E-txt2db: Giving Structure to Unstructured Data (Extended Abstract)

Gonçalo Simões
INESC-ID and Technical University of Lisbon,
Avenida Prof. Cavaco Silva,
Tagus Park, 2780-990
Porto Salvo, Portugal
goncalo.simoes@ist.utl.pt

ABSTRACT
A large amount of information handled in organizations is stored as unstructured or semi-structured documents (e.g., reports or e-mail messages). It is not easy, for a software tool, to use these type of documents, which means that much of this information is ignored. The Information Extraction area offers a set of techniques that help extracting text segments from these documents in order to make them useful for a user in a given domain. Typically, to specify Information Extraction programs, a user can use: (i) procedural languages to develop the program from scratch; (ii) ETL (Extraction, Transformation and Loading) tools; (iii) Information Extraction frameworks or; (iv) declarative languages for Information Extraction. All these solutions have advantages and disadvantages.

This work proposes a framework for Information Extraction called E-txt2db, that tries to combine the advantages of the existing solutions for the specification of Information Extraction programs. E-txt2db offers declarative operators based on the semantics of the following Information Extraction tasks: segmentation, classification, association, normalization, coreference resolution. The work of this thesis focuses on the development of the classification operator of E-txt2db, proposing its formalization, a Java API to create, execute and evaluate classification models and an SQL-like syntax to specify classification programs.

Keywords
Information Extraction, Declarative Languages, Machine Learning

1. INTRODUCTION
Statistics of 2005 indicate that more than 80% of business knowledge was stored in digital format [14]. With the increasing use of information technologies in business, it is expected that this percentage grows even more. Much of the information stored is vital to the business only after properly processed. Unfortunately, no Human Being is able to read, understand and synthesize large amounts of data in a reasonable time [5]. A solution for this problem is the use of software tools for searching, accessing and analyzing data. However, much data in companies (such as e-mail messages, reports and Web pages) are not organized in a structured way.

Information Extraction is a scientific area that encloses a set of techniques to extract relevant information in a given domain from unstructured or semi-structured text, and represent it in a coherent format that is understood by other software applications. Our work proposes a framework for the specification and execution of Information Extraction programs called E-txt2db. E-txt2db offers declarative operators based on the semantics of the following Information Extraction tasks [16]: segmentation, classification, association, normalization, coreference resolution. The focus of this work is on the classification operator of E-txt2db. We propose its formalization, a Java API to create, execute and evaluate classification models and an SQL-like syntax to specify classification programs.

The rest of this paper is organized into five sections. In Section 2, the basic concepts about information extraction are described. Section 3 presents the state of the art of existing solutions to specify Information Extraction programs. Section 4 describes the E-txt2db framework, namely the implementation of the classification operator and its Java implementation and an SQL-like syntax for the creation, execution and evaluation of classification programs. Section 5 summarizes the experiments performed to validate the instantiation of E-txt2db for the classification operator. Finally, Section 6 summarizes the main topics addressed in this work.

2. CONCEPTS
This section describes the basic concepts related to Information Extraction that were used in this work. Section 2.1 presents a division of the Information extraction into tasks and the techniques used for each task. Section 2.2 describes the measures used to evaluate the quality of Information Extraction results.

2.1 Information Extraction tasks
There is no unanimity in what concerns the decomposition of the Information Extraction activity into tasks. In this thesis, we use the decomposition of the Information Extraction activity that is usually adopted by the machine learning community [16]. The considered tasks are: Segmentation, Classification, Association, Normalization and Coreference resolution.

2.1.1 Segmentation
The Segmentation task divides the text into atomic elements, called segments or tokens. Even though this task
is simplified for Western languages due to the existence of whitespaces separating words, there are some cases in which simple whitespace separation may not be enough [20]. Usually, segmentation for these cases is performed using rules that show how to handle each case.

The major problems related to this task can be found in oriental languages. For example, Chinese does not have whitespaces between words [11]. For this reason, solving the problems described above is not enough in this language. In these cases, it is typically necessary to use external resources. Lexicons and grammars can also be used in order to accomplish the task of segmentation using syntactic or lexical analysis. Another approach for segmentation in Chinese uses techniques based on statistics. An example is the system described in [11], which uses N-grams and the Viterbi algorithm [8] applied to segmentation.

### 2.1.2 Classification

The Classification task determines the type of each segment obtained in the segmentation task. In other words, it determines the field of the output data structure where the input segment fits. The result of this task is the classification of a set of segments as entities, which are elements of a given class potentially relevant for the extraction domain.

The rule-based techniques used in the classification task are usually based on linguistic resources, such as lexicons and grammars [7].

One of the most popular approaches to undertake classification is machine learning. Machine learning techniques used in this task are usually supervised, which means that an annotated corpus is needed. Five of the most common supervised learning techniques are the Hidden Markov Models (HMM), Maximum Entropy Markov Models (MEMM) [17], Conditional Random Fields (CRF) [15], Support Vector Machines [13] and Decision Trees [21].

### 2.1.3 Association

The Association task seeks to find how the different entities found in the classification task are related. The systems that perform extraction of relationships are less common than the ones that perform the classification task [16]. This happens due to the difficulty in achieving good results in this task.

Many techniques in the association task are based on rules. The simplest approach uses patterns to extract a limited set of relationships. A more generic rule-based approach for association is based on syntactic analysis. Often, the relationships that we want to extract are grammatical relationships [10]. For example, a verb may indicate a relationship between two entities.

The association task can also use machine learning techniques. One of the first machine learning approaches was based on probabilistic context-free grammars [18]. These grammars differ from regular context-free grammars, because they have a probability value associated to each rule. When the syntactic analysis is undertaken, it is possible to find many syntactic trees. By using probabilistic rules, the probability of each tree is computed and the most probable tree is chosen.

### 2.1.4 Normalization

The Normalization task is required because some information types do not conform to a standard format. For instance, there can be several representations of hours like “3pm”, “3h”, “15:00”. This format heterogeneity may pose difficulties in the comparison between entities.

The normalization task transforms information to a standard format defined by the user. In the example of the hours mentioned above, the user can define that all the hours shall be converted to a standard format, for example, “15h00”. This task is typically achieved through the use of conversion rules that produce a standard format previously chosen.

#### 2.1.5 Coreference resolution

Coreference arises whenever the same real world entity is referred in different ways in a text fragment. This problem may arise due to the use of: (i) different names describing the same entity (e.g., the entity “Bill Gates” can be found in the text as “William Gates”), (ii) classification expressions (e.g., a few years ago, “Bill Gates” was referred as “the world’s richest man”), (iii) pronouns (e.g., in the sequence of sentences “Bill Gates is the world’s richest man. He was a founder of Microsoft”, the pronoun “He” refers to “Bill Gates”).

Rule-based approaches for coreference resolution usually take into account semantic information about entities. A machine learning approach for coreference resolution is described in [3]. This approach is based on clustering algorithms for grouping similar entities.

### 2.2 Evaluation of Information Extraction systems

The measures used to evaluate Information Extraction systems are based on the metrics from Information Retrieval: recall and precision. Although the names remain, the method for calculating the measures are amended in order to consider the general cases of Information Extraction. This section defines these measures.

#### 2.2.1 Recall

Recall gives the ratio between the amount of information correctly extracted from the texts and the information available in texts. Thus, recall measures the amount of relevant information extracted and is given by Equation (1):

\[
\text{recall} = \frac{C}{P}
\]

where \(C\) represents the number of correctly extracted records while \(P\) represents the total number of records that should be filled.

The disadvantage of this measure is the fact that it returns high values when we extract all the correct and incorrect information from the text.

#### 2.2.2 Precision

Precision is the ratio between the amount of information correctly extracted from the texts and all the information extracted. The precision is then a measure of confidence on the information extracted and is given by Equation (2):

\[
\text{precision} = \frac{C}{C + I + O}
\]

where \(C\) represents the number of records correctly filled, \(I\) represents the number of records incorrectly filled and \(O\) represents the number of records that have been filled and should not have been.
The disadvantage of this measure is that we can get high results extracting only information that we are sure to be right and ignoring information that are in the text and may be relevant

2.2.3 F-measure

The values of recall and precision may enter in conflict. When we try to increase the recall, the value of precision may decrease and vice versa. The F-measure was adopted to measure the general performance of a system, balancing the values of recall and precision. It is given by Equation (3):

\[ F_{\text{measure}} = \frac{(\beta^2 + 1) \times P \times R}{\beta^2 \times P + R} \]  

where \( R \) represents the recall, \( P \) represents the precision, \( \beta \) is an adaptation value of the equation that allows to define the relative weight of recall and precision. The value \( \beta \) can be interpreted as the number of times that the recall is more important than accuracy. A value for \( \beta \) that is often used is 1, in order to give the same weight to recall and precision. In this case, the F-measure value is obtained through Equation (4):

\[ F_1 = \frac{2 \times P \times R}{P + R} \]  

3. RELATED WORK

There are several solutions to specify Information Extraction programs. The following four solutions are among the most popular ones: (i) programming languages; (ii) declarative languages for Information Extraction; or (iii) ETL (Extract Transform and Load) tools.

The first approach is to use a programming language, like C or Java to write an Information Extraction program. With these languages, a user can write a program from scratch for a given input data source because they have high expressivity. However, implementing Information Extraction techniques from scratch is an hard work that takes too much time. For this reason, users who use these languages to develop Information Extraction programs are typically assisted by Information Extraction frameworks. In these frameworks, some Information Extraction functionalities are pre-defined and implemented. The code produced is more reusable, the development time and the number of code lines developed are significantly shorter when compared with the solution of writing a program from scratch. Many of these frameworks offer a wide selection of state-of-the-art Information Extraction techniques. The user may select which one to use in each situation. The main disadvantage of these frameworks is the fact that each Information Extraction technique is implemented by a fixed algorithm. Thus, it cannot be optimized as the queries in a Relational Database Management System. Another disadvantage is that these frameworks are usually hard to learn due to their complexity and sometimes to the lack of documentation. Following, we present some of these frameworks:

1. Lingpipe\(^1\): Java framework for Natural Language Processing. It encloses modules for segmentation, classification and coreference. It offers rule-based and machine learning algorithms.

\(^1\)http://alias-i.com/lingpipe/

2. Minorthird\(^2\): Java framework that offers machine learning algorithms for document and sequence classification. It contains a wide variety of algorithms.

3. Mallet\(^3\): Java framework that offers machine learning algorithms for document and sequence classification. The variety of algorithms is not as wide as in Minorthird.

4. Gate [6]: infrastructure for Natural Language Processing software development. Gate Information Extraction techniques are included in a module called ANNIE. ANNIE divides Information Extraction processes into tasks offering operators for them.

The second approach consists of using a declarative language to specify an Information Extraction program. Declarative languages allow a separation between the logic specification (what is executed) and the physical execution (how it is executed). With this kind of approach, it is possible to have multiple execution possibilities for a given logical specification of an Information Extraction program. It is then possible to choose the more efficient execution alternative for each situation. Another advantage of declarative languages is the high level of abstraction they offer. They do not require much programming knowledge to create an application. The inconveniences of these languages are the following two. First, their expressivity is limited, since there are some tasks for which the predicates or operators supported are not enough. Second, the techniques offered are usually limited. For example, these languages usually do not support machine learning techniques. Next, we present some of the declarative languages for Information Extraction that have been proposed in the literature:

1. XLog [23]: language based on Datalog [4]. Offers a set of predicates for Information Extraction and supports the addition of new ones written with a procedural language.

2. ALog [22]: extension of XLog that tries to overcome the complexity incurred in the development of new Information Extraction predicates. It supports a new methodology that makes the specification of predicates as declarative as possible.

3. ELog [9]: language internally used by the Listo system [2]. It is based in Datalog but it only extracts information from HTML documents.

4. System T [19]: offers a simple data model and operators for rule-based Information Extraction. System T algebra is implemented by the AQL (Annotation Query Language), a SQL-like language under development by IBM.

Finally, it is possible to use the functionalities of ETL tools such as Microsoft SQL Server Integration Services [12]. These tools offer pre-defined operators for extraction, transformation and loading of data from several sources. The operators offered by these tools make it easy to specify and maintain an ETL program. Moreover, they are able to deal

\(^2\)http://minorthird.sourceforge.net/

\(^3\)http://mallet.cs.umass.edu/
with multiple data sources. However, the tools have limited expressive power. In fact, ETL tools were not designed specifically for Information Extraction and many of the state of the art Information Extraction techniques are not available (in particular the machine learning techniques). To extend the available techniques and functionalities of ETL tools, the user is usually forced to write ad-hoc code. Another drawback of this approach is the difficulty to optimize programs. Since the supported operators run fixed algorithms, it is not possible to choose, among different execution possibilities, the most efficient for a given input data set.

4. THE E-TXT2DB FRAMEWORK FOR CLASSIFICATION

E-txt2db is a framework for specifying and executing Information Extraction programs. This framework offers a declarative operator for each task that composes an Information Extraction process, namely Segmentation, Classification, Association, Normalization and Coreference Resolution, as described in Section 2.1. For each operator, there is a set of pre-implemented techniques available to create models that describe how an Information Extraction task is executed (extraction models).

In order to specify an Information Extraction process using E-txt2db, the user writes a specification using an SQL-like language. There is a specific syntax for creating, executing and evaluating a model for each Information Extraction operator. The user that creates an extraction model chooses one of the pre-implemented techniques available according to the needs of the model he wants to develop. After the creation, he stores the model for further use. Someone who uses a previously created classification model does not need to know which technique it uses.

The approach used by E-txt2db mixes the advantages of the Information Extraction frameworks and the declarative languages for Information Extraction described in Section 3. Similarly to the Information Extraction frameworks, the fact that there are several pre-implemented techniques for each operator turns it easy to specify and maintain Information Extraction programs. The code produced is highly reusable since the created extraction models are stored for further reuse. Since the language offered by E-txt2db for specifying programs is declarative, the programs are amenable to optimization. E-txt2db supports a set of operators for extracting information. These operators can be composed to produce an Information Extraction program. This way, the user is able to specify an Information Extraction activity at a high level of abstraction. As a consequence, Information Extraction programs are easily specified.

The expressive power of E-txt2db is limited because the library of techniques available is not exhaustive. However, the E-txt2db library of techniques is easily extensible. In fact, new techniques can be added by creating one new Java class and adapting the SQL-like syntax to support it.

Figure 1 presents the architecture of the C-txt2db framework. The input consists of a text. The output consists of relevant data extracted from the input and annotated according to classes. The architecture encloses three main modules: the Java CEE Engine, the library of techniques, and the parser. The text segmentation module corresponds to a basic implementation of the segmentation task. The Java CEE Engine is responsible for the main functionalities of the operator: Creation, Execution and Evaluation (CEE) of classification models. The available techniques for each operator are stored in the library of techniques. When creating a model, the Java CEE Engine accesses the library of techniques in order to use the algorithms available to implement the chosen technique. The module responsible for interpreting the specification is the parser. The parser accepts as input the specification program in the corresponding SQL-like syntax and translates it in order to call methods of the Java CEE Engine.

Figure 1: Architecture of the E-txt2db framework

This Section reports on the implementation of the classification operator. Section 4.1 describes the Architecture of the classification operator CEE Engine. Section 4.2 proposes an SQL-like syntax for the creation, execution and evaluation of classification models.

4.1 The Architecture of the Classification operator CEE Engine

The CEE engine for each operator is responsible for the Creation, Execution and Evaluation of extraction models. Figure 2 shows the UML class diagram of the Classification operator CEE engine.

The functionalities of the CEE Engine for the classification operator are the creation, the execution and the evaluation of a classification model. Therefore, the concept of classification model is the most important in the classification operator CEE Engine. Classification models are represented by instances of classes that implement the interface ClassificationModel. The creation of classification models is achieved through objects of the ClassificationModelCreator class.
4.2 SQL-like syntax for the classification operator

The E-txt2db classification language offers three types of queries: (i) creation queries, that are responsible for the creation of classification models; (ii) execution queries, that perform a classification operator execution for a given corpus and using for a given model. (iii) evaluation queries, that evaluate the results obtained by a classification model. In this section, we describe the three types of queries.

4.2.1 Creation queries

The creation queries implement the functionalities of the CEE Engine ClassificationModelCreator. The syntax for the creation queries supports several variations in order to produce classification models based on the available classification techniques.

The creation of a classification model based on regular expressions is achieved by providing a regular expression that must be matched by the resulting classified segments and the class that must be assigned to the segments that match the regular expression. The following code creates a classification model called timeModel that classifies as time all the segments that match the regular expression "([0-9][0-9]):([0-9][0-9][a-pm])?".

CREATE CLASSIFICATION MODEL AS timeModel USING REGEX "([0-9][0-9]):([0-9][0-9][a-pm])?"
TO FIND time

A second possibility is to create a dictionary-based classification model. In this case, the user must provide the path to the dictionary file. The following code creates a classification model called namesModel that classifies segments of the text that match an entry of the dictionary in "/resources/namesDictionary.txt". The class that is assigned to the segment is indicated in the dictionary file.

CREATE CLASSIFICATION MODEL AS namesModel USING DICTIONARY "/resources/namesDictionary.txt"

In order to take full advantage of the potentialities of MEMM, CRF and SVM, it is common to use additional knowledge about the segments of the corpus when training and executing the classification models. Each additional piece of knowledge included in a classification process is called a feature. The syntax for the creation of classification models supports the introduction of features in the process. E-txt2db offers a wide variety of features that include, among others, information about the value of a segment, information about an automatically generated regular expression the segment matches, information about the type of characters that constitute the segment, information of whether the segment is present in a dictionary or the classification provided by a previously trained classification model.

Additionally to the syntax variations for each of the techniques, E-txt2db supports the creation of classification models as the union of previously created classification models. To create such classification model, a user must provide a list of classification models. The following code creates a model to classify segments as time (using the timeModel model), speaker or location (using the seminarsModel mode):

CREATE CLASSIFICATION MODEL AS seminarsTimeModel USING CLASSIFICATION MODEL UNION timeModel, seminarsModel

4.2.2 Execution queries

The execution queries are used to perform a classification task using a previously created classification model. These queries applied to a given text produce a classified copy of...
the text that is composed by a set of classified segments marked with XML tags. The functionality of these queries correspond to the functionalities of the CEE Engine \texttt{ClassificationEvaluator} class.

Unlike the creation queries, the syntax for the execution of classification models does not change according to the technique used. The following code executes the \texttt{seminarsTimeModel} model created as an example in Section 4.2.1 in order to classify segments of the text in "/.text.txt" as speaker or time:

\begin{verbatim}
CLASSIFY speaker, time
FROM "text.txt"
USING seminarsTimeModel
\end{verbatim}

### 4.2.3 Evaluation queries

The evaluation queries determine the effectiveness of a classification model by measuring the recall, precision and the f-measure with a given testing corpus. The functionality of these queries correspond to the functionalities of the CEE Engine \texttt{ClassificationEvaluator} class.

The basic syntax for the evaluation queries is very similar to the syntax for the execution queries. The user must provide the classes, the path for the testing corpus (that must be a text file with XML tags classifying text segments) and the model used in the evaluation. The following code evaluates the \texttt{seminarsTimeModel} model created as an example in Section 4.2.1 in order to classify segments of the text in "/.text.txt" as speaker or time:

\begin{verbatim}
EVALUATE CLASSIFICATION OF speaker, time
FROM "text.txt"
USING seminarsTimeModel
\end{verbatim}

The result of this evaluation query is a text with three lines. The first line shows the value of recall, the second line presents the value of precision and the third line displays the value of the f-measure with \( \beta = 1 \).

In order to change the measures used in the evaluation, a user must write an additional line indicating the measures to be used. The measures available are recall, precision and f-measure with any value of \( \beta \). The following code evaluates the \texttt{seminarsTimeModel} model using only recall and f-measure with \( \beta = 2 \).

\begin{verbatim}
EVALUATE CLASSIFICATION OF speaker, time
FROM "text.txt"
USING seminarsTimeModel
MEASURING recall, f-measure 2
\end{verbatim}

## 5. EVALUATION

In order to demonstrate the advantages of E-txt2db, we performed the set of experiments that are described in this Section. Since we only developed a proof-of-concept for the classification operator, all the experiments performed are focused in the classification task. This decision imposed two limitation to the evaluation of the E-txt2db framework. First, it is not possible evaluate the customization of the Information Extraction process, since it is only possible to select and compose classification models. Second, even though we tested the modification of the execution algorithm without changing the specification of the program, this change was performed manually and thus no cost model or heuristics were considered.

### 5.1 Easy specification

To determine how easy it is to specify a classification task, we compare the specification of classification programs with E-txt2db and Minorthird. During the tests, ten Computer Science students were divided into two groups. One group used Minorthird to develop a classification program while the other group used the Java API of the E-txt2db CEE Engine. Each element of the group wrote his own program individually. To help them, we provided a document with the description of the assignment, the corpus format and the functionalities of the main classes that we recommended them to use.

To evaluate how easy it is to create a classification program with each framework, we considered the following two measures: (i) the time spent by each student to write the program; (ii) the number of code lines of the program produced.

Table 1 shows the results of the evaluation of the programs written by students who used E-txt2db. Table 2 presents the results of the evaluation of the programs written by students who used Minorthird.

#### Table 1: Evaluation of the programs written by the students using E-txt2db

<table>
<thead>
<tr>
<th>Student</th>
<th>Time spent (minutes)</th>
<th>Number of code lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student 1</td>
<td>15</td>
<td>21</td>
</tr>
<tr>
<td>Student 2</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>Student 3</td>
<td>14</td>
<td>21</td>
</tr>
<tr>
<td>Student 4</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>Student 5</td>
<td>27</td>
<td>21</td>
</tr>
<tr>
<td>Average</td>
<td>21.2</td>
<td>21</td>
</tr>
</tbody>
</table>

#### Table 2: Evaluation of the programs written by the students using Minorthird

<table>
<thead>
<tr>
<th>Student</th>
<th>Time spent (minutes)</th>
<th>Number of code lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student 6</td>
<td>78</td>
<td>72</td>
</tr>
<tr>
<td>Student 7</td>
<td>59</td>
<td>32</td>
</tr>
<tr>
<td>Student 8</td>
<td>73</td>
<td>39</td>
</tr>
<tr>
<td>Student 9</td>
<td>83</td>
<td>39</td>
</tr>
<tr>
<td>Student 10</td>
<td>65</td>
<td>38</td>
</tr>
<tr>
<td>Average</td>
<td>71.6</td>
<td>44</td>
</tr>
</tbody>
</table>

By comparing the results shown in Table 1 and 2, we observe that the average time spent by the students to create the classification program with Minorthird is almost three times higher than the average time spent to create a classification program with E-txt2db. Moreover, the resulting Minorthird programs have an average number of code lines that is almost the double of the average number of code lines of the E-txt2db programs (which curiously was the same for all the students). These results further suggests that E-txt2db can help reduce the time it takes to develop a simple classification program using machine learning techniques programs. Moreover, the code produced by E-txt2db is more concise than the code produced by Minorthird.

### 5.2 Local refinement

We performed the experiment described in this section to show how easy it is to refine the results of an Information Extraction program with the E-txt2db framework to improve the accuracy of the results obtained. First, we specified a simple Information Extraction program that uses a corpus of 52 documents to train a Support Vector Machine model
to extract information about the location, the speaker, the starting time and the ending time of a seminar. This model is then evaluated with another corpus of 52 documents to determine the precision, recall and f-measure (with $\beta = 1$) values. Both the training and the testing corpora are documents that contain e-mails with Carnegie Mellon University seminar announcements.\footnote{http://www.cs.cmu.edu/~dayne/SeminarAnnouncements/} The corpus uses the default data format accepted by the E-txt2db framework which is plain text with XML tags. The Information Extraction program is the following one:

```sql
CREATE CLASSIFICATION MODEL AS seminarsModel
USING SVM TRAINED WITH ".//SeminarsTraining/"
TO FIND speaker, location, stime, etime
EVALUATE CLASSIFICATION OF speaker, location, stime, etime
FROM ".//SeminarsTesting/"
USING seminarsModel
```

The accuracy results returned by the evaluation of the classification model are shown in Table 3.

Table 3: Accuracy of a Support Vector Machines classification program

<table>
<thead>
<tr>
<th>speaker</th>
<th>location</th>
<th>stime</th>
<th>etime</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td>recall</td>
<td>72.12%</td>
<td>58.18%</td>
<td>83.94%</td>
<td>66.27%</td>
</tr>
<tr>
<td>precision</td>
<td>72.12%</td>
<td>58.18%</td>
<td>83.94%</td>
<td>66.27%</td>
</tr>
<tr>
<td>f-measure</td>
<td>72.12%</td>
<td>58.18%</td>
<td>83.94%</td>
<td>66.27%</td>
</tr>
</tbody>
</table>

The values presented in Table 3 show that the accuracy of the model is far from good. The values of recall are very weak when extracting speaker values. The values of precision are slightly better but can still be improved. Our objective during this test is to iteratively refine the results of the program. At each iteration, we analyzed the accuracy results and explored some properties of the data that could be used to influence the Support Vector Machine model. After four iterations the resulting program was as follows:

```sql
CREATE CLASSIFICATION MODEL AS seminarsModel
USING SVM TRAINED WITH ".//SeminarsTraining/"
TO FIND speaker, location, stime, etime
EVALUATE CLASSIFICATION OF speaker, location, stime, etime
FROM ".//SeminarsTesting/"
USING seminarsModel
```

This test leads us to two main conclusions about the E-txt2db refinement capabilities: (i) it is easy to locally refine a classification model; and (ii) when refining a model, a different program that uses the resulting model to perform a classification task does not need to change. The later property is a very important property because, if the classification model is used as a small part of a more complex Information Extraction program, a user can refine the classification model without any impact in the rest of the program.

5.3 Independence of the data format

To evaluate the capability of E-txt2db to deal with several text formats, we conducted a test in which information about the location, the speaker, the starting time and the ending time of a seminar was extracted from two different corpora. With this test, we wanted to show the following two properties of E-txt2db: (i) the E-txt2db classification models can be used to classify segments from texts with any data format; and (ii) the code to specify, execute or evaluate E-txt2db classification models does not need to undergo many changes when the data source is modified.

The first corpus was composed by 484 e-mails containing CMU seminar announcements. The format of these e-mails is plain text with XML tags classifying text segments. From the 484 e-mails, we separated 50% for training and 50% for testing. The second corpus is composed by 88 Web pages from the UCLA Department of Materials Science and Engineering. In this case, we could not use the same format because the HTML tags would be confused by the XML tags that classify the segments. For this reason, we replaced the symbols "<" and ">" by the symbols "\" and \\\. From the 88 Web pages, we separated 50% for training and 50% for testing.

For each of the corpora, a program was written to create a classification model that uses machine learning techniques. Because our interest in this test was not accuracy, HMM were chosen for fast training and evaluation. The resulting Information Extraction program to classify e-mails corpus is shown next:

```sql
CREATE CLASSIFICATION MODEL AS seminarsHMMModel
USING HMM TRAINED WITH ".//SeminarsTraining/"
TO FIND speaker, location, stime, etime
EVALUATE CLASSIFICATION OF speaker, location, stime, etime
FROM ".//SeminarsTesting/"
USING seminarsModel
```

E-txt2db can deal with both plain text and HTML input texts. The fact that the user must provide the corpus with XML tags classifying segments is no problem when dealing
with HTML or XML texts, because the user can configure the program to use variations of XML tags to classify the segments. This configuration is shown in the first two lines of the program that classifies web pages.

Since machine learning techniques are available, a classification program specified with E-txt2db does not change much when the input data source changes. This can be observed in this test since the two programs created are very similar. The differences in the code produced are the lines to configure the format of the training corpus and the paths to the files.

5.4 Amenability to optimization

In the context of this work, we did not study any optimization technique for the classification task. Therefore, we do not evaluate the use of optimization techniques in E-txt2db. Instead, we prove that E-txt2db is amenable to optimization.

The scenario built for this purpose consists of an Information Extraction program to extract person names from texts. This program uses a dictionary of person names. We use two algorithms for implementing the dictionary-based classification technique. The goal is to show that it is possible to internally replace the dictionary-based classification algorithm without changing the specification of the Information Extraction program to extract proper names. We used a non-classified version of the Brown corpus with a set of 400 English texts. The Information Extraction program written is the following one:

```
CREATE classIFICATION MODEL AS namesModel
USING DICTIONARY ./resources/namesDictionary.txt

CLASSIFY name
FROM ./brownUnlabeled/

USING namesModel
```

The first implementation of the dictionary-based classification algorithm was developed from scratch using the Map Java class to store the entries of the dictionary and their classification. This implementation is shown in the first two lines of the program that classifies web pages.

The scenario built for this purpose consists of an Information Extraction program to extract person names from texts. This program uses a dictionary of person names. We use two algorithms for implementing the dictionary-based classification technique. The goal is to show that it is possible to internally replace the dictionary-based classification algorithm without changing the specification of the Information Extraction program to extract proper names. We used a non-classified version of the Brown corpus with a set of 400 English texts. The Information Extraction program written is the following one:

```
CREATE CLASSIFICATION MODEL AS namesModel
USING DICTIONARY "./resources/namesDictionary.txt"

CLASSIFY name
FROM "/brownUnlabeled/"

USING namesModel
```

The second implementation of the dictionary-based classification algorithm uses a Lingpipe class called ExactDictionaryChunker. This class implements the Aho-Corasick algorithm for the dictionary-based classification. This algorithm is linear in the number of segments in the input plus the number of classified segments. When using the Aho-Corasick algorithm, the Information Extraction program finished the classification task in 4:49 minutes. The second implementation of the dictionary-based classification algorithm uses a Lingpipe class called ExactDictionaryChunker. This class implements the Aho-Corasick algorithm for the dictionary-based classification. This algorithm is linear in the number of segments in the input plus the number of classified segments. When using the Aho-Corasick algorithm, the program finished the classification task much faster, in 2:01 minutes. An E-txt2db optimizer would choose the second implementation of the classification operator.

The most important idea to take from this test is that it is possible to change the algorithm used in the classification task without changing the specification.

5.5 Customization

Since we only develop the classification operator of E-txt2db in the context of this thesis, it was not possible to evaluate the possibility to customize the whole Information Extraction process. However, by analyzing the code produced during the test described in Section 5.2, we can see how a classification task can be customized. In this test, the following two types of customization are highlighted: (i) union of different classification models; and (ii) use of classification models to generate features.

The union of classification models creates a new model that is capable of classifying segments with all the classes from the original ones. The experiment reported in Section 5.2 illustrates this type of customization. In fact, the classification of speaker and location is separated from the classification of starting time and ending time.

Using classification models to generate features is another form of composition of classification models. The objective of this type of customization is to produce hints that may help in the classification process of another model. The experiment reported in Section 5.2 shows this type of customization. A classification model for Part-of-speech tagging, called HMMPOSTagger, gives hints to classify segments as speaker or location.

Even if these customizations are local to the classification task, they may be very useful because they offer the possibility to generate complex classification models using simple ones that can be locally refined.

6. CONCLUSIONS

This work proposes a framework called E-txt2db that aims at combining both the advantages of Information Extraction frameworks and the advantages of declarative languages for Information Extraction for the specification and execution of Information Extraction programs. Analogously to the Information Extraction frameworks, E-txt2db offers a wide variety of pre-implemented rule-based and machine learning techniques that can be used to specify Information Extraction programs. This turns the specification and maintenance of the programs easy because the user does not need to develop the Information Extraction algorithms from scratch. Code reusability is possible since extraction models are created and then stored for further use. Similarly to the declarative languages for Information Extraction, the resulting Information Extraction programs are amenable to optimization. Finally, due to the high level of abstraction offered by the E-txt2db operators, it is very easy to learn how to use.

This paper focused only on the E-txt2db framework classification operator. There is considerable amount of work left to be done until E-txt2db reaches its full potential. However, the evaluation results presented in Section 5, gives optimistic perspectives about the advantages of this framework. In the future, we expect that the work developed for the creation of the other Information Extraction operators and for the automatic optimization of the Information Extraction process turns E-txt2db into a very comprehensive solution for the specification and execution of Information Extraction programs.

7. REFERENCES


