Projet d'introduction au machine learning
Out-Of-Domain PoS Tagging
Rapport Final

Le Montagner Roman
Turbiau Guilhem

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Le Part of speech tagging (PoS ou encore étiquetage morpho-syntaxique en français) consiste à construire un algorithme capable d'identifier et d'extraire les informations syntaxiques des mots contenus dans une phrases. Ainsi, notre algorithme doit être capable d'associer une étiquette à chaque mot d'une phrase, cette étiquette permettant de connaître la classe syntaxique du mot en question.
Le PoS est un problème bien connue en linguistique. Il s'agit de la façon dont, nous autres humains, comprenons les mots et leurs fonctions dans une phrases. Il s'agit d'un processus d'apprentissage que nous faisons en étant enfant lorsque nous lisont ou écoutons pour la première fois. Lorsque nous arrivons à identifier les structures syntaxiques d'une phrase, cela signifie que nous avons compris le contexte qui entoure la phrases, sa significations et donc de mieux la comprendre. Ainsi, avec le developpement des techniques algorithmiques moderne et l'augmentation des vitesses de traitements des ordinateurs, Nous voulons donner à nos ordinateurs la capacité de pouvoir comprendre le langage humain que ce soit écrit ou oral. Cela permettrait d'améliorer les algorithmes de detection des erreurs, les assistants personnels tel Siri ou Cortana ou encore le domaine de la robotique avec par exemple Nao.

1.2 Approche naive et identification des problèmes

Nous pourrions nous arrêter la et construire un algorithme général qui, étend donné tout les mots possible du dictionnaire d'un langage, renvoie la bonne classe syntaxique du mot, une sorte de grandes tables de hashage qui associe chaque mot du dictionnaire avec sa classe syntaxique. Nous pouvons tout de suite identifier une limite à cette algorithme.

Prenons l'exemple de deux phrases simple :

— Aujourd'hui, Il a noté beaucoup de rendez-vous dans son carnet.

— Le son d'un avion au décollage est très très fort.

Nous pouvons voir que le mot "son" est utilisé dans les deux phrases or leurs classes syntaxique à changé entre les deux. Dans la première phrase, "son" est utilisé comme étant un adjectifs possessif alors que dans la seconde, il s'agit du nom commun "son" qui décrit le phénomène physique. Nous avons identifier un premier problème à notre approche naive. Un même mot peut avoir plusieurs classes syntaxique qui dépend du contexte de la phrase. Il ne suffit plus simplement de se focaliser sur le mot seul mais sur toute la phrase en général.

Un autre problème est celui du use-mention distinction. Ce problème peut être illustré de la façon suivante : Le mot "voiture" est composé de trois syllabes. . Dans cette exemple, le mot "voiture" peut être remplacé par n'importe quel mot contenant également trois syllabes. Ainsi, la différences entre "utilisé" un mot et y faire "référence" est dénotée dans cette exemple (le "Brown Corpus tag set" ajoute le tag -NC dans ces cas). Le problème est qu'on peut remplacé le mot par n'importe quelle autre, il s'agit d'une référence et doit donc avoir un tag syntaxique différent du reste des mots de la phrase.

De plus, si nous utilisons l'approche naive de la table de hash. Il se pose la question du nombre de mot d'une langue mais également de la définition d'un mot. Il convient également d'identifier de quelles langues ont parle. L'humanité,

au cours de son histoire, s'est scindée en de multiple groupes sur toute la planète et chacun à évolué avec sa propre histoire, culture et langue. Ainsi, notre algorithme doit-il se focaliser sur une seul langue ou essayer de toute les englobers ? Sur la question de la nature des mots d'une langue, il y a le cas des langues qui choisissent de composer des mots par associations comme l'allemand ou encore les langues agglutinantes comme le turc. Le français n'est pas en reste non plus.

Cette question sur la nature des mots est intrinsèquement liée avec celle du nombre de mot d'une langue. Par exemple, *le Larousse* rescense plus de 59 000 mots pour le français alors que *Le Littré* en compte pas moins de 132 000. Le nombre de mot d'une langue change en fonction de la définition que l'on donne à un mot.

Il est donc essentielle de trouver une autre approche à ce problème.

1.3 Historique du Part Of Speech Tagging

Le tout premier corpus de langue a été construit dans les années 60 à l'université Brown à Providence dans le Rhode Island au Etats-Unis. Il était initialement composé uniquement de mot avec un total de 500 données de texte en langue anglaise pour plus de 1 000 000 de mots.Chacune des données ayant au moins 2000 mots. Ce corpus avait pour but de fournir des statistiques sur la langue anglaise comme la fréquence des mots dans la langue. Très vite, les scientifiques et linguistique ajoute l'étiquetage morpho-syntaxique au brown corpus. Pour cela, un algorithme de PoS tagging est developpé par B.B. Green et G.M. Rubin. L'algorithme fonctionnait selon les possibilités de co-occurence des mots. PAr exemple, un determinant suit le plus souvent un mot en revanche un verbe n'est pas suivie d'un determinant. Ainsi, l'algorithme était capable d'atteindre 70% de précision mais cela imposait une relecture des tags à la main pour permettre d'annoter correctement le corpus. Ce corpus est à la base de beaucoup de recherche en traitement automatique des langues et a inspiré

de nombreux autres corpus.

Du côté de l'Europe, au alentour des années 80, les scientifiques commence à utilisé des modèles de markov caché. Les modèles de markov caché sont des automates à états dont les transitions sont des fonctions de probabilité. Le principe de l'algorithme était donc de construire une table de probabilité de certaines séquence de mot dans une phrase. Par exemple, connaissant un mot dans la phrase, qu'elle serait le mot suivant ?. Ces modèles permettent d'atteindre une précision de l'ordre de 90%. En revanche, ces modèles s'appuie sur des corpus spéciaux permettant d'atteindre un grand pourcentage de précision. Le problème est maintenant de savoir comment se comporte ces modèles lorsqu'on les utilisent sur des textes ayant des contextes très différents. Notamment, nous étudierons l'impact du changement de domaine entre l'ensemble d'apprentissage et l'ensemble de test. Nous passerons de corpus spécialisé dans des textes bien construits comme des articles de journaux à des corpus de tweets, phrases en langages courant ayant un nombre de charactères limité.

Dans la suite de ce rapport, nous allons étudié l'utilisation d'algorithme d'apprentissage machine pour traité le problème du PoS tagging. Plus particulièrement, nous utiliserons le modèle de génération d'arbre de décision et développerons un ensemble de feature permettant d'atteindre un bon score de précision. Avant cela, nous passerons en revue les corpus dont nous disposons pour entrainer notre modèles et étudierons quelques valeurs statistiques sur ceux-ci.

Chapitre 2

Etude des corpus

2.1 Présentation général

Dans ce projet, nous nous sommes attardés sur les corpus français. Nous disposons d'un total de 6 coorpus ayant chacun un ensemble de données d'entrainement, un ensemble de données de test et un ensemble de données de développement. L'ensemble de données d'entrainement étant utilisé pour ajuster les paramètres de nos modèles dans le but de faire les meilleurs prédictions possibles, l'ensemble de test est utilisé pour évaluer notre modèles à chaque modifications des hyper-paramètre. En effet, les algorithmes de machine-learning dispose d'un ensemble de paramètre permettant de modifier leurs comportements, on appele ceux-ci des hyper-paramètres. Une certaine instance d'hyper-paramètre peut donner de très bon résultat alors qu'une autre peut donner des résultats catastrophique. Un des enjeux est alors de trouver la meilleurs instances d'hyper-paramètre pour notre modèle.Nous verrons cela plus en détaille dans le chapitre dédié aux résultats de nos modèles.

Pour en revenir sur nos ensembles de corpus après cette petite parenthèse, nous avons donc un ensemble de 6 corpus de texte français appelé ftb, gsd, partut, pud, sequoia et spoken. Nous disposons ensuite de deux corpus supplémentaire de test appelé foot et natdis. Ces corpus sont utilisé pour évaluer l'impact du changement de context entre l'ensemble d'apprentissage et l'en-

semble de test/dev.

2.2 Contexte des différents corpus

— Corpus sequoia : Ce corpus contient des phrases provenant d'extrait de séance du parlement européen, du site Wikipédia FR, du corpus Est républicain contenant des extrait de journaux et de l'agence européenne du médicament. Les phrases de ce corpus proviennent donc de source diverse mais dans un contexte assez politique / scientifique avec des phrases bien structuré. Ce corpus vient en complément du corpus FTB pour augmenté le nombre de domaine et de genre couvert.

— Corpus FTB French TreeBank : Ce corpus est un projet initié en 1997. Il regroupe un ensemble d'article du Monde sur une période de 1987 jusqu'en 1995. Par exemple, la première phrase de l'ensemble d'entrainement provient d'un article du Monde de 1992 traitant d'un sujet politique. On peut s'attendre à avoir un ensemble de phrase traitant de domaine varié mais centré principalement sur l'actualité dans la période concerné lors de la création du corpus.

— Corpus Partut : Le corpus Partut est un corpus produit par l'université de Turin. Il comprend des extraits de phrases provenant des textes de loi de l'union européenne, de la déclaration universelle des droits de l'homme, de la licence Creative Commons, des extraits de pages du site Facebook, des extraits de réunion du parlement européen et finalement des extraits oraux pris sur le web.

— Corpus PUD : Ce corpus provient du projet Universal Dependencies et le nom de ce corpus particulier est Parallèle Universal Dependencies. Il est composé de phrases provenant du domaine journalistique plus particulièrement de l'actualité. Il est également composé de phrases provenant de pages Wikipédia pris aléatoirement.

— Corpus GSD : Ce corpus provient également du projet Universal De-

pendencies. Il couvre des phrases de domaine varié avec des articles de journaux, de pages Wikipédia, de blogs et de critiques de divers produit.

— Corpus Spoken : Ce corpus comprend un ensemble de phrase dans un langage oral. Il comprend donc une retranscription fidèle de la parole des interlocuteurs avec notamment la présence d'interjection avec le "heu" que l'on peu très souvent entendre à l'oral. En revanche, avec la multitude de corpus oral présent dans la litérrature, nous n'avons pas été en mesure de déterminé précisément la provenance de celui-ci. Notre principal hypothèse est qu'il s'agit du corpus CFPP2000 contenant un ensemble d'extrait d'interview sur les quartiers de Paris et sa proche banlieue notamment au regard des phrases présente dans le corpus.

Nous disposons également de deux ensembles de tests supplémentaires. Il s'agit de l'ensemble foot et natdis. L'ensemble de test foot est constitué de tweet de fan de football. Les phrases sont donc consitué d'élément ne pouvant pas se retrouver dans les autres corpus comme les émoticons. Le second ensemble de test appelé natdis est également un ensemble de tweet traitant cette fois du domaine des catastrophe naturelle. Ces ensembles de tests seront principalement utilisé pour évaluer l'impact du changement de domaine entre l'ensemble d'apprentissage et l'ensemble de test.

2.3 Les classes du problème

Comme nous l'avons vu en introduction, notre problème est l'étiquetage morpho-syntaxique qui consiste à donner une étiquette à chaque mot d'une phrases en fonction de sa nature syntaxique. L'objectif de cette section est donc d'étudier ces différentes étiquettes qui constitueront les classes que notre modèle devra prédire.

Nous avons un total de 26 étiquettes différentes dans les corpus français dont voici la liste ci-dessous :

— Noun : nom commun

- SCONJ : conjonction de subordination
- ADP : adposition
- PROPN : nom propre
- CCONJ : conjonction de coordination
- ADJ : adjectif
- DET : determinant
- PRON : pronom
- VERB : verbe
- CONJ : conjonction
- NUM : nombre
- ADV : adverbe
- PART : particule
- PUNCT : ponctuation
- AUX : auxiliaire
- SYM : symbole
- INTJ : interjection
- ADP + ADP : adposition liée, Il n'existe qu'un seul cas de ce tag dans tous les corpus et il est présent dans l'ensemble de test FTB et le mot associé est "jusqu'au"
- ADP + DET : adposition pouvant également être un determinant, exemple tiré du corpus foot : {'Du', 'Au', 'AU', 'du', 'DES', 'aux', 'au', 'DU', 'des' }
- ADP + PRON : adposition pouvant également être un pronom, exemple tiré du corpus de developpement FTB : {'auxquelles', 'duquel', 'desquelles', 'auxquels', 'auquel' }
- DET + NOUN : determinant pouvant également être un nom : il n'existe qu'un seul cas dans tous les corpus, le mot "des" dans l'ensemble d'apprentissage du corpus PUD.
- AT : @, utilisé dans les corpus de tweet pour faire une référence à quelque chose
- URL : lien hypertexte

— E : emoticon
— SHARP : # sur twitter
— X : autre

Au delà des étiquettes standard comme Noun ou Verb, il y a quelque étiquettes spécial qu'il convient d'expliquer plus en détails. L'étiquette X est notamment utilisé pour désigner les mots inconnues du corpus ou ceux ayant une ortographe différente du mot habituelle. L'étiquette E est utilisé dans les corpus de tweet pour désigner les émoticons, ces étiquettes vont probablement posé certain problème lors du changement de domaine. L'étiquette SHARP est utilisé pour désigner les balises # utilisé pour faire référence à d'autre tweet. Cette étiquette ne devrait également pas être retrouvé dans les autres corpus. L'étiquette URL permet de désigner les liens hypertexte dans une phrase. L'étiquette SYM sert à désigner des symboles spéciaux dans les tweets comme des drapeaux. L'étiquette AT sert à désigner le symbole utilisé par les tweets pour faire référence à quelque chose.

2.4 Analyse statistique des corpus

Nous avons identifiés les corpus et les différentes classes que notre algorithme devra prédire. Nous allons maintenant étudier plus en détails l'organisation et la structure des corpus au moyens de certaines observables statistique. Nous agrémenterons cette partie de quelques graphes et tableau pour rendre ceci plus lisible.

Dans un premier temps, nous allons étudié la taille de chacun des corpus en nombre de phrases, nombre de mot et nombre de mot unique.

Comme nous pouvons le voir dans le tableau 2.1, les deux corpus les plus fourni en données d'entrainement sont les corpus GSD et FTB. Ce sont les corpus ayant le plus grand nombre mots/ mots uniques.

Dans le jupiter-notebook, deux autres mesures statistiques ont été effectué. La première permet de determiné les mots et labels les plus fréquents appa-

Corpus \ Statistique		nombre de phrases	Nombre de mots	Nombre de mots uniques
gsd	train	14450	345009	41072
	test	416	9742	3172
	dev	1476	34664	8964
sequoia	train	2231	49173	8185
	test	456	9740	2922
	dev	412	9724	2789
ftb	train	14759	442228	27127
	test	2541	75073	9845
	dev	1235	38763	6545
spoken	train	1153	14952	2529
	test	726	10010	1980
	dev	907	10010	1894
pud [a]	train	803	23324	3709
	test	1000	24138	6004
partut	train	803	23324	3709
	test	110	2515	815
	dev	107	1822	715
foot	test	743	13985	2638
natdis	test	622	12044	1631

a. Ce corpus ne dipose pas d'ensemble de developpement d'où son absence

TABLE 2.1 – Récapitulatif de la taille des corpus

raissant dans chacun des corpus. La seconde permet de déterminé les mots les plus ambigu de chaque corpus. Cette mesure statistique permet de montrer un des problème que nous soulevions dans l'introduction . Si l'on prend le corpus foot, l'ensemble de test contient le mot "tout", celui-ci est associé à pas moins de 6 labels différents. Ce mot sera alors très difficile à discriminé par notre algorithme d'apprentissage et risque de donner des prédictions incorrectes dans pas mal de cas.

Out Of vocabulary word

Nous allons maintenant passer à l'étude de la qualité de l'ensemble d'apprentissage sur l'ensemble de test grâce à la mesure d'out of vocabulary word (OOV). Ce sont les mots qui apparaissent dans l'ensemble de test mais pas dans l'ensemble d'apprentissage. Ainsi, plus le pourcentage est faible, plus l'ensemble de test et d'apprentissage ont un vocabulaire proche. Inversement, un pourcentage élevé indique une grande différence entre le vocabulaires de l'ensemble d'apprentissage et de l'ensemble de test. La meilleur solution serait d'avoir un ensemble d'apprentissage ayant le vocabulaire le plus riche possible

training set \ test set	test	dev	foot test	natdis test
gsd	1.27927	4.9984	2.53946	1.22708
sequoia	7.66183	7.28996	15.9475	9.1687
ftb	5.60695	3.1985	4.39778	2.10376
spoken	26.5691	23.5813	39.0943	28.3173
pud	44.9089		31.7945	20.618
partut	4.99558	6.08047	31.7945	20.618

TABLE 2.2 – Pourcentage d'OOV entre l'ensemble d'apprentissage de chaque corpus et son ensemble de test/developpement, l'ensemble de test foot et natdis

pour pouvoir faire tendre ce pourcentage d'OOV vers zéro. Le modèle serait alors en mesure de pouvoir prédire plus précisement la classe syntaxique des mots car il aurait appris les meilleurs paramètres pour ces mots.

La tableau 2.2 nous donne les résultats du calcul du pourcentage d'OOV entre les ensembles d'apprentissage de tous les corpus avec leurs ensembles de test et de developpement. Au dela des ensembles spoken et pud, le pourcentage d'OOV entre le trainset et le testset est très faible ce qui nous fait dire que les performance de notre modèle sera plus correct sur ces corpus. En revanche, le corpus PUD a un pourcentage d'OOV très élevé, le plus élevé après spoken. Cela signifie qu'un grand nombre de mots inconnue vis-à-vis de l'entrainement du modèle sont présent dans l'ensemble de test. On s'attend à avoir des performance plus faible de notre modèle sur ces deux corpus.

Pour les ensembles de test foot et natdis, nous avons un pourcentage d'OOV relativement faible avec les corpus gsd, sequoia et ftb. Nous serions tenté de dire que notre modèle fera un score correct avec ces corpus ci. Il ne faut pas oublier en revanche que cette mesure ne s'appuie que sur la présence des mots, la structure même de phrases, l'agencement des mots et leurs ambiguité n'est pas représenter par cette mesure.

Divergence de KullBack-Leibler

La divergence de KullBack-Leiber tire son origine de la théorie de l'information. L'objectif de la théorie de l'information est de déterminer la quantité d'information qu'il y a dans un ensemble de données. La métrique la plus importante de la théorie de l'information est la mesure de l'entropie notée H. Sa définition est : $H = -\sum_{i=1}^{N} p(x_i) \cdot log_2(p(x_i))$ L'utilisation d'un log en base 2 nous permet d'approcher la quantité minimal, la borne inférieur de bits nécessaire pour encoder la donnée x_i.

Maintenant que l'on peut connaitre la quantité d'information contenue dans notre ensemble, nous voulons quantifier la quantité d'information que l'on perd lorsque l'on passe de notre modèle à des données inconnues. C'est l'objectif de

training set \ test set	test	foot test	natdis test
gsd	0.000481962	0.000314367	0.000381745
sequoia	0.000241021	0.000163088	0.000218926
ftb	3.23858e-05	0.00033936	0.000410431
spoken	0.000124028	6.26248e-05	0.000151464
pud	2.62551e-05	0.000114489	0.000185953
partut	0.00121712	0.000114489	0.000185953

TABLE 2.3 – Valeur de la divergence de KullBack-Leibler entre l'ensemble d'apprentissage et les ensembles de test pour chaque corpus

la divergence de KullBack-Leibler. Sa définition est la suivante : $D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \cdot [log_2(p(x_i)) - log_2(q(x_i))] = \sum_{i=1}^{N} p(x_i) \cdot log_2(\frac{p(x_i)}{q(x_i)})$

Lorsque la valeur de cette métrique augmente, cela indique que l'on perd plus d'information en passant de la distribution observée à la distribution paramétrisé et inversement lorsque la métrique diminue. A noté que la divergence de KullBack-Leibler n'est pas symetrique, ainsi $DKL(observed||param) \neq DKL(param||observed)$.

Dans notre exemple, l'utilisation de la divergence de Kullback-Leibler entre un ensemble d'apprentissage et de test nous permettra de quantifier la quantité d'information que le classifier n'aura pas réussi à récupéré par le biais de son apprentissage. Ainsi, il sera judicieux d'utiliser un ensemble d'apprentissage ayant une divergence la plus faible possible vis-à-vis de l'ensemble de test.

Le tableau 2.3 nous donne les valeurs de la divergence de Kullback-leibler suivante dans l'ordre du tableau :

$DKL(test|train), DKL(foottest|train), DKL(natdistest|train)$.

On constate que les valeurs de DKL sont relativement faible entre le train-set et le testset pour la plupart des corpus hormis gsd et sequoia. Si l'on établit un parallèle avec la table 2.2, sequoia avait un haut pourcentage d'OOV, ce

qui indique une très grande différences dans le vocabulaire entre le trainset et le testset de Sequoia. La DKL nous indique qu'en plus, il y une grande quantité d'information différentes entre les deux ensembles, nous nous attendons à de faibles performances pour notre modèle. En revanche, gsd avait un faible pourcentage d'oov or sa divergence est grande donc une grande quantité d'information sera perdu entre le trainset et le testset ce qui peut nous faire dire que notre modèle peut faire de mauvais résultat entre ces deux ensembles.

Pour ce qui est des ensembles de test foot et natdis, les valeurs de divergence sont relativement élevé ce qui n'est pas une surprise étant données qu'il s'agit d'ensemble traitant de domaine différents de l'ensemble d'apprentissage. On s'attend évidemment à avoir de faible performance sur ces ensembles de test ci.

Perplexité

Nous passons maintenant à la dernière metrique permettant d'évaluer nos ensembles de données. La perplexité est une autre métrique de la théorie de l'information et s'appuie, comme pour la divergence de KullBack-Leibler, sur l'entropie de Shannon. la perplexité au sens de la théorie de l'information est tous simplement PP[1] $= 2^{H(p)}$ où $H(p)$ est l'entropie. Nous avons vu que l'entropie est la quantité d'information moyenne requise pour coder le résultat d'une variable aléatoire. La perplexité est donc l'exponentiation de l'entropie et correspond donc au nombre d'élément que l'on peut coder en sortie de notre distribution de probabilité p. Ainsi, dans le contexte de l'apprentissage machine, la perplexité nous permet d'approcher la quantité d'information que l'on peut espérer via l'apprentissage de notre modèle sur cette distribution de probabilité p(ou notre ensemble d'apprentissage dans notre cas). Une faible valeur de perplexité nous indique que notre modèle pourra prédire correctement des données, à l'inverse une grande perplexité signifie que notre modèle aura

1. On note PP pour la perplexité dans la suite du rapport

corpus	trainset	testset	devset
foot		1.57913	
gsd	2.40795	1.38588	1.65074
sequoia	1.79879	1.39286	1.42592
ftb	2.63512	2.04522	1.84254
spoken	1.67794	1.57774	1.59158
pud	1.73847	1.63067	
natdis		1.44732	
partut	1.73847	1.23857	1.19951

TABLE 2.4 – Valeur de perplexité pour l'ensemble des corpus

du mal à prédire des données, il y aura une plus grande incertitude sur les prédictions.

Pour les calculs de perplexité, nous sommes passer en échelle logarithimique en base e pour évité l'effet d'underflow du à de faible valeurs de probabilité. Nous utilisons du coup une exponentiation par la constante e car nous utilisons le logarithme népérien. De plus, la distribution de probabilité que nous utilisons est celle du modèle trigram $P(w_i|w_{i-2}, w_{i-1}) = \frac{cout(w_{i-2},w_{i-1},w_i)}{count(w_{i-2},w_{i-1})}$ et le calcul de perplexité que nous faisons est donc le suivant :

$PP = e^{(-\sum_{i=1}^{N} ln(P(w_i|w_{i-2},w_{i-1})))/N}$

La table 2.4 nous permet d'avoir les valeurs de perplexité pour chacun des ensembles de données de chaque corpus. On constate que l'ensemble d'entrainement de gsd et l'ensemble d'entrainement et test de ftb ont les valeurs les plus grandes de perplexité. On peut expliquer cela grâce au mots les plus ambigu de chaque corpus. Si l'on regarde le jupyter-notebook sur les mots les plus ambigu de chaque corpus, ces ensembles possèdent les mots les plus ambigu des corpus avec notamment le mot 'A' ayant plus de 6 labels différents associée dans l'ensemble de train ftb.

Chapitre 3

Classifieurs multi-classe

3.1 Features

Bien que nous ayons deux classifieurs, nous avons décidé de garder les mêmes features pour les deux, essentiellement par manque de temps. Si nous avions pu passer plus de temps sur le classifieur naïf bayésien, nous aurions probablement fini par avoir deux ensembles de features spécialisés. Dans la suite du rapport, l'ensemble des features sera noté F.

Les features sont :

— word : le mot haché
— pos : la position du mot dans la phrase
— first : si le mot est positionné en premier. Concrètement la valeur de cette feature dépend uniquement de la position du mot et paraît donc superflu au premier abord, mais nous avons pensé que cela pourrait aider l'arbre de décision sur lequel nous nous sommes concentrés.
— last : si le mot est positionné en dernier
— len : la longueur du mot en nombre de caractères
— startUpper : si le mot démarre par une majuscule, ce qui peut aider à distinguer les noms propres
— hasUpper : si le mot contient une majuscule
— allUpper : si le mot n'est constitué que de majuscules

— all-special : le mot n'est constitué que de caractères spéciaux, i.e ni alphabétiques, ni numériques
— w-1, w-2, w+1, w+2 : les 4 mots environnants, hachés
— prefix-1, prefix-2, prefix-3 : respectivement le premier caractère, les deux premiers, et les trois premiers, tous hachés. Si le mot a moins de 3 caractères, certaines de ces features seront égales.
— suffix-1, suffix-2, suffix-3 : similaires mais pour la fin du mot et pas le début.

3.2 Arbre de décision

Pour résoudre notre problème de PoS tagging, nous avons choisi d'implémenter un arbre de décision. Il s'agit d'un algorithme qui fabrique un arbre binaire dont un chemin dans cette arbre est une suite de test à effectuer sur une donnée pour ensuite arriver à une prédiction au niveau des feuilles de l'arbre. Contrairement à d'autre algorithme de machine learning, celui-ci est un algorithme qui permet de comprendre les prédictions effectué par celui-ci. C'est un algorithme boite blanche contrairement à d'autre qui sont des algorithme boite noir comme les réseaux de neurone. Si l'on souhaite comprendre une prédiction, il suffit de refaire le chemin de décision dans l'arbre.

La construction de l'arbre en revanche est un peu plus subtile que pour la prédiction. Le principe est de construire un arbre où chaque noeud contient un test qui doit permettre de séparé l'ensemble des données en deux. Nous aurons d'un côté les labels ayant passé le test et de l'autre ceux n'ayant pas réussi à passer le test. Chaque test doit permettre de réduire de plus en plus l'ensemble des données de sorte à ce qu'une fois arriver à une feuille, on doit permettre de déterminer le label de la donnée. La génération des tests à chaque noeud ainsi que le choix du meilleurs test se font grâce à deux hyper-paramètre qui seront explicité dans la section suivante.

Algorithm 1 Algorithme de contruction de l'arbre de décision

```
1: function BUILDTREE(X, Y, max_depth)
2:     if taille(X) < threshold ∨ taille(Y) < threshold ∨ max_depth == 0 then
3:         return max{P(c) [a] | ∀c ∈ Y}
4:     else
5:         bestTest, Xyes, Yyes, Xno, Yno ← meilleurTest(X, Y)
6:         left ← buildTree(Xyes, Yyes, max_depth − 1)
7:         right ← buildTree(Xno, Yno, max_depth − 1)
8:         return (bestTest, left, right)
9:     end if
10: end function
```

a. où P(c) est la probabilité d'apparition de la classe c dans Y

Hyper-paramètre

Le pseudo-code 1 permet de construire l'arbre de décision. Plusieurs hyper-paramètres permettent de changer le comportement du pseudo-code. Dans un premier temps, nous avons deux hyper-paramètre permettant de limiter la récursion de l'algorithme. Le paramètre max_depth permet de limiter la profondeur maximale de l'arbre tandis que le paramètre threshold permet de stopper une branche de l'arbre dès que l'un des ensembles d'entrée est inférieur au threshold. L'hyper-paramètre max_depth rend sensible la construction de l'arbre au phénomène de sur-apprentissage. Si la limite de profondeur de l'arbre est trop grande, celui-ci risque d'avoir des tests trop spécifiques à l'ensemble d'apprentissage et la précision des prédictions diminueront sur l'ensemble de test.

Nous avons également deux autres hyper-paramètres importants permettant de changer radicalement la construction de l'arbre et l'efficacité de celui-ci. Le premier concerne la génération de test. Nous pouvons générer un test pour splitter les étiquettes en fonction de la moyenne ou de la médiane des features.

Algorithm 2 Algorithme de choix du meilleur test

1: **function** MEILLEURTEST(X, Y)
2: $bestTest \leftarrow (0, 0)$
3: $bestXyes \leftarrow \emptyset$
4: $bestYyes \leftarrow \emptyset$
5: $bestXno \leftarrow \emptyset$
6: $bestYno \leftarrow \emptyset$
7: **for all** $features \in X$ **do**
8: $\overline{f} \leftarrow select\_test(features)$
9: $yes\_answer \leftarrow \{y_x | y \in Y, \forall x \in X, x \leq \overline{f}\}$
10: $no\_answer \leftarrow \{y_x | y \in Y, \forall x \in X, x > \overline{f}\}$
11: $score \leftarrow computeScore(yes\_answer, no\_answer)$
12: **if** score is better than bestScore **then**
13: $bestTest \leftarrow (\overline{f}, features)$
14: $bestYyes \leftarrow yes\_answer$
15: $bestYno \leftarrow no\_answer$
16: $bestXyes \leftarrow \{x_y | \forall x \in X, y \in bestYyes\}$
17: $bestXno \leftarrow \{x_y | \forall x \in X, y \in bestYno\}$
18: **end if**
19: **end for**
20: **return** $bestTest, bestXyes, bestYyes, bestXno, bestYno$
21: **end function**

Cela est représenté par l'hyper-paramètre nommée gen_test dans le code python et par la fonction *select_test* dans le pseudo-code 2

Lorsqu'un nouveau test est généré, nous calculons son score. Nous souhaitons gardé le test qui sépare au mieux l'ensemble de données et la méthode de calcul du score est un hyper-paramètre. Celui ci est représenté par l'hyper-paramètre nommée split_criterion dans le code python et par la fonction *computeScore* dans le pseudo-code 2.

Nous avons la méthode de calcul utilisant l'entropie et qui calcul le gain d'information entre l'ensemble de donnée avant la séparation et les ensembles de données après la séparation. La formule du calcul du gain d'information que nous effectuons est la suivante :

$$I_g = H(Y) - P(yes\_answer) * H(yes\_answer) + P(no\_answer) * H(no\_answer) \tag{3.1}$$

La probabilité P(yes_answer) correspond au nombre de labels présents dans l'ensemble yes_answer divisé par le nombre de labels dans l'ensemble des labels Y avant la séparation. H(yes_answer) correspond à l'entropie de l'ensemble yes_answer. Au final, le meilleur test est celui ayant le gain d'information maximal donc la valeur maximum de I_g.

La seconde méthode de calcul que nous utilisons est l'indice de gini que nous calculons de la façon suivante :

$$Gini = 1 - \sum_{y \in yes\_answer} P(y) \tag{3.2}$$

$$scoreTest = P(yes\_answer) * Gini_{yes\_answer} + P(no\_answer) * Gini_{no\_answer} \tag{3.3}$$

Le coefficient de gini est utilisé pour mesurer les inégalités de répartition des labels entre l'ensemble yes_answer et no_answer. Le but est de réduire

ce degré d'inégalité, c'est pour cela qu'on cherche le test ayant le score la plus faible valeur de scoreTest.

Optimisation et validation croisée

Concernant l'implémentation de notre arbre de décision, celle-ci a changé lorsque nous avons voulu effectué nos tests pour la section résultat. La construction de l'arbre prenait en effet beaucoup trop de temps. La fonction fit() de la classe DecisionTreeClassifier était codé essentiellement avec des boucles for en python qui ralentissait sensiblement le temps d'exécution. Nous avons alors changé l'implémentation et nous sommes passés sur des fonctions de la bibliothèque numpy. Cela a permis de réduire de façon significative le temps de calcul de nos arbres. En revanche, le temps de calcul sur les gros corpus GSD et FTB est toujours aussi long.

Pour la génération de nos résultats, nous utilisons la méthode de la validation croisée pour determiner les meilleurs hyper-paramètres de notre modèle. Elle consiste à entrainer notre modèle sur une instance d'hyper-paramètres donnée grâce à l'ensemble puis à tester notre modèle sur l'ensemble de test. On sauvegarde alors les résultats de cette session de test puis on change d'instance d'hyper-paramètres et on recommence. Le but est de trouver l'instance d'hyper-paramètres maximisant le score du modèle sur l'ensemble de test. La question que l'on peut se poser est alors : pourquoi ne pas utiliser l'ensemble d'apprentissage pour évaluer le score du modèle ? Tout simplement pour éviter le phénomène de sur-apprentissage du modèle, un problème récurrent en machine-learning. Le sur-apprentissage intervient lorsque le modèle commence à ajuster un peu trop finement ces paramètres d'apprentissage sur l'ensemble d'apprentissage. Le modèle perd alors les paramètres lui permettant d'avoir les caractéristiques générales du problème et gagne en précision sur les données spécifiques de l'ensemble d'apprentissage.

Le but de la validation croisée est donc d'éviter ce phénomène en monitorant l'évolution du modèle grâce au jeu de données de test. Lorsque le

score sur le jeu de données de test diminue alors cela signifie que le modèle est probablement en train de sur-apprendre. L'ensemble de développement est lui plutôt utilisé pour évaluer le modèle final une fois la meilleure instance d'hyper-paramètres trouvée. Parfois certains jeux de données peuvent ne pas avoir besoin d'ensemble de développement, mais il est toujours préférable de tester une dernière fois notre modèle sur d'autres jeux de données inconnue.

Résultats

Nous allons présenter dans cette section les différents résultats de nos tests sur les différents corpus avec notre modèle d'arbre de décision et les différentes features.

Pour la présentation de nos résultats,nous dessinons des graphes au moyen de la bibliothèque python matpotlib qui permettent d'observer l'évolution du score de notre modèle en fonction des différentes instance d'hyper-paramètre possible. Nous dessinons 4 sous-graphes qui permettent d'observer l'évolution de la courbe de score en fonction de la profondeur maximale de l'arbre. Les 4 sous-graphes permettent de rendre compte des deux hyper-paramètres concernant le calcul du score des tests et de la méthode de génération des tests.

Nous dessinons également la matrice de confusion qui nous permet de nous rendre compte des différents labels ayant été difficiles à classer par notre modèle. Nous calculons la matrice de confusion entre l'ensemble d'entraînement et l'ensemble de test de chaque corpus. Nous calculons ensuite la matrice de confusion entre l'ensemble d'entraînement de chaque corpus avec les meilleurs hyper-paramètres et l'ensemble de test foot et natdis afin d'évaluer l'impact du changement de domaine.

Sur les performances de notre modèle d'arbre de décision sans changement de contexte, nous obtenons entre 70% et un peu moins de 90% de précision.

Les performances relativement élevés entre l'ensemble d'apprentissage et de test du même corpus sont cohérentes vis-à-vis des mesures avec les métriques que nous avions calculées dans le chapitre 2. Seul l'ensemble pud a des perfor-

test set / training set	test	foot test	natdis test
sequoia	89.68%	59.92%	X
spoken	84.75%	47.21%	X
pud	74.07%	57.70%	X
partut	88.31%	58.05%	X

TABLE 3.1 – Performance du modèle en pourcentage de bonne prédiction sur les labels avec les meilleurs hyper-paramètre

mances inférieures à 80% mais cela s'explique par la mesure de l'OOV. Il s'agit de l'ensemble ayant la valeur d'OOV la plus élevée et donc ces performances sont plus faibles que les autres.

Pour les performances concernant les ensembles foot et natdis, il s'agit des ensembles ayant un contexte différent des ensembles d'apprentissage. Cela explique les faibles performances de notre modèle.

L'ensemble des graphes est disponible dans le dossier results/DecisionTree à partir de la racine du projet.

3.3 Classifieur naïf bayésien

En plus de l'arbre de décision, nous avons voulu implémenter un classifieur naïf bayésien. Cependant, par manque de temps notamment à cause des autres projets que nous devions réaliser en même temps, nous avons passé beaucoup moins de temps dessus ce qui explique ses très mauvais résultats, si mauvais que le classifieur est complètement inutilisable en l'état.

La première étape pour réaliser un tel classifieur consiste à analyser l'ensemble de données d'apprentissage. On le subdivise d'abord par étiquette pour obtenir un sous-ensemble pour chaque classe. Sur chacun de ces ensembles, on

calcule ensuite le cardinal, la moyenne et l'écart type.

Le classifieur est ensuite prêt à prédire la classe d'une donnée qu'on lui donne. On calcule la probabilité que celle-ci fasse partie de chaque classe puis on prend la meilleure. La probabilité que la donnée x fasse partie de la classe C est estimée avec la formule suivante :

$$P(C \mid x) = \frac{P(x \mid C) \cdot P(C)}{P(x)} \tag{3.4}$$

$P(C \mid x)$ est ici la probabilité que x soit de la classe C, soit exactement ce qu'on veut calculer. $P(x \mid C)$ est la probabilité d'observer x sachant que la donnée observée est de la classe C. $P(C)$ est la probabilité de tomber sur une donnée de la classe C globalement, et similairement $P(x)$ la probabilité de trouver x sans hypothèse sur la classe.

Cette formule est le théorème de Bayes, d'où le nom du classifieur. Puisqu'on ne s'intéresse qu'à la probabilité la plus grande parmi toutes les classes, il est inutile de la calculer complètement, étant donné que la valeur de $P(x)$ ne changera pas d'une classe sur l'autre. Au final il suffit donc de calculer :

$$P(x \mid C) \cdot P(C) \tag{3.5}$$

Malheureusement, $P(x \mid C)$ nécessiterait un nombre extrêmement grand de données pour pouvoir être calculé correctement. C'est pourquoi il faut faire une hypothèse supplémentaire sur les données, d'où le qualificatif "naïf" de la méthode de classification ; ici, on suppose que chaque feature suit une loi normale et qu'elle est indépendante des autres, ce qui n'est évidemment pas le cas la plupart du temps ; malgré tout, la classification naïve bayésienne est réputée assez efficace pour le PoS Tagging. En supposant une telle distribution, on peut utiliser l'équation suivante, avec x_i la feature i de x, et $\mu_{C,i}$ et $\sigma_{C,i}$ respectivement la moyenne et l'écart type de cette feature pour les données de la classe C :

$$P(x \mid C) = \prod_{i \in F} \frac{1}{\sigma_{C,i}\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x_i - \mu_{C,i}}{\sigma_{C,i}})^2} \tag{3.6}$$

Hyper-paramètres

Lors de l'implémentation, nous avons rencontré un souci sous la forme d'une division par zéro lorsque l'écart type vaut 0. Certaines de nos features étant des booléens, le cas doit être géré. Notre première approche pour résoudre le problème était intuitive : si l'écart type vaut 0 et que la feature observée vaut la moyenne, alors la probabilité que la donnée appartienne à la classe considérée n'est a priori pas nul ; au contraire, si les deux valeurs sont différentes, il y a très peu de chances que ce soit le cas.

Après de rapides recherches, en remplaçant l'écart type par une valeur prédéfinie σ dans ce cas, on obtient une formule avec des résultats satisfaisants. Cependant σ ne doit être ni trop petit, ni trop grand. Nous avons décidé de faire de cette valeur un hyper-paramètre.

Résultats

Idées de développement

Les résultats de notre classifieur naïf bayésien sont très mauvais. Un des problèmes (en-dehors des bugs probables) pourrait être que nous utilisons une fonction de hachage sur les mots pour les convertir en entier, ce qui entre très probablement en contradiction avec l'hypothèse que chaque feature suit une loi normale. De même, il est absurde de parler de distribution normale sur des variables booléennes.

Une solution serait de considérer différents types de features, par exemple booléennes, catégoriques, résultats de hachage, etc. Le calcul de probabilité

effectué ensuite diffèrerait selon le type de la variable. Il reste à déterminer quelles catégories retenir, et quelles sont les fonctions qui y sont adaptées.