Projet 7 : AIR PARADIS

Challenge : Détectez les Bad Buzz grâce au Deep Learning

https://github.com/blanchonnicolas/IA_Project7_Openclassrooms_IA_SentimentAnalysis

Agenda

01

Classifications par apprentissage

Régression Logistique
Decision Tree
Gradient Boosting

02

Réseau de Neurones

1DConvNet
Bi-LSTM
Transfer Learning Glove
Transfer Learning Fasttext

03

Modèles BERT

AutoModel (BERT)
RoBERTa
CamemBERT

04

Déploiement

FastAPI

(Blog)

Descriptif du contexte projet

Détecter les sentiments

Créer un produit IA permettant d'anticiper les bad buzz sur les réseaux sociaux (et plus spécifiquement Tweeter), en prédisant le sentiment associé à un tweet.

Différences entre les modèles

Justifier les choix des modèles sur leurs performances, ainsi que la complexité de mise en œuvre et de maintenance (Sur étagère, Customisés, From scratch).

Détectez les Bad Buzz grâce au Deep Learning

Air Paradis veut un prototype d'un produit IA permettant de prédire le sentiment associé à un tweet

Préparer un prototype de modèle : Envoie un tweet et récupération de la prédiction de sentiment (positif ou négatif), incluant le Traitement du texte et l'Analyse de la performance.

Pas de données clients chez Air Paradis.
Utilisation de données Open Source provenant de tweeter.

3 Approches comparatives:

- Simple : Modèle de classification de texte
- Avancé: Modèle de classification basé sur des réseaux de neurones
- BERT: Bidirectional Encoder Representations from Transformers

Back to AGENDA

Préparation

Présentation du jeu de données

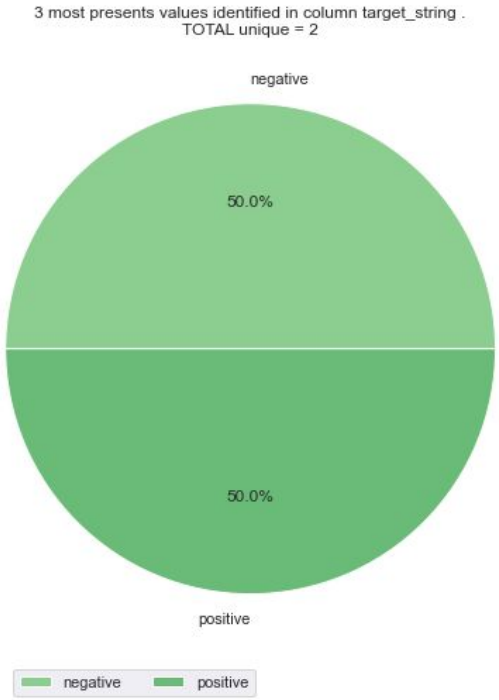
Compréhension du jeu de données

Sentiment140 dataset with 1.6 million tweets

- 227 Mo
- 2 champs intéressants (text et target)
- Distribution équilibrée (50/50%)
- Longueur des champs traitée de 30 Mots

Séparation du jeu d'entraînement et du jeu de test (80/20%).

Réduction du jeu de donnée durant les phases de Dev



Présentation de la démarche

Démarche du Projet

Prototypage

Déploiement d'une solution prototype permettant de recueillir les premiers commentaires utilisateurs

Text Preprocessing

Comparaison de différentes méthodes de preprocessing de texte, afin de le nettoyer et d'uniformiser pour des réutilisations futures

Article de Blog explicatif

Présentation des approches de modélisation et Guide utilisateur du prototype.

Modèles de classification

Comparaisons des approches de modélisation:

- Architecture : Modèle de classification simple vs Deep Learning vs BERT
- Techniques de plongements & Transfer Learning
- Critères d'évaluations

Pré-traitement du texte

Text Preprocessing - Typique NLP

Special Character Cleaning
StopWords Cleaning

2 Options:
- Stemmatisation
- Lemmatisation

consolidation chaîne de texte

3 Jeux de données disponibles, contenant les chaînes de texte recomposées:
- Avec StopWords seul
- Avec Stop_words + Stemmatisation
- Avec Stop_words + Lemmatisation

Notes:
- *Caractères spéciaux = URLs + @mention*
  @\S+|https?:\S+|http?:\S|[^A-Za-z0-9]+

Text Preprocessing - Librairie pour Tweet

Tweet Preprocessor Library

4 Options:

- Avec Text preprocess simple: StopWords ou Stem ou Lem
- Sans Text preprocess

consolidation chaîne de texte

“All in One” Library specifically developed for tweet preprocessing, used to clean, parse or tokenize the tweets:

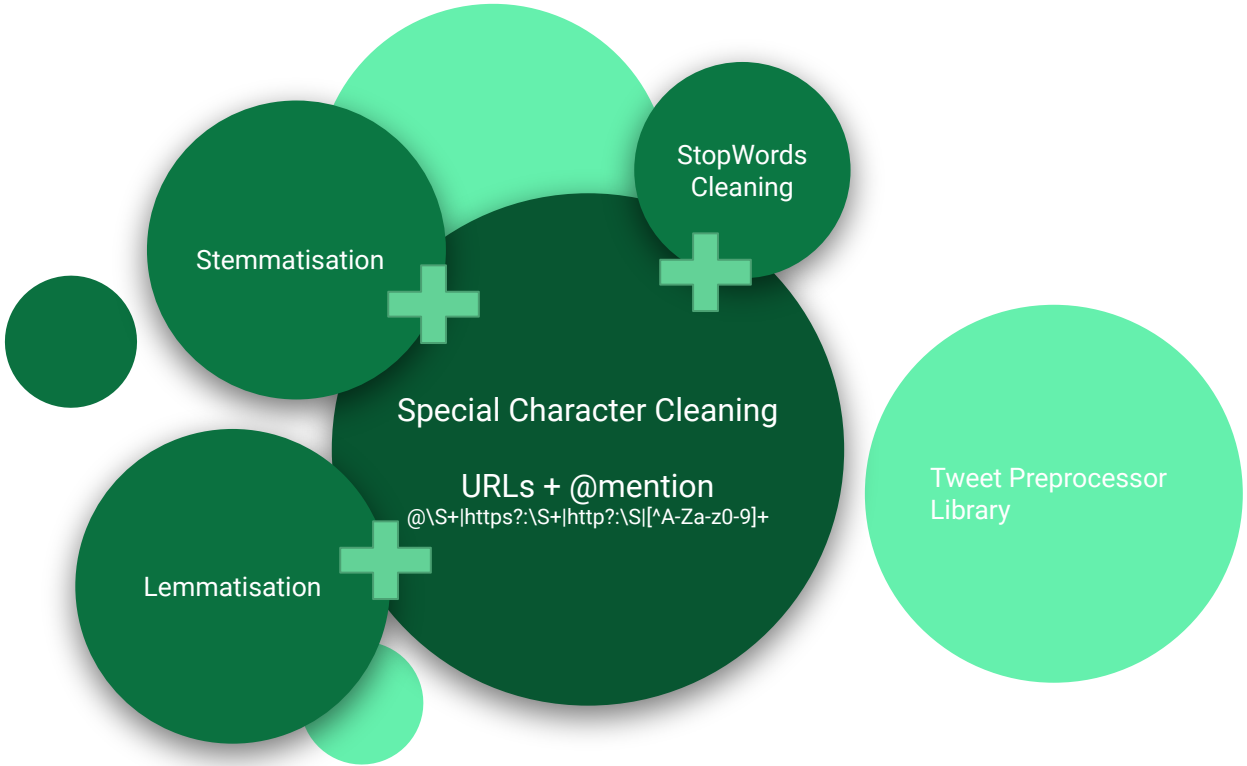
- URLs
- Hashtags
- Mentions
- Reserved words (RT, FAV)
- Emojis
- Smileys

Text Preprocessing

4 Jeux de données présentés, contenant les chaînes de texte recomposées:

- StopWords seul
- Stop_words + Stemmatisation
- Stop_words + Lemmatisation
- Tweet Preprocess

<u>Notes:</u> Plusieurs combinaisons réalisées, non présentées dans cette présentation.



Back to AGENDA

Classifications par apprentissage

Modèles de Classification “simples”

Modèles sur mesure “Simple”

Régression Logistique

Matrice de Confusion

Mesures de l’Accuracy, ROC_AUC, …

01

Decision Tree

Matrice de Confusion

Mesures de l’Accuracy, ROC_AUC, …

02

Gradient Boosting

Matrice de Confusion

Mesures de l’Accuracy, ROC_AUC, …

03

Nous mesurons spécifiquement l’Accuracy, ce qui nous permet de valider l’efficacité de nos modèles à prédire correctement à la fois les sentiments positifs et négatifs, à partir du jeu de données équilibré.

Preprocess de Texte

Nous repartons des 4 jeux de données pré-processées

Séparation Train, Test

Nous séparons nos jeux d'entraînements

Bag Of Words : TFIDF

Nous pondérons les mots en fonction de leur fréquence et singularité (min_df=0.005,max_df=0.8)

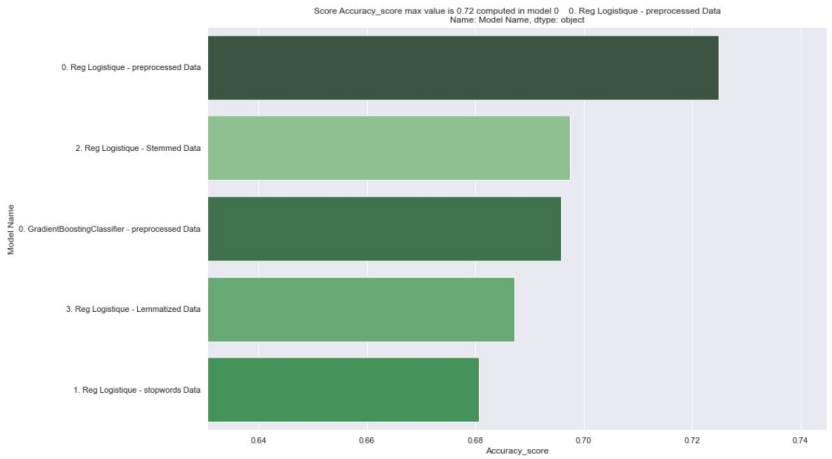
Entrainement du modèle

Nous entraînons nos 3 modèles supervisés, sur la base des 4 jeux de données d'entraînement

Evaluation du modèle

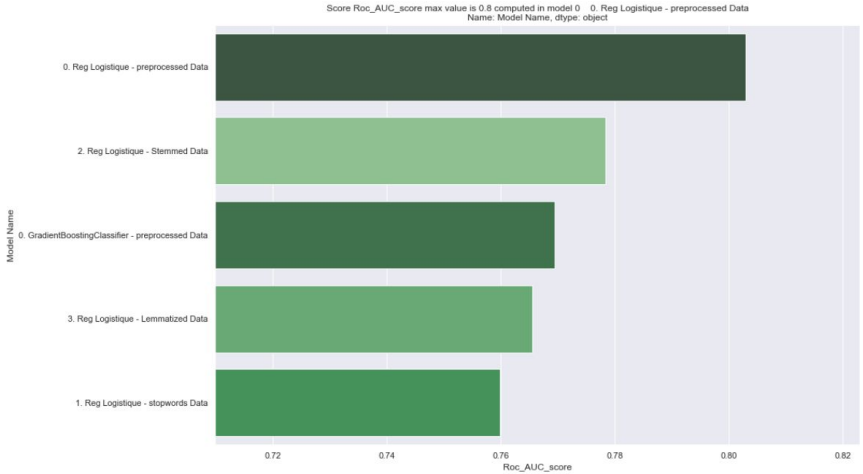
Nous mesurons la capacité de prédiction des différents modèles entraînés

Modèles sur mesure “Simple” : Top 5 Résultats



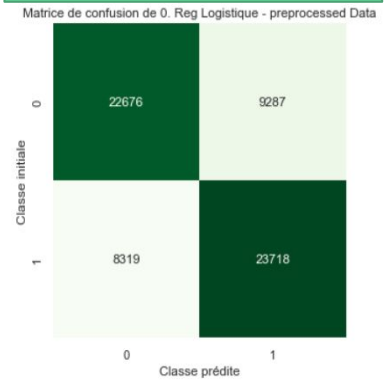
Max Accuracy = 0.7249
Le modèle de Reg Log est le plus performant dans ce cadre (4 sur le Top 5).

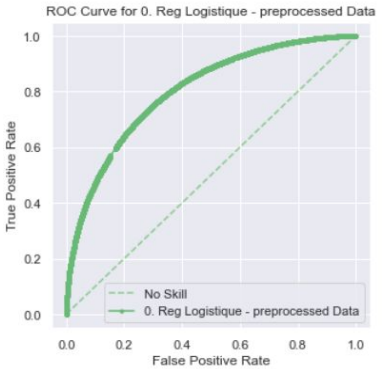
Max ROC_AUC = 0.8031
Les modèles ayant pré-traités les données avec *Tweet Preprocessor* sont plus performant que ceux avec NLP classiques.



Modèles sur mesure “Simple” : Best and Worst Baseline

Meilleur Modèle “Simple”

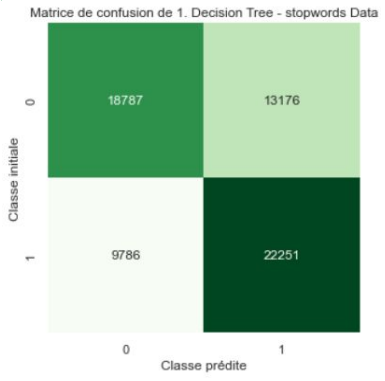




Nativement, le modèle Reg Log permet de réduire considérablement le taux de faux positifs, lorsque les données sont correctement pré-traitées.

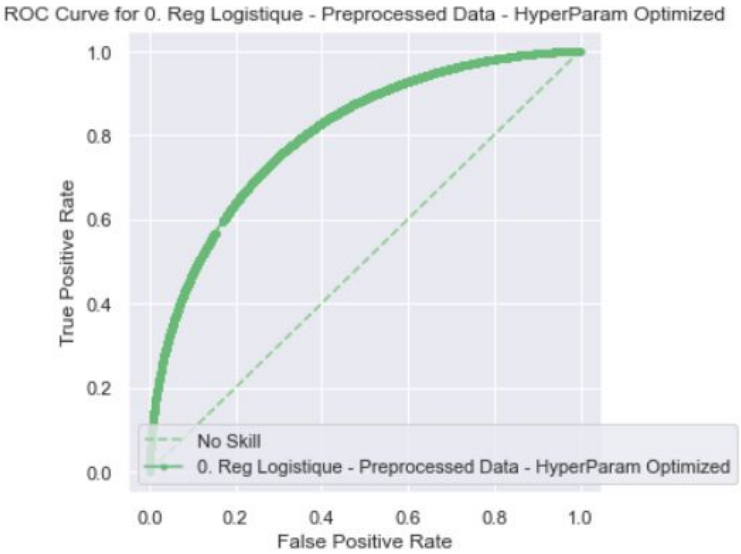
En revanche, le Decision Tree avec suppression des stopwords détecte très mal les sentiments négatifs

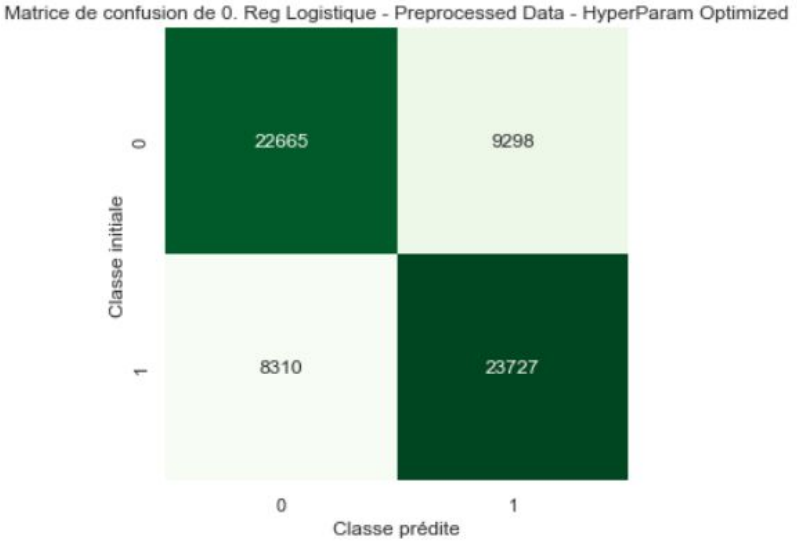
Pire Modèle “Simple”



Choix du modèle “simple” et Optimisation

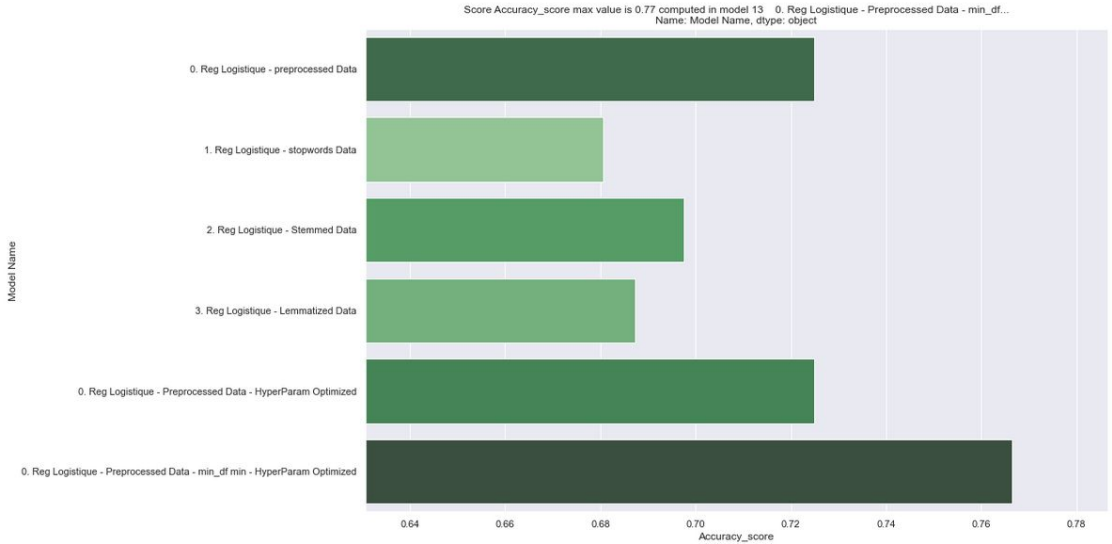
Optimisation des Hyperparamètres

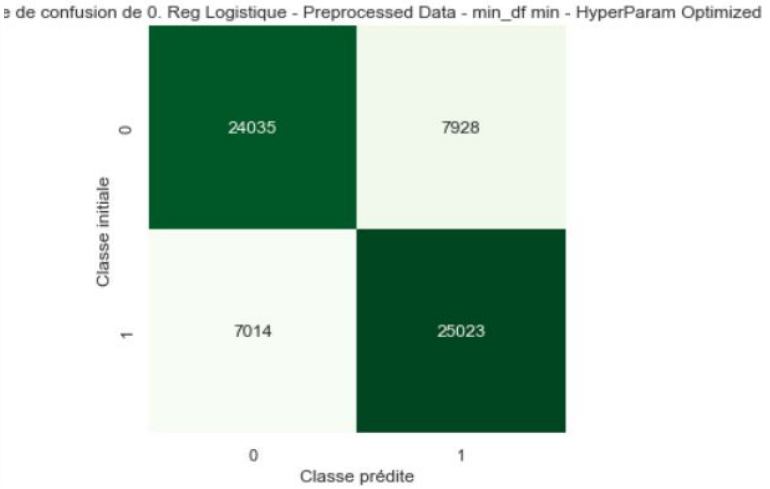




Nous avons effectué un gridsearchcv sur la régression logistique, afin d'optimiser les hyper-paramètres de notre modèle. Ceci n'a pas d'effet significatif sur la performance du modèle.

Optimisation du BoW





Nous avons diminué le min_df utilisé lors de la vectorisation des mots, ceci afin d'exclure moins de mots ayant une fréquence faible. Ceci améliore grandement les performances, comme visualisés dans le graphique résumant l'ensemble des performances des reg Log.

Back to AGENDA

Réseau de Neurones

Modèles de Classification “avancés”

Modèles sur mesure “Avancé”

1DConvNet

Matrice de Confusion

Mesures de l’Accuracy, ROC_AUC, …

01

Bi-LSTM

Matrice de Confusion

Mesures de l’Accuracy, ROC_AUC, …

02

Transfer Learning Glove

Matrice de Confusion

Mesures de l’Accuracy, ROC_AUC, …

03

Transfer Learning Fasttext

Matrice de Confusion

Mesures de l’Accuracy, ROC_AUC, …

04

Nous utilisons les librairies Keras de Tensorflow pour définir 2 architectures distinctes.
Nous sélectionnons la plus performante, et utilisons 2 embedding pré-entraînées pour améliorer nos performances.

Preprocess

Nous pré-traitons le texte et réalisons le Padding des séquences

Architecture

Nous définissons l’architecture RNN, et configurons nos plongements

Compilation

Nous configurons le processus d’apprentissage (adam, loss, accuracy)

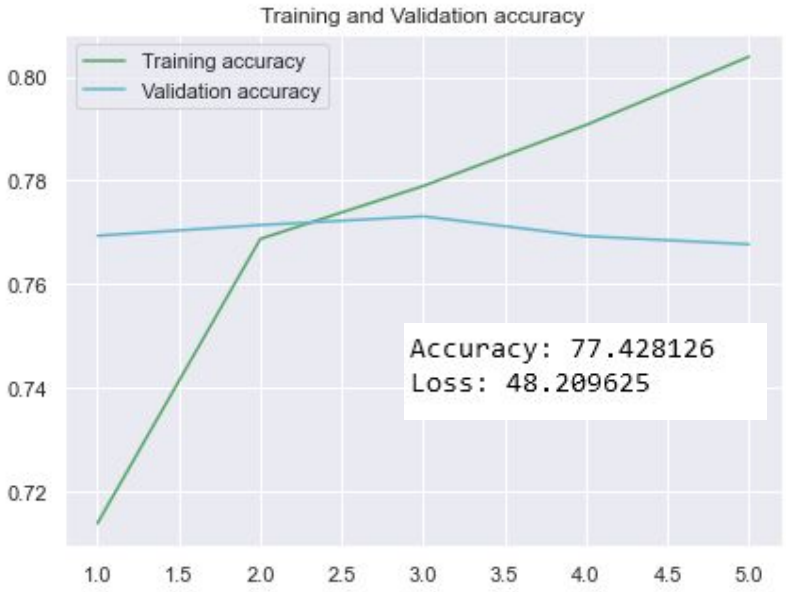
Entrainement

Nous entraînons nos 4 modèles supervisés, sur la base du training dataset

Evaluation

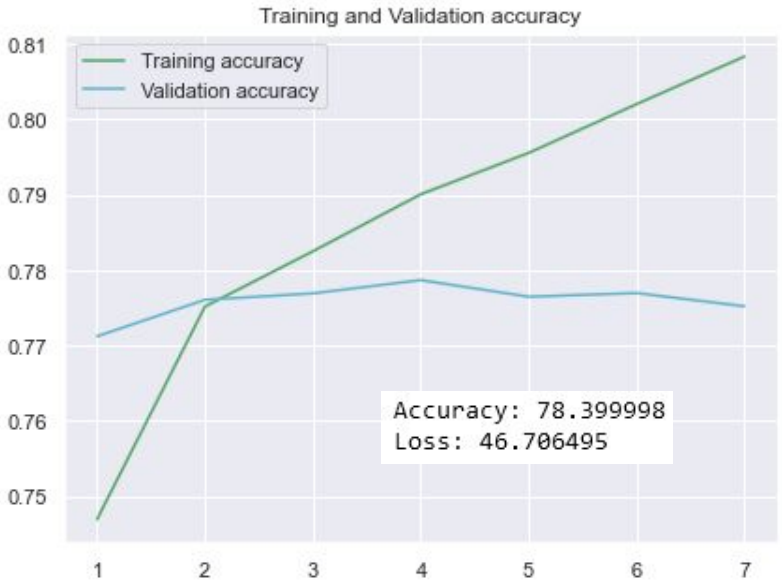
Nous mesurons la capacité de prédiction des différents modèles entraînés

Evaluation du modèle 1DConvNet



Risque de sur-apprentissage avéré. Utilité des callbacks pour stopper le processus !
L'accuracy et Loss évoluent normalement sur le training set, mais pas sur le validation set.

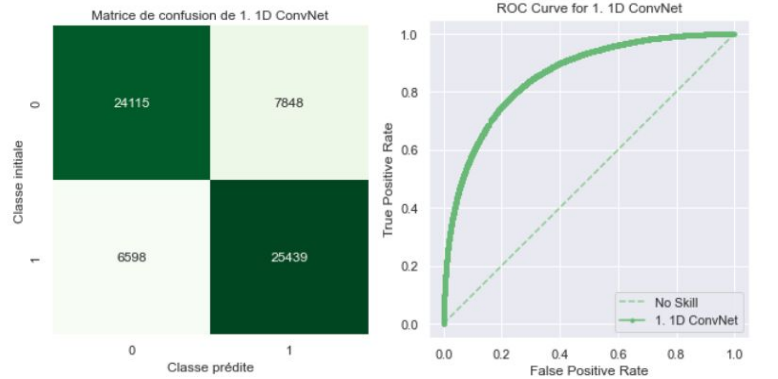
Evaluation du modèle Bi-LSTM



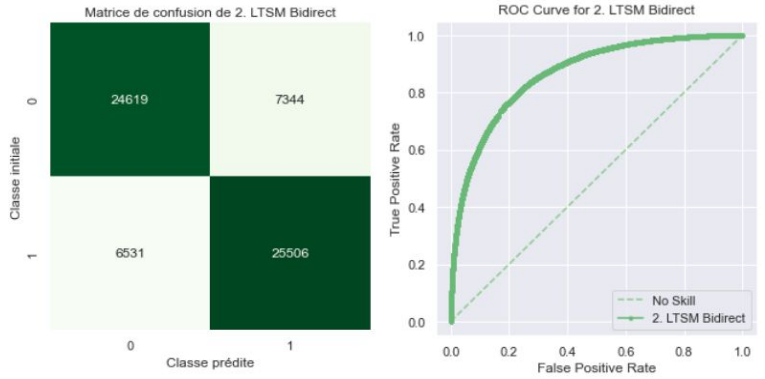
Risque de sur-apprentissage avéré. Utilité des callbacks pour stopper le processus !
L'accuracy et Loss évoluent normalement sur le training set, mais pas sur le validation set.

Comparaison des modèles sur mesure “Avancé”

1D ConvNet



Bi-LSTM



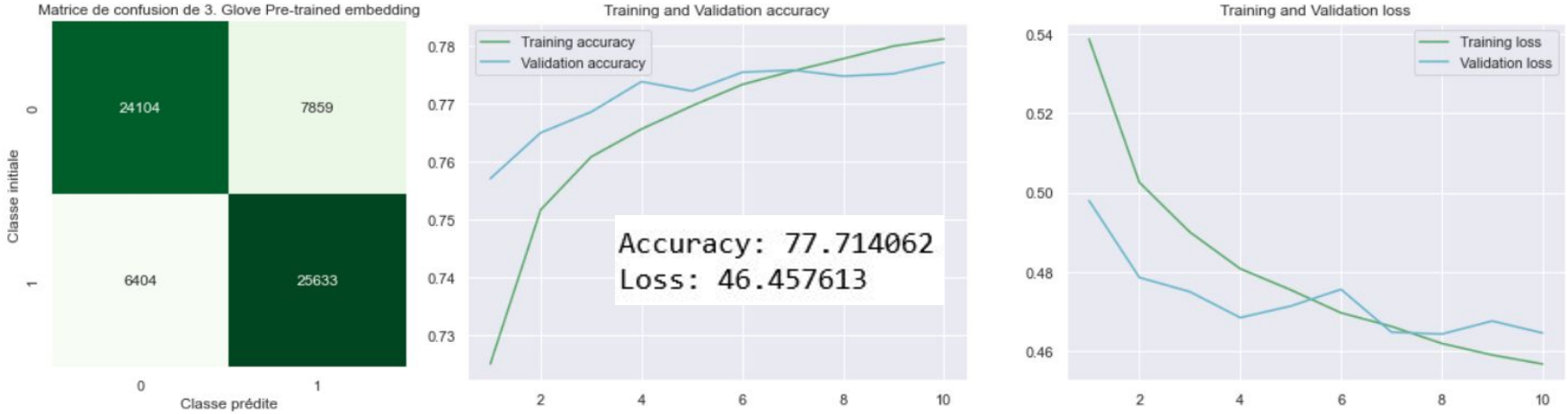
Model Name	seuil	F1-Score	FBeta-Score	Recall_score	Precision_score	Accuracy_score	Roc_AUC_score
1. 1D ConvNet	0.5	0.778856	0.770015	0.794051	0.764232	0.774281	0.857541
2. LTSM Bidirect	0.5	0.786167	0.780301	0.796142	0.776438	0.783203	0.867276

Nativement, le modèle Bidirectionnel LTSM est plus performant

Note: ici, le RNN ne tire pas encore profit de la taille du jeu d'entraînement

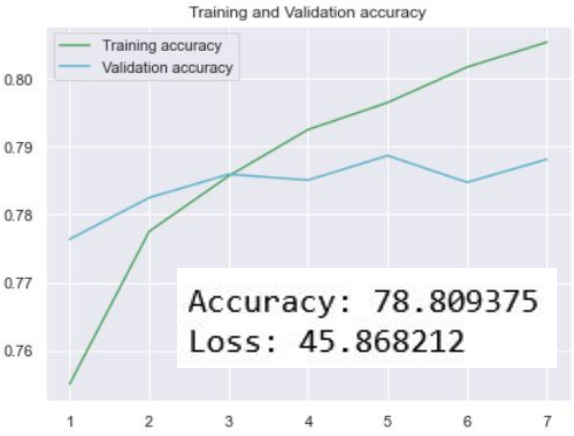
Choix du modèle “avancé” et Optimisation des méthodes de plongements

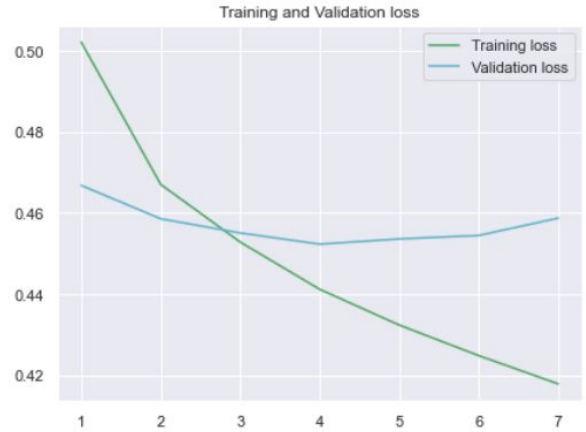
Evaluation du modèle Pre-entrainé GloVe



L'apprentissage semble cohérent ici, même si la valeur de Loss n'est pas la plus faible (46.4)
L'accuracy et Loss évoluent normalement sur le training set, ainsi que sur le validation set.

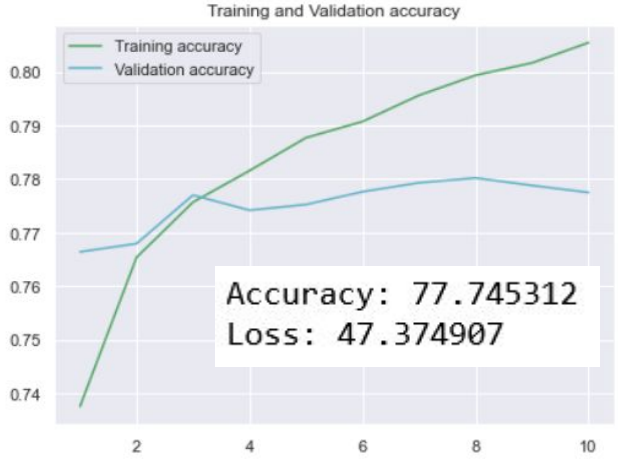
Evaluation du modèle Ré-entrainé GloVe





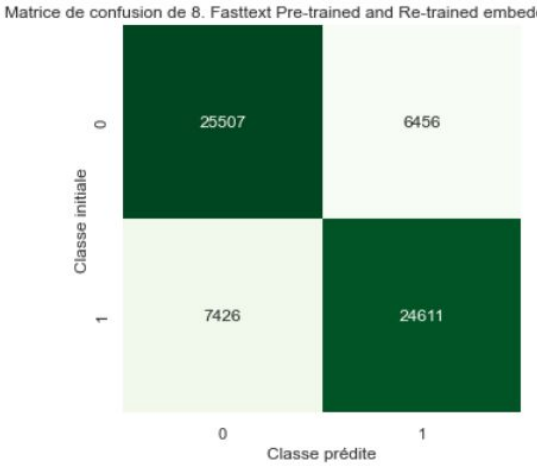
Le ré-entrainement du modèle nous permet d'améliorer encore l'Accuracy et Loss du modèle.
Le nombre d'Epochs est lui moins important pour stopper le processus d'apprentissage.

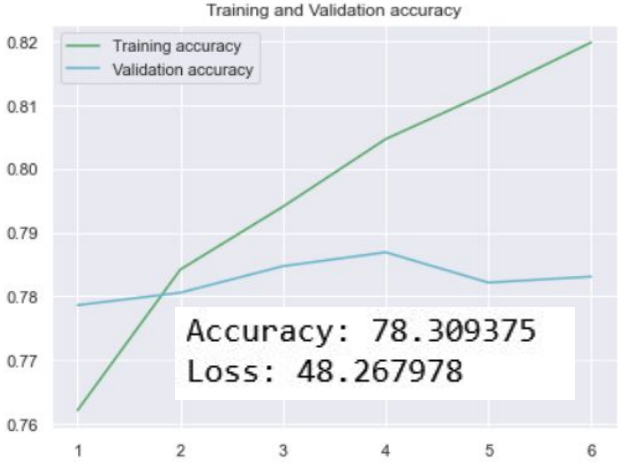
Evaluation du modèle Pré-entrainé Fasttext



L'apprentissage semble cohérent ici, même si la valeur de Loss est plus importante (47.4 au lieu de 46.4 et 45.8)
L'accuracy et Loss évoluent normalement jusqu'à 3 Epochs.

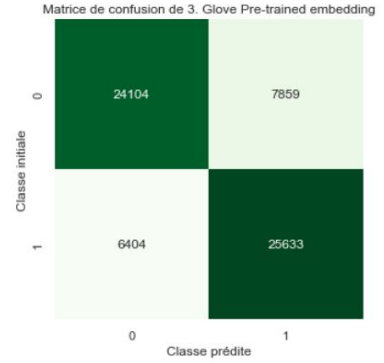
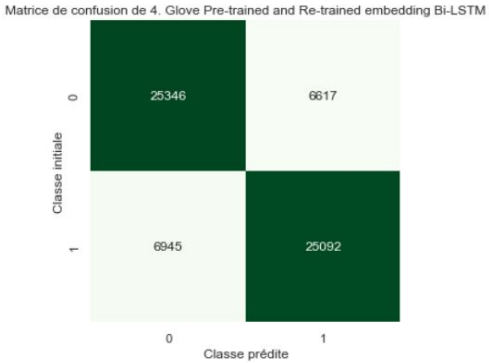
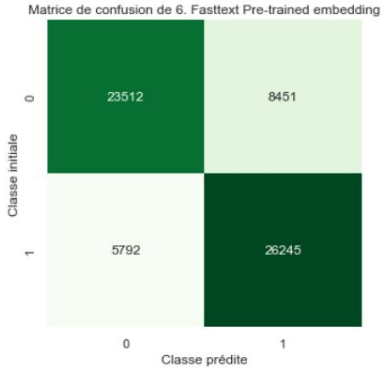
Evaluation du modèle Ré-entrainé Fasttext





Le ré-entrainement du modèle ne nous permet pas d'améliorer la Loss du modèle (48.3 au lieu de 47.4).
En revanche, l'Accuracy est elle meilleure qu'avec le modèle seulement pré-entrainé.

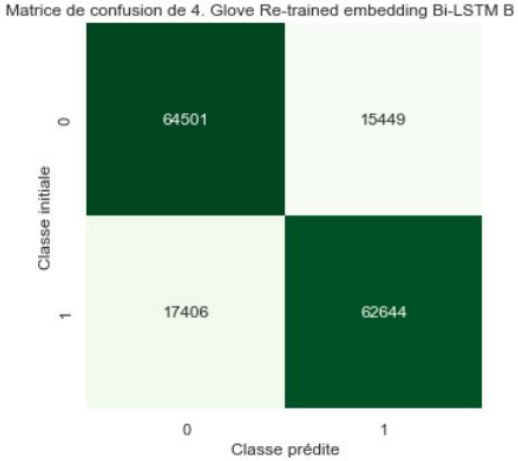
Comparaison des modèles sur mesure “Avancé”

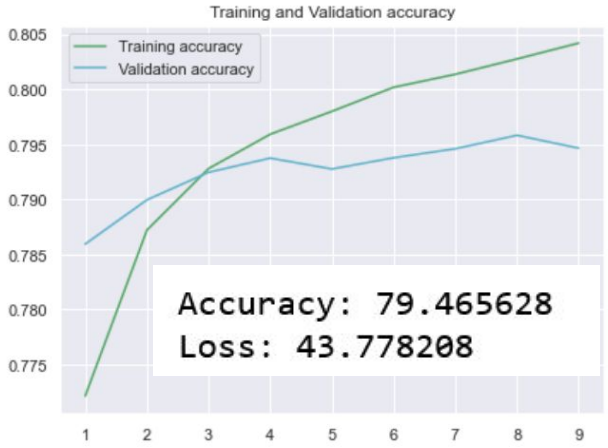
Pre-trained Glove	Re-trained Glove	Pre-trained Fasttext	Re-trained Fasttext
			
Accuracy_score: 0.777141	Accuracy_score: 0.788094	Accuracy_score: 0.777453	Accuracy_score: 0.783094
Roc_AUC_score: 0.862803	Roc_AUC_score: 0.871671	Roc_AUC_score: 0.862956	Roc_AUC_score: 0.866065
4	1	3	2

L’optimisation des plongements par Transfer Learning, accompagné du ré-entraînement sur notre jeu de données nous permet d’augmenter l’efficacité de notre modèle Deep learning de détection de sentiment.

Bénéficier de la taille du jeu de donnée

Evaluation du modèle Ré-entrainé GloVe BigData





Model Name	Accuracy_score	Roc_AUC_score
4. Glove Re-trained embedding Bi-LSTM BigData	0.794656	0.878792
4. Glove Pre-trained and Re-trained embedding	0.788094	0.871671

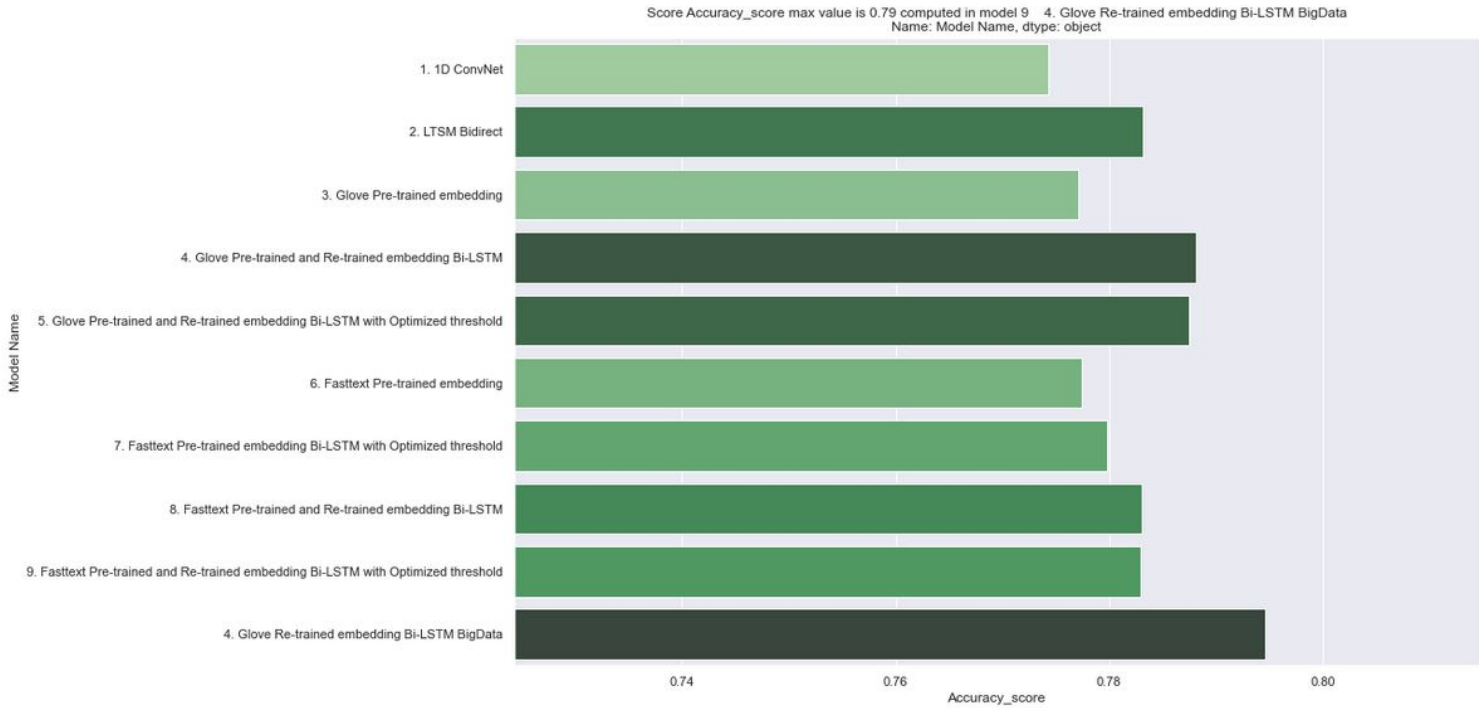
L'augmentation de la taille du jeu d'entraînement nous permet de tirer profit de la puissance du réseau de neurones.

L'accuracy gagne ici +1% , et la Loss diminue à 43.8 (au lieu de 45.9).

Récapitulatif des modèles “Avancés” Deep Learning

10 Modèles de Réseaux de neurones comparés !

2 Méthodes de plongements analysées !



Back to AGENDA

Modèle BERT

Modèles de Classification “BERT”

Modèles BERT Fine-tuned

AutoModel ForSequenceClassification

"bert-base-uncased"

01

AutoModel ForSequenceClassification

"distilbert-base-uncased"

02

Roberta ForSequenceClassification

"bert-base-uncased"

03

CamemBert ForSequenceClassification

"distilbert-base-uncased"

04

Nous utilisons les librairies Transformers de HuggingFace
Nous améliorerons la performance des modèles via l'utilisation de la class "Trainer" sur notre jeu de données.

Prepare Data

Nous changeons le format des données pour le rendre compatible avec les librairies HuggingFace

Preprocess

Nous pré-traitons les données à l'aide du tokenizer BERT, incluant padding et tronquage.

Compilation

Nous chargeons le modèle pré-entraîné de classification de texte BERT, et définissons les hyperparamètres et le nombre de label attendus.

Entrainement

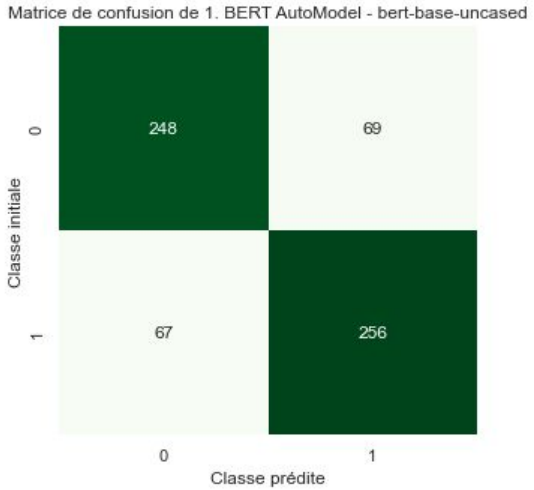
Nous entraînons nos 4 modèles supervisés, sur la base du training dataset

Evaluation

Nous mesurons la capacité de prédiction des différents modèles entraînés

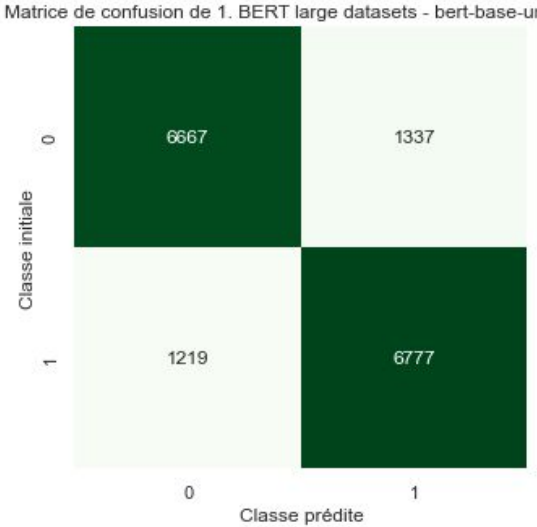
Comparaison des modèles BERT Fine-tuned

Model Name	seuil	F1-Score	FBeta-Score	Recall_score	Precision_score	Accuracy_score
1. BERT AutoModel - bert-base-uncased	0.5	0.790123	0.788663	0.79257	0.787692	0.7875
2. BERT AutoModel - distilbert-base-uncased	0.5	0.774295	0.780164	0.764706	0.784127	0.775
3. ROBERTA - bert-base-uncased	0.5	0.660767	0.658436	0.664688	0.656891	0.640625
4. CAMEMBERT - bert-base-uncased	0.5	0.702128	0.698307	0.708589	0.695783	0.69375



Le modèle le plus performant après fine tuning est le modèle BERT.
Nous allons ré-exécuter son entraînement sur une base de données plus importante, et évaluer son gain de performance.

Evaluation du modèle BERT



Nous notons que l'accuracy du modèle BERT est très supérieure à nos précédents modèles.

Nous recommandons donc d'investir dans ce type de modèle, afin d'améliorer les hyperparamètres, et augmenter la taille du jeu d'entraînement.

Model Name	seuil	F1-Score	FBeta-Score	Recall_score	Precision_score	Accuracy_score
1. BERT large datasets - bert-base-uncased	0.5	0.841341	0.837659	0.847549	0.835223	0.84025

Back to AGENDA

Déploiement du prototype (FastAPI)

Déploiement du Pipeline via FastAPI

Test en Local sur 2 phrases

Récupération des sentiments et scores

```
test_text
```

```
['We are very happy to test the great transformers library.',
 'We are very frustrated to see how complex it is.']
```

```
pipe = pipeline("text-classification")

pipe(test_text)
```

```
[{'label': 'POSITIVE', 'score': 0.9997225999832153},
 {'label': 'NEGATIVE', 'score': 0.9996737241744995}]
```

```
y_prob_test = trainer5.predict(tokenized_test_text_di
y_prob_test
```

```
PredictionOutput(predictions=array([[-3.6616232,  3.29
       [ 2.7986479, -2.3708546]], dtype=float32), labe
d': 23.594})
```

```
y_pred = np.argmax(y_prob_test.predictions, axis=-1)
y_pred
```

```
array([1, 0], dtype=int64)
```

```
result=[]
for pred in y_pred:
    if pred == 0:
        result.append('NEGATIVE')
    elif pred == 1:
        result.append('POSITIVE')
print(result)
```

```
['POSITIVE', 'NEGATIVE']
```

Name	Description
text * required string (query)	We are very frustrated to see how complex it

Response body

```
{
  "result": [
    {
      "label": "NEGATIVE",
      "score": 0.9996737241744995
    }
  ]
}
```

Name	Description
text * required string (query)	We are very happy to test the great transform

Response body

```
{
  "result": [
    {
      "label": "POSITIVE",
      "score": 0.9996731877326965
    }
  ]
}
```

Déploiement du modèle sur un serveur, et utilisation de la détection de sentiment et du niveau de confiance atteint.

Voir Démo

Conclusions

Entraîner plusieurs modèles Deep Learning sur des données textuelles.

Sélectionner des méthodes de prétraitement de texte pour améliorer les performances des modèles de classification.

Évaluer et comparer les performances de modèles distincts de Deep Learning.

Importance des choix des méthodes de plongement de mots (word embeddings)