Rapport de projet d'Algorithme de texte

Leonardo NASSABAIN et Karim DELLALI

1 Introduction

Pour ce projet nous devions réaliser un système de recherche d'information dans une collection de débats de l'Assemblée nationale en France. Le travail se composait de deux étapes. La première étape consistait à préparer les données, à les étiqueter correctement puis de developper à l'aide des librairies scikit-learn un outil de classification automatique des tours de parole selon l'orientation politique. Et pour la deuxième étape il nous était demandé d'indexer les différents tours de parole fournis dans une instance du serveur de recherche Solr, puis de créer une interface de recherche en texte intégral qui fournira également de filtrer les données selon l'orientation politique et les années.

Ce travail était effectué en binôme. Les tâches étaient réparties de manière équilibrée. Karim travaillait sur l'étape 1 et Leonardo travaillait sur l'étape 2 du projet. A la rencontre d'un problème, on s'entraidait.

2 Etape 1

Cette première étape pour le projet, concernait la mise en place d'un modèle, dans le but de prédire l'orientation politique des différents partie à partir de tours de parole de différents groupes politiques.

Nous avions à disposition un fichier csv, contenant dans la date de la séance à l'assemblée nationale, le "contenu" textuel du débat, ainsi que le groupe politique qui était à la parole. Dans un premier temps nous avons traité les données, en supprimant les mauvaises colonnes, puis on a seulement garder 4 parties politiques, qui étaient SOC, GDR, LR, et FI.

Le but n'était pas de prédire le groupe politique auteur du débat, mais de prédire à l'aide de notre modèle, l'orientation politique du groupe en question. Pour cela on a également du supprimer la colonne groupe politique, pour avoir une colonne orientation, qui correspondant à "droite" ou "gauche" en fonction du parti. Après avoir pré-traité nos données, nous étions donc en mesure de pouvoir débuter la construction de notre modèle.

Les données traitées serviront également dans la partie 2, pour cela on a crée un nouveau fichier csv, contenant les nouvelles colonnes, et avec le "nettoyage" de celles-ci.

On sépare les données en attributs et classes, où notre seul attribut est le texte du tour deparole, et la classe devant être prédit étant l'orientation politique.

Tout d'abord, il a fallu compter sur les bonnes pratiques nécessaire pour élaborer notre modèle, c'est à dire le traitement du texte avant l'apprentissage sur celui-ci. Nous avons procédé à une étape de **tokénisation** puis de **lemmatisation**. En effet, représenter le texte sous forme de tokens est nécessaire pour notre modèle, et concernant la lemmatisation, il est utile de la mettre en pratique pour des langues hautement

fléchies, ce qui est le cas pour le français et nous avons donc pensé qu'il pourait être utile de rajouter cette étape.

Une fois cela fait, on avait plus qu'à transformer notre texte en vecteur pour le pipeline spécifique à la colonne texte, qui sera d'ailleurs le seul de notre modèle, car on cherche à prédire la classe seulement à partir de cette colonne.

A noter que le nombres de traits extraits à partir de la colonne texte était de 790.

Le fait que nous ne prédisons l'orientation politique qu'à partir d'un seul attribut, fait qu'on pourra avoir des problèmes de précisions. Dans ce cas, il a été nécessaire de tester, éxperimenter notre modèle dans le but d'améliorer sa précision.

Nous avions donc choisi deux modèles, qui avaient une précision assez bonne, un modèle basé sur l'algorithme d'apprentissage **RandomClassifier** et un autre sur **Multinomial NB**

Étant donné que nous étions face à un problème de classification, nous avons également procédé à une **validation croisée stratifiée**. Pour illustrer cela, vous trouverez à disposition une partie contenant les modèles sans validation et avec dans notre rendu. On a pu voir une légère augmentation de la précision de nos modèles quand on appliquait celle-ci au préalable.

2.1 Comparaison des modèles

Ces deux modèles, ils arrivaient plus ou moins à prédire correctement l'orientation politique, avec un léger avantage pour le modèle basé sur l'Algorithme d'apprentissage RandomClassifier.

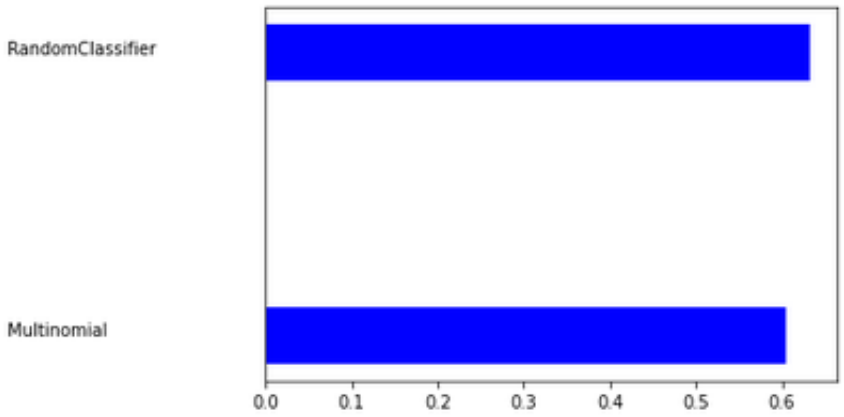


Figure 1: Graphique de comparaison des deux modèles

On peut observer que la précision du modèle basé sur le RandomClassifier est supérieure d'environ 3%, une marge non négligeable quand on a du mal à améliorer la précision de notre modèle. Nous étions à l'origine parti sur un modèle basé sur le Multinomial, mais suite à l'expérience faite, nous avions choisi le RandomClassifier, en gardant aussi l'autre modèle dans le but d'illustrer et mettre en valeur la précision de celui-ci.

Il a été difficile d'améliorer la précision de notre modèle, étant donné qu'on ne se basait que sur une colonne, bien que la lemmatisation nous a été utile, ainsi que la validation croisée stratifiée. Nous n'avions pas eu à la base ces résultats et en mettant en pratique ces étapes, on a pu obtenir un gain de précision non négligeable.

La précision du modèle que l'on garde, le RandomClassifier, est d'un peu plus de 63,5%

2.2 Analyse des résultats

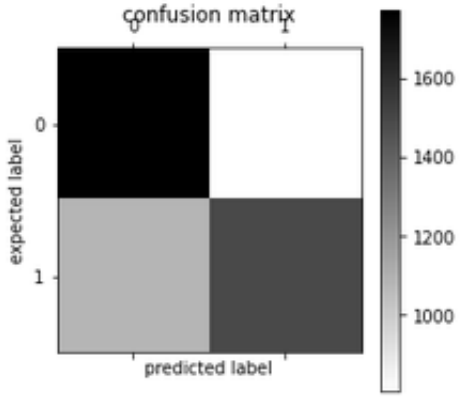


Figure 2: Matrice de confusion du modèle avec Multinomial

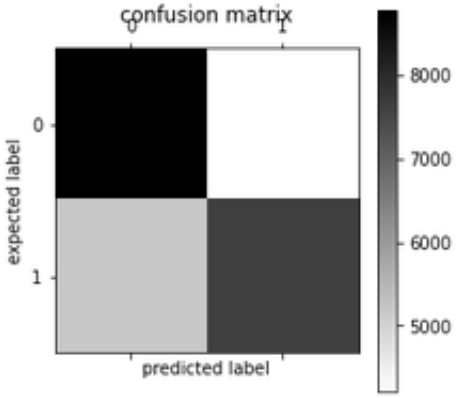


Figure 3: Matrice de confusion du modèle avec RandomClassifier

L'etiquette 0 représentant la "gauche" et l'étiquette 1 représentant la "droite", nous pouvons voir dans ces deux matrices de confusion que les discours qui sont de "gauche" sont prédits de manière fiable car la case de coordonnées (0,0) noire et la case de coordonnées (0,1) est blanche. Cela veut dire que quand il s'agit d'un discours de "gauche" le classement se trompe rarement.
Cependant, pour les discours de "droite", la prédiction n'est pas aussi fiable, le taux d'erreur étant relativement élevé.

Dans les deux modèles, la majorité d'erreurs provient donc d'un classement à gauche des discours qui sont de droite.
On peut observer une meilleure précision (bien que légère) pour les textes de droites pour la matrice de confusion du modèle utilisant le RandomClassifier, comme illustré dans la partie précédente où celle-ci était supérieure d'environ 3%.
Nous pouvons déduire à partir de ces résultats que les termes employés dans les discours des personnes orientées politiquement à gauche sont plus faciles à classer, donc il y a des caractéristiques plus fortes et plus saillantes dans le vocabulaire utilisé que dans les discours de personnes orientées à droite. Les personnes à droite utilisent probablement un vocabulaire proche du celui utilisé à gauche.

3 Etape 2

Pour l'étape 2, nous avons commencé par nous connecter à Solr, puis par créer un "core" appelé "debats" avec la commande `create_core`. Ensuite, pour l'indexation nous avions le choix entre deux méthodes. Dans l'étape 1 du projet un fichier CSV nettoyé était crée, sans NaN et sans colonnes et instances indésirables.A partir de ce fichier CSV, nous aurions pu transformer chaque instance en fichier XML puis de suivre la méthode qui était éxpliquée en TP en indexant chaque instance (fichier XML) un par un.
Cependant, après avoir cherché s'il n'éxistait pas de méthode plus élégante, nous sommes tombés sur une méthode rendant possible l'indexation directement à partir d'un fichier CSV. Donc l'indexation était faite avec la commande suivante:

```
curl 'http://localhost:8983/solr/debats/update?commit=true' --data-binary @debats_transf.csv
    -H 'Content-type:application/csv'
```

debats étant le core Solr dans lequel nous travaillons et `debats_transf.csv` étant le fichier CSV obtenu dans l'étape 1 du projet. Nous remarquons par ailleur qu'après cette commande, il n'est pas nécessaire d'effectuer une commande pour faire un **commit** des changements car au sein de la commande nous voyons la partie `"update?commit=true"` qui nous permet d'effectuer l'indexation et le commit en une seule commande.
Une fois que les données étaient indéxées, il était question d'améliorer le fonctionnement des recherches et l'apparence de l'interface afin de rendre l'utilisation plus intuitive et facile.
Le corpus étant en français il fallait donc changer la langue des champs textuels en français car elle est en anglais par défaut. Pour faire cela, dans le fichier **managed-schema**, il fallait retrouver les champs textuels et changer leur type de **text_general** à **text_fr**.

Or dans notre jeu de données, nous avons 3 champs:
dateSeance: l'année quand la prise de parole était faite
texte: les paroles composant la prise de parole
orientation: l'orientation politique de la personne faisant le discours

Donc nous avions un seul champs textuel **texte** dont le type était donc à modifier. Une fois que cela était fait, il fallait supprimer les contenus du core afin de pouvoir reindexer correctement les données. La suppression était faite en utilisant les commandes suivantes, fournies préalablement en TP:

Pour autoriser la suppression du contenu, il faut faire la commande:

```
curl -H 'Content-type:application/json' -d '{"set-property": {"requestDispatcher.requestParsers
    .enableRemoteStreaming": true}, "set-property": {"requestDispatcher.requestParsers.
    enableStreamBody": true}}' http://localhost:8983/api/cores/debats/config
```

Ensuite on supprimme les données avec les commandes suivantes:

```
curl "http://localhost:8983/solr/debats/update?stream.body=%3Cdelete%3E%3Cquery%3E*:*%3C/query
    %3E%3C/delete%3E"
```

```
curl "http://localhost:8983/solr/debats/update?commit=true"
```

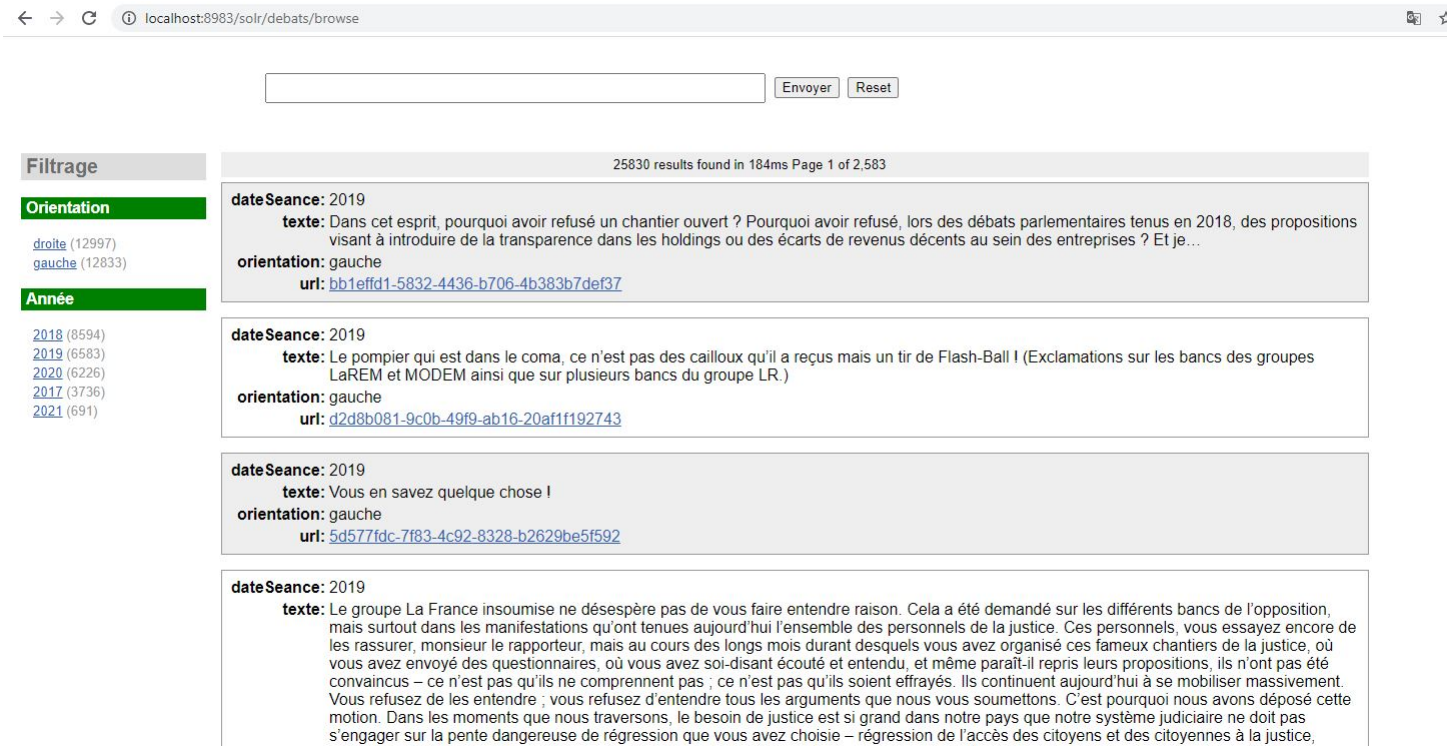
Les données supprimées, nous pouvons de nouveau les indexer en utilisant la commande expliquée plus haut. Maintenant, nous pouvons faire des requêtes avec des résultats corrects mais l'interface reste basique,

et pour l'améliorer il va falloir faire des modifications dans le fichier **solrconfig.xml** qui se trouve dans le dossier **conf** de notre core Solr. Les modifications qui étaient faites dans notre fichier sont similaires aux modifications qui étaient faites lors du TP.

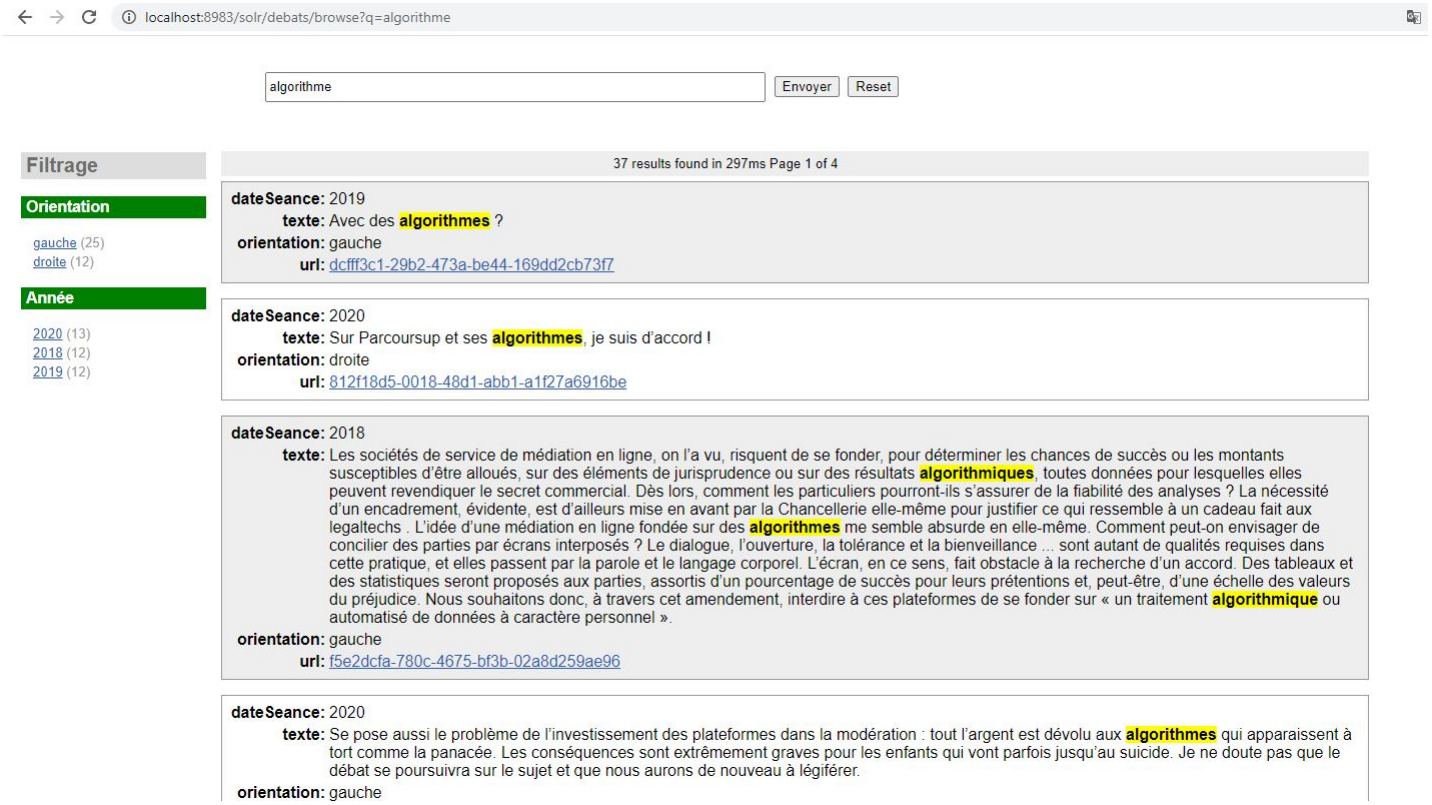
Nous avons d'abord défini le champ **texte** comme le champ de recherche par défaut, pour éviter de devoir préciser le champ à chaque requête. Pour faire cela nous avons introduit la ligne suivante dans **solrconfig.xml**:
`<str name="df">texte</str>`
Ensuite nous avons utilisé le code fourni en TP pour introduire les facettes et la mise en surbrillance en modifiant uniquement le champ **hl.fl**, qui défini les champs où la surbrillance sera utilisée, de la manière suivante:
`<str name="hl.fl">texte</str>`

Comme dans le TP, l'analyseur de requêtes edismax est utilisé, et pour le champ **"qf"** c-a-d query fields, nous avons mis:
`<str name="qf">texte5.0</str>`
Afin d'avoir l'affichage que l'on voulait, avec des couleurs et des titres à notre choix, nous avons décidé d'implémenter des templates qui utilisent la librairie **Velocity**. Pour faire cela, nous avons repris les templates disponibles sur le GitLab de monsieur Ruiz-Fabo (`https://git.unistra.fr/ruizfabo/ri_atexte/-/tree/master/templates/velocity_modifie`), en les adaptant à nos besoins.
Nous avons enlevé la facette droite qui était fournie tout en gardant la facette gauche afin de permettre le filtrage selon l'orientation et l'année et pour l'affichage des résultats nous avons précisé que nous garderons uniqument les champs **dateSeance**, **texte**, et **orientation**, afin de ne pas afficher les champs **score**, **_version_** et **id**.

Nous allons maintenant vous montrer nous résultats:
Une fois que Solr est lancé, nous utilisons la commande suivante pour accéder à l'interface de recherche:
`http://localhost:8983/solr/debats/browse`



En faisant une recherche, on voit bien que le terme est surligné, que le nombre de résultats est renvoyé et que les champs revoyés pour chaque instance sont ceux que nous avons précisé:



4 Bibliographie

https://solr.apache.org/guide/7_1/uploading-data-with-index-handlers.html
https://nowontap.wordpress.com/2013/11/25/solr-ingesting-a-csv-file-part-1/
https://git.unistra.fr/ruizfabo/ri_atexte/-/blob/master/tp/tp_ri.md
https://colab.research.google.com/drive/1enqIHVFHv175k8OUsGV1VmB84V9hv1HP?usp=sharing