Algorithmes du Texte et Recherche d'Information : Projet

Florian Falkner
Matthias Foyer

Université de Strasbourg — 18 mai 2021

Les Données

Les données reprennent des textes de discours politiques, ainsi que leurs groupes politiques associés et la date à laquelle ils ont été prononcés.

1 Étape 1

Le fichier IPYNB ainsi que le CSV "debats" sont accessibles dans le dossier "Partie 1".

La structure générale de notre implémentation se base sur les travaux faits durant les TP d'Algorithmes du Texte et Recherche d'Information.

1.1 Pré-traitement

La première partie est consacrée au pré-traitement des données. Après l'importation classique du fichier CSV et son nettoyage (suppression des données aux valeurs manquantes), nous avons substitué les groupes politiques à leurs orientations respectives (en ne gardant que les groupes politiques mentionnés dans le sujet). Les données sont ensuite séparées en jeux d'entrée (X) et de sortie (y) puis un "split" est effectué pour extraire un jeu de test contenant 20% des données (permettant la validation croisée).

1.1.1 Implémentation

Après avoir effectué les tests sur différentes méthodes, nous avons choisi de sélectionner un "stemmer" qui nous assurait les meilleurs résultats. Il permet d'enlever le suffixe des mots constituants les discours étudiés. Bien que son implémentation française ne soit pas optimale, les performances générales en restaient notablement améliorées. Lors de son utilisation, il effectue également une suppression des caractères seuls et spéciaux, ainsi que des nombres. Nous avons fait ce choix car nous considérons que ces caractères n'apportent aucune valeur sémantique, et n'aident donc pas notre modèle à repérer des structures logiques.

Enfin, le résultat est "tokenisé". C'est-à-dire que le texte est découpé en "tokens", ici les mots. Cela nous permet de le "vectoriser" afin qu'il soit interprétable par notre modèle.

Ces deux dernières étapes sont éxecutées lors de l'appel à l'objet "TfidfVectorizer" avec quelques paramètres supplémentaires. D'abord, les mots courants français sont ignorés (stopwords) pour éviter de surcharger notre input de données inutiles. Des fréquences d'apparitions minimales et maximales pour les mots sont définies pour éviter de prendre en compte les mots trop rares ou au contraire trop fréquents. Le

paramètre “ngram_range” agit comme une fenêtre glissante (de taille 2 dans notre cas) pour conserver le contexte des phrases lorsque la sémantique dépend de l’association de plusieurs mots. Enfin, la casse des textes est modifiée pour que tous les caractères soient minuscules (et qu’ils n’induisent donc pas plusieurs valeurs de vectorisation pour une seule et même lettre).

Ces étapes sont effectuées à travers une “pipeline” que nous pourront appliquer à de nouvelles données facilement si nécessaire.

1.1.2 Expériences Réalisées

Pour aboutir à ce résultat nous avons effectué les essais suivants :

- Utilisation de la Lemmatization, du Stemming (2 bibliothèques différentes), et sans rien (le Stemmer était le plus efficace)
- Ajout de la prise en compte des statistiques (“stats_pipeline” lors des TPs) mais n’a pas amélioré les résultats
- Différents paramètres pour Tfidf Vectorizer (fréquences de mots et ngram_range)

1.2 Modèles et Apprentissage

Nous avons exécuté plusieurs modèles afin de comparer leurs performances. Nous récupérons le score et la précision de chaque modèle, qu’un graphique permet ensuite de lire facilement.

1.2.1 Implémentation Retenue

Le modèle le plus efficace était dans notre cas la Regression Logistique, que nous utilisons donc dans la suite du projet. Une validation croisée permet de nous assurer que nous pouvons faire confiance à ces précisions en les confrontant à un jeu non-utilisé lors de l’apprentissage.

Pour affiner les ajustements des paramètres de notre modèle, nous avons utilisé un gridsearch. Il nous a permis de nous approcher des valeurs optimales pour les paramètres “C” (inverse de la force de régularisation) et de poids des différentes classes. Nous avons ainsi gagné encore en précision.

Le tout est ordonné dans une pipeline (pré-traitement puis modèle).

1.2.2 Expériences Réalisées

Pour aboutir à ce résultat nous avons effectué les essais suivants :

- Les modèles : Baseline, Multinomial NB, CART, LR, KNN, Random Forest, Linear SVC
- La gridsearch sur les paramètres : C, poids

Ensuite, des tests manuels ont été effectués pour régler les paramètres dans les derniers détails.

1.3 Performances

Nous validons notre modèle en étudiant sa précision sur un jeu de données “test” non-utilisé lors de l’apprentissage. Les performances de notre modèle sur ces données doivent être représentatives d’un cas

réel de prédiction sur de nouvelles données. Dans notre cas cette précision est satisfaisante puisqu'elle est égale à celle constatée sur le jeu d'apprentissage à **66%** en moyenne.

Dans sa première version, sans optimisation des choix et paramètres, notre modèle offrait une précision d'environ 52%. Les recherches et différents essais nous ont donc permis d'améliorer notre précision de 14%.

1.4 Problèmes Observés

La précision semble plafonner à 67% quoi que l'on fasse. Nous estimons que les termes utilisés dans les discours sont trop proches d'un groupe politique à l'autre, certainement parce qu'ils traitent des mêmes sujets. Le modèle n'est de toute évidence pas capable d'en extraire suffisamment la sémantique globale / le sens abstrait / l'orientation générale.

2 Étape 2

Le projet SOLR est accessible depuis le dossier "Partie 2" tout comme le CSV modifié pour cette partie.

Une fois le projet lancé (à partir de notre dossier SOLR), l'interface est accessible à partir de ce lien : `http://localhost:8983/solr/debats/browse`

2.1 Implémentation

La construction de notre projet SOLR s'est faite en grande partie selon les étapes détaillées dans le TP associé.

Le fichier CSV contenant les données a été modifié pour convenir à nos attentes :

- Suppression de la colonne inutile "Unnamed: 0" (représentant les index, mais que nous rajoutons par la suite avec un nom de colonne adéquat)
- Suppression des lignes comportant des données manquantes
- Pour assurer la cohérence du projet, nous n'avons conservé que le groupes politiques qui nous intéressent (cités dans le sujet)
- Ajout d'une colonne indiquant l'orientation politique pour chaque ligne (en fonction du groupe politique, comme dans la partie 1 du projet)
- Re-formatage des dates (pour conserver uniquement l'année)
- Ajout de la colonne des index (servira pour générer des URL courtes et logiques par la suite)

Nous avons fait le choix de conserver les groupes politiques. Puisqu'ils sont renseignés dans le dataset original, il nous semblait judicieux de garder cette information. Elle vient enrichir le degré de détail de nos données et permettent un filtrage encore plus précis si nécessaire.

3 Répartition du Travail

Le développement devant se faire une tâche après l'autre, nous avons travaillé en simultané la plupart du temps. Nous avons utilisé une plateforme de discussion vocale ainsi que Google Colab pour travailler en même temps sur le même travail. Nous avons ainsi confronté constamment nos avis pour aboutir aux

meilleurs choix possibles à chaque étape du projet. Quand nécessaire, nous avons également fait appel à des partages d'écrans à distance pour suivre en direct ce que notre binôme faisait. Ce fût utile en particulier pour la seconde partie où le développement ne pouvait pas se faire sur Google Colab.

Références

[1] *Classifier comparison*. URL: https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html (visited on 05/18/2020).

[2] *Gitunistra TP Markdown*. URL: https://git.unistra.fr/ruizfabo/ri_atexte/-/tree/master/tp (visited on 05/18/2020).

[3] *Hyperparameter Tuning with GridSearchCV*. URL: https://www.mygreatlearning.com/blog/gridsearchcv/ (visited on 05/18/2020).

[4] *Lemmatize French Text*. URL: https://stackoverflow.com/questions/13131139/lemmatize-french-text (visited on 05/18/2020).

[5] *Logistic Regression*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (visited on 05/18/2020).

[6] *Parameter estimation using grid search with cross-validation*. URL: https://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_digits.html (visited on 05/18/2020).

[7] *Partie TP1 Google Colab*. URL: https://bit.ly/2Rvv2EF (visited on 05/18/2020).

[8] *Pipelines & Custom Transformers in scikit-learn: The step-by-step guide (with Python code)*. URL: https://towardsdatascience.com/pipelines-custom-transformers-in-scikit-learn-the-step-by-step-guide-with-python-code-4a7d9b068156 (visited on 05/18/2020).

[9] *POD Réalisation compléte TP*. URL: https://pod.unistra.fr/video/40692-realisation-complete-du-tp-pas-a-pas-avec-solr/ (visited on 05/18/2020).

[10] *PYTHON SKLEARN PRE-PROCESSING + PIPELINE (22/30)*. URL: https://www.youtube.com/watch?v=OGWwzm304Xs (visited on 05/18/2020).

[11] *Stemming and Lemmatization in Python*. URL: https://www.datacamp.com/community/tutorials/stemming-lemmatization-python (visited on 05/18/2020).

[12] *Tuning the hyper-parameters of an estimator*. URL: https://scikit-learn.org/stable/modules/grid_search.html (visited on 05/18/2020).