Entity Linking Over Short Texts

Ricardo Ascenso Lavareda

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Bruno Emanuel da Graça Martins
Prof. Pável Pereira Calado

Examination Committee

October 2016

Acknowledgements

First, I would like to thank professor Bruno Emanuel da Graça Martins and professor Pável Pereira Calado for their guidance during this work, in which they gave me the tools, knowledge and motivation necessary to achieve success.

Second, I would like to thank my family, in particular my parents and grandparents, for their support and for giving me the opportunity to have a degree, which will be very important for my future life.

I would also like to thank to Instituto Superior Tećnico for all the time I spent there. There I gained skills and knowledge, which I believe that can contribute to a bright future.

Finally, I have to thank all my friends, who supported me and sometimes gave me the strength not to give up, despite some difficulties I had during all this time.

Lisboa, October 2016

Ricardo Ascenso Lavareda

For my parents, grandparents and friends,

Resumo

Dada uma entidade (por exemplo, uma localização, uma marca, um clube de futebol, entre outras) mencionada num texto, o objectivo da ligação de entidades é associar a referida entidade à entrada correspondente numa base de conhecimento existente. Este trabalho foca-se em aplicar ligação de entidades em tweets. Neste tipo de texto curto, o seu tamanho (máximo de 140 caracteres) leva à existência de desafios adicionais, nomeadamente o facto destes sofrerem de falta de contexto, ou o facto de usarem linguagem não regular. O trabalho desenvolvido resultou num protótipo, capaz de reconhecer e ligar entidades em textos curtos à página da Wikipedia correspondente. Ao longo deste documento, algum trabalho relacionado nos campos de aprendizagem automática e processamento de linguagem natural vão ser apresentados, tal como a estratégia usada para desenvolver o sistema proposto. O sistema foi construido usando software previamente desenvolvido, adaptado para ser executado especificamente para texto do Twitter. As experiências feitas neste trabalho provam que, treinando um model de reconhecimento de entidades com dados do Twitter e realizando alterações no texto dos tweets, melhora as performances de ligação de entidades neste tipo de textos.

Abstract

Given an entity (e.g., a location, a brand, a football club, among others) mentioned in a text, the task of entity linking is to associate the entity reference to the corresponding entry in an existing knowledge base. This work focus on applying entity linking in tweets. In this type of short text, their size (maximum of 140 characters) leads to the existence of additional challenges, namely the fact that these texts lack on string contextual evidence, or that they may use non-standard vocabulary. The work developed resulted in a prototype system, capable of recognizing and linking entities in short texts to the corresponding Wikipedia page. Throughout this document, some related work in the fields of machine learning and natural language processing will be presented, as well as the approach used to develop the proposed system. The system was built by using previously built software, adapted to perform specifically for Twitter text. The experiments performed in this work prove that by training named entity recognition models with Twitter data and by performing changes the text on tweets improves the performance of entity linking for this kind of texts.

Palavras Chave
Keywords

Palavras Chave

Processamento de Linguagem Natural

Aprendizagem Automática

Extração de Informação

Reconhecimento de Entidades

Ligação de Entidades

Keywords

Natural Language Processing

Machine Learning

Information Extraction

Named Entity Recognition

Entity Linking

Contents

List of Figures

List of Tables

1 Introduction

Twitter is a widely-know social network in the world. It has over 300 million active users (March 2016 numbers). The number of daily tweets also achieves millions. So, it is normal that some entities are usually mentioned on them. This is what entity linking is concerned about. The idea is to detect those mentions and link them to their correct Wikipedia page. However, tweets (and other types of short texts) have additional challenges for this task. The three main challenges of tweets for this task are: (1) ambiguity; (2) lack of context, since it is limited to 140 characters; (3) noisy data, namely abbreviations and slangs. For these reasons, many of the systems developed for entity linking perform poorly on this type of texts. The proposal of this work is to take an entity linking system trained for news text and adapt him to tweets. I trained a model for named entity recognition with Twitter data and then performed some changes in the tweets to try to improve the entity linking's perfomance. I compared the before and after changes results to have a better idea if it helps the task or not. Finally, I compared it with another approaches previously developed.

1.1 Motivation

Beside recognizing mentions, there are many challenges associated to performing the disambiguation. First, the problem of *ambiguity*. Take as example a Facebook post that says *Dante and David make a good duo in defense*. How can we know if *Dante* is Dante Alighieri, the character from Devil May Cry or the brazilian football player named Dante? And if *David* is David Copperfield, David Guetta or David Luiz?

This is a problem on both long and short texts, but with one small difference, that leads us to the second problem: *lack of context*. If we consider a text (e.g., a news article), it is easier to contextualize the mentions. If the same Facebook post is a sentence from a news article, we can easily understand who Dante and David are. But in a Facebook post, the input is just a small sentence, which makes it harder to contextualize. This is why entity linking is more challenging in small texts.

Finally, the problem of noisy data. Imagine a Twitter post that says *OMG, that #GameOfThrones episode killed me :O John Snow is dead! Bae is not gonna like it #GOT #nw.* This tweet is full of noisy data (e.g., slangs), which makes it harder to understand the tweet (e.g., OMG stands for *Oh My God* or #nw to *now watching*) and to perform entity linking (e.g., it is harder to link #GOT to Game of Thrones).

1.2 Problem Description

Entity linking can be very useful. If we can build systems that are able to perform the task in a good and effective way, we can, for example, enable business-intelligence or create a convenient gateway to encyclopedic information.

There already are some systems that perform this task in an effective way, which we present in Chapter 2. However, these systems work well with larger texts, relying on a larger context to help linking the entity mentions found to the corresponding entries within the knowledge base.

Nowadays, specially with the continuously growing use of social media, we have more and more short texts (e.g., things like tweets or short Facebook posts) that sometimes have mentions to entities (e.g., a person, a location, or a movie).

The main objective of this work is to find a way to take these mentions, perform the necessary intermediate steps to represent the mentions, and perform entity linking in tweets.

1.3 Contributions

The main contributions of this work are:

- Training a named entity recognition model with Twitter data improves the results of this task when applied to short texts like tweets;
- By making changes in the tweets text, named entity recognition systems have better chances of identifying the correct entities in a tweet, which helps improving the entity linking results;

- An increase of context of a tweet helps in the results of entity linking for tweets

1.4 Document Structure

This document is divided into 5 different chapters. Chapter 2 presents fundamental concepts required to understand the proposed work and work related to the subject discussed in the document. This section is divided into 2 sections. In Section 2.1, the fundamental concepts are presented, while in Section 2.2, the related work is divided into 3 categories: General Entity Linking (2.2.1), Named Entity Resolution in Tweets (2.3.1) and Entity Linking in Tweets (2.3.2). Chapter 3 presents detailed information about the Entity Linking system developed in this work. Section 3.1 presents a General Architecture of the System, while Sections 3.2 and 3.3 present the Named Entity Recognition and the Entity Linking Subsystems, respectively. Chapter 4 presents the datasets (Section 4.1), evaluation measures (Section 4.2) and the results of the work (Section 4.3). Finally, Chapter 5 presents conclusions (Section 5.1) and some ideas for future work (Section 5.2).

2 Fundamental Concepts and Related Work

In this chapter, I will present some concepts that are fundamental to better understand this work, in addition to related work developed on the subject. First, concepts fundamental to better understand this work are provided in Section 2.1. The chapter starts by covering ways of representing documents in Section 2.1.1, followed by concepts regarding named entity recognition in Section 2.1.2, and finalizes with concepts for named entity disambiguation in Section 2.1.3. Then, important related work in the area is presented in Section 2.2, covering two main categories: general entity linking (Section 2.2.1) and named entity resolution in Tweets (Section 2.2.2). This last section is also divided into two main categories: named entity recognition in Tweets (Section 2.2.2.1) and entity einking in Tweets (Section 2.2.2.2).

2.1 Fundamental Concepts

To better understand this work, it is imperative to first understand some fundamental concepts.

In particular, *Information Extraction* refers to the general task of the automatic extraction of structured information from unstructured and/or semi-structured machine-readable documents, often relying on *Natural Language Processing* (NLP). In turn, NLP is a field of artificial intelligence aiming to develop computer programs with the ability to understand human language. Sentence segmentation, part-of-speech tagging, parsing, and named entity extraction are good examples of NLP tasks in software programs.

Machine Learning is a field of artificial intelligence that provides computers with the ability to learn. Machine learning focuses on the development of computer programs that can teach themselves to grow and change when exposed to new data. Applications of machine learning include natural language processing.

2.1.1 Representing Documents

In information extraction and retrieval, the content of a document may be represented as a collection of terms: words, stems (i.e., words stripped of a suffix), phrases, or other units derived or inferred from the text of the document.

The *Vector Space Model* (VSM) is an algebraic model for representing text documents as vectors of identifiers such as, for example, index terms. In information retrieval, documents d_j and queries q are represented as vectors $\vec{d_j} = (w_{1,j}, w_{2,j}, ..., w_{t,j})$ and $\vec{q} = (w_{1,q}, w_{2,q}, ..., w_{n,q})$. Each dimension corresponds to a separate term, and typically terms are single words, keywords, or longer phrases. If words are chosen to be the terms, the dimensionality of the vector is the number of words in the vocabulary. When a term occurs in the document, its value in the vector is non-zero. Many different ways of computing these values have been developed, being TF-IDF weighting one of the best known schemes.

TF-IDF stands for term frequency-inverse document frequency, and it corresponds to a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. TF-IDF is often used as a weighting factor in information retrieval and text mining.

One good application to better understand the relation between VSM and TF-IDF is relevance rankings of documents in a keyword search engine. Consider a small collection C that consists of the following three documents:

- d_1 : "new york times"
- d_2 : "new york post"
- d_3 : "los angeles times"

Representing the documents in VSM with the TF scores computed gives us the output shown by Table 2.1.

Now the weight vector must be calculated using the following formula for each index term:

$$w_{t,d} = \mathrm{tf}_{t,d} \cdot \log \left(\frac{|D|}{|d' \in D | t \in d'} \right) \tag{2.1}$$

The first term in the equation is a local parameter that represents term frequency of term t in document d (TF score) and the log term is the inverse document frequency, which is a global parameter, where the total number of documents in the document set is divided by the number of documents containing the term t. In our example, the total of documents is 3. For each vocabulary term:

- **angeles** $\rightarrow \log_2(3/1) = 1.584$
- **los** $\rightarrow \log_2(3/1) = 1.584$
- **new** $\rightarrow \log_2(3/2) = 0.584$
- **post** $\rightarrow \log_2(3/1) = 1.584$
- **times** $\rightarrow \log_2(3/2) = 0.584$
- **york** $\rightarrow \log_2(3/2) = 0.584$

Now that both the TF and IDF scores are calculated, it is time to calculate the TF-IDF score, by applying Equation 1 (see Table 2.2).

Let us now consider the query *new new times*. The TF-IDF vector for the query must be calculated, and we also need to calculate the score of each document in C relative to this query, using the *Cosine Similarity* measure, which is a measure of similarity between two vectors that measures the cosine of the angle between them. When computing the TF-IDF values for the query terms, the frequency is divided by the maximum frequency (2) and multiplied with the IDF values (see Table 2.3).

For computing the cosine similarity, the length of each document and of the query must also be calculated, according to

$$||d|| = \sqrt{\sum_i d_i^2} \tag{2.2}$$

Table 2.1: Term Frequency scores

	angeles	los	new	post	times	york
d_1	0	0	1	0	1	1
d_2	0	0	1	1	0	1
d_3	1	1	0	0	1	0

Table 2.2: TF-IDF scores

	angeles	los	new	post	times	york
d_1	0	0	0.584	0	0.584	0.584
d_2	0	0	0.584	1.584	0	0.584
d_3	1.584	1.584	0	0	0.584	0

- **Length of d1** $\rightarrow$ $\sqrt[2]{0.584^2 + 0.584^2 + 0.584^2}$=1.011
- **Length of d2** $\rightarrow$ 1.786
- **Length of d3** $\rightarrow$ 2.316
- **Length of q** $\rightarrow$ 0.652

Finally, the cosine similarity values are calculated:

$$\text{sim}_{d_j,q} = \frac{d_j \cdot q}{||d_j|| * ||q||} \tag{2.3}$$

We would get the following values for each document:

- **cosSim(d1,q)** $\rightarrow$ (0.584*0.584+0.584*0.292+0.584*0) / (1.011*0.652) = 0.776
- **cosSim(d2,q)** $\rightarrow$ 0.292
- **cosSim(d3,q)** $\rightarrow$ 0.112

Thus, according to the similarity values, the final order in which the documents are presented, as result to the query, will be: d1, d2, d3.

The representations produced by the vector space model can be also used directly as input to different types of machine learning algorithms, for instance in tasks related to document classification.

To classify an object, we use the cosine similarity metric to search for the K (positive integer, typically small) most similar examples and, by a majority voting procedure of its neighbors, the

Table 2.3: TF-IDF score for the query

	angeles	los	new	post	times	york
q	0	0	$(2/2) * 0.584 = 0.584$	0	$(1/2) * 0.584 = 0.292$	0

object is assigned to a class. If k = 1, then the object is simply assigned to the class of that single nearest neighbor. For regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

2.1.2 Named Entity Recognition

Named Entity Recognition (NER) is a sub task of information extraction that aims to find and classify elements in text into a set of pre-defined categories (e.g., names of persons, organizations or locations) For example, consider the following sentence: *Morgan Freeman will be in London during 2016 filming a new movie.* A NER system has to be able to identify *Morgan Freeman* as a person's name, *London* as a location, and *2016* as temporal reference.

There are many different ways to perform this task. Traditional NER methods on well-formatted documents depend heavily on a phrase's local linguistic features, such as capitalization or part-of-speech (POS) tags of previous words.

Carreras et al. (2003) developed a simple named entity extractor using AdaBoost, where the NER task is performed as a combination of local classifiers which test simple decisions on each word in the text. It starts with a greedy sequence tagging procedure under the well-known BIO labeling scheme, where each word is tagged as either the beginning of a Named Entity (B), a word inside a Named Entity (I) or a word outside a Named Entity (O). The tagging process makes use of three binary classifiers trained to be experts on the recognition of B, I, and O labels. During the tagging process, the sentence is processed from left to right, greedily selecting, for each word, the tag with maximum confidence that is coherent with the current solution (e.g., I tags cannot appear after O tags).

There are actually many other ways to address this task using machine learning, namely through *Hidden Markov Models* (HMM) or through *Conditional Random Fields* (CRF).

An HMM is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states, and it can be presented as the simplest dynamic Bayesian network. Formally, an HMM can be defined as $\lambda = (A, B, \pi)$, where A represents the transition probabilities, B represents emission probabilities and π represents the start probabilities. Also

$$[A = a_{ij}] = \frac{\text{Number of transitions from state } s_i \text{ to } s_j}{\text{Number of transitions from state } s_i} \tag{2.4}$$

and

$$B = b_j(k) = \frac{\text{Number of times in state } j \text{ and observing symbol } k}{\text{Expected number of times in state } j} \tag{2.5}$$

Usually, HMMs are associated with another important algorithm used for decoding, i.e., the *Viterbi* algorithm. The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states - called the Viterbi path - that results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov models. The algorithm has 5 parameters:

1. set of states S where $|S| = N$, being N the total number of states
2. observations O where $|O| = k$, being k the number of Output Alphabets
3. transition probabilities A
4. emission probabilities B
5. initial state probabilities π

An HMM example can be found in Zhou and Su (2002). Through the HMM, the author's system is able to apply and integrate four types of internal and external evidences: 1) simple deterministic internal feature of the words, such as capitalization and digitalization; 2) internal semantic feature of important triggers; 3) internal gazetteer feature; 4) external macro context feature. The evaluation of the system on MUC-6 and MUC-7 English NE tasks achieves F-measures of 96.6

However, HMMs have difficulty modeling overlapping, non-independent features. For example, an HMM might specify which word x is likely for a given label z (i.e., tag or state) via $P(x|z)$. But often the part-of-speech of the word, as well as that of the surrounding words, character n-grams or capitalization patterns all carry important information. HMMs cannot easily model these, because the generative story limits what can be generated by a state variable. That is where Conditional Random Fields (CRFs) can help.

CRFs are discriminative graphical models that can model these overlapping, non-independent features. A special case, linear-chain CRF, can be thought of as the undirected graphical model version of HMM. It is as efficient as HMMs, where the sum-product algorithm and max-product algorithm still apply. Along a different dimension, HMMs are the sequence version of Naive Bayes models, while linear-chain CRFs are the sequence version of logistic regression.

Formally, the model can be defined as

$$P(z_{1:N}|x_{1:N}) = \frac{1}{Z}\exp\left(\sum_{n=1}^{N}\sum_{i=1}^{F}\lambda_i f_i(z_{n-1}, z_n, x_{1:n}, n)\right) \tag{2.6}$$

In the equation, λ_i is the weight for feature $f_i()$, $x_{1:N}$ are the observations (e.g., words in a document), $z_{1;n}$ are the hidden labels (e.g., tags) and Z is a scalar normalization factor defined by

$$Z = \sum_{z_{1:N}}\exp\left(\sum_{n=1}^{N}\sum_{i=1}^{F}\lambda_i f_i(z_{n-1}, z_n, x_{1:n}, n)\right) \tag{2.7}$$

To train a CRF model, the λ parameters in the model need to be found. To do so, a fully labeled data sequence $\{(x^{(1)}, z^{(1)}), ..., (x^{(m)}, z^{(m)})\}$ is required and since CRFs define the conditional probability $P(z|x)$, the appropriate objective for parameter learning is to maximize the conditional likelihood of the training data as

$$\sum_{j=1}^{m}\log p(z^{(j)}|x^{(j)}) \tag{2.8}$$

A common practice in NLP is to define a very large number of candidate features, and let the training data select a small subset to use in the final CRF model. An example of CRF for Named Entity Recognition can be found in Ritter et al. (2011), which is analyzed in section 2.2.2.1

2.1.3 Named Entity Disambiguation

Named Entity Disambiguation (NED) is concerned with associating mentions to named entities in a textual document to entries in a knowledge base.

The problem has 2 steps: detecting whether a proper name refers to a named entity included in a knowledge base (detection phase) and disambiguating between multiple named entities that can be denoted by the same proper name (disambiguation phase).

NED is different from NER in that NER identifies the occurrence or mention of a named entity in text but it does not identify which specific entity is being referenced.

To perform NED, it is mandatory to have a repository of entities called *Knowledge Base* (KB). A KB is a repository used to store complex structured and unstructured information. KBs usually contain a large number of entities, which are assigned to semantic types (e.g., a character from a movie, a singer form a music band), as well as an even bigger number of relational facts between entities. In addition, they also provide dictionaries of short names and paraphrases for entities, which allows a quicker way to identify candidate entities for mentions. YAGO[1], Freebase[2], and DBpedia[3] are good examples of KBs. Regarding the linking between named entities and the corresponding entity in a KB, there are also some concepts that are important to know.

Support Vector Machines (SVMs) are one way to perform NED. SVMs are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on. More formally, a support vector machine constructs a hyperplane[4] or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since, in general, the larger the margin the lower the generalization error of the classifier.

To better perform disambiguations, a collection of features needs to be used. Some of the most common are Entity Prior, Name Probability, and Textual Context (the use of all these

[1]`https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/`

[2]`https://www.freebase.com/`

[3]`http://wiki.dbpedia.org/`

[4]subspace of one dimension less than its ambient space

features in different systems is mentioned in Section 2). *Entity Prior* is a simple approach to entity disambiguation which ranks candidate entities in terms of their popularity. For example, if we search Miami, one percentage of links in Wikipedia will point to the city, while another percentage point to the NBA team (Miami Heat). Chisholm and Hachey (2015) define the entity prior as the probability of a link pointing to entity e:

$$f_{prior}(e) = \log \frac{|I_{*,e}|}{|I_{*,*}|} \tag{2.9}$$

In the equation $I_{*,e} \in I_{*,*}$ is the set of pages that link to entity e.

Name Probability models the relationship between a name and an entity. It was introduced as an initial score in coherence-driven disambiguation and is nowadays used in most state-of-the-art systems. Chisholm and Hachey (2015) defines it as the conditional probability of a name referring to an entity:

$$f_{name}(e, n) = \log \frac{|M_{n,e}|}{|M_{n,*}|} \tag{2.10}$$

In the equation, $M_{\mathrm{n,e}}$ is the set of mentions with name n that refer to entity e and $M_{\mathrm{n},*}$ is the set all mentions with name n.

Finally, the *Textual Context* provides a means to distinguish between entities based on the words with which they occur. This way, we can go beyond intrinsic entity and name popularity. For example, references to *Miami* the city appear in passages with words like *USA* or *beach*. References to the NBA team appear with words like *basketball*, *Heat*, or *sports*.

When linking a textual entity mention in a given document to an entry in a KB, the system may return that there is no matching entry. This problem is called *NIL mention detection*. Further details about the NIL detection problem are given at Section 2.2.

2.1.4 Evaluation Measures

Before explaining the measures used in the work, it is important to understand three basic concepts. *True Positive* refers to an entity that is supposed to be caught and its link is correctly made. *False Positive* refers to an entity that is not supposed to be caught, but it is. Finally, *False Negative* refers to an entity that is supposed to be caught, but the link is incorrectly made.

Three different evaluation measures were used. The first one is *Precision*. Precision is the fraction of retrieved instances that are relevant. For this work, it is calculated as

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.11}$$

In the equation, TP refers to True Positives and FP refers to False Positives.

Recall is the fraction of relevant instances that are retrieved. For this work, it is calculated as

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.12}$$

In the equation, TP refers to True Positives and FN refers to False Negatives.

Finally, F1 is a measure of a test's accuracy. For this work, it is calculated as

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.13}$$

2.2 Related Work

Now that some fundamental concepts has been explored, it is time to explore previous work related to the subject of entity linking. This section will start by exploring general entity linking, followed by an analysis of systems focusing on tweets.

2.2.1 General Entity Linking

Yosef et al. (2011) presented AIDA, a robust framework centered around collective disambiguation, exploiting the prominence of entities, similarity between the context of the mention and its candidates, and the coherence among candidate entities for all mentions.

AIDA leverages the YAGO2 KB as an entity catalog and a rich source of relationships between entities. In AIDA, mentions are automatically detected using the Stanford NER tagger[5]. For collective mapping, they use a graph-based approach. The graph is constructed with mentions and their candidate entities as nodes, with 2 types of edges: *mention-entity* edges between mentions and their candidate entities, with weights that capture the similarity between

[5]http://nlp.stanford.edu/software/CRF-NER.shtml

the context of a mention and a candidate, and *entity-entity* edges between different entities, with weights that capture the coherence between two entities.

The authors define entity linking as the task of reducing the graph to a dense sub-graph where each mention node is connected to one and only one candidate entity node. Once the graph is constructed, they use a two-step greedy algorithm to compute the sub-graph. In the first step, this procedure identifies the entity node that has the lowest weighted degree. In the second step it removes that node and its incident edges from the graph unless it is the last remaining candidate entity for one of the mentions.

The authors also apply some features and measures for computing the edge weights. The *mention-entity* similarity is computed as a linear combination of two ingredients. The first is the prominence of an entity, which acts as a prior probability for each potential mapping. This prior is computed by collecting statistics on anchor texts and their link targets in Wikipedia. The second ingredient is based on the overlap between a mention's context and a candidate's entity context. For a mention, the authors consider the full input as its context. For entities, and since the preferred similarity of AIDA is keyphrase-based (keyphrases are obtained from the collection of Wikipedia anchor text that links to the entity and from the titles of the articles that have links to the entity's article), the authors consider entity keyphrases pre-computed from the Wikipedia articles that YAGO's entities connect to. However, and despite the fact that the keyphrase overlap between the contexts of a mention and entity is an expressive similarity measure, it will rarely find perfect matches for multi-word keyphrases in the input text. So, the score for one keyphrase relies on partial matches of the keyphrase with the document text (including the mention) and their mutual information. For *entity-entity*, two different edge weights can be used, in order to capture coherence. The *Milne-Witten relatedness* measure (Milne and Witten (2008)) uses the Wikipedia link structure, taking into account the entities that have links to n_1 or n_2 and entities that have links to n_1 and n_2 at the same time:

$$\mathrm{MW}(a, b) = 1 - \frac{\log(\max(|A|, |B|)) - \log(|A \cap B|)}{\log(|W|) - \log(\min(|A|, |B|))} \tag{2.14}$$

In the equation, a and b are two articles of interest, A and B are the sets of all articles that link to a and b and W is the entire Wikipedia. The *Keyphrase Overlap Relatedness* (KORE) measure is based on the overlap between the keyphrases of each entity. The higher the overlap between the keyphrases of each entity, the higher the edge weight.

The authors built a demo that allows the users to interact with the system with two types of input: text or tables. For text (e.g., ambiguous names or short text), the user can choose different methods, vary configurations, explore effects in terms of candidate entities, the weighted graph of mentions and entities, and the output quality. They concluded that *prior-only* (i.e., choose the most frequent meaning for a mention in Wikipedia) and the *prior-and-similarity* (i.e., using not only prior probability, but also keyphrases and including distance to mention) methods make many mistakes, while their *graph-based* method achieves better results in the disambiguations performed. This aspect is due to the use of entity-entity coherence (in addition to the previous features).

For tabular input, it is possible to copy-and-paste Web tables in HTML (or XML) format. However, tabular input has a bigger data ambiguity. For example, if we have a table comparing the attendance at the stadiums of football teams during the last 5 seasons, it is harder to contextualize it, since we only have the name and the attendance numbers. However, the graph-based approach again achieves higher values of accuracy, in comparison with prior-only and prior-and-similarity. Detailed information about how AIDA works is presented at Chapter 3.

Lin et al. (2012), investigated entity linking over millions of high-precision extractions from a corpus of 500 million Web documents, with the goal of crating a useful knowledge base of general facts. As in many state of the art systems, given a textual assertion, they aim to find the Wikipedia entity that corresponds to the argument. To improve execution speed, the authors decided to start with a faster system leveraging linking features (e.g., string matching, or prominence or context matching) instead of designing a system that relies on the full Wikipedia graph structure. As demonstrated by Ratinov et al. (2011), these local features already provide a baseline that is very difficult to beat with the more sophisticated (and slower to compute) global features (0.9222 accuracy when using a Wikipedia dataset), which lead the authors to focus on the faster techniques, leaving the incorporation of *corpus-level* features to increase precision to a later stage.

On their basic linker, given an entity string, they first obtain the most prominent Wikipedia entities that meet a string matching criteria, using *inlink count*, which is the number of Wikipedia pages that link to a Wikipedia entity's page. Then, with the list of candidate entities, they employ a context matching step that uses cosine similarity to measure the semantic distance between the assertion and each candidate's Wikipedia article. For example, if the assertion is

New York is a city in the USA, the system would calculate the similarity between this sentence and the Wikipedia articles for *New York* (considering we are using New York as the mention to look in the knowledge base). Finally, they calculate a *link score* for each candidate as a product of string match level, prominence score, and context match score. They calculate a *link ambiguity score* as the second highest link score divided by the highest link score.

Regarding *corpus-level* linking, they use two features: *collective contexts* and *link count expectation*. Collective contexts can help providing stronger context signal for entity linking, by collecting together the various source sentences from different documents where the assertion is present. This way, the assertion is more contextualized, and the entity linking achieves better results. Link count expectation is formalized by calculating an inlink ratio for each entity as the number of assertions linking to it divided by its inlink prominence. Since the authors use a extract then entity-link pipeline, it allows them to capture assertions concerning the entities that are not prominent enough to have their own Wikipedia article. To deal with that, they follow 3 steps. First, to *detect entities*, they trained a classifier to separate entities that are not prominent enough and non-entities, by using features derived from the Google Books Ngrams corpus[6] . Second, they found that they could *predict types* (from Freebase) of unlinkable entities using instance-to-instance class propagation from the linked entities (i.e., if a mention cannot be linked, they can predict its semantic type by observing if the collection of relations it appears with also occurs with linkable entities and thus propagate the semantic types from these similar entities). For example, if the unlinkable entity is *prune juice*, we can predict its semantic type by observing the collection of relations it appears with (e.g., *is a good source of*). If this relation also appears with linkable entities (e.g., *apple juice* and *orange juice*), we can propagate the semantic type and predict the type for the unlinkable one. Third, to help *disambiguation*, they use clustering to recover the most likely types of the multiple entities and then divide assertions among them.

After testing the system, they observed that entity linking of 15 million textual extractions enables several benefits. The first is the *Freebase Selectional Preferences.* Each Wikipedia entity that gets linked is easily annoted with its Freebase semantic types using data from the Freebase Wikipedia Extraction (WEX)[7] project. On the 15 million extractions, the entities that they linked to encompassed over 1300 Freebase types. Knowing these entity types allows them to

[6]https://books.google.com/ngrams
[7]http://wiki.freebase.com/wiki/WEX

compute the Freebase selectional preferences of all their textual relations.

The second main benefit is the *improved instance ranking function.* They observed that the link score is a better ranking function than assertion frequency for presenting query results. When results are sorted by link score, they are noticeably more specific and generally more correct than the top-frequency-sorted instances.

Chisholm and Hachey (2015) explored entity linking with web links versus linking with Wikipedia. Their evaluation was done with the CoNLL[8] and TAC[9] data sets. Regarding Wikipedia, a wide range of systems use approaches to take advantage of the clean, well-edited information in Wikipedia, which includes *entity prior* models derived from popularity metrics, *disambiguation* pages and inter-article links, or *entity coherence* models derived from the Wikipedia inter-article link graph. The authors survey these models and describe a new baseline system that instantiates them from existing Wikipedia-derived data sets. For Wikipedia, their components include *entity prior* and *name probability*, as well as *textual context.* Entity prior is derived from DBpedia's Wikipedia Pagelinks [10], which contains the link graph between Wikipedia pages. Entity prior alone achieves 68.4 p@1 (precision at 1, i.e., precision at the first relevant result) on the CoNLL development data. For name probability computation, the authors use existing conditional probability estimates from the DBpedia Lexicalizations data set[11]. The textual context is modeled as a weighted bag of words (BOW), with each term $\vec{t}$ containing TF-IDF weights. The authors derived the term frequency for an entity e from the corresponding article content in the Kopiwiki[12] plain text extraction, where candidate entities are scored using cosine distance between a mention context $\vec{t_m}$ and the entity model t_e

$$f_{bow}(m,e) = 1 - \cos(\vec{t_m}, \vec{t_e}) \tag{2.15}$$

This BOW context derived from Wikipedia article text achieves 50.6 p@1 on the CoNLL development data. In addition, the authors also built entity models from their mention contexts, (i.e., the combined text surrounding all incoming links) by projecting mentions into Kopiwiki article text (which yields more contexts than the actual Wikipedia links). For an article a, the

[8]http://ifarm.nl/signll/conll/
[9]http://www.nist.gov/tac/
[10]http://oldwiki.dbpedia.org/Downloads2014#wikipedia-pagelinks
[11]http://wiki.dbpedia.org/lexicalizations
[12]http://kopiwiki.dsd.sztaki.hu/

authors tag as mentions all aliases of entities linked to from a. The term frequency for an entity e is calculated over the concatenation of all contexts for e. This BOW context derived from mentions achieves 55.8 p@1 on the CoNLL development data.

They also introduced a simple distributed bag-of-words (DBOW) model that represents the context as the TFIDF-weighted average over word vectors (collective name for a set of language modeling and feature learning techniques in natural language processing where words or phrases from the vocabulary are mapped to vectors of real numbers in a low-dimensional space relative to the vocabulary size):

$$\vec{v_p} = \frac{1}{|T_p|} \sum_{t \in T_p} \text{tfidf}(t, p) \cdot \vec{v_t} \tag{2.16}$$

In the formula, T_p is the set of terms in passage p, and $\vec{v}_t \in V$ is the word vector for term t. The candidates are scored using the cosine distance between mention context $\vec{v}_m$ and the entity model $\vec{v}_e$

$$f_{\text{dbow}}(m, e) = 1 - \cos(\vec{v}_m, \vec{v}_e) \tag{2.17}$$

On the CoNLL development data, DBOW context models derived from article text achieve 49.9 p@1, while derived by mentions context achieves 51.2 p@1.

Web links are a comparatively noisy source (e.g., anchors are less likely to be well-formed). To instantiate the entity prior (f_{prior}), the authors built a page-entity link graph from Wikilinks. Pages and entities are the same as in the Wikipedia graph, and with Wikilinks they have an unweighted bipartite graph of links from web pages to Wikipedia articles. The link-derived entity prior achieves 63.0 p@1 on the CoNLL development data. To instantiate the name probability (f_{name}), they build a name-entity graph from Wikilinks, where the structure is the same as the corresponding model from Wikipedia (i.e., both are bipartite graphs with co-occurrence frequencies on edges). However, names here are sourced from link anchors in web pages rather than from Wikipedia articles. On the CoNLL development data, link-derived name probability achieved 58.4 p@1. For textual context, to instantiate f_{bow} and f_{dbow}, they follow the same methodology used for Wikipedia mention contexts. The term frequency for an entity e is calculated over the concatenation of mention contexts for e and document frequency is calculated across aggregated entity contexts. On the CoNLL development data, BOW context achieves 62.2 p@1 and DBOW context achieves 54.0 p@1.

To perform disambiguation, the authors first extract a set of real-valued features (6 from

Wikipedia and 4 froam Wikilinks, previously mentioned and selected by the optimal combination through exhaustive search) for each candidate entity e given a training set of mentions M. Then, they train a Support Vector Machine (SVM) classifier to perform pairwise ranking, where for each mention in the training set, they derive training instances by comparing the feature vector of the gold link $(\vec{f_g})$ with each non-gold candidate (f_c). Finally, they create instances for the top-ten non-gold candidates by adding the absolute feature values

$$\text{activation}(c) = \sum_{i=1}^{|\vec{f_c}|} |\vec{f}_{c,i}| \tag{2.18}$$

To add coherence, the authors employ a simple approach based on conditional probabilities and the candidate-level feature is the average

$$f_{cond}(e) = \frac{1}{|C|} \sum_{c \in C} \log p_{coh}(e|c) \tag{2.19}$$

In the formula, C is the set of context entities for candidate entity e. For Wikipedia and Web link coherence, I_e models are derived respectively from the set of other articles that link to e and from the set of web pages that link to e. Both models achieve, respectively, 84.7 and 76.6 p@1, given the same initial ranking from the optimal base model. To incorporate the coherence in the model, the authors used a two-stage classifier, where first they obtain an initial candidate ranking for each mention and populate a set C of content entities for candidate entity e, from the top-one candidate for each unique context name, and where the second classifier incorporates all features, including basic components and coherence. Adding coherence to the model improves the results of coherence alone to 89.2 (Wikipedia) and 81.4 (Web link), which suggests that coherence may not contribute much on top of an already strong set of basic features.

To report on the results of their experiments, they used p@1 for CoNLL and A_{KB} (knowledge base accuracy, i.e., the percentage of correctly linked non-NIL mentions). To check if link components can replace KB components, they compared both Popularity (i.e., using just entity prior and name probability features) and Context (i.e., using just BOW and DBOW features) models. The authors found that the popularity models from Wikipedia are better, while the context models from web links are better. Also, they observed that web link Popularity is significantly indistinguishable from Wikipedia Popularity on TAC data (73.3 and 72.6 respectively), which may be attributed to the fact that TAC selectively samples difficult mentions. To check if links can replace a curated KB, they performed tests over both datasets, using a basic feature set alone (i.e., the 6 Wikipedia and 4 Web link features previously mentioned) and with coherence added. The authors also found that a link data set can replace a curated KB with only a small impact on accuracy (it drops from 82.7 to 77.0 in the CoNLL dataset and from 78.6 to 78.5 in the TAC dataset), and that adding coherence improves performance in all cases (77.0 to 80.7 in the CoNLL dataset and 78.5 to 80.2 in the TAC dataset).

To check if links complement article text, they made a comparison between a standard Wikipedia-only model to a model that also includes features derived from Web link data. They found that adding Web link data has a strong impact on CoNLL dataset (82.7 to 86.1 with base features only and 86.1 to 88.7 adding coherence) while it has less impact on TAC dataset (less than 1 point of difference). The results (especially the ones obtained with CoNLL data recommend using both feature sets when available.

Finally, the authors compared their Wikipedia and Web link combinations to state-of-the-art numbers from the literature. First, they found that adding coherence to their base model results in a significant improvement on the CoNLL test data (86.1 to 88.7), but not on TAC data (79.6 and 80.7). Results on TAC data are competitive with He et al. (2013) (who achieved 81.0), while the results on CoNLL data revealed to be a new state of the art (the previous higher value was 87.6 achieved by Alhelbawy and Gaizauskas (2014)). This way, the authors found that the best base model is competitive with previous methods that use complex collective approaches to coherence (e.g., graph ranking used at Alhelbawy and Gaizauskas (2014)).

Lazic et al. (2015) presented Plato, a probabilistic model for entity resolution that includes a novel approach for handling noisy or uninformative features, also supplementing labeled training data derived from Wikipedia with a very large unlabeled text corpus (a collected Web corpus of 50 million pages from a source similar to the Clueweb09 corpus[13]) . Plato is designed to be resilient to irrelevant features (i.e., the authors assume that most of the context features of a mention are irrelevant to its disambiguation) and can be seen as a selective extension of the naive Bayes model. These assumptions contrast with the naive Bayes assumption that all features are generated from a class-conditional distribution and are thus all relevant to the class assignment.

The authors of Plato trained the system in a semi-supervised manner through an expectation-maximization (EM) algorithm.The procedure starts with labeled data, which is

[13]http://lemurproject.org/clueweb09/

derived from Wikipedia, and continues with a very large unlabeled corpus of Web documents. The use of unlabeled data enables the authors to obtain a better estimate of feature distributions and discover new features that are not present in the (labeled) training data. Plato does not include a full coherency component. Instead, mentions in a given document are sorted into coreference clusters by a simple within-document coreference algorithm, and each coreference cluster is then resolved independently of the resolution of the other clusters in the document.

The system starts with a naive Bayes model that serves as an evaluation baseline. The phrase and context of a mention are conditionally independent given the entity in the model. The authors implemented a naive Bayes model and used it for resolving entities in the CoNLL corpus (Hoffart et al. (2011b)) and the TAC KBP corpora (Ji et al. (2011)).

Plato also has a *selective context model*, which assumes that most features that appear in the context of a mention are not discriminative for entity disambiguation, i.e., they make a simplifying modeling assumption that exactly one context feature is relevant for disambiguation, and the remaining features are drawn from a background distribution. Let k be the random variable corresponding to the index of the relevant features for a mention entity e. The model can be written as

$$p(k, e, w, b) = p(w)p(e|w)p(k|e)\prod_j p(b_j|k) \tag{2.20}$$

$$p(b_j|k) = \begin{cases} b_j, & if k = j \\ \beta_j^{b_j}(1-\beta_j)^{1-b_j}, & if k \neq j \end{cases} \tag{2.21}$$

In the formula, β_j parameterizes the background probability of observing a feature that is not relevant. The authors impose the constraint that the relevant k-th feature must be on. Mentions (or mention clusters) are treated as independent and identically distributed. Let w_m represent the phrase of mention m and $\vec{b}_m$ be a binary feature presence vector to represent the context of mention m. Given a test mention $(w^{'}, b^{'})$, the entity posterior can be computed by marginalizing out the latent variable k, and thus, the entity posterior is a product of the name score $p(e|w^{'})$ and context score $\sum_k b_k^{'} p(k|e)/\beta_k$. Finally, the context score is the sum of the scores for all features present in the context.

For training, the authors proposed a memory-efficient EM algorithm, where only $q_m(e)$ (i.e., the probability that mention m resolves to entity e) is updated. The E-step is performed according to the entity posterior. In the M-step, rather than updating parameters β_j, they use empirical marginals. Notice that semi-supervised learning includes many informative context features that are not present in the contexts of labeled training mentions, which can be a very large model that might not fit in a single computer's memory. Thus, their inference algorithm is parallelizable. They partition the model across multiple servers in a distributed client-server architecture. Each entity is assigned to a server according to a heuristic algorithm and model parameters for the entity are stored on the assigned server. Each client stores a lookup a table that maps mention phrases to the servers containing entities for which the mention phrase probability is non-zero. To resolve a mention, the lookup table is consulted to identify the servers that could resolve the mention. Once an entity server receives a query for a mention m, it looks up the candidate entities for phrase w_m, retrieves model parameters and returns the entities. These lists are then merged across all the servers responsive to w_m to yield the n top-scoring entities for m.

For NIL detection, Plato decides whether to resolve a mention m to an entity in the KB or to NIL, based on context alone. The following score is first computed

$$a^*_m = \max_e \sum_k b_{m,k} \frac{p_{e,k}}{\beta_k} \tag{2.22}$$

If $a^*_m < \alpha$, being α an empirical value of $1e^{-5}$, then i is resolved to NIL. This rule is used both during EM-based learning and at inference time.

To test the system, the authors used Freebase as their KB, restricted to the set of entities that appear in both Freebase and Wikipedia. They used all pages in Wikipedia that contain a corresponding topic in Freebase as *labeled data* and, for a given Wikipedia page, they treat the target Wikipedia page of the link as the entity, and the anchor text as a mention of that entity. They also collected a Web corpus of 50 million pages for use as *unlabeled data* and considered 3 evaluation corpora: CoNLL 2003 (contains 1393 articles with about 34K mentions), TAC 2011 KBP (contains 2250 mentions of which 1123 are linkable to the reference KB), and TAC 2012 KBP (contains 2226 gold mentions, of which 1177 are linkable to the reference KB).

The experimental setup starts with mention prior. The authors initialized the mention phrase parameters from links in Wikipedia by counting how many times a given phrase w is used to refer to entity e, and normalizing appropriately. Then, to extract context features, the free text in each document was POS-tagged and dependency parsed. Next, named mentions and common noun phrases were identified using a simple rule-based tagger with rules over POS tag sequences and dependency parses, and a within-document coreference resolver was used to cluster all identified mentions. They initialized the context parameters using only labeled data, and re-estimated them using unlabeled data. Then, to determine the set of candidate entities for each coreference cluster, they use the mention with the longest phrase in the cluster and copy the label of the highest-scoring entity to all mentions in the cluster.

Regarding results, their baseline for TAC 2011 was the system from Cucerzan (2012), which achieved the highest overall accuracy and second-highest $B^{3+}F1$ score (B-cubed + measure) in the 2011 competition. They compared these results with all of their experiments (mention prior, Naive Bayes, supervised selective context model and Plato). Mention prior achieves a good performance alone, since it does not uses all the system's features and achieves 78.7 of accuracy. Supervised Naive Bayes performs better (79.6) and selective context model achieves an even higher performance (84.1). These results support the hypothesis that the performance of naive Bayes suffers in the presence of irrelevant features. Plato outperforms all the other models in accuracy, suggesting that incorporating unlabeled data helps the model generalize through feature discovery. On TAC 2011 data, Plato achieves the highest in-KB accuracy, and the same overall accuracy as the previous best system (86.5). On TAC 2011, Plato also achieves the highest in-KB accuracy and has a similar overall accuracy as to that of the system by Cucerzan (2012). For CoNLL, they discovered that coherency is a good direction for improving Plato, since the system with the highest reported accuracy performs slightly worse than Plato without coherency. Plato achieved an accuracy on the whole CoNLL 2003 dataset of 86.4.

2.2.2 Named Entity Resolution in Tweets

Performing entity linking within tweets involves different challenges. First of all, tweets are short texts (i.e., 140 characters maximum), which makes it harder to use context information to map the mentions with the correct entities. Second, tweets are a free text form, i.e., tweets usually do not follow standard spelling and grammatical conventions, so it is harder to perform a good segmentation of the tweet, since many times they are ambiguous.

Third, tweets are very noisy and have an informal nature, since they might have abbreviations (e.g., LFC for Liverpool Football Club), hashtags (e.g., #Oscars2015, #Summer) or URLs, all mixed in just 140 characters. Finally, tweets can use rich variations of named entities, which leads to many of them being hard to perform the task with the dictionaries used for recognition and/or disambiguation.

This section focuses on previous work related to Twitter entity linking and is divided into 2 subsections. First, ways to perform named entity recognition with tweets are presented, followed by ways of performing entity linking task in the second subsection.

2.2.2.1 Named Entity Recognition in Tweets

Ritter et al. (2011) found three main problems when performing NER with tweets. First, tweets contain a huge amount of distinctive named entity types (e.g., movies, companies, products), many of which are relatively infrequent. Second, due to the 140 characters limit, tweets often lack sufficient context to determine an entity's type without the aid of background knowledge. Finally, the fact that capitalization can be a key orthographic feature for the NER task and that, in tweets, capitalization is much less reliable than in structured texts.

To address these issues, the authors propose a distantly supervised approach which applies LabeledLDA (Ramage et al. (2009)) to leverage unlabeled data in addition to large dictionaries of entities gathered from Freebase, and combines information about an entity's context across its mentions. LabeledLad is a topic model that constrains Latent Dirichlet Allocation by defining a one-to-one correspondence between LDA's latent topics and user tags. The author's NER approach is divided into 2 subtasks: *segmentation* and *classification* of named entities. To perform the segmentation task, they use *part of speech tagging*, *noun-phrase chunking*, and *capitalization* as features.

Regarding part of speech tagging, they manually annotated a set of 16K tokens with tags from the Penn TreeBank tag set, for use as in-domain data for training a POS tagging system called T-POS. They also add new tags for Twitter specific features (i.e., retweets, @usernames, #hashtags, and URLs). To address the issue of out-of-vocabulary words, they perform hierarchical clustering (using Jcluster[14], a fast simple clustering program that produces hierarchical, binary branching, tree structured clusters) to group together words which are similar. Each word is uniquely represented by a bit string based on the path from the root of the resulting

[14]http://research.microsoft.com/en-us/downloads/0183A49D-C86C-4D80-AA0D-53C97BA7350A/default.aspx

hierarchy to the word's leaf (i.e., they use Brown clusters (Brown et al. (1992)) resulting from prefixes of 4, 8, and 12 bits). T-POS uses Conditional Random Fields (CRFs), because of their ability to model strong dependencies between adjacent POS tags, and also to make use of highly correlated features.

Chunking regards identifying non-recursive phrases, such as noun phrases, verb phrases and prepositional phrases in text. The chunking system is called T-CHUNK. To start, the authors annotated the same set of 16K tokens (800 tweets) with tags from the CoNLL shared task on chunking. The authors used a set of shallow parsing features (described by Sha and Pereira (2003)), in addition to the Brown clusters. POS tag features are extracted based on cross-validation output predicted by T-POS, and CRFs are again used for inference and learning.

Finally, for capitalization, they build a capitalization classifier (T-CAP), to predict whether or not a tweet is informatively capitalized. The output of this model is used as a feature for NER. Again, they manually labeled the 800 tweets as having *informative* or *uninformative* capitalization with the following criteria: if a tweet contains any non-entity words which are capitalized, but do not begin a sentence, or it contains many entities which are not capitalized, then its capitalization is uninformative, otherwise it is informative. For learning, they use SVMs.

In the system from Ritter et al. (2011), NER is divided into segmentation and classification of named entities. Theses tasks are treated separately, which allows one to more easily apply techniques better suited towards each task. For segmentation, T-SEG models named entity segmentation as a sequence-labeling task using the IOB(each word is either Inside a named entity, Outside a named entity or Begins a named entity) encoding for representing segmentations, and uses CRFs for learning and inference. In addition, Brown clusters and the outputs of T-POS, T-CHUNK, and T-CAP are used for generating features. Compared with the state-of-the-art Stanford NER (Finkel et al. (2005)), which was trained on news text, T-SEG obtains an increase of almost 52% in F1 (0.67, compared with 0.44 of Stanford NER). For classification, they leverage large lists of entities and their types gathered from an open-domain ontology (i.e., from Freebase) as a source of distant supervision, allowing the use of unlabeled data in learning. To model unlabeled entities and their possible types, the authors apply LabeledLDA (which models each entity string as a mixture of types rather than using a single hidden variable to represent the type of each mention), constraining each entity's distribution over topics based on its set of possible types according to Freebase. Each entity string in their data is associated with a bag of words found within a context window around all of its mentions, and also within the entity itself. They use Gibbs Sampling to estimate the posterior distribution over types. In order to make predictions, for each entity, they use an informative Dirichlet prior based on θ_e^{train} (i.e., distribution over topics) and perform 100 iterations of Gibbs Sampling holding the hidden topic variables in the training data fixed. To evaluate T-CLASS, the authors annotated 2400 tweets with 10 types, both popular on Twitter and with good coverage in Freebase (PERSON, GEO-LOCATION, COMPANY, PRODUCT, FACILITY, TV-SHOW, MOVIE, SPORTSTEAM, BAND, and OTHER)

In the experiments, the authors found that T-CHUNK achieves a higher accuracy (0.875 against 0.839) and a greater value of error reduction (22%) when compared to the off the shelf OpenNLP chunker[15]. T-CAP was compared against the majority baseline, and achieves higher values of Precision (0.77 against 0.70) and F1 (0.86 against 0.82), but a lower value of Recall (0.98 against 1.00). T-SEG was compared with Stanford NER and achieves higher values in Precision (0.73 against 0.62), Recall (0.61 against 0.35) and F1 (previously mentioned), being the higher values achieved when all features all included. To evaluate T-CLASS, the authors annotated 2400 tweets with 10 types popular on Twitter and with good coverage in Freebase. Also, to gather unlabeled data for inference, they ran T-SEG on 60 million tweets and kept the entities which appear 100 or more times, which results in a set of 23651 strings. Next, for each string, they collected words occurring in a context window of 3 words from all mentions in the data. T-CLASS outperforms all the other baselines (i.e., the Majority Baseline, Freebase Baseline, Supervised Baseline and DL-Cotrain (Collins and Singer (1999)) and achieves a 25% increase in F1 score over DL-Cotrain (0.66 against 0.53). To test the end-to-end performance on segmentation and classification (T-NER), the authors compared their approach with the 10 types (against COTRAIN-NER) and the MUC types (PERSON, LOCATION, and ORGANIZATION, against COTRAIN-NER and Stanford NER). For the 10 types, T-NER achieved higher values of Precision, Recall, and F1 (0.65 against 0.55, 0.42 against 0.33 and 0.51 against 0.41, respectively). For the 3 types, T-NER again achieved higher values on all metrics. It achieved 0.73 on Precision (against 0.57 from COTRAIN-NER and 0.30 from Stanford NER), 0.49 on Recall (against 0.42 from COTRAIN-NER and 0.27 from Stanford NER) and 0.59 on F1 (against 0.49 from COTRAIN-NER and 0.29 from Stanford NER).

[15]http://incubator.apache.org/opennlp/

Godin et al. (2015) proposed to use 400 million Twitter posts to generate word embeddings[16] as input features for NER. These word embeddings were used as input to a neural network to classify the words in the posts. Finally, a post-processing step is executed to check the coherence of individual tags within a named entity.

In general, a word embedding of a particular word is inferred by using the previous or future words within a number of posts/sentences. The algorithm iterates a number of times over a large dataset while updating the word embeddings of the words within the vocabulary of the dataset. The final result is a look-up table which can be used to convert every word $w(t)$ into a feature vector $w_c(t)$. In the next step, a Feed-Forward Neural Network (FFNN) is used as the underlying classification algorithm. The BIO notation is used to classify the words, and since the authors considered 10 different NE categories (each one with a Begin and Inside tag), the FFNN assigns a $tag(t)$ to every word $w(t)$ out of 21 different tags. The concatenated word embeddings $w_c(t)$ are used as the input layer of the neural network. The main design parameters of the neural network are the *type of activation function*, the *number of hidden units* and the *number of hidden layers*. The authors considered two types of activation functions, namely the *tanh* activation function and *Rectified Linear Units* (ReLUs)[17]. The output layer is a standard softmax function.

Finally, the postprocessing step to correct inconsistencies is based on 2 rules: 1) if the NE does not start with a word that has a B-tag, they select the word before the word with the I-tag and replace the O-tag with a B-tag and copy the category of the I-tag; 2) if the individual words of a NE have different categories, they select the most frequently occurring category. If it is a tie, they select the category of the last word within the NE.

For inferring the word embeddings, a set of Twitter posts was used, collected during 300 days using the Twitter Streaming API (400 million English Twitter posts). For preprocessing the posts, the authors used the tokenizer used by Ritter et al. (2011) and they replaced tokens for URL mentions and numbers. To train the model, they considered 2 phases. First the look-up table containing per-word feature vectors was constructed. To that end, they applied the word2vec[18] software on the preprocessed dataset of 400 million Twitter posts to generate word

[16]collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers in a low-dimensional space relative to the vocabulary size (”continuous space”).

[17]

[18]https://code.google.com/p/word2vec/

embeddings. Next, they trained the neural network. They used mini-batch stochastic gradient descent with a batch size of 20, a learning rate of 0.01, a momentum of 0.5 and the standard negative log-likelihood cost function to update the weights.

To evaluate the system, they used two different baselines. The word embeddings and the neural network architecture were evaluated in terms of word level accuracy. On these components, the baseline system simply assigned the O-tag to every word, which yield an accuracy of 0.9353. For the postprocessing step and the overall system evaluation, they used the baseline provided by the challenge for which the system was developed[19]. The baseline system uses lexical features and gazetteers, and, yields an F1-score of 0.3429.

For inferring the word embeddings from the 400 million posts, they followed the suggestion of Godin et al. (2014) that the best word embeddings are inferred using a Skip-gram architecture and negative sampling. The authors trained an initial version of the neural network using an input window of five words and 300 hidden nodes. They concluded that a smaller context window consistently gave a better result, since it achieves the higher M-1 value (clustering-based tagging accuracy) of 86.6. Next, they evaluated the neural network architecture. The most important parameters are the size of the input layer, the size of the hidden layers and the type of hidden units. First, they evaluate the activation function and the efectiveness of dropout ($p = 0.5$), and found that both the *tanh* function and the leaky ReLU with a leak rate of 0.01^2 activation functions perform equally good when no dropout is applied during training, but when dropout is applied, the gap between the configurations becomes larger, which lead to the conclusion that the combination of ReLUs and dropout is the best one (0.9564 accuracy). Next, they evaluated a number of neural network configurations for which they varied the input layer size and the hidden layer size, and found that the best configuration seems to be a neural network with five input words and 500 hidden nodes.

Finally, they evaluated their best model at the NE level instead of on the word level. The F1-score of their best model is 0.4515, which is an improvement of 0.11 and an error reduction of 0.1652 over the baseline provided by the challenge. Finally, they corrected the output of the neural network for inconsistencies when categorizing named entities. They found that postprocessing causes a significant error reduction, yielding a final F1-score of 0.4909 and an error reduction of 0.2252 over the baseline. In summary, their best model achieved a F1-score

[19]https://noisy-text.github.io/

of 0.5882 when detecting NEs without categorizing them and 0.4375 when categorizing the NEs into one of 10 types.

Li et al. (2012) presented TwiNER, a 2-step unsupervised NER system for targeted Twitter streams, i.e., streams constructed by filtering tweets with user-defined selection criteria (e.g., tweets published by users from a selected region, or tweets that match one or more predefined keywords. In a first step, the method leverages on global context obtained from Wikipedia and from the Web N-Gram corpus[20] to partition tweets into valid segments (phrases). Each such tweet segment is a candidate named entity. In the second step, TwiNER constructs a random walk model to exploit the local context derived from the Twitter stream. To address the problem of automatically identifying the true entities from non-entities, they learn a function that assigns a confidence score to each candidate named entity. Candidate named entities are then ranked according to this score. The highly-ranked segments have a higher chance of being true named entities. However, there is a difference between TwiNER and the other systems presented before, due to the fact that TwiNER does not categorize the entities (i.e., TwiNER does not say if a named entity is a location, a person). The 2 steps consist of *tweet segmentation* and *segmentation ranking*. For segmentation, the goal is to split an individual tweet $t \in T_i$ into m consecutive segments $t = s_1 s_2 ... s_m$ where each segment contains one or more words. To obtain the optimal segmentation, the authors use the following objective function

$$\arg\max_{s_1 ... s_m} \mathrm{C(t)} = \sum_{i=1}^{m} C(s_i) \tag{2.23}$$

In the equation, C is the function that measures the *stickiness* of a segment or a tweet defined based on word collocations. If the word length of tweet t is l, there are 2^{l-1} possible segmentations, and it is inefficient to iterate over all of them and compute their stickiness. Thus, the authors use a dynamic programming algorithm to tackle this problem. The basic idea is to recursively conduct binary segmentations and then evaluate the stickiness of the resultant segments. Formally, given any segment s from t and $s = w_1, w_2, ...w_n$, their solution is to conduct a binary segmentation by splitting into two adjacent segments $s^1 = w_1, ..., w_j$ and $s^2 = w_{j+1}, ..., w_n$ by satisfying

$$\arg\max_{s^1, s^2} \mathrm{C(t)} = C(s^1) + C(s^2) \tag{2.24}$$

[20]http://research.microsoft.com/en-us/collaboration/focus/cs/web-ngram.aspx

Regarding the stickiness function C, a high score for a segment s indicates that further splitting segment s would break the correct word collocation. In this work, they use two generalized collocation measures: *Point Manual Information* (PMI) and *Symmetric Conditional Probability* (SCP). PMI measures the degree that two words occur together more often than by chance. Mathematically, PMI for a segment $s = w_1...w_n$

$$\text{PMI}(s) = \log \frac{P(w_1...w_n)}{\frac{1}{n-1}\sum_{i=1}^{n-1} P(w_1...w_i)P(w_{i+1}...w_n)} \tag{2.25}$$

SCP measures the cohesiveness of bigram w_1w_2 by considering both conditional probabilities for the bigram given each single term. Given a segment $s = w_1...w_n$, SCP of s is defined of similarity by

$$\text{SCP}(s) = \log \frac{P(s)^2}{\frac{1}{n-1}\sum_{i=1}^{n-1} P(w_1...w_i)P(w_{i+1}...w_n)}, SCP(s) \in (-\infty, 0) \tag{2.26}$$

So far, the calculation of the stickiness is reduced to estimating P for the prior probabilities of segments $P(s), P(s^1)$ and $P(s^2)$ for any segment $s \in t$, which are prior probabilities of segments. To accurately estimate these prior probabilities, they needed a large enough corpus as the global context of each segment. Thus, they exploited the one provided by Microsoft Web N-Gram Services[21] as approximation. In addition, and to tackle the problem of only using Web documents as a priori knowledge, they leverage a knowledge base in the World Wide Web (Wikipedia), as another source of global context. Now, the stickiness function is defined as follows

$$C^{'}(s) = C(s) \cdot e^{Q(s)} \tag{2.27}$$

In the equation, $Q(s)$ is the probability that s appears as anchor text in its mentioning Wikipedia article and $C(s)$ is measured by exploiting the global context captured in the Microsoft Web N-Gram corpus. Finally, they introduce length normalization $\mathscr{L}(s)$ to the stickiness function, to favor moderately long segments in both Web N-gram and Wikipedia. Thus, the final stickiness function is as follows:

$$C^{'}(s) = C(s) \cdot e^{Q(s)} \cdot \mathscr{L}(s) \tag{2.28}$$

For the step of segmentation ranking, the authors first start by performing a noise filtering,

[21]https://www.microsoft.com/cognitive-services/en-us/web-language-model-api

where they remove three types of segments which are not named entities: segments containing slang words (like *lol* and *omg*) are removed using a compilation of internet slangs[22]. Segments containing words with consecutive repeating characters (like *hahahaha* or *noooooo*) are removed with the help of regular expressions. Finally, the segments containing words with # as prefix (i.e., hastags like #*summer* or #*nw*) are removed by locating the # prefix. Then, to tackle the problem of the global context alone being insufficient to recognize a named entity, they use the local context of a segment in tweets. They exploit a *gregarious* property that exists among named entities in targeted twitter streams. This property refers to the interaction between entities and their co-existence among the targeted twitter streams. For example, if *Barack Obama* is a true named entity, it often occurs with *Michelle Obama* and *United States*, but rarely with *please look* (which is a valid segment but not a named entity). So, at the i-th interval they build an undirected segment graph $G(V, E)$ using all segments V extracted from the tweet set T_i. Each node in the graph is a valid segment after noise filtering and the edge $e_{a,b} \in E$ between two segments s_a and s_b is weighted by the Jaccard index

$$w_{ab} = w(e_{ab}) = \frac{|M(s_a) \cap M(s_b)|}{|M(s_a) \cup M(s_b)|} \tag{2.29}$$

In the equation, $M(s)$ is the set of tweets in T_i containing segment s. Finally, a random walk model is applied on graph $G(V, E)$ to compute the stationary probability of each segment being a true named entity, by considering the graph bidirectional.

To test the system, the authors started by gathering collections of tweets to simulate targeted twitter streams. The first collection is *SIN* and was collected to simulate a targeted twitter stream of one particular geolocation by monitoring the tweets from a number of Singapore-based users published in June 2010. The second is *SGE* and was collected to simulate a targeted twitter stream of one particular event by monitoring a set of predefined keywords related to Singapore General Election 2011. For both collections, 5000 tweets were randomly sampled from the tweets published on a random day. Each tweet was then labeled by two human annotators, which leaves 4422 tweets for SIN and 3328 for SGE (which will now be referred as SIN_g and SGE_g).

Regarding tests, the authors first compared TwiNER with Stanford-NER, LBJ-NER[23] and T-NER (Ritter et al. (2011)). They found that 1) the overall performance is much better with

[22]http://www.noslang.com/dictionary/
[23]http://cogcomp.cs.uiuc.edu/m2repo/edu/illinois/cs/cogcomp/lbj-ner-tagger/

SGE_g that with SIN_g (0.929 against 0.419 on Precision, 0.660 against 0.329 on Recall and 0.772 against 0.419). The main reason is the impact of error-prone local context of tweets, since SIN_g has more tweets of informal style; 2) TwiNER achieves comparable F1 performance with the other supervised systems; 3) TwiNER performs much better than LBJ-NER and T-NER on SGE_g (outperforming these systems both on Recall (0.660 against 0.595 form LBJ–NER and 0.359 from T-NER) and F1 (0.772 against 0.727 from LBJ-NER and 0.478 from T-NER) although loses on Precision (0.929 against 0.933 from LBJ-NER and 0.713 from T-NER); and 4) TwiNER achieves a comparable performance with Stanford-NER on SIN_g, in terms of the F1 metric (0.419 against 0.423).

Next, they tested the performance of tweet segmentation. They studied the impact of the collocation measures (PMI or SCP), the Wikipedia dictionary (Wiki) and the length normalization (Norm) factor based on the ground truth in SIN_g and SGE_g. They measured the percentage of named entities that are correctly extracted as the performance metric Precision. They observed that: 1) SCP significantly outperforms PMI for tweet segmentation; 2) Norm is effective and improves the accuracy of tweet segmentation in the two collections for SCP-based stickiness; 3) Wikipedia's broad coverage and high quality knowledge helps reclaim incorrect decisions made by the lexical statistics or reinforce the correct decisions. For example, a word that might not be extracted with SCP can be extracted when SCP+Wiki is used; 4) The combination of Wiki and Norm boosts up the performance of tweet segmentation of SCP-based stickiness.

Next, the authors analyzed the impact of the random walk process on the performance of segment ranking. They investigated 5 schemes for segment ranking: 1) MFS, a naive method that ranks the segments based on their frequency in the collection; 2) Wiki, a naive method that ranks the segments based on their Wikipedia-based teleportation probability; 3) RW, a simple random walk with uniform teleportation; 4) RWW, a random walk with Wikipedia-based teleportation; and 5) RWW+Wiki, a random walk with Wikipedia-based teleportation, while the segments are ranked based on

$$y(s) = e_s \cdot \pi(s) \tag{2.30}$$

They concluded that: 1) MFS works considerably well in SGE_g, but its performance degrades significantly on SIN_g; 2) RW outperforms MFS significantly on SGE_g in terms of F1, which validates their assumption that named entities are more likely to co-occur with one another than with non-entities. 3) Wiki achieves an impressive performance, since it achieves the second best results in F1 on both collections (0.415 for SIN_g against the higher value of 0.423 obtained by RWW+Wikiv, and 0.712 for SGE_g against the higher value of 0.762 obtained by RWW+Wiki), which helps concluding that random walk with Wikipedia-based teleportation boosts up many named entities that may not be covered by Wikipedia, by leveraging the cooccurrence of local context. In addition, combining the random walk with Wiki achieves the higher F1 results (0.423 for SIN_g and 0.762 for SGE_g)

Finally, they tested the impact of the K value,i.e., the parameter in the segment ranking step. Since TwiNER's performance is related to the choice of K, larger K will increase Recall and decrease Precision, and vice-versa. Thus, they calculate $Prec@K$ (the percentage of top K segments returned by TwiNER that are real named entities) for each K value form 1 to 200. They concluded that major proportion of segments returned when $K < 50$ are true named entities. TwiNER achieves a stable Precision performance when K is in the range of $[50, 100]$, and after 100, $Prec@K$ starts to degrade slowly on SGE_g, while $Prec@K$ is still stable at around 0.5 on SIN_g.

2.2.2.2 Entity Linking in Tweets

Liu et al. (2013) developed a collective inference method that jointly links a set of tweet mentions to their corresponding entities. Procedures for collective inference make simultaneous statistical judgments about the same variables for a set of related data instances. The main innovation in this work is that they integrate 3 kinds of similarity (i.e., mention-entity, entity-entity, and mention-mention) to enrich the context for entity linking and to address irregular mentions that are not covered by the entity-variation dictionary, called out-of-vocabulary mentions. Given as input $\vec{M}$ = (m_1, m_2, ..., m_{n}), the system outputs the entity sequence $\vec{E}^*$ = (e_1^*, e_2^*, ..., e_{n}^*) using

$$\vec{E}^* = \mathrm{argmax}_{\forall \vec{E} \in C(\vec{M})} \lambda \sum_{i=1}^{n} \vec{w} \cdot \vec{f}(e_i, m_i) + (1-\lambda) \sum_{i \neq j} r(e_i, e_j) s(m_i, m_j) \tag{2.31}$$

In the equation, C($\vec{M}$) is the set of all possible entity sequences for the mention sequence $\vec{M}$, $\vec{E}$ denotes an entity sequence instance, $\vec{f}(e_i, m_i)$ is the feature vector that models the mention-entity similarity, $\vec{w}$ is the feature weight vector related to $\vec{f}$, $\vec{r}(e_i, e_j)$ is the function that returns the entity-entity similarity, $\vec{s}(m_i, m_j)$ is the function that returns the mention-mention entity and $\lambda \in (0,1)$ is a systematic parameter (i.e., a parameter to guarantee the stability of the whole control system), determined on development data set, used to adjust the trade off between local compatibility and global consistence (experimentally set to 0.8 in the work).

To represent the search space, the authors built an inverted index of all entity variation lists. Then, for any mention m, they look up the index, and get all possible entities, denoted by $C(m)$. If $|C(m)| = 0$, they have an OOV mention. To solve this, they generate a list of candidates for an OOV mention using its similar mentions ($S(m)$) and, if still $C(m) = 0$, they remove m from $\vec{M}$.

As stated before, this system integrates 3 kinds of similarity scores. Regarding *mention-entity* similarity (i.e., the local features), the authors use *Prior Probability*, *Context Similarity*, *Edit Distance Similarity*, *Mention Contains Title* (i.e., if the mention contains the entity title this function returns 1 and 0 otherwise) and *Title Contains Mention* (i.e., if the entry title contains the mention returns 1, 0 otherwise). Regarding *entity-entity* similarity, they adopted in-link-based similarity. Finally, for *mention-mention* similarity, they define 5 features: (1) the cosine similarity of $t(m_i)$ and $t(m_j)$ with tweets represented as TF-IDF vectors, (2) the cosine similarity of $t(m_i)$ and $t(m_j)$ with tweets represented as topic distribution vectors, (3) whether $t(m_j)$ and $t(m_i)$ are published by the same account, (4) whether $t(m_i)$ and $t(m_j)$ contain any common hashtags and (5) edit distance-related similarity between m_i and m_j. In the end, they use the following formula to integrate all features

$$s(m_i, m_j) = \sum_{k=1}^{5} a_k s_k(m_i, m_j) \tag{2.32}$$

To test the system, the authors used Wikipedia as the knowledge base. An inverted index with about 60 million entries for the entity variation lists is also built. For tweets, they used a data set annotated manually[24]. First, they compared their method with Wikify! (Mihalcea and Csomai (2007)) and with the method proposed on Meij et al. (2012), concluding that their system outperforms both these baselines in terms of Precision (0.752 against 0.734 from Meij and 0.375 from Wikify!), Recall (0.675 against 0.632 from Meij and 0.421 from Wikify!) and F1 (0.711 against 0.679 from Meij and 0.396 from Wikify!).

Second, they evaluated the method adding local features, where they found that using only the

[24]http://edgar.meij.pro/dataset-adding-semantics-microblog-posts/

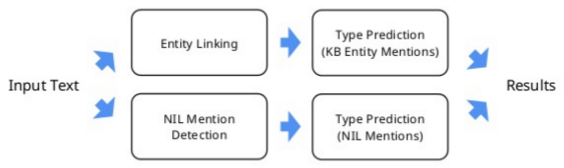
Input Text
Entity Linking
NIL Mention Detection
Type Prediction (KB Entity Mentions)
Type Prediction (NIL Mentions)
Results

Figure 2.1: System Architecture

Prior Probability feature already yields a reasonable F1 (0.646) and that Context Similarity and Edit Distance Similarity features have little contribution to the F1 (0.642 and 0.643, respectively), in contrast with Mention and Entity Title Similarity features, which boost the F1 metric (0.680).
Third, they evaluated their method using various mention similarity features (previously mentioned), and found that these features improve F1 significantly (from 0.680 to values between 0.700 and 0.704, depending on what features are used). Combining all 5 features yields the best performance (0.704). Finally, they made experiments with out-of-vocabulary (OOV) mentions. To do so, they used the strategy of guessing their possible entity candidates using similar mentions. They found that with their OOV strategy, Recall improves (0.653 to 0.675), while Precision is slightly dropped (0.764 to 0.752) and the overall F1 gets better (0.704 to 0.711).

Yamada et al. (2015a) developed a Twitter-specific entity linking architecture based on 4 supervised machine-learning models as show in Figure 2.1. The system consists in 5 steps: 1) preprocessing, 2) mention candidate generation, 3) mention detection and disambiguation, 4) NIL mention detection and 5) type prediction. In Step 1, they tokenize a tweet and assign part-of-speech tags to the resulting tokens.
In Step 2, the candidates of the entity mentions are generated from the tweet. First, the system uses a mention-entity dictionary that maps each mention surface form to the possible referent entities. This way, the system generates candidates using a mention-entity dictionary. To do so, it first takes all the n-grams ($n < 10$) from the tweet to perform queries to the dictionary using the text-surface of each of the n-grams. Then, they use 4 methods to deal with mentions in irregular forms (i.e., exact search, fuzzy match, approximate token search and acronym search).
In Step 3, the authors use supervised machine-learning to classify mention candidates, using *random forests* (Breiman (2001)) as the machine-learning algorithm. Here they also introduced some features to enhance performance, like string similarity measures (i.e., to measure the

similarity between the title of the entity and the mention) or temporal popularity knowledge of an entity extracted from Wikipedia page view data.
Step 4 is dedicated to NIL mention detection, where they use supervised machine-learning (again with random forests) to assign a binary label to each of all possible n-grams. Again, they use some features to enhance the performance, like the ratio of capitalized words in the tweet or the output of Stanford NER.
Finally, Step 5 is divided into 2 separate classification steps to predict the entity types for each type of mention: KB entity mentions (type of an entity that exists in the KB) and NIL mentions (type of an entity that does not exist in the KB). Both systems use logistic regression and random forest classifiers, and a final model corresponds to an ensemble model of these methods. Again, features specific to these tasks are used (KB entity classes for KB entity mentions, word embeddings for NIL entity mentions and NER entity types for both).
To evaluate the system, they used the #Microposts2015 dataset[25] split into a training set (3498 tweets) and a test set (500 tweets). The system was evaluated with 3 measures: *strong_link_match* to evaluate the performance of linking entities, *strong_typed_mention_match* to measure the performance of mention detection and entity typing and *mention_ceaf* for calculating the performance of clustering detected mentions into entity mentions or NIL mentions. *Strong_link_match* achieves 0.786 on Precision, 0.656 on Recall and 0.715 on F1. *Strong_typed_mention_match* achieves 0.656 on Precision, 0.630 on Recall and 0.642 on F1. Finally, *mention_ceaf* achieves 0.857 on Precision, 0.823 on Recall and 0.840 on F1.

In a subsequent study, Yamada et al. (2015b) proposed a novel method to enhance the performance of named entity recognition in Twitter by using an end-to-end entity linking approach (i.e., both segmentation and classification tasks are performed together). Typically, entity linking is performed *after* NER. However, the authors found that performing entity linking *before* enhances the performance of NER, because resolving the entity mentions to the KB entries enables them to use the high-quality knowledge in the KB (e.g., they can use the popularity of the entity, the classes of the entity).
The system consists of 3 steps: 1) pre-processing, 2) entity linking and 3) named entity recognition. Steps 1 and 2 are equal to the first 3 steps of Yamada et al. (2015a). Step 3 is the innovation in this paper, where NER is performed. This step is divided into 2 different

[25]http://scc-research.lancaster.ac.uk/workshops/microposts2015/challenge/index.html

stages: *segmentation* and *classification of named entities*. For *segmentation*, they use some features based on the results of entity linking, like *EL relevance score* or *link probability*. For *classification*, the only feature from EL that is used is the *EL relevance score*. The authors concluded that data retrieved from entity linking is highly effective for segmentation and classification of named entities, which shows that entity linking can be used to enhance the performance of NER.
To train the EL method, they used the #Microposts2015 dataset (which contains 3998 tweets and 3993 annotations of entities). Testing their segmentation task and comparing the results with the 5 top-ranking systems, their system clearly outperforms all of them in Precision (0.7220 against the second higher value of 0.6774), Recall (0.6914 against the second higher value of 0.5628) and F1 (0.7063 against the second higher value of 0.6029). Comparing the results with the same systems, but this time in terms of end-to-end results (i.e., combining both segmentation and classification), their system had better Recall (0.5522 against the second higher value of 0.4112) and F1 values (0.5641 against the second higher value of 0.5140), but ranks second in Precision (0.5766 against the higher value of 0.6362). Finally, they tested the performance of the system in both segmentation and classification tasks, broken down by entity types. They found that Precision and F1 achieve the higher value in the *person* category (0.7097 and 0.7395, respectively) and lower value in *tvshow* (0.1429 and 0.2222, respectively), while Recall achieves the higher value in *geo-loc* (0.7845) and lower in *product* (0.2162).

Guo et al. (2013) proposed a structural SVM algorithm for entity linking that merges the tasks of mention detection and entity disambiguation into a single end-to-end entity linking approach. The difference between a SVM and a structural SVM is that whereas the SVM classifier supports binary classification, multiclass classification and regression, the structured SVM allows training of a classifier for general structured output labels (which is an umbrella term for supervised machine learning techniques that involves predicting structured objects, rather than scalar discrete or real values). This way, they are able to apply a large set of features that are conventionally used only for the initial detection of mentions.
The system starts by generating candidate entity mentions. Given a tweet t, they extract all k-grams of size $\leqslant k$. Then, for each k-gram, they find all entities where the k-gram is an anchor phrase[26]. If the k-gram is an anchor phrase for at least one entity, then it also constitutes

[26]The anchor text, link label, link text, or link title is the visible, clickable text in a hyperlink

a candidate entity mention. Let $U(t)$=$\{c_1$,c_2,...$\}$ denote the set of candidates in tweet t, c_i a candidate, y_i the output of c_i, where $y_i \in E(c_i)$, $E(ci)$=$\{e_1$,e_2,...,NIL$\}$ denotes the set of entities which candidate i may be linked to, plus NIL, $T = |U(t)|$ and y = $\{y_1$,y_2,...,$y_T\}$. After generating the candidates, the system performs filtering and optimization to process the list of candidates (i.e., formally, given a tweet **t** and $U(t)$, the system predicts $y_i \in$ E(c_i), $\forall_{c_i} \in U(t)$). As stated before, the system uses structural learning to capture the relationship between entities. The learning algorithm that is used is a Structural SVM (SSVM), where to train the weight vector w, the authors minimize the following objective function

$$\underset{\mathbf{w}}{min} \frac{||w||^2}{2} + C\sum_{i=1}^{l} \xi_i^2 \tag{2.33}$$

In the equation, l is the number of labeled examples, w is the weight vector and ξ is a slack variable
Regarding features, the authors define their feature vector as follows

$$\Phi(t, U(t), y) = \sum_i \phi(t, c_i, y_i) + \sum_{i<j} \phi(t, c_i, y_i, c_j, y_j) \tag{2.34}$$

In the equation, c_i and c_j are the i-th and j-th candidates in $U(t)$. There are also first and second order features for $\Phi(t, c_i, e)$. The first order ones can be classified into 2 types: *mention-specific* (only consider the surface form of c_i and the tweet t) and *entity-specific* (consider the knowledge base content of entity e). The authors considered *base and capitalization rate features* (like normalized link count or number of tokens), *popularity features* (like normalized page view count), *context capitalization features* (3 features indicating if the current candidate and the words before and after are capitalized), and *entity type* (six binary features for each entity type), and *tf-idf* features (2 features for the similarity between the word vectors of the entity and the tweet). In terms of second order features, they include 4 features to capture relations between entities and candidates (like the Jaccard measure).
To address the difficulty of using lexical features (i.e., they do not have many labeled examples, which can lead to overfitting, also because of the diverse language in tweets), they use a very simple method to mine context words for different entities from an unlabeled tweet corpus. The algorithm involves the following steps: 1) train an end-to-end entity linking system and then apply it to a large, unlabeled tweet corpus; 2) extract contextual words for each entity

type based on the pseudo-labeled data; and 3) train the entity linking system again with new contextual features (where they add an additional feature for the contextual words).

In addition to the second order features, the authors also consider a modified *cohesiveness* score, to estimate the correlation between different entities by using weighted Jaccard scores (i.e., in the experiments, the authors only applied the cohesiveness score technique on candidates which pass the filtering procedure). They first estimate the most probable entity for each candidate given all the other candidates in the same tweet. Then, the cohesiveness score is produced with respect to the most most probable entity computed in the first round. With the cohesiveness score, they increased the disambiguation ability of the model without using the second-order information.

Due to the fact that the inference problem becomes NP-hard with the second order features, the authors used a beam search algorithm that first arranges the candidates from left to right, and then solves the inference problem approximately.

To test the system, the authors collected unlabeled Twitter data from two resources and then asked human annotators to label each tweet with the set of entities present. Duplicate entities per tweet, ambiguous entity mentions and entities not present in Wikipedia were ignored. They used 2 data sets, plus a set of 200 randomly sampled tweets (which were used as a development set). The first set consists in 473 (train set, named Train) and 500 tweets (named Test 1) from the data used in Ritter et al. (2011). The second set is used to check if the system has the ability to generalize across different domains (from election results and democracy movements to health issues and disease spreading, as well as tracking product feedback and sentiment) and consists in 488 tweets related to entertainment entities and is called Test 2. The tweets were then processed (e.g., remove retweet symbols, hashtags) and the evaluation focus primarily on six categories (i.e., person, location, organization, TV show, book/magazine and movie).

The first test they performed compared their system with other state-of-the-art systems (Cucerzan and TagMe[27]). With both the Test 1 and Test 2 sets, their system outperforms the other in Precision (0.788 against 0.648 from Cucerzan and 0.388 from TagMe for Test 1; 0.750 against 0.649 from Cucerzan and 0.349 from TagMe for Test2) and F1 (0.680 against 0.511 from Cucerzan and 0.497 from TagMe for Test 1; 0.652 against 0.495 from Cucerzan and 0.467 from TagMe for Test2), but is behind TagMe in Recall (0.599 against 0.422 from Cucerzan and 0.690 from TagMe for Test 1; 0.577 against 0.397 from Cucerzan and 0.703 from TagMe for

[27]http://tagme.di.unipi.it/

Test2). The second test concerned with calculating F1 scores to both data sets by studying the contribution of each feature group in the system. They found that adding capitalization rate to the base features already improves F1 (0.359 to 0.384 for Test 1; 0.477 to 0.499 for Test 2), and that the entity related features (i.e., popularity, entity type and Tf-Idf) are crucial and that F1 achieves higher values when all features are used (0.532 for Test 1; 0.568 for Test 2).
Also, by finding that entity-specific are among the most important features, they proved their hypothesis that addressing mention detection and entity disambiguation as a single problem, rather than two separate problems, can bring benefits. Finally, they tested the system adding mined context, cohesiveness and second order features (all previously detailed). Adding the mined context to the system improves the F1 performance (0.532 to 0.539 for Test 1; 0.568 to 0.586 for Test 2), which also occurs with cohesiveness (0.556 for Test 1; 0.597 for Test 2) and second order features (0.581 for Test 1; 0.606 for Test 2).

Shen et al. (2013), proposed a system called Kauri, which integrates the intra-tweet local information with the inter-tweet user interest information into a graph-based framework to collectively link all the named entity mentions in all tweets posted by a user, by modeling the user's topics of interest.
Intra-tweet local information considers three key properties: 1) the *prior probability* of the entity being mentioned is high; 2) the *similarity* between the context associated with the candidate mapping entity is high; and 3) the candidate mapping entity is topically *coherent* with the mapping entities of the other entity mentions within the same tweet.
Inter-tweet exploits the user interest information across tweets published by one individual user by modeling the user's topics of interest. This way, if a candidate mapping entity is topically related to entities the user is interested in, they assume this user is likely to be interested in this candidate entity as well. For example, consider 2 tweets from a user, the first being *FC Barcelona must win La Liga this year* and the second *Leo and Luis make a perfect duo.* Since *FC Barcelona* is the entity the user is mentioning on the first tweet, the authors assume that the user is interested in that entity, so, we can assume is more likely that *Leo* and *Luis* refer to Leo Messi and Luis Suarez, rather than another Luis or Leo.
To perform entity linking, the authors first make 3 assumptions: 1) Each Twitter user has an underlying topic interest distribution over various topics of named entities, i.e., for each Twitter user, each named entity is associated with an interest score, indicating the strength of

the user's interest in it (therefore Assumption 1); 2) If some named entity is mentioned by a user in his tweet, that user is likely to be interested in this named entity (therefore Assumption 2); 3) If one named entity is highly topically related to the entities that a user is interested in, that user is likely to be interested in this named entity (therefore Assumption 3). Kauri's entity linking approach is divided into 3 steps. First, for each Twitter user, they construct a graph encoding the interdependence information between different named entities. Next, they estimate the initial interest score for each named entity in the graph based on the intra-tweet local information (i.e., according to Assumption 2). Finally, they propose a graph-based algorithm (i.e., user interest propagation) to propagate the user interest score among different named entities across tweets using the interdependence structure of the graph, according to Assumption 3. This way, the system integrates the intra-tweet local information with the inter-tweet user interest information into a unified graph-based model via modeling the user's interest.

Regarding the graph construction, for one Twitter user, let R be the candidate mapping entity set. They construct a graph $G = (V, A, W)$ to represent all the interdependence information between these candidate mapping entities in R, where V denotes the set of nodes, A is the set of edges and $W : A \to R^+$ is the weight function which assigns a positive weight to each edge in A. One node in the graph represents a candidate mapping entity $r^i_{j,q}$ (i.e., candidate mapping entity with index q in R^i_j) and is associated with an interest score $s^i_{j,q}$, which indicates the strength of the user's interest in each node is also associated and with an initial interest score $p^i_{j,q}$ (estimated from the intra-tweet local information). Then, for each pair of nodes $(r^i_{j,q}, r^{i'}_{j',q'})$ in the graph, if the weight between them is larger than 0, they add an undirected edge $(r^i_{j,q}, r^{i'}_{j',q'})$ to A, with $W(r^i_{j,q}, r^{i'}_{j',q'})$ indicating the strength of interdependence between them. If the two candidate entities of the node pair are candidates of the same entity mention (i.e., $i = i' and j = j'$), a edge is not added (since only one can be the true mapping entity). The edge weight is defined as the *topical relatedness* between the two candidate entities. In the framework, since all the candidate entities are Wikipedia articles, the topical relatedness (TR) is defined by Equation 2.11.

After the graph is built, the next stage of Kauri is the initial interest score estimation. The initial interest score $p^i_{j,q}$ for each candidate mapping entity $r^i_{j,q} \in R^i_j$ is calculated as the weighted sum of the 3 intra-tweet features (i.e., prior probability, context similarity and topical coherence). Prior probability $Pp(r^i_{j,q})$ for each candidate mapping entity $r^i_{j,q} \in R^i_j$ with respect to the entity mention m^i_j as the proportion of the links with the surface form m^i_j, is defined as the anchor text which point to the candidate mapping entity $r^i_{j,q}$

$$Pp(r^i_{j,q}) = \frac{\text{count}(r^i_{j,q})}{\sum_{c=1}^{|R^i_j} \text{count}(r^i_{j,c})} \tag{2.35}$$

In the formula, $count(r^i_{j,q})$ is the number of links which point to entity $r^i_{j,q}$ and have the surface form m^j_i as the anchor text. To calculate context similarity $\text{Sim}(r^i_{j,q})$ for each candidate mapping entity $r^i_{j,q}$, the authors compare the context associated with the candidate entity $r^i_{j,q}$ with the context around the entity mention m^j_i in tweet t_i. For each candidate entity $r^i_{j,q}$, the authors collect each occurrence of entity $r^i_{j,q}$ in the Wikipedia page corpus, and extract the context of $r^i_{j,q}$ in a short window to compose a bag of words vector for $r^i_{j,q}$. To get the bag of words vector, the authors first pre-process the tweet and then extract the short window of words around each occurrence of $r^i_{j,q}$ in tweet t_i to compose the bag of words vector for entity mention m^i_j. Finally, the authors measure the cosine similarity of the two vectors (bag of words vector for $r^i_{j,q}$ and bag of words vector for entity mention m^i_j) weighted by TF-IDF as the context similarity. Finally, for a given tweet t_i where an entity mention m^i_j appears, the authors formally define the notion of topical coherence $\text{Coh}(r^i_{j,q})$ for each candidate mapping entity $r^i_{j,q} \in R^i_j$ as

$$\text{Coh}(r^i_{j,q}) = \frac{1}{|M^i|-} \sum_{c=1,c\neq j}^{|M^i|} \text{TR}(r^i_{j,q}, e^i_c) \tag{2.36}$$

In the formula, $|M^i|$ is the number of entity mentions in tweet t_i, e^i_c is the mapping entity for the entity mention $m^i_c \in M^i$ and $\text{TR}(r^i_{j,q}, e^i_c)$ is the topical relatedness between the candidate entity $r^i_{j,q}$ and the mapping entity e^i_c for other entity mention $m^i_c \in M^i$ (where $c \neq j$) which can be calculated by Formula **??**.

Finally, the initial interest score is calculated by:

$$p^i_{j,q} = \alpha * \text{Pp}(r^i_{j,q}) * \beta * \text{Sim}(r^i_{j,q}) + \gamma * \text{Coh}(r^i_{j,q}) \tag{2.37}$$

In the equation α, β and γ are weight factors that give different weights for different feature values in the formula ($\alpha + \beta + \gamma = 1$).

A user interest propagation algorithm is used to propagate the interest score among different candidate mapping entities across tweets using the interdependence structure of the constructed graph. First, they define the interest propagation strength $\mathrm{NW}(v_k, v_k^{'})$ from node v_k to $v_k^{'}$ as follows:

$$\mathrm{NW}(v_k, v_k^{'}) = \frac{W(v_k, v_k^{'})}{\sum_{v_c \in V_{v_k}} W(v_k, v_c)} \tag{2.38}$$

In the equation, V_{v_k} is the set of nodes, each of which has an edge with node v_k. To compute the final interest score vector $\vec{s} = (s_1, ..., s_k, ..., s_{|V|})$, where s_k is the final interest score for node v_k, that indicates the strength of the user's interest in it, the authors used the following procedure:

$$\vec{s} = \lambda \vec{p} + (1 - \lambda) B \vec{s} \tag{2.39}$$

In the equation, $\lambda \in [0, 1]$ is a parameter that balances the relative importance of $\vec{p}$ (i.e., initial interest score) and $B\vec{s}$ (i.e., the propagated score of other related notes) and B is a $|v| \times |V|$ interest propagation strength matrix. Finally, with the final interest score $s_{j,q}^{i}$ calculated, the mapping of entity e_i^j for mention m_i^j can be identified

$$e_i^j = \arg \max_{r_{j,q}^i \in R_j^i} s_{j,q}^i \tag{2.40}$$

To test the system, the authors created a gold standard data set for the Twitter EL task. They used the Twitter API to collect the 3200 most recent tweets for each of a set of randomly sampled 71937 Twitter users. To create the gold standard, they randomly sampled 20 non-verified Twitter users and, for each sampled user, they annotated the 200 most recent tweets. For the Twitter user who has fewer than 200 tweets they annotated all his tweets, and finally obtained 3818 tweets from 20 users forming the gold standard set.

To construct the knowledge base D, the authors downloaded the May 2011 version of Wikipedia. They first tested the accuracy and the number of correctly linked entity mentions for all mentions with different types (linkable, unlinkable and all). They compared the results from Kauri with LINDEN (Shen et al. (2012)) and LOCAL (which is a truncated Kauri framework, with the initial interest score as the final score). For Kauri and LOCAL they not only tested the performance by leveraging all the intra-tweet local features (refered as $LOCAL_{full}$ and $KAURI_{full}$), but also present their performance by leveraging a subset of the intra-tweet local features. For instance, $LOCAL_{\beta=0,\gamma=0}$ and $KAURI_{\beta=0,\gamma=0}$ means that the initial interest score calculated using Formula 2.37 sets $\beta = 0$ and $\gamma = 0$. Instead of Precision, Recall and F1 metrics, Kauri

is evaluated in terms of Accuracy, defined as the number of correctly linked entity mentions divided by the total number of all entity mentions.
They concluded that $KAURI_{full}$ outperforms LINDEN (0.858 against 0.824) and all the different configurations of LOCAL (0.836 against 0.799 for $\beta = 0, \gamma = 0$; 0.841 against 0.803 for $\gamma = 0$;0.853 against 0.828 for $\beta = 0$; 0.858 against 0.829 for $full$). They also found that the difference between LOCAL and Kauri means the inter-tweet user interest information is very important and that their algorithm is very effective for the tweet EL task and integrating intra-tweet local information with the inter-tweet user interest information considerably boosts the performance. Finally, they concluded that every intra-tweet local feature has a positive impact on the performance of KAURI and LOCAL, and with the combination of all intra-tweet local features $KAURI_{full}$ and $LOCAL_{full}$ can obtain the best results. The highest value achieved by the system was 0.858.

2.3 Overview

This section presented fundamental concepts and related work for entity linking. The related work is divided into two main categories: general entity linking and named entity resolution in tweets. For the first category, we observed four different systems. The first was AIDA, the system by Yosef et al. (2011), which is a robust framework centered around collective disambiguation, exploiting the prominence of entities, similarity between the context of the mention and its candidates, and the coherence among candidate entities for all mentions. AIDA was the system chosen for this work, so further details about it are presented in the next chapter. Lin et al. (2012) investigated entity linking over millions of high-precision extractions from a corpus of 500 million Web documents, with the goal of crating a useful knowledge base of general facts. Chisholm and Hachey (2015) explored entity linking with web links versus linking with Wikipedia. Their evaluation was done with the CoNLL and TAC data sets. The authors found out that Wikipedia popularity models are better, but web link context models are better and that a link data set can replace a curated KB with only a small impact on accuracy. At last, Lazic et al. (2015) presented a probabilistic model for entity resolution that includes a novel approach for handling noisy or uninformative features, also supplementing labeled training data derived from Wikipedia with a very large unlabeled text corpus (a collected Web corpus of 50 million pages). On TAC 2011 data, Plato achieves the highest in-KB accuracy, and the same overall accuracy as the previous

	Approach(es) used	Precision	Recall	F1	Accuracy
Yamada et al. (2015a)	Supervised-Machine-Learning	0.857	0.823	0.840	-
Yamada et al. (2015b)	Supervised-Machine-Learning	0.5766	0.5522	0.5641	-
Guo et al. (2013)	Supervised-Machine-Learning SVM	0.788	0.599	0.680	-
Liu et al. (2013)	Collective Inference	0.764	0.675	0.711	-
Shen et al. (2013)	Graph-based Framework	-	-	-	0.858

Table 2.4: Approaches used by all the methods and higher values achieved for each metric

best system (86.5), while on CoNLL 2003 dataset it achieves an accuracy of 86.4.

The second category is divided into two subcategories: named entity recognition in tweets and entity linking in tweets. For named entity recognition in tweets, Ritter et al. (2011) propose a distantly supervised approach which applies LabeledLDA to leverage huge amounts of unlabeled data in addition to large dictionaries of entities gathered from Freebase, and combines information about an entity's context across its mentions. When performing tests on 10 types of categories, T-NER achieved values of Precision, Recall and F1 (0.65, 0.42 and 0.51, respectively). For 3 types, T-NER achieved 0.73 on Precision, 0.49 on Recall and 0.59 on F1. Godin et al. (2015) proposed to leverage 400 million Twitter posts to generate powerful word embeddings as input features for NER. These word embeddings were used as input to a neural network to classify the words in the posts, followed by a post-processing step to check the coherence of individual tags within a named entity. Their best model achieved a F1-score of 0.5882 when detecting named entities without categorizing them and 0.4375 when categorizing the named entities into one of 10 types. Li et al. (2012) is a 2-step unsupervised NER system for targeted Twitter Streams. In a first step, the method leverages on global context obtained from Wikipedia and from the Web N-Gram corpus to partition tweets into valid segments (phrases) using a dynamic programming algorithm. In the second step, TwiNER constructs a random walk model to exploit the local context derived from the Twitter stream.

Finally, for entity linking in tweets(which is the main focus for this work), I observed that Yamada et al. (2015a) achieved the best results. This might explained by the fact that they do not use a named entity recognition system for the first step of entity linking because they are proved to perform poorly for tweets. The same authors tried to reverse the order of this systems, by applying named entity recognition after entity linking. Although it improved the performance of NER, it dropped the results on entity linking. Guo et al. (2013) proposed a structural SVM algorithm that merges mention detection and entity disambiguation into a single end-to-end entity linking approach. Although it achieved a Precision value close to 0.8, Recall only achieved 0.599. Liu et al. (2013) developed a collective inference method that jointly links a set of tweet mentions to their corresponding entities. The results were similar to to the system from Guo et al. (2013), but Recall was almost 0.1 higher. Table 2.4 summarizes this systems.

3 Entity Linking Over Short Texts

This section describes how the system used for this work was built. It starts with a general architecture of the system, followed by detailed description of both subsystems used for this work: Named Entity Recognition (NER) and Named Entity Disambiguation (NED).

Given a sequence of tweets, denoted $\vec{T} = (t_1, t_2..., t_n)$, the task is, for each tweet, to first identify a sequence of mentions denoted $\vec{M} = (m_1, m_2...m_i)$ and then output a sequence of entities denoted by $\vec{E} = (e_1, e_2..., e_i)$, where e_i is the entity corresponding to m_i. Each entity refers to an item of a knowledge base. The KB used is based on Wikipedia. Each mention denotes a sequence of tokens in a tweet than might possibly have an entity related in the KB.

The following example illustrates the task. Given the tweet *@BarackObama will be missed, we can't let Trump be the head of the USA! Go Hillary!* and the mentions *BarackObama*, *Trump* and *Hillary*, the expected output is the Wikipedia links of *Barack Obama*, *Donald Trump* and *Hillary Clinton*, respectively.

3.1 Genereal Architecture of the System

As stated before, Entity Linking requires two main steps: recognizing named entities in text (in this case, tweets) and disambiguate them by linking to the correct page in Wikipedia.

As showed in Figure 3.1, the tweets start by entering the main system. There, they start by entering the pre-processing step. There, operations are executed to increase context for disambiguation. The main assumption made was that if a user posts a tweet, he is possibly interested in that subject, so it is normal that he posts something related to the tweet in another time. This way, it is possible to have 2/3 tweets regarding one subject instead of just 1, which increases the chance of a correct linking between the named entity and the corresponding Wikipedia link. To do this, the Java library for Twitter API Twitter4J [1] was used. The strategy

[1]http://twitter4j.org/en/index.html

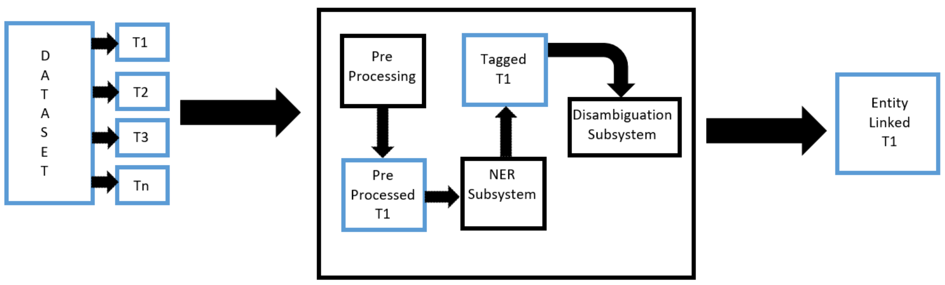

Figure 3.1: General Architecture of the Named Entity Recognition and Disambiguation System

used was to extract the tweet ID and use the library to get the user who posted the tweet. After identifying the user, the same library is used to extract 5 more tweets from that user.

The next step is to clean up the tweets, to try to make them less noisy. To do so, some changes in the tweet's text are performed. The following changes were made to the tweets:

- **All usernames (@) are replaced by the user's name**: This way, the system has better chances of disambiguate the named entity, because it would be harder with the standard representation of a username in Twitter (for example, @justibieber is replaced by Justin Bieber). Twitter4J was also used to perform this task;
- **The hashtag symbol (#) is removed**: same reason used at the usernames;
- **URLs are removed from the tweet**: Since URLs are never identified as entities, it is useless that they enter the disambiguation process;
- **The symbol / is removed**: Sometimes users use this symbol to talk about two different things, which can be two entities intended to be identified (for example, Twitter/Facebook);
- **The word RT (used when mentioning a retweet) is removed**: As stated before, capitalization is normally a good indicator of a named entity. Since RT usually appears in uppercase, it might be confused with a named entity, so removing it reduces the false positive cases.

All this changes clean the tweet, making it easier to understand, and thus to be interpreted by both systems, which can lead to an improvement of the results.

For example, consider the tweet *RT @barackobama will be missed. Clinton/Trump can't replace him #USElections https://t.co/MC8WvosMTT*. After the clean step, it would end up as *Barack Obama will be missed. Clinton Trump can't replace him USElections.*

After this, the tweets go to the NER subsystem so that the candidate named entities can be identified (more details in Section 3.2). As a final step, the tweets go to the disambiguation subsystem, where all the named entities identified are evaluated to perform the linking. The final result is an HTML file, where we have the tweets with the named entities highlighted, with the corresponding link performed by the system (more details in Section 3.3).

3.2 Named Entity Recognition Subsystem

After the pre-processing described in the previous section is applied to the tweets, the next step is the processing through the Named Entity Recognition subsystem. Previous sections explained what named entity recognition is and why it is important in the entity linking task. NER is the first main step in this task. At this step, the candidate named entities in the tweets are identified.

To perform this task, I chose to adapt Stanford NER[2], a Java implementation of a Named Entity Recognizer. Stanford NER provides a general implementation of (arbitrary order) linear chain Conditional Random Field (CRF) sequence models. That is, by training specific models on labeled data, it is possible to use this code to build sequence models for NER or any other task.

Taking this last point into account, I started by training my own NER model, specifically for tweets. To do so, I mixed two different datasets. The first one was the dataset used by Ritter et al. (2011) and the second one was the dataset from ACL 2015 Workshop on Noisy User-generated Text (W-NUT) - NER Shared Task[3] (more details about both datasets presented in Section 4.1) to use as training data. For the first dataset, I used both train and eval sets, and for the second one I used the train set. The dataset combination was divided into 4 categories: LOCATION, PERSON, ORGANIZATION and MISC. In addition, two more models were used during this task: one trained on the CoNLL 2003 *eng.train* and the same model, but in a caseless

[2]http://nlp.stanford.edu/software/CRF-NER.shtml
[3]http://noisy-text.github.io/2015/ner-shared-task.html

version (i.e., a version that ignores capitalization). Both systems are also divided into the same 4 categories, which supports their choice. To train the model for Twitter (using tools provided by the Stanford NER software), and since the datasets used were all in the CoNLL format (explained above), the features used to train CoNLL 4 Class and CoNLL 4 Class Caseless were used, in addition to others that to try to improve the model. The features used are described at Appendix A.1

In addition to all those features, and to the token categories already defined in the dataset, all usernames (tokens who started with @) were classified as PERSON. This decision was made because in the Entity Linking dataset, some of the entities in the gold set refer to the usernames in the tweets, which is why it is important to identify them in this subsystem. Since my disambiguation subsystem does not categorize the entities, it was almost irrelevant which category was chosen for the usernames. PERSON was chosen, because, in most cases, usernames refer to a person, although many times they can also refer to organizations.

On the other hand, hashtags where not used in the training data. This means that all tokens that represented an hashtag (start with #) where categorized as *O*. To prevent this from affecting the final results, all tweets passed through a pre-processing stage before being evaluated in terms of Named Entity Recognition. The pre-processing removed some invalid chars in the tweets, as well as the hashtag symbol in the tokens. This way, the tweets were more clean, which may increase the chances of better results.

To test the models to be used (Twitter model, CoNLL 4 class and CoNLL 4 class caseless), Stanford NER was again used. The software provides a flag which allows one to evaluate NER performance on a file with a chosen model and output a file where the first column has the token evaluated, the second column has the expected category and on the third the category the system obtained. The input file corresponds to the gold set, and contains two columns, where the first one is the token to be evaluated and the second contained the expected category. Finally, to obtain the evaluation results, the script *conlleval*[4] was used. This script evaluated the files and outputted the Precision, Recall and F1 scores. All 3 systems where evaluated for both datasets used separately. The results for the 3 systems used are presented in Table 3.1.

As we can see in the table, the Twitter model achieves the higher results, which is not a surprise. All the three metrics in both datasets achieve results over 0.9. The low results from the

[4]http://www.cnts.ua.ac.be/conll2000/chunking/output.html

	Ritter dataset			W-NUT dataset		
	P	R	F1	P	R	F1
Twitter	0.9745	0.9259	0.9496	0.9712	0.9270	0.9486
CoNLL 4 Class	0.3431	0.2507	0.2897	0.3183	0.2159	0.2573
CoNLL 4 Class Caseless	0.3690	0.2507	0.2986	0.3482	0.1905	0.2462

Table 3.1: Named Entity Recognition results obtained with individual models

	Ritter dataset			W-NUT dataset		
	P	R	F1	P	R	F1
Twitter	0.9745	0.9259	0.9496	0.9712	0.9270	0.9486
Twitter CoNLL 4 Class	0.7693	0.9501	0.8502	0.7861	0.9450	0.8582
Twitter CoNLL 4 Class Caseless	0.7294	0.9444	0.8231	0.7739	0.9418	0.8496
Twitter CoNLL 4 Class CoNLL 4 Class Caseless	0.6533	0.9530	0.7752	0.6892	0.9481	0.7982

Table 3.2: Comparison between Twitter model and Named Entity Recognition results obtained by combining models

other two models is explained by the fact that both models are not trained with Twitter data. This means they are not prepared for short texts. In addition, they are not ready to identify usernames and categorize them, so there is a lot of tokens that the model does not recognize, which decreases the results.

Even though the Twitter model achieved these results, I also decided to check what results would achieve if I tried to combine the models, by using the class $NERClassifierCombiner$, available at Stanford NER, which allows for multiple CRFs to be used together. I decided to do check this option, because I wanted to know if both CoNLL models could help improving the results when combined with the Twitter model. The class combines the models and outputs the named entities identified, based on the order in which the models are (i.e., the first model has higher power of decision, followed by the second and so on). Since the focus of this work is on tweets, I always decided to input the Twitter model in first place. The tools provided by Stanford NER were again used, as well as *conlleval*. The results obtained are presented at Table 3.2.

By looking at the table, the main result we see is that Recall improved on both datasets, compared with the results from the individual models. We also observe that, when combined with the Twitter model, both CoNLL 4 Class and CoNLL 4 Class Caseless almost doubled their Precision scores and almost quadrupled their Recall and F1 (although F1 is impacted by the improvement of Recall).

Considering the results in both tables, we see that the Twitter model achieves the higher values of Precision and F1 for both datasets, while the combination of the three models achieves the higher Recall. Since the Twitter model is used in all the best score achieved, I decided to use the combination of the three models to perform the entity linking task. Although Precision and F1 drop in the combination, Recall achieves 0.9530, which is the higher value achieved in all the experiences performed. In addition, and despite the fact that they are not Twitter specific, both CoNLL 4 Class and CoNLL 4 Class Caseless are important to identify some entities, since they are trained with data that may help identifying entities that the data used to train the Twitter model does not provide. For example, acronyms like *SLB* or *QPR* are not caught by the Twitter model, but when combining the 3 models, the entities are recognized.

After choosing the models to use for Named Entity Recognition, the next step was to enter the disambiguation subsystem to perform the final step for Entity Linking.

3.3 Entity Linking Subsystem

Now that tweets are pre-processed and the candidate named entities are identified, the final step is to identify the Wikipedia links and perform disambiguations.

To build this system, software previously available was again used. I chose to use AIDA[5] (mentioned at section 2.2.1), a framework and online tool for entity detection and disambiguation. AIDA was developed to perform entity linking on structured texts (e.g., news articles), which is proved to perform poorly in tweets. Since the source code is available at GitHub[6], it was possible to adapt the system to increase its performance on short texts (in this case, tweets).

As stated before, the entity linking subsystem's goal is to map mentions of named entities with their proper entries in a knowledge base. AIDA first starts by identifying noun phrases that potentially denote named entities. In the standard version, it uses Stanford NER Tagger.

[5]https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/aida/
[6]https://github.com/yago-naga/aida

Then, for each of the possible entities that a mention could denote, the system provides a set of short names (e.g., Apple for Apple Inc.) and paraphrases (e.g., El Mago for Pablo Aimar). After identifying the entities, and to aim for the best disambiguation mappings, the system combines prior, similarity and coherence measures into a combined objective function: for each mention m_i, $i = 1...k$ select entity candidates e_j, one per mention, such that

$$\begin{aligned} \alpha \cdot \sum_{i=1..k} prior(m_i, e_{j_i}) + \\ \beta \cdot \sum_{i=1..k} sim(cxt(m_i), cxt(e_{j_i})) + \\ \gamma \cdot (e_{j_i} \in cnd(m_1) ... e_{j_k} \in cnd(m_k)) = max! \end{aligned} \tag{3.1}$$

In the equation, $\alpha + \beta + \gamma = 1$, $cnd(m_i)$ is the set of possible meanings of m_i, $cxt()$ denotes the context of mentions and entities, respectively, and $coh()$ is the coherence function for a set of entities.

The authors consider that the key for mapping mentions onto entities are the contexts on both sides of the mapping. So, the authors consider two different approaches for context. First, for each mention, the authors construct a context from all words in the entire input text. This way, it is possible to represent a mention as a set of weighted words or phrases that it co-occurs with. Second, the authors alternatively consider similarity scores based on syntactically-parsed contexts (based on Thater et al. (2010)). On the entity side of the mapping, the authors associate each entity with characteristic keyphrases or salient words, precomputed from Wikipedia articles and similar sources (e.g., Kurt Cobain may have keyphrases like Nirvana, Hard rock). This way, it is possible to define and compute similarity measures between a mention and an entity candidate (e.g., weighted word overlap, n-gram-based measures, etc). To calculate coherence, the authors consider that on the entity side, each entity has a context in the underlying knowledge base, i.e., other entities that are connected via semantic relationships or have the same semantic type. Since YAGO[7] provides the same-as cross-referencing to Wikipedia, the authors can quantify the coherence between two entities by the number of incoming links that their Wikipedia article share.

Regarding features and measures, the system uses three different types: *popularity prior*,

[7]semantic knowledge base, derived from Wikipedia WordNet and GeoNames. Currently, YAGO has knowledge of more than 10 million entities (like persons, organizations, cities, etc.) and contains more than 120 million facts about these entities.

mention-entity similarity, and *entity-entity coherence.* For popularity prior, for each surface form that constitutes an anchor text, the authors count how often it refers to a popular entity. For each name, these counts provides an estimate for a probability distribution over candidate entities. For mention-entity similarity, two different similarity types are used: *keyphrase-based similarity* and *syntax-based similarity.* For the first one, on the mention side, the authors use all tokens in the document as context (except stopwords and the mention itself), while on the entity side, the knowledge base knows authoritative sources for each entity (e.g., the corresponding Wikipedia article). These are the inputs for an offline data-mining step to determine keyphrases for each entity and their statistical weights. The set of keyphrases of an entity is denoted as K P(e). For each word w that occurs in a keyphrase, the authors compute a specificity weight with regard to the given entity: the Mutual Information (MI) between the entity e and the keyword w, calculating the joint probabilities for MI as follows:

$$p(e,w) = \frac{|w \in (\text{K P}(e) \cup \cup_{e' \in \text{IN}_e} \text{K P}(e'))|}{N} \tag{3.2}$$

In the equation, IN(e) reflects if w is contained in the keyphrase set of e or any of the keyphrase sets of an entity linking to e, with N denoting the total number of entities. Since a keyphrase may occur only partially in an input text, the score of partially matching phrase q in a text is set to:

$$score(q) = z(\frac{\sum_{w \in cover} \text{weight}(w)}{\sum_{w \in q} \text{weight}(w)})^2 \tag{3.3}$$

In the equation, $weight(w)$ is the MI weight and

$$z = \frac{\#\text{matching words}}{\text{length of cover}(q)} \tag{3.4}$$

Finally, the similarity score is calculated by:

$$\text{simscore}(m,e) = \sum_{q \in \text{K P}(e)} \text{score}(q) \tag{3.5}$$

For the second one, the authors leverage information about the immediate syntactic context in which an entity mention occurs. They derive vector representations of words in syntactic contexts. Then, the authors consider a set of substitutes for each possible entity e, which are used as its context $ctx(e)$. For each substitute, a standard distributional vector and a contextualized vector are computed. Syntax-based similarity between $cxt(e)$ and the context $cxt(m)$ of the mention is then defined as the sum of the scalar-product similarity between these two vectors for each substitute.

Finally, for entity-entity coherence, the authors consider the number of incoming links that the entities Wikiepdia's articles share, refined by Milne and Witten (2008), taking into account the total number N of entities in the Wikipedia collection (Equation 2.14)

For all the previous mentioned measures, the authors construct a weighted, undirected graph with mentions and candidate entities as nodes. As stated before, it has two kinds of edges: *mention-entity* and *entity-entity.* Given a mention-entity graph, the goal is to compute a dense subgraph that would ideally contain all mention nodes and exactly one mention-entity edge for each mention, disambiguating all mentions. To specify a notion of density that is best suited for capturing the coherence of the resulting entity nodes, the authors define a *weighted degree* of a node in the graph to be the total weight of its incident edges. Then, the density of a subgraph is defined to be equal to the minimum weighted degree among its nodes. The goal is to compute a subgraph with maximum density, while observing constraints on the subgraph structure.

To address the problem that the discovery of a dense-subgraph is NP-hard, the authors adopt and extend an approximation algorithm that starts from the full mention-entity graph and iteratively removes the entity node with the smallest weighted degree. Among the subgraphs obtained in the various steps, the one maximizing the minimum weighted degree will be returned as output. To guarantee that the algorithm arrives at a coherent mention-entity mapping for all mentions, the authors enforce each mention node to remain connected to at least one entity. However, this may lead to suboptimal results, which is why a pre-processing phase is done to prune the entities that are only remotely related to the mention nodes. For each entity node, the authors compute the distance from the set of all mention nodes in terms of the sum of the corresponding squared shortest path distances. Then, the input graph is restricted to the entity nodes that are closest to the mentions. After this pre-processing, the greedy algorithm described is then applied. The output of the main loop would often be close to the desired result, but it might still have more than one mention entity-edge for one or more mentions. However. the subgraph is small enough to consider an exhaustive enumeration and assessment of all possible solutions. This is one of the options that the authors implemented as post-processing.

Alternatively, it is possible to perform a faster local-search algorithm, where candidate entities are randomly selected with probabilities proportional to their weighted degress. This step is repeated for a number of iterations and the best configuration with the highest total edge-weight is used as final solution.

Finally, and since the algorithm may be misled in specific situations (e.g., input text very short), the authors introduced two *robustness tests* for individual mentions and, depending on the test's outcomes, the system only uses a subset of features and techniques. The first is a *prior test*, where the authors check, for each mention, whether the popularity prior for the most likely candidate entity is above some threshold. If this is not the case, then the prior is completely disregarded for computing the mention-entity edge weights. Otherwise, the prior is combined with the context-based similarity computation to determine edge weights. The second one is a *coherence test*, where the authors compare the popularity prior and the similarity only measure, on a per-mention basis. For each mention, the authors compute the $L1$ distance between the popularity-based vector of candidate probabilities and the similarity-only-based vector of candidate probabilities:

$$\sum_{i=1...k} |prior(m, e_i) - \text{simscore}(m, e_i)| \tag{3.6}$$

If the equation returns a value that exceeds a specified threshold λ, the disagreement between popularity and similarity-only indicates that there is a situation that coherence may be able to fix. If it does not exceed the threshold, there is hardly any disagreement, using coherence as an additional aspect would be risky for thematically heterogeneous texts and should better be disabled. In that case, the authors choose an entity for the mention at hand, using the combination of prior and similarity.

Now that we know how AIDA works, it is time to explain how I set up the system to perform all the tasks I needed for this work. First, to use AIDA with YAGO2 (Hoffart et al. (2011a)), I downloaded the repository provided on the AIDA website as a Postgres dump[8]. AIDA uses nearly 3 million named entities harvested from Wikipedia for disambiguation. The dump used was AIDA_entity_repository_2014-01-02v10, which was built from a Wikipedia dump from 2014. For recognizing named entities, I loaded a combination of three different NER models: a model

[8]https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/aida/downloads/

Algorithm 1 Dataset Break Algorithm

```
1: procedure DATASET BREAK
2:     $i \leftarrow 1$
3:     while i != length(dataset) do
4:         $tweet \leftarrow datasetTweet_i$;
5:         create file with single tweet;
6:         $i \leftarrow i+1$;
7:     end while
8: end procedure
```

trained specifically for Twitter, CoNLL 4 Class and CoNLL 4 Class Caseless (all explained in the previous section).

For this work, different configurations of AIDA where used, to see how each one performs for tweets. AIDA offers four different configurations to perform entity linking:

- PRIOR: uses *PriorOnlyDisambiguationSettings*, meaning that the system annotates each candidate mention with the most prominent entity;
- LOCAL: uses *LocalDisambiguationSettings*, meaning that the system uses the entity prominence and the keyphrase-context similarity to disambiguate;
- GRAPH: uses *CocktailPartyDisambiguationSettings*, meaning that the system uses a graph algorithm on the entity coherence graph (Milne-Witten link coherence) to disambiguate;
- GRAPH-KORE: uses *CocktailPartyKOREDisambiguationSettings*, meaning that the system uses a graph algorithm on the entity coherence graph (KORE link coherence) to disambiguate;

The first test performed was to check the values of the different measures without performing any changes in AIDA (except the models, which were already loaded to increase performance for tweets). I developed a Java algorithm (see Algorithm 1) that breaks the dataset and sends each tweet individually for AIDA. To perform these test, I used the dataset from #Microposts2015 (more details on the dataset are provided at Section 4.1).

Results are presented in Section 4.2. After analyzing the results, the next step was to try to increase the performance. To do so, the pre-processing steps explained above where performed before executing the system. The Java program was modified (see Algorithm 2) to send to the system not only the individual tweet, but also the five more tweets added in the pre-processing.

Algorithm 2 Dataset Break Algorithm for Changed AIDA

```
1: procedure DATASET BREAK
2:     $i \leftarrow 1$
3:     while i != length(dataset) do
4:         $tweet \leftarrow datasetTweet_i$;
5:         perform pre-processing steps;
6:         extract tweets from user to increase context;
7:         create file with $datasetTweet_i$ cleaned and 5 more tweets from the user;
8:         $i \leftarrow i+1$;
9:     end while
10: end procedure
```

In this case, since the tweets belong to the same user who tweeted the original tweet, AIDA is supposed to consider them as context for the original tweet, which is why they are all processed at once by AIDA. The results for this final experiment and their discussion are also presented in Section 4.2.

3.4 Summary

This section presented all the details regarding the system developed for this work. It stated with the NER subsystem, where the candidate named entities where identified, using a model trained specially for Twitter. Then, it passed through the pre-process step to try to increase the disambiguation results, after performing an execution as-is, so that the results could be compared and discussed how the changes performed affected the results.

4 Experimental Validation

This chapter starts by presenting and describing the datasets (Section 4.1) and metrics used to evaluate the system (Section 4.2). Section 4.3 presents the results obtained with the framework developed for this work.

4.1 Datasets

Different datasets where used accordingly to each subsystem. As said before, the combination of two different datasets was used to train the Named Entity Recognition subsystem. The first one was the dataset used by Ritter et al. (2011). The combination of both train and eval sets ended up with 2400 tweets, with approximately 48K tokens. The second dataset was the one used at the ACL 2015 Workshop on Noisy User-generated Text (W-NUT) - NER Shared Task[1]. The train set was used, which contains approximately 35K tokens. As said before, both datasets were annotated with 4 types of categories: PERSON, LOCATION, ORGANIZATION, and MISC.

For the Disambiguation of Entities Subsystem, I used the dataset from Making Sense of Microposts (#Microposts2015)[2]. I used the 2015 version instead of the 2016, because the most recent dump of Wikipedia available for AIDA is from 2014, which would result in many unknown entities with the 2016 version (which I tested). The dataset comprises three sets of tweets (train, test and dev), extracted from a collection of over 18 million tweets. They include event-annotated tweets provided by the Redites project[3], covering multiple noteworthy events from 2011 to 2013 (including the death of Amy Winhehouse, the London Riots, the Oslo bombing, and the Westgate Shopping Mall shootout), and tweets extracted from the Twitter firehose in 2014.. Instead of the test set, I had to use the train set to evaluate the system, since almost every tweet from

[1]http://noisy-text.github.io/2015/ner-shared-task.html
[2]http://scc-research.lancaster.ac.uk/workshops/microposts2015/
[3]http://demeter.inf.ed.ac.uk/redites/

	Precision	Recall	F1
GRAPH	0.5541	0.5235	0.5383
LOCAL	0.5725	0.5620	0.5672
PRIOR	0.5739	0.5648	0.5693
KORE	0.5604	0.5357	0.5478

Table 4.1: Results produced by AIDA without changes performed

	Precision	Recall	F1
My Framework	0.5739	0.5648	0.5693
Yamada et al. (2015a)	0.8570	0.8230	0.840
Yamada et al. (2015b)	0.5766	0.5522	0.5641
Guo et al. (2013)	0.7880	0.5990	0.6800
Liu et al. (2013)	0.7640	0.6750	0.7110

Table 4.2: Comparison between the best performance of my framework and other systems

the test set was not available anymore, which would make impossible to increase context by getting more tweets from the user. The initial set had 3498, but ended up with 2644 tweets after checking which ones were still available to be used the way I needed.

4.2 Results

The results obtained for the first test (AIDA without changes performed) are presented at Table 4.1. By looking at the table, we see that PRIOR is the configuration that achieves the higher results for all three measures. The results are not very high, which may have some explanations. The first one is the fact that usernames do not suffer changes for processing, i.e., the system tries to link a username (which is caught by the NER models, as explained before) in his standard version (including the @ symbol), which will hardly achieve success. This ends up being a downside of using the Twitter model without any pre-processing. The second one is similar to the usernames and is related to the hashtags, The system also tries to link the hashatgs with the # symbol, which also makes it hard to achieve success. Another reason might be related with some noise (specially abbreviations and some symbols) in the tweets, which might led the NER models to identify some entities which it shouldn't or to not identify others that should be caught. All the steps performed at the pre-processing (described at Section 3.3) where made precisely taking all this possible causes into account.

Table 4.2 presents a comparison between the best performance achieved by my system (PRIOR) and the systems developed for Entity Linking in Tweets described at Section 2.2.2.2. Although my system did not achieve very high results, we can see that it outperforms Recall and F1 from Yamada et al. (2015b) and an almost equal Precision score. It also achieves a Recall similar to the one from Guo et al. (2013), but Precision and Recall are still at some distance.

Regarding the test after the changes performed, the results obtained are explained below.

Looking at Table 4.3, we can directly see that all configurations improved their results. This already proves that the changes performed were useful to increase the performance of AIDA. Another thing that is possible to see is that, while PRIOR achieved the best performance on the first execution, at this one LOCAL achieved the higher results. Regarding each metric individually, Precision was the one with the lower increase values. This might be explained by the fact that Precision includes False Positives in its calculations (i.e., entities that were not supposed to be caught and linked), and although the changes in the pre-processing where meant to reduce noise (which had success), the truth is that it also made False Positives easier to appear. It might happen with usernames. Imagine that @username (for example @Mike), when replaced by the screen name, is transformed into *Mike the Man With Golden Boots*. AIDA passes from identifying a True Positive (i.e., an entity that is supposed to be caught and linked) to a false positive, which increases their number and prejudices Precision. On the other hand, Recall and Precision achieve good increase values (which for F1 is directly proportional to an increase of Recall). Since Recall uses False Negatives (i.e., entities that are supposed to be caught, but linked incorrectly), the improvements on their value supports the thesis that the changes performed increased the number of correct entities caught. Again, I compared the best performance (LOCAL in this case) with the systems from Section 2.2.2.2. The results can be seen at Table 4.4. The first thing observable is that my framework, with all the changes performed, now outperforms the system by Yamada et al. (2015b) in all three metrics (while the previous execution only outperformed Recall and F1). In addition, my framework also outperformed the Recall values from Guo et al. (2013) and Liu et al. (2013), while also achieved a closer F1 value to the one at Guo et al. (2013). However, both these frameworks were evaluated with different datasets.

After analyzing both executions, it is possible to see that, although the results increase

	Precision	Recall	F1
GRAPH	0.5978	0.6575	0.6262
LOCAL	0.6117	0.6942	0.6503
PRIOR	0.6050	0.6757	0.6384
KORE	0.6035	0.6732	0.6365

Table 4.3: Results produced by AIDA, after the changes performed

	Precision	Recall	F1
My Framework	0.6117	0.6942	0.6503
Yamada et al. (2015a)	0.8570	0.8230	0.840
Yamada et al. (2015b)	0.5766	0.5522	0.5641
Guo et al. (2013)	0.7880	0.5990	0.6800
Liu et al. (2013)	0.7640	0.6750	0.7110

Table 4.4: Comparison between the best performance of my framework (after changes) and other systems

when changes are performed, they still are a bit far from some other systems. Nevertheless, it is also shown that my system outperformed some values from the same systems. Some facts that might support these results were already explained during the work (e.g., the username replacement that might increase the number of False Positives). Another explanation is related with the increase of context. The idea was to increase the context of the original tweet by adding more tweets from the user. However, these tweets were supposed to be closer to the date of the original tweet (e.g., add the five next tweets the user made after the tweet to evaluate). The problem is that the Twitter API only allows to extract tweets that are, at maximum, seven days old from the date that the request is made. Since the dataset is composed by tweets made until 2014, it was impossible to get tweets closer to the date of the original tweet. This way, context might not be fully increased, since it is assumed that it is more likely to a user to tweet about related subjects in consecutive tweets. Finally, and despite the fact that I tried to clean up the tweets, they still have noise, which as stated before, makes it hard for the system to perform the linking correctly.

4.3 Summary

In this chapter, some fundamental information was presented, as well as the final discussion of the results obtained by the system developed for this work. The datasets used for both NER and Entity Linking were described, as well as the Wikipedia dump used as KB and the measures used to evaluate the system.

5 Conclusions and Future Work

This chapter presents conclusions of the work developed (Section 5.1) and possible future research that might improve the results of this kind of tasks (Section 5.2).

5.1 Conclusions and Main Contribution List

In this work, I presented an alternative use for AIDA, a framework and online tool for entity detection and disambiguation. I started by training a model specifically for Twitter, because both Stanford NER and AIDA were developed for rich text, which leads to poor performances on short texts like tweets. This model achieved very high results, and I decided to combine it with another two models: CoNLL 4 Class and CoNLL 4 Class Caseless. After loading these models to AIDA, two tests were made: one with standard configurations and another with changes performed.

Results improved from the first to the second execution, which allows me to conclude that the changes performed were effective. Although, as explained before, it might increase the number of False Positives, the idea of replacing a username by his screen name proved to be effective, since almost every username was correctly linked, which would be impossible in the standard form. The same effect happens with hashtags, because removing the # symbol made it possible to correctly link some hashtags. However, in this case, the fact that hashtags are rarely used for possible entities, in addition to the fact that many times they are written with capitalization, might lead to an increase of False Positives as well. Another contribution is the Twitter model. It would be very hard to obtain these results if a model trained with Twitter data was not included, since tweets suffer from a lack of context and noisy data. Finally, since Recall improved in values higher than 0.1 in all four configurations of AIDA, it supports that my framework misses less linking of entities.

In conclusion, I think the results were good, since I was able to outperform some of the systems I analyzed, allied to the improvement of results that was achieved by performing the changes I chose.

5.2 Future Work

There are some ways to improve both Named Entity Recognition and Entity Linking in Tweets. Since one of the mains problems of tweets is the noisy data, one way to try to increase performance is by normalizing the tweets, to make them cleaner and to try to force them to have a good textual structure. Kaufmann and Kalita (2010) present a novel system which normalizes Twitter posts, converting them into a more standard form of English, so that standard machine translation (MT) and natural language processing (NLP) techniques can be more easily applied to them. In order to normalize Twitter tweets, the authors take a two step approach. First, the authors pre-process tweets to remove as much noise as possible and then feed them into a machine translation model to convert them into standard English.

Regarding Named Entity Recognition, at Ammar et al. (2016) the authors introduce two new neural architectures: one based on bidirectional LSTMs (Long Short-term Memory Networks) and conditional random fields, and the other that constructs and labels segments using a transition-based approach inspired by shift-reduce parsers. Their models rely on two sources of information about words: character-based word representations learned from the supervised corpus and unsupervised word representations learned from unannotated corpora. For a given sentence $(x_1, x_2, ..., x_n)$ containing n words, each represented as a d-dimensional vector, an LSTM computes a representation $\vec{h}_t$ of the left context of the sentence at every word t. Since generating a representation of the right context $\vec{h}_t$ as well should add useful information, the authors use a second LSTM that reads the same sequence in reverse. The representation of a word using this model is obtained by concatenating its left and right context representations and this effectively include a representation of a word in context, which is useful for numerous tagging applications.

Other techniques for entity linking can now also be explored. Since a key challenge in entity linking is making effective use of contextual information to disambiguate mentions that might refer to different entities in different contexts, Francis-Landau et al. (2016) present a model that uses convolutional neural networks to capture semantic correspondence between the context of a mention and a proposed target entity. These convolutional networks operate at multiple granularities to exploit various kinds of topic information, and their rich parameterization gives them the capacity to learn which n-grams characterize different topics. These networks are then combined with a sparse linear model, which allows them to achieve state-of-the-art performances. The authors model semantic similarity between a source document context of a mention and its potential entity targets using convolutional neural networks (CNNs). Their model focuses on two core ideas: first, that topic semantics at different granularities in a document are helpful in determining the genres of entities for entity linking, and second, that CNNs can distill a block of text into a meaningful topic vector. Their entity linking model is a log-linear model that places distributions over target entities t given a mention x and its containing source document. For each of three text granularities in the source document (the mention, the immediate context of that mention, and the entire document) and two text granularities on the target entity side (title and Wikipedia article text), the authors produce vector representations with CNNs.

Bibliography

Alhelbawy, A. and Gaizauskas, R. (2014). Graph ranking for collective named entity disambiguation. In *Proceeding of the Annual Meeting of the Association for Computational Linguistics*, pages 75–80.

Ammar, W., Mulcaire, G., Ballesteros, M., Dyer, C., and Smith, N. A. (2016). Many languages, one parser. In *Transactions of the Association for Computational Linguistics*, volume 4, pages 431–444.

Breiman, L. (2001). Random forests. *Machine Learning*, 45:5–32.

Brown, P. F., deSouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. (1992). Class-based n-gram models of natural language. In *Compulalional Linguistics*, pages 467–479.

Carreras, X., Màrquez, L., and Padró, L. (2003). A simple named entity extractor using adaboost. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, volume 4, pages 152–155.

Chisholm, A. and Hachey, B. (2015). Entity disambiguation with web links. In *Transactions of the Association for Computational Linguistics*, volume 3, pages 152–155.

Collins, M. and Singer, Y. (1999). Unsupervised models for named entity classification. In *In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 100–110.

Cucerzan, S. (2012). The MSR System for Entity Linking at TAC 2012 . In *Proceedings of the Text Analysis Conference 2012*.

Finkel, J. R., Grenager, T., and Manning, C. (2005). 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.

Francis-Landau, M., Durrett, G., and Klein, D. (2016). Capturing semantic similarity for entity linking with convolutional neural networks. In *Proceedings of the North American Association for Computational Linguistics.*

Godin, F., Vandersmissen, B., Jalalvand, A., Neve, W. D., and de Walle, R. V. (2014). Alleviating manual feature engineering for part-of-speech tagging of twitter microposts using distributed word representations. *In Workshop on Modern Machine Learning and Natural Language Processing (NIPS 2014)*, pages 146–153.

Godin, F., Vandersmissen, B., Neve, W. D., and de Walle, R. V. (2015). Multimedia Lab @ ACL W-NUT NER Shared Tasks: Named Entity Recognition for Twitter Microposts using Distributed Word Representations. In *Proceedings of the ACL 2015 Workshop on Noisy User-generated Text*, pages 146–153.

Guo, S., Chang, B. M.-W., and Kiciman, E. (2013). To Link or Not to Link? A Study on End-to-End Tweet Entity Linking. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1020–1030.

He, Z., Liu, S., Li, M., Zhou, M., Zhang, L., and Wang, H. (2013). Learning entity representation for entity disambiguation. In *Annual Meeting of the Association for Computational Linguistics*, pages 30–34.

Hoffart, J., Suchanek, F. M., Berberich, K., Lewis-Kelham, E., de Melo, G., and Weikum, G. (2011a). Yago2: Exploring and querying world knowledge in time, space, context, and many languages. In *Proceedings of the 20th International Conference Companion on World Wide Web*, pages 229–232.

Hoffart, J., Yosef, M. A., Bordino, I., Fürstenau, H., Pinkal, M., Spaniol, M., Taneva, B., Thater, S., and Weikum, G. (2011b). Robust disambiguation of named entities in text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP11*, pages 782–732.

Ji, H., Grishman, R., and Dang, H. T. (2011). Overview of the tac 2011 knowledge base population track. In *Proceedings of the 4th Text Analysis Conference, TAC 11.*

Kaufmann, M. and Kalita, J. (2010). Syntactic normalization of twitter messages. In *International conference on natural language processing*, pages 149–158.

Lazic, N., Subramanya, A., Ringgaard, M., and Pereira, F. (2015). Plato: A Selective Context Model for Entity Resolution. In *Transactions of the Association for Computational Linguistics (TACL)*, volume Volume 3, pages 503–515.

Li, C., Weng, J., He, Q., Yao, Y., Datta, A., Sun, A., and Lee, B.-S. (2012). TwiNER: Named Entity Recognition in Targeted Twitter Stream. In *Proceedings of the 35th international Association for Computational Linguistics SIGIR conference on Research and development in information retrieval*, pages 721–730.

Lin, T., Mausam, and Etzioni, O. (2012). Entity Linking at Web Scale. In *Proceeding AKBC-WEKEX '12 Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, pages 84–88.

Liu, X., Li, Y., Wu, H., Zhou, M., Wei, F., and Lu, Y. (2013). Entity Linking for Tweets. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, volume 1: Long Papers, pages 1304–1311.

Meij, E., Weerkamp, W., and de Rijke, M. (2012). Adding semantics to microblog posts. In *Proceedings of the Fifth Association for Computational Linguistics International Conference on Web Search and Data Mining*, pages 563–572.

Mihalcea, R. and Csomai, A. (2007). Wikify!: Linking documents to encyclopedic knowledge. In *Proceedings of the Sixteenth Association for Computational Linguistics Conference on Conference on Information and Knowledge Management*, pages 233–242.

Milne, D. and Witten, I. H. (2008). Learning to link with wikipedia. In *Proceedings of the 17th Association for Computational Linguistics Conference on Information and Knowledge Management*, pages 509–518.

Ramage, D., Hall, D., Nallpatu, R., and Manning, C. D. (2009). Labeled lda: a supervised topic model for credit attribution in multi-labeled corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, volume 1, pages 248–256.

Ratinov, L., Roth, D., Downey, D., and Anderson, M. (2011). Local and global algorithms for disambiguation to wikipedia. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, pages 1375–1384.

Ritter, A., Clark, S., Mausam, and Etzioni, O. (2011). Named entity recognition in tweets: An experimental study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1524–1534.

Sha, F. and Pereira, F. (2003). Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1, NAACL '03*.

Shen, W., Wang, J., Luo, P., and Wang, M. (2012). Linden: Linking named entities with knowledge base via semantic knowledge. In *Proceedings of the 21st International Conference on World Wide Web*, pages 449–458.

Shen, W., Wang, J., Luo, P., and Wang, M. (2013). Linking named entities in tweets with knowledge base via user interest modeling. In *Proceedings of the 19th Association for Computational Linguistics SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 68–76.

Thater, S., Fürstenau, H., and Pinkal, M. (2010). Contextualizing semantic representations using syntactically enriched vector models. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 948–957.

Yamada, I., Takeda, H., and Takefuji, Y. (2015a). An end-to-end entity linking approach for tweets. In *Proceedings of the the 5th Workshop on Making Sense of Microposts co-located with the 24th International World Wide Web Conference (WWW 2015), Florence, Italy, May 18th, 2015.*, pages 55–56.

Yamada, I., Takeda, H., and Takefuji, Y. (2015b). Enhancing Named Entity Recognition in Twitter Messages Using Entity Linking. In *Proceedings of the ACL 2015 Workshop on Noisy User-generated Text*, pages 136–140.

Yosef, M. A., Hoffart, J., Bordino, I., Spaniol, M., and Weikum, G. (2011). AIDA: an online tool for accurate disambiguation of named entities in text and tables. In *Proceedings of the International Conference on Very Large Databases*, volume 4, pages 1450–1453.

Zhou, G. and Su, J. (2002). Named entity recognition using an hmm-based chunk tagger. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 473–480.

Appendix A

A.1 Features used to train the Twitter model using Stanford NER

- **wordFunction= edu.stanford.nlp.process.LowercaseAndAmericanizeFunction** → this function maps words in the training or test data to new words, and it is used at the feature extractor level. Same as used in CoNLL 4 Class Caseless model and chosen instead of the one in CoNLL 4 Class model (edu.stanford.nlp.process.AmericanizeFunction) because since tweets are highly informal, it is important to be able to identify entities when in lowercase

- **useDistSim = true** → DistSim refers to using features based on word classes/clusters, built using distributional similarity clustering methods. This feature is used to say if we want to load a file of distributional similarity classes (specified by *distSimLexicon*) and use it for features. *egw4-reut.512.clusters* are used

- **useTitle = true** → used to match a word against a list of name titles (Mr, Mrs, etc.)

- **useClassFeature=true** → includes a feature for the class (as a class marginal) and puts a prior on the classes which is equivalent to how often the feature appeared in the training data. Used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useWord=true** → gives feature for word. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useNGrams=true** → used to make features from letter n-grams, i.e., substrings of the word. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **noMidNGrams=true** → use to not include character n-gram features for n-grams that contain neither the beginning or end of the word. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **usePrev=true** → gives features for (position of word, class), and together with other options enables other previous features. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useNext=true** → gives features for (n-grams of word, class), and together with other options enables other next features. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useSequences=true** → set true to use class combination features. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useLongSequences=true** → uses plain higher-order state sequences out to minimum of length or maxLeft. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useNextSequences=true** → set true to use class combination features using next classes. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **usePrevSequences=true** → set true to use class combination features using previous classes. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useSymTags=true** → gives features (position of tag, tag, n-grams for tag, class), (tag, n-grams for tag, class), (position of tag, tag, class)

- **useSymWordPairs=true** → gives features (position of word, n-grams of word, class)

- **useLemmas=true** → used to include the lemma (canonical form, dictionary form, or citation form) of a word as a feature

- **usePrevNextLemmas=true** → used to include the previous/next lemma of a word as a feature

- **normalizeTerms=true** → set true so that some words are normalized: day and month names are lowercased and some British spellings are mapped to American English spellings (e.g., -our/-or, etc.)

- **useTypeySequences=true** → used to apply some first order word shape patterns. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useTypeSeqs=true** → used to apply basic zeroeth order word shape feature. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useTypeSeqs2=true** → used to add additional first and second order word shape features. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useTypeSeqs3=true** → used to add one more first order shape sequence

  Examples of word shape features are common letter sequences and word lengths, along with the presence of key words within the proper noun phrase

- **useOccurrencePatterns=true** → used to capture multiple references to names. If the current word isn't capitalized, followed by a non-capitalized word, and preceded by a word with alphabetic characters, it returns NO-OCCURRENCE-PATTERN. Otherwise, if the previous word is a capitalized NNP, then if in the next 150 words this PW-W sequence is found, we get XY-NEXT-OCCURRENCE-XY, else if W is found we get XY-NEXT-OCCURRENCE-Y. Similarly for backwards and XY-PREV-OCCURRENCE-XY and XY-PREV-OCCURRENCE-Y. Else (if the previous word isn't a capitalized NNP), under analogous rules we get one or more of X-NEXT-OCCURRENCE-YX, X-NEXT-OCCURRENCE-XY, X-NEXT-OCCURRENCE-X, X-PREV-OCCURRENCE-YX, X-PREV-OCCURRENCE-XY, X-PREV-OCCURRENCE-X. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useLastRealWord=true** → if the previous word is of length 3 or less, it adds an extra feature that combines the word two back and the current word's shape. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useNextRealWord=true** → if the next word is of length 3 or less, it adds an extra feature that combines the word after next and the current word's shape. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useBoundarySequences** = true → adds extra second order class sequence features when previous is CoNLL boundary, so entity knows it can span boundary

- **wordShape=WordShapeClassifier.WORDSHAPEDAN2USELC** → rovides static methods which map any String to another String indicative of its "word shape" (e.g., whether capitalized, numeric, etc). Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

- **useDisjunctive=true** → used to include in features giving disjunctions of words anywhere in the left or right *disjunctionWidth* words (preserving direction but not position). Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless
- **useGazettes = true**, **gazette= Dictionary.txt**, **sloppyGazette=true** → features to use gazettes. In includes a flag to say that gazetes will be used, what file to load and that a gazette feature fires when any token of a gazette entry matches. In the file, each line is an entity class name, followed by whitespace followed by an entity, which might be a phrase of several tokens with a single space between words (e.g., PERSON Messi). The gazette was built by mixing some of the dictionaries provided with the source code from Ritter et al. (2011) available at GitHub[1]. The 4 base categories where used in the gazette
- **disjunctionWidth=4** → defines the number of words on each side of the current word that are included in the disjunction features. Same value used in CoNLL 4 Class and CoNLL 4 Class Caseless
- **useObservedSequencesOnly=true** → same value used in CoNLL 4 Class and CoNLL 4 Class Caseless
- **sigma = 20** → same value used in CoNLL 4 Class and CoNLL 4 Class Caseless
- **featureDiffThresh=0.05** → same value used in CoNLL 4 Class and CoNLL 4 Class Caseless

[1]https://github.com/aritter/twitter_nlp