Gautier Hoarau

Rapport : Classification de Commentaires Toxiques

Sommaire :

1. Contexte et Présentation du Projet

Les plateformes en ligne doivent gérer une grande quantité de commentaires laissés par les utilisateurs. Certains d'entre eux peuvent contenir des propos toxiques, ce qui pose un problème de modération. Une modération manuelle est coûteuse et inefficace à grande échelle. L'objectif de ce projet est donc de développer un modèle de classification de texte capable de détecter automatiquement les commentaires toxiques.

L'approche retenue repose sur le traitement automatique du langage naturel (NLP) et l'utilisation d'un modèle de deep learning basé sur des RNN (LSTM) pour analyser et classer les commentaires en toxiques ou non toxiques.

2. Jeu de Données Utilisé

Nous avons utilisé un jeu de données, qui contient des commentaires annotés selon plusieurs catégories de toxicité pour pouvoir entraîner notre modèle :

- Toxic : commentaire général toxique
- Severe toxic : commentaire particulièrement agressif
- Obscene : contenu obscène
- Threat : contenu menaçant
- Insult : insulte directe
- Identity hate : discours haineux basé sur l'identité

Dans notre approche, nous avons simplifié la tâche en regroupant ces catégories en une seule variable binaire (1 = toxique, 0 = non toxique).

3. Approche Utilisée

3.1 Prétraitement des Données

Avant d'entraîner un modèle de classification, nous avons appliqué plusieurs transformations sur les commentaires :

- Nettoyage du texte :
  - Passage en minuscules
  - Suppression de la ponctuation
- Suppression des stopwords :
  - Élimination des mots les plus courants n'apportant pas d'information (ex. "le", "de", "et").
- Tokenization et Vectorisation :
  - Transformation du texte en séquences numériques exploitables par un réseau de neurones.
  - Application d'un padding pour uniformiser la taille des séquences.

3.2 Construction du Modèle

Le modèle utilisé est un réseau de neurones récurrent (RNN) basé sur des LSTM (Long Short-Term Memory). Il est constitué des couches suivantes :

- Embedding : Convertit les mots en vecteurs de taille fixe pour capturer le sens du texte.
- LSTM (64 unités, return_sequences=True) : Capture la structure du texte sur plusieurs étapes.
- Dropout (0.2) : Réduit le surapprentissage.
- LSTM (32 unités) : Raffine l'apprentissage du texte.
- Dense (32 neurones, activation ReLU) : Couche intermédiaire.
- Dropout (0.2) : Encore une fois pour éviter l'overfitting.
- Dense (1 neurone, activation sigmoïde) : Produit une probabilité entre 0 et 1 (classification binaire).

Modèle :

```python
# Définition du modèle simple de réseau de neurones avec Keras
model = Sequential([
    # Couche d'embedding pour transformer les indices des mots en vecteurs
    Embedding(input_dim=max_words, output_dim=64, input_length=max_len),
    # Couche LSTM avec 64 neurones
    LSTM(64, return_sequences=True),
    Dropout(0.2),
    # Couche LSTM avec 32 neurones
    LSTM(32),
    # Couche dense avec 16 neurones et activation ReLU
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

3.3 Entraînement et Évaluation

L'ensemble des données a été divisé en deux parties afin d'assurer un apprentissage efficace et une évaluation pertinente du modèle. Ainsi, 80 % des données ont été consacrées à l'entraînement du modèle, lui permettant d'apprendre à différencier les commentaires toxiques des non-toxiques. Les 20 % restants ont été réservés pour tester et mesurer la capacité du modèle à généraliser ses prédictions sur de nouvelles données. Pour optimiser les performances du modèle, l'algorithme Adam a été utilisé afin d'ajuster progressivement les poids des neurones. Ce choix permet une mise à jour efficace des paramètres et améliore la convergence du modèle lors de l'entraînement. La fonction de coût employée pour ce modèle est la Binary Cross-Entropy, une métrique adaptée à la classification binaire. Elle permet d'évaluer la précision des prédictions en comparant les sorties du modèle aux valeurs réelles. Le modèle a été entraîné sur 5 époques, ce qui signifie qu'il a parcouru l'ensemble des données d'entraînement cinq fois pour ajuster progressivement ses paramètres. Ce nombre peut être ajusté en fonction des performances observées afin d'améliorer encore la qualité des prédictions.

Le modèle est évalué à l'aide des métriques suivantes :

- Accuracy (précision globale du modèle)
- F1-score (compromis entre précision et rappel)
- Matrice de confusion (analyse des faux positifs/négatifs)

4. Pipeline de Classification

Pour permettre une utilisation en temps réel, nous avons créé une pipeline de classification, capable de prendre une phrase brute en entrée et de retourner directement si elle est toxique ou non.

Les étapes du pipeline sont :

1. Nettoyage du texte
2. Suppression des stopwords
3. Tokenization et conversion en séquence numérique
4. Padding
5. Prédiction avec le modèle entraîné

Exemple de fonction de classification :

```python
# Fonction de prétraitement du texte et de prédiction de la toxicité
def preprocess_and_predict(text, tokenizer, model, max_len=100):
    # Nettoyage du texte
    text_cleaned = clean_text(text)
    # Stopwords
    text_cleaned = remove_stopwords(text_cleaned)
    # Conversion du texte en séquence d'indices de mots
    sequence = tokenizer.texts_to_sequences([text_cleaned])
    # Vérifie si le texte est vide
    if len(sequence[0]) == 0:
        return "Inconnu (aucun mot reconnu)"
    # Padding
    padded_sequence = pad_sequences(sequence, maxlen=max_len, padding='post', truncating='post')
    # Prédiction
    prediction = model.predict(padded_sequence)
    # Classification
    prob = prediction[0][0]
    classification = "Toxique" if prob > 0.5 else "Non toxique"
    # Retourne le résultat : "Toxique" si la probabilité est supérieure à 0.5, sinon "Non toxique"
    return f"{classification} (Score : {prob:.2f})"
```

5. Résultats et Analyse

Les résultats obtenus montrent que le modèle parvient à identifier efficacement les commentaires toxiques avec une précision satisfaisante. Cependant, certaines erreurs persistent, notamment la classification erronée de certains commentaires non toxiques comme toxiques, ce qui génère des faux positifs. De même, certains commentaires toxiques, en particulier ceux contenant des menaces implicites ou des formulations plus subtiles, ne sont pas toujours détectés, entraînant des faux négatifs. Pour améliorer ces performances, l'intégration d'un modèle pré-entraîné comme BERT permettrait d'affiner la compréhension du contexte et d'accroître la fiabilité des prédictions.

6. Améliorations Possibles

Voici quelques pistes pour améliorer mon modèle :

- Utilisation de modèles pré-entraînés : Intégration de BERT ou GPT pour une meilleure compréhension du contexte.
- Augmentation des données : Génération automatique de nouvelles phrases toxiques pour enrichir l'apprentissage.
- Optimisation des hyperparamètres : Ajustement du nombre de couches, de la taille des embeddings, et des taux de dropout.
- Approche multi-catégories : Plutôt que d'avoir une classification binaire, utiliser un modèle qui prédit plusieurs types de toxicité en parallèle.

7. Conclusion

Ce projet m'a permis d'utiliser des réseaux de neurones récurrents (LSTM) pour classifier automatiquement les commentaires toxiques avec une précision satisfaisante. Cette approche, basée sur le prétraitement du texte, la tokenization, et l'entraînement d'un modèle de deep learning, m'a permis d'obtenir de très bon résultats.

Cependant, certaines limitations persistent, notamment la difficulté à détecter des formes subtiles de toxicité et la présence de faux positifs. Des améliorations peuvent être envisagées, notamment en intégrant des modèles NLP pré-entraînés comme BERT et en optimisant les hyperparamètres du modèle.

Pour conclure ce type de solution pourrait être intégré dans des systèmes de modération automatique pour améliorer la sécurité des plateformes en ligne et réduire le temps nécessaire à la modération manuelle.