Text Analysis to Support Structuring and Modeling a Public Policy Problem: Outline of an Algorithm to Extract Inferences from Textual Data

Claudia Ehrentraut, Osama Ibrahim, Hercules Dalianis

Abstract

Policy making situations are real-world problems that exhibit complexity in that they are composed of many interrelated problems and issues. To be effective, policies must holistically address the complexity of the situation rather than propose solutions to single problems. Formulating and understanding the situation and its complex dynamics, therefore, is a key to finding holistic solutions. Analysis of text based information on the policy problem, using Natural Language Processing (NLP) and Text analysis techniques, can support modelling of public policy problem situations in a more objective way based on domain experts’ knowledge and scientific evidence. The objective behind this study is to support modelling of public policy problem situations, using text analysis of verbal descriptions of the problem. We propose a formal methodology for analysis of qualitative data from multiple information sources on a policy problem to construct a causal diagram of the problem. The analysis process aims at identifying key variables, linking them by cause-effect relationships and mapping that structure into a graphical representation that is adequate for designing action alternatives, i.e., policy options. This study describes the outline of an algorithm used to automate the initial step of a larger methodological approach, which is so far done manually. In this initial step, inferences about key variables and their interrelationships are extracted from textual data to support a better problem structuring. A small prototype for this step is also presented.

Keywords—Public policy, problem structuring, qualitative analysis, Natural Language Processing, algorithm; inference extraction.

Introduction

Policy analysis in an evidence-based policy making approach needs to draw upon a wide range of existing data and knowledge (including factual information, scientific knowledge, and expert knowledge), in order to provide
public policy makers and stakeholders with informative hindsight, insight and foresight. In order to make well-informed decisions about public policy programmes and projects, it is important to put the best available evidence from research at the heart of policy development and implementation, this attempt extends the idea of governing based on facts [1].

Since policies are designed to address problems in society, the problem must be kept in mind as the foundation to any policy analysis both if the intent of the analysis is prescriptive or evaluative. If the problem is not accurately understood and stated, it is hard to recommend policy alternatives addressing the underlying problem situation. The manner in which a problem is set or framed constrains its solution and the generation of action alternatives. For that reason, a large part of the decision support activities occurring within a policy cycle is about understanding, formulating and structuring “problems”. The accuracy of the definition of the problem allows identifying appropriate policy alternatives or evaluating the success of an existing policy.

The focus of this study lies on outlining an algorithm used to automate the process of decomposing text into a series of inferences. The step supports the structuring and modelling of public policy problems and is so far performed manually. The paper intends to provide the basic structure for a future implementation by:

- Illustrating the framework and workflow of the algorithm as well as
- Addressing key questions that have to be considered during implementation.

Background

System Dynamics is one of the operations research (OR) areas that have made significant contributions to policy making (e.g. [2], [3], [4]). Although system dynamics models are mathematical representations of problems and policy alternatives, it is recognized that most of the information available to the modeler is not numerical in nature, but qualitative [2]. By examining the system dynamics modelling process, it is clear that the use of qualitative data is not just appropriate but essential to facilitate the conceptualization and the formulation stages of the modelling process. The four stages of modeling are outlined below: [5]

The conceptualization stage (problem definition and system conceptualization): Sterman [6] recognizes the need to access the client’s mental database, and the written database during the problem definition process.

The formulation stage (model formulation and decision dynamics): positing a detailed structure and selecting the parameter values, can also contain elements of qualitative data. Sterman [6] suggests that omitting structures or variables known to be important because numerical data are unavailable is actually less scientific and less accurate than using judgment to estimate
their values. Nonetheless, this is the area in which system dynamics practitioners have questioned the use of qualitative variables.

The testing stage (model testing and evaluation): Model testing should draw upon all sources of available knowledge. The model must not contradict knowledge about the structure of the real system. Structure verification may include review of model assumptions by domain experts or by comparing model assumptions to descriptions of decision-making and organizational relationships found in relevant literature.

The implementation stage (policy analysis and model use): Testing the model’s response to different policies and transferring study insights to the users of the model in an accessible form. The interpretation and use of simulation results by policy makers pose several important challenges associated with understanding the many types of judgments needed during the model-building process, and the judgments needed to assess and use the output of the model [7].

There is a lack of an integrated set of procedures to obtain and analyze qualitative data, which creates a gap between the problem modeled and the model of the problem. The application of these procedures with textual data to support the modeling process in one or more case studies could lead to specific recommendations to enrich system dynamics practice through the development and testing of reliable formal protocols that can be replicated and generalized [8].

There are some systems, however, that have approached this problem. Xiao et al [9], describes the extraction of security policies from documents with almost 90% precision and recall. In Brodie et al. [10], they use a system called SPARCLE to extract Privacy Policy Rules from documents. SPARCLE uses a shallow parser for the extraction and obtains a precision ranging from 82 to 91% and a recall from 87 to 97% depending on domain as for example government, finance, high tech etc.

Approach

In this study, qualitative information refer to text based information on the policy problem, which is simply either a recording of information from the mental database of policy decision-makers, stakeholders and domain experts, or concepts and abstractions that interpret scientific evidence and facts from various information sources.

Key information sources for a public policy problem structuring include: (i) Public policy evaluations and impact assessment reports from governmental institutions’ websites; (ii) Reports from industry, research institutions and NGO’s; and (iii) Published literature (mainly from refereed journals).

Text analysis of verbal descriptions of problems has been the subject of considerable work in the cognitive sciences, with related although somewhat different goals, with a focus on understanding and summarization of text into
“units” and then connecting these units in a summary or a graph, or in a sequence of inferences. Herein, we define a methodology for text synthesis based on a methodology by Câmara et al. [11] for grammatical and semantic analysis of the verbal description of the problem in order to construct a representation of the problem that is adequate for its solution, a causal diagram. The methodology can be summarized with the following steps:

Step 1. Decomposing text into a series of inferences.
- The text is browsed searching simultaneously for trigger words from one of the following categories. Read in sentence-by-sentence and extract sentences which contain a word from at least one, two or all the three categories.
  - Inference indicating words: because, thus, then, however, mean, compare, etc.
  - Modal words: will, would, can, could, must, may, etc.
  - Influence indicating words: result, impact, influence, relate, cause, affect, increase, decrease, reduce, hinder, improve, support, benefit, important, etc.
- Put chosen inference sentences and attached reference into desired format.
- GUI that displays all inferences with references. (User engagement)

Step 2. Inferences are scanned for nouns, adjective-noun combinations to identify variables and entities.

Step 3. The initial set of variables (entities) is processed and refined by the user to identify synonyms, group and rank entities in order to reach the final set of key variables.

Step 4. An adjacency matrix $A = [a_{ij}]$, is constructed with the identified key variables to indicate interdependencies, where $a_{ij} = 0$ if a variable $x_i$ is not related to variable $x_j$; $a_{ij} = 1$ if $x_i$ is related to $x_j$ and changes are in the same direction; $a_{ij} = -1$ if $x_i$ is related to $x_j$ and changes are in opposite directions.

Step 5. Translate the matrix into a causal diagram with directed polarized relationships.

The approach presented in the paper at hand focuses on Step 1. Decomposing text into a series of inferences is up to now performed manually, which is a time consuming task. In order to scale up that process, the idea is to automate this task. The remaining part of the paper will outline the algorithm that is to be implemented in order to automate this step.
The Algorithm Framework

The following sections will outline the algorithm used to automate the process of decomposing text into a series of inferences. The algorithm can be pictured as a pipeline consisting of seven modules. The modular construction aims at making the algorithm easily extendable.

The framework of the algorithm is a Java program that incorporates the seven modules. These modules reflect the different tasks that have to be performed in order to extract the inference sentences and their references from the input data. Depending on the module, either an already existing program is incorporated, or new code is written that implements the processing steps of the task. The modules are described in more detail in Section V.

![Possible GUI for inference sentences generated by the Output Generation Module.](image)

Incorporation of existing tools and programs

The algorithm comprises tasks for which tools are already available, e.g., effective PDF to XML converters exists. The goal, when implementing the algorithm, is to make as extensive use of existing programs and methods as possible.

The premise for already existing programs to be used is that they are open source, i.e., they are covered by licenses that comply with the Open Source Definition, which is approved by the Open Source Initiative\(^3\).

Open source programs applicable to our task were searched for via http://sourceforge.net/ or a standard search engine, while ensuring, for the latter, that the found program complies with the premise stated above.

---

\(^3\)http://opensource.org/about. 2014-09-08.
Principles of operation – the control module

The algorithm is designed in the way that it processes one text at a time, i.e., one text passes all modules before the next text is being processed. The algorithm can take either a single or multiple texts as input. In case multiple texts are given as input, they are to be saved in a folder that is then passed as an argument to the algorithm.

The algorithm will be controllable either via a command-line interface (CLI), in the initial version, or a graphical user interface (GUI). From there, the user will be able to provide (CLI) or choose (GUI):

- The input text or folder,
- The reference insertion mode (see Section Reference Capturing Module),
- The desired text normalization steps (see Section Text Normalization Module),
- The favored output format (see Section Output Generation Module).

From this control module, the user will also be prompted in case errors appear or if further information needs to be provided for the algorithm to continue processing the data.

The Algorithm Modules

Converting Module

The first task of the algorithm is to convert the existing policy documents from their original format into a format that is suitable for processing the texts.

The policy documents are available in PDF-format, while, for instance, XML is a suitable processing format. XML refers to Extensible Markup Language and is a simple and flexible format [12]. The XML format was chosen because it provides the most appropriate format for further processing, with the actual content as well as metadata about the text being extracted from the PDF-file. Although the module will only support PDF-files in the initial version, it will be implemented so that it is easily extendible to process further document formats, such as DOC, or ODT.

For the PDF to XML conversion, the pdf2xml converter[^1] could be incorporated.

Reference Capturing Module

This module reflects the task of capturing the reference for each inference sentence. The module implements different capturing techniques: (1) the references are inserted manually or (2) the references are extracted and captured automatically, which the user will be able to choose in the initially described control module.

When processing the data, the entire text is read into this module. The Metadata Extraction Tool\(^5\) could be used to extract the references automatically, either directly from the PDF or from the XML file. The tool allows the user to specify which metadata to extract. When extracting the reference of a text, tags such as \texttt{<author>}, \texttt{<title>} or \texttt{<year>} would be of interest. The relevant reference tags are, initially, extracted from the XML file, assigned to a reference identifier, and saved program internally.

The reference identifier has the following structure: name of the first author, year of publication, and an optional identifier, in case there are multiple publications of the same author during one year, e.g., Doe\_2008\_1.

In case no reference can be found automatically, the user is prompted to insert it manually via the control module interface. Once the text has been processed in the Text Normalization and Inference Extraction Modules, the reference tag will be added to each inference sentence extracted from this text in the Merging Module.

Text Normalization Module

In this module, multiple standard Natural Language Processing (NLP) tools can be applied to the input text. The module is designed to be extendable in order to incorporate different NLP tasks if necessary. Initially, this module implements the removal of XML-tags, sentence detection and lemmatisation.

The text is passed to this module in XML format. In a first step, all XML-tags are removed from the input text, yielding the plain content of the text that is to be further processed.

By applying a sentence detector, the plain text is segmented into its composing sentences. Sentence segmentation is a central preprocessing step in natural language processing with many applications such as parsing, document summarization or machine translation relying on it [13]. Lemmatisation is a method to reduce inflected words to their lemma or base form [14]. For example, the inflected words \textit{increasing} and \textit{increases} are reduced to \textit{increase}. By lemmatizing the text, all variations of inference indicating words, modal words and influence indicating words, which are described in Section V.D, can be retrieved.

XML removal, sentence detection and lemmatisation are integrated in the Stanford CoreNLP package\(^6\) that could be used for the task. All three pre-processing methods will be executable from the initial version.

Step 2 of the overall methodology aims at scanning inferences for nouns and adjective-noun combinations to identify variables and entities. In NLP terms, such a task is called Named Entity Recognition (NER), i.e., the task of recognizing expressions denoting entities such as diseases, drugs, or people’s names in free text documents [15]. So far, even this task is performed manually but will ideally be automated. In this regard, the module needs to be extendable. A NER module that can be incorporated is available via the Stanford CoreNLP package or Apache OpenNLP\(^7\). The Text Normalization Module will implement NER from the initial version, since it represents a vital step with regard to subsequent steps, yet it will not be executable. In order to recognize entities that are relevant for the present task, the NER system needs to be trained on manually annotated data; annotation referring to the task of adding notes to existing text documents. Thus, relevant nouns, and adjective-noun combinations in a training dataset with inference sentences will need to be annotated before the NER system could be used for the policy documents.

Other NLP subtasks that could be embedded into this module are tokenization, part-of-speech tagging or parsing. For all these tasks, open source programs are available via Apache OpenNLP, or within the Stanford NLP package\(^8\). For stemming, the Porter stemmer\(^9\) is available open source.

The output of this module, in the initial version, will be the text(s) in TXT format, from which XML tokens are removed and which has been lemmatized and sentence-segmented.

**Inference Extraction Module**

The input to this module is the plain, preprocessed text. The module will extract the actual inference sentences from the input text, by applying a keyword matching technique. This means, given sentence \(n\) in text \(A\), the program will scan the sentence for keywords or trigger words, which are available to the program from external files.

The trigger words can be classified into three different categories.

- **Inference indicating words**, e.g., because, thus, then, however, mean and compare.
- **Modal words**, e.g., will, would, can, could, must or may

---


Influence indicating words, e.g., result, impact, influence, relate, cause, affect, increase, decrease, reduce, hinder, lock, improve, support, provide, benefit or important to.

The key words are stored in separate files with one file per category. For a sentence to be classified as an inference sentence, it must contain a word from at least one, two or all three categories. Thus, the trigger words define what is an inference sentence and what not. For instance, the following sentence, containing the modal word can and the influence indicating word provide, would be classified as an inference sentence:

*Biomass is a non-intermittent Renewable Energy Source that can provide energy to be used for heating and cooling, electricity and transport.*

On the other hand, the subsequent sentence would not be classified, as it does not contain the mandatory trigger words:

*Biomass is the Renewable Heat source for small, medium and large scale solutions.*

A lot of research has focused on sentence extraction. Extracting sentences can be a subtask for various purposes but is often considered as part of automatic text summarization\(^\text{10}\). Automatic text summarization refers to “the shortening of texts by computer, while still retaining the most important points of the original text.” [16, 17] In our approach, however, the focus is not on extracting sentences that summarize the text in any way but to extract sentences that meet certain constraints.

For this module, Java code will be written that takes the input text and process it sentence by sentence. The output text is split into inference and non-inference sentences, respectively. All inference sentences are saved into one file, and are retrievable from there for the Merging Module and Output Generation Module. The non-inference sentences are saved into a separate file and archived.

**Merging module**

This module merges the extracted inference sentences with their respective text reference. Keeping the information from which text the inference sen-

\(^{10}\) This assessment is based on a Google Scholar search, September 10th 2014. Entering “sentence extraction” into the search field and choosing results from 2010 and later yielded 1670 results, which were then sorted by relevance. Scanning the first 20 papers, 9 out of 20 discussed sentence extraction as part of automatic text summarization.
tence was extracted is important, in order to enable the user to access the source text if necessary.

For this module, Java code will be written. The input is a file containing the extracted inference sentences, one sentence per line. The output is a file with all the inference sentences and their respective reference tag.

Fig. 2. Possible GUI for inference sentences generated by the Output Generation Module.

Output Generation Module

This module is designed to convert the extracted inference sentences into the desired output format. It will be possible for the user to choose the output format in the initially described control module. Possible output formats will be:

- **File**, i.e., the inference sentences will be output into a desired file format, for example, TXT, XML etc.,
- **GUI**, i.e., the inference sentences will be loaded into the screen of a GUI,
- **External program**, i.e., the inference sentences will be loaded into an existing program for further processing, as for example brat rapid annotation tool\(^\text{11}\).

The GUI as it is depicted is already implemented in Java and can be extended and used in the final algorithm implementation.

Trial Implementation

A trial version of the algorithm was implemented using Python. A randomly chosen policy document was used as input. In a sample output [18], inference sentences that were identified by the system are highlighted in gray. The trigger words are marked in different colors depending on the three categories, yellow for inference indicating words, blue for modal words and red for influence indicating words.

The trial showed that it is feasible to implement this approach in a reasonable amount of time. Further, it helped to specify requirements in the algorithm framework, such as using lemmatisation when preprocessing the data.

Conclusions

We have presented the outline of an algorithm that is, in its initial step, supposed to automate the process of decomposing text into a series of inferences, to support structuring and modelling a public policy problem. We have provided the basic structure for a future implementation by illustrating the framework and workflow of the algorithm as well as addressing key questions that have to be considered during implementation. Being a task that is performed manually so far, automating this step entails time saving for the end user (policy makers, stakeholders etc.), and less dependence on human factors.

The next task will focus on implementing the system in its initial version. The system will be evaluated by comparing the extracted output sentences of a text to the inferences extracted by a human expert from the same text. Once an initial version is up and running, further steps such as Named Entity Recognition can be incorporated.

Acknowledgement

The authors would like to thank Maria Skeppstedt for providing very valuable comments and advice and for implementing a simple version of the algorithm.

References

13. L. Lemnitzer. “Sentence Segmentation.” Internet: