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 phrase. 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 connu en linguistique. Il s'agit de la façon dont les humains comprennent les mots et leur fonction dans une phrase. Il s'agit d'un processus d'apprentissage réalisé pendant l'enfance lorsque nous lisons ou écoutons pour la première fois. Lorsque nous arrivons à identifier les structures syntaxiques d'une phrase, cela signifie que nous avons compris son contexte et sa signification. Avec le développement des techniques algorithmiques modernes et l'augmentation des vitesses de traitement des ordinateurs, il est naturel de vouloir donner à nos ordinateurs la capacité de pouvoir analyser le langage humain que ce soit écrit ou oral. Cela permettrait d'améliorer les algorithmes de détection des erreurs, les assistants personnels tels que Siri ou Cortana ou encore le domaine de la robotique avec par exemple Nao.

1.2 Approche naïve et identification des problèmes

Un algorithme naïf consisterait à, étant donné tout les mots possibles du dictionnaire d'un langage, renvoyer la bonne classe syntaxique du mot, à l'aide d'une sorte de grande table de hashage qui associerait chaque mot du dictionnaire avec sa classe syntaxique. Nous pouvons tout de suite identifier une limite à cette approche.

Prenons l'exemple de deux phrases simples :

— 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 sa classe syntaxique n'est pas la même. Dans la première phrase, "son" est utilisé comme étant un adjectif 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 naïve. Un même mot peut avoir plusieurs classes syntaxiques 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 cet exemple, le mot "voiture" peut être remplacé par n'importe quel mot contenant également trois syllabes. Ainsi, la différence entre "utiliser" 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 remplacer le mot par n'importe quel 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 naïve de la table de hash, il se pose la question du nombre de mots d'une langue mais également de la définition d'un mot. Il convient également d'identifier de quelles langues on parle. L'humanité, au cours de son histoire, s'est scindée en de multiple groupes sur toute la

planète et chacun a évolué avec sa propre histoire, culture et langue - sans compter sur les variantes liées à la localisation géographique, au contexte social et culturel, etc. Ainsi, notre algorithme doit-il se focaliser sur une seule langue ou essayer de généraliser ? Sur la question de la nature des mots, il y a le cas des langues qui choisissent de composer des mots par association comme l'allemand ou encore les langues agglutinantes comme le turc. Le français n'est pas en reste non plus..

Il est donc essentiel de trouver une autre approche, plus générale, 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. Très vite, les scientifiques et linguistes ajoutent 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 déterminant précède le plus souvent un mot et pas un verbe. 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, aux alentours des années 80, les scientifiques commencent à utiliser des modèles de Markov cachés. Les modèles de Markov cachés 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 séquences de mot dans une phrase. Par exemple, connaissant un mot dans la phrase, quel est le mot suivant le plus probable ? Ces modèles permettent d'atteindre une précision de l'ordre de 90%. En revanche, ces modèles s'appuient sur des corpus spéciaux permettant d'atteindre un grand pourcentage de précision. Le problème est maintenant de savoir comment se comportent ces modèles lorsqu'on les utilise sur des textes ayant des contextes très différents.

Ce souci de généralisation est central à ce projet. Nous passerons de corpus spécialisés dans des textes bien construit comme des articles de journaux à des corpus de tweets, phrases en langage courant ayant un nombre de caractères limités.

Dans la suite de ce rapport, nous allons étudier l'utilisation d'algorithmes d'apprentissage machine pour traiter le problème du PoS Tagging. Plus particulièrement, nous utiliserons le modèle de génération d'arbres de décision et développerons un ensemble de features permettant d'atteindre un bon score de précision. Avant cela, nous passerons en revue les corpus dont nous disposons pour entraîner notre modèle 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 des corpus en langue française. Nous disposons d'un total de 6 corpus ayant chacun un ensemble de données d'entraînement, un ensemble de données de test et un ensemble de données de développement. L'ensemble de données d'entrainement est utilisé pour ajuster les paramètres de nos modèles dans le but de faire les meilleures prédictions possibles, l'ensemble de test est utilisé pour évaluer notre modèle à chaque modification des hyper-paramètres. En effet, les algorithmes de machine-learning dispose d'un ensemble de paramètres permettant de modifier leur comportement, on appelle ceux-ci des hyper-paramètres. Une certaine instance d'hyper-paramètres peut donner de très bons résultats alors qu'une autre peut se révéler catastrophique. Un des enjeux est alors de trouver la meilleure instance d'hyper-paramètres pour notre modèle. Nous verrons cela plus en détail 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 textes français appelé ftb, gsd, partut, pud, sequoia et spoken. Nous disposons ensuite de deux corpus supplémentaires de test appelés foot et natdis. Ces corpus sont utilisés pour évaluer l'impact du changement de contexte entre l'ensemble d'apprentissage et les

ensembles de test et développement.

2.2 Contexte des différents corpus

— Corpus sequoia : Ce corpus contient des phrases provenant d'extraits de séances 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 sources diverses mais dans un contexte assez politique / scientifique avec des phrases bien structurées. Ce corpus vient en complément du corpus FTB pour augmenter le nombre de domaines et de genre couvert.

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

— 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 réunions 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 prises aléatoirement.

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

pendencies. Il couvre des phrases de domaines variés avec des articles de journaux, de pages Wikipédia, de blogs et de critiques de divers produits.

— Corpus Spoken : Ce corpus comprend un ensemble de phrases dans un langage oral. Il comprend donc une retranscription fidèle de la parole des interlocuteurs avec notamment la présence d'interjections avec le "heu" que l'on peut très souvent entendre à l'oral. En revanche, avec la multitude de corpus oraux présents dans la littérature, nous n'avons pas été en mesure de déterminer précisément la provenance de celui-ci. Notre principale hypothèse est qu'il s'agit du corpus CFPP2000 contenant un ensemble d'extraits d'interviews sur les quartiers de Paris et sa proche banlieue, notamment au regard des phrases présentes dans le corpus.

Nous disposons également de deux ensembles de test supplémentaires. Il s'agit des ensembles foot et natdis. L'ensemble de test foot est constitué de tweets de fans de football. Les phrases sont donc constituées d'éléments ne pouvant pas se retrouver dans les autres corpus comme des émoticônes. Le second ensemble de test appelé natdis est également un ensemble de tweets traitant cette fois du domaine des catastrophes naturelles. Ces ensembles de tests seront principalement utilisés pour évaluer l'impact du changement de domaines 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 phrase 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 : déterminant
- 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 tous corpus confondus et il est présent dans l'ensemble de test FTB. 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 développement FTB : {'auxquelles', 'duquel', 'desquelles', 'auxquels', 'auquel' }
- DET + NOUN : déterminant 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 : émoticône
— SHARP : # sur twitter
— X : autre

Au delà des étiquettes standard comme Noun ou Verb, il y a quelque étiquettes spéciaesl qu'il convient d'expliquer plus en détails. L'étiquette X est notamment utilisée pour désigner les mots inconnus du corpus ou ceux ayant une orthographe différente du mot habituel. L'étiquette E est utilisée dans les corpus de tweets pour désigner les émoticônes, ces étiquettes vont probablement poser certains problèmes lors du changement de domaines. L'étiquette SHARP est utilisée pour désigner les balises # utilisé pour faire référence à d'autres tweets. Cette étiquette ne devrait également pas être retrouvée dans les autres corpus. L'étiquette URL permet de désigner les liens hypertextes 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 identifié les corpus et les différentes classes que notre algorithme devra prédire. Nous allons maintenant étudier plus en détail l'organisation et la structure des corpus aux moyens de certaines observations statistiques. Nous agrémenterons cette partie de quelques graphes et tableaux pour rendre le tout plus lisible.

Dans un premier temps, nous allons étudier la taille de chacun des corpus en nombre de phrases, nombre de mots et nombre de mots uniques.

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

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 développement d'où son absence

TABLE 2.1 – Récapitulatif de la taille des corpus

Dans le jupyter-notebook, deux autres mesures statistiques ont été effectuées. La première permet de determiner les mots et étiquettes les plus fréquents apparaissant dans chacun des corpus. La seconde permet de déterminer les mots les plus ambigus de chaque corpus. Cette mesure statistique permet de montrer un des problèmes 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 étiquettes différentes. Ce mot sera alors très difficile à discriminer par notre algorithme d'apprentissage et risque de générer des prédictions incorrectes dans un certain nombre de cas.

Out Of Vocabulary words

Nous allons maintenant passer à l'étude des différences entre ensembles d'apprentissage et ensembles de test grâce à la mesure d'Out of Vocabulary words (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 vocabulaire des

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 / développement / foot / natdis

ensembles d'apprentissage et de test. Le cas idéal serait d'avoir un ensemble d'apprentissage ayant le vocabulaire le plus riche possible pour pouvoir faire tendre ce pourcentage d'OOV vers zéro. Le modèle serait alors en mesure de pouvoir prédire plus précisément la classe syntaxique des mots car il aurait appris les meilleurs paramètres pour ces mots, sans pour autant perdre en généralité.

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-delà 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 performances de notre modèle seront plus correctes 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 inconnus vis-à-vis de l'entrainement du modèle sont présents dans l'ensemble de test. On s'attend naturellement à avoir des performances plus faibles 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. Il est probable 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 des phrases, l'agencement des mots et leurs ambiguïtés ne sont pas représentées par cette mesure.

Divergence de KullBack-Leibler

La divergence de KullBack-Leiber tire son origine de la théorie de l'information, dont l'objectif est de déterminer la quantité d'informations 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 minimale, la borne inférieure de bits nécessaires pour encoder la donnée x_i.

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

Maintenant que l'on peut connaître la quantité d'informations contenue dans notre ensemble, nous voulons quantifier la quantité d'informations que l'on perd lorsque l'on passe à des données inconnues. C'est l'objectif de 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'informations en passant de la distribution observée à la distribution paramétrisée et inversement lorsque la métrique diminue. A noter que la divergence de KullBack-Leibler n'est pas symétrique, 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'informations que le classifier n'aura pas réussie à récupérer 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 faibles entre le trainset 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 différence dans le vocabulaire entre le trainset et le testset de Sequoia. La DKL nous indique qu'en plus, il y une grande quantité d'informations différentes entre les deux ensembles, nous nous attendons donc 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'informations sera perdue entre le trainset et le testset : notre modèle donnera probablement de mauvais résultats entre ces deux ensembles.

Pour ce qui est des ensembles de test foot et natdis, les valeurs de divergence sont relativement élevées ce qui n'est pas une surprise étant donné qu'il s'agit d'ensembles traitant de domaines différents de l'ensemble d'apprentissage. On s'attend évidemment à avoir de faibles performances sur ces ensembles.

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 tout simplement PP [1] $= 2^{H(p)}$ où $H(p)$ est l'entropie. Nous avons vu que l'entropie est la quantité d'informations moyenne requise pour coder le résultat d'une variable aléatoire. La perplexité est donc l'exponentiation de l'entropie et correspond au nombre d'éléments 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'informations que l'on peut espérer obtenir via l'apprentissage de notre modèle sur cette distribution de probabilité p(ou l'ensemble d'apprentissage dans notre cas). Une faible valeur de perplexité nous indique que notre modèle pourra prédire

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

correctement des données, à l'inverse une grande perplexité signifie que notre modèle aura du mal à prédire les données, il y aura une plus grande incertitude sur les prédictions.

Pour les calculs de perplexité, nous sommes passé en échelle logarithimique en base e pour éviter l'effet d'underflow dû à de faibles valeurs de probabilité. Nous utilisons par conséquent 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'entraînement de gsd et l'ensemble d'entraînement et test de ftb ont les valeurs les plus grandes de perplexité. On peut expliquer cela grâce aux mots les plus ambigus de chaque corpus. Si l'on regarde le jupyter-notebook sur les mots les plus ambigus de chaque corpus, ces ensembles possèdent les mots les plus ambigus des corpus avec notamment le mot 'A' ayant plus de 6 étiquettes différentes associées dans l'ensemble de train ftb.

Chapitre 3

Classifieurs multi-classe

3.1 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 cet arbre est une suite de tests à effectuer sur une donnée pour ensuite arriver à une prédiction au niveau des feuilles de l'arbre. Contrairement à d'autres algorithmes de machine learning, celui-ci est un algorithme qui permet de comprendre les prédictions effectuées. C'est un algorithme "boîte blanche" contrairement à d'autres qui sont des algorithmes "boîte noir" (dont les exemples les plus connus sont 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. Nous suivons l'algorithme C4.5 qui est une amélioration de l'algorithme ID3.

Hyper-paramètres

Le pseudo-code 1 permet de construire l'arbre de décision. Plusieurs hyper-paramètres permettent de changer le comportement du pseudo-code. Dans

Algorithm 1 Algorithme de contruction de l'arbre de décision

1: **function** BUILDTREE(X, Y, max_depth)
2: **if** $taille(X) < threshold \vee taille(Y) < threshold \vee max\_depth == 0$ **then**
3: **return** $max\{P(c)$ [a] $|\forall c \in Y\}$
4: **else**
5: $bestTest, Xyes, Yyes, Xno, Yno \leftarrow meilleurTest(X, Y)$
6: $left \leftarrow buildTree(Xyes, Yyes, max\_depth - 1)$
7: $right \leftarrow buildTree(Xno, Yno, max\_depth - 1)$
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

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: $yes\_answer \leftarrow yes\_answer \cup \{y | \forall y \in Y, y \leq \overline{f}\}$
10: $no\_answer \leftarrow no\_answer \cup \{y | \forall y \in Y, y > \overline{f}\}$
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**

3.2 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 en partie ses mauvais résultats.

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 :

$$P(C \mid x) = \frac{P(x \mid C) \cdot P(C)}{P(x)} \tag{3.1}$$

$P(C \mid x)$ 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 :

$$P(x \mid C) \cdot P(C) \tag{3.2}$$

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 :

$$P(x \mid C) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)} \tag{3.3}$$

Avec μ la moyenne et σ l'écart type, tous deux calculés plus tôt.

Résultats

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.

Idées de développement

Les résultats de notre classifieur naïf bayésien sont très mauvais. Un des problèmes 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.

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.

Plus tard

Dans notre projet, nous utilisons la méthode de la validation croisée. Elle consiste à entraîner notre modèle sur une instance d'hyper-paramètres donnée grâce à l'ensemble puis à tester notre modéle 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 gagne alors en précision spécifiquement sur l'ensemble d'apprentissage, mais perd en généralité : testé sur d'autres ensembles de données, les résultats seront mauvais.

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'ensembles de développement, mais il est toujours préférable de tester une dernière fois notre modèle sur d'autre jeu de données inconnue.