Titre: Title:	Amélioration de la précision de systèmes d'extraction de relations en utilisant un filtre générique basé sur l'apprentissage statistique
Auteur: Author:	Kevin Lange Di Cesare
Date:	2016
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Lange Di Cesare, K. (2016). Amélioration de la précision de systèmes d'extraction de relations en utilisant un filtre générique basé sur l'apprentissage statistique [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2115/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/2115/
Directeurs de recherche: Advisors:	Michel Gagnon, Amal Zouaq, & Ludovic Jean-Louis
Programme: Program:	Génie informatique

UNIVERSITÉ DE MONTRÉAL

AMÉLIORATION DE LA PRÉCISION DE SYSTÈMES D'EXTRACTION DE RELATIONS EN UTILISANT UN FILTRE GÉNÉRIQUE BASÉ SUR L'APPRENTISSAGE STATISTIQUE

KEVIN LANGE DI CESARE
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AVRIL 2016

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

AMÉLIORATION DE LA PRÉCISION DE SYSTÈMES D'EXTRACTION DE RELATIONS EN UTILISANT UN FILTRE GÉNÉRIQUE BASÉ SUR L'APPRENTISSAGE STATISTIQUE

présenté par: LANGE DI CESARE Kevin
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

M. DESMARAIS Michel C., Ph. D., président
M. GAGNON Michel, Ph. D., membre et directeur de recherche
Mme ZOUAQ Amal, Ph. D., membre et codirectrice de recherche
M. JEAN-LOUIS Ludovic, Ph. D., membre et codirecteur de recherche
Mme INKPEN Diana, Ph. D., membre externe

REMERCIEMENTS

Je tiens à remercier mon directeur de recherche, Michel Gagnon, et mes codirecteurs de recherche, Amal Zouaq et Ludovic Jean-Louis, pour leur encadrement et l'aide précieuse qu'ils m'ont apportée tout au long de mes travaux. Je tiens également à remercier Netmail Inc. de m'avoir offert un lieu de travail convivial et de m'avoir supporté financièrement au cours de la dernière année et demie. J'aimerais remercier Nick Stefan et Frédéric Bourget d'être allés de l'avant avec l'intégration du projet chez Netmail Inc. Enfin, je tiens à remercier mes parents qui ont cru en moi et qui ont su me soutenir moralement depuis le tout début de mes études.

RÉSUMÉ

L'extraction de relations contribue à l'amélioration de la recherche sémantique, recherche basée sur la compréhension du sens des termes de recherche. Puisque la recherche d'information est principalement axée sur des mots-clés, l'extraction de relations offre un éventail de possibilités en identifiant les liens entre les entités. L'extraction de relations permet entre autres de transformer de l'information non structurée en information structurée. Les bases de connaissances, telles que *Google Knowledge Graph* et *DBpedia*, permettent un accès plus précis et plus direct à l'information. Le *slot filling*, qui consiste à peupler une base de connaissances à partir de textes, a été une tâche très active depuis quelques années faisant l'objet de plusieurs campagnes évaluant la capacité d'extraire automatiquement des relations prédéfinies d'un corpus de documents. Malgré quelques progrès, les résultats de ces compétitions demeurent modestes. Nous nous concentrons sur la tâche de *slot filling* dans le cadre de la campagne d'évaluation *TAC KBP* 2013. Cette tâche vise l'extraction de 41 relations prédéfinies basées sur les *infobox* de Wikipédia (par exemple: *title*, *date of birth*, *countries of residence*, etc.) liées à des entités nommées spécifiques (personnes et organisations). Une entité nommée (l'entité requête) et une relation sont soumises à un système (extracteur de relations) qui doit automatiquement trouver, parmi un corpus de plus de deux millions de documents, toute entité liée à l'entité requête par la relation donnée. Le système doit également retourner un segment textuel justifiant cette relation. Ce mémoire présente un filtre basé sur l'apprentissage statistique dont l'objectif principal est d'améliorer la précision d'extracteurs de relations tout en minimisant l'impact sur le rappel. Notre approche consiste à filtrer la sortie des extracteurs de relations en utilisant un classifieur. Notre filtre est annexé à la sortie de l'extracteur de relations, pouvant ainsi être facilement testé sur n'importe quel système. Notre classifieur est basé sur un large éventail de caractéristiques (*features*), incluant des caractéristiques statistiques, lexicales, morphosyntaxiques, syntaxiques et sémantiques extraites en majorité des phrases justificatives soumises par les systèmes. Nous proposons également une méthode efficace permettant d'extraire les patrons les plus fréquents (ex.: catégories morphosyntaxiques, dépendances syntaxiques) afin d'en dériver des caractéristiques booléennes utiles pour notre tâche de filtrage. Les caractéristiques utilisées pour l'entraînement des classifieurs sont soit génériques. Ainsi, notre méthode peut être utilisée pour la classification de toute relation prédéfinie. Nous avons testé le filtre sur 14 systèmes ayant participé à la tâche de *slot filling*. Le filtre permet d'améliorer la précision pour chacun de ces systèmes. Nos résultats démontrent également que le filtre permet d'améliorer la précision du meilleur système de plus de 20% (points de pourcentage) et d'améliorer le F-score pour 20 relations.

ABSTRACT

Relation extraction is becoming a very important challenge for enhanced semantic search. In fact, while traditional information retrieval is mainly focused on keywords, relation extraction opens a whole range of possibilities by identifying the links between concepts and entities. Unstructured data can be transformed into structured data by using effective relation extraction to populate a knowledge base (ex: *Google Knowlegde Graph* and *DBpedia*). Slot filling, which mainly consists in the population of a knowledge base, has been a very active task in recent years and has been subject to several evaluation campaigns that assess the ability of automatically extracting previously known relations from corpora. Despite some progress, the results of these competitions remain limited. In this thesis, we focus on the English slot filling track within *TAC KBP* 2013 evaluation campaign. This track targets the extraction of 41 pre-identified Wikipedia infobox relations (e.g. *title*, *date of birth*, *countries of residence*, etc.) related to specific named entities (persons and organizations). A named entity and a relation are submitted to a system (relation extractor), which must automatically find, within a corpus containing over 2 million documents, every other entity that is linked to the query entity with this particular relation, and must return a textual segment that justifies this result.

This thesis presents a machine learning filter whose main objective is to enhance the precision of relation extractors while minimizing the impact on recall. Our approach consists in the filtering of relation extractors' output using a binary classifier. Our filter is appended to the end of the relation extractor's pipeline, thus allowing the filter to be tested and operated on any system. Another objective of this research is the identification of the most important features for the filtering step. Our classifier is based on a wide array of features including statistical, lexical, morphosyntactic, syntactic and semantic features. We also present a method for extracting the most frequent patterns (ex: part-of-speech, syntactic dependencies) between the query and the answer within the justification sentence from which we create boolean features indicating the presence of such patterns. The features used for training our classifiers are mostly generic and could be utilized to classify any pre-defined relation. We experimented the classifier on 14 systems participating in the English slot filling track of TAC KBP 2013 campaign. The filter allowed an increase in precision for every tested system. Our results also show that the classifier is able to improve the precision of the best system by more than 20% (in percentage points) and improve the F1-score for 20 relations.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

DS	Distant Supervision
ESF	English Slot Filling
GPE	Entité géopolitique (*Geo-Political Entity*)
HTML	HyperText Markup Language
KBP	Knowledge Base Population
LDC	Linguistic Data Consortium
NE	Entité nommée (*Named Entity*)
ORG	Organisation (*Organisation*)
POS	Catégorie morpho-syntaxique (*Part-of-speech*)
PER	Personne (*Person*)
SFV	Slot Filler Validation
SMO	Sequential Minimal Optimisation
SMOTE	Synthetic Minority Over-sampling Technique
SVM	Machine à vecteurs de support (*Support Vector Machine*)
TAC	Text Analysis Conference
XML	Extensible Markup Language

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

À l'époque des bases de connaissances structurées telles que *Google Knowledge Graph* [43], *DBpedia* [7] et le nuage *Linked Open Data* [6], l'extraction de relations devient un enjeu très important pour améliorer la recherche sémantique, recherche basée sur la compréhension du sens des termes de recherche. Puisque la recherche d'informations est principalement axée sur des mots-clés, l'extraction de relations offre un éventail de possibilités en identifiant les liens entre les entités, permettant ainsi d'effectuer des recherches complexes.Les entités peuvent être, par exemple, des personnes, des organisations, des emplacements géographiques ou des dates. L'extraction de relations demeure une tâche difficile présentant plusieurs défis sur lesquels les groupes de recherche de renommée mondiale se penchent actuellement. Malgré le progrès récent de la capacité des systèmes à reconnaître efficacement les entités nommées dans les textes, l'appariement de ces entités nommées à des noeuds dans une base de connaissances demeure une tâche difficile. L'identification de relations sémantiques entre les entités est également un défi important. L'extraction de relations et sa sous-tâche, le *slot filling*, ont récemment été l'objet de plusieurs campagnes d'évaluation qui évaluent la capacité d'extraire automatiquement des relations prédéfinies d'un corpus de documents Le *slot filling* consiste à remplir une base de connaissances dont certaines ressources sont manquantes. Une base de connaissances regroupe des connaissances spécifiques à un domaine de manière structurée en mettant en relation diverses ressources existant sur le Web. Les relations sont représentées par des triplets (sujet, prédicat, objet) dont chacun des membres pointe vers une ressource. Une base de connaissances est souvent représentée par un modèle objet comprenant des classes, sous-classes et instances. L'objectif d'un système de *slot filling* est de trouver automatiquement les entités qui sont en relation avec une entité présente dans la base de connaissances à partir d'un corpus de textes. Par exemple, pour une requête composée de l'entité *Barack Obama* et de la relation *state of birth*, le système doit automatiquement trouver l'entité qui correspond à l'état de naissance de *Barack Obama*. Malgré quelques progrès, les performances de ces systèmes demeurent modestes. Dans ce mémoire, nous nous concentrons sur la campagne d'évaluation *TAC KBP*, plus particulièrement sur la tâche de *English Slot Filling* qui cible l'extraction de 41 relations prédéfinies basées sur les *infobox* de Wikipédia (par exemple: *title*, *date of birth*, *countries of residence*, etc.) liées à des entités nommées spécifiques (*personnes* et *organisations*) [44]. Dans le cadre de cette tâche, une entité nommée et une relation sont soumises à un extracteur de relations qui doit automatiquement trouver, parmi un corpus de plus de deux millions de documents, toute entité qui est liée à l'entité de requête par la relation spécifiée. Le système doit également retourner

un segment textuel justifiant ce résultat [44, 46]. Le but de cette tâche est de peupler une base de connaissances, dans ce cas-ci, une image des *infobox* de Wikipédia en date de 2008 [21], pour laquelle les réponses à une liste prédéfinie de relations pour 100 entités nommées sont manquantes. Les systèmes sont évalués selon les métriques suivantes: rappel, précision et F1. Le rappel est égal au nombre de réponses *correctes* renvoyées par le système sur le nombre total de réponses *correctes* existant dans la référence. La référence est construite à partir des sorties de tous les systèmes ayant participé à la tâche ainsi que les réponses que les évaluateurs ont trouvées manuellement. La précision est égale au nombre de réponses *correctes* renvoyées par le système sur le nombre total de réponses renvoyées par le système (*correctes* et *incorrectes*). Le F1 est une moyenne harmonique du rappel et de la précision ($F_1 = 2(précision * rappel)/(précision + rappel)$). En 2014, le meilleur système, soumis par Stanford, a obtenu un rappel de 0,277, une précision de 0,546 et un F1 de 0,368 [46, 4]. Ces résultats sont quelque peu similaires à ceux soumis lors de l'année précédente, pour laquelle la tâche était légèrement moins difficile, par le meilleur système, *Relation Factory*, qui a réalisé un rappel de 0,332, une précision de 0,425 et un F1 de 0,373 [44, 38].

1.1 Exemple de la structure d'un système de *slot filling*

Pour la plupart des systèmes, la tâche de *slot filling* est effectuée principalement en deux étapes: la génération de candidats et la validation de candidats [38]. La génération de candidats peut elle-même être décomposée en trois étapes: l'extension des requêtes, la récupération de documents et la correspondance de candidats. L'extension de requêtes vise à trouver les formes alternatives d'expressions des entité nommées. Par exemple, l'entité *Barack Obama* pourrait être désignée par les expressions suivantes: *Président Obama*, *Barack Hussein Obama*, *Obama*, etc. Ces formes alternatives constituent l'extension de la requête sous une de ses formes. Le système retient par la suite les documents contenant une référence à l'entité requête. Les documents retenus sont annotés avec un annotateur d'entités nommées qui associe le type d'entité nommée correspondant (ou *OTHER* dans le cas où un mot n'est pas une entité nommée) à chacun des mots du corpus. Durant la phase de correspondance de candidats, les candidats, c'est-à-dire les phrases contenant une référence à l'entité requête (référence originale ou extension) et un type d'entit énommée correspondant au type d'une réponse potentielle pour la relation en question, sont retenus par le système [38]. Par exemple, la relation *city of birth* requière en guise réponse une entité nommée de type *GPE-CITY*. La phrase "*Barack Obama was born in Honolulu, Hawaii in 1961.*" serait annotée par l'annotateur d'entités nommées comme suit: "*Barack/PERSON Obama/PERSON was/OTHER born/OTHER in/OTHER Honolulu/GPE-CITY, Hawaii/GPE-STATE in/*

OTHER 1961/DATE." Cette phrase serait un candidat valide pour la relation *city of birth* et l'entité requête *Barack Obama* puisqu'en plus de contenir une référence à l'entité requête, elle contient l'entité *Honolulu*, qui est du type *GPE-CITY*. Pour chaque relation, les phrases candidates retenues sont envoyées en entrée au module effectuant la validation de candidats. L'implémentation de cette étape peut varier selon le système, mais est habituellement, depuis quelques années, basée sur la supervision distante et/ou sur des patrons [44].

1.2 Problématique et objectifs de recherche

Le rappel d'un système peut être augmenté considérablement en configurant une des deux étapes du processus, soit la génération ou la validation des candidats, de manière à ce qu'elle soit plus tolérante, par exemple en réduisant le seuil de confiance minimum du système. Cependant, cela entraîne une baisse de la précision. Ceci est un problème bien connu en extraction d'information où il faut trouver un compromis entre la précision et le rappel. Ce mémoire décrit une approche de filtrage pouvant être appliquée sur la sortie d'un extracteur de relations afin d'améliorer sa précision. Nous traitons les questions de recherche suivantes:
QR1: Quelles sont les caractéristiques les plus importantes pour le filtrage de relations?
QR2: Y a-t-il une méthode générique permettant d'augmenter la précision des extracteurs de relations sans dégrader considérablement le rappel?
QR3: Est-ce que certaines relations sont plus sensibles au filtrage?

1.3 Plan du mémoire

La suite de ce mémoire est constituée de six chapitres. Le deuxième chapitre présente une revue de la littérature survolant le domaine de l'extraction de relations et se concentrant sur le *slot filling*. Quelques travaux récents visant à améliorer les performances de systèmes existants sont présentés. Le troisième chapitre présente la démarche de l'ensemble du travail de recherche effectué. Ce chapitre décrit la méthode de recherche, présente une explication approfondie de quelques concepts abordés dans les articles présentés aux quatrième et cinquième chapitres. Le quatrième chapitre est une reproduction intégrale de l'article *A Machine Learning Filter for Relation Extraction* publié dans le cadre de la conférence *WWW2016: 25th International World Wide Web Conference*. Cet article explique brièvement une première version du filtre et présente les résultats de l'application de ce filtre aux différents extracteurs de relations ayant participé à la tâche de *slot filling* dans le cadre de TAC KBP 2013. Le cinquième chapitre est une reproduction intégrale de l'article *A Machine Learning*

Filter for the Slot Filling Task soumis à la revue *Data & Knowledge Engineering* en date du 9 mars 2016. Cet article explique en plus de détails la méthode de filtrage d'extracteurs de relations et présente des résultats exhaustifs de la recherche menée. Enfin, le cinquième chapitre contient une discussion générale sur les résultats obtenus par rapport aux questions de recherche présentées dans la section précédente. Plusieurs suggestions de pistes à explorer pour des travaux futurs sont également proposées. Le dernier chapitre, quant à lui, conclut le mémoire.

CHAPITRE 2 REVUE DE LITTÉRATURE

Ce chapitre présente une revue de la littérature sur l'extraction et le filtrage relations. La première partie porte sur les différentes approches utilisées pour le développement de systèmes de *slot filling*, notamment le *bootstrapping* et la supervision distante, les deux approches dominantes. La seconde partie porte sur les différentes méthodes permettant l'amélioration des performances de ces systèmes.

2.1 Extraction de relations et *slot filling*

L'extraction de relations étant un enjeu majeur, plusieurs groupes de recherche ont travaillé sur la tâche durant ces quelques dernières années. L'extraction de relations et le *slot filling* sont des tâches complexes nécessitant des systèmes multifonctionnels. Les méthodes telles que l'extraction de relations supervisées notamment, posent certains problèmes. Premièrement, il y a très peu de données manuellement annotées disponibles pour l'entraînement de ces systèmes puisqu'elles sont coûteuses à produire. De plus, il y a généralement un biais lié au domaine sur lequel l'annotation des données est effectuée [25]. Contrairement aux approches supervisées, les différentes approches d'extraction de relations non supervisées utilisent une très grande quantité de données et permettent l'extraction d'une grande variété de relations. Toutefois, ces relations ne sont pas nécessairement facilement appariables à une base de connaissances [22]. Ainsi les systèmes modernes (par exemple [38], [4], [34]) utilisent différentes techniques telles que le *bootstrapping* de patrons [11] et la supervision distante [32], qui ne nécessitent pas de données manuellement annotées et permettent l'extraction de relations prédéfinies.

Les techniques de *bootstrapping* utilisent un nombre restreint de paires d'entités en relation (*seed instances*) comme amorce du processus d'apprentissage [11, 1]. Ces paires servant d'amorces sont utilisées pour extraire des patrons d'un vaste corpus, qui sont par la suite utilisés pour extraire d'autres paires d'entités permettant de générer d'autres patrons de manière itérative. En fonction des paramètres utilisés pour le *bootstrapping* (nombre d'itérations, nombre de patrons conservés), un phénomène de dérive sémantique peut être observé : les nouvelles relations extraites sont de plus en plus différentes des relations de l'ensemble d'amorce [32]. Cette dérive sémantique a pour conséquence de pénaliser la précision du système. Certaines approches s'appuient sur des patrons de dépendances syntaxiques. Par exemple, certains auteurs tel que [30] utilisent le *bootstrapping* pour générer des patrons de dépendances syntaxiques. Ils entraînent des classifieurs à partir de l'arbre de dépendances

issu de l'analyse syntaxique d'une phrase candidate en sélectionnant tous les chemins contenant moins de trois noeuds entre la requête et la réponse potentielle. Leur système a obtenu un rappel de 0,223, une précision de 0,199 et un F1 de 0,21 à la tâche de *slot filling* dans le cadre de TAC KBP 2014 pour laquelle le F1 médian était de 0,198.

La plupart des systèmes de *slot filling* utilisent la supervision distante pour acquérir des données pour l'entraînement de leurs classifieurs. La supervision distante est une technique d'apprentissage qui ne requiert pas de données manuellement annotées, mais requiert plutôt une base de connaissances ainsi qu'un corpus de textes. Pour une relation donnée, des paires d'entités connues pour être en relation sont extraites de la base de connaissances. Des phrases contenant les deux entités sont extraites du corpus et utilisées en tant que données annotées pour entraîner un classifieur [32]. Cette approche semble être plus efficace que le *bootstrapping*, puisque la plupart des systèmes utilisant la supervision distante ont obtenu de meilleures performances en 2013 et 2014 [46]. Par exemple, le système Relation Factory [38], qui a obtenu les meilleures performances de la compétition KBP utilise des machines à vecteurs de support entraînées par supervision distante pour effectuer la validation de candidats. Les paires d'arguments de supervision distante sont obtenues en appariant les relations de *Freebase* à celles de TAC et en faisant correspondre des patrons d'amorce créés manuellement (*hand-crafted seed patterns*) au corpus de textes de TAC 2009 pour chaque relation [39]. La figure 2.1 présente un exemple de relation de Freebase correspondant à une relation de TAC KBP. [38] utilisent un maximum de 20k paires d'arguments par relation qui sont par la suite mises en correspondance avec le corpus de TAC 2009 duquel un maximum de 500 phrases par paire sont utilisées comme données d'entraînement. Malgré le fait que la qualité des exemples générés est inférieure à celle des exemples des approches utilisant les techniques de *bootstrapping*, le nombre d'exemples est largement supérieur. Afin d'améliorer les performances des classifieurs entraînés sur des données bruitées, Relation Factory effectue de l'entraînement par agrégats. Au lieu de traiter chaque phrase comme un exemple d'entraînement individuel, le système regroupe toutes les phrases par paire d'entités et extrait un vecteur normalisé de caractéristiques pour chaque paire. En plus d'accélérer l'entraînement, cette méthode permet de réduire l'influence de caractéristiques qui sont grandement corrélées à une seule paire d'arguments d'amorce.

2.2 Approches visant l'amélioration d'extracteurs de relations

Au lieu de développer des systèmes complets, il est également commun pour les groupes de recherche de viser l'amélioration d'extracteurs de relations existants, puisqu'il y a une grande marge d'amélioration potentielle. [15] a développé un modèle de reclassement supervisé basé

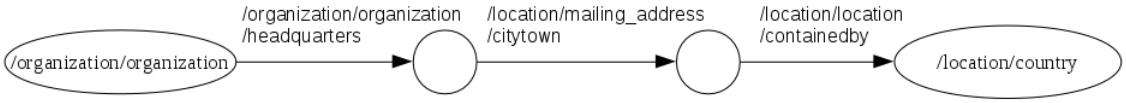

Figure 2.1 Représentation de la relation de Freebase correspondant à la relation *org:cities_of_headquarters* de TAC KBP. Les noeuds correspondent aux restrictions sur le type de chacune des entités de Freebase. Les arêtes correspondent aux restrictions sur la relation. Le noeud de départ correspond à l'entité de la requête dans la relation de TAC KBP et le noeud d'arrivée correspond à l'argument de réponse

sur l'entropie maximale afin de reclasser des réponses candidates pour une relation. Ce modèle est entraîné sur des paires requête-réponse annotées afin de prédire la confiance de chacune des réponses. Les auteurs utilisent des caractéristiques basées premièrement sur un score de confiance assigné par *OpenEphyra*, un module web d'un système de question-réponse à domaine ouvert (*open domain*) [41], sur la co-occurrence dans la collection de documents, sur les types d'entités nommées et finalement sur le type de relation. Testée sur les requêtes de la tâche de *slot filling* de TAC 2009, cette approche a obtenu une précision 3,1% plus élevée et un rappel 2,6% plus élevé que le système ayant originalement obtenu les meilleurs résultats. Les auteurs dans [33] ont travaillé sur un filtre basé sur des règles sémantiques. C'est une méthode non supervisée évolutive permettant de construire automatiquement une représentation sémantique d'une relation donnée. Les auteurs comparent des approches utilisant *Wordnet* et *BabelNet*. L'approche utilisant *BabelNet* permet une amélioration substantielle de la précision et seulement une légère perte de rappel. Leur approche fonctionne bien sur les relations de type *acquisition* et de type *naissance/mort*, mais elle n'est pas efficace sur les relations qui ont une diversité lexicale plus élevée comme le type *lieu de résidence*. Ils n'ont cependant testé leur approche que sur sept relations. Un module d'apprentissage actif est présenté dans [5] combinant l'apprentissage supervisé à la supervision distante. En fournissant seulement un faible échantillon de données annotées, ce système permet une amélioration de 3,9 de F1 pour le système de Stanford ayant participé à la tâche de *slot filling* de 2013.

Toutes ces méthodes de rehaussement sont incorporées directement au système et visent, dans la plupart des cas, un système en particulier ou fonctionnent mieux sur quelques relations particulières. En revanche, ce mémoire décrit une approche générique pouvant être facilement intégrée à n'importe quel extracteur de relation et pouvant être entraînée sur n'importe quelle relation.

Une autre possibilité pour le rehaussement d'un système unique est l'apprentissage par ensemble (*ensemble learning*). Dans une incitation au développement de techniques d'apprentissage

par ensemble, TAC propose également la tâche de *Slot Filler Validation (SFV)*, qui consiste à raffiner la sortie d'extracteurs de relations [45]. [49] présente un système ayant participé à la tâche de SFV de 2013 qui utilise une approche d'*agrégation*. Pour une requête donnée, les auteurs agrègent les scores de confiances "bruts" produits par divers extracteurs de relations afin de produire un score de confiance agrégé. Cette valeur est basée sur le nombre de fois qu'une réponse est retournée par un système et sur le score de confiance attribué à chaque instance. Ils assignent un poids à chacune des occurrences en se basant sur la "confiance générale" du système ayant soumis la réponse. Leur approche obtient de bons résultats pour les relations à réponse unique (précision: 0,797, rappel: 0,794, F1: 0,795). Toutefois, ce n'est pas le cas pour les relations à multiples réponses. De manière similaire, [37] a développé un système utilisant l'empilage (*stacking*) pour l'entraînement d'un classifieur combinant les résultats de plusieurs systèmes de manière optimale. Leur système combine la sortie de plusieurs systèmes de *slot filling* en utilisant un "méta-classifieur" et en utilisant les scores de confiances des différents systèmes. Ils dérivent également des caractéristiques à partir de la provenance de chacune des réponses (document de provenance) et du recoupement des décalages de caractère (par rapport à l'origine dans le document) de chacune des réponses. Pour une requête donnée (entité/relation), lorsque N systèmes fournissent des réponses et n de ces systèmes fournissent le même identifiant de document en guise de provenance, alors le score de provenance associé à cet identifiant de document est égal à n/N. Leur système s'appuie entre autres sur le score de provenance pour le filtrage d'instances. De même, ils utilisent le coefficient de Jaccard afin de mesurer la fiabilité de la position d'une réponse dans le document par rapport aux autres pour une requête (entité/relation). Leur système s'appuie également sur ce coefficient de Jaccard pour la classification. De manière comparable à Relation Factory [38], ils utilisent une méthode d'expansion basée sur les textes d'ancrage (*anchor text*) de Wikipédia et appliquée à la réponse afin de détecter les redondances, c'est-à-dire des justifications différentes désignant la même relation entre les mêmes entités, augmentant ainsi la précision. Cette méthode leur permet d'atteindre un F1 de 50,1% sur les soumissions d'ESF TAC 2014, ce qui est une amélioration considérable par rapport au meilleur système de la campagne (39,5%). De manière générale, les méthodes par ensemble atteignent de meilleures performances par rapport à celles obtenues par les différents systèmes individuellement. Toutefois, elles nécessitent l'accès à plusieurs extracteurs de relations, ce qui, en pratique, n'est généralement pas possible. Ce mémoire décrit une approche moins complexe et moins gourmande en matière de ressources qui vise l'amélioration de la précision des extracteurs de relations tout en limitant la perte de rappel de ces derniers.

CHAPITRE 3 MÉTHODOLOGIE

Ce chapitre présente la démarche de l'ensemble de l'expérience effectuée ainsi que méthodologie de l'expérience présentée dans les articles des chapitres 4 et 5 en expliquant plus en détail certains concepts pertinents. En plus de la solution retenue, il est également question de pistes de solutions explorées qui ne font pas l'objet des articles aux chapitres subséquents.

3.1 Description générale de l'approche

Notre approche consiste à filtrer la sortie d'extracteurs de relations et a été testée sur les systèmes ayant participé à la tâche de *slot filling* de TAC KBP 2013. Pour chaque relation, un classifieur est joint à la sortie d'un extracteur de relations, éliminant les instances classifiées comme fausses. La figure 3.1 présente le flux de données général: les justifications aux réponses soumises par les extracteurs de relations sont prétraitées et les caractéristiques sont extraits pour la classification. Cette approche vise à améliorer la précision d'un extracteur de relations. Une augmentation du rappel n'est pas possible puisque le filtre ne peut pas générer d'instances supplémentaires. La sortie des extracteurs de relations contenant une phrase justificative pour chaque réponse est envoyée à l'annotateur d'entités nommées qui fournit le type d'entité nommée pour chacun des mots. La sortie des extracteurs de relations est également passée à l'annotateur de catégories morphosyntaxiques et à l'analyseur syntaxique qui retournent les catégories morphosyntaxiques pour chacun des mots ainsi que l'arbre de dépendances syntaxiques. Ensuite, les patrons les plus fréquents sont extraits de la justification brute ainsi que de la sortie de l'annotateur de catégories morphosyntaxiques et de l'analyseur syntaxique. Ces patrons ainsi que les phrases prétraitées sont utilisés pour l'extraction de caractéristiques. Les vecteurs de caractéristiques résultants sont par la suite utilisés pour l'entraînement du classifieur. Les données de test (sorties de l'extracteur de relations testé) qui contiennent également des phrases justificatives sont soumises au même processus, c'est-à-dire la reconnaissance d'entités nommées, l'annotation de catégories morphosyntaxiques et l'analyse syntaxique. Les mêmes caractéristiques sont extraites et le classifieur retient ou rejette les réponses en se basant sur son modèle de classification.

3.2 Données d'entraînement

L'ensemble des données est composé de réponses des extracteurs de relations aux requêtes visant les 41 relations prédéfinies de la tâche de *slot filling* de TAC KBP 2013. De ces 41

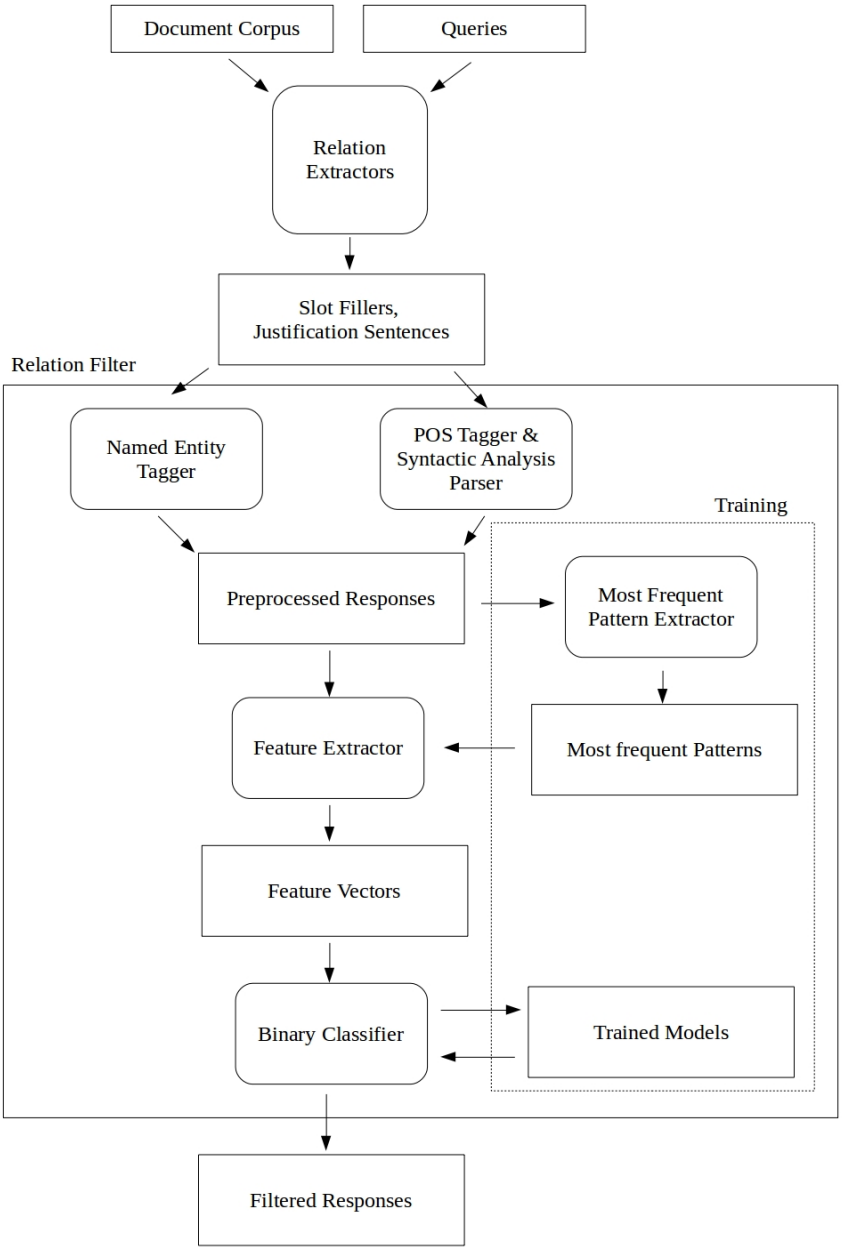

Figure 3.1 Structure et flux de données du filtre

relations, il y a 25 relations pour les requêtes dont l'entité est de type *personne* et 16 relations pour les requêtes dont l'entité est de type *organisation* [20]. Les réponses sont évaluées comme *correctes* lorsque la réponse et la justification sont exactes et *incorrectes* dans le cas contraire. La quantité de données d'entraînement varie grandement d'une relation à l'autre, car premièrement, certaines relations requièrent de multiples réponses alors que d'autres nécessitent une réponse unique et deuxièmement parce que les extracteurs de relations trouvent plus facilement des réponses potentielles pour certaines relations en comparaison avec d'autres. Il est également important de noter qu'il y a un déséquilibre des données favorisant la classe *incorrecte* pour la plupart des relations.

Dans le cadre de la tâche de *slot filling*, 100 requêtes composées d'une entité nommée (50 *personnes* et 50 *organisations*) et d'une liste de relations pour chacune d'elles ont été fournies aux extracteurs de relations. En guise de réponse à ces requêtes, les soumissions des systèmes doivent fournir les champs suivants: l'identifiant de la requête, l'identifiant du document de provenance de la réponse, la réponse à la relation spécifiée dans la requête (identifiée comme le *slot*) sous sa forme normalisée (nom, valeur ou chaîne de caractères), la position de l'entité requête dans le document, la position de la réponse, la position de la justification ainsi que le score de confiance attribué à la réponse par le système [44]. Toutes les réponses proviennent du corpus du *Linguistic Data Consortium* intitulé *TAC 2013 KBP Source Corpus (LDC2013E45)* constitué de 2.1M de documents dont 1M de fils de nouvelles provenant de *Gigaword*, 1M de pages web et 100K documents provenant de forums de discussions [44]. Les systèmes sont tenus de soumettre entre une et cinq soumissions pour lesquelles la configuration du système peut varier. Chaque soumission est évaluée et annotée par des évaluateurs du *LDC*. Dans le cadre de notre expérience, nous avons utilisé les données des requêtes d'évaluation pour la tâche de *Slot Filler Validation (SFV)*, qui sont composées des soumissions annotées de la tâche de *slot filling* pour lesquelles le système ayant soumis chaque instance est explicitement indiqué. L'ensemble de données complet utilisé dans le cadre de notre expérience est l'agrégat de données annotées composé de la meilleure soumission de chaque système participant, c'est-à-dire la soumission ayant réalisé le meilleur F1.

Comme mentionné, les systèmes doivent fournir pour chaque sortie, la réponse sous sa forme normalisée, la justification, l'identifiant du document de provenance ainsi que la position de la réponse, de la requête et de la justification. L'identifiant de document et les positions dans le document en question constituent les données d'entraînement brutes de nos classifieurs. Ces données sont, dans certains cas, inutilisables puisque certains systèmes fournissent des positions erronées. Dans la plupart des cas, ces positions erronées sont causées par une omission des balises HTML dans le calcul des décalages de caractères. Les justifications comportant un nombre de caractères supérieur à 600 sont retirées afin d'assurer un fonc-

tionnement convenable de l'annotateur de catégories morphosyntaxiques. Pour cette même raison, nous retirons également les balises HTML dans les justifications ainsi que les points, points d'interrogation et points d'exclamation au début des justifications. Par exemple, le segment textuel justificatif suivant contient des balises HTML et un point d'exclamation qui seraient retirés: "*! Ramazan Bashardost </P> <P> A lawmaker and former Cabinet minister*".

Une fois le nettoyage de données effectué, nous retirons, pour un système donné, sa sortie de l'ensemble d'entraînement complet et utilisons ce sous-ensemble de données pour des fins de tests. Cette méthode nous permet de comparer les performances initiales du système testé avec celles après que celui-ci ait été soumis au filtrage en utilisant le script d'évaluation officiel de TAC KBP [1].

3.3 Prétraitement des données

Les justifications doivent être annotées avec des catégories morphosyntaxiques et des types d'entités nommées afin de permettre l'extraction de plusieurs caractéristiques. L'arbre de dépendances syntaxiques est également nécessaire pour l'extraction de caractéristiques syntaxiques. L'analyse syntaxique par dépendance consiste à représenter la structure d'un texte écrit langue naturelle. Cette opération suppose une formalisation du texte défini par un ensemble de règles de syntaxe formant une grammaire formelle [27]. La structure révélée par l'analyse indique précisément la manière dont les règles de syntaxe sont combinées dans le texte. Cette structure est souvent une hiérarchie de syntagmes, représentables par un arbre syntaxique. Nous utilisons le *Stanford Parser* pour obtenir les annotations de catégories morphosyntaxiques ainsi que l'arbre syntaxique pour chacune des justifications [19]. Les types d'entités nommées, quant à eux, sont obtenus avec *Sequor* [16]. La figure 3.2 présente un exemple d'une phrase justificative pour laquelle les types d'entités nommées, les catégories morphosyntaxiques et l'arbre de dépendances syntaxiques ont été extraits.

Afin d'éviter le surentraînement, nous ne filtrons pas les relations pour lesquelles il y a moins de 15 instances pour une classe (*correcte*/*incorrecte*). Le déséquilibre de données amène les classifieurs à favoriser la classe majoritaire [13]. Pour la plupart des relations, il y a une plus grande quantité de données d'entraînement pour la classe *incorrecte* que pour la classe *correcte*. Pour chaque relation dont le nombre d'instances pour la classe *incorrecte* est supérieur à au moins deux fois le nombre d'instances pour la classe *correcte*, nous sous-échantillonnons l'ensemble d'instances de la classe *incorrecte* en cinq sous-ensembles. Ces

[1] http://www.nist.gov/tac/2013/KBP/SentimentSF/tools/SFScore.java

Relation:	**Query:**	**Filler:**
City of birth	Barack Obama	Honolulu

Raw justification sentence:

Nicole Kidman and **Barack Obama** were both born in **Honolulu**, Hawaii.

Named-entity tags:

Nicole/*B-PERSON* Kidman/*I-PERSON* and/*O* **Barack/*B-PERSON*** **Obama/*I-PERSON*** were/*O* both/*O* born/*O* in/*O* **Honolulu/*B-GPE:CITY*** ,/*O* Hawaii/B-GPE:STATE_PROVINCE ./*O*

Part-of-speech tags:

Nicole/*NNP* Kidman/*NNP* and/*CC* **Barack/*NNP*** **Obama/*NNP*** were/*VBD* both/*DT* born/*VBN* in/*IN* **Honolulu/*NNP*** ,/, Hawaii/NNP ./.

Syntactic dependency tree:

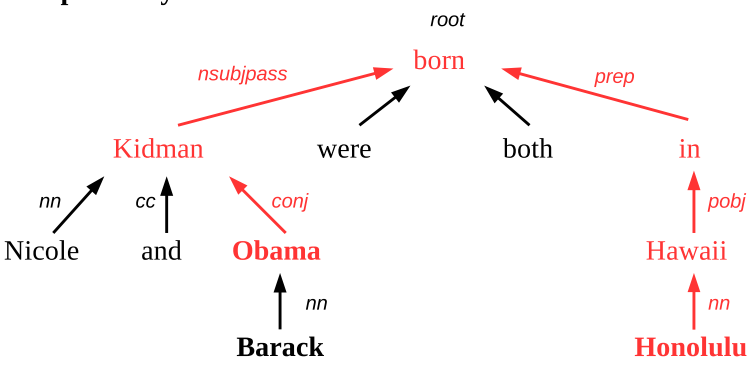

Figure 3.2 Exemple d'une phrase justificative pour laquelle les types d'entités nommées, les catégories morphosyntaxiques et l'arbre de dépendances syntaxiques ont été extraits

sous-ensembles sont échantillonnés aléatoirement et comportent le même nombre d'instances *incorrectes* que *correctes*. Le tableau A.4 en annexe montre, pour chacun des systèmes, les relations qui ont été éliminées du processus de filtrage ainsi que celles dont l'ensemble de données a été rééquilibré.

3.4 Identification des caractéristiques

Un des objectifs de notre approche est d'expérimenter différents ensembles de caractéristiques et de proposer une sélection de caractéristiques génériques. Notre filtre s'appuie sur plusieurs caractéristiques dans les catégories suivantes: statistique, sémantique, lexicale et syntaxique. Ces caractéristiques sont présentées et décrites brièvement dans le tableau 5.2. Les caractéristiques seront expliquées en détail au chapitre 5. La plupart des caractéristiques sont dérivées des patrons les plus fréquents (mots, catégories morphosyntaxiques et dépendances syntaxiques) dans l'ensemble d'entraînement. La section suivante décrit plus en détail la méthode utilisée afin d'identifier les patrons les plus fréquents et fait lumière sur les différents types de patrons.

3.5 Patrons les plus fréquents

Afin de bien modéliser chacune des différentes relations, il est pertinent de retenir les patrons les plus fréquents pour chacune de ces relations. Nous nous concentrons principalement sur les contextes entre les entités (requête et réponse). Par exemple, pour les patrons de catégories morphosyntaxiques, nous extrayons les ensembles de catégories morphosyntaxiques entre les entités pour chacune des instances dans l'ensemble d'entraînement. Le but est d'extraire les patrons les plus fréquents et de dériver des caractéristiques booléens indiquant la présence ou l'absence de ces patrons. Apriori est un algorithme qui permet d'extraire les sous-ensembles les plus fréquents tout en limitant le temps d'exécution [2]. L'algorithme permet d'identifier tous les sous-ensembles d'éléments qui sont présents au-delà d'une certaine proportion dans l'ensemble d'entraînement.

Dans le cadre de l'expérience, l'objectif est de discerner les instances *correctes* des instances *incorrectes*. Nous voulons donc déterminer les patrons les plus fréquents pour chacune des deux classes. Nous voulons également nous assurer que les patrons retenus pour une classe donnée soient exclusifs à cette classe. La première étape consiste à séparer en deux ensembles les instances d'entraînement pour chacune des relations selon la classe de celles-ci (*correcte*/*incorrecte*). Pour chacun des ensembles, nous extrayons les patrons les plus fréquents. La seconde étape est de déterminer pour chacun des exemples s'il est exclusif à la classe en

question. Pour ce faire, nous calculons le support du sous-ensemble retenu sur la classe opposée. Par exemple, pour un sous-ensemble retenu pour la classe *correcte*, nous déterminons son support sur la classe *incorrecte*, c'est-à-dire la proportion des instances de la classe *incorrecte* il correspond. Nous fixons un ratio *support* (*classe visée*)/*support* (*classe opposée*) minimum que nous appellerons la *spécificité*. Ainsi, nous rejetons les patrons qui ont une spécificité inférieure à la spécificité minimale que nous avons fixée.

Afin de ne retenir que les patrons les plus discriminants, un nombre maximum de patrons à retenir est fixé pour chacune des relations. Un support et une spécificité de départ sont également fixés. Suite à l'exécution d'Apriori et à la sélection de patrons basé sur la spécificité de ceux-ci, lorsque le nombre de patrons retenus est supérieur au maximum fixé, nous incrémentons le support et la spécificité par des valeurs spécifiées et réitérons jusqu'à ce que le nombre de patrons retenus soit inférieur ou égal au nombre de patrons maximum spécifié.

3.5.1 Exemple de patrons fréquents avec Apriori

À titre d'exemple, considérons cinq phrases justificatives pour une relation donnée dont chacun des mots a été annoté avec un annotateur de catégories morphosyntaxiques. Pour chacune de ces phrases, nous avons retenu la catégorie morphosyntaxique des mots se trouvant entre les entités (requête et réponse) dans la phrase pour former un ensemble de catégories morphosyntaxiques pour chaque phrase, soit cinq ensembles au total: {NNP, NN, VBG, NN}, {NNP, NN}, {NNP, VBG, IN, NN}, {NN, VBG IN, NNP}, {VBG, DT, NN, IN, DT, NNP, NNP}. Nous voulons retenir les sous-ensembles qui sont présents, par exemple, au moins une fois dans 60% des ensembles de base, c'est-à-dire dont le *support* ≥ 0.6. Premièrement, évaluons le support de chacun des éléments: {NNP}: 1, {NN}: 0.8, {VBG}: 0.8, {IN}: 0.6, {DT}: 0.2. Tous les éléments sauf {DT} ont un support supérieur ou égal à 0.6. La prochaine étape est de déterminer les paires dont le support ≥ 0.6. Nous pouvons déjà exclure {DT}. Voici le support pour chacune des paires: {NNP, NN}: 1, {NNP, VBG}: 0.8, {NNP, IN,}: 0.4, {NN, VBG}: 0.6, {NN, IN}: 0.4, {VBG, IN}: 0.6. Nous voulons maintenant trouver les triplets dont le support ≥ 0.6. Nous pouvons déjà exclure les triplets contenant des paires dont le support < 0.6. Nous obtenons donc: {NNP, NN, VBG}: 0.6. Nous avons déterminé tous les sous-ensembles dont le support ≥ 0.6: {NNP}, {NN}, {VBG}, {IN}, {NNP, NN}, {NNP, VBG}, {NN, VBG}, {NNP, NN, VBG}. Apriori permet de calculer le support seulement pour les ensembles de longueur l dont chacun des sous-ensembles de longueur $l-1$ satisfont les contraintes limitant ainsi le temps d'exécution.

3.5.2 N-grammes

Nous avons utilisé ce processus pour retenir différents contextes au niveau de la phrase: les mots dans la phrase (excluant les entités nommées et les *stop-words*) et les catégories morphosyntaxiques des mots se situant entre les entités (requête et réponse). Nous avons également utilisé ce processus pour retenir des contextes au niveau de l'arbre de dépendances syntaxiques. Nous avons premièrement retenu les dépendances syntaxiques se situant entre les entités au niveau de l'arbre syntaxique. Puisque les patrons retenus à l'aide d'Apriori ne tiennent pas compte de l'ordre d'apparition des éléments, nous avons ajouté de l'information aux patrons de dépendances syntaxiques. Nous avons donc ajouté aux simples patrons de dépendances syntaxiques les catégories morphosyntaxiques du mot ainsi que la direction du lien dans l'arbre. Pour déterminer la direction, nous suivons un chemin dans l'arbre qui débute à la requête et qui se termine à la réponse. Si la dépendance syntaxique du mot courant est liée à un mot qui se situe plus près de la réponse que le mot courant, la direction sera "↑". Si la dépendance syntaxique du mot courant est liée à un mot qui se situe plus près de la requête que le mot courant, la direction sera "↓". Si le mot courant est situé à la racine de l'arbre, la direction sera "-". Ainsi, nous retenons des patrons dont l'élément unitaire est un triplet de dépendance syntaxique, catégorie morphosyntaxique et direction. Pour l'exemple dans la figure 3.2 nous observons le patron suivant: {*(conj,↑,NNP), (nsubjpass,↑,NNP), (root,-,VBN), (prep,↓,IN), (pobj,↓,NNP), (nn,↓,NNP)*}.

En retenant des *n-grammes*, une sous-séquence de n éléments construite à partir d'une séquence donnée, il est également possible d'obtenir de l'information sur la séquence des différents éléments. Dans ce cas-ci nous avons effectué l'expérience pour n=2 (bigrammes) et n=3 (trigrammes). Toujours en utilisant Apriori tel que spécifié préalablement, nous avons remplacé les contextes liés aux mots dans la phrase par des n-grammes de mots se situant entre les entités au niveau de la phrase pour l'entraînement du filtre. Les entités (requête et réponse) sont également représentées dans les n-grammes (*QUERY* et *FILLER*) afin de fournir l'information quant à la proximité des mots à chacune des entités. Pour l'exemple dans la figure 3.2 nous obtenons le patron suivant (pour n=2): {*QUERY_were, were_both, both_born, born_in, in_FILLER*}. Nous retenons également les n-grammes de catégories morphosyntaxiques entre les entités au niveau de la phrase dans lesquelles les entités sont également représentées. Pour l'exemple dans la figure 3.2 nous obtenons le patron suivant (pour n=2): {*QUERY_VBD, VBD_DT, DT_VBN, VBN_IN, IN_FILLER*}. Nous avons retenu les n-grammes pour les dépendances syntaxiques ainsi que pour les triplets de dépendances syntaxiques, catégories morphosyntaxiques et direction. Pour l'exemple dans la figure 3.2 nous obtenons respectivement les patrons suivants (pour n=2): {*conj_nsubjpass, nsub-*

jpass_prep, prep_pobj, pobj_nn} et {*(conj, ↑, NNP)_(nsubjpass, ↑, NNP), (nsubjpass, ↑, NNP)_(root, -, VBN), (root, -, VBN)_(prep, ↓, IN), (prep, ↓, IN)_(pobj, ↓, NNP), (pobj, ↓, NNP)_(nn, ↓, NNP)*}.

Afin qu'il n'y ait pas de redondance ou de surreprésentation des différents patrons retenus, nous avons remplacé les caractéristiques dérivées des unigrammes en premier lieu par ceux dérivés des bigrammes et ensuite par ceux dérivés des trigrammes. Nous avons testé nos classifieurs pour *Relation Factory*. Les bigrammes permettent d'obtenir de meilleures performances. Ainsi, nous les avons gardés et avons rejeté les trigrammes.

3.6 Description de l'expérimentation

Parmi les 18 systèmes disponibles, nous en avons omis quatre puisque les positions des entités et de la justification dans le texte fourni ne nous permettaient pas d'extraire les requêtes, réponses et justifications adéquatement. L'expérience a été répétée pour chacun des 14 systèmes restants, en retirant la sortie du système testé de l'ensemble d'entraînement tel que mentionné préalablement. La description suivante indique la procédure expérimentale pour un seul système.

Après avoir séparé par relation les données d'entraînement prétraitées, les caractéristiques sont extraits pour chaque instance. Dans certains cas, les caractéristiques ne peuvent pas être extraits puisqu'ils dépendent de la position de la requête par rapport à la réponse dans la justification. C'est le cas lorsqu'une des deux entités ne se trouve pas dans la justification ou que les deux entités ne se trouvent pas dans la même phrase. Le filtre ne traite que les instances pour lesquelles les deux entités (requête et réponse) sont présentes dans la même phrase. Lorsque les deux entités ne sont pas trouvées dans la même phrase, la plupart des caractéristiques ne peuvent pas être extraits. Ainsi, l'instance est retirée de l'ensemble d'entraînement. La procédure pour l'ensemble de test est similaire: le filtre n'est pas appliqué aux instances pour lesquelles les entités ne se trouvent pas dans la même phrase; elles sont retenues par défaut. Le tableau A.5 en annexe présente pour chaque relation le nombre d'instances rejetées de l'ensemble d'entraînement pour Relation Factory. Les instances rejetées correspondent à environ 15% des données et sont inégalement distribuées parmi les relations. Pour les relations telles que *org:website*, les systèmes ne retournent généralement pas des phrases complètes en guise de justification, mais plutôt de courts segments textuels contenant seulement la réponse. Quelques relations spécifiques telles que *per:alternate_names* et *org:alternate_names*, pour lesquelles une justification n'est pas exigée afin d'obtenir une évaluation *correcte* sont responsables pour la plupart des instances rejetées de l'ensemble d'entraînement.

3.7 Classifieurs

Pour chacune des relations, nous avons entraîné une série de classifieurs parmi lesquels le meilleur est appliqué sur les données de test, c'est-à-dire pour filtrer la sortie des extracteurs de relations. Les différents modèles sont sélectionnés parmi les classifieurs de base suivants: *machine à vecteurs de support*, *forêt d'arbres décisionnels*, *arbre bayésien naïf*, *table de décision*, *J48* et *K** [24] (détails dans la section suivante). L'évaluation est effectuée par a) validation croisée sur l'ensemble d'entraînement (10 partitions) et b) validation sur l'ensemble de test. Nous retenons le classifieur présentant le F1 le plus élevé sur la classe *correcte* à l'entraînement. Nous avons sélectionné cette métrique afin de limiter le nombre de faux négatifs produits par le classifieur. Tel que mentionné préalablement, le but du filtre est d'augmenter la précision globale de l'extracteur de relations en limitant la perte de rappel. Une quantité considérable de faux négatifs produits par le filtre résulterait en une grande perte de rappel et serait nuisible au F1 de l'extracteur de relations. Le tableau A.6 en annexe présente pour chaque relation, les performances du classifieur retenu à l'entraînement estimées par validation croisée pour Relation Factory. Pour les relations dont l'ensemble d'entraînement a été rééquilibré, nous entraînons les six classifieurs mentionnés pour chacun des cinq sous-ensembles échantillonnés. Nous retenons la paire classifieur/sous-ensemble qui a le F1 sur la classe *correcte* le plus élevé.

3.8 Algorithmes de classification

La section suivante présente un aperçu de chacun des algorithmes de classification de la librairie Weka [24] utilisés lors de l'expérience: machine à vecteurs de support, forêt d'arbres décisionnels, arbre bayésien naïf, table de décision, J48 et K*.

3.8.1 Machine à vecteurs de support

Une machine à vecteurs de support (*support vector machine* - SVM), dans sa forme la plus simple, est un hyperplan séparant un ensemble d'exemples positifs d'un ensemble d'exemples négatifs avec une marge maximale [47]. Les SVM peuvent être généralisés en des classifieurs non linéaires [8]. La sortie d'un SVM non linéaire est calculée à partir des multiplicateurs de Lagrange selon l'équation suivante: $u = \sum_{j=1}^{N} y_j \alpha_j K(\vec{x}_j, \vec{x}) - b$, où K est une fonction de noyau mesurant la similarité ou la distance entre le vecteur en entrée $\vec{x}$ et le vecteur d'entraînement en mémoire $\vec{x}_j$ et y_i est +1 pour un exemple positif et -1 pour un exemple négatif. La fonction de noyau K peut être, par exemple, une fonction gaussienne ou une fonction polynomiale [12]. Typiquement, l'entraînement de SVM nécessite la résolution d'un

problème d'optimisation quadratique de grande ampleur. L'algorithme SMO permet de fractionner ce problème en plusieurs problèmes d'optimisation quadratique les plus petits possible résolubles analytiquement, permettant ainsi d'éviter une optimisation quadratique numérique gourmande [35]. Nous utilisons l'implémentation par défaut de Weka qui utilise une fonction de noyau linéaire [24]. SMO permet une meilleure adaptation à une quantité grandissante de données par rapport au *chunking*, un autre algorithme permettant l'apprentissage de SVM.

3.8.2 Forêt d'arbres décisionnels

Comme le stipule son nom, l'algorithme de forêt d'arbres décisionnels (*Random Forest*) est une combinaison d'arbres de décision, de manière à ce que chaque arbre dépende de la valeur d'un vecteur aléatoire échantillonné indépendamment et avec la même distribution pour chacun des arbres [10]. En d'autres mots, lors de l'entraînement, chaque arbre est construit à partir d'un nombre fixe de caractéristiques. Nous utilisons l'implémentation par défaut de Weka qui utilise 100 arbres s'appuyant chacun sur $F = int(log_2 M + 1)$ caractéristiques où M est la dimensionnalité de l'ensemble d'entraînement (nombre total de caractéristiques) [24]. La sélection des exemples d'entraînement est faite par *bagging* (*bootstrap aggregation*) ce qui augmente l'exactitude des prédictions lorsque des caractéristiques aléatoires sont sélectionnées [9]. Par ce processus, chaque arbre est appris sur une portion des données et non sur la totalité de l'ensemble d'entraînement, ce qui permet également d'obtenir une estimation de l'erreur de généralisation de l'ensemble des arbres de décision. Cette estimation est obtenue en utilisant, pour chaque exemple, un vote effectué par les arbres qui ne sont pas entraînés sur cet exemple en question.

3.8.3 Arbre bayésien naïf

Comme le sous-entend son nom, l'arbre bayésien naïf (*Naive Bayes Tree*) est un hybride entre un arbre de décision et un classifieur bayésien naïf. Le classifieur bayésien naïf est basé sur une forte hypothèse d'indépendance des dimensions et lorsque cette hypothèse est violée, cela peut avoir un effet néfaste sur l'exactitude de la classification lorsque la taille de l'ensemble d'entraînement augmente [29]. Lors de la construction de l'arbre, ses noeuds sont divisés en se basant sur une seule caractéristique (*univariate split*) et chacune des feuilles de l'arbre représente un classifieurs bayésien naïf [26]. Un classifieur bayésien naïf suppose que l'existence d'une caractéristique pour une classe est indépendante de l'existence d'autres caractéristiques. L'avantage du classifieur bayésien naïf est qu'il nécessite relativement peu de données d'entraînement pour estimer les moyennes et variances des différentes variables, paramètres nécessaires à la classification. L'hypothèse d'indépendance des variables permet

de se contenter de la variance de chacune des variables pour chaque classe sans avoir à calculer de matrice de covariance. L'implémentation par défaut de Weka suppose que les lois de probabilité relatives aux variables sont des gaussiennes [24]. L'espérance, μ, se calcule avec $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$, où N est le nombre d'instances et x_i est la valeur d'une instance donnée. La variance, σ^2, se calcule avec $\sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2$ [26].

3.8.4 Table de décision

La table de décision (*Decision Table*) est construite sur un espace de caractéristiques réduit [28]. Le schéma est défini comme l'ensemble de caractéristiques qui sont incluses dans la table et le corps est défini comme l'ensemble de données annotées projetées dans l'espace défini par les caractéristiques du schéma. Les caractéristiques optimales peuvent être trouvées avec différents algorithmes de sélection de caractéristiques. Le schéma optimal est celui qui minimise l'erreur d'exactitude de prédiction qui est évaluée par validation croisée sur l'ensemble d'entraînement. Lorsque les exemples de test ne peuvent être appariés à aucune entrée dans la table de décision, leur prédiction correspond à la classe majoritaire [28]. Dans le contexte de l'expérience, nous utilisons les paramètres par défaut de l'implémentation de Weka. La sélection de caractéristiques est effectuée en utilisant un algorithme de recherche *best-first* [40] et l'erreur associée à chaque sous-ensemble de caractéristiques est évaluée par validation croisée *leave-one-out* [24]. L'algorithme de recherche *best-first* parcourt un graphe en explorant le noeud le plus prometteur selon une règle spécifique. Dans ce cas-ci, la règle est basée sur le gain en information de l'ensemble de caractéristiques suite à l'ajout d'une caractéristique à l'ensemble.

3.8.5 J48

L'algorithme J48 est l'implémentation de Weka de l'algorithme C4.5 [36, 24]. L'algorithme permet la construction d'un arbre de décision en se basant sur le concept d'entropie de Shannon [36]. À chacun des noeuds de l'arbre, C4.5 sélectionne la caractéristique qui sépare le mieux l'ensemble d'entraînement en sous-ensembles contenant respectivement un maximum d'exemples appartenant à chacune des classes. Ce critère de sélection est le gain en information. Le processus est ensuite répété sur chacun des sous-ensembles de données. L'implémentation par défaut de Weka répète l'exercice, jusqu'à ce que le nombre d'éléments par feuille soit de deux [24]. Toutefois, la complexité de l'arbre est réduite en effectuant l'élagage statistique, en retirant un noeud lorsqu'il ne fournit pas suffisamment de pouvoir de classification, permettant ainsi d'éviter le surapprentissage [36].

3.8.6 K*

L'algorithme K* est un algorithme d'apprentissage par instance (*instance-based learner*). Dans le contexte d'apprentissage par instance, la classe d'une instance de test est basée sur la classe d'instances d'entraînement similaires à celle-ci et est déterminée par une fonction de similarité [17]. Dans cette catégorie, nous retrouvons également les algorithmes d'apprentissage tels que la méthode de k plus proches voisins (*k-nearest neighbors* - kNN), IB1 et IBk [18, 3]. K* se distingue de ces algorithmes par l'utilisation d'une fonction de distance basée sur l'entropie [17]. Cet algorithme permet de manipuler des attributs dont la valeur est réelle, des attributs symboliques ainsi que des instances pour lesquelles des valeurs sont manquantes [42]. L'implémentation par défaut de K* utilise un *blend* $b = 20\%$ [24]. Le blend est échelonné de 0% à 100% et représente une "sphère d'influence" spécifiant le nombre de voisins de l'instance de test qui devraient être considérés importants (plus le blend est élevé, plus le nombre de voisins considérés important est grand) [17].

L'article présenté au chapitre 4 est une version intégrale de l'article publié dans le cadre de la conférence *WWW2016: 25th International World Wide Web Conference*. Il présente une première version du filtre. L'article présenté au chapitre 5 est une version intégrale de l'article soumis à la revue scientifique *Knowledge & Data Engineering* en date du 9 mars 2016. Cet article explique en plus de détails la méthode de filtrage d'extracteurs de relations et présente des résultats exhaustifs de la recherche menée.

CHAPITRE 4 ARTICLE 1: A MACHINE LEARNING FILTER FOR RELATION EXTRACTION

Kevin Lange Di Cesare, Michel Gagnon, Amal Zouaq, Ludovic Jean Louis
Proceedings of the 25th International Conference Companion on World Wide Web

The TAC KBP English slot filling track is an evaluation campaign that targets the extraction of 41 pre-identified relations related to specific named entities. In this work, we present a machine learning filter whose aim is to enhance the precision of relation extractors while minimizing the impact on recall. Our approach aims at filtering relation extractors' output using a binary classifier based on a wide array of features including syntactic, lexical and statistical features. We experimented the classifier on 14 of the 18 participating systems in the TAC KBP English slot filling track 2013. The results show that our filter is able to improve the precision of the best 2013 system by nearly 20% (in percentage points) and improve the F1-score for 17 relations out of 33 considered.

4.1 Introduction

In this paper, we focus on the TAC KBP English slot filling (ESF) track, an evaluation campaign that targets the extraction of 41 pre-identified Wikipedia infobox relations (*title, spouse*, etc.) related to specific named entities (persons and organizations) [44]. A named entity (the query entity) and a relation (the slot) are submitted to the system, which must find every other entity (the filler) that is linked to this entity with this particular relation, and must return a textual segment that justifies this result[1] [44].

In 2013, the top performing system, Relation Factory, achieved a recall of 33.2%, precision of 42.5% and F1 of 37.3% [38]. We propose to complement the output of a relation extraction system with a filtering step. Our aim is to tackle the following research questions:
(1) *Which features can be used for filtering the output of relation extractors?*
(2) *Can we increase the precision of slot filling systems without impacting their F1-score?*

4.2 Methodology

Our approach consists in using a machine learning filter trained on the output of the ESF participating systems. For each relation, a binary classifier is appended to the end of a

[1] Examples of filtered responses, the list of relations and further results are available at: http://westlab.herokuapp.com/extractionfilter/appendix

Table 4.1 Full set of generic features used for filtering

Name	Description
Statistical features	
Sentence length	Number of tokens in the sentence
Query/filler length	Number of tokens within the query and filler (strings in the justification)
Entity order	Order of appearance of query and filler
#tokens left/between/right	Number of tokens left/right or between the query and the filler in the sentence
Confidence score	Score given by the relation extractor [44]
Lexical/POS features	
POS fractions	Proportion of nouns, verbs, adjectives and others (left/between/right/sentence) [32]
POS subsets	Most frequent subsets of POS tags between the query and the filler
Word subsets	Most frequent subsets of words in the sentence (excluding stop-words and named entities)
Syntactic features	
Distance between entities	Distance between the query and the filler in the syntactic dependency tree
Entity level difference	Level difference in the syntactic dependency tree between the query and the filler
Entities are ancestors	The query and the filler are ancestors in the syntactic dependency tree
Syntactic dependencies subsets	Syntactic dependencies in the syntactic dependency tree between the query and the filler
Multilevel subsets	{POS tag, syntactic dependency & relation} tuples in the syntactic dependency tree

relation extractor's pipeline. It eliminates responses which it classifies as *wrong* with the goal to improve its precision. The filter cannot increase recall scores as it does not generate additional responses.

Responses from the relation extractors are passed to the part-of-speech (POS) tagger and syntactic analysis parser which return the POS tags and the syntactic dependency tree [19]. Our approach is based on a wide selection of generic features which are listed in Table 4.1 ranging from statistical, lexical to syntactic features. As we are using syntactic-based features, we only focus on intra-sentence relations. To enhance our classifier, we captured the context between the query entity and the filler at the lexical, the morpho-syntactic and the syntactic levels. The contexts are extracted from the raw justification and from the POS/syntactic analysis output. To limit the number of contexts, we retained the most frequent subsets of part-of-speech tags between the query entity and the filler for each class (*correct/wrong*) by using the Apriori algorithm after separating *wrong* instances from *correct* ones, for each relation [2]. The Apriori algorithm allows us to retain the desired subsets with reduced computational resources, by first identifying individual items, in this case a single POS tag, for which the support is greater than the minimum support and second, by extending them to larger item sets by retaining only the subsets which have a support greater than a minimum support, 15% in this case. Similarly, we used the Apriori algorithm to retain the most frequent subsets of words within the sentence (excluding stop-words & named entities) as well as syntactic dependencies and {POS tag, syntactic dependency & relation} tuples within the syntactic dependency tree. Each retained subset is translated into a boolean feature indicating the presence of the subset. For each relation, we experimented various

classifiers and kept the one with the highest F1 score on the *correct* class. Models are selected from one of the following base classifiers: RandomForest, SMO, NBTree, DecisionTable, J48 and K* [24]. The evaluation is done on the training set using a 10-fold cross-validation. To avoid over-fitting, we do not filter relations for which the number of training instances is inferior to 15 for either class (*correct/wrong*). To prevent favoring the *wrong* class, for each relation where the number of *wrong* instances is more than double the number of *correct* instances, we down-sample the subset of *wrong* instances to five subsets randomly selected. Each subset contains the same number of instances as the *correct* class subset. For relations which have been rebalanced, we train six classifiers for the five subsets of the majority class and retain the classifier/subset pair which has the greatest F1 score on the *correct* class.

4.3 Experiments & Evaluation

We excluded 4 systems out of 18 as the offsets they provided did not allow us to extract the justifications correctly. Our training data is obtained by merging all systems outputs. The experiment was executed for each of the 14 systems by holding out its own output from the training data. Our filter increases the precision score for every tested relation extractor compared with their original performance. The filter results in an average increase in precision of 8.8% (in percentage points) for the 14 systems[2]. We tested different combinations of features. Table 5.6 reports our results. We used as baseline Relation Factory's highest F1 run, on which we applied our filter for all the feature sets. We also consider its highest precision run for comparison. The combination of statistical, lexical/POS and syntactic features render the greatest performance: precision increases from 42.5% to 61.6% for a 19.1% increase. The F1 is increased from 37.2% to 37.7%. The precision achieved is also 10.7% higher than Relation Factory's highest precision run. From the 33 filtered relations, the precision was increased for 26 relations and F1 was increased for 17 relations. Following the filtering, F1 and precision were decreased for only four relations. Performances could not be measured for three relations due to a lack of test data.[3]

4.4 Conclusion

We present a generic method to increase the precision of relation extractors by appending a machine-learning-based filter to the relation extractor's pipeline. The filter utilizes a wide

[2]See footnote 1
[3]See footnote 1

Table 4.2 Results obtained by Relation Factory after filtering when using different feature sets

Feature Set	**R**	**P**	**F1**
Relation Factory (best F1)	0.332	0.425	0.373
Relation Factory (best precision)	0.259	0.509	0.343
Statistical	0.256	0.574	0.354
Statistical + Lexical/POS	0.266	0.614	0.371
Statistical + Syntactic	0.253	0.582	0.352
Statistical + Lexical/POS + Syntactic	**0.271**	**0.616**	**0.377**

scope of features, including statistical, lexical/POS and syntactic features. The features used to train the classifiers are generic and could be used for classifying any pre-defined relation. Our filter has allowed an increase in precision of nearly 20% (in percentage points) for the best performing system in the 2013 slot filling track. We believe that the performance of the filter can be improved by exploiting more training samples of better quality. We are working on improving the features using various feature selection approaches.

4.5 Acknowledgments

This research was partially funded by the Industrial Innovation Scholarships Program.

CHAPITRE 5 ARTICLE 2: A MACHINE LEARNING FILTER FOR THE SLOT FILLING TASK

Kevin Lange Di Cesare, Amal Zouaq, Michel Gagnon, Ludovic Jean Louis
Data & Knowledge Engineering

Slot Filling, a subtask of Relation Extraction, represents a key aspect for building structured knowledge bases usable for semantic-based information retrieval. In this work, we present a machine learning filter whose aim is to enhance the precision of relation extractors while minimizing the impact on the recall. Our approach consists in the filtering of relation extractors' output using a binary classifier. This classifier is based on a wide array of features including syntactic, semantic and statistical features such as the most frequent part-of-speech patterns or the syntactic dependencies between entities. We experimented the classifier on the 18 participating systems in the TAC KBP 2013 English Slot Filling track. The TAC KBP English Slot Filling track is an evaluation campaign that targets the extraction of 41 pre-identified relations (e.g.*title*, *date of birth*, *countries of residence*, etc.) related to specific named entities (*persons* and *organizations*). Our results show that the classifier is able to improve the global precision of the best 2013 system by 20.5% (in percentage points) and improve the F1-score for 20 relations out of 33 considered.

5.1 Introduction

In the age of structured knowledge bases such as Google Knowledge Graph [43], DBpedia [7] and the Linked Open Data cloud [6], relation extraction is becoming a very important challenge for enhanced semantic search. Relation extraction and its sub-task, slot filling, have been very active in recent years and have been subject to several evaluation campaigns that assess the ability of automatically extracting previously known relations from corpora. Despite some progress, the results of these competitions remain limited. In this paper, we focus on the TAC KBP English Slot Filling (ESF) track which is an evaluation campaign that targets the extraction of 41 pre-identified Wikipedia info-box relations (e.g. *title*, *date of birth*, *countries of residence*, etc.) related to specific named entities (*persons* and *organizations*) [44]. In this task, a named entity (*the query entity*) and a relation (*the slot*) are submitted to the system, which must find every other entity (*the filler*) that is linked to this entity with this particular relation, and must return a textual segment that justifies this result [44, 46]. The goal of this task is to populate a reference knowledge base, in this case a Wikipedia

infobox 2008 snapshot [21], for which fillers to a pre-defined list of relations for 100 selected named entities are missing. In 2014, the top performing system, submitted by Stanford, achieved a recall of 0.277, a precision of 0.546 and a F1 score of 0.368 [46, 4]. These results are somewhat similar to those submitted the previous year by the best performing system, Relation Factory, which achieved a recall of 0.332, precision of 0.425 and F1 score of 0.373 [44, 38].

For most systems, the slot filling task is done in two main steps: candidate generation and candidate validation [38]. The candidate generation stage can be further decomposed into the following steps: i) query expansion, ii) document retrieval and iii) candidate matching. Query expansion is performed by finding alternate reference forms that designate the named entity contained in the query. For example, the entity *Barack Obama* could be referenced as *President Obama*, *Barack Hussein Obama* or simply *Obama*. These alternative forms constitute the named entity expansion. The system then retrieves documents containing a reference to the query entity. The retained documents are annotated with a name entity recognizer which associates a named-entity tag (or the tag *OTHER* if the word is not a named entity) to each word. In the candidate matching stage, candidates, e.g. sentences containing a reference to the query entity or to some item of its expansion, along with a named-entity tag of a potential slot filler type for a given relation, are retrieved [38]. For example, the sentence *Barack Obama was born in Honolulu, Hawaii in 1961.* would be tagged by the named entity tagger as follows: "*Barack/PERSON Obama/PERSON was/OTHER born/OTHER in/OTHER Honolulu/GPE-CITY*, *Hawaii/GPE-STATE in/OTHER 1961/DATE*." This sentence would be a candidate for the relation *city of birth* and the query entity *Barack Obama* since it contains a named-entity tag of type *GPE-CITY*. For each relation, the candidates are passed to the validation stage. This system-dependent step is usually based on distant supervision (DS) classifiers and/or hand-crafted patterns, among other approaches. Distant supervision is a learning technique which doesn't require labeled data. In the context of slot filling, DS requires a knowledge base and a large text corpus. For a given relation, entity pairs are extracted from the knowledge base and are matched to a text corpus from which a number of sentences containing each entity pair are retained as positive training data [32].

A system's recall can be increased significantly by a configuring a more lenient candidate generation step or candidate validation step. However, its precision is then expected to drop. This is a well-known issue in information retrieval where there is a trade-off between precision and recall. Our research describes a filtering module that could be applied to the output of relation extractors in order to increase their precision. It is driven by the following research questions:

RQ1: What are the most important features for the filtering step?

RQ2: Is there a generic method to increase the precision of slot filling systems without significantly degrading their recall (overall performance across several relations)?
RQ3: Are some relations more sensitive to our filter?

The paper is organized as follows. In section 5.2, we present a brief overview of related work in the field of relation extraction and slot filling. In section 5.3, we explain our approach, describing the dataset and the set of features used to train our classifiers. We compare the evaluation of systems' for the TAC ESF track before and after filtering, measuring the impact of the feature sets in section 5.4. In section 5.5, we discuss our motivation to focus on systems' precision along with other aspects of our work. We also give a brief conclusion and discuss future work.

5.2 Related Work

In section 5.2.1, we give a brief overview of the most common and most successful approaches for relation extraction. In section 5.2.2, we focus on relation extractors' enhancement and present different methods to increase a relation extractor's performance.

5.2.1 Relation Extraction

Relation extraction being a prevailing concern, multiple research groups have worked on the task lately. Relation extraction and slot filling are complex tasks which require rich modular automatic systems. Typical methods such as supervised relation extraction encounter certain problems. Primarily, there is very few labeled data available for training such systems since it is expensive to produce. Also, there is generally a text or domain bias since relations are labeled on a particular corpus [25]. Alternatively, unsupervised relation extraction approaches use very large amounts of data and can extract a wide scope of relations. However those relations are not necessarily easy to map to a knowledge base [22]. Therefore, modern systems (eg. [38], [4], [34]) use different techniques such as pattern bootstrapping and distant supervision, which do not require labeled datasets but extract pre-defined relations.

Bootstrap learning uses a small number of seed instances, which are entities known to fill a given relation [11, 1]. The seed pairs are used to extract patterns from a large corpus, which are then used to extract more seed entity pairs to generate additional seed patterns in an iterative manner. Depending on the pattern types and other factors, the resulting patterns often suffer from low precision because of semantic drift [32]. Some approaches rely on dependency parsing patterns. For example, [30] uses bootstrapping to generate syntactic

dependencies patterns. The authors train classifiers by parsing the dependency tree and selecting all paths containing less than 3 vertices between the query and the candidate filler. Their system achieved a recall of 0.223, a precision of 0.199 and a F1 of 0.21 for the 2014 TAC ESF task for which the median F1 was 0.198.

Most systems throughout different evaluation campaigns use distant supervision to acquire data to train their classifiers. This approach, briefly described in the previous section, seems to work better than bootstrapping, since most systems that use distant supervision scored higher in the ESF task in 2013 and 2014 [46]. For example, [38] uses distant supervision support vector machine classifiers for candidate validation. Distant supervision argument pairs are obtained by mapping Freebase relations to TAC relations and by matching hand-crafted seed patterns against the TAC 2009 text collections [39]. [38] uses a maximum of 10k argument pairs per relation for each of the two sets of seed pairs that are then matched against the TAC 2009 text corpora. A maximum of 500 sentences per pair are used as training data. Though the quality of generated examples is inferior to the quality of examples generated using bootstrapping techniques, the number of examples is much greater. In order to increase the performances of classifiers trained on noisy false positive sentences, the authors use aggregate training. Instead of treating each sentence as a single training example, they group all sentences per entity pair, extract the features, sum the feature counts of all the sentences and normalize the feature vector for that pair so that the highest feature has weight 1.0. While increasing training speed, this method reduces the influence of features that are highly correlated with a single distant supervision pair. Their system, Relation Factory, scored the highest in the 2013 task.

5.2.2 System Enhancement

Rather than developing whole systems, it is also common for research groups to look to enhance existing relation extractors since there is a great margin of potential improvement. [15] developed a Maximum Entropy based supervised re-ranking model to re-rank candidate answers for the same slot. The model is trained on labeled query-answer pairs to predict the confidence of each answer's correctness. The authors used features based on confidence assigned by OpenEphyra, a web module of an open domain QA system [41], co-occurrence in the document collection, named-entity tags and slot type. When tested on the 2009 TAC ESF task, this approach achieved a 3.1% higher precision and a 2.6% higher recall compared to the best performing system. [33] worked on a semantic rule filter. It is an unsupervised, scalable learning method for automatically building relation-specific lexical semantic graphs representing the semantics of the considered relation. The authors compare approaches using

WordNet and BabelNet. The approach using BabelNet yields a considerable precision gain with only a slight loss of recall. This approach worked best on relations of type *acquisition* and type *birth/death* and struggled for relations of type *place of residence*, which have a larger lexical diversity. It approach was tested on only seven relations. [5] worked on an active learning module which combines supervised learning with distant supervision. By providing only a small sample of annotated data, they have increased the F1 of Stanford's 2013 slot filling system on the 2013 ESF task by 3.9.

All these enhancement methods are incorporated within the pipeline of an individual system and aim, in most cases, at a particular system or work best for particular relations. In contrast, this paper describes a generic approach which can be easily integrated to any relation extractor and which can be trained on any relation, regardless of the domain. An alternative to single system enhancement is ensemble learning. In an incentive to encourage the developpement of ensemble learning techniques, TAC also proposes the Slot Filler Validation (SFV) task, which consists in refining the output of ESF systems [45]. [49] has developped a system participating in the TAC 2013 SFV task that uses an aggregation approach. For a given query, it aggregates "raw" confidence scores produced by individual slot fillers to produce an aggregated confidence score. This value is obtained based on the number of occurences of a response and the confidence score for each occurence. A variant of their method assigns a weight to each occurence based on the "general confidence" of the system that submitted the response. Their approach achieves great performances for single-value slots (Precision: 0.797, Recall: 0.794, F1: 0.795) but does not perform as well for list-value slots. Similarly, [37] has developped a system using *stacking* which trains a classifier to optimally combine the results of multiple systems. Their system combines the output of various systems in the TAC ESF task using a "meta-classifier" and using the systems' confidence score. The authors also derive features from document provenance and offset overlap, i.e the proportion of characters which are common between justifications of every response submitted for the slot. For a given query/relation pair (slot), if N systems provide answers and n of those systems provide the same *docid* as provenance, then the provenance score associated with this *docid* is equal to n/N. Similary, they use the Jaccard coefficient to measure the offset reliability of a response relatively to other responses for the same slot. Similary to Relation Factory [38], they use an expansion method based on Wikipedia anchor text applied on the filler, to detect redundant responses, thus increasing precision. This method allows them to achieve a F1 of 50.1% on the 2014 SF submissions which is greater than the best performing system (39.5%).

In general, ensemble methods achieve better performances compared to individual systems. However they require accessing multiple relation extractors. This paper describes a more

light-weight approach and focuses on increasing precision while limiting loss of recall.

5.3 Methodology

We start by introducing our general framework in section 5.3.1. We then present the input and expected output of the TAC KBP ESF track in section 5.3.2. In sections 5.3.3 and 5.3.4, we explain the preprocessing on our dataset. In section 5.3.5, we present the features used for filtering. Finally, we present an experiment description in section 5.3.6 and our classification method in section 5.3.7.

5.3.1 General Framework

Our approach consists in using a machine learning filter trained on the output of the TAC ESF participating systems. For each relation, a binary classifier filter is appended to the end of a relation extraction pipeline, from where it eliminates responses which it classifies as *wrong*. Figure 5.1 shows the general data flow in which justifications from relation extractors are preprocessed and from which features are extracted for classification. This approach aims at improving the precision of a relation extractor. An increase in recall is not possible because there are not any additional responses generated by our filter. The output from the relation extractors which contains a justification sentence for each response are passed to the named-entity recognizer (NER), which gives the NE tags for each word in the justification sentence (*OTHER* if the word is not a named entity). The output from the relation extractors is also passed to the POS tagger and syntactic analysis parser, which return the POS tags as well as the syntactic dependencies tree (details in section 5.3.3). Most frequent patterns are extracted from the raw justification sentence as well as the POS/syntactic analysis output. Sentence level and syntactic dependency tree level frequent patterns and the preprocessed responses are used for feature extraction (details in section 5.3.5). The resulting feature vectors are then passed to the binary classifier for training. Test data from the tested relation extractor's output, containing once again, justification sentences, go through the NER, tagging and parsing process. The same features are extracted and the binary classifier retains or rejects the responses based on its classification model.

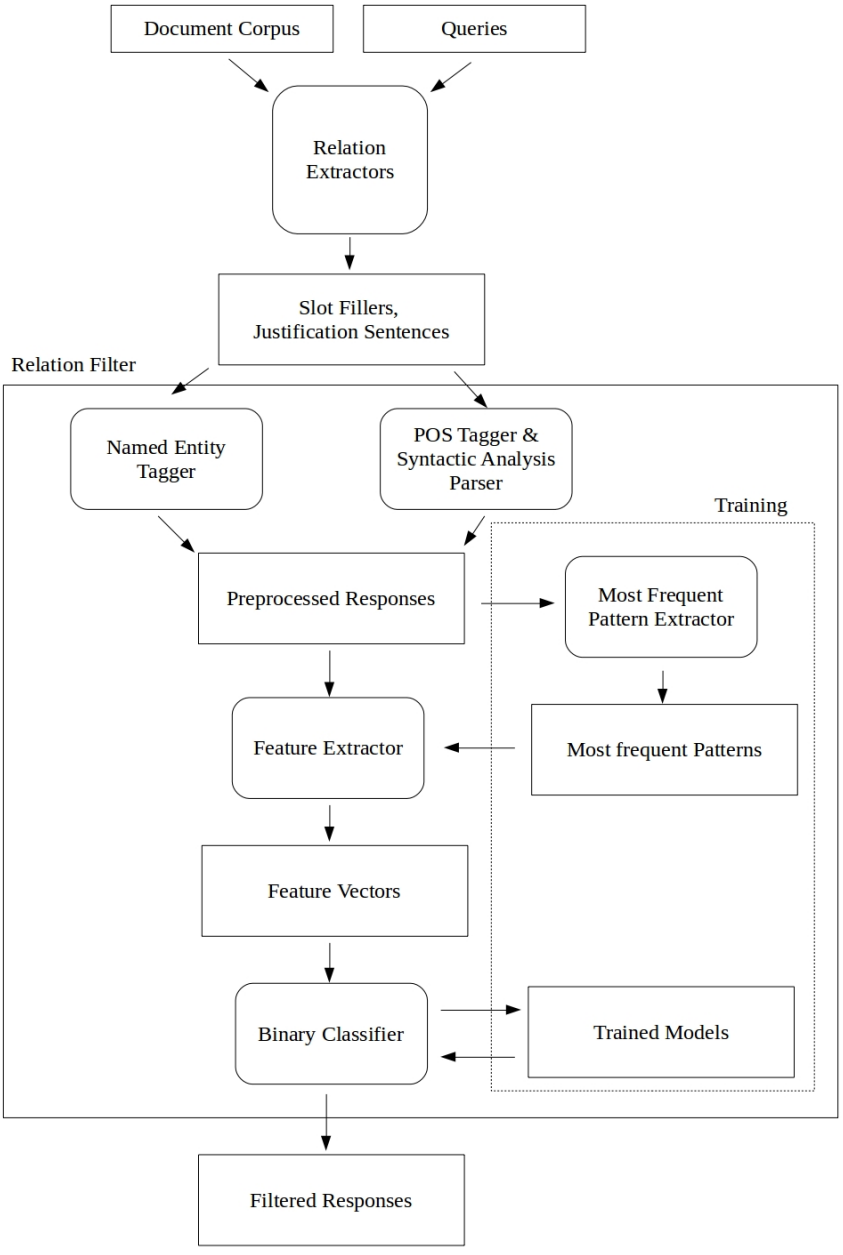

Figure 5.1 Pipeline of our approach

5.3.2 Dataset

5.3.2.1 The TAC KBP English Slot Filling Dataset

As mentioned previously, the dataset is composed of responses from relation extractors to queries related to 41 pre-defined relations. Table 5.1 shows a complete list of the relations for the TAC KBP 2013 ESF task. There are 25 relations expected to be filled for requests of entity type *person* and 16 relations for requests of entity type *organization* [20]. Responses are assessed as either *correct* if they satistfy the assessment validity criterion and *wrong* elsewise (details in section 5.3.2.2). The table also shows the distribution of the complete training dataset, comprising the preprocessed responses of each relation extractor's best run after the dataset is cleaned (details in section 5.3.2.3). The amount of training data varies widely across relations, primarily because some relations expect multiple values whereas others only expect a single value as filler, and secondly because relation extractors have more ease in generating and retaining candidates for certain relations compared to others. Another important fact to consider is the class imbalance towards the *wrong* class in most relations.

5.3.2.2 System Output

The relation extractors were provided a list of 100 queries which consisted of a named entity and a list of relations to fill [44]. Figure 5.2 shows an example of a query. The XML query contains the query ID, the named entity, the ID of a document in which an instance of the entity can be found, and the beginning and end character offset of this instance in the document. The query also contains the entity type (*organization* or *person*) indicating which relations have to be extracted. If the entity is already in the knowledge base, it also contains a node ID and a list of slots that must be ignored (because they are already available in the knowledge base). In response to the input queries, systems' submissions were required to provide the following information in each individual answer: the query ID, the ID of the document that contains the answer, the name, string or value that is related to the relation specified in the query (we will call this the slot filler), the offset of the query entity in the document, the offset of the slot filler, the beginning/end character offset of the justification text, and, finally, a confidence score [44]. For a given query, slots are expected to be filled for all relations associated with the entity type (except for relations to be ignored). Some relations accept multiple answers, e.g. *alternate_names*, *children*, *title*, etc. Name slots are required to be filled with the name of a person, organisation or geo-political entity; value slots are required to be filled by either a numerical value or a date; string slots are a "catch all" meaning that their fillers cannot be classified as names or values [20]. All

Table 5.1 List of 41 pre-defined relations in TAC KBP 2013 English Slot Filling track and their categorization [20]

Type	**Relation**	**Content**	**Quantity**	**#Correct**	**#Wrong**	**Total**
ORG	org:alternate_names	Name	List	100	157	257
ORG	org:city_of_headquarters	Name	Single	62	118	180
ORG	org:country_of_headquarters	Name	Single	73	114	187
ORG	org:date_dissolved	Value	Single	0	15	15
ORG	org:date_founded	Value	List	36	48	84
ORG	org:founded_by	Name	List	49	127	176
ORG	org:member_of	Name	List	5	195	200
ORG	org:members	Name	List	49	195	244
ORG	org:number_of_employees_members	Value	Single	19	28	47
ORG	org:parents	Name	List	34	270	304
ORG	org:political_religious_affiliation	Name	List	1	39	40
ORG	org:shareholders	Name	List	15	255	270
ORG	org:stateorprovince_of_headquarters	Name	Single	47	76	123
ORG	org:subsidiaries	Name	List	46	259	305
ORG	org:top_members_employees	Name	List	392	612	1004
ORG	org:website	String	Single	79	17	96
PER	per:age	Value	Single	226	22	248
PER	per:alternate_names	Name	List	58	114	172
PER	per:cause_of_death	String	Single	127	110	237
PER	per:charges	String	List	58	149	207
PER	per:children	Name	List	169	202	371
PER	per:cities_of_residence	Name	List	71	356	427
PER	per:city_of_birth	Name	Single	44	56	100
PER	per:city_of_death	Name	Single	105	97	202
PER	per:countries_of_residence	Name	List	77	164	241
PER	per:country_of_birth	Name	Single	11	23	34
PER	per:country_of_death	Name	Single	26	50	76
PER	per:date_of_birth	Value	Single	63	22	85
PER	per:date_of_death	Value	Single	123	124	247
PER	per:employee_or_member_of	Name	List	257	554	811
PER	per:origin	Name	List	120	320	440
PER	per:other_family	Name	List	19	184	203
PER	per:parents	Name	List	80	167	247
PER	per:religion	String	Single	9	44	53
PER	per:schools_attended	Name	List	78	93	171
PER	per:siblings	Name	List	43	154	197
PER	per:spouse	Name	List	169	266	435
PER	per:stateorprovince_of_birth	Name	Single	27	48	75
PER	per:stateorprovince_of_death	Name	Single	57	62	119
PER	per:statesorprovinces_of_residence	Name	List	50	176	226
PER	per:title	String	List	888	1277	2165
	Total			3962	7359	11321

Content: filler type (Name, Value or String), Quantity: number of fillers expected for the relation (Single or List), #Correct: number of *correct* responses, #Wrong: number of *wrong* responses, Total: number of responses

answers come from the LDC Corpus entitled "the TAC 2013 KBP Source Corpus" (catalog ID: LDC2013E45) composed of 2.1M documents: 1M newswire documents from Gigaword, 1M web documents and 100K documents from web discussion forums [44]. Systems were also required to submit between 1 and 5 system runs for which the system configuration could vary. Each run is annotated by the Linguistic Data Consortium (LDC) assessors who classify each response as *correct* if the slot filler and its justification are accurate and *wrong* elsewise. For our experiment we used the data from the evaluation queries for the TAC KBP 2013 Slot Filler Validation (SFV) track, which are composed of the annotated ESF submissions. The complete dataset utilized for our experiment is the pool of annotated data made of each participating systems' best run, i.e. the run which achieved the highest F1 score.

5.3.2.3 Dataset Cleaning and Partition

As mentioned, the TAC KBP English Slot Filling task only requires systems, for every output, to provide the query entity, the slot filler and justification, the ID of the document where they are found and the beginning/end character offset of the filler, the query and the justification in this document. The document ID and offsets constitute the data that is used to train and test our classifiers and are in some cases noisy, since some relation extractors provide erroneous offsets. For example, although some source documents are in HTML, some systems do not consider HTML tags in their computation of offsets. Therefore, the justification cannot be properly recovered. To ensure the cleanliness of the data, such outputs are removed from the training set. Responses containing a justification with a character length greater than 600 are removed from the dataset as well to ensure a smooth usage of the Stanford Parser. We removed HTML tags from responses in the query entity, the slot filler, or the justification. We also removed periods, question marks and exclamation marks from responses which contained such punctuation at the beginning of their justification.

For a given system, we hold out its own output from the training set and use it as test data in order to be able to compare the system's initial performance to its performance after filtering using the official evaluation script. This means no actual data coming from the system is used in the training set. The evaluation script calculates the global recall, precision and F1 by considering only the slots for which the system submitted a response (including *NIL* which indicates no response). We have also experimented segmentation where 75% of slots were used for training and 25% of slots were held for the test dataset. This enables us to include a portion of the tested system's output in the training data. However we can only test the filter's performance on 25% of the system's output. We have not noticed a significant increase nor decrease in performance for the tested slots compared to the previous segmentation. We

```
<query id="SF13_ENG_010">
  <name>Ronnie James Dio</name>
  <docid>NYT_ENG_20090629.0075</docid>
  <beg>4016</beg>
  <end>4031</end>
  <enttype>PER</enttype>
  <nodeid>E0654593</nodeid>
  <ignore>per:age per:date_of_birth per:city_of_birth per:stateorprovince_of_birth
per:country_of_birth</ignore>
</query>
```

Figure 5.2 TAC ESF query example [44]

therefore make the assumption that there is no significant variability between outputs of different systems affecting the filter's performance and proceed using the former train/test segmentation to be able to compare the systems' performances after filtering to their original performance.

5.3.3 Linguistic Processing of Justifications

The various features require the justification to be annotated with part-of-speech tags as well as named-entity tags. The syntactic dependency tree is also necessary for feature extraction. Systems are allowed to submit up to two sets of character offsets within the specified document to designate query entity, slot filler and justification. Therefore, since most features depend on the query entity and the slot filler's position in the justification, we must ensure a mapping between these positions and the justification sentence. This process is necessary especially in cases where there are multiple references to either the query entity or the slot filler in the justification or if either entity is designated by a pronoun. We used the Stanford Parser to obtain the part-of-speech annotated justification as well as the syntactic dependencies [19]. The named-entity annotations are obtained by applying Sequor, a perceptron named-entity tagger which also considers special entity types such as *job titles* [16]. For this reason we have chosen to use Sequor instead of more known named-entity recognizers such as the Stanford Named Entity Recognizer [23] which does not consider some special entity types which are useful for certain relations. Figure 5.3 shows an example of a justification sentence for which the named-entity tags, the part-of-speech tags and the syntactic dependency tree

Relation:	**Query:**	**Filler:**
City of birth	Barack Obama	Honolulu

Raw justification sentence:

Nicole Kidman and **Barack Obama** were both born in **Honolulu**, Hawaii.

Named-entity tags:

Nicole/*B-PERSON* Kidman/*I-PERSON* and/*O* **Barack/*B-PERSON*** **Obama/*I-PERSON*** were/*O* both/*O* born/*O* in/*O* **Honolulu/*B-GPE:CITY*** ,/*O* Hawaii/B-GPE:STATE_PROVINCE ./*O*

Part-of-speech tags:

Nicole/*NNP* Kidman/*NNP* and/*CC* **Barack/*NNP*** **Obama/*NNP*** were/*VBD* both/*DT* born/*VBN* in/*IN* **Honolulu/*NNP*** ,/, Hawaii/NNP ./.

Syntactic dependency tree:

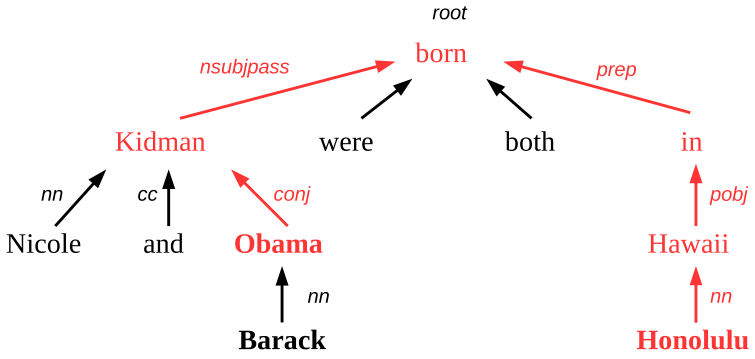

Figure 5.3 Example of NE tags, POS tags and syntactic dependency tree for a justification sentence

have been extracted.

5.3.4 Down-Sampling and Selective Filtering

In order to avoid over fitting, we do not filter relations for which the number of instances is inferior to 15 for either class (*correct/wrong*). Due to the lack of available examples the following relations were omitted from the filtering process for all systems: *org:date_dissolved*, *org:member_of*, *org:political_religious_affiliation*, *per:county_of_birth*, *per:religion*. Data imbalance has been shown to cause classifiers to favor the majority class [13]. For most relations, there are more *wrong* instances than *correct* instances. Therefore, for each relation where the number of *wrong* instances is greater than 2 times the number of *correct* instances, we down-sample the subset of *wrong* instances to 5 subsets randomly sampled containing the same number of instances as the *correct* class subset. The usage of these 5 subsets will be further detailed in Section 5.3.7. The training dataset was rebalanced for the following relations for most systems: *org:founded_by*, *org:members*, *org:parents*, *org:subsidairies*, *per:charges*, *per:cities_of_residence*, *per:countries_of_residence*, *per:employee_or_member_of*, *per:origin*, *per:other_family*, *per:parents*, *per:siblings*, *per:statesorprovinces_of_residence*. The relation *org:shareholders* was rebalanced for 8/14 systems and eliminated for 6/14 systems. Other relations were either rebalanced or eliminated for few systems.

5.3.5 Features Identification

The main goal of our approach is to experiment various sets of features and ultimately propose a relation-independent selection of features. In this section, we present a set of generic features used in our experiment.

Our approach is based on a wide selection of generic features ranging from statistical, named-entity, lexical to syntactic features. These features are listed in Table 5.2.

Statistical features. Our filter utilizes generic features such as sentence length (C1), the number of words of both the query entity and the slot filler (two features) (C2), and their order of appearance in the justification sentence, either query or filler first (C3). Another set of features is based on the segmentation of the sentence according to the position of the query entity and the slot filler, resulting for most sentences in three segments (before, between and after the query entity and slot filler). For each sentence segment, we capture the number of tokens (including words and punctuation) (C4). Since the training data is directly derived from relation extractors that participated in the TAC KBP English Slot Filling task, and given that the task requires every submitted response to be accompanied by a confidence

Table 5.2 Full set of generic features used for filtering

ID	Name	Description
Statistical features		
C1	Sentence length	Number of tokens in sentence
C2	Answer/query length	Number of tokens within answer and query references
C3	Entity order	Order of appearance of query and answer references
C4	#tokens left/between/right	Number of tokens left/right or between entities in the sentence
C5	Confidence score	Score given by the relation extractor [44]
Named-entity features		
N1	#person left/between/right	Number of person left/right or between entities in the sentence [48]
N2	#gpe left/between/right	Number of Geo-political entities left/right or between entities in the sentence [48]
N3	#orgs left/between/right	Number of organizations left/right or between entities in the sentence [48]
Lexical (POS) features		
L1	POS fractions left/between/right/sentence	Fraction of nouns, verbs, adjectives and others left/right/between answer and query references or in the whole sentence [32]
L2	POS subsets	Most frequent subsets of POS tags between query and answer references in the sentence. (boolean feature indicating the presence of the subset)
L3	Word subsets	Most frequent subsets of word (excluding stop-words and named entities) in the sentence. (boolean feature indicating the presence of the subset)
L4	POS bigram subsets	Most frequent subsets of POS tag bigrams (excluding stop-words) between query and answer references in the sentence. (boolean feature indicating the presence of the subset)
L5	Word bigram subsets	Most frequent subsets of word bigrams between query and answer references in the sentence. (boolean feature indicating the presence of the subset)
Syntactic features		
S1	Distance between entities	Distance between entities at the syntactic dependency tree level
S2	Entity level difference	Level difference within syntactic dependency tree between query and answer references
S3	Ancestors	One entity is an ancestor of the other at the syntactic dependency tree level
S4	Syntactic dependency subsets	Most frequent subsets of syntactic dependencies between query and answer references at the syntactic dependency tree level (boolean feature indicating the presence of the subset)
S5	Multilevel subsets	Most frequent subsets, where each token is composed of a POS tag, syntactic dependency and direction, between query and answer references at the syntactic dependency tree level (boolean feature indicating the presence of the subset)
S6	Syntactic dependencies bigram subsets	Most frequent subsets of syntactic dependencies bigram between query and answer references at the syntactic dependency tree level (boolean feature indicating the presence of the subset)
S7	Multilevel bigram subsets	Most frequent subsets, where each token is composed of a POS tag, syntactic dependency and direction bigram, between query and answer references at the syntactic dependency tree level (boolean feature indicating the presence of the subset)

score, we exploit this score as a feature (C5).

Named entity features. This subset of features captures the location of named entities in the justification relative to the query and the filler for the following types: person (N1), geo-political entity (N2) and organization (N3) [48]. Similarly to the feature C4, this set of features is based on the segmentation of the sentence according to the position of the query entity and the slot filler. For each sentence segment, we capture the number of named entities of each of the specified types, resulting into three features for each entity type. For example, as shown in figure 5.3, given the relation *city of birth*, in the following justification, "*Nicole Kidman and Barack Obama were both born in Honolulu, Hawaii.*", where the query entity is *Barack Obama* and the slot filler is *Honolulu*, the sentence would be segmented in the following way: 1) *Nicole Kidman and* 2) *were both born in* 3)*, Hawaii.* The first segment contains one entity of type person (*Nicole Kidman*), the second segment does not contain any named entity and the final segment contains one named entity of type geopolitical entity (*Hawaii*).

Lexical (POS) features. Our filter also relies on a subset of features that captures the ratio of part-of-speech tags in each sentence segment and in the whole sentence. The part-of-speech tags are generalized into 4 categories: nouns, verbs, adjectives and other (L1) [48]. If we take figure 5.3 for example, the distribution of POS tags for the sentence segment before the query entity ("*Nicole/noun Kidman/noun and/other*") is the following: nouns: 2/3, verbs: 0/3, adjectives: 0/3, other: 1/3. We also extracted the most frequent subsets of part-of-speech tags between both entities at the sentence level in the training dataset(L2). This is done using the Apriori algorithm after separating the instances of each relation by class (*correct* and *wrong*) [2]. We used a support of 0.15, meaning that the subset must be present in at least 15% of data related to the wanted class. The Apriori algorithm allows to retain the desired subsets with reduced computational resources, by first identifying individual items, in this case a single POS tag, for which the support is greater than the minimum support and extending them to larger item sets by retaining only the subsets which have a support greater than the minimum support, 15% in this case. In order to ensure the specificity of the previously withheld subset, a subset is only retained if its support on the opposite class is equal or less than 0.5 of its support on the wanted class (we will call this value *specificity*), which gives us a minimum specificity of 2. These parameters were determined experimentally. In order to maintain a reasonable training execution time, we have limited the number of retained subsets to 100. As long as the number of retained subsets is greater than the limit, we rerun the Apriori algorithm with a 2% support increment and a 10% specificity increment. The number of retained subsets ranges from 1 to 100 in the 41 relations. We derive boolean features that indicate the presence or absence of each subset between the query and the

filler at the sentence level. A similar method is used to extract the most frequent word subsets contained in the whole sentence (bag of words)(L3). Stop words, named-entities and punctuation are excluded from potential subsets. Figure 5.3 shows an example of a justification sentence for which the query is *Barack Obama* and the filler is *Honolulu*. In this case the word set is the following : {*born*}. The other words between the query and the filler are either named entities or stop-words and are therefore not retained. Since word subsets are more specific to responses compared to POS subsets, we used Apriori with more lenient parameters to retain the most frequent subsets in the training data: support 10%, specificity 1.5, limit of retained subsets 200, support increment 1% and specificity increment 0. Similarly, we extract POS tag bigram subsets and word bigram subsets between the query and the filler and retain the most frequent subsets using the Apriori algorithm once again (L4 & L5). We also add two additional tokens representing respectively the query and the filler to the bigram subset. In order to retain a sufficient amount of subsets we use the following settings: support 5%, specificity 1.5, limit of retained subsets 200, support increment 1% and specificity increment 0. The word bigram subset is the following for the example shown in figure 5.3: {*QUERY_were, were_both, both_born, born_in, in_FILLER*}. The POS bigram subset is constituted of the corresponding POS tags: {*QUERY_VBD, VBD_DT, DT_VBN, VBN_IN, IN_FILLER*}.

Syntactic features. These features are based on the syntactic dependency tree obtained by parsing the justification sentence. The distance between both entities, i.e. the query and the filler (S1), is the number of links that separate both entities in the syntactic dependency tree. The entity level difference (S2) is the difference between the level (depth) of each entity in the tree. We use a feature that indicates if one entity is an ancestor of the other (S3). We also focus on frequent subsets between the entities in the syntactic dependency tree. The subsets retained are comprised of syntactic dependencies of words present in the subtree between both entities (path in the tree between both entities) (S4). In cases where either entity is composed of more than one word, we consider only the word with the lowest level to represent the query or the filler. For example, in figure 5.3, the query has two words, *Barack* and *Obama*. Since the level of *Barack* is 3 and the level of *Obama* is 2 (root level = 0), we consider only *Obama* as the query for the subset extraction. This is the case for every syntactic feature. We, once again, use the Apriori algorithm, with the following parameters: support 15%, specificity 2, limit of retained subsets 100, support increment 1%, specificity increment 10%, to obtain the most frequent subsets. The syntactic dependency subset for the example in figure 5.3 is: {*conj, nsubjpass, prep, pobj, nn*}. Another set of features derived from most frequent subsets captures the syntactic dependencies, direction and part-of-speech tag of each node in the subtree common to both entities

(S5). The direction is obtained by following the path starting from the query and ending at the filler. The Apriori algorithm is used with the same parameters as the previous set of features. Each token is comprised of the syntactic dependency, the direction (↑: the next token in the path is closer to the root, ↓: the next token in the path is further from the root, -: the token is at root level) and the POS tag. The subset for the example in figure 5.3 is: {*(conj,↑,NNP), (nsubjpass,↑,NNP), (root,-,VBN), (prep,↓,IN), (pobj,↓,NNP), (nn,↓,NNP)*}. In the same way, we extract the syntactic dependencies bigrams as well as the syntactic dependencies, direction and part-of-speech tuple bigrams within the path between the query and the filler in the syntactic dependency tree (S6 & S7). The syntactic dependency bigram subset for the example in figure 5.3 is: *conj_nsubjpass, nsubjpass_prep, prep_pobj, pobj_nn*. The syntactic dependencies, direction and part-of-speech tuple bigrams are the following: {*(conj,↑, NNP)_(nsubjpass,↑,NNP), (nsubjpass,↑,NNP)_(root,-,VBN), (root,-,VBN)_(prep,↓,IN), (prep,↓,IN)_(pobj,↓,NNP), (pobj,↓,NNP)_(nn,↓,NNP)*}. We use the Apriori algorithm to retain the most frequent subsets with the same configuration as L4 and L5 feature subsets.

5.3.6 Run Description

From the 18 available systems, we excluded 4 systems because the offsets provided did not allow us to extract the queries, fillers and justifications. In fact, the following systems' outputs are corrupted by a character offset error and were omitted in order to evaluate the filter's performances correctly: SINDI, IIRG, CohenCMU & Compreno. The experiment is executed for each of the remaining 14 participating systems by holding out its own output from the training data. The following description indicates the procedure for one system. Once the preprocessed training data is separated by relation, the features are extracted from each response. In some cases, some features may not be extracted if they depend on the position of the query entity and the slot filler in the justification sentence (e.g. the distance between entities in the syntactic tree, the ratio of part-of-speech tags of each type between the entities at the sentence level, etc.). In general, when this happens, it is due to the fact that one of the entities (query, slot filler) cannot be found in the justification, or because neither entity can be found in the same sentence. The filter only processes responses in which a reference to both entities is present in the same sentence. When both entities are not found in the same sentence, most features cannot be extracted. Therefore, the response is removed from the training set. The procedure for the test data is quite similar: responses for which entities are not found in the same sentence are simply not passed through the filter and are kept by default. Cases for which the responses are rejected correspond to approximately 15% of the instances and are unevenly distributed amongst relations. For relations such as

org:website, systems do not typically return complete sentences as justifications, but only return short textual segments generally containing only the filler. Specific relations such as *per:alternate_names* and *org:alternates_names*, for which justifications are not required to obtain a *correct* assessment are the main cause of examples being rejected from the training set.

5.3.7 Classifiers

For each relation, we trained a series of classifiers from which we selected the best classifier to apply on test data, i.e. used for filtering the relation extractors' output. Models are selected from one of the following base classifiers: RandomForest, Sequential Minimal Optimization (SMO), NBTree, DecisionTable, J48 and K* [24]. The evaluation is done on the training set using a 10-fold cross-validation. We retain the classifier which has the greatest F1 measure on the *correct* class. We selected this metric amongst others for the sole purpose that we want to limit the number of false negatives produced by the classifier. As mentioned in the previous section, the aim of the filter is to increase the relation extractors' global precision while limiting loss on recall. A significant amount of false negatives produced by the filter would result in a great loss of recall which would be detrimental to the relation extractors' F1. For relations which have been rebalanced, we train these six classifiers for the five randomly sampled subsets. We retain the classifier/subset pair which has the greatest F1 measure on the *correct* class.

5.4 Experiments & Evaluation

We performed experiments for each participating system in the TAC KBP Slot Filling task. We present results for all systems, then we focus on Relation Factory, the best performing system [38].

5.4.1 Overall Systems Results

The filtered response is evaluated using the TAC KBP 2013 English Slot Filling evaluation script version 1.4.1[1] with the *strict* configuration, for which the response is assessed as *correct* only if the answer is provided with a *correct* justification. As mentioned previously, from the 18 systems, we excluded 4 systems because their output was not processable. Our filter enables an increase in precision for every relation extractor that participated in the TAC KBP 2013 English Slot Filling task compared to its original performance. We have attempted

[1] http://www.nist.gov/tac/2013/KBP/SentimentSF/tools/SFScore.java

Table 5.3 Global Evaluation using the TAC KBP 2013 Slot Filling scorer on the test dataset for all systems before and after filtering

System ID	Pre-filtering			Post-filtering		
	Recall	Precision	F1	Recall	Precision	F1
Uwashington	0.103	0.634	0.177	0.079	0.725	0.143
BIT	0.232	0.511	0.319	0.185	0.668	0.290
CMUML	0.107	0.323	0.161	0.082	0.553	0.142
lsv (Relation Factory)	**0.332**	**0.425**	**0.373**	**0.276**	**0.630**	**0.384**
TALP_UPC	**0.057**	**0.131**	**0.080**	**0.048**	**0.262**	**0.082**
NYU	0.168	0.538	0.256	0.139	0.620	0.227
PRIS2013	0.276	0.389	0.323	0.211	0.535	0.303
Stanford	0.279	0.357	0.314	0.208	0.491	0.293
UNED	0.093	0.176	0.122	0.061	0.249	0.098
Umass_IESL	**0.185**	**0.109**	**0.137**	**0.153**	**0.159**	**0.156**
SAFT_Kres	0.150	0.157	0.153	0.095	0.167	0.122
CUNY_BLENDER	0.290	0.407	0.339	0.221	0.519	0.310
utaustin	0.081	0.252	0.123	0.050	0.310	0.087
ARPANI	0.275	0.504	0.355	0.215	0.600	0.316
Average	0.188	0.351	0.231	0.144	0.463	0.211

multiple feature configurations to train our classifiers. We have achieved the greatest precision increase using a combination of statistical and lexical/POS features using bigrams as a base for the extraction of most frequent subsets (subsets composed respectively of POS and word bigrams). We will discuss in depth our process for determining the best feature subset in section 5.4.3 (Results throughout the paper for which the feature subset is not specified use the *Statistical + Lexical/POS (bigrams)* configuration). Table 5.3 shows the evaluation for participating systems before and after filtering using this feature configuration. The filter results in an average increase in precision of 11.2% (in percentage points)and an average decrease in recall of 4.4% (in percentage points) for the 14 systems. It allows an increase in precision for every system and a F1 increase for 3 systems including Relation Factory, the best performing system. In the case of Relation Factory, the precision increases from 42.5% to 63% with a 20.5% increase. The F1 is increased from 37.3% to 38.4%.

5.4.2 Relation Factory Results

Table 5.4 shows the filter's accuracy and F1 for both *correct* and *wrong* classes evaluated by cross-validation using 10 folds on the training data. The training data contains the output of every participating system from which Relation Factory's output was held out. Table 5.5 details the performances of Relation Factory, the best system of the 2013 campaign, for each relation before and after filtering using its best feature set configuration (detailed in section 5.4.3) individually for every relation. From the 33 relations out of 41 for which there was a trained classifier, the filter increases the precision as well as the F1 for 20 relations. There is an increase in precision but a decrease in F1 for 9 relations. There was no change in precision

Table 5.4 Classifier performances on train set for Relation Factory by cross-validation (10-folds).

Relation	**Algorithm**	**Accuracy (%)**	**F1 (Correct)**	**F1 (Wrong)**
org:alternate_names	NBTree	94.7368	0.933	0.957
org:city_of_headquarters	RandomForest	74.7368	0.755	0.739
org:country_of_headquarters	NBTree	78.6207	0.739	0.819
org:date_founded	NBTree	77.4648	0.704	0.818
org:founded_by	RandomForest	92.9412	0.936	0.921
org:members	SMO	90	0.879	0.915
org:number_of_employees_members	SMO	84.375	0.828	0.857
org:parents	SMO	80.7692	0.808	0.808
org:shareholders	SMO	68.9655	0.69	0.69
org:stateorprovince_of_headquarters	RandomForest	81.25	0.747	0.851
org:subsidiaries	SMO	86.3636	0.847	0.877
org:top_members_employees	RandomForest	76.799	0.698	0.812
per:alternate_names	SMO	92.4528	0.917	0.931
per:cause_of_death	J48	84.2932	0.84	0.845
per:charges	SMO	73.7864	0.743	0.733
per:children	RandomForest	82.6087	0.797	0.848
per:cities_of_residence	SMO	75.8929	0.765	0.752
per:city_of_birth	SMO	77.0115	0.744	0.792
per:city_of_death	J48	81.6092	0.814	0.818
per:countries_of_residence	SMO	73.1343	0.746	0.714
per:country_of_death	SMO	82.2222	0.818	0.826
per:date_of_birth	J48	84	0.895	0.667
per:date_of_death	NBTree	63.8498	0.703	0.539
per:employee_or_member_of	NBTree	66.4269	0.689	0.635
per:origin	RandomForest	79.1045	0.806	0.774
per:other_family	J48	87.5	0.846	0.895
per:parents	J48	91.8033	0.918	0.918
per:schools_attended	RandomForest	79.1367	0.785	0.797
per:siblings	RandomForest	86.9565	0.877	0.862
per:spouse	RandomForest	79.2105	0.727	0.832
per:stateorprovince_of_birth	J48	84.058	0.766	0.879
per:stateorprovince_of_death	SMO	83.8095	0.825	0.85
per:statesorprovinces_of_residence	SMO	77.5281	0.773	0.778
per:title	RandomForest	70.96	0.644	0.755

Table 5.5 Evaluation for Relation Factory using the TAC KBP 2013 Slot Filling scorer on the test dataset before and after filtering for each relation.

Relation	Pre-filtering			Post-filtering		
	Instances	Precision	F1	Precision	F1	Classifier
org:country_of_headquarters	30	0.267	0.250	0.636	0.311	NBTree
org:date_founded	7	0.714	0.500	1.000	0.556	NBTree
org:number_of_employees_members	11	0.273	0.273	0.375	0.316	SMO
org:parents	16	0.25	0.276	0.364	0.333	SMO
org:subsidiaries	22	0.364	0.291	0.700	0.326	SMO
org:top_members_employees	153	0.386	0.417	0.663	0.505	RandomForest
per:alternate_names	19	0.632	0.293	0.667	0.296	SMO
per:cause_of_death	29	0.759	0.710	0.880	0.759	J48
per:charges	8	0.375	0.113	0.600	0.120	SMO
per:children	28	0.429	0.282	0.733	0.306	RandomForest
per:city_of_birth	13	0.615	0.640	0.875	0.700	SMO
per:city_of_death	25	0.800	0.702	0.909	0.741	J48
per:countries_of_residence	7	0.571	0.160	0.800	0.167	SMO
per:date_of_death	27	0.037	0.032	0.040	0.033	NBTree
per:schools_attended	22	0.364	0.314	0.615	0.381	RandomForest
per:siblings	11	0.545	0.522	0.600	0.545	RandomForest
per:spouse	22	0.500	0.400	0.750	0.400	RandomForest
per:stateorprovince_of_birth	3	0.667	0.308	1.000	0.333	J48
per:statesorprovinces_of_residence	11	0.455	0.256	0.625	0.278	SMO
per:title	345	0.348	0.417	0.580	0.445	RandomForest
org:alternate_names	62	0.710	0.583	0.722	0.545	NBTree
org:city_of_headquarters	24	0.458	0.468	0.571	0.432	RandomForest
org:founded_by	8	0.625	0.345	0.800	0.308	RandomForest
org:stateorprovince_of_headquarters	8	0.625	0.357	0.750	0.250	RandomForest
per:cities_of_residence	50	0.22	0.214	0.375	0.174	SMO
per:employee_or_member_of	70	0.257	0.185	0.360	0.120	NBTree
per:origin	19	0.526	0.339	1.000	0.298	RandomForest
per:parents	21	0.524	0.478	0.692	0.474	J48
per:stateorprovince_of_death	12	0.667	0.533	0.750	0.462	SMO
org:members	1	0.000	0.000	0.000	0.000	SMO
per:date_of_birth	7	0.857	0.600	0.857	0.600	J48
per:other_family	1	1.000	0.125	1.000	0.125	J48
per:country_of_death	3	1.000	0.462	1.000	0.333	SMO

or F1 for 3 relations. Finally, the precision along with the F1 are decreased for 1 relation.

5.4.3 Measuring the impact of features

We proceeded iteratively to measure the impact of the different features throughout the experiment. We compared the results using the full set of features to those obtained using different combinations of feature subsets presented in Section 5.3.5. We started by varying the feature subsets in our classifiers when filtering Relation Factory's output. Table 5.6 shows the performance achieved by Relation Factory after filtering using different feature configurations. For the *Lexical/POS* and *Syntactic* feature subsets, we specified if the most frequent subset based features were bigram subsets. Cases for which nothing is specified indicate that the most frequent subset based features were unigrams subsets. For the *Lexical/POS* feature subset, we also specify if only word or POS patterns were used and for the *Syntactic* feature subset, we specify if only syntactic or multilevel subsets were used. In addition to the different feature subsets applied to our filter, the table also shows as a baseline the evaluation obtained by Relation Factory's highest F1 run (to which the filter is applied). The run only uses its main components, which are the SVM classifier, the distant supervision patterns, the hand-written patterns and the alternate name expansion modules [38]. We also compare our results to Relation Factory's highest precision run, which uses syntactic patterns instead of the SVM classifier [38]. We first measured the impact of each feature subset used in combination with the statistical features subset. When using bigrams instead of unigrams for lexical/POS features, we obtain the best results when combined with statistical features. The precision is 0.63, and the F1 is 0.384 which is a 12.1% precision increase and a 4.1% F1 increase compared to Relation Factory's precision run (baseline 2). It is also a 20.5% precision increase and a 1.1% F1 increase compared to Relation Factory's F1 run (baseline 1).

To evaluate the statistical significance of our results (recall loss and precision gain), we used the chi-square test [31]. Relation Factory submitted a total of 1145 responses on which the filter was applied. Considering that the precision increases to 0.63 from 0.425 (baseline 1) and from 0.509 (baseline 2) we obtain p-values of respectively 5.6e-05 and 5.0e-09. Since the post-filtering subset is not independent from the pre-filtering subset in the case of baseline 1, we created two independent data subsets. We split Relation Factory's output into two randomly sampled subsets: we calculated the initial precision on the first subset and applied the filter to the second subset on which we then calculated the precision. We repeated the process 100 times to obtain an average p-value. We can therefore reject the null hypothesis ($p < 0.1$) and consider those results as statistically significant. Considering the precision difference between the different feature subsets 0.63 (Statistical + Lexical/POS (bigrams)) and the other subsets

Table 5.6 Results obtained by Relation Factory using the TAC KBP 2013 Slot Filling scorer on the test dataset after filtering when using different feature sets

Feature Set	**R**	**P**	**F1**	↑↑	↑↓	↓↓	–	**NT**
Baseline 1: Relation Factory (best F1 run)	0.332	0.425	0.373					
Baseline 2: Relation Factory (best precision run)	0.259	0.509	0.343					
Statistical	0.256	0.574	0.354	12	13	4	4	8
Statistical + NE	0.260	0.579	0.359	13	11	5	4	8
Statistical + Lexical/POS	0.266	0.614	0.371	16	10	4	3	8
Statistical + Syntactic	0.253	0.582	0.352	9	18	1	5	8
Statistical + Lexical/POS + Syntactic	0.271	0.616	0.377	16	10	4	3	8
Statistical + Lexical/POS + Syntactic + NE	0.264	0.591	0.365	15	12	3	3	8
Statistical + Lexical/POS (bigrams) + Syntactic (bigrams)	0.272	0.623	0.379	16	9	4	4	8
Statistical + Lexical/POS (bigrams)	**0.276**	**0.630**	**0.384**	**20**	**9**	**1**	**3**	**8**
Statistical + Syntactic (bigrams)	0.255	0.597	0.357	9	16	5	3	8
Statistical + Lexical/POS (bigrams) + Syntactic (bigrams) + NE	0.268	0.591	0.369	14	13	3	3	8
Statistical + Lexical/POS (bigrams)+ Syntactic (bigrams)+ Specific	0.266	0.600	0.369	13	15	3	2	8
Statistical + Lexical/POS (bigrams) + Syntactic (bigrams) + NE + Specific	0.267	0.597	0.369	15	13	3	2	8
Statistical + Lexical/POS (bigrams) + Syntactic (unigrams)	0.272	0.607	0.376	16	11	3	3	8
Statistical + Lexical/POS (POS bigrams only)	0.272	0.582	0.370	15	12	2	4	8
Statistical + Lexical/POS (word bigrams only)	0.266	0.618	0.372	18	11	2	2	8
Statistical + Lexical/POS (POS bigrams only) + Syntactic (unigrams)	0.274	0.617	0.379	15	13	3	2	8
Statistical + Lexical/POS (word bigrams only) + Syntactic (unigrams)	0.270	0.608	0.374	16	12	2	3	8
Statistical + Lexical/POS (bigrams) + Syntactic (syntactic dependencies unigrams only)	0.269	0.603	0.372	16	13	2	2	8
Statistical + Lexical/POS (bigrams) + Syntactic (multilevel unigrams only)	0.279	0.614	0.383	19	9	3	2	8

↑↑: Precision and F1 increase, ↑↓: Precision increase and F1 decrease, ↓↓: Precision and F1 decrease, –: No change in Precision and F1, NT: No trained classifier

for which the precision ranges from 0.574 to 0.623, we obtain p-values greater than 0.1. Thus, we cannot state that the Statistical + Lexical/POS (bigrams) allows a significant precision increase over the other feature subsets. Since the different subsets are not independent, we once again split Relation Factory's output into two randomly sampled subsets. We then applied the filter using the Statistical + Lexical/POS (bigrams) feature configuration on one subset and alteratively applied the filter using every other feature configuration on the other subset. Initially, Relation Factory has submitted 487 *correct* responses from 1468 responses contained in the goldstandard obtaining a recall of 0.332 (baseline 1). Considering that the recall decreases from 0.332 to 0.276 (Statistical + Lexical/POS (bigrams)), we obtain a p-value of 0.0287 indicating the recall decrease is statistically significant.

Based on the results obtained by filtering Relation Factory's output, for the 7 feature subsets that allow an increase in F1 for Relation Factory, we have tested our filter for the other systems as well. Table 5.7 shows the average performance for the 14 systems which were tested. The greatest average precision increase was obtained using the *Statistical + Lexical/POS (bigrams)* feature subset which is the same feature subset providing the greatest precision and F1 increase for Relation Factory. However, we assume there is no statistical significance between the precision difference obtained using the different subsets.

5.4.4 Filtering all system runs

As mentioned previously, relation extractors were required to submit between one to five runs. This gave the opportunity to the teams to test their different system configurations. The systems which submitted multiple runs generally have precision-oriented and recall-oriented configurations. Table 5.8 shows the evaluation before and after filtering for all system runs.

Table 5.7 Average evaluation obtained by the participating systems using the TAC KBP 2013 Slot Filling scorer on the test dataset after filtering when using different feature sets

Feature Subset	**Recall**	**Precision**	**F1**
Pre-Filtering	0.188	0.351	0.231
Statistical + Lexical/POS + Syntactic	0.147	0.439	0.211
Statistical + Lexical/POS + Syntactic (bigrams)	0.144	0.453	0.209
Statistical + Lexical/POS (bigrams)	**0.144**	**0.463**	**0.211**
Statistical + Lexical/POS (bigrams) + Syntactic (unigrams)	0.146	0.457	0.212
Statistical + Lexical/POS (POS bigrams only) + Syntactic (unigrams)	0.146	0.453	0.211
Statistical + Lexical/POS (word bigrams only) + Syntactic (unigrams)	0.141	0.446	0.206
Statistical + Lexical/POS (bigrams) + Syntactic (multilevel unigrams only)	0.145	0.458	0.211

Precision is increased for 43 out of 44 runs and F1 is increased for 11 out of 44 runs (25%). The filter has increased the F1 for at least one run for 5 out of 14 systems. When applied to Relation Factory's highest Recall run, the precision is increased by 20.9% and the F1 is increased by 3%. This post-filtering F1 of 39.4% is also a 2.1% increase relative to Relation Factory's highest F1 run. Table 5.9 shows the average results for respectively every system's F1-tuned, precision-tuned and recall-tuned run. Our filter consistently increases the original system precision by an average of at least 10% for any system configuration. However, the filter decreases the systems' F1 by an average of 2.4% for precision-tuned runs, 1.8% for F1-tuned runs and 1.3% for recall-tuned runs.

5.5 Discussion and Conclusion

This paper presents a method to increase the precision of relation extractors. This generic method consists of appending a machine learning-based filter to the relation extractor's pipeline in order to increase its performance. It utilizes a wide scope of features, including statistical, lexical/POS, named entity and syntactic features. A detailed analysis showed that precision could be increased for every system having participated in the TAC KBP 2013 ESF track. The filter has allowed a 20.5% precision increase and a 1.1% F1 increase for the best performing system from the 2013 slot filling track when applied to the F1-tuned run. The filter has allowed a 14.9% precision increase and a 3% F1 increase for the best performing system when applied to the recall-tuned run. In addition, the filter obtains a 2.1% F1 increase relative to the original F1-tuned run, the best global performance in the TAC KBP 2013 ESF track. Our filter performs best on high recall runs. The features used for training are mostly generic and could be utilized for any pre-defined relation. However classification is more successful when there is a sufficient amount of training data which is balanced between the *correct* and *wrong* class.

One of the main objectives of this research was to determine the most important features for the filtering step (*RQ1*). Based on the results obtained in Table 5.6, which show the post-filtering performances for Relation factory, the greatest performances were achieved using a combination of statistical and POS/lexical features. This is also true on average for every tested system as shown in Table 5.7. These results also show that the most frequent subsets are more informative when considering bigrams over unigrams. Bigram subsets implicitly capture the token sequence whereas unigram subsets do not. However we cannot state that this feature subset is better than the other feature subsets tested in table 5.6 for which the precision is above 0.582. We have also compared our results with those obtained by filtering Relation Factory's output using a confidence score threshold heuristic only which are shown

Table 5.8 Global evaluation for all systems using the TAC KBP 2013 Slot Filling scorer on the test dataset before and after filtering for all submitted runs

System ID	Run	Pre-filtering			Post-filtering		
		Recall	Precision	F1	Recall	Precision	F1
Uwashington	F1	0.103	0.634	0.177	0.079	0.725	0.143
	Precision	0.086	0.646	0.152	0.067	0.742	0.122
	Alternate	0.076	0.633	0.136	0.057	0.694	0.106
BIT	F1	0.217	0.613	0.321	0.176	0.751	0.286
	Recall	**0.260**	**0.234**	**0.246**	**0.197**	**0.395**	**0.263**
	Alternate 1	0.225	0.539	0.318	0.181	0.693	0.287
	Alternate 2	0.232	0.511	0.319	0.185	0.668	0.290
	Alternate 3	**0.251**	**0.258**	**0.254**	**0.192**	**0.445**	**0.268**
CMUML	F1	0.107	0.323	0.161	0.082	0.553	0.142
	Precision	0.053	0.443	0.095	0.042	0.633	0.079
	Alternate	0.097	0.303	0.147	0.073	0.525	0.128
lsv (Relation Factory)	**F1**	**0.332**	**0.425**	**0.373**	**0.276**	**0.630**	**0.384**
	Precision	0.259	0.509	0.343	0.216	0.637	0.322
	Recall	**0.378**	**0.351**	**0.364**	**0.304**	**0.560**	**0.394**
	Alternate 1	**0.366**	**0.369**	**0.368**	**0.295**	**0.591**	**0.393**
	Alternate 2	**0.358**	**0.381**	**0.369**	**0.286**	**0.595**	**0.386**
TALP_UPC	**F1**	**0.098**	**0.077**	**0.086**	**0.078**	**0.148**	**0.102**
	Precision	0.020	0.291	0.038	0.016	0.387	0.031
	Alternate	**0.057**	**0.131**	**0.080**	**0.048**	**0.262**	**0.082**
NYU	F1	0.168	0.538	0.256	0.139	0.620	0.227
PRIS2013	F1	0.276	0.389	0.323	0.211	0.535	0.303
	Recall	**0.335**	**0.267**	**0.297**	**0.240**	**0.395**	**0.298**
	Alternate 1	**0.324**	**0.227**	**0.267**	**0.232**	**0.341**	**0.276**
	Alternate 2	0.266	0.221	0.242	0.181	0.319	0.231
	Alternate 3	0.257	0.218	0.236	0.170	0.319	0.222
Stanford	F1	0.284	0.359	0.317	0.215	0.498	0.300
	Precision	0.267	0.382	0.314	0.204	0.530	0.295
	Alternate 1	0.279	0.357	0.314	0.208	0.491	0.293
	Alternate 2	0.267	0.351	0.303	0.200	0.483	0.283
	Alternate 3	0.256	0.353	0.297	0.189	0.494	0.274
UNED	F1	0.093	0.176	0.122	0.061	0.249	0.098
	Alternate	0.089	0.167	0.116	0.058	0.234	0.093
Umass_IESL	**F1**	**0.185**	**0.109**	**0.137**	**0.153**	**0.159**	**0.156**
SAFT_Kres	F1	0.150	0.157	0.153	0.095	0.167	0.122
	Precision	0.088	0.277	0.133	0.051	0.439	0.092
	Alternate	0.078	0.122	0.096	0.054	0.119	0.074
CUNY_BLENDER	F1	0.292	0.396	0.336	0.224	0.500	0.310
	Precision	0.268	0.443	0.334	0.207	0.543	0.300
	Alternate 1	0.275	0.400	0.326	0.212	0.498	0.297
	Alternate 2	0.290	0.407	0.339	0.221	0.519	0.310
	Alternate 3	0.258	0.435	0.324	0.196	0.555	0.290
utaustin	F1	0.081	0.252	0.123	0.050	0.310	0.087
	Alternate	0.076	0.186	0.108	0.043	0.228	0.072
ARPANI	F1	0.275	0.504	0.355	0.215	0.600	0.316
Average		0.206	0.349	0.239	0.156	0.472	0.223

Table 5.9 Average global evaluation for all systems using the TAC KBP 2013 Slot Filling scorer on the test dataset before and after filtering for F1, precision and recall oriented runs

System Configuration	Pre-filtering			Post-filtering		
	Recall	Precision	F1	Recall	Precision	F1
F1-tuned	0.190	0.354	0.231	0.147	0.460	0.213
Precision-tuned	0.167	0.398	0.218	0.129	0.510	0.194
Recall-tuned	0.201	0.313	0.224	0.152	0.420	0.211

in table 5.10. By filtering Relation Factory's output considering only the confidence score, there is no confidence score threshold that achieves a higher precision or F1 than those obtained using our filter. This confirms that our filter achieves a much greater precision than tuning an individual system by highering the threshold on the confidence score which also severely hinders the recall. To evaluate the statistical significance of our results (precision gain), we once again used the chi-square test [31]. Column *P-value (P/baseline)* indicates the p-value of the precision increase using the heuristic filter compared to the baseline. We measured this value using the same process as described in section 5.4.3. The precision increase is statistically significant for a confidence score threshold of 0.4 to 1.0. Column *P-value (P/filter)* indicates the p-value of the precision difference between our filter and the heuristic filter for different confidence score values. The precision difference is statistically significant for a confidence score threshold of 0.1 to 0.7. However, using a heuristic filter based on a confidence score threshold above 0.7 hinders the recall much more ($>$ 10 percentage points) than our filter. This recall difference is statistically significant as well.

Another objective of our research was to evaluate whether a method based on a generic set of features could lead to an increase in precision without a major setback in recall (*RQ2*). As mentioned in previous sections, our approach aims mainly at increasing the relation extractor's precision. Our filter is appended to the end of the relation extractor's pipeline, thus allowing the filter to be tested and operated on any system. This approach, however, imposes an upper bound on recall, which is the recall of the system before filtering. As shown in Table 5.8, post-filtering performances are optimal for recall-oriented system configurations. In order to increase the relation extractor's recall, the filter would have to be directly inserted inside the pipeline after the candidate generation stage. Our modular approach gives us the flexibility to operate upstream or downstream of the candidate validation stage, whereas typical ensemble learning methods can only be applied downstream. This approach would require further specificity in order to adapt to each individual system. The system's original candidate validation module could be improved by taking into account our filter's confidence score for each response. An upstream integration could be a promising avenue to explore given that the filter performs best on high-recall outputs.

Table 5.10 Results obtained by Relation Factory using the TAC KBP 2013 Slot Filling scorer on the test dataset after filtering when using a confidence score threshold heuristic (Precision increase is significant for p-value < 0.1)

Confidence Score Threshold	**Recall**	**Precision**	**F1**	**P-value (P/baseline)**	**P-value (P/filter)**
Pre-filtering (baseline)	0.332	0.425	0.373		5.6e-05*
0.1	0.319	0.452	0.374	0.46	0.000222*
0.2	0.292	0.473	0.361	0.32	0.00164*
0.3	0.275	0.494	0.353	0.18	0.00863*
0.4	0.262	0.515	0.347	0.079*	0.0302*
0.5	0.252	0.535	0.343	0.038*	0.0727*
0.6	0.210	0.528	0.300	0.074*	0.0566*
0.7	0.196	0.539	0.287	0.060*	0.0898*
0.8	0.171	0.532	0.259	0.046*	0.121
0.9	0.160	0.529	0.246	0.053*	0.120
1.0	0.147	0.539	0.231	0.039*	0.165
Using our filter	0.276	0.630	0.384	5.6e-05*	

In the same order of ideas, since the relation extractor's recall can only decrease after filtering downstream, it is imperative to limit the loss of recall. The loss of recall is caused by the filter's rejecting *correct* instances provided by the relation extractor. This case is recognized as a False Negative. Therefore, the selection of a convenient classifier for filtering is based upon the metric which is correlated with the quantity of false negatives. A high recall on the *correct* class indicates that a low rate of false negatives are generated at training time by the filter (Recall = TP/(TP + FN)). However, we also want to limit the number of False Positives, which in this case is a *wrong* response retained by the filter. A high precision on the *correct* class indicates a low rate of false positives (Precision = TP/(TP + FP)). Therefore we use the F1, which is a harmonic mean of both recall and precision, to determine the base classifier. Classifiers that exhibit a high F1 on the *correct* class at training time and are trained on a sufficient amount of training data usually perform well on the test data. To measure the impact of our algorithm selection, we tested our filter consistently using alternately each algorithm for all relations. Results of performances on Relation Factory's output are shown in Table 5.11. From the 6 machine learning algorithms, Random Forest provides the best performances with a 62.2% precision and a 38% F1 compared to a 63% precision and a 38.4% F1 when using our proposed algorithm selection method (Statistical and Lexical/POS (bigrams) features). For applications using limited resources, a filter using Random Forest as the default classifier could be considered without expecting a significant decrease in performances.

Finally, we wanted to determine whether some relations were more sensitive to our filter (*RQ3*). We separated relations based on slot quantity, slot content, number of training instances, initial recall and initial precision. Table 5.12 shows the difference in recall, precision

Table 5.11 Performance using the TAC KBP 2013 Slot Filling scorer on the test dataset when using alternatively different algorithms to filter responses for every relation (Relation Factory)

Algorithm	**Recall**	**Precision**	**F1**
Pre Filtering	0.332	0.425	0.373
Decision table	0.263	0.556	0.357
J48	0.271	0.621	0.377
Kstar	0.272	0.605	0.376
NBTree	0.254	0.572	0.352
Random Forest	**0.274**	**0.622**	**0.380**
SMO	0.256	0.596	0.358
Combination	0.276	0.630	0.384

and F1 after filtering Relation Factory's output for the different relation groups. The filter has a greater impact on list-value slots for which the precision is increased by 18.3% on average relative to 13.5% for single-value slots. List-value slots have a greater contribution on global performances since they represent 73.5% of the system's recall, whereas single slots represent 26.5% of the system's recall. The filter increases the precision of name and string slots by respectively 17.3% and 19.3% whereas it increases precision of value slots by only 9.8%. Table 5.12 also indicates that relations with a greater number of training instances achieve a greater increase in precision on average. Initial recall has a slight influence on precision increase. However, relations with a high initial recall achieve greater F1 increase. Relations with high initial precision ($\geq$65) achieve a lesser precision increase than relations with a lower initial precision. Relations for which initial precision is less than 40% achieve the greatest F1 increase (2.4%).

Another important point to consider is the fact that the quantity of data is unevenly distributed across relations. For relations that do not have enough data, models could suffer overtraining. Therefore, it is wiser not to apply the filter in those cases, to avoid a significant decrease in performance. In addition to that, there is a great potential of improvement for the filter, considering the small amount of labelled data available for this experiment. Obviously, with a greater amount of training data, the filter would cover a greater scope of examples and would be more flexible. Improvements which were not included in this paper suggest an optimization of the current features, especially for the selection of the most frequent subsets.

Another interesting path to explore in the future is feature selection. Currently, most features are generated by capturing most frequent patterns at different levels using the Apriori algorithm. The selection criteria allow us to retain patterns for which the number of occurrences in the subset of the specified class is above a certain threshold and for which the number of occurrences in the subset of the complementary class is below a certain threshold.

Table 5.12 Evaluation variation using the TAC KBP 2013 Slot Filling scorer on the test dataset after filtering for different relation groups (Relation Factory)

Relation Group	**ΔR**	**ΔP**	**ΔF1**	**#Relations**
List	-0.035	0.183	0.009	22
Single	-0.046	0.135	-0.005	11
Name	-0.044	0.173	-0.002	26
String	-0.054	0.193	0.028	3
Value	0.000	0.098	0.025	4
#train $\geq$ 300	-0.071	0.233	0.015	5
300 > #train $\geq$ 100	-0.041	0.193	-0.003	12
100 > #train	-0.028	0.124	0.005	16
Recall $\geq$ 0.5	-0.049	0.155	0.040	5
0.5 > Recall $\geq$ 0.25	-0.054	0.161	-0.008	14
0.25 > Recall	-0.021	0.174	0.003	14
Precision $\geq$ 0.65	-0.030	0.105	-0.008	9
0.65 > Precision $\geq$ 0.4	-0.050	0.197	-0.007	12
0.4 > Precision	-0.036	0.181	0.024	12

#train: number of training instances, ΔR: $\text{Recall}_{\text{post-filtering}} - \text{Recall}_{\text{pre-filtering}}$, ΔP: $\text{Precision}_{\text{post-filtering}} - \text{Precision}_{\text{pre-filtering}}$, ΔF1: $\text{F1}_{\text{post-filtering}} - \text{F1}_{\text{pre-filtering}}$, #Relations: number of relations

However, for certain relations such as *per:title* for instance, there is a wide array of patterns that exist to represent the relation. Therefore, in order to ensure an optimal coverage, an option would be to be more lenient on Apriori selection parameters and instead use feature selection algorithms such as *information gain* [50] to empirically discover the most significant features for each relation.

The simplest way to increase the amount of training would be to append to the current training dataset the responses for the systems having participated in the TAC KBP English Slot Filling track in the subsequent years when they are available. However, the data would still be unevenly distributed across relations and the observable imbalance towards the *wrong* class would persist and we would lose data by down-sampling the dataset. We could use existing methods for up-sampling the minority class instead. One way to do so could be by using the SMOTE method [14]. This method consists in creating "synthetic" examples rather than over-sampling by duplicating existing examples. This is done by creating a feature vector between a randomly selected minority class example and its nearest neighbour and adding this vector multiplied by a random number between 0 and 1 to the original selected minority class example. Finally, making use of distant supervision to increase the amount of training data would be an avenue worth considering.

5.6 Acknowledgments

This research was partially funded by the Industrial Innovation Scholarships Program.

CHAPITRE 6 DISCUSSION GÉNÉRALE

Ce chapitre contient une discussion générale sur les recherches effectuées. On y retrouve tout d'abord une courte discussion reprenant les points avancés dans la discussion du chapitre précédent faisant l'objet, entre autres, des questions de recherches. Il sera également question d'une solution alternative basée sur des caractéristiques spécifiques à certaines relations n'ayant pas été explorée dans les articles aux chapitres précédents. Finalement, quelques de possibilités de travaux futurs seront explorées.

6.1 Retour sur les questions de recherche

Un des objectifs principaux de ce mémoire était de déterminer quels étaient les caractéristiques les plus importants pour le filtrage de la sortie des extracteurs de relations (*QR1*). Le tableau 5.6 présente l'évaluation de la sortie de Relation Factory après qu'un filtrage ait été effectué avec différents ensembles de caractéristiques. Les meilleures performances ont été obtenues lorsque le filtre utilise une combinaison de **caractéristiques statistiques et lexicales**. Ceci est également vrai en moyenne pour tous les systèmes tel que démontré par les résultats dans le tableau 5.7. Ces résultats montrent également que les patrons les plus fréquents contiennent plus d'information lorsque ceux-ci sont des **bigrammes** et non des unigrammes. Les sous-ensembles de bigrammes capturent implicitement la séquence des éléments alors que ce n'est pas le cas pour les unigrammes. Toutefois, pour généraliser l'hypothèse évoquée par ces résultats, l'expérience devra être répétée sur un nombre d'instances plus élevé puisque ces résultats ne sont pas statistiquement significatifs.

Un autre objectif de ce mémoire était d'évaluer si une méthode basée sur un ensemble de caractéristiques génériques pouvait mener à une augmentation de la précision sans perte majeure de rappel (*QR2*). Tel que mentionné préalablement, notre approche vise principalement à améliorer la précision d'un extracteur de relations. Notre filtre est annexé à la sortie de l'extracteur de relations, permettant ainsi au filtre d'être intégré à tout système. Toutefois, cette approche impose une borne supérieure sur le rappel égale au rappel du système avant filtrage. Tel que présenté dans le tableau 5.8, les performances des systèmes après filtrage sont optimales pour les configurations de systèmes optimisant le rappel. Afin de permettre une augmentation du rappel, le filtre doit être intégré directement dans l'extracteur de relations après l'étape de génération de candidats. Notre approche modulaire, permet une flexibilité en ce qui concerne l'intégration de notre filtre en amont ou en aval relativement à l'étape de validation de candidats, tandis que les méthodes d'apprentissage par ensemble n'offrent

pas cette flexibilité et peuvent seulement être appliquées en aval. Une intégration de notre filtre en amont nécéssitait toutefois davantage de spécificité afin que le filtre soit adapté à chacun des systèmes. De cette manière, le module de validation de candidats original du système pourrait être amélioré s'il prenait en considération la confiance du filtre quant à chaque instance. L'exploration d'une intégration en amont pourrait être une avenue intéressante considérant que le filtre obtient ses meilleures résultats sur des sorties de systèmes dont le rappel est élevé.

Dans le même ordre d'idées, puisque le rappel de l'extracteur de relations ne peut que diminuer suite à un filtrage en aval, il est impératif de limiter la perte de rappel. Cette perte est causée par le rejet d'instances *correctes* fournies par l'extracteur de relations. Ces cas peuvent être reconnus comme des *faux négatifs*. Ainsi, la sélection d'un classifieur optimal au filtrage pour une relation donnée est basé sur une métrique qui est corrélée avec la quantité de *faux négatifs*. Un rappel élevé sur la classe *correcte* indique un faible taux de *faux négatifs* générés par le filtre lors de l'entraînement évaluée par validation croisée. Le rappel est égal au nombre de *vrais positifs* sur le nombre total d'instances *correctes* ce qui comprend les *vrais positifs* et les *faux négatifs* ($Rappel = VP/(VP + FN)$). Cependant, nous voulons également limiter le nombre de faux positifs, c'est-à-dire les instances *incorrectes* retenues par le filtre. Une précision élevée sur la classe *correct* indique un faible taux de *faux positifs*. La précision est égale au nombre de *vrais positifs* sur le nombre total d'instances considérées *correctes* par le filtre, c'est-à-dire le total de *vrais positifs* et de *faux positifs* ($Précision = VP/(VP + FP)$). Ainsi, nous utilisons le F1, qui est une moyenne harmonique du rappel et de la précision, afin de déterminer le classifieur de base à utiliser pour le filtrage. Les classifieurs qui présentent un F1 élevé sur la classe *correcte* lors de l'entraînement et qui sont entraînés sur une quantité suffisante de données obtiennent habituellement de bonnes performances sur les données de test. Afin de mesurer l'impact de notre sélection d'algorithmes, nous avons testé notre filtre en utilisant tour à tour le même algorithme de classification pour toutes les relations. Les résultats du filtrage de la sortie de Relation Factory sont présenté dans le tableau 5.11. Des six algorithmes testés, l'algorithme forêt d'arbres décisionnels (*Random Forest*) obtient les meilleures performances avec une précision de 62,2 % et un F1 de 38% comparé à une précision de 63% et un F1 de 38,4% lorsqu'une sélection d'algorithmes parmi les six algorithmes testés est effectuée (Le sous-ensemble de caractéristiques utilisé est le suivant: *statistiques et lexicaux/morpho-syntaxiques (bigrammes)*). En considérant ces résultats, des applications utilisant des ressources limitées pourraient utiliser un filtre dont le classifieur par défaut est une forêt d'arbres décisionnels sans qu'il n'y ait une diminution significative de la performance.

Finalement, nous voulions déterminer si certaines relations étaient plus sensibles au filtrage

(*QR3*). Nous avons segmenté les relations selon les critères suivants: quantité de réponses attendues (*réponse unique* ou *liste*), contenu des réponses attendues (*nom*, *valeur* ou *chaîne de caractères*), nombre d'instances d'entraînement, rappel avant filtrage et précision avant filtrage. Le tableau 5.12 présente la différence entre le rappel, la précision et le F1 de la sortie de Relation Factory avant et après filtrage pour différents groupes de relations. D'après nos résultats, le filtre a un plus grand impact sur les relations dont la réponse attendue est une liste pour lesquelles la précision est augmentée de 18,3% en moyenne par rapport à 13,5% pour les relations dont la réponse attendue est unique. Les relations dont la réponse attendue est une liste ont une plus grande contribution sur les performances globales du système puisqu'elles représentent 73,5% du rappel total du système tandis que les relations dont la réponse attendue est unique ne représentent que 26,5% du rappel. Le filtre améliore la précision des relations dont les réponses attendues sont des noms et des chaînes de caractères respectivement par 17,3% et 19,3% alors qu'il n'améliore la précision des relations dont les réponses attendues sont des valeurs par seulement 9,8%. Le tableau 5.12 indique également que les relations pour lesquelles le nombre de données d'entraînement est plus grand voient leur précision davantage améliorée par le filtre par rapport à celles dont le nombre d'instances d'entraînement est plus faible. Le rappel initial d'une relation a une légère influence sur l'amélioration de la précision suite au filtrage. Cependant, les relations qui ont un rappel initial élevé affichent une plus grande augmentation du F1. Les relations dont la précision initiale est élevée ($\geq$65) présentent une moins grande amélioration de la précision par rapport à celles dont la précision initiale est plus faible. Toutefois, les relations pour lesquelles la précision initiale est inférieure à 40% présentent l'augmentation de F1 la plus élevée (2,4%).

6.2 Caractéristiques spécifiques

Bien que nous présentons une méthode générique pour filtrer la sortie d'extracteurs de relations, nous avons aussi tenté de mesuré l'impact de caractéristiques spécifiques sur les classifieurs pour certaines relations. Pour ce faire, nous nous sommes concentrés sur les relations pour lesquelles il y avait le plus grand nombre de données. Ainsi nous avons testé l'impact de l'ajout de caractéristiques spécifiques pour 6 relations parmi les 10 relations pour lesquelles le plus grand nombre de réponses ont été soumises par les différents systèmes. Ces caractéristiques ont été déterminées en analysant un échantillon d'instances pour chaque relation. Cet échantillon est constitué de 15 instances de chaque classe (*correcte*/*incorrecte*) pour chaque relation traitée. Puisque le nombre d'instances total dans l'ensemble d'entraînement varie pour chaque relation, les caractéristiques ajoutées peuvent avoir une couverture variable du nombre total de patrons représentant la relation. Cette couverture est également inhérente à

la relation elle-même puisque les différentes relations peuvent être exprimées par une quantité variable de patrons. Le tableau B.1 en annexe présente la liste complète des caractéristiques spécifiques pour chacune des relations ciblées (*per:title, org:top_members_employees, per:employee_or_member_of, per:cities_of_residence, per:statesorprovinces_of_residence and org:shareholders*). Ces caractéristiques visent à capturer la présence de contextes, soit de groupe de mots, de types d'entités nommées ou de catégories morphosyntaxiques dans la phrase justificative ou dans le segment de phrase situé entre les entités. Les caractéristiques présentées capturent des patrons pour les instances positives (*correctes*) ou négatives (*incorrectes*).

Nous avons évalué les performances du filtre sur la sortie de Relation Factory avec plusieurs combinaisons de caractéristiques qui comprennent les caractéristiques spécifiques pour les relations énumérées précédemment. Le tableau B.2 en annexe présente les résultats de cette évaluation. Le F1 est légèrement dégradé pour deux sous-ensembles de caractéristiques sur trois et la précision est dégradée pour les trois sous-ensembles lorsque l'on ajoute les caractéristiques spécifiques. Ces résultats nous ont incités à abandonner l'exploration des caractéristiques spécifiques déterminées analytiquement sur des petits échantillons de données. Nous nous sommes plutôt tourné vers une méthode générique permettant de retenir automatiquement les patrons les plus fréquents pour chacune des relations.

6.3 Pistes à explorer

La sélection de caractéristiques serait une piste intéressante à explorer. Présentement, la plupart des caractéristiques sont générées en capturant les patrons les plus fréquents à différents niveaux en utilisant l'algorithme Apriori. Les critères de sélection permettent de retenir des patrons pour lesquels le nombre d'occurrences d'une certaine classe dans l'ensemble de données est supérieur à un certain seuil et pour lesquels le ratio du nombre d'occurrences de cette classe sur le nombre d'occurrences de la classe complémentaire est inférieur à un certain seuil. Cependant, pour les relations telles que *per:title* par exemple, il existe une grande quantité de patrons représentant la relation. Ainsi, dans le but d'assurer une couverture optimale, une option est d'utiliser des critères de sélection d'Apriori (support et spécificité) moins restrictifs et d'appliquer des algorithmes de sélection de caractéristiques basés, par exemple, sur le *gain en information* [50] afin de découvrir empiriquement les caractéristiques les plus significatives pour chaque relation.

Une autre voie à explorer vise l'augmentation du nombre de données. La manière la plus simple pour augmenter le nombre de données d'entraînement serait d'ajouter à l'ensemble de données actuel les ensembles provenant des sorties des systèmes ayant participé à la tâche

de *slot filling* aux années subséquentes si celles-ci deviennent éventuellement disponibles. Cependant les données seraient encore distribuées inégalement sur les relations. De plus, le déséquilibre vers la classe *incorrecte* persisterait et il y aurait tout de même une perte causée par le sous-échantillonnage des données. En plus d'ajouter des données, nous pourrions utiliser les méthodes existantes pour sur-échantillonner la classe minoritaire. Ceci pourrait être fait en employant la méthode *SMOTE* [14]. Cette méthode consiste en la création d'exemples "synthétiques" plutôt que de sur-échantillonner en dupliquant des instances existantes. Ceci est réalisé en créant un vecteur entre une instance de la classe minoritaire sélectionnée aléatoirement et son plus proche voisin et en ajoutant ce vecteur multiplié par un nombre aléatoire entre 0 et 1 à l'instance sélectionnée. Finalement, faire usage de la supervision distante afin d'augmenter le nombre de données d'entraînement serait une avenue à considérer.

CHAPITRE 7 CONCLUSION

Ce mémoire présente une méthode visant à améliorer la précision d'extracteurs de relations. Cette méthode est générique, premièrement puisqu'elle est facilement applicable à n'importe quel extracteur de relation, deuxièmement puisqu'elle est applicable à n'importe quelle relation prédéfinie. Cette méthode générique consiste à annexer un filtre basé sur l'apprentissage statistique à un extracteur de relations. Notre filtre utilise un large éventail de caractéristiques incluant des caractéristiques statistiques, lexicales, morphosyntaxiques, syntaxiques et sémantiques. Nous proposons également une méthode permettant d'extraire les patrons les plus fréquents entre les entités afin d'en dériver des caractéristiques booléennes indiquant la présence de ceux-ci dans les phrases justificatives soumises par les extracteurs de relations. Nos résultats démontrent que le filtre est optimal, c'est-à-dire qu'il apporte la plus grande amélioration de la précision et du F1, lorsqu'il s'appuie sur un ensemble de caractéristiques statistiques, lexicales et morphosyntaxiques. Également, les patrons les plus fréquents sont optimaux lorsque ceux-ci sont des bigrammes. Toutefois, ces résultats ne sont pas statistiquement significatifs. Le filtre utilisant différentes combinaisons de caractéristiques statistiques, lexicales, morphosyntaxiques, syntaxiques et sémantiques permet dans tous les cas d'améliorier significativement la précision du système sur lequel il est appliqué.

Une analyse détaillée a démontré que la précision peut être augmentée pour chacun des systèmes ayant participé à tâche de *slot filling* dans le cadre de la campagne TAC KBP 2013. Le filtre a permis une augmentation de 20,5% et de 1.1% de la précision et du F1 respectivement pour le système ayant originalement obtenu la meilleure évaluation dans le cadre de la campagne. Ces résultats sont obtenus en appliquant le filtre à la sortie du système configuré en vue d'optimiser le F1. Lorsque le système est configuré afin d'optimiser le rappel, le filtre permet une augmentation de 14,9% de précision et de 3% de F1 par rapport à l'évaluation originale. De cette manière, le filtre permet d'obtenir une augmentation du F1 de 2,1% par rapport à l'évaluation originale du système configuré pour optimiser le F1.

Nos résultats nous ont permis de déterminer que le filtre obtient les meilleurs résultats sur les sorties des systèmes dont le rappel est élevé. Nos résultats démontrent également que la classification est plus efficace lorsqu'il y a une quantité suffisante de données d'entraînement et lorsque ces données sont balancées entre les classes (*correctes* et *incorrectes*). Il serait donc intéressant de comparer les résultats de notre filtre lorsque nous utilisons des méthodes permettant de générer des données en sur-échantillonnant la classe minoritaire. En appliquant certaines modifications, la méthode de filtrage proposée permettra d'obtenir des données

structurées dont la précision pourrait être largement plus élevée.

RÉFÉRENCES

[1] Eugene Agichtein and Luis Gravano. Snowball: Extracting Relations from Large Plain-text Collections. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, pages 85–94. ACM, 2000.

[2] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast Algorithms for Mining Association Rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499, 1994.

[3] David W Aha, Dennis Kibler, and Marc K Albert. Instance-based Learning Algorithms. *Machine learning*, 6(1):37–66, 1991.

[4] Gabor Angeli, Sonal Gupta, Melvin Jose, Christopher D Manning, Christopher Ré, Julie Tibshirani, Jean Y Wu, Sen Wu, and Ce Zhang. Stanford's 2014 Slot Filling Systems. *TAC KBP*, 2014.

[5] Gabor Angeli, Julie Tibshirani, Jean Y Wu, and Christopher D Manning. Combining Distant and Partial Supervision for Relation Extraction. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[6] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story so Far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pages 205–227, 2009.

[7] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - A Crystallization Point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009.

[8] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM, 1992.

[9] Leo Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.

[10] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[11] Sergey Brin. Extracting Patterns and Relations from the World Wide Web. In *The World Wide Web and Databases*, pages 172–183. Springer, 1999.

[12] Christopher JC Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[13] Nitesh V Chawla. Data Mining for Imbalanced Datasets: An Overview. In *Data Mining and Knowledge Discovery Handbook*, pages 853–867. Springer, 2005.

[14] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, pages 321–357, 2002.

[15] Zheng Chen, Suzanne Tamang, Adam Lee, Xiang Li, Marissa Passantino, and Heng Ji. Top-Down and Bottom-Up: A Combined Approach to Slot Filling. In *AIRS*, pages 300–309. Springer, 2010.

[16] Grzegorz Chrupała and Dietrich Klakow. A Named Entity Labeler for German: Exploiting Wikipedia and Distributional Clusters. In *Proceedings of the Conference on International Language Resources and Evaluation (LREC)*, pages 552–556, 2010.

[17] John G. Cleary and Leonard E. Trigg. K*: An Instance-based Learner Using an Entropic Distance Measure. In *12th International Conference on Machine Learning*, pages 108–114, 1995.

[18] Thomas M Cover and Peter E Hart. Nearest Neighbor Pattern Classification. *Information Theory, IEEE Transactions On*, 13(1):21–27, 1967.

[19] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. Generating Typed Dependency Parses from Phrase Structure Parses. In *Proceedings of LREC*, volume 6, pages 449–454, 2006.

[20] Joe Ellis. TAC KBP 2013 Slot Descriptions, 2013.

[21] Joe Ellis. TAC KBP Reference Knowledge Base LDC2009E58, April 2013.

[22] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying Relations for Open Information Extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545. Association for Computational Linguistics, 2011.

[23] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[24] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA Data Mining Software: An Update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[25] Jing Jiang. *Domain Adaptation in Natural Language Processing*. ProQuest, 2008.

[26] George H. John and Pat Langley. Estimating Continuous Distributions in Bayesian Classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann, 1995.

[27] Martin Kay. Parsing in Functional Unification Grammar. *Natural Language Parsing*, pages 251–278, 1985.

[28] Ron Kohavi. The Power of Decision Tables. In *Machine Learning: ECML-95*, pages 174–189. Springer, 1995.

[29] Ron Kohavi. Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid. In *Second International Conference on Knoledge Discovery and Data Mining*, pages 202–207, 1996.

[30] Yan Li, Yichang Zhang, Xin Tong Doyu Li, Jianlong Wang, Naiche Zuo, Ying Wang, Weiran Xu, Guang Chen, and Jun Guo. PRIS at Knowledge Base Population 2013. In *Proc. TAC 2013 Workshop*, 2013.

[31] Nathan Mantel. Chi-square tests with one degree of freedom; extensions of the mantel-haenszel procedure. *Journal of the American Statistical Association*, 58(303):690–700, 1963.

[32] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant Supervision for Relation Extraction without Labeled Data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, volume 2, pages 1003–1011. Association for Computational Linguistics, 2009.

[33] Andrea Moro, Hong Li, Sebastian Krause, Feiyu Xu, Roberto Navigli, and Hans Uszkoreit. Semantic Rule Filtering for Web-scale Relation Extraction. In *The Semantic Web–ISWC 2013*, pages 347–362. Springer, 2013.

[34] Thien Huu Nguyen, Yifan He, Maria Pershina, Xiang Li, and Ralph Grishman. New York University 2014 Knowledge Base Population Systems. In *Proc. Text Analysis Conference (TAC2014)*, 2014.

[35] John Platt. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Technical Report MSR-TR-98-14, Microsoft Research, April 1998.

[36] J Ross Quinlan. *C4. 5: Programs for Machine Learning*. Elsevier, 2014.

[37] Nazneen Fatema Rajani, Vidhoon Viswanathan, Yinon Bentor, and Raymond J. Mooney. Stacked Ensembles of Information Extractors for Knowledge-base Population. volume 1, pages 177 – 187, Beijing, China, 2015. Knowledge base;Multiple systems;.

[38] Benjamin Roth, Tassilo Barth, Michael Wiegand, Mittul Singh, and Dietrich Klakow. Effective Slot Filling Based on Shallow Distant Supervision Methods. *CoRR*, 2014.

[39] Benjamin Roth, Grzegorz Chrupala, Michael Wiegand, Mittul Singh, and Dietrich Klakow. Generalizing from Freebase and Patterns Using Distant Supervision for Slot Filling. In *Proceedings of the Text Analysis Conference (TAC)*, 2012.

[40] Stuart Russell, Peter Norvig, and Artificial Intelligence. A Modern Approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[41] Nico Schlaefer, Jeongwoo Ko, Justin Betteridge, Manas A Pathak, Eric Nyberg, and Guido Sautter. Semantic Extensions of the Ephyra QA System for TREC 2007. In *TREC*, volume 1, page 2, 2007.

[42] Trilok Chand Sharma and Manoj Jain. WEKA Approach for Comparative Study of Classification Algorithm. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(4):1925–1931, 2013.

[43] Amit Singhal. Introducing the Knowledge Graph: Things, not Strings. *Official Google Blog, May*, 2012.

[44] Mihai Surdeanu. Overview of the TAC2013 Knowledge Base Population Evaluation: English Slot Filling and Temporal Slot Filling. In *TAC 2013*, 2013.

[45] Mihai Surdeanu. Slot Filler Validation at TAC 2014 Task Guidelines, April 2014.

[46] Mihai Surdeanu and Heng Ji. Overview of the English Slot Filling Track at the TAC2014 Knowledge Base Population Evaluation. 2014.

[47] Vladimir Naumovich Vapnik and Samuel Kotz. *Estimation of Dependences Based on Empirical Data*, volume 40. Springer-Verlag New York, 1982.

[48] Nikos Voskarides, Edgar Meij, Manos Tsagkias, Maarten de Rijke, and Wouter Weerkamp. Learning to Explain Entity Relationships in Knowledge Graphs. 2015.

[49] I-Jeng Wang, Edwina Liu, Cash Costello, and Christine Piatko. JHUAPL TAC-KBP2013 Slot Filler Validation System. In *Proceedings of the Sixth Text Analysis Conference (TAC 2013)*, volume 24, 2013.

[50] Yiming Yang and Jan O Pedersen. A Comparative Study on Feature Selection in Text Categorization. In *ICML*, volume 97, pages 412–420, 1997.

ANNEXE A Complément aux articles présentés

The following is a list of examples of responses submitted by Relation Factory which our filter has properly identified as *wrong*:

Slot ID: per:statesorprovinces_of_residence
Query: Adam Gadahn
Filler: Texas
Justification: In a recent video, *Gadahn* urged Muslims to follow the example of Maj. Nidal Malik Hasan, the American charged with the shooting that killed 13 people in Fort Hood, *Texas*, in November.

Slot ID: org:founded_by
Query: Galleon Group
Filler: Anil Kumar
Justification: *Anil* Kumar, a former director at the consulting firm McKinsey & Co., pleaded guilty on Thursday to providing inside information to Raj Rajaratnam, the founder of the *Galleon Group*, in exchange for payments of at least $1.75 million from 2004 through 2009.

Slot ID: org:number_of_employees_members
Query: Swiss Bankers Association
Filler: 360
Justification: "The model would generate tax revenues while respecting the privacy of bank clients and it would represent an efficient alternative to a system of automatic information exchange," said Urs Roth, chief executive of the *Swiss Bankers Association* which has almost *360* members from the financial industry.

Slot ID: per:schools_attended
Query: Nancy Kissel
Filler: High Court
Justification: If a *High Court* judge agrees with that argument, *Kissel* could ultimately be set free, barring a successful appeal by prosecutors, Chan added.

Slot ID: per:children

Query: Girija Prasad Koirala

Filler: Man Mohan Singh

Justification: Indian President Pratibha Patil and Prime Minister *Man Mohan Singh* sent condolences message to *Koirala*'s daughter and Deputy Prime Minister and Minister for Foreign Affairs Sujata Koirala on Saturday.

Table A.1 List of 41 pre-defined relations, categorization and number of responses per relation in the complete dataset

Type	**Relation**	**Content**	**Quantity**	**#Correct**	**#Wrong**	**Total**
ORG	org:alternate_names	Name	List	100	157	257
ORG	org:city_of_headquarters	Name	Single	62	118	180
ORG	org:country_of_headquarters	Name	Single	73	114	187
ORG	org:date_dissolved	Value	Single	0	15	15
ORG	org:date_founded	Value	List	36	48	84
ORG	org:founded_by	Name	List	49	127	176
ORG	org:member_of	Name	List	5	195	200
ORG	org:members	Name	List	49	195	244
ORG	org:number_of_employees_members	Value	Single	19	28	47
ORG	org:parents	Name	List	34	270	304
ORG	org:political_religious_affiliation	Name	List	1	39	40
ORG	org:shareholders	Name	List	15	255	270
ORG	org:stateorprovince_of_headquarters	Name	Single	47	76	123
ORG	org:subsidiaries	Name	List	46	259	305
ORG	org:top_members_employees	Name	List	392	612	1004
ORG	org:website	String	Single	79	17	96
PER	per:age	Value	Single	226	22	248
PER	per:alternate_names	Name	List	58	114	172
PER	per:cause_of_death	String	Single	127	110	237
PER	per:charges	String	List	58	149	207
PER	per:children	Name	List	169	202	371
PER	per:cities_of_residence	Name	List	71	356	427
PER	per:city_of_birth	Name	Single	44	56	100
PER	per:city_of_death	Name	Single	105	97	202
PER	per:countries_of_residence	Name	List	77	164	241
PER	per:country_of_birth	Name	Single	11	23	34
PER	per:country_of_death	Name	Single	26	50	76
PER	per:date_of_birth	Value	Single	63	22	85
PER	per:date_of_death	Value	Single	123	124	247
PER	per:employee_or_member_of	Name	List	257	554	811
PER	per:origin	Name	List	120	320	440
PER	per:other_family	Name	List	19	184	203
PER	per:parents	Name	List	80	167	247
PER	per:religion	String	Single	9	44	53
PER	per:schools_attended	Name	List	78	93	171
PER	per:siblings	Name	List	43	154	197
PER	per:spouse	Name	List	169	266	435
PER	per:stateorprovince_of_birth	Name	Single	27	48	75
PER	per:stateorprovince_of_death	Name	Single	57	62	119
PER	per:statesorprovinces_of_residence	Name	List	50	176	226
PER	per:title	String	List	888	1277	2165
	Total			3962	7359	11321

Table A.2 Global Evaluation for all systems before and after filtering

System ID	Recall	Precision	F1	Recall	Precision	F1
Uwashington	0.103	0.634	0.177	0.084	0.707	0.15
BIT	0.232	0.511	0.319	0.196	0.641	0.3
CMUML	0.107	0.323	0.161	0.088	0.44	0.147
lsv (Relation Factory)	0.332	0.425	0.373	0.271	0.616	0.377
TALP_UPC	0.057	0.131	0.08	0.044	0.237	0.075
NYU	0.168	0.538	0.256	0.135	0.602	0.22
PRIS2013	0.276	0.389	0.323	0.22	0.509	0.307
Stanford	0.279	0.357	0.314	0.218	0.45	0.294
UNED	0.093	0.176	0.122	0.059	0.228	0.093
Umass_IESL	0.185	0.109	0.137	0.142	0.148	0.145
SAFT_Kres	0.15	0.157	0.153	0.115	0.183	0.141
CUNY_BLENDER	0.29	0.407	0.339	0.211	0.505	0.298
utaustin	0.081	0.252	0.123	0.049	0.299	0.084
ARPANI	0.275	0.504	0.355	0.221	0.582	0.321
Average	0.188	0.351	0.231	0.147	0.439	0.211

Table A.3 Precision and F1 variation for Relation Factory after filtering for considered relations

	Pre-filtering		Post-filtering		
Relation	**Precision**	**F1**	**Precision**	**F1**	**Classifier**
org:country_of_headquarters	0.267	0.250	0.538 ↑	0.298 ↑	NBTree
org:founded_by	0.625	0.345	0.833 ↑	0.370 ↑	SMO
org:number_of_employees_members	0.273	0.273	0.750 ↑	0.400 ↑	NBTree
org:parents	0.250	0.276	0.308 ↑	0.308 ↑	SMO
org:subsidiaries	0.364	0.291	0.750 ↑	0.293 ↑	SMO
org:top_members_employees	0.386	0.417	0.667 ↑	0.488 ↑	RandomForest
per:cause_of_death	0.759	0.710	0.913 ↑	0.750 ↑	J48
per:charges	0.375	0.113	0.500 ↑	0.118 ↑	DecisionTable
per:children	0.429	0.282	0.750 ↑	0.329 ↑	SMO
per:cities_of_residence	0.220	0.214	0.345 ↑	0.244 ↑	RandomForest
per:city_of_birth	0.615	0.640	0.875 ↑	0.700 ↑	DecisionTable
per:employee_or_member_of	0.257	0.185	0.368 ↑	0.172 ↑	RandomForest
per:schools_attended	0.364	0.314	0.667 ↑	0.390 ↑	RandomForest
per:siblings	0.545	0.522	0.600 ↑	0.545 ↑	SMO
per:stateorprovince_of_birth	0.667	0.308	1.000 ↑	0.333 ↑	NBTree
per:statesorprovinces_of_residence	0.455	0.256	0.833 ↑	0.294 ↑	SMO
per:title	0.348	0.417	0.597 ↑	0.450 ↑	RandomForest
org:members	0.000	0.000	0.000 -	0.000 -	J48
per:date_of_birth	0.857	0.600	0.857 -	0.600 -	RandomForest
per:other_family	1.000	0.125	1.000 -	0.125 -	SMO
org:city_of_headquarters	0.458	0.468	0.500 ↑	0.465 ↓	SMO
org:date_founded	0.714	0.500	0.800 ↑	0.444 ↓	SMO
org:stateorprovince_of_headquarters	0.625	0.357	0.667 ↑	0.174 ↓	NBTree
per:city_of_death	0.800	0.702	0.882 ↑	0.612 ↓	DecisionTable
per:countries_of_residence	0.571	0.160	1.000 ↑	0.130 ↓	SMO
per:origin	0.526	0.339	1.000 ↑	0.298 ↓	DecisionTable
per:parents	0.524	0.478	0.615 ↑	0.421 ↓	SMO
per:spouse	0.500	0.400	0.600 ↑	0.375 ↓	SMO
per:stateorprovince_of_death	0.667	0.533	0.750 ↑	0.462 ↓	KStar
org:alternate_names	0.710	0.583	0.704 ↓	0.531 ↓	J48
per:alternate_names	0.632	0.293	0.625 ↓	0.253 ↓	SMO
per:country_of_death	1.000	0.462	0.000 ↓	0.000 ↓	J48
per:date_of_death	0.037	0.032	0.000 ↓	0.000 ↓	J48

Table A.4 Relations which are rebalanced or eliminated for each system.

Relation	Uwashington	BIT	CMUML	lsv (Relation Factory)	TALP_UPC	NYU	PRIS2013	Stanford	UNED	Umass_IESL	SAFT_Kres	CUNY_BLENDER	utaustin	ARPANI
org:alternate_names	-	-	-	r	-	-	-	-	-	-	-	-	-	-
org:city_of_headquarters	-	-	-	r	-	r	-	r	-	-	-	-	-	-
org:country_of_headquarters	-	-	-	-	-	-	-	-	-	-	-	-	-	-
org:date_dissolved	e	e	e	e	e	e	e	e	e	e	e	e	e	e
org:date_founded	-	-	-	-	-	-	-	-	-	-	-	-	-	-
org:founded_by	r	r	r	r	r	r	r	r	r	-	r	r	r	r
org:member_of	e	e	e	e	e	e	e	e	e	e	e	e	e	e
org:members	r	r	r	r	r	r	r	r	r	r	r	r	r	r
org:number_of_employees_members	-	-	-	-	-	-	-	-	-	-	-	-	-	-
org:parents	r	r	r	r	r	r	r	r	r	r	r	r	r	r
org:political_religious_affiliation	e	e	e	e	e	e	e	e	e	e	e	e	e	e
org:shareholders	r	e	r	r	e	r	r	r	e	e	r	e	r	e
org:stateorprovince_of_headquarters	-	-	-	-	-	-	-	-	-	-	-	-	-	-
org:subsidiaries	r	r	r	r	r	r	r	r	r	r	r	r	r	r
org:top_members_employees	-	-	-	-	-	-	-	-	-	-	-	-	-	-
org:website	-	-	-	-	-	-	e	-	-	-	-	e	-	-
per:age	-	-	-	e	-	-	-	-	-	-	-	-	-	-
per:alternate_names	r	-	-	r	-	-	r	-	-	-	-	r	-	r
per:cause_of_death	-	-	-	-	-	-	-	-	-	-	-	-	-	-
per:charges	r	r	r	r	r	r	r	r	r	r	r	r	r	r
per:children	-	-	-	-	-	-	-	-	-	-	-	-	-	-
per:cities_of_residence	r	r	r	r	r	r	r	r	r	r	r	r	r	r
per:city_of_birth	-	-	-	-	-	-	-	-	-	-	-	-	-	-
per:city_of_death	-	-	-	-	-	-	-	-	-	-	-	-	-	-
per:countries_of_residence	r	r	r	r	r	r	r	r	r	-	r	r	r	r
per:country_of_birth	e	e	e	e	e	e	e	e	e	e	e	e	e	e
per:country_of_death	r	-	-	r	-	r	r	-	-	-	-	-	-	r
per:date_of_birth	-	-	-	-	-	-	-	-	-	-	-	-	-	e
per:date_of_death	-	-	-	-	-	-	-	-	-	-	-	-	-	-
per:employee_or_member_of	r	r	r	r	r	r	r	r	-	r	r	r	r	r
per:origin	r	r	r	r	r	r	r	r	-	r	r	r	r	r
per:other_family	r	r	r	r	r	r	r	r	r	r	r	r	r	e
per:parents	r	r	r	r	r	r	r	r	r	-	r	r	r	r
per:religion	e	e	e	e	e	e	e	e	e	e	e	e	e	e
per:schools_attended	-	-	-	-	-	-	-	-	-	-	-	-	-	-
per:siblings	r	r	r	r	r	r	r	r	r	-	r	r	r	r
per:spouse	-	-	-	-	-	-	-	-	-	-	-	-	-	-
per:stateorprovince_of_birth	-	-	-	-	-	-	-	-	-	-	-	r	-	-
per:stateorprovince_of_death	-	-	-	-	-	-	-	-	-	-	-	-	-	-
per:statesorprovinces_of_residence	r	r	r	r	r	r	r	r	r	r	r	r	r	r
per:title	-	-	-	-	-	-	-	-	-	-	-	-	-	-

e: eliminated, r: rebalanced, -:no special treatment

Table A.5 Number of instances retained and rejected for each relation in the training set by Relation Factory

Relation	**#Correct**	**#Wrong**	**#Instances**	**#Correct Rejected**	**#Wrong Rejected**	**#Rejected**	**Total**
org:country_of_headquarters	56	85	141	9	7	16	157
org:date_founded	27	39	66	4	7	11	77
org:number_of_employees_members	14	17	31	2	3	5	36
org:stateorprovince_of_headquarters	42	69	111	0	4	4	115
org:top_members_employees	225	444	669	99	83	182	851
org:website	1	3	4	60	14	74	78
per:cause_of_death	98	97	195	7	6	13	208
per:children	136	184	320	15	8	23	343
per:city_of_birth	36	50	86	0	1	1	87
per:city_of_death	84	89	173	1	3	4	177
per:date_of_birth	57	18	75	0	3	3	78
per:date_of_death	119	92	211	3	7	10	221
per:schools_attended	67	71	138	2	9	11	149
per:spouse	145	232	377	12	24	36	413
per:stateorprovince_of_birth	23	46	69	2	1	3	72
per:stateorprovince_of_death	47	58	105	2	0	2	107
per:title	591	910	1501	143	176	319	1820
org:alternate_names	15	23	38	40	32	72	110
org:city_of_headquarters	43	44	87	8	7	15	102
org:founded_by	41	41	82	3	3	6	88
org:members	35	44	79	14	5	19	98
org:parents	26	24	50	3	5	8	58
org:shareholders	14	14	28	1	1	2	30
org:subsidiaries	31	31	62	7	7	14	76
per:alternate_names	25	28	53	21	18	39	92
per:charges	52	52	104	3	3	6	110
per:cities_of_residence	58	50	108	2	10	12	120
per:countries_of_residence	66	66	132	7	7	14	146
per:country_of_death	22	23	45	1	0	1	46
per:employee_or_member_of	202	205	407	31	28	59	466
per:origin	95	95	190	15	15	30	220
per:other_family	15	18	33	3	0	3	36
per:parents	61	57	118	6	10	16	134
per:siblings	34	35	69	2	1	3	72
per:statesorprovinces_of_residence	43	43	86	2	2	4	90

#Correct: Number of *correct* considered instances, #Wrong: Number of *wrong* considered instances, #Instances: Number of total considered instances, #Correct Rejected: Number of *correct* rejected instances, #Wrong Rejected: Number of *wrong* rejected instances, #Rejected: Number of total rejected instances

Table A.6 Classifier performances at training time by cross-validation (10-folds) for Relation Factory

Relation	**Algorithm**	**Correctly classified**	**P(C)**	**R(C)**	**F1(C)**	**P(W)**	**R(W)**	**F1(W)**	**#C**	**#W**	**TP**	**FP**	**FN**	**TN**
org:alternate_names	NBTree	94.7368	1	0.875	0.933	0.917	1	0.957	16	22	14	2	0	22
org:city_of_headquarters	RandomForest	74.7368	0.74	0.771	0.755	0.756	0.723	0.739	48	47	37	11	13	34
org:country_of_headquarters	NBTree	78.6207	0.733	0.746	0.739	0.824	0.814	0.819	59	86	44	15	16	70
org:date_founded	NBTree	77.4648	0.731	0.679	0.704	0.8	0.837	0.818	28	43	19	9	7	36
org:founded_by	RandomForest	92.9412	0.88	1	0.936	1	0.854	0.921	44	41	44	0	6	35
org:members	SMO	90	0.935	0.829	0.879	0.878	0.956	0.915	35	45	29	6	2	43
org:number_of_employees_members	SMO	84.375	0.8	0.857	0.828	0.882	0.833	0.857	14	18	12	2	3	15
org:parents	SMO	80.7692	0.84	0.778	0.808	0.778	0.84	0.808	27	25	21	6	4	21
org:shareholders	SMO	68.9655	0.667	0.714	0.69	0.714	0.667	0.69	14	15	10	4	5	10
org:stateorprovince_of_headquarters	RandomForest	81.25	0.756	0.738	0.747	0.845	0.857	0.851	42	70	31	11	10	60
org:subsidiaries	SMO	86.3636	0.893	0.806	0.847	0.842	0.914	0.877	31	35	25	6	3	32
org:top_members_employees	RandomForest	76.799	0.704	0.692	0.698	0.808	0.816	0.812	312	494	216	96	91	403
per:alternate_names	SMO	92.4528	0.957	0.88	0.917	0.9	0.964	0.931	25	28	22	3	1	27
per:cause_of_death	J48	84.2932	0.868	0.814	0.84	0.82	0.872	0.845	97	94	79	18	12	82
per:charges	SMO	73.7864	0.736	0.75	0.743	0.74	0.725	0.733	52	51	39	13	14	37
per:children	RandomForest	82.6087	0.791	0.803	0.797	0.852	0.843	0.848	137	185	110	27	29	156
per:cities_of_residence	SMO	75.8929	0.772	0.759	0.765	0.745	0.759	0.752	58	54	44	14	13	41
per:city_of_birth	SMO	77.0115	0.69	0.806	0.744	0.844	0.745	0.792	36	51	29	7	13	38
per:city_of_death	J48	81.6092	0.805	0.824	0.814	0.828	0.809	0.818	85	89	70	15	17	72
per:countries_of_residence	SMO	73.1343	0.726	0.768	0.746	0.738	0.692	0.714	69	65	53	16	20	45
per:country_of_death	SMO	82.2222	0.818	0.818	0.818	0.826	0.826	0.826	22	23	18	4	4	19
per:date_of_birth	J48	84	0.895	0.895	0.895	0.667	0.667	0.667	57	18	51	6	6	12
per:date_of_death	NBTree	63.8498	0.65	0.765	0.703	0.616	0.479	0.539	119	94	91	28	49	45
per:employee_or_member_of	NBTree	66.4269	0.68	0.698	0.689	0.646	0.626	0.635	222	195	155	67	73	122
per:origin	RandomForest	79.1045	0.791	0.821	0.806	0.791	0.758	0.774	106	95	87	19	23	72
per:other_family	J48	87.5	1	0.733	0.846	0.81	1	0.895	15	17	11	4	0	17
per:parents	J48	91.8033	0.918	0.918	0.918	0.918	0.918	0.918	61	61	56	5	5	56
per:schools_attended	RandomForest	79.1367	0.791	0.779	0.785	0.792	0.803	0.797	68	71	53	15	14	57
per:siblings	RandomForest	86.9565	0.821	0.941	0.877	0.933	0.8	0.862	34	35	32	2	7	28
per:spouse	RandomForest	79.2105	0.729	0.724	0.727	0.831	0.834	0.832	145	235	105	40	39	196
per:stateorprovince_of_birth	J48	84.058	0.75	0.783	0.766	0.889	0.87	0.879	23	46	18	5	6	40
per:stateorprovince_of_death	SMO	83.8095	0.8	0.851	0.825	0.873	0.828	0.85	47	58	40	7	10	48
per:statesorprovinces_of_residence	SMO	77.5281	0.773	0.773	0.773	0.778	0.778	0.778	44	45	34	10	10	35
per:title	RandomForest	70.96	0.643	0.644	0.644	0.755	0.755	0.755	683	994	440	243	244	750

P: Precision, R: Recall, C: *Correct*, W: *Wrong*, TP: True Positive, FP: False Positive, FN: False Negative, TN: True Negative

ANNEXE B Caractéristiques spécifiques aux relations

Table B.1: List of relation specific features

Feature (description)	Category	Class	Example
per:title			
Presence of one of the following between POS tags at a sentence level: VBD, VBN, VBZ	Lexical (POS)	Negative	After retiring from the Navy in 1962, Anderson (query) ran (VBD) for governor (filler) of Tennessee as an independent and lost.
Presence of only a comma between entities	Lexical (POS)	Positive	Gilbert Gude (query), a former Republican (filler) from Montgomery County, Md, died June 7 of congestive heart failure at Sibley Memorial Hospital in Washington D.C.
Number of PERSON groups in sentence	Named-entity	Negative	"Jennifer Dunn (query), a former Republican congresswoman from Washington state; Jack Fuller (PERSON), former president of Tribune Publishing and editorial page editor of The Chicago Tribune; and Nicholas Negroponte (PERSON), former chairman of the Massachusetts Institute of Technology's Media Lab and a founder (filler) of Wired magazine."
org:top__members__employees			
Title one of the following : spokesman, official, secretary, analyst	Lexical (POS)	Negative	ConAgra (query) spokesman Chris Kircher (filler) declined to comment on specifics of the lawsuit
Following verbs between entities: to become, to serve as, to be, assigned to, to lead, to serve as, to be appointed	Lexical (POS)	Positive	Alcatel-Lucent (query) said Tuesday it has appointed former BT chief executive Ben Verwaayen to replace outgoing CEO Patricia Russo (filler).
Presence of title between QUERY and FILLER	Named-entity	Positive	Mohamed Abdallah (filler), commander (title) of the rebel Justice and Equality Movement (JEM) (query), was killed on Sunday
Number of PERSON in sentence	Named-entity	Negative	Patrick Stewart (filler), Arsenal Football Club (query) manager Arsene Wenger (PERSON), athlete Kelly Holmes (PERSON), model Penny Lancaster (PERSON) and rapper Sean Kingston (PERSON) were among the celebrities going online for the British-based Beatbullying campaign.
per:employee__or__member__of			
Presence of title between QUERY and FILLER	Named-entity	Positive	Cathleen P. Black (query), the president (title) of Hearst Magazines (filler).
Filler is of type ORGANIZATION or GPE:COUNTRY	Named-entity	Positive	Austrian media say former U.N (filler / ORG) Secretary-General and ex-Austrian president Kurt Waldheim (query) has died.
Following verbs between entities: to become, to serve as, to be, assigned to, to lead, to serve as, to be appointed	Named-entity	Positive	In her later life, King (query) became a national spokeswoman for the American Stroke Association (filler).
Presence of PERSON between QUERY and FILLER	Named-entity	Negative	He also arranged for opera singer Beverly Sills (query), country singer Kenny Rogers (PERSON) and the Cincinnati Pops (filler) orchestra.
per:cities__of__residence			

Presence of residence residence, house, home, apartment between QUERY and FILLER	Lexical (POS)	Positive	Lorraine Rothman (query) died Sept. 25 at her home in Fullerton (filler), Calif.
Following verbs between entities: lived, resided, grew up, moved	Lexical (POS)	Positive	Crisp (query), who lived in Washington, D.C. (filler)
Following verbs between entities: comes, travels	Lexical (POS)	Negative	"Up-and-coming soprano Beverly Sills (query) comes to Fort Worth (filler) to sing the title role of Franz von Suppe's ""The Beautiful Gala"
FILLER of type GPE:CITY	Named-entity	Positive	Harry Shuler Dent (query), grew up in St. Matthews (filler / GPE:CITY), S.C., and graduated cum laude from Presbyterian College in Clinton, S.C
per:statesorprovinces_of_residence			
Presence of residence residence, house, home, apartment between QUERY and FILLER	Lexical (POS)	Positive	Bradford Washburn (query), an explorer and cartographer who, as director of the Museum of Science in Boston, led a landmark mapping of the Grand Canyon, died on Wednesday at his home in Lexington, Mass. (filler)
Following verbs between entities: lived, resided, grew up, moved	Lexical (POS)	Positive	Jefferson J. DeBlanc (query) was 86 and lived St. Martinville, La. (filler)
FILLER of type GPE:STATE_PROVINCE	Named-entity	Positive	Juanita Millender-McDonald (query), a seven-term California Democrat who chaired the Committee on House Administration, died of cancer April 22 at her home in Carson, Calif. (filler / GPE:STATE_PROVINCE)
Presence of "GPE:CITY," before FILLER	Named-entity	Positive	Irene Morgan Kirkaldy (query), whose defiance of bus segregation laws more than a decade before Rosa Parks' landmark case helped lay the foundation for later civil rights victories, died Friday at her home in Hayes (GPE:CITY), Va. (filler)
Presence of title between FILLER and QUERY	Named-entity	Positive	Former Rep. Steven B. Derounian (query), who represented Nassau County in Congress from 1953 to 1965 and was later a judge (title), died on Tuesday in Austin, Texas (filler).
org:shareholders			
FILLER of type PER or ORG	Named-entity	Positive	Nigel Phillips (filler / PER), Arsenal (query) shareholder and member of the Arsenal Supporters Trust, the club's off-field watchdog.

Table B.2 Results obtained by Relation Factory after filtering when using different feature sets (specific features)

Feature Set	**R**	**P**	**F1**	↑↑	↑↓	↓↓	–	**NT**	**Tot**
Baseline 1: Relation Factory (best F1 run)	33.2	42.5	37.3						
Baseline 2: Relation Factory (best precision run)	25.9	50.9	34.3						
Statistical	0.256	0.574	0.354	12	13	4	4	8	41
Statistical + Specific	0.265	0.546	0.357	12	13	4	4	8	41
Statistical + Lexical/POS + Syntactic	0.271	0.616	0.377	16	10	4	3	8	41
Statistical + Lexical/POS + Syntactic + Specific	0.263	0.574	0.361	14	13	3	3	8	41
Statistical + Lexical/POS + Syntactic + NE	0.264	0.591	0.365	15	12	3	3	8	41
Statistical + Lexical/POS + Syntactic + NE + Specific	0.265	0.566	0.361	13	13	4	3	8	41

↑↑: Precision and F1 increase, ↑↓: Precision increase and F1 decrease, ↓↓: Precision and F1 decrease, –: No change in Precision and F1, NT: No trained classifier