POLITECNICO DI TORINO

Master degree course in Cinema and Media Engineering

Master Degree Thesis

NERD for NexGenTV

Ensemble Learning for Named Entity Recognition and Disambiguation

Supervisors:
Laura Farinetti
Raphaël Troncy

Candidates
Lorenzo Canale

January 2018

Summary

The following graduation thesis “NERD for NexGenTV” documents the contribution I gave to the NexGenTV project during my internship at Graduate school and research centre EURECOM.

In particular, the thesis focuses on two NexGenTV subtasks: Named Entity Recognition (NER) and Named Entity Disambiguation (NED). It presents two multilingual ensemble methods that combines the responses of web services NER and NED in order to improve the quality of the predicted entities. Both represent the information got by the extractor responses as real-valued vector (features engineering) and use Deep Neural Networks to produce the final output.

In order to evaluate the proposed methods, I created a gold standard which consists in a corpus of subtitles transcripts of French political debates. About the matter, I described the ground truth creation phase, explaining the annotation criteria.

I finally tested the quality of my ensemble methods defining standard metrics and my own defined ones.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Information Retrieval has been a crucial research topic in scientific community and, in particular, in the Data Science department of EURECOM. One of the related research projects to which the department contributed is called **NexGenTV** [1]. The project aims to enrich the experience of the television viewer on a support device. In fact, facing the revolution concerning the progressive abandon of television single screen in favor of multiple ones, NexGenTV was thought to offer new possibilities about media consumption:

1. the automatic detection of highlights for a program;

2. the (semi)-automatic enrichment of it with complementary information;

3. the optimization of the user experience via a new interaction in line with viewer's expectations.

These services are offered delayed or live. In the fist case the most important aspect is the quality of the information presented to the user in addition to the program, while the main challenge in the live scenario is the ability of producing real-time contents. Examples of this scenario could be a political debate, in which, additional information related to the debate topic could be useful to help the understanding and to stimulate viewers' interest.

The contribution presented in this master thesis improve a part of the pipeline used to enrich information, with the aim of specializing it for political debates.

[1] http://www.nexgentv.fr/about-us/about-nexgentv

1.1 Motivation and context

Considering the context of a political debate, the first issue to solve is how to represent it. Moving the attention from the political debates to the news in general, in [24], [25] and [9] the authors developed a pipeline that aimed to offer this representation. In particular they introduced the concept of **News Semantic Snapshot (NSS)** and they defined the way to built it starting from a specific news:

- using the newscast broadcasters offered metadata about the items they publish, they build the query $q = [h, t]$, where h is the video heading, and t is the publication date. They use this query as input for **Google Custom Search Engine (CSE)**,[2] collecting event-related documents as result of the query;
- they perform a named entity recognition analysis on the set of documents retrieved by the query q and on the original news transcript extracting a bag of entities. Each entity is a triplet (*surface form*[3], *type*, *link*).
- named entities are then clustered applying a centroid-based clustering operation based on strict string similarity over the *link*, and in case of mismatch, the Jaro-Winkler string distance over the *surface form*, filtering out clusters according to their entity centroids
- at the end the entities are filtered according to their type, keeping only PERSON, LOCATION, and ORGANIZATION, removing the ones with low confidence score keeping only the ones with capitalized surface form.
- NSS can now be modeled following a schema of concentric entity layers. In particular they define two entity layers that can annotate a news item:
  - **Core**: it is composed of a small number of key entities which are essential to identify an event; those entities have the highest degree of representativeness, are frequently mentioned in related documents and are therefore spottable via frequency-based functions;
  - **Crust**: it is composed of a larger number of entities that describe particular details of news.

[2] https://developers.google.com/custom-search/

[3] In a textual document, the surface forms are linear sequences of characters. The surface form related to an entity is the sequence of characters representing this entity in the analyzed text. For instance, in the sentence *François Hollande condamne avec la plus grande fermeté les allégations François Hollande* is the surface form related to the entity referred to the French politician François Hollande

One of the most important step presented in this pipeline is the identification of entities in the documents set. In fact the precision of retrieved entities clearly affects the semantic snapshot generation.

The term "named entities" identifies information units like names, including person, organization and location names, and numeric expressions including time, date, money and percent expressions.

Identifying references to these entities in text was recognized as one of the important sub-tasks of Information Extraction and was called **Named Entity Recognition and Classification (NERC)** or more simply **NER**. In this context a type (a tag) is assigned to each named entity. Early work formulates the NERC problem as recognizing "proper names" in general. Overall, the most studied types are three specializations of "proper names": names of "persons", "locations" and "organizations". These types are collectively known as ENAMEX. Each of these types can in turn be divided into multiple subtypes. The type "miscellaneous" includes proper names falling outside the classic ENAMEX. The class is also sometimes augmented with the type "product", the TIMEX types ("date" and "time") and the NUMEX types ("money" and "percent"). Depending on the context of the extraction, these types can be further augmented.

Let's consider that the types here discussed are the most common, but the list of types depends on the extraction purposes. For this reason, each NER extractor specifies the list of types tags that it uses. In most cases it happens that two different extractors uses different tags but related to the same concepts; for example let's suppose that, to identify a geographical region, the extractor e_1 uses the tag PLACE and the extractor e_2 uses the tag LOCATION.

In these cases, it is necessary to align the different type lists (taxonomies) by mapping the tags retrieved by the first extractor to the ones retrieved by the second, or vice versa. This process is called **type alignment** or **type mapping**.

In the first applications that tried to compare different extractors, this step was done manually. However using a manual type mapping is an iterative and time consuming task because, as the type taxonomies evolve over time, mappings may need to be updated. For this reason, an automatically learned type alignment is usually preferred.

Named entity linking (NEL) or **Named entity disambiguation (NED)** is instead the task of determining the identity of entities mentioned in text. This is done by linking the entities to a **knowledge base**[4] containing them. Each entity in Knowledge Base is idntified by an **Uniform Resource Identifier (URI)**, a string of characters used to identify a resource We take as example the example: *Il va y avoir une dynamique avec Benoît Hamon.* The surface form *Benoît*

[4]A knowledge base is a centralized repository for information: a public library, a database of related information about a particular subject.

Hamon corresponds to an entity and it is linkable with the Wikipedia resource corresponding to it (https://it.wikipedia.org/wiki/Beno%C3%AEt_Hamon).

The term **Named entity recognition and disambiguation (NERD)** identifies both task NERC and NED.

The core of this report is focused on NERD task applied to the debates transcripts of TV programs.

1.2 Research problems

Different web services are available for performing NER and NED tasks. Is it possible to combine their responses in order to create a new more powerful extractor? Which response data are useful for NER purposes? Which ones for NED purposes? Considering that extractors return list of named entities – together with the type and the identifier of each of them –, how this data can be numerically represented? In particular, how a node in a knowledge base can be embedded as a features vector? Which ensemble method should be adopted to exploit all the collected information? Are there other data sources – different from the extractor responses – that are useful for the ensemble method?

In terms of NER, considering that each extractor associates each NE to a type depending on its ontology, is it possible to define an ensemble method that avoids a type alignment step or that computes it automatically, without the human intervention?

In terms of NED, considering that each entity should be disambiguate with a proper entity in knowledge base, which knowledge base should be used?

1.3 Approach and methodology

The NER approaches can be subdivided into three main categories: *dictionary-based*, *rule-based* and *machine learning* approaches.

The *dictionary-based* approaches are based on a list of named entities that they try to identify in the text; for example, in [1] the authors build a complete lexicon containing all the words found in general newspaper text. They intended to provide the reader with an understanding of the inherent limitations of existing vocabulary collection methods and the need for greater attention to multi-word phrases as the building blocks of text. In particular, *dictionary-based* approaches show problems in recognizing proper names.

Rule-based approaches rely on hand-crafted rules. For example, [5] proposes a natural language processing (NLP) system, **FUMES**, which makes use of the internal structure of names and the descriptive information that regularly accompanies them to produce lexical and knowledge base entries for unknown proper names.

Nowadays, hand-crafted rules for recognizing NEs are usually implemented when no training examples are available for the domain or language to process.

When training examples are available, the methods of choice are borrowed from supervised *machine learning* approaches. [39] proposes a Hidden Markov Model (HMM) and an HMM-based chunk tagger for NER. Other approaches, as Entropy Models and Conditional Random Fields were presented respectively in [6] and [8]. More recently deep learning models started to be used for NER. For example, the Python library *spaCy* [5] uses BiLSTM with residual connections, layer normalization and maxout non-linearity.

Also for NED purposes, there is a big variety of approaches. In the seminal approach of Milne and Witten, supervised learning is employed using the anchor texts of Wikipedia entities as training data [17]. Other approaches [38] collect training data based on unambiguous synonyms. [31] describes in detail many other approaches used for NED.

A deep learning approach that was recently proposed and that seems interesting to be explored is described in [13]. Here the authors propose a type-guided semantic embedding approach to boost collective entity linking. They used bidirectional Long Short-Term Memory (BiLSTM) to model the context, and use dynamic convolutional neural network (DCNN) to model the categories, in order to obtain respectively a mention embedding and an entity embedding. Their resulting vector representations x_m and x_e are used to compute a mention-entity similarity score through a similarity matrix M. Then, a join layer concatenates x_m, x_e and the similarity score into a single vector, which is then passed through three fully connected hidden layers. Finally, the output of the hidden layers is further fed to the softmax classification layer, which generates a initial mention-entity linking probability.

Since the 90's, an increasing number of services have been developed for NER and NED. Recently, those tools have been transformed into web services, opening their APIs for public research or commercial use and contributing to the development of a new set of semantic applications.

In this report I will not focus on the creation on a new tool to directly do NER and NED, but I want to combine the outputs of the existing ones in an ensemble method. For this reason, I will not focus in detail on NER and NED models, but I will pay more attention on ensemble methods in literature.

[5] https://spacy.io/

1.4 Contributions

In this report, I present a new ensemble approach to combine the output of the NER and NED extractors available on the Web to improve the F1 scores of the single extractors. It allows to treat the extractors as "black boxes", ignoring how they work and being able to easily add new extractors to further improve the scores. The definition of these ensemble models will be strictly based on deep learning and neural networks.

To reach this purpose, I define a way to numerically represent the types and the entities retrieved by each extractor. At this scope, I analyse different techniques to embed the Wikidata knowledge graph.

In order to evaluate my approach on the political debates transcripts, I annotate part of these files creating a gold standard by this corpus. For this purpose, I define algorithms to help in speeding up the annotation phase.

I test the ensemble approach using the ground truth created by myself and other gold standards commonly used in literature, to be able to compare my results with previous works.

In conclusion, I analyze the reasons why the single extractors responses scores are outperformed by my ensemble methods.

1.5 Report structure

The report is divided into 5 chapter.

Chapter 2 describes the extractors combined in the ensemble methods and analyze related works that experiment an ensemble approach.

Chapter 3 describes the NexGenTV political debates dataset, the ground truth creation, the annotations guidelines and the methods used to speed up this phase, together with its final result.

Chapter 4 analyze some methodologies to embed the Wikidata graph in order to offer a numerical representation of each entity presented in it.

Chapter 5 is focused on the way I built the features from the extractors outputs. These features are useful to form the input samples for the neural networks at the base of my ensemble method.

Chapter 6 describes these neural networks and it focuses on the core of the ensemble methods I adopt for NER and NED purposes. In addition, I evaluate the described method for different gold standards.

Chapter 2

State of the art and related work

2.1 Named entity recognition and disambiguation extractors

As the interest for Named Entity Recognition and Disambiguation grow up, a number of services have been developed to extract structured information from resources published on the Web. Those tools have been transformed into web services, opening their APIs for public research or commercial use and contributing to the development of a new set of semantic applications. These tools generally provide an output composed of a set of extracted named entities, their type and potentially a URI disambiguating each named entities (o = (NE, type, URI)). In this section, I focus on describing the ones I used as part of my ensemble method.

To have an idea about the output of each extractor, I query each of them for the same sentence z:

> *A. Juppé : Vous me posez la question de savoir qui je suis, vous voyez qu je ne suis pas... Je pense que cette outrance est mauvais signe, il y a de la panique à bord. La seule chance de Nicolas Sarkozy de gagner, c'est de capter les voix de l'extrême droite, c'est ce que disait une étude récente. Je ne veux pas m'engager dans cette bataille, je veux parler de mon projet.*

Each extractor returns the output in its own specific format. The output tables in the following shows it after a parsing step, in order to represent all extractors output in the same way.

2.1.1 ADEL

ADEL[1] [22] is a robust and efficient entity recognition and linking framework that is adaptive to text genres and language. Receiving in input a text or an URL pointing to a document (from which it extract text), it analyzes it, extracts named entities and links them to the their disambiguation KB node. The text can be in French or English; however, linking task is not implemented for French language. Because I am working on a French corpus, I only consider the Named Entity Recognition part of ADEL.

ADEL models are trained on many datasets and a query parameter allows to specify which one to use to get the predicted entities. The types vocabulary depends on the used dataset; so, the types associated to the returned named entities differ by changing the dataset. By default, the returned types are PERSON, ORGANIZATION and LOCATION.

The framework architecture is composed of multiple modules spread into two main parts: *Entity Extraction* and *Entity Linking*.

For *Entity Extraction*, the authors used three kinds of extractors: (*i*) Dictionary, (*ii*) POS Tagger and (*iii*) NER. Each of these extractors run in parallel. The entity dictionary reinforces the extraction by bringing a robust spotting for well-known proper nouns or mentions that are too difficult to be extracted for the other extractors (e.g. Role-type mentions). The two other extractors use an external NLP system based on Stanford CoreNLP [14] and particularly the POS [36] and NER taggers.

For *Entity Linking*, they create an index over a targeted knowledge base (e.g. the April 2015 DBpedia snapshot) using the *Indexing Module*. This index is used to select possible candidates with the *Candidate Generation Module*. If no candidates are provided, this entity is passed to the *NIL Clustering Module*, while if candidates are retrieved, they are given to the *Linkers Module*.

The approach is interesting because shows the benefit of combining different CRF models[2] to improve the entity recognition, and to use them as a filter for improving the linking.

Table 2.1 shows the Adel output for the sentence z.

[1] http://adel.eurecom.fr/api/

[2] Conditional random fields (CRFs) are a class of statistical modeling method often applied in pattern recognition and machine learning and used for structured prediction.

Table 2.1: Adel output example

text	start	end	type	uri
A. JUPPÉ	0	7	PERSON	NaN
NICOLAS SARKOZY	30	44	PERSON	NaN

2.1.2 Alchemy

AlchemyAPI[3] has been launched in 2009. Today it is also known as **Natural Language Understanding**, because the original API was acquired by IBM in 2015 and is now part of IBM's Watson Developer Cloud. AlchemyAPI is composed by three principal modules: (i) *AlchemyLanguage* or *Watson Natural Language Understanding*, (ii) *Watson Discovery News*, (iii) *AlchemyVision* or *Watson Visual Recognition*. *AlchemyLanguage* performs Named Entity Recognition, but not Named Entity Linking. Given a text or a URL as input, it founds named entities and assigns types to them. Figure 2.1 contains the complete list of entity types. AlchemyAPI uses deep learning in order to detect entities and assign types. However the model and architecture details are not public, making impossible to provide here a better insight.

AlchemyAPI supports these languages: Arabic, English, French, German, Italian, Portuguese, Russian, Spanish, and Swedish.

Table 2.2 shows the Alchemy output for the sentence z.

Table 2.2: Alchemy output example

text	start	end	type	uri	relevance
NICOLAS SARKOZY	30	44	Person	NaN	0.978347

2.1.3 Babelfy

Babelfy[4] is a unified, multilingual, graph-based approach to Word Sense Disambiguation and Entity Linking based on a loose identification of candidate meanings coupled with a densest subgraph heuristic which selects high-coherence semantic interpretations. The Babelfy approach is explained in [18].

At first the authors create a widecoverage semantic network which encodes structural and lexical information both of encyclopedic and lexicographic kind.

[3] https://www.ibm.com/watson/alchemy-api.html

[4] http://babelfy.org/login

Figure 2.1: Alchemy Types

Academic, AcademicInstitution, Accommodation, Actor, Adherents, AdministrativeDivision, AircraftDesigner, AircraftManufacturer, Airline, Airport, AirportOperator, AmericanIndianGroup, Anatomy, Anniversary, Appellation, Appointee, Appointer, Architect, ArchitecturalContractor, ArchitectureFirm, ArchitectureFirmPartner, ArmedForce, Astronaut, Astronomer, AstronomicalObservatory, Athlete, Automobile, AutomobileCompany, AutomobileModel, AutomotiveDesigner, Award, AwardDiscipline, AwardJudge, AwardNominee, AwardPresentingOrganization, AwardWinner, BasketballCoach, BasketballConference, BasketballPlayer, Bassist, Beer, Beverage, BicycleManufacturer, Blog, Blogger, BoardMember, BoardMemberTitle, BodyOfWater, Book, Boxer, Brand, Bridge, BritishRoyalty, Broadcast, BroadcastArtist, BroadcastContent, BroadcastDistributor, Building, BuildingComplex, CandyBarManufacturer, Cardinal, CauseOfDeath, Celebrity, Cemetery, Chancellor, CharacterOccupation, CharacterSpecies, Cheese, Chef, ChemicalCompound, ChineseAutonomousCounty, ChivalricOrderFounder, ChivalricOrderMember, ChivalricOrderOfficer, ChristianBishop, City, CityTown, Club, Collector, College, CollegeCoach, CollegeUniversity, Comedian, ComicBookCreator, ComicBookEditor, ComicBookFictionalUniverse, ComicBookInker, ComicBookLetterer, ComicBookPenciler, ComicBookPublisher, ComicBookSeries, ComicBookWriter, ComicStripArtist, ComicStripSyndicate, Company, CompanyAdvisor, CompanyDivision, CompanyFounder, CompanyShareholder, CompetitiveSpace, Composer, Composition, ComputerDesigner, ComputerPeripheral, ComputerScientist, ComputingPlatform, ConcertFilm, ConcertTour, ConductedEnsemble, Conductor, ConferenceSeries, ConsumerProduct, Continent, Country, CreativeWork, CricketAdministrativeBody, CricketBowler, CricketCoach, CricketPlayer, CricketTeam, Crime, Criminal, CriminalOffense, Cuisine, CulinaryTool, Cyclist, Dedicatee, Dedicator, Degree, Deity, DietFollower, DisasterSurvivor, DisasterVictim, Disease, DiseaseCause, DiseaseOrMedicalCondition, Dish, Distillery, DrinkingEstablishment, Drug, Drummer, DutchMunicipality, ElectionCampaign, ElementDiscoverer, EmailAddress, EndorsedProduct, Engine, Engineer, EngineeringFirm, EnglishCivilParish, EnglishMetropolitanBorough, EntertainmentAward, Facility, Family, FashionDesigner, FashionLabel, FictionalUniverse, FictionalUniverseCreator, FieldOfStudy, FieldTerminology, FileFormat, Film, FilmActor, FilmArtDirector, FilmCastingDirector, FilmCharacter, FilmCinematographer, FilmCompany, FilmCostumerDesigner, FilmCrewmember, FilmCritic, FilmDirector, FilmDistributor, FilmEditor, FilmFestival, FilmFestivalFocus, FilmMusicContributor, FilmProducer, FilmProductionDesigner, FilmScreeningVenue, FilmSeries, FilmSetDesigner, FilmTheorist, FilmWriter, FinancialMarketIndex, FootballCoach, FootballLeague, FootballManager, FootballOrganization, FootballPlayer, FootballTeam, FoundingFigure, FrenchDepartment, FrenchRegion, Game, GameDesigner, GamePublisher, GeographicFeature, GermanState, GermanUrbanDistrict, Golfer, GovernmentAgency, GovernmentOfficeOrTitle, GovernmentalBody, GovernmentalJurisdiction, Governor, Guitar, Guitarist, HallOfFame, HallOfFameInductee, Hashtag, HealthCondition, HistoricPlace, Hobbyist, HockeyCoach, HockeyConference, HockeyPlayer, HockeyTeam, Holiday, HonoraryDegreeRecipient, Hospital, House, HumanLanguage, IPAddress, Illustrator, IndianCity, IndonesianCity, Industry, InfectiousDisease, Invention, Inventor, Island, IslandGroup, ItalianComune, ItalianRegion, JapaneseDesignatedCity, JapanesePrefecture, JobTitle, Journal, Journalist, Judge, Kingdom, Lake, LandscapeArchitect, LandscapeProject, LanguageCreator, LanguageWritingSystem, Lighthouse, Location, Lyricist, Magazine, ManufacturingPlant, MartialArt, MartialArtist, Mayor, MedicalSpecialty, MedicalTreatment, MemberOfParliament, MembershipOrganization, MeteorologicalService, MilitaryCommander, MilitaryConflict, MilitaryPerson, MilitaryPost, MilitaryUnit, Model, Monarch, Monastery, Money, Mountain, MountainPass, MountainRange, Mountaineer, Movie, Museum, MusicFestival, MusicGroup, MusicalAlbum, MusicalArtist, MusicalGameSong, MusicalGroup, MusicalGroupMember, MusicalInstrumentCompany, MusicalPerformanceRole, MusicalRelease, MusicalTrack, NaturalDisaster, NaturalOrCulturalPreservationAgency, Neighborhood, Newspaper, NoblePerson, NobleTitle, Non-ProfitOrganisation, OfficeHolder, OlympicAthlete, OlympicBiddingCity, OlympicEvent, OlympicHostCity, OlympicVenue, OperaCharacter, OperaCompany, OperaDirector, OperaHouse, OperaSinger, OperatingSystem, OperatingSystemDeveloper, Orchestra, OrganismClassification, Organization, OrganizationSector, Park, PerformanceVenue, PeriodicalEditor, PeriodicalPublisher, Person, Philosopher, Physician, PlaceOfWorship, PlaceWithNeighborhoods, Play, PoliticalAppointer, PoliticalDistrict, PoliticalParty, Politician, President, PrintMedia, ProcessorManufacturer, Product, ProductionCompany, Profession, ProfessionalDegree, ProfessionalField, ProfessionalSportsTeam, ProgrammingLanguage, ProgrammingLanguageDesigner, ProgrammingLanguageDeveloper, ProjectParticipant, ProtectedArea, Protocol, ProtocolProvider, PublicLibrary, PublishedWork, Quantity, RadioNetwork, RadioProgram, RadioStation, Rank, RecordLabel, RecordProducer, RecordingEngineer, RecurringCompetition, RecurringEvent, Region, Religion, ReligiousLeader, ReligiousOrder, ReligiousOrganization, ReportIssuingInstitution, RiskFactor, River, Road, RocketEngineDesigner, RocketManufacturer, RoyalLine, RugbyPlayer, Saint, Satellite, School, SchoolDistrict, SchoolFounder, SchoolMascot, SchoolNewspaper, SchoolSportsTeam, Scientist, ScottishCouncilArea, Senator, ShipBuilder, ShipDesigner, ShoppingCenter, SkiArea, Skyscraper, SoccerClub, Software, SoftwareDeveloper, SoftwareLicense, Songwriter, Soundtrack, SpaceAgency, SpacecraftManufacturer, Spaceport, Sport, SportingEvent, SportsAssociation, SportsFacility, SportsLeagueAwardWinner, SportsOfficial, SportsTeam, Stadium, StateOrCounty, Station, StudentOrganization, Supercouple, Surgeon, Symptom, TVActor, TVChannel, TVCharacter, TVCrewmember, TVDirector, TVEpisode, TVNetwork, TVPersonality, TVProducer, TVThemeSong, TVWriter, Technology, TelevisionShow, TelevisionStation, TennisPlayer, TennisTournamentChampion, Theater, TheaterActor, TheaterCharacter, TheaterChoreographer, TheaterDesigner, TheaterDirector, TheaterProducer, TheaterProduction, TheatricalComposer, TheatricalLyricist, TopLevelDomainRegistry, TouristAttraction, TradeUnion, TransitLine, TransportOperator, TransportTerminus, TwitterHandle, U.S.Congressperson, UKOverseasTerritory, USCounty, USIndianReservation, USPresident, USState, USTerritory, USVicePresident, University, VaccineDeveloper, VentureFundedCompany, VideoGame, VideoGameActor, VideoGameDesigner, VideoGameDeveloper, VideoGameEngine, VideoGameEngineDeveloper, VideoGamePlatform, VideoGamePublisher, VietnameseProvincialCities, VisualArtist, Waterfall, Website, WineProducer, WineRegion, WorkOfFiction, Wrestler, Writer

The network name is **BabelNet** and it is obtained from the automatic seamless integration of Wikipedia and WordNet. The network is built not considering relation types, but only the end points – i.e. vertices – connected by these relations. In addition, the authors provide a structural weighting of the network's edges and for each vertex they create a set of related vertices (*semantic signature*) using random walks with restart.

Given a text, the frameworks extracts all the linkable fragments from this text and, for each of them, lists the possible meanings according to the semantic

network. This output is obtained thanks to the loose candidate identification routine, based on superstring matching instead of exact matching.

To finally predict the right candidate, the framework creates a graph-based semantic interpretation of the whole text by linking the candidate meanings of the extracted fragments using the previously-computed semantic signatures. Then it extracts a dense subgraph of this representation and select the best candidate meaning for each fragment. Through this step, Babelfy fulfills Named Entity Linking purposes, supporting the majority of languages. On the other hand, it does not support type recognition.

Table 2.3 shows the Babelfy output for the sentence z.

Table 2.3: Babelfy output example

text	start	end	type	uri	relevance
CHANCE	20	25	NaN	Q7632586	0.0
NICOLAS SARKOZY	30	44	NaN	Q329	1.0
VOIX	77	80	NaN	Q7390	0.0
EXTRÊME DROITE	87	100	NaN	Q204481	1.0
ÉTUDE	127	131	NaN	Q207841	0.0

2.1.4 Dandelion

Dandelion[5] is a framework composed by a set of APIs, that involve also Entity Linking [21]. The linking goal is reached by using a knowledge graph of places, events, organizations, people and other information. The data forming the graph are collected among numerous proprietary and open data sources. Those data pass by a data normalisation process of data normalisation that includes several steps, among which data cleaning and data harmonisation. Then the entities are deduplicated using Silk Framework [3]. Dandelion API supports the majority of languages.

Table 2.4 shows the Dandelion output for the sentence z. Dandelion doesn't return types for the named entities.

2.1.5 DBpedia Spotlight

DBpedia Spotlight [6] is an open source project responsible for the developing a system for automatic annotation of DBpedia entities in natural language text.

[5] https://dandelion.eu/

[6] http://demo.dbpedia-spotlight.org/

Table 2.4: Dandelion output example

text	start	end	type	uri	confidence
CHANCE	20	25	NaN	Q1970348	0.0328
NICOLAS SARKOZY	30	44	NaN	Q329	0.9185
VOIX	77	80	NaN	Q189760	0.5889
ÉTUDE	127	131	NaN	Q30612	0.0578

It provides programmatic interfaces for phrase spotting (recognition of phrases to be annotated) and disambiguation (entity linking) as well as various output formats (XML, JSON, RDF, etc.) in a REST-based web service. The standard disambiguation algorithm is based upon cosine similarities and a modification of TF-IDF weights (using Apache Lucene[7]). The main phrase spotting algorithm is exact string matching, which uses Aho-Corasick implementation of LingPipe[8]. The detailed description of DBpedia Spotlight is reported by the authors in [7] and [15].

Table 2.5 shows the Dbspotlight output for the sentence z.

Table 2.5: Dbspotlight output example

text	start	end	type	uri	relevance
JUPPÉ	3	7	NaN	Q215569	0.999550
LA	11	12	NaN	Q263478	0.950094
SEULE	14	18	NaN	Q310890	0.999989
CHANCE	20	25	NaN	Q1970348	0.979174
DE	27	28	NaN	Q188	0.306102
NICOLAS SARKOZY	30	44	NaN	Q329	1.000000
DE	46	47	NaN	Q188	0.306102
DE	63	64	NaN	Q188	0.306102
DE	82	83	NaN	Q188	0.306121
EXTRÊME	87	93	NaN	Q465978	0.697596
UNE	123	125	NaN	Q630790	0.999775
ÉTUDE	127	131	NaN	Q12483	0.804187
RÉCENTE	133	139	NaN	Q11084414	0.999640

[7] http://lucene.apache.org/

[8] http://alias-i.com/lingpipe

2.1.6 Meaning Cloud

MeaningCloud[9] is a *Software-as-a-Service* product that enables users to embed text analytics and semantic processing in any application or system. It supports Entity Recognition, but not Entity Linking. It finds named entities in a specified input text and assigns types to them. The used types derive from a specific ontology (Figure 2.2). In addition, it supports many languages: English, Spanish, French, Italian, Portuguese, Catalan.

Table 2.6 shows the MeaningCloud output for the sentence z.

Table 2.6: MeaningCloud output example

text	start	end	type	uri	relevance
A. JUPPÉ	0	7	FullName	NaN	1.0
NICOLAS SARKOZY	30	44	FullName	NaN	1.0

2.1.7 Opencalais

Opencalais[10] is a sophisticated Thomson Reuters web service that attaches intelligent metadata-tags to unstructured content, enabling powerful text analytics. OpenCalais identifies and tags mentions (text strings) in a text based on a list of predefined metadata types: *Anniversary*, *City*, *Company*, *Continent*, *Country*, *Editor*, *EmailAddress*, *EntertainmentAwardEvent*, *Facility*, *FaxNumber*, *Holiday*, *IndustryTerm*, *Journalist*, *MarketIndex*, *MedicalCondition*, *MedicalTreatment*, *Movie*, *MusicAlbum*, *MusicGroup*, *NaturalFeature*, *OperatingSystem*, *Organization*, *Person*, *PersonCarrer*, *PersonEducation*, *PharmaceuticalDrug*, *PhoneNumber*, *PoliticalEvent*, *Position*, *Product*, *ProgrammingLanguage*, *ProvinceOrState*, *PublishedMedium*, *RadioProgram*, *RadioStation*, *Region*, *SportsEvent*, *SportsGame*, *SportsLeague*, *TVShow*, *TVStation*, *Technology*, *URL* .

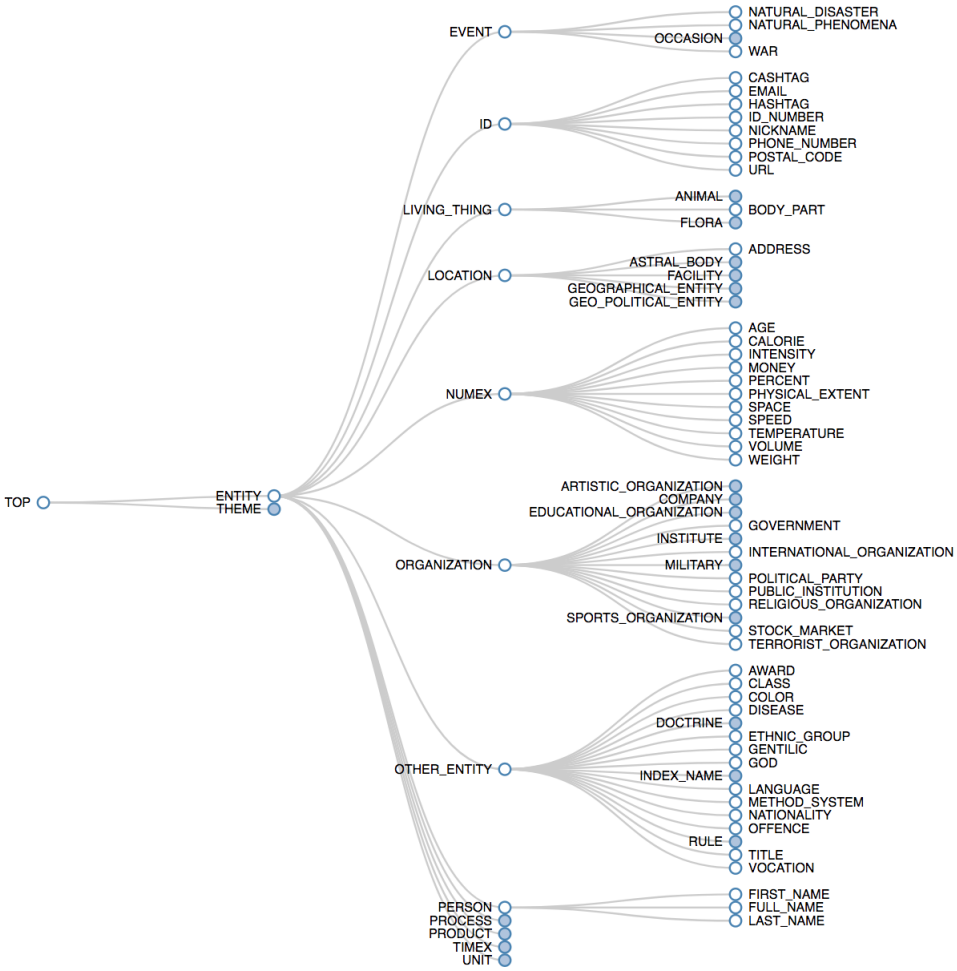
OpenCalais does not perform Named Entity Linking.

For the sentence z, Opencalais do not retrieve NE.

[9] https://www.meaningcloud.com/demos/text-analytics-demo

[10] http://www.opencalais.com/opencalais-api/

Figure 2.2: Meaning cloud ontology



2.1.8 TextRazor

TextRazor[11] uses state-of-the-art Natural Language Processing and Artificial Intelligence techniques to parse, analyze and extract semantic metadata from a textual content. TextRazor is the only tool between the ones mentioned that performs both Named Entity Recognition and Disambiguation. For a specific named entity, it returns:

1. the types associated to the DBpedia ontology;
2. the list of DBpedia types;
3. the list of Freebase types;
4. the disambiguated Freebase ID;
5. the disambiguated Wikidata ID;
6. the link to Wikipedia page.

For the ensemble method that I will describe in the following chapters, I use only the DBpedia types for type recognition purposes and the Wikidata ID for disambiguation purposes. The DBpedia ontology (from which the types are derived) is represented at `http://mappings.dbpedia.org/server/ontology/classes/`.

Table 2.7 shows the TextRazor output for the sentence z.

Table 2.7: TextRazor output example

text	start	end	type	uri	confidence	relevance
NICOLAS SARKOZY	30	44	Politician	Q329	15.870	0.26610
A. JUPPÉ	0	7	OfficeHolder	Q215569	1.385	0.28010
EXTRÊME DROITE	87	100	NaN	Q204481	4.618	0.07472

2.2 Ensemble Approaches

2.2.1 NERD

NERD[12], enable human beings to evaluate the most popular Linked Data named entity extractors [27]. It allows users to analyze any textual resource published on

[11] `https://www.textrazor.com/`

[12] `http://nerd.eurecom.fr/`

the web and accessible through a URI, and to extract from the text the named entities that are detected, typed and disambiguated by various NE extractor APIs. It provides a user interface for assessing the performance of each of those tools according to the pattern (NE, type, URI).

Each service provides its own taxonomy of named entity types it can recognize. The authors therefore designed the NERD ontology which provides a set of mappings between these various classifications. Mapping the original types to the NERD ontology makes possible an evaluation of the quality of each extractor.

The type alignment step was originally designed as static: the authors manually mapped the taxonomies of 12 of these systems to a single ontology, namely the NERD ontology. However in [29], they proposed a more sophisticated and robust approach called **Inductive Entity Typing Alignment** that avoids to manually map types; considering a specific text, let's denote with E the entity list, with T the entity type list, with S the source extractor types, and with GS the types observed in the gold standard. $(E, T)_S$ indicates the ordered list of entities and types given by the source extractor, while O_S is the schema used by the source extractor to type the entities. Let's define $A : T_S \rightarrow T_{GS}$ as the set of alignments given to which they apply a transformation from the T_S to the T_{GS}.

The authors adopted a machine learning induction approach, that aim to learn which entity types as assigned by the extractor outputs correspond to which entity classes in the gold standard. The considered algorithms are k-Nearest Neighbour (k-NN) and Naive Bayes (NB).

In [28] and [26], NERD-ML tool is described. It is a related tool that allows to combine the extractors applying three different machine learning algorithms: Naive Bayes (NB), k-Nearest Neighbor (k-NN) and Support Vector Machines (SVM).

2.2.2 FOX

FOX[13] is a framework that integrates the Linked Data Cloud and makes use of the diversity of NLP algorithms to extract RDF triples of high accuracy out of NL. In its current version, it integrates and merges the results of Named Entity Recognition tools, relying on ensemble learning [32, 34]. California

At the moment, FOX integrates four NER tools: the **Stanford Named Entity Recognizer** (Stanford) [8], the **Illinois Named Entity Tagger** (Illinois) [23], the **Ottawa Baseline Information Extraction** (Balie) and the **Apache OpenNLP Name Finder** (OpenNLP). It only considers the performance of these tools on the classes LOCATION, ORGANIZATION and PERSON. For achieving this goal, the authors mapped the entity types of each of the NER tools to these

[13] https://github.com/dice-group/FOX

three classes. In the literature, it is not clearly mentioned how the type alignment step has been performed.

Given any input text t, FOX forwards t to each of the n tools it integrates. The result of each tool T_i is a piece of annotated text t_i, in which each token is assigned either a particular class or a zero class (not part of the label of a named entity). Each token in t is then represented as a vector of length n which contains the classification assigned to it by each tool.

Once these vectors are formed, in [32] the authors tried 15 ensemble learning algorithms in order to get the final type. They showed as Multilayer Perceptron approach gave the best results. So in [34] the focused on this ensemble learning method. The token vectors previously discussed are forwarded to the multilayer perceptron (MLP), whose input layer contains one neuron for each possible combination of tool and class. The output layer of the network in the MLP contains exactly as many classes as recognized by FOX. The trained neural network returns a classification for each token of t, which is the final classification assigned by FOX for the token under consideration. In a final step, sequences of token which belong to the same class are merged to a single entity.

The FOX MLP approach is similar for many aspects to the one that I implemented and I will discuss in Chapter 6.1.

Chapter 3

Ground truth generation

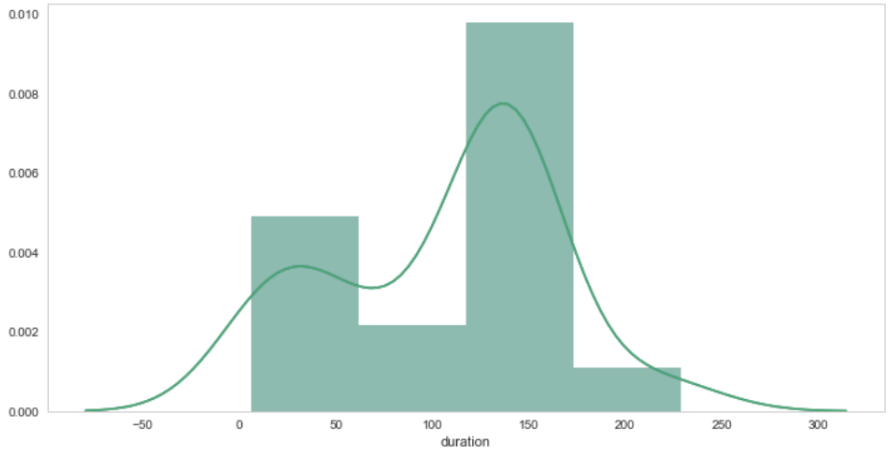
3.1 NextGenTV dataset analysis

NexGenTV offers a dataset formed by 33 subtitles transcripts related to politician television debates (French presidential election, 2017). My goal was to annotate the ground truth in order to evaluate the quality of the ensemble method that I designed (Chapter 6).

I analyzed the debates duration, discovering that the average duration is 107 min and the median duration is 133 min; the distribution is plotted in Figure 3.1.

Considering that these transcripts files are really long, I divided them into temporal fragments lasting 2 minutes; let's identify with $f_{i,j}$ the fragment j for the transcript file i, where $1 \leq i \leq 33$. I randomly select N fragments between them. This random selection avoids to select fragments only coming from a small number of files and, as a consequence, to create a corpus highly sensitive to the content discussed in a specific debate. In total, I annotated 77 fragments, that is the 23% of the entire original dataset.

Figure 3.1: Subtitles duration



3.2 Wikidata ontology description

In order to guarantee an high level of consistency of annotations, I took as reference **Wikidata**[1]. The items and properties in Wikidata that are used to structure the ontology are class (Q16889133),[2] entity (Q35120),[3] Wikidata metaclass (Q19361238)[4], instance of (P31)[5] and subclass of (P279)[6].

Classes are items that conceptually group together similar items, as human (Q5)[7] groups together humans. The items in a class are known as its instances, and they are related to the class via instance of (P31). Classes are related to more-general classes using subclass of (P279), i.e. human (Q5) is a subclass of person (Q215627).[8]

If a class is a subclass of another, then it is transitively a subclass of any

[1] https://www.wikidata.org/wiki/Wikidata:Main_Page

[2] https://www.wikidata.org/wiki/Q16889133

[3] https://www.wikidata.org/wiki/Q35120

[4] https://www.wikidata.org/wiki/Q19361238

[5] https://www.wikidata.org/wiki/Property:P31

[6] https://www.wikidata.org/wiki/Property:P279

[7] https://www.wikidata.org/wiki/Q5

[8] https://www.wikidata.org/wiki/Q215627

more-general classes, so human (Q5) is a subclass of animal (Q729)[9] because it is a super class of person (Q215627). It is not necessary to explicitly state these subclass relationships, so human (Q5) does not have animal (Q729) as a value for subclass of (P279) even though it is a subclass of animal (Q729).

An other important aspect of Wikidata is that each entities is identified by a specific URI (e.g. `http://www.wikidata.org/entity/Q5577`) that is composed by the Wikidata prefix (`http://www.wikidata.org/entity`) and the entity identifier (e.g. Q5577).

3.3 Annotations problems and related tools

In order to annotate the subtitles files, I used **brat rapid annotation tool**.[10] `brat` is a web-based tool for text annotation or, in other words, for adding notes to existing text documents [35]. For my purposes, `brat` is useful to a have a nice interface to annotate documents and because it offers the possibility to upload a collection of files to be annotated. The collection is a folder that contains the text files (**.txt** suffix) that has to be annotated. For each of these files, it is necessary to add an annotation file (**.ann** suffix) in the folder with the same base name and it could be also empty; for example, in the folder *f* you need to find the file *DOC-1000.txt* and its annotation file*DOC-1000.ann*.

Annotations created in `brat` are stored in the annotation files separately from the annotated document text, which is never modified by the tool. All annotations follow the same structure: each line contains one annotation, and each annotation is given an ID that appears first on the line, separated from the rest of the annotation by a single `TAB` character. The rest of the structure varies by annotation type. An example of annotation file is shown below.

```
T1 PERSON 6 12 Salame
T2 ORGANIZATION 44 54 entreprise
T4 PERSON 62 70 salaries
T6 MISC 111 117 themes
T7 MISC 137 148 suppression
T9 MISC 152 158 compte
T10 PERSON 183 200 Francois Hollande
```

[9] https://www.wikidata.org/wiki/Q729

[10] http://brat.nlplab.org/index.html

3.4 Annotations guidelines

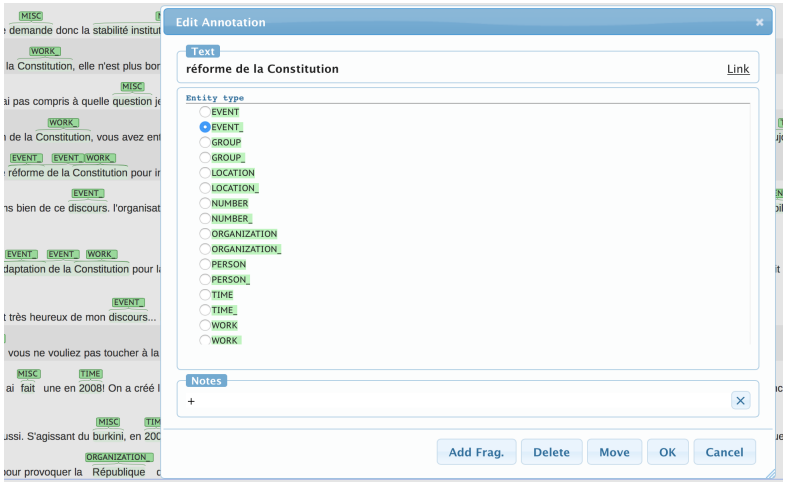
During the annotation phase, I define some criteria to create a consistent ground truth, in particular related to these issues:

1. **Overlapping mentions**, which are two different entity type labels or URIs that are applied to a single phrase in a document. For example, given the sentence:

   > *Lors de l'affaire Kerviel, vous avez demandé la démission du PDG dela Société Générale*

   you would overlap mentions if you annotate *Kerviel* as PERSON and *affaire Kerviel* as EVENT for the single phrase *affaire Kerviel*. Building the ground with `brat`, I annotate both the overlapping mentions, but I mark the one that is more interesting (generally the longest one in terms of chars) with a specific note, in order to easily create a secondary ground truth without overlapping mentions (Figure 3.2).

Figure 3.2: Note example for overlapping mentions



In the example above *réforme de la Constitution* is an EVENT, while *Constitution* is a WORK; adding the symbol + in the notes related to *réforme de la Constitution* allows to recreate a ground truth without overlapping mentions, preserving only the annotation with + as note

2. **Coreferences**, which are two or more expressions in a text refer to the same person or thing. If we consider the example below:

> *- D. Pujadas : Matteo Renzi à sa manière, aller à Bruxelles pour changer l'Europe et ils n'ont jamais réussi. Comment feriez-vous, vous? Avez-vous une baguette magique?*
> *- A. Montebourg : Il y a d'abord des récriminations internationales qui montent contre l'Union européenne.*

the proper noun *Matteo Renzi* and the pronoun *Il* refer to the same person, namely to Matteo Renzi.

I choose to consider also the case of coreferences. However, as done for the overlapping mentions, I mark them with a specific note, in order to easily build a ground truth without coreferences.

3.4.1 Entity annotations

The first step to create the ground truth is to look for all possible named entities in the subtitles documents and then to link each of them with the correspondent Wikidata entity, assigning them the related identifier. To perform the entity research I used the Wikidata search engine, with the surface form of the named entity as input.

However, when the surface form does not exactly match the entity label, Wikidata search engine is not able to find the appropriate disambiguation entity. For example, this is the case for a surface form like *N. Sarkozy*: using it as input string of the search engine I don't get results (Figure 3.3). Further attempt of using third-part tools like **wikidata-autocomplete**[11] have not provided solutions to this issue. For this reason, I implemented a workaround that relies on a search on Google, that usually returns the correct Wikipedia page with the proper label. Using this label as input of the Wikidata search engine or the Wikidata autocomplete tool is possible to retrieve the right entity and the associated URI (Figure 3.4 and 3.5).

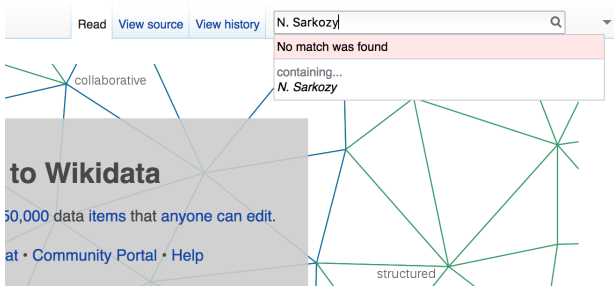
When I find an entity related to a mention, I insert the related Wikidata ID as `brat` note.

3.4.2 Type annotations

I consider as Wikidata classes all the entities that are subject or object of at least one triple with subclass of (P279) as predicate. The target type that I choose corresponds to the subset of Wikidata classes mentioned in Table 3.1.

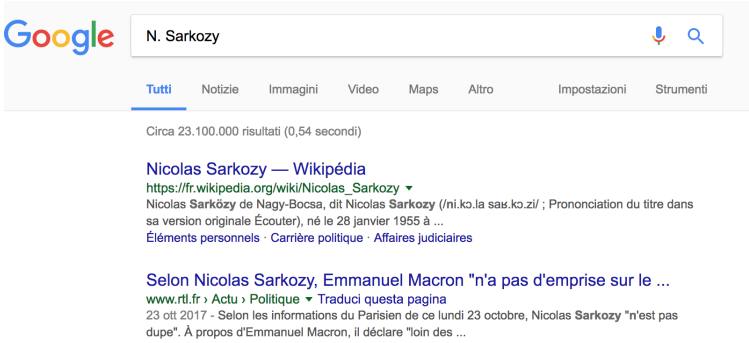
[11] https://athalhammer.github.io/wikidata-autocomplete/wikidata.html

Figure 3.3: Wikidata Search engine (no results)



Looking for *N. Sarkozy* the Wikidata search engine doesn't return result

Figure 3.4: Google Search



Considering as before the example of *N. Sarkozy*, the referred Wikidata entity is Q329[12], that is an instance of human (Q5); so in this case the correct type is PERSON.

Let's consider instead the case in which the surface form presented in the text is *country*. It corresponds to Wikidata entity country (Q6256)[13]. It is not an instance of a class, as France (Q142)[14] could be, but directly a class that is a subclass of geographical point (Q2221906).[15] In this case I add to the correct type

[12] https://www.wikidata.org/wiki/Q329

[13] https://www.wikidata.org/wiki/Q6256

[14] https://www.wikidata.org/wiki/Q142

[15] https://www.wikidata.org/wiki/Q2221906

Figure 3.5: Wikidata Search engine (with results)

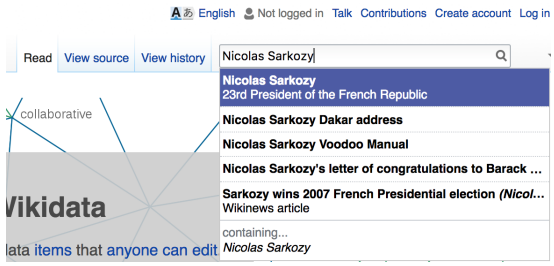


Table 3.1: Wikidata interesting classes

uri	label
http://www.wikidata.org/entity/Q215627	person
http://www.wikidata.org/entity/Q43229	organization
http://www.wikidata.org/entity/Q11471	time
http://www.wikidata.org/entity/Q186081	time interval
http://www.wikidata.org/entity/Q11563	number
http://www.wikidata.org/entity/Q309314	quantity
http://www.wikidata.org/entity/Q2221906	geographical point
http://www.wikidata.org/entity/Q1190554	occurrence
http://www.wikidata.org/entity/Q1656682	event
http://www.wikidata.org/entity/Q15621286	intellectual work
http://www.wikidata.org/entity/Q214339	role
http://www.wikidata.org/entity/Q12737077	occupation
http://www.wikidata.org/entity/Q16334295	group of humans
http://www.wikidata.org/entity/Q231002	nationality

GEOGRAPHICAL POINT a letter C, that distinguishes the between instances and classes mentions.

If the entity annotation step is done before, the types can be dynamically assigned because both steps depend on the same knowledge base (Wikidata). For this purpose I wrote a Python script called *assign_types.py* that looks for the annotated disambiguation URIs and automatically adds the type information; to perform this action is necessary to query the **Wikidata Query Service**[16] for each of the classes mentioned in Table 3.1 in order to get all subclasses and then the instances of each subclass.

[16] https://query.wikidata.org/

Considering for example the class/type organization (Q43229), the query to get all subclasses is:

```
SELECT DISTINCT *
WHERE {
 ?s wdt:P279* wd:Q43229.
}
```

Then I got all instances for each of the subclasses of organization (Q43229), through the query:

```
SELECT DISTINCT *
WHERE {
 ?s wdt:P31 wd:CLASS_IDENTIFIER.
}
```

By processing and managing the results of the queries, I produce a table that represents the association between entities and types. A sample of this table is reported in Table 3.2.

Table 3.2: Uri-type mapping table

WD Id	WD Type
Q37175	PERSON
Q192909	EVENT C
Q142	LOCATION, ORGANIZATION

Looking at the last sample of the table, the identifier Q142, that corresponds to the France's entity, is linked to more than one type. In fact using a set of Wikidata classes as types causes an issue: the types are not mutually exclusive. It happens for two reasons:

1. some selected classes are subclasses of other selected classes; for instance, let's consider group of humans (Q16334295) and organization (Q43229); the latter is a subclass of the former; in these cases I assign as type the most specific, so the subclass (organization (Q43229) in the considered example);

2. some entities are instances of more than one class; taking the entity France (Q142) for example, it is an instance of both organization (Q43229) and geographical point (Q142); in these cases I annotate at first leaving both types, creating a first ground truth where both types are present.

   Then, in a secondary annotation step, I select the most appropriate type depending on the context of the mentions, creating a second ground truth.

In order to to remove arbitrariness from the choice of the most appropriate type, I decided to also take in consideration the DBpedia type. Having already the information about the Wikidata ID, I can retrieve the correspondent DBpedia entity by querying the DBpedia SPARQL Endpoint[17]:

```
SELECT ?db_uri
WHERE {
 ?db_uri owl:sameAs WD_URI.
}
```

Once I get the DBpedia URI, I can derive the type of the respective entity through the query:

```
SELECT ?db_type
WHERE {
 DB_URI rdf:type ?db_type.
}
```

The only limit is that if the DBpedia entity has not a correspondent Wikidata entity, I cannot retrieve the correspondent type starting by the Wikidata entity. However, this case is really rare.

3.4.3 Optimizations

In order to speed-up the annotation phase, I wrote another script called *propagate_annotations.py* that looks for specific notes in the annotation files and execute an action linked to the type of note. For example, let's consider to have the word *F. Lenglet* in the subtitle file a at starting char n and to have already linked it with its disambiguation uri. It is really probable that each time that the surface form *F. Lenglet* is present in the subtitles files, the annotation will be the same. Adding the note *-a* to this mention, the annotation will be propagated into all annotation corpus.

For example, in *fragment_0_48* I annotate the phrase *F. Lenglet* linking it with its Wikidata entity represented by the identifier Q3085156 and accordingly by the URI `https://www.wikidata.org/entity/Q3085156`. I add *-a* in the annotation note, after the identifier (Figure 3.6). It is interpreted as a command by the script *propagate_annotations.py* that replicates the annotation note in all corpus files; the Figure 3.7 shows that the note is replicated in *fragment_0_50*.

Specifying *-f* rather than *-a*, the annotation is only replicated in the fragment that is currently annotated.

[17] `https://dbpedia.org/sparql`

Figure 3.6: fragment_0_48

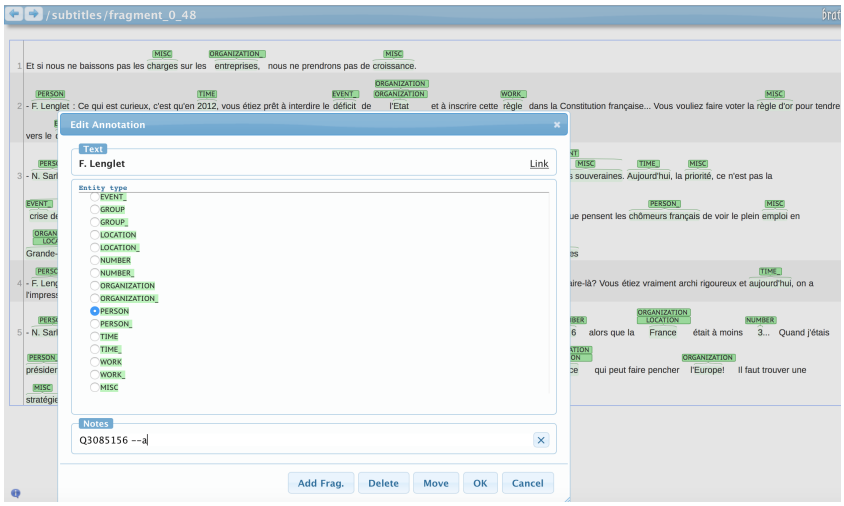
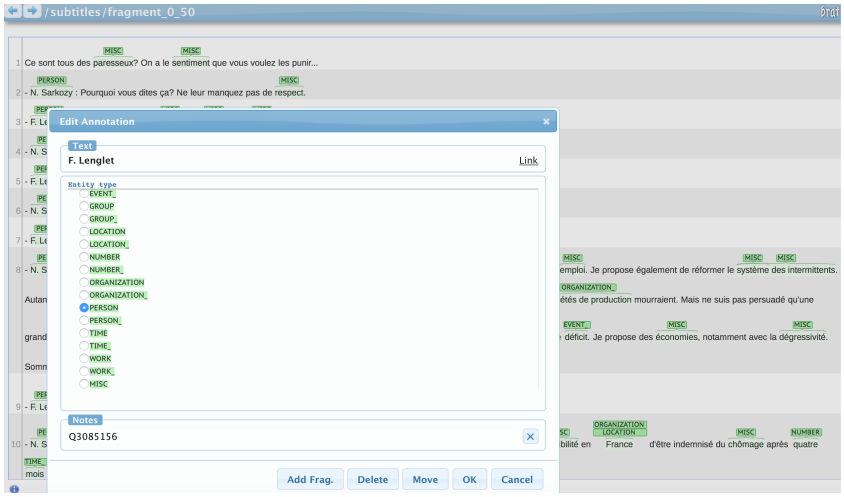


Figure 3.7: fragment_0_50



This propagation procedure sometimes causes mistakes; in fact, supposing to replicate the annotations referred to the phrase *France* founded in the fragment a, it is reasonable to think that in the majority of cases it corresponds to the country France. However, it could happen that the French television channel *France 2*

appears in another subtitle file b; in this case the word *France* in b is wrongly annotated as in a. However the annotation is implemented in order that the same mention is not annotated more then once in the same fragment. So supposing that the file b is already annotated the word *France* presented in *France 2* will not be annotated. At the opposite, if file b is not already annotated, the word *France* will be wrongly annotated. However, it is in charge to the human annotator the deletion of wrong annotations. This process can appear expensive in time consumption, but it is much less than manually looking for the right identifier each time the word *France* appears.

A second optimization that I use to speed up the annotation phase is to avoid to start with completely empty annotation files: the script called **generate_annotations.py** automatically generates annotations using as input sources:

1. the entities extracted by the extractors mentioned in Chapter 2.1 with an high relevance or confidence score or that are extracted by more than n extractors; increasing n, the precision of the automatically annotated mentions increases but the recall decreases;

2. the entities previously annotated; if you have already annotated a set of file F and you are currently annotating a file c, the algorithm uses the phrases that are already annotated in F and looks for them in the new file c and, if it finds once of them, copies the related annotations.

   For example, let's consider the case in which you are annotating the fragment i; if the phrase *affaire Kerviel* was already annotated in a fragment p and if the same phrase is present in i, the annotations notes are copied for *affaire Kerviel* in i.

   It is also possible to set an option o that allows to propagate the annotation also when the phrases are not exactly the same but are really similar, using as similarity criteria the Levenshtein distance and defining a threshold *theta* beyond which the entities are considered different; this usually increases the recall, but decreases the precision.

   Considering the previous phrase (*affaire Kerviel*), the option allows to propagate the associated annotations also when the phrase in the new document is not exactly *affaire Kerviel*, as for *affaire Gerome Kerviel*. However there is the risk to propagate the annotation also when the phrase *Kerviel* appears, and for this reason, it is better to set an high threshold.

   In addition, setting up the option o increases the time to generate the automatic annotation; in this case, considering the phrase *affaire Kerviel*, the corresponding entity link l and a new fragment i, the scripts acts like this:

   - it forms all possible phrases of d consecutive words in i, where $d \leq D$ and D is a constant, that indicates the maximum considered phrase

length (I usually set this parameter to 7); I identify the set of these phrases as P;

- for each of phrase p in P, it computes the Levenshtein distance from the phrase *affaire Kerviel* and if it is higher than the threshold *theta*, it considers p as matched.

Instead, when the option o is not set up the script only checks if the phrase *affaire Kerviel* is in i.

3. the Wikidata entities that have a French label similar to a phrase found in the new file. If option o is set up, a set S of all possible phrases of d consecutive words in the new file is created. Then, the script looks for each phrase of S in the Wikidata French labels, and if it founds it assign the URI of the entity related to the matching label. This is the way to create automatically annotations that causes more mistakes because the Wikidata French labels include also creative works titles.

3.5 Summary

Accordingly with the ground truth creation criteria previously discussed, I annotated the files in such a way it is possible to generate different ground truths, depending on the final needs:

1. with or without overlapping mentions;
2. with or without coreferences;
3. with or without types;
4. with or without disambiguation URIs.

In Chapter 6, I define two ensemble methods, for type recognition and for NE disambiguation. I test both them using a final ground truth $GT_{debates}$. This derives from the annotations that I discussed in this Chapter. This is formed by not considering overlapping mentions and coreferences and assigning, for each NE, the Wikidata disambiguation ID and the DBpedia type.

Chapter 4

Wikidata embedding

In order to pursue scopes as Named Entity Recognition and Linking, I look for a way to represent each entity returned by the extractors as a vector. In particular, I look for a representation in which more the entities are similar, more easily they are confused by the extractors predictions.

More generally Knowledge Base Embedding is a process that serves the purpose to compute vector representations of Knowledge Graph entities; the goal is usually to preserve semantic similarities between them, in order to perform tasks as classification or recommendation. In this chapter I focus on describing how I embed the Wikidata KB, representing the entities as features vectors.

4.1 State of the art

In [37], for instance, the authors investigate how to leverage the heterogeneous information in a knowledge base to improve the quality of recommender systems. By performing knowledge base embedding and collaborative filtering jointly, they can simultaneously extract feature representations from the knowledge base and capture the implicit relationship between users and items. In particular, for the knowledge base embedding, the information stored in the knowledge base can be divided into three parts:

1. **structural knowledge**: this knowledge can be regarded as a heterogeneous network with multiple types of entities and multiple types of links to express the structure of the knowledge base. For structural embedding component, they apply a network embedding procedure called Bayesian TransR, a state-of-the-art embedding approach for heterogeneous network [12].

2. **textual knowledge**: for an item entity such as book or movie in the knowledge base, they use the textual summary to represent the textual knowledge. They apply an unsupervised deep learning model called stacked denoising auto-encoders (SDAE) to get item entities' textual representations from the textual knowledge.

3. **visual knowledge**: for an item entity, except for previous textual description, there are usually some images in the knowledge base, they use a book's front cover image or a movie's poster image to represent its visual knowledge and similarly to previous textual embedding part, they apply another unsupervised deep learning model, termed as stacked convolutional auto-encoders (SCAE), to extract item entities' semantic representations from the visual knowledge.

In [19] the author presents the web service **Webemebedder** [1] for querying an embedding of entities in the Wikidata knowledge graph. To compute the embedding the authors downloaded the Wikidata truthy dump, they filtered it according to some specific properties, stripping the prefixes `http://www.wikidata.org/entity/` and `http://www.wikidata.org/prop/direct/`, getting a list of triples, as shown below:

```
Q22 P1546 Q2016568
Q22 P610 Q104674
Q22 P1151 Q8143311
Q22 P31 Q3336843
Q22 P36 Q23436
Q22 P47 Q21
...
```

Each line can be regarded as a very simple graph walk consisting of a single step from one Wikidata item through a typed property to the next Wikidata item. These triple data are now regarded as a sentence of three words which can be treated by Word2Vec model [16] in the Gensim program to compute the embedding. This is a word embedding method; a word embedding is a learned representation for text where words that have the same meaning have a similar representation. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network, and hence the technique is often lumped into the field of deep learning.

Another approach that is generally used for graph embedding is **`node2vec`** [10]. This framework learns low-dimensional representations for nodes in a graph by

[1] `https://tools.wmflabs.org/wembedder/`

optimizing a neighborhood preserving objective. The objective is flexible, and the algorithm works by simulating a random walk on the graph, generating sequences of nodes, which are then fed in Word2Vec model [16] as if they were sentences of a document to learn vector representations of the nodes. From these representations, the relatedness between two nodes can easily be computed using vector similarity measures. However, in a knowledge graph different properties have different semantic values and should have different weights in judging the relatedness between two entities.

In [20] the authors proposed a framework (`entity2rec`) that solves this issue. They learn property-specific vector representations of knowledge graph entities in a completely unsupervised way via `node2vec`. For each property, node vector representations is computed with `node2vec` and these representation are used as property-specific relatedness scores between users and items. Then the property-specific relatedness scores are combine din a global relatedness score using a supervised learning to rank approach optimizing top-N item recommendation.

All the mentioned approaches consider two entities as similar when they are semantically correlated. As stated before, my notion of similarity is different because it reflects the probability that an extractor confuses the right entity with another in the prediction. Using one of the discussed methods means to consider the two concepts of similarity are near and infers that if two entities are semantically similar, the probability that the extractors predict one rather than the other is high. Is that the case?

I tried to understand which are the typical reasons why the extractors fails to predict the right entity. Obviously, depending on the extractor, the reasons could be different but through the observation of the extractors output and intuitively reasoning about which could be the principal factors that affect the predictions, I considered some possible causes:

1. the extractors could confuse two entities that represent similar concepts; in particular they could take an human for another, or a geographical point with another. In this context it could be useful to keep in consideration the KB structural part, so the properties that express hierarchical relationships: subclass of (P279), instance of (P31), part of (P361).[2] In addition, it could be useful to consider some specific properties related to a specific class; for example, considering the class human (Q5), the property occupation (P106)[3] could be useful to more precisely distinguish between two human instances.

2. the extractors could confuse two entities with a similar label; (e.g., Paris

[2] https://www.wikidata.org/wiki/Property:P361

[3] https://www.wikidata.org/wiki/Property:P106

(Q90)[4], the city, with Paris Hilton (Q47899)[5], the actress). For this purpose, it could be useful to represent the label as features vector. In addition, also considering the abstract describing each entity will be easier to distinguish two different entities.

In order to establish more surely the causes of the wrong predictions it would be necessary to look at the model of the single extractors; but considering that the models are not always public, this solution is unfeasible. In addition, the idea of my ensemble methods is to not focus too much on the single extractors but see them as a "black box", in order to be easily able to add new extractors without thinking about how they work.

Another difference in pursuing recommendation purposes rather than correcting the wrong entity returned by the extractors is that the former case is nearer to clustering perspective in which the items in the same cluster are similar, so it is important to have a global view on all the KB; in the latter, it is not enough to establish that two entities are similar, but it is also important to differentiate two entities that are globally similar, keeping into account features that ensure a good local precision.

In the following of the chapter, I describe the approaches that I experimented to get an embedding that presents the features discussed before.

4.2 Node2Vec

The section describes some experiments I did to test `node2vec` [10] scalability. In particular I focused on understanding if this framework allows a Wikidata embedding in a timely fashion. My internship work lasts six months and, for this reason, I was interested to get the final embedding in a month at most.

To test the `node2vec` feasibility I used two different graphs A and B. In order to form these graphs, I followed these steps:

1. I got a Wikidata dump composed by the unified output of two queries:

```
SELECT DISTINCT *
WHERE {
 ?s wdt:P279 ?o.
}
```

[4] https://www.wikidata.org/wiki/Q90

[5] https://www.wikidata.org/wiki/Q47899

```
SELECT DISTINCT *
WHERE {
 ?s wdt:P31 ?o.
}
```

The first query contains all subclass of relationships; the second one all the instance of relationships. So the dump contains all the hierarchical relationships formed by these properties and presented in Wikidata KB and it considers all the relations among classes and between classes and instances;

2. by using the edgelist got by the query response I formed the graph A;

3. in order to form B, I restricted A, considering only the classes that are instances or subclasses of the Wikidata classes shown in Table 3.1;

4. for each of these classes I got, using the property subclass of (P279), all the superclasses and subclasses and the connections between them, through some SPARQL queries;

5. the formed graph corresponds to B.

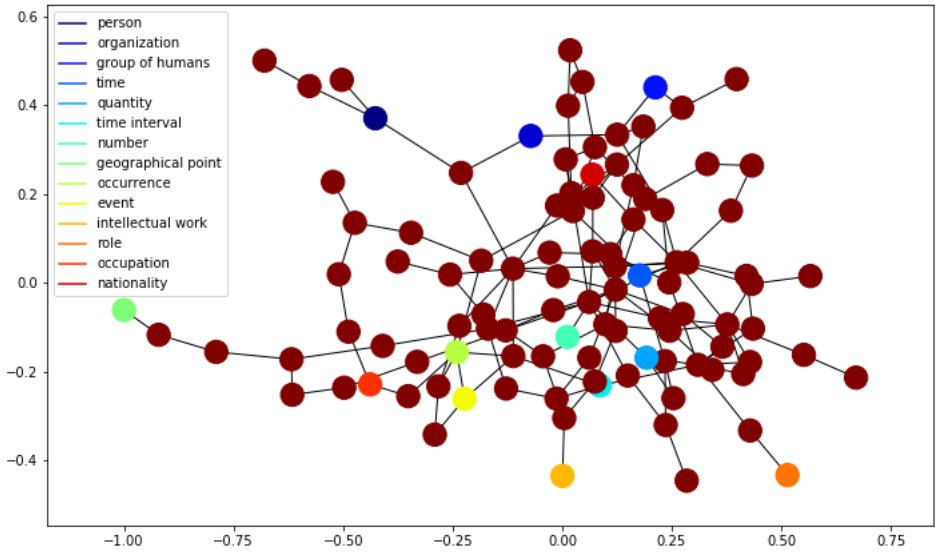
Graph A contains around 1500000 nodes, while B contains around 100000 nodes. The part of B representing the top classes is visible in Figure 4.1.

At this point I tried using `node2vec` to obtain the embedding for both graphs.

As previously mentioned, `node2vec` works simulating many random walks. The framework provides a way of balancing the exploration-exploitation tradeoff that in turn leads to representations obeying a spectrum of equivalences from homophily to structural equivalence. In addition it allows to set some parameters:

- **p** and **q** parameters: while a low **q** encourages outward exploration (homophily emphasis), a low **p** ensures that the walk does not go too far from the start node (structural equivalence emphasis);
- the number of features **d** (default: 128);
- the number of walks **r** (default: 10);
- the walk length **l** (default: 80);
- the neighborhood size **k** (default: 10);
- the number of workers **w** (default: 1).

Figure 4.1: Top classes



In addition, there is another option (**preprocessing**) that allows to preprocess all transition probabilities or compute them on the fly. The first choice is the faster one but requires more RAM capacity. By default the **preprocessing** option is disabled.

In the next section, I analyze the `node2vec` performances for both A and B.

4.2.1 Performance analysis

Before to directly focus on `node2vec` behaviour for A and B, I analyze as the learning quality varies according to the parameters previously described (Figure 4.2).

With the aim of understanding how much each parameter affects the learning time, I iteratively tested `node2vec` changing the parameters one by one and setting with a fix number of edges (around 1,000); this edgelist is got by taking the first 1000 edges from the edgelist that forms the graph A . The experiment was conducted considering both the cases of preprocessing or not the transition probabilities. The results are reported in the Tables 4.1 and 4.2 and Figures 4.3 and 4.4 .

Looking at the correlation values, it is possible to realise that all parameters seem directly or inversely correlated to the duration when the preprocessing option

Figure 4.2: Results extrapolated by [10]

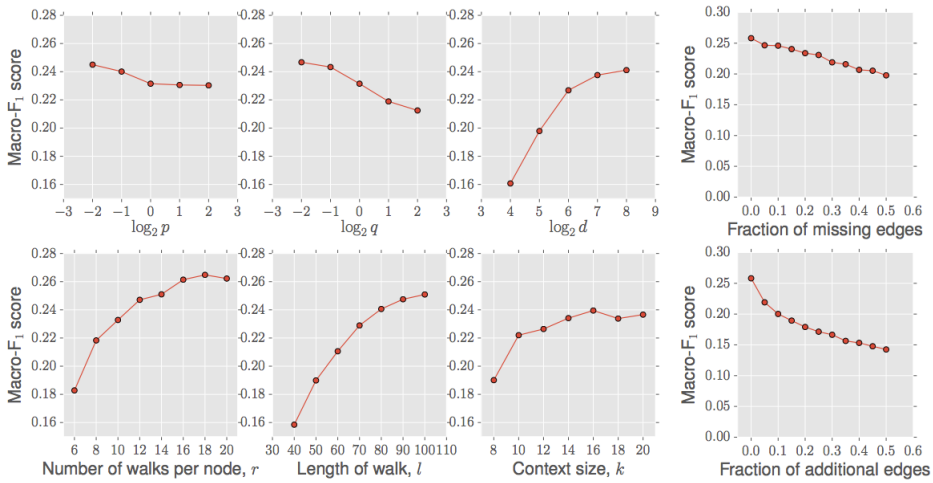
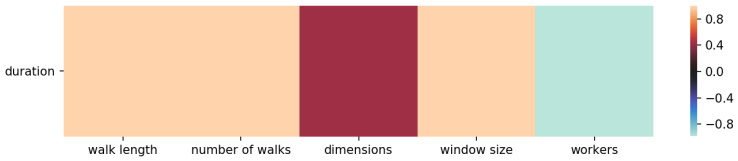


Table 4.1: Correlation values without preprocessing

parameter	correlation with duration
walk length	0.999990524513316
number of walks per entity	0.9999929333856371
dimensions	0.428024784058904
window size	0.9938776427280335
workers	-0.9853259012698113

Figure 4.3: Correlation visualization without preprocessing



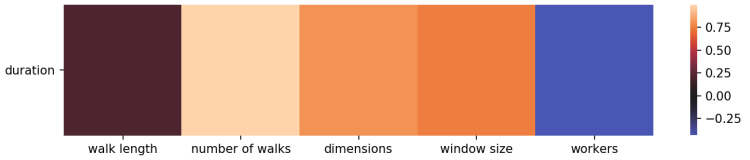
is disabled. The number of dimensions is the least correlated parameter.

Instead, using the preprocessing option, the number of walks (0.99), the number of dimensions (0.8) and the window size (0.75) are the values more correlated. Looking at both cases and at 4.2 it is possible deducing that improving the embedding quality also means increasing the learning time. So it is necessary to look

Table 4.2: Correlation values with preprocessing

parameter	correlation with duration
walk length	0.2104442835579027
number of walks	0.9999710569307252
dimensions	0.8061769084780231
window size	0.7482112524087928
workers	-0.4372913161718269

Figure 4.4: Correlation visualization with preprocessing



for a good tradeoff.

In addition I examined as the learning time varies according to the number of edges and nodes (Figure 4.5), running `node2vec` iteratively and changing the length of the edgelist forming the graph and using the default parameters values. Also in this case, the edgelists that I used derive from the ones that forms the graph A by cutting it depending on the number of edges I wanted to consider.

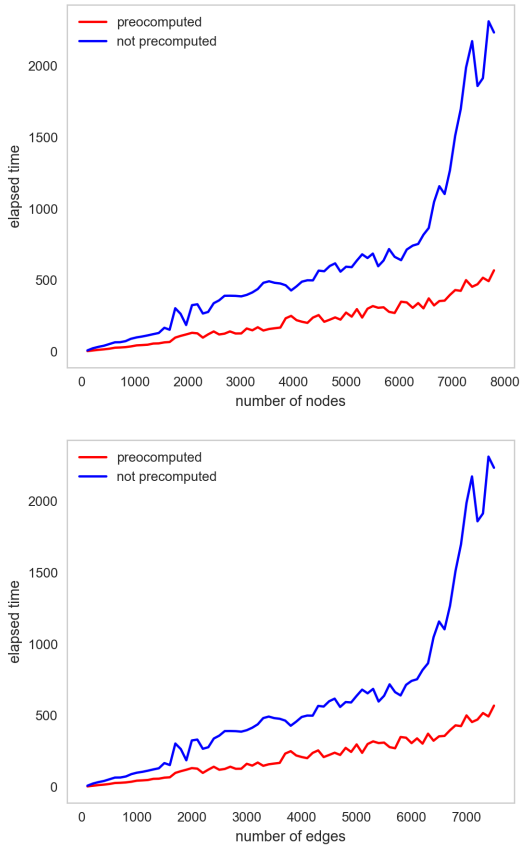
Examining the learning time variation, I observe that the elapsed time linearly increases with the number of edges when the preprocessing option is set, otherwise it exponentially increases.

I can conclude that the preprocessing option makes the algorithm faster and the only limitation is the risk to fit the memory, in particular when dealing with big graphs. In addition I observe that the elapsed time exponentially increases with the number of edges.

To understand how the memory consumption varies according to the number of edges, I run `node2vec` iteratively, changing the length of the edgelist that forms A from 2000 to 10000, with the preprocessing option and the default values for the other parameters.

In Figure 4.6 two trends are shown: the blue one represent the real trend of the memory consumption respect to the number of edges, the green one is a polynomial of degree 2 that approximates the real trend. The latter is useful to have a generic idea about the time consumption when the number of edges is bigger than 10,000 (Figure 4.7). In fact it is not possible to analyze the memory consumption variation for graphs bigger than 10,000 because running the algorithm will require too much time. Considering this, we can not be sure that the trend will be the same for bigger

Figure 4.5: Elapsed time (in seconds) respect to the number of edges and nodes



graphs; in Figure 4.7 a linear variation is visible, but this is a rough approximation that considers the case in which the trend continues to be linear, so the best one.

Reasoning about these behaviours, I can now establish if `node2vec` can compute the embedding for A and B in a timely fashion.

200,000 MiB (21 GB) of RAM would be necessary for a graph composed by 100,000 nodes (Figure 4.7); considering that A is composed by millions of nodes, it's clear that the RAM would be saturated using the preprocessing option. However, not using it really affects the embedding time, that would become greater than 2

Figure 4.6: Trend of the memory consumption according to number of edges

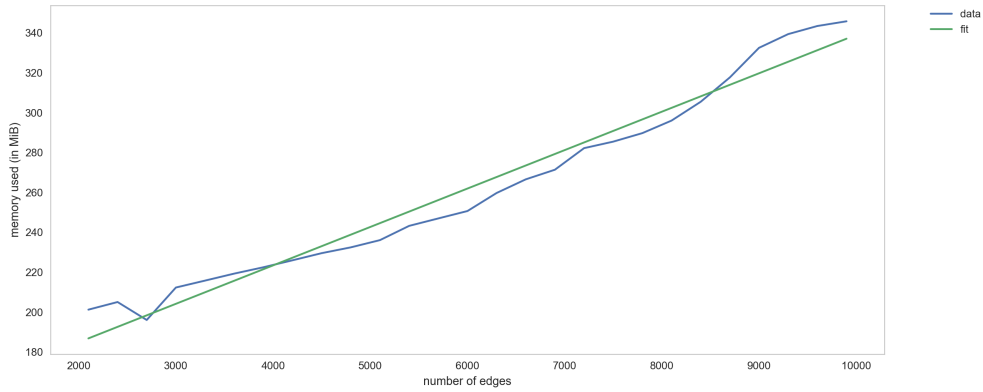
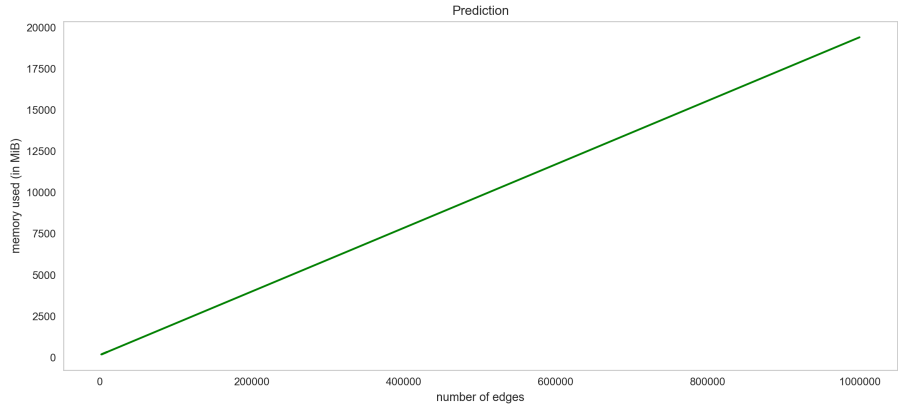


Figure 4.7: Prediction of the memory consumption according to the number of edges



months. This means that computing the embedding for graph A is not feasible, according with the constrains described at the beginning of this section.

On the contrary, graph B is only composed by 100,000 nodes. Considering that I used a machine that has more than 21 GB of RAM, I was able to compute the embedding for this graph with these options:

1. walk length = 100
2. number of walks per entity = 30
3. p = 0.25
4. q = 0.25
5. preprocessing = True
6. dimensions = 150
7. window size = 10
8. iterations = 3
9. workers = 3

The estimated learning duration is one week. In terms of timing, this solution is feasible. As disadvantage, the dump restriction causes that there is not embedding for the instances, but only for some specific classes. In addition, the formed graph contains only structural information and it is composed only by the properties subclass of (P279) and instance of (P31), not considering all the other properties that I considered being useful in Section 3.2.

For these limits, I decided to test other methods to get the Wikidata Embedding Representation.

4.3 All pairs shortest path length matrix and dimensionality reduction

This approach based on two steps: considering a graph G, I compute the all min shortest path matrix and I reduce the matrix dimensionality using PCA or autoencoders.

4.3.1 All pairs shortest path length matrix

Let's consider the Wikidata set of properties P, and a graph G composed by the Wikidata entities linked by at least one of the properties in P. I computed **All-Pairs Shortest Paths Length** for G_P and I obtained a matrix $n \times n$ (symmetric), where the value of each cell $c_{i,j}$ identify the shortest paths length between the node i and the node j. I call this matrix **All pairs shortest path length matrix** (M).

Each matrix row represents a specific node i, and the values contained in the row cells could be seen as a features vector representing the node i.

In such a way, the node representation is strictly correlated with the node position in the graph. However, each features vector is composed by n cells, where is the number of nodes in the graph. For big graphs, as for the Wikidata one, considering this number of features is unfeasible; so it is necessary to reduce the number of features through a dimensionality reduction method. This step leads to delete all the features related to the nodes that are not really discriminative.

Table 4.3: All pairs shortest path length matrix sample

	Q1	**Q2**	**...**	**Qn**
Q1	0	5	...	2
Q2	5	0	...	4
...	...	...	...	...
Qn	2	4	...	0

Each Q represents an entity in the Wikidata graph

4.3.2 Incremental Principal Component Analysis

Because of its huge dimension, loading in memory all M is not possible. Some dimensionality reduction methods are required for managing and processing the matrix.

I used **Incremental PCA** rather than the simple **Principal Component Analysis**. IPCA is typically used as a replacement for PCA when the dataset to be decomposed is too large to fit in memory. IPCA builds a low-rank approximation for the input data using an amount of memory which is independent of the number of input data samples. It is still dependent on the input data features, but changing the batch size allows for control of memory usage. I used the IPCA implementation described in [30], where the authors present an incremental model that is an extension of the **Sequential Karhunen-Loeve Transform** [11].

In order to evaluate the dimensionality reduction quality and the amount of original data that is preserved I performed an experiment. I built a Wikidata subgraph G_S in such a way:

1. using the property subclass of (P279), I considered the Wikidata set of classes C, where each element c in C is subclass or a superclass of the classes specified in Table 3.1; so I formed an edgelist E_1 from which you can derive a graph G_1;

2. using the property instance of (P361), I added to E_1 the edges in which the instances are super classes (subclass of (P279)) of other entities; so I got a new edgelist E_2 and a correspondent graph G_2;

3. using the property part of (P31), I added the edges in which at least one node is in G_2; so I got the edgelist E_3 and the graph G_3.

4. for each item x of the subset X of entities that are objects of the triples formed by the property part of (P31) and that has an associated node in G_3, I associated a virtual node v representing all instances of x. At the end of this step I got I the edgelist E_S and the graph G_{small}.

G_{small} contains 338,202 nodes. This graph represents a part of the hierarchical and structural information presented in Wikidata. However it considers only the entities linked to the classes in Table 3.1 and it doesn't make distinction between two instances of the same class because they are associated to the same virtual node. However, for the goal of evaluating the dimensionality reduction, we can consider it a good compromise.

In order to evaluate how much data is lost during the dimensionality reduction phase, I compute M for G_{small}. Then I define f_i as the matrix vector corresponds to the row i, f_i^r as the reconstructed vector f_i after computing the PCA and n as the number of nodes (338,202) in G_{small}. At this point I compute 2 different measures:

1. $Mean\ Square\ Error = MSE(f_i, f_i^r) = \sum_{i=1}^{n}(f_i - f_i^r)^2$

2. $Coefficient\ of\ determination = R2(f_i, f_i^r) = 1 - \frac{MSE(f_i, f_i^r)}{\sum_{i=1}^{n}(f_i - \frac{\sum_{i=1}^{n} f_i}{n})^2} = 1 - \frac{MSE(f_i, f_i^r)}{\sum_{i=1}^{n}(f_i - \overline{f})^2}$

$MSE(f_i, f_i^r)$ has to be as low as possible and $R2(f_i, f_i^r)$ is a metric that assumes scores comprised between 0 and 1, where 1 indicates the best case.

In this specific case, $MSE(f_i, f_i^r) = 0.004$ and $R2(f_i, f_i^r) = 0.998$. I can conclude that my dimensionality reduction method works because $MSE(f_i, f_i^r)$ assumes a low value and $R2(f_i, f_i^r)$ is really close to 1.

4.3.3 Autoencoders

An autoencoder is an artificial neural network, whose aim is is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction. Architecturally, the simplest form of an autoencoder is a feedforward, non-recurrent neural network very similar to the multilayer perceptron (MLP)[6],

[6] A multilayer perceptron is a neural network composed by an input layer, an output layer and one or more hidden layers connecting them.

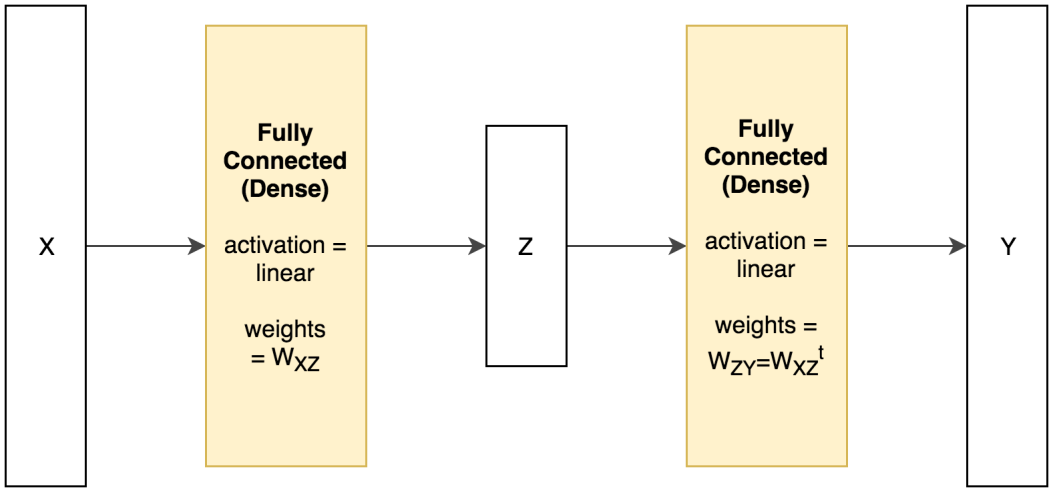
but with the output layer having the same number of nodes as the input layer, and with the purpose of reconstructing its own inputs instead of predicting the target value Y given inputs X.

The advantage of using a NN to reduce the dimensionality is that, as for the PCA, I can pass the input data to the network batch by batch, loading in memory a batch per time.

I used a simple architecture as shown in Figure 4.8: the network is formed by an input layer X that is fully connected to the hidden layer Z, the output of Z is the compressed representation of the input data; Z is fully connected to the output layer Y. The activation functions are always linear. The reason why I opted for a simple and linear network is that, trying more complex architectures with more layers and different activations, I realised that the learning time drastically increases.

In addition I used "tied weights" ($W_{Z,Y}^T = W_{X,Z}$) to reduce the model complexity, the number of parameters and because they act as regularizers. Autoencoders with "tied weights" in linear case are equivalent to PCA and this may lead to more geometrically adequate coding.

Figure 4.8: autoencoder architecture



I computed $MSE(f_i, f_i^r)$ and $R2(f_i, f_i^r)$ where f_i^r is the f_i reconstructed by the autoencoder. In this case $MSE(f_i, f_i^r) = 0.005$ and $R2(f_i, f_i^r) = 0.991$. I can conclude that the values are really similar to the ones got by using IPCA. However IPCA is really faster to compute than training the autoencoder.

4.3.4 Performances analysis

The approach described in this section resent some improvements respect than using `node2vec`. In fact it reduces the time to compute the embedding and it produces a data representation more appropriate for my purposes. In addition it allows to load in memory only a part of data, avoiding to fit it. However the approach still presents some limits; at first I experimented the approach using a graph composed by around 300,000 nodes. If the number of nodes increases, some problems occurs:

1. the computation time increases exponentially with the number of nodes; in addition the percentage of data than the memory is able to handle is less; so the number of batches increases and it causes that the time to read batches from the disk also increases;

2. the available space on disk has to be more.

4.4 Summary

In this chapter I discussed about some criteria to embed Wikidata entities. In particular I tried running `node2vec` for Wikidata and I evaluated the limits. Then I tried adopting a different approach based on all pairs shortest path length matrix and dimensionality reduction. Both approaches are limited to represent the Wikidata information related to the class hierarchy and a structural knowledge but fails to represent a semantic knowledge.

For this reason, I do not use the criteria described before to represent the entity information, but in 5.4 I will discuss an alternative method to represent this kind of information.

Chapter 5

Ensemble approach: features engineering

5.1 Introduction

In this chapter I discuss how to numerically represent the information retrieved by extractors in order to generate features arrays. In Section 2.1, I described the extractors, highlighting that some of them performs only type recognition and some others returns only the disambiguation URIs (Table 5.1).

Table 5.1: Extractors output

Extractor	Type recognition	NE disambiguation
AlchemyAPI	✓	✗
DandelionAPI	✗	✓
DbSpotlight	✗	✓
TextRazor	✓	✓
Babelfy	✗	✓
MeaningCloud	✓	✗
Adel	✓	✗
OpenCalais	✓	✗

N.B. = ✓indicates that the extractor supports the action for the French language

In addition, I mentioned as some extractors return scores. The last source of information that I want keep into consideration is the text passed to the extractors to extract NE, in particular the word embedding associated to each textual token.

We can summarize the sources of information that I want to consider: surface form, type, entity, scores.

In the following, I will analyze them one by one, describing how I can numerically represent this information. The adopted notation will be also used in Chapter 6.

5.2 Surface form features

The surface form features are strictly related to the text used to extract named entity. In fact the text is divided into tokens. For example, let's consider the sentence: *Lors de l'affaire Kerviel*. The sentence is tokenized as shown in Table 5.2.

Table 5.2: Tokens

Surface	Start Char	End Char
Lors	0	3
de	5	6
l	8	8
'	9	9
affaire	10	16
Kerviel	18	24

Each token can be seen as a word, and the goal here is to embed each word/token.

In [2] the authors highlight the advantages of representing a - by a real-valued vector, often tens or hundreds of dimensions. This is contrasted to the thousands or millions of dimensions required for sparse word representations, such as one-hot encoding.

There are many algorithms that follow this goal. The one that I use is called *FastText*[1] [4]. It produce several advantages:

1. *FastText* proposes a new approach based on the skipgram model, where each word is represented as a bag of character n-grams. A vector representation is associated to each character n-gram; words being represented as the sum of these representations; doing this it doesn't ignore the morphology of words as many others word embedding methods;

2. it is fast, allowing to train models on large corpora quickly;

[1] https://github.com/facebookresearch/fastText

3. it allows us to compute word representations for words that did not appear in the training data;
4. it is possible to represent each word in 300 dimensions, that is really a few number of dimensions compared to one-hot encoding representations;
5. it offers the possibility of using precomputed vectors or models trained using Wikipedia for a specific language;

Let's define $\vec{s^n}$ as the real-valued vector associated to a specific word/token n. For the corpus described in 3 I got this vector by concatenating two others vectors:

$$\vec{s^n} = \left[\vec{s_p^n} | \vec{s_c^n}\right]$$

$\vec{s_p^n}$ (300 dimensions) corresponds to the token embedding computed using the pre-computed *FastText* French model. I used this model for computing the vector associated to word presented in my corpus.

$\vec{s_c^n}$ (100 dimensions) is the token embedding computed training *FastText* using all subtitles corpus.

$\vec{s^n}$ is composed by 400 dimensions.

I tested the quality of my ensemble method using also other datasets. They are in English and in this case $\vec{s^n} = \vec{s_p^n}$; this means that I used only the pre-trained English model.

5.3 Type features

Let's define T as the set of extractors that return type information and U as the set of extractors that return disambiguation URIs. TextRazor is the only extractor that is in both sets: $T \cap U = \{TextRazor\}$.

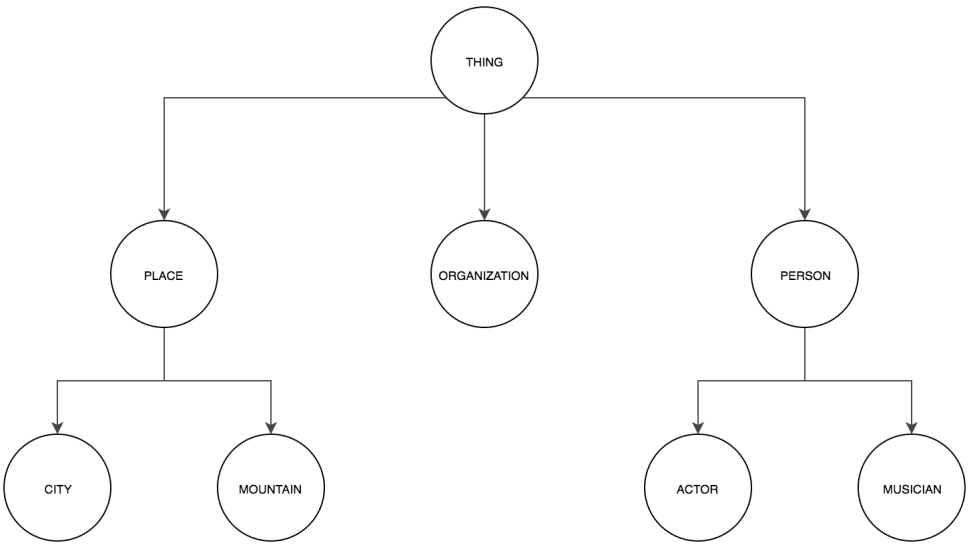
As I mentioned in Chapter 2.1, each extractor in T assigns types accordingly to the ontology that it uses. It could be a specific extractor ontology, as for Meaning Cloud, or a known knowledge base as DBpedia for DandelionAPI. It means that each extractor in T has a different set of types.

In these cases a type alignment step is usually done, in the simplest case manually. However the purposes of my ensemble method is to avoid type alignment in order to be easily able to add new extractors without mapping types. In chapter 6 I will discuss how some layers of my Neural Network learn to align type. For the moment I focus only on how to get vector representation from a type.

Let's consider an extractor $e \in T$ and its own ontology o. The ontology is a tree formed by many levels of hierarchy (L represents the number of levels in the ontology). For simplicity let's assume that the ontology o is formed as shown in 5.1. This is not a real ontology, but it is useful to clarify the idea. It is composed by 2 levels:

1. Level 1 includes three types: PLACE, ORGANIZATION and PERSON.
2. Level 2 includes four types: CITY and MOUNTAIN (subtypes of PLACE) and ACTOR and MUSICIAN (subtypes of PERSON).

Figure 5.1: Ontology example



Let's define C_i as the number of different types for the level i (e.g. $C_1 = 3$ in o). I infer a one-hot encoding representation for each level as shown in Table 5.3.

Table 5.3: Type features

LEVEL 1		**LEVEL 2**	
Type	**Representation**	**Type**	**Representation**
PERSON	001	ACTOR	0001
ORGANIZATION	010	MUSICIAN	0010
PLACE	100	CITY	0100
		MOUNTAIN	1000

Let's consider a generic type τ in the last layer (e.g. ACTOR in o). I get the features vector representing τ walking in tree from the root to the leaf associate to τ, concatenating the one-hot representation of each type founded on the walk. So, the features vector for ACTOR is `0010001`, where the first three values `001` derive from PERSON and the last four values `0001` derive from ACTOR. The features

vector length V is defined as: $V = \sum_i^L C_i$. If the extractor $e \in T$ returns a type that is not the last level in the hierarchy, as PERSON, I fill the missing vector positions with `0`. So the feature vector associated to PERSON is `0010000`.

What described for the ontology o is extensible to all ontologies. However the features vector length is different for each extractor, depending on the ontology that it uses.

What I discussed until now is valid for extractors that return type information. Considering a generic extractor e, where $e \in U \wedge e \notin T$, it returns an URI for each entity returned. The URI is related to a specific knowledge base.

1. if the URI derives from Wikidata, I leave it unchanged;

2. if the URI derives from another KB, I remap it with its corresponding Wikidata URI; for example, DBSpotlight returns URIs related to DBpedia Fr, (e.g `http://fr.dbpedia.org/resource/Nicolas_Sarkozy`); in this case I use the property `foaf:isPrimaryTopicOf`[2] to get the Wikipedia URI (`https://fr.wikipedia.org/wiki/Nicolas_Sarkozy`), using the latter to get the corresponding Wikidata identifier and URI through the MediaWiki API[3] (e.g. Q329);

Table 5.4: Wikidata matching

Extractor	Disambiguation KB	Percentage of URIs matched in WD
Dandelion	Wikipedia	99%
DBSpotlight	DBpedia Fr	98%
TextRazor	Wikidata	100%
Babelfy	DBpedia	100%

In the latter case there is the risk that no one Wikidata node corresponds to the original URI, which it means this information is not present in Wikidata knowledge base. However this case is really rare, as Table 5.4 shows.

Once all the URIs are remapped to the Wikidata KB, I can infer a type for each entity represented by the URIs as described in Section 3.4.2.

I define $\vec{t_e^w}$ as the the features vector representing the type for the named entity w and the extractor e where $e \in U \vee e \in T$.

[2] `http://xmlns.com/foaf/0.1/isPrimaryTopicOf`

[3] `https://www.mediawiki.org/wiki/API:Main_page`

5.4 Entity features

In Chapter 4, I described how to embed Wikidata entities in order to have a numerical representation for each of them. I pointed out a possible solution in Chapter 4.3, but it is still presenting a problem: it allows to represent only the structural knowledge of Wikidata.

In this section I consider the problem of representing the entities information returned by the extractors from another point of view, considering that the goal is to limit the number of disambiguation and recognition mistakes, remembering the possible causes described at the end of Section 4.1.

The approach is based on the idea that, in a ensemble perspective, my goal consists more in representing how the extractors differ in the named entities prediction than in directly representing the single entity. For this reason, I define a similarity metric between two entities based on the extractors outputs and that is able to correct the possible mistakes.

Let's consider two generic entities w_1 and w_2; the similarity between them is expressed as a features vector of 5 dimensions (Figure 5.2). The first four dimensions represent **semantic knowledge**:

1. the first dimension value represents if the compared entities share the same URI; so it is a boolean. I can express this value as $S_{uri}(w_1, w_2)$;

2. the second dimension value is the **Levenshtein distance**[4] between the labels associated to the compared entities. I can express this value as $S_{Lev}(w_1, w_2)$. In order to get the labels of the entities, I used the property `rdfs:label`[5];

3. the third dimension value is the **TF-IDF Cosine Similarity** between the abstracts associated to the compared entities; I can express this value as $S_{TfIdf}(w_1, w_2)$. In order to get the abstracts of the entities, I used the property `schema:description`[6]; this dimension represents a textual knowledge as in [37];

4. the fourth dimension value represents if the compared entities share the same occupation. I can express this value as $S_{occ}(w_1, w_2)$. The occupation is got through a specific Wikidata property: P106[7]. This property is specific for

[4]The Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other

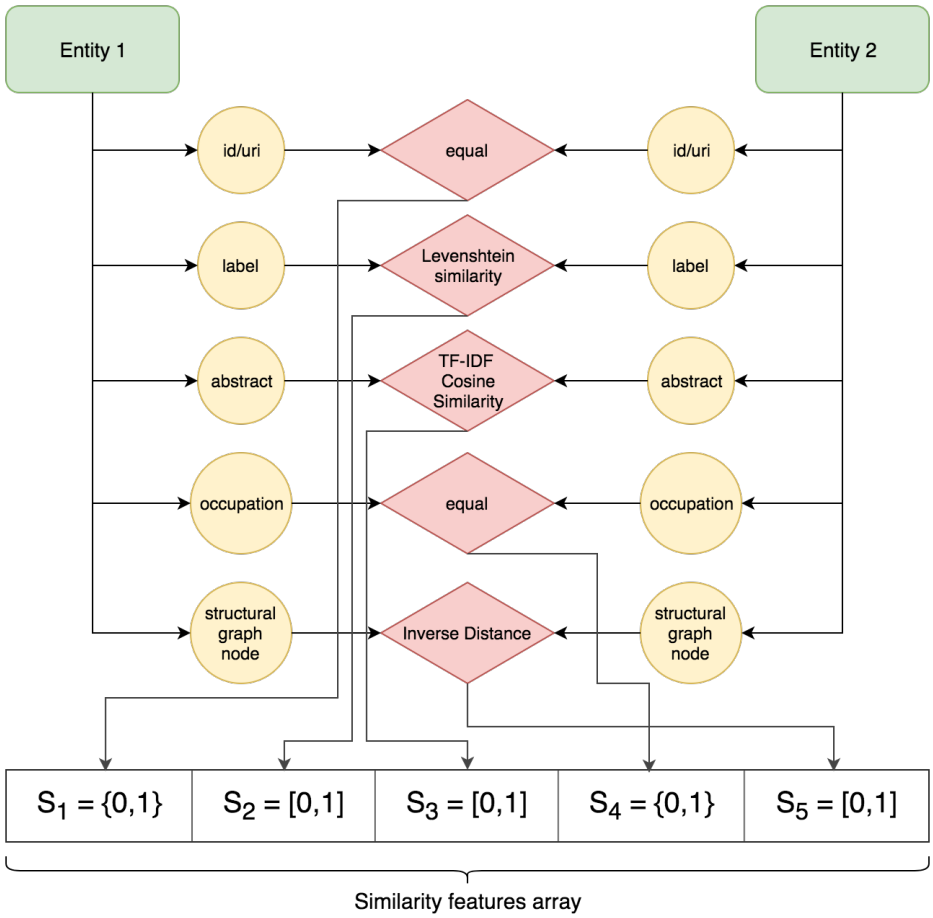
[5]`http://www.w3.org/2000/01/rdf-schema#label`

[6]`http://schema.org/description`

[7]`https://www.wikidata.org/wiki/Property:P106`

entities of type PERSON and it helps in the disambiguation of people with similar names but different professions. This means that $S_{occ}(w_1, w_2)$ can be 1 only when two entities referred to people that practice the same profession are compared.

Figure 5.2: Entities similarity



Last dimension represents the structural knowledge as in [37]. Its value is extracted from a Wikidata subgraph G. Considering the properties set P, the graph is realised by considering all triples in which a property in P appears. The properties in P are:

1. subclass of (P279)
2. instance of (P31)
3. part of (P361)

Once I form this graph, I define the distance d_{w_1,w_2} between two generic entities w_1 and w_2 as the shortest path length that links w_1 and w_2. Then I compute the maximum distance between 2 nodes in the graph G, defining it as d_{max}. Now I can define the structural similarity between w_1 and w_2:

$$S_{struct}(w_1, w_2) = -\frac{d_{w_1,w_2}}{d_{max}} + 1$$

The total similarity between w_1 and w_2 can be expressed like this:

$$\vec{S}(w_1, w_2) =$$
$$= [S_{uri}(w_1, w_2), S_{Lev}(w_1, w_2), S_{TfIdf}(w_1, w_2), S_{occ}(w_1, w_2), S_{struct}(w_1, w_2)]$$

This definition of similarity is useful to generate the entities features. The way in which I will use the similarity vectors to form entities features vector varies accordingly to the NN purposes (type recognition or NE disambiguation) and, for this reason, they will be described directly in Chapter 6. For the moment, let's only consider the notion of similarity and that I will use the similarity vectors rather than the entities embedding to represent the information returned by the extractors.

5.5 Score features

As stated before, some extractors return scores associated to the named entities. We can name these set of extractors as K. I define $k_e^{\vec{w}}$ as the the features vector representing the scores for the named entity w and the extractor e where $e \in K$. The length of $k_e^{\vec{w}}$ depends on the considered extractors, and more precisely, on the number of scores returned by a specific extractor. So, if the extractors w returns two scores, where the former indicates the relevance and the second indicates the confidence, $dim(k_e^{\vec{w}}) = 2$.

Chapter 6

Ensemble methods

6.1 Type recognition

6.1.1 Adopted technique

In order to reach type recognition purposes, I designed a Neural Network that uses the extractors output as input features to outperform the single extractors results.

We define a generic ground truth GT formed by N textual fragments, such that I can split each fragment in tokens. I express the number of tokens for a fragment i as F_i. The total number of tokens is $\sum_{i=0}^{N} F_i$. Let's consider x_i as a generic token and X_i as the ordered list of tokens for fragment i. Concatenating the lists X_i, I get a list X, that is the ordered list of tokens for all corpus. Let's identify x as a token in X.

o_{GT} is the ontology associated to GT and it contains H types. GT associates a type in o_{Gt} to each token. Let's identify the NN target as Y_t. The number of samples in Y_t is equal to the total number of tokens ($\sum_{i=0}^{N} M_i$).

The NN goal is to assign the right type to each token. I identify this NN with the acronym **ENNTR (Ensemble Neural Network for Type Recognition)** and I represent its architecture in Figure 6.1.

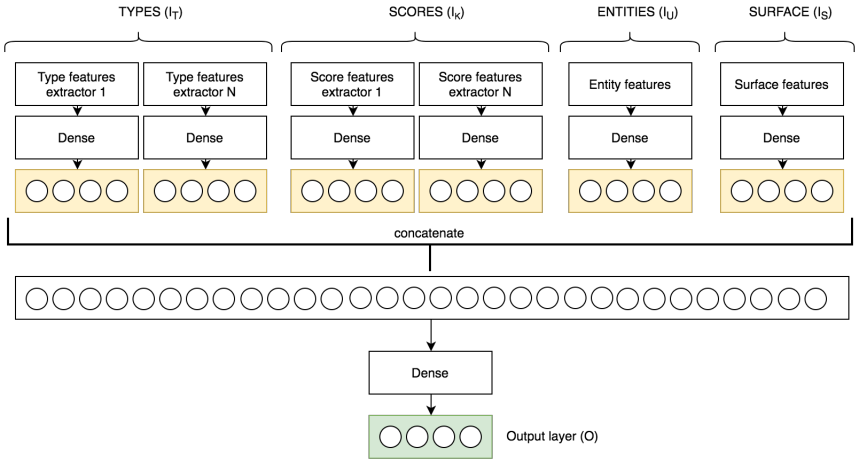
ENNTR has an output layer O formed by H neurons: the number of neurons in the output layer is equal to the number of types in the ground truth Gt. For this reason each value returned by a neuron in the output layer corresponds to the probability that a token x is of a specif type.

Hence each target sample $\vec{y_t}$ is a vector formed by H values, where each value corresponds to a type and a neuron.

In Figure 6.1 I am assuming that $H = 4$.

ENNTR presents many input layers. Using the same notation that I used in Chapter 5, T is the set of extractor that return type information, K is the set of

Figure 6.1: ENNTR architecture



extractors that return score information, U is the set of extractors that performs disambiguation. Let's define the variable I, as the set of input layers of ENNTR. I can identify four different types of input depending on the kind of features enter inside.

$$I = I_T \cup I_K \cup I_U \cup I_S$$

$$|I| = |I_T| + |I_K| + |I_U| + |I_S| = |T \cup U| + |K| + 1 + 1$$

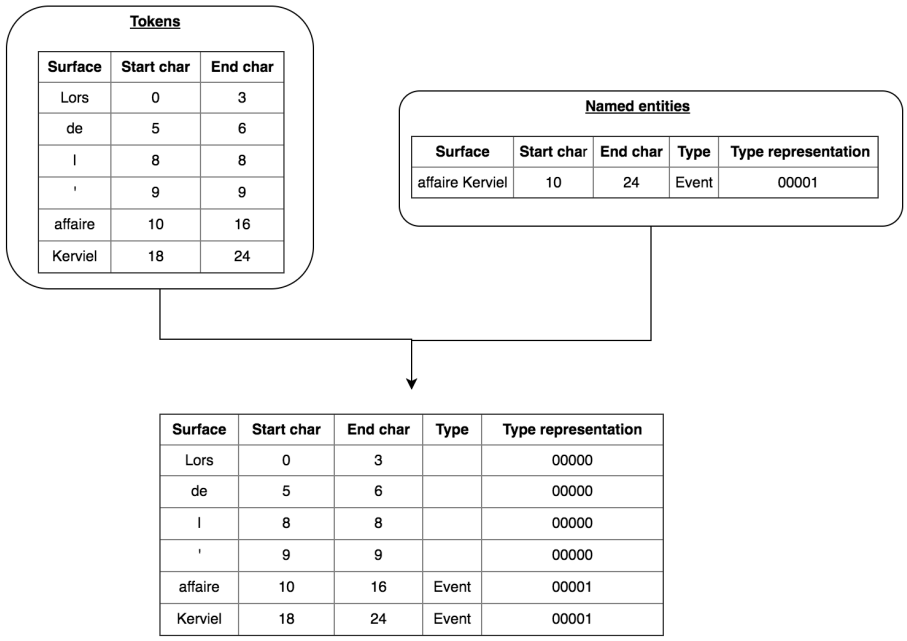
Let's start looking at the input layers I_T. These layers receive the features representing the type information got by the extractors. The number of samples that enter in these layers is equal to $\sum_{i=0}^{N} M_i$. It means that there is a sample per token x and that the tokens related to each fragment g in GT are vertically concatenated to form the input data.

Each sample is represented by a features vector $\vec{t_e^x}$, which is the features vector representing the type of the token x for the extractor e where $e \in U \vee e \in T$. In Section 5.3, $\vec{t_e^w}$ is defined as the features vector representing the type for the named entity w and the extractor e where $e \in U \vee e \in T$. Now I focus on how to get $\vec{t_e^x}$ from $\vec{t_e^w}$. At each named entity w corresponds one or more tokens. So if I tokenize w, I get a set of tokens X_w. Let's define a default vector $\vec{d_t}$, that has the same dimensionality than $\vec{t_e^w}$ but contains only `0` values. I can get $\vec{t_e^x}$ like this:

$$\vec{t_e^x} = \begin{cases} \vec{t_e^w} & if\ x \in X_w \\ \vec{d_t} & if\ x \notin X_w \end{cases} \tag{6.1}$$

An example of this step is represented in Figure 6.2. Here the sentence *Lors de l'affaire Kerviel* is splitted in tokens { *Lors, de, l, ', affaire, Kerviel* } and a generic extractor retrieves the named entity associated to the surface form *affaire Kerviel* and assigns to it the type EVENT, represented by the vector `00001`. For each token in the original sentence that is also part of the surface form related to the found NE I assign the vector `00001`, otherwise the vector `00000`.

Figure 6.2: From NE to type features



What I described for input layers I_T is also valid for I_K. In this case the input layers I_K receive the scores features. The number of samples that enter in these layers is still equal to $\sum_{i=0}^{N} M_i$. $\vec{k_e^x}$ is the features vector representing the scores for the token x and the extractor e where $e \in K$. From $\vec{k_e^w}$ it is possible to derive $\vec{k_e^x}$:

$$\vec{k_e^x} = \begin{cases} \vec{k_e^w} & if\ x \in X_w \\ \vec{d_k} & if\ x \notin X_w \end{cases}$$

The input layers I_U receive the features representing the entity information got by the extractors and more precisely the similarity between the extracted entities.

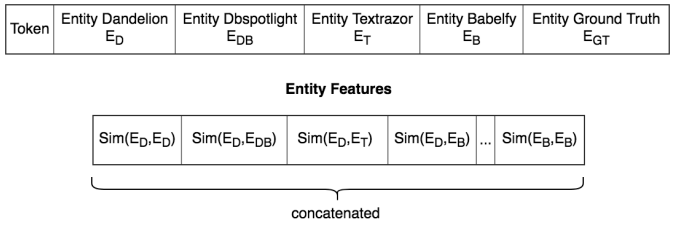
For each fragment g in Gt, each extractors e, such that $e \in U$, returns a list of named entities, in which each entity has its specific disambiguation identifier.

Now I concatenate the lists of named entities of each fragments f in order to form a final list of named entities representing all corpus: so there is a list of named entities l_e for each extractor e.

I tokenize the l_e and I get a set of tokens X_e for each extractor. I assign an entity identifier per extractor e to each token, $e \in U$, in such a way that I obtain a list of entity identifiers u_e for each e of the same length of the list of tokens ($\sum_{i=0}^{N} M_i$). When the extractor does not return an entity for a particular toke, I declare a null identifier (NAN).

After this step, I get the entity w_e predicted by each extractor for a specific token. In Chapter 5.4 I defined the similarity $S(w_1, w_2)$ between two entities $w1$ and $w2$. I can derive the entity features vector for a token x as shown in Figure 6.3.

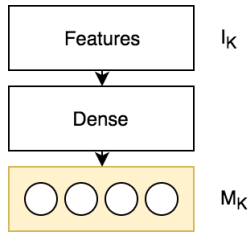
Figure 6.3: Entity features for NER



The input layers I_S receive the surface features as described in Chapter 5.2 ; a Fasttext features vector $\vec{s^w}$ corresponds to each token x.

After describing the features creation and the way in which these features enter in the network, it is time to focus on the other part of the network. In particular, each input layer I_k is fully connected with a layer M_k as shown in Figure 6.4. M_k, as O, is formed by H neurons, where H is the number of types in the ground truth. The activation of M_k neurons is linear.

Figure 6.4: Alignment block



I refer to this part of the network as **alignment block**. It is useful for many

aspects. At first, let's consider that each I_k is formed by a different number of neurons depending on the related features vector; they are mapped on H neurons in M_k. This avoids that the NN risk to privilege features vectors with higher dimensionality – it happens directly concatenating different features vectors. In addition, the value assumed by each neuron of M_k represents the probability that the predicted type is right, using only the features that enter in the related input layer I_k.

The alignment block is also useful to align the type between the ones retrieved by a specific extractor and the ground truth ones. To demonstrate this, I run the NN only considering the alignment block related to the Alchemy type features and I get the predicted types by the values of the neurons in M_k. Table 6.1 shows a sample of the predicted ground truth types when Alchemy returned a type for a specific token.

Table 6.1: Alignment block output sample

surface	alchemy output	alignment block output
president	JobTitle	Role
eugenio	Person	Person
canfari	Person	Person
enrico	Person	Person
university	Organization	Organization
of	Organization	Organization
turin	Organization	Organization
foreign	JobTitle	Role
member	JobTitle	Role

This part of the NN aligns the types between the extractors and the ground truth ontologies. This is pretty similar to the *Inductive Entity Typing Alignment* discussed in [29]. The difference here is that the alignment step is learned by a fully connected layer, not using k-Nearest Neighbour (k-NN) or Naive Bayes (NB). In addition, the alignment and recognition phase are no more separated, because they are part of the same network. This aspect make a further difference between my ensemble method and the one described in [32,34], in which the ontology mapping is not done by the MLP.

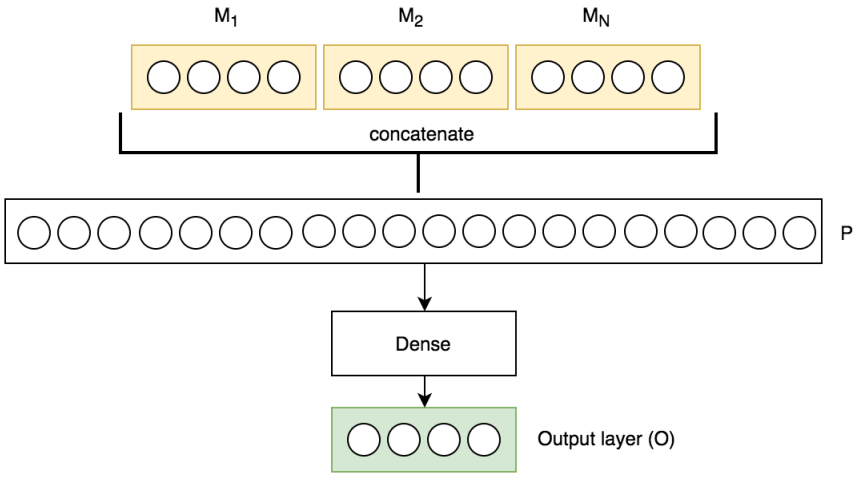
Last part of the network is the **ensemble block** (Figure 6.5).

At first, M_k layers are concatenated forming a new layer P. $|o_{GT}|$ is the number of types in the ground truth, $|I|$ the number of input layers and $|P|$ the number of neurons in P:

$$|P| = |o_{GT}| \cdot |I|$$

P is fully connected to the output layer O. The activation of the neurons in O

Figure 6.5: Ensemble block



is linear. This means that ENNTR is basically a linear combinations of features; the key is the way in which the features are generated and enter in the network.

The values v_h of the H output neurons in O correspond to the probability that a specif type is correct. I take the highest value v_{max} between them and if it is greater than a threshold θ, I set the type related to its neuron as the predicted one.

The final output of the ensemble method is a list of predicted type l_p for each token x.

In a final step, sequences of token which belong to the same type are merged to a single entity, at the same way of [32,34]. From each entity, there is an associated type, start and end char and file from which it derives. So I can generate a brat annotation file for each file in GT. I identify this corpus of files as A_p.

6.1.2 Evaluation

In order to understand the quality of ENNTR I evaluated it with some gold standards. Considering a generic gold standard as GT, I compute two different kinds of scores: *token based* scores and *brat based* scores.

The *token based* are computed similarly to [33], [34] and [32]. From GT I can derive a list of types l_t (target), as long as the number of tokens $|X|$ (each type in the list correspond to a token x in the corpus). Considering the two lists of tokens l_t (target) and l_p (predicted), I compute, for each type t_{GT} in GT, the precision $Precision(l_t, l_p, t_{GT})$, the recall $Recall(l_t, l_p, t_{GT})$ and the F1 score $F1(l_t, l_p, t_{GT})$.

Then I compute macro and micro averaged measures:

1. $Precision_{macro}(l_t, l_p)$: macro precision
2. $Recall_{macro}(l_t, l_p)$: macro recall
3. $F1_{macro}(l_t, l_p)$: macro F1 score
4. $Precision_{micro}(l_t, l_p)$: micro precision
5. $Recall_{micro}(l_t, l_p)$: micro recall
6. $F1_{micro}(l_t, l_p)$: micro F1 score

The *brat based* score are got by using BratUtils.[1] I allows calculating inter-annotator agreement in brat annotation files. This library roughly follows the definitions of precision and recall calculation from the **MUC-7 test scoring**.[2] A_t represent the corpus of brat annotations files got by the ground truth GT. $Precision_{brat}(A_t, A_p)$, $Recall_{brat}(A_t, A_p)$ and $F1_{brat}(A_t, A_p)$ respectively correspond to the precision, the recall and the F1 score got by using brat scorer.

I was also interested to understand if the ensemble method outperforms the single extractors outputs. The problem here is that each extractor uses a specific set of types. To solve this issue I mapped the extractor types to the ground truth ones using the alignment block of ENNTR, as shown in Table 6.1 for Alchemy.

The loss function used to train the network is the Mean Square Error.

OKE2016

The first gold standard that I used to test my ensemble method was **OKE2016**. This ground truth was created for 2016 Open Knowledge Extraction (OKE) Challenge. The OKE2016 types set is composed by PERSON, ORGANIZATION, PLACE and LOCATION. Considering that the number of types is 4, the number of output neurons in ENNTR will be 4 as well. The OKE2016 set is already splitted in train and test data. So, I trained ENNTR using the features formed by the train data, and I evaluated it on the test set, computing both *token based* and *brat based* scores. Tables 6.2, 6.3 and 6.4 show respectively the F1, precision and recall *token based* scores for single extractors and the ensemble.

From this data, I can derive some conclusions. For macro and micro *token based* scores the ensemble method always outperform the single extractors in terms of precision, recall and F1.

[1] https://github.com/savkov/BratUtils

[2] https://aclweb.org/anthology/M/M98/M98-1024.pdf

Table 6.2: F1 scores (OKE2016)

	Organiz.	Person	Place	Role	macro	micro
alchemy	0,83	0,76	0,81	0,67	0,77	0,79
adel	0,94	0,75	0,91	0,83	0,86	0,87
opencalais	0,68	0,52	0,71	0	0,48	0,56
dandelion	0,71	0,65	0,72	0,36	0,61	0,64
dbspotlight	0,61	0,56	0,66	0,52	0,59	0,59
babelfy	0,73	0,63	0,74	0,38	0,62	0,66
textrazor	0,8	0,76	0,82	0,34	0,68	0,74
meaning_cloud	0,64	0,61	0,77	0,18	0,55	0,59
ensemble	0,93	0,98	0,88	0,96	0,94	0,94

Table 6.3: Precision scores (OKE2016)

	Organiz.	Person	Place	Role	macro	micro
alchemy	0,96	0,95	0,89	0,87	0,92	0,93
adel	0,96	0,89	0,85	0,72	0,86	0,88
opencalais	0,96	0,98	1	0	0,73	0,97
dandelion	0,92	0,97	0,81	0,68	0,84	0,89
dbspotlight	0,77	0,91	0,67	0,65	0,75	0,75
babelfy	0,88	0,94	0,8	0,83	0,87	0,88
textrazor	0,82	0,9	0,89	0,95	0,89	0,86
meaning_cloud	0,98	0,82	0,9	0,91	0,9	0,91
ensemble	0,97	0,99	0,96	0,98	0,98	0,98

Table 6.4: Recall scores (OKE2016)

	Organiz.	Person	Place	Role	macro	micro
alchemy	0,74	0,63	0,75	0,54	0,67	0,68
adel	0,92	0,64	0,98	0,98	0,88	0,87
opencalais	0,53	0,35	0,55	0	0,36	0,39
dandelion	0,58	0,49	0,66	0,25	0,49	0,51
dbspotlight	0,5	0,41	0,66	0,44	0,5	0,49
babelfy	0,63	0,47	0,69	0,25	0,51	0,53
textrazor	0,79	0,66	0,77	0,21	0,6	0,65
meaning_cloud	0,48	0,49	0,67	0,1	0,43	0,44
ensemble	0,89	0,97	0,81	0,95	0,91	0,91

I cannot derive the same conclusions looking at the single types scores. Looking

at the F1 score in particular, that is the most important because is the trade-off between precision and recall, I realized that ADEL outperforms the ensemble method for ORGANIZATION and PLACE types. However this conclusion is not completely true, or at least is limited to this specific case. In fact, as explained in 2.1.1, I can query ADEL specifying the model I want to use. In this case I querying ADEL using the model trained on OKE2016, the same dataset that I used for the evaluation. Looking at the ADEL response I have realized that ADEL gives really higher scores on the test set than on the training. However, I am using the features formed by ADEL training response to train my Neural Network.

The *brat based* scores for OK E2016 are reported in Table 6.5.

Table 6.5: Brat based scores (OKE2016)

	fsc	**pre**	**rec**
adel	0,84	0,85	0,83
alchemy	0,88	0,92	0,86
babelfy	0,74	0,79	0,7
dandelion	0,78	0,83	0,75
dbspotlight	0,6	0,77	0,52
meaning_cloud	0,72	0,78	0,69
opencalais	0,69	0,71	0,68
textrazor	0,77	0,81	0,74
ensemble	0,94	0,95	0,92

Looking at these scores, I can conclude that the ensemble method outperforms the single extractors for F1, precision and recall.

To understand which are the features that contribute more to the obtained result, I trained 4 other different NNs:

1. the first one is equal to ENNTR but it considers only the input layers related to type features: $I = I_T$
2. the second one is equal to ENNTR but it considers only the input layers related to score features: $I = I_K$
3. the third one is equal to ENNTR but it considers only the input layers related to entity features: $I = I_U$
4. the fourth one is equal to ENNTR but it considers only the input layers related to surface features: $I = I_S$

The F1 *token based* scores on test set for each of these NNs are reported in Table 6.6. They clearly show that the most useful features are the types ones (F1

Table 6.6: Features F1 (OKE2016)

	Organization	Person	Place	Role	macro	micro
type	0,86	0,88	0,92	0,91	0,89	0,88
score	0,34	0,30	0,00	0,00	0,16	0,23
uris	0,57	0,56	0,00	0,00	0,28	0,40
surface	0,39	0,65	0,38	0,58	0,50	0,49

macro is 0.89 and F1 micro is 0.88). Then, there are the surface ones (F1 macro is 0.50 and F1 micro is 0.49). Entity features contribute slightly less to the final result (F1 macro is 0.28 and F1 micro is 0.40), while the score features are the least useful (F1 macro is 0.00 and F1 micro is 0.16). The reason is in the fact that scores features serve only to reinforce the type features and are not intended to be considered them alone.

French Debates Subtitles

The second ground truth for which I tested ENNTR is the one I annotated. It is in French, so that it allows to evaluate the ensemble method on a language different from English. I splitted the dataset in train set (64 fragments) and test set (13 fragments). Then I trained ENNTR on the training framents and I tested it on the test fragments.

Tables 6.7, 6.8 and 6.9 show respectively the F1, precision and recall *token based* scores for the single extractors and for the ensemble.

Table 6.7: F1 scores (subtitles transcripts)

	Organization	Person	Place	macro	micro
alchemy	0,00	0,87	0,00	0,29	0,80
adel	0,00	0,75	0,00	0,25	0,68
opencalais	0,00	0,62	0,51	0,37	0,58
dandelion	0,00	0,30	0,00	0,10	0,26
dbspotlight	0,08	0,55	0,00	0,21	0,48
babelfy	0,00	0,59	0,51	0,36	0,55
textrazor	0,36	0,88	0,10	0,45	0,81
meaning_cloud	0,33	0,86	0,61	0,60	0,82
ensemble	0,44	0,95	0,82	0,74	0,92

Looking at these Tables, the ensemble method always outperforms the single extractors in terms of precision, recall and F1, for both macro and micro scores and single types scores. The single extractors scores appear much lower then the

Table 6.8: Precision scores (subtitles transcripts)

	Organization	Person	Place	macro	micro
alchemy	0,00	0,92	0,00	0,31	0,77
adel	0,00	0,68	0,00	0,23	0,57
opencalais	0,00	0,48	0,41	0,30	0,45
dandelion	0,00	0,19	0,00	0,06	0,16
dbspotlight	0,04	0,40	0,00	0,15	0,34
babelfy	0,00	0,42	0,54	0,32	0,41
textrazor	0,25	0,85	0,05	0,39	0,74
meaning_cloud	0,21	0,84	0,51	0,52	0,77
ensemble	0,29	0,97	0,76	0,67	0,90

Table 6.9: Recall scores (subtitles transcripts)

	Organization	Person	Place	macro	micro
alchemy	0,00	0,92	0,00	0,31	0,77
adel	0,00	0,68	0,00	0,23	0,57
opencalais	0,00	0,48	0,41	0,30	0,45
dandelion	0,00	0,19	0,00	0,06	0,16
dbspotlight	0,04	0,40	0,00	0,15	0,34
babelfy	0,00	0,42	0,54	0,32	0,41
textrazor	0,25	0,85	0,05	0,39	0,74
meaning_cloud	0,21	0,84	0,51	0,52	0,77
ensemble	0,29	0,97	0,76	0,67	0,90

ones computed for OKE2016. On the other side, this fact allows a much stronger improvement of results when the ensemble method is applied. In particular, the recall improvement shows that ENNTR is able to sum up the knowledge deriving from different extractors.

Table 6.10 shows the *brat based* scores for OKE2016.

Looking at these scores, the ensemble method outperforms the single extractors for F1, precision and recall. Only Alchemy reaches scores comparable to the ensemble ones.

Also in this case, I wonder which features contribute more to the ensemble method. So I trained four NNs, each one with a single feature enabled. Table 6.11 reveals once again that type features are the ones that contribute more to the result. Score and entity features seem to mainly contribute on PERSON type, while surface features are more useful for PLACE recognition.

Table 6.10: Brat based scores (subtitles transcripts)

	fsc	pre	rec
adel	0,75	0,83	0,7
alchemy	0,87	0,97	0,81
babelfy	0,65	0,74	0,59
dandelion	0,51	0,69	0,42
dbspotlight	0,5	0,61	0,45
meaning_cloud	0,8	0,87	0,76
opencalais	0,81	0,9	0,76
textrazor	0,75	0,8	0,72
ensemble	0,92	0,98	0,87

Table 6.11: Features F1 (subtitles transcripts)

	Organiz.	Person	Place	macro	micro
type	0,29166667	0,94620253	0,64864865	0,62883928	0,87533157
score	0	0,83860759	0	0,27953586	0,70291777
entity	0	0,79746835	0	0,26582278	0,66843501
surface	0	0,60759494	0,45945946	0,35568480	0,55437666

6.2 Entity Linking

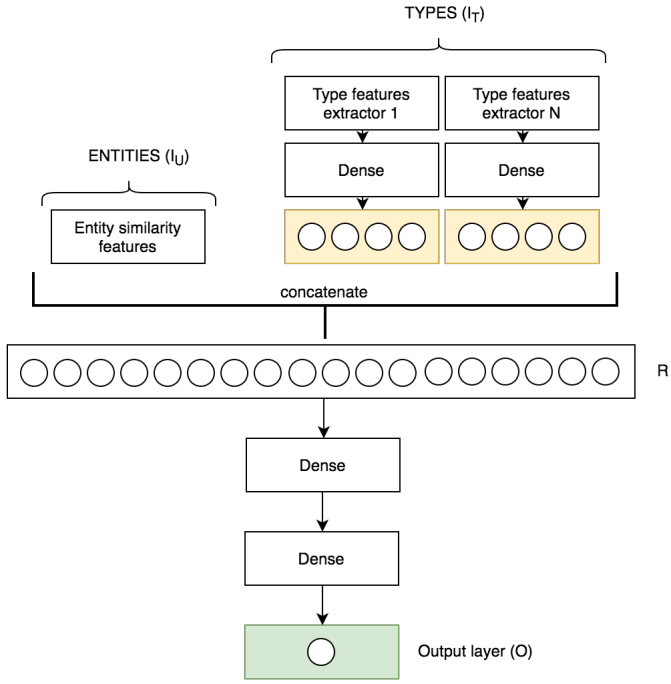
6.2.1 Adopted technique

In order to reach Wikidata entity linking purposes, I designed a Neural Network, to which i will refer as **ENND (Ensemble Neural Network for Disambiguation)**. The architecture is shown in Figure 6.6.

Let's define a generic ground truth GD formed by N textual fragments, such that I can split each fragment in tokens. I express the number of tokens for a fragment i as M_i. The total number of tokens is $\sum_{i=0}^{N} M_i$. Let's consider x_i as a generic token and X_i as the ordered list of tokens for fragment i. Concatenating the lists X_i, I get a list X, that is the ordered list of tokens for all corpus. Let's identify x as a token in X. GD associates a Wikidata entity identifier (URI) to each token. Let's identify the target as Y_d.

The ENND goal is not to directly predict the right disambiguation Wikidata entity but to determine if the predicted entity by an extractor e, where $e \in U$, is correct. For this reason the number of samples in target Y_d is not more equal to the number of tokens. Let's consider a token x; each extractor e ($e \in U$) returns a predicted entity w_{x_e}: I call the set of predicted entities for token x C_x, and the correct entity v_x; $|C_x| \leq |U|$ because more extractors could predict the same

Figure 6.6: ENND architecture



entity. For each candidate $c_{x,j} \in C_x$, where $0 < j \leq |C_x|$, I generate a target sample $y_d \in Y_d$:

$$y_d = \begin{cases} 1 & if\ c_{x,j} = v_x \\ 0 & if\ c_{x,j} \neq v_x \end{cases}$$

The output layer O contains a single neuron that should converge to y_d. The O activation is a sigmoid.

ENND has many input layers. Let's define the variable I, as the set of input layers of ENND. I can identify two different types of input depending on the kind of features enter inside.

$$I = I_U \cup I_T$$

$$|I| = |I_U| + |I_T| = 1 + |T \cup U|$$

The entity similarity features enter through I_U. A similarity features sample is associated to each target sample y_d, that allows to compute an entity similarity features vector for each candidate $p_{x,j}$ in such a way:

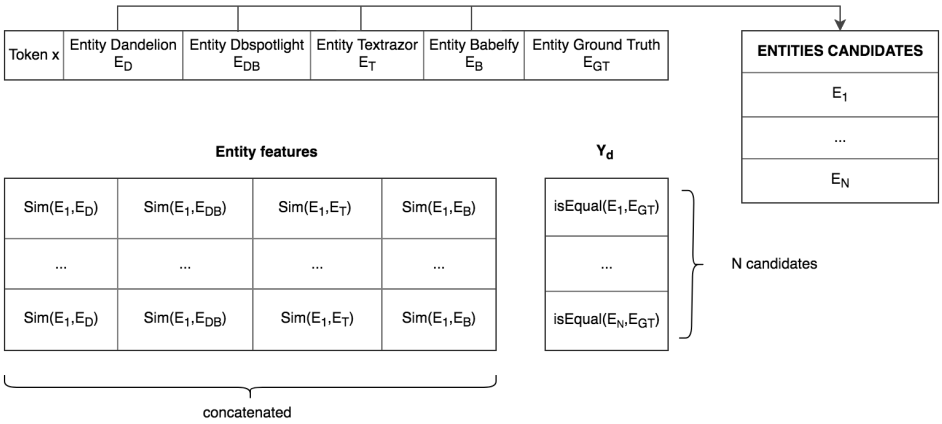
$$\vec{u_{x,j}} = [\vec{S}(p_{x,j}, w_{x_1})|\vec{S}(p_{x,j}, w_{x_2})|...|\vec{S}(p_{x,j}, w_{x_n})] \; where \; n = |U|$$

Considering my specific case, in which the full set of extractor is equal to $U = \{Dandelion, DbSpotlight, TextRazor, Babelfy\}$, and remembering that a generic $\vec{S}(w_1, w_2)$ is composed by 5 dimensions, we can state that:

$$dim(\vec{u_{x,j}}) = dim(\vec{S}(w_1, w_2)) \cdot |U| = 5 \cdot 4 = 20$$

$dim(\vec{v})$ expresses the dimensionality of a generic vector $\vec{v}$.
The way in which $\vec{u_{x,j}}$ is formed is represented in Figure 6.7.

Figure 6.7: Entity features for NED



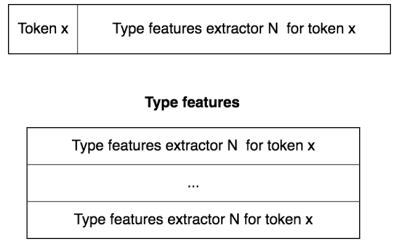
The input layers I_T receive the features representing the type information got by the extractors. The number of samples that enter in these layers is equal to the number of samples in target.

As for type recognition, for each $e \in U \vee e \in T$, the feature vector $\vec{t_e^w}$ represents the type for the named entity w, while $\vec{t_e^x}$ represents the type for the token x. I can get $\vec{t_e^x}$ from $\vec{t_e^w}$ as described in 6.1.

It can happen that, for each token x, there are multiple entity candidates. I assign at each candidate $p_{x,j}$ the vector $\vec{t_e^w}$ correspondent to the token x, from which $p_{x,j}$ derives (Figure 6.8).

I_T layers are fully connected to the layers M_k as in ENNTR (Figure 6.4). M_k, as O, is formed by H neurons; however, in this case, H is no more the number of types in the ground truth, but it indicates the number of dimensions to encode the type information got by each extractor. In fact, it is not mandatory that GD contains also the types associated to each entity because it serves for entity linking

Figure 6.8: Type features for NED



and not for type recognition. Clearly, if GD has type information, the best option is to set H equal to the number of types. As for ENNTR, the M_k activation is linear.

After this step, the I_U layer and the M_k layers are concatenated forming a new layer R. In this layer some neurons some neurons represent the type information, some other the entity features; the idea is to exploit the fact that some extractors are more precise on certain types. The number of neurons in P is equal to $dim(\vec{u_{x,j}}) + |T \cup U| \cdot H$.

The last part of the network is composed by two dense layers[3] and the output layer O discussed before. The activation functions of the dense layers are **Scaled Exponential Linear Units (selu)**:

$$selu(x) = \lambda \begin{cases} x & if\ x > 0 \\ \alpha e^x - \alpha & if\ x \leq 0 \end{cases}$$

The loss function used to train the network is the Mean Square Error.

The neural network goal is to determine the probability that an entity candidate is right. In fact for each sample, I get an output value that corresponds to this probability.

$p_{x,j}$ corresponds to the output value of the input sample associated to the candidate entity j for token x and P corresponds to the output value list. More candidates C_x are associate to each token x. I select the candidate associated with the highest value $p_{x,max}$ among all output values $\{p_{x,1}, p_{x,2}, ..., p_{x,|C_x|}\}$.

Defining a threshold τ_d, if $p_{x,max} > \tau_d$, I select as predicted entity for token x the one related to $p_{x,max}$; otherwise I set that the token x does not correspond to a named entity. I can identify this selection method with the name **candidate selection**. It returns the list of predicted entities identifiers z_p

[3] A *dense layer* is a layer fully connected to the previous one.

6.2.2 Evaluation

From the disambiguation ensemble algorithm description presented in the previous section, I can distinguish two parts: the neural network **ENND**, whose goal is to understand if the entities candidates are right, and the **candidate selection** that uses the ENND output to set the final predicted entity.

For this reason I decided to perform two different kinds of evaluations: the first one evaluates if the NN is able to identify the wrong and the right candidates, the second one evaluates instead the final output of the disambiguation ensemble method.

For the former I consider the neural network target list Y_d and the predicted values list P. Y_d values are boolean, instead P values are continuous values included between 0 and 1. Rounding P, I get P_{round}.

Using this list I compute two different metrics:

1. the **categorical cross entropy** between two probability distributions. It measures the average number of bits needed to identify an event from a set of possibilities, if a coding scheme is used based on a given probability distribution P, rather than the "true" distribution Y_d. It follows the formula:

$$H_{ENND}(Y_d, P) = -\sum_k Y_d(k) \log(P(k))$$

2. the F1 score between Y_d and P_{round}, according to the formula:

$$F1_{ENND}(Y_d, P_{round})$$

To evaluate the final output of the disambiguation ensemble method, I use a *token based* approach. Considering a generic gold standard GD, I can derive a list of entities identifiers z_t (target), as long as the number of token $|X|$. Considering the two lists of tokens z_t (target) and z_p (predicted), I can compute F1, precision and recall scores. I respectively identify them with $F1(z_t, z_p)$, $precision(z_t, z_p)$, $recall(z_t, z_p)$. To have an idea about how z_t and z_p are formed, have a look at the following example:

$$z = [NAN, Q21, Q21, ..., NAN]$$

`Q21` is the Wikidata identifier of a specific entity, `NAN` means that no entity was predicted for the corresponding token.

To compute precision, recall and F1, I define the true positives, the true negatives, the false positives and the false negatives as shown in Algorithm 1.

From z_t I derive z_{pt} (pseudo-target): for a token x, $z_{pt}(x) = z_t(x)$ if at least an extractor $e \in U$ returned the correct entity identifier, otherwise $z_{pt}(x) = NAN$. Using $z_{pt}(x)$ I can compute $F1(z_{pt}, z_p)$, $precision(z_{pt}, z_p)$ and $recall(z_{pt}, z_p)$. These

Algorithm 1 Compute true positives, true negatives, false positives and false negatives

```
1: procedure GET_TP_TN_FP_FN(target_identifiers_list, predicted_identifiers_list number_of_tokens)
2:     true_positive ← 0
3:     false_positive ← 0
4:     true_negative ← 0
5:     false_negative ← 0
6:     i ← 0
7:     while i < number_of_tokens do
8:         target_id ← target_identifiers_list[i]
9:         predicted_id ← target_identifiers_list[i]
10:        if target_id = predicted_id = NAN then
11:            true_negative ← true_negative + 1
12:        end if
13:        if target_id = predicted_id ≠ NAN then
14:            true_positive ← true_positive + 1
15:        end if
16:        if target_id ≠ predicted_id then
17:            if predicted_id = NAN then
18:                false_negative ← false_negative + 1
19:            else if target_id = NAN then
20:                false_positive ← false_positive + 1
21:            else
22:                false_negative ← false_negative + 1
23:                false_positive ← false_positive + 1
24:            end if
25:        end if
26:        i = i + 1
27:    end while
28:    Return true_positive,true_negatives,false_positive,false_positive
29: end procedure
```

scores have the advantage that they consider the fact that my ensemble method is not able to predict entities that no one extractor predicts. This means that $F1(z_{pt}, z_p) = 1$ is always reachable, instead $F1(z_t, z_p) = 1$ is reachable only when for each token x at least one extractor predicted the right entity.

I tested the disambiguation ensemble method using three different gold standards: OKE2016, AIDA and the French subtitles fragments that I annotated.

In the following, I will report and discuss the obtained result.

OKE2016

OKE2016 has already been described in Section 6.1.2, but now I consider the identifiers associated to each NE rather than the types.

The ENND scores got for the test set are here reported:

$$H_{ENND}(Y_d, P) = 0.92$$

$$F1_{ENND}(Y_d, P_{round}) = 0.94$$

The *token based* scores are reported in the following tables: in particular in Table 6.12 are reported $F1(z_t, z_p)$, $precision(z_t, z_p)$, $recall(z_t, z_p)$ scores for each single extractor and for the ensemble.

In Table 6.13 the scores are computed respect to the pseudo target: $F1(z_{pt}, z_p)$, $precision(z_{pt}, z_p)$ and $recall(z_{pt}, z_p)$.

Table 6.12: Token based scores (OKE2016) for target

	F1	**precision**	**recall**
babelfy	0,52	0,46	0,62
dandelion	0,52	0,45	0,62
dbspotlight	0,39	0,33	0,49
textrazor	0,67	0,67	0,67
ensemble	0,76	0,89	0,66

Table 6.13: Token based scores (OKE2016) for pseudo target

	F1	**precision**	**recall**
babelfy	0,57	0,46	0,74
dandelion	0,56	0,45	0,75
dbspotlight	0,42	0,33	0,59
textrazor	0,73	0,67	0,8
ensemble	0,84	0,89	0,79

The ensemble method outperforms the single extractors in terms of F1 score, improving it by 12% (from 0.67 to 0.76); this because the ensemble method really increases the precision, from 0.67 (the highest precision of the single extractors) to 0.89. Looking at the recall, TextRazor returned the highest value; for this reason, I can conclude that the ensemble model for OKE2016 improves the TextRazor response by using the responses of the other extractors to filter out the wrong predicted entities.

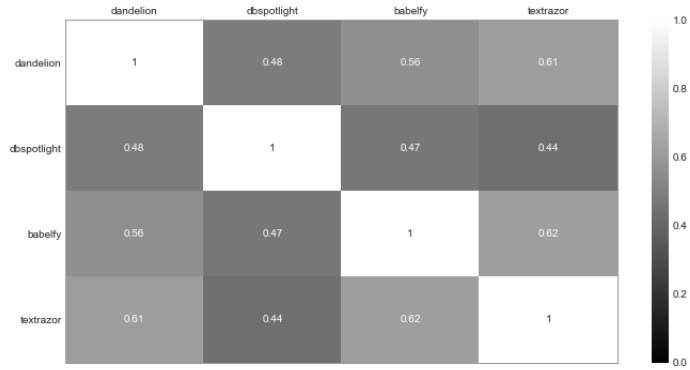
I define the similarity between the identifiers lists,$z_p^{e_1}$ and $z_p^{e_2}$, respectively returned by extractors e_1 and e_2, as:

$$sim(e_1, e_2) = F1(z_p^{e_1}, z_p^{e_2})$$

I computed the similarities between all possible extractors responses pairs in Figure 6.9.

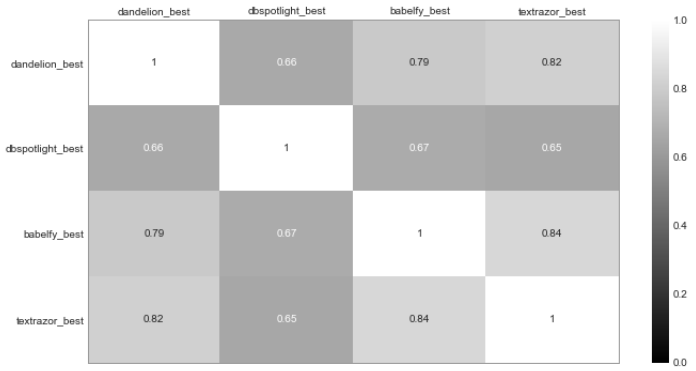
Instead, Figure 6.10 shows the similarities between the extractors responses filtering out by the responses the wrong predictions. In other words, it shows how much two extractors are similar when they predict the right entities: if the similarity is high, they generally identify the same entities, hence to combine their responses in an ensemble method will not be really convenient for an improvement in prediction.

Figure 6.9: Similarity extractors on OKE2016



Looking at both similarities figures, I can deduce that some extractors are really similar in terms of retrieved entities. For instance, the Figure 6.10 show an high similarity value between TextRazor and Babelfy (0.79) and TextRazor and Dandelion (0.82). Knowing that TextRazor is the best single extractor in terms of F1 score for OKE2016, improving its output response recall is difficult because the other extractors are similar in terms of retrieved entities.

Figure 6.10: Similarity extractors on OKE2016 (right predicted)



The last research task investigates how much the type features contribute to improve the final prediction. To understand this, I run ENND removing the I_T input layers and considering only the I_U input layers. It means that $R = I_U$. Applying these changes, the $H_{ENND}(Y_d, P)$ cross-entropy score decreased from 0.92 to 0.88 and therefore $F1(z_t, z_p)$ score decreased from 0.76 to 0.67. Hence I can establish that the types features have a 12% contribution on $F1(z_t, z_p)$. In practice, using only the entity features, the ensemble method does not outperform TextRazor. I can conclude that types are really useful also in terms of disambiguation and some extractors are probably reaching greater results on some specific types.

French Debates Subtitles

French Debates Subtitles corresponds to the corpus annotated by myself. This corpus allows to test the disambiguation ensemble method also on the French language.

The ENND scores got for the test set are here reported:

$$H_{ENND}(Y_d, P) = 0.98$$

$$F1_{ENND}(Y_d, P_{round}) = 0.98$$

The **token based** scores are reported in the following tables: in particular in Table 6.14 are reported $F1(z_t, z_p)$, $precision(z_t, z_p)$, $recall(z_t, z_p)$ scores for each single extractor and for the ensemble. In Table 6.15 the scores are computed respect to the pseudo target: $F1(z_{pt}, z_p)$, $precision(z_{pt}, z_p)$ and $recall(z_{pt}, z_p)$.

Also for French debates corpus, the ensemble method outperforms the single extractors in terms of F1 score, improving it by 25% (from 0.66 to 0.8); this

Table 6.14: Token based scores (French subtitles) for target

	F1	precision	recall
babelfy	0,28	0,23	0,35
dandelion	0,18	0,18	0,17
dbspotlight	0,2	0,14	0,36
textrazor	0,66	0,62	0,7
ensemble	0,8	0,97	0,68

Table 6.15: Token based scores (French subtitles) for pseudo target

	F1	precision	recall
babelfy	0,31	0,23	0,47
dandelion	0,2	0,18	0,23
dbspotlight	0,22	0,14	0,48
textrazor	0,74	0,62	0,93
ensemble	0,94	0,97	0,9

improvement is bigger than the OKE2016 one (12%). The reason is that the ensemble method really increases the precision, from 0.62 (the highest precision of the single extractors) to 0.97. Looking at the recall, TextRazor returned the highest value. The conclusion is similar to the previous case: the ensemble model improves TextRazor response by using the responses of the other extractors to filter out the wrong predicted entities.

As for OKE2016, I report in Figure 6.11 and 6.12 the extractors similarities. In this case, only Babelfy and Dandelion responses seem similar (0.81 in 6.12). They probably reinforces the TextTazor response to avoid false positive prediction.

To sum up, I run ENND without the types features to monitor if these features are also significant in this case. Once again, I note that both $H_{ENND}(Y_d, P)$ and $F1(z_t, z_p)$ decrease, respectively from 0.98 to 0.95 and from 0.8 to 0.76. Hence the types features contributes by the 5%. This improvement is lower than the one got for OKE2016, but it still persists.

AIDA

Last gold standard I used to test the ensemble method is the English AIDA dataset. In this case, I could not use the DBspotlight response because the service was not available during my test.

The ENND scores got for the test set are here reported:

$$H_{ENND}(Y_d, P) = 0.95$$

Figure 6.11: Similarity extractors on French debates

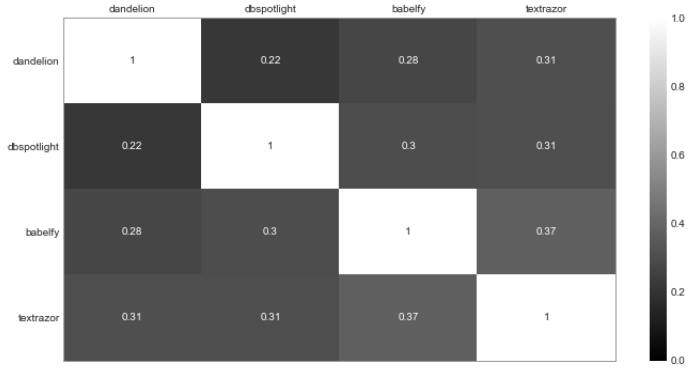
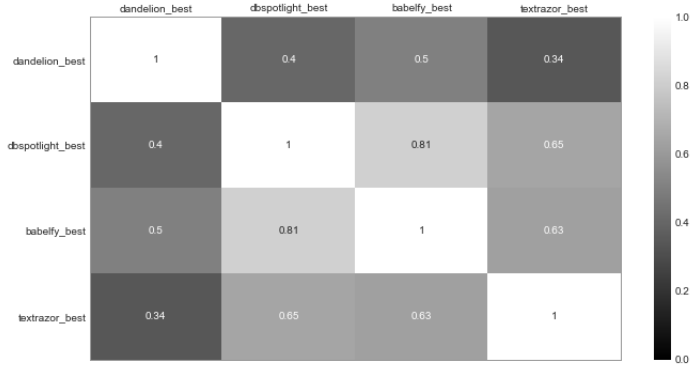


Figure 6.12: Similarity extractors on French debates (right predicted)



$$F1_{ENND}(Y_d, P_{round}) = 0.97$$

The **token based** scores are reported in the following tables: in Table 6.16 are reported $F1(z_t, z_p)$, $precision(z_t, z_p)$, $recall(z_t, z_p)$ scores for each single extractor and for the ensemble. In Table 6.17 the scores are computed respect to the pseudo target: $F1(z_{pt}, z_p)$, $precision(z_{pt}, z_p)$ and $recall(z_{pt}, z_p)$.

The ensemble method outperforms the single extractors in terms of F1 score, improving it by 18% (from 0.6 to 0.73); this improvement is bigger than the one I estimated on OKE2016 (12%) but lower to the one I estimated on French subtitles corpus (25%); also in this case the ensemble method increases the precision, from 0.5 (the highest precision of the single extractors) to 0.8, while the recall falls down (from 0.79 to 0.88). As for the previous gold standards, the ensemble model

Table 6.16: Token based scores (AIDA) for target

	F1	precision	recall
babelfy	0,37	0,27	0,59
dandelion	0,4	0,29	0,66
textrazor	0,6	0,5	0,75
ensemble	0,73	0,8	0,68

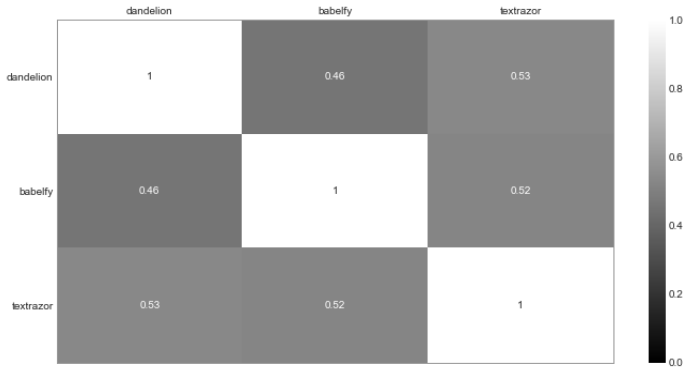
Table 6.17: Token based scores (AIDA) for pseudo target

	F1	precision	recall
babelfy	0,38	0,27	0,69
dandelion	0,42	0,29	0,77
textrazor	0,64	0,5	0,88
ensemble	0,8	0,8	0,79

improves the TextRazor response by using the responses of the other extractors to filter out the wrong predicted entities.

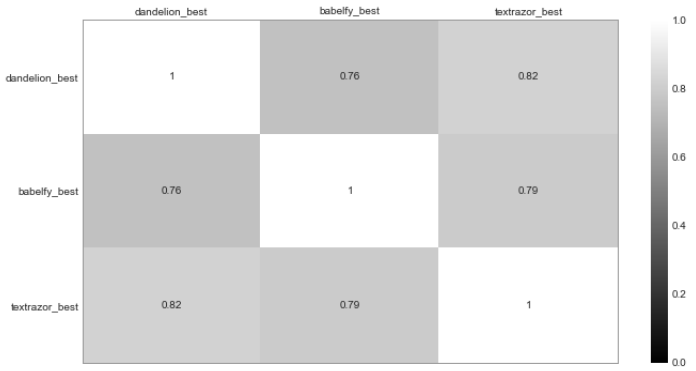
I report in Figure 6.13 and 6.14 the extractors similarities. In this case, all the extractors responses seem similar.

Figure 6.13: Similarity extractors on AIDA



To conclude, I run ENND without the types features to monitor if these features are also significant in this case. I note that both $H_{ENND}(Y_d, P)$ and $F1(z_t, z_p)$ decrease, respectively from 0.95 to 0.92 and from 0.73 to 0.6. Hence the types features contributes by the 18%.

Figure 6.14: Similarity extractors on AIDA (right predicted)



6.2.3 Summary

By looking at the F1 **token based** scores got by the previous evaluations, I can conclude that the ensemble method improves the extractors response and it generally happens because the precision increases.

In addition, I showed as for all considered ground truth, the role of type features is crucial because its absence makes null the improvement of the ensemble method respect to a single extractor prediction.

Chapter 7

Conclusions and future work

In this master thesis I presented two multilingual ensemble methods which combine the responses of web services (extractors) performing Named Entity Recognition and Disambiguation.

The first ensemble method is based on a neural network called **Ensemble Neural Network for Type Recognition (ENNTP)** and it outperforms the single extractors scores for NER. It allows to avoid the manually type alignment between the type ontologies of each extractor and the ground truth ontology. In particular, a particular block of the network – the **alignment block** – performs this task.

The second ensemble method is based on a neural network called **Ensemble Neural Network for Disambiguation (ENND)** and it outperforms the single extractors scores for NED; Wikidata is the knowledge base containing the entities to which entity mentions can be linked.

I demonstrated that the features generation is crucial for the success of these ensemble methods. In particular I presented four different kinds of features: **surface features** related to a specific text, **type**, **entity**, **score features** related to the extractor outputs. In addition I showed how much each of these features contributes to the scores improvements.

Despite having reached the goal to outperform the single extractors, some steps forward could be done as future work:

1. in Chapter 2.1, I described the extractors that I am currently using for my ensemble methods. However, this list of extractors is extensible and should be updated to add the new services presented on the Web; for instance, *spaCy*;

2. in Chapter 3, I explained the annotation criteria I used to annotate 77 fragments extrapolated by the debates transcripts corpus. Using the same criteria, other fragments could be annotated in order to increase the number of

training data;

3. in Chapter 4, I focused on finding a way to embed Wikidata knowledge base. I solved the problem for a graph composed by around 300000 nodes but I did not use this embedding method due to scalability problems with big graphs. The possibility of running the algorithm in a distributed way should be explored;

4. in Chapter 5, I described the different sources of information I used to create features. A step further could be adding Part of Speech tags features to each token presented in anaayzed corpus;

5. in Chapter 6.1, I presented the architecture of ENNTR and I showed as the input layer that receive the surface features is connected to a dense layer. This part of the network could be replaced by a BiLSTM that takes in consideration the token context. I start performing an experiment in this direction but it failed in improving the result, mainly because of the increased (and not fulfilled) training time required by the network. A further step could be to understand why it happens and to try different word embedding methods rather than the Fasttext one.

Bibliography

[1] R. A. Amsler. Research toward the development of a lexical knowledge base for natural language processing. In *Proceedings of the 12th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '89, pages 242–249, New York, NY, USA, 1989. ACM.

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[3] Christian Bizer, Julius Volz, Georgi Kobilarov, and Martin Gaedke. Silk - a link discovery framework for the web of data. In *18th International World Wide Web Conference*, April 2009.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[5] Sam Coates-Stephens. The analysis and acquisition of proper names for the understanding of free text. *Computers and the Humanities*, 26(5):441–456, Dec 1992.

[6] James R. Curran and Stephen Clark. Language independent ner using a maximum entropy tagger. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 164–167, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[7] Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N. Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th International Conference on Semantic Systems (I-Semantics)*, 2013.

[8] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 363–370, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[9] José Luis Redondo García, Giuseppe Rizzo, and Raphaël Troncy. Capturing

news stories once, retelling a thousand ways. In *Proceedings of the 8th International Conference on Knowledge Capture*, K-CAP 2015, pages 34:1–34:4, New York, NY, USA, 2015. ACM.

[10] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

[11] Avraham A Levy and Michael Lindenbaum. Sequential karhunen-loeve basis extraction and its application to images. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 9 8:1371–4, 1998.

[12] Hailun Lin, Yong Liu, Weiping Wang, Yinliang Yue, and Zheng Lin. Learning entity and relation embeddings for knowledge resolution. *Procedia Computer Science*, 108(Supplement C):345 – 354, 2017. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

[13] Weiming Lu, Yangfan Zhou, Haijiao Lu, Pengkun Ma, Zhenyu Zhang, and Baogang Wei. Boosting collective entity linking via type-guided semantic embedding. In *Natural Language Processing and Chinese Computing: 6th CCF International Conference, NLPCC 2017, Dalian, China, November 8–12, 2017, Proceedings*, pages 541–553, Cham, 2018. Springer International Publishing.

[14] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[15] Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. Dbpedia spotlight: Shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems*, I-Semantics '11, pages 1–8, New York, NY, USA, 2011. ACM.

[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[17] David Milne and Ian H. Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 509–518, New York, NY, USA, 2008. ACM.

[18] Andrea Moro, Alessandro Raganato, and Roberto Navigli. Entity linking meets word sense disambiguation: a unified approach. *TACL*, 2:231–244, 2014.

[19] F. Å. Nielsen. Wembedder: Wikidata entity embedding web service, 2017.

[20] Enrico Palumbo, Giuseppe Rizzo, and Raphaël Troncy. entity2rec: Learning user-item relatedness from knowledge graphs for top-N item recommendation. In *RECSYS 2017, 11th ACM Conference on Recommender Systems, August 27-31, 2017, Como, Italy*, Como, ITALY, 08 2017.

[21] Stefano Parmesan, Ugo Scaiella, Michele Barbera, and Tatiana Tarasova. Dandelion: From raw data to datagems for developers. In *Proceedings of the 2014 International Conference on Developers - Volume 1268*, ISWC-DEV'14, pages 1–6, Aachen, Germany, Germany, 2014. CEUR-WS.org.

[22] Julien Plu, Giuseppe Rizzo, and Raphaël Troncy. Enhancing entity linking by combining models. In *ESWC 2016, 13th European Semantic Web Conference (ESWC'16), Open Extraction Challenge, May 29-June 2, 2016, Heraklion, Greece*, Heraklion, GREECE, 05 2016.

[23] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, CoNLL '09, pages 147–155, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[24] José Luis Redondo García, Giuseppe Rizzo, Lilia Perez Romero, Michiel Hildebrand, and Raphaël Troncy. Generating semantic snapshots of newscasts using entity expansion. In *Engineering the Web in the Big Data Era: 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015, Proceedings*, pages 410–419, Cham, 2015. Springer International Publishing.

[25] José Luis Redondo García, Giuseppe Rizzo, and Raphaël Troncy. The concentric nature of news semantic snapshots: Knowledge extraction for semantic annotation of news items. In *Proceedings of the 8th International Conference on Knowledge Capture*, K-CAP 2015, pages 16:1–16:8, New York, NY, USA, 2015. ACM.

[26] Giuseppe Rizzo. Learning with the web: Spotting named entities on the intersection of nerd and machine learning, 01 2013.

[27] Giuseppe Rizzo and Raphaël Troncy. Nerd: A framework for unifying named entity recognition and disambiguation extraction tools. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, pages 73–76, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[28] Giuseppe Rizzo, Marieke van Erp, and Raphaël Troncy. Benchmarking the extraction and disambiguation of named entities on the semantic web. In *LREC 2014, 9th International Conference on Language Resources and Evaluation, May 26-31, 2014, Reykjavik, Iceland*, Reykjavik, ICELAND, 05 2014.

[29] Giuseppe Rizzo, Marieke van Erp, and Raphaël Troncy. Inductive entity typing alignment. In *ISWC 2014, 1st International Workshop on Linked Data for Information Extraction (LD4IE 2014), October 20, 2014, Riva del Garda, Italy*, Riva del Garda, ITALY, 10 2014.

[30] David A. Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *Int. J. Comput. Vision*, 77(1-3):125–141, May 2008.

[31] Wei Shen, Jianyong Wang, and Jiawei Han. Entity linking with a knowledge

base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460, 2015.

[32] René Speck and Axel-Cyrille Ngonga Ngomo. Ensemble learning for named entity recognition. In *The Semantic Web – ISWC 2014*, volume 8796 of *Lecture Notes in Computer Science*, pages 519–534. Springer International Publishing, 2014.

[33] René Speck and Axel-Cyrille Ngonga Ngomo. *Ensemble Learning for Named Entity Recognition*, pages 519–534. Springer International Publishing, Cham, 2014.

[34] René Speck and Axel-Cyrille Ngonga Ngomo. Ensemble Learning of Named Entity Recognition Algorithms using Multilayer Perceptron for the Multilingual Web of Data. In *K-CAP 2017: Knowledge Capture Conference*, page 4. ACM, 2017.

[35] Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. brat: a web-based tool for NLP-assisted text annotation. In *Proceedings of the Demonstrations Session at EACL 2012*, Avignon, France, April 2012. Association for Computational Linguistics.

[36] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. 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, pages 173–180, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[37] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 353–362, New York, NY, USA, 2016. ACM.

[38] Wei Zhang, Jian Su, Chew Lim, Tan Wen, and Ting Wang. Entity linking leveraging automatically generated annotation. In *In Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010*, pages 1290–1298, 2010.

[39] GuoDong Zhou and Jian Su. 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, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.