Extraction et programmation statistique de l'information¶

Rapport de projet :¶
Prédiction de sentiments de tweets¶

Introduction¶

Ce projet a pour objectif de réaliser un modèle de prédiction du sentiments de tweets anglophones, selon 4 catégories distinctes : '*irrelevant*', *'positive'*, *'neutre'* et *'négative'*. Ce projet est réalisé en Python 3.8.

Méthode d'évaluation

Un modèle de prédiction sera évalué par plusieurs métriques:

- La précision moyenne sur le jeu d'entraînement
- La précision moyenne et la variance sur le jeu de validation (Cross validation)
- La matrice de confusion sur le jeu d'entraînement

L'intérêt de disposer de la métrique de précision sur plusieurs jeux est de pouvoir détecter les cas de surapprentissage. Le score de cross validation permet d'effectuer la sélection d'hyper paramètres d'un modèle ou d'effectuer le choix entre plusieurs modèles.

↡¶

Tests et résultats

Baseline:

Afin de juger correctement les modèles de prédiction, il est important d'avoir une idée des performances facilement atteignables par un modèle très simple. Pour cela on utilise une simple tokenisation par espace, avec un modèle bayésien naïf en bag-of-word.

performances:

Précision moyenne (train)	Précision moyenne (CV)	Variance de précision (CV)
87%	75%	10^-4

Notebook associé: **Baseline.ipynb**

Prétraitement de données :

Plusieurs types communs de Transformers sont utilisés. On appelle Transformer les foncteurs chargés de transformer les données avant l'apprentissage/classification.

Filter

Les Filter permettent de supprimer des mots du tweet.

HashtagFilter : Retire tous les hashtags (toute suite de lettres après #).

MentionFilter: Retire les mentions (toute suite de lettres après @).

URLFilter : Retire les url.

StopWordFilter : Retire les mots communs de syntaxe sans intérêt sémantique.

Flagger

Les Flagger fonctionnent de façon très similaire aux Filter, mais au lieu de supprimer les mots, ils sont remplacés par un générique <Type>.

MentionFlagger : Toute mention est remplacée par <mention>.

NumberFlagger : Tout nombre (écriture numérique seulement, “2” est transformé mais pas “two”) est remplacé par <number>.

URLFlagger : Toute url est changée par <url>.

Autres

LowerCaseTransformer : Change simplement la casse du texte en minuscule.

NegationTransformer : Détecte les négations (mots finissant par “n’t”) et transforme ces mots en “not”. L’intérêt de ce transformer réside surtout pour les modèles utilisant des bi-gram (ou plus).

SplitterPunctuation : Tokenizer sur les ponctuations. Change les tweets du type string en liste de tokens.

HashtagToWords : Change les hashtags en phrases en séparant les mots par majuscules. Exemple : #HelloWorldTwitter sera converti en “Hello World Twitter”

Traitement du cas “irrelevant” : prédiction de la langue

Parmi les catégories à classifier, on distingue deux groupes particuliers:

- Les tweets qui ne sont pas pertinents “*irrelevant*”
- Les trois autres classes neu, neg, pos

On peut supposer que identifier les tweets pertinents est très différent de la détection de sentiment, car l’on se base sur des règles arbitraires qui n’ont pas forcément de lien avec le sentiment:

- On considère les tweets non anglophones comme non pertinents
- Le spam est également considéré comme non pertinents (ex: plusieurs url sans message)

Il est alors possible d'imaginer que deux modèles travaillant sur deux tâches séparées seront plus efficaces qu'un seul gros modèle qui doit tout apprendre par lui-même.

La stratégie choisie ici est d'éliminer d'office les tweets qui ne sont pas écrits en anglais. En effet, conserver ces tweets oblige le modèle de prédiction à apprendre à reconnaître les mots étrangers, ce qui augmente considérablement la taille du vocabulaire. On utilise la bibliothèque langdetect[1] qui est un portage de la bibliothèque Java language-detection.

Deux façons de traiter le problème ont été explorées : filtrer les tweets *irrelevant*, et les identifier en chaînage de modèle.

Filtrage des tweets étrangers

Le filtrage correspond à la séparation des tâches, si un tweet est détecté comme n'étant pas écrit en anglais, il sera prédit comme *'irrelevant'* immédiatement.

Performance de la détection des langues:

L'évaluation de la détection de la langue est difficile à effectuer puisque l'on ne dispose pas des étiquettes de langage associées aux tweets. On peut cependant afficher une matrice de confusion :

	irr	oth
irr	0.794816	0.205184
oth	0.040948	0.959052

Tableau: Matrice de confusion pour langdetect

[1] Mimino666/langdetect: Port of Google's language-detection library to Python, https://github.com/Mimino666/langdetect.

La classe “oth” représente toutes les classes différentes de *irrelevant*. Seule une case est facile à interpréter, la case [oth, irr] représente les tweets filtrés par erreur car détectés comme étrangers alors qu’ils étaient anglophones, ici **langdetect** se trompe 4% du temps sur les tweets pertinents. Les autres cases sont difficiles à interpréter car un tweet peut être anglophone mais *irrelevant*. Lorsque **langdetect** laisse passer un tweet anglais non pertinent, la matrice de confusion va le compter comme une erreur (on va augmenter [irr, oth]), mais il s’agit du comportement attendu pour ce prédicteur.

langdetect est perturbé par le format des tweets, et aura tendance à détecter comme anglais un tweet possédant des hashtags (souvent anglophone dans le corpus comme #Twitter). Exemple : “Buenas noches a todos #Twitter off” est détecté comme anglais.

Une possible solution est d’enlever les hashtags puisqu’ils introduisent du bruit dans les tweets. Sur cet exemple en particulier, **langdetect** comprend le tweet comme espagnol si #Twitter est enlevé. Puisque l’on ne possède pas les étiquettes réelles, il est cependant difficile de déterminer si cette solution marche de façon globale. On peut cependant réexaminer la matrice de confusion.

	irr	oth
irr	0.873290	0.126710
oth	0.083693	0.916307

Tableau : Matrice de confusion avec filtrage des hashtags

On voit qu’avec cette solution, **langdetect** capture davantage de *irrelevant* (on passe de 79% à 87%), cependant on gagne également 4% en tweets pertinents filtrés par erreur. En moyenne on semble donc gagnant avec cette stratégie.

Résultats en modélisation:

On peut tenter d’appliquer la méthode à la baseline pour observer l’impact sur les performances.

Modèle	Précision moyenne train	Précision moyenne CV 5-fold
Baseline	0.873	0.748
Filtrage *irrelevant* + Baseline	0.861	0.755

Tableau des performance baseline vs baseline avec Filtrage des irrelevant

La méthode fait gagner ~0.7% sur le score de cross validation.

Prédiction de la langue avec chaînage de modèle

On a pu observer avec le filtrage des tweets étrangers que l'on avait des faux positifs et faux négatifs sans possibilité de les corriger via le prédicteur de langue. On souhaite éviter de rejeter des tweets anglais. Une possible solution est de chaîner les différents modèles de sorte à fournir au modèle prédictif le langage détecté du tweet. Dans l'idéal on peut développer un modèle qui sache rattraper les erreurs communes du détecteur de langage.

Dans notre situation, nous n'avons pas besoin de réaliser l'entraînement sur plusieurs sous-ensembles de données car **langdetect** ne se *fit* pas aux données, ses prédictions sont toujours les mêmes.

Résultats en modélisation:

Comme avec la méthode précédente on applique cette solution à la baseline pour observer le résultat:

Modèle	Précision moyenne train	Précision moyenne CV 5-fold
Baseline	0.873	0.748
Filtrage *irrelevant* + Baseline	0.861	0.755
Détection de langue + Baseline	0.863	0.754

Tableau des performance baseline vs baseline avec détection de langue en chaînage

Les performances sont similaires lorsqu'on utilise la baseline.

Cependant, lors de l'utilisation des transformations avec de véritables pipelines de prétraitement, on remarque un très grand overfit lorsqu'on utilise la méthode par chaînage. C'est pourquoi nous utiliserons le filtrage lors de l'optimisation des hyper paramètres des modèles.

Modèles réseaux de neurones :

- Bert :

Pour aller plus loin, on a décidé d'utiliser un modèle BERT, il s'agit d'un modèle NLP pré-entrainé sur un large corpus anglais (non labellisé) auquel on va ajouter une nouvelle couche qui représente nos tweets et l'entraîner là dessus. Ce modèle produit les résultats suivants, bien qu'il est long à réentraîner :

Précision Moyenne (train)	Précision Moyenne (validation)
90%	82%