ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

BACHELOR RESEARCH ASSIGNMENT

Argument Mining

Author:
Klio Fragkedaki

Supervisor:
Prof. Panagiotis Louridas

An assignment submitted as part of
Bachelor degree

in the

Department of Management Science and Technology

September 26, 2019

Contents

List of Abbreviations

SVM	**S**upport **V**ector **M**achine
LR	**L**ogistic **R**egression
NB	**N**aive **B**ayes classifier
RF	**R**andom **F**orest
RNN	**R**ecurrent **N**eural **N**etworks for Language Models
RF	**R**andom **F**orest
CRF	**C**onditional **R**andom **F**orest
ML	**M**aximum **L**ikelihood
TES	**T**extual **E**ntailment **S**uites
P	**P**arsing Using a Context-Free Grammar
LSTM	**L**ong **S**hort-**t**erm **M**emory
POS	**P**art **O**f **S**peech

Chapter 1

Introduction

1.1 Definition

Argument mining is a relatively new research field in natural language processing. The aim of this research is the auto detection and identification of argumentative structures expressed in text. In order to perform extraction and evaluation of arguments, computer science and artificial intelligence is used.

An argument is a group of premises conducted to support a claim (Palau and Moens, 2009). When it comes to real world, arguments are hardly identified even by experts (Lippi and Torroni, 2015). The ambiguity of natural language, the implicit content, the different ways of expressing and the complex structure of arguments are the main reasons why argument mining is a challenging research field. Labeled corpora are scarce which is a fact that slows down field's potential growth (Lippi and Torroni, 2015).

The purpose of argument mining is to understand what kind of views have been expressed in the examined text and why they are held. Argument mining has derived from opinion mining and sentiment analysis research area, in which the only goal is to understand the opinions about a certain topic (Lawrence and Reed, 2015).

1.2 Research Goal

My research goal is to identify argumentative statements by using two different approaches; the structural approach which is based in hand coded rules and the machine learning approach.

The structural approach uses lexical cues that have been identified by linguists as signs of argumentative speech. As an example, words such as "because", "therefore", "in order to" are common cues of arguments. However, these argumentative patterns are rarely used in practice, since human discourse involves a lot of information which is being implied rather than being explicitly stated.

On the other hand, the machine learning approach relies on examples of pieces of text that have been manually labeled as argumentative or non-argumentative. These are used for training models in order to automatically identify arguments in free text without the use of predefined lexical cues and rules. The challenging part is the construction of a manually annotated data-set, given the fact that a large amount of data are required for training such models.

The fundamental research questions that will be addressed in this assignment are the following:

- To what extent are the lexical rules drafted by a structural approach capable of successfully identifying arguments in existing resources of labeled data?
- Do the statistical approaches outperform these results?

1.3 Assignment's Structure

This paper of research is organized into 6 chapters. Chapter 2 presents the state of the art in argument mining, and introduces the two different approaches; the structural and the machine learning approach. Chapters 3 and 4 describe in detail the methods and results of both approaches implemented in the scope of this study. Chapter 5 contains the corpora created for the supervised algorithm, while chapter 6 concludes with a look to future work.

Chapter 2

State-of-the-Art

Arguments do not have a universally accepted definition; though there are plenty of well-described proposals. According to (Walton, 2009), an argument is a group of statements which splits into three portions, which are conclusion, set of premises, and an inference leading from premises to conclusion. These concepts have been widely accepted in literature, but they are defined in slightly different ways. Conclusions are also referred to as claims, premises as evidence or reasons, while the link between claims and evidence is the argument (Lippi and Torroni, 2015).

A claim is supported or argued by one or more premises and it is the main part of an argumentative text. Claims are controversial in terms of validity and need premises to endorse readers' acceptance (Stab and Gurevych, 2014). Argumentation schemes and their common patterns provide a way to both identify and determine arguments (Lawrence and Reed, 2015).

The term of argumentation used to be connected with the process of argument construction (Lippi and Torroni, 2016). After the emergence of text mining procedures, this term defines the process of argument identification in text (Lippi and Torroni, 2016). The research field of argument mining is about the automatic recognition of argumentative structures expressed in natural language texts. Argument mining utilizes methods and techniques used in natural language processing, such as machine learning and sentiment analysis (Lippi and Torroni, 2015).

In general, argument mining procedure is separated into linguistic and computational part, as described in figure 2.1. Regarding the linguistic part, large corpora of manually annotated argument data are being created based on a common agreement among annotators about argument's structure. On the other hand, the computational part is separated into two main styles of automation, the structural and the statistical approach. (Budzynska and Villata, 2015)

In **structural or grammar approach**, linguists aim to retrieve lexical patterns, rules or categories while annotating a training corpus. For example, it might be noticed that words like "because", "since", "however" are signs of arguments inside a specific corpus (Budzynska and Villata, 2015). These signs are called indicators, and point out the connection between claims and premises inside a text (Lawrence and Reed, 2015). Indicators are declared as linguistic expressions that connect statements and provide an unambiguous recognition of argumentative structure (Webber, Egg, and Kordoni, 2012).

A lot of research has been applied in order to be found words and expressions revealing argumentative structure (Van Eemeren, Houtlosser, and Henkemans, 2007, Knott and Dale, 1994). Apart from indicators, other structural techniques have been applied for argument mining. Such techniques are argumentation schemes (Feng and Hirst, 2011), dialogical context (Budzynska et al., 2014), and semantic context (Cabrio and Villata, 2012) or a combination of them.

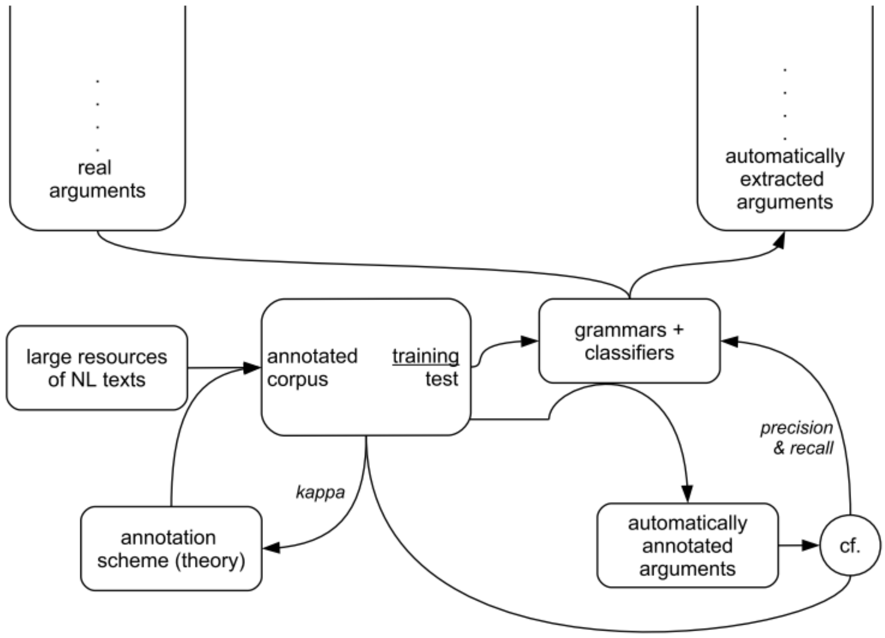

FIGURE 2.1: Natural language processing techniques
Source: Budzynska and Villata, 2015

In **statistical approach**, linguists are replaced by algorithms. These algorithms are basically classifiers developed for automating the argument annotation procedure (Budzynska and Villata, 2015). The first attempts for the before mentioned automation were made in (Moens et al., 2007), in which text is separated into sentences, and then each sentence is classified as argumentative or non-argumentative based on its lexical or syntactic features. As a result, (Palau and Moens, 2009) presented an additional separation of argumentative sentences as premises or conclusions. As regards the automatic recognition of argumentative schemes, it was introduced in (Walton, 2011) and it was based on the idea of connecting each scheme with a group of indicators. The paper's proposal is first indicating the arguments included in text, and then matching them to a given list of argument schemes. (Feng and Hirst, 2011) classifies annotated argumentation structures into a list of five common argumentation schemes. In (Lippi and Torroni, 2015), the authors describe a framework for claim detection in unstructured data-sets without any contextual information. Because arguments are often expressed through rhetorical structures, the previously mentioned framework was built based on an SVM classifier which captures similarities among parse trees via Tree Kernels. This method is used for measuring likeliness of two trees regarding their common substructures. Furthermore, Habernal and Gurevych (Habernalt and Gurevych, 2016) try to evaluate argument convincingness by assessing their qualitative properties. Using an annotated corpus of 26,000 sentences, their purpose is to predict which argument is more convincing between a pair of arguments and to rank arguments regarding the topic and their convincingness, through the usage of SVM and LSTM algorithms.

Various traditional machine learning algorithms have been employed in the context of argument mining (Figure 2.2). More specifically, most of the algorithms that have been implemented are Support Vector Machines (Mochales and Moens, 2011; Park and Cardie, 2015; Stab and Gurevych, 2014; Eckle-Kohler, Kluge, and Gurevych, 2015), Logistic Regression (Levy et al., 2014; Rinott et al., 2015), Naive Bayes classifiers (Mochales and Moens, 2011; Biran and Rambow, 2011; Park and Cardie, 2015; Eckle-Kohler, Kluge, and Gurevych, 2015), Maximum Entropy classifiers (Mochales and Moens, 2011), and Decision Trees and Random Forests (Stab and Gurevych, 2014; Eckle-Kohler, Kluge, and Gurevych, 2015). All mentioned classifiers are trained in labeled corpora. Thus, some parts of the annotated text are given, alongside with the associated label, and during training stage a model is being produced. This model is used to perform predictions on new unlabeled text. (Lippi and Torroni, 2016)

	SC							BD		SP			
System	SVM	LR	NB	ME	DT	RF	RNN	CRF	ML	TES	P	SVM	NB
Eckle-Kohler et al. [2015]	X		X			X							
Lippi and Torroni [2015]	X												
Rinott et al. [2015]		X											
Sardianos et al. [2015]	X						X	X					
Boltuzic and Snajder [2014]										X		X	
Goudas et al. [2014]	X							X					
Levy et al. [2014]		X							X				
Stab and Gurevych [2014b]	X		X		X	X						X	
Cabrio and Villata [2012a]										X			
Rooney et al. [2012]	X												
Biran and Rambow [2011]			X								X		X
Mochales Palau and Moens [2011]	X		X	X							X		

FIGURE 2.2: Machine learning algorithms that have been used for argument mining
Source: Lippi and Torroni, 2016

Despite the fact that researchers have tried to make a comparison between these algorithms, there is no clear proof of which classifier is more appropriate for argumentation mining. In fact, most of the research efforts have been settled down on finding appropriate features for improving performance instead of implementing new specifically designed models and algorithms for solving argument identification problem (Lippi and Torroni, 2016).

To sum up, a number of different approaches have been applied to argument identification problem. The research community solutions are ranging from linguistic techniques (Garcia Villalba and Saint-Dizier, 2012) and topic modeling (John Lawrence, Chris Reed, Colin Allen, Simon McAlister, Andrew Ravenscroft, 2014), to supervised machine learning algorithms(firstly implemented by Moens et al., 2007).

Chapter 3

Methods

In this research paper, we attempt to apply two different approaches for recognizing argumentative sentences. These approaches cover both a structured methodology, which is related to the selection of hand-coded linguistic rules, and a statistical one, that includes the implementation of supervised algorithms; namely, Random Forest classifier and sequence classification with LSTM.

3.1 Structural Approach

The structural approach is based on lexical cues, rules or patterns for identifying arguments inside a given text. These cues are also referred to as argument indicators, since they are connecting claims and premises, signaling argumentative relations.

<table>
<tr><th colspan="8">Argumentative Indicators based on (Knott and Dale, 1994)</th></tr>
<tr><th>Indicator</th><th>POS</th><th>Indicator</th><th>POS</th><th>Indicator</th><th>POS</th><th>Indicator</th><th>POS</th></tr>
<tr><td>even though</td><td>none</td><td>first</td><td>adv</td><td>against</td><td>none</td><td>last</td><td>adv</td></tr>
<tr><td>naturally</td><td>none</td><td>most</td><td>{"[a-z]*ly": "adv"}</td><td>if</td><td>none</td><td>(T | t)(he more).+?(the more)</td><td>none</td></tr>
<tr><td>once more</td><td>none</td><td>more</td><td>{"[a-z]*ly": "adv"}</td><td>once again</td><td>none</td><td>(T | t)(he more).+?(the less)</td><td>none</td></tr>
<tr><td>surely</td><td>none</td><td>second</td><td>adv</td><td>so</td><td>mark</td><td>third</td><td>adv</td></tr>
<tr><td>should say</td><td>none</td><td>too</td><td>(too)($ | [\\.])</td><td>might say</td><td>none</td><td>may say</td><td>none</td></tr>
<tr><td>could say</td><td>none</td><td>while</td><td>mark</td><td>as a start</td><td>none</td><td>in order to</td><td>none</td></tr>
<tr><td>still</td><td>adv</td><td>that is</td><td>none</td><td>since</td><td>mark</td><td>yet</td><td>(Y | y)(et)[\̂\.].</td></tr>
<tr><td>that</td><td>mark</td><td>above all</td><td>none</td><td>actually</td><td>none</td><td>after all</td><td>none</td></tr>
<tr><td>afterwards</td><td>none</td><td>all in all</td><td>none</td><td>also</td><td>none</td><td>although</td><td>none</td></tr>
<tr><td>anyway</td><td>none</td><td>as a consequence</td><td>none</td><td>as a result</td><td>none</td><td>at any rate</td><td>none</td></tr>
<tr><td>at first blush</td><td>none</td><td>at first view</td><td>none</td><td>at the outset</td><td>none</td><td>because</td><td>none</td></tr>
<tr><td>by comparison</td><td>none</td><td>by the same token</td><td>none</td><td>certainly</td><td>none</td><td>consequently</td><td>none</td></tr>
<tr><td>correspondingly</td><td>none</td><td>despite the fact that</td><td>none</td><td>either</td><td>none</td><td>equally</td><td>none</td></tr>
<tr><td>even then</td><td>none</td><td>every time</td><td>none</td><td>except insofar as</td><td>none</td><td>firstly</td><td>none</td></tr>
<tr><td>for a start</td><td>none</td><td>for instance</td><td>none</td><td>further</td><td>none</td><td>for the simple reason</td><td>none</td></tr>
<tr><td>accordingly</td><td>none</td><td>admittedly</td><td>none</td><td>after that</td><td>none</td><td>all the same</td><td>none</td></tr>
<tr><td>alternatively</td><td>none</td><td>always assuming that</td><td>none</td><td>as</td><td>none</td><td>as a corollary</td><td>none</td></tr>
<tr><td>at first</td><td>none</td><td>at first sight</td><td>none</td><td>at the moment when</td><td>none</td><td>at the same time</td><td>none</td></tr>
<tr><td>but</td><td>none</td><td>by contrast</td><td>none</td><td>by the way</td><td>none</td><td>clearly</td><td>none</td></tr>
<tr><td>conversely</td><td>none</td><td>despite that</td><td>none</td><td>essentially</td><td>none</td><td>even so</td><td>none</td></tr>
<tr><td>eventually</td><td>none</td><td>except</td><td>none</td><td>finally</td><td>none</td><td>first of all</td><td>none</td></tr>
<tr><td>for example</td><td>none</td><td>for one thing</td><td>none</td><td>for this reason</td><td>none</td><td>furthermore</td><td>none</td></tr>
<tr><td>hence</td><td>none</td><td>in actual fact</td><td>none</td><td>in any case</td><td>none</td><td>in conclusion</td><td>none</td></tr>
<tr><td>in fact</td><td>none</td><td>in other words</td><td>none</td><td>in short</td><td>none</td><td>in sum</td><td>none</td></tr>
<tr><td>incidentally</td><td>none</td><td>instead</td><td>none</td><td>merely because</td><td>none</td><td>just as</td><td>none</td></tr>
<tr><td>meanwhile</td><td>none</td><td>it might appear that</td><td>none</td><td>as long as</td><td>none</td><td>as well</td><td>none</td></tr>
<tr><td>notably</td><td>none</td><td>moreover</td><td>none</td><td>of course</td><td>none</td><td>nevertheless</td><td>none</td></tr>
<tr><td>on one hand</td><td>none</td><td>not only</td><td>none</td><td>now that</td><td>none</td><td>no doubt</td><td>none</td></tr>
<tr><td>on the grounds that</td><td>none</td><td>on the assumption that</td><td>none</td><td>on the one side</td><td>none</td><td>on the other side</td><td>none</td></tr>
<tr><td>plainly</td><td>none</td><td>otherwise</td><td>none</td><td>so that</td><td>none</td><td>providing that</td><td>none</td></tr>
<tr><td>such that</td><td>none</td><td>secondly</td><td>none</td><td>sure enough</td><td>none</td><td>simply because</td><td>none</td></tr>
<tr><td>thereafter</td><td>none</td><td>summing up</td><td>none</td><td>therefore</td><td>none</td><td>suppose that</td><td>none</td></tr>
<tr><td>thirdly</td><td>none</td><td>the fact is that</td><td>none</td><td>to be sure</td><td>none</td><td>though</td><td>none</td></tr>
<tr><td>to sum up</td><td>none</td><td>to conclude</td><td>none</td><td>undoubtedly</td><td>none</td><td>to take an example</td><td>none</td></tr>
<tr><td>whenever</td><td>none</td><td>to the extent that</td><td>none</td><td>whereas</td><td>none</td><td>what is more</td><td>none</td></tr>
<tr><td>wherever</td><td>none</td><td>for the reason that</td><td>none</td><td>besides</td><td>none</td><td>(E | e)(ither).+?(or)</td><td>none</td></tr>
<tr><td>in one hand</td><td>none</td><td>(N | n)(either).+?(nor)</td><td>none</td><td>on one side</td><td>none</td><td>in this case</td><td>none</td></tr>
<tr><td>in point of fact</td><td>none</td><td>as a matter of fact</td><td>non</td><td>provided that</td><td>none</td><td>presumably</td><td>none</td></tr>
<tr><td>rather than</td><td>none</td><td>regardless</td><td>none</td><td>as an example</td><td>none</td><td>simply</td><td>none</td></tr>
<tr><td>in order that</td><td>none</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

A list of indicators were extracted from the corpus created by (Knott and Dale, 1994). This corpus includes often-used words or phrases in arguments according to paper's authors. Based on these words, a dictionary was developed containing as keys the extracted words, and as values, their specific part of speech in argumentative sentences. It needs to be mentioned that words, considered by us as usual or non-usual in argumentative structures, were added or removed respectively from the dictionary. For this purpose, there were created five methods in Python for paper's extraction, modification, as well as dictionary's creation (Appendix 6). The indicators that demonstrate the previously referred dictionary is presented in the table above.

Apart from dictionary's development, a way to handle and encapsulate corpora into the same format was necessary, and the code developed for this purpose is shown in Appendix 7. Each data-set was differently displayed, from unstructured text to sentence labeled data. This is the reason why there was created a *datasets.ini* file containing information about data, for example the number of column indicating the sentence or/and the label, which sheet includes the desired data, or which is the data-set's path. The key of each record was the name of every corpora as it was saved in local file. So, depending on the data-set's type (excel, csv or txt file) and its configurations, other actions were applied in order to returned a list of sentences and their labels in case corpora was annotated.

By using the previously created dictionary and corpora handler, argument identification had to take place (Appendix 8). For this reason, part of speech tagging was necessary, so as a sentence's words and their POS to be compared to those words included in the dictionary. Statements tokenization was achieved through the usage of a library called *spaCy*, which is an open-source NLP library written in Python and Cython, and it was selected due to its performance and efficiency comparing to other libraries. If any of matches between the dictionary and a given sentence occur, the sentence is characterized as argumentative, otherwise as non-argumentative. As regards the labeled corpora, the algorithm's outcomes and the given labels, which is considered to be the truth, are correlated so as four counters to be calculated; False Positives, False Negatives, True Positives and True Negatives. These counters are used for encountering precision, recall and f1_score, that are metrics for reviewing algorithm's results. These metrics will be presented in more details at Chapter 5.

3.2 Machine Learning Approach

3.2.1 randomForest.py

the Random Forest algorithm introduced a robust, practical take on decision-tree learning that involves building a large number of specialized decision trees and then ensembling their outputs. Random forests are applicable to a wide range of problemsyou could say that theyre almost always the second-best algorithm for any shallow machine-learning task. Chollet, 2017

3.2.2 LSTM

It can be understood as either a sequence of characters or a sequence of words, but its most common to work at the level of words. The deep-learning sequence-processing models introduced in the following sections can use text to produce a basic form of natural-language under-standing, sufficient for applications including document classification, sentiment analysis, author identification, and even question-answering (QA) (in a constrained context). Of course, keep in mind throughout this chapter that none of these deep-learning models truly understand text in a human sense; rather, these models can map the statistical structure of written language, which is sufficient to solve many sim-ple textual tasks. Deep learning for natural-language processing is pattern recognition applied to words, sentences, and paragraphs, in much the same way that computer vision is pattern recognition applied to pixels. Chollet, 2017

Collectively, the different units into which you can break down text (words, characters, or n-grams) are called tokens, and breaking text into such tokens is called tokenization. All text-vectorization processes consist of applying some tokenization scheme and then associating numeric vectors with the generated tokens. These vectors, packed into sequence tensors, are fed into deep neural networks. There are multiple ways to associate a vector with a token. In this section, Ill present two major ones: one-hot encoding of tokens, and token embedding (typically used exclusively for words, and called word embedding). Chollet, 2017

Note that Keras has built-in utilities for doing one-hot encoding of text at the word level or character level, starting from raw text data. You should use these utilities, because they take care of a number of important features such as stripping special characters from strings and only taking into account the N most common words in your dataset (a common restriction, to avoid dealing with very large input vector spaces). Chollet, 2017

A variant of one-hot encoding is the so-called one-hot hashing trick, which you can use when the number of unique tokens in your vocabulary is too large to handle explicitly. Instead of explicitly assigning an index to each word and keeping a reference of these indices in a dictionary, you can hash words into vectors of fixed size. This is typically done with a very lightweight hashing function. The main advantage of this method is that it does away with maintaining an explicit word index, which saves memory and allows online encoding of the data (you can generate token vectors right away, before youve seen all of the available data). The one drawback of this approach is that its susceptible to hash collisions: two different words may end up with the same hash, and subsequently any machine-learning model looking at these hashes wont be able to tell the difference between these words. The likelihood of hash collisions decreases when the dimensionality of the hashing space is much larger than the total number of unique tokens being hashed. Chollet, 2017

Another popular and powerful way to associate a vector with a word is the use of dense word vectors, also called word embeddings. Whereas the vectors obtained through one-hot encoding are binary, sparse (mostly made of zeros), and very high-dimensional (same dimensionality as the number of words in the vocabulary), word embeddings are low-dimensional floating-point vectors (that is, dense vectors, as opposed to sparse vec-tors); see figure 6.2. Unlike the word vectors obtained via one-hot encoding, word embeddings are learned from data. Its common to see word embeddings that are 256-dimensional, 512-dimensional, or 1,024-dimensional when dealing with very large vocabularies. On the other hand, one-hot encoding words generally leads to vectors that are 20,000-dimensional or greater (capturing a vocabulary of 20,000 tokens, in this case). So, word embeddings pack more information into far fewer dimensions. Chollet, 2017

the geometric relationships between word vectors should reflect the semantic relationships between these words. Word embeddings are meant to map human language into a geometric space. For instance, in a reasonable embedding space, you would expect synonyms to be embedded into similar word vec-tors; and in general, you would expect the geometric distance (such as L2 distance) between any two word vectors to relate to the semantic distance between the associ-ated words (words meaning different things are embedded at points far away from each other, whereas related words are closer). In addition to distance, you may want specific directions in the embedding space to be meaningful. To make this clearer, lets look at a concrete example. Chollet, 2017

Its thus reasonable to learn a new embedding space with every new task. Fortu-nately, backpropagation makes this easy, and Keras makes it even easier. Its about learning the weights of a layer: the Embedding layer. Chollet, 2017

Chapter 4

Data Curation

In order to successfully apply the statistical learning approach, a well-structured training data-set is needed. In this section, the process of data curation is elaborated so as both argumentative and non-argumentative sentences to be found.

4.1 Data Used

Most of the argumentative sentences included in the our corpora were found on two IBM data-sets created for this purpose (Table 4.1), including three main topics; Video Games, Democracy and Multiculturalism. It has to be mentioned that some of the data were duplicated, and thus Python code of Appendix 2 was created so as to be removed.

Source Data	file	Total Data	Duplicated	Arguments	Non-Arguments
Aharoni et al., 2014	CDEdata.xls	1292	967	325	-
Bar-Haim et al., 2017	claim_stance_dataset_v1	2394	56	2366	-

TABLE 4.1: Argumentative Data Used

However, the exploitation of already existing annotated data-sets referred to argument detection has the obstacle of lacking non-argumentative instances. The previously mentioned IBM corpora contain only phrases that have been manually annotated as positive instances of arguments, which makes it impossible to train a supervised algorithm classifier in identifying non-arguments without any negative examples.

So, our purpose was to gather an equal number of argumentative and non-argumentative sentences that have the same context with the IBM curated data. Therefore, plain text referring to Video Games was found in additional IBM data-sets. These raw data files were split into sentences, and each of this sentence was labeled as argument or non-argument by authors of this research paper, and the Table 4.2 depicts the results. The data referred as "Not used" were blank or incomplete lines.

The non-arguments collected were not enough, thus it was decided to scrap data from Wikipedia articles. The topics of these articles were similar to the previously gathered data, and are more specifically about Video games, Democracy and Multiculturalism. The code used for the scraping process is aligned at Appendix 1.

Source Data	file	Total Data	Not Used	Arguments	Non-Arguments
Mirkin et al., 2017	asr/DJ_1_ban-video-games_pro.wav.asr.txt	9	3	5	1
Mirkin et al., 2017	asr/EH_1_ban-video-games_pro.wav.asr.txt	20	3	12	5
Mirkin et al., 2017	asr/HE_1_ban-video-games_pro.wav.asr.txt	21	1	12	8
Mirkin et al., 2017	asr/SN_1_video-games_pro.wav.asr.txt	28	10	15	3
Mirkin et al., 2017	asr/TL_1_ban-video-games_pro.wav.asr.txt	19	6	8	5
Mirkin et al., 2017	asr/YB_1_ban-video-games_pro.wav.asr.txt	19	4	7	8
Aharoni et al., 2014	wiki12_articles/Gender_representation_in_video_games	39	4	5	30

TABLE 4.2: Argumentative and Non-Argumentative Data Used

Assuming that Wikipedia articles' authors are objective and do not express their point of view, most of the data scraped were included as non-arguments in the data-set. It has to be mentioned that the data collected from this procedure were previously checked from the python code-described in the Chapter 3 that is aligned with Appendix 6. The sentences classified as non-arguments were added to the created data-set, while the others were considered as controversial (Table 4.3).

Topic of Wikipedia	Total Data	Not Used	Controversial sentences	Non-Arguments
Early Hstory of video games	143	19	59	65
Fourth generation of video game consoles	65	6	32	27
Game Boy	84	22	23	39
Game design	223	32	71	120
Game	191	28	89	74
Gaming Computer '	129	8	62	59
Gaming disorder	12	6	-	6
History of video games	604	70	224	310
Home computer	359	236	54	69
Nintendo	393	70	133	190
PC game	248	60	87	101
Video game	434	82	154	198
Video game addiction in China	58	4	9	45
Video game addiction	257	138	70	49
Video game console	337	261	41	35
Video game culture	292	74	104	114
Video game development	446	229	102	115
Video game industry	275	49	91	135
Video game music	460	176	146	138
Video game programmer	164	19	72	82
Video game-related health problems	52	7	22	23
Video gaming in Japan	300	102	82	116
Video gaming in the United States	119	31	27	61
The Game Awards	36	2	18	16
Multicultural transruption	45	3	20	22
Multicultural and diversity management	41	7	17	17
Multicultural education	248	28	104	116
Multiculturalism in Australia	156	37	55	54
Criticism of multiculturalis	237	56	96	85
Cultural pluralism	28	7	12	9
Multiculturalism in Canada	205	28	84	93
Multiculturalism	449	62	160	227
Democracy Index	59	18	17	24
Direct democracy	162	22	59	81
Types of democracy	25	7	9	9
Representative democracy	56	7	26	23
Criticism of democracy	191	102	55	34
Athenian democracy	335	41	147	147
History of democracy	394	81	126	187
Democracy	452	71	193	188

TABLE 4.3: Non-Argumentative Data Used & Controversial Sentences

The previously described data were concatenated (Appendix 3), and the corpora that will be used in machine learning algorithms is finally composed of both arguments and non-arguments (Table 4.3).

Argumentative and No-Argumentative Data Used				
File	Total Data	Arguments	Non-Arguments	False-Positive Arguments
dataset.csv	6318	2755	3563	-
found_fp.csv	2952	-	-	2952

TABLE 4.4: Argumentative and No-Argumentative Data Used

Additionally, a file that includes all the controversial sentences was created (Appendix 4). This file indicates all the keywords responsible for these controversial results. The indicators- described in previous Chapter and found in Wikipedia articles- were counted using code of Appendix 5, and the following table was the outcome of that enumeration.

either	53	for example	93
(E|e)(ither).+?(or)	44	**as a result**	**991**
also	**456**	**as well**	**1065**
because	121	notably	13
for instance	17	but	326
that is	**582**	while	197
actually	26	against	51
still	85	**so that**	**558**
though	87	besides	4
furthermore	66	eventually	41
clearly	5	if	107
since	34	**on the grounds that**	**527**
although	170	third	6
consequently	4	**as a consequence**	**946**
rather than	66	instead	42
nevertheless	9	except	5
otherwise	12	in fact	10
too	3	not only	22
on one side	1	simply	26
moreover	9	hence	4
therefore	25	every time	1
in other words	4	despite the fact that	488
in order to	43	**just as**	**877**
at the same time	7	of course	2
finally	13	**as an example**	**846**
first	29	even though	95
lastly	**823**	equally	7
whereas	**828**	(T|t)(he more).+?(the more)	2
regardless	7	**for this reason**	**803**
simply because	109	**by contrast**	**781**
naturally	4	at any rate	1
in short	2	in this case	727
such that	398	essentially	5
(N|n)(either).+?(nor)	41	whenever	4
second	5	**as long as**	**609**
even then	3	**as a matter of fact**	**578**
accordingly	3	provided that	283
conversely	3	alternatively	3
afterwards	6	thereafter	1
meanwhile	127	once again	3
once more	1	above all	1
by comparison	7	surely	2
undoubtedly	2	on the one side	1
at first	8	presumably	2
after all	1	what is more	1
certainly	1	anyway	1

TABLE 4.5: Indicators found in Wikipedia articles

The goal of this process was to test in Wikipedia articles the argumentative indicators included in the Chapter 3's dictionary, and upon cross-examination to remove indicators that do not usually reveal argumentative sentences. Based on the Table 4.5, the indicators marked as bold are the ones found in the majority of sentences in Wikipedia. A representative number of them was examined by the authors of this research papers, and the keywords that did not pointed out argumentative statements and removed from the dictionary are represented in the following table. (+ table)

Chapter 5

Results

Results for both structured and statistical implementations presented in Chapter 3, are applied to a set of corpora in order to be evaluated.

5.1 Structural Approach

The structural approach described in previous chapters is being assessed into this section. For this purpose, the code of Appendix 8 alongside with a group of annotated data were executed.

A set of metrics, recall, precision and F1 score, were used in order to evaluate the indicators selected for recognizing argumentative sentences. These metrics are using four counters, true positives (tp) is counting the times both algorithm and analyst labeled a sentence as argumentative, true negatives (tn) how often both algorithm and analyst labeled as non-argumentative, false positive (fp) the times the algorithm assigned as argumentative a sentence that expert recognized as non-argumentative, while false negative (fn) how many times human identified a sentence as argumentative while algorithm did not.

- **Precision** indicates a metric of correctly identified instances:

$$P = \frac{tp + fp}{tp}$$

- **Recall** measures the times algorithm missed out an instance:

$$R = \frac{tp + fn}{tp}$$

- **F1 Score** presents the mean of precision and recall:

$$P = \frac{2 * (R * P)}{R + P}$$

The results based on these three metrics are presented in the table bellow, and they were applied to ten corpora in total. It has to be mentioned that the data-sets were manually annotated by this paper's authors, and included a small amount of both argumentative and non-argumentative sentences.

Source Data	file	Precision	Recall	F1 Score
Mirkin et al., 2017	asr/DJ_1_ban-video-games_pro.wav.asr.txt	1.0	1.0	1.0
Mirkin et al., 2017	asr/EH_1_ban-video-games_pro.wav.asr.txt	1.0	1.0	1.0
Mirkin et al., 2017	asr/HE_1_ban-video-games_pro.wav.asr.txt	0.8	0.727	0.762
Mirkin et al., 2017	asr/SN_1_video-games_pro.wav.asr.txt	1.0	0.928	0.963
Mirkin et al., 2017	asr/TL_1_ban-video-games_pro.wav.asr.txt	0.727	0.888	0.799
Mirkin et al., 2017	asr/YB_1_ban-video-games_pro.wav.asr.txt	0.555	0.833	0.667
Aharoni et al., 2014	wiki12_articles/Gender_representation_in_video_games	0.167	0.4	0.235

5.2 Machine Learning Approach

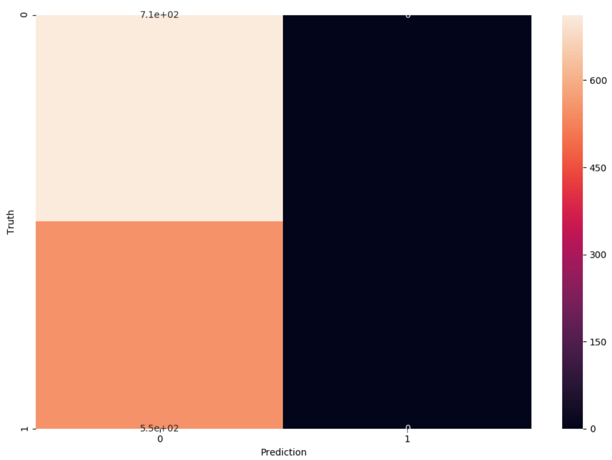

FIGURE 5.1: Results of Random Forest Algorithm in the created dataset

Chapter 6

Conclusion

6.1 Future Work

Appendix

Appendix 1

Scrap Data from Wikipedia

```
import urllib.request
import re
from inscriptis import get_text


def wiki(theme):
  url = "https://en.wikipedia.org/wiki/" + theme
  html = urllib.request.urlopen(url).read().decode('utf-8')

  text = get_text(html)

  with open('../datasets/wiki_' + theme + '.txt', 'w') as out:
    for row in text.split('\n'):
      if len(row) >= 80 and not row[0].isdigit() and not row[1].
          isdigit() and not row[2] == '*':
        row = re.sub(r'\[\d+]', '', row)
        row = row.rstrip('\n')
        out.write(row)
        out.write('\n')


if __name__ == '__main__':

  # Video games topic
  wiki('Video_game-related_health_problems')
  wiki('Video_game_addiction_in_China')
  wiki('Video_game_addiction')
  wiki('Gaming_disorder')
  wiki('2017_in_video_gaming')
  wiki('2019_in_video_gaming')
  wiki('History_of_video_games')
  wiki('PC_game')
  wiki('Video_game_culture')
  wiki('Gaming_computer')
  wiki('Video_game_console')
  wiki('Video_game')
  wiki('Video_gaming_in_the_United_States')
  wiki('Video_game_music')
  wiki('Video_game_industry')
```

```
wiki('Video_game_development')
wiki('Game_design')
wiki('Video_game_programmer')
wiki('Early_history_of_video_games')
wiki('Video_gaming_in_Japan')
wiki('Video_game_crash_of_1983')
wiki('Sixth_generation_of_video_game_consoles')
wiki('Video_gaming_in_China')
wiki('1980s_in_video_gaming')
wiki('Home_computer')
wiki('Nintendo')
wiki('Game_Boy')
wiki('Fourth_generation_of_video_game_consoles')
wiki('Game')
wiki('The_Game_Awards')

# Democracy topic
wiki('Democracy')
wiki('History_of_democracy')
wiki('Athenian_democracy')
wiki('Representative␣democracy')
wiki('Direct_democracy')
wiki('Types_of_democracy')
wiki('Democracy_Index')
wiki('Criticism_of_democracy')

# Multiculturalism
wiki('Multiculturalism')
wiki('Criticism_of_multiculturalism')
wiki('Cultural_pluralism')
wiki('Multiculturalism_in_Canada')
wiki('Multicultural_education')
wiki('Multicultural_and_diversity_management')
wiki('Multiculturalism_in_Australia')
wiki('Multicultural_transruption')
```

Appendix 2

Remove Duplicate Sentences

```
import os

FILE_PATH = os.path.abspath(os.path.dirname(__file__))


def remove_duplicate_rows(file_name):
  sentences = []

  with open(os.path.join(FILE_PATH, '../Results/' + file_name),
     mode='r') as txt_file:
    reader = txt_file.read()

    for sentence in reader.split("\n"):
      sentences.append(sentence)

  with open ( os.path.join ( FILE_PATH, '../Results/new_' +
     file_name ), mode='w' ) as txt_file_new:
    for i in range(len(sentences)):
      if i == 0:
        txt_file_new.write(sentences[i])
        txt_file_new.write("\n")
      elif i < len(sentences) and sentences[i] != sentences[i-1]:
        txt_file_new.write(sentences[i])
        txt_file_new.write("\n")


if __name__ == '__main__':
  remove_duplicate_rows('CDEdata.csv')
```

Appendix 3

Concat Results of checked Data

```
import os
import glob
import csv

FILE_PATH = os.path.abspath(os.path.dirname(__file__))


def combine_results():

  csvfiles = glob.glob(FILE_PATH + '/../Results/checked/*.csv')
  dataset = csv.writer(open(FILE_PATH + '/../Results/dataset.csv', 'w'), delimiter=',')

  for files in csvfiles:
    rd = csv.reader(open(files, 'r'))

    for row in rd:
      if 'wiki' in files:
        if "False" in row:
          dataset.writerow(row)
      else:
        if "True" or "False" in row:
          dataset.writerow(row)


def find_false_positives():

  csvfiles = glob.glob(FILE_PATH + '/../Results/checked/*.csv')
  found_fps = csv.writer(open(FILE_PATH + '/../Results/found_fp.csv', 'w'), delimiter=',')

  for files in csvfiles:
    rd = csv.reader(open(files, 'r'))

    for row in rd:
      if 'wiki' in files:
        if "True" in row:
          found_fps.writerow(row)
```

```
if __name__ == '__main__':
    combine_results()
    find_false_positives()
```

Appendix 4

False-Positives Creation

4.1 ../Results/found_fp_results.csv

In the **terminal** type:

```
python3 getFalsePositives.py > ../Results/found_fp_results.csv
```

4.2 getFalsePositives.py

```python
import json
from configParser import choose_function
from getArguments import spaCy, pos_tagged, check_regex

"""
Description: by using a dictionary that includes words often used
in arguments, it is identified if given sentences are
arguments or not. Part of speech tagging from spaCy is used
for this purpose as well and it is imported by getArguments.py file.
"""


def check_dictionary(doc, dictionary):
    """None
    This function checks if any of the words in the sentence exists
     in the dictionary given. If it does, then it is checked if
     this word's part of speech match with its value given in
     dictionary. If they match, then the word is added in a
     list named keyword_found.

    :param doc: pos tagged sentence from spacy function
    :param dictionary: dictionary that has as keywords words
      and as value their part of speech
    :return: keywords found in the given sentence
    """

    keyword_found = []

    for key, value in dictionary.items():
        if check_regex(doc, key) is not None:
```

```
      if len(value) == 1:
        for key2 in value:
          if checz_regex(doc, key + ' ' + key2) is not None
           and check_regex(doc, key2) is not None and
           pos_tagged(doc, check_regex(doc, key2).text) is not 'None':
            if value[key2] == pos_tagged(doc, check_regex(doc, key2).text)[1] or \
             value[key2] == pos_tagged(doc, check_regex(doc, key2).text)[2]:
              keyword_found.append(key + " + " + key2)
      elif value == 'none':
        keyword_found.append(key)
      elif len(value) != 1 and check_regex(doc, value)
       is not None:
        keyword_found.append(key)
      elif pos_tagged(doc, key)[1] == value or \
       pos_tagged(doc, key)[2] == value:
        keyword_found.append(key)

  return keyword_found


if __name__ == '__main__':

  with open('../dict/dictionary.json', 'r') as dict:
    dictionary = json.load(dict)

  sentences = choose_function("found_fp.csv")

  if sentences != 'No dataset found':
    for sentence in sentences:

      print('"' + str(sentence[0]).strip('b') + '",' +
        str(check_dictionary(spaCy(sentence[0]), dictionary))

  else:
    print(sentences)
```

Appendix 5

Enumerate false positives

```
import os
import csv
import json

FILE_PATH = os.path.abspath(os.path.dirname(__file__))


def enumerate_false_positives(file_name):
  with open('../dict/dictionary.json', 'r') as dict:
    dictionary = json.load(dict)

  with open(os.path.join(FILE_PATH, '../Results/' + file_name), mode='r') as file:
    reader = csv.reader(file)
    counters = []

    for row in reader:
      # first cell includes sentences, not keywords
      for column in row[1:]:
        print(column)
        column = column.strip("['] ")
        column = column.strip('"')

        if column in dictionary:
          flag = True

          for counter in counters:
            if column in counter[0]:
              counter[1] += 1
              flag = False

          if flag:
            counters.append([column, 1])

  save_array_to_csv(counters)


def save_array_to_csv(arrayOfArrays):
  file = csv.writer(open(FILE_PATH +
```

```
      '/../Results/found_FP_keywords.csv', 'w'), delimiter=',')

    for array in arrayOfArrays:
      file.writerow([array[0]] + [array[1]])


if __name__ == '__main__':
    enumerate_false_positives('found_fp_results.csv')
```

Appendix 6

Extract keywords from pdf

```
"""
Description: extract words which are often used in arguments
(based on a paper),and create a dictionary based on these words
(key of the dict) and their specific, if they have one, part
of speech (value of the dict) in arguments
"""

import json
import textract
import os
import csv

FILE_PATH = os.path.abspath(os.path.dirname(__file__))  # path of
    this file


def extract_data():
  """
  By using textract library, this function extracts the whole pdf
      file

  pdf: paper called 'Using Linguistic Phenomena to Motivate a Set
  of Coherence Relations'
  """

  text = textract.process(os.path.join(FILE_PATH,
  "../Reading/cues-
      UsingLinguisticPhenomenaMotivateCoherenceRelations_Knott93.
      pdf"))
  save(text)


def save(text):
  """
  This function saves extracted text to a csv file
  """
  if not os.path.exists("../dict" ):
    os.makedirs("../dict")

```

```
  with open(os.path.join(FILE_PATH, "../dict/data.csv"),
   mode='wb') as csv_file:
    csv_file.write(text)

  modify_csv_file("../dict/data.csv")  # modify extracted text


def modify_csv_file(data):
  """
  This function modifies csv file in order to keep those words
  we are interested in
  """

  flag = 0

  with open(os.path.join(FILE_PATH, data)) as inp:
    reader = csv.reader(inp)

    with open(os.path.join(FILE_PATH, "../dict/data2.csv"),
     mode='w') as out:
      for row in reader:
        if len(row) > 0 and row[0] == "Phrase":
          flag = 1
          continue
        if len(row) == 0 or row[0].isdigit():
          flag = 0
        if flag == 1 and len(row) > 0:
          out.write(row[0])
          out.write("\n")
  check_words("../dict/data2.csv", "../dict/data.csv")


def check_words(data2, data):
  """
  This function adds or removes words that considered as useful
  or not
  """

  exclude_words = ['after', 'and', 'as␣soon␣as', 'before',
          'at␣first', 'at␣first␣sight', 'earlier',
          'fisrt␣of␣all', 'for', 'inasmuch␣as',
          'later', 'much␣sooner', 'not␣because',
          'now', 'if␣not', 'if␣so',
          'in␣the␣beginning', 'in␣the␣end',
          'in␣the␣meantime', 'in␣turn',
          'much␣later', 'not', 'notwithstanding␣that',
          'suppose', 'the␣more␣often', 'this␣time',
          'presumably␣because', 'when', 'where',
          'previously', 'regardless␣of␣that', 'rather',
          'after␣that', 'as', 'simply␣because', 'then',
          'true', 'until', 'again', 'and/or', 'or',
```

```
            'else', 'even']

    include_words = ['for the reason that', 'besides',
            '(E|e)(ither).+?(or)','(N|n)(either).+?(nor)',
            'in one hand', 'in this case',  'on one side',
            'as a matter of fact', 'in point of fact',
            'presumably', 'provided that',
            'regardless', 'rather than', 'simply',
            'as an example', 'in addition']

    test_words = {'even though': 'none', 'first': 'adv',
          'against': 'none', 'last': 'adv',
          'more': {'[a-z]*ly': 'adv'},
          'most': {'[a-z]*ly': 'adv'}, 'if': 'none',
          '(T|t)(he more).+?(the more)': 'none',
          '(T|t)(he more).+?(the less)': 'none',
          'naturally': 'none', 'once again': 'none',
          'once more': 'none', 'surely': 'none',
          'second': 'adv', 'so': 'mark', 'third': 'adv',
          'too': '(too)($|[\.])', 'should say': 'none',
          'might say': 'none', 'may say': 'none',
          'could say': 'none', 'while': 'mark',
          'as a start': 'none', 'in order to': 'none',
          'in order that': 'none', 'still': 'adv',
          'that is': 'none', 'since': 'mark',
          'yet': '(Y|y)(et)[^\.].', 'that': 'mark'}

    with open(os.path.join(FILE_PATH, data2), 'r') as inp, \
      open(os.path.join(FILE_PATH, data), 'w') as out:

      for row in csv.reader(inp):
        if row[0] in exclude_words:
          continue
        else:
          out.write(row[0])
          out.write("\n")

      for word in include_words:
        out.write(word)
        out.write("\n")

    create_dictionary("../dict/data.csv", test_words)


def create_dictionary(data, test_words):
    """
    This function creates a .json file that includes a dictionary of
    the words from the csv file created before and some additional
    words for testing
    """

```

```
    dictionary = test_words

    with open(os.path.join(FILE_PATH, data), 'r') as inp:

        for row in csv.reader(inp):
            if "\x05" in row[0]:
                row[0] = row[0].replace('\x05', 'fi')  # correct words from pdf extraction

            if row[0] in test_words.keys():
                continue
            else:
                dictionary.update({row[0]: 'none'})

    with open('../dict/dictionary.json', 'w') as dict:
        json.dump(dictionary, dict)


if __name__ == '__main__':
    extract_data()
```

Appendix 7

Extract Keywords from pdf

```
from xlrd import open_workbook
import configparser
import os
import re
import csv

FILE_PATH = os.path.abspath(os.path.dirname(__file__))


def py23_str(value):
    """
    This function tries to convert a string to unicode. Because
    of the fact that this conversion differ from python 3
    to python 2, here are checked both possibilities so as
    the program to run in both python 3 and 2.

    :param value: sentence to be converted from string to unicode
    :return: converted input
    """

    try:  # Python 2
        return unicode(value, errors='ignore', encoding='utf-8')
    except NameError:  # Python 3
        try:
            return str(value, errors='ignore', encoding='utf-8')
        except TypeError:  # Wasn't a bytes object, no need to decode
            return str(value)


def get_sentences_csv(dataset_number):
    """
    This function reads files with .csv extension

    :dataset_number: number that refers to order (starts from 0)
    of a dataset in datasets.ini

    :return: a list of sentences
    """
    sentences = []
```

```

    path, _, column, is_argument = get_parameters_dataset(
        dataset_number)

    with open(os.path.join(FILE_PATH, path), mode='r') as dataset:
        reader = csv.reader(dataset)
        for sentence in reader:
            if is_argument is not None:
                sentences.append([str(sentence[int(column)]), str(sentence
                    [int(is_argument)])])
            else:
                sentences.append([str(sentence[int(column)]), 'True'])

    sentences.pop(0)
    return sentences


def get_sentences_xls(dataset_number):
    """
    This function reads files with .xls extension

    :dataset_number: number that refers to order (starts from 0)
    of a dataset in datasets.ini
    :return: a list of sentences
    """
    sentences = []

    path, sheet, column, is_argument = get_parameters_dataset(
        dataset_number)

    reader = open_workbook(path, on_demand=True)
    sheet = reader.sheet_by_name(sheet)
    if is_argument is not None:
        for cell, cell2 in zip(sheet.col(int(column)), sheet.col(int(
            is_argument))):
            sentences.append([cell.value.encode("utf-8"), cell2.value.
                encode("utf-8")])
    else:
        for cell in sheet.col(int(column)):
            sentences.append([cell.value.encode("utf-8"), 'True'])

    sentences.pop(0)
    return sentences


def get_sentences_txt(dataset_number):
    """
    This function reads files with .txt or none extension

    :dataset_number: number that refers to order (starts from 0)
            of a dataset in datasets.ini
```

```

    :return: a list of sentences
    """
    sentences = []

    path, _, _, _ = get_parameters_dataset(dataset_number)

    with open(os.path.join(FILE_PATH, path), mode='r') as txt_file:
        reader = txt_file.read()

        for sentence in reader.split('.'):
            sentences.append([sentence])

    return sentences


def get_parameters_dataset(dataset):
    """
    This function gets the arguments of a specific dataset from
        datasets.ini

    :dataset: number that refers to order (starts from 0)
            of a dataset or the name of dataset in datasets.ini
    :return: section['path'] + file_name: path of dataset
    sheet: sheet that data are in it if it is an .xls file
    column: column of sentences to be identified as arguments or not
    is_argument: column which reveals if a specific sentence is
            an argument or not
    """
    dataset_number, config = check_validity_of_dataset(dataset)

    section = config.sections()[dataset_number]  # each section is a
        name of a file with data
    section = config[section]
    file_name = re.match(r".*:␣(.*)>", str(section), re.MULTILINE)
    file_name = file_name.group(1)

    try:
        sheet = section['sheet']
    except KeyError:
        sheet = None

    try:
        is_argument = section['is_argument']
    except KeyError:
        is_argument = None

    try:
        column = section['column']
    except KeyError:
        column = None
```

```
    return section['path'] + file_name, sheet, column, is_argument


def check_validity_of_dataset(dataset):
    """
    This function checks of a dataset exists in dataset.ini or not

    :dataset: number that refers to order (starts from 0) of
    a dataset or the name of dataset in datasets.ini
    :return: dataset_number: returns the order of given
    dataset in datasets.ini config: returns object config
    from datasets.ini
    """
    config = configparser.ConfigParser()
    config.read('../datasets/datasets.ini')

    if dataset in config:
        dataset_number = config.sections().index(dataset)
    elif dataset < len(config.sections()):
        dataset_number = dataset

    return dataset_number, config


def choose_function(dataset):
    """
    This function checks the extension of a datasets and chooses
    an appropriate method to read the file

    :dataset: number that refers to order (starts from 0)
    of a dataset or the name of dataset in datasets.ini
    :return: a list of sentences if dataset exits
    otherwise 'No dataset found'
    """

    try:
        dataset_number = int(check_validity_of_dataset(dataset)[0])

        try:
            _, extension = dataset.rsplit('.', 1)
        except ValueError:
            extension = None

        if extension == 'xls':
            return get_sentences_xls(dataset_number)
        elif extension == 'csv':
            return get_sentences_csv(dataset_number)
        elif extension == 'txt' or extension is None:
            return get_sentences_txt(dataset_number)

```

```
    except TypeError:
        return 'No dataset found'
```

Appendix 8

Find Argumentative Sentences

```
import json
import spacy
from __future__ import division
from configParser import choose_function, os, py23_str, re

"""
Description: by using a dictionary that includes words often used
in arguments, this file identifies if given sentences are
arguments or not. Part of speech tagging from spaCy is used
for this purpose as well.
"""

FILE_PATH = os.path.abspath(os.path.dirname(__file__))

tp = 0
tn = 0
fp = 0
fn = 0


def spaCy(sentence):
    """
    By using spaCy, this function gets a sentence and returns every
    word's part of speech

    :param sentence: input to be tokenized
    :return: tokenized sentence
    """

    nlp = spacy.load('en')
    doc = nlp(py23_str(sentence))

    return doc


def pos_tagged(doc, word):
    """
    This function gets a tagged sentence from spaCy and a specific
    word and return its part of speech and its dependency
```

```

    :param doc: pos tagged sentence from spacy function
    :param word: a word that we are interested to learn its part of
      speech
    :return: word, its part of speech(pos) and its dependence in the
      given sentence or None
    """

    word = word.lower()

    for token in doc:
        if token.text.lower() == word:
            return [token.text, token.pos_.lower(), token.dep_.lower()]

    return 'None'


def check_regex(doc_regex, regex):
    """
    By using spaCy's function called match, this function is
    checking if a specific regular expression is represented
    by a given sentence

    :param doc_regex: pos tagged sentence from spacy function
    :param regex: a regular expression
    :return: the part of the sentence that is indicated in the given
      regex otherwise None
    """

    regex = re.compile(r''+regex)

    for match in re.finditer(regex, doc_regex.text.lower()):
        start, end = match.span()  # get matched indices
        word_found = doc_regex.char_span(start, end)  # create Span from indices

        return word_found

    return None


def check_dictionary(doc, dictionary):
    """
    This function checks if any of the words in the sentence exists
     in the dictionary given. If it does, then it is checked if
     this word's part of speech match with its value given in
     dictionary. If they match, then the word is added in a
     list named keyword_found.

    :param doc: pos tagged sentence from spacy function
    :param dictionary: dictionary that has as keywords words
```

```
      and as value their part of speech
    :return: True if the list keyword_found is not empty or False
        if it is empty
    """

    keyword_found = []

    for key, value in dictionary.items():
        if check_regex(doc, key) is not None:
            if len(value) == 1:
                for key2 in value:
                    if checz_regex(doc, key + ' ' + key2) is not None
                     and check_regex(doc, key2) is not None and
                     pos_tagged(doc, check_regex(doc, key2).text) is not 'None':
                        if value[key2] == pos_tagged(doc, check_regex(doc, key2).text)[1] or \
                         value[key2] == pos_tagged(doc, check_regex(doc, key2).text)[2]:
                            keyword_found.append(key + " + " + key2)
            elif value == 'none':
                keyword_found.append(key)
            elif len(value) != 1 and check_regex(doc, value)
             is not None:
                keyword_found.append(key)
            elif pos_tagged(doc, key)[1] == value or \
             pos_tagged(doc, key)[2] == value:
                keyword_found.append(key)

    if len(keyword_found) != 0:
        return 'True'
    else:
        return 'False'


def check_validity(real_value, given_value):
    """
    This function checks if two given values match

    :param real_value:  real value is the result of check_dictionary
      function
    :param given_value: given value is the value given by analysts
      into dataset
    :return: Correct results if they match or Wrong results if they
      do not match
    """
    global tn, tp, fn, fp

    if real_value in given_value:
        if real_value == 'True':
            tp = tp + 1
```

```
    else:
      tn = tn + 1
    return "Correct results"
  else:
    if real_value == 'False':
      fp = fp + 1
    else:
      fn = fn + 1
    return "Wrong results"


def precision():
  if (tp + fp) != 0:
    return tp/(tp + fp)
  else:
    return "Integer division by zero"


def recall():
  if (tp + fn) != 0:
    return tp/(tp + fn)
  else:
    return "Integer division by zero"


def f1_score(precision, recall):
  if (precision + recall) != 0:
    return 2 * (precision * recall) / (precision + recall)
  else:
    return "Integer division by zero"


if __name__ == '__main__':

  with open('../dict/dictionary.json', 'r') as dict:
    dictionary = json.load(dict)

  sentences = choose_function("found_fp.csv")
  labeled_data = False

  if sentences != 'No dataset found':
        for sentence in sentences:

      if len(sentence) == 2:
        print('"' + str(sentence[0]).strip('b') + '",' + 'True') # for csv
        # print ( str ( sentence[0] ).strip ( 'b' ) + ',' + 'True' )
        labeled_data = True

      else:
```

```
            print('"' + str(sentence[0]).strip('b') + '",' +
             check_dictionary(spaCy(sentence[0]), dictionary)) # for
                 csv

   else:
      print(sentences)
```

Appendix 9

Machine Learning Approach: Random Forest

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import OneHotEncoder
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
import csv


def encode_dataset(sentences):
  enc = OneHotEncoder(handle_unknown='ignore')
  print(sentences)
  enc.fit([[line.strip()] for line in sentences])
  enc.categories_
  print(enc.categories_)
  result = enc.transform([[line.strip()] for line in sentences]).
     toarray()
  return result


def my_dataset():

  with open('../Results/dataset.csv', 'r') as dataset:
    dataset = csv.reader(dataset,  delimiter=",")
    # for row in dataset:
    #      print(row)
    df = pd.DataFrame(dataset)
    del df[2]
    y = pd.factorize(df[1])[0]
    df[1] = y
    target = df[1]
    df[0] = encode_dataset(df[0])

    x_train, x_test, y_train, y_test = train_test_split(df.drop
       ([1], axis='columns'),
    target, test_size=0.2)
```

```
    print(x_train, y_train)
    print(x_test, y_test)
    model = RandomForestClassifier(n_estimators=20)
    model.fit(x_train, y_train)
    print(model.score(x_test, y_test))

    y_predicted = model.predict(x_test)
    cm = confusion_matrix(y_test, y_predicted)
    print(cm)

    plt.figure(figsize=(10, 7))
    sn.heatmap(cm, annot=True)
    plt.xlabel('Prediction')
    plt.ylabel('Truth')
    plt.show()


if __name__ == '__main__':
  my_dataset()
  # False 0 -True 1
```

References

Aharoni, Ehud et al. (2014). "A Benchmark Dataset for Automatic Detection of Claims and Evidence". In: *Proceedings of the 25th International Conference on Computational Linguistics (COLING)*, pp. 1489–1500. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.672.6321.

AI, Explosion. *spaCy*. URL: https://spacy.io/.

Bar-Haim, Roy et al. (2017). "Stance Classification of Context-Dependent Claims". In: *"Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers"*. Valencia, Spain: Association for Computational Linguistics, pp. 251–261. URL: https://www.aclweb.org/anthology/E17-1024.

Biran, Or and Owen Rambow (2011). "Identifying justifications in written dialogs". In: *Proceedings - 5th IEEE International Conference on Semantic Computing, ICSC 2011* October 2011, pp. 162–168. DOI: 10.1109/ICSC.2011.41.

Budzynska, Katarzyna and Serena Villata (2015). "Argument Mining". In: *IEEE Intelligent Informatics Bulletin*.

Budzynska, Katarzyna et al. (2014). "A model for processing illocutionary structures and argumentation in debates". In: *Proceedings of the 9th International Conference on Language Resources and Evaluation, LREC 2014*, pp. 917–924.

Cabrio, Elena and Serena Villata (2012). "Natural language arguments: A combined approach". In: *Frontiers in Artificial Intelligence and Applications* 242, pp. 205–210. ISSN: 09226389. DOI: 10.3233/978-1-61499-098-7-205.

Chollet, Francois (2017). *Deep Learning with Python*. Manning Publications.

Eckle-Kohler, Judith, Roland Kluge, and Iryna Gurevych (2015). "On the role of discourse markers for discriminating claims and premises in argumentative discourse". In: *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing* September, pp. 2236–2242.

Feng, Vanessa Wei and Graeme Hirst (2011). "Classifying arguments by scheme". In: *ACL-HLT 2011 - Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* 1, pp. 987–996.

Garcia Villalba, Maria Paz and Patrick Saint-Dizier (2012). "Some facets of argument mining for opinion analysis". In: *Frontiers in Artificial Intelligence and Applications* 245.1, pp. 23–34. ISSN: 09226389. DOI: 10.3233/978-1-61499-111-3-23.

Habernalt, Ivan and Iryna Gurevych (2016). "Which argument is more convincing? Analyzing and predicting convincingness of Web arguments using bidirectional LSTM". In: *54th Annual Meeting of the Association for Computational Linguistics, ACL 2016 - Long Papers* 3, pp. 1589–1599.

John Lawrence, Chris Reed, Colin Allen, Simon McAlister, Andrew Ravenscroft, David Bourget (2014). "Mining Arguments From 19th Century Philosophical Texts Using Topic Based Modelling". In: *Proceedings of the First Workshop on Argumentation Mining*, pp. 79–87.

Knott, Alistair and Robert Dale (1994). "Using linguistic phenomena to motivate a set of coherence relations". In: *Discourse processes* 18.1, pp. 35–62.

Lawrence, John and Chsris Reed (2015). "Combining Argument Mining Techniques". In: Association for Computational Linguistics (ACL), pp. 127–136. DOI: 10.3115/v1/w15-0516.

Levy, Ran et al. (2014). "Context dependent claim detection". In: *COLING 2014 - 25th International Conference on Computational Linguistics, Proceedings of COLING 2014: Technical Papers*, pp. 1489–1500.

Lippi, Marco and Paolo Torroni (2015). "Context-independent claim detection for argument mining". In: *IJCAI International Joint Conference on Artificial Intelligence*. Vol. 2015-January. International Joint Conferences on Artificial Intelligence, pp. 185–191. ISBN: 9781577357384.

— (2016). "Argumentation mining: State of the art and emerging trends". In: *ACM Transactions on Internet Technology* 16.2. ISSN: 15576051. DOI: 10.1145/2850417.

Mirkin, Shachar et al. (2017). "A Recorded Debating Dataset". In: *arXiv preprint arXiv:1709.06438*.

Mochales, Raquel and Marie Francine Moens (2011). "Argumentation mining. MARGOT: a web server for argumentation mining". In: *Artificial Intelligence and Law* 19.1, pp. 1–22. ISSN: 09248463. DOI: 10.1007/s10506-010-9104-x. arXiv: arXiv:1502.07526v1. URL: http://argumentationmining.disi.unibo.it/resources.html.

Moens, Marie Francine et al. (2007). "Automatic detection of arguments in legal texts". In: *Proceedings of the International Conference on Artificial Intelligence and Law*, pp. 225–230. ISBN: 1595936807. DOI: 10.1145/1276318.1276362.

Palau, Raquel Mochales and Marie Francine Moens (2009). "Argumentation mining: The Detection, Classification and Structuring of Arguments in Text". In: *Belgian/Netherlands Artificial Intelligence Conference*, pp. 351–352. ISSN: 15687805.

Park, Joonsuk and Claire Cardie (2015). "Identifying Appropriate Support for Propositions in Online User Comments". In: pp. 29–38. DOI: 10.3115/v1/w14-2105.

Rinott, Ruty et al. (2015). "Show me your evidence - An automatic method for context dependent evidence detection". In: *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing* September, pp. 440–450.

Stab, Christian and Iryna Gurevych (2014). "Identifying argumentative discourse structures in persuasive essays". In: *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 46–56.

Van Eemeren, Frans H, Peter Houtlosser, and AF Snoeck Henkemans (2007). *Argumentative indicators in discourse: A pragma-dialectical study*. Vol. 12. Springer Science & Business Media.

Walton, Douglas (2009). "Argumentation Theory: A Very Short Introduction". In: *Argumentation in Artificial Intelligence*.

— (2011). "Argument mining by applying argumentation schemes". In: *Studies in Logic* 4.April 2012, pp. 38–64. URL: http://www.studiesinlogic.net/english/UploadFiles{_}1698/201104/20110415074459727.pdf.

Webber, B., M. Egg, and V. Kordoni (2012). "Discourse structure and language technology". In: *Natural Language Engineering* 18.4, pp. 437–490. ISSN: 13513249. DOI: 10.1017/S1351324911000337.