IFT3335 – TP2

Application des algorithmes d'apprentissage automatique à la classification de textes

Ce TP est à réaliser en groupe de 2-3 personnes
Il correspond à 15% de la note globale
Date de la remise : 19 avril avant 23:59.

Ce TP a pour but de pratiquer les algorithmes d'apprentissage automatique. Nous allons traiter le problème de classification de textes, qui vise à classifier chaque texte dans une des classes prédéfinies. Dans ce TP, on utilise la classification pour déterminer si un texte contient de la parole offensive. La plupart des algorithmes de classification sont déjà implantés dans la bibliothèque Scikit-Learn (scikit-learn: https://scikit-learn.org/stable/), ou peuvent être implantés avec une bonne base disponible (pour le modèle basé sur BERT). Votre travail principal dans ce TP consiste à utiliser Scikit-Learn et BERT sur une collection de données et à examiner la performance de ces algorithmes et l'impact des différents hyperparamètres des algorithmes.

1. La tâche de classification de textes

Dans cette tâche, on suppose qu'un ensemble de classes sont déjà définies, et qu'on dispose d'un ensemble d'exemples pour chaque classe. On veut entraîner un modèle de classification avec les exemples fournis, et ensuite appliquer le modèle aux nouveaux textes (test). La classification de textes est une tâche utilisée dans beaucoup d'applications : déterminer le sujet d'un article de presse (économie, sport, etc.), déterminer le sentiment véhiculé dans un commentaire sur un produit (positif/négatif), etc. Pour ce TP, le but est de construire des modèles pour détecter si un *tweet* sur Twitter contient un langage offensif. Comme l'entrée, on utilise seulement le texte brut du tweet.

Il y a plusieurs algorithmes d'apprentissage automatique pour la classification. Dans ce TP, on vous demande de tester les algorithmes présentés dans le cours et quelques variantes : Naive Bayes, arbre de décision, forêt aléatoire, SVM, perceptron multicouche, et un modèle basé sur BERT. Le but est de pratiquer l'application de ces algorithmes, et d'analyser les résultats obtenus pour voir les forces et faiblesses de différents algorithmes, ainsi que l'impact de leurs hyperparamètres.

2. Préparatifs

Les algorithmes de classification de textes classiques requièrent certains prétraitements afin de convertir un texte en un vecteur de caractéristiques (features). Une fois ces caractéristiques sont créées, les algorithmes d'apprentissages standard peuvent être utilisé pour déterminer la meilleure classe pour un texte. La plupart de ces algorithmes sont déjà implémentés dans Scikit-Learn, que vous pouvez utiliser dans ce TP.

Pour vous familiariser avec Scikit-Learn, on vous conseille de lire la documentation de Scikit-Learn en ligne : https://scikit-learn.org/stable/

Pour la classification basée sur BERT, un texte est d'abord converti en une représentation distribuée (un vecteur dense appelé plongement ou embedding). Ce vecteur sera utilisé comme l'entrée d'un perceptron pour faire la classification.

Dans les paragraphes suivants, on décrit le prétraitement que vous devez faire sur le texte avant d'appliquer un algorithme de classification.

1. Prétraitement

Le prétraitement permet de charger les textes à traiter et le transformer en des caractéristiques utilisables par les algorithmes. Ce prétraitement contient les étapes principales suivantes : tokenisation (pour découper un texte en mots), troncature de mots (pour transformer un mot en une forme standard), filtrage de mots outils inutiles, et pondération de mots. Il y a de différentes options : faire la troncature ou non, filtrer les mots outils ou non, différentes pondérations. Toutes ces options peuvent être utilisées dans un modèle. Les effets de chaque option ne sont pas toujours les mêmes. Cela dépend de la collection à traiter. Ainsi, on va vous demander de tester différentes options dans ce TP.

Pour commencer à vous familiariser avec le prétraitement, essayez pratiquer avec Scikit-Learn avec les données incluses dans le package (e.g. iris dataset, digits pour de la reconnaissance d'écriture…).

2. Sélection et pondération des mots comme caractéristiques (features)

En général, pour la classification de textes, chaque mot peut être considéré comme une caractéristique. Cependant, certaines méthodes peuvent être utilisées pour standardiser, sélectionner et pondérer les mots.

A) Convertir un texte en un vecteur, et appliquer une pondération

Si aucune de ces méthodes n'est appliquée, on va simplement transformer un texte en un vecteur de caractéristiques, où chaque mot est une caractéristique. Comme un mot peut apparaître plusieurs fois dans un texte, son importance peut être différente selon sa fréquence d'occurrences. Ainsi, il est approprié d'utiliser la fréquence pour pondérer l'importance d'un mot dans un texte. Cette transformation en un vecteur de caractéristiques et de pondérer les caractéristiques est réalisée dans Scikit-Learn grâce aux classes CountVectorizer, et TfidfVectorizer. La première classe permet de créer un vecteur de caractéristiques pondérées par leur fréquence, et la deuxième va pondérer les caractéristiques selon le schéma TF-IDF : TF – Term Frequency, IDF – Inverse Document Frequency. L'ajout du facteur IDF est souvent utile pour la classification de textes. Voir https://fr.wikipedia.org/wiki/TF-IDF pour une brève description de cette pondération.

Lisez les options proposées dans Scikit-Learn. Lors de votre choix, il vous est notamment possible de préciser si le résultat de cette transformation produit un ensemble d'attributs (mots) binaire (présent ou absent) ou avec un poids numérique (fréquence, tf transformé et avec idf).

B) Troncature de mots

Une pratique courante dans le domaine de classification de textes et de recherche d'information est de tronquer les mots pour ne garder que les racines. Par exemple, le mot « computer » sera tronqué en « comput ». Ceci a pour but de créer une représentation unique pour une famille de mots semblables (computer, computers, computing, compute, computes, computed, computation), ainsi standardiser ces mots similaires Il est supposé que les différences morphologiques ne changent pas le sens de ces mots. Ce processus est appelé « stemming » (troncature).

Il y a des méthodes de stemming standards disponibles en python, dont celle proposée par la bibliothèque NLTK (voir ici pour un tutoriel sur la troncature avec NLTK). Vous allez utiliser le stemmer Porter – le plus répandu pour cette tâche. Pour intégrer le

stemmer de NLTK, voir la page https://scikit-learn.org/stable/modules/feature_extraction.html.

C) Filtrage de mots outils

Certains mots dans un texte ne jouent qu'un rôle syntaxique, sans une signification particulière. Ces mots sont appelés « mots outils » ou *stopwords.* Par exemple, *of, and, the, …* en anglais sont des mots outils. Il est possible de ne pas les garder comme caractéristiques. Cette option est incluse dans les algorithmes Scikit-Learn.

3. Corpus à traiter pour le TP

Pour ce TP, nous allons utiliser un ensemble de tweets en anglais annotées en texte offensif (OFF) ou non (NOT). Ce sont des données utilisées dans la compétition sur la détection de textes offensifs – OffensEval dans l'atelier SemEval 2019

Voici quelques exemples en format tsv :

```
Doctors' interest in medical marijuana far outpaces available
research https://t.co/JLdEuhdyBG via @thecannifornian_com
https://t.co/qJIgzq4HiD    NOT   NULL  NULL

A must-read and a must-share for all your friends who support common
sense" gun control. https://t.co/hiCzzpbdjy"     NOT   NULL  NULL

@Jo2timess Now that's the dumbest shit I have ever heard in my
life!!!LIKE WTF!!!!   OFF   UNT   NULL

Agreed! When all of this drama was unfolding agents I know said
Comey had stacks of resignation letters piling up on his desk &amp;
no one wanted to interact with him anymore. When he let Clinton off
the hook there were agents throwing shit at their TV's! You are a
disgrace @Comey https://t.co/rZFBlgEWBy    OFF   UNT   NULL
```

Les 3 derniers éléments (en rouge) correspondent aux classes annotées pour différentes sous-tâches :

- A : Est-ce que le texte est offensif ? OFF-offensif / NOT – non-offensif.
- B : Si c'est offensif, est-ce qu'il cible une entité, ou non ? TIN – ciblé / UNT - non-ciblé
- C : Si l'offense est ciblée, qui est la cible ? IND – individu / GRP – groupe / ORG – organisation / OTH – autre.

Pour ce TP, on se limite à la sous-tâche A – déterminer si un texte est offensif (OFF) ou non (NOT).

Un article qui décrit l'expérimentation dans cette tâche dans SemEval est disponible ici : https://aclanthology.org/S19-2010.pdf. Vous pouvez voir le nombre d'exemples et de test pour chaque sous-tâche, ainsi qu'un résumé des résultats de différents participants (et des algorithmes typiques).

4. Entraînement d'un modèle de classification

En utilisant les données d'entraînement, on peut construire un modèle de Naive Bayes, d'arbre de décision, de forêt aléatoire, etc. La construction de modèles selon les algorithmes classiques sont déjà implantées dans Scikit-Learn. Il suffit d'appeler la fonction appropriée.

Pour la construction d'un modèle basé sur BERT, vous aurez besoin de télécharger un modèle BERT pré-entraîné (utiliser la version réduite *bert-base-uncased* pour faciliter la tâche). BERT va créer une représentation (embedding) pour le texte au complet, et cette représentation sera celle du jeton [CLS]. Vous devez ajouter un perceptron par-dessus, en utilisant [CLS] comme entrée, pour déterminer une classe. L'utilisation de BERT pour la classification sera présentée à une séance de démonstration.

Le modèle entraîné sera appliqué aux données de test pour déterminer la performance du modèle (exactitude et score de F1).

5. Les tâches à réaliser

1. **Préparatif :**
   Vous devez faire un programme capable d'extraire les caractéristiques à partir des textes annotés. Réfléchissez et testez en particulier le choix des éléments à utiliser comme caractéristique pour la classification.
2. **Tâche de base**
   Vous devez tester la performance de différents algorithmes de classification. Pour ce TP, on vous demande de tester les algorithmes suivants disponibles dans Scikit-Learn (voir aussi le cours MOOC):
   a. Naive Bayes : multinomial https://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes
   b. Arbre de décision : https://scikit-learn.org/stable/modules/tree.html#classification
   c. Forêt aléatoire : https://scikit-learn.org/stable/modules/ensemble.html#forests-of-randomized-trees
   d. SVM : https://scikit-learn.org/stable/modules/svm.html
   e. et MultiLayerPerceptron (à une couche cachée) : https://scikit-learn.org/stable/modules/neural_networks_supervised.html#multi-layer-perceptron
3. **Tâches d'exploration**
   Les algorithmes de classifications contiennent des options. Vous êtes encouragés à explorer différentes options pour voir leurs effets dans le test. Les options principales que vous devez tester sont :
   - Naive Bayes : modèle multinomial ou Gaussien
   - Arbre de décision : avec ou sans limite de profondeur, critère *gini*/*entropy*
   - Forêt aléatoire : nombre d'arbre, limite de profondeur
   - SVM : linéaire ou avec noyau *rbf*
   - Perceptron : nombre de neurones dans la couche cachée (tester quelques variations), fonction d'activation (*logistic* ou *relu*)
4. **Méthode basée sur l'apprentissage profond (optionnelle – bonus)**
   En plus des méthodes d'apprentissage automatique classiques, on utilise de plus en plus l'apprentissage profond pour des tâches en traitement de la langue naturelle. Notamment, les études récentes exploitent généralement un modèle pré-entraîné comme BERT. Dans cette tâche, vous devez construire un modèle de classification avec un BERT pré-entraîné. (Voir la présentation dans la séance de démonstration)
5. **Analyse dans un rapport**
   Analysez les résultats de classification avec différents algorithmes, et comparer leur performance. Comme réflexion globale, vous êtes demandé à analyser la

performance des algorithmes, d'observer leurs forces et faiblesses, ainsi que l'effet potentiel des hyperparamètres.

Les tâches qu'on vous demande de réaliser sont déjà implémentées sur ce Github : https://github.com/ZeyadZanaty/offenseval. Vous pouvez y retrouver les données d'entraînement pour différentes sous-tâches, les données de test, le programme pour la lecture de données, l'entraînement de modèles de classification, ainsi que le test sur les données de test. Un rapport décrivant ce qui est réalisé est disponible sur ce site : https://github.com/ZeyadZanaty/offenseval/blob/master/docs/OFFENSEVAL.pdf.
Dans ce rapport, l'auteur a utilisé la validation croisée (cross validation) pour chercher les meilleurs hyperparamètres. Pour ce TP, on ne vise pas à identifier les meilleurs hyperparamètres, mais d'observer l'impact de ces paramètres sur le résultat.

Dans cette collection de données, seules les données d'entraînement ont été annotées des classes. Les données de test ne sont pas annotées. Ainsi, vous devez utiliser une partie (70%) de données d'entraînement pour l'entraînement, et le reste (30%) pour le test. Vous pouvez utiliser la fonction *split* pour diviser les données d'entrainement annotées en 70% et 30%. Pour permettre de comparer entre différents modèles, on vous conseille de faire cette division d'abord, et utiliser la même division pour l'entraînement et le test de chaque modèle.

Le code fourni sur ce Github n'est plus fonctionnel maintenant. Cependant, il vous donne un bon exemple sur la façon d'utiliser Scikit-Learn et des modèles d'apprentissage profond (CNN et LSTM) pour la classification. Vous allez utiliser une partie de ces algorithmes et utiliser BERT sur Google Colab.

6. Remise

Vous devez rendre tous les programmes que vous avez construits. Ils doivent fonctionner avec Python 3.10 ou ultérieur.

Lors de la correction, vos programmes seront lancés tels que reçus, sans modification. Assurez-vous donc qu'ils tournent sans que le correcteur ait à modifier le code (commenter des lignes, modifier des lignes, changer des chemins de fichiers, etc.). Vous pouvez par exemple faire un fichier par question, ou avoir un seul programme central qui prend un argument (p.ex. « tp2.py svm »), c'est votre choix. Incluez tous les fichiers de données nécessaires à votre code dans votre remise.

Si vous utilisez Colab, téléchargez le notebook Jupyter (.ipynb) et incluez-le dans votre remise.

En plus des programmes, vous devez aussi rendre un rapport, de longueur d'environ 5 pages. Allouez une petite portion de votre rapport pour donner une description de ces programmes et de leur utilisation. Dans votre rapport, vous devez décrire votre prétraitement avec différentes options, les expériences que vous avez réalisées, les résultats obtenus, et enfin des comparaisons et des analyses sur les résultats. Vos analyses doivent porter, au minimum, sur la performance des différents algorithmes, l'impact de différentes options indiquées dans les tâches d'exploration. Vous êtes encouragés à faire des analyses sur d'autres aspects que vous jugez intéressants.

7. Barèmes d'évaluation

Ce TP correspond à 15% de la note globale. Voici les critères utilisés pour noter ce TP :

- Implémentation de modèles classique, et le test sur les données avec Scikit-Learn: 2 points sur chacun des algorithmes suivants :
    - Naïve Bayes
    - arbre de décision
    - forêt aléatoire
    - SVM
    - MultiLayerPerceptron
- Bonus : Implémentation et tests du modèle basé sur BERT : 2 points de bonus
- Rapport : 5 points (on tient compte de la description, les analyses et comparaisons entre différents algorithmes et options, les analyses de résultats et les conclusions. La bonne structure et la clarté de description sont attendues.)

Vous devez former des groupes de 2-3 personnes. Exceptionnellement, une personne peut travailler seul avec une justification raisonnable et ce doit être approuvé par le professeur. Si vous remettez ce travail seul(e) sans l'autorisation préalable du professeur, vous serez pénalisé(e).