FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Dynamic Detection of Ambiguities and Contradictions in Job Posting

Catarina Pinheiro Correia

DISSERTATION



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Henrique Lopes Cardoso

Second Supervisor: Gil Rocha

Skeeled's Supervisor: Joana Urbano

July 19, 2019

Dynamic Detection of Ambiguities and Contradictions in Job Posting

Catarina Pinheiro Correia

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. Daniel Augusto Gama de Castro Silva
External Examiner: Diego Nascimento Esteves da Silva
Supervisor: Prof. Henrique Daniel de Avelar Lopes Cardoso

July 19, 2019

Abstract

In job recruitment, finding the right candidate for a given job position is no easy task. Although recruiting experts often perform this task, the number of tools using machine learning (ML) to solve this problem is on the rise.

skeeled is one company that makes use of Artificial Intelligence (AI) technology to provide software to centralize and digitize the recruitment process. The company analyzes a large amount of data in order to match candidates to jobs. Analyzing job postings involves understanding their content. Since natural language is inherently ambiguous, this is a complex task for computers. Besides, these job postings may contain contradictions that may lead to an imprecise matching between candidates and job postings.

The analysis of the mentioned data is a relatively intuitive task for human beings. Nevertheless, the manual evaluation of job postings is time-consuming and error-prone, and not only it depends on the reviewer's knowledge and his or her interpretation, but also there can be a mismatch between the analysis of two different reviewers. Thus, the purpose of this dissertation is to develop a module capable of detecting ambiguities and contradictions, in several languages, when a recruiter creates a job posting and alert the recruiter to such irregularities before the publication of the job ad.

Our proposed approach involves the manual annotation of contradictions and ambiguities present in *skeeled*'s job postings, the usage of natural language processing techniques and the creation of rule-based models to detect such inconsistencies.

Resumo

Em recrutamento, encontrar o candidato ideal para uma determinada proposta de emprego é uma tarefa complexa. Apesar de esta tarefa ser frequentemente executada por peritos, o número de ferramentas baseadas em *machine learning* tem vindo a aumentar.

A *skeeled* é uma empresa que usa tecnologias de Inteligência Artificial para disponibilizar software que centraliza e digitaliza o processo de recrutamento. Para atribuir candidatos a propostas de emprego, a empresa analisa um vasto número de dados. Analisar propostas de empregos implica compreender os seus conteúdos. Esta tarefa é complexa para computadores, já que linguagem natural é inerentemente ambígua. Além disso, as propostas de emprego podem conter contradições que levam a uma correspondência imprecisa entre candidatos e ofertas.

A análise destes dados é uma tarefa relativamente intuitiva para o ser humano. Contudo, para além de esta avaliação manual ser demorada, também está sujeita ao conhecimento de cada avaliador e à interpretação de cada um, podendo duas pessoas diferentes detetar falhas diferentes. Como tal, o objetivo desta dissertação é desenvolver um módulo capaz de detetar ambiguidades e contradições, em várias línguas, quando um recrutador cria uma proposta de emprego, e alterar o recrutador para tais irregularidades antes que este publique a oferta.

A nossa abordagem consiste na anotação manual de contradições e ambiguidades existentes nas propostas de emprego da *skeeled*, na utilização de técnicas de processamento de linguagem natural e na criação de modelos baseados em regras que detetem tais inconsistências.

Acknowledgements

I would like to thank my supervisors, Professor Henrique Lopes Cardoso and Gil Rocha, for their guidance throughout the development of my dissertation.

I would also like to thank Joana Urbano, for all the support and guidance throughout my journey in *skeeled* and for helping with the annotations.

I would also like to extend my gratitude to Adeline Herbey, António Gomes, Carolina Costa, Josephine Schonberger, Michael Wiech and Penelope Figueira for their help during the annotations phase.

I would also like to show my gratitude to *skeeled* for all the help and welcoming environment they have provided me. Finally, I would like to thank my family for their support. I would not have been able to do this without your support.

Catarina Pinheiro Correia

"Those who have learned to walk on the threshold of the unknown worlds, by means of what are commonly termed par excellence the exact sciences, may then, with the fair white wings of imagination, hope to soar further into the unexplored amidst which we live."

Ada Lovelace

Contents

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Goals . . . 3
- 1.3 Contributions . . . 3
- 1.4 Structure of the Dissertation . . . 4

2 Background **5**
- 2.1 Ambiguities . . . 5
- 2.2 Contradictions . . . 8
- 2.3 Natural Language Processing . . . 8
  - 2.3.1 The stages of natural language analysis . . . 9
- 2.4 Machine Learning applied to NLP . . . 14
  - 2.4.1 Feature Extraction . . . 15
  - 2.4.2 Training a model . . . 15
  - 2.4.3 Training, Test and Validation Sets . . . 16
  - 2.4.4 Evaluation of the Model . . . 17
  - 2.4.5 Text classification . . . 18
- 2.5 Inter-Annotator Agreement . . . 21
  - 2.5.1 A Common Notation . . . 22
  - 2.5.2 Metrics . . . 22
- 2.6 Summary . . . 24

3 Related Work **25**
- 3.1 Contradiction Detection . . . 25
- 3.2 Ambiguity Detection . . . 29
- 3.3 Summary . . . 31

4 Dataset **33**
- 4.1 Dataset Description . . . 33
- 4.2 Contradictions and ambiguities areas . . . 34
- 4.3 Ambiguities . . . 35
  - 4.3.1 Types . . . 35
  - 4.3.2 Examples . . . 36
- 4.4 Contradictions . . . 37
  - 4.4.1 Types . . . 37
  - 4.4.2 Examples . . . 37
- 4.5 Annotating the Dataset . . . 40
  - 4.5.1 Annotation tool . . . 40

4.5.2 Annotation Statistics . . . 42
4.6 Inter-Annotator Agreement . . . 43
4.7 Summary . . . 45

5 Detecting Contradictions **47**
5.1 Overall method . . . 47
5.2 Preprocessing . . . 48
5.3 Languages Model . . . 52
5.3.1 Identification of the language-related sentences . . . 53
5.3.2 Second preprocessing . . . 55
5.3.3 Extracting modifiers . . . 56
5.3.4 Extracting the mandatory and optional languages . . . 58
5.3.5 Rules for detecting contradictions . . . 59
5.3.6 Results . . . 63
5.3.7 Error analysis . . . 66
5.3.8 Summary . . . 67
5.4 Experience Model . . . 67
5.4.1 Identification of the experience-related sentences . . . 68
5.4.2 Preprocessing & Extracting modifiers . . . 68
5.4.3 Rules for detecting contradictions . . . 69
5.4.4 Results . . . 70
5.4.5 Error Analysis . . . 71
5.4.6 Summary . . . 72

6 Conclusion and Future Work **73**
6.1 Future Work . . . 74

References **75**

A Dataset **79**
A.1 Dataset Fields . . . 79
A.1.1 **Structured Fields** . . . 79
A.1.2 Unstructured Fields . . . 84
A.2 Job Posting . . . 85

B Annotation Manual **87**

List of Figures

2.1 The stages of analysis in processing natural language . . . 9
2.2 Constituency parsing tree for the sentence *Good knowledge of English and basic French* by *StanfordCoreNLP* . . . 12
2.3 Two possible syntax trees for the same (ambiguous) sentence. [BLK09] . . . 12
2.4 Basic dependencies for *Good knowledge of English and basic French.* . . . 13
2.5 Enhanced++ dependencies for *Good knowledge of English and basic French.* . . . 13
2.6 Iterative trail-and-error process to train a model [Dev] . . . 16
2.7 Workflow [Dev] . . . 17
2.8 Conventional Supervised Classification [BLK09] . . . 18
2.9 Support Vector Machines . . . 19
2.10 Feed-forward NN with two hidden layers [Gol16] . . . 21

3.1 Architecture of the contradiction detection system [PNS13] . . . 26
3.2 Architecture for contradiction detection [Dra17] . . . 27
3.3 Clustering approach overview [MKS18] . . . 28
3.4 High Level Solution Approach and Research Framework Diagram - LSTM based feature extraction [LBN+18] . . . 28
3.5 High Level Solution Approach and Research Framework Diagram - Contradiction and Neutral Detection in Text using Neural Network and Five Features [LBN+18] 29

4.1 Distribution of job postings by language in the dataset . . . 34
4.2 Example of an annotation in *brat* . . . 41

5.1 Summary of the approach to detect contradictions . . . 49
5.2 Preprocessing . . . 52
5.3 Dependency parsing for the sentence *Good knowledge of English.* . . . 56
5.4 Dependency parsing for the sentence *Good knowledge of English.* . . . 57
5.5 Dependency parsing for the sentence *Good knowledge of English and basic level of French.* . . . 57
5.6 Dependency parsing for the sentence *Good knowledge of English and basic level of French.* after adding the missing punctuation . . . 57
5.7 Dependency parsing for the sentence *Good knowledge of English and basic level of French.* after adding the verb *be* . . . 57
5.8 Parse tree for *Good knowledge of English or French and Portuguese.* . . . 60
5.9 Dependency parser for *Our Programme is dedicated for people who have a minimum of 2 years of experience.* . . . 69
5.10 Dependency parser for *Our Programme is a 6 month learning programme designed for students at any level (BSc, MSc, MBA and PhD) with limited working experience.* . . . 69

LIST OF FIGURES

List of Tables

2.1 Examples of contradiction types adapted from Marneffe et al. [MRM08] 8
2.2 Some of the tools used for segmentation tasks 10
2.3 Some of the tools used for lexical analysis 10
2.4 Some of the tools used for syntactic analysis 14

4.1 Attribution of language levels according to different expressions 38
4.2 Language levels according to the *Common European Framework of Reference for Languages* 39
4.3 Description of the fields of an annotation 41
4.4 Summary of the distribution of annotators and annotations per language 42
4.5 Inter-annotator agreement values obtained using Krippendorff's alpha 43
4.6 Summary of the distribution of contradictions by type and by language in the Gold Standard 44
4.7 Summary of the distribution of ambiguities by type and by language in the Gold Standard 45

5.1 HTML Entities and respective descriptions and replacement expressions. 50
5.2 Language levels and respective replacement expressions 51
5.3 Example of some of the words we use to describe the language levels in English and French 61
5.4 Number of total sentences identified versus the number of language-related sentences 64
5.5 Confusion matrix for English (binary classification problem) 64
5.6 Confusion matrix for French (binary) 65
5.7 Metrics obtained for our languages model in English and French (binary classification) 65
5.8 Metrics obtained for baselines models in English (binary classification) 65
5.9 Confusion matrix for English (multiclass classification problem) 66
5.10 Metrics obtained for our language models in English (multiclass classification problem) 66
5.11 Number of total sentences identified versus the number of experience-related sentences 70
5.12 Confusion matrix for experience model in English (binary) 71
5.13 Confusion matrix for experience model in French (binary classification) 71
5.14 Metrics obtained for our experience model in English and French (binary) 71

Abbreviations

AI	Artificial Intelligence
BoW	Bag-of-words
GloVe	Global Vectors for Word Representation
IAA	Inter-Annotator Agreement
LSTM	Long Short-term Memory
ML	Machine Learning
NLP	Natural Language Processing
NN	Neural Networks
POS	Part-of-speech
RNN	Recurrent Neural Networks
RTE	Recognizing Textual Entailment
SF	Structured Fields
SRL	Semantic Role
UF	Unstructured Fields
WSD	Word Sense Disambiguation
WK	World Knowledge

Chapter 1

Introduction

Traditionally, the recruitment process is long and time-consuming. It starts by determining the company's hiring needs, publishing a job position, searching for potential candidates, and choosing the most suitable candidates for the job. However, choosing the best candidates is not as trivial as it might seem. Such a decision requires a deep analysis of not only the candidate's grades or years of experience but also of the candidate's motivation, personality profile and even the cultural fit, i.e., the ability the candidate has to adapt to the values and beliefs that define the company's culture. To do so, the recruiting company needs to analyze the candidate's CV, followed by a series of interviews that aim to evaluate the candidate's competencies, soft skills and whether the candidate fits the company's needs or not. At last, after the selection of the right candidate, the company hires the chosen candidate, if the candidate decides to accept the job offer.

However, with the development of technology and new tools and frameworks, recruitment processes have been undergoing a revolution in the past few years. The emergence of professional networks and other online platforms have been helping job seekers and employers to establish a connection and speeding up the whole process of searching for employees. Nevertheless, some companies take a step further and make use of Machine Learning (ML) tools and algorithms to expedite the whole recruitment process.

skeeled is a company whose purpose is to enhance the experience for both candidates and recruiters by providing an all-in-one hiring software that digitizes the recruitment process. It offers an innovating technology that automates and speeds up the first steps of the recruitment process, saving both recruiters and candidates a large amount of time and facilitating collaborative hiring.

Basically, when a recruiter posts a job position in *skeeled*'s application, *skeeled* makes use of Artificial Intelligence (AI) methods and powerful ML algorithms to analyze the job, determine the job's needs and search for suitable candidates among the platform. Then, the application rates the candidates and provides a list of candidates ordered by relevance. This rating takes into account

the candidate's profile that is built based on three factors: the applicant's CV, the candidate's personality test and the applicant's video interview.

Using Artificial Intelligence in order to select the most suitable candidates allows the analysis of more potential candidates in less time and it improves the quality and accuracy of the selection process, as the algorithms are not influenced by the same conditions humans are, such as tiredness and the favoring of some candidates over others.

1.1 Motivation

The process of matching a candidate to a given job position is complicated. On the one hand, *skeeled* must analyze the job posting to see what the recruiter is looking for. On the other hand, the company needs to analyze several candidates' profiles to see how they fit the job's requirements.

Determining a job posting's requirements demands a thorough analysis of its fields. In *skeeled*'s platform, creating a job posting consists of filling several fields. These fields can be divided into two classes: structured fields (SF) and unstructured fields (UF). In the structured fields, recruiters can choose from a limited range of options, while in the unstructured fields, the employer has the freedom to write freely, as these are open-text fields.

The complexity of the analysis of such fields is uneven, as examining the UF implies analyzing natural language and, subsequently, understanding the content of these fields. Comprehending natural language is a challenging task not only for computers but also for humans, as natural language is inherently ambiguous. Despite this, each recruiter has a unique writing style and is susceptible to making mistakes, which can lead to incoherent and error-prone job postings that contain language errors, contradictions and/or ambiguities.

The presence of such defects degrades the quality of the job postings, which may lead to an imprecise matching between job postings and candidates. Because of that, it is crucial that these job postings are reviewed not only to guarantee that the incoherence of a job posting is reduced but also to ensure that its quality is improved. In other words, the presence of ambiguities and contradictions should be nonexistent or minimal. However, the task of reviewing a job posting is not easy and has several implications when applied manually, especially if the reviewers lack proper guidelines and training. Some of the difficulties that can arise are as follows:

- The reviewers must have linguistic and technical knowledge;
- Two different reviewers can produce different analysis for the same job posting, especially if clear instructions are not provided;
- Reviewing is a time-consuming task;
- This process is error-prone as the reviewers are influenced by different factors, such as fatigue.

Due to these reasons, it is of best interest to automate this process. Some of the advantages were already referred to above. For once, the computer can analyze more and in a faster way, it is

not influenced by fatigue and it produces the same results for the same job posting regardless of the circumstances.

1.2 Goals

The purpose of this dissertation is to develop a module capable of detecting contradictions, in different languages, with high accuracy and precision values before the recruiter publishes the job offer. Thus, we expect our module to produce significant enhancements in the *skeeled*'s job matching algorithm when integrated with *skeeled*'s system.

Our research work intends to answer the following questions:

1. Can we reliably annotate contradictions and ambiguities in a corpus of job postings?
    - To do so, we will annotate the corpus, and use metrics such as Krippendorff's alpha coefficient to calculate the agreement between the different annotators.
2. What are the most common inconsistencies found in job postings?
    - After annotating the dataset and calculating its reliability, we will analyze the annotated dataset and count the types and subtypes of inconsistencies that occur.
3. Are models based on lexical and syntactic rules suitable for the detection of contradictions in job postings?
    - This problem will be addressed by comparing the contradictions detected by our models with the contradictions annotated in the dataset. We will use the accuracy, recall, precision, and f-score metrics to determine the performance of our model.

1.3 Contributions

With this project, we intend to contribute to the following:

- **Creation of a dataset**. As far as we are concerned, there are no datasets annotated with ambiguities nor contradictions related to the recruitment area. Therefore, this work provides a dataset composed of job postings annotated with the mentioned inconsistencies in two different languages: English and French;
- **Present an analysis of the most common types of inconsistencies in job postings**. There are several ambiguities and contradictions that may jeopardize the quality of job postings. Understanding the main inconsistencies that may occur is of great importance to tackle such a problem;
- **Provide a means of detecting contradictions in job postings**. There are different types of contradictions that can occur in the recruitment area. We analyze the most common types and provide a solution to identify such contradictions and its results.

1.4 Structure of the Dissertation

The present document contains five more chapters. In Chapter 2, some key concepts necessary for the understanding of the subsequent chapters are provided. In Chapter 3, we provide the current state of the art and other related works that provide useful insights for the implementation of our solution. Chapter 4 contains a more detailed analysis of the available data, alongside with the creation and annotation of the dataset. Chapter 5 contains a detailed explanation of the implementation procedure, the tools used and an evaluation of the developed solution. At last, Chapter 6 presents the conclusions withdrawn from the work done and the future work needed to improve the current implementation.

Chapter 2

Background

In this chapter, we explain with further detail what ambiguities and contradictions are, and show the different sources that can cause each alongside with some examples. We also present some challenges that can arise in the field of natural language processing (NLP) and provide an overview of the techniques that have been used to address NLP tasks. Then, we dedicate a section to explain the uses of machine learning (ML) techniques for NLP tasks, some key concepts needed to understand these techniques and we provide a shallow definition of some algorithms that we consider relevant for the understanding of the next chapters.

The problem of detecting contradictions and ambiguities, as well as to propose fixes for both, falls into the domains of artificial intelligence, machine learning, and natural language processing and understanding.

2.1 Ambiguities

When someone uses the term *natural language*, that person refers to the language used by humans on a daily basis to communicate (e.g.: Portuguese, English, French). In contrast, there are artificial languages, such as the so-called programming languages.

One of the many characteristics of natural languages is that they are in constant evolution. As a consequence, it is difficult to pin them down with rules. Besides this, natural languages are inherently ambiguous, which makes their understanding a difficult and complex task even for a human, let alone a computer.

Ambiguity can be defined as a doubtfulness of meaning or intention[1]. For example, if one sentence or expression can have several interpretations, then the meaning of this expression is ambiguous and causes uncertainty. There is a vast number of natural languages, and ambiguities can come in many forms. Furthermore, some ambiguities are language-dependent.

[1] Definition withdrawn from *https://www.dictionary.com/browse/ambiguity*.

According to Shah and Jinwala [SJ15], ambiguities can occur from two different sources: missing information and communication errors. Errors can be *author-independent*, if no domain knowledge is needed, or *author-dependent* if its resolution requires domain knowledge.

Besides ambiguities, the Ambiguity Handbook [BKK] states vagueness and language errors as further sources of language misinterpretation. These sources are the following:

- **Lexical ambiguity** - This ambiguity occurs when a word has multiple meanings. It can be subdivided into two categories: homonymy and polysemy. Two words are **homonyms** if they have the same written and phonetic representation, but unrelated meanings and different etymologies.

  – *E.g.: Bank, newspaper, network, position*

  A word is **polysemous** if it has several related meanings but just one etymology.

  – *E.g.: Man*
  This word has several related meanings: the human species as a whole, one of the genders of the human species, or the adult male of the human species (opposing to the word *boy*)

- **Syntactic ambiguity** - This ambiguity occurs when an expression has more than one possible structural analysis or parse tree, and each has a different meaning. This ambiguity can be distinguished as analytical, attachment or coordination ambiguity.

  Analytical ambiguity arises when a given word can be analyzed as more than one part-of-speech (e.g.: *book* can be either a *noun* or a *verb*).

  – *E.g.: We are looking for an English history expert*
  This expression can be read as *We are looking for an (English history) expert* or *We are looking for an English (history expert)*

  Attachment ambiguity occurs when a certain syntactic constituent of a sentence can be attached to two parts of a sentence.

  – *E.g.: We are looking for someone with a car*
  The expression "with a car" can be either a modifier of the word "someone" or a modifier of the verb "are looking"

  Coordination ambiguity takes place when two or more conjunctions (*and* and *or* are combined in a sentence, or when one conjunction is used with a modifier.

  – *E.g.: We want someone who speaks Portuguese and English or Chinese*
  This expression can be read as either *We want someone who speaks (Portuguese and English) or Chinese* or *We want someone who speaks Portuguese and (English or Chinese)*

- **Semantic ambiguity** - This ambiguity occurs when a sentence has several ways of interpretation within its context although there is no lexical or syntactic ambiguity. This kind of ambiguity can arise from coordination ambiguity (previously discussed), referential ambiguity, and scope ambiguity. For the purpose of this work, scope ambiguity is not relevant, so it will not be discussed.

  Referential ambiguity (or anaphoric ambiguity) is a type of ambiguity that is close to both semantic and pragmatic ambiguity, due to its occurrence within a sentence or between a sentence and its context. It occurs when an *anaphor*[2] can have as antecedent more than one element.

  - *E.g.: The candidates should treat their subordinates respectfully so that they succeed*

    The antecedent for the anaphor *they* can be either *candidates* or *subordinates*.

- **Pragmatic ambiguity** - This type of ambiguity occurs when the meaning of a sentence depends upon the context in which it is uttered. In this case, the context comprises the language context. It can be caused by referential ambiguity or deictic ambiguity. The latter is not important for the present work and shall not be detailed.

  - *E.g.: Can you open the window?*

    This phrase can act both as a request and as a question, depending on the contextual situation.

- **Vagueness** - A sentence is vague when it has a single meaning from a grammatical point of view, but still leaves room for different interpretations. If a requirement is vague, this means that it is not clear how to measure whether the requirement is fulfilled or not.

  - *E.g.: We are looking for someone with **high** knowledge in systems administration*

    There is no way to describe or measure the word "high", leaving room to wonder whether the essence of this requirement has been captured.

- **Language Error** - These errors represent grammatically wrong constructions, such as misplacement of punctuation, word choice or other mistakes, resulting in a misinterpretation of the intended meaning.

  - *E.g.: Every candidate has their bachelor degree*

    Every candidate is singular, but is incorrectly considered as plural. It could have multiple interpretations, such as *Every candidate has its bachelor degree* or *All candidates have their common bachelor degree*.

[2] Element of a sentence that depends upon another element, possibly one that belongs to another sentence

2.2 Contradictions

The logical definition of contradiction defines two sentences as contradictory if there is no possible world in which they could both be true. Marneffe et al. [MRM08] gives us a looser and more intuitive definition: two sentences are contradictory if the probability of both being simultaneously true is almost null. However, two sentences can only be contradictory if they refer to the same event. Given a limited context, two sentences are assumed to be coreferent if there is no clear countervailing evidence that proves otherwise.

Contradictions can be categorized into two major categories, depending on whether they arise from more (1) or less (2) evident features. In case (1), they can occur via antonymy, negation, and numeric mismatch, while in case (2) they can arise from differences in the structure of assertions, use of factive or modal words, discrepancies based on world-knowledge, or structural and lexical contrasts. According to Marneffe et al. [MRM08], automatically detecting contradictions in (1) is an easier task than in (2), because in the former there is no need for a full sentence understanding. Table 2.1 reflects the different types of contradictions.

Table 2.1: Examples of contradiction types adapted from Marneffe et al. [MRM08]

Type	**Sentence 1**	**Sentence 2**
Antonymy	The candidate must be cold-headed and capable of taking difficult decisions	The candidate must be passionate and filled with excitement.
Negation	We need someone with experience	No experience is required for this job.
Numeric	We need someone with more than 5 years of experience	Candidates with 1 year of experience will be considered for the position.
Structure	The candidate will be guided by a colleague in the first weeks.	A colleague will be guided by the candidate in the first weeks.
World-Knwledge	*Skeeled* was founded in 2014	*Skeeled* was founded on 2017
Lexical	The candidate must have a Bachelor's degree in Informatics	The candidate needs a Master's degree in Informatics

2.3 Natural Language Processing

NLP is concerned with the way natural languages and computers interact, and it focuses its efforts on finding ways to enable computers to process and analyze natural language data. Basically, it refers to any type of computer manipulation of natural language.

As natural language is present in our daily lives, NLP-based technologies are becoming of great importance and its usage has been increasing. NLP has a broad range of purposes, such as predict text, speech recognition, search engines, machine translation, natural language generation, ambiguity resolution, information extraction, among others [BLK09].

However, there are many challenges and limitations regarding natural language understanding. One of the biggest limitations, according to Bird et al. [BLK09], is that natural language systems still cannot perform common-sense reasoning or draw on world knowledge in a general and robust

manner. The challenges can vary from determining the sense of a word in a given sentence (**word sense disambiguation**), to identifying the antecedent of a pronoun (**pronoun resolution**), or even checking whether a sentence (hypothesis) is supported by the text (**textual entailment**), among others.

NLP tasks were traditionally carried out by manually constructed rules. According to Khan et al. [KDNA16], NLP tasks are currently performed by statistical techniques and ML methods, which automatically induce rules from training data. Statistical approaches incorporate stochastic and probabilistic methods. The dominant ML techniques to perform NLP tasks fall into the domain of supervised learning. These supervised ML techniques include, among others, Support Vector Machines (SVM) and Naive Bayes (NB), further detailed in Section 2.4.5.

More recently, Goldberg [Gol16] stated that the use of neural-network models using dense inputs has seen some improvements over the usage of linear models that use sparse inputs when applied to NLP techniques.

2.3.1 The stages of natural language analysis

As Clark et al. [CFL13] stated, the process of language analysis can be decomposed into five stages: segmentation, lexical analysis, syntactic analysis, semantic analysis, and pragmatic analysis, as seen in Figure 2.1. The latter determines the meaning of the sentence in context.



Figure 2.1: The stages of analysis in processing natural language

2.3.1.1 Text Preprocessing

Natural language is not always presented in an obvious way, with well-delimited sentences and words that are neatly delimited by spaces. To perform a linguistic analysis, a clear definition of the characters, words, and sentences is necessary. Nevertheless, depending on the language being processed, defining such units can raise different difficulties. Text preprocessing is an important step towards preparing raw data for further processing. The preprocessing step usually involves tasks such as normalization and segmentation.

Normalization Normalizing the text consists on converting text into a canonical form, such as transforming every word to lower case, removing markups, among other operations.

Segmentation According to [BLK09], segmentation can be subdivided into different categories: tokenization, word segmentation, and sentence segmentation. **Tokenization** is the task of breaking the text into basic units - tokens - that can be words or punctuation. **Word segmentation**

consists of dividing the text into words, which can be a difficult task depending on the language, as some languages have no visual representation of word boundaries (e.g.: Chinese). **Sentence segmentation** consists on identifying sentence boundaries thus dividing the text into sentences. Segmentation tasks, such as word segmentation, are needed to perform part-of-speech (POS) tagging, discussed in Section 2.3.1.3.

Table 2.2 shows some of the tools used for segmentation tasks, respective features, the programming language in which they are implemented and if these tools provide support for several natural languages.

Name	**Features**	**Programming Language**	**Multilanguage support**
Natural Language Toolkit (NLTK)	Tokenization, sentence segmentation, word segmentation	Python	Little[1]
StanfordCoreNLP	Tokenization, sentence segmentation, word segmentation	Java	Yes
SpaCy	Tokenization, sentence segmentation, word segmentation	Python	Yes

[1] Only one of the available tokenizer offers multilanguage support.

Table 2.2: Some of the tools used for segmentation tasks

2.3.1.2 Lexical analysis

Lexical analysis is related to the techniques used for performing text analysis at the level of the word, as words themselves are not atomic and are open to further analysis.

Lexical analysis often encompasses tasks such as lemmatization and *stemming*. **Lemmatization** consists on relating *morphological variants* (delivers, delivering, delivered) to their lemma(deliver), while **stemming** is the process of reducing words to their word *stem*/base/root. [CFL13]

Table 2.3 summarizes some of the currently available tools used for lexical analysis, with respective features, implementation programming language and whether there is multilingual support.

Name	**Features**	**Programming Language**	**Multilanguage support**
Natural Language Toolkit (NLTK)	Lemmatization, stemming	Python	Yes
StanfordCoreNLP	Lemmatization, stemming	Java	Yes
SpaCy	Lemmatization, stemming	Python	Yes

Table 2.3: Some of the tools used for lexical analysis

2.3.1.3 Syntactic analysis

When analyzing text, one important step is to determine the structure of its sentences. Formal grammars, such as Context Free Grammars (CFG) [Hop08], can be used to describe language syntax, by subsequently dividing the sentence into smaller parts until all there is left are single words.

According to Jurafsky et al. [JM14], there are two ways of parsing one sentence: by performing a complete parsing or a partial/shallow one. Whereas the full complete parsing attempts to extract all the possible information in the sentence by using CFG, shallow parsing focuses on a less detailed division by using, for example, finite-state automata instead. Although the latter parsing lacks coverage, it gains in efficiency.

Syntactic parsing can be addressed as an intermediate step to achieve the purpose of finding the meaning of a given sentence. It encompasses tasks such as part-of-speech tagging (POS tagging), dependency and constituency parsing.

Part-of-Speech Tagging The process of classifying and labeling words into their parts of speech is known as **POS tagging** [CFL13]. Parts of speech are also called word classes (e.g. noun, adjective, verb) or lexical categories. A tagset is a set of tags used for a particular task. Depending on the language, different tagsets can be used [3].

Constituency Parsing A constituency parser is a common way of representing the grammatical structure of a sentence, for instance, dividing the sentences into smaller phrases (e.g. nominal, prepositional, verbal) and, for each word, explaining what is its part-of-speech [ZZC+13]. There are possible constituent tags according to the language[4].

Figure 2.2 shows the output provided by StanfordCoreNLP's[5] constituency parser for the sentence *Good knowledge of English and basic French*. As it is possible to see, this sentence is complex and can be divided into two phrases: a nominal phrase (NP) – *Good knowledge of English* – and an adjective phrase (ADJP) – *basic French*. These two sentences are connected by the word *and* whose POS tag is a coordinating conjunction (CC). The leaves of the constituency tree show the words that compose the sentence.

[3] The *Universal Dependencies* is a framework for cross-linguistically consistent grammatical annotation. It provides documentation for the possible tags according to the language. The current possible POS tags for English can be found in https://universaldependencies.org/treebanks/en_ewt/index.html, while the French tags can be found in https://universaldependencies.org/treebanks/fr_ftb/index.html.

[4] The possible constituent tags for English can be found in https://surdeanu.info/mihai/teaching/ista555-fall13/readings/PennTreebankConstituents.html. The exact constituent tags for French could not be found.

[5] https://github.com/stanfordnlp/CoreNLP

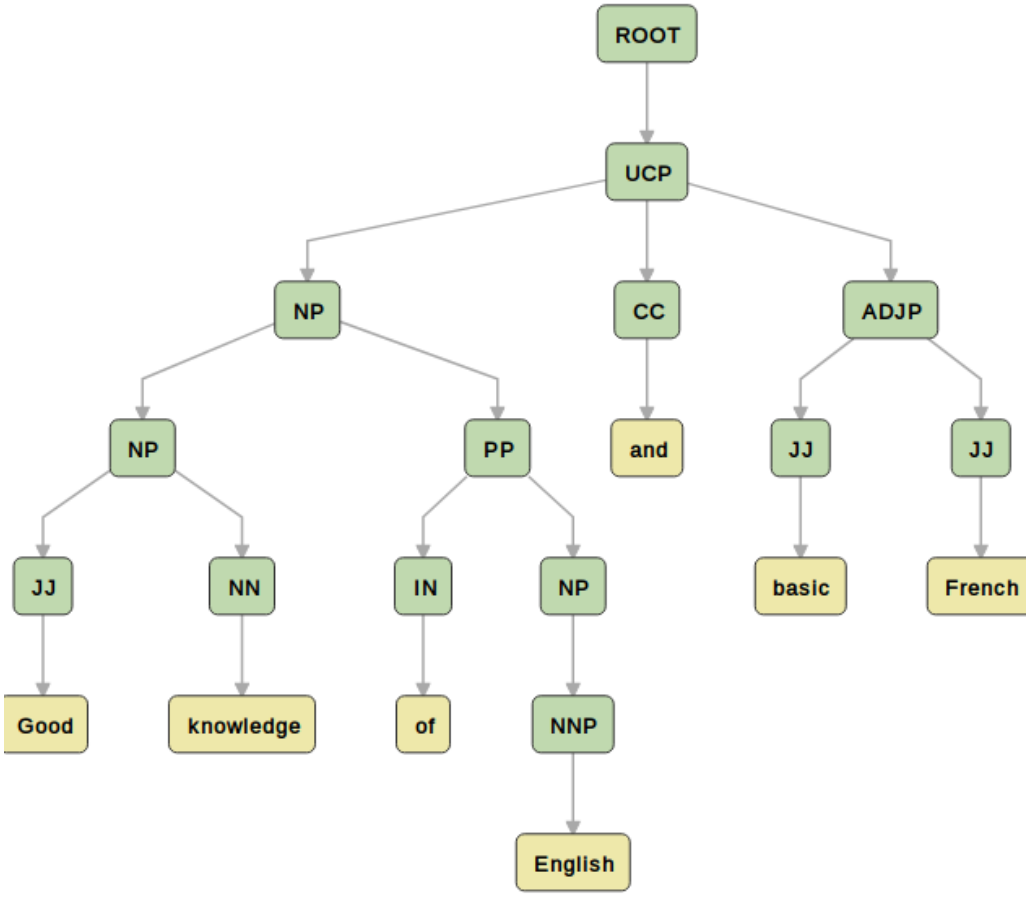


Figure 2.2: Constituency parsing tree for the sentence *Good knowledge of English and basic French* by *StanfordCoreNLP*

When one sentence has more than one syntax tree, as demonstrated in Figure 2.3, it is said that this sentence is ambiguous.

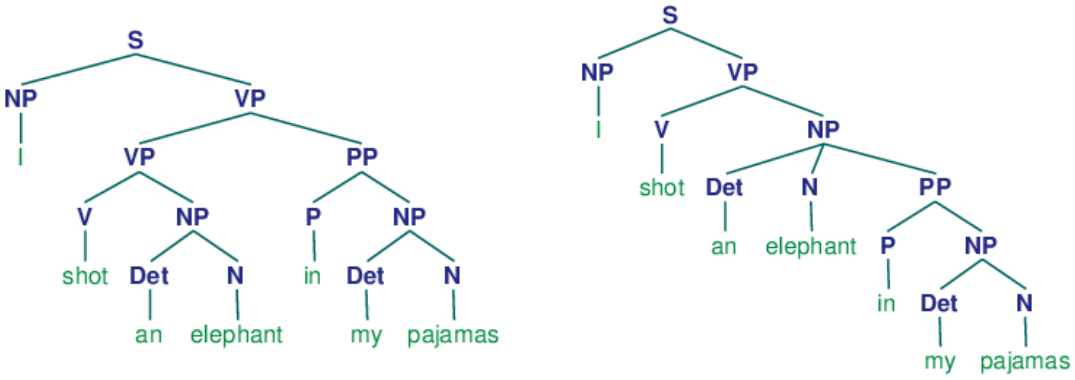


Figure 2.3: Two possible syntax trees for the same (ambiguous) sentence. [BLK09]

Dependency Parsing As stated by Chen and Manning [CM14], a dependency parser analyzes the grammatical structure of a sentence, providing the existent relationships between words. The relation occurs between "head" words and their modifiers.

StanfordCoreNLP's dependency parser provides different styles of dependency representation, such as the basic typed dependencies and the enhanced dependencies. The first provides a small set of relationships that are easier to parse, while the latter provides more detailed relationships, making it easier to work with[6].

Figure 2.4 represents the basic relationships that result from the application of StanfordCoreNLP's dependency parser to the sentence *Good knowledge of English and basic French.*, while Figure 2.5 provides the enhanced dependencies for the same sentence. As it is possible to observe, even though the sentence is relatively simple to analyze, the two figures show different relationships between the words. Figure 2.4 contains the dependencies between words with less detailed than Figure 2.5. For example, in the first one, it is possible to observe that there is a relationship between the words *knowledge* and *English* of the type *noun modifier (nmod)*. However, the latter provides additional information about this relationship, as it shows that these two words are connected by the word *of*.

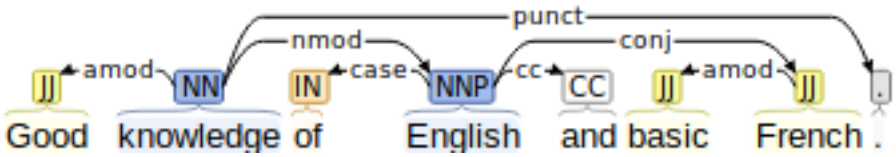


Figure 2.4: Basic dependencies for *Good knowledge of English and basic French.*

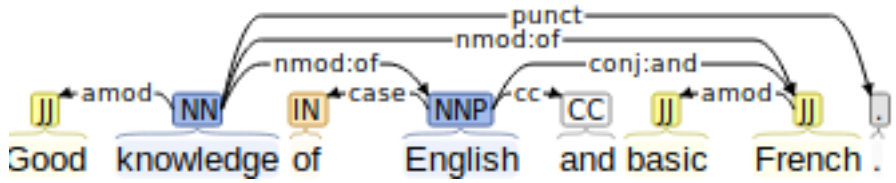


Figure 2.5: Enhanced++ dependencies for *Good knowledge of English and basic French.*

Table 2.4 summarizes some of the tools that can be used to perform syntactic analysis, with the respective features, implementation programming language and whether the tool provides support for different languages.

[6]The possible existent relationships between words can be found in the Universal Dependencies framework in https://universaldependencies.org/treebanks/en_ewt/index.html and https://universaldependencies.org/treebanks/fr_partut/index.html for English and French, respectively.

Name	Features	Programming Language	Multilanguage support
Natural Language Toolkit (NLTK)	POS tagging, dependency parsing[1], constituency parsing[1]	Python	Yes
StanfordCoreNLP	POS tagging, dependency parsing, constituency parsing	Java	Yes
SpaCy	POS tagging, dependency parsing, constituency parsing	Python	Yes

[1] Uses the parser implemented by StanfordCoreNLP

Table 2.4: Some of the tools used for syntactic analysis

2.3.1.4 Semantic analysis

Semantic analysis is concerned with understanding the meaning of single words, expressions, whole sentences, and utterances in context. Semantic parsing consists of translating original expressions into representations that can be understandable by a machine. [CFL13, p. 94]

A **shallow semantic analysis** is concerned with labelling phrases of a sentence with the semantic roles they play regarding a target word. For example, the sentence *Shaw Publishing offered Mr. Smith a reimbursement last March.* is labeled as *[AGENTShaw Publishing] offered [RECEPIENTMr. Smith] [THEMEa reimbursement] [TIMElast March].* [PWH+04]

Traditionally, there is a division between lexical semantics, which concerns itself with the meaning of single words, and combinational semantics, which concerns itself with the meaning of sentences or expressions. Although it is reasonable to make this division, the interaction between world-level semantics and grammatical semantics has gained some recognition. The use of lexicogrammar as a whole has been gaining followers as many grammatical constructions have construction-specific meanings.

Ambiguity resolution is usually a requirement used for semantic analysis.

2.4 Machine Learning applied to NLP

According to [APA+17], Machine learning (ML) techniques are widely used for major NLP tasks (e.g. for linguistic knowledge disambiguation). ML involves the study of methods that are able to extract information automatically. ML algorithms accept as parameters data from which features are synthesized, and these features can later be used in prediction or classification. As mentioned in [KDNA16], learning problems fall into three categories: supervised learning, semi-supervised learning, and unsupervised learning. Supervised learning is the current dominant method to address NLP tasks.

Unsupervised ML algorithms try to discover groups of similar examples within unlabeled data [PVG+11]. The most common approach of unsupervised learning is clustering.

Supervised machine learning is about learning some properties of a data set and then test these properties against an unseen data set. Basically, ML systems learn how to combine inputs to produce useful predictions on unknown data. Supervised learning can address either a classification or regression problem. Whereas regression models predict continuous values, classification models are used to deduce discrete values.

Before further details, let us introduce the ML terminology that consists on *labels* **y** (what is being predicted), on *features* **x1, x2, ..., xn** (the input variables describing the data), on *examples* (particular instances of the data), and on *models*. The examples can be labeled if they include both features and the label, or unlabeled if they only contain the features. Supervised models define the relationship between features and labels.

A model's life cycle consists on two phases: the **training** phase, where labeled examples are given to the model so it learns the relationships between features and labels; and an **inference** phase, where the trained model is applied to an unseen data set and it predicts the labels for these unlabeled examples.

2.4.1 Feature Extraction

Since ML algorithms cannot deal directly with raw text, in order to perform machine learning on text documents, we first need to convert the text into numerical feature vectors that reflect linguistic properties of the text. This step is called **feature extraction**.

Conventionally, supervised approaches take a word, convert it to a symbolic ID and then transform it into a one dimension feature vector with the same length as the size of the vocabulary. The features are independent of one another. This is called a **one-hot representation**. The use of one-hot representation comprises some disadvantage: data sparsity that results from the high number of distinct words present in the corpus; the number of features is equal to the number of distinct words in the corpus (vocabulary), which can be computationally expensive; the inability of handling words that are not present in the labeled training data [TRB10] [PVG+11]. However, one-hot representations may come in handy when there are relatively few different uncorrelated features.

As an alternative, features can be represented as **dense vectors** instead, i.e., each feature is *embedded* into a *d* dimensional space, and then represented as a vector (*embedding)* in that same space. According to Goldberg [Gol16], this representation has the advantage of sharing information between similar features, as they have similar vector representations.

2.4.2 Training a model

Training a model means learning good values for all the weights from labeled examples. In supervised learning, a ML algorithm builds a model that tries to minimize loss while examining many examples. Loss indicates how bad the model's prediction was on a single example.

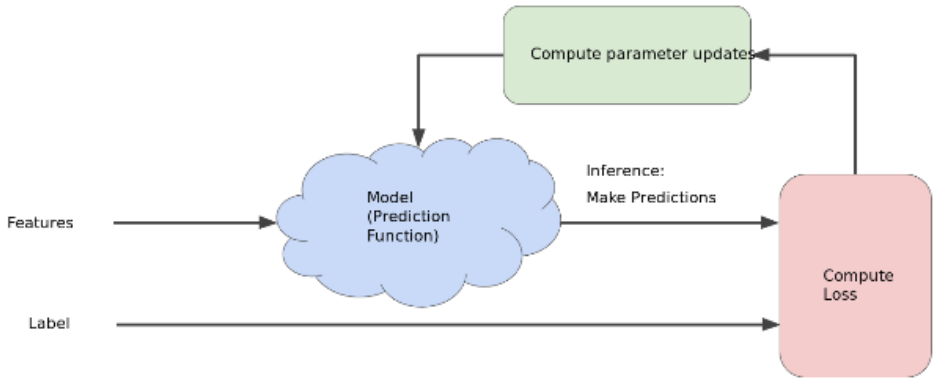


Figure 2.6: Iterative trail-and-error process to train a model [Dev]

Figure 2.6 summarizes the process of training a model. In order to minimize the loss, several ML techniques make use of a mechanism called gradient descent. This algorithm calculates the gradient of loss, a vector of partial derivatives. The negative gradient indicates the direction to update the parameters of the model in order to reduce loss. The algorithm takes a step in that direction. These steps are repeated, while the algorithm edges closer to the minimum.

The ML model should not be too complex, otherwise, there is a risk that the model is overfitting. Although an overfit model has a low loss during the training phase, it fails to do a good job when predicting new data. The simpler the model, the more likely the model will be able to generalize when seeing new data.

2.4.3 Training, Test and Validation Sets

Like seen before, a model has a training and an inference phase. Each of these phases needs different datasets so that the results predicted are accurate. To do so, we divide our randomized dataset into three different datasets: the training set, the validation set, and the test set. The validation set is used to reduce the risks of overfitting our model. The balance between these three sets is important, but the larger the training set, the better the model can generalize. This process can be synthesized in Figure 2.7.

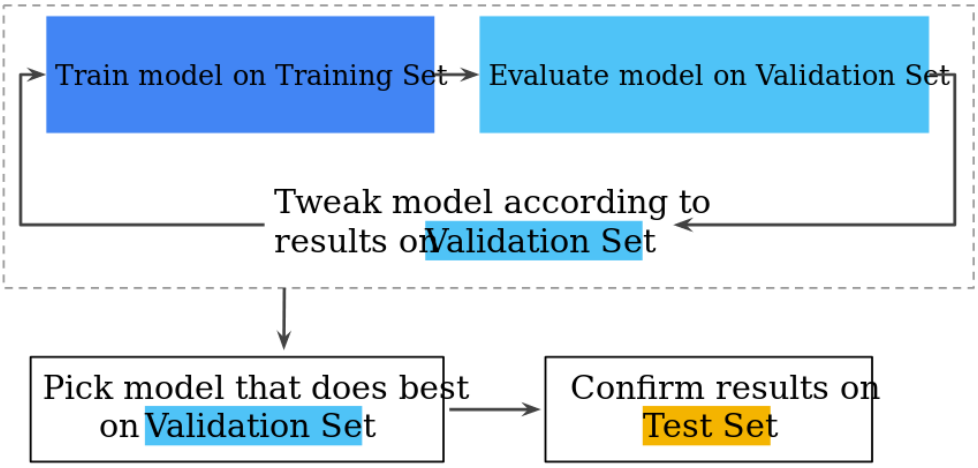


Figure 2.7: Workflow [Dev]

As stated in [Dev], one of the problems of dividing the data into three subsets is that it can reduce drastically the number of examples which may not be enough to learn a good model. A solution to this is called **cross-validation**. When the number of samples is not big enough, it is possible to split the training set into *k* smaller sets. For each of the *k* sets, the model is trained using *k - 1* of the folds as training data. Then, the model is validated in the remaining part of the data. This approach can be computationally expensive, but a good solution to problems where the number of samples is not enough.

2.4.4 Evaluation of the Model

In order to evaluate the model, there are several metrics one can use. A **confusion matrix** is a NxN table that summarizes how successful the classification model's predictions were; that is, it contains the correlation between the label and the model's classification. One axis of the matrix represents the predicted labels, while the other contains the actual labels. For a binary classification problem, N is equal to 2 and we can classify the classes as positive or negative. For example, if we want to determine whether there is a contradiction or not, we define *contradiction* as the positive class and *no contradiction* as the negative class.

On the one hand, a true positive (TP) happens the model predicts correctly the *positive* class. Similarly, a true negative (TN) occurs when the model predicts successfully the *negative* class. On the other hand, a false positive (FP) is the result of an incorrect prediction of the *positive* class, while a false negative (FN) is an outcome where the model incorrectly predicts the *negative* class.

One of the metrics used to evaluate the model is **accuracy**. Accuracy can be defined as the fraction of successful predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

It is important to notice that accuracy alone does not pay attention to class-imbalanced data sets. For this purpose, one should also use metrics as precision, recall and F-score. **Precision** can be defined as the proportion of positive identification that was actually correct, while **recall** can be defined as the proportion of actual positives that were identified correctly. **F-score** considers both precision and recall.

$$Precision = \frac{TP}{TP+FP} \tag{2.2}$$

$$Recall = \frac{TP}{TP+FN} \tag{2.3}$$

$$F-score = 2 * \frac{precision * recall}{precision + recall} \tag{2.4}$$

2.4.5 Text classification

Text classification (or categorization) can be performed by extracting features from the text, training a classifier to generate a model, and evaluating the performance of the model on the test set. A possible way to perform text classification can be visualized in Figure 2.8.

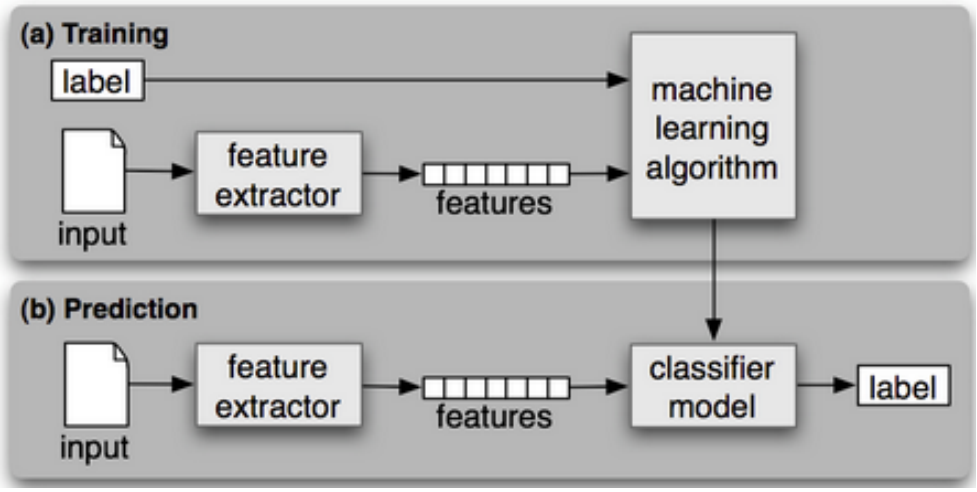


Figure 2.8: Conventional Supervised Classification [BLK09]

There are several ML algorithms that can be used to tackle text classification problems. Some of these algorithms are mentioned below as their understanding is required in Chapter 3.

Naive Bayes Naive Bayes methods are a set of supervised algorithms based on the Bayes' theorem [7] that make the *naive* assumption of conditional independence between the features given the class [PVG+11]. Bayes classifier attributes the most likely class on new examples.

Although independence is a poor assumption, naive Bayes often competes well with more advanced classifiers. Rish et al. [R+01] demonstrated that naive Bayes works well for certain nearly-functional feature dependencies, reaching its best performance in two contrasting situations: when features are completely independent and when features are functionally dependent.

Support Vector Machine Support vector machines (SVM) are supervised learning techniques used to separate data linearly. A single SVM can only separate two classes of data, a positive and a negative classes. This algorithm attempts to find a hyperplane as far as possible from both negative and positive examples - the support vectors. The support vectors specify the actual location of the hyperplane. When the classification problem is not linearly separated, SVM try to determine a hyperplane that minimizes the number of document vectors located in the wrong side [APA+17]. A visual representation of SVM can be seen in Figure 2.9.

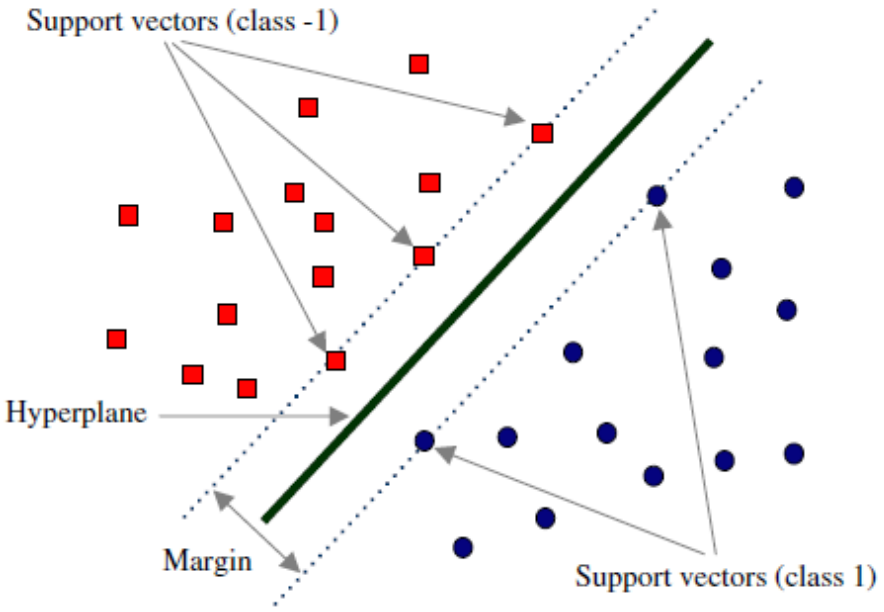


Figure 2.9: Support Vector Machines

One advantage of this method is that learning is almost independent of the dimensionality of the feature space, which is an ideal choice regarding text data due to the sparse high dimensional nature of the text with little irrelevant features.

[7] Bayes' theorem describes the probability of an event occurring based on prior knowledge of conditions that might be related to the event.

K-means K-means clustering is an unsupervised learning approach whose purpose is to cluster the data into k different groups. More recently, it has been applied in classification tasks.

The algorithm takes as input a set of k center-points (averages of the data) randomly defined. Then, the observations are assigned to their nearest center-point. When the distance from the observation is the same for several center-points, a random one is chosen. After this, each center-point is recalculated by calculating the centroid of the assigned observations. The two previous steps are recomputed until all observations remain at the assigned center-points, more specifically, until the center-points are no longer updated [MKS18].

Because the k values are randomly distributed, they can be computationally expensive and they could be set in a way that the data would still be fairly distributed although it may not be the right solution.

2.4.5.1 Neural Networks applied to NLP

Neural networks (NN) are powerful learning models that can be applied to NLP tasks, pattern recognition, and ML problems very effectively. As stated by Goldberg [Gol16], all NN act as classifiers, but each with different strengths. Non-linear NN can replace linear classifiers, obtaining usually better accuracy values when integrated with pre-trained word embeddings. In this section, two NN will be discussed: feed-forward networks and recurrent/recursive networks.

When moving from sparse-input linear models to neural-network based models, features are no longer represented as one-hot representations, but instead as dense vectors - *embeddings*. In NN, these embeddings are used as model parameters and they need to be trained along with other components of the network. Dense vectors are used because, in general, neural networks do not deal well with sparse vectors.

When dealing with multi-class classification problems with k classes, the NN produces a vector with k dimensions as the output in which every dimension represents the strength of a certain output class.

NN are inspired by the brain's computation mechanism. The basic computation units are known as neurons - imitations of real neural cells. The neuron is constituted by scalar inputs (dendrites), a body, and scalar outputs (axon). Each input has an associated weight that represents its importance. The neuron multiplies each input by its weight, sums them, feeds the result to the activation function (usually a non-linear function), passing its value to its output. Sometimes a bias term is added to the total weighted sum of inputs to act as a threshold for changing the activation function. Neurons connected to each other form a network where the output of a neuron can be fed into the input of one or more neurons. A neural network is arranged in layers reflecting the flow of information. There is an input layer and an output layer, whereas the layers in between are named hidden layers.

In a **feed-forward NN**, the input layer does not receive any values and the output layer does not send any information. When each neuron is connected to every neuron in the following layer, this is called a *fully-connected layer* or an *affine layer*. Besides fully-connected layers, feed-forward NN also include convolutional and pooling layers.

Feed-forward networks and convolutional networks allow us to encode arbitrary length sequences as fixed-size vectors. The first one makes use of vector concatenation and vector addition, which has the limitation of disregarding the order of features. The latter that can capture most salient features and allows some sensitivity to word order, although most of the structural information is sacrificed. An example of a feed-forward network can be seen in Figure 2.10.

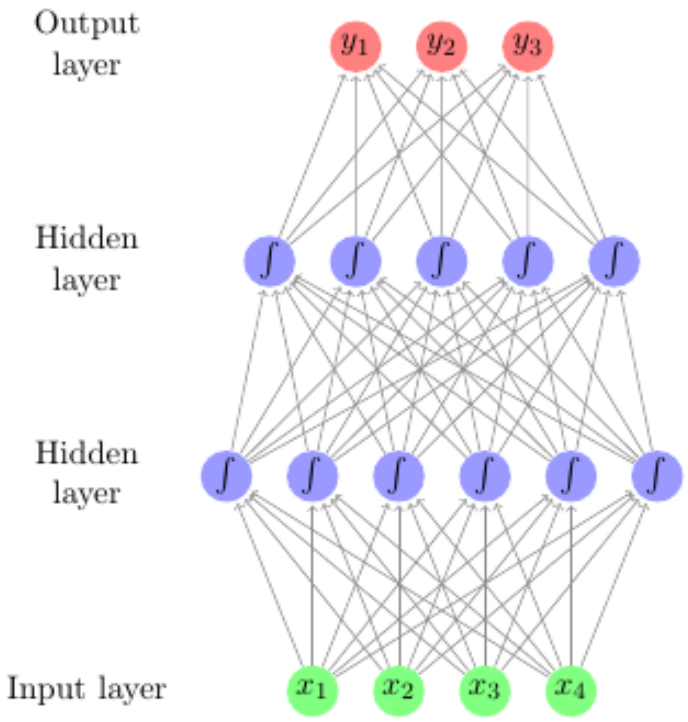


Figure 2.10: Feed-forward NN with two hidden layers [Gol16]

As well as feed-forward networks, **recurrent neural networks (RNN)** also allow the representation of arbitrary size inputs into a fixed-size vector. However, they do so by preserving most of the structural information.

2.5 Inter-Annotator Agreement

As stated before in this Chapter, in text classification tasks, the development of a model requires testing and validation. The labeled data used to validate the model must be reliable, i.e., the data must have been annotated by different people that consistently agree on the labels assigned to the parts of the text. If the annotations are consistent, it is possible to infer that the different annotators have a similar understanding of the annotation guidelines they were provided with, i.e., the instructions that must be followed throughout the annotation process. If this condition fails to happen, then the data is not fit to validate the model as it will provide unreliable results.

Inter-annotator agreement (IAA) [SDL80] is the degree of agreement amongst the different annotators. If there is perfect agreement between annotators, then the IAA has a value of 1. If there is no agreement at all, this value is 0. In order to consider data as reliable, the inter-rater agreement must be close to 1, although the acceptance values vary according to different metrics.

There are several factors that may lead to disagreement between annotators [AP08]:

- Lack of domain knowledge
- Uncertainty in deciding on a category
- Some cases are difficult to classify
- Difficulty in understanding the provided guidelines or unclear guidelines
- Tiredness or lack of concentration
- Personal opinions that may bias the interpretation

2.5.1 A Common Notation

Calculating the IAA involves a set of items to which are attributed categories by a set of annotators. There are different notations that can be used to express such concepts, but we use the notation provided by Artstein and Poesio [AP08]:

1. The set of items is **{i | i ε I}** and is of cardinality **i**;
2. The set of categories is **{k | k ε K}** and is of cardinality **k**;
3. The set of annotators is **{c | c ε C}** and is of cardinality **c**
4. The observed agreement is A_o
5. The observed disagreement is D_o
6. The expected agreement is A_e
7. The expected disagreement is D_e
8. The number of annotators who assigned item i to category k is n_{ik}
9. The number of items assigned by annotator c to category k is n_{ck}
10. The total number of items assigned by all annotators to category k is n_k
11. The agreement value for item i is agr_i

2.5.2 Metrics

There are several metrics that can be used to calculate the inter-annotator agreement. Some metrics take into account the probability of the annotators agreeing by chance.

2.5.2.1 Agreement without chance correction

The simplest method to calculate the percentage of agreement between distinct annotators is by simply calculating the sum of the number of items on which the annotators agree divided by the total number of items.

If the coder agree with the assignment, then $agr_i = 1$, else $agr_i = 0$.

$$A_o = \frac{1}{i} \sum_{i \varepsilon I} agr_i \tag{2.5}$$

The problem with this metric is that it is does not take into account the probability of the annotators agreeing by chance when they guess randomly.

2.5.2.2 Chance-corrected coefficients for measuring agreement for more than two annotators

The chance-corrected coefficients of agreement are based on the same idea: the agreement is equal to the agreement beyond chance found divided by the attainable chance-corrected agreement. This can be translated to the following formula:

$$\frac{A_o - A_e}{1 - A_e} \tag{2.6}$$

Although there are some agreement metrics that can be used to calculate the agreement between two annotators, such metrics will not be discussed as their analysis is not relevant for the purpose of this dissertation.

Krippendorff's alpha coefficient The coefficient α [Kri11] measures the observed disagreement and the expected disagreement. Its form can be seen below:

$$\alpha = 1 - \frac{D_o}{D_e} \tag{2.7}$$

where:

$$D_o = \frac{1}{ic(c-1)} \sum_{i \in I} \sum_{j=1}^{k} \sum_{l=1}^{k} n_{ik_j} n_{ik_l} d_{k_j k_l} \tag{2.8}$$

$$D_e = \frac{1}{ic(c-1)} \sum_{j=1}^{k} \sum_{l=1}^{k} n_{k_j} n_{k_l} d_{k_j k_l} \tag{2.9}$$

$d_{k_j k_l}$ corresponds to the distance function. As Krippendorff's alpha is suitable for different types of data, such as binary, nominal, ordinal, interval, ratio and others, the distance function's form varies according to the type of data. For binary or nominal data, $d_{k_j k_l} = (k_j == k_l ? 0 : 1)$, that is, if both categories are equal, the distance between them is 0, otherwise it is 1. For ordinal data, the distance function is equal to the square distance between categories: $d_{k_j k_l} = (k_j - k_l)^2$.

α can take values between 0 and 1. If there is no disagreement, then α is equal to 1. If there is little disagreement, then α takes values higher than 0.8. If alpha takes values lower than 0.67, then there is little IAA and the data cannot be reliable[8].

There are several reasons that justify the use of Krippendorff's alpha coefficient:

- It is suitable for any number of annotators;
- It handles missing data;
- It deals with different types of data.

2.6 Summary

In this Chapter, we present some key concepts that are necessary to understand the following Chapters. We do not provide an exhaustive and detailed explanation of the concepts as this is not required.

Some of the theoretical knowledge hereby presented is not required to understand the proposed approach detailed in Chapter 5. However, these key concepts may be required to understand some of the approaches specified in Chapter 3.

[8]These values were taken from a tutorial on IAA held as part of the doctoral program "Language and Knowledge Engineering" (LKE) at the Technische Universität Darmstadt, Germany on November 9, 2009 by Christian M. Meyer.

Chapter 3

Related Work

This chapter provides an overview of the state of the art and other related works concerning the detection of contradictions and ambiguities.

3.1 Contradiction Detection

There are several approaches to tackle the problem of detecting contradictions.

Marneffe et al. [MRM08] describe a system for automatically detecting contradictions given two sentences (text and hypothesis). For these authors, detecting contradictions is a more complex task than detecting textual entailments, as contradictions require deeper inferences and model building.

Their approach requires little external information and it involves a dataset obtained through different resources and techniques, such as WordNet[1] and VerbOcean[2].

They apply a multiple-step approach: first, they convert the text and hypothesis to typed dependency graphs produced by the Stanford parser [DMMM+06]; this step is followed by the alignment between text and hypothesis graphs; their next step consists of removing pairs of sentences that do not correspond to the same event to avoid finding false evidence of contradiction; their final step consists on the extraction of many contradiction features (polarity, number, antonymy, modality, and relational features) from the text followed by the application of logistic regression models to classify the pair of sentences as contradictory or not. The main problem of the overall method results from the lack of feature generalization.

In [PNS13], the authors propose a new rule-based method based on shallow semantic representations for contradiction detection. The low-coverage problem involving shallow semantic representations is addressed by using a backup module that relies on binary relations extracted

[1] WordNet is a lexical database for the English language. It groups English words into sets of synonyms called synsets and provides short definitions and usage examples. [Uni10]

[2] VerbOcean is a broad-coverage semantic network of verbs [CP04]

from sentences for finding contradictions. An overview of their system's architecture can be seen in Figure 3.1.

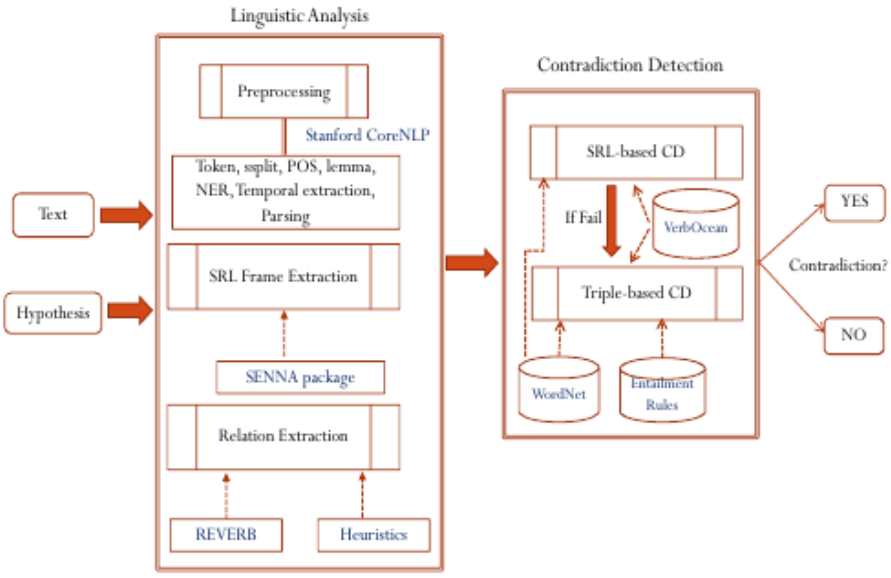


Figure 3.1: Architecture of the contradiction detection system [PNS13]

Their approach was proven to achieve better precision and recall values than most of the baseline methods on standard datasets of Recognizing Textual Entailment (RTE) challenges due to their ability to change between contradiction detection strategies when one of them fails. However, the values obtained for the metrics chosen vary between 14% and 23% (precision) and 19.44% and 28.89% (recall).

Some limitations of their system lie on the fact that the system does not take into account unary relations and it lacks common sense knowledge.

Dragos [Dra17] studies the problem of detecting a contradiction in uncertain statements and tackles the problem as a categorization problem. In the paper, contradictions that happen in objective statements are differentiated from contradictions that occur in subjective statements. In the first case, contradictions cover different relationships between concepts, properties, and attributes of entities and require deep processing and reasoning to examine the consistency of information. In the second, contradictions are related to different dimensions such as vagueness, ambiguity, imprecision or uncertainty.

Dragos defines a typology of relations between statements: agreement, disagreement and conflict. Then, a model is developed considering two facts: the factual information carried by sentences and its classification as certain or uncertain. This model analyzes disparities over facts reported and certainty estimations and then combines previously extracted features to provide an overall characterization. The architecture of their system can be seen in Figure 3.2.

This approach has the advantage of separating the contradictions that result from disagreement over how certain a fact is and the contradictions related to more conflictual opinions about the facts

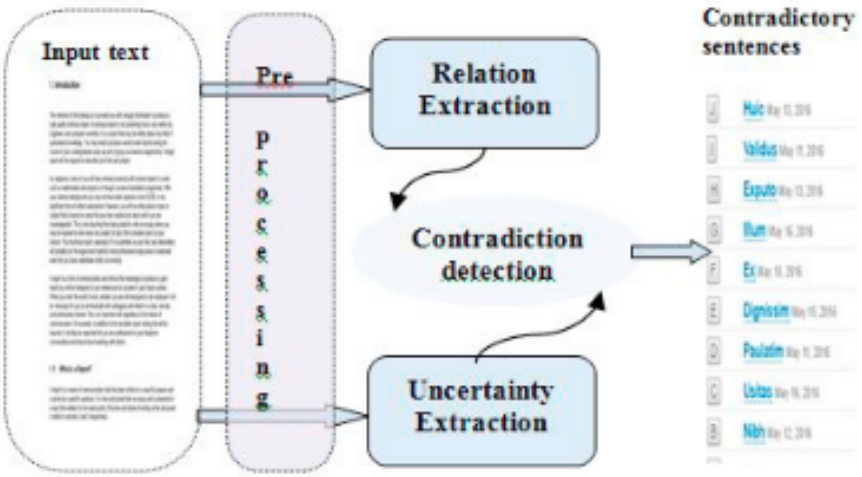


Figure 3.2: Architecture for contradiction detection [Dra17]

reported. In addition, this method overcomes the "limitations related to the inconsistencies in the interpretation of words having fuzzy semantic boundaries" [Dra17, p. 77]. However, relational features cannot be extended easily for broad coverage and robust detecting in large corpora.

Mezghani et al. [MKS18] developed a system to deal automatically with redundancy and inconsistency between requirements. They make use of classification approaches to deal with this problem, more precisely they make use of k-means clustering algorithm. Their approach consists of several steps (Figure 3.3). First, they focus on the extraction of requirements using a predefined function in SEMIOS software[3]. Then they apply a preprocessing step to erase "noisy" requirements followed by a POS tagging and noun chunking to extract business terms. At last, the k-means algorithm is applied.

The use of the k-means algorithm is a bigger contribution as it improves the overall efficiency of the system, especially with the filtering and preprocessing steps. It also is domain-independent and does not require *a priori* knowledge. A drawback is that it demands predefined values for the set k and finding its optimal values is a complex task, one yet to be improved. Another shortcoming is this solution cannot distinguish redundancy or inconsistency in quite similar clusters.

Lingam et al. [LBN+18] propose a solution to detect three types of contradiction: negation, antonymy, and numeric mismatch. This problem is tackled as a classification problem. Their approach is two main phases: the extraction of features and the detecting of contradictions.

For the feature extraction phase, as seen in Figure 3.4, first, the input text is converted to lower case. Then, they create a corpus by combining both training and test instances for a particular dataset. Each term in the corpus is given an index id so that sentences are transformed into vectors containing the index id of the sentence's words. Then vectors are transformed in fixed-size vectors

[3]SEMIOS is a software used for error detection in specifications from the conception phase.

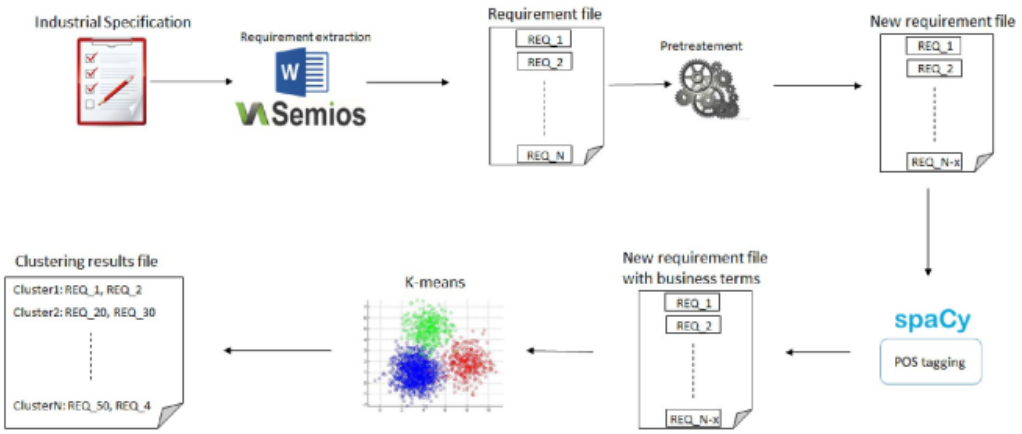


Figure 3.3: Clustering approach overview [MKS18]

in the padding step. After this, they perform GloVe embedding[4] followed by the combination of RNN with bidirectional long short-term memory(LSTM) to encode an English sentence into a vector that contains the semantic meaning of that same sentence. This technique was chosen because learning a good representation of the sentence is an essential task for detecting contradictions.

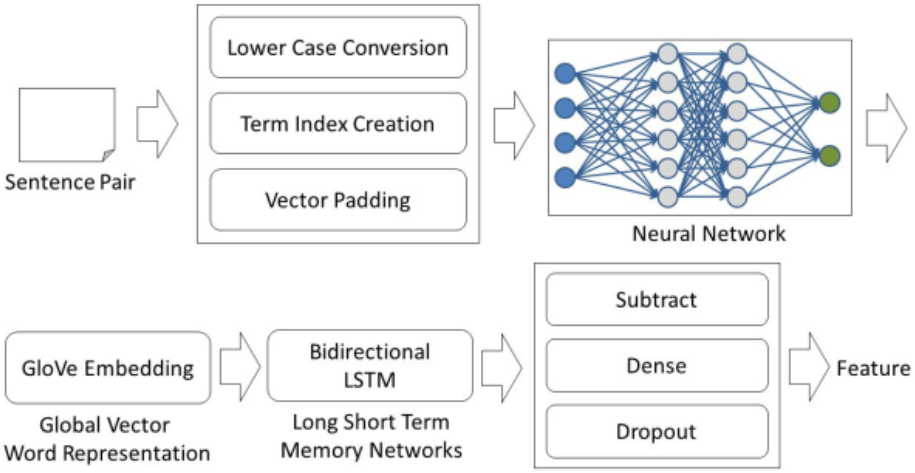


Figure 3.4: High Level Solution Approach and Research Framework Diagram - LSTM based feature extraction [LBN+18]

In the detecting contradictions phase (Figure 3.5), they use the features extracted alongside with some manually created features as inputs and obtain as output two possible values: "contradiction" or "neutral".

[4]GloVe embedding helps to create vectors from words that capture linguistic regularities. https://nlp.stanford.edu/projects/glove/

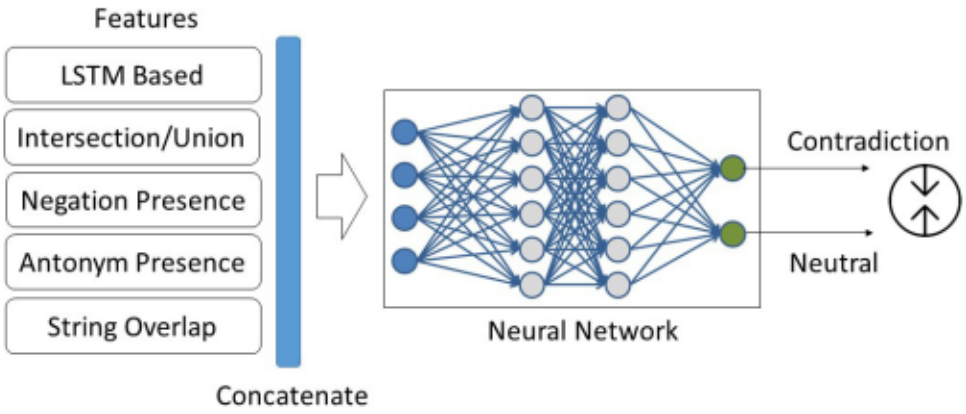


Figure 3.5: High Level Solution Approach and Research Framework Diagram - Contradiction and Neutral Detection in Text using Neural Network and Five Features [LBN+18]

The model is measured using the accuracy metric, demonstrating that is possible to find contradictions accurately (the minimum value obtained for accuracy was 71.99%, being the maximum 98.39%). The usage of both features generated by the neural network and the manually engineered features increases significantly the accuracy of the model. However, one thing yet to confirm is if this solution can also be applied to other types of contradictions and if the features are indeed responsible for the observations seen in the obtained outputs.

3.2 Ambiguity Detection

Gleich et al. [GCK10] propose a tool to automatically detect ambiguities and explain ambiguity sources in English and German documents. Their tool investigates the input text line by line and checks whether the line matches certain patterns, using regular expressions. If the expression matches a pattern, then the match is marked as ambiguous.

Their solution is based on different detection techniques: they define single keywords to deal with ambiguities that result from the multiple meanings a word can have; they define regular expressions to deal with ambiguities resulting from either possible word combinations or the occurrence of a given word in a certain position; they used TreeTagger [Sch] to perform POS tagging in order to detect ambiguities resulting from the usage of the passive voice, adjectives, and adverbs. The solution for German documents only differs in the fact that the keywords are translated and that the regular expressions for passive detection are changed to fit the German grammar.

To evaluate the tool developed, they created a data set with both English and German sentences and asked different people to annotate the data set. Then these manually written annotations were compared to the output of their tool. Their tool was analyzed based on two different metrics: precision and recall. The results obtained show that the tool is fit for practical use, correctly identifying 6 out of 7 ambiguities (recall is 86%) and obtaining precision values between 95% and

97%. However, the tool fails to identify sometimes pragmatic and language errors, as pragmatic ambiguities require a profound knowledge of the application domain.

Rojas and Sliesarieva [RS10] propose a method to verify whether software requirements written in English satisfy three desired properties: accuracy, non-ambiguity, and verifiability. To do so, they combine NLP techniques with specialized dictionaries. Because we are focused on techniques to detect ambiguities, we will just focus on that part of their solution.

In their research, they focus on lexical, syntactic, vagueness and language errors. First, they performed a lexical analysis followed by a syntactic analysis. The first one makes use of regular expressions to identify keywords or phrase structures, such as vague adverbs [5] and non-deterministic language constructs [6]. The second one takes language statements and transforms them into their grammatical components.

In order to identify syntactic ambiguities, they used a parser and studied the output of that same parser. Besides this, they also incorporated dictionaries to support their analysis: WordNet and VerbNet[7], as dictionaries allow the identification of ambiguous verbs. This happens because the parser used identifies the main verb for a requirement, while queries into dictionaries complete the task.

The tool developed is able to examine hundreds of requirements in minutes, while this task performed manually takes hours or even days. One problem that the researchers faced was the high degree of discrepancies and disagreement between reviewers that hampers the comparison between the tool's results and the reviewer's results.

Although rule-based NLP techniques have been improved to automatically detect defects in natural language requirements, Rosadini et al. [RFG+17] mention that there is a lack of empirical studies on the application of these approaches. Therefore, their goal is to search to which extent NLP can be practically applied to detect defects in requirement documents of a railway signaling manufacturer.

Their approach consists on first identifying a set of common classes that cause defects in natural language. These classes are anaphoric ambiguity, coordination ambiguity, vague terms, modal adverbs[8], passive voice, excessive length, missing condition, missing unit of measurement, missing reference, and undefined term. Then, for each class, they define a set of defect-detection patterns based on Java Annotation Patterns Engine (JAPE) rules[9] and other NLP technologies, such as tokenization, POS tagging, shallow parsing[10]

From the previously defined classes, the ones that cause uncertainty and, therefore, can be sources of ambiguities are anaphoric ambiguity, coordination ambiguity, vague terms, modal verbs, passive voice, and excessive length, i.e., longer sentences are harder to process than short sentences.

[5] E.g.: *acceptable, high, low, fast, sufficient, normal* and others

[6] E.g.: *and/or, any, not limited to* and others

[7] VerbNet is the largest online verb lexicon currently available for English. [ver]

[8] Modifiers that express a quality associated with a predicate. E.g.: *Positively, permanently, clearly*, among others

[9] JAPE rules allow defining rules over tokens and other constituents in a text. It is based on JAPE grammar, that is similar to regular expressions.

[10] This technology only identifies the noun and verb phrases.

After implementing the patterns previously defined, an evaluation of the system was performed in two different datasets, a smaller and a bigger one. This was done by first annotating the datasets and then comparing the annotated datasets with the results provided by the system developed. They used recall and precision as metrics to evaluate the system.

For the smaller dataset, their results show that this system can detect defects in requirements caused by anaphoric and coordination ambiguities, modal verbs and passive voice, with recall values of 100%. Nevertheless, this system shows lower values (around 60%) when recognizing defects whose sources are vague terms and excessive length. These low results are due to the absence of certain words and limitations regarding the tokenizer, respectively. In order to detect with higher recall values vague terms, the researchers defend a lexical-based technique over a statistics-based method.

For the bigger dataset, containing 1866 requirements, the number regarding FP was very high, and their system had a precision of only 5.7%. The researchers later concluded that this low performance could be associated with a poorly manual annotated dataset that did not focus on linguistic defects, but instead on conceptual defects. However, the results obtained show that this tool cannot be used in a real-world setting, contrary to the expected.

The researchers concluded that in their large-scale case, NLP alone is not sufficient to detect certain defects, such as conceptual defects that require domain knowledge. Although some of these defects could be detected if a correlation between linguistic and conceptual defects had been computed.

Sabriye and Zainon [SZ17] propose a rule-based framework to detect syntactic ambiguities in software requirements specification document. Initially, the proposed framework performs POS tagging on the requirements by using the Stanford POS tagger. Then, they check if the POS tagger returns more than one parse tree for each sentence and checks whether any sentence contains AND and OR logics. If so, then the requirements are considered as ambiguous. Lastly, the number of ambiguities are calculated as well as the percentage of ambiguities detected in order to evaluate the model. The results of this evaluation are not shown, although the researchers conclude that further investigation must be done to implement the developed framework to automatically detect ambiguities.

3.3 Summary

From what is present, we can conclude that contradiction detection is a technical and challenging NLP task. Finding contradictions can be tackled as a classification problem. A common method to solve this problem includes deriving linguistic-based features from text and then train or learn a classifier from annotated examples to perform the categorization task. Traditionally, approaches were based on the elaboration of rule-based models. Nowadays, approaches are commonly based on linear ML models or non-linear neural networks. However, recently, neural network approaches using dense vectors are starting to have better results than linear ML models that

use one-dimensional sparse vectors. One problem regarding ML linear approaches is related to the lack of feature generalization in a large corpus.

Ambiguity detection is also a challenging NLP task. In the past years, some NLP techniques have been developed to assist ambiguity detection. Part of these techniques focuses on rule-based approaches. The problem regarding the use of rule-based approaches is that they fail to detect ambiguities that require world knowledge and domain knowledge. Because of this, it is possible to conclude that NLP alone is not enough to detect this type of ambiguities.

As far as we are concerned, no works related to ambiguity and contradiction detection in job postings were found. Nevertheless, although the works discussed in this chapter are not related to the job recruitment domain specifically, they provide an interesting overview of some methods that can be used to tackle the problem we have in hands.

However, the methods present in this Chapter are data-dependant. Therefore, the lack of annotated data influences the different approaches that can be taken. As we initially did not have an annotated dataset, we detail, in the next Chapter, how we annotated a dataset from scratch.

Chapter 4

Dataset

As mentioned in Chapter 1, when a recruiter creates a job posting in *skeeled*'s platform, the application makes use of Machine Learning algorithms to rank candidates and present them by order of relevance.

Nevertheless, finding suitable candidates for a job opening is a complex process. On the one hand, it requires an analysis of the job posting's several fields to obtain its requirements. On the other hand, it involves the analysis of the several potential candidates' profiles, based on three main factors: the applicant's CV, the candidate's personality test and the applicant's video interview. Despite the inherent complexity of these two analyzes, there are some factors that might jeopardize the matching process: the inconsistencies that might occur not only in the job posting's fields but also in the candidate's profile. For the purpose of this dissertation, only the inconsistencies coming from job postings are considered and discussed.

Surpassing such inconsistencies is a crucial matter. Our solution consists of elaborating a model that classifies several phrases as contradictory with one another or not. Then, we compare our model's output with a labeled dataset to evaluate the performance of the developed model. As *skeeled*'s dataset had not been annotated before, our approach's first step comprises the annotation of contradictions and ambiguities in the dataset.

In this chapter, we provide a detailed explanation of the several steps taken to annotate the dataset, starting with a description of the dataset and its fields, followed by an explanation of the possible inconsistencies that might occur in the dataset. Then we provide the annotations guidelines alongside a discussion of the results of the annotations and some of the difficulties encountered.

4.1 Dataset Description

skeeled's dataset is composed of a set of job postings stored in *json*. This dataset is constantly being updated as recruiters write new job postings. For the purpose of this dissertation, we used a

fraction of that same dataset, composed of 805 job postings published by *Skeeled*'s recruiters. As *skeeled* is a multilingual platform, the job postings are written in different languages. A distribution of the number of job postings per language can be seen in Figure 4.1.

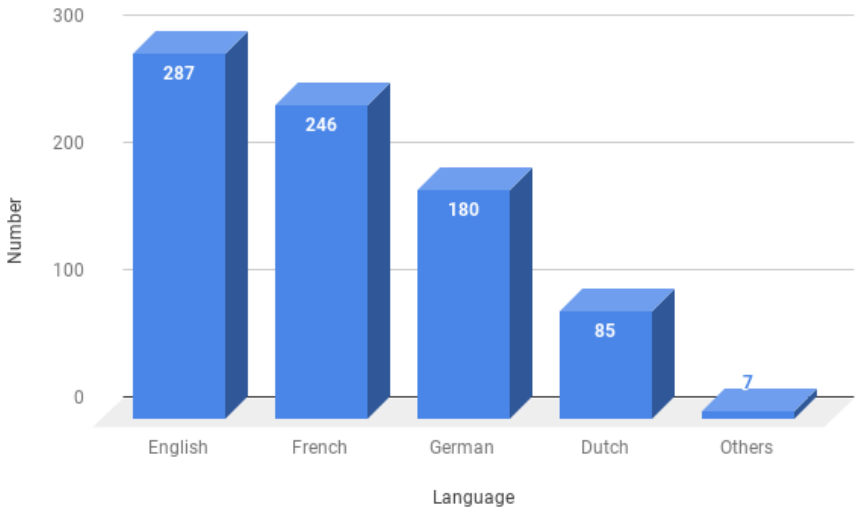


Figure 4.1: Distribution of job postings by language in the dataset

As mentioned before, each job posting is composed of open text fields and fields where recruiters can choose from a limited number of options. We name these fields as **unstructured** (UF) and **structured** fields (SF), respectively. Each field that is relevant for the annotation process is further detailed in Appendix A, containing its brief description, whether it is mandatory, its possible values and if it is important for the annotation task. Understanding a job posting's fields is required in order to comprehend the following sections. A fictitious example of a job posting can be found in Section A.2.

4.2 Contradictions and ambiguities areas

It is possible to categorize ambiguities and contradictions by the areas they are related to. These areas depend on the fields of a job posting, and its number may vary if more fields are added to job postings in the future. We can distinguish the following areas:

- **Education** - if the ambiguity or contradiction is related with education. For example, if there is a mismatch between the diplomas specified in the SF (*minReqEducation)* and the UF;
- **Language** - if the ambiguity or contradiction is related to the languages specified in different fields. For example, in one field the recruiter specifies a language as mandatory, while in the other field this language is specified as optional. The recruiter can specify the required and optional languages in the

UF or in the respective SF (*languages_number.code, languages_number.speak, languages_number.write, languages_number.read, languages_number.exclusionFactor, languages.mandatory, optional_languages_number.code, optional_languages_number.speak, optional_languages_number.write and optional_languages_number.read*);

- **Experience** - if the contradiction or ambiguity is related to the experience a candidate must have. For example, some recruiters ask for little experience in one field and, in another field, they ask for the opposite thing. The recruiter can specify the required experience in the *reqExperience* field.

4.3 Ambiguities

We consider an expression as ambiguous if it has more than one possible interpretation and if its meaning is unclear or causes uncertainty to the reader. Ambiguities can occur in an UF or between a SF and an UF.

An ambiguity can be distinguished by its type or by the area in which it occurs.

4.3.1 Types

In Chapter 2, we present the possible types of ambiguities that can occur in natural language. However, not all of these types might be relevant depending on the task one has in hands. Because of that, we first decided to look at a significant number of random job postings to see which of these types stood out and which did not. We noticed that the most significant ambiguities are a result of the poor structure of a sentence and the vagueness present in some expressions or sentences. Hence, we decided to distinguish ambiguities in three different categories:

- **Syntactic Ambiguity**
- **Vagueness**
- **Other** - A generic category for all the ambiguities that are a result of any other type of ambiguity not covered be the two previous ones.

Some of the syntactic ambiguities encountered in job postings occur when specifying which languages are mandatory in the UF. Let us consider the following example:

- In the *info.en.jobRequirements.mustHave* column, the recruiter asks for a candidate proficient in **Dutch and French or English**.

Looking at this sentence, it is not certain whether the recruiter wants a candidate that speaks *Dutch and (French or English)* or a candidate proficient in *(Dutch and French) or English*, as both interpretations are plausible. Analyzing the structured fields could help us understand the meaning of this sentence if it were not for the fact that *skeeled*'s job posting form does not allow

the recruiter to specify the second case. For that same reason, we decided to not consider syntactic ambiguities that may occur when specifying the required languages.

Regarding the vagueness type, if an expression is vague but can be complemented by a SF, then we do not mark such expression as dubious. For example, if the recruiter asks for a candidate with *little experience*, although *little* can be considered a vague term, this word may be complemented and quantified in the *reqExperience* field.

4.3.2 Examples

We enumerate some of the many ambiguities that can occur in the dataset fields:

4.3.2.1 Syntactic Ambiguities

1. In the *info.en.jobRequirements.mustHave* column, the recruiter asks for **an English history expert**.

   This expression can be read as *We are looking for an (English history) expert* or *We are looking for an English (history expert)*

4.3.2.2 Vagueness

1. *info.en.jobRequirements.mustHave*: "Already hold the **correct permit** to accept an internship in Luxembourg"

   This information is ambiguous as it is unclear what the correct permit is. Is it the necessary documents for working in Luxembourg or for doing an internship?

2. *info.en.jobRequirements.mustHave*: "Knowledge of the current languages of Luxembourg"

   This sentence is vague as it is not defined which languages the recruiter effectively refers to.

4.3.2.3 Other Ambiguities

1. *info.en.jobRequirements.niceToHave*: "Any previous **experiences related to the internship description** are a plus"

   reqExperience: **1-10**

   This information is ambiguous as it can have several interpretations. In the field *info.en.jobRequirements.niceToHave* the recruiter says experience related to the internship description is a plus. However, is the *1-10* years of *reqExperience* specified related to the internship description or is it the overall experience one candidate has? This experience is not specified in any other field.

4.4 Contradictions

To define contradictions, we use the definition provided by De Marneffe et al. [MRM08]. We define two expressions as contradictory if they are extremely unlikely to be true simultaneously. From what has been analyzed, a contradiction can occur between a structured field and an unstructured field, between two different unstructured fields, and even within the same unstructured field. A contradiction can be divided by its type or by the area (Section 4.2) in which it occurs.

4.4.1 Types

For the reasons mentioned in Section 4.3.1, we distinguish contradictions in the following categories:

- **Numeric mismatch** - Numeric mismatches between quantities, numbers, and dates.
- **World-knowledge** - Discrepancies based on world knowledge, i.e., the non-linguistic information that helps a reader or listener interpret the meanings of words and sentences. The detection of such contradictions require external knowledge.
- **Lexical** - These contradictions occur when recruiters ask for different competence levels (e.g.: education, language, domain) in the same job posting.
- **Language-not-specified** - Contradictions that occur when the recruiters ask for a mandatory language in the unstructured fields but do not specify the language in the structured fields.
- **Other** - These contradictions encompass all the other types of contradictions that are not previously defined.

4.4.2 Examples

We enumerate some of the contradictions that might be found in the dataset fields:

4.4.2.1 Numeric Contradictions

1. *languages.mandatory*: **3**

   info.en.jobRequirements.mustHave: Proficient in **Dutch/French & English**

   These fields are contradictory because three languages are mandatory although the expression can be interpreted as *Dutch OR (French AND English)* - one or two languages are required, respectively - or *(Dutch OR French) AND English* - two languages are required. Either way, there is not a case where the three are required.

2. *info.en.jobRequirements.mustHave*: "The candidate must have **more than three years of experience**."

   reqExperience: **0-3**

These two fields are contradictory, as the candidate must have between 0 and 3 years of experience in one field and more than 3 years in the other field.

4.4.2.2 World-knowledge Contradictions

1. *info.en.title*: *Bus Driver*

   requireDrivingLicense: *FALSE*

   These fields are contradictory since the recruiter is searching for a bus driver that does not need to have a driving license.

2. *info.en.jobRequirements.mustHave*: We hire **engineers and architects**.

   minReqEducation: **High School**

   These fields are contradictory because engineers and architectures have higher degrees like *Bachelor* or *Master*. This is a world-knowledge contradiction since one needs to know that to be an engineer/architect one needs a higher education degree.

4.4.2.3 Lexical Contradictions

1. *info.en.jobRequirements.mustHave*: "A **Master degree** is required".

   minReqEducation: "**Bachelor**"

   The recruiter asks for a candidate with a Master degree in one field and then asks for a Bachelor degree in another field.

2. *info.en.jobRequirements.mustHave*: "**High level** in spoken and written English & Portuguese".

   languages_1.write: "**B1**"

   The recruiter asks for a candidate with a high level of written English but then asks for *B1*.

   We attribute different language levels to different expressions as it is possible to check in Table 4.1. This table was based on the language levels according to the *Common European Framework of Reference for Languages* (Table 4.2).

Expression	Example	Levels
Basic	*Basic knowledge of ... / Basic level of ...*	A1 - A2
Good	*Good knowledge of ... / Good level of ...*	B1 - C2
High	*High knowledge of ... / High level of ...*	B2 - C2
Fluency	*Fluency in ... / Fluent in ...*	B2 - C2
Proficient	*Proficient in ... / Proficient knowledge of ...*	C1 - C2

Table 4.1: Attribution of language levels according to different expressions

Level	Description
A1	Beginner
A2	Elementary
B1	Intermediate
B2	Upper Intermediate
C1	Advanced
C2	Mastery or Proficiency

Table 4.2: Language levels according to the *Common European Framework of Reference for Languages*

3. *info.en.jobRequirements.mustHave*: "High knowledge in **systems administration**"

   fieldsOfStudy: **philosophy**

   These two fields are contradictory, as the fields *system administration* and *philosophy* are not related.

4.4.2.4 Language-Not-Specified Contradictions

1. *info.en.jobRequirements.mustHave*: "Good knowledge of English and French."
   languages_1.code: en

   The field *info.en.jobRequirements.mustHave* is contradictory because both English and French are mandatory languages, however, only English is specified in the structured fields.

4.4.2.5 Other Contradictions

1. *requireWorkPermit*: **TRUE**

   workingPermitIsKiller: **FALSE**

   info.en.jobRequirements.mustHave: "Already **hold the work permit** to accept an internship in Luxembourg."

   These fields are contradictory because in the *info.en.jobRequirements.mustHave* and *requireWorkPermit* the recruiter says that a work permit is required for the job, but then in *workingPermitIsKiller* says it is not an exclusion factor.

2. *info.en.jobRequirements.mustHave*: "Knowledge of Portuguese is optional."

   languages_1.code: **pt**

These fields are contradictory because, since Portuguese is an optional language, it should be specified in the optional languages field instead of the mandatory languages field.

4.5 Annotating the Dataset

After analyzing the dataset fields and choosing the relevant types of inconsistencies, the next step consisted of deciding on which tool to use, which languages to annotate, the number of annotators required, and the minimum number of annotations needed to validate the developed models later.

4.5.1 Annotation tool

Regarding the tool, we thought of different solutions: either use one markup language, such as XML or HTML or use a tool that would be more intuitive for other people with little to no knowledge of these languages. Because not all people that work in *Skeeled* and that volunteered to help us build the dataset have knowledge of the markup languages previously mentioned, we opted for the last option. We found a tool specifically designed for the task of annotating, named *brat*[1].

brat is a web-based tool for text annotation. In other words, it is a tool for adding notes to existing documents. There are many advantages in using *brat*, such as:

- It offers a comprehensive and **intuitive visualization** of the underlying annotation;
- Annotating is easily done and in an **intuitive way**. By simply using the mouse (selecting, dragging or double-clicking), it is possible to mark a span of text;
- It allows text annotation in **any language**. This is an important feature since Skeeled's software is multilingual;
- It allows **comparison** between multiple sets of annotations for the same documents, which includes an automatic comparison to identify and mark differences and side-by-side visualization;
- It allows **real-time collaboration**.

brat contains some configuration files that can be changed to annotate different entities and relations, in our case, different types of inconsistencies and their relationships, respectively. To annotate a file, *brat* requires two files: a *.txt* file that contains the text of the job posting that is going to be annotated, and a *.ann* file – the file where the annotations are saved. Table 4.3 shows the fields that compose an annotation.

Additionaly, Figure 4.2 shows a visual representation of a lexical contradiction that was annotated between the span text *minimal C1* and the structured fields *languages_1.write*, *languages_1.speak* and *languages_1.read*. The annotations made were added to the respective *.ann* file and their representation can be seen in Listing 1.

[1] http://brat.nlplab.org/

Table 4.3: Description of the fields of an annotation

Name	Description	Can be null
Type	Type of inconsistency; either contradiction or ambiguity.	FALSE
Subtype	Subtype of the inconsistency. Its values vary according to the inconsistency	FALSE
Annotator	Name of the annotator.	FALSE
Offset_begin	Position of the first character of the annotated span text in the job posting	TRUE
Offset_end	Position of the last character of the anotated span text in the job posting	TRUE
Field	Name of the field that contains the annotated span text	FALSE
Text	Span text that is being annotated	FALSE
Contradictory_field	Name of the field that is contradictory with the annotated span text.	TRUE
Contradictory_text	Span text that is contradictory with the annotated span text.	TRUE

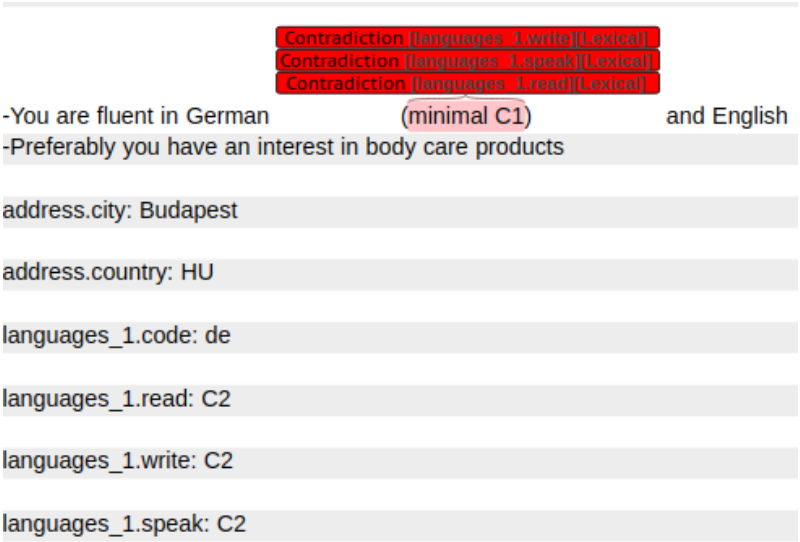


Figure 4.2: Example of an annotation in *brat*

```
T1      Contradiction 2202 2212 minimal C1
A1      Contradiction-with T1 languages_1.read
T2      Contradiction 2202 2212 minimal C1
A2      Contradiction-with T2 languages_1.speak
T3      Contradiction 2202 2212 minimal C1
A3      Contradiction-with T3 languages_1.write
A4      Contradiction-type T3 Lexical
A5      Contradiction-type T2 Lexical
A6      Contradiction-type T1 Lexical
```

Listing 1: Example of annotations saved in a *.ann* file

An *ann* file can have several lines. Each line starts with an identifier that expresses the type of fields defined:

- If the line starts with a **T**, then it contains the annotation's ID, type, beginning and ending offsets, and finally the piece of text that contains either a contradiction or ambiguity.
- If the line starts with an **A**, then it contains an identifier, the type of relation[2], the annotation ID that it refers to, and either it contains the subtype of inconsistency or the field it is contradictory/ambiguous with, depending on the type of relation.
- If the line starts with a **R**, it contains the relation identifier, the type of relation and the IDs of two annotations. This annotations usually refer to two different spans of text that are either contradictory or ambiguous with one another.

4.5.2 Annotation Statistics

Looking at Figure 4.1, it is possible to see that the languages that have a greater impact on the platform currently are English, French, and German. Due to this reason, we opted to annotate job postings written in these languages.

The task of annotating a job posting is not an easy one. It is important that the annotators are familiar not only with the annotation tool but also with the context of the job posting and types of inconsistencies that might occur. The lack of knowledge may lead to inconsistent and inaccurate annotations. Therefore, we decided it was important to deal with this problem beforehand and we wrote an annotation manual (see Appendix B) that provides guidelines that each annotator must follow. Besides providing key concepts about contradictions and ambiguities, this manual explains how to use *brat* to annotate such inconsistencies. Additionally, we gave training to all the annotators.

With the help of some of *Skeeled*'s workers, we then annotated 80 job postings in English, 80 job postings in French, and 40 job postings in German. Each job posting was annotated by 3 different annotators, except for German, since we only had 1 annotator. In total, we had 8 different annotators for the three languages. However, some people annotated in two different languages. This information is summarized in Table 4.4.

Table 4.4: Summary of the distribution of annotators and annotations per language

Language	English	French	German	
Nº Annotations	80	80	40	200
Nº Annotators	5	5	1	8
Annotators/job posting	3	3	1	-

We decided to have 3 different annotators per job posting due to possible disagreements between them when annotating a contradiction or ambiguity. This way, in case of discrepancy, the

[2]It can be one of the following values: *contradiction-with*, *contradiction-type*, *ambiguity-with*, *ambiguity-type*

majority voting decides whether there is a contradiction/ambiguity or not. In cases where all annotators disagree from one another, such annotations are discarded.

4.6 Inter-Annotator Agreement

As mentioned before, the decision of having 3 different annotators per job posting was made so that, in case of doubt or disagreement between annotators, one could use the majority voting to create the *Gold Standard Corpus* [3].

Nevertheless, it is important to guarantee that the annotated data being used is reliable and trustworthy. Otherwise, this data cannot be used to evaluate the performance of our developed model. In other words, it is essential to calculate the agreement between the different annotators to assure the quality and usability of the labeled dataset. Moreover, if there is agreement between annotators, we can assume that the annotators understood the provided guidelines and, therefore, the task is feasible, i.e., it is well defined.

To do that, we decided to apply a measure widely used named *Krippendorff*'s alpha to calculate the Inter-annotator agreement (IAA). An explanation of this metric is detailed in Section 2.5.2.2. We chose this metric because it is a complete measure and it fulfills our requirements: it is applicable for more than two annotators and for handling the data we are working with. However, there are other metrics for more than 2 annotators that could also be applied.

We only applied this metric to French and English annotations, since there is only one annotator for German. This evaluation was performed at different levels: at phrase level, i.e., whether the annotators identify the same sentences as problematic sentences; at type level, i.e., if the annotators agree with the type of inconsistency (contradiction or ambiguity) chosen for the identified sentence; and at subtype level, i.e., if the coders agree with the subtype label chosen.

Table 4.5: Inter-annotator agreement values obtained using Krippendorff's alpha

Level	Sentence	Type	Subtype
Krippendorff	0.13	0.12	0.09

Looking at Table 4.5, there are some conclusions that can be drawn. As expected, we observe that there is more agreement when looking at the sentence level, as the annotators only need to agree whether the sentence contains inconsistencies or not. As this evaluation gets more specific, it is possible to see that the values for agreement start decreasing. It is also possible to observe that even the highest value (0.13) does not fall under the minimum value required to state that there is agreement between the annotators (0.67), as stated in Chapter 2. Thus, we can conclude that our annotations are not reliable. Hence, the data should not be used to validate the developed models.

These low values obtained in Inter-annotator agreement can be explained by the following factors:

[3]The Gold Standard Corpus refers to trustworthy data that can be used to train and/or test algorithms/models

- Since our annotators are from different areas inside the company, they have different perceptions regarding the way they see a job posting. Annotators that review job postings daily have different viewpoints from people who have little experience analyzing such documents;

- The annotators are not always on the best of their performances (e.g.: tiredness, lack of concentration);

- Notwithstanding, although formation was given and guidelines were provided, we noticed that some annotators did not always follow those rules.

Since it was not possible to use the annotated corpus to obtain the Gold Standard corpus, we decided to try another approach. We gathered three experts in analyzing job postings and manually reviewed the previously annotated job postings written in French and in English. We decided not to build a Gold Standard Corpus for German since only one person annotated the 40 German job postings, which could lead to some biased annotations. The choice of having three different people creating the Gold Standard corpus was made because of dubious annotations. In other words, in case there is disagreement between the experts, it is possible to tie up these cases.

This new approach of generating the Gold Standard Corpus is costly and time-consuming. Since the available annotators are workers from *skeeled* that have several distinctive responsibilities, different schedules, and different competencies, it is not easy to gather three people with different responsibilities to focus exclusively on the annotation process. Besides, fewer annotators mean a slower annotation process that causes a decreasing of the number of annotations.

The distribution of the types of contradiction and types of ambiguities resulted from the new Gold Standard corpus can be seen in Table 4.6 and Table 4.7, respectively. On the one hand, we can observe that there are 84 and 103 contradictions for the job postings written in English and French, respectively. On the other hand, the job postings contain 16 ambiguities in English and 11 ambiguities in French. From the results obtained, we can infer that the number of ambiguities (27) present in job postings is by far less significant than the number of contradictions (187). Besides, we can also conclude that the most meaningful types of contradictions that occur in our dataset result from numeric and lexical mismatches and that most ambiguities result from a lack of specification and usage of vague terms.

Table 4.6: Summary of the distribution of contradictions by type and by language in the Gold Standard

	Numeric	**World-knowledge**	**Lexical**	**Language-not-specified**	**Other**	
English	32	1	36	12	3	84
French	24	5	50	14	10	103
	56	6	86	26	13	187

Table 4.7: Summary of the distribution of ambiguities by type and by language in the Gold Standard

	Syntactic Ambiguity	**Vagueness**	**Other**	
English	1	15	-	16
French	1	8	2	11
	2	23	2	27

4.7 Summary

As it was explained, building a Gold Standard Corpus is a time-consuming and demanding task. Besides, there are some factors that delay this task:

- *skeeled* is a small company. For that same reason, it was not easy to find as many annotators as we intended to at the beginning. The **lack of annotators** influenced the rhythm of the annotating process and, therefore, the number of job postings annotated;
- the number of annotators throughout the annotation process **was not constant** due to the (un)availability of some annotators. Besides, some people left the company, so this number diminished over time;
- during the annotation process, we noticed the need to **change the guidelines** as some types of inconsistencies were not being covered. This constant changing of guidelines implied reviewing the previously annotated documents;
- the **lack of IAA** led to the necessity of rebuilding the Gold Standard corpus.

We can conclude that the annotation process is a complex and continuous task that needs time and resources. As we encountered some difficulties throughout the whole process, the annotation of job postings took a slower pace than expected. As a result, the lack of annotated documents influenced the approach mentioned in Chapter 5.

Chapter 5

Detecting Contradictions

This Chapter provides an exhaustive explanation of the approach taken to automatically detect contradictions, the difficulties that arose when developing the contradiction models and gives an overview of the technologies used for each task.

5.1 Overall method

A job posting is composed of structured (SF) and unstructured fields (UF). In the latter, the data is already standardized so its interpretation is pretty straightforward when compared with the unstructured fields. However, the complexity of analyzing open-text fields and, therefore, detect contradictions and ambiguities, lies in different reasons:

- The hardship of understanding natural language, because of its inherent ambiguity that results in several possible interpretations;
- The diversity of the writing styles due to the unique writing of each recruiter;
- Language errors might occur as a result of misspelled words or grammatical issues.

Although ambiguities and contradictions may occur in the same job posting, its occurrence is independent of one another. Due to their different nature, detecting contradictions and ambiguities requires two different solutions.

We started by tackling the problem of contradictions as the number of contradictions (187) exceeds the occurrences of ambiguities (27), as it is possible to see in Table 4.6 and Table 4.7. In fact, for this same reason, we decided to postpone the resolution of ambiguities as future work.

Once again, we found the problem of contradiction identification a complex one that should be divided into smaller problems. Thus, we decided to approach this problem by distinguishing the detection of contradictions by area (Section 4.2). For each area, we developed a model to detect

contradictions related to languages and experience, in different idioms. We frame each of these problems as a classification problem.

Initially, we thought of developing models based on Machine Learning (ML) techniques. We would tackle such a problem by extracting features from the job postings and feed them as input to train the models. At last, we would test the outputs given by our trained models against a testing set to see how well the models were performing. Nonetheless, the delay of building the Gold Standard (GS) corpus mentioned in Chapter 4 and, therefore, the lack of annotated data, made it difficult to implement such a solution. Because of that, we decided to take a new approach: the creation of rule-based models. After all, models based on ML are models whose rules are induced automatically.

Thereby, our overall approach, for each of the models, consists of identifying the language of the job posting, preprocessing the data and understanding which sentences are related to the different areas in which the inconsistencies may occur. To identify the language of a job posting, we use a library named *langdetect*[1] that, given a sentence or text, returns the language that most closely resembles the text language. Alternatively, this library can return a list of possible languages and respective probabilities of matching the text language.

After the identification of the sentences that are related to the models (either experience-related or language-related), we apply some NLP techniques to those same sentences and we focus on the extraction of relevant features from them. Strictly speaking, the extraction of features varies from model to model. For example, extracting features for the *experience* model (focused on numbers), detailed in Section 5.4, is less complex than extracting words that characterize languages in the *language* model, further discussed in Section 5.3.

Upon obtaining the relevant features of the targeted sentences, our approach comprises the creation of a set of rules that define the model, whose complexity depends once again on the model. Finally, we use several metrics to test the outputs generated by the models against the outputs obtained in the GS corpus and refine these same results by updating the rules. Our approach is summarized in Figure 5.1.

As mentioned in Chapter 4, the GS corpus was only built for French and English. Hence, our models only detect contradictions in these two languages, although they can be later adapted to work with other languages. We opted to develop these models in *Python* since there is a lot of support and available tools to perform NLP tasks.

5.2 Preprocessing

Before starting to target the problem of finding contradictions, the data must be prepared, regardless of the model. One of the challenges that our dataset represents is that it is composed of a set of job postings created by different people. Contrary to the SF, where data is uniform, the UF have unique writing styles. Hence, some factors must be taken into account:

[1] https://github.com/Mimino666/langdetect

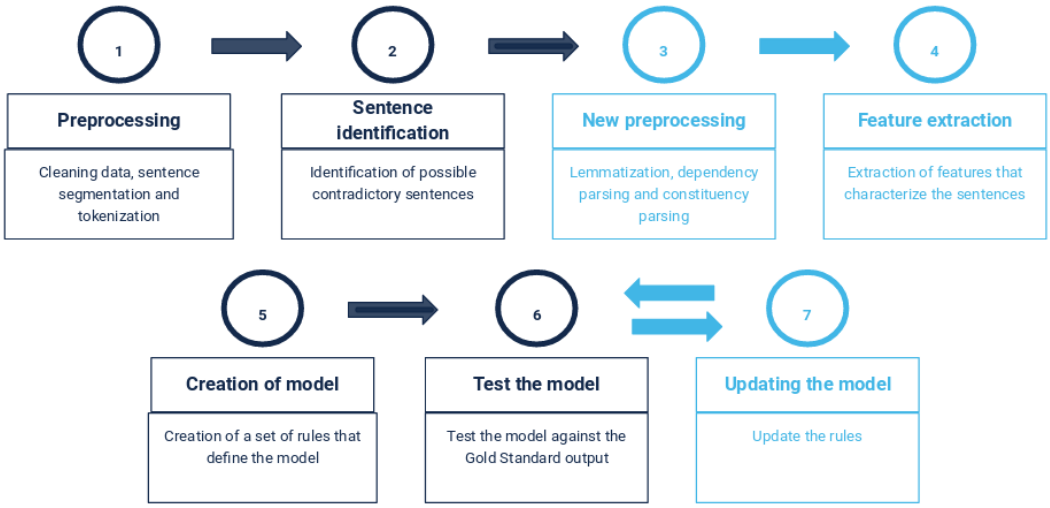


Figure 5.1: Summary of the approach to detect contradictions

- The data contains noise, such as language errors (e.g.: misuse of punctuation, missing punctuation, misspelled words);
- The data has different encodings;
- The data is written in markup languages, such as HTML. Sometimes, the tags are inadequate;
- The data contains some information that is not relevant;
- The data is not consistent.

The existence of such problems lead to the urge to clean the data so it can be used in later stages. For that, we created a function that iterates over the fields of a job posting and cleans the UF. When cleaning the data, the following steps are fulfilled:

- **Removal of unnecessary fields**. A job posting is composed of several fields. However, in order to detect contradictions, not all the fields need to be analyzed (such as the job posting's ID and the recruiter company's ID);
- **Replacement or removal of special symbols**. Essentially, we replace "`;`" or more than one "`.`" in a row with "`.`", we remove "–" from the beginning of sentences and we remove some special characters that are not numeric nor alphabetic characters. However, we keep the following special characters "`&,.?!:=<>\/ '"()+-`" as they are often relevant to understand the meaning of a sentence. We replace the "`;`" as it is a symbol that can separate two simple sentences in a complex sentence (e.g.: *Excellent presentation and communication skills; strong communicator whether face to face, phone or by email*) or list items. We

do this, because the sentence segmentation tool that is being used (*StanfordCoreNLP*) cannot do this on its own. We remove "–" form the beginning of sentences because it influences the relationships obtained through the dependency parser.

- **Addition of missing punctuation characters**. We noticed that some recruiters separate sentences with a "`\n`" or with HTML tags, such as "`</p>`" or "`</li>`" (e.g.: (e.g.: *You should have: \n- Good knowledge of French \n- Additionally, you know German*)). In such cases, we replace such expressions with a "`.`". However, we also noticed that such characters sometimes are incorrectly inserted in the middle of sentences (e.g.: *Good \n knowledge of French*). In these cases, we just remove the HTML tag (following bullet);

- **Replacement or removal of HTML tags**;

- **Substitution of HTML entities**. Table 5.1 summarizes the entities that are replaced, their description and replacement characters;

Table 5.1: HTML Entities and respective descriptions and replacement expressions.

Entity name	**Description**	**Replacement**
	non-breaking space	
<	less than	<
>	greater than	>
&	ampersand	&
&quot	double quotation mark	"
'	apostrophe	'
£	pound	pound
¥	yen	yen
€	euro	euro

- **Removal of non-ASCII characters**;

- **Replacement of some fixed expressions** that create some challenges in further stages. Expressions that represent *language levels*, such as *A1, A2, B1, B2, C1* and *C2*, are replaced in the job postings written in French. This is necessary because the current version of the dependency parser we are using for French does not recognize these levels as single words and divides them in two different tokens, which is problematic when extracting the modifiers of a language (explained in Section 5.3.3); for example, "A1" is divided in "A" and "1".

These requirements were accomplished by creating some rules that use regular expressions. A practical example of cleansing a job offer can be seen in Listing 2 and Listing 3.

After cleaning the raw data, it is important to divide the text into smaller units, such as sentences and, subsequently, tokens. In Chapter 2, we mentioned some tools that can be used to perform some NLP tasks. Table 2.2 summarizes some of the current tools used to perform segmentation tasks, such as sentence segmentation and tokenization. Although we did not perform an exhaustive comparison between these tools, we opted to choose the *StanfordCoreNLP* tool as

Table 5.2: Language levels and respective replacement expressions

Expression	Replacement
A1	A-un
A2	A-deux
B1	B-un
B2	B-deux
C1	C-un
C2	C-deux

```
{
    "info.en.jobRequirements.mustHave": "<ul><li>Some bullet point</li>
                                         <li>A second bullet point</li></ul>",
    "info.en.jobRequirements.niceToHave": "<ul><li>One condition &
                                           another condition</li></ul>",
    "info.en.title": "Title of the job",
    "info.en.description": "Description of the job",
    "info.en.responsibilities": "Responsibilities of the job",
    "languages_1.speak": "C1",
    "languages_1.read": "C1",
    "languages_1.write": "C1",
    "languages_1.exclusionFactor": "C1"
}
```

Listing 2: Example of some job posting's fields before the cleaning step

```
{
    "info.en.jobRequirements.mustHave": "Some bullet point.
                                         A second bullet point.",
    "info.en.jobRequirements.niceToHave": "One condition & another condition.",
    "info.en.title": "Title of the job",
    "info.en.description": "Description of the job",
    "info.en.responsibilities": "Responsibilities of the job",
    "languages_1.speak": "C1",
    "languages_1.read": "C1",
    "languages_1.write": "C1",
    "languages_1.exclusionFactor": "C1"
}
```

Listing 3: Example of some job posting's fields after the cleaning step

it provides all the features we need for this and future tasks. Moreover, this toolkit has good documentation, it is continuously being improved and updated and its results are generally good. Despite this tool being written in *Java*, there are some Python wrappers that can be used[2].

Figure 5.2 summarizes the overall process of denoising and segmenting the data.

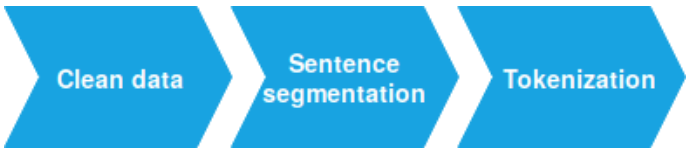


Figure 5.2: Preprocessing

Once the preprocessing stage is over, the data is ready to be used by the model.

5.3 Languages Model

Although the languages and experience models follow the same structure, their implementations differ from one another. The languages model focuses on detecting contradictions that may occur when specifying languages, due to the misuse of a job posting's field, or to the lack of coherence when specifying which languages are mandatory and which languages are not, or due to a mismatch of the levels of a language.

As it is possible to observe in Algorithm 1, first, we detect the language-related sentences (line 3). For each sentence, we compare the words that modify each specified language with the language levels asked in the SF, i.e., *languages_number.speak, languages_number.write* and *languages_number.read* fields[3]. Then, we compare the mandatory and optional languages the recruiter specifies in the multiple UF with the required and optional languages specified in the SF. To do so, we compare the languages in the UF with the *languages.mandatory* and *languages_number.exclusionFactor* fields. This way, we obtain the semantic contradictions and the numeric contradictions (lines 5-6), respectively. The semantic contradictions refer to the contradictions that occur when two words do not have the same meaning, while numeric contradictions are related to numeric mismatches of the required languages specified in the UF and in the SF.

The number of comparisons done increases with the number of language-related sentences, the detected languages, and the words used to describe such languages. If a language is specified in the SF and in one of the UF sentences and can be described by one word (the modifier), then, the model only needs to compare that one modifier with the levels specified in the SF. That is, the model will compare the modifier with the language levels specified in the *write, read* and *speak* fields (3 combinations). However, if a language is described by two different modifiers, then, the model needs to compare each modifier with the language levels in the SF (in this case, the

[2]In https://stanfordnlp.github.io/CoreNLP/other-languages.html, under the Python section, there are some packages that implement *Stanford CoreNLP* in Python.

[3]These fields are detailed in Chapter A.

number of combinations doubles to 6). A further explanation of the number of combinations will be detailed in Section 5.3.5.

Algorithm 1: `Get_Language_Contradictions` pseudo-code

Data: key: name of field in variable fields
fields: fields of the job posting
n_total_sentences: total number of sentences
sentences_related_to_languages: sentences related to languages
job_language: language in which the job is written
Input: *job_posting*: the job posting to detect contradictions from
Output: *contradictions*: list containing the contradictions detected in the job posting

```
1  Function Get_Language_Contradictions(job_posting):
2      contradictions ← EMPTY LIST;
3      sentences ← Get_Sentences_Related_To_Languages(job_posting);
4      for each s in sentences do
5          semantic_contradictions ← Get_Semantic_Contradictions(s);
6          numeric_contradictions ← Get_Numeric_Contradictions(s);
7          contradictions ← contradictions + semantic_contradictions +
             numeric_contradictions;
8      end
9      return contradictions
```

5.3.1 Identification of the language-related sentences

The first step towards the implementation of the model is based on the identification of sentences that are related to languages as these are the sentences that might be contradictory with other fractions of text.

First, our function obtains all the language names and codes (lines 2-3 of Algorithm 2) listed in *ISO 639-1*[4]. We use a Python's library named *pycountry*[5] to obtain the standardized codes and we make use of a library named *Langcodes*[6] to get the respective language names. This library provides language names in different languages, which is useful as we are dealing with a multilingual context and need to find language-related sentences in different languages. For example, for the code "en", we can get either "English" or "Anglais". Even though our model's current implementation only detects contradictions in French and English, this feature can be used with other languages as well.

Then, we iterate over the UF of the job posting (lines 4-7) and, for each of the UF sentences, we check if that sentence contains any language name or language code (lines 10-11). If no languages are found, the current sentence is not considered as language-related (lines 12-14). Else, we use a filter to discard some sentences that are not related to languages (lines 15-19). If the sentence

[4]ISO 639 is a standardized nomenclature used to classify languages. ISO stands for *International Organization for Standardization*. A list of the available language codes can be found in https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes

[5]https://pypi.org/project/pycountry/

[6]https://github.com/LuminosoInsight/langcodes

Algorithm 2: Get sentences related to languages

Input: key: name of the field in variable field (initial value: None),
field: an unstructured field of the job posting (initial value: unstructured fields of the job posting)
Globals: sentences_related_to_languages: sentences related to languages,
job_language: language in which the job is written

```
1  Function Get_Sentences(key, fields):
2      languages_group ← get_languages_regex_group(job_language);
3      languages_codes ← get_codes_regex_group(job_language);
4      if field is a dictionary then
5          for field_key, field_value in field.items() do
6              Get_Sentences(field_key, field_value)
7          end
8      else
9          for stanford_sentence in fields.sentence do
10             sentence ← stanford_sentence.text;
11             languages_found_u_f ← regex.findall(languages_group, sentence) +
                regex.findall(languages_codes, sentence) ;          // searches for
                languages_group in sentence and for languages_codes in
                sentence, respectively
12             if length(languages_found_u_f) == 0 then
13                 continue
14             end
15             expression ← filter(sentence);
16             non_related_sentences_with_languages ←
                regex.findall(expression, sentence);
17             if length(non_related_sentences_with_languages > 0 then
18                 continue
19             end
20             new_stanford_sentence ← preprocessing(sentence, options =
                ['parse','depparse']);
21             sentence_dict ← create_sentence_dictionary(new_stanford_sentence);
22             sentences_related_to_languages ←
                sentences_related_to_languages + sentence_dict;
23         end
24     end
```

gets through the filter, it is processed again and then it is added to a set of all the language-related sentences of the job posting (lines 20-22), along with the respective field name, the languages found, and its delimiters.

The filter in line 15 was applied because not all sentences that contain language names or codes are language-related sentences. On the one hand, in some cases, language words may be used to characterize something. For example, some recruiters might use languages as adjectives while describing their company's offices (e.g.: "The Luxembourgish office is (...)"). The inaccurate identification of language-related sentences was leading to an increased number of false positives. To solve this, we thought of using the languages' POS tags to identify the languages that were being targeted as adjectives. However, the POS tagger used sometimes predicts incorrect tags. Instead, to solve this problem, if the algorithm identifies languages in a sentence, we decided to apply some basic regular expressions to guarantee that the language is not being used as an adjective (lines 15-19). For example, if a language is related to certain words such as *office, teacher, labour code*, among others, we assume that this language is being used to describe such words. Despite the lack of sophistication of this method, it serves its purpose. However, this is a problem that may require more attention in the future.

On the other hand, there are some words present in job postings that can represent language codes, words or acronyms. For example, the word "et" can represent the language code for Estonian or the conjunction "and" in French. In these cases, our model was detecting sentences containing such words as language-related, even if they referred to conjunctions and not language codes. To solve this, we decided to identify sentences as language-related only if they contain more than one language code in a row (e.g.: *knowledge of NL-EN* is considered as a language-related sentence, while *knowledge of NL* is not).

After improving the identification of language-related sentences, we noticed a common pattern that was causing problems in the following stages. Writing natural language is an error-prone task, especially if the recruiters are not careful while doing so. Therefore, some sentences had misspelled words that needed to be addressed. We opted to correct only the targeted sentences, as correcting the whole job posting would be a time and resource-consuming task that would not add additional value. For that, we used a Python's library named *pyspellchecker*[7] that, given a word, it returns the most probable result for the word, if it is misspelled. This library provides support for English, Spanish, German, French, and Portuguese.

5.3.2 Second preprocessing

When identifying contradictions in a language-related context, there are two major tasks that need to be performed: the correct identification of the modifiers of a language (detailed in Section 5.3.3) and the recognition of the optional and mandatory languages specified in the SF as well as in the UF (detailed in Section 5.3.4). Solving the first requires a further preprocessing as the identification of the relationships between words are needed. The latter implies a further syntactic analysis

[7] https://github.com/barrust/pyspellchecker

in which the syntactic components are recognized. To do so, we applied a second preprocessing on top of the one we had. More precisely, we used lemmatization, dependency parsing and constituency parsing provided by *StanfordCoreNLP*, respectively. *StanfordCoreNLP* does not support lemmatization for the French language. Because of that, we use the lemmatization given by *SpaCy* for this language.

Although there was no need to divide the preprocessing into two different steps, we thought this would be wise as the dependency parser is a complex and time-consuming parser. In fact, we tried both approaches and came to the conclusion that the performance results were significantly better when only applying the dependency parsing to the language-related sentences.

5.3.3 Extracting modifiers

The proper identification of the words that modify a language is not an easy task. First as mentioned in the previous section, we applied the dependency parser to each language-related sentence. We are using the EnhancedPlusPlusDependencies provided by the dependency parser instead of the BasicDependencies (Section 2.3.1.3). After, for each language in each sentence, we extracted the modifiers that characterize the language. Basically, the extraction function takes as input the sentence and the language and searches recursively for relations that modify that same language, such as name modifiers (nmod), adjectival modifiers (amod), among others[8].

For example, if a recruiter asks for a *Good knowledge of English*, then the modifier that should be targeted is the word *good*.

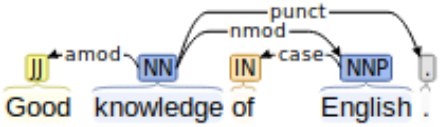


Figure 5.3: Dependency parsing for the sentence *Good knowledge of English.*

Figure 5.3 shows that the word *English* has two relations: a relation of the type *case* with the word *of* and a relation of the type *nmod* with the word *knowledge*. Subsequently, the word *knowledge* has a relation with the word *good*. In this case, by recursively calling the extraction function, the model obtains the word *good* as a modifier of *English*.

The previous example is a simple one but Figure 5.4 shows a more complex case. In cases like this, obtaining the right modifiers is not as straightforward as the first example. The sentence *Good knowledge of English (minimum level of B2)* has as modifiers *good* and *B2*.

When identifying the modifiers of a sentence several problems arose. These problems are related to the parser being used and the way sentences are written. We noticed that the more the complexity of the sentence, the more probable it is for the parser to incorrectly detect the relations between words. Nonetheless, the parser often fails to identify complex/composed sentences that are verbless. This happens because the parser being used was trained with well-written sentences

[8] A special thanks to Professor Rui Sousa Silva for clarifying some doubts we had related to these relationships.

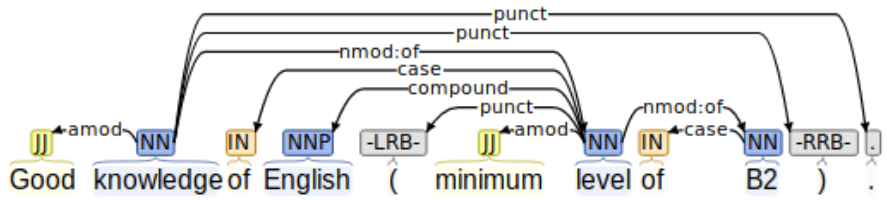


Figure 5.4: Dependency parsing for the sentence *Good knowledge of English.*

that contain a subject and a predicate. We also detected how the misuse or omission of punctuation such as "," might influence the results given by the parser.

For example, when we tested our parser with the following input *Good knowledge of English and basic level of French.*, we were expecting to obtain the modifier *good* for *English* and *basic* for French. Instead, we obtained both *good* and *basic* as modifiers for each of the words (Figure 5.5). However, when adding a "," or the verb *to be* before or after the conjunction *and*, respectively, we noticed an improvement of the quality of the relations (Figures 5.6 and 5.7, respectively). The position where the verb is added influences the results. If not added after the conjunction, it does not improve the relationships obtained through the dependency parser.

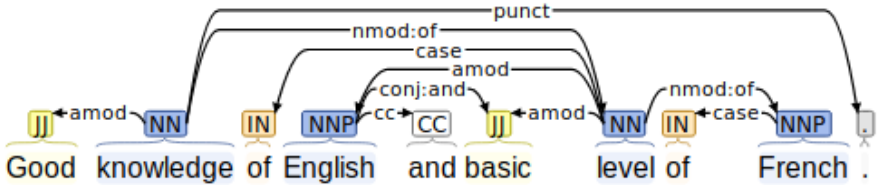


Figure 5.5: Dependency parsing for the sentence *Good knowledge of English and basic level of French.*

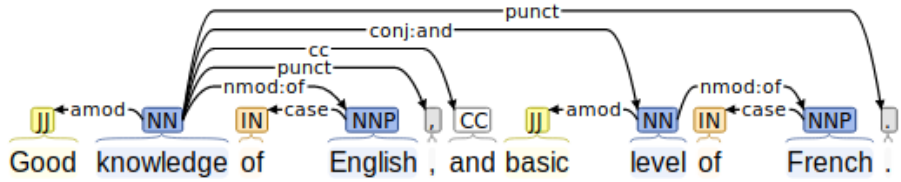


Figure 5.6: Dependency parsing for the sentence *Good knowledge of English and basic level of French.* after adding the missing punctuation

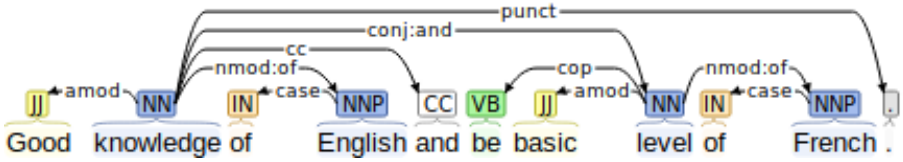


Figure 5.7: Dependency parsing for the sentence *Good knowledge of English and basic level of French.* after adding the verb *be*

5.3.4 Extracting the mandatory and optional languages

In our approach, we classify languages as mandatory or as optional. In the first, either a language or a subset of languages (or both) can be mandatory. For example, if the recruiter writes that the candidate must have a *good knowledge of French and Dutch or English* and that *knowing German is a plus*, then French is considered a mandatory language, German is considered an optional language, while Dutch and English are part of a set in which only a subset is mandatory, as only one of these languages is required.

On one hand, we need to classify the languages in the SF and UF. On the other hand, we need to compare both classifications and see if these differ from one another, ergo understand if there is a contradiction.

As explained in Chapter 4, there is a structured field in the job posting where the recruiter can specify the number of mandatory languages and, for each language in the SF, there is an exclusion factor that indicates whether the language is mandatory or not. We use the following nomenclature:

- **n-mandatory-languages**: Number of mandatory languages specified by the recruiter in the field *languages.mandatory*;
- **n-listed-languages**: Number of languages listed by the recruiter in the field *languages.list*;
- **n-languages-exclusion-factor**: Number of listed languages that have the exclusion factor set to true;
- **subset-mandatory-languages**: languages that belong to a set of languages whose subset is mandatory, i.e., only some of the languages specified in this set are mandatory, not all.

To detect which languages are mandatory in the SF, we iterate over the different listed languages and check the following conditions (the understanding of the following requires a deeper comprehension of *Skeeled*'s fields that can be found in Appendix A):

1. First, we check if *n-mandatory-languages* is equal to *n-listed-languages*. If this condition is met, all the languages specified in the SF are mandatory;
2. If the preceding condition does not occur, we check if the exclusion factor of that language is set to true. If this condition is met, then the language is mandatory;
3. If not, then we check if the *n-languages-exclusion-factor* is equal to *n-mandatory-languages*. If so, then the languages whose exclusion factors are set to true are mandatory, while the others are considered optional;
4. If the previous condition fails, then this means that *n-languages-exclusion-factor* is less than *n-mandatory-languages*, therefore, the languages whose exclusion factor is set to false belong to the *subset-mandatory-languages*.

Classifying languages in the UF differs from the previous implementation, as in this case, we need to analyze natural language. Besides, there are some factors that need to be considered. As we are analyzing different fields, one would expect that the recruiters use the fields correctly. For example, it would be expected that the recruiter specifies the mandatory languages in the *info.en.jobRequirements.mustHave* field and the optional languages in the *info.en.jobRequirements.niceToHave* field. However, it is quite frequent to find required and optional qualifications specified in the same field (usually, the *info.en.jobRequirements.mustHave* field). Because these cases are quite frequent, we do not consider the misuse of the UF as contradictions.

Our approach takes into account the aforementioned aspects. It consists of iterating over the language-related sentences previously identified and applying the following steps:

1. First, we check whether the field of the sentence is the 'niceToHave' field or if the sentence is optional. By optional, we mean whether the sentence contains any lemmas that represent optional words (*plus, optional, asset, advantageous, prefer, advantage, preferably*) that are related to the identified languages. If this condition applies, the languages of that sentence are marked as optional languages;

2. If the first condition fails, then we assume that the languages we are analyzing are a prerequisite. That way, we check if the number of languages in that same sentence is equal to one. If so, that language is marked as mandatory;

3. If the number of languages specified is greater than 1, then we make use of the constituency parser to determine which languages are mandatory and which languages belong to a set of languages whose subset is required. Our approach consists of going through each node of the parse tree and verify which words connect the languages. For example, in Figure 5.8, it is possible to see that the word connecting *English* and *French* is ***or***, and these two languages are connected to *Portuguese* by the word ***and***. Although this sentence can be ambiguous, as we discussed before, we are not considering syntactic ambiguities related to languages as there is a limitation in *skeeled*'s platform. Besides, the constituency parser disambiguates this sentence by choosing the most probable tree. In this case, the parse tree shows us that *Portuguese* is a mandatory language while *English* and *French* belong to the *subset-mandatory-languages*.

5.3.5 Rules for detecting contradictions

Once the modifiers are identified and the languages are classified as mandatory or optional, our method focuses on the creation of rules to detect contradictions. These rules were divided into two major groups: detecting semantic contradictions and detecting numeric contradictions. The first refers to all the contradictions that occur due to semantic differences, while the second points to the contradictions that result from different specifications of languages between the SF and the UF.

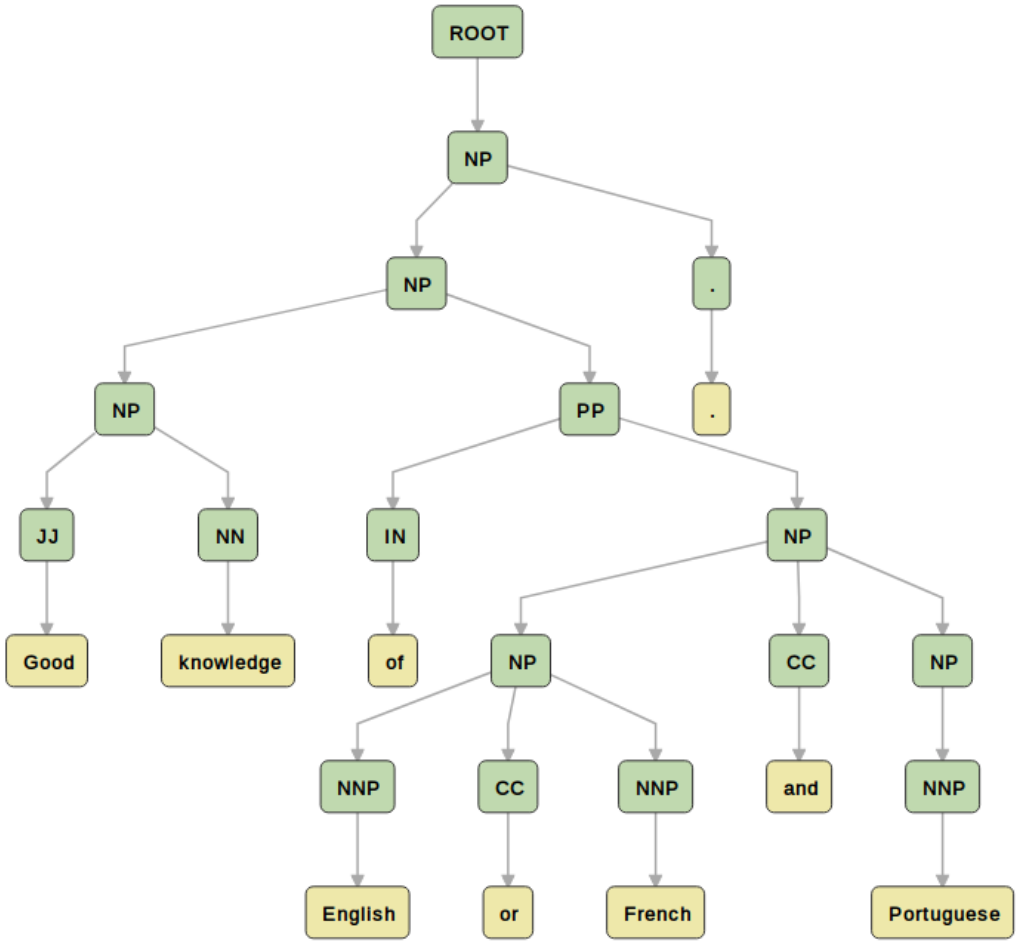


Figure 5.8: Parse tree for *Good knowledge of English or French and Portuguese.*

We use the following nomenclature to explain the set of created rules:

- **n-mandatory-languages**: Number of mandatory languages specified by the recruiter in the field *languages.mandatory*;
- **mandatory-languages**: Languages that are mandatory;
- **subset-mandatory-languages**: Languages that belong to a set of languages in which only a subset is mandatory.
- **optional-languages**: Languages that are optional.

The first method consists of comparing, for each language, the semantic similarity of its modifiers with the language levels (Table 4.2) specified in the SF. For example, in Listing 4, *English* is modified by the adjective *good* and the language level *C1*. In order to detect whether there is any semantic contradictions, the model compares the semantic similarity between the modifiers (*good* and *C1*). As the two words are similar, the model then compares the language level with the language levels specified in the SF (note that comparing the two modifiers with the values of

the SF would be redundant). This way, the model compares *C1* with the reading level (C1), the writing level (A1) and the speaking level (C1). As the writing level is different than the modifier, the model detects a lexical contradiction between the modifier *C1* and the language level defined in *languages_1.write*.


```
{"info.en.jobRequirements.mustHave": "Good knowledge of English (minimal C1)",
"languages_1.code": "en",
"languages_1.read": "C1",
"languages_1.write": "A1",
"languages_1.speak": "C1",
"languages_1.exclusionFactor": "False",
"languages.mandatory": "1"}
```


Listing 4: Example of some fields of a job posting

The similarity between words is given by calculating the distance between those words. *Wordnet*[9] has a series of distance functions that calculate this distance. We are using the *path_similarity* function that indicates how similar two word senses are, based on the shortest path that connects them. The possible values may vary from 0 to 1 (0 indicates no similarity).

However, to calculate the similarity between words, they must exist in the database. Since the language levels are not recognized by *Wordnet*, we created a dictionary, for both English and French, containing a word cloud that describes these same levels. Table 5.3 contains some of the words we use to describe the language levels.

Levels	**English**	**French**
A1	beginner, simple, basic, elementary	simple, novice, base, élémentaire
A2	beginner, simple, basic, elementary	simple, novice, base, élémentaire
B1	intermediate, independent, skill	intermédiaire, indépendant, connaissance
B2	good, fluent, clear, knowledge	maîtrise, courant, compétence
C1	competent, proficient, advanced, concise	indépendant, compétence, courant, avancée
C2	proficient, advanced, concise, profound, exceptional	parfait, compétence, courant, avancéé, maîtrise

Table 5.3: Example of some of the words we use to describe the language levels in English and French

We consider two words similar if their semantic similarity has a value equal or higher than 0.6. However, *Wordnet*'s measures of distance between words restrict their applicability to the syntactic categories of noun and verb, excluding the adjectives. As our modifiers are constituted by those three syntactic categories, to surpass such limitation, we first get a list of all the nouns

[9] https://wordnet.princeton.edu

that are related to the words we want to compare. Only then, we iterate over each noun of the modifier and each noun of the language level and calculate the similarity for the two words. The similarity between the two is the maximum value obtained when comparing each pair of words.

To **determine semantic contradictions**, we developed a function that goes through the following steps:

1. We first divide the set of modifiers (e.g.: *good, C1, proficient, A2*) into two different groups: the language levels (e.g.: *C1, A1*) and the rest of the modifiers (e.g.: *good, proficient*).

2. Then, for each level and modifier specified, we check their semantic similarity. If the words are similar, that is, the result is equal or higher than 0.6 (value previously defined by us), then there is no contradiction between the language level and the modifier. Otherwise, we create a contradiction of the **lexical** type.

3. Afterwards, if any language level was detected, we check if its value is equal to the levels specified in the respective structured language field in the *speak, read* and *write* subfields. If so, no contradiction is created, otherwise, a contradiction of the type **lexical** arises.

4. At last, if no language level was defined, we calculate the similarity between the modifier and each language level in the respective structured field. If they are semantically similar, no contradiction is detected. Elseways, a contradiction of the type **lexical** is created.

To **find numeric contradictions**, we do the following:

- We iterate over each language specified in the structured fields. If that language is not specified in the unstructured fields, then we decrease *n-mandatory-languages* by one. We do this because, as mentioned before in Chapter 4, the **Language-not-specified** contradictions only occur when a language is specified in the UF, but not in the SF. The contrary is not marked as contradictory.

- After iterating over all the language fields, we check if the number of languages specified in the unstructured fields is equal to the new *n-mandatory-languages*. If these variables differ, then we create a contradiction of the **Numeric** type. Else, no contradiction is detected.

- The next steps consists of iterating over the languages specified in the unstructured fields and check the following:

  - If the language belongs to the *mandatory-languages* of the UF and is classified as mandatory in the SF, there is no contradiction.
  - If the language is mandatory in the UF and it belongs to *subset-mandatory-languages* or *optional-languages* in the SF, there is a **Lexical** contradiction.
  - If the language is classified as *subset-mandatory-languages* in the UF and it is classified as mandatory in the SF, then we check if the exclusion factor is set to True. If so, there is a contradiction of the type **Other**; else, there is no contradiction.

- If the language belongs to the *subset-mandatory-languages* in the UF and it does not belong to the *mandatory-languages* in the SF, then the exclusion factor is set to False and there is no contradiction.
- If the language is optional and it is not specified in the right SF, that is, in the optional SF, then the job posting has a contradiction of the type **Other**.
- If the language is not specified in the SF, then there is a contradiction of the type **Language-not-specified** in the job position.

These rules were first created for the detection of contradictions in English. After testing the model for English language, we adapted the developed model for the French language:

- We translated the optional words used for determining if a sentence is optional (*atout, avantage*);
- We translated the word cloud used to describe the language levels (as seen in Table 5.3);
- We adjusted the external tools being used to work according to the job posting language;
- We adapted the relationships obtained in the dependency parser as their tags are different from the tags in English.

5.3.6 Results

As discussed in Chapter 4, we annotated 80 job postings in English and 80 job postings in French using the *brat* tool. We used these annotations to test our model separately for each language.

As we needed to compare our model contradictions with the contradictions annotated in the GS corpus, we started by iterating over the *ann* files. Each file was parsed to transform the annotated contradictions and ambiguities in a format our model could understand:

- For lines starting with a **T**, we obtained the name of the field where the text appears, and then we created an annotation with ID, type, offsets and the span text.
- For lines starting with an **A**, we updated the subtype or the contradictory-field of the annotation with id equal to ID.
- For lines starting with a **R**, we updated the contradictory-field and contradictory-text of the annotation whose id is ID.

Afterward, we filtered the annotations by contradictions that are related to languages. Thereafter, we checked the outputs of our model and compared them with the annotations from the GS corpus. This comparison was made on two different levels:

1. As if our problem is a **binary classification problem**, that is, the model detects whether a sentence is contradictory or not, regardless of whether there is a match between the subtypes.

2. As if our problem is a **multiclass classification problem**, more specifically, we checked if the model correctly identifies a sentence as contradictory and if the subtypes match.

Two annotations are considered equal either if they are symmetric (i.e., if the text of one is equal to the contradictory text of the other and vice-versa) or if both texts have an overlap (character-level) of at least 70% and their types match. In a multiclass classification context, additionally, the two annotations' subtypes must match.

In Table 5.4, it is possible to see the number of sentences identified by our model, for both English and French. From the 2467 English sentences, 108 sentences were identified as language-related sentences, that is, only 4% of the sentences were identified as language-related. For French, this ratio is slightly higher, with 110 identified sentences out of 1385. Each one of these 218 sentences was compared with other fields in order to see if there were any contradictions.

	No. total sentences	**No. language sentences**	**% of identified sentences**
English	2467	108	4
French	1385	110	7

Table 5.4: Number of total sentences identified versus the number of language-related sentences

5.3.6.1 Binary classification results

Out of the 108 English sentences detected, our model tested 529 possible combinations, as seen in Table 5.5, from which 51 were correctly identified as contradictions, 453 were correctly identified as non-contradictions, while 25 were incorrectly classified. We used several metrics (accuracy, recall, precision and f-score) to evaluate our model, as observed in Table 5.7. Although the accuracy has a high value, it is important to have into account that this value on its own says little about the performance of our model, as it is possible to observe how imbalanced our class distribution is. For that same reason, the most relevant metric is f-score as it considers both recall and precision metrics.

	Contradiction (Actual)	**No Contradiction (Actual)**	
Contradiction (Predicted)	51	11	62
No Contradiction (Predicted)	14	453	467
	65	464	529

Table 5.5: Confusion matrix for English (binary classification problem)

Regarding the French contradictions, our model tested 718 possible combinations, from which 77 were actual contradictions. Our model correctly identified 52 contradictions, 632 non-contradictions, and failed to correctly classify 34 annotations.

Table 5.6: Confusion matrix for French (binary)

	Contradiction (Actual)	No Contradiction (Actual)	
Contradiction (Predicted)	52	25	77
No Contradiction (Predicted)	9	632	641
	61	657	718

The performance of our model for the French language can be observed in Table 5.6, with high accuracy and recall values.

Table 5.7: Metrics obtained for our languages model in English and French (binary classification)

	Accuracy (%)	Recall (%)	Precision (%)	F-score (%)
English	95.3	78.5	82.3	80.3
French	95.3	85.2	67.5	75.4

In order to detect how good our model is, we decided to compare it with some baseline models. For instance, for English:

- If the model **only predicted the most observed class** (non-contradictions), we would have 464 true negatives and 65 false negatives. This way, the model would have 87.7% of accuracy, 0% of recall, and it would not have precision nor f-score values, as precision would be *NaN*. Despite the proximity between the accuracy values, we can see in Table 5.8 that the values for the other metrics fall far behind from the ones obtained in the current implemented model.

- If the model **only predicted contractions**, then we would have 65 true positives and 464 false positives. This way, the model would have 12.3% of both accuracy and precision, 100% of recall and 21.9% of f-score, as observed in Table 5.8. Although the recall value is higher, all the other values drop significantly.

Table 5.8: Metrics obtained for baselines models in English (binary classification)

	Accuracy (%)	Recall (%)	Precision (%)	F-score (%)
Non-contradictions	87.7	0	NaN	NaN
Contradictions	12.3	100	12.3	21.9

As expected, we can conclude that our model surpasses the performances of the baseline models.

5.3.6.2 Multiclass classification results

When treating the detection of contradictions as a multiclass classification problem , we obtained the confusion matrix in Table 5.9.

Table 5.9: Confusion matrix for English (multiclass classification problem)

	Non-contradiction (Actual)	**Lexical (Actual)**	**Numeric (Actual)**	**Language-not-specified (Actual)**	**Other (Actual)**	
Non-contradiction (Predicted)	453	0	5	3	6	467
Lexical Contradiction (Predicted)	1	30	0	0	0	31
Numeric Contradiction (Predicted)	3	0	10	0	0	13
Language-not-specified Contradiction(Predicted)	6	0	0	9	0	15
Other Contradiction (Predicted)	1	0	0	0	2	3
	464	30	15	12	8	529

Table 5.10: Metrics obtained for our language models in English (multiclass classification problem)

	Accuracy (%)	Recall (%)	Precision (%)	F-score (%)
Non Contradictions	95.3	97.6	97.0	97.3
Lexical Contradictions	99.8	100	96.8	98.4
Numeric Contradictions	98.4	66.7	76.9	71.4
Language-not-specified Contradictions	98.2	75.0	60.0	66.7
Other Contradictions	98.6	25.0	66.7	36.4

Looking at Table 5.10, it is possible to observe that the model detects *Lexical* contradictions with high f-score values (98.4%). However, the model does not perform so well when detecting *Numeric*, *Language-not-specified* and *Other* contradictions.

5.3.7 Error analysis

Our current model failed to correctly classify a total of 59 cases out of the 1247 combinations for the two languages. There are several factors that might have contributed to that:

- The model fails to correctly divide sentences as there might be language errors in the job postings;
- The model fails to identify sentences that are related to languages but which do not explicitly express any language. For example, *Any European language is a plus.*;

- The model incorrectly identifies sentences that have languages despite not being language-related. Although we already apply some filters, these are based on regular expressions, so it might not cover all cases;
- The dependency parser fails to identify correctly the relationships between words. This is more frequent in French, as the models for English are more optimized than for the French;
- The constituency parser fails to identify the grammatical components of a sentence;
- The library we are using to fix misspelled words fails to correct some of the words;
- The model fails to identify a language's modifiers due to different writing styles that give different relationships between words that the model has not foreseen.

5.3.8 Summary

There were some obstacles during the development of this model, where specific circumstances influenced the output and diminished the quality of the results. Nonetheless, we overcame some of these obstacles and incrementally improved the detection of inconsistencies in sentences related to languages. Looking at the obtained outcomes, there are some conclusions that we can draw:

- Detecting discrepancies in sentences related to language is susceptible to errors.
- The model that detects contradictions in job postings written in English has better results than the model for French postings.
- Relatively to binary contradiction detection, the models achieve an F-score of 80.3% and 75.4% in job postings written in English and French, respectively.
- When we treat contradiction detection as a multiclass problem, the model detects non-contradictions and lexical contradictions with a high F-score. However, the performance is poorer when trying to identify contradictions of the *Numeric*, *Language-not-specified* and *Other* types.

5.4 Experience Model

The current implementation of the experience model focuses on the detection of contradictions that are a result of a mismatch between the years of experience specified in the UF and in the SF *reqExperience*.

Due to little time, we are only identifying sentences whose experience can be quantified. In other words, sentences like "Little experience is a plus." are not being targeted, but sentences like "3-5 years of experience" are. However, the processing of such sentences can be later implemented in the current model without many changes.

5.4.1 Identification of the experience-related sentences

Just as we do in the *languages* model (Section 5.3), we first need to find all the sentences that are experience-related. This requires that we iterate over all the sentences detected in a job posting and find the ones that contain given keywords, that is, words related to experience[10].

The presence of keywords and numbers alone does not mean that the sentences are related to experience. For example, let us look at two different cases:

- "Our Programme is dedicated for people who have a minimum of 2 years of experience."
- "Our Programme is a 6-month learning program designed for students at any level (BSc, MSc, MBA, and Ph.D.) with limited working experience."

In the first case, it is clear that the sentence is related to experience. However, in the second one, although we can find the relevant keywords and the presence of numbers, we cannot identify such sentence as experience-related, as *6* is not a modifier of experience.

Due to this, we need to distinguish such sentences by identifying the words that characterize the keywords (detailed in Section 5.4.2). If no modifiers are found, the sentence being analyzed is discarded; otherwise, the sentence is added to the list of experience-related sentences

5.4.2 Preprocessing & Extracting modifiers

As we have just mentioned, having experience-related words does not mean the sentence is indeed related to experience. Because of that, sentences need to be distinguished by identifying the words that characterize the keywords, that is, by finding their modifiers. Thus, we add a further preprocessing step on top of the one we already have (this latter is discussed in Section 5.2), just like we did in the implementation of the languages model. In this model, it is important to obtain the relationships between words. Therefore, the new preprocessing step consists of the use of lemmatization and dependency parsing.

After preprocessing the sentences, we are ready to extract the keywords' modifiers. This extraction task is not as complex as the one we have implemented in the previous model since our focus is on recognizing only sentences that have modifiers that quantify the experience. In spite of, the extraction function receives as input a sentence and the experience-related word found, searching recursively for possible numeric modifiers (*nummod*) that quantify the mentioned word.

For example, for the sentence *Our Programme is dedicated to people who have a minimum of 2 years of experience*, our function should be able to identify the number *2* as a modifier of the word *experience*. Nevertheless, for the sentence *Our Programme is a 6-month learning program designed for students at any level (BSc, MSc, MBA, and Ph.D.) with limited working experience.*, the dependency parser should fail to identify any modifiers related to *experience* as there are none.

As seen in Figure 5.9, for the first case, the word *experience* has the word *years* as modifier, and subsequently this has *2* as its numeric modifier. Basically, the successive calls of the extraction function will obtain *2* as a modifier of the word *experience*.

[10]In English, we can search for the words *experience* and *working*, while in French we search for *expérience*

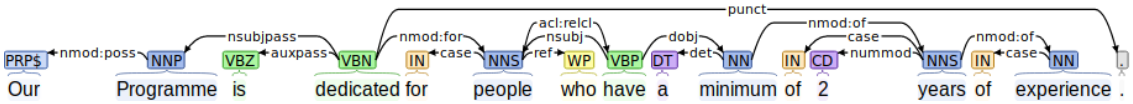


Figure 5.9: Dependency parser for *Our Programme is dedicated for people who have a minimum of 2 years of experience.*

Contrarily, Figure 5.10 shows that in the second case the word *experience* has the words *working* and *students* as modifiers. Regardless, there is no path in which the number *6* is a modifier of the word *experience*. This way, there are no numeric modifiers for the word *experience*.

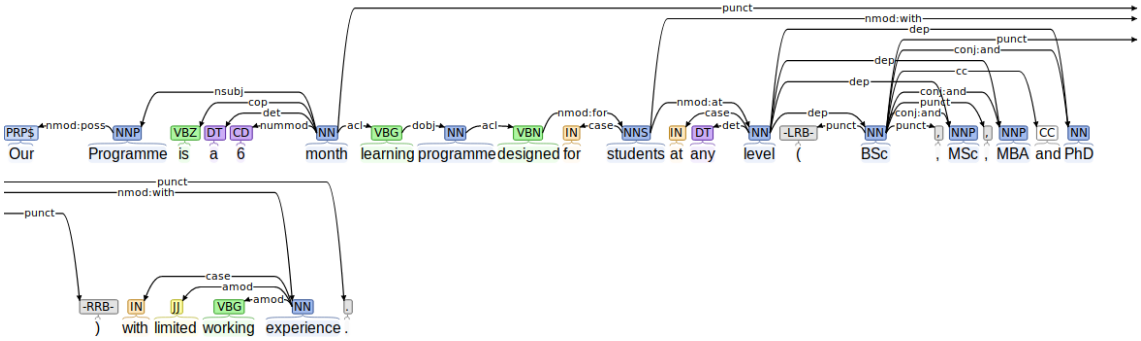


Figure 5.10: Dependency parser for *Our Programme is a 6 month learning programme designed for students at any level (BSc, MSc, MBA and PhD) with limited working experience.*

5.4.3 Rules for detecting contradictions

After establishing the sentences that are related to experience, we centered our approach on creating rules to properly uncover the contradictions that exist.

As explained in Chapter 4, there is a field named *reqExperience* whose purpose is to establish the range of years of experience that a candidate must have in order to be considered for the job by the matching algorithm. In the structured fields, the recruiter can either specify a minimum, a maximum, a fixed number or an interval. If a minimum is specified, then the maximum is equaled to a constant (for now, this constant is 50). If a maximum is defined, then the minimum is set to 0. If the recruiter indicates a fixed-number, then the recruiter must have exactly that number of years of experience. Besides this, the recruiter can also define such values in the unstructured fields.

Our approach comprises the following steps:

- In the first place, we obtain the number of experience that is stipulated in the *reqExperience* structured field.
- Afterwards, our function iterates over each sentence. For each sentence, we obtain the minimum and maximum values defined in that same sentence. If only the minimum value is set, then the maximum value takes the value specified in the structured fields. If only the maximum is defined, then the minimum takes the value of the minimum in the structured field. If only one value is specified, then the maximum is equal to the minimum.

- As we stated in Section 5.3, the recruiters misuse the UF and sometimes they specify optional qualifications in the *info.en.jobRequirements.mustHave* field. Because of that, we check whether the sentence is optional or mandatory. If it is mandatory, we check whether the minimum and the maximum values specified in that sentence and in the SF match. If there is a mismatch between these values, then a numeric contradiction is created. If the sentence is optional, we do the reverse reasoning, i.e., if the minimum experience in the SF matches the minimum experience specified in the sentence and the maximum experience in the SF matches the maximum experience in the sentence, then a contradiction arises.

Once again, these rules were first elaborated for the English language and later adapted for the French language. To adapt the model to detect experience-related contradictions in French, we did some adjustments:

- We adapted the external tools that are being used to work according to the language in which the job posting is written;

- We translated the English word cloud to French.

5.4.4 Results

The output from the experience model is also tested with the GS corpus, that, as mentioned before, is constituted by a total of 160 annotated job postings, half in English and half in French. The table represents the percentage of sentences that actually correspond to the sentences our model associated with the experience.

Table 5.11: Number of total sentences identified versus the number of experience-related sentences

	No. total sentences	No. experience sentences	% of identified sentences
English	2551	33	1
French	1500	18	1

5.4.4.1 Binary Classification Results

Table 5.12 represents the confusion matrix for the problem of identifying a sentence as contradictory or not in English. As it is possible to see, the developed model for English classified correctly 13 contradictions, 18 non-contradictions, and failed to identify 6 contradictions. The values for the metrics used to test the performance of the model can be found in Table 5.14.

Table 5.12: Confusion matrix for experience model in English (binary)

	Contradiction (Actual)	No Contradiction (Actual)	
Contradiction (Predicted)	13	2	15
No Contradiction (Predicted)	4	18	22
	17	20	37

Table 5.13 summarizes the confusion matrix values for French. In summary, the experience model correctly identifies 6 contradictions, 8 non-contradictions, failing to correctly classify 6 annotations.

Table 5.13: Confusion matrix for experience model in French (binary classification)

	Contradiction (Actual)	No Contradiction (Actual)	
Contradiction (Predicted)	6	4	10
No Contradiction (Predicted)	2	8	10
	8	12	20

Table 5.14: Metrics obtained for our experience model in English and French (binary)

	Accuracy (%)	Recall (%)	Precision (%)	F-score (%)
English	83.8	76.5	86.7	81.3
French	70.0	75.0	60.0	66.7

5.4.5 Error Analysis

As shown in at Table 5.14, we can verify that the model performs better for the English language than for the French one. Out of the total 57 possible combinations, the model fails to correctly classify 12 cases. Such values can be explained by the following factors:

- The model fails to correctly divide sentences as there might be language errors in the job postings;
- The model fails to identify some sentences related to experience;
- The dependency parser being used might establish some incorrect relationships between words. Nevertheless, these models perform better for English than French, since the latter are not as developed as the first.

5.4.6 Summary

Taking into account the process of development of this model and its results, we can mention the following:

- Due to time limits, we were only able to implement contradiction detection in sentences related to experience when the experience is numerically quantified.
- As there is only one structured field to compare inconsistencies with, the *reqExperience* field, the number of possible contradictions between this field and the unstructured fields regarding experience is a lot lower than in the languages model.
- Because of what is mentioned in the previous point, the distribution of results for this model is more balanced than the distribution of the results from the languages model. Therefore, we can assume that the accuracy metric is more relevant and supplies more significant values.

Chapter 6

Conclusion and Future Work

Finding the ideal candidate for a certain job position is a complex task. *Skeeled* is a company that tries to facilitate this task by providing a list of favorable candidates to a given job. The problem arises when recruiters publish postings that contain wither ambiguities or contradictions which antagonizes the precision of *skeeled*'s matching algorithm.

Hence, the initial purpose of this dissertation was to develop a solution that satisfied two requirements:

- Detect contradictions automatically between two different fields or inside the same unstructured field;
- Automatically find ambiguities inside the same unstructured field.

Although we did not find any works regarding detecting such irregularities in job postings, we found some useful approaches to detect contradictions and ambiguities. These techniques fall in the domains of natural language processing (NLP) and machine learning (ML).

We started by building a labeled dataset that could be used to test the output from our solution. We were able to assemble a dataset that indicates the existing contradictions and ambiguities in job postings written in different languages. However, we noticed that building a reliable dataset is a time-consuming and meticulous task. To the extent of our knowledge, there are no datasets annotated with contradictions and ambiguities in job postings (in different languages).

Afterwards, our method consisted of creating models for the different areas in which the contradictions might occur, that is, languages and experience. Due to a delay in the annotation of the dataset, the approach we took was different from the one we initially had foreseen. As ML models need labeled data to be trained, instead we opted to develop models that would not have such a limitation and we developed rule-based models that make use of NLP techniques. Nonetheless, creating rules to detect inconsistencies takes more time than automatically deducing these rules. Due to the short time we had in hands and the faced problems, the education model was not developed.

At last, we compared the results of our models with the annotations of our Gold Standard Corpus and we computed different metrics to see their performance. For the language models, we noticed that our model has a good performance and is able to correctly identify contradictions both for English and for French. Although the results of the experience model are not as good as the results from the language model, this model performs reasonably well taking into account the time we had to develop it.

With the development of this dissertation, we were able to answer all the initial proposed questions except one. We successfully identified what are the most common inconsistencies that occur in a job posting. Moreover, our results prove that the taken approach, despite not being a very efficient solution, is suitable to detect contradictions in job postings. As we did not develop a solution to detect ambiguities, due to the presence of little ambiguities in our dataset, we are unable to understand if our approach is suitable or not. By detecting contradictions, we reckon the improvement of the job postings as the recruiter must correct its initial job proposal.

To conclude, the developed project provides some important contributions that must be referred:

- We provide a corpus annotated with contradictions and ambiguities in job postings in English and French;
- We detect which types of contradictions are more common in job postings;
- We provide a means of detecting contradictions related to language and experience in job postings;
- The models we developed can be used later as a baseline models to test new approaches;

6.1 Future Work

Despite the developed models, there is still a lot of work that needs to be done:

- The annotation of our dataset needs to be continued, not only for French and English but also for other languages, as *skeeled* is a multilingual platform;
- Other approaches can be developed and compared with the currently developed models. For example, compare ML models with our rule-based models;
- With the increasing number of annotated ambiguities, a model to detect such inconsistencies must be developed;
- Improve the current experience model as this one is only capable of detecting contradictions in which the experience can be quantified;
- Build a model able to detect contradictions related to education.

References

[AP08] Ron Artstein and Massimo Poesio. Inter-coder agreement for computational linguistics. *Computational Linguistics*, 34(4):555–596, 2008.

[APA+17] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth D Trippe, Juan B Gutierrez, and Krys Kochut. A brief survey of text mining: Classification, clustering and extraction techniques. *arXiv preprint arXiv:1707.02919*, 2017.

[BKK] Daniel M Berry, Erik Kamsties, and Michael M Krieger. From contract drafting to software specification: Linguistic sources of ambiguity, 2003. *A Handbook*.

[BLK09] Steven Bird, Edward Loper, and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

[CFL13] Alexander Clark, Chris Fox, and Shalom Lappin. *The handbook of computational linguistics and natural language processing*. John Wiley & Sons, 2013.

[CM14] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750, 2014.

[CP04] Chklovski and Pantel, 2004.

[Dev] Google Developers. Machine learning crash course with tensorflow apis.

[DMMM+06] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454. Genoa Italy, 2006.

[Dra17] Valentina Dragos. Detection of contradictions by relation matching and uncertainty assessment. *Procedia Computer Science*, 112:71–80, 2017.

[GCK10] Benedikt Gleich, Oliver Creighton, and Leonid Kof. Ambiguity detection: Towards a tool explaining ambiguity sources. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 218–232. Springer, 2010.

[Gol16] Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

[Hop08] John E Hopcroft. *Introduction to automata theory, languages, and computation*. Pearson Education India, 2008.

[JM14] Dan Jurafsky and James H Martin. *Speech and language processing*, volume 3. Pearson London, 2014.

[KDNA16] Wahab Khan, Ali Daud, Jamal A Nasir, and Tehmina Amjad. A survey on the state-of-the-art machine learning models in the context of nlp. *Kuwait journal of Science*, 43(4), 2016.

[Kri11] Klaus Krippendorff. Computing krippendorff's alpha-reliability. 2011.

[LBN+18] Vijay Lingam, Simran Bhuria, Mayukh Nair, Divij Gurpreetsingh, Anjali Goyal, and Ashish Sureka. Deep learning for conflicting statements detection in text. Technical report, PeerJ Preprints, 2018.

[MKS18] Manel Mezghani, Juyeon Kang, and Florence Sèdes. Industrial requirements classification for redundancy and inconsistency detection in semios. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pages 297–303. IEEE, 2018.

[MRM08] Marie-Catherine Marneffe, Anna N Rafferty, and Christopher D Manning. Finding contradictions in text. *Proceedings of ACL-08: HLT*, pages 1039–1047, 2008.

[PNS13] Minh Quang Nhat Pham, Minh Le Nguyen, and Akira Shimazu. Using shallow semantic parsing and relation extraction for finding contradiction in text. 2013.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PWH+04] Sameer S Pradhan, Wayne H Ward, Kadri Hacioglu, James H Martin, and Dan Jurafsky. Shallow semantic parsing using support vector machines. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, 2004.

[R+01] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.

[RFG+17] Benedetta Rosadini, Alessio Ferrari, Gloria Gori, Alessandro Fantechi, Stefania Gnesi, Iacopo Trotta, and Stefano Bacherini. Using nlp to detect requirements defects: an industrial experience in the railway domain. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 344–360. Springer, 2017.

[RS10] Allan Berrocal Rojas and Gabriela Barrantes Sliesarieva. Automated detection of language issues affecting accuracy, ambiguity and verifiability in software requirements written in natural language. In *Proceedings of the NAACL HLT 2010 Young Investigators Workshop on Computational Approaches to Languages of the Americas*, pages 100–108. Association for Computational Linguistics, 2010.

[Sch] Helmut Schmid. TreeTagger - a part-of-speech tagger for many languages.

[SDL80] Frank E Saal, Ronald G Downey, and Mary A Lahey. Rating the ratings: assessing the psychometric quality of rating data. *Psychological bulletin*, 88(2):413, 1980.

[SJ15] Unnati S Shah and Devesh C Jinwala. Resolving ambiguities in natural language software requirements: a comprehensive survey. *ACM SIGSOFT Software Engineering Notes*, 40(5):1–7, 2015.

[SZ17] Ali Olow Jim'ale Sabriye and Wan Mohd Nazmee Wan Zainon. A framework for detecting ambiguity in software requirement specification. In *Information Technology (ICIT), 2017 8th International Conference on*, pages 209–213. IEEE, 2017.

[TRB10] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

[Uni10] Princeton University. "About WordNet", 2010.

[ver] A Class-Based Verb Lexicon.

[ZZC+13] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 434–443, 2013.

REFERENCES

Appendix A

Dataset

A.1 Dataset Fields

A.1.1 Structured Fields

- **drivingLicenseIsKiller**
  - **Description**: If the driving license is a killer factor, this field is set to TRUE; if not, this field is set to FALSE; if no value is specified, this field is omitted from the *.json* file.
  - **Mandatory**: FALSE
  - **Possible values**: *TRUE, FALSE*
  - **Relevant for annotation**: TRUE
- **endDate**
  - **Description**: End date of acceptance of candidates for the job posting.
  - **Mandatory**: TRUE
  - **Relevant for annotation**: TRUE
- **fieldsOfStudy**
  - **Description**: Fields of study needed for the job position
  - **Mandatory**: FALSE
  - **Relevant for annotation**: TRUE
- **industry**
  - **Description**: The industry of the job posting
  - **Mandatory**: TRUE
  - **Relevant for annotation**: TRUE

- **languages.mandatory**
  - **Description**: Number of languages the candidate needs to know.
  - **Mandatory**: TRUE
  - **Possible values**: Integer numbers
  - **Relevant for annotation**: TRUE
- **languages_*number*[1].code**
  - **Description**: Country codes of the mandatory languages the recruiter specified (e.g., pt, en, nl). For example, if the recruiter specified "1" in the "languages.mandatory" field, then, there needs to be at least one code specified in the "languages_number.code" fields.
  - **Mandatory**: At least, one of these fields is mandatory
  - **Possible values**: *fr, en, lb, de, nl*
  - **Relevant for annotation**: TRUE
  - **Examples**: languages_1.code: *en*; languages_2.code: *fr*
- **languages_*number*.exclusionFactor**
  - **Description**: When the number specified in *languages.mandatory* is **equal** to the number of languages specified in *languages_number.code* (see Example 1), these fields are set to FALSE, i.e., they are irrelevant.

    However, when the number of languages specified in *languages_number.code* is **bigger** than the number specified in the *languages.mandatory* field (see example 2), then these fields are used to specify which languages are indeed mandatory (values set to TRUE) and which ones are part of a set whose subset is mandatory (values set to FALSE)[2]. If the number of these fields whose values are set to TRUE is **less** (see examples 3 and 4) than the number specified in *languages.mandatory*, then we can look at the fields set to TRUE as AND, and the fields set to FALSE as OR.
  - **Mandatory**: Mandatory if *languages_number.code* field exists
  - **Possible values**: TRUE, FALSE
  - **Relevant for annotation**: TRUE
  - **Examples**:
    1. *languages.mandatory: 2*
       languages_1.code: en

[1] *number* corresponds to a natural number that starts in 1 and its maximum value is the number of languages available in the application.

[2] For example, if the recruiter asks for English or French, both these languages belong to a set whose subset is mandatory - only one of these languages is mandatory.

languages_2.code: fr
languages_1.exclusionFactor: FALSE
languages_2.exclusionFactor: FALSE

From this example, we can conclude that both *en* and *fr* are mandatory.

2. *languages.mandatory: 2*
   languages_1.code: en
   languages_2.code: fr
   languages_3.code: nl
   languages_1.exclusionFactor: ***TRUE***
   languages_2.exclusionFactor: ***TRUE***
   languages_3.exclusionFactor: FALSE

   There are two mandatory languages but three languages specified. We can conclude that both *en* and *fr* are mandatory, while *nl* is optional. However, there is a specific field for optional languages, so *nl* should be in that same field.

3. *languages.mandatory: 2*
   languages_1.code: en
   languages_2.code: fr
   languages_3.code: nl
   languages_1.exclusionFactor: ***TRUE***
   languages_2.exclusionFactor: FALSE
   languages_3.exclusionFactor: FALSE

   Since there are two mandatory languages and only one is specified as mandatory, the necessary languages are: *en* **AND** (*fr* **OR** *nl*).

4. *languages.mandatory: 2*
   languages_1.code: en
   languages_2.code: fr
   languages_3.code: nl
   languages_1.exclusionFactor: FALSE
   languages_2.exclusionFactor: FALSE
   languages_3.exclusionFactor: FALSE

   Since there are two mandatory languages and none is specified as mandatory, the necessary languages are: (*en* **AND** *fr*) **OR** (*fr* **AND** *nl*) **OR** (*en* **AND** *nl*).

- **languages_*number*.read**
  - **Description**: Reading minimum level needed for the languages specified in the field *languages_number.code*
  - **Mandatory**: Mandatory if *languages_number.code* exists
  - **Possible values**: *A1, A2, B1, B2, C1, C2*
  - **Relevant for annotation**: TRUE
  - **Example**: languages_1.read: *C1*; languages_2.read: *B1*
- **languages_*number*.speak**
  - **Description**: Speaking minimum level needed for the languages specified in the field *languages_number.code*
  - **Mandatory**: Mandatory if *languages_number.code* exists
  - **Possible values**: *A1, A2, B1, B2, C1, C2*
  - **Relevant for annotation**: TRUE
  - **Example**: languages_1.speak: *C1*; languages_2.speak: *B1*
- **languages_*number*.write**
  - **Description**: Writing minimum level needed for the languages specified in the field *languages_number.code*
  - **Mandatory**: Mandatory if *languages_number.code* exists
  - **Possible values**: *A1, A2, B1, B2, C1, C2*
  - **Relevant for annotation**: TRUE
  - **Example**: languages_1.write: *C1*; languages_2.write: *B1*
- **optional_languages_*number*.code**
  - **Description**: Country codes of the optional languages the recruiter specified (e.g., pt, en, nl).
  - **Mandatory**: FALSE
  - **Possible values**: *fr, en, lb, de, nl*
  - **Relevant for annotation**: TRUE
  - **Examples**: optional_languages_1.code: *en*; optional_languages_2.code: *fr*
- **optional_languages_*number*.read**
  - **Description**: Reading minimum level needed for the languages specified in the field *optional_languages_number.code*

  - **Mandatory**: Mandatory if *optional_languages_number.code* exists
  - **Possible values**: *A1, A2, B1, B2, C1, C2*
  - **Relevant for annotation**: TRUE
  - **Example**: optional_languages_1.read: *C1*; optional_languages_2.read: *B1*

- **optional_languages_*number*.speak**
  - **Description**: Speaking minimum level needed for the languages specified in the field *optional_languages_number.code*
  - **Mandatory**: Mandatory if *languages_number.optional.code* exists
  - **Possible values**: *A1, A2, B1, B2, C1, C2*
  - **Relevant for annotation**: TRUE
  - **Example**: optional_languages_1.speak: *C1*; optional_languages_2.speak: *B1*

- **optional_languages_*number*.write**
  - **Description**: Writing minimum level needed for the languages specified in the field *optional_languages_number.code*
  - **Mandatory**: Mandatory if *optional_languages_number.code* exists
  - **Possible values**: *A1, A2, B1, B2, C1, C2*
  - **Relevant for annotation**: TRUE
  - **Example**: optional_languages_1.write: *C1*; optional_languages_2.write: *B1*

- **minReqEducation**
  - **Description**: The minimum required education the candidate needs to have.
  - **Mandatory**: TRUE
  - **Possible values**: *High School, Technical, Associate, Bachelor, Postgraduate, Master, Doctorate, Erasmus, Certificate of Professional Ability (CCP), CAP, BEV ou equivalent, Diploma of Professional Aptitude (DPA), Niveau Bac, Bac Obtenu, Educator Diploma (14e), Bac +2, Bac +3, Bac +4, Bac +5, > Bac +5, Other*
  - **Relevant for annotation**: TRUE

- **reqExperience**
  - **Description**: Minimum experience, in years, a candidate must have.
  - **Mandatory**: TRUE
  - **Possible values**: "[0-10]-[0-10]", "[0-10]+" (minimum value or more) "[0-10]-" (maximum value or less)
  - **Relevant for annotation**: TRUE

  - **Examples**: 2-10: two to ten years of experience; 0+; zero or more experience; 2-: up to two years of experience.

- **requireDrivingLicense**
  - **Description**: If the driving license is required, this field is set to TRUE; if not, this field is set to FALSE; if no value is specified, this field is omitted from the *.json* file.
  - **Mandatory**: FALSE
  - **Possible values**: *TRUE, FALSE*
  - **Relevant for annotation**: TRUE
- **requireWorkPermit**
  - **Description**: If a work permit is required, this field is set to TRUE; if not, this field is set to FALSE; if no value is specified, this field is omitted from the *.json* file.
  - **Mandatory**: FALSE
  - **Possible values**: *TRUE, FALSE*
  - **Relevant for annotation**: TRUE
- **startDate**
  - **Description**: Start date of acceptance of candidates for the job posting.
  - **Mandatory**: TRUE
  - **Relevant for annotation**: TRUE
- **workingPermitIsKiller**
  - **Description**: If the working permit is a killer factor, this field is set to TRUE; if not, this field is set to FALSE; if no value is specified, this field is omitted from the *.json* file.
  - **Mandatory**: FALSE
  - **Possible values**: *TRUE, FALSE*
  - **Relevant for annotation**: TRUE

A.1.2 Unstructured Fields

- **info.*language*[3].description**
  - **Description**: Job position description.
  - **Mandatory**: TRUE

[3] *language* is a variable that can take the value of one of the language's codes. It can be *en*, *nl*, among others. Example: "info.en.description" or "info.nl.description".

  - **Relevant for annotation**: TRUE
  - **Examples**: *info.en.description*, *info.fr.description*

- **info.*language*.responsibilities**
  - **Description**: Responsibilities that the selected candidate will have in the job position.
  - **Mandatory**: TRUE
  - **Relevant for annotation**: TRUE

- **info.*language*.weOffer**
  - **Description**: What the company offers to the selected candidate.
  - **Mandatory**: FALSE
  - **Relevant for annotation**: TRUE

- **info.*language*.jobRequirements.mustHave**
  - **Description**: Requirements a candidate must have in order to be selected.
  - **Mandatory**: TRUE
  - **Relevant for annotation**: TRUE

- **info.*language*.jobRequirements.niceToHave**
  - **Description**: Characteristics favourable to the candidate in order to be selected.
  - **Mandatory**: FALSE
  - **Relevant for annotation**: TRUE

- **info_title**
  - **Description**: Title of the job posting.
  - **Mandatory**: TRUE
  - **Relevant for annotation**: TRUE
  - **Examples**: *senior software engineering, m/f 100%*; *chauffeur poids lourds (m/f)*

A.2 Job Posting

An example of a fictitious job posting with flat fields can be found in Listing 5. An original job posting has nested fields.

```
{"info.en.jobRequirements.mustHave": "Requirement 1, ...",
"info.en.jobRequirements.niceToHave": "Advantage 1, ...",
"info.en.title": "A title test",
"info.en.description": "Description of the job",
"info.en.responsibilities": "Responsibility 1, ...",
"info.en.emailsLanguage": "en",
"endDate": "2020-01-21T23:59:59.999Z",
"languages_1.code": "fr",
"languages_1.read": "C1",
"languages_1.write": "C1",
"languages_1.speak": "C1",
"languages_1.exclusionFactor": "False",
"languages.mandatory": "1",
"optional_languages_1.code": "lb",
"optional_languages_1.read": "B1",
"optional_languages_1.write": "B1",
"optional_languages_1.speak": "B1",
"industry": "some_industry",
"fieldsOfStudy": "Field 1",
"minReqEducation": "technical",
"reqExperience": "0+",
"requireDrivingLicense": "True",
"drivingLicenseIsKiller": "True",
"requireWorkPermit": "False",
"workPermitIsKiller": "False",
"startDate": "2019-01-21T14:24:02.693Z"}
```


Listing 5: Example of a flat job posting

Appendix B

Annotation Manual

ANNOTATION MANUAL FOR SKEELED'S DATASET



Contents

1 Introduction **2**

2 Concepts **3**
- 2.1 Dataset fields . . . 3
- 2.2 Contradictions . . . 3
  - 2.2.1 Types . . . 3
  - 2.2.2 Examples of Possible Contradictions in the Dataset . . . 4
- 2.3 Ambiguities . . . 7
  - 2.3.1 Types . . . 7
  - 2.3.2 Examples of Possible Ambiguities in the Dataset . . . 8

3 *brat* annotation tool **10**
- 3.1 Features . . . 10
- 3.2 Download and Setup . . . 11
- 3.3 Data . . . 11
- 3.4 How to Use . . . 12
  - 3.4.1 Examples of Annotations in the Dataset . . . 29

Chapter 1

Introduction

Skeeled is a company whose purpose is to enhance the experience for both candidates and recruiters by providing an all-in-one hiring software that digitalizes the recruitment process. It offers an innovating technology that automates and speeds up the first steps of the recruitment process, saving both recruiters and candidates a large amount of time and facilitating collaborative hiring.

Nevertheless, the process of matching a candidate to a certain job position is complex. To do so, *Skeeled* analyzes several data in their dataset, including job proposals. One problem that jeopardizes this process is the presence of ambiguities and contradictions among the job proposals. To address this problem, *Skeeled* needs to manually annotate these job postings.

In *Skeeled*'s dataset, job descriptions can be written in different languages, such as English (en), French (fr), Luxembourgish (lb), Dutch (nl), German (de). The number of languages can increase in the future.

The present document serves as a **guide** to any person who needs to make annotations in *Skeeled*'s dataset.

The present document is structured in the following way: first, we explain each field in *Skeeled*'s dataset, what contradictions and ambiguities are and give some examples for each; then, we introduce the tool used for annotating the dataset and explain how to use it with illustrative examples.

Chapter 2

Concepts

In this chapter, there is a brief introduction to the dataset fields, alongside with an explanation of what a contradiction and an ambiguity are, their types and examples that occur in the dataset.

2.1 Dataset fields

Skeeled's dataset is composed of open text fields and fields where recruiters can choose from a limited number of options. In this manual, we name these fields as **unstructured** and **structured** fields, respectively.

A further description of each field of *Skeeled*'s dataset is detailed in **??**.

2.2 Contradictions

Two or more expressions are contradictory if they are extremely unlikely to be true simultaneously. A contradiction can occur in the following cases:

- Between structured fields;
- Between unstructured fields;
- Between both structured and unstructured fields;
- Inside the same unstructured field.

2.2.1 Types

Contradictions can arise from less or more evident features. We defined several different types of contradictions that occur in *skeeled*'s dataset.

Numeric mismatch These contradictions occur from numeric mismatches between quantities, numbers and dates. See examples in 2.2.2.

World-knowledge These contradictions arise from discrepancies based on world-knowledge, i.e., external knowledge is needed to detect them. See examples in 2.2.2.

Lexical These contradictions occur when recruiters ask for different competence levels (e.g.: education, language, domain) in the same job posting. See examples in 2.2.2.

Language-not-specified These contradictions occur when the recruiters ask for a mandatory language in the unstructured fields but do not specify the language in the structured fields or vice-versa. See examples in 2.2.2

Other These contradictions encompass all the other types of contradictions that are not previously defined. See examples in 2.2.2.

2.2.2 Examples of Possible Contradictions in the Dataset

Numeric Contradictions

1. *languages.mandatory*: **3**

   info.en.jobRequirements.mustHave: Proficient in **Dutch/French & English**

   These fields are contradictory because three languages are mandatory although the expression can be interpreted as *Dutch OR (French AND English)* - one or two languages are required, respectively - or *(Dutch OR French) AND English* - two languages are required. Either way, there is not a case where the three are required.

2. *info.en.jobRequirements.mustHave*: "The candidate must have **more than three years of experience**."

   reqExperience: **0-3**

   These two fields are contradictory, as the candidate must have between 0 and 3 years of experience in one field and more than 3 years in the other field.

World-knowledge Contradictions

1. *info.en.title*: *Bus Driver*

   requireDrivingLicense: *FALSE*

   These fields are contradictory since the recruiter is searching for a bus driver that does not need to have a driving license.

2. *info.en.title*: *Bus Driver*

   languages_1.speak: "**C1**"

   These fields are contradictory because the recruiter does not specify in any other field (e.g.: responsibilities) that the driver must speak with customers. Therefore, it is extremely unlikely that a bus driver needs such a high level of English.

   NOTE: If it said in any other field that the driver needs to speak with clients, then this would not be considered a contradiction.

3. *info.en.jobRequirements.mustHave*: We hire **engineers and architects**.

   minReqEducation: **High School**

   These fields are contradictory because engineers and architectures have higher degrees like *Bachelor* or *Master*. This is a world-knowledge contradiction since one needs to know that to be an engineer/architect one needs a higher education degree.

4. *info.en.jobRequirements.mustHave*: "**High knowledge** in systems administration"

   reqExperience: **0-1**

   minReqEducation: **High School**

   These two fields are contradictory, as it is extremely unlikely that a candidate with *0-1 years of experience* and whose degree of education is only *high school* has a high knowledge of *systems administration*.

   NOTE: On the one hand, if the *minReqEducation* was set to Bachelor, for example, this would not be a contradiction as *high knowledge* can be acquired from studying. On the other hand, if *reqExperience* was set to 3-6, for example, then these fields would not be contradictory as *high knowledge* can be acquired from experience.

Lexical Contradictions

1. *info.en.jobRequirements.mustHave*: "A **Master degree** is required".

   minReqEducation: "**Bachelor**"

   The recruiter asks for a candidate with a Master degree in one field and then asks for a Bachelor degree in another field.

2. *info.en.jobRequirements.mustHave*: "**High level** in spoken and written English & Portuguese".

   languages_1.write: "**B1**"

   The recruiter asks for a candidate with a high level of written English but then asks for *B1*.

   NOTE: Check the levels we attribute according to different expressions in table 2.1. This table was based on the language levels according to the *Common European Framework of Reference for Languages* (table 2.2).

Expression	Example	Levels
Basic	*Basic knowledge of ... / Basic level of ...*	A1 - A2
Good	*Good knowledge of ... / Good level of ...*	B1 - C2
High	*High knowledge of ... / High level of ...*	B2 - C2
Fluency	*Fluency in ... / Fluent in ...*	B2 - C2
Proficient	*Proficient in ... / Proficient knowledge of ...*	C1 - C2

Table 2.1: Attribution of language levels according to different expressions

Level	Description
A1	Beginner
A2	Elementary
B1	Intermediate
B2	Upper Intermediate
C1	Advanced
C2	Mastery or Proficiency

Table 2.2: Language levels according to the *Common European Framework of Reference for Languages*

3. *info.en.jobRequirements.mustHave*: "High knowledge in **systems administration**"

   fieldsOfStudy: **philosophy**

   These two fields are contradictory, as the fields *system administration* and *philosophy* are not related.

Language-Not-Specified Contradictions

1. *info.en.jobRequirements.mustHave*: "Good knowledge of English and French."
   languages_1.code: en

The field *info.en.jobRequirements.mustHave* is contradictory because both English and French are mandatory languages, however, only English is specified in the structured fields.

2. *info.en.jobRequirements.mustHave*: "Good knowledge of English and French."
   languages_1.code: en
   languages_2.code: fr
   languages_3.code: pt

   The fields *info.en.jobRequirements.mustHave* and *languages_3.code* are contradictory because Portuguese is mandatory in the structured field, but it is not specified in the unstructured field.

Other Contradictions

1. *requireWorkPermit*: **TRUE**

   workingPermitIsKiller: **FALSE**

   info.en.jobRequirements.mustHave: "Already **hold the work permit** to accept an internship in Luxembourg."

   These fields are contradictory because in the *info.en.jobRequirements.mustHave* and *requireWorkPermit* the recruiter says that a work permit is required for the job, but then in *workingPermitIsKiller* says it is not an exclusion factor.

2.3 Ambiguities

An ambiguity occurs when a specific piece of text is ambiguous and does not express clearly its meaning, having multiple possible interpretations and causing uncertainty to the reader. Ambiguities can occur in the following cases:

- In an unstructured field;
- Between two different fields.

2.3.1 Types

Ambiguities can occur from several different sources, such as:

- **Syntactic ambiguity** - This ambiguity occurs when an expression has more than one possible structural analysis, and each has a different meaning. See examples in 2.3.2.

- **Vagueness** - A sentence is vague when it has a single meaning from a grammatical point of view, but still leaves room for different interpretations. If a requirement is vague, this means that it is not clear how to measure whether the requirement is fulfilled or not. See examples in 2.3.2
- **Other** - These ambiguities are a result of any other type of ambiguity not previously mentioned. See examples in 2.3.2

2.3.2 Examples of Possible Ambiguities in the Dataset

Syntactic Ambiguities

1. In the *info.en.jobRequirements.mustHave* column, the candidate is valued if the candidate is proficient in **Dutch and French or English**.

   Does this mean that the candidate should speak *Dutch and (French or English)* or *(Dutch and French) or English*? This ambiguity results from the different possible structural analysis.

2. In the *info.en.jobRequirements.mustHave* column, the recruiter asks for **an English history expert**.

   This expression can be read as *We are looking for an (English history) expert* or *We are looking for an English (history expert)*

Vagueness

1. *info.en.jobRequirements.mustHave*: "Already hold the **correct permit** to accept an internship in Luxembourg"

   This information is ambiguous as it is unclear what the correct permit is. Is it the necessary documents for working in Luxembourg or for doing an internship?

Other Ambiguities

1. *info.en.jobRequirements.niceToHave*: "Any previous **experiences related to the internship description** are a plus"

   reqExperience: **1-10**

   This information is ambiguous as it can have several interpretations. In the field *info.en.jobRequirements.niceToHave* the recruiter says experience related to the internship description is a plus. However, is the *1-10* years of *reqExperience* specified related to the internship description or is it the overall experience one candidate has? This experience is not specified in any other field.

NOTE: If it is specified in *info.en.jobRequirements.mustHave* the type of experience, then we assume that the type of experience in *reqExperience* is the same.
For example, if it said *It is needed experience related to the field*, then we assume the *reqExperience* is related to the field and not the overall experience one candidate has.

Chapter 3

brat annotation tool

brat is a web-based tool for text annotation. It is designed in particular for structured annotation, where the notes are not freeform text but have instead a fixed form that can be interpreted by a computer.

3.1 Features

There are many advantages in using *brat* tool, such as:

- It offers a comprehensive and **intuitive visualization** of the underlying annotation;
- Annotating is easily done and in an intuitive way. By simply using the mouse (selecting, dragging or double-clicking), it is possible to mark a span of text;
- It allows text annotation in **any language**. This is an important feature since Skeeled's software is multilingual;
- It allows **comparison** between multiple sets of annotations for the same documents, which includes an automatic comparison to identify and mark differences and side-by-side visualization;
- Each annotation has its own link;
- It implements a simple interface for integrating the output of automatic text annotation tools accessible as web services into annotation efforts;
- It allows **real-time collaboration**;
- Amongst many other features.

3.2 Download and Setup

NOTE: Annotators should skip this step and go directly to section 3.4.

Before you are able to use brat, you need to download it. Go the brat downloads page (http://brat.nlplab.org/downloads.html) and download the version 1.3. After downloading, you can go to the brat installation page (http://brat.nlplab.org/installation.html) and follow the steps mentioned there.

3.3 Data

NOTE: Annotators should skip this step and go directly to section 3.4.

After installing the *brat* tool, in the *brat*'s folder, you should see a folder named *bratdata*. We propose the folder hierarchy present in figure 3.1. The folder *skeeled* should contain sub-folders with the names of the different annotators. Each annotator has inside the respective folder several job postings that need to be annotated.

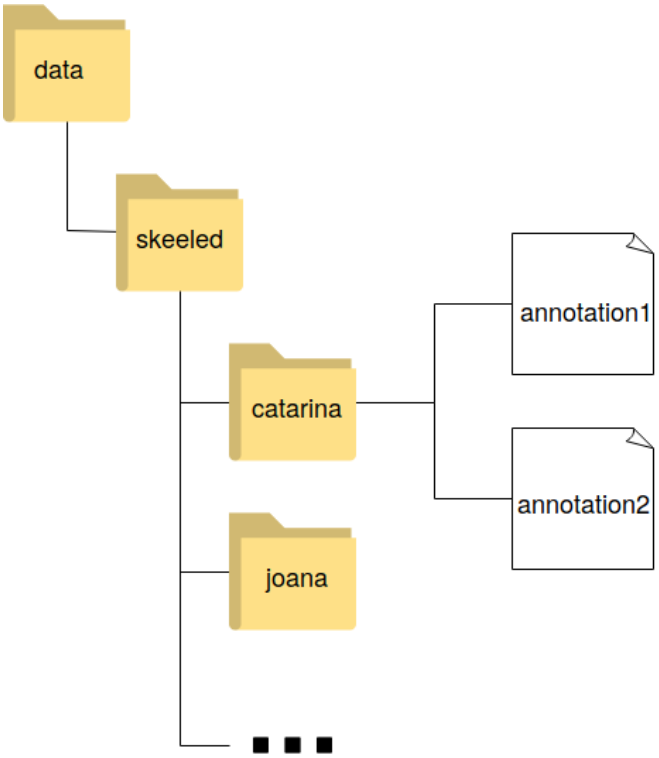


Figure 3.1: Folder structure

These job postings were generated with the script seen in **??**. This script reads a *.json* file that contains the data and converts each *json* object into two different files: an *.ann* file and a *.txt* file needed for annotating in *brat*.

3.4 How to Use

Step 1 Access `https://doodles.ci.skeeled.com`. A pop-up will appear asking for an username and password, respectively. After signing in, a welcome message (figure 3.2) will appear. Then read it and press *OK*.

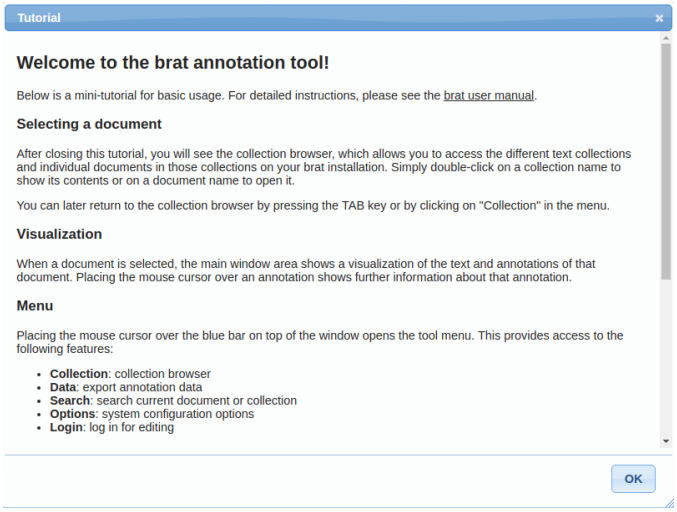


Figure 3.2: Welcome message

Step 2 Select the collection *skeeled - YOUR_NAME - ANNOTATION_-FILE* as seen in figure 3.9.

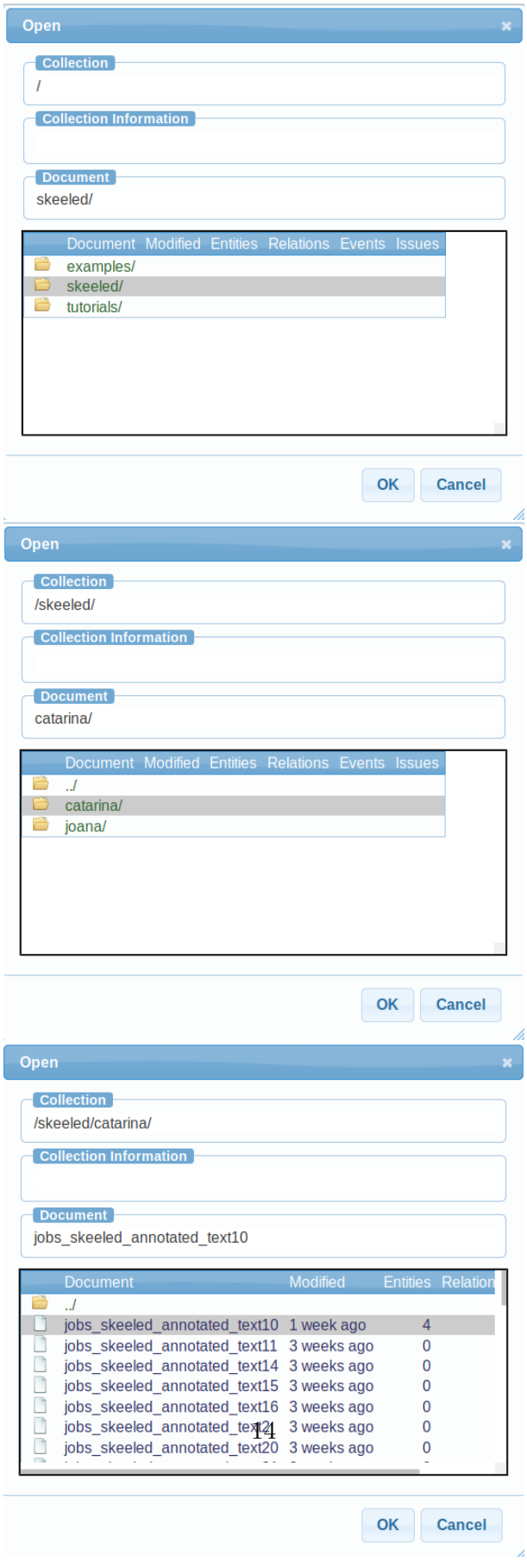


Figure 3.3: Open annotation

Step 3 Now, you need to log in by pressing the *login* button in the upper right corner (figure 3.4). Credentials will be provided to you. After that, you can start annotating.

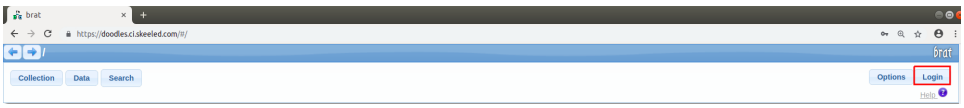


Figure 3.4: Select the ambiguous/contradictory expression

Create an annotation To mark text as contradictory or ambiguous, simply select the text. A pop-up will appear and you can choose whether it is a contradiction or an ambiguity (figure 3.5).

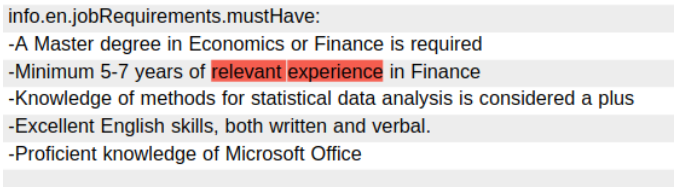


Figure 3.5: Select the ambiguous/contradictory expression

An ambiguity can occur inside the same field **(1)**, but it can also happen between a structured and an unstructured field **(2)**.

(1) In the first case, an **ambiguity** can be annotated by simply clicking in *Ambiguity* inside the *Entity type* box or by pressing *A* in the keyboard (figure 3.6). Then, you choose the type of ambiguity (figure 3.7). To conclude the ambiguity annotation click in the *OK* button (figure 3.8).

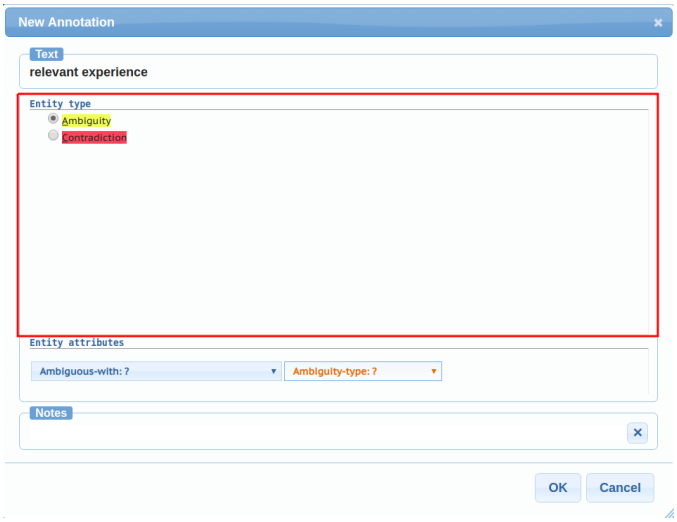


Figure 3.6: Choose the option *Ambiguity* in *Entity type*

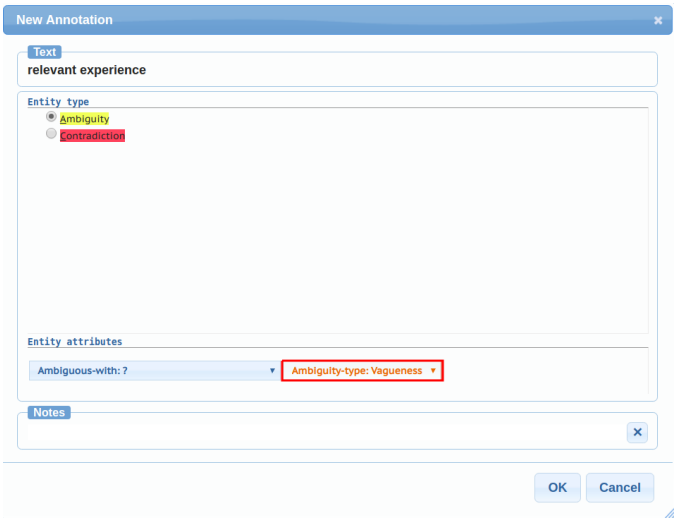


Figure 3.7: Choose the type of ambiguity in *Entity attributes*

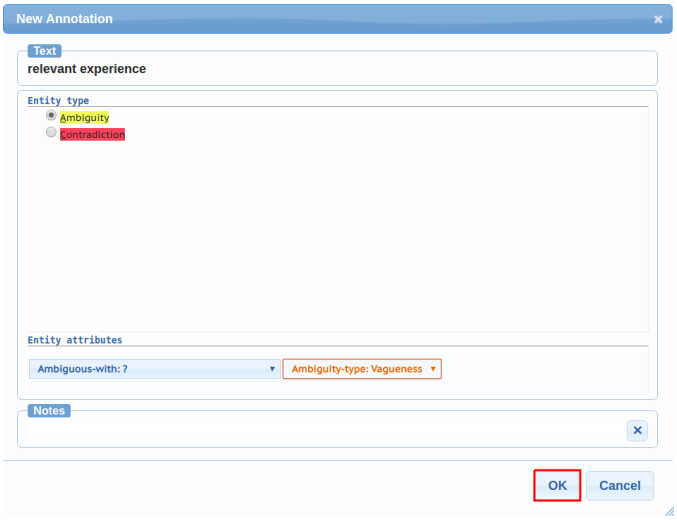


Figure 3.8: Select the *OK* button

(2) In the latter case, we select the ambiguous expression inside the unstructured field (figure **??**). Then the **ambiguity** is annotated by simply clicking in *Ambiguity* inside the *Entity type* box or by pressing *A* in the keyboard (figure 3.10). Then, select the structured field that causes ambiguity in *Ambiguous-with* (figure 3.11) and choose the type of ambiguity (figure 3.12) in *Entity attributes*. To conclude the ambiguity annotation click in the *OK* button (figure 3.13).

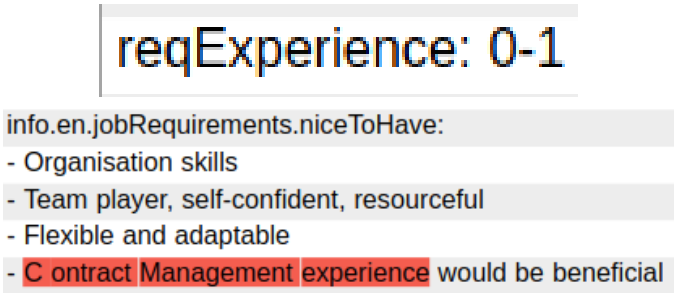


Figure 3.9: Select the ambiguous expression in the unstructured field

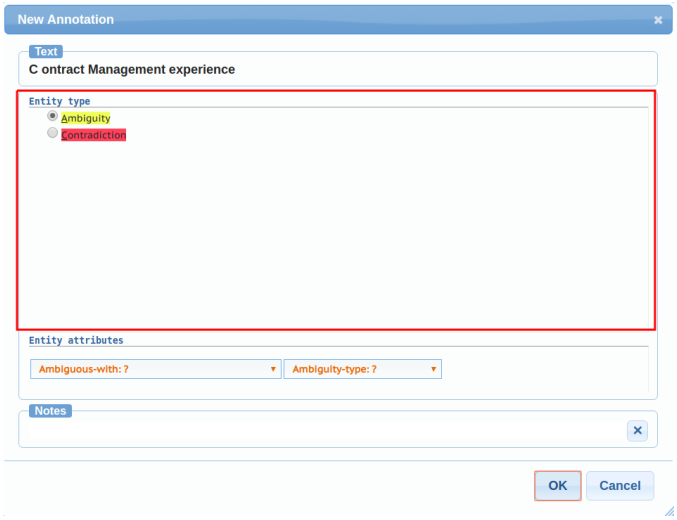


Figure 3.10: Select the option *Ambiguity* in *Entity type*

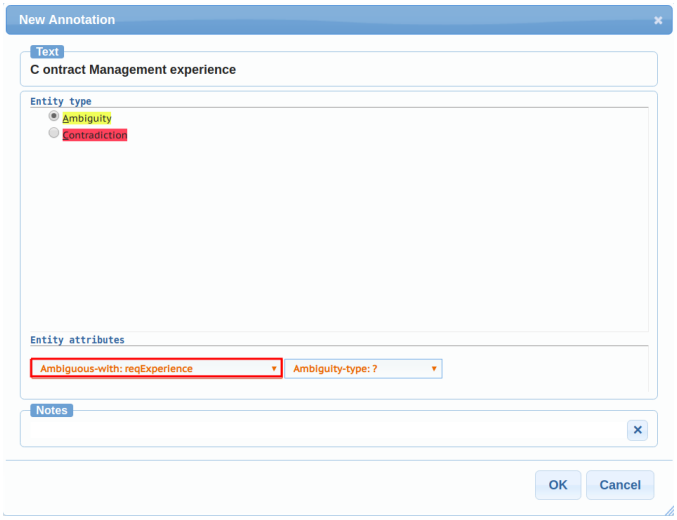


Figure 3.11: Choose the ambiguous field in *Ambiguous-with* in *Entity attributes*

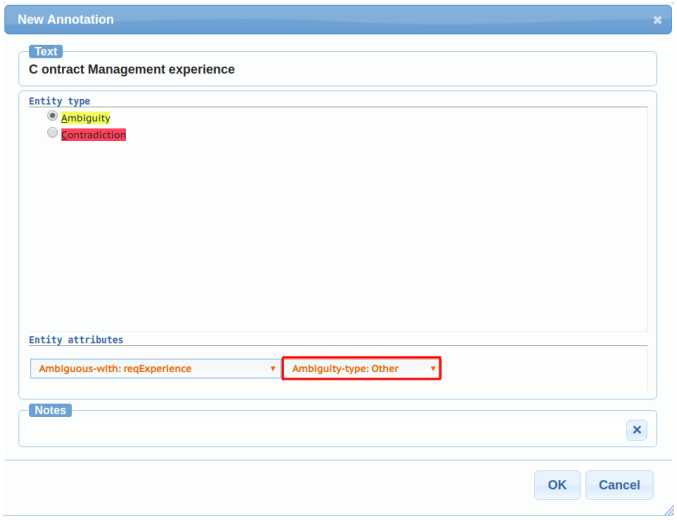


Figure 3.12: Choose the ambiguity type in *Ambiguity-type* in *Entity attributes*

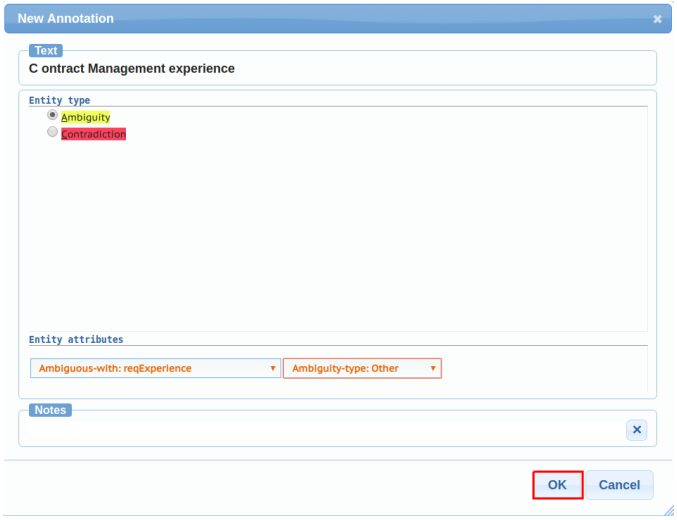


Figure 3.13: Select the *OK* button

A **contradiction** can be annotated by clicking in *Contradiction* inside the *Entity type* box or by pressing *C* in the keyboard. A contradiction occurs between two different expressions, as mentioned above. Because of this, there are certain rules that one must take into account when annotating contradictions:

- If a contradiction occurs between two expressions inside **two different unstructured fields** or **inside the same unstructured field**,

then the annotator needs to **mark** both expressions as *Contradiction* (figures 3.14 3.15 3.16 3.17 3.18 3.19) and then click in one of them (which one is irrelevant) and **drag** the arrow to the other contradictory expression (figure 3.20). Finally, press the *OK* button (figure 3.21), and observe the contradiction (figure 3.22).



Figure 3.14: Select one of the contradictory expressions

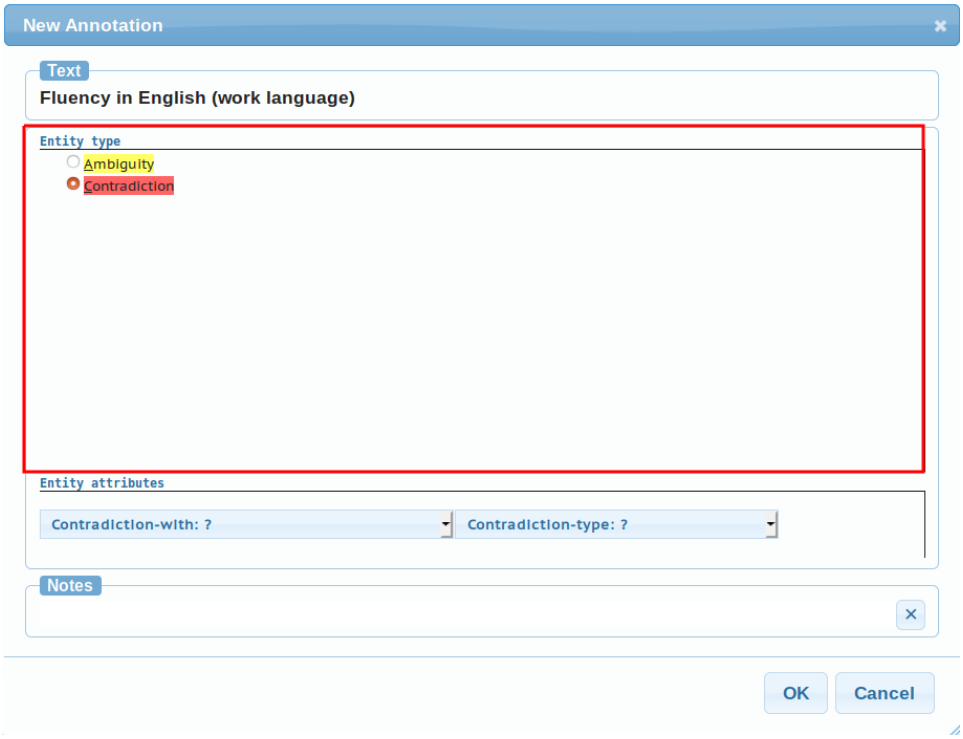


Figure 3.15: Select the option *Contradiction* in *Entity type*

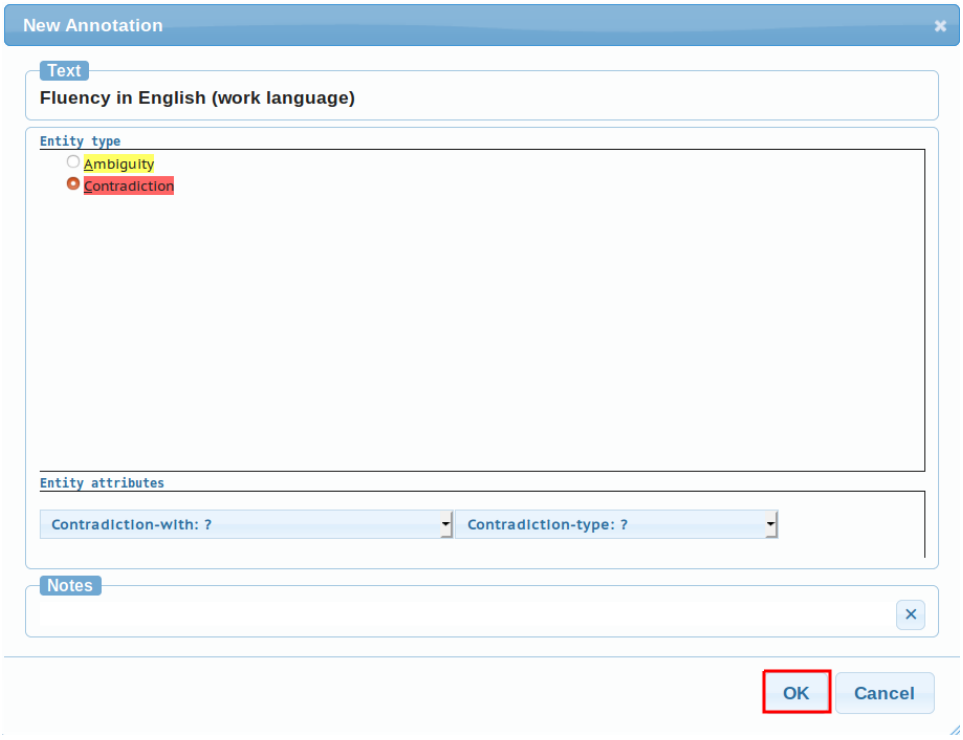


Figure 3.16: Press the *OK* button

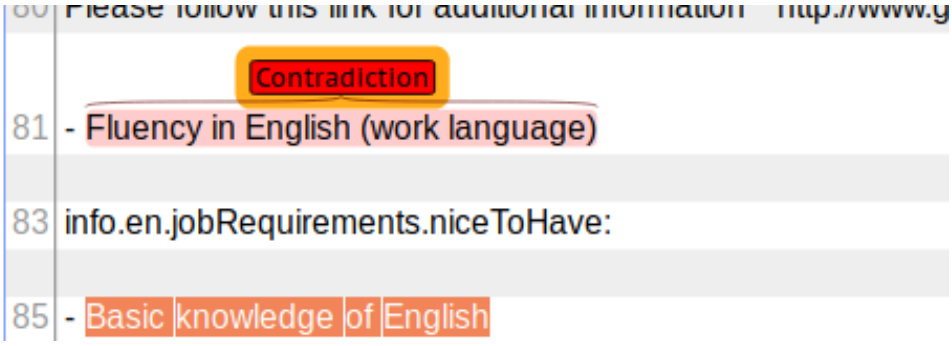


Figure 3.17: Select the other contradictory expressions

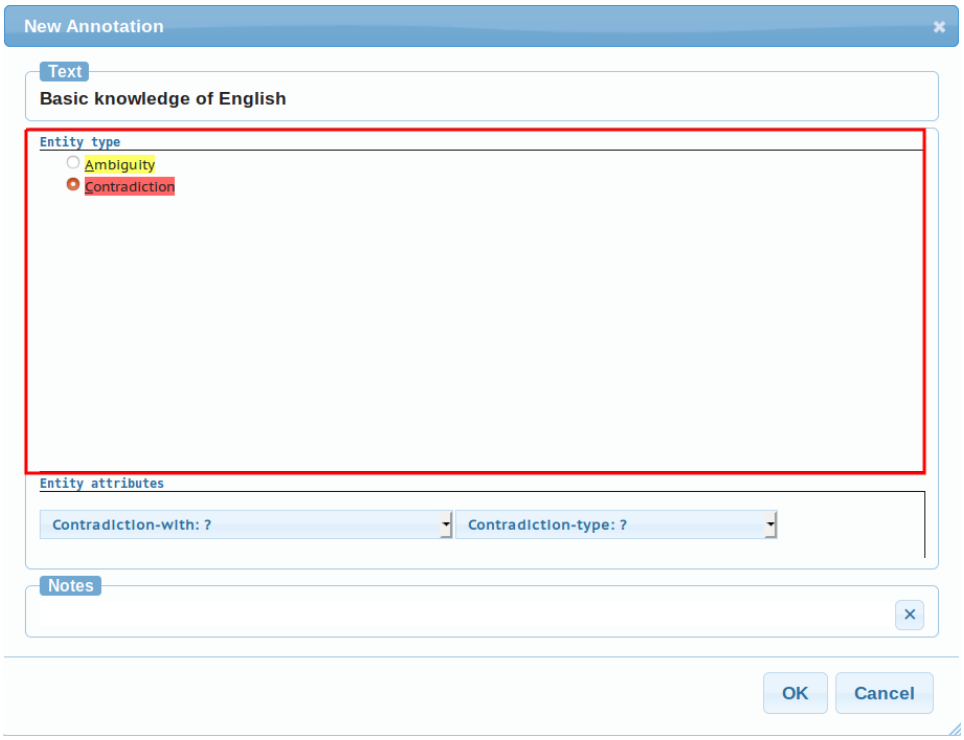


Figure 3.18: Select the option *Contradiction* in *Entity type*

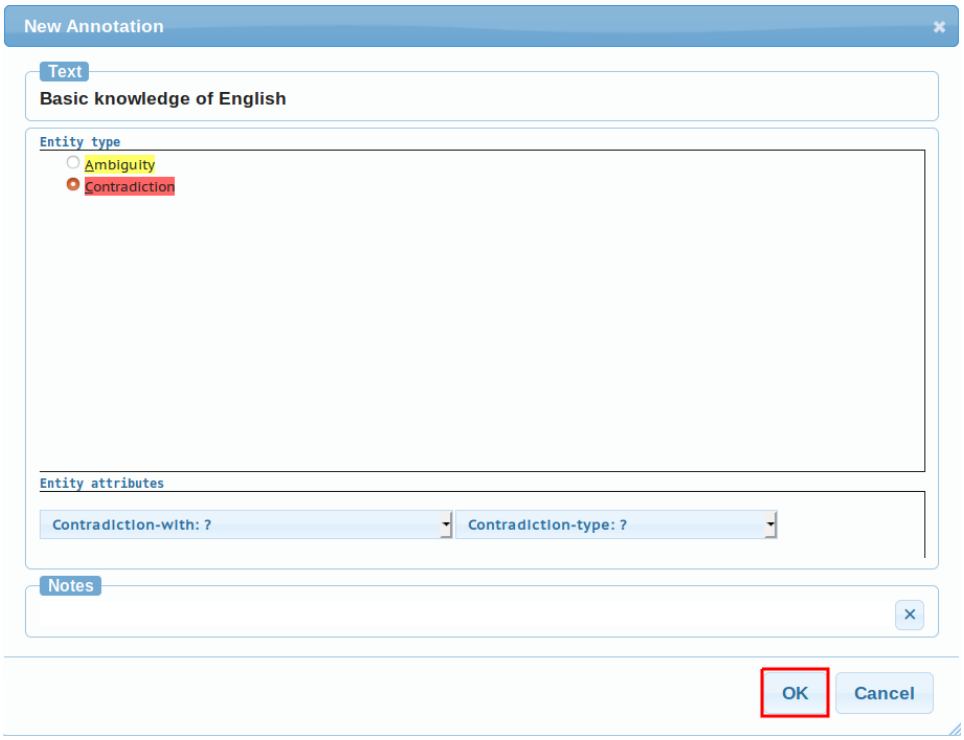


Figure 3.19: Press the *OK* button

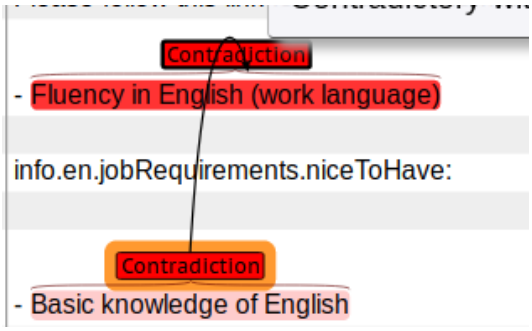


Figure 3.20: Click on one of the annotated contradictions and drag it to the other one

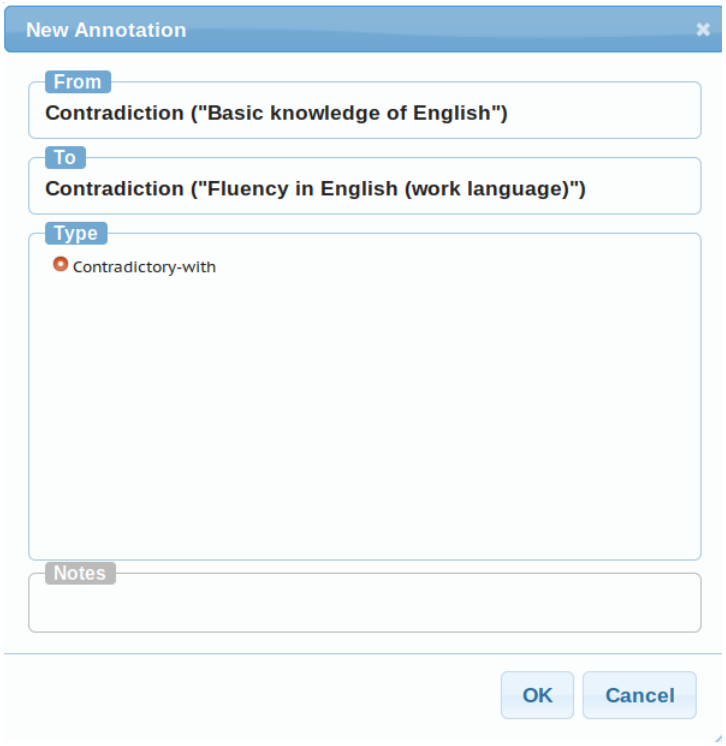


Figure 3.21: Press the *OK* button

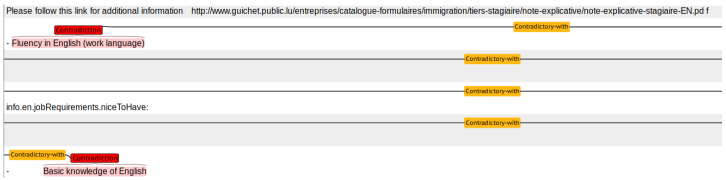


Figure 3.22: Observe the contradictory expressions

- If a contradiction occurs between **two different structured fields**, then the annotator just needs to select one of the contradictory expressions (no matter which one) (figure 3.23), mark it as a contradiction (figure 3.24) and select the contradictory field in *Contradiction-with* (figure 3.25) in the *Entity attributes* section and the type of contradiction in *Contradiction-type* (figure 3.26). At last, press *OK* as seen in figure 3.27.

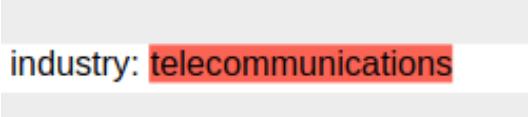


Figure 3.23: Select the contradictory value of one of the contradictory structured fields

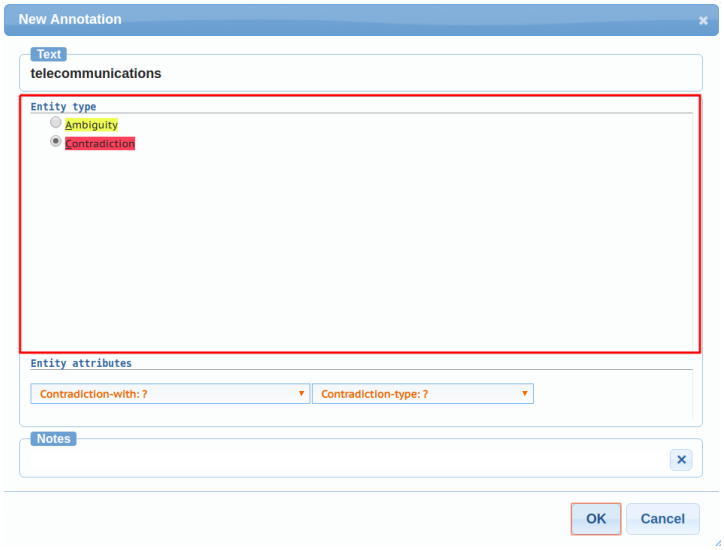


Figure 3.24: Select the option *Contradiction* in the *Entity type*

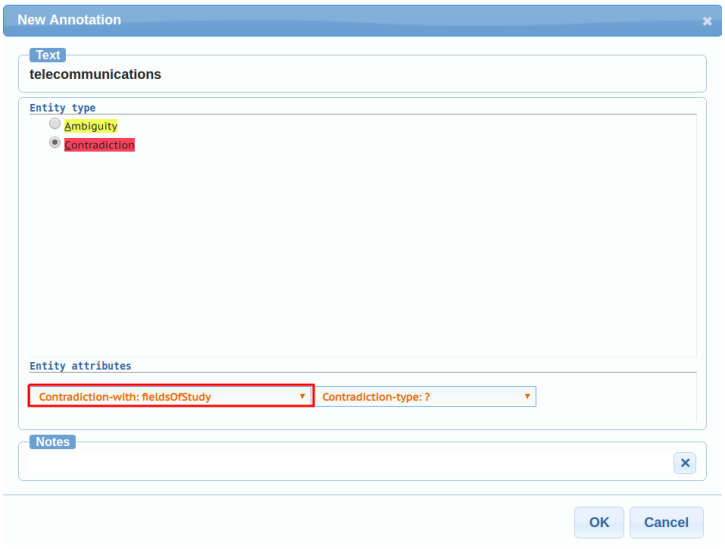


Figure 3.25: Select the contradictory field in *Contradiction-with* in *Entity attributes*

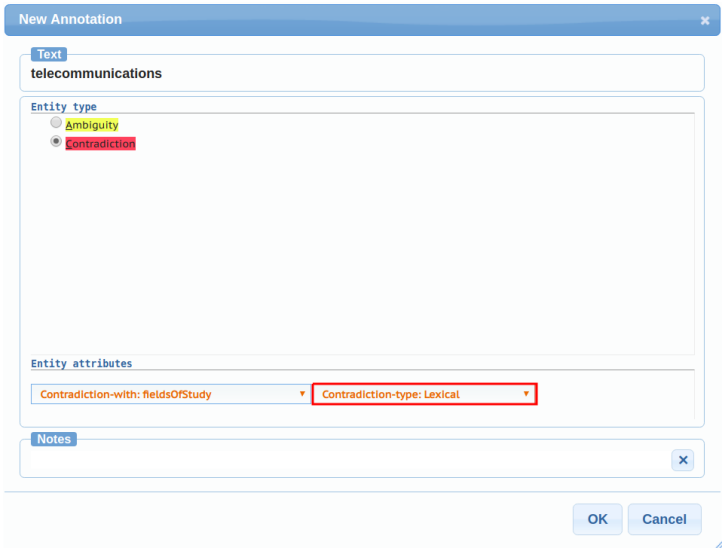


Figure 3.26: Select the type of contradiction in *Contradiction-type* in *Entity attributes*

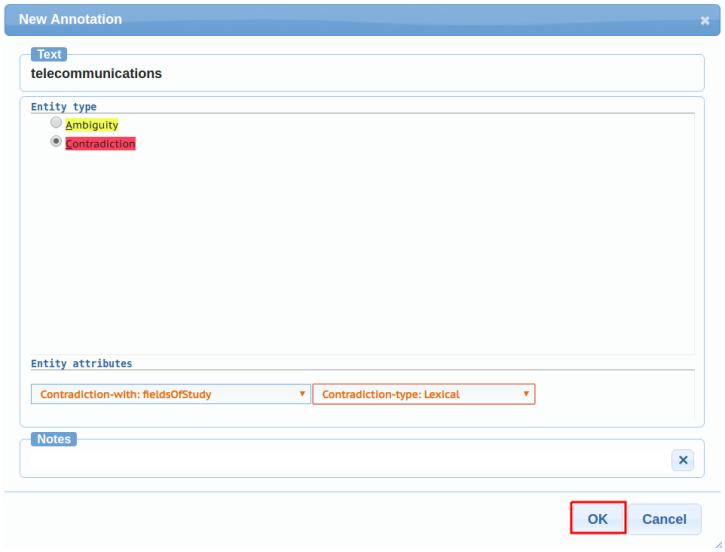


Figure 3.27: Press the *OK* button to conclude the annotation

- If a contradiction occurs between two different fields, **one structured and the other unstructured**, then the annotator needs to select the contradictory expression in the unstructured field, mark it as a contradiction and select the contradictory field in *Contradiction-with* in the *Entity attributes* section. For better understanding, examples are shown in section 3.4.1.

Note: One expression can be marked as both a contradiction and an ambiguity.

Edit an annotation To edit an annotation, double click the annotation and after you are done editing, press the *OK* button.

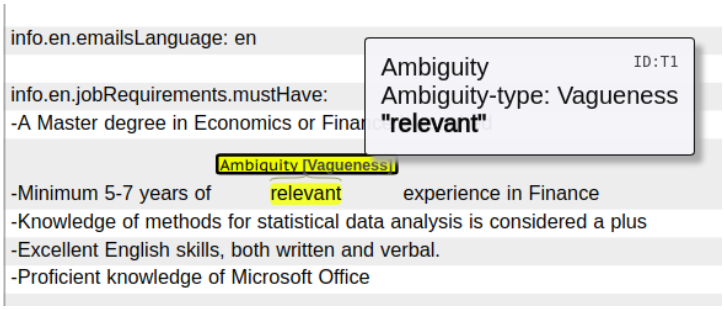


Figure 3.28: Double click the annotation

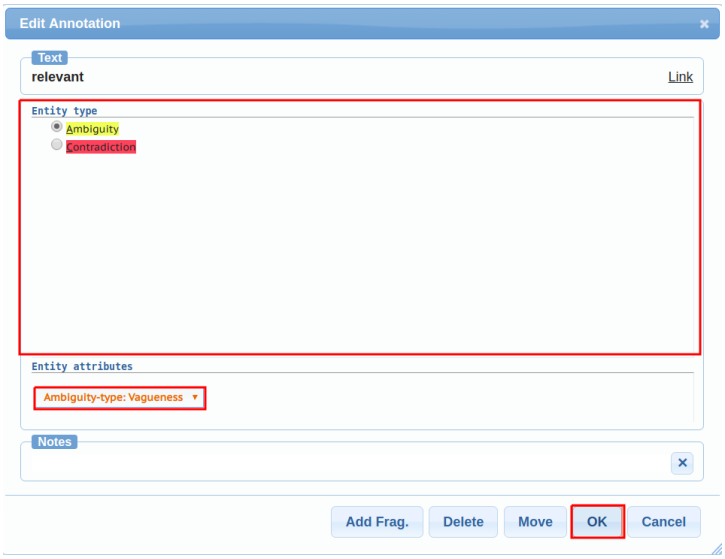


Figure 3.29: Edit the marked values and then press the *OK* button

Delete an annotation To delete an annotation, double click it and press the *Delete* button.

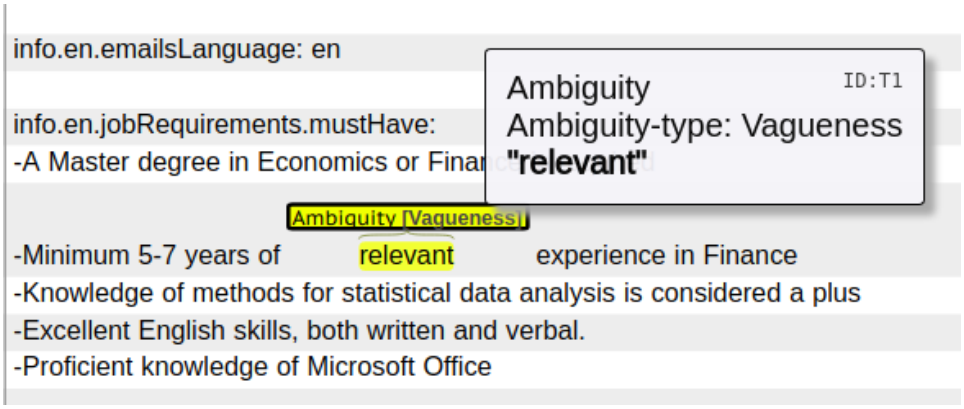


Figure 3.30: Double click the annotation

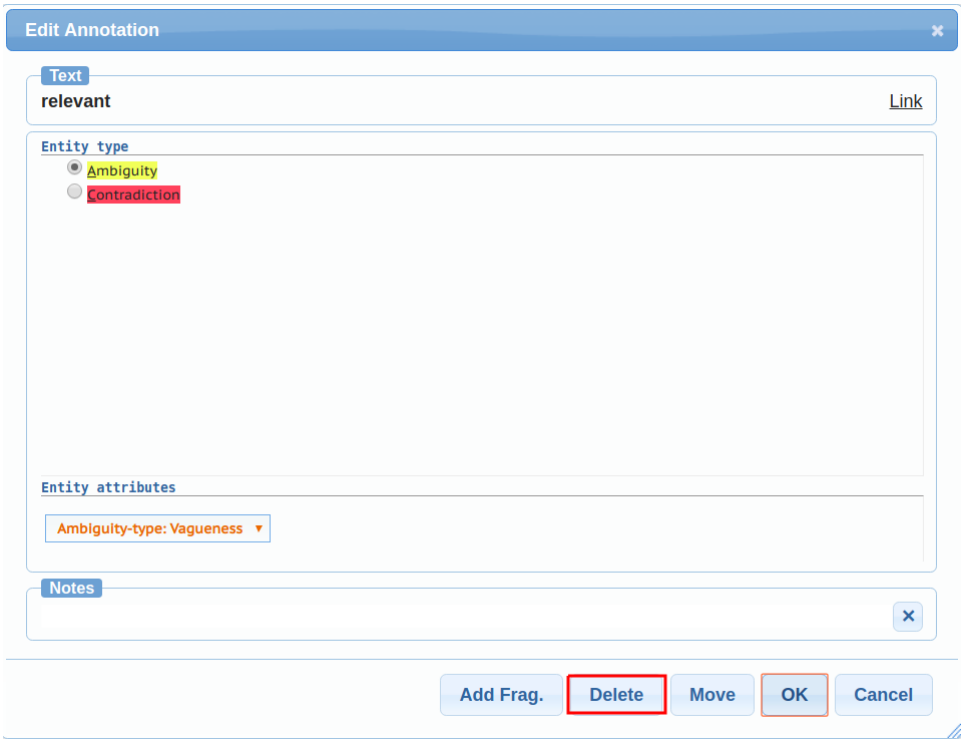


Figure 3.31: Press the "Delete" button

3.4.1 Examples of Annotations in the Dataset

Contradiction 1

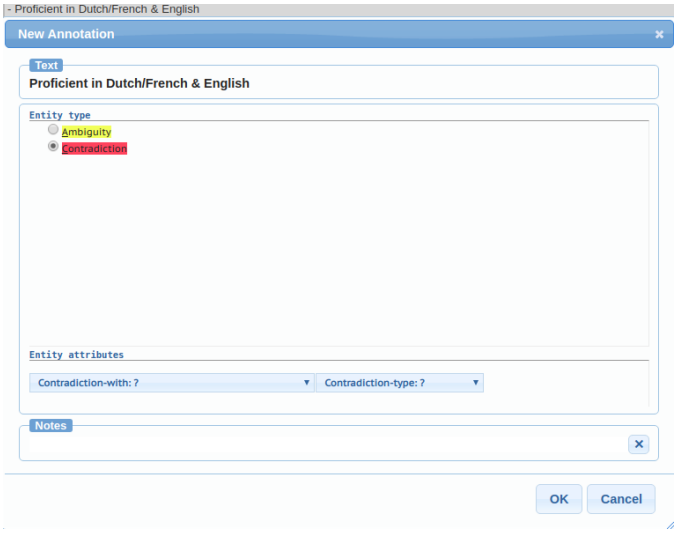


Figure 3.32: **Step 1** - Select the contradiction expression in the unstructured field *Proficient in Dutch/French & English*



Figure 3.33: **Step 2** - Select the contradictory structured field in *Contradiction-with*

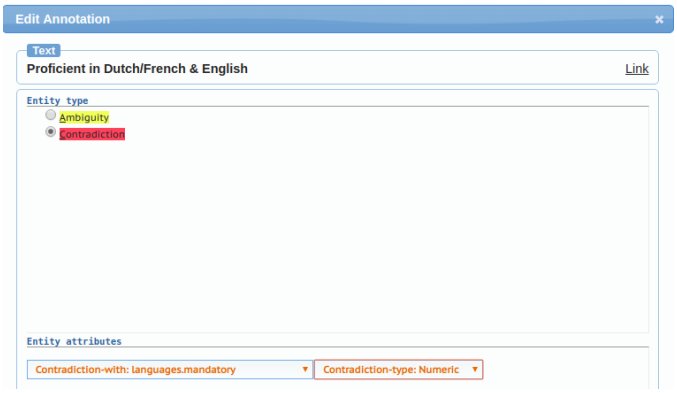


Figure 3.34: **Step 3** - Select the type of contradiction in *Contradiction-type*

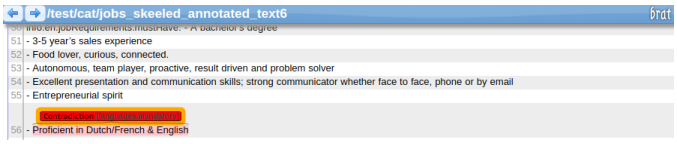


Figure 3.35: **Step 4** - Press the *OK* button and observe the annotated contradiction

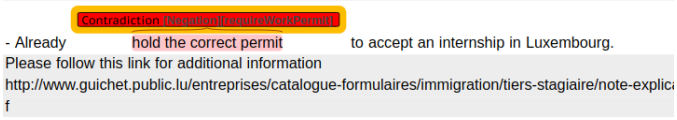


Figure 3.36: Step 6

Ambiguity 1

Step 1 Select the ambiguous expression:

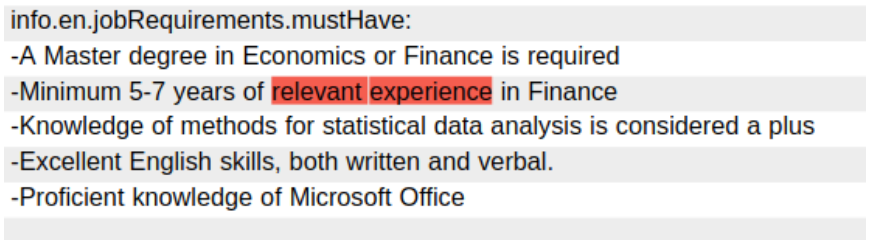


Figure 3.37: Step 1

Step 2 Select the *Entity type* "Ambiguity" or press the letter "A":

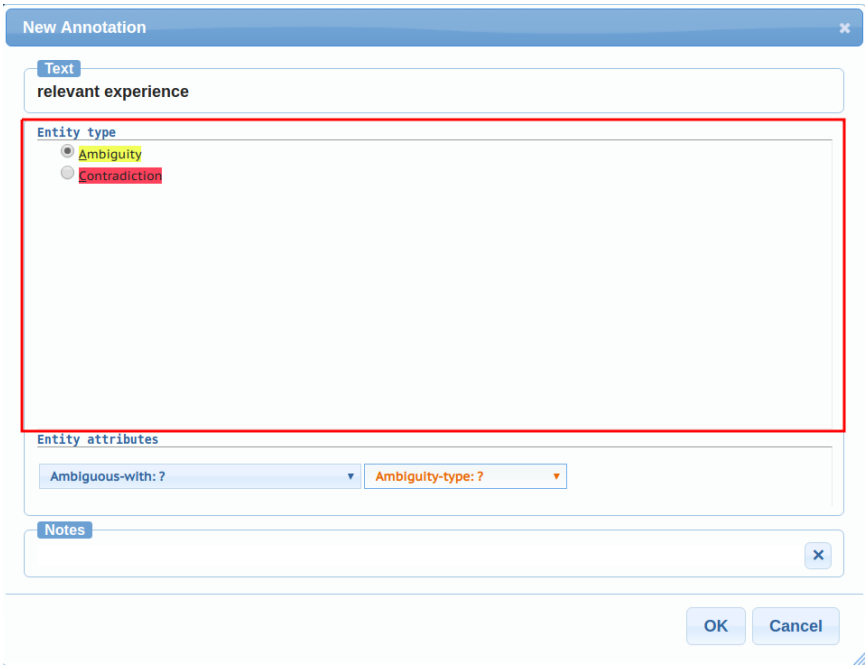


Figure 3.38: Step 2

Step 3 Select the type of ambiguity in *Entity attributes*:

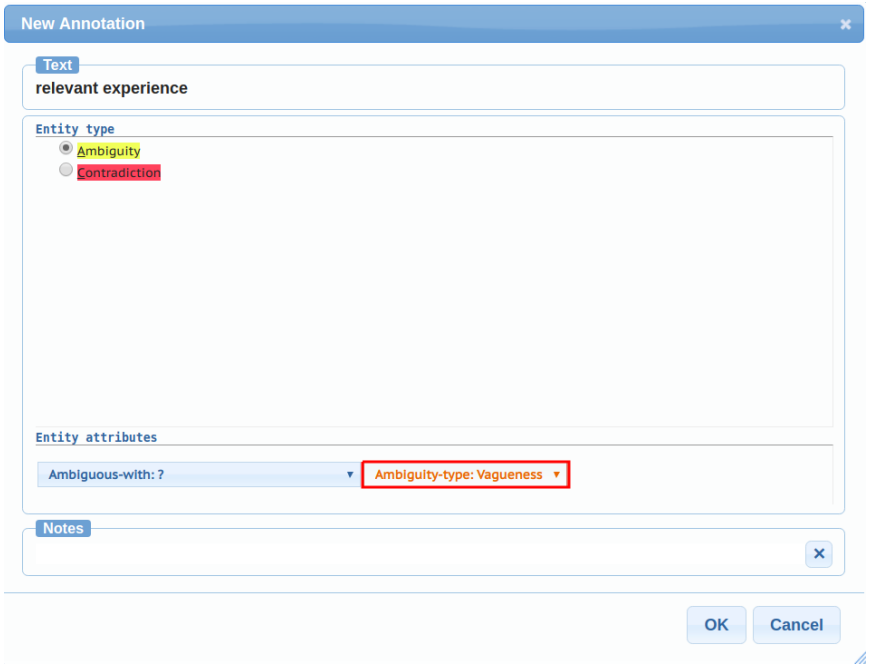


Figure 3.39: Step 3

Step 4 Click on the *OK* button:

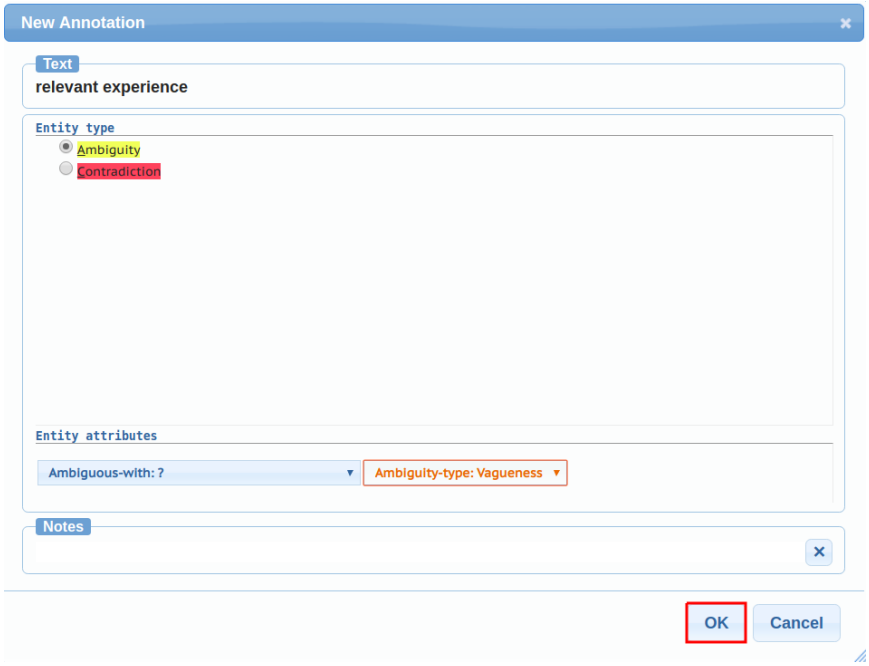


Figure 3.40: Step 4

Step 5 Now you can observe the ambiguity in the text:

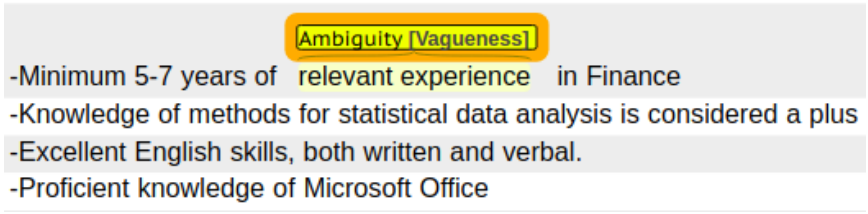


Figure 3.41: Step 5

Glossary

world-knowledge The non-linguistic information that helps a reader or listener interpret the meanings of words and sentences. 4, 5