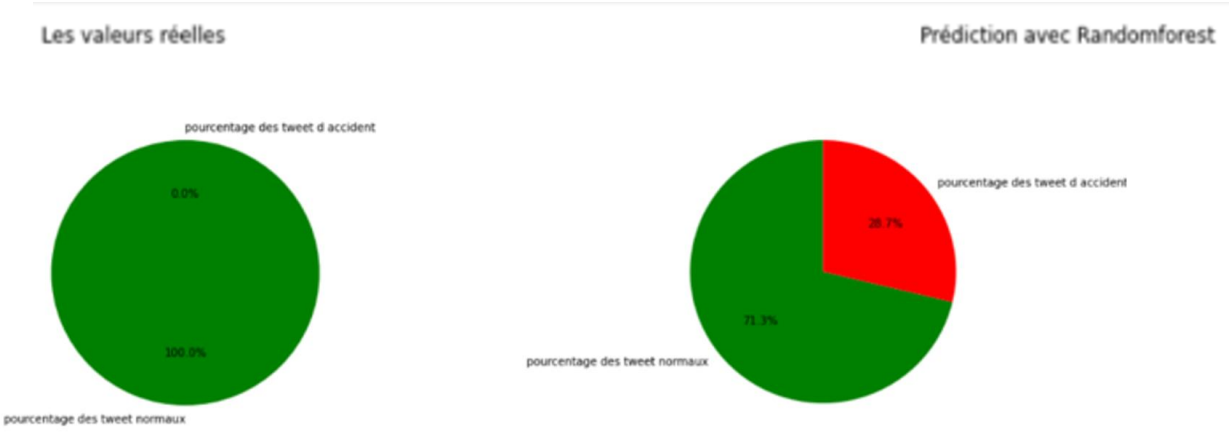
VIII-GRAPHES

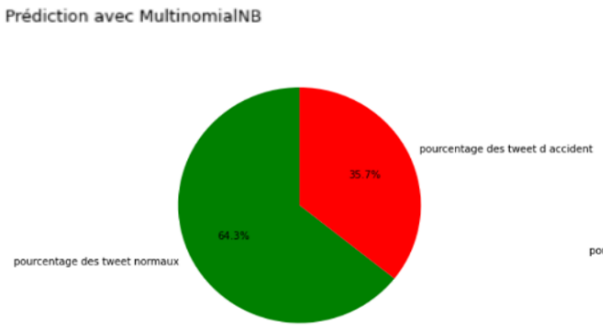
On a implémenté les données pour le test du modèle final, et les réponses, on va montrer deux types de graphes, pour illustrer les donner et voir les donner l'algorithme qui a le mieux marché :

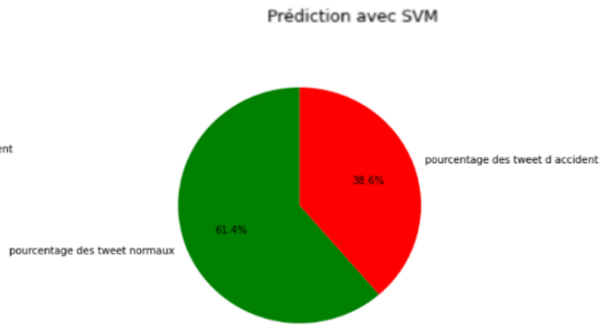
A-graphe histogramme



B-graphe en camembert







Les craintes concernant le meilleur algorithme, malgré les bons résultats obtenus, se sont avérées justifiées. En effet, cela est en grande partie dû au surapprentissage. L'algorithme RandomForestClassifier est moins sujet au surapprentissage par rapport aux autres. D'autre part, nos données d'entraînement étaient plus importantes par rapport aux données de validation, avec un ratio de 85 % et 15 %.

Un autre point intéressant à souligner est que les algorithmes que nous avons utilisés sont considérés en général comme classiques. Ils ne sont pas très optimaux dans le domaine du NLP en comparaison avec d'autres. C'est pourquoi nous allons introduire BERT, qui est plus optimal que CountVectorizer, et TensorFlow, qui se base sur le Deep Learning.

IX- Classification en Utilisant TensorFlow et BERT

Dans cette dernière partie, on va travailler avec Tensorflow et Bert, qui sont plus poussé que les trois algorithmes précédents.

DEEP LEARNING : Le Deep Learning est une branche de l'apprentissage automatique (machine learning) qui se concentre sur l'utilisation de réseaux de neurones artificiels pour résoudre des problèmes complexes de traitement de données.

Le neurone artificiel est l'unité de base du réseau de neurones. Il reçoit des entrées pondérées, applique une fonction d'activation et produit une sortie.

Les neurones sont organisés en couches, notamment une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie

Le Deep Learning vise à simuler le fonctionnement du cerveau humain en utilisant des modèles mathématiques complexes pour apprendre des représentations de haut niveau à partir de données non structures, C’est une approche puissante pour résoudre des problèmes complexes de traitement de données.

Il a apporté des avancées significatives dans de nombreux domaines tels que la vision par ordinateur et le traitement du langage naturel. Le domaine est en constante évolution avec de nouvelles architectures et techniques émergentes.

TENSORFLOW : TensorFlow est une bibliothèque open-source développée par Google pour le calcul numérique et l'apprentissage automatique (machine learning). Elle est largement utilisée pour créer, former et déployer des modèles d'apprentissage automatique, en particulier des réseaux de neurones profonds, qui sont couramment utilisés dans des domaines tels que la reconnaissance d'images, le traitement du langage naturel, la prédiction et bien plus encore.

Pour l'entraînement de modèles, TensorFlow utilise le processus de rétropropagation (backpropagation). Les gradients sont calculés automatiquement pour les opérations afin d'ajuster les poids et les biais du modèle à l'aide d'algorithmes d'optimisation tels que Stochastic Gradient Descent (SGD) ou ses variantes plus avancées, comme Adam. La fonction de perte est utilisée pour évaluer l'écart entre les sorties prédites et les étiquettes réelles.

BERT : BERT, qui signifie Bidirectional Encoder Representations from Transformers, est un modèle de langage pré-entraîné développé par Google en 2018. Il s'agit d'un modèle de traitement du langage naturel (NLP) basé sur l'architecture des Transformers. BERT est célèbre pour ses performances révolutionnaires dans une grande variété de tâches liées au traitement du langage naturel

Contrairement aux modèles traditionnels, BERT prend en compte le contexte de gauche à droite (avant) et de droite à gauche (arrière) lors de l'entraînement. Donc BERT comprend le contexte des mots précédents et suivants pour une tâche donnée, ce qui améliore sa capacité à saisir le sens global d'une phrase.

Mathématiques : les aspects mathématiques et statistiques de TensorFlow et de BERT sont larges et couvrent des domaines tels que les calculs matriciels, les opérations d'optimisation, les fonctions d'activation, les mécanismes d'attention et les transformations linéaires. Ces éléments fondamentaux sont au cœur de la construction et de l'entraînement de ces modèles avancés.

TENSORFLOW :

TENSEURS : TensorFlow est centré autour de la notion de tenseurs, qui sont des objets multidimensionnels similaires à des tableaux ou des matrices. Les opérations mathématiques effectuées sur ces tenseurs sont au cœur des calculs réalisés par les modèles.

Graphes computationnels : il utilise des graphes computationnels pour décrire les opérations mathématiques et les flux de données entre les nœuds. Cela permet d'optimiser l'exécution des calculs et de les répartir sur des dispositifs matériels tels que des processeurs ou des GPU.

Optimisation : Lors de l'entraînement de modèles, TensorFlow utilise des algorithmes d'optimisation tels que la descente de gradient pour ajuster les paramètres du modèle afin de minimiser la fonction de perte.

Fonctions d'activation : Les fonctions d'activation, telles que ReLU (Rectified Linear Activation) et Sigmoid, sont des éléments fondamentaux des couches neuronales dans TensorFlow. Elles introduisent des non-linéarités qui permettent au modèle d'apprendre des relations complexes.

BERT (Bidirectional Encoder Representations from Transformers):

Attention : BERT repose sur le mécanisme d'attention, qui permet au modèle de prendre en compte les dépendances contextuelles entre les mots d'une séquence. Les aspects mathématiques de l'attention incluent la multiplication de matrices pour pondérer les interactions entre les différents tokens.

Transformers : BERT est basé sur l'architecture de modèle de langage Transformer, qui implique des calculs de multiplication de matrices et des opérations de somme pour calculer les représentations des tokens.

Pré-entraînement et fine-tuning : Le processus de pré-entraînement de BERT implique de prédire les mots manquants dans une séquence à l'aide de l'objectif de Masked Language Modeling (MLM). Cela nécessite des calculs probabilistes et des ajustements des paramètres du modèle. Ensuite, pour des tâches spécifiques, le modèle est fine-tuné en utilisant des données spécifiques à la tâche, ce qui peut impliquer des techniques statistiques pour adapter le modèle.

Embeddings : BERT génère des embeddings de mots (représentations vectorielles) pour capturer les informations sémantiques. Les aspects mathématiques ici incluent les transformations linéaires des embeddings, les opérations de somme et les calculs de similarité cosinus pour mesurer les relations entre les mots.

Application des modèles

[5]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import cosine_similarity
```

De sklearn, cosine_similarity permet d'identifier deux vecteurs similaire, plus cosine_similarity est proche de 1 plus les vecteurs sont similaire. Si c'est 0 alors on est proche de l'indépendance des vecteurs et plus c'est -1, plus les vecteurs sont opposés. Et un vecteur représente un mot ou une phrase, donc le cosine_similarity permet de savoir les relations (ressemblance ou les contexte) entre les mots ou phrases.

In [4]:
```
bert_preprocess = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3")
bert_encoder = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4")
```

Ces codes utilisent TensorFlow Hub pour charger deux couches pré-entraînées de BERT (Bidirectional Encoder Representations from Transformers), pour le prétraitement et l'encodage de texte en anglais. Le modèle BERT pré-entraîné est capable de capturer des informations sémantiques riches pour une variété de tâches de traitement du langage naturel.

```
# Bert layers

text_input = tf.keras. layers. Input (shape= (), dtype=tf.string, name="text")

preprocessed_text = bert_preprocess (text_input)

outputs = bert_encoder (preprocessed_text)
# Neural network layers
l = tf.keras.layers.Dropout(0.1, name='dropout')(outputs['pooled_output'])
l = tf.keras.layers.Dense(1, activation='sigmoid', name='output', input_shape=(768,))(l)

# construct final model
model = tf.keras.Model(inputs=[text_input], outputs=[l])
```

```
METRICS = [
    tf.keras.metrics.BinaryAccuracy(name='accuracy'),
    tf.keras.metrics.Precision(name='precision'),
    tf. keras.metrics. Recall (name= ' recall')
]

model. compile (optimizer="adam",
                loss='binary_crossentropy',
                metrics=METRICS)
```

Les codes précédents, sont utilisé pour créer un modèle de classification de texte en utilisant le modèle **BERT**. La première ligne de code **text_input = tf. keras.layers.Input(shape=(), dtype=tf.string, name="text")** est utilisée pour définir l'entrée du modèle.

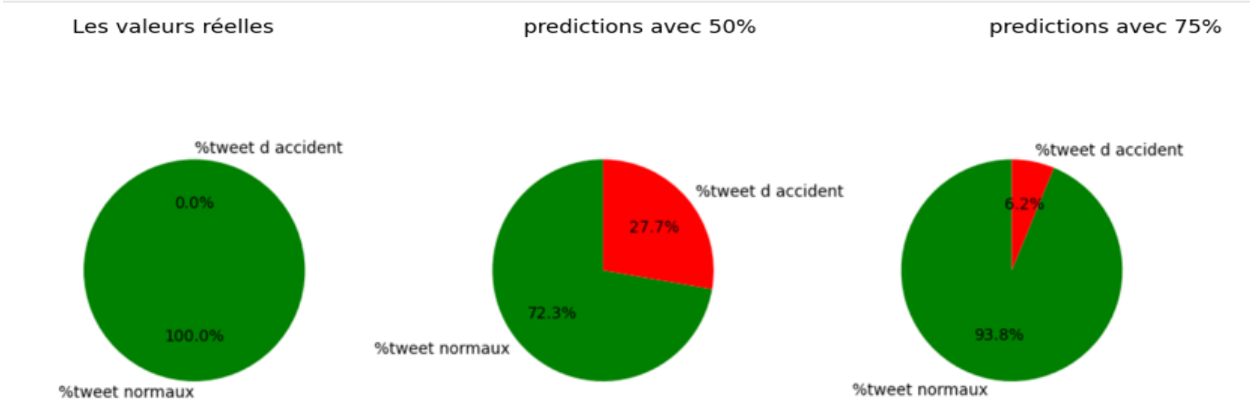
La deuxième ligne de code **preprocessed_text = bert_preprocess(text_input)** est utilisée pour prétraiter le texte d'entrée à l'aide du modèle BERT.

La troisième ligne de code **outputs = bert_encoder(preprocessed_text)** est utilisée pour encoder le texte prétraité à l'aide du modèle BERT.

Les lignes suivantes sont utilisées pour ajouter des couches de réseau neuronal au modèle. La Couche Dropout est utilisée pour éviter le sur apprentissage et la couche Dense est utilisée pour produire une sortie binaire.

Le modèle final est construit en utilisant les entrées et les sorties définies précédemment. Les métriques sont définies pour évaluer les performances du modèle. Le modèle est compilé en utilisant l'optimiseur Adam, la perte binaire croisée et les métriques définies précédemment

Après l'entrainement et l'application du model, aux données test, on a les resultats suivant :




```
In [80]:  y_predicted_075 = np. where(y_predict_f > 0.75, 1, 0)
          y_predicted_050 = np. where(y_predict_f > 0.5, 1, 0)
```

Les valeurs prédit par le modèle, sont en réalité entre 0 à 1 ([0,1]).

Comme on voit plus haut, il y'a une prédiction à 75%, qui veut dire que qu'un tweet est classé urgent donc 1 ; si la valeur prédite par le modèle, est au moins supérieure à 0.75, et celle de 50%, pour un seuil de 0.5.

Cela parait évident qu'il faut choisir un seuil élevé, après tout le model donne des bonnes réponses en comparant au seuil de 50%. Pourtant la réalité est qu'il faut faire attention, en général on utilise le seuil de 50%, mais ça dépend du contexte au quel on travail. Et d'autre part, nos données tests sont tous classés normaux, ce qui peut jouer un effet de biais dans notre model

X-Conclusion

En guise de conclusion, le langage naturel comme autre sujets, tels que la vision ordinatrice, ou l'intelligence artificiel tout court, peuvent être traiter avec plusieurs type d'algorithme qui sont autant diffèrent mais aussi en cours amélioration. On a spécifiquement travaillé sur des algorithmes de classification.

Dans nos modèles, on a fait exprès de travailler avec des modèles simples, sans tenir compte de des complexité qui aurait pu être rajouter : je pense aux features ingénieries possible, et les utiliser pour la modélisation. On aurait pu aussi manipuler les métas paramètre des classificateurs classiques : svm RandomForesclassifier etc. et aussi les métas paramètre de Tensorflow et Bert. Cela est parce que je suis nouveau dans ce domaine et parce que je voulais expliquer la solution en mentionnant les bases des modèles et en étant aussi claire et bref.

BERT et TENSORFLOW sont plus important car ils font intervenir le deep learning. Bert et Tensorflow sont plus efficaces et prometteurs, car ils sont capables de traiter des données non structurées telles que du texte et des images. De plus, ces modèles sont en constante évolution et continuent d'être améliorés pour offrir des performances encore meilleures.

Les perspectives futures pour le traitement du langage naturel sont très prometteuses. Les modèles de deep learning tels que BERT et TensorFlow ont montré des résultats impressionnants dans la classification de texte et la compréhension du langage naturel. Cependant, il reste encore beaucoup à faire pour améliorer ces modèles et les rendre plus efficaces. Par exemple, l'utilisation de techniques telles que l'attention et la mémoire à long terme peut aider à améliorer les performances des modèles de deep learning. De plus, l'utilisation de données supplémentaires telles que des données multimodales (texte, image, audio) peut aider à améliorer la compréhension du langage naturel. Enfin, l'utilisation de techniques telles que le renforcement de l'apprentissage peut aider à améliorer la capacité des modèles à interagir avec leur environnement.

En somme, le domaine de l'apprentissage automatique est un domaine passionnant en constante évolution. Les futures recherches et applications devront tenir compte de l'interprétabilité, de l'apprentissage fédéré, des synergies avec d'autres technologies émergentes(comme la réalité virtuelle) et de la responsabilité éthique. Ces perspectives élargiront l'horizon des possibilités pour résoudre des problèmes complexes et relever les défis de demain avec des solutions d'intelligence artificielle avancées et socialement responsables.