Rapport du projet

Extraction et programmation statistique de l'information

M2 ISD
Groupe1

1. Introduction

Nous faisons de la Classification de texte multi-classe sur un jeu de données en anglais (contenu), écrit par des personnes de nationalités différentes (tag). Chaque échantillon de texte contient le tag au début de cet échantillon. Bien entendu, certains ont l'anglais comme de la langue maternelle, et d'autres non.

Concrètement, on veut prédire de quelle nationalité s'agit un texte nouvellement écrit par une personne de nationalité inconnue.

Pour ce dernier, nous avons réalisé un pré-processing des données, puis du feature engineering. Et par la suite, nous avons construit de nombreux modèles et les entraîner avec nos jeux de données d'entraînement, puis les évaluer avec le score de précision afin d'évaluer leur performance respectives.

Après la soutenance, nous avons prise en compte certaines suggestions d'autres groupes. Nous avons ajouté de nouvelles caractéristiques (l'explication en bleue ci-dessous), puis essayé différentes façons de normalisation et enfin refait la partie d'évaluation des features.

2. Feature Engineering

2.1 Extraction des informations

Nous avons fait 4 différents types de représentations des mots qui nous a servi de caractéristiques par la suite.

1. **One-hot** : Un **encodage one-hot** ou **encodage 1 parmi n** consiste à représenter des états d'existence en utilisant pour chacun une valeur dont la représentation est binaire de 0 ou 1 ;
2. **TF (de l'anglais term frequency)**: le nombre d'occurrences d'un terme dans l'échantillon actuelle ;
3. **Word embedding** : Glove qui est un modèle pré-entraîné ; word2vec, un modèle entrainant avec le jeu de données du projet ;
4. Environ 400 caractéristiques **statistiques** extraites manuellements ;

Plus spécifiquement, ces caractéristiques extraites manuellements avec les méthodes statistiques sont:

- **Au niveau de Text**
  - ➔ **Statistique générale**

1) char_count : Nombre de caractères par texte
2) word_count : Nombre de mots par texte
3) word_density : Longueur moyen de mots par texte (*word_density = char_count / word_count*)
4) punctuation_count : nombre de ponctuations par texte
5) punctuation_count_apostrophe : nombre d'apostrophe par texte
6) punctuation_count_bracket_right : nombre de parenthèse (droite) par texte
7) punctuation_count_bracket_left : nombre de parenthèse (gauche) par texte
8) punctuation_count_exclamation : nombre de point d'exclamation par texte
9) punctuation_count_question: nombre de point d'interrogation par texte
10) punctuation_count_dash : nombre de tiret par texte

➔ **Analyse de sentiment**

L'analyse des sentiments a fait par la package TextBlob qui est une bibliothèque open source pour le traitement des données textuelles.
On peut connaître deux scores sur le sentiment : Polarité et Subjectivité. Polarité fait référence au niveau de positivité, entre -1 à + 1, avec -1 la plus négative et + 1 la plus positive. Subjectivité fait référence aux subjective comment les taux d'instructions de 0 à 1, 1 étant très subjective.

11) polarity_negative : 1 si le score de prolarity entre -1.0 et -0.3, sinon 0
12) Polarity_neutre : 1 si le score de prolarity entre -0.3 et 0.3, sinon 0
13) Polarity_positive : 1 si le score de prolarity entre 0.3 et 1.0, sinon 0
14) Subjectivity_objective : 1 si le score de subjectivity entre 0.0 et 0.5, sinon 0
15) Subjectivity_subjective : 1 si le score de subjectivity entre 0.0 et 0.5, sinon 0

- **Au niveau de Phrase**

16) sentence_count : Nombre de phrase par texte
17) sentence_density_char : Longueur moyenne de phase (en nombre de caractère) par texte
18) sentence_density_word : Longueur moyenne de phrase (en nombre de mot) par texte

- **Au niveau de Mot**

➔ **Majuscule**

19) upper_case_word_count : Nombre de mot commencé par majuscule par texte

➔ **Article**

20) quantity_a : Nombre d'occurrence du mot « a » par texte
21) quantity_an : Nombre d'occurrence du mot « an » par texte
22) quantity_the : Nombre d'occurrence du mot « the » par texte

➔ **Pos-tagger :** réalisé par nltk.pos_tag(x)

23) noun_count : Nombre de noms par texte
24) verb_count : Nombre de verbes par texte
25) adj_count : Nombre de adjectives par texte
26) adv_count : Nombre d'adverbes par texte
27) pron_count : Nombre de pronoms par texte

➔ **Stop Word**

28) stopwords_count : Nombre de stopwords par texte
29) stopwords_frequency : Fréquence des stopwords (*stopwords_frequency = stopwords_count/word_count*)

- **Nombre de mot concernant une langue maternelle (nom de pays/ville)**

On a 11 langues maternelles: German,Turkish, Chinese, Telugu, Arabic, Hindi, Spanish, Japanese, Korean, French, Italian.

Pour nous, il peut être intéressant de chercher des localisations qui sont culturellement ou géographiquement en rapport avec les langues maternelles, par exemple les pays, et les villes et régions qui parlent une certaine langue. Ces villes ou régions peuvent être inclus ou pas dans les pays.

Langue	Pays	ville/région
German	Germany,Austria,Liechtenstein,Switzerland...	Berlin, Hamburg, Munich, Cologne, …
Chinese	China,Singapore	Hongkong, Tibet, Taiwan, Beijing, Shanghai….
Telugu	India	…..
…	...	...

Nous avons donc construit un dictionnaire avec ces mots concernant les langues. Puis, nous avons calculé l'occurrence de chacun des mots dans chaque corpus de texte. De ce fait, nous avons trouvé 2 features de plus :

30) word_country_about_GER : l'occurrence de pays qui parlent l'allemand dans chaque texte
31) word_region_about_GER : l'occurrence de villes et régions célèbres qui parlent l'allemand dans chaque texte

Nous avons appliqué les même traitements pour les autres langues et nous avons finalement obtenu 22 features en rapport avec les pays, villes et régions.

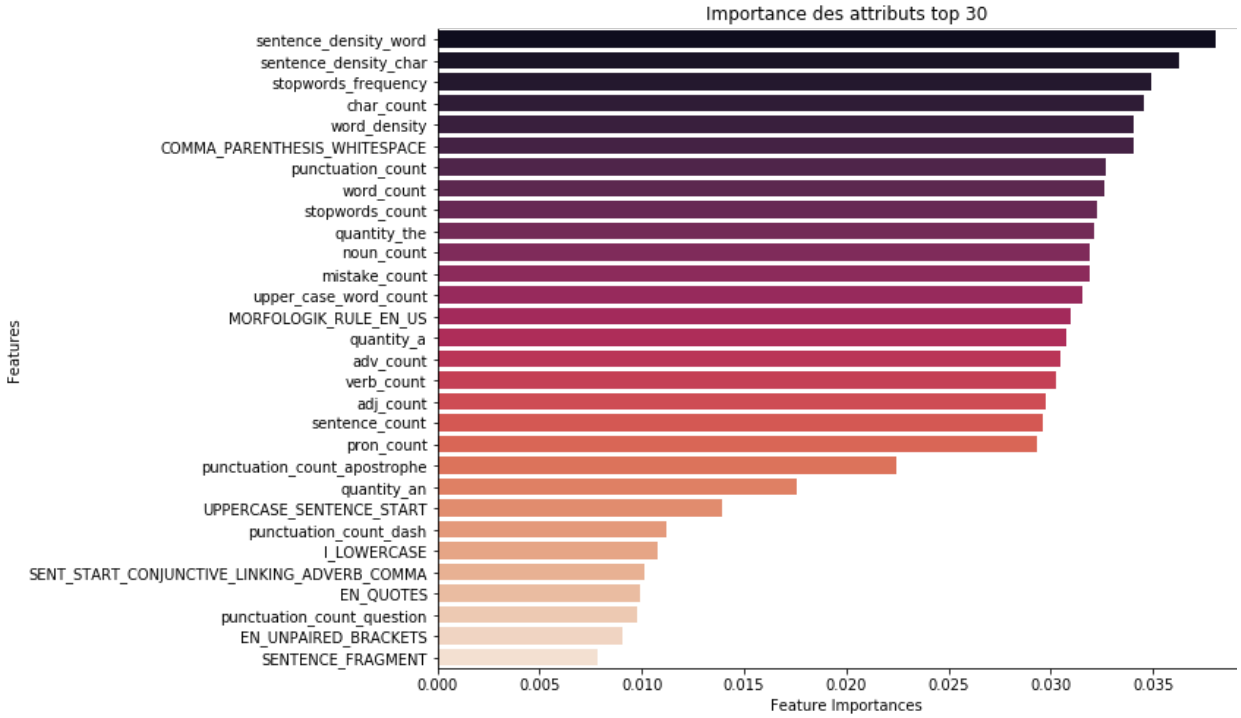
- **Faute d'orthographes et de grammaires**

Les fautes d'orthographes sont repérées par le package *language-check,* une bibliothèque open source développé par la société LanguageTool. Cet outil nous aide à retrouver les fautes d'orthographes et les fautes de grammaires d'un texte, et nous donne les catégories de ces fautes.

32) mistake_count : Nombre d'orthographes et de grammaires par texte
33) WHITESPACE_RULE : Le nombre de faute "Whitespace repetition (bad formatting)"
34) ENGLISH_WORD_REPEAT_RULE : Le nombre de faute "Word repetition (e.g. 'will will')"
35) ……..on a environ 350 types de la faute d'orthographes

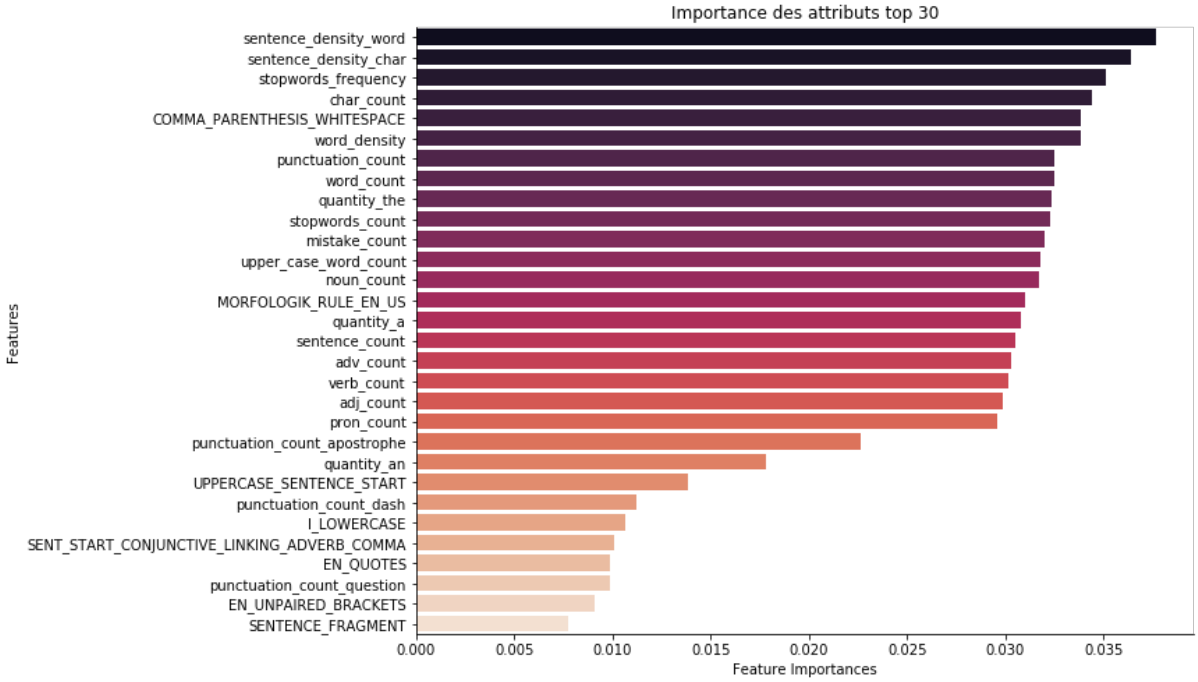
2.2 Évaluation des caractéristiques

En utilisant le modèle Random Forest, nous avons trié les caractéristiques par l'importance.



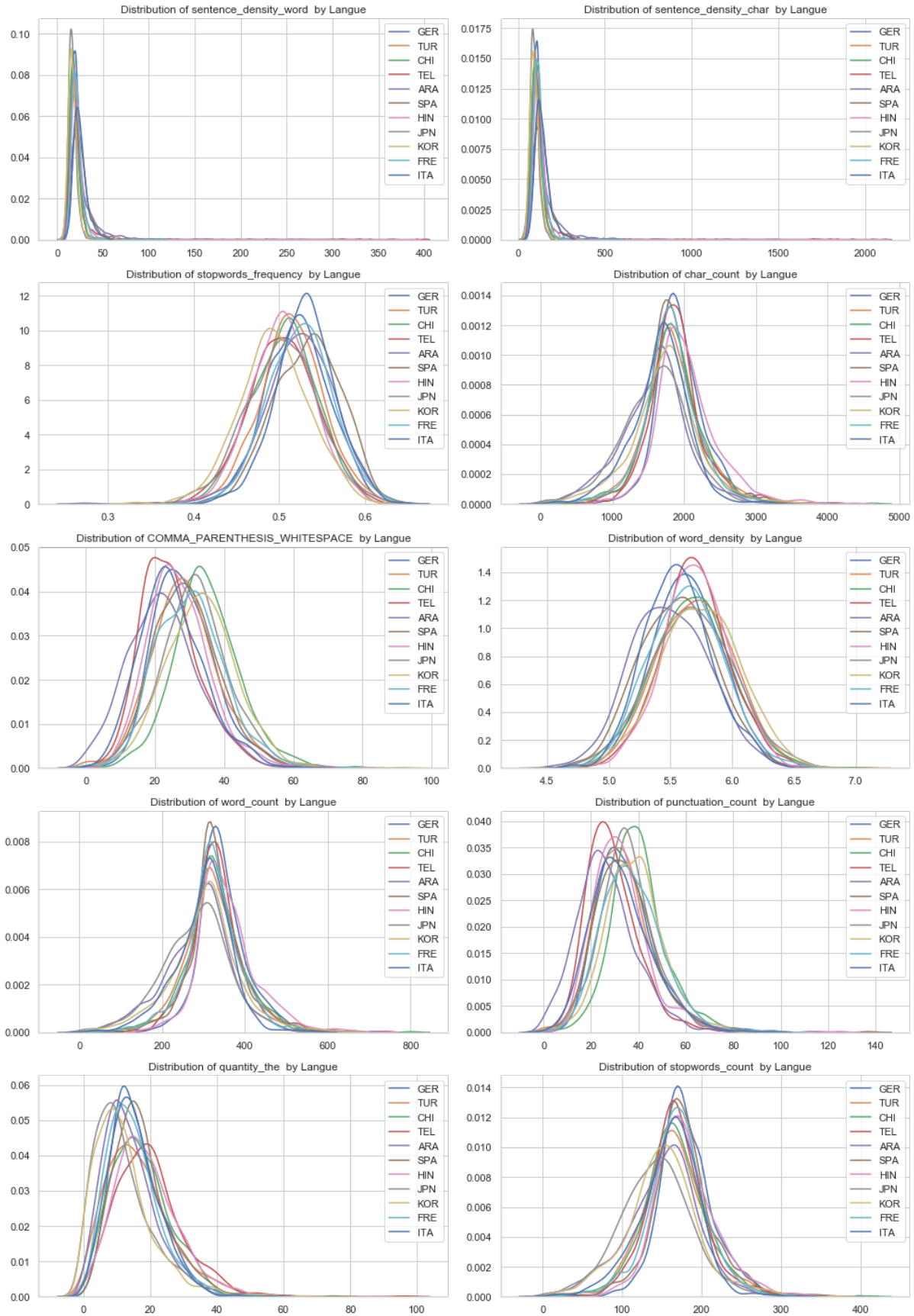
Le graphe ci-dessus nous permet de conclure que les caractéristiques sur le longueur moyenne de phase a une grande influence sur les résultats de prédiction. Les informations sur les stopwords, le nombre de texte et les pos-tagger sont aussi des caractéristique très importantes. Concernant les fautes de grammaire, COMMA_PARENTHESIS_WHITESPACE (“Use of whitespace before comma and before/after parentheses”) et MORFOLOGIK_RULE_EN_US (“Possible spelling mistake") sont relativement utiles.

Grâce à la remarque de notre camarade pendant la soutenance, nous avons normalisé les données et remettre en ordre les attributs.



Après normalisation, nous pouvons voir sur le graphe ci-dessus, que l'ordre des attributs n'applique pas de changement significatif.

Puis nous avons réalisé les histogrammes des 10 caractéristiques les plus importantes afin de voir si ces derniers nous permettent de distinguer les différentes langues.



Nous avons vu effectivement que ces caractéristiques ont des différences selon leurs langues correspondants, ce qui pourrait nous permet de les distinguer.

3. Construction, entraînement et évaluation des modèles

caractéristique	Modèle	Précision du jeu de test	Dataset shape	commentaire
Les mots (one hot)	BernoulliNB	0.637	(9900, 55240)	
Les mots (one hot) + 10 caractéristiques extraites manuellement	BernoulliNB	0.668	(9900, 55250)	Nombre d'apprentissage ne change pas la précision
10 caractéristiques extraites manuellement	BernoulliNB	0.1385	(9900, 10)	
391 caractéristiques extraites manuellement	BernoulliNB	0.266	(9900, 391)	
	MultinomialNB	0.297		
	DecisionTreeClassifier	0.167		
100 caractéristiques les plus importantes trouvés par Random Forest	BernoulliNB	0.239	(9900,100)	
Remplacer les mots one-hot par TF				
TF(term frequency)	GaussianNB	0.3885	(9900, 51827)	Pour le "term frequency", si on enlève les stop_words ou bien on ignore les mots qui apparaissent dans plus de 85% des documents ou bien on ne prend que les 10 000 mots les plus fréquents, le score devient
	BernoulliNB	0.6395	(9900, 51827)	
	MultinomialNB	0.647	(9900, 51827)	

				pire.
	DecisionTreeClassifier	0.253	(9900, 51827)	
TF30000 + features391	MultinomialNB	0.6215	(9900, 30391)	
TF50000 + features391	BernoulliNB	0.6415	(9900, 52218)	
	MultinomialNB	0.6135	(9900, 52218)	
	DecisionTreeClassifier	0.267	(9900, 52218)	
Word-embedding				
glove-wiki-gigaword-100	Keras réseau de neurones	0.3075000047683716	(9900, 100)	
Gensim.word2vec	Keras réseau de neurones	0.3015	(9900, 150)	
TF50000+word2vec150+glove100	BernoulliNB	0.657	(9900, 52077)	
	MultinomialNB	0.633	(9900, 52077)	
	DecisionTreeClassifier	0.261	(9900, 52077)	
TF1000 + features391 + word2vec150 + glove100	BernoulliNB	0.561	(9900, 1641)	La raison pour laquelle on fait la réduction de dimension de 51827 au 1000 pour TF est d'augmenter la proportion des features manuellement extraites, ainsi augmenter son importance.
	MultinomialNB	0.345	(9900, 1641)	
	DecisionTreeClassifier	0.208	(9900, 1641)	

TF30000 + features391 + word2vec150 + glove100	Keras réseau de neurones	0.533	(9900, 30641)	4 heures d'exécution avec un réseau de neurone basique.
TF51827 + features391 + word2vec150 + glove100	BernoulliNB	0.685	(9900, 52468)	
	MultinomialNB	0.552	(9900, 52468)	
	DecisionTreeClassifier	0.216	(9900, 52468)	
	SVM			
	Keras réseau de neurones			

Nous avons aussi testé les différentes méthodes de normalisation sur les modèles bernoulliNB, MultinomialNB et DecisionTreeClassifier, que vous trouverez les codes dans le ficher “tous les models.ipynb”, et nous avons constaté des phénomènes étranges.

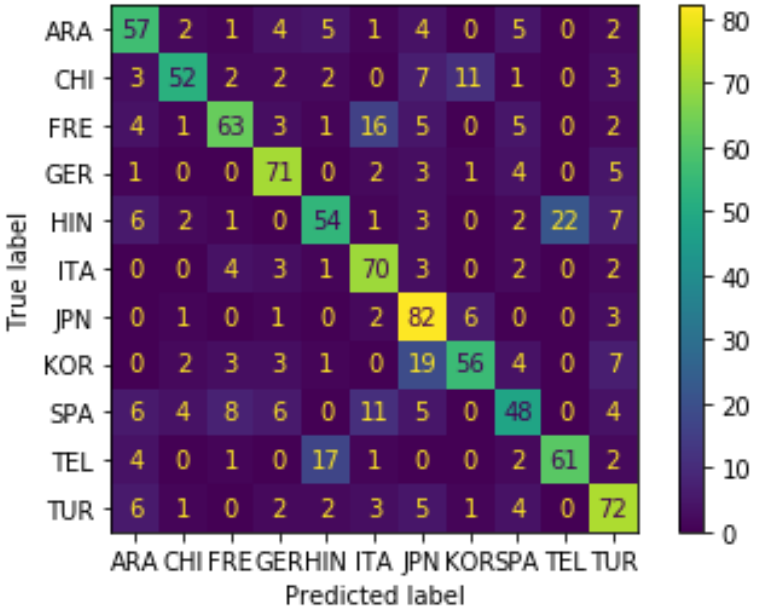
Nous avons d’abord appliqué la fonction “sklearn.preprocessing.StandardScaler” sur les 3 groupes de caractéristiques (TF+glove+word2vec).

Si on normalise après avoir concaténé toutes ces caractéristiques, les précisions, initialement 0.6, sont devenu mauvaises : 0.46 pour bernoulliNB.

Si on ne normalise que les word-embedding(glove+word2vec) mais pas TF, les précisions reste moyennement appréciables, à peu près 0.6 pour bernoulliNB et MultinomialNB.

Après la normalisation, certaines valeurs de caractéristiques deviennent négatives, l’intervalle est à peu près [-3, 100]. Comme le modèle MultinomialNB ne prend pas de valeur négative, nous avons fait donc +3 pour toutes les caractéristiques, l’intervalle de valeurs est donc devenu [0,103]. Avec cette action +3, la précision de MultinomialNB ne change pas beaucoup(0.59), mais la précision de bernoulliNB devient hyper mauvaise qui est 0.07. Sans l’action +3, la précision de bernoulliNB est pas mal qui est 0.67. Nous pensons que ceci est extrêmement bizarre, nous n’avons toujours pas compris pourquoi l’action +3 a un impact si énorme sur le modèle bernoulliNB.

Ci-dessous la matrice de confusion.



Les éléments diagonaux représentent le nombre de points pour lesquels l'étiquette prédite est égale à la véritable étiquette, tandis que les éléments hors diagonale sont ceux qui sont mal étiquetés par le classificateur. Plus les valeurs diagonales de la matrice de confusion sont grandes, plus les prédictions sont correctes.

Dans notre cas, le japonais est le mieux prédit, l'espagnol est le moins bien prédit.

4. Extra

1) **D'autres hyperparamètres pour nos modèles de deep-learning ?**

   Limité par la puissance de nos PC personnels, on n'a pas pu tester tous les hyperparamètres que nous avons voulu tester. De ce fait, nous avons envie d'améliorer la performance de notre modèle de deep learning, keras réseaux de neurones, keras.sequentiel, en faisant des tests avec des tunings différents des hyperparamètres. Pour l'instant nous avons dû limiter le nombre de **dimension (nodes)** en 30614 au lieu de 52218, le nombre d'**époque** à 3 au lieu de 30 pour que le résultat puisse être généré dans un temps raisonnable (à peu près 4 heures). Il est également possible de tester la performance des autres **fonctions d'activations**, ainsi que d'autres nombres de **nodes** et de **couches**. Mais si la puissance est suffisante, nous aimerions les tester.

2) **D'autres modèles neuronaux ?**

   Il est possible d'utiliser d'autres modèles neuronaux comme le MLP(multi-layer perceptron), ou les CNN.

3) **Ajouter K-means cluster.**

   Nous permet de regrouper les textes ayant les caracteristiques similaires en plusieurs clusters, pour savoir quelles langues ont des ressemblances.

4) **Améliorer notre correction d'orthographe.**

   TF -

   Après avoir observé les données, nous avons remarqué que les fautes d'orthographe s'agit souvent d'une erreur d'épellation, par exemple on écrit "Helo",

au lieu de “Hello”. Afin d’améliorer la précision de TF (term frequency), on pourrait d’abord identifier les mots mal écrits, et compter l'occurrence d’un mot mal écrit sous le nom du bon mot correspondant.

Nombre d’écriture d’un même mot -

On pourrait également compter le nombre d’erreur s’agissant d’un même terme d’une nationalité spécifique. Par exemple, les français ont tendance à écrire le mot “gouvernement” au lieu de “government”.

5) **SVM. temps d’execution trop long.**

5. Conclusion

Ce projet a été très enrichissant et formateur pour nous. Grâce à ce projet, nous avons eu l'occasion de pratiquer les différentes techniques d’extraction et programmation d’information tel que des méthodes de feature engineering, word-embedding, construction, entraînement et évaluation des modèles…

Cependant, il y a eu également de nombreuses découvertes de surprises et de points d’améliorations au fur et à mesure de l'accomplissement du projet. Nous avons été un peu déçue par nos résultats obtenus car nous avons eu beaucoup d’espoirs après avoir consacré de nombreux efforts dans la recherche de différents features. Surtout nous avons été contraints par la recherche de bons paramètres d'entraînements qui peuvent à la fois être convenable à la puissance de nos PC personnels, sans trop compromettre les résultats obtenus. Mais malgrés notre énergie, certains résultats restent décevant à cause de ses contraintes.

Malgrés ces déceptions, le fait de pouvoir réaliser un processus end-to-end depuis les données brutes jusqu’à l’évaluation des modèles reste très précieux car cela nous permet de voir la réalité d’un projet. L’avoir réalisé dans un cadre académique nous permet de mieux statuer les problèmes en amont dans un cadre industriel.

Bibliographie

https://www.nationsonline.org/oneworld/countries_by_languages.htm
https://pypi.org/project/language-check/#description
https://languagetool.org/dev/