Open-Source Implementation of a Document Structuring System for NLTK

Nicholas FitzGerald
UBC
nfitz@interchange.ubc.ca

Abstract

Natural Language Generation (NLG) is the task of generating coherent text outputs which express information such that it can be easily read and understood by human users. Most NLG systems which have been developed are highly domain-specific, and few adapable generalized algorithms exist, and none which have been implemented for open-source toolkits. For this project, I implement the Bottom-Up Document Structuring algorithm presented in (Reiter and Dale, 2000) for the python-based Natural Language ToolKit (NLTK). I also test how this implementation can be incorporated into a pipeline for abstractive summarization of evaluative opinions.

1 Introduction

The Natural Language ToolKit (NLTK) (Bird et. al, 2009) is a collection of libraries and tools written in Python to facilitate various Natural Language Processing tasks. Currently, NLTK contains modules for many tasks such as Context-Free grammar parsing, Part-of-Speech tagging, Resolution Theory Proving, and many other tasks related to the study of computational semantics and syntax. As of the publication of the most recent version of NLTK, only limited NLG capabilities exist. NLG has been indicated as an area of fruitful future development for NLTK.

2 Background

2.1 Natural Language Generation

Natural Language Generation (NLG) is the task of generating coherent text outputs which express information such that it can be easily read and understood by human users. This process is generally thought to involve several distinct tasks (Reiter and Dale, 2000).

1. **Text Planning** in which the most important and relevant information is selected from the knowledge base (*Content Determination*) and this information is ordered and structured in such a way as to maximise coherence and informativeness (*Document Structuring*).

2. **Microplanning** in which specifics of word selection, referring expressions, and the finalisation of ordering are determined.

3. **Realization** in which internal representations of the above decisions are realised in actual text output.

The focus of this project was Document Structuring, which is the problem of ordering the information presented in a text in such a way as to be maximally coherent and informative to the reader. Texts cannot be simply a random bag of sentences the order in which messages are presented, and the ways these messages are linked together can have a large impact on the meaning conveyed. In order to generate useful and meaningful outputs, NLG systems must incorporate models of language which go beyond the level of sentences.

The dominant paradigm informing much work Document Planning is based on Rhetorical Structure Theory (RST) (Mann and Thompson, 1988) in which messages are joined by Discourse Relations paradigmatic schemas which indicate certain semantic relationships between the two messages. For example, two opposing messages might be linked by a Contrast relationship, or a general message might be followed by a more specific one as an Example.

Two main implementation approaches exist (Reiter and Dale, 2000). The Schema approach is a top-down approach where article structures are specified as a hierarchy of patterns. This is quite similar to the top-down approach to Context-Free Grammar parsing, but in this case

```
Let POOL = *messages produced by content determination mechanism*
while(size(POOL)$\geq$1)do
    find all pairs of elements in POOL which can be
        linked by a discourse relation
    assign each such pair a desirability score, using a
        heuristic preference function
    find the pair $E_i$ and $E_j$ with the highest preference
        score
    combine $E_i$ and $E_j$ into a new DocumentPlan $E_k$,
        using and appropriate discourse relation
    remove $E_i$ and $E_j$ from POOL and replace them
        with $E_k$
end while
```

Figure 1: The pseudo-code Bottom-Up Document Structuring algorithm (from (Reiter and Dale, 2000))

the non-terminals are Schemas and the terminals are messages. This approach is most appropriate when the structure of the desired output document is fairly predictable and stereotyped. In contrast, the bottom-up approach (see fig. 1) (Marcu, 1997) is useful when the output document structure is more varied and difficult to predict in advance.

Another way of conceptualising the different algorithms is as data-driven or hypothesis driven. The bottom-up algorithm is essentially data-driven, since all content selection occurs before document planning, so this task is easily isolated and pipelined. On the other-hand, the Top-Down algorithm is generally hypothesis-driven, so content selection is embedded into the Document Structuring. Since content-selection is highly domain-specific, and not easily generalised, a generalised Top-Down Hypothesis-Driven document structuring system is not easily achieved. Therefore, the Data-Driven Bottom-Up algorithm was chosen for this system.

2.2 Summarization of Evaluative Opinion

One of the many applications of NLG systems is in Abstractive Summarization systems. In Abstractive summarization, information is first extracted from the source texts into an abstract internal format. This is then expressed via a NLG system: Content Selection determines the most relevant and important information to be expressed, Document Structuring orders this information, and a Lexicalizer and Surface Realiser generates the text output.

One domain where this has been used is in summarization of Evaluative Opinions, such as opinions regarding a product expressed in user reviews (Carenini et. al, 2006; FitzGerald, 2009). As with other domains, the NLG systems used for these tasks tend to be highly domain specific.

In order to evaluate the usefulness of the software package, tests were made investigating whether these simple algorithms could be used as a useful part of a pipeline for summarization of evaluative opinions.

3 Design Considerations

3.1 Main Classes

The important main classes for the generalised bottom-up Document Structuring package are as follows:

- Messages are the content bearing units which make up the document. What these are is determined by the domain for which the document is being structured.
- Rules are a representation of the relationships which can hold between Messages
- ConstituentSets are constructs which consist of a set of Messages and/or ConstituentSets related via some discourse relation. They can be thought of as the output of a rule acting on Messages and ConstituentSets.
- DocPlans are the output of the Document Structuring algorithm, and encompass the tree-like structure which results from successive application of Rules

In what follows, each of these objects are discussed in further detail:

3.1.1 Messages

Messages are the content bearing units which make up the document. What specifically will make up the Messages is domain specific. Every Message has a `msgType`, which indicates the kind of information expressed by the Message. Beyond this, other features to be included in the message are open to the user. There are many design choices to be made about the kind of information captured by the message specification. For example, the information Messages expressed by a message could be captured by a fine-grained data representation (fig. 2a), or as high-level lexical phrases (fig. 2b). As well as the information they wish to express, Messages could also include lexical features such as verb tense or referring expressions which will be used later during Surface Realisation.

3.1.2 Rules

Rules specify the relationships which can hold between Messages and/or constructs of related Messages called ConstituentSets (see 3.1.3). These are based on the Discourse Relations of RST (Mann and

(a)

```
MonthlyRainfallMsg
    period
        month 05
        year 1995
    attribute
        type 'RelativeVariation'
        magnitude
            unit 'inches'
            number 4
        direction '-'
```

(b)

```
MonthlyRainfallMsg
    text '|May 1995 was drier
             than average|'
```

Figure 2: Two different approaches to Message construction. (a) shows a fine-grained data representation, while (b) is a high-level lexical representation

> **Elaboration Rule.** A TotalRainfallMsg can be added as an Elaboration to a MonthlyRainfallMsg if both messages have the same value for Direction (the direction of variation). This rule has a heuristic score of 3.

Figure 3: An example rule for the WeatherExplainer domain

Thompson, 1988). Each Rule specifies a set of inputs (usually two) which are either Messages or ConstituentSets, and certain conditions which must hold between these inputs for the rule to apply. In addition, for the Bottom-Up planning, a Rule will specify a heuristic score or function which will be used to select the best rule to apply when generating the plan. An example rule is given in Figure 3.1.2

3.1.3 ConstituentSets

ConstituentSets can be thought of as the output of applying a Rule to a set of Messages and ConstituentSets. A Rule will select a subset of these to satisfy it's input requirements and the conditions set on those inputs, and output a new ConstituentSets which ties those inputs together with a certain relationship type. For example, the result of applying the rule specified in Figure 3.1.2 to a TotalRainfallMsg and MonthlyRainfallMsg which satisfy the requirement on their Direction would be a ConstituentSet with those two Messages as children and `relType = 'Elaboration'`.

3.2 NLTK FeatStructs

NLTK includes an implementation of a data-structure called FeatStruct which is a mapping from feature identifiers to feature values, where each feature value is either a basic value (such as a string or an integer), or a nested feature structure. Since both Messages and ConstituentSets have this structure, they were implemented as subclasses of `nltk.featstruct.FeatDict`. This package an important method, `subsume` which assists in the implementation. `subsume` relies on the method `unify`.

3.2.1 unify

Unification is an "adding together" of two FeatStructs. Specifically, the unification of two FeatStructs will be the minimal feature structure that contains all feature value assignments from both FeatStructs. If the two features specify different values for some feature, then unification is impossible.

3.2.2 subsume

`fs1` subsumes `fs2` if the unification of the two is equivalent to `fs2`. That is to say, `fs1` contains no features which `fs2` does not, and `fs1` and `fs2` do not differ on the value of any feature.

3.2.3 Using subsume

This `subsume` method is useful for determining which Messages or ConstituentSets can serve as inputs for a given Rule. The condition "ConstituentSets whose nucleus is a MonthTemperatureMsg" could be rendered as "all constituent sets which are subsumed by `ConstituentSet(nucleus= Message('MonthlyTemperatureMsg'))`".

3.3 Input Formats

In keeping with the design of NLTK modules for Context-Free Grammar and other tasks, a design feature I wished to incorporate would be a simple string-based input format for Messages and Rules. This means that users would not have to have detailed knowledge about the workings of the passage in order to be able to specify the Messages and Rules which would make up the Document Structuring system.

3.3.1 Message Input Format

The input for Messages is straight forward. The input format is the nested-tab tree structure shown in Figure 4, where the first line is the msgType, and each subsequent line is either a feature-value pair, or a category name for a nested feature structure. Nested structures are further indented.

3.3.2 Rule Input Format

The input format for Rules is more complicated, since for each rule there are many things which need to be specified. The basic template is as follows:

`Name`(*Input1* `Name1`, *Input2* `Name2`) (*conditions*) : (*return*) : (*heuristic*)

The elements of this template are the following:

- `Name` is the name of the rule. This is a string of alphanumeric characters which helps identify the rule, but is not reflected in the parsed representation
- *Input1* and *Input2* are the specifications of the 'templates' which will subsume the inputs to the rule. See 3.2.3 for an example. Each input can optionally be one of several types, in which case the templates are separated by '|' (see Section 4.1.2 and Figure 6 for examples of this.
- `Name1` and `Name2` are names for the inputs which will be used in the specifications of the *conditions*, *return* and *heuristic*
- *conditions* - Within the brackets is a string which will be evaluated to determine whether potential inputs match this rule. Indexes into the input FeatStructs can be specified with dot indexed (ie. `M1.attribute.direction`), and these while be converted into the proper format for evaluation by the python interpreter.
- *return* specifies the ConstituentSet which will be returned by the Rule. Format is `ConstituentSet('relType', 'nucleusName', 'auxName')`.
- *heuristic* specifies a string which, like conditions, will be evaluated on the inputs to produce the heuristic value.

3.4 Using Document Planner

An example usage for the document planner package using the string input readers would be as follows:

```
with open('msg_file', 'r') as f:
    msg_string = f.read()

with open('rule_file', 'r') as f:
    rule_string = f.read()

msgs = read_messages(msg_string)
rules = read_rules(rule_string)

plan = bottom_up_plan(msgs, rules)
```

msgfile and *rulefile* are files containing the string representations of the messages and rules, respectively.

4 Test Data Sets

As well as implementing the Bottom-Up algorithm, this project included testing the package on two sample domain models. The first is the very simple WeatherExplainer example provided in (Reiter and Dale, 2000). The second is an attempt to use this document planning system as part of a pipeline for the summarization of Evaluative Statements (Carenini et. al, 2006; FitzGerald, 2009).

4.1 WeatherExplainer

The first test data set used was the very simple WeatherExplainer example provided in (Reiter and Dale, 2000). This is a hypothetical NLG system for producing reports of weather conditions which is used throughout the textbook to illustrate the various techniques under discussion. For Document Structuring, they define an example Message, and three Rules.

4.1.1 Messages

(Reiter and Dale, 2000) did not specify a specific set of Messages for the Document Planner. In order to develop and test the system, I created three simple messages which would satisfy the rules specified in 4.1.2 and produce a Document Plan. These messages are meant to represent various information about the weather during the month of June 1996. The messages are shown in Figure 4.

4.1.2 Rules

Three rules are specified in an abstract description (quoted verbatim):

- **Elaboration Rule.** A TotalRainfallMsg can be added as an Elaboration to a MonthlyRainfallMsg if both messages have the same value for Direction (the direction of variation).
- **Contrast Rule.** A TotalRainfallMsg can be added as an Contrast to a MonthlyRainfallMsg if both messages have different Direction values.
- **Sequence Rule.** A MonthlyTemperatureMsg (or a DocumentPlan whose nucleus is a MonthlyTemperatureMsg) and a MonthlyRainfallMsg (or a DocumentPlan whose nucleus is a MonthlyRainfallMsg) can be combined using the Sequence

These rules were rendered into the input format as per Fig. 6. Note that since the **SequenceRule** specifies two possible types for each of its inputs (a Message or a DocumentPlan), rendering these rules makes use of the '|' operator described in 3.3.2.

```
TotalRainfallMsg
    period
        year 1996
        month 06
    attribute
        type 'RelativeVariation'
        magnitude
            unit 'inches'
            number 4
        direction '+'

MonthlyRainfallMsg
    period
        year 1996
        month 06
    attribute
        type 'RelativeVariation'
        magnitude
            unit 'inches'
            number 2
        direction '+'

MonthlyTemperatureMsg
    period
        year 1996
        month 06
    temperature
        category 'hot'
```

Figure 4: Messages for ModelExplainer test data set.

```
bottom_up_plan(messages,rules)
    CS = set(messages)
    return bottom_up_search(CS, rules)

bottom_up_plan(CS,rules)
    if len(CS) == 1:
        return CS
    else:
        options = get_options
        if options == []
            return None
        for o in options:
            fringe = CS - [inputs of o]
            ret = bottom_up_plan(fringe,rules)
            if ret:
                return ret
        return None
```

Figure 5: A rough pseudo-code description of the best-first-search strategy used to implement the bottom-up algorith. [get_options] returns all the possible options for applying the rules to the current set CS, ordered by heuristic score from highest to lowest.

4.1.3 Bottom-Up Implementation

In order to ensure that the planner could back-track if a wrong choice was made during search, the algorithm described in Figure 1 was modified. The algorithm is implemented as a best-first local search using recursive calls to the search method. See Figure 5 for the implementation.

4.1.4 Output

The output of running the Bottom-Up Planner with the above Rules and Messages is displayed in Figure 7. This is the expected output and shows that the algorithm is functioning correctly.

4.2 ASSESS Pipeline

In order to investigate whether this system could be used as part of a real Natural Language Generation system, I experimented with using the Document Planner as part of the ASSESS pipeline (Carenini et. al, 2006; FitzGerald, 2009). ASSESS is a system for Abstractive Summarization of Evaluative statements, such as opinions about a product expressed in user reviews. In Abstractive Summarization, information is first extracted into an abstract data-representation, which is then expressed via an NLG system. This is in contrast to Extractive Summarization, where information is expressed simply by extracting representative sentences from the inputs.

For the ASSESS pipeline, the information extraction system (described in (FitzGerald, 2009)) tags input sen-

```
Elaboration(Message(’MonthlyRainfallMsg’) M1, Message(’TotalRainfallMsg’) M2)
(M1.attribute.direction == M2.attribute.direction) : ConstituentSet(’Elaboration’, M1, M2) : 3

Contrast(Message(’MonthlyRainfallMsg’) M1, Message(’TotalRainfallMsg’) M2)
(M1.attribute.direction != M2.attribute.direction) : ConstituentSet(’Contrast’, M1, M2) : 2

Sequence(Message(’MonthlyTemperatureMsg’)|ConstituentSet(nucleus=Message(’MonthlyTemperatureMsg’)) M1,
    Message(’MonthlyRainfallMsg’)|ConstituentSet(nucleus=Message(’MonthlyRainfallMsg’)) M2)
() : ConstituentSet(Sequence, M1, M2) : 1
```

Figure 6: Rules for ModelExplainer test data set.

```
[ *type*   = ’DPDocument’                                                                          ]
[                                                                                                  ]
[            [             [             [ *msgType* = ’TotalRainfallMsg’                    ] ] ] ]
[            [             [             [                                                   ] ] ] ]
[            [             [             [             [ direction = ’+’                   ] ] ] ] ]
[            [             [             [             [                                   ] ] ] ] ]
[            [             [             [ attribute = [ magnitude = [ number = 4        ] ] ] ] ] ]
[            [             [ *aux*     = [             [             [ unit   = ’inches’ ] ] ] ] ] ]
[            [             [             [             [                                   ] ] ] ] ]
[            [             [             [             [ type      = ’RelativeVariation’   ] ] ] ] ]
[            [             [             [                                                   ] ] ] ]
[            [             [             [ period    = [ month = 6    ]                      ] ] ] ]
[            [             [             [             [ year  = 1996 ]                      ] ] ] ]
[            [             [                                                                   ] ] ]
[            [ *aux*     = [             [ *msgType* = ’MonthlyRainfallMsg’                  ] ] ] ]
[            [             [             [                                                   ] ] ] ]
[            [             [             [             [ direction = ’+’                   ] ] ] ] ]
[            [             [             [             [                                   ] ] ] ] ]
[ children = [             [             [ attribute = [ magnitude = [ number = 2        ] ] ] ] ] ]
[            [             [ *nucleus* = [             [             [ unit   = ’inches’ ] ] ] ] ] ]
[            [             [             [             [                                   ] ] ] ] ]
[            [             [             [             [ type      = ’RelativeVariation’   ] ] ] ] ]
[            [             [             [                                                   ] ] ] ]
[            [             [             [ period    = [ month = 6    ]                      ] ] ] ]
[            [             [             [             [ year  = 1996 ]                      ] ] ] ]
[            [             [                                                                   ] ] ]
[            [             [ *relType* = "’Elaboration’"                                       ] ] ]
[            [                                                                                   ] ]
[            [             [ *msgType*   = ’MonthlyTemperatureMsg’ ]                             ] ]
[            [             [                                       ]                             ] ]
[            [ *nucleus* = [ period      = [ month = 6    ]        ]                             ] ]
[            [             [               [ year  = 1996 ]        ]                             ] ]
[            [             [                                       ]                             ] ]
[            [             [ temperature = [ category = ’hot’ ]    ]                             ] ]
[            [                                                                                   ] ]
[            [ *relType* = ’Sequence’                                                            ] ]
[                                                                                                  ]
[ title    = [ text = None ]                                                                       ]
[            [ type = None ]                                                                       ]
```

Figure 7: Output document plan for the ModelExplainer test data set.

tences with the product features mentioned in the sentence (termed 'Crude Features' or CFs) and a numerical measure of the opinion expressed about that feature. Also, a mapping is calculated between these CFs and a user-defined hierarchy of features (UDFs) which represents the features of the target product. The intuition is that by mapping CFs to UDFs which represent the same feature, opinions can be aggregated, and redundancy reduced. Once this has been achieved, the NLG system must express the opinion trends apparent in the corpus.

In order to be able to test the use of the Document Planner in this domain, I first had to develop a simple method of Content Selection. This is described below, followed by a discussion of the rules used, and the resulting Document Plan output.

4.2.1 Content Selection

I implemented a very simple system for content selection. The numerical opinion scores for each CF were enumerated. These CF scores were there aggregated based on the UDF to which they were mapped. The scores for each UDF were averaged, giving an average opinion score. For each UDF for which opinions were expressed, a AverageOpinionMessage score was created, which contained the following features:

- **udf** - The UDF being evaluated
- **udf_parent** - The parent of this UDF in the UDF hierarchy
- **valence** - The absolute value of the average strength of opinions expressed about this UDF
- **polarity** - '+' or '-' depending on whether the average opinion was positive or negative
- **numOpinions** - The number of opinions which were expressed about this UDF

An example message produced by this simple Content Selection scheme is shown in Figure 8.

4.2.2 Rules

The Rules used for the ASSESS Document Planner were conceptualised as follows:

- **Conjunction** - An AverageOpinionMessage can be related to another AverageOpinionMessage if they share `udf_parents` and their opinion `polarity` is the same.

  This represents the situation where two similar UDFs could be conjucted, perhaps within the same sentence. For example: "Users liked both the Sound Quality and Video Quality".

- **Conjunction** - An AverageOpinionMessage can be related to another AverageOpinionMessage if they share `udf_parents` and their opinion `polarity` is different.

  This represents the situration where two similar UDFs have different opinions, and these could be contrasted. For example: "Users really liked the Sound Quality but did not like the Video Quality".

- **WeakExplanation** - An AverageOpinionMessage can be related to another AverageOpinionMessage if the UDF of the first is the UDF parent of the other and they share the same opinion polarity.

  This represents the situation where the opinions about one UDF could be explained by the opinions about one of its child UDFs. For example: "Users were negative about the Picture Quality because they felt that Progressive Scan was poor."

- **StrongExplanation** - An AverageOpinionMessage can be related to another AverageOpinionMessage if the UDF of the first is the UDF parent of the other and they share the same opinion polarity.

  This represents the situation where the opinions about one UDF could be explained by the opinions about two of its child UDFs. For example: "Users were negative about the Picture Quality because they felt that Progressive Scan and Colour Contrast were poor."

- **Sequence** - Any Message or ConstituentSet can be linked to another Message or ConstituentSet in a Sequence. This is to ensure that a DocumentPlan is always possible, even if the conditions for any of the above rules cannot be met. This Rule is given the lowest heuristic priority.

The implementation of these rules can be seen in the Appendix code for `assess_test.py`.

4.2.3 Results

The simple Content Selection scheme and Document Planning system described above were run on the Apex corpus (Hu and Liu, 2004) and CF-UDF mapping gold-standard. Content Selection produces 12 AverageOpinionMessages. These are then input to the Bottom-Up Planner, with the Rules specified above.

The Document Planner successfully produces a Document Plan for these Rules and Messages. All rules are used at least one, with the exception of **StrongExplanation** (since cases where this Rule would be applicable are rare for such a small number of messages). In terms of the goal of this project, this constitutes a proof-of-concept. Further evaluation of the produced Document Plan in terms of coherency is not possible at this point,

```
[ *msgType*   = 'AverageOpinionMessage'     ]
[ numOpinions = 17                          ]
[ polarity    = '-'                         ]
[ udf         = 'Universal Remote Control'  ]
[ udf_parent  = 'Extra Features'            ]
[ valence     = 1.1764705882352942          ]
```

Figure 8: An example message produced by the simple Content Selection scheme for ASSESS

nor would it be especially enlightening as it would say more about the Content Selection scheme and specified rules than the Document Planning system itself.

5 Discussion and Future Work

From the tests I have run on the two sample domains presented in 4, the implemented Document Planner seems to work well, and be fairly easy to use. However, the two test data sets are relatively simple, and do not represent the complexity that would be present in a real NLG task. Furthermore, the Rules and heuristics specified were arbitrary, and were not evaluated for coherence of the produced Document Plan. In future work, it would be necessary to see whether this simple Document Planning frame-work could be scaled up to produce results which perform well on coherence tests. One line of research would be to see if a system similar to SEA (Carenini et. al, 2006) could be implemented using this system.

In order for the system to scale-up efficiently, input formats must be robust and easy to use. The text input format makes specification of Messages very easy. The input format for Rules, however, is not to the level of ease-of-use and robustness that I would like. More complicated rules become unwieldy and syntax heavy. This is a difficult problem, as ease of specification must be balanced against expressiveness and generalizability. Through use with more realistic problems, it should become more apparent which features are necessary, and what capabilities are required for the input format.

A task which remains to be completed is the creation of comprehensive documentation for the Document Planning system. If this is to be a useful contribution to NLTK, the code must be well commented and documented. Code commenting is almost to a high-enough standard, but a comprehensive manual remains to be created. Once this is finished, the system will be contributed to NLTK.

6 Conclusion

In this project I have presented in implementation of a generalized bottom-up Document Planner, based on the ideas presented in (Reiter and Dale, 2000). This system was tested with two relatively simple examples, and seemed to peform well, although more work is needed to see if this simple framework can scale up to more complex NLG tasks. Once the the code is well-documented, this system will be contributed to NLTK.

References

Bird, S., Klein, E., and Loper, E. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc. Print and online.

Carenini, G., and Moore, J.D. 2006. Generating and Evaluating Evaluative Arguments. *Artificial Intelligence*, 170(11):925-952

Carenini, G., Ng, R., and Pauls, A. 2006. Multi-Document Summarization of Evaluative Text. *Proc. of the Conf. of the European Chapter of the Association for Computational Linguistics*.

FitzGerald, N. 2009. A Complete Pipeline for Semantic Evaluation Summarization.

Hu, M., and Liu, B. 2004. Mining opinion features in customer reviews. *Proc. AAAI*.

Mann, W. and Thompson, S. 1988. Rhetorical Structure Theory: Toward a fuctional theory of text organization. *Text*.3:243-281

Marcu, D. 1997. From local to global coherence: A bottom-up approach to test planning. *Proceesings of the Fourteenth National Conference on Artificial Intelligence*.629-635.

Reiter, E. and Dale, R. 1997. Building applied natural language generation systems. *Natural Language Engineering*.3(1):57-87

Reiter, E. and Dale, R. 2000. *Building Natural Language Generation Systems (Studies in Natural Language Processing)*. Cambridge University Press, New York. Print.

A Code

The following appendices (and `code` directory) include the source code used for this project. There are 5 files:

- `documentplanner.py` is the main module which contains the Bottom-Up planning method and necessary classes

- `weather_test.py` is the script to test the WeatherExplainer example
- `assess_test.py` is the script to test the ASSESS example, and contains the simple Content Selector
- `util.py` includes various utility methods used by the other scripts
- `MyTree.py` is a module file used by ASSESS

NLTK must be installed in order to run these scripts. The two data files needed for the ASSESS test are in the 'data' folder.

To run the two test scripts, the following commands are used:

```
>python weather_test.py
>python assess_test.py
```