Rapport de Projet : Système Intelligent de Détection de Spams

Table des Matières

1. Introduction

Ce rapport détaille le développement d'un système intelligent de détection de spams à partir d'emails. L'objectif principal est de classifier automatiquement les emails comme "spam" (malveillant) ou "ham" (légitime) en utilisant des techniques de Traitement du Langage Naturel (NLP) et d'apprentissage automatique supervisé. Nous présenterons les étapes clés du projet, de l'analyse des données à l'évaluation du modèle final.

2. Analyse Exploratoire des Données (EDA)

L'EDA est cruciale pour comprendre la structure, les caractéristiques et les patterns de notre jeu de données.

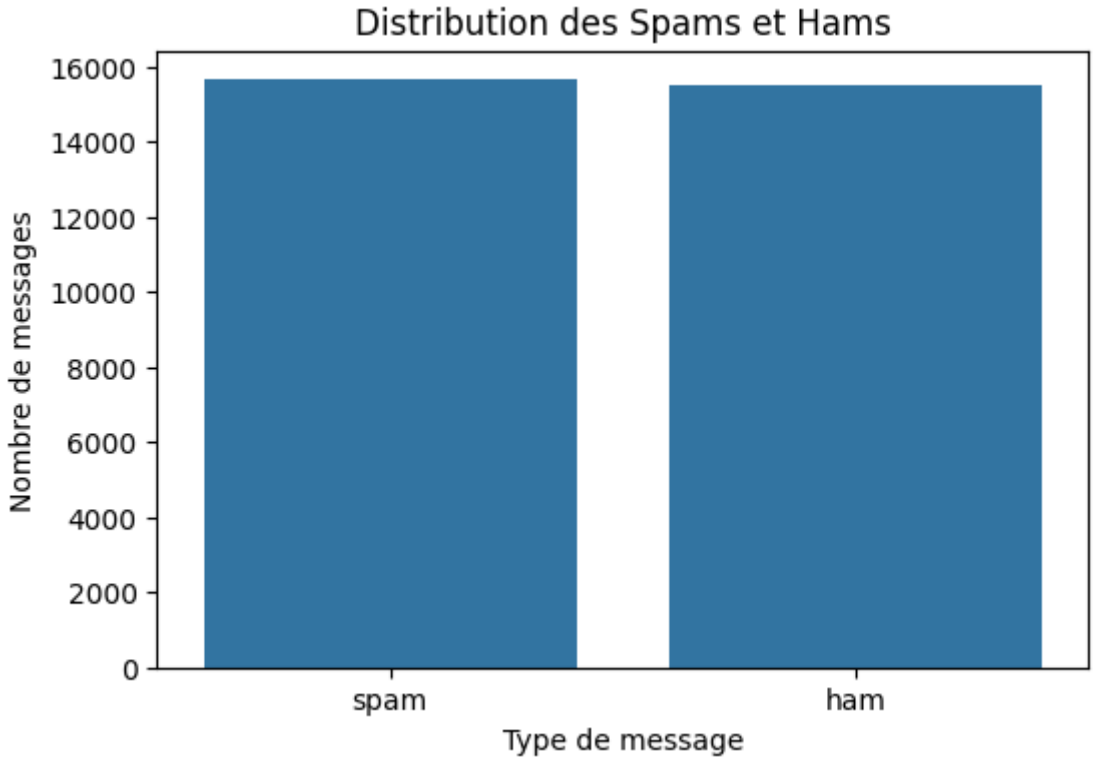
2.1 Aperçu des Données

- **Description du jeu de données :**
  - le nombre de lignes : 31715
  - le nombre de colonnes : 8
  - le type de données de chaque colonne.
    - Unnamed: int64
    - message_id int64
    - text object
    - label int64
    - label_text object
    - subject object
    - message object
    - date object
  - **text (type object):** C'est le corps de l'email, le texte principal que notre modèle analysera.
  - **label (type int64)** et **label_text (type object):** Ces deux colonnes sont nos variables cibles (les étiquettes).
  - **label** est l'étiquette numérique (probablement 0 pour ham et 1 pour spam), tandis que label_text est la version textuelle (ham, spam).
  - **subject (type object)**: Le sujet de l'email. Il y a 274 valeurs manquantes. Le sujet est une information très pertinente pour la détection de spams, donc il serait judicieux de l'intégrer au texte principal de l'email après avoir géré les valeurs manquantes.
  - **date (type object):** La date d'envoi de l'email. Cette colonne est complète. Sauf si vous prévoyez une analyse temporelle, cette variable n'est probablement pas nécessaire pour la détection de spam en elle-même.
- **Gestion des valeurs manquantes et des doublons :**

les valeurs manquantes pour les colonnes:

1. **text** 51 valeurs manquantes (il est mieux de les supprimer puisque ce nombre est négligeable devant 31715)
2. **subject** 274 valeurs manquantes
3. **message** 345 valeurs manquantes

2.2 Distribution des Catégories (Spam vs. Ham)



[le graphique de distribution des labels]

- **Interprétation :**

1. Le graphique montre que notre jeu de données est **parfaitement équilibré** en termes de nombre d'emails de type "spam" et "ham". Il y a un nombre quasiment égal d'exemples dans chaque classe.
2. C'est une excellente nouvelle pour l'apprentissage automatique, car cela signifie que notre modèle ne sera pas biaisé en faveur de l'une des classes. ce qui simplifie le processus d'entraînement.

2.3 Nuages de Mots (Word Clouds)

[le nuage de mots pour les emails HAM]

1. **Interprétation :**
    - les nuages de mots est une excellente façon de visualiser les termes les plus fréquents et d'obtenir une première intuition sur les différences linguistiques entre les spams et les emails légitimes.
    - **Mots fréquents dans le "Ham" :** les mots qui apparaissent le plus souvent dans les emails légitimes (ex: "hi", "know", "need", "hou", "ect"). Cela peut refléter le langage courant des communications non sollicitées.
    - **Mots fréquents dans le "Spam" :** les mots caractéristiques des spams (ex: "make", "money", "company", "email", "one", "thank"). Ces mots sont des indicateurs clés pour le modèle.

3. Prétraitement du Texte

Dans cette partie nous avons décrivis les étapes de prétraitement appliquées aux emails. Et Mentionnons l'ordre et la raison de chaque étape :

- **Conversion en minuscules :** Standardisation du texte.
- **Suppression des chiffres et de la ponctuation :** pour la réduction du bruit et de la complexité.
- **Tokenisation :** Découpage du texte en unités (mots).
- **Suppression des *stopwords* :** Élimination des mots courants qui n'apportent pas de sens discriminatif (ex: "the", "this", "a", "of").
- **Stemming (ou Lemmatisation) :** Réduction des mots à leur racine ou forme canonique (ex: "running", "runs", "ran" -> "run").

4. Vectorisation des Caractéristiques

Une fois que le texte a été nettoyé et standardisé, il doit être converti en un format numérique que les algorithmes d'apprentissage automatique peuvent traiter. Pour ce projet, nous avons utilisé **la méthode TF-IDF** (Term Frequency-Inverse Document Frequency), une technique couramment employée en NLP.

- **Méthode utilisée :** `TfidfVectorizer`
- **Explication de la méthode :** Le **TfidfVectorizer** de la bibliothèque **scikit-learn** a été employé pour transformer la collection de documents (emails) en une matrice de caractéristiques TF-IDF. Chaque ligne de cette matrice représente un email, et chaque colonne correspond à un mot (un "token") unique dans notre vocabulaire. La valeur à l'intersection d'une ligne et d'une colonne indique l'importance de ce mot pour cet email.
- **Taille du vocabulaire :** Nous avons utilisé le paramètre **max_features** pour limiter le nombre de mots du vocabulaire à un seuil défini (dans notre cas nous avons choisi 5000). Cela permet de réduire la dimensionnalité de la matrice de caractéristiques, ce qui rend l'entraînement du modèle plus rapide et peut aider à éviter le surapprentissage.

X (matrice TF-IDF) est définie.

1. Forme de X (Caractéristiques) : (31148, 5000)
2. Forme de y (Cible) : (31148,)

5. Modélisation et Évaluation

Cette section décrit le processus d'entraînement et les performances des modèles.

5.1 Division des Données

- **Méthode :** `train_test_split`.
- **Proportion :** La proportion utilisée pour l'ensemble de test (est 20%).
- **Stratification :** La stratification est le processus qui consiste à diviser les données de manière à ce que chaque sous-ensemble (l'ensemble d'entraînement et l'ensemble de test) ait la même proportion de classes que le jeu de données d'origine.
- **Exemple :** Pour un jeu de données de détection de spam :
- Si 90% des emails sont des "ham" et 10% sont des "spam" dans le jeu de données total. Alors stratify=y garantit que l'ensemble d'entraînement et l'ensemble de test contiendront également environ 90% de "ham" et 10% de "spam".

  Forme de X_train : (24918, 5000)
  Forme de X_test : (6230, 5000)
  Forme de y_train : (24918,)
  Forme de y_test : (6230,)

  Distribution des classes dans y_train:
  label_text
  spam 0.502288
  ham 0.497712
  Name: proportion, dtype: float64

  Distribution des classes dans y_test:
  label_text
  spam 0.502247
  ham 0.497753
  Name: proportion, dtype: float64

5.2 Algorithmes Testés

La liste des algorithmes de classification que nous avons expérimentés :

- **Decision Tree Classifier**
- **Naïve Bayes Classifier (MultinomialNB)**
- **Support Vector Classifier (SVC)**

  Récapitulatif des précisions :
  - Decision Tree: 0.9547
  - Naïve Bayes (Multinomial): 0.9831
  - SVC (Linear): 0.9891

5.3 Validation Croisée et Métriques

- **Méthode de validation croisée :** `StratifiedKFold` (nombre de folds). Pour évaluer la robustesse et la généralisation de nos modèles, nous avons utilisé la validation croisée k-fold. Plutôt que de diviser les données une seule fois en un ensemble d'entraînement et de test, cette technique divise le jeu de données en k parties (ou "folds"). Le modèle est ensuite entraîné k fois, chaque fois sur k−1 folds, et testé sur le fold restant. La performance finale est la moyenne des scores obtenus sur les k itérations, ce qui donne une estimation plus fiable.
- **Métriques d'évaluation :**
    - **Précision (Accuracy) :** Taux de bonnes prédictions.
    - **Précision (Precision) :** Vrais positifs / (Vrais positifs + Faux positifs). Utile pour éviter de classer des hams comme des spams.
    - **Rappel (Recall) :** Vrais positifs / (Vrais positifs + Faux négatifs). Utile pour détecter le plus de spams possible.
    - **Score F1 (F1-score) :** Moyenne harmonique de la précision et du rappel. Bon indicateur pour les classes déséquilibrées.
    - **ROC AUC (Area Under the Receiver Operating Characteristic Curve) :** Mesure la capacité du modèle à distinguer les classes.
- **Tableau des Résultats de Validation Croisée :** un tableau récapitulatif des performances moyennes (et écart-type) pour chaque modèle et chaque métrique.

Modèle	Précision Moyenne	Précision (weighted) Moyenne	Rappel (weighted) Moyenne	F1-Score (weighted) Moyenne	ROC AUC Moyenne
Decision Tree Classifier	0.9612 (± 0.0018)	0.9613 (± 0.0018)	0.9612 (± 0.0018)	0.9612 (± 0.0018)	0.9630 (± 0.0017)
Naïve Bayes (MultinomialNB)	0.9821 (± 0.0018)	0.9822 (± 0.0018)	0.9821 (± 0.0018)	0.9821 (± 0.0018)	0.9977 (± 0.0002)
SVC (Linear)	0.9889 (±0.0008)	0.9889 (±0.008)	0.9889 (±0.0008)	0.9889 (±0.0008)	0.9989 (±0.0003)

- **Interprétation :**

1. **Support Vector Classifier (SVC Linear)** : Le SVC se positionne comme le meilleur modèle de l'ensemble. Avec une précision moyenne de 98,89% et un score ROC AUC de 0,9989, il surpasse légèrement le Naïve Bayes. Ses scores de précision, de rappel et de F1-score sont également les plus élevés, ce qui indique une excellente capacité à correctement classer les spams et les hams.
2. **Support Vector Classifier (SVC) :** C'est le modèle le plus stable. Un écart-type de seulement 0,0008 pour la précision et de 0,0003 pour le ROC AUC indique que la performance du SVC a très peu varié d'un fold à l'autre. Il est donc très fiable.

6. Optimisation des Hyperparamètres

Décrivez le processus d'optimisation des hyperparamètres pour affiner les performances des modèles sélectionnés.

- **Méthode :** `GridSearchCV`.
- **Modèles optimisés :** Naïve Bayes et SVC.
- **Hyperparamètres testés :**
  - **Naïve Bayes :** `alpha` (`[0.01, 0.1, 0.5, 1.0, 5.0, 10.0]`).
    - **Alpha :** Paramètre de lissage de Laplace
  - **SVC :** `C` (`[0.1, 1, 10]`), `kernel` (`['linear']`).
    - C : Le compromis entre une marge large et une classification correcte
    - Nous nous concentrons sur le noyau linéaire, souvent efficace pour le texte
- **Métriques d'optimisation :** la métrique utilisée par `GridSearchCV` pour sélectionner la meilleure combinaison (`f1_weighted`).
- **Résultats de l'optimisation :**
  - **Naïve Bayes :**
    - Meilleurs hyperparamètres : `{ 'alpha': 0.01 }`
    - Meilleur score F1 (pondéré) : `0.9826`
  - **SVC :**
    - Meilleurs hyperparamètres : `{ 'C': 1 , 'kernel': 'linear' }`
    - Meilleur score F1 (pondéré) : `0.9889`
- **Interprétation :**

  --- Comparaison des scores des modèles optimisés ---
  - Meilleur score pour Naïve Bayes : 0.9836
  - Meilleur score pour SVC : 0.9897

  Le modèle le plus performant est SVC.

7. Résumé des résultats

Ce projet a permis de développer un système de détection de spams robuste et performant. Après une analyse exploratoire des données, un prétraitement minutieux et une vectorisation via TF-IDF, nous avons entraîné et évalué plusieurs modèles de classification. Le modèle Support Vector Classifier (SVC) s'est révélé être le plus performant, avec des scores de validation croisée exceptionnels. Sa précision moyenne de 98,89% et son score ROC AUC de 0,9989, combinés à un écart-type très faible, confirment sa fiabilité et sa capacité à généraliser sur des emails non vus.