École Centrale Casablanca

Rapport Projet Deep Learning

Élèves :
Zineb Aissaoui
Ilyass El Fourati

Enseignant:
Vincent Lefieux

22 décembre 2023

Table des matières

1 Problème de classificaton : SPAM

1.1 Préparation des données

Après l'aquisition des données à partir du fichier 'spam.csv'en utilisant la fonction 'read_csv' de la librairie Pandas pour lire les données. Nous avons commencé la préparation de nos données avant l'entraînement des modèles :

1.1.1 Contrôle des valeurs nulles et aberrantes

Après avoir vérifié les données et confirmé l'absence de valeurs manquantes, il est important de noter que les fréquences de mots dans les e-mails présentent souvent des valeurs aberrantes prévisibles. Cette particularité découle de la diversité du langage utilisé : certains mots sont fréquents dans certains e-mails, tandis qu'ils sont absents ou peu fréquents dans d'autres.

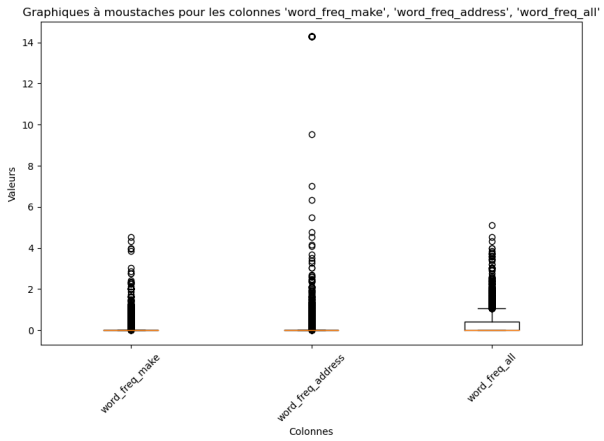


FIGURE 1 – Graphique à moustache

1.1.2 Etude des corrélations

Pour étudier les corrélations, nous allons générer la matrice de corrélation en utilisant 'correlation_matrix = df_spam.corr()' et ensuite nous allons représenter cette matrice sous forme de 'heatmap'..

Nous pouvons observer que certaines variables sont particulièrement corrélées, surtout au centre de la heatmap.

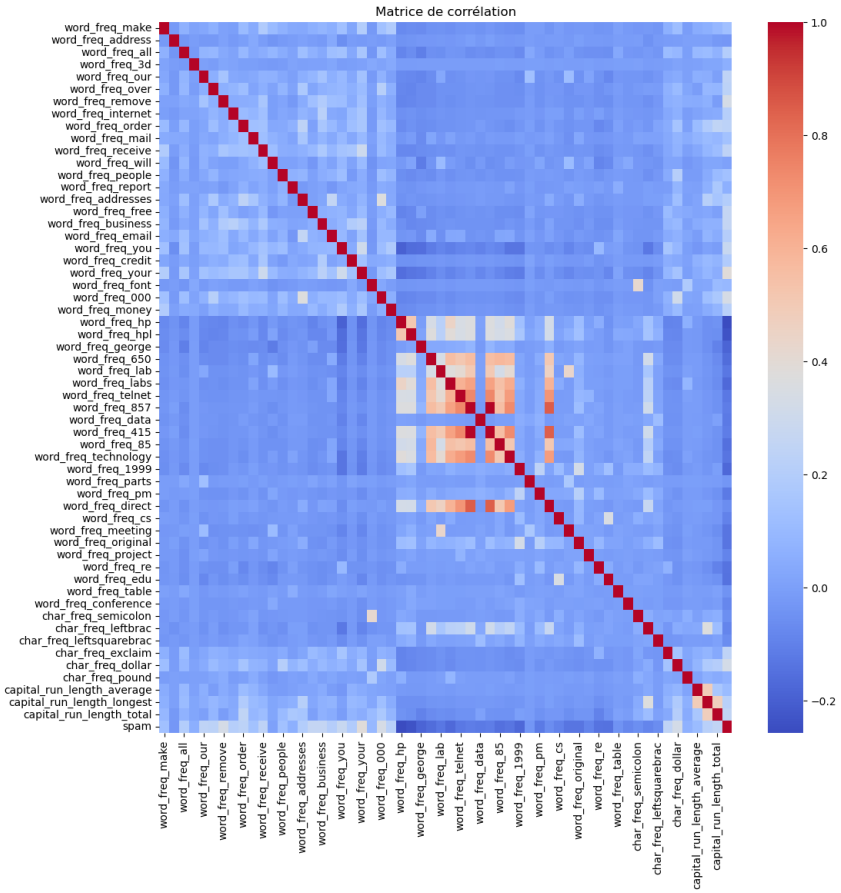


FIGURE 2 – Heatmap

1.2 Régression logistique

1.2.1 Implémentation

Nous avons employé GridSearchCV pour identifier la meilleure combinaison d'hyperparamètres pour notre modèle de Régression logistique. Les résultats de cette recherche ont démontré que les paramètres optimaux sont les suivants :

- **Penalty** : L1.
- **L'hyperparamètre C** : 10.
- **max_iter** : 500.

1.2.2 Analyse des Performances du Modèle

Dans cette section, nous examinerons de près les performances de notre modèle de régression logistique pour classer les e-mails en spam ou non-spam. L'évaluation de l'efficacité de notre modèle se fera à l'aide de plusieurs métriques clés, notamment l'exactitude (accuracy), la précision, le rappel et la matrice de confusion.

Avant l'utilisation GridSearchCV :

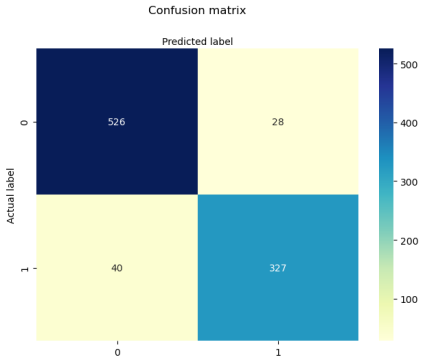


FIGURE 3 – Matrice de Confusion - régression logistique

Nous avons choisit les paramètres : penalty='l2', max_iter=1000, solver='lbfgs'et C=0.1. Avec ce premier modèle nous avons obtenu une accuracy égale à 92,5%

- **Accuracy ou Exactitude** : 92,6%
- **Précision** : 89,1%
- **Recall ou Rappel** : 92,1% .
- **F1 score** : 90,5%.

Après l'utilisation GridSearchCV

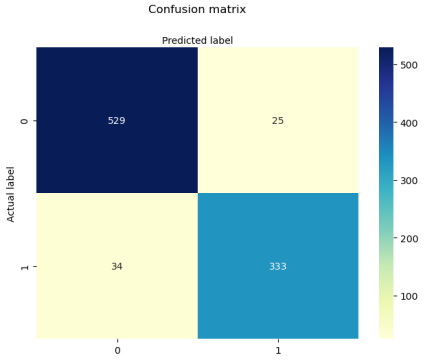


FIGURE 4 – Matrice de Confusion - régression logistique (GridSearchCV)

- **Accuracy ou Exactitude** : Le modèle nous donne une exactitude de 93,59%
- **Précision** : 90,7%
- **Recall ou Rappel** : 93%
- **F1 score** :91,8%.

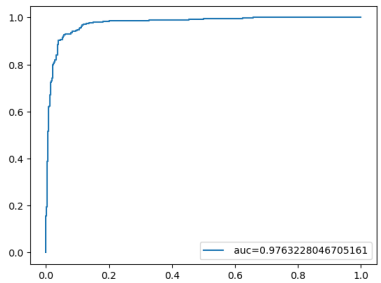


FIGURE 5 – Courbe ROC - régression logistique

- **Courbe ROC du modèle** : Nous avons tracé la courbe ROC du modèle sur l'ensemble de test et calculé la surface sous la courbe qui est égale à 0,976. Cette valeur est très élevée. Cela indique que le modèle a une très bonne capacité à classifier correctement les échantillons positifs et négatifs.

1.3 Gradient Boosting

1.3.1 Implémentation

Nous avons utilisé le modèle de boosting par gradient (GradientBoostingClassifier) fourni par la bibliothèque scikit-learn, en particulier à travers la classe sklearn.ensemble. Nous avons testé d'entraîner le modèle sur X_train origine, X_train_normalized qui

correspon à X_train normalisé et X_train_PCA qui correspon à X_train après avoir appliqué une ACP.

1.3.2 Analyse des Performances du Modèle

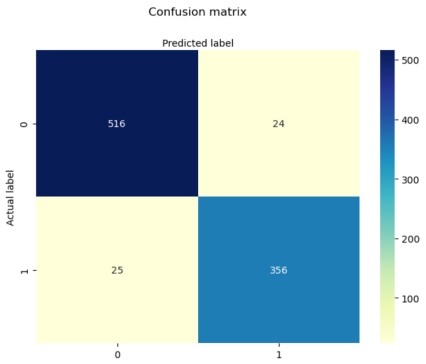


FIGURE 6 – Matrice de Confusion : Gradient Boosting

Pour évaluer ce modèle nous avons pris comme critères : l'exactitude, la précision, le rappel et la matrice de confusion.

- **Exactitude** : L'exactitude de ce modèle est de 95%
- **Précision** : 95%
- **Rappel** : 95%
- **Matrice de confusion** :

Les approches d'entraînement sur des données normalisées ainsi que sur des données avec une dimension réduite ont conduit à des performances très médiocres en termes de précision. Par conséquent, nous avons exclu ces modèles.

1.4 Réseau de neurones

1.4.1 Implémentation

Pour mettre en œuvre ce modèle, nous avons utilisé Keras, une interface de programmation d'applications (API) de la bibliothèque TensorFlow.

Nous avons mis en œuvre trois modèles et avons sélectionné le plus performant parmi eux.

- **Modèle 1 :** Nous avons manuellement optimisé les hyperparamètres de ce modèle.
- **Modèle 2 :** Nous avons utilisé Keras Tuner pour ajuster de manière optimale les hyperparamètres, en mettant particulièrement l'accent sur le nombre de neurones dans chaque couche et le taux d'apprentissage.
- **Modèle 3 :** Similaire au modèle 2, à l'exception que dans ce cas, nous avons optimisé le nombre de couches ainsi que le nombre de neurones dans chacune de ces couches.

TABLE 1 – Détails des modèles

Modèle	Modèle 1	Modèle 2	Modèle 3
Nombre de Neurones (couche1)	100	190	250
Nombre de Neurones (couche2)	100	120	200
Learning Rate	Par défaut	0.01	0.05
Accuracy	94.1%	94.4%	94.8%

D'après le tableau, le meilleur modèle est le modèle 3.

1.4.2 Analyse des Performances du Modèle

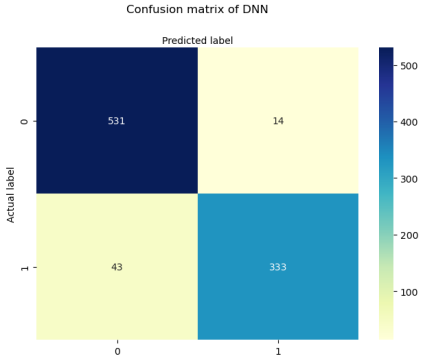


FIGURE 7 – Matrice de Confusion

Dans cette section, nous présenterons les scores obtenus par le modèle optimal.

- **Exactitude :** 95,33%
- **Précision :** 85,56%
- **Rappel :** 92,88%

1.5 Conclusion

En conclusion, les modèles ont produit des résultats satisfaisants. Nous avons entrepris une comparaison pour déterminer le modèle victorieux, celui affichant les meilleures performances. Les valeurs sont légèrement différentes car nous avons réparti à nouveau les données en ensembles d'entraînement et de test, en raison d'un paramètre aléatoire ('Random'). Cela a entraîné des modifications dans les ensembles X_test et y_test.

Modèle	Gradient Boosting	Régression Logistique	Réseau de neurones
Exactitude	0.97	0.93	0.94
Précision	0.95	0.88	0.90
Rappel	0.96	0.94	0.95
F_1	0.95	0.91	0.93

Ainsi, nous pouvons conclure que le **gradient boosting** se distingue en tant que modèle le plus performant, affichant les scores les plus élevés dans pratiquement toutes les métriques évaluées.

2 Problème de régression : Ozone

2.1 Préparation des données

Avant d'initier la création des modèles visant à prévoir le pic quotidien d'ozone pour le lendemain, nous avons effectué un prétraitement des données afin de les rendre plus significatives.

- **Transformation des types de données :** Nous avons modifié les types de données pour les convertir en float, et pour la colonne date, nous l'avons transformée en type datetime.
- **Gestion des valeurs aberrantes :** Lors de l'inspection du graphique en boîte à moustaches, nous avons constaté que les valeurs aberrantes étaient normales et explicables.

— **Colonne cible :** Dans nos données initiales, la colonne du pic d'ozone pour le lendemain était absente. Pour remédier à cela, nous avons ajouté cette colonne en nous basant sur la ligne suivante à celle pour laquelle nous souhaitions ajouter des informations à la nouvelle colonne.

— **Gestion des valeurs manquantes :** Les lignes présentant des valeurs manquantes pour la colonne cible ont été supprimées. Pour les autres colonnes, nous avons remplacé les valeurs manquantes par la moyenne de la colonne.

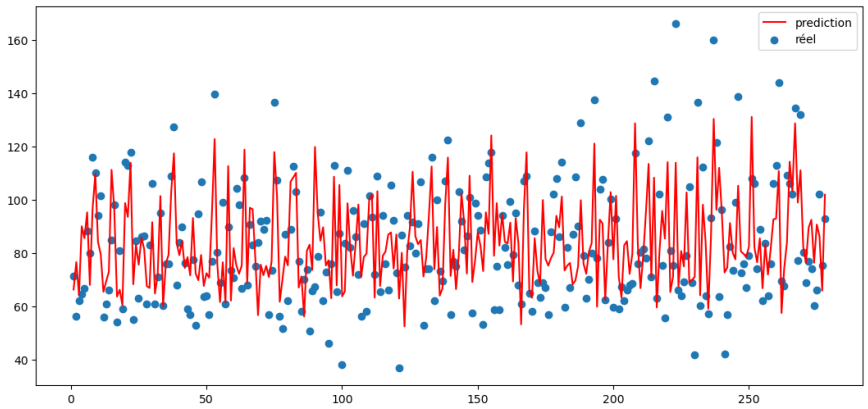
— **Suppression de la colonne date :** Étant donné que nos modèles ne tolèrent pas les colonnes de type date, nous avons retiré cette colonne et prévoyons de la remplacer par des indices.

— **Normalisation des données :** Nous avons normalisé les données (sauf la colonne cible) en utilisant la méthode StandardScaler de la bibliothèque sklearn.preprocessing.

2.2 Les modèles

2.2.1 Régression linéaire

Nous avons utilisé le modèle **LinearRegression** (LR) de sklearn.linear_model. Ce modèle nous a donné les résultats suivant en terme de performance :



Modèle	RL
RMSE	15.89
MAE	12.81
R^2 Score	0.49

FIGURE 8 – y predit par LR sur les données test et y réelle

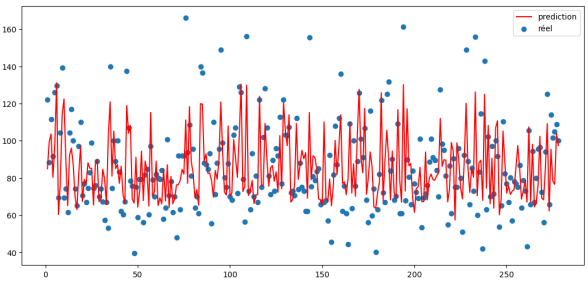
2.2.2 Random Forest

Nous avons utilisé **RandomForestRegressor** (RF) de sklearn.ensemble. Nous avons utilisé GridSearchCV pour optimiser les paramètres. Le modèle qui nous a donnée les paramètres du meilleur modèle en fonction de la mae est : **max depth** =10,**n estimators** =250, **min samples leaf**=4 , **min samples split**=2, **random state** =42.

2.2.3 DNN

Pour le modèle DNN, nous avons suivi les mêmes étapes qu'avec les données 'spam'. Cependant, les résultats n'ont pas été très performants, donc nous avons conservé uniquement le modèle optimisé manuellement.

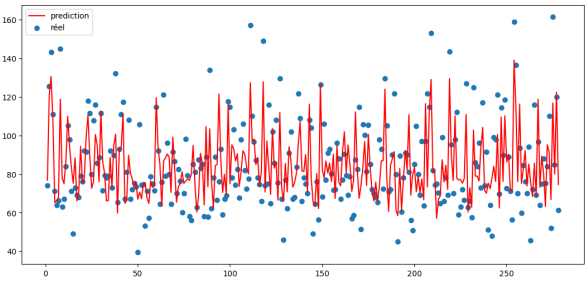
Constitution :



Modèle	Random Forest
RMSE	15.36
MAE	11.79
R^2 Score	0.57

FIGURE 9 – y predit par RF sur les données test et y réelle

— **Couche 1 :** constitué de 64 neuronnes.
— dropout avec une valeur de 0.3 pour réduire le surajustement.
— **Couche 2 :** constitué de 32 neuronnes.
— **learning rate :** est de 0.01.



— RMSE :15.15
— MAE :11.61
— R^2 Score :0.55

FIGURE 10 – y predit par DNN sur les données test et y réelle

2.3 Conclusion

En résumé, les performances des modèles se sont avérées satisfaisantes. Une analyse comparative a été menée pour identifier le modèle victorieux, celui qui a manifesté la meilleure efficacité.

Modèle	Random Forest	Régression Linéaire	Réseau de neurones
MAE	11.79	12.81	11.61
RMSE	15.36	15.89	15.15
R^2 Score	0.57	0.49	0.55

En se basant sur les données fournies dans le tableau précédent, on peut tirer la conclusion que le **réseau de neurones** se distingue en tant que modèle le plus performant, affichant des scores notablement élevés dans la plupart des métriques évaluées.