PROJET DE SEMESTRE

IPRoduct : Product Name Identification

Jules COURTOIS

September 8, 2017

Superviseur : Dr. David PORTABELLA

Professeurs : Prof. Jean-Cédric CHAPPELIER, Prof. Gaétan DE RASSENFOSSE

Printemps 2017

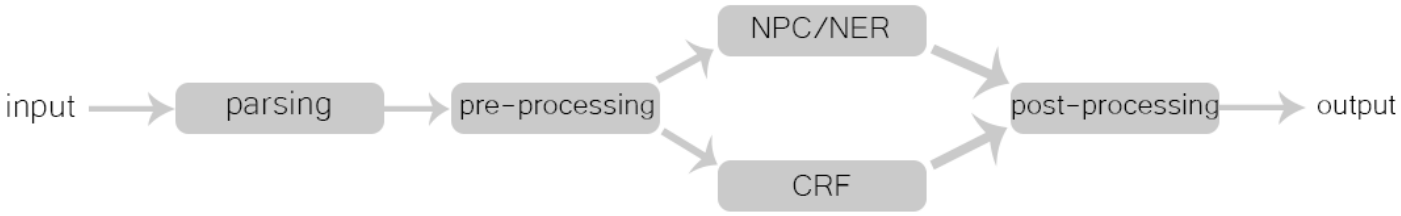
EPFL

ABSTRACT

Dans le cadre d'application du projet ***IPR*oduct**, nous souhaitons reconnaître la liste des noms de produits mentionnés sur des pages anglaises de Virtual-Patent Marking (VPM). Pour ce faire, nous utiliserons des techniques de traitement automatique du language naturel (NLP) ainsi que des méthodes probabilistes tels les Conditional Random Fields (CRF).

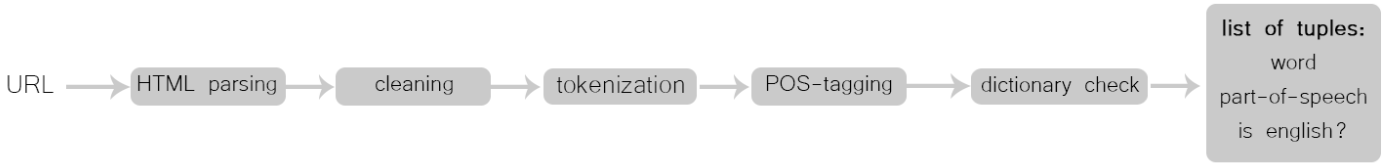
Les pages de VPM contiennent une liste de produits ainsi que leurs numéros de brevets associés. De telles pages suivent en général une structure simple et limitée, avec peu de texte extérieur à celui recherché. Par exemple : "[product-name] is covered by the following patents: [patents-numbers]".

Le projet nécessitera l'exploration et la compréhension de multiples outils tels NLTK ou Wapiti, ainsi que l'acquistion de nouvelles compétences comme la compréhension et amélioration du Part-of-Speech tagging, la création de corpora d'entraînement, réalisation de "feature engineering" pour les CRF.



CONTENTS

1 Extraction et Prétraitement



Le premier aspect de la résolution du problème présenté est le format d'entrée des données : une liste d'URLs. Trivialement, on peut transformer cette liste en une séquence d'URL uniques, à partir desquels on peut facilement visualiser des pages web à l'aide d'un browser, ou les télécharger. Une page téléchargée depuis son URL n'est pas dans un language naturel : elle présente du code HTML sous forme de balises, ainsi que souvent du JavaScript et du CSS. De plus, en situation réelle, tout le texte d'une page ne nous est pas utile. Au contraire : bien que l'on puisse trouver des noms de produits par exemple dans un menu déroulant, celui-ci ne nous intéresse pas et risque de desservir le modèle obtenu par entraînement. Ainsi, il est nécessaire de trouver une manière efficace de parser les données obtenues avant d'entamer les méthodes de NLP.

Le language utilisé pour la plupart du projet sera du Python 3, choisi pour sa simplicité d'écriture, la grand nombre de librairies et outils existants (dont BeautifulSoup, NLTK, Wapiti que nous verrons plus tard) ainsi que son support pour les expressions régulières et tous les avantages typiques d'un language de script.

1.1 Extraction de la Page et BS4

BeautifulSoup 4 (BS4) est une librairie Python permettant d'extraire des données HTML ou XML de manière extrêmement simplifiée. Nous nous en servirons pour transformer notre URL en un document HTML complet. Nous sommes confrontés à plusieurs problèmes : tout d'abord, le document contient encore des balises de scripts ou de style, des syntaxes qui ne nous sont d'aucune utilité et ne ressemblent en rien à un language naturel. Nous pouvons les supprimer entièrement en utilisant BS4 ainsi qu'une autre librairie, *html2text*..

Ensuite, beaucoup de parties de la page web contiennent des noms propres ou de produits bien que ceux-ci ne soient pas associés à des numéros de brevet. Par exemple, on peut trouver dans un menu déroulant global au site des liens vers les pages de chaque produit. Comme vu précédemment, ceci risque d'être contre-productif. En plus d'être source de plus de temps de calcul, ce phénomène pourra donner des résultats contradictoires sur quels mots définir comme noms de produits marqués. Pour résoudre cela, on remarque que les informations recherchées sont presque toujours dans le 'corps' de la page (c'est à dire entre les balises <body> et </body>). BS4 implémente facilement l'extraction de contenu entre deux balises, ce problème est donc en grande partie résolu. Il reste tout de même des difficultés avec les pages mal formatées, ou avec le contenu intéressant en dehors du body. Afin de les reconnaître, on peut rechercher la présence de numéros de brevets dans le <body>, et si celle-ci est nulle ou

inférieure à un certain seuil, considérer la page entière.

Finalement, on utilise des expressions régulières pour supprimer les derniers résidus qui n'ont pas encore été attrapés par BS4. Nous avons désormais réussi la première étape dans la pipeline des données : convertir un simple URL en un texte dans un format similaire à du language naturel, tout en traitant et éliminant un maximum des données qui ne nous seront jamais d'aucune utilité.

1.2 TOKENIZATION

La tokenization est la conversion d'un string en une liste de sous-strings censés représenter des "mots" en language naturel. L'extraction de texte depuis une page implique des erreurs de formattage. Par exemple, un saut de ligne peut être vu sans espace, et un texte de la forme "patent[saut de ligne]number" serait extrait en tant que "patentnumber". La vraie difficulté est que cela n'a pas lieu dans tout les cas, même très similaires, selon les choix d'implémentation du webmaster. Une solution assez fructueuse est d'utiliser comme séparateur dans BS4 le string : " . " (espace point espace) puis de supprimer les doublons. Cela nous permet de mettre des "fins de phrase" là où sont donnés en HTML des fins de pargraphes, de sections, etc...

Une approche naïve à la tokenization serait de définir le caractère 'space' comme séparateur de tokens. Ceci révèle plusieurs problèmes. Prenons comme exemple la phrase : "Barack H. Obama was an american President" Une tokenization naïve résultera avec le token 'President.' ce qui n'est pas correct. En effet, le '.' de fin de phrase devrait être considéré comme son propre token. On peut donc supposer utiliser chaque signe de ponctuation comme un séparateur, mais cela donnerait le token 'H' qui n'est désormais plus correct, on voudrait 'H.'

La tokenization est un problème complexe pour lequel il existe déjà de nombreuses grammaires et algorithmes bien plus efficaces que celles que nous aurions pu créer dans le cadre de ce projet. Nous avons donc décidé d'utiliser la fonction **word_tokenize()** de NLTK, par sa facilité d'utilisation et sa compatibilité avec d'autres fonctions de NLTK que nous utiliserons plus tard. Le word tokenizer de NLTK utilise une grammaire d'expressions régulières.

Nous avons désormais atteint la troisième étape dans la pipeline : une liste de chaque "mot" dans le texte de la page web, sur laquelle nous allons désormais pouvoir appliquer des algorithmes probabilistes entrainés pour différentes utilisations.

1.3 PART-OF-SPEECH TAGGING

L'étiquetage morpho-syntaxique (POS-tagging) est un procédé ayant pour but d'assigner à chaque mot d'une phrase une étiquette désignant son rôle syntaxique (nom commun, nom propre, verbe, déterminant...)

En pratique, nous utiliserons NLTK pour le tagging. Il existe dans NLTK une fonction "black-box" dénotée **pos_tag()**. Afin de pouvoir comprendre, voir les limites et améliorer ce processus nous allons regarder son fonctionnement.

Cette fonction utilise un "Perceptron Tagger", une méthode probabiliste qui consiste à utiliser un modèle pré-entrainé sur une liste de features. Un feature est une fonction booléenne à laquelle est associée un poids positif, nul ou négatif pour chaque sortie possible. Les sorties possibles sont chacune des étiquettes selon la norme 'Penn Treebank', visible ici. La liste des features utilisés par le Perceptron Tagger est la suivante :

```
'bias' : feature constant
'i suffix' : les 3 derniers caractères du token
'i pref1' : le 1er caractère du token
'i-1 tag' : le tag du token précédent
'i-2 tag': le tag du token 2 positions avant
'i-1tag+i-2 tag' : le couple des deux tags précédents
'i word' : le token lui-même
'i-1 tag+i word' : le couple du token et du tag précédent
'i-1 word' : le token précédent
'i-1 suffix' : 3 derniers caractères du token précédent
'i-2 word' : le token 2 positions avant
'i+1 word' : le token suivant
'i+1 suffix' : les 3 derniers caractères du token suivant
'i+2 word' : le token deux positions après
```

Le tagger additionne ensuite les poids de chaque feature pour chaque sortie possible, et sélectionne la sortie avec le score le plus grand. Nous allons désormais critiquer son fonctionnement par des exemples :
"dog" : le mot "dog" n'a qu'un seul role dans le dictionaire anglais, celui d'un nom commun. Ainsi, le feature 'i word' associera à la sortie "nom commun" un poids extrêmement élevé, ce qui assurera que "dog" soit finalement taggé "nom commun". Ainsi, là où le modèle à le plus d'utilité est lorsqu'un mot peut avoir plusieurs roles ou lorsqu'il est inconnu.
Par exemple, pour la phrase "This is the Logitech", le tagger commencera tout d'abord par réaliser les observations suivantes :

```
 'bias' (4443319352)
 'i pref1 L' (4529159344)
 'i suffix ech' (4529158960)
 'i tag+i-2 tag DT VBZ' (4528973480)
 'i word logitech' (4529159216)
 'i+1 suffix ND-' (4529159792)
 'i+1 word -END-' (4529159728)
 'i+2 word -END2-' (4529159856)
 'i-1 suffix the' (4529159600)
 'i-1 tag DT' (4529159280)
 'i-1 tag+i word DT logitech' (4528967968)
 'i-1 word the' (4529159408)
 'i-2 tag VBZ' (4529159024)
 'i-2 word is' (4529159664)
```

Desquelles il extraiera, entre autres, les poids suivants (on ne montre que les tags qui nous intéressent, NN et NNP) :

```
weights['i suffix ech']
 'NN' (4507579592) = {float} 1.719
 'NNP' (4498679712) = {float} -1.162

weights['i-2 word is']
 'NN' (4498678480) = {float} -0.06
 'NNP' (4502981408) = {float} -0.748
```

Ainsi, on comprend pourquoi Logitech est taggé comme un nom commun dans cette phrase au lieu d'un nom propre comme il devrait l'être. En revance, dans la phrase "He talks the Logitech", so only changing "This is" into "He talks", 'Logitech' is now correctly tagged as a proper noun. There only different Ceci nous montre qu'il serait important d'améliorer le tagging à l'avenir, car il est imparfait.

Finalement, nous avons également créé notre propre extension spécialisée au tagger par défaut de NLTK, qui est utilisée pour reconnaître les numéros de brevets (et les tagger avec 'PATENT') ainsi que les symboles spéciaux tels ®, ©ou ™et de les tagger avec 'SIGN'. Ceci nous permet de détecter des parties de texte intéressantes tôt dans la pipeline.

1.4 Dictionary Check

Une dernière information est la présence de chaque token dans le dictionaire anglais. En effet, les noms de produits ou de sociétés sont souvent inventés de toute pièce ou un mélange de plusieurs mots. Ainsi, l'absence d'un mot dans le dictionaire pourrait être un indicateur intéressant.

Il existe quelques subtilités : d'abord, un même mot peut avoir plusieurs formes. Ensuite, comment gérer les nombres, ponctuations, etc ?

Conernant le premier défi, le plus flagrant sont les verbes. Un même verbe peut être conjugué à plusieurs temps et à plusieurs personnes. De plus, la plupart des noms ont une forme au pluriel différente de la forme singulière. Pour attraper chacune de ces possibilités, nous avons décidé d'utiliser trois corpora différents : *PyEnchant, NLTK.words* et *WordNet*. Après plusieurs tests sur différents ensembles d'essai, cet assemblement de trois a semblé être le plus complémentaire et large.
De plus, cet ensemble gère également les chiffres et ponctuation écris de manière naturelle, ce qui sera utile pour les numéros de brevet.

1.5 Sortie

On a désormais réussi à transformer notre URL en une liste de 3-tuple : {token, tag, appartenance to english dictionary (I in or O out)}. Afin de pouvoir utiliser ce résultat dans les parties suivantes, nous allons le stocker dans un fichier texte (encodé en UTF-8) avec un format spécifique : chaque ligne contiendra chacun des membres du 3-tuple dans l'ordre précisé ci-dessus, séparés par un espace. De plus, on rajoute un dernier membre au tuple dont nous discuterons plus tard. Exemple :

```
Barack      NNP   O   #
Obama       NNP   O   #
...
President   NN    I   I
```

Ce format sera particulièrement pratique pour utiliser Wapiti, plus tard dans la pipeline des données.

2 NP-CHUNKING ET NAMED ENTITY RECOGNITION

Maintenant que nous avons séparé le texte en plusieurs tokens et associé à chacun des étiquettes morpho-syntaxiques, nous pouvons commencer à rechercher les noms de produits. Dans un contexte de language naturel, un produit sera toujours compris dans, ou égal, à une phrase nominale. L'objectif est désormais d'extraire les phrases nominales du texte, puis de les filtrer à la recherche de produits.

2.1 NLTK NP-CHUNKING

Rappel:
- F-mesure = 2*precision*recall/(precision + recall)

Actuellement, l'état de l'art des systèmes de NP-Chunking atteint jusqu'à 95% de F-mesure. NLTK offre plusieurs options de chunkers, les plus intéressantes étant le RegexpChunker et le UnigramChunker.

Le RegexpChunker permet d'utiliser des expressions régulières sur des séquences d'étiquettes afin de reconnaître des phrases nominales. Par exemple, prenons la phrase "He saw the yellow dog." On a deux phrases nominales : l'une composée seulement d'un pronom ("He"), la seconde d'un déterminant, un adjectif puis un nom commun ("the yellow dog"). Un RegexpChunker fonctionnel dans ce cas là devrait détecter "PR" mais aussi "DT/JJ/NN". Cette solution peut sembler optimale si on la travaille assez, mais en situation réelle il faut prendre en compte que la syntaxe des phrases étudiées n'est pas parfaite (surtout pas dans notre cas), que l'étiquettage n'est pas optimal et que finalement qu'il existe beaucoup de subtilités et d'exceptions dans le language qui faussent les résultats.

L'UnigramChunker utilise un modèle probabiliste et un training set pour générer des fonctions capables de détecter les phrases nominales. NLTK offre plusieurs corpora en complément de ses algorithmes sur lesquels il est facile d'entrainer un UnigramChunker. Encore une fois, malgré la quantité de données sur lesquelles s'entrainer, la difficulté vient du language spécifique des pages que nous analysons ainsi que leur manque de syntaxe correcte.

Une fois les phrases nominales extraites via l'un ou l'autre des deux chunkers, il faut encore choisir lesquelles représentent un nom de produit. L'idée la plus simple est d'éliminer toute phrase nominale ne contenant que des mots présents dans le dictionnaire anglais. Les résultats au mot-par-mot sont d'environs 8% de précision et 2% de recall dans les meilleurs cas, en utilisant le corpus d'entrainement original sur un UnigramChunker et en évaluant sur la page de 'TiVo'. La plupart des RegexpChunkers par défaut sont encore moins efficace à cause de la syntaxe erratique des pages étudiées. Bien qu'il soit facile d'augmenter la taille du corpus ou

d'en créer un plus adapté, c'est une solution coûteuse en temps. De plus, il sera toujours très difficile de filtrer ce qui représente un nom de produit de ce qui représente autre chose. Nous allons donc nous tourner vers l'état-de-l'art en matière d'extraction de phrases nominales.

2.2 SpaCy NP-Chunking and Named Entity Recognition

"SpaCy's mission is to make cutting-edge NLP practical and commonly available."

SpaCy est un logiciel récent offrant un système complet de NLP extrêmement simplifié, en Python. Bien qu'il ne soit pas le logiciel le plus précis actuellement disponible, les auteurs affirment qu'il est en moyenne 20 fois plus rapide que n'importe quel logiciel plus précis. Source : https://spacy.io/docs/api/

SpaCy offre des outils pré-entrainés de NPC et de NER, mais qui peuvent être modifiés à souhait. Un test simple des paramètres par défaut donne les résultats suivant (en détectant la précision mot-par-mot ou produit-par-produit) :

Table 2.1: SpaCy Evaluations

Implementation	Np-Chunking	NER
Runtime (s)	53.00	60.00
Phrases Prec	0.023	0.080
Phrases Rec	0.042	0.081
Words Prec	0.090	0.137
Words Rec	0.136	0.074

L'évaluation a été faite sur le corpus initial pour l'entrainement, puis en évaluant la page "TiVo". Le pourcentage indique la correspondance mot-par-mot, il est donc encore moins précis qu'une correspondance exacte. On remarque des résultats légèrement supérieurs à ceux par défaut de NLTK, mais pas conséquents pour autant. Il est important de noter que SpaCy utilise également un PerceptronTagger. Lors du projet, l'outil était encore assez mal documenté car très récent et par conséquent difficile à comprendre et utiliser, en revanche la documentation a beaucoup évolué depuis, et SpaCy semble être une option intéressante à ré-étudier dans le futur. Parmis les foncitonnalités disponibles, on peut trouver :

- Custom tokenization
- Visualizers
- Custom-format text processing
- PoS/NER training

En particulier, il pourrait être intéressant d'utiliser SpaCy pour le pre-processing du texte extrait de l'HTML au lieu de, ou en conjonction avec, NLTK.

3 Conditional Random Fields

3.1 Introduction

"Les champs aléatoires conditionnels (conditional random fields ou CRFs) sont une classe de modèles statistiques utilisés en reconnaissance des formes et plus généralement en apprentissage statistique." - Wikipedia

Les CRFs sont utilisés pour modéliser des probabilités dans un contexte déjà connu grâce à des observations. Il existe de nombreux outils différents qui permettent de mettre en place un modèle CRF, par exemple :

- Wapiti
- CRFSuite
- CRF++
- Mallet
- Factorie

Cette source $http://fnl.es/a-review-of-sparse-sequence-taggers.html$ résumant bien les différences entre les outils et les points forts de chacun, en plus d'études personelles sur la documentation et de tests de chaque outil, nous a permis de choisir celui qui semblait le plus approprié : Wapiti. Il est également important de noter l'existence de WebStruct, un autre logiciel intéressant pour le futur basé sur Wapiti, et qui permet l'utilisation de CRFs directement sur des pages web.

3.2 Wapiti Patterns and Features

L'interaction principale entre l'utilisateur et Wapiti est l'utilisation de fichiers "patterns", grâce auxquels Wapiti pourra extraire des "feature functions" (FF) du corpus d'entraînement. Comprendre le fonctionnement exact de ces patterns est la première étape vers la création d'un modèle compétent.

Exemple du fichiers patterns utilisé pour la tâche partagée de ConLL_2000, auquel nous avons ajouté une simple ligne pour notre troisième colonne:

```
*

U:Wrd-1 X=%x[ 0,0]

U:wrd-1LL=%X[-2,0]
U:wrd-1 L=%X[-1,0]
U:wrd-1 X=%X[ 0,0]
U:wrd-1 R=%X[ 1,0]
U:wrd-1RR=%X[ 2,0]

U:wrd-2 L=%X[-1,0]/%X[ 0,0]
U:wrd-2 R=%X[ 0,0]/%X[ 1,0]

*:Pos-1LL=%x[-2,1]
*:Pos-1 L=%x[-1,1]
*:Pos-1 X=%x[ 0,1]
*:Pos-1 R=%x[ 1,1]
*:Pos-1RR=%x[ 2,1]

U:Pos-2 L=%X[-1,1]/%X[ 0,1]
U:Pos-2 R=%X[ 0,1]/%X[ 1,1]

*:Pre-1 X=%m[ 0,0,"^.?"]
*:Pre-2 X=%m[ 0,0,"^.?.?"]
*:Pre-3 X=%m[ 0,0,"^.?.?.?"]
*:Pre-4 X=%m[ 0,0,"^.?.?.?.?"]

*:Suf-1 X=%m[ 0,0,".?$"]
*:Suf-2 X=%m[ 0,0,".?.?$"]
*:Suf-3 X=%m[ 0,0,".?.?.?$"]
*:Suf-4 X=%m[ 0,0,".?.?.?.?$"]

*:Caps? L=%t[-1,0,"\u"]
*:Caps? X=%t[ 0,0,"\u"]
*:Caps? R=%t[ 1,0,"\u"]

*:AllC? X=%t[ 0,0,"^\u*$"]
*:BegC? X=%t[ 0,0,"^\u"]

*:Punc? X=%t[ 0,0,"\p"]

*:Dict? X=%x[ 0,2]
```

Les fichiers patterns suivent une structure bien précise : chaque ligne correspond à une règle d'extraction d'une FF. Les lignes blanches ou précédées d'un caractère '#' sont ignorées.

Chaque ligne possède plusieurs éléments qui composent le pattern :

- Un premier caractère, qui peut être 'U' pour unigramme, 'B' pour bigramme, ou '*' pour les deux. Ce caractère représente la portée des fonctions extraites. Définissons |L| comme la taille de l'ensemble des étiquettes, et |F| comme le nombre de valeurs possibles qui peuvent être extraites du pattern. Ainsi, un pattern 'U' produira au maximum |L|*|F| feature functions, soit une fonction par valeur possible par étiquette possible. Un pattern 'B' associe également la valeur à l'étiquette précédente, donc extraiera au maximum |L|*|L|*|F| feature functions. Finalement, un pattern '*' extraiera au maximum |L|*|F| + |L|*|L|*|F| feature functions, et est équivalent à écrire deux fois le même pattern avec un 'U' et un 'B'.

- Un second string, optionnel, qui débute par ':' et qui permet de définir un identificateur aux FF extraites. Cet identificateur a plusieurs utilités : tout d'abord, il permet d'avoir plusieurs patterns différent aux environnements s'intersectants. Ensuite, il permet également de plus facilement comprendre le modèle à l'oeil humain. Finalement, cette fonction de Wapiti est également une manière d'implémenter des "Bag-of-words features", c'est à dire des features qui ne prennent pas en compte l'ordre relatif ou absolu des mots mais seulement leur ensemble et leur multiplicité. Prenons un exemple :

  ```
  U:my-bag=%x[0,0]
  U:my-bag=%x[1,0]
  U:my-bag=%x[2,0]
  ```

  Avec un fichier de patterns ainsi, deux séquences de la forme :

Hello	I
am	am
I	Hello

  Génèreront la même feature function, du type my-bag = {Am, I, Hello}

- Un indicateur du type d'information à extraire, précédé d'un '='. L'indicateur peut prendre l'une des 6 valeurs : '%x', '%t', '%m', '%X', '%T' ou '%M'. En majuscule, on applique la même opération qu'en minuscule mais en ignorant la casse. Attention, la casse est ignorée en mettant l'information entière en majuscule ! Pour les indicateurs même :

  - %x : prends deux arguments, de la forme [off, col] avec off signifiant la ligne d'offset 'off' à partir du mot étudié (peut être négatif, positif ou nul), et 'col' signifiant la colonne des données à regarder, en commençant à 0. %x extrait le mot entier comme information, par exemple :

Barack	NNP	I
Obama	NNP	I
was	VBZ	O
President	NN	O

Lorsque la prédiction aura lieu sur le mot "Obama", les transformations suivantes se feront :

```
x[ 0,0] => Obama
x[-1,0] => Barack
x[ 1,1] => VBZ
```

– %t : signifie "regexp test", prend cette fois trois arguments: [off,col,"re"] avec 'off' et 'col' comme vus précédemment, et 're' une expression régulière compréhensible pour Wapiti. Si les données en [off,col] correspondent à l'expression régulière, les données seront transformées en "true". Sinon, en "false". Par exemple, encore une fois appliqué sur le mot "Obama" :

```
%t[0,0,"\^O"] => true
%t[1,0,"\^O"] => false
```

– %m : signifie "regexp match", avec les mêmes arguments que '%t'. Cette fois, si les données en [off,col] correspondent à l'expression régulière, les données seront transformées en la partie du string qui a été reconnue. Par exemple, encore une fois appliqué sur le mot "Obama" :

```
%t[0,0,"\^Obama"] => Obama
%t[1,0,"\^Obama"] =>  (string vide)
```

- Finalement, plusieurs clauses peuvent être ajoutées sans limites de nombre et avec n'importe quels schémas de caractères, afin de créer des observations composées. Par exemple :

```
%t[off1,col1,"re1"]+%t[off1,col1,"re2"] => true+false
%t[off1,col1,"re1"]%t[off1,col1,"re2"]  => truefalse
%x[0,0] %t[off1,col1,"re1"] => Obama true
```

Nous savons désormais créer des patterns pour Wapiti. Le reste du travail consistera donc à trouver ces-dits patterns, les évaluer, et les perfectionner.

3.3 Graphes des Poids

Une manière intéressante de visualiser un modèle Wapiti est d'ordonner ses poids puis de les plotter afin d'en faire une courbe.

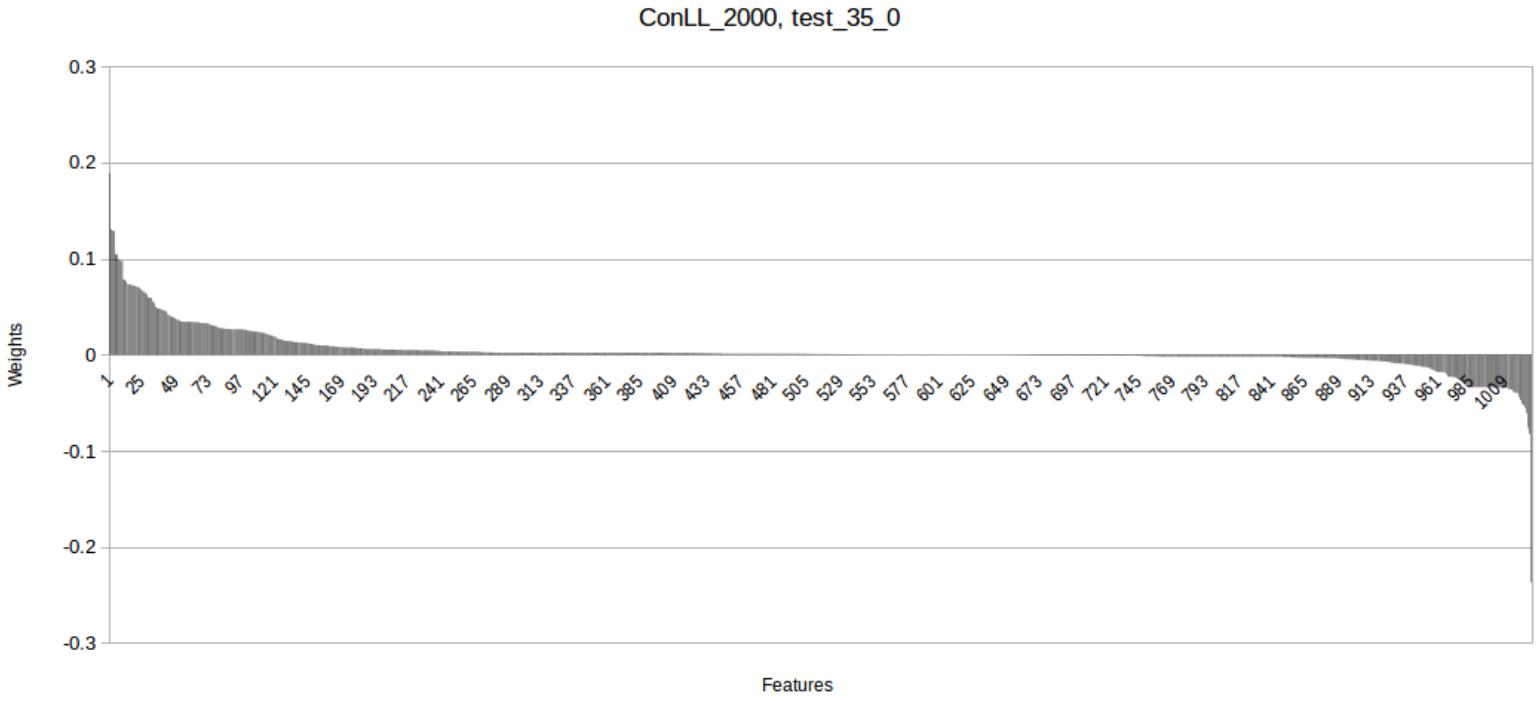
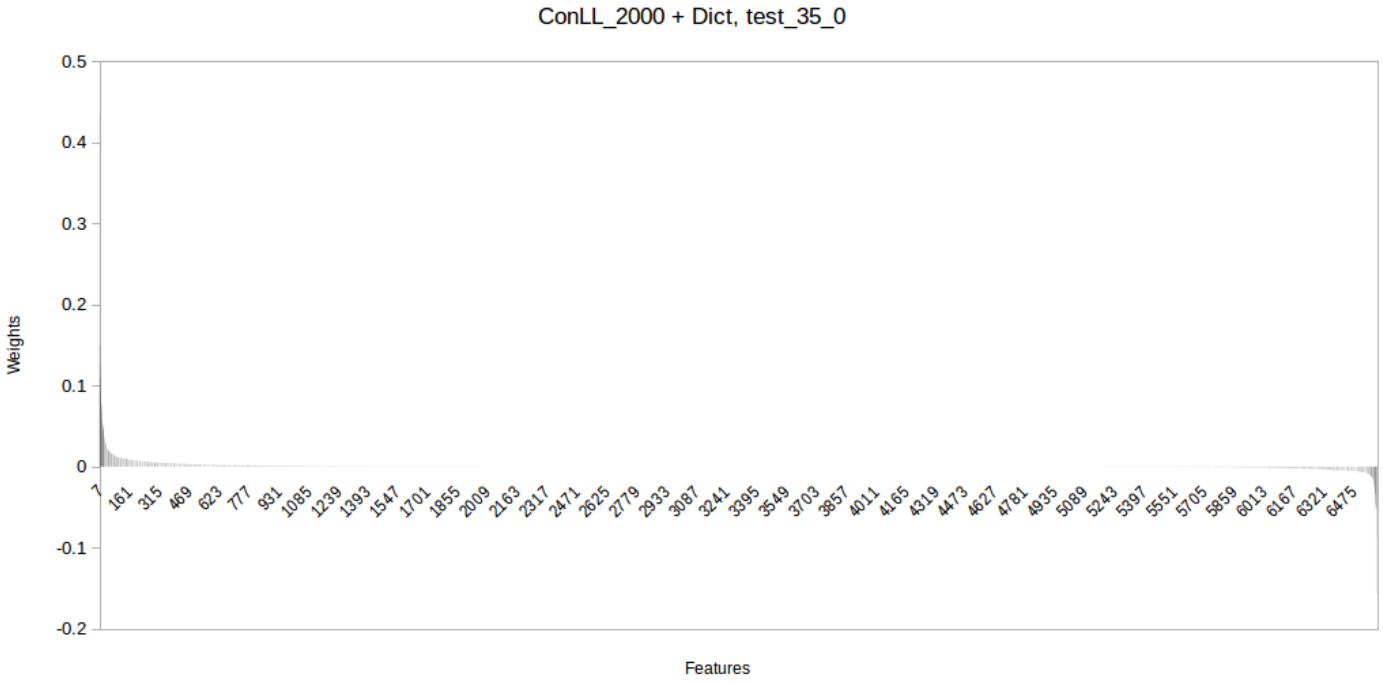
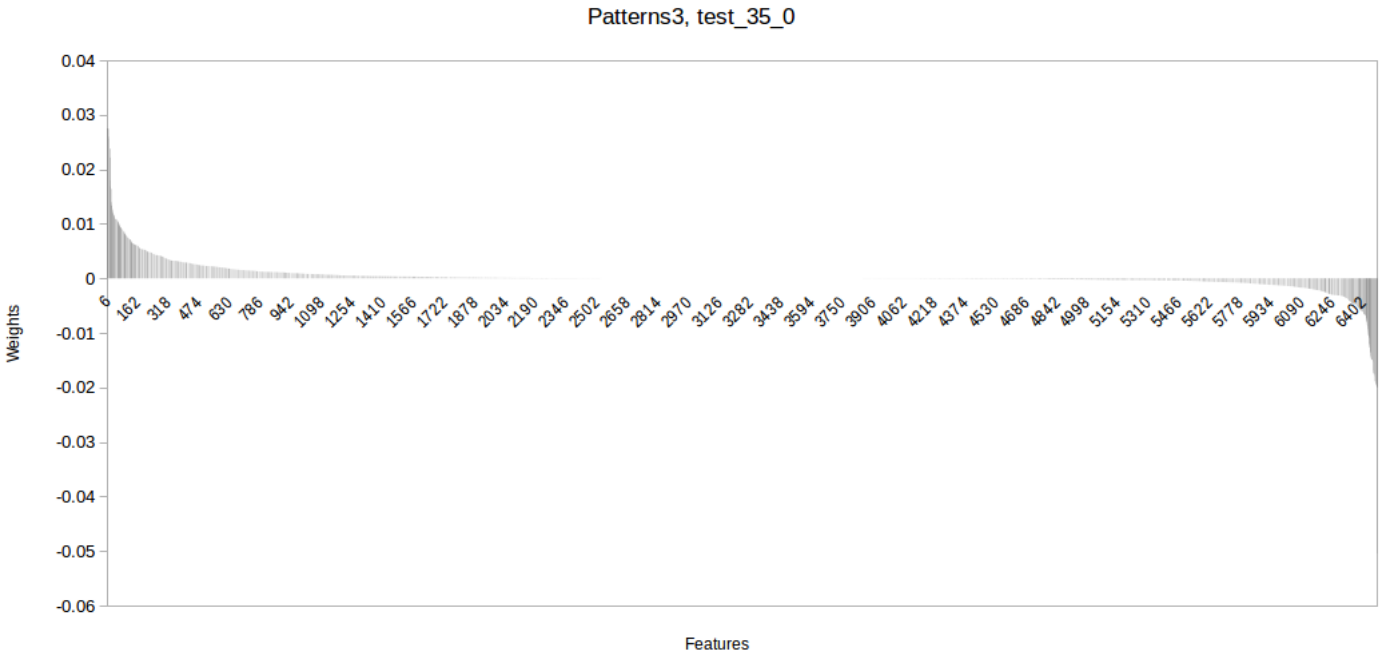


Table 3.1: F-values for test_35_0

Label	X	B	I	P
ConLL	0.9679	0.2519	0.1244	**0.9715**
ConLL + Dict	**0.9779**	0.2247	0.0847	0.9517
Patterns3	0.9713	**0.3408**	**0.2427**	0.8198


ConLL_2000 + Dict, test_35_0
Weights
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
7
161
315
469
623
777
931
1085
1239
1393
1547
1701
1855
2009
2163
2317
2471
2625
2779
2933
3087
3241
3395
3549
3703
3857
4011
4165
4319
4473
4627
4781
4935
5089
5243
5397
5551
5705
5859
6013
6167
6321
6475
Features


Patterns3, test_35_0
Weights
0.04
0.03
0.02
0.01
0
-0.01
-0.02
-0.03
-0.04
-0.05
-0.06
6
162
318
474
630
786
942
1098
1254
1410
1566
1722
1878
2034
2190
2346
2502
2658
2814
2970
3126
3282
3438
3594
3750
3906
4062
4218
4374
4530
4686
4842
4998
5154
5310
5466
5622
5778
5934
6090
6246
6402
Features

Note : le test_35_0 est une instance aléatoire d'une division du corpus de 75 textes en 40 textes d'entrainement, et 35 textes d'évaluation.

Nous pouvons tirer plusieurs informations qualitatives de ces graphes. Tout d'abord, la largeur du "plat" autour de 0 indique la proportion de features avec très peu d'impact. Ensuite, nous pouvons observer la forme des deux extrémités, qui représentent les features avec beaucoup d'impact positif ou négatif sur une étiquette. Un modèle parfait aurait très peu de features inutiles, pour améliorer la performance, quelques features très utiles des côtés positifs et négatifs, et un changement abrupt d'un côté à l'autre. Ainsi, on pourrait penser que le premier graphe, qui représente les features selon la norme ConLL_2000, proposerait les meilleurs résultats. En revanche, il est important de pondérer car Patterns3, qui est beaucoup plus spécialisé dans la détection de numéros de brevet, trouve de meilleurs résultats. Il est important de ne pas confondre des valeurs extrêmes différentes à un changement important de la forme du graphe, parfois quelques très bons features changent radicalement le tout comme dans l'exemple de ConLL_2000 avec ou sans dictionnaire.

Le "feature engineering" est le processus le plus important pour l'instant, et celui pour lequel le plus d'ameliorations sont encore sûrement possible. L'une des plus grandes faiblesses de Wapiti est le manque de flexibilité de ses expressions régulières. Par exemple, nous souhaitons détecter la sous-phrase "US Patent Numbers", mais aussi "EU Patent Numbers" sans doubler le poids de "Patent numbers" artificiellement. Pour ce faire, nous devons créer un quadrinome d'expressions régulières qui génère $2^4 = 16$ features de la forme true/false/true/true (EU, US, Patent, Number). Si Wapiti permettait l'utilisation de l'opérateur "OR" dans une expression régulière, le nombre de features serait divisé par deux. Au premier abord, cela peut sembler peu nécessaire, mais si nous souhaitons traduire une expression de la forme : ('EU' OR 'US' OR 'UK')/'patent'/('number' OR 'no' OR '#'), on pourrait diviser le nombre de features par 16. De plus, les patterns seraient beaucoup plus facile à écrire et permettraient aussi la création de plus de features intéressant avec par exemple la possibilité de détecter de numéros de brevets directement par Wapiti. Il serait envisageable de modifier le code source (disponible sur github) pour améliorer sa compréhension d'expressions régulières, même si cela n'a pas été abordé durant le projet.

Exemple d'équivalence :

```
%x[1,0,"(EU) OR (US)"]/%x[2,0,"patent"]/%x[3,0,"number"] =>
    %x[1,0,"EU"]/\%x[1,0,"US"]/%x[2,0,"patent"]/%x[3,0,"number"]
```

Cette équivalence n'est vraie si et seulement si les ensembles sont parfaitement disjoints. Ainsi dans le cas de EU et US, vu que le string est différent, c'est le cas. En revanche, pour des expressions régulières plus flexibles, il faudra le garder en mémoire.

Ci-joint, des tableaux de comparaisons de plusieurs fichiers de patterns aux fonctions et apparences différentes. Pour chaque ligne, le modèle est entrainé sur 40 documents du corpus "wp_corpus" puis évalué sur les 35 autres, au hasard. Chaque cellule contient la F-value pour l'étiquette correspondant à la colonne.

Table 3.2: Random results on Patterns 3

Label	X	B	I	P
1	0.9640	0.5460	0.3963	0.9162
2	0.9366	0.2953	0.1807	0.7086
3	**0.9869**	**0.6348**	0.6319	0.8853
4	0.9685	0.2656	0.1352	0.8749
5	0.9722	0.6000	**0.6390**	0.8874
6	0.9752	0.3418	0.1222	**0.9307**
7	0.9622	0.3594	0.3577	0.9058
Total	0.9665	0.4347	0.3519	0.8727

Table 3.3: Random results on Patterns 4

Label	X	B	I	P
1	**0.9863**	0.3655	0.3989	0.9720
2	0.9700	0.5305	0.5200	0.9681
3	0.9714	**0.5975**	0.5333	**0.9823**
4	0.9766	0.5342	0.4592	0.8856
5	0.9656	0.5838	0.4448	0.9608
6	0.9676	0.5405	0.5039	0.9839
7	0.9734	0.5424	**0.5930**	0.9504
Total	0.9633	0.5278	0.4933	0.9576

Table 3.4: Random results on Patterns 5

Label	X	B	I	P
1	0.9623	0.4208	0.531	0.9772
2	0.9646	0.3718	0.3337	0.9807
3	0.9748	0.3144	0.2604	0.9305
4	0.9324	0.2498	0.3443	0.9275
5	0.9844	0.3816	0.3924	0.9637
6	**0.9879**	**0.6868**	**0.7123**	**0.9951**
7	0.9352	0.4311	0.4149	0.9687
Total	0.9631	0.4080	0.427	0.9633

Table 3.5: Random results on Patterns 6

Label	X	B	I	P
1	0.9822	0.6281	0.6244	0.9964
2	0.9790	0.6474	0.5385	0.9954
3	0.9762	0.5029	0.5238	0.9910
4	0.9850	0.4127	0.3463	**0.9970**
5	**0.9855**	**0.7447**	**0.8020**	**0.9970**
6	0.9833	0.7212	0.7368	0.9952
7	0.9802	0.6748	0.6934	0.9967
Total	0.9816	0.6188	0.6093	0.9955

Chaque ligne représente une instance du test 'WP40_35', qui consiste à diviser le corpus de 75 pages VPM en 40 pages d'entrainement et 35 pages d'évaluation.

On peut voir que les résultats peuvent être très épars, mais sont plutôt bons. Patterns3 représente des schémas assez simples, et peu de règles générales, Patterns4 très peu de schémas précis mais des règles générales poussées (copiées de ConLL), Patterns5 des schémas imples et des règles générales simples, et finalement Patterns6 les schémas les plus développés que nous ayons pu créer durant le projet, ainsi que les règles générales de ConLL. On remarque que Patterns6 est le plus efficace, ce qui tend donc à penser qu'avoir de nombreux patterns spécifique est un avantage dans notre cas.

3.4 Données et Formats

La seconde interaction principale que l'utilisateur peut avoir avec Wapiti est la forme des données d'entrées. L'un des features qui nous est particulièrement utile est le "forcing", qui permet de définir des contraintes sur les résultats. En rajoutant une colonne d'entrée à Wapiti, il est possible de forcer n'importe quel mot à être étiqueté d'une certaine manière. Cette fonction nécessite une connaissance préalable : l'ensemble des étiquettes possibles présentes dans le modèle. Ensuite, il suffit de rajouter une colonne au fichier à étiqueter avec un symbole inconnu pour les mots nécéssitant un étiquettage, ou un symbole connu pour forcer une contrainte. Voici un exemple, en utilisant un modèle contenant l'ensemble {X, P, B, I} :

protected	VBZ	#
by	PP	#
patent	NN	#
number	NN	#
D123,456	NNP	P

Dans cet exemple, Wapiti n'étiquettera seulement les lignes avec un '#' comme dernier caractère. Ici, 'D123,456' sera automatiquement étiqueté P. Grâce à cette fonction, nous sommes capable de détecter par CRF une partie des numéros de brevet, ce qui améliore non seulement énormément les résultats dans le cas des patterns, mais également qui permet d'augmenter la précision des autres étiquettes. Il n'y a pas de point négatif visible à utiliser cette méthode.

L'autre impact que l'utilisateur peut avoir est sur le format et l'importance des données d'entrée et de la méthode d'étiquetage. Nous avons testé plusieurs méthodes d'entrée, qui sont reportées dans le tableau suivant. Par exemple, l'une impliquait n'utiliser qu'une seule étiquette pour définir un nom de produit, tandis qu'une autre essayait d'enlever entièrement les numéros de brevet. De plus, nous utilisons pour l'instant seulement trois données sur chaque mot (Mot, Part-of-speech, appartenance au dictionnaire) mais d'autres informations pourraient encore être rajoutées. Par exemple, la position d'un mot dans une page, ou encore les balises dans lesquel il est contenu (par exemple mot en gras, titre, etc.)

Ci-joint, un tableau des résultats montrant les différentes méthodes d'étiquetage et leurs résultats sur le corpus original de 12 textes.

Table 3.6: Default, BI

Label	X	B	I	P
avid	0.93	0.10	0.28	0.91
cnd	0.98	0.50	0.63	0.89
dolby	0.97	0	0.02	0.98
ignite	0.90	0	0	1
ipc	0.98	0	0.45	0.51
mallin	0.97	0.55	0.53	0.99
rms	0.99	0	0.22	0.37
speckip	0.99	0	0	0.70
sun	0.93	0.40	0.56	0.98
swisslog	0.98	0.56	0.53	1
symantec	1	0.91	0.93	1
tivo	0.96	0.43	0.41	1

Table 3.7: I only

Label	X	I	P
avid	0.92	0.56	0.91
cnd	0.93	0.06	0.79
dolby	0.97	0.56	0.98
ignite	0.69	0	0.68
ipc	0.97	0.24	0.48
mallin	0.97	0.25	0.96
rms	0.98	0.24	0.40
speckip	0.91	0.12	0.60
sun	0.94	0.54	0.99
swisslog	0.98	0.37	1
symantec	0.99	0.54	1
tivo	0.94	0.47	0.99

Table 3.8: B, I and E

Label	X	B	I	E	P
avid	0.91	0.09	0.11	0.03	0.93
cnd	0.95	0.29	0.07	0.14	0.82
dolby	0.97	0	0	0	0.82
ignite	0.68	0	0	0	0.58
ipc	0.96	0.22	0.19	0.47	0.53
mallin	0.98	0.50	0.49	0.55	0.99
rms	0.94	0.21	0.14	0.24	0
speckip	0.92	0	0	0	0.69
sun	0.91	0.55	0.50	0.48	0.98
swisslog	0.99	0.67	0.61	0.24	1
symantec	0.98	0.91	0.37	0	1
tivo	0.97	0.71	0.49	0.43	0.99

Ces résultats ne sont pas très significatifs, simplement de par la taille du corpus d'apprentissage. En revanche, nous pouvons voir que la méthode "BI" semble être celle qui produit les meilleurs résultats, du moins à cette échelle. Il serait nécessaire de conduire des tests à plus grande échelle pour avoir un comparatif plus valide.

On remarque que nous retrouvons un problème majeur qui change beaucoup les résultats : selon la manière de parsing, on peut retrouver des ’ . ’ en milieu de phrases dérangeants, causés par des balises inconnues, des sauts de lignes invisibles, etc. Il est difficile de supprimer de tels points sans tous les supprimer, et on remarque que les vrais points qui déterminent les fins de phrases aident en général beaucoup à la détection des noms de produit. C’est un aspect à travailler dans le futur, mais qui a déjà été résolu dans le wp_corpus.
Il est important de noter que nous avons également testé avec différentes manières d’étiqueter les produits, comme plusieurs tags PB, PI, etc ou encore sans les étiqueter du tout, mais les résultats étaient toujours inférieurs à ceux offerts par d’autres méthodes. C’est pourquoi ils ne sont pas montrés dans ce rapport.

Finalement, Wapiti offre encore un feature qui pourrait être intéressant : dans le fichier d’entrée, une ligne blanche signifie une fin de séquence. Ainsi, du contenu ne prendra pas en compte la suite d’un fichier si celle-ci est séparée par une ligne vide. Il pourrait être prometteur de pouvoir isoler des parties du texte que nous estimons comme contenant des noms de produit. Cet aspect n’a pas été exploré durant le projet.

4 Sortie et Conclusion

4.1 Fichiers de sortie

Afin de facilement visualiser les résultats pour chaque fichier, nous avons créé deux types de sortie extrêmement simples : le premier est simplement un fichier texte affichant un produit (de type B-I-I-...) par ligne, le second une page HTML qui montre selon différentes couleurs la manière dont le fichier a été étiqueté.

Pour l'instant, aucun post-processing n'est appliqué. Il serait possible et très clairement envisageable de créer des règles simples de post-processing, comme par exemple enlever tout les mots qui font partie de et viennent après un "protected by", car on remarque que cette erreur apparaît très souvent. Il pourrait également être très intéressant de croiser les résultats de plusieurs modèles, tout en leur associant un certain poids selon leur performance lors d'entrainement et d'évaluation. Cet aspect n'a pas été abordé dans le projet.
Finalement, en utilisant une expression régulière plus complète, on pourrait encore réduire les erreurs sur les noms de produits qui contiennent parfois des numéros de brevets.

4.2 Futur du projet

En plus de ce qui a déjà été vu tout au long du rapport, il existe encore un moyen efficace et important d'avoir des résultats définitifs : créer un modèle avec précision. Les deux corpus utilisés ont tout les deux de très grands défauts, l'un étant bien trop petit et l'autre parfois imprécis, avec d'immenses pages vides qui déservent à l'intégrité du corpus.

C'est également la raison pour laquelle nous n'avons pas encore appliqué de post-processing. L'avenir proche du projet est la construction d'un modèle puissant et versatil via l'utilisation d'un corpus large, mais précis et correct. Les patterns à utiliser sont en grande partie implémentés ou annotés, mais ils sont également à améliorer lors de la construction du modèle final selon les nouveaux phénomènes encontrés. Une fois le modèle créé et travaillé, nous pourrons commencer à éliminer en post-processing ses erreurs fréquentes.

Parmis les modèles générés au hasard sur 40 textes et évalués sur 35, nous avons réussi à obtenir dans un instance particulière du test WP40_35 une moyenne des résultats de plus de 75% de F-value, avec environs 80% de recall et 75% de précision. Il paraît envisageable de pouvoir créer un modèle avec une moyenne de F-value allant jusqu'à 90%. Cela semblerait également correct par rapport aux valeurs de précision générales que l'on retrouve dans le domaine de l'apprentissage machine. Sur la voie de ces résultats, il sera important de garder en tête que le recall nous intéresse légèrement plus que la précision, car il est plus important d'être capable d'affirmer avoir détectés tout les produits de la page et d'autres en trop que ne pas être certain d'avoir tout les produits, même si l'on peut affirmer qu'ils sont corrects. Perfectionner le recall est donc un but majeur du modèle final.

5 APPENDICE

Corpus Originel :

http://www.cnd.com/patents
http://www.avid.com/en/legal/patent-marking
http://www.dolby.com/us/en/about/virtual-patent-marking.html
http://www.ignitepatents.com
http://www.ipc.com/legal/patents/
http://www.mallinckrodt.com/patents
http://www.rmspumptools.com/about-us/product-patents.php
http://speckip.com
https://us.sunpower.com/solar-panels-technology/patents/
http://www.swisslog.com/en/Products/HCS/Patents
https://www.symantec.com/about/legal/virtual-patent
https://www.tivo.com/legal/patents

Wapiti

NLTK

SpaCy

WebStruct