Московский Государственный Университет
имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра системного программирования

Курсовая работа

Извлечение информации из неструктурированных текстовых источников с использованием имеющихся табличных данных

Выполнил:
Студент 327 группы
Бабичев Антон Витальевич

Научный руководитель:
Недумов Ярослав Ростиславович

Москва
2014

Оглавление

Введение

Анализ информации является важной задачей во многих отраслях человеческой деятельности, таких как наука, производство, экономика. Информации в наши дни очень много, и накапливается она усиленными темпами. Однако большинство информации имеет неудобную для интерпретации форму. В частности, текст может быть абсолютно неструктурирован и не подчинен каким-либо правилам. Работа с такими данными занимает большое количество времени, поэтому естественно желание упростить, ускорить и автоматизировать эту задачу. Далее эти данные используются для дальнейшего анализа.

Наибольшее количество информации сосредоточено в Интернете, и в основном она представлена в виде web-страниц. Данные об одном и том же объекте могут встречаться на разных сайтах, и информация, которую в себе несут эти данные, может друг друга дополнять. Использование единой структурированной базы знаний об объектах какого-либо типа очень полезно и упрощает анализ неструктурированных данных о них. Информация, полученная в результате анализа такого неструктурированного текста, несет пользу и помогает дополнить наши знания об определенном объекте. Например, мы можем дополнить базу знаний записью о новом объекте, если его параметры схожи по типу с таковыми у объектов в базе знаний. Либо мы можем дополнить запись об уже существующем в базе объекте новыми параметрами-характеристиками объекта. Также полезно уметь свзывать данные об одном и том же объекте с разных страниц в Интернете, чтобы, например, проверить имеющиеся знания в базе либо же дополнить ее.

В данной курсовой работе сконцентрировано внимание на извлечении информации из неструктурированного источника с использованием базы знаний

Глава 1

Постановка задачи

Целью данной работы является исследование и разработка методов извлечения информации из неструктурированного текстового источника. Для достижения данной цели были поставлены следующие задачи:

1 Исследовать существующие методы извлечения информации из текстовых источников

2 Разработать алгоритм извлечения информации из неструктурированных текстовых данных об объекте какого-либо типа с использованием доступной базы знаний об объектах того же типа

3 Создать прототип системы извлечения информации, подтверждающий работоспособность данного метода

4 Произвести оценку результатов разработанного метода

Глава 2

Обзор существующих решений

2.1 Методы, основанные на распозавании именованных сущностей

Это группа методов, в основе которых лежат техники распознавания именованных сущностей. Именованная сущность - это группа слов в тексте, которая описывает реальный объект. Например, “Apple Inc.”, “John Broun”, “information extraction” и т.д. Поиск каких-либо именованных сущностей ведется в тексте с помощью паттерна. По способу нахождения паттерна методы делятся на подходы, основанные на правилах, и на статистические подходы.

Методы, основанные на правилах, (например, [1]) находят паттерн в тексте, редуцируя обобщенные правила. Например, из правила “является числом” получаются более специфичные правила “является 4-значным числом” или “является дробным числом”. Для вычисления правил используются размеченная вручную обучающая база, поэтому для больших текстов данный подход неприменим.

Статистические подходы (например, система Nymble [2]) используют вариации EM-алгоритма для нахождения распределений токенов по сущностям. В частности, в Nymble используются скрытые Марковские модели. Поскольку предположение, что токены сущности распределены по нормальному закону во всем тексте может быть неприемлемо в случае неструктурированного источника, данный подход нам также неинтересен.

2.2 Методы, основанные на NLP

Методы (например, [3]) этой группы используют в вычислениях предположение о наличии структуры естественного языка в текстовом источнике, поэтому для нашей задачи не годятся.

2.3 Методы, основанные на использовании базы знаний

Данная группа методов представлена подходом, описанным Matthew Michelson и Craig A. Knoblock [4] и характеризуется использованием при анализе содержания текста некой базы знаний об объектах какого-либо типа. Анализируемый текст авторы подхода рассматривают просто как набор токенов без какой-либо структуры — так называемый “текстовый пост”. База знаний представлена записями об объектах в виде совокупности пар {атрибут: значение} и в рамках статьи называется “множество элементарных исходов”. В своей статье авторы описывают рещение задачи разметки токенов поста заданным набором меток-атрибутов и особой метки “junc”, символизирующей непринадлежность токена ни одному из атрибутов объекта.

Решается задача в несколько этапов:

1 Из множества элементарных исходов с помощью приблизительного сравнения выбирается подмножество записей.

2 По посту и каждой отобранной записи строится вектор признаков

3 Вектор признаков попадет на вход SVM-классификатору, обученному относить запись к одному из 2 классов: *запись является схемой поста* и *запись не является схемой поста*

4 Лучший кандидат становится схемой поста

5 По схеме поста и каждому токену поста строится вектор признаков

6 Вектор признаков попадет на вход Multi-Class SVM-классификатору, обученному каждому токену поста ставить в соответствие метку-атрибут

7 Для группы токенов одного атрибута проводится чистка — отсеиваются токены с меткой "junc"

8 Остается размеченный пост - требуемый результат работы.

Результат работы алгоритма меряется F_1мерой и составляет 0.7704.

Данный подход интересен тем, что решает задачу извлечения информации для полностью лишенного структуры источника. Также для обучения классификаторов не требуется размеченных вручную данных.

Глава 3

Исследование и построение решения

Для реализации решения был выбран подход, предложенный АФФТАРОМ, поскольку он создан для решения проблемы извлечения информации именно в неструктурированном тексте. Авторы подхода предлагают находить решение с помощью так называемого множества элементарных исходов. Каждый элемент этого множества представляет собой совокупность пар {атрибут: значение} и описывает ровно один объект из имеющейся базы знаний.

В рамках решения поставленной задачи необходимо решить следующие подзадачи:

1 Поиск примерной схемы поста

2 Извлечение информации

Рассмотрим подробнее каждую из подзадач.

3.1 Поиск примерной схемы поста

Сперва необходимо выяснить структуру поста. Для этого пост сравнивается с элементами множества элементарных исходов (назовем его множеством кандидатов). Для каждого кандидата строится вектор признаков, который затем подается на вход классификатору, обученному относить вектор к одному из 2 классов: *кандидат соответствует посту* и *кандидат не соответствует посту*. Искомой схемой поста является кандидат, отнесенный к классу схем с наибольшей вероятностью.

Эта задача решается в 2 шага;

1 Предобработка (построение подмножества кандидатов)

2 Связывание (построение векторов и классификация)

Рассмотрим подробнее каждый из шагов.

3.1.1 Предобработка

Каждый элемент вектора признаков представляет собой значение определенной меры, вычисленной для поста и конкретного атрибута кандидата либо кандидата в целом. Поскольку кандидат содержит в среднем около 20 атрибутов, а набор мер состоит из 6 мер, вычисление каждой из которых подразумевает сравнение строк, построение вектора признаков является недешевой операцией. В множестве кандидатов находится около десятка тысяч элементов, поэтому построение векторов признаков для каждого кандидата - достаточно длительная задача. Поэтому с помощью приблизительного сравнения выбирается подмножество кандидатов, для которого в дальнейшем и будет производиться классификация.

Для этого множество кандидатов кластеризуется по значениям нескольких конкретных атрибутов. В один кластер попадает группа кандидатов с одинаковыми значениями атрибутов, указанных при кластеризации. Один кластер может быть описан в виде правила, которому удовлетворяют все элементы этого кластера. Правило удобно представлять в виде конъюнкции пар {атрибут: значение}. Например, пусть множество кандидатов состоит из 4 элементов:

{имя: ”стол”}	{длина: 120}	{ширина: 80}	{материал: ”дерево”}	[1]
{имя: ”стол”}	{длина: 120}	{ширина: 60}	{материал: ”стекло”}	[2]
{имя: ”табурет”}	{длина: 70}	{ширина: 70}	{материал: ”дерево”}	[3]
{имя: ”табурет”}	{длина: 60}	{ширина: 60}	{материал: ”сталь”}	[4]

Тогда правилу {имя: “стол”}{длина: 120} соответствуют элементы 1 и 2, а правилу {имя: “табурет”}{материал: “сталь”} - 4 элемент.

Обрабатывать и классифицировать в дальнейшем предстоит элементы одного выбранного кластера. Выбираемый кластер должен соответствовать 2 требованиям:

1 Он должен состоять из как можно меньшего числа элементов (чтобы строить меньше векторов признаков)

2 Он должен иметь достаточно элементов, чтобы содержать кандидатов, наиболее похожих на содержание поста - *перспективных кандидатов*

Для контроля требований над правилом вводятся 2 меры:

- $ReductionRatio(RR) = 1 - \frac{C}{N}$, где C — мощность кластера, описываемого правилом, а N — мощность множества элементарных исходов
- $PairCompleteness(PC) = \frac{T_n}{S_n}$, где T_n — мощность множества перспективных кандидатов, содержащихся в кластере, описываемом правилом, а S_n — мощность множества перспективных кандидатов в множестве всех кандидатов.

Задача нахождения подмножества кандидатов сводится к нахождению правил, описывающих систему кластеров, содержащих всех перспективных кандидатов. Для этого используется sequentional covering algorythm, суть которого в следующем:

перспективные кандидаты *= множество элементов, требующих покрытия*
система правил *= пустое множество*
пока ***перспективные кандидаты*** *не пусто, выполнить:*
    найти одно правило
    убрать из ***перспективные кандидаты*** *элементы, которые оно покрывает*
    добавить найденное правило к ***система правил***
результат - система правил

На каждом шаге ищется правило, максимизирующее RR и имеющее $PC >= 0.5$. Для его поиска используется следующий алгоритм:

атрибуты *= множество всех атрибутов*
кандидаты *= множество кандидатов для покрытия*
правило *= пустое множество*
пока ***атрибуты*** *не пусто, выполнить:*
    часть правила *=* ***правило***
    для каждой пары {атрибут: значение} из ***атрибуты*** *выполнить:*
        добавить к ***часть правила*** *пару {атрибут: значение}*
        *если RR(****часть правила****) > RR(****правило****) и PC(****часть правила****) >= 0.5, то*
            правило *=* ***часть правила***
        удалить из ***часть правила*** *добавленную ранее пару*
    если ***правило*** *не изменилось, то*
        результат - правило
    иначе
        удалить атрибут, добавленный в ***правило****, со всеми значениями из* ***атрибуты***
результат - правило

3.1.2 Связывание

Для поста и каждого кандидата из подмножества, полученного в ходе предобработки, строится вектор признаков V_{rl}, имеющий следующую структуру:

$V_{rl}(post, candidate) = ($
$\quad RL\_scores(post, amp_1)$
$\quad RL\_scores(post, amp_2)$
$\quad \ldots$
$\quad RL\_scores(post, amp_1 amp_2 \ldots amp_n)$

Вектор $RL\_scores(str_1, str_2)$ имеет следующую структуру:

$RL\_scores(str_1, str_2) = ($
$\quad JaccardSim(str_1, str_2)$
$\quad LevensteinDist(str_1, str_2)$
$\quad JaroWinklerSim(str_1, str_2)$
$\quad SmithWatermanSim(str_1, str_2)$
$\quad Soundex(str_1, str_2)$
$\quad PorterStemmer(str_1, str_2)$

Для полученных векторов проводится процедура binary rescoring, которая заключается в приведении векторов действительных чисел к двоичным векторам. Правило приведения таково: компонента вектора становится 1, если ее значение максимально среди значений этой же компоненты всех остальных векторов, и 0, если иначе. Если несколько векторов обладают максимальным значением какой-либо компоненты, то у всех этих векторов компонента в процессе binary rescoring станет равной 1. Например, следующие вектора

(0.21 0.0 0.5 -0.2 3.0)
(0.5 0.0 0.5 -0.3 2.0)
(-0.12 0.3 0.3 -0.7 3.0)

после binary rescoring выглядят так:

(0 0 1 1 1)
(1 0 1 0 0)
(0 1 0 0 1)

Далее вектора подаются на вход классификатору SVM, обученному относить вектор к одному из 2 классов: *кандидат соответствует посту* и *кандидат не соответствует посту*. Среди всех кандидатов, отнесенных к классу *кандидат соответствует посту* выбирается единственный, чье отнесение к данному классу сделано классификатором с большей вероятностью. Выбранный кандидат является схемой поста.

3.2 Извлечение информации

На этом шаге для найденной схемы и каждого токена поста строится вектор признаков V_{ie}, имеющий следующую структуру:

$V_{ie}(token, scheme) = ($
$\quad IE\_scores(token, amp_1)$
$\quad IE\_scores(token, amp_2)$
$\quad \ldots$
$\quad IE\_scores(token, amp_n)$

Вектор $IE\_scores(str_1, str_2)$ имеет следующую структуру:

$RL\_scores(str_1, str_2) = ($

$LevensteinDist(str_1, str_2)$
$JaroWinklerSim(str_1, str_2)$
$SmithWatermanSim(str_1, str_2)$
$Soundex(str_1, str_2)$
$PorterStemmer(str_1, str_2)$

Далее вектора подаются на вход классификатору Multi-Class SVM, обученному каждому токену поста ставить в соответствие метку того атрибута, к которому наиболее вероятно токен принадлежит. После этого для группы токенов, отнесенных к одному и тому же атрибуту, проводится процедура чистки, а именно изменения метки токена, который на самом деле не являются значением выбранного атрибута, на "пустышку". "Пустышками" в посте являются токены, которые не принадлежат значениям ни одного из атрибутов. Например, в объявлении о продаже товара это могут быть токены "cheap", "for sale", "used" и т. д. Суть чистки следующая:

1 Из группы токенов формируется значение атрибута в посте (например, строка"tok1 tok2 ... tokn")

2 Для полученной строки и значения соотнесенного атрибута из схемы поста вычисляется 2 меры: JaccardSim и JaroWinklerSim. Полученные значения считаются базовыми

3 Затем по очереди из строки убирается по одному токену, и для полученной строки и значения атрибута заново вычисляются меры

4 Если значения обоих мер возросли по сравнению с базовыми, то полученные значения сами становятся базовыми, а убранный токен становится кандидатом на удаление (замещая предыдущего кандидата, если он был)

5 После проверки всех токенов кандидату на удаление ставится метка "пустышка"

6 Процесс повторяется, пока в при проверке токенов удается найти хотя бы одного кандидата на удаление

7 Когда кандидатов на удаление больше нет, проверяются текущие значения мер. Если они меньше необходимых минимумов (для каждой меры свой), то всем токенам атрибута ставится метка "пустышка"

После очистки атрибутов остается пост, каждый токен которого помечен меткой-атрибутом либо меткой-"пустышкой" - требуемый в задаче результат.

Глава 4

Описание практической части

4.1 Инструментарий

- В качестве языка реализации выбран язык программирования Java
- В качестве реализации классификаторов используется библиотека libsvm
- В качестве реализации мер над строками используется библиотека secondstring
- Для сбора данных для обучения и тестирования написан краулер на Python с использованием библиотеки scrapy
- В качестве реализации перевода данных для обучения из формата JSON во внутреннее представление используется библиотека json-simple

4.2 Общая схема работы

1 Из JSON-файла с базой знаний загружаются записи для множества элементарных исходов

2 Из входного файла считываются текстовые посты для анализа (по одному на каждой строке) Далее для каждого поста:

2.1. Предобработка множества элементарных исходов и получение подмножества кандидатов

2.2. Связывание поста со схемой

2.3. Классификация токенов поста в соответствии со схемой

2.4. Чистка атрибутов поста

2.5. Результат записывается в файл в виде “токен метка”

4.3 Архитектура системы

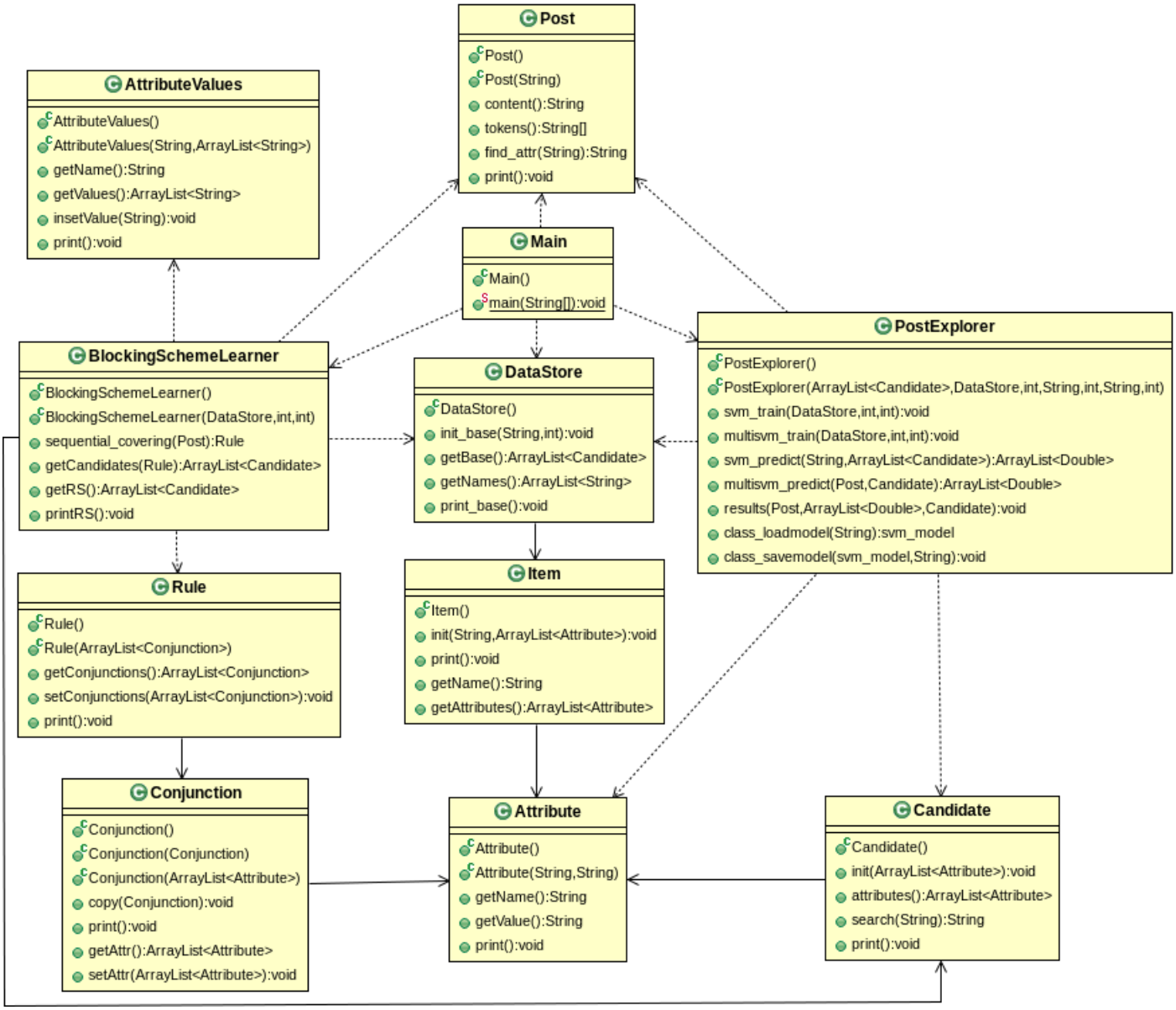
Работа программы начинается с класса *Main*.

Класс *DataStore* служит внутренним представлением базы знаний, с помощью метода *init_base* загружаются данные для обучения и множества элементарных исходов.

Класс *BlockSchemeLearner* с помощью метода *sequential_covering* осуществляет предобработку множества элементарных исходов.

Класс *PostExplorer* осуществляет связывание и извлечение нинформации. Метод *results* производит чистку и запись результатов в выходной файл

На рисунке изображена упрощенная диаграмма классов системы



4.4 Анализ результатов

Для тестирования работы алгоритма системе была поставлена задача разметки постов о ноутбуках в интернет-магазинах.

Для обучения SVM-классификатора использовались 8055 записей о ноутбуках с сайта amazon.com[1]. Для обучения Multi-Class SVM-классификатора по 8055 записям были искусственно созданы посты вида {значение_атрибута$_1$ . . . значение_атрибута$_n$}. Для тестирования использовались 104 вручную размеченные записи о ноутбуках с сайта bestbuy.com[2]

Тестировалось качество разметки поста метками 6 атрибутов: цвет, диагональ экрана, RAM, объем внешней памяти, производитель, линейка

Для оценки качества использовалась F_1мера($\frac{2*Precision*Recall}{Precision+Recall}$). Значение меры вычислялось для каждого атрибута по-отдельности.

[1]www.amazon.com
[2]www.bestbuy.com

Таблица 4.1: Результаты тестирования.

Атрибут	Precision	Recall	F_1
Цвет	0.4462	0.7632	0.5631
Диагональ экрана	0.9216	0.9592	0.94
RAM	0.3789	0.72	0.4966
Память	0.6094	0.7959	0.6903
Производитель	0.8621	1.0	0.9259
Линейка	0.2621	0.6585	0.375
Среднее	0.58	0.8162	0.6652

В целом результат работы несколько хуже, чем таковой у авторов подхода. Этому может быть 2 причины.

1 причина заключается в неоднородности обучающей выборки Multi-Class SVM-классификатора и сильном разбросе значений атрибутов. Лучшее значение F_1меры достигнуто на атрибуте “диагональ экрана”, этот атрибут есть почти у всех моделей и диапазон его значений довольно узок (около 10). В то время как атрибута “линейка” нет почти у половины записей в обучающей выборке, и диапазон значений очень широкий (его ширина сравнима с количеством записей).

2 причина заключается в процедуре чистки атрибутов и косвенно зависит от 1 причины. Из-за того, что в обучающей выборке недостаточно записей, содержащих, например, атрибут “линейка”, то классификатор токен входного поста, относящийся к атрибуту “линейка” с большой вероятностью разметит другим атрибутом. В процессе чистки, таким образом, данный токен в лучшем случае станет “пустышкой”. Поскольку пороговый минимум в конце чистки определяется исходя из обучающей выборки, а средняя длина значения атрибута “линейка” велика (более 15 символов), значения мер в чистке несколько больше, чем должно быть (так как “мусорных” токенов в посте обычно мало и они коротки). В результате страдает разметка токенов атрибутами, длина чьих значений очень мала (около 4 символов) - такие токены становятся “пустышками”. Об этом свидетельствует низкая точность определения атрибута “RAM”.

Заключение

В рамках данной курсовой работы были получены следующие результаты:

1 Исследованы существующие подходы к решению задачи извлечения информации из текстового источника.

2 Разработан метод извлечения информации из неструктурированных текстовых данныых об объекте определенного типа с использованием доступной базы знаний по объектам того же типа.

3 Создан прототип системы извлечения информации, подтверждающий работоспособность данного метода

4 Произведена оценка качества результатов разработанного метода

Список литературы

1 Mary Elaine Califf and Raymond J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the 16th National Conference on Artificial Intelligence and the 11th Innovative Applications of Artificial Intelligence Conference*, pages 328-334, 1999.

2 Daniel M. Bikel, Scott Miller, Richard Schwartz, Ralph Weischedel. Nymble: a High-Performance Learning Name-finder. In *Proceeding ANLC 1997 Proceedings of the fifth conference on Applied natural language processing* pages 194-201, 1997.

3 Frederik Hogenboom, Flavius Frasincar, Uzay Kaymak. An Overview of Approaches to Extract Information from Natural Language Corpora. *DIR 2010 January 25, 2010, Nijmegen, the Netherlands.*

4 Matthew Michelson and Craig A. Knoblock. Creating Relational Data from Unstructured and Ungrammatical Data Sources. In *Journal of Artificial Intelligence Research 31 (2008)*, pages 543-590, 2008.