Institut National des Langues et Civilisations Orientales

Master 2 Ingénierie Multilingue

Cours :

Apprentissage automatique /
Outils de traitement statistique de corpus

dispensés par

Damien Nouvel

Rapport du projet

Catégorisation automatique des questions dans le domaine juridique

---

Auteur : JIANG Chunyang — *N° étudiant :* 21701222

Auteur : XU Yizhou — *N° étudiant :* 21701223

26 avril 2019

Table des matières

Loi de Hofstadter : Il faut toujours plus de temps que prévu, même en tenant compte de la Loi de Hofstadter.

— *GEB*, Douglas Hofstadter

1 Introduction

1.1 Contexte

Ce travail est réalisé dans le cadre des cours de Apprentissage automatique et Outils de traitement statistique de corpus, dispensés par Damien Nouvel, du Master 2 Ingénierie Multilingue à l'INaLCO.

1.2 Objectif

1.2.1 Objectif général

Le projet « Chatlaw » a pour l'objectif de réaliser un chatbot dans le domaine du juridique. Le diagramme [*fig. 1*] donne un aperçu des sous tâches variées effectuées par des groupes différentes. Dans ce travail, nous nous appuyons sur la catégorisation automatique des questions.

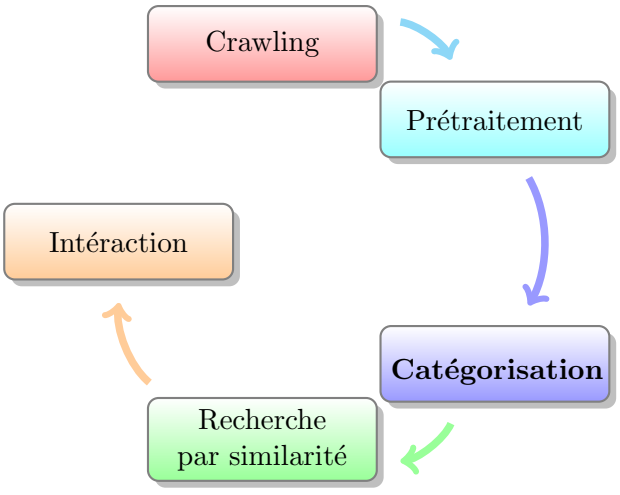

FIGURE 1 – Tâches du projet Chatlaw

1.2.2 Objectif spécifique

Notre tâche consiste à entrainer un modèle qui est capable de catégoriser automatiquement des questions d'un jeu de questions/réponses issues d'un forum juridique,

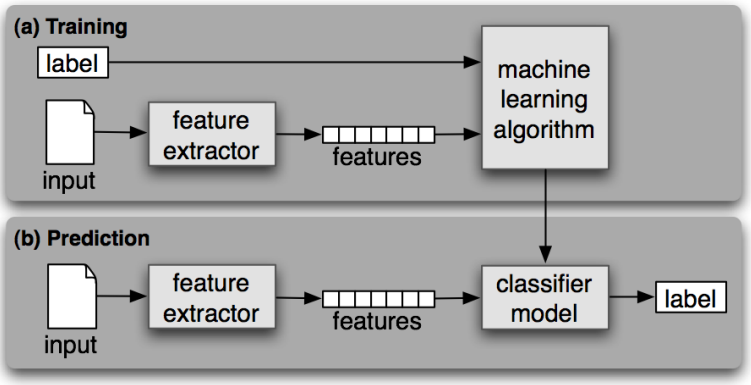

FIGURE 2 – Catégorisation supervisée [1]

ainsi que fournir au groupe suivant un module qui permet de prédire la classe d'une nouvelle question en entrée, comme le montre le schéma [*fig. 2*].

Il est essentiel que le modèle et le module soient robuste, léger et rapide lors de leur déploiement sur le serveur.

1.3 Plan

Dans ce travail, nous commencerons par une analyse exploratoire sur le jeu de données afin de reconnaître les caractéristiques pertinentes. Nous essayerons de déterminer ensuite avec scikit-learn [2] le classifieur le plus performant selon nos objectifs. Nous proposerons dans un dernier temps une évaluation du modèle choisi, suivie d'une discussion des pistes d'amélioration.

2 Jeu de données

2.1 Corpus sélectionné

Le corpus soumis à notre travail est procuré par le groupe qui s'occupe du *Crawling*. Il est composé de questions/réponses provenant d'un forum juridique français. Tous les textes bruts sont regroupés dans un fichier *xml* avec des balises précisant l'id, le titre, l'URL, ainsi que la classe et la sous-classe. Ce corpus sert à entrainer et tester le modèle de prédiction avec de différents algorithmes d'apprentissage automatique.

1. source : http://www.nltk.org/
2. release 0.20.2, cf https://scikit-learn.org/stable/index.html

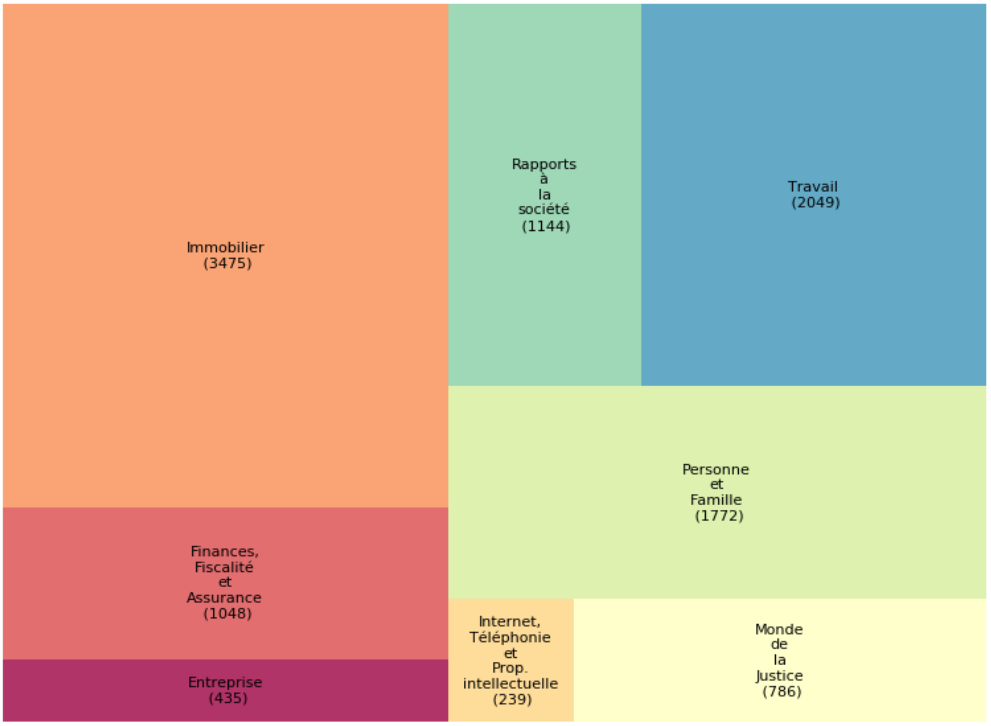

FIGURE 3 – Distribution [3] par classe

2.2 Classes (*Labels*) sélectionnées

Les échantillons du corpus sélectionné représentent 8 classes et 22 sous-classes préalablement attribuées par le forum. La carte [*fig. 3*] montre la distribution de textes (paires de questions/réponses) par classe. Le travail est donc une tâche de catégorisation de multi-classes dans un contexte de déséquilibre de données. Nous utiliserons ces 8 classes comme *labels*[listing 1].

```
DOC_CLASS = {
    'immobilier' : 'imm',
    'travail' : 'trv',
    'personne et famille' : 'per',
    'finances, fiscalité et assurance' : 'fin',
    'rapports à la société' : 'soc',
    'monde de la justice' : 'jus',
    'entreprise' : 'ent',
    'internet, téléphonie et prop. intellectuelle' : 'int'
}
```

Code Listing 1 – *Mapping* des classes

2.3 Répartition de corpus

Une fois les textes dupliqués et/ou vides supprimés, le corpus est répartis aléatoirement en deux parties [table 1] pour l'entrainement et le test.

	Corpus train	**Corpus test**	**Total**
nombre	9848	2460	12308
pourcentage	80%	20%	100%

TABLE 1 – Répartition du corpus

3 Sélection de caractéristiques (*features*)

3.1 Type de caractéristiques

Avant que nous puissions appliquer des techniques d'apprentissage automatique, il faut construire une représentation numérique des textes pour que la machine peut comprendre [2]. Le modèle de sac de mots, entre autres modèles de représentation des textes, est souvent utilisé quand il s'agit de classification thématique ou domaniale [1].

Il faut admettre que la sélection des caractéristiques est essentiellement un processus itératif qui va souvent de pair avec la catégorisation. En l'occurrence, nous essayons sac de X : token/lemma[4], lemma+pos, ngram de lemmas(1-3) avec des algorithmes de catégorisation dans la Section 4.

3.2 Représentation numérique

Une fois le type de caractéristiques déterminé, il convient de transformer les textes par un certain codage. Cela peut être binaire(présence/absence), fréquence, tf-idf, chi-2, information mutuelle, etc. Nous optons pour le tf-idf.

3.3 Filtrage et nettoyage

Puisque les données sont issues de forum de discussion, il existe souvent des mots non standards, mal-orthographiés et/ou vides, qui influent sur la performance de catégorisation. Nous tentons de réduire ces bruits à l'aide de *sklearn.feature extraction.text.TfidfVectorizer* :

3. sur l'ensemble du jeu de données.
4. La lemmatisation est effectué avec treetaggerwrapper (un Python *wrapper* du TreeTagger) , qui est réalisé par le groupe qui s'occupe le Prétraitement, et est réfactorisé pour mieux intégrer ce travail.

(i) mots vides et mots fréquents(*corpus-specific stop words*)
   * ignorer les termes qui apparaissent dans plus de 70% des documents
   * `max_df=0.7`

(ii) mots rares (*cut-off*)
   * ignorer les termes qui apparaissent dans moins de 5 documents
   * `min_df=5`

(iii) accents
   * supprimer les accents sur les caractères qui ont un mappage ASCII
   * `strip_accents='ascii'`

4 Entrainement

Dans cette Section, nous présenterons les algorithmes pour l'entrainement, leur paramétrage à l'aide de *Grid Search*, ainsi que l'évaluation des modèles entrainés sur le jeu de test.

4.1 Algorithmes de catégorisation de textes

Certains des algorithmes d'apprentissage automatique les plus populaires pour la création de modèles de classification de texte incluent la famille des algorithmes naïfs bayes, les machines à vecteurs de support (SVM), la régression logistique, l'arbre de décision et ses extensions(*bagging*, forêts aléatoires), ainsi que l'apprentissage en profondeur. Dans ce travail, nous en essayons certains à l'aide des implémentations correspondantes dans la bibliothèque scikit-learn.

4.1.1 *Baseline*

Classifier *sklearn.dummy.DummyClassifier*
DummyClassifier est un classifieur qui fait des prédictions en utilisant des règles simples. Il est utile comme base simple pour comparer avec d'autres classifieurs.

Stratégie *stratify*
Stratifié permet de générer des prédictions aléatoires en respectant la répartition des classes de corpus d'entraînement.

4.1.2 Naïf Bayes

La classification naïve bayésienne est basée sur le théorème de Bayes. Il existe plusieurs classifeurs dans cette famille, tels que *GaussianNB*, *MultinomialNB*, *Ber-*

nouilliNB et *ComplementNB*[5] [3]. Ils se diffèrent principalement par la distribution de vraisemblance. Dans ce travail, nous optons pour *ComplementNB*.

Classifier *sklearn.naive_bayes.ComplementNB*

ComplementNB est une adaptation de multinomial naïf de Bayes(MNB), particulièrement adapté aux ensembles de données non équilibrés. Plus précisément, CNB utilise les statistiques du complément de chaque classe pour calculer les pondérations du modèle.

4.1.3 Régression logistique

Classifier *sklearn.linear_model.LogisticRegression*

La régression logistique est un modèle linéaire de classification, également connu dans la littérature sous le nom de régression *logit*, classification d'entropie maximale (*MaxEnt*) ou classifieur log-linéaire.

Stratégie Optimisation et régularisation

Pour les problèmes multi-classes, seuls *newton-cg*, *sag*, *saga* et *lbfgs* gèrent les pertes multinomiales ; *liblinear* se limite à des schémas *one-versus-rest*.

newton-cg, *lbfgs* et *sag* ne gèrent que la pénalité de L2, tandis que *liblinear* et *saga* gèrent la pénalité de L1. Pour notre corpus d'entrainement, nous avons deux combinaisons possibles :

* newton-cg + l2 + multinomial
* liblinear + l1 + ovr

4.1.4 SVM

Les machines à vecteurs de support (SVM) sont une classe de méthodes d'apprentissage statistique basées sur le principe de la maximisation de la marge (séparation des classes). Il existe plusieurs formulations (linéaires, versions à noyaux) qui peuvent s'appliquer sur des données séparables (linéairement) mais aussi sur des données non séparables[6].

Dans Scikit-learn, les SVM sont implémentés dans le module sklearn.svm. Dans cette partie nous allons nous concentrer sur la version linéaire.

Classifier *sklearn.svm.LinearSVC*

Les modèles linéaires *LinearSVC()* et *SVC(kernel='linear')* produisent des résultats légèrement différents à cause du fait qu'ils optimisent des fonctions de coût différentes mais aussi à cause du fait qu'ils gèrent les problèmes multi-classe de manière différente (linearSVC utilise *One-vs-All* et SVC utilise *One-vs-One*).

5. *ComplementNB* est implémenté dans scikit-learn 0.20.
6. cf http://cedric.cnam.fr/vertigo/Cours/ml2/tpSVMLineaires.html

4.1.5 Forêts aléatoires

Classifier *sklearn.ensemble.RandomForestClassifier*

Une forêt aléatoire est un méta-estimateur qui crée un ensemble d'arbres de décision à partir d'un sous-ensemble sélectionné d'un corpus d'apprentissage. Il regroupe ensuite les votes de différents arbres de décision pour décider de la classe(*label*) finale.

Cela permet de régler plusieurs problèmes inhérents aux arbres de décision uniques comme l'altération du résultat selon l'ordre des paramètres prédicteurs dans les noeuds, ou encore de réduire leur compléxité.

Stratégie Contrôle du nombre et de la taille des arbres

* *n_estimators*
* *max_depth*
* *min_samples_leaf*
* etc

4.2 Paramétrage

Nous avons déjà rencontré certains paramètres tels que *max_df* dans *TfidfTransformer*. Les classifieurs ont également tendance à avoir de nombreux paramètres.

Au lieu de peaufiner les paramètres des différents composants de la chaîne de manière « manuelle » , il est possible de lancer une recherche exhaustive des meilleurs paramètres sur une grille de valeurs possibles à l'aide de *GridSearchCV*.

Outil *sklearn.model_selection.GridSearchCV*

Nous essayons tous les classifieurs sur des lemmes et des ngrams(1,3) de lemmes, et avec recherche exhaustive sur les valeurs de paramètre spécifiées pour un estimateur. Cette procédure est couteuse en terme de temps.

A titre d'exemple, pour *RandomForestClassifier*

* *n_estimators* : 20,30,40,...,200 ⇒ *130*
* *max_depth* : 10,20,30,...,100 ⇒ *60*
* *min_samples_split* : 2, 5, 10 ⇒ *10*
* *min_samples_leaf* : 1, 2, 4 ⇒ *2*

4.3 Évaluation

Mesure d'évaluation micro F1-mesure

S'agissant d'une tâche de catégorisation multi-classe dans un contexte de déséquilibre de données, il convient d'utiliser la micro F1-mesure pour évaluer la performance des modèles.

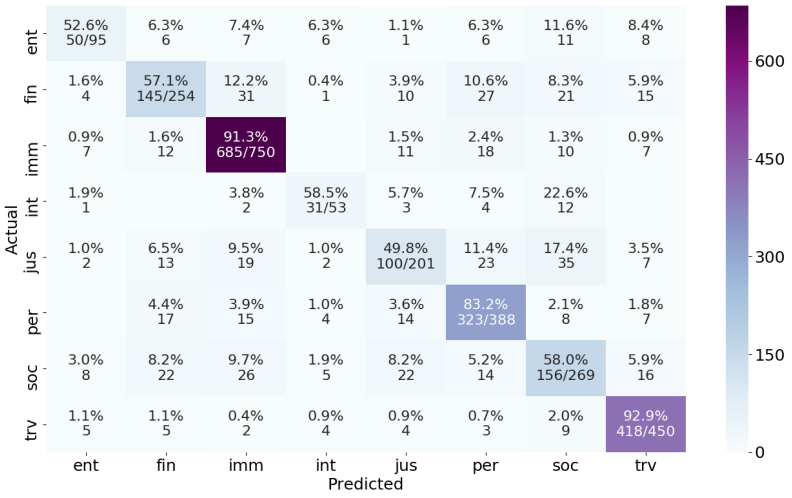

FIGURE 4 – SVM - matrice de confusion sur le jeu de test

Le tableau [*table 2*] montre la micro moyenne de F1-mesure pour chaque classifieur bien paramétré sur le jeu de test.

	BL	**RF**	**LR**	**NB**	**SVM**
Token	0.18	0.72	0.77	0.76	**0.78**
Lemma+POS	0.18	0.72	0.78	0.76	**0.78**
N-Gram	0.19	0.72	0.76	0.69	**0.79**

TABLE 2 – Micro F1-mesure sur le jeu de test

Qu'est-ce un bon modèle (dans le cadre du projet) ?

— **Critères**

* Léger & rapide (à la fois pour charger et *fitting*)
  ⇒ économie de mémoire RAM
  ⇒ prévention de surcharge serveur
* Bonne micro F1-mesure

— **Modèle choisi**

* **Caractéristiques** ⇒ lemma + POS
* **Classifieur** ⇒ SVM
* **Taille** ⇒ 1,8 Mo
* **F1-mesure (micro)** ⇒ 78%

Matrice de confusion pour le modèle choisi La matrice de confusion [*fig. 4*] montre que le modèle fonctionne mieux pour les classes qui ont plus de échantillons (telles que *immobilier*, *travail*, *personne et famille*) ; quant aux classes « petites », (*internet, téléphonie et prop. intellectuelle* et *monde de la justice*) sont souvent classées comme *rapport à la société*.

5 Prédiction

Une fois le modèle créé, nous pouvons prédire la classe d'une question en entrée. Dans ce travail, nous proposons un module qui permet au groupe suivant de faire la prédiction tout simplement en deux étapes.

— **Scripts**
  * make_prediction.py
  * test_make_prediction.py

— **Fonctionnement**
  * load_model
    - Chargement d'un modèle en mémoire
      ⇒ « une fois pour toutes »
  * make_prediction
    - Prétraitement de phrase en entrée
      ⇒ identique au prétraitement sur le modèle entrainé
    - *Fit* et prédiction

6 Conclusion

Ce rapport compare plusieurs algorithmes de catégorisation de texte, propose un modèle et un module pour prédire la classe d'une question en entrée. Toutefois, les résultats peuvent être améliorés.

Améliorations envisagées : 3 pistes envisageables

(i) *Jeu de données* : Puisque la classe d'une question est attribuée par un utilisateur de forum, il est très probable qu'elle soit mal-étiqueté. Il serait intéressant de réexaminer et réétiqueter les classes.

(ii) *Features* : Il convient de contrôler la qualité des *features*
bon nettoyage, réduction de dimension, ajout de features, usage d'autres types de représentation (plongement par exemple)

(iii) Autres approches : il serait intéressant aussi d'essayer l'apprentissage en profondeur.

Références

[1] CLEUZIOU, G., AND POUDAT, C. Classification de textes en domaines et en genres en combinant morphosyntaxe et lexique. *Actes du quatrième DÉfi Fouille de Textes* (2008), 57.

[2] IBEKWE-SANJUAN, F. *Fouille de texte.* Hermès-Lavoisier, 2007.

[3] RENNIE, J. D., SHIH, L., TEEVAN, J., AND KARGER, D. R. Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of the 20th international conference on machine learning (ICML-03)* (2003), pp. 616–623.