# Projet 8 OC : Compétition kaggle - Jigsaw Multilingual Toxic Comment Classification

Présenté par : NAMA NYAM Guy Anthony

Mentor: Julien Hendrick

24 Juin 2020

## **DPENCLASSROOMS**



## Sommaire

- Introduction
- Source de données
- Modélisation
- 4 Résultats et discussions
- Conclusion



## Mise en situation

- Dans les conversations en ligne, Il suffit d'un seul commentaire toxique pour aigrir une discussion.
- La toxicité est définie comme tout ce qui est grossier, irrespectueux ou autrement susceptible de pousser quelqu'un à quitter une discussion.
- Ses toxicités dans les conversations en ligne peuvent être détectées par les modèles de machine learning.
- Les utilisateurs des plateformes de discussion en ligne proviennent de tous les quatre coins du monde, c'est dire la diversité des commentaires multilingue qu'on y retrouvent.

## Objectif

Contribuer à l'identification de commentaires toxiques dans différentes langues pour un Internet plus sûr et collaboratif.

# **Approches**

- Pour ce faire, nous utilisons les représentations vectorielles denses du langage naturel par les architectures de réseaux de neurones profond pré-entraînés.
- Dans ce contexte multilingue, nous utilisons les capacités croissantes et impressionnantes de modèles des derniers innovations.
- Il s'agit principalement des modèles multilingues BERT et XLM
- Nous tirons profit des ressources informatiques grandissantes TPU pour améliorer notre capacité de modélisations.

## Sommaire

- Introduction
- 2 Source de données
- Modélisation
- 4 Résultats et discussions
- Conclusion

# Description

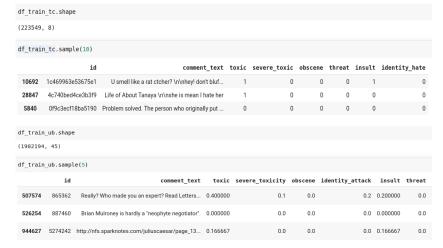
- Nous avons 4 jeux de données :
  - jigsaw-toxic-comment-train.csv: dataset de commentaires en anglais provenant des discussions Wikipedia(Dataset d'entraînement)
  - jigsaw-unintended-bias-train.csv : second dataset de commentaires provenant de commentaires civils(Dataset d'entraînement)
  - validation.csv : commentaires des pages de discussion Wikipedia dans différentes langues non anglaises(Dataset de validation)
  - test.csv: commentaires des pages de discussion Wikipedia dans différentes langues non anglaises(Dataset de test)

Colonnes prépondérantes à la suite du travail :

- id identifiant à l'intérieur de fichier correspondant au commentaire.
- comment\_text le texte du commentaire à classer
- lang le langage du commentaire
- toxic si le commentaire est classé comme toxique ou non.

# Preprocessing - fusion train dataset

• Taille des dataset et mise en évidence de la colonne cible toxic.



# Preprocessing - fusion train dataset

Mettre 0 si probabilité inférieur à 0.5, 1 sinon

```
for f in ['toxic']:
    df_train_ub.loc[df_train_ub[f] >= 0.5, f] = 1
    df_train_ub.loc[df_train_ub[f] < 0.5, f] = 0</pre>
```

Observation des proportions de labels dans chaque dataset

 Fusionner l'ensemble du dataset de wikipédia avec une partie du dataset comment civils.

```
df_train = tm.pd.concat([df_train_tc, df_train_ub[df_train_ub.toxic == 1]], axis=0, join='inner', ignore_index=True)
df_train.shape
(375660, 3)
```

4 D F 4 B F 4 B F

# Preprocessing - multiprocessing

Train dataset

L'ensemble des tâches de nettoyage sont en multiprocessing.

```
from multiprocessing import Pool

def parallelize(data, func, num_of_processes=4):
    data_split = np.array_split(data, num_of_processes)
    pool = Pool(num_of_processes)
    data = tm.pd.concat(pool.map(func, data_split))
    pool.close()
    pool.join()
    return data
```

df train = parallelize(df train, emoji replace)

# Preprocessing - nettoyage urls

#### Suppression des urls

```
import re

def remove_URL(df):
    """Remove_URLs from a sample string"""
    for i, row in df.iterrows():
        text = row["comment_text"]
        text = re.sub(r'http\S+', "", text)
        text = re.sub(r'www.\S+', "", text)
        df.loc[i, "comment_text"] = text
    return df
```

#### • Exemples textes avant et après suppression urls.

```
1d **Comment_text

"In In Snowflakes are NOT always symmetrical! In full inder Geometry & is stated that "A coogliske always but also symmetric arms." This statertion is simply not true! According to Kenneth Libbrocht. "The rather because I still see a decent number of failules on this page, forgine mit in more at these above that see that see and get PACTS stiff of it in the page. The state is the page of the
```

"In In Snowflakes are NOT always symmetrical! In In Index Geometry 8 is stated that "A snowflake always has six symmetric atrms." This assertion is simply not true! According to Kerneth Libbrecht, "The rather liddi995809/fa unattractive irregular crystals are by far the most common variety." Someone really need to take a look at his site and get FACTS off of it because I still see a decent number of fallelites on this page. (Forgive me In mow

I was able to port the above list so quickly because a larsedy had it in a test file in my had drive he been mening to get around to quideling the sound list for some time now. What far as penerating interest the significant part of the property of the p

# Preprocessing - nettoyage contract forms

Forme longue des contractions

```
import contractions

def contract(df):
    """Resolving contractions"""
    for i, row in df.iterrows():
        df.loc[i, "comment_text"] = contractions.fix(str(row["comment_text"]), slang=False)
    return df
```

• Exemples textes avant et après remplacement forme contractée.

toxic	comment_text	id	
0	Explanation\n\Why the edits made under my username Hardcore Metallica Fan were reverted? They wern't yandalisms, just closure on some GAs after I voted at New York Dolls FAC. And please the properties of the talk page since the retired properties of the properties of the talk page ince the retired ones. See 15 ones.	0000997932d777bf	o
0	D'aww! He matches this background colour seemingly stuck with. Thanks. (talk) 21:51, January 11, 2016 (UTC)	000103f0d9cfb60f	1
0	Hey man, to really not trying to edit war to just that this guy is constantly removing relevant information and talking to me through edits instead of my talk page. He seems to care more about the formatting than the actual info.	000113f07ec002fd	2
toxic	comment text		
		id	
		id	
0	Explanation\nWhy the edits made under my username Hardcore Metallica Fan were reverted? They less not you and alisms, just closure on some GAs after I voted at New York Dolls FAC. And please member the template from the talk page move. 89.205.38.27	id 0000997932d777bf	0
0	Explanation\nWhy the edits made under my username Hardcore Metallica Fan were reverted? They were not vandalisms, just closure on some GAs after I voted at New York Dolls FAC. And please dar of remove the template from the talk page since I am retired		0
	Explanation\nWhy the edits made under my username Hardcore Metallica Fan were reverted? They ware rouvandalisms, just closure on some GAs after i voted at New York Dolls FAC. And please to not remove the template from the talk page since! imprestied now.89.205.38.27  D'aww! He matches this background colour impresses seemingly stuck with. Thanks. (talk) 2.15.	0000997932d777bf	•

## Preprocessing - nettoyage emoji

• Remplacement des *emoji* par la forme textuelle.

```
import emoii
def emoii replace(df):
    for i, row in df.iterrows():
        text = str(row["comment text"])
        if emoji.emoji count(text):
            dem = emoil.demoilze(text)
            tab = dem.split(":")
            tab = [' '.join(tab[j].split('_')) for j in range(len(tab))]
            # Delete emoii successive
            while k < (len(tab) - 2):
                if (tab[k+1] == '') and (tab[k] == tab[k+2]):
                    del tab[k+2]
                    del tab[k+1]
                else:
                    k += 1
            res = ' '.join(tab).strip()
            df.loc[i, "comment text"] = res
    return df
```

• Exemples textes avant et après remplacement émoji.



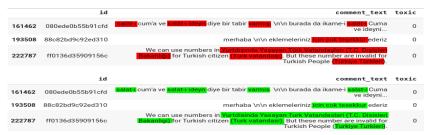
# Preprocessing - nettoyage Unidecode

Représentation l'ensemble du texte dans l'encodage ASCII

```
from unidecode import unidecode

def unicode_characters(df):
    for i, row in df.iterrows():
        df.loc[i, "comment_text"] = unidecode(str(row["comment_text"]))
    return df
```

Exemples textes avant et après encodage ASCII



24 juin 2020

# Preprocessing - nettoyage misspelling words

Détection des mots "supposés" mal orthographiés.

```
from spellchecker import SpellChecker
spell = SpellChecker()

def suggest_misspelled(df):
    misspelt = dict()
    for i, row in df.iterrows():
        doc = np(str(row["comment text"]))
        misspelted = spell.unknown([token.text for token in doc])
    for unknown in misspelted:
        if unknown in misspelte:
        misspelt[unknown] + 1
        else:
        misspelt[unknown] = 1

with open("miss{}.txt'.format(i), 'w') as outfile:
        Json.dump(misspelt, outfile)
```

Remplacer les mots d'occurrences inférieurs ou égale à 3(+400 milles)

```
def misspelling_replace(words):
    misspelling_repl = []
    for word in words:
        misspelling_repl.append((word, spell.correction(word)))
    return misspelling_repl

misspelling_replace

('tapdance', 'tapdance'),
('zumpta', 'humpty'),
('doli', 'doll'),
('chiffer', 'schiffer'),
('nabila', 'nabil'),
('jresult', 'result'),
```

('alfentanil', 'alfentanil'), ('toomas', 'thomas').

# Preprocessing - nettoyage astericks words

Détection des mots avec astérisques

Remplacement à la main les mots avec astérisques. Exemple d'extrait.

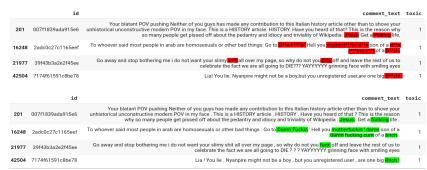
```
len(asterick_replace)

asterick_replace

('d**n', 'damn'),
('big*ted', 'bigoted'),
('ClusterF**K', 'ClusterFucK'),
('N**gga', 'Nigga'),
('a*sholes', 'assholes'),
('di*ks', 'dicks'),
('F*uck', 'Fuck'),
('sc*m', 'scam'),
('g***Y', 'GAY'),
('1**sin', 'ticking'),
('fustan', 'fucking'),
```

# Preprocessing - nettoyage multiple space

 Exemples textes avant et après remplacement des mots avec astérisques.



Suppression des espaces multiples.

```
def remove space multiple(df):
    """Remove space multiple"""
    for i, row in df.iterrows():
        text = str(row["comment_text"]).strip()
        df.loc[i, "comment_text"] = ' '.join(text.split())
    return df
```

## Sommaire

- Introduction
- Source de données
- Modélisation
  - m-Bert
  - XLM-Roberta
- 4 Résultats et discussions
- Conclusion



### **FullTokenizer**

• Récupération de l'objet de la classe *FullTokenizer* qui contient la méthode de *tokenisation* et l'encodeur de token en *id*.

Paramètres : bert\_path : str

Chemin vers Bert Saved Model

Retour: tokenizer: Objet bert.tokenization.FullTokenizer
Tokenizer bert

```
def get_tokenizer(bert_path=BERT_PATH_SAVEDMODEL):
    """Obtenez le tokenizer pour une couche BERT."""
    bert_layer = tf.saved_model.load(bert_path)
    bert_layer = hub.KerasLayer(bert_layer, trainable=False)
    vocab_file = bert_layer.resolved_object.vocab_file.asset_path.numpy()
    cased = bert_layer.resolved_object.do_lower_case.numpy()
    tf.gfile = tf.io.gfile # bert.tokenization.load_vocab dans tokenizer
    tokenizer = bert.tokenization.FullTokenizer(vocab_file, cased)
    return tokenizer

tokenizer = get_tokenizer()
```

# Encodage

Paramètres : sentence : str - Texte à encoder

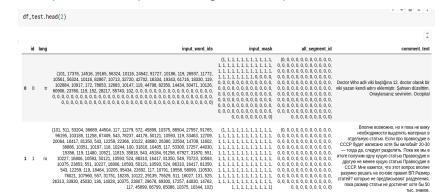
 Convertir chaque token en id avec ajout de tokens spéciaux. max seq length : int - Longueur maximale de la séquence encodée

tokenizer: Obiet bert.tokenization.FullTokenizer - Tokenizer bert

```
Retour : input_ids : list - Liste contenant ids des tokens
        input mask: list - Liste contenant 1 pour les tokens et 0 pour le rembourrage
        segment ids : list - Liste de zéros
 def process_sentence(sentence, max_seq_length=SEQUENCE_LENGTH, tokenizer=tokenizer):
     """Convertit la phrase sous la forme ['input word ids', 'input mask', 'segment ids']."""
     # Tokenize, et tronque à max_seq_length si necessaire.
     tokens = tokenizer.tokenize(str(sentence))
     if len(tokens) > max_seq_length - 2:
         tokens = tokens[:(max seg length - 2)]
     # Convertir les tokens de la phrase en IDs
     input_ids = tokenizer.convert_tokens_to_ids(["[CLS]"] + tokens + ["[SEP]"])
     # 1 pour les vrais tokens et un 0 pour les tokens de rembourrage.
     input_mask = [1] * len(input_ids)
     # Compléter par des zéros si la séquence est inférieur à max_seq_length
     pad_length = max_seq_length - len(input_ids)
     input_ids.extend([0] * pad_length)
     input_mask.extend([0] * pad_length)
     # Nous avons un seul segment d'entrée
     segment_ids = [0] * max_seq_length
     return (input ids. input mask, segment ids)
```

# Représentation vectorielle dense du texte

• Exemple de sortie après encodage du texte par m-Bert.



# Modèle multilingue Bert

Construire l'architecture du modèle multilingue bert

Paramètres : max\_seq\_len : int

Utiliser pour la forme des entrées du modèle.

trainable\_bert : bool

Entraîner la partie représentation ou pas.

Retour : \_ : tf.keras.Model Modèle bert

```
def multilingual_bert_model(max_seq_length=SEQUENCE_LENGTH, trainable_bert=True):
    """Construit et retourne un modèle Bert multilingue"""
   input_word_ids = tf.keras.layers.Input(shape=(max_seg_length,), dtype=tf.int32, name="input_word_ids")
    input_mask = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="input_mask")
   segment_ids = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="all_segment_id")
   bert_layer = tf.saved_model.load(BERT_GCS_PATH_SAVEDMODEL)
   bert_layer = hub.KerasLayer(bert_layer, trainable=trainable_bert)
   pooled_output. _ = bert_layer([input_word_ids. input_mask. segment_ids])
   output = tf.keras.layers.Dense(32, activation='relu')(pooled_output)
   output = tf.keras.lavers.Dense(1, activation='sigmoid', name='labels')(output)
    return tf.keras.Model(inputs={'input_word_ids': input_word_ids,'input_mask': input_mask,
                                  'all_segment_id': segment_ids},outputs=output)
```

## **TPU**

#### Détection TPUs ou GPUs

```
# Detect hardware, return appropriate distribution strategy
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
    # Default distribution strategy in Tensorflow. Works on CPU and single GPU.
    strategy = tf.distribute.get_strategy()
print("REPLICAS: ", strategy.num_replicas_in_sync)
```

```
Running on TPU grpc://10.0.0.2:8470
REPLICAS: 8
```

Chargement et compilation du modèle sur TPU.

# Training and predict

 Notre stratégie d'entraînement est d'adaptée au dataset en anglais puis au dataset de validation multilingue.

Format du contenu du fichier soumission.

```
Generating submission file...
id,toxic
0,0.083437
1,0.128149
2,0.256390
3,0.082323
4,0.005595
5,0.358424
```

# Encodage AutoTokenizer

 AutoTokenizer une classe de tokeniseur générique en fonction du modèle. XLM-Roberta, MODEL = 'jplu/tf-xlm-roberta-large'.

```
def get_tokenizer(MODEL):
    """Obtenez le tokenizer pour XLM-Roberta"""
    tokenizer = AutoTokenizer.from_pretrained(MODEL)
    return tokenizer

tokenizer = get_tokenizer(MODEL)
```

Conversion des tokens en ids. Méthode encode\_plus du tokeniseur.

```
def encode(texts, tokenizer, maxlen=512):
    enc_di = tokenizer.batch_encode_plus(
        texts,
        return_attention_masks=False,
        return_token_type_ids=False,
        pad_to_max_length=True,
        max_length=maxlen
)
return np.array(enc_di['input_ids'])
```

## Représentation vectorielle dense du texte

#### Encodage

```
x_train = encode(train.comment_text.values, tokenizer, maxlen=SEQUENCE_LENGTH)
x_valid = encode(valid.comment_text.values, tokenizer, maxlen=SEQUENCE_LENGTH)
x_test = encode(test.comment_text.values, tokenizer, maxlen=SEQUENCE_LENGTH)

y_train = train.toxic.values
y_valid = values
```

Exemple de sortie après encodage du texte par XLMRobertaTokenizer.

```
x_valid[:1]
array([[
              0.
                           54367.
                                     300.
                                                     47612.
                                                                144, 161265,
                   3688.
                             236.
                                                      1818.
                                                                459.
              8.
                                        6.
                                                 5.
                                                                        4104.
         11035.
                   1651.
                             520.
                                                                       91614.
                                    10155.
                             121, 217332,
                                               42.
                                                       166,
                                                              22068,
                                                                         113,
         13130, 129402, 178705,
                                                       166.
                                                                 23, 141933,
                  91958.
                                                      1039.
                                                              88684.
                                                                      58744.
                                    3126,
                   6053,
                                             8090,
                                       95,
         34605.
                            1108.
                                                     45962.
```

## tf.data.Dataset

• L'utilisation efficace de la mémoire et TPUs avec l'api tf.data.Dataset

```
train_dataset = (
   tf.data.Dataset
    .from_tensor_slices((x_train, y_train))
    .repeat()
    .shuffle(2048)
    .batch(BATCH_SIZE)
    .prefetch(AUTO)
valid dataset = (
   tf.data.Dataset
    .from_tensor_slices((x_valid, y_valid))
    .batch(BATCH_SIZE)
    .cache()
    .prefetch(AUTO)
test_dataset = (
   tf.data.Dataset
    .from_tensor_slices(x_test)
    .batch(BATCH_SIZE)
```

## La classe AutoModel

 Construire le modèle grâce aux couches de l'architecture XLM-Roberta

```
def build_model(transformer, max_len=512):
    """
    Construire le modèle à partir des couches Roberta
    """
    input_word_ids = Input(shape=(max_len,), dtype=tf.int32, name="input_word_ids")
    sequence_output = transformer(input_word_ids)[0]
    cls_token = sequence_output[: 0, :]
    x = tf.keras.layers.Dropout(0.5)(cls_token)
    out = Dense(1, activation='sigmoid')(x)
    model = Model(inputs=input_word_ids, outputs=out)
    model.compile(tf.keras.optimizers.Adam(lr=1e-5), loss='binary_crossentropy', metrics=[tf.keras.metrics.AUC()])
    return model
```

Load model into TPU

```
%%time
with strategy.scope():
    transformer_layer = TFAutoModel.from_pretrained(MODEL)
    model = build_model(transformer_layer, max_len=SEQUENCE_LENGTH)
```

27/33

## **Training**

 Tout d'abord, nous entraînons notre modèle sur le dataset d'entrainement qui est entièrement en anglais.

 Puis, nous l'entraînons sur l'ensemble de validation, qui est beaucoup plus petit mais contient un mélange de différentes langues.

## Sommaire

- 1 Introduction
- Source de données
- Modélisation
- Résultats et discussions
- Conclusion

## Résultats et discussions

- AUC public modèle jplu/tf-xlm-roberta-large est de : 0.9238
- Résultat obtenu au bout juste de 6 soumissions, nombre faible comparé au trois premiers de cette compétition respectivement (385, 295, 354).
- Le résultat jugé moyen car dut à une mauvaise organisation (méconnaissance de la plateforme kaggle, utilisation entière de la donnée, preprocessing de base, etc).
- La phase de preprocessing nécessitait davantage d'opérations avancées.
- Une connaissance accrue du fonctionnement du modèle XLM-Roberta aurait été un plus ainsi qu'une meilleure construction du modèle (partie classifieur).

## Sommaire

- Introduction
- Source de données
- Modélisation
- 4 Résultats et discussions
- Conclusion



## Conclusion

- Expérience enrichissante d'échange avec la communauté Kaggle.
- Fondamentaux des compétitions Kaggle acquis et contraintes.
- Utilisation de deux hub importants de modèles NLU et NLG : transformers de HuggingFace et TFHub de TensorFlow.
- Utilisation des TPUs : accélérer le temps d'exécution d'entraînement.
- Construire efficacement les modèles multilingues conversation IA
- Appropriation des opérations de pré-traitement de base
- Stratégie d'entrainement des modèles multilingues



# Perspectives

- Intégrer davantage les lois de distributions dans la modélisation entre les différents dataset.
- Utiliser la technique des n-grammes et des features linguistiques lors de la phase de preprocessing de la data.