

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 30, 2019

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Definition | 1 |
| 1.2 | Research Goal | 1 |
| 1.3 | Assignment's Structure | 2 |
| 2 | State-of-the-Art | 3 |
| 3 | Methods | 6 |
| 3.1 | Structural Approach | 6 |
| 3.2 | Statistical Approach | 7 |
| 3.2.1 | Random Forest Classification Algorithm | 8 |
| 3.2.2 | LSTM-RNN Algorithm | 8 |
| 4 | Data Curation | 10 |
| 5 | Results | 14 |
| 5.1 | Structural Approach | 14 |
| 5.2 | Statistical Approach | 15 |
| 5.2.1 | Random Forest Algorithm | 15 |
| 5.2.2 | LSTM-RNN Algorithm | 15 |
| 6 | Conclusion | 18 |
| | Appendices | 20 |
| 1 | Scrap Data from Wikipedia | 23 |
| 2 | Remove Duplicate Sentences | 25 |
| 3 | Concat Results of checked Data | 26 |
| 4 | Gather Ambiguous Sentences found in Wiki articles | 27 |
| 5 | Enumerate ambiguous sentences | 29 |
| 6 | Extract keywords from pdf | 31 |
| 7 | Load Data-sets based on their configurations | 34 |
| 8 | Find Argumentative Sentences | 37 |
| 9 | Statistical Approach: Random Forest algorithm | 40 |
| 10 | Statistical Approach: LSTM-RNN algorithm | 43 |

Acknowledgments

I would like to thank Vasiliki Efstathiou for being the second annotator and a great supervisor, as well as my Professor Panos Louridas.

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-term 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 on hand coded rules and the statistical approach, which based on supervised and deep learning algorithms.

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 statistical 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.

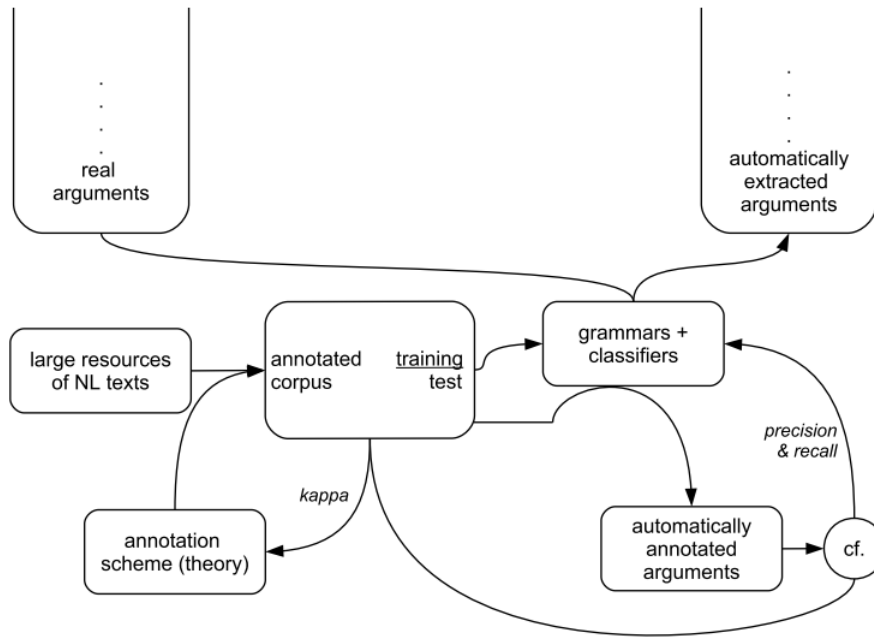


FIGURE 2.1: Natural language processing techniques
 Source: Budzynska and Villata, 2015

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 (Habernal 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 Torrioni, 2016)

| System | SC | | | | | | | BD | | SP | | | |
|---------------------------------|-----|----|----|----|----|----|-----|-----|----|-----|---|-----|----|
| | SVM | LR | NB | ME | DT | RF | RNN | CRF | ML | TES | P | SVM | NB |
| Eckle-Kohler et al. [2015] | X | | X | | | X | | | | | | | |
| Lippi and Torrioni [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 Torrioni, 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 Torrioni, 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.

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

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 measuring accuracy that is a metric for reviewing algorithm's results, which will be used and described at Chapter 5.

3.2 Statistical Approach

The statistical approach is relying on algorithms developed for automating the argument annotation process. These algorithms receive as input, a representation of data that can be understood by computers. For this reason, textual data of our corpora need to be transformed into numeric tensors. (Chollet, 2017)

Text is considered as a form of sequence data, and tokens are the different units into which a text can be split. These units can be either words, characters or n-grams (Chollet, 2017). After text tokenization, the next step is connecting numeric vectors with the occurred tokens. Then, the sequences of vectors instead of words is fed into the selected algorithm.

There are a variety of ways for vector and token association. Two and the most known ones for forms of sequence data are the one-hot encodingText and word-embedding methods (Chollet, 2017), which will be used in the implemented algorithms later on.

As regards one-hot encodingText is a basic way to transform tokens into vectors. Every word is connected with a unique integer index, which is turned into a binary vector of vocabulary's size. Another version of this method is the one-hot hashing trick, which is used when vocabulary's unique tokens are too much that is difficult to be handled. This approach hashes words into a fixed-sized vector through a hashing function, rather than allocating an index to each word, and then creating a dictionary for these indexes. The disadvantages of this method is the hash collisions, and in general the inability of word's correlation that leads to high dimensionality. (Chollet, 2017)

Word embedding, or dense word vectors, is another popular way to associate vectors with words. In contrast with one-hot encodingText, this technique has low-dimensionality

and float vectors. Furthermore, dense word vectors are trained by the data provided, and thus packing more and more data into the same dimensions. This is achieved by minimizing the geometric distance between related words, like synonyms, while maximizing it when words have different semantic (Chollet, 2017).

The implemented algorithms for argument mining problem are the Random Forest classification algorithm, using one-hot encodingText, and the LSTM-RNN algorithm, using word embedding method. The annotation of argumentative and non-argumentative sentences can be considered as a binary classification in Random Forest (Stab and Gurevych, 2014) or a statistical structure mapping of written language in LSTM (Chollet, 2017).

3.2.1 Random Forest Classification Algorithm

The Random Forest algorithm is a collection of decision trees, where each one of them are slightly different from the others either by selecting the data-points or by selecting different features (Müller and Guido, 2016). Every decision tree, built during the training period, predicts to which class a specific input is more suitable (argument or non-argument). After this process is completed, the chosen prediction for each sentence is the one that has the majority of votes across the decision trees. Chollet, 2017

By using the `sklearn.ensemble` library, a Random Forest classification algorithm was implemented (Appendix 9). First of all, an annotated corpora containing both argumentative and non-argumentative sentences is loaded and features regarding how many words, punctuation characters and uppercase characters were created (see Chapter 5). Afterwards, the sentences were tokenized in order to be transformed into a format that computer can understand, as mentioned before. For the tokenization `OneHotEncoder` of `sklearn.preprocessing` library was used. The corpora was separated into a training and a test set of data, and then model was defined and trained using the train data. Finally, the model is tested and a matrix showing the real and the predicted data was presented in a heatmap plot.

The model is applied by using two different methods for training. The one method is using the features created, like in (Lawrence and Reed, 2016), while the other is by using the tokenized sentences.

3.2.2 LSTM-RNN Algorithm

Deep learning in natural-language processing problems is aiming to identify patterns of words or sentences (Chollet, 2017). The most appropriate deep learning model for sequence data processing, like sequences of words in our case, is the Recurrent Neural Network algorithm. RNN is a neural network algorithm that is implementing internal iterations. The RNN resets its state every time a new independent process occurs, which means that each sequence is a different data input to the network. However, network include loops over each sequence's elements. RNN includes the so-called LSTM layer. The Long Short-Term Memory algorithm was developed by Hochreiter and Schmidhuber in 1997, and the purpose of creation was to solve the "vanishing-gradient problem" of SimpleRNN algorithm. LSTM is basically allowing previous information to be re-injected across different timesteps. (Chollet, 2017)

By using the `tensorflow.keras` API, an LSTM algorithm was conducted and implemented (Appendix 10). First of all, the corpora created in Chapter 4 is loaded, and data applied to one list of sentences and one list of labels. Following, the corpora is split into a training and a testing data-set. Furthermore, the lists of trained sentences were tokenized, taking into account only the first ten-thousand most frequent words. The lists of the occurred trained sequences, which are basically lists of integers after the tokenization process, are transformed into a two-dimensions numpy array that has a shape of (number of

sequences, number of timesteps). The number of timesteps is basically the length of the longest sequence. It has to be mentioned that shorter sequences are padded with values at the end, so every sequence has the same shape.

As regards word embedding, Global Vectors for Word Representation (*Glove*) was used, which is a file containing 100-dimensional pre-computed embedding vectors for 400,000 of English words. This file helped in building an index that associates words with number vectors, and through which an embedding matrix of shape (max_words= 10000, embedding_dim = 100) was created and loaded in Embedding layer later on. The Embedding layer is a dictionary-like that maps integer indexes for certain words to dense vectors, and it is the first layer of our model. The second layer applies to LSTM method, while the third layer of Dense was added to end our stack of layers with an equal number of units and classes created. After constructing our model, we are freezing the Embedding layer to avoid deleting what has already been learned.

Finally, the model was trained based on the training data, which are explicitly separated into training and validation samples used for learning word embedding based on our corpora. The last step is testing the model on the test data by firstly tokenizing the sentences, and then evaluating the results occurred.

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.

Most of the argumentative sentences included in our corpora were found on two IBM data-sets created for this purpose (Table 4.1), and are about 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., 2014a | 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 the existing annotated data-sets regarding 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. This means that it is impossible to train a supervised algorithm in classifying 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 curated data of IBM. 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 these sentences 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 more specifically about Video games, Democracy and Multiculturalism. The code used for the scraping process, as well as the scraped pages, are 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., 2014a | 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 (Appendix 6). The sentences classified as non-arguments were added to the created data-set, while the others were considered as ambiguous (Table 4.3).

| Topic of Wikipedia | Total Data | Not Used | Controversial sentences | Non-Arguments |
|--|------------|----------|-------------------------|---------------|
| Early History 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), so as to be used in the statistical approach algorithms. The corpora is composed by 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: Data included in our corpora

As regards the ambiguous sentences of Wikipedia articles that was mentioned before, they were all saved in a file named `found_ambiguous.csv` (Appendix 4). This file indicates all the keywords responsible for these controversial results. The indicators- described in the previous Chapter and found in Wikipedia articles- were counted using code of Appendix 5, and the following table was the outcome of that enumeration.

| Indicator Found | Number of Sentences | Indicator Found | Number of Sentences |
|-------------------------|---------------------|-------------------------------|---------------------|
| as | 968 | that | 542 |
| also | 440 | because | 121 |
| as well | 101 | for example | 92 |
| (E e)(ither).+?(or) | 44 | as a result | 23 |
| because | 121 | notably | 13 |
| for instance | 17 | but | 326 |
| that is | 40 | while | 197 |
| actually | 26 | against | 3 |
| still | 85 | so that | 12 |
| though | 87 | besides | 4 |
| furthermore | 13 | eventually | 41 |
| more + | 40 | either | 53 |
| clearly | 5 | if | 107 |
| since | 34 | on the grounds that | 3 |
| although | 85 | third | 6 |
| consequently | 4 | as a consequence | 2 |
| 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 | 2 |
| in order to | 43 | as | 9 |
| at the same time | 7 | of course | 2 |
| finally | 13 | as an example | 2 |
| first | 29 | even though | 14 |
| lastly | 3 | equally | 7 |
| whereas | 13 | (T t)(he more).+?(the more) | 2 |
| regardless | 7 | for this reason | 2 |
| simply because | 1 | by contrast | 3 |
| naturally | 4 | at any rate | 1 |
| in short | 2 | in this case | 3 |
| such that | 5 | essentially | 5 |
| (N n)(either).+?(nor) | 4 | whenever | 4 |
| second | 5 | as long as | 2 |
| even then | 3 | as a matter of fact | 1 |
| accordingly | 3 | provided that | 1 |
| conversely | 3 | alternatively | 3 |
| afterwards | 6 | thereafter | 1 |
| meanwhile | 10 | once again | 3 |
| once more | 1 | above all | 1 |
| by comparison | 1 | surely | 2 |
| undoubtedly | 2 | on the one side | 1 |
| at first | 1 | presumably | 2 |
| after all | 1 | what is more | 1 |
| certainly | 1 | anyway | 1 |
| so | 16 | most | 22 |
| further | 53 | | |

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 Wikipedia sentences. More than 10% of each indicator's sentences was examined by the authors of this research papers, and the keywords, that did not usually pointed out argumentative statements and removed from the dictionary, are represented in the following. The annotations are described in detail inside the file Results/found_ambiguous_results_annotated.xlsx.

| Indicator | Total Number of Sentences | Number of Sentences Examined | Arguments | Non-Argument | Depending on the context | Removed from the dictionary |
|-----------------------|---------------------------|------------------------------|-----------|--------------|--------------------------|-----------------------------|
| as | 968 | 131 | 22 | 98 | 11 | Yes |
| that | 542 | 91 | 20 | 54 | 17 | Yes |
| also | 440 | 64 | 8 | 46 | 10 | Yes |
| but | 326 | 50 | 7 | 36 | 7 | Yes |
| while | 197 | 35 | 6 | 24 | 5 | Yes |
| because | 121 | 33 | 20 | 10 | 3 | No |
| if | 107 | 20 | 6 | 12 | 2 | No |
| as well | 101 | 20 | 5 | 14 | 1 | No |
| for example | 92 | 8 | 5 | 3 | 0 | No |
| further | 53 | 20 | 4 | 15 | 1 | Yes |
| against | 51 | 21 | 3 | 16 | 2 | Yes |
| more +a dverb in 'ly' | 40 | 16 | 4 | 12 | 0 | No |

TABLE 4.6: Indicators found in Wikipedia articles

It was also observed that the following indicators were used in combination with punctuation, and that's why they were modified into the formats of the table 4.7.

| Indicator | Modified to |
|-------------|--------------|
| while | , while |
| that is | , that is, |
| still | , still |
| as a result | as a result, |

TABLE 4.7: Indicators found in Wikipedia articles

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 two IBM corpora was applied. It has to be mentioned that these two IBM corpora include only argumentative sentences, and we aim to check how many of those arguments will be identified correctly by our algorithm.

The metric of accuracy was used in order to evaluate the indicators selected for recognizing argumentative sentences. For measuring accuracy, four counters were used; **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, **false negative** (fn) how many times human identified a sentence as argumentative while algorithm did not.

- **Accuracy** represents the percentage of correctly classified sentences:

$$A = \frac{tp + tn}{tp + tn + fp + fn}$$

| Source Data | file | Accuracy |
|-----------------------|-------------------------|----------|
| Bar-Haim et al., 2017 | claim_stance_dataset_v1 | 17.50% |
| Aharoni et al., 2014b | CDEdata.xls1.00.1480.25 | 14.81% |

TABLE 5.1: Results of Structural Approach

5.2 Statistical Approach

5.2.1 Random Forest Algorithm

Supervised machine learning algorithms need a number of labeled data in order to be trained. That was the reason corpora of Chapter 4 was created, and used in the implementation of Random Forest classifier algorithm (Appendix 9). A number of 33% of the data used to train the model, while the rest of them to evaluate the results.

It has to be mentioned that two methods were used for argument's classification. The first one was by tokenizing the sentences, so as to be in a format that a computer can understand, and then training the model based on the tokenized sentences. This method had an accuracy of 56.83%, and the results are displayed in the heatmap of figure 5.1 (A). The other technique is by determining some features of each sentence and then feed the algorithm with these features instead of the sentences. In this way the accuracy increased to 79.42%, and it deprecates to the heatmap of figure 5.1 (B). The results of the first approach are not that high, and that is probably because of the unique words.

The features used for classification are the following based on the paper (Lawrence and Reed, 2016):

- **Word Counter:** the number of words in a sentence
- **Uppercase Characters Counter:** the number of uppercase characters found
- **Punctuation or Special Characters Counter:** the number of presence punctuation characters like ""

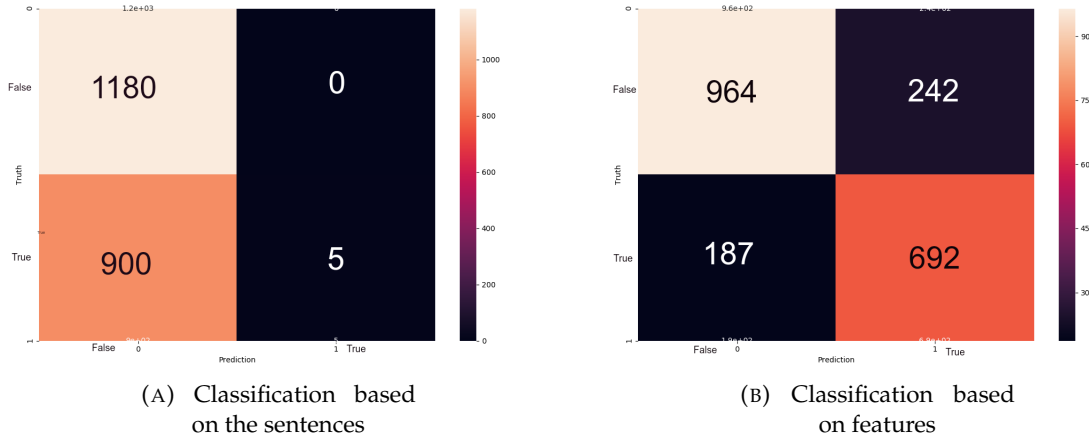


FIGURE 5.1: Predicted and Real results of Random Forest classification algorithm

5.2.2 LSTM-RNN Algorithm

The LSTM-RNN algorithm described in previous chapters is being assessed into this section. For this purpose, the code of Appendix 10 alongside with the data-set described in Chapter 4 was executed.

The data-set used contains an equal number of argumentative and non-argumentative sentences, as well as their labels. The twenty percent of the data were used for training the model, while the rest of them for evaluating it. The model's performance over time is represented in the following plots by using the metrics of accuracy and loss.

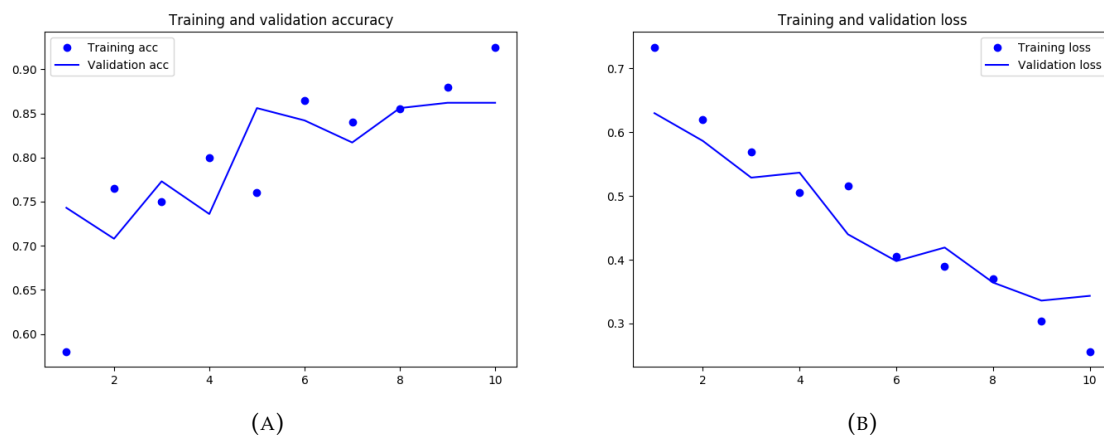


FIGURE 5.2: Training and validation accuracy and loss when using pre-trained word embeddings

After testing the algorithm in test data, the following results occurred with an accuracy of **85.44%**.

```
Found 10859 unique tokens.
Shape of data tensor: (5054, 235)
Shape of label tensor: (5054,)
Found 400000 word vectors.

Model: "sequential"
-----
Layer (type)                Output Shape          Param #
-----
embedding (Embedding)       (None, None, 100)    1000000
-----
lstm (LSTM)                  (None, 100)           80400
-----
dense (Dense)                (None, 1)             101
-----
Total params: 1,080,501
Trainable params: 1,080,501
Non-trainable params: 0

Train on 200 samples, validate on 1000 samples
Epoch 1/10
200/200 [=====] - 2s 10ms/sample - loss: 0.7334 - acc: 0.5800 -
val_loss: 0.6299 - val_acc: 0.7430
Epoch 2/10
200/200 [=====] - 1s 7ms/sample - loss: 0.6195 - acc: 0.7650 -
val_loss: 0.5864 - val_acc: 0.7080
Epoch 3/10
200/200 [=====] - 1s 7ms/sample - loss: 0.5688 - acc: 0.7500 -
val_loss: 0.5288 - val_acc: 0.7730
Epoch 4/10
200/200 [=====] - 1s 7ms/sample - loss: 0.5049 - acc: 0.8000 -
val_loss: 0.5366 - val_acc: 0.7360
Epoch 5/10
200/200 [=====] - 1s 7ms/sample - loss: 0.5152 - acc: 0.7600 -
val_loss: 0.4399 - val_acc: 0.8560
Epoch 6/10
200/200 [=====] - 1s 7ms/sample - loss: 0.4049 - acc: 0.8650 -
val_loss: 0.3978 - val_acc: 0.8420
Epoch 7/10
200/200 [=====] - 1s 7ms/sample - loss: 0.3900 - acc: 0.8400 -
val_loss: 0.4191 - val_acc: 0.8170
Epoch 8/10
200/200 [=====] - 1s 7ms/sample - loss: 0.3700 - acc: 0.8550 -
val_loss: 0.3642 - val_acc: 0.8560
Epoch 9/10
```

```
200/200 [=====] - 1s 7ms/sample - loss: 0.3031 - acc: 0.8800 -  
      val_loss: 0.3359 - val_acc: 0.8620  
Epoch 10/10  
200/200 [=====] - 1s 7ms/sample - loss: 0.2558 - acc: 0.9250 -  
      val_loss: 0.3433 - val_acc: 0.8620  
1264/1264 [=====] - 1s 547us/sample - loss: 0.3467 - acc: 0.8544  
Accuracy: 85.44%
```

Chapter 6

Conclusion

I have implemented three separate argument mining techniques, applicable to both structural and statistical approach. For the **structural approach**, there was created a dictionary of lexical cues that usually characterize argumentative structures based on (Knott and Dale, 1994). The algorithm created was tested in two IBM corpora containing only argumentative sentences, the accuracy seems to be around **15 to 17 %**. As regards the statistical approach, a corpora has to be curated in order to include both argumentative and non-argumentative instances. That is why Wikipedia articles were scrapped assuming that there are no arguments included, and the sentences gathered were cross-checked by the algorithm created for the structural approach. Every sentence that did not have argumentative cues, based on the algorithm, was added to the corpora as non-argumentative, while the others considered as ambiguous and were checked by two annotators so as to remove misleading indicators of dictionary. Afterwards, two algorithms were built by using as training data a part of this corpora. **Random forest classification algorithm** was approached by two different methods of training. The first technique was by using the tokenized sentences which leads to an accuracy of **56.83%**, while the second was by using features like counter of words, uppercase and punctuation characters that have an accuracy of **79.42%**. Finally, the other algorithm of statistical approach that was built is the **LSTM-RNN** that concludes to an accuracy of **85.44%**.

As the results of both structural and statistical algorithms reveal, linguistic rules are not capable of successfully identifying arguments. On the other hand, learning algorithms are a better fit to argument mining, with the most accurate one to be the LSTM-RNN algorithm. One of the main goals as regarding future work is to apply the algorithms built in a fully annotated data-set found in GitHub's issues.

References

- Aharoni, Ehud et al. (2014a). “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>.
- (2014b). “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](https://doi.org/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](https://doi.org/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](https://doi.org/10.3233/978-1-61499-111-3-23).
- Glove.
- 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 Chris Reed (2016). "Argument Mining Using Argumentation Scheme Structures". In: DOI: [10.3233/978-1-61499-686-6-379](https://doi.org/10.3233/978-1-61499-686-6-379).
- Lawrence, John and Chrsis Reed (2015). "Combining Argument Mining Techniques". In: Association for Computational Linguistics (ACL), pp. 127–136. DOI: [10.3115/v1/w15-0516](https://doi.org/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](https://doi.org/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](https://doi.org/10.1007/s10506-010-9104-x). arXiv: [arXiv:1502.07526v1](https://arxiv.org/abs/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](https://doi.org/10.1145/1276318.1276362).
- Müller, Andreas C and Sarah Guido (2016). *Introduction to Machine Learning with Python*. Tech. rep. URL: www.wowebook.orgWOW!eBookwww.wowebook.org.
- 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](https://doi.org/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](https://doi.org/10.1017/S1351324911000337).

Appendices

Appendicies 1

Scrap Data from Wikipedia

```

1  import urllib.request
2  import re
3  from inscriptis import get_text
4
5
6  def wiki(theme):
7      url = "https://en.wikipedia.org/wiki/" + theme
8      html = urllib.request.urlopen(url).read().decode('utf-8')
9
10     text = get_text(html)
11
12     with open('../datasets/wiki_' + theme + '.txt', 'w') as out:
13         for row in text.split('\n'):
14             if len(row) >= 80 and not row[0].isdigit() and not row[1].isdigit() and not row[2] ==
15                 '*':
16                 row = re.sub(r'\s{1,}', ' ', row)
17                 row = row.rstrip('\n')
18                 out.write(row)
19                 out.write('\n')
20
21 if __name__ == '__main__':
22
23     # Video games topic
24     wiki('Video_game-related_health_problems')
25     wiki('Video_game_addiction_in_China')
26     wiki('Video_game_addiction')
27     wiki('Gaming_disorder')
28     wiki('2017_in_video_gaming')
29     wiki('2019_in_video_gaming')
30     wiki('History_of_video_games')
31     wiki('PC_game')
32     wiki('Video_game_culture')
33     wiki('Gaming_computer')
34     wiki('Video_game_console')
35     wiki('Video_game')
36     wiki('Video_gaming_in_the_United_States')
37     wiki('Video_game_music')
38     wiki('Video_game_industry')
39     wiki('Video_game_development')
40     wiki('Game_design')
41     wiki('Video_game_programmer')
42     wiki('Early_history_of_video_games')
43     wiki('Video_gaming_in_Japan')
44     wiki('Video_game_crash_of_1983')
45     wiki('Sixth_generation_of_video_game_consoles')
46     wiki('Video_gaming_in_China')
47     wiki('1980s_in_video_gaming')
48     wiki('Home_computer')
49     wiki('Nintendo')
50     wiki('Game_Boy')
51     wiki('Fourth_generation_of_video_game_consoles')
52     wiki('Game')
53     wiki('The_Game_Awards')
54
55     # Democracy topic
56     wiki('Democracy')

```

```
57 wiki('History_of_democracy')
58 wiki('Athenian_democracy')
59 wiki('Representative_democracy')
60 wiki('Direct_democracy')
61 wiki('Types_of_democracy')
62 wiki('Democracy_Index')
63 wiki('Criticism_of_democracy')
64
65 # Multiculturalism
66 wiki('Multiculturalism')
67 wiki('Criticism_of_multiculturalism')
68 wiki('Cultural_pluralism')
69 wiki('Multiculturalism_in_Canada')
70 wiki('Multicultural_education')
71 wiki('Multicultural_and_diversity_management')
72 wiki('Multiculturalism_in_Australia')
73 wiki('Multicultural_transruption')
74
```

Appendices 2

Remove Duplicate Sentences

```
1  import os
2
3  FILE_PATH = os.path.abspath(os.path.dirname(__file__))
4
5
6  def remove_duplicate_rows(file_name):
7      sentences = []
8
9      with open(os.path.join(FILE_PATH, '../Results/' + file_name), mode='r') as txt_file:
10         reader = txt_file.read()
11
12         for sentence in reader.split("\n"):
13             sentences.append(sentence)
14
15     with open ( os.path.join ( FILE_PATH, '../Results/new_' + file_name ), mode='w' ) as
16         txt_file_new:
17         for i in range(len(sentences)):
18             if i == 0:
19                 txt_file_new.write(sentences[i])
20                 txt_file_new.write("\n")
21             elif i < len(sentences) and sentences[i] != sentences[i-1]:
22                 txt_file_new.write(sentences[i])
23                 txt_file_new.write("\n")
24
25 if __name__ == '__main__':
26     remove_duplicate_rows('CDEdata.csv')
27
```

Appendices 3

Concat Results of checked Data

```
1 import os
2 import glob
3 import csv
4
5 FILE_PATH = os.path.abspath(os.path.dirname(__file__))
6
7
8 def combine_results():
9
10     csvfiles = glob.glob(FILE_PATH + '/../Results/checked/*.csv')
11     dataset = csv.writer(open(FILE_PATH + '/../Results/dataset.csv', 'w'), delimiter=',')
12
13     for files in csvfiles:
14         rd = csv.reader(open(files, 'r'))
15
16         for row in rd:
17             if 'wiki' in files:
18                 if "False" in row:
19                     dataset.writerow(row)
20             else:
21                 if "True" or "False" in row:
22                     dataset.writerow(row)
23
24
25 def find_ambiguous():
26
27     csvfiles = glob.glob(FILE_PATH + '/../Results/checked/*.csv')
28     found_ambiguous = csv.writer(open(FILE_PATH + '/../Results/found_ambiguous.csv', 'w'),
29                                     delimiter=',')
30
31     for files in csvfiles:
32         rd = csv.reader(open(files, 'r'))
33
34         for row in rd:
35             if 'wiki' in files:
36                 if "True" in row:
37                     found_ambiguous.writerow(row)
38
39 if __name__ == '__main__':
40     combine_results()
41     find_ambiguous()
```

Appendices 4

Gather Ambiguous Sentences found in Wiki articles

4.1 ../Results/found_ambiguous_results.csv

In the terminal type:

```
$python3 getAmbiguousSentences.py > ../Results/found_ambiguous_results.csv
```

4.2 getAmbiguousSentences.py

```

1  import json
2  from configparser import choose_function
3  from getArguments import spaCy, pos_tagged, check_regex
4
5  """
6  Description: by using a dictionary that includes words often used
7  in arguments, it is identified if given sentences are
8  arguments or not. Part of speech tagging from spaCy is used
9  for this purpose as well and it is imported by getArguments.py file.
10 """
11
12
13 def check_dictionary(doc, dictionary):
14     """None
15     This function checks if any of the words in the sentence exists
16     in the dictionary given. If it does, then it is checked if
17     this word's part of speech match with its value given in
18     dictionary. If they match, then the word is added in a
19     list named keyword_found.
20
21     :param doc: pos tagged sentence from spacy function
22     :param dictionary: dictionary that has as keywords words
23     and as value their part of speech
24     :return: keywords found in the given sentence
25     """
26
27     keyword_found = []
28
29     for key, value in dictionary.items():
30         if check_regex(doc, key) is not None:
31             if len(value) == 1:
32                 for key2 in value:
33                     if check_regex(doc, key + ' ' + key2) is not None
34                     and check_regex(doc, key2) is not None and
35                     pos_tagged(doc, check_regex(doc, key2).text) is not 'None':
36                         if value[key2] == pos_tagged(doc, check_regex(doc, key2).text)[1] or \
37                         value[key2] == pos_tagged(doc, check_regex(doc, key2).text)[2]:
38                             keyword_found.append(key + " + " + key2)
39             elif value == 'none':
40                 keyword_found.append(key)
41             elif len(value) != 1 and check_regex(doc, value)
42                 is not None:

```

```
43     keyword_found.append(key)
44     elif pos_tagged(doc, key)[1] == value or \
45         pos_tagged(doc, key)[2] == value:
46         keyword_found.append(key)
47
48     return keyword_found
49
50
51 if __name__ == '__main__':
52
53     with open('../dict/dictionary.json', 'r') as dict:
54         dictionary = json.load(dict)
55
56     sentences = choose_function("found_ambiguous.csv")
57
58     if sentences != 'No dataset found':
59         for sentence in sentences:
60
61             print('"' + str(sentence[0]).strip('b') + '", ' +
62                 str(check_dictionary(spaCy(sentence[0]), dictionary))
63
64     else:
65         print(sentences)
```


Appendicies 5

Enumerate ambiguous sentences

```

1  import os
2  import csv
3  import json
4
5  FILE_PATH = os.path.abspath(os.path.dirname(__file__))
6
7
8  def enumerate_ambiguous(file_name):
9      with open('../dict/dictionary.json', 'r') as dict:
10         dictionary = json.load(dict)
11
12         with open(os.path.join(FILE_PATH, '../Results/' + file_name), mode='r', encoding='utf-8') as file:
13             reader = csv.reader(file)
14             counters = {}
15
16             count = 0
17             count_rows = 0
18
19             for row in reader:
20                 firstArgumentWordFlag = False
21
22                 count_rows += 1
23                 # first cell includes sentences, not keywords
24                 for column in row[1:]:
25
26                     if firstArgumentWordFlag == False:
27                         testStr = "["
28                         if column.startswith(testStr):
29                             firstArgumentWordFlag = True
30                         else:
31                             continue
32
33                     column = column.strip("[ ' ")
34                     column = column.strip('"')
35
36                     if column in dictionary:
37                         count += 1
38                         if column in counters:
39                             counters[column] += 1
40                     else:
41                         counters[column] = 1
42                 else:
43                     if ' + ' in column:
44                         count += 1
45                         firstPart = column.split(" + ")[0]
46
47                         if firstPart in counters:
48                             counters[firstPart] += 1
49                         else:
50                             counters[firstPart] = 0
51                 else:
52                     print(row)
53                     print('Not found: ', column)
54
55         save_array_to_csv(counters)
56

```

```
57
58 def save_array_to_csv(my_dict):
59     with open(FILE_PATH + '/../Results/found_Ambiguous_keywords.csv', 'w', newline='') as f:
60         w = csv.writer(f)
61         w.writerows(my_dict.items())
62
63
64 if __name__ == '__main__':
65     enumerate_ambiguous('found_ambiguous_results.csv')
```

Appendices 6

Extract keywords from pdf

```

1  """
2  Description: extract words which are often used in arguments
3  (based on a paper),and create a dictionary based on these words
4  (key of the dict) and their specific, if they have one, part
5  of speech (value of the dict) in arguments
6  """
7
8  import json
9  import textract
10 import os
11 import csv
12
13 FILE_PATH = os.path.abspath(os.path.dirname(__file__)) # path of this file
14
15
16 def extract_data():
17     """
18     By using textract library, this function extracts the whole pdf file
19
20     pdf: paper called 'Using Linguistic Phenomena to Motivate a Set
21     of Coherence Relations'
22     """
23
24     text = textract.process(os.path.join(FILE_PATH,
25     "../Reading/cues-UsingLinguisticPhenomenaMotivateCoherenceRelations_Knott93.pdf"))
26     save(text)
27
28
29 def save(text):
30     """
31     This function saves extracted text to a csv file
32     """
33     if not os.path.exists("../dict"):
34         os.makedirs("../dict")
35
36     with open(os.path.join(FILE_PATH, "../dict/data.csv"),
37         mode='wb') as csv_file:
38         csv_file.write(text)
39
40     modify_csv_file("../dict/data.csv") # modify extracted text
41
42
43 def modify_csv_file(data):
44     """
45     This function modifies csv file in order to keep those words
46     we are interested in
47     """
48
49     flag = 0
50
51     with open(os.path.join(FILE_PATH, data)) as inp:
52         reader = csv.reader(inp)
53
54         with open(os.path.join(FILE_PATH, "../dict/data2.csv"),
55             mode='w') as out:
56             for row in reader:
57                 if len(row) > 0 and row[0] == "Phrase":

```

```

58         flag = 1
59         continue
60         if len(row) == 0 or row[0].isdigit():
61             flag = 0
62         if flag == 1 and len(row) > 0:
63             out.write(row[0])
64             out.write("\n")
65     check_words("../dict/data2.csv", "../dict/data.csv")
66
67
68 def check_words(data2, data):
69     """
70     This function adds or removes words that considered as useful
71     or not
72     """
73
74     exclude_words = ['after', 'and', 'as soon as', 'before',
75                      'at first', 'at first sight', 'earlier',
76                      'firsrt of all', 'for', 'inasmuch as',
77                      'later', 'much sooner', 'not because',
78                      'now', 'if not', 'if so',
79                      'in the beginning', 'in the end',
80                      'in the meantime', 'in turn',
81                      'much later', 'not', 'notwithstanding that',
82                      'suppose', 'the more often', 'this time',
83                      'presumably because', 'when', 'where',
84                      'previously', 'regardless of that', 'rather',
85                      'after that', 'as', 'simply because', 'then',
86                      'true', 'until', 'again', 'and/or', 'or',
87                      'else', 'even']
88
89     include_words = ['for the reason that', 'besides',
90                      '(E|e)(ither).+?(or)', '(N|n)(either).+?(nor)',
91                      'in one hand', 'in this case', 'on one side',
92                      'as a matter of fact', 'in point of fact',
93                      'presumably', 'provided that',
94                      'regardless', 'rather than', 'simply',
95                      'as an example', 'in addition']
96
97     test_words = {'even though': 'none', 'first': 'adv',
98                  'against': 'none', 'last': 'adv',
99                  'more': {'[a-z]*ly': 'adv'},
100                 'most': {'[a-z]*ly': 'adv'}, 'if': 'none',
101                 '(T|t)(he more).+?(the more)': 'none',
102                 '(T|t)(he more).+?(the less)': 'none',
103                 'naturally': 'none', 'once again': 'none',
104                 'once more': 'none', 'surely': 'none',
105                 'second': 'adv', 'so': 'mark', 'third': 'adv',
106                 'too': '(too)($|[\.] )', 'should say': 'none',
107                 'might say': 'none', 'may say': 'none',
108                 'could say': 'none', 'while': 'mark',
109                 'as a start': 'none', 'in order to': 'none',
110                 'in order that': 'none', 'still': 'adv',
111                 'that is': 'none', 'since': 'mark',
112                 'yet': '(Y|y)(et)[^\.] .', 'that': 'mark'}
113
114     with open(os.path.join(FILE_PATH, data2), 'r') as inp, \
115             open(os.path.join(FILE_PATH, data), 'w') as out:
116
117         for row in csv.reader(inp):
118             if row[0] in exclude_words:
119                 continue
120             else:
121                 out.write(row[0])
122                 out.write("\n")
123
124         for word in include_words:
125             out.write(word)
126             out.write("\n")
127
128     create_dictionary("../dict/data.csv", test_words)
129

```

```
130
131 def create_dictionary(data, test_words):
132     """
133     This function creates a .json file that includes a dictionary of
134     the words from the csv file created before and some additional
135     words for testing
136     """
137
138     dictionary = test_words
139
140     with open(os.path.join(FILE_PATH, data), 'r') as inp:
141
142         for row in csv.reader(inp):
143             if "\x05" in row[0]:
144                 row[0] = row[0].replace('\x05', 'fi') # correct words from pdf extraction
145
146             if row[0] in test_words.keys():
147                 continue
148             else:
149                 dictionary.update({row[0]: 'none'})
150
151     with open('../dict/dictionary.json', 'w') as dict:
152         json.dump(dictionary, dict)
153
154
155 if __name__ == '__main__':
156     extract_data()
```

Appendices 7

Load Data-sets based on their configurations

```

1  from xlrd import open_workbook
2  import configparser
3  import os
4  import re
5  import csv
6
7  FILE_PATH = os.path.abspath(os.path.dirname(__file__))
8
9
10 def py23_str(value):
11     """
12     This function tries to convert a string to unicode. Because
13     of the fact that this conversion differ from python 3
14     to python 2, here are checked both possibilities so as
15     the program to run in both python 3 and 2.
16
17     :param value: sentence to be converted from string to unicode
18     :return: converted input
19     """
20
21     try: # Python 2
22         return unicode(value, errors='ignore', encoding='utf-8')
23     except NameError: # Python 3
24         try:
25             return str(value, errors='ignore', encoding='utf-8')
26         except TypeError: # Wasn't a bytes object, no need to decode
27             return str(value)
28
29
30 def get_sentences_csv(dataset_number):
31     """
32     This function reads files with .csv extension
33
34     :dataset_number: number that refers to order (starts from 0)
35     of a dataset in datasets.ini
36
37     :return: a list of sentences
38     """
39     sentences = []
40
41     path, _, column, is_argument = get_parameters_dataset(dataset_number)
42
43     with open(os.path.join(FILE_PATH, path), mode='r') as dataset:
44         reader = csv.reader(dataset)
45         for sentence in reader:
46             if is_argument is not None:
47                 sentences.append([str(sentence[int(column)]), str(sentence[int(is_argument)])])
48             else:
49                 sentences.append([str(sentence[int(column)]), 'True'])
50
51     sentences.pop(0)
52     return sentences
53

```

```

54
55 def get_sentences_xls(dataset_number):
56     """
57     This function reads files with .xls extension
58
59     :dataset_number: number that refers to order (starts from 0)
60     of a dataset in datasets.ini
61     :return: a list of sentences
62     """
63     sentences = []
64
65     path, sheet, column, is_argument = get_parameters_dataset(dataset_number)
66
67     reader = open_workbook(path, on_demand=True)
68     sheet = reader.sheet_by_name(sheet)
69     if is_argument is not None:
70         for cell, cell2 in zip(sheet.col(int(column)), sheet.col(int(is_argument))):
71             sentences.append([cell.value.encode("utf-8"), cell2.value.encode("utf-8")])
72     else:
73         for cell in sheet.col(int(column)):
74             sentences.append([cell.value.encode("utf-8"), 'True'])
75
76     sentences.pop(0)
77     return sentences
78
79
80 def get_sentences_txt(dataset_number):
81     """
82     This function reads files with .txt or none extension
83
84     :dataset_number: number that refers to order (starts from 0)
85     of a dataset in datasets.ini
86
87     :return: a list of sentences
88     """
89     sentences = []
90
91     path, _, _, _ = get_parameters_dataset(dataset_number)
92
93     with open(os.path.join(FILE_PATH, path), mode='r') as txt_file:
94         reader = txt_file.read()
95
96         for sentence in reader.split('.'):
97             sentences.append([sentence])
98
99     return sentences
100
101
102 def get_parameters_dataset(dataset):
103     """
104     This function gets the arguments of a specific dataset from datasets.ini
105
106     :dataset: number that refers to order (starts from 0)
107     of a dataset or the name of dataset in datasets.ini
108     :return: section['path'] + file_name: path of dataset
109     sheet: sheet that data are in it if it is an .xls file
110     column: column of sentences to be identified as arguments or not
111     is_argument: column which reveals if a specific sentence is
112     an argument or not
113     """
114     dataset_number, config = check_validity_of_dataset(dataset)
115
116     section = config.sections()[dataset_number] # each section is a name of a file with data
117     section = config[section]
118     file_name = re.match(r".*: (.*)>", str(section), re.MULTILINE)
119     file_name = file_name.group(1)
120
121     try:
122         sheet = section['sheet']
123     except KeyError:
124         sheet = None
125

```

```

126     try:
127         is_argument = section['is_argument']
128     except KeyError:
129         is_argument = None
130
131     try:
132         column = section['column']
133     except KeyError:
134         column = None
135
136     return section['path'] + file_name, sheet, column, is_argument
137
138
139 def check_validity_of_dataset(dataset):
140     """
141     This function checks of a dataset exists in dataset.ini or not
142
143     :dataset: number that refers to order (starts from 0) of
144     a dataset or the name of dataset in datasets.ini
145     :return: dataset_number: returns the order of given
146     dataset in datasets.ini config: returns object config
147     from datasets.ini
148     """
149     config = configparser.ConfigParser()
150     config.read('../datasets/datasets.ini')
151
152     if dataset in config:
153         dataset_number = config.sections().index(dataset)
154     elif dataset < len(config.sections()):
155         dataset_number = dataset
156
157     return dataset_number, config
158
159
160 def choose_function(dataset):
161     """
162     This function checks the extension of a datasets and chooses
163     an appropriate method to read the file
164
165     :dataset: number that refers to order (starts from 0)
166     of a dataset or the name of dataset in datasets.ini
167     :return: a list of sentences if dataset exists
168     otherwise 'No dataset found'
169     """
170
171     try:
172         dataset_number = int(check_validity_of_dataset(dataset)[0])
173
174         try:
175             _, extension = dataset.rsplit('.', 1)
176         except ValueError:
177             extension = None
178
179         if extension == 'xls':
180             return get_sentences_xls(dataset_number)
181         elif extension == 'csv':
182             return get_sentences_csv(dataset_number)
183         elif extension == 'txt' or extension is None:
184             return get_sentences_txt(dataset_number)
185
186     except TypeError:
187         return 'No dataset found'

```


Appendices 8

Find Argumentative Sentences

```

1  import json
2  import spacy
3  from __future__ import division
4  from configparser import choose_function, os, py23_str, re
5
6  """
7  Description: by using a dictionary that includes words often used
8  in arguments, this file identifies if given sentences are
9  arguments or not. Part of speech tagging from spaCy is used
10 for this purpose as well.
11 """
12
13 FILE_PATH = os.path.abspath(os.path.dirname(__file__))
14
15 tp = 0
16 tn = 0
17 fp = 0
18 fn = 0
19
20
21 def spaCy(sentence):
22     """
23     By using spaCy, this function gets a sentence and returns every
24     word's part of speech
25
26     :param sentence: input to be tokenized
27     :return: tokenized sentence
28     """
29
30     nlp = spacy.load('en')
31     doc = nlp(py23_str(sentence))
32
33     return doc
34
35
36 def pos_tagged(doc, word):
37     """
38     This function gets a tagged sentence from spaCy and a specific
39     word and return its part of speech and its dependency
40
41     :param doc: pos tagged sentence from spacy function
42     :param word: a word that we are interested to learn its part of
43     speech
44     :return: word, its part of speech(pos) and its dependence in the
45     given sentence or None
46     """
47
48     word = word.lower()
49
50     for token in doc:
51         if token.text.lower() == word:
52             return [token.text, token.pos_.lower(), token.dep_.lower()]
53
54     return 'None'
55
56
57 def check_regex(doc_regex, regex):

```

```

58 """
59 By using spaCy's function called match, this function is
60 checking if a specific regular expression is represented
61 by a given sentence
62
63 :param doc_regex: pos tagged sentence from spacy function
64 :param regex: a regular expression
65 :return: the part of the sentence that is indicated in the given
66         regex otherwise None
67 """
68
69 regex = re.compile(r''+regex)
70
71 for match in re.finditer(regex, doc_regex.text.lower()):
72     start, end = match.span() # get matched indices
73     word_found = doc_regex.char_span(start, end) # create Span from indices
74
75     return word_found
76
77 return None
78
79
80 def check_dictionary(doc, dictionary):
81     """
82     This function checks if any of the words in the sentence exists
83     in the dictionary given. If it does, then it is checked if
84     this word's part of speech match with its value given in
85     dictionary. If they match, then the word is added in a
86     list named keyword_found.
87
88     :param doc: pos tagged sentence from spacy function
89     :param dictionary: dictionary that has as keywords words
90                     and as value their part of speech
91     :return: True if the list keyword_found is not empty or False
92             if it is empty
93     """
94
95     keyword_found = []
96
97     for key, value in dictionary.items():
98         if check_regex(doc, key) is not None:
99             if len(value) == 1:
100                 for key2 in value:
101                     if check_regex(doc, key + ' ' + key2) is not None
102                     and check_regex(doc, key2) is not None and
103                     pos_tagged(doc, check_regex(doc, key2).text) is not 'None':
104                         if value[key2] == pos_tagged(doc, check_regex(doc, key2).text)[1] or \
105                         value[key2] == pos_tagged(doc, check_regex(doc, key2).text)[2]:
106                             keyword_found.append(key + " " + key2)
107             elif value == 'none':
108                 keyword_found.append(key)
109             elif len(value) != 1 and check_regex(doc, value)
110             is not None:
111                 keyword_found.append(key)
112             elif pos_tagged(doc, key)[1] == value or \
113             pos_tagged(doc, key)[2] == value:
114                 keyword_found.append(key)
115
116     if len(keyword_found) != 0:
117         return 'True'
118     else:
119         return 'False'
120
121
122 def check_validity(real_value, given_value):
123     """
124     This function checks if two given values match
125
126     :param real_value: real value is the result of check_dictionary
127                     function
128     :param given_value: given value is the value given by analysts
129                     into dataset

```

```

130 :return: Correct results if they match or Wrong results if they
131 do not match
132 """
133 global tn, tp, fn, fp
134
135 if real_value in given_value:
136     if real_value == 'True':
137         tp = tp + 1
138     else:
139         tn = tn + 1
140     return "Correct results"
141 else:
142     if real_value == 'False':
143         fp = fp + 1
144     else:
145         fn = fn + 1
146     return "Wrong results"
147
148
149 def precision():
150     if (tp + fp) != 0:
151         return tp/(tp + fp)
152     else:
153         return "Integer division by zero"
154
155
156 def recall():
157     if (tp + fn) != 0:
158         return tp/(tp + fn)
159     else:
160         return "Integer division by zero"
161
162
163 def f1_score(precision, recall):
164     if (precision + recall) != 0:
165         return 2 * (precision * recall) / (precision + recall)
166     else:
167         return "Integer division by zero"
168
169
170 if __name__ == '__main__':
171
172     with open('../dict/dictionary.json', 'r') as dict:
173         dictionary = json.load(dict)
174
175     sentences = choose_function("found_fp.csv")
176     labeled_data = False
177
178     if sentences != 'No dataset found':
179         for sentence in sentences:
180
181             if len(sentence) == 2:
182                 print('"' + str(sentence[0]).strip('b') + '", ' + 'True') # for csv
183                 # print ( str ( sentence[0] ).strip ( 'b' ) + ', ' + 'True' )
184                 labeled_data = True
185
186             else:
187                 print('"' + str(sentence[0]).strip('b') + '", ' +
188                     check_dictionary(spaCy(sentence[0]), dictionary)) # for csv
189
190     else:
191         print(sentences)

```

Appendices 9

Statistical Approach: Random Forest algorithm

```

1  import pandas as pd
2  import seaborn as sn
3  from sklearn.preprocessing import OneHotEncoder
4  from sklearn.model_selection import train_test_split
5  from sklearn.ensemble import RandomForestClassifier
6  from sklearn.metrics import confusion_matrix
7  import matplotlib.pyplot as plt
8  import string
9  import csv
10 import os
11 import numpy as np
12
13
14 """
15 Description: implementing Random Forest classifier model
16 to an annotated corpora that contains
17 both argumentative and none sentences
18 """
19
20 FILE_PATH = os.path.abspath(os.path.dirname(__file__)) # path of this file
21
22
23 def load_dataset(path):
24     """
25     Loading dataset of a given path, and creating features based on the sentences given.
26     The first feature is a counter of words included in each sentence,
27     the second feature is a counter of uppercase characters, while
28     the third feature is a counter of special characters (punction)
29     :param path: full path of dataset
30     :return: a list of sentences and their labels, and a set of features
31     """
32
33     # load & prepare data
34     with open(path, 'r') as file:
35         dataset = csv.reader(file, delimiter=",")
36         df = pd.DataFrame(dataset)
37
38     del df[2] # column full of non labels = df[1]
39     df[1] = [len(sentences.split()) for sentences in df[0]] # Word Count'
40     df[2] = [sum(char.isupper() for char in sentence) \
41             for sentence in df[0]] # 'Uppercase Char Count'
42
43     df[3] = [sum(char in string.punctuation for char in sentence) \
44             for sentence in df[0]] # 'Special Char Count'
45     df[4] = pd.factorize(labels)[0] # switch False to 0 and True to 1
46
47     return df
48
49
50 def tokenize_sentences(df):
51     """
52     Tokenizing raw sentences by using One-hot encoding
53     into a format that a computer can understand

```

```

54
55 :param df: the dataframe that includes 4 columns
56         [sentences, word Counter, uppercase counter,
57         special char counter, label]
58 :return: tokenized sentences, labels
59 """
60 vectorizer = OneHotEncoder(handle_unknown='ignore')
61 vectorizer.fit([[line.strip()] for line in df[0]])
62 sentences = vectorizer.transform([[line.strip()] for line in df[0]]).toarray()
63
64 # sentences = vectorizer.fit_transform(df[0])
65 labels = df[4]
66 return sentences, labels
67
68
69 def define_model(x_train, y_train):
70     """
71     This function defines and trains the Random Forest model
72
73     :param x_train: the training sentences
74     :param y_train: the sentences' labels
75     :return: trained model
76     """
77     model = RandomForestClassifier()
78     model.fit(x_train, y_train)
79     return model
80
81
82 def testing_model(model, x_test, y_test):
83     """
84     This function evaluates the model on the test set
85
86     :param model: trained model
87     :param x_test: the testing set of sentences
88     :param y_test: the sentences' labels
89     :return: a matrix of the predicted and real values
90     """
91     score = model.score(x_test, y_test)
92     print("Accuracy: %.2f%%" % (score * 100))
93
94     y_predicted = model.predict(x_test)
95     cm = confusion_matrix(y_test, y_predicted)
96     print(cm)
97     return cm
98
99
100 def plot_results(cm):
101     """
102     This function is showing a heatmap plot based on
103     the values of the confusion matrix that contains
104     the real and predicted values.
105     :param cm: matrix of both real and predicted values
106     """
107     plt.figure(figsize=(10, 7))
108     sn.heatmap(cm, annot=True)
109     plt.xlabel('Prediction')
110     plt.ylabel('Truth')
111     plt.show()
112
113
114 if __name__ == '__main__':
115     df = load_dataset('../Results/dataset.csv')
116     sentences, labels = tokenize_sentences(df)
117
118     # using features instead of the sentences
119     #
120     features = np.asarray(df[df.columns[1:4]].values)
121     # split dataset into test and train data
122     x_train, x_test, y_train, y_test = \
123         train_test_split(features, labels, test_size=0.33)
124
125     model = define_model(x_train, y_train)

```

```
126 cm = testing_model(model, x_test, y_test)
127 plot_results(cm)
128
129 # using sentences without their features
130 #
131 # split dataset into test and train data
132 x_train, x_test, y_train, y_test = train_test_split(sentences, labels, test_size=0.33)
133
134 model = define_model(x_train, y_train)
135 cm = testing_model(model, x_test, y_test)
136 plot_results(cm)
```

Appendices 10

Statistical Approach: LSTM-RNN algorithm

```

1  import csv
2  import os
3  import pandas as pd
4  import numpy as np
5  import matplotlib.pyplot as plt
6  from sklearn.model_selection import train_test_split
7  from tensorflow.python.keras.preprocessing.text import Tokenizer
8  from tensorflow.python.keras.preprocessing.sequence import \
9      import pad_sequences
10 from tensorflow.python.keras.layers import Dense, LSTM, Embedding
11 from tensorflow.python.keras.models import Sequential
12
13 """
14 Description: implement lstm model to an annotated corpora
15 that contains both argumentative and none sentences
16 """
17
18 # path of this file
19 FILE_PATH = os.path.abspath(os.path.dirname(__file__))
20
21
22 def load_dataset(path):
23     """
24     Loading dataset of a given path
25     :param path: full path of dataset
26     :return: a list of sentences and their labels
27     """
28
29     # load & prepare data
30     with open(path, 'r') as file:
31         dataset = csv.reader(file, delimiter=",")
32         df = pd.DataFrame(dataset)
33
34         del df[2] # column full of none
35
36         df[1] = pd.factorize(df[1])[0] # False switched to 0 and True to 1
37         texts = list(df[0].values)
38         labels = list(df[1])
39
40         return texts, labels
41
42
43 def get_tokenized_text(max_words, texts_train):
44     """
45     Tokenizing raw sentences.
46     :param max_words: considering only the top max_words of dataset
47                       (most frequent ones)
48     :param texts_train: sentences to for tokenization
49     :return: tokenizer, sequences, word_index
50     """
51     tokenizer = Tokenizer(num_words=max_words)
52     tokenizer.fit_on_texts(texts_train)
53     sequences = tokenizer.texts_to_sequences(texts_train)

```

```

54 word_index = tokenizer.word_index
55
56 print('Found %s unique tokens.' % len(word_index))
57 return tokenizer, sequences, word_index
58
59
60 def pad_sentences(sequences, labels_train):
61     """
62     This function transforms a list of num_sentences sequences
63     (lists of integers) into a 2D Numpy array of shape
64     (num_sentences, num_timesteps).
65     The num_timesteps is either the maxlen argument if provided,
66     or the length of the longest sequence otherwise.
67     Sequences that are shorter than num_timesteps are padded
68     with value at the end.
69
70     :param sequences: list of tokenized sentences
71     :param labels_train: list of sequences' labels
72     :return: a 2D Numpy array of sentences and labels
73     """
74     data = pad_sequences(sequences)
75     labels_train = np.asarray(labels_train)
76     print('Shape of data tensor:', data.shape)
77     print('Shape of label tensor:', labels_train.shape)
78
79     # Splits the data into a training set and a validation set,
80     # but first shuffles the data
81     indices = np.arange(data.shape[0])
82     np.random.shuffle(indices)
83     data = data[indices]
84     labels_train = labels_train[indices]
85
86     return data, labels_train
87
88
89 def word_embedding(glove_file, max_words, word_index):
90     """
91     Parsing the GloVe's word-embeddings file in order
92     to build an index that maps words (as strings)
93     to their vector representation (as number vectors),
94     and then build an embedding matrix that will
95     be loaded into an Embedding layer later on.
96
97     :param glove_file: path of GloVe's word-embedding
98     :param max_words: considering only the top max_words
99                       of dataset (most frequent ones)
100     :param word_index: index of words after tokenization
101     :return: embedding_dim, embedding_matrix
102             of shape (max_words, embedding_dim)
103     """
104
105     embeddings_index = {}
106     f = open(os.path.join(FILE_PATH, glove_file))
107     for line in f:
108         values = line.split()
109         word = values[0]
110         coefs = np.asarray(values[1:], dtype='float32')
111         embeddings_index[word] = coefs
112     f.close()
113     print('Found %s word vectors.' % len(embeddings_index))
114
115     embedding_dim = 100
116     embedding_matrix = np.zeros((max_words, embedding_dim))
117     for word, i in word_index.items():
118         if i < max_words:
119             embedding_vector = embeddings_index.get(word)
120             if embedding_vector is not None:
121                 # Words not found in the embedding
122                 # index will be all zeros.
123                 embedding_matrix[i] = embedding_vector
124
125     return embedding_dim, embedding_matrix

```



```

126
127
128 def lstm_model(max_words, embedding_dim, embedding_matrix):
129     """
130     This function defines the model and
131     loads pretrained word embedding into the Embedding layer
132
133     :param max_words: considering only the top max_words
134                       of dataset (most frequent ones)
135     :param embedding_dim: number of embedding dimensions used
136     :param embedding_matrix: a matrix of shape shape
137                           (max_words, embedding_dim)
138     :return: the defined model
139     """
140
141     # define model
142     model = Sequential()
143     model.add(Embedding(max_words, embedding_dim))
144     model.add(LSTM(100))
145     model.add(Dense(1, activation='sigmoid'))
146     print(model.summary())
147
148     # Loading pretrained word embeddings into the Embedding layer
149     model.layers[0].set_weights([embedding_matrix])
150     model.layers[0].trainable = False
151
152     return model
153
154
155 def training_model(x_train, y_train, x_val, y_val, model):
156     """
157     This function is responsible for compiling and
158     training the model
159
160     :param x_train: sentences for training model (200 sentences)
161     :param y_train: labels of training sentences
162     :param x_val: sentences for validating trained model
163                  (1000 sentences)
164     :param y_val: labels of validation sentences
165     :param model: defined model
166     :return: historing of training, trained model
167     """
168
169     model.compile(loss='binary_crossentropy', optimizer='rmsprop',
170                  metrics=['accuracy'])
171     history = model.fit(x_train, y_train, epochs=10, batch_size=128,
172                       validation_data=(x_val, y_val))
173     model.save_weights('pre_trained_glove_model.h5')
174
175     return history, model
176
177
178 def plotting_results(history):
179     """
180     This function is showing plots of the training and
181     validation results based on accuracy and loss metrics.
182
183     :param history: history of training procedure
184     """
185     acc = history.history['acc']
186     val_acc = history.history['val_acc']
187     loss = history.history['loss']
188     val_loss = history.history['val_loss']
189     epochs = range(1, len(acc) + 1)
190
191     plt.plot(epochs, acc, 'bo', label='Training acc')
192     plt.plot(epochs, val_acc, 'b', label='Validation acc')
193     plt.title('Training and validation accuracy')
194     plt.legend()
195     plt.figure()
196
197     plt.plot(epochs, loss, 'bo', label='Training loss')

```

```

198 plt.plot(epochs, val_loss, 'b', label='Validation loss')
199 plt.title('Training and validation loss')
200 plt.legend()
201 plt.show()
202
203
204 def testing_model(texts_test, labels_test, tokenizer, model):
205     """
206     This function tokenizes the data of test set,
207     and evaluates the model on the test set
208
209     :param texts_test: sentences for testing model
210     :param labels_test: labels of the testing sentences
211     :param tokenizer: tokenizer
212     :param model: trained model
213     :return: accuracy of the created model
214     """
215
216     # Tokenizing the data of the test set
217     sequences = tokenizer.texts_to_sequences(texts_test)
218     x_test = pad_sequences(sequences)
219     y_test = np.asarray(labels_test)
220
221     # Evaluating the model on the test set
222     model.load_weights('pre_trained_glove_model.h5')
223     scores = model.evaluate(x_test, y_test)
224     print("Accuracy: %.2f%%" % (scores[1] * 100))
225
226
227 def main():
228     texts, labels = load_dataset('../Results/dataset.csv')
229     texts_train, texts_test, labels_train, labels_test = \
230         train_test_split(texts, labels, test_size=0.2)
231
232     max_words = 10000 # Considers only the top 10,000 words
233                     # in the dataset
234     training_samples = 200
235     validation_samples = 1000\usepackage{xcolor}
236     \definecolor{maroon}{cmyk}{0, 0.87, 0.68, 0.32}
237     \definecolor{halfgray}{gray}{0.55}
238     \definecolor{ipython_frame}{RGB}{207, 207, 207}
239     \definecolor{ipython_bg}{RGB}{247, 247, 247}
240     \definecolor{ipython_red}{RGB}{186, 33, 33}
241     \definecolor{ipython_green}{RGB}{0, 128, 0}
242     \definecolor{ipython_cyan}{RGB}{64, 128, 128}
243     \definecolor{ipython_purple}{RGB}{170, 34, 255}
244
245     tokenizer, sequences, word_index = \
246         get_tokenized_text(max_words, texts_train)
247     data, labels_train = pad_sentences(sequences, labels_train)
248
249     x_train = data[:training_samples]
250     y_train = labels_train[:training_samples]
251     x_val = \
252         data[training_samples: training_samples + validation_samples]
253     y_val = \
254         labels_train[training_samples: training_samples + validation_samples]
255
256     embedding_dim, embedding_matrix \
257     = word_embedding('../Reading/glove.6B/glove.6B.100d.txt', \
258                     max_words, word_index)
259
260     model = lstm_model(max_words, embedding_dim, embedding_matrix)
261     history, model = training_model(x_train, y_train, \
262                                   x_val, y_val, model)
263     plotting_results(history)
264     testing_model(texts_test, labels_test, tokenizer, model)
265
266
267 if __name__ == '__main__':
268     main()

```