

Rapport du projet NLP : Insurance Reviews



Membres de l'équipe :

- GUO Zhaojun
- DENG Muchan

Introduction :

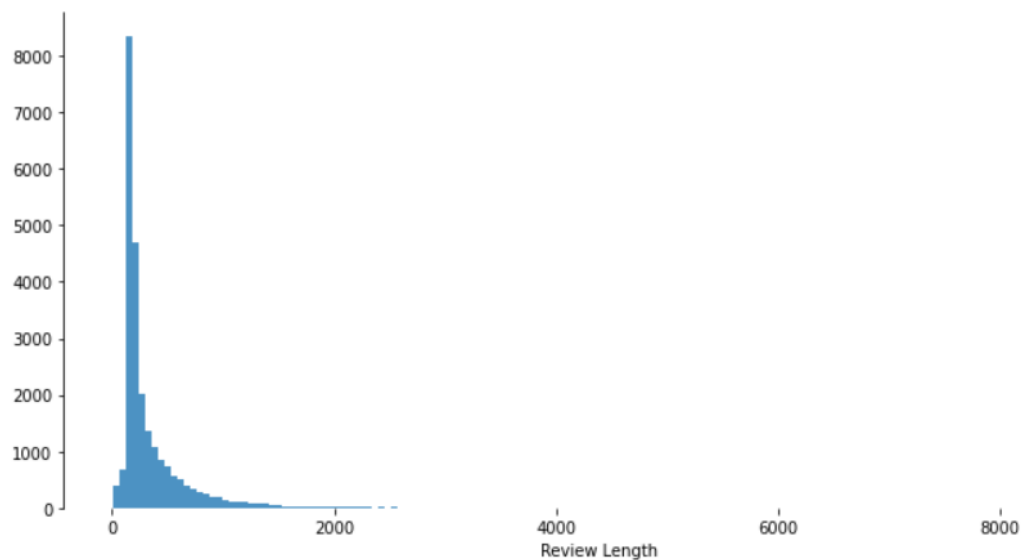
Nous travaillons sur une base de données appelé Insurance Reviews, ce dataset est constitué de 6 features : date, note, auteur, avis, assureur et produit, ces derniers constituent les éléments importants d'un avis client d'assurance.

Afin de mener une étude sur ce dataset, nous allons dans un premier temps chercher à comprendre et visualiser ces données, en suite effectuer du data cleaning pour que nous puissions par la suite travailler via les deux approches unsupervised learning et supervised learning.

Data exploration :

Avant de commencer, il faut remarquer que le dataset nous est fourni sous forme de deux subset : training set et test set, la variable note a été retirée dans le test set afin d'effectuer plus tard la prédiction.

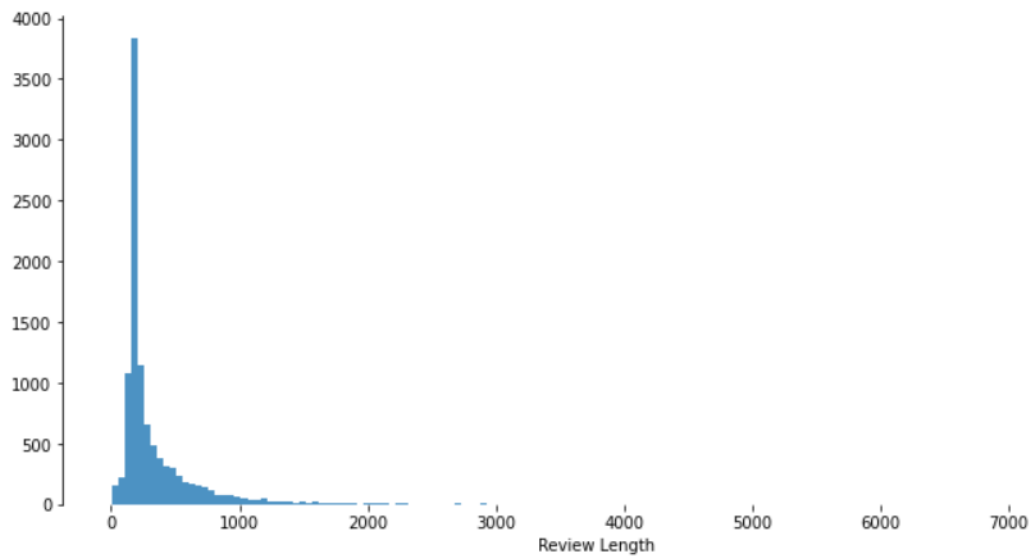
Après avoir supprimé les données manquantes (les null) et les données dupliquées et après avoir converti les dates sous forme Datetime, nous pouvons enfin avoir une première vue sur ces deux subset, d'abord le train set :



Dont seulement 0,95% des commentaires dépassent une longueur de 2000 caractères.

`230 reviews with LEN > 2000 (0.95 % of total data)`

Alors que dans le test set 0,77% des commentaires dépassant les 2000 caractères :

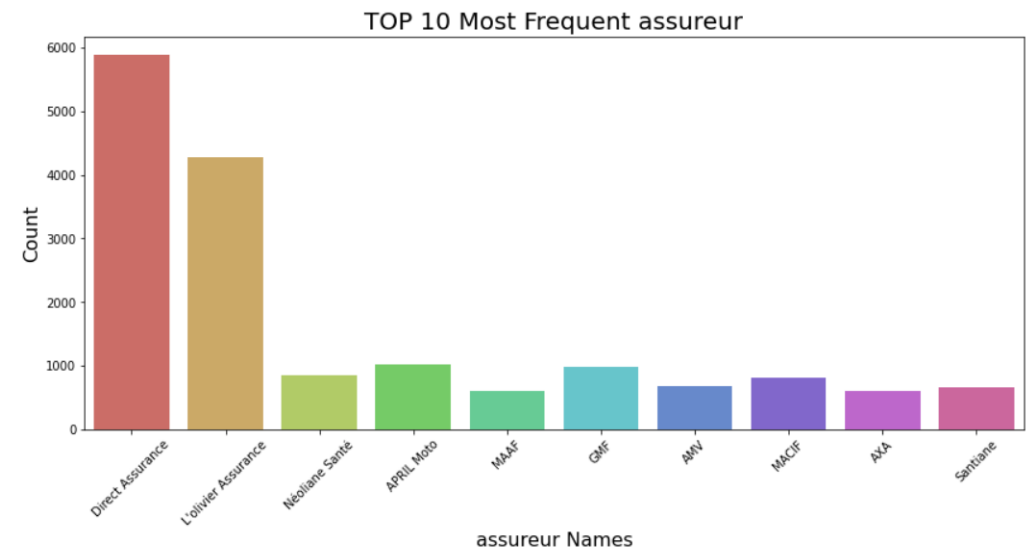


80 reviews with LEN > 2000 (0.77 % of total data)

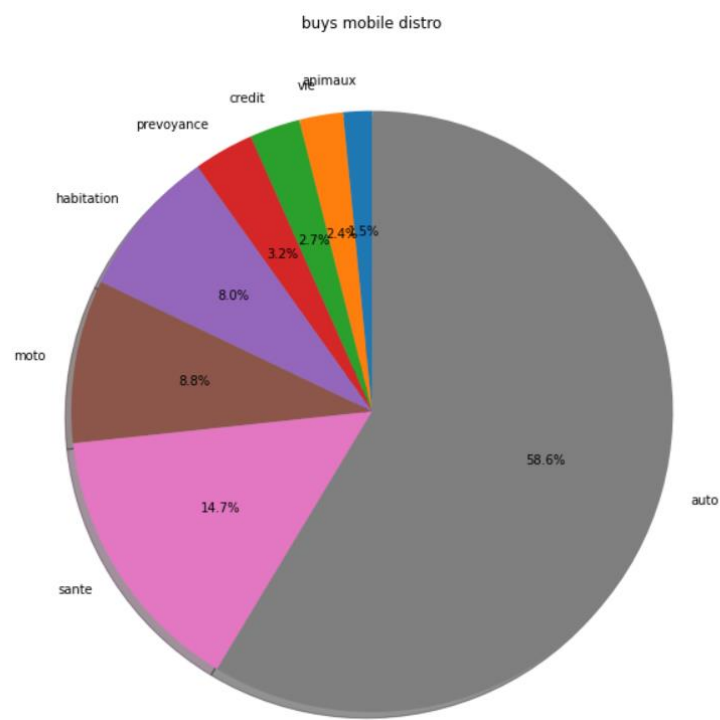
Nous avons décidé de pas prendre en compte les commentaires tres longs que ce soit le train set ou le test set.

Data visualisation :

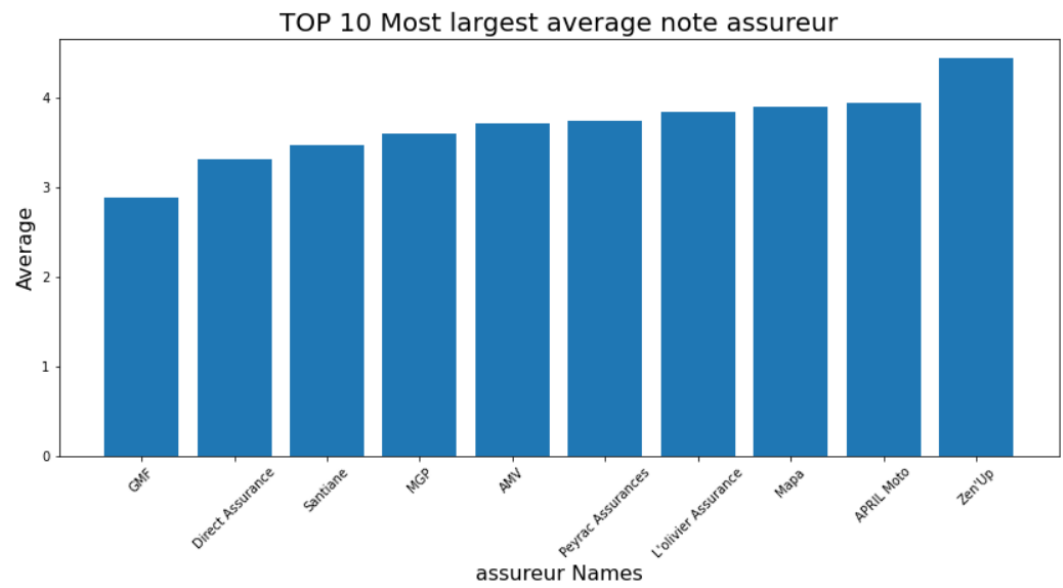
Nous pouvons voir que les assureurs les plus souvent apparus dans les commentaires sont : Direct assurance et l’Olivier assurance.



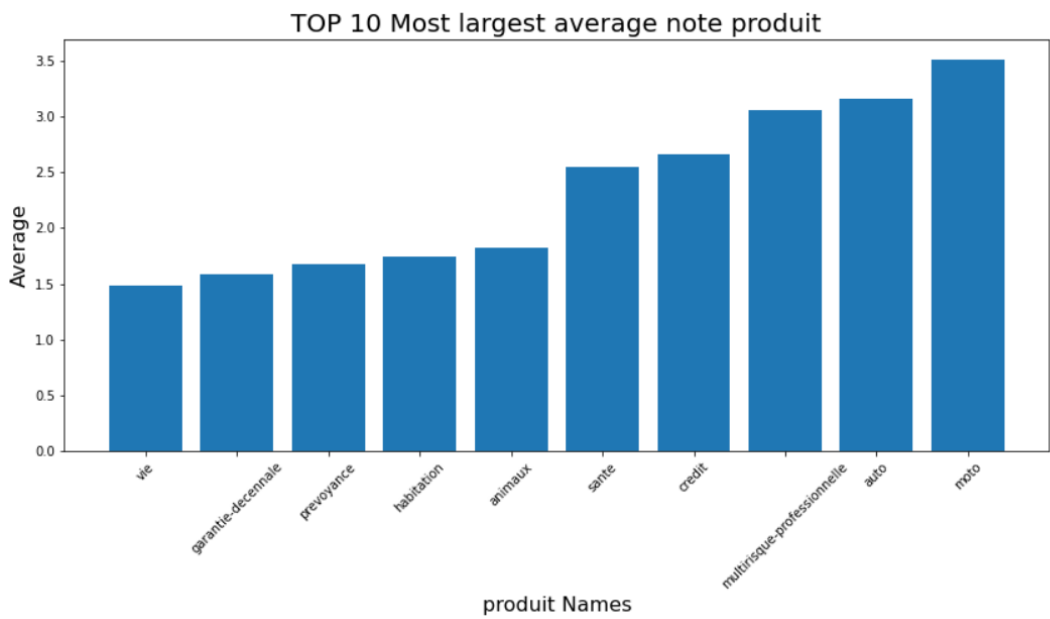
Nous pouvons voir sur le pieplot que les produits les plus souvent apparus dans les commentaires sont les produits auto les produits de santé et les moto.



Nous avons par la suite affiché le top 10 des assureurs les mieux notés (Moyenne des notes)



Et de la même manière le top 10 des produits d'assurances les mieux notés (Moyenne des notes)





Word Cleaning:

Nous utilisons treetagger afin de préparer nos données pour pouvoir par la suite effectuer du unsupervised learning.

Nous ajoutons d'abord le sens de chaque mot via le library `treetaggerwrapper`, et nous supprimons les mots qui rapportent peu de sens dans la phrase d'un point de vue algorithmique.

✓ 0 s result

	index_content	word	nature	note
0	1	meilleurs	tADJ	5
1	1	assurances	tNOM	5
3	1	prix	tNOM	5
5	1	solutions	tNOM	5
7	1	écoute	tNOM	5
...
1498029	23868	procédure	tNOM	1
1498032	23868	faire	tVER:infl	1
1498034	23868	plus	tADV	1
1498035	23868	grand	tADJ	1
1498036	23868	bien	tADV	1

614668 rows x 4 columns

En suite en gardant seulement les noms et les adjectifs, nous obtenons enfin notre dataset prêt pour effectuer du unsupervised learning.

Pour mieux illustrer cela, nous l'avons affiché sous forme d'un nuage de mots en fonction de leur fréquence d'apparition.



Ainsi, grâce au model Word2Vec, nous pouvons comparer le sens des mots entre eux.

Voici les mots ayant un sens proche du mot « meilleur ».

```
▶ w = 'meilleur'
  r = model.similar_by_word(w)

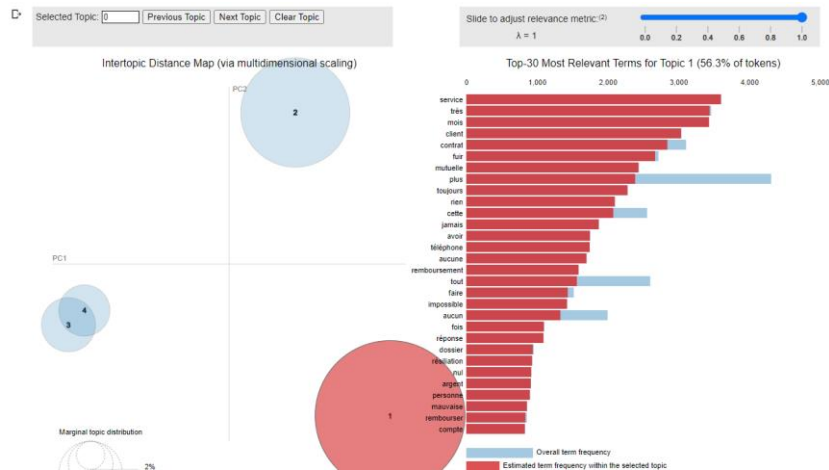
  print("Le plus proche word avec "+w+" est: ")
  for i in r:
    print(i)
```

```
↳ Le plus proche word avec meilleur est:
('imbattable', 0.656801700592041)
('interessant', 0.6258652210235596)
('qualité-prix', 0.6219062805175781)
('marché', 0.6216448545455933)
('concurrence', 0.618495523929596)
('meilleures', 0.6162171363830566)
('parmis', 0.6151142716407776)
('meilleurs', 0.611415445804596)
('attiré', 0.6098800897598267)
('recherchais', 0.6095585823059082)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py
```

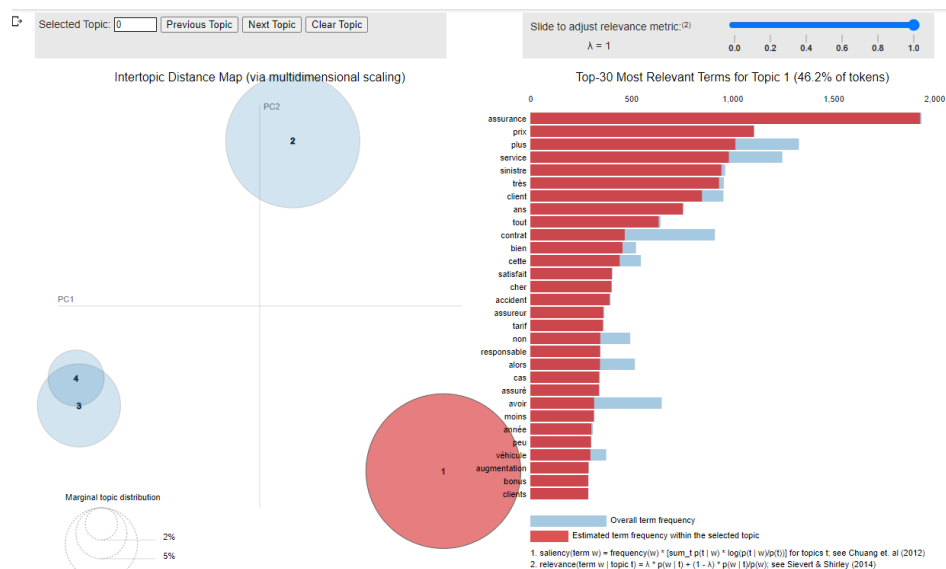

Unsupervised Learning :

La partie unsupervised learning nous permet d'étudier les topics présents dans le dataset, or nous avons pu remarquer que la note attribuée par l'auteur des commentaires joue un rôle assez important.

Nous avons donc décidé d'utilisé Linear discriminant analysis (LDA) et de travailler cas par cas, en distinguant les 5 notes de 1 à 5.



Le graph au-dessus est le résultat de prédiction pour les commentaires dont la note est égale à 1. Le graph en dessous correspond à la note 2 attribuée, nous pouvons voir que les deux graphs sont différents.



Les topics ainsi obtenus pour respectivement les notes de 1 à 5 sont : (chaque ligne correspond un topic constitue par un groupe de mot)

The note 1

```
[(0,
  '0.013*"vie" + 0.010*"dossiers" + 0.008*"respect" + 0.007*"traitement" + 0.006*"pacifica"'),
 (1,
  '0.058*"assurance" + 0.022*"sinistre" + 0.022*"prix" + 0.021*"ans" + 0.018*"plus"'),
 (2,
  '0.018*"service" + 0.017*"très" + 0.017*"mois" + 0.015*"client" + 0.014*"contrat"'),
 (3,
  '0.013*"macif" + 0.012*"expert" + 0.011*"matmut" + 0.010*"maif" + 0.009*"indemnisation"')]
```

The note 2

```
[(0,
  '0.015*"charge" + 0.011*"macif" + 0.010*"prise" + 0.009*"franchise" + 0.008*"expert"'),
 (1,
  '0.022*"mois" + 0.018*"toujours" + 0.012*"téléphone" + 0.011*"dossier" + 0.010*"remboursement"'),
 (2,
  '0.011*"tres" + 0.006*"meme" + 0.006*"apres" + 0.006*"choix" + 0.005*"choses"'),
 (3,
  '0.033*"assurance" + 0.019*"prix" + 0.017*"plus" + 0.017*"service" + 0.016*"sinistre"')]
```

```
print("The note 3")
lda_model3.print_topics(num_words=5)
```

The note 3

```
[(0,
  '0.013*"sinistre" + 0.012*"ans" + 0.010*"rapidité" + 0.009*"accueil" + 0.008*"tres"'),
 (1,
  '0.032*"prix" + 0.032*"assurance" + 0.029*"satisfait" + 0.026*"service" + 0.021*"très"'),
 (2,
  '0.008*"telephone" + 0.005*"telephonique" + 0.005*"ca" + 0.004*"ecoute" + 0.004*"espece"'),
 (3,
  '0.029*"mutuelle" + 0.015*"augmentation" + 0.011*"concurrence" + 0.010*"espérant" + 0.005*"proposition"')]
```

The note 4

```
[(0,
  '0.062*"assurance" + 0.061*"service" + 0.054*"satisfait" + 0.053*"prix" + 0.029*"satisfaite"'),
 (1,
  '0.027*"plus" + 0.016*"ans" + 0.015*"véhicule" + 0.015*"sinistre" + 0.013*"facilité"'),
 (2,
  '0.018*"attentes" + 0.017*"espérant" + 0.016*"conditions" + 0.013*"l'écoute" + 0.012*"automobile"'),
 (3,
  '0.063*"très" + 0.022*"bien" + 0.019*"satisfait" + 0.017*"bonne" + 0.016*"prix"')]
```

The note 5

```
[(0,
  '0.028*"mutuelle" + 0.024*"amis" + 0.022*"entourage" + 0.021*"besoins" + 0.021*"recommander"'),
 (1,
  '0.046*"cordialement" + 0.011*"raisonnables" + 0.009*"zéro" + 0.009*"utiliser" + 0.007*"monsieur"'),
 (2,
  '0.061*"prix" + 0.057*"satisfait" + 0.056*"assurance" + 0.046*"service" + 0.021*"rapide"'),
 (3,
  '0.104*"très" + 0.043*"satisfaite" + 0.034*"merci" + 0.021*"écoute" + 0.018*"bonne"')]
```

Supervised Learning :

Nous avons essayé 2 modèles de régression (pretrain embedding hub et lstm) et un modèle de classification (CamemBERT) mais nous estimons que cette tâche a plus de sens en tant que tâche de régression, car la loss (RMSE) de faussement prédire la note 1 au lieu de 5 est supérieure à la loss de la fausse prédiction de la note 2 à la note 5; mais en tant que tâche de classification, la loss (CrossEntropy) est la même entre les catégories quel que soit le nombre de points qu'elles prédisent.

Pretrain embedding hub

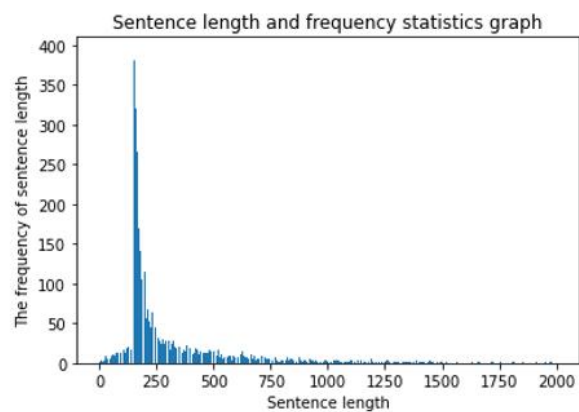
Dans ce premier model, nous utilisons un pre trained text-embedding modèle de TensorFlow Hub pour créer embedding vector de mots. Il s'agit d'un modèle multilingue pré-entraîné. Il s'agit d'un modèle de bout en bout.

```
✓ 秒 model_pretrain_embedding_hub.summary()
```

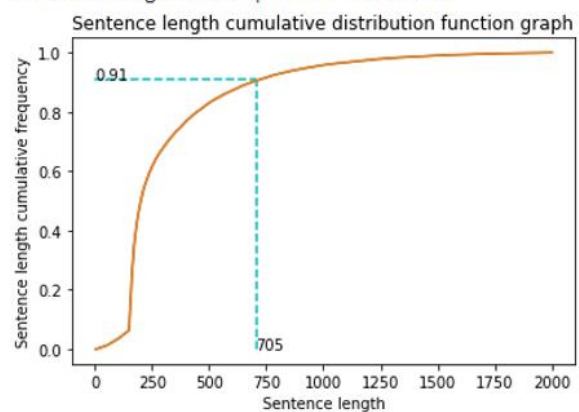
```
Model: "sequential"
Layer (type)                Output Shape              Param #
=====
keras_layer_1 (KerasLayer)   (None, 512)              68927232
dense (Dense)                (None, 16)               8208
dense_1 (Dense)              (None, 1)                17
=====
Total params: 68,935,457
Trainable params: 8,225
Non-trainable params: 68,927,232
```

LSTM

Nous utilisons le vocabulary de nous avons nettoyé pour représenter l'index de chaque mot. Si le mot ne fait pas partie de ce vocabulaire, nous le traiterons comme 0. Ensuite, nous effectuons le padding de longueur 300 à chaque input review, car comme nous pouvons le voir ci-dessous, la plupart des reviews ont une longueur à l'alentour de 300.



Sentence length with quantile 0.91:705.



Voici la performance du modèle LSTM :

Il y a un risque de overfitting, embedding length a été modifiée à 50 et les paramètres ont été réduits. La quantité de données est trop petite, il est donc facile de overfitting.

```
Model: "sequential_1"
Layer (type)                Output Shape                Param #
-----
embedding (Embedding)       (None, 300, 50)            6000100

spatial_dropout1d (SpatialD  (None, 300, 50)            0
ropout1d)

lstm (LSTM)                  (None, 128)                 91648

dense_2 (Dense)              (None, 1)                   129

Total params: 6,091,877
Trainable params: 6,091,877
Non-trainable params: 0

Epoch 1/10
38/38 [=====] - 48s 1s/step - loss: 1.9542 - root_mean_squared_error: 2.0563 - val_loss: 1.5254 - val_root_mean_squared_error: 1.5256
Epoch 2/10
38/38 [=====] - 45s 1s/step - loss: 1.4523 - root_mean_squared_error: 1.4533 - val_loss: 1.3276 - val_root_mean_squared_error: 1.3278
Epoch 3/10
38/38 [=====] - 45s 1s/step - loss: 1.0683 - root_mean_squared_error: 1.0732 - val_loss: 0.9996 - val_root_mean_squared_error: 0.9999
Epoch 4/10
38/38 [=====] - 45s 1s/step - loss: 0.9197 - root_mean_squared_error: 0.9202 - val_loss: 0.9525 - val_root_mean_squared_error: 0.9530
Epoch 5/10
38/38 [=====] - 45s 1s/step - loss: 0.8661 - root_mean_squared_error: 0.8665 - val_loss: 0.9229 - val_root_mean_squared_error: 0.9232
Epoch 6/10
38/38 [=====] - 45s 1s/step - loss: 0.8323 - root_mean_squared_error: 0.8334 - val_loss: 0.9351 - val_root_mean_squared_error: 0.9354
Epoch 7/10
38/38 [=====] - 45s 1s/step - loss: 0.7778 - root_mean_squared_error: 0.7783 - val_loss: 0.9218 - val_root_mean_squared_error: 0.9221
Epoch 8/10
38/38 [=====] - 45s 1s/step - loss: 0.7452 - root_mean_squared_error: 0.7458 - val_loss: 0.8847 - val_root_mean_squared_error: 0.8851
Epoch 9/10
38/38 [=====] - 45s 1s/step - loss: 0.7240 - root_mean_squared_error: 0.7247 - val_loss: 0.9192 - val_root_mean_squared_error: 0.9198
Epoch 10/10
38/38 [=====] - 45s 1s/step - loss: 0.7084 - root_mean_squared_error: 0.7092 - val_loss: 0.8997 - val_root_mean_squared_error: 0.9003
<keras.callbacks.History at 0x7fdf39a0f610>
```

Camembert : <https://huggingface.co/camembert-base> CamemBERT est un modèle pour le français basé sur le modèle RoBERTa.

```
Model: "tf_camembert_for_sequence_classification"
Layer (type)                Output Shape                Param #
-----
roberta (TFRobertaMainLayer multiple
)                             110031360

classifier (TFRobertaClassi multiple
ficationHead)                 594437

Total params: 110,625,797
Trainable params: 110,625,797
Non-trainable params: 0
```

Colab :

<https://colab.research.google.com/drive/1HhiZF3pkUV1gvbf3SnjivL3OkBM4J8HW?usp=sharing>

Résultat :

<https://drive.google.com/drive/folders/1qjSQIY6Elj9pt6krV7oU0lpfDIYgvlAA?usp=sharing>