Domain Model Generation With the Help of Supervised Machine Learning

Viliam Simko, Petr Kroha, Petr Hnetynka

Abstract: This technical report aims at describing how to generate a domain model from natural language specification using supervised machine learning. The elicitation process consists of several steps (classification tasks) each contributing a piece of information to the generated domain model. We explain the design, training application and evaluation of the relevant classification models.

This work was partially supported by the Grant Agency of the Czech Republic project P103/11/1489 and by the Charles University institutional funding SVV-2012-265312.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>2</td>
</tr>
<tr>
<td>2. Commonly used approaches</td>
<td>2</td>
</tr>
<tr>
<td>2.1 Named Entity Recognition</td>
<td>3</td>
</tr>
<tr>
<td>3. Classification</td>
<td>3</td>
</tr>
<tr>
<td>4. Overview of the method</td>
<td>3</td>
</tr>
<tr>
<td>4.1 Feature Selection</td>
<td>4</td>
</tr>
<tr>
<td>4.1.1 Experimental Samples</td>
<td>5</td>
</tr>
<tr>
<td>4.2 Training</td>
<td>6</td>
</tr>
<tr>
<td>4.3 Domain Model Elicitation</td>
<td>7</td>
</tr>
<tr>
<td>5. List of Classification Tasks</td>
<td>7</td>
</tr>
<tr>
<td>5.1 Task: Identifying words forming a domain entity</td>
<td>8</td>
</tr>
<tr>
<td>5.2 Task: Identifying multi-word entities</td>
<td>8</td>
</tr>
<tr>
<td>5.3 Task: Identifying the aggregation relation</td>
<td>8</td>
</tr>
<tr>
<td>5.4 Task: Direction of the aggregation relation</td>
<td>8</td>
</tr>
<tr>
<td>5.5 Task: Identification of attributes</td>
<td>8</td>
</tr>
<tr>
<td>5.6 Task: Merging duplicate entities</td>
<td>8</td>
</tr>
<tr>
<td>6. Evaluating the classification performance</td>
<td>8</td>
</tr>
<tr>
<td>6.1 Evaluation metrics</td>
<td>9</td>
</tr>
<tr>
<td>6.2 The Experiment</td>
<td>9</td>
</tr>
<tr>
<td>6.2.1 Measuring a single model</td>
<td>12</td>
</tr>
<tr>
<td>7. Experimental Data</td>
<td>12</td>
</tr>
<tr>
<td>7.1 Text of the specification</td>
<td>12</td>
</tr>
<tr>
<td>7.2 Summary of domain model entities</td>
<td>13</td>
</tr>
<tr>
<td>7.3 Manually created domain model</td>
<td>13</td>
</tr>
<tr>
<td>8. Conclusions and Future work</td>
<td>14</td>
</tr>
</tbody>
</table>
1 Introduction

At the very beginning of software development, there should be a document containing a detailed textual description of requirements. The software analyst writes it in cooperation with other stakeholders of the project, i.e. with the customer, the end users, and the domain experts. A natural language is mainly used because this is the only form of requirements specification which the user really understands. Even though the text may be accompanied by other graphical elements, such as diagrams or images, they are also described in the text. Thus, we can say that the text carries the information relevant for the development of a project.

Unfortunately, the requirements specification in a natural language is informal, imprecise and vague. Every sentence, every word could be understood in a different way by the user and by the analyst. It is very well known that the most serious mistakes and failures in software products have their origin in analyst’s bad understanding of user’s needs. The analysts describe how they understand the problem but they do not really understand the problem deep enough at this time, i.e. at the very beginning of the project. The analyst’s understanding cannot be easily confronted with the user’s understanding because these groups of specialists do not have a common formalism how to describe systems.

To solve the semantic gap between stakeholders, the domain model became a mandatory part of the software engineering process. The elements of the domain model serve as a common vocabulary in the communication among technical and non-technical stakeholders throughout all project phases. This helps them come to an agreement on the meaning of important concepts. In [9] (p. 23), the domain model is defined as a live, collaborative artifact which is refined and updated throughout the project, so that it always reflects the current understanding of the problem space.

From the modeling point of view, the domain model consists of classes, associations (aggregation, inheritance) and other elements commonly found in UML class-diagrams. The main difference is that a domain model focuses on entities in the problem-space and should always be independent of any particular implementation technology. On the other hand, UML class-diagrams are tight to the solution space and may well entail caching- or UI-related classes.

A domain model is usually not constructed en bloc, yet it undergoes refinement starting from the first prototype elicited from text. By reading the specification documents, the analyst tries to identify important concepts that may be included to the domain model. The analyst has to read the text multiple times while each time focusing on a certain aspect of the domain model – naming of entities, aggregation, dependency etc. …

Naturally, due to the vagueness of a textual specification, it is not uncommon that two analysts may derive significantly different models from the same input texts. However, this is not an issue since the prototype domain model is further refined as the project development progresses. Even the original specifications may be updated provided that inconsistencies are identified during the elicitation process.

Apart from the creative nature of the refinement phase, which involves humans, the initial prototype can be derived automatically to some extent. The motivation for the automation is to save the initial effort of the analyst ranging from a couple of hours to a couple of days. Instead of starting from a plain text, the analyst can starts from a model which points to specific locations in the source text.

In this paper, our goal is to derive such prototype domain model automatically. We formulate the problem as a supervised machine-learning classification task. Our approach combines (i) linguistic features gathered from text by existing Natural Language Processing (NLP) tools together with (ii) features related to software engineering such as domain model entities and glossary entries.

2 Commonly used approaches

Grammatical inspection is an approach commonly used by practitioners and extraction tools to get a quick start. The text is scanned for nouns, adjectives, verbs or other linguistic units. For example, the book [9] suggests that 80% of domain classes can be discovered in the initial domain modeling session.

The analyst should:

1. Create a list of candidates for domain entities by scanning the text for nouns and noun phrases as a representatives of classes/objects. (The simplest approach would be to consider a simple equivalence noun=class)

It is advised that the analyst does not spend too much time in this phase and that the rest of the objects are identified during the robustness analysis, i.e. when analyzing the textual use-cases.
2. Identify named entities such as places, addresses, names of organizations, years, etc. This is important since these entities should usually be excluded from the domain model.

3. Remove obviously duplicate terms, such as "User Account" = "Customer Account", or "Book Review" = "Review Comment"

4. Remove generic terms (e.g. "Internet") and UI-related terms. (e.g. "Password")

2.1 Named Entity Recognition

In computational linguistics, the term Named Entity Recognition (NER) represents a variety of tasks related to extracting relevant entities and their relations (aka Relation Extraction) from unstructured or semi-structured text.

Historically, the task was focused on the identification of real-world entities such as places, organizations or person names, hence the term "Named". Our method, discussed in this paper, also belongs to the NER/RE family. In particular, we use the supervised machine learning approach. Rather than identifying places or names, we identify entities and their relations forming a potential domain model. Since this is a statistical approach it also needs to be properly evaluated.

3 Classification

As usual for the information extraction techniques, their performance is evaluated by computing the precision, recall and F-measure.

Inspired by the state-of-the-art NLP tools, such as the Stanford part-of-speech tagger [10] or other approaches, such as [11], [8], [2], we utilized Maximum Entropy (MaxEnt) models [6] for the classification. However, the ideas presented in our contribution should also work with other classifiers such as Bayes classifiers [5] (p. 253), Support Vector Machines (SVMs) [3] (p. 319), Conditional Random Fields (CRFs) [4], Perceptrons [7], each having their specific strengths and weaknesses.

We opted for the MaxEnt approach because it is well suited for classification tasks comprising vast amount of dependent binary features. This has been demonstrated by a number of MaxEnt implementations in field of NLP, e.g. [2]. To explain MaxEnt approach, the following citation from [6] (p. 589) is usually mentioned in literature:

Maximum entropy modeling is a framework for integrating information from many heterogeneous information sources for classification. The data for a classification problem is described as a (potentially large) number of features. These features can be quite complex and allow the experimenter to make use of prior knowledge about what types of informations are expected to be important for classification. Each feature corresponds to a constraint on the model. We then compute the maximum entropy model, the model with the maximum entropy of all the models that satisfy the constraints. This term may seem perverse, since we have spent most of the book trying to minimize the (cross) entropy of models, but the idea is that we do not want to go beyond the data. If we chose a model with less entropy, we would add ‘information’ constraints to the model that are not justified by the empirical evidence available to us. Choosing the maximum entropy model is motivated by the desire to preserve as much uncertainty as possible.

As an implementation, we use the OpenNLP framework [3] (chapter "Machine Learning / Maximum Entropy") which allows us to integrate Maximum Entropy classifiers to Java applications.

As we show later in the text, we use multiple dependent classification tasks. Unknown values to be classified are connected in a Markov chain rather than being conditionally independent of each other. This approach is called Maximum Entropy Markov Models (MEMM). It should also be pointed out that with MEMMs we can easily make use of lexicalization which means that the features are constructed from an unrestricted set of all words rather than from a restricted set of classes. The unlexicalized approach is time- and space-efficient, but often with lower prediction performance.

4 Overview of the method

Figure[1] provides an overview of our method divided into 3 phases – (1) Feature Selection, (2) Training, (3) Domain Model Elicitation (classification).
4.1 Feature Selection

At this point, it is important to clarify what we mean under the term "feature". In general, feature \( \phi(c,d) \) is an elementary piece of evidence that links some observed data \( d \in D \) with a category \( c \in C \). It is a function with a bounded real value.

\[
\phi : C \times D \mapsto \mathbb{R}
\]  

(1)

In practice, especially in the field of NLP, the features are of a particular form – an indicator function (boolean matching function) where \( P \) is a matching predicate against the data and \( c_j \) is some category.

\[
\phi(c,d) \equiv [P(d) \land c = c_j]
\]  

(2)

Finally, it is useful to restrict the predicate \( P \) to a conjunctive form:

\[
\phi(c,d) \equiv [f_1 \land f_2 \land \ldots \land f_n \land c = c_j]
\]  

(3)

An example could be the following formula which indicates that the word represents a domain entity if its POS-tag is "NN" and the lemma-form is "user".

\[
\phi(c,d) \equiv [\text{pos} = \text{NN} \land \text{lem} = \text{user} \land c = \text{DOMENT}]
\]  

(4)

When we train a classifier, we need to encode such formulas as training samples for the classifier. For example, the formula (4) would be encoded as a sample:

\[
\text{pos=NN lemma=use DOMENT}
\]  

(5)

In machine learning, the samples are also called feature-vectors that encode relevant information about a given data-point \( x \). In our case, this would be a tuple composed of \( n \) values representing a single data-point \( x \).

\[
\tilde{y} = (f_1(x), \ldots, f_n(x))
\]  

(6)
4.1 Feature Selection

The media administration contains an entry for each medium in the library.

Figure 2: Links between manually annotated text and manually created domain model

"The library system contains a user administration and a media administration"

Figure 3: Stanford typed dependencies representation of a sentence.

We use the term “feature extractor” for an algorithm that computes the value of a single \( f(x) \) in the context \( x \). Thus, in contrast to the Formulas (1) and (2), we will use the term “feature” for the individual \( f_1, \ldots, f_n \) rather than the whole \( \phi \). (e.g. POS-tag of a word would be an \( f_{pos} \) feature for us).

To design a classifier\(^3\) we need to specify which features will contribute their value to the feature-vector. Thus we need to fix a feature-set \( \phi^\lambda = \{ f_1, \ldots, f_n \} \) for the classifier \( \lambda \).

The goal of the Feature Selection phase is to identify \( \phi^\lambda_{\text{best}} \) among alternative feature-sets \( \phi_1^\lambda, \ldots, \phi_n^\lambda \) for each \( \lambda \) in terms of their prediction-performance. This is a hill-climbing process guided by our intuition when we measured the prediction-performance of candidate feature-sets:

\[
\phi^\lambda_{\text{best}} = \arg \max_x \text{Perf}(\phi^\lambda_x) \tag{7}
\]

Later, in the section\(^4\) we show how \( \text{Perf} \) is computed.

4.1.1 Experimental Samples

To experiment with the performance of classifiers, we need experimental samples. Therefore, we manually prepared annotated texts and domain models\(^4\) according to the specification of a Library System. Our annotations represent links between a sequence of words and some element from the domain model as depicted in Figure 2. In this example, the words "media administration" are linked to the MediaAdministration domain entity. Annotations are encoded in the text in the following form:

The [media administration|MediaAdministration] contains an entry for each [medium|Medium] in the [library|Library].

When parsing the annotated text, each sentence is enriched with automatically generated data from the Stanford parser applied on each sentence found in the input text (Figure 3). All the information is stored together in a single graph we call the "specification model" \( M \). (see Figure 4 showing its metamodel)

\(^2\)As we will see later, the feature extractors can be arbitrarily complex. (In particular \( f_{pos} \) has a fairly simple extractor.)
\(^3\)Since each classification task requires a single classifier we use the notation \( \lambda \) interchangeably for both – task and classifier.
\(^4\)We use EMF.ecore model for encoding domain models
For the given classification task $\lambda$, we construct the sample-set $S$ as a matrix which contains features $f_1, \ldots, f_n$ of $k$ data-points $x_1, \ldots, x_k$:

$$
S = \begin{bmatrix}
\vec{y}_1 \\
\vdots \\
\vec{y}_k
\end{bmatrix} = 
\begin{bmatrix}
f_1(x_1) & \ldots & f_n(x_1) \\
\vdots & \ddots & \vdots \\
f_1(x_k) & \ldots & f_n(x_k)
\end{bmatrix}
$$

(8)

by applying relevant feature extractors on the graph nodes of the "specification model". Then the performance of each candidate feature-set $\phi^\lambda$ is evaluated on samples from $S/\phi^\lambda$ (restricted sample-set with only those columns from $\phi^\lambda$). Further discussion on the evaluation can be found in a separate section 6.

The results of the "Feature Selection" phase are: (i) feature extractors (ii) ranking of feature-sets for each classifier (iii) best feature-set for each classifier.

4.2 Training

While looking for the best feature-set $\phi^\lambda_{best}$ in the previous phase, we already went through the process of training and evaluating a classifier $\lambda$. (Formula (7)). For this purpose, we used manually annotated experimental data. Knowing $\phi^\lambda_{best}$, we can now automatically retrain the same classifier with new input samples. We assume that such data come from already finished projects (denoted as "training projects" in Figure 1). Specification model $M$ is therefore an aggregation of training projects $P_1, P_2, \ldots$

$$
M = \bigsqcup_i P_i
$$

(9)

According to the Formula (8), each column in the matrix $S$ represents a single feature, whereas rows represent individual samples. By convention, the last column is regarded as the "outcome" while the rest is the "context". The classifier learns which context leads to which outcome. A typical input required by the OpenNLP MaxEnt API conforms to the following format:

```
pos=NN  indep=agent  true
pos=VBZ indep=auxpass false
...```

---

**Figure 4:** Meta-model of the graph containing input data for feature extraction.
4.3 Domain Model Elicitation

Only in this phase, the real users are involved. The process starts with an empty specification model \( M_0 = \emptyset \). Specification documents \( D_1, \ldots, D_m \) written in natural language describing the intended system are gathered, parsed and then added to \( M_0 \):

\[
M_0 = \bigcup_{i=1}^{m} D_i
\]

(10)

Now the trained classifiers \( \lambda_1, \ldots, \lambda_n \) from the Training phase can be used. Each classifier in the chain adds new information based on the data generated by the previous classifier in the chain.

\[
M_i = \lambda_i(M_{i-1}); i = 1 \ldots n
\]

(11)

After applying all the classifiers, the model \( M_n \) (containing prototype domain model) can be further refined by human analysts iteratively as the project progresses. Finally, when the project is finished, the refined version of all artifacts (text, UML models, code) representing the final product, can be included to the set of training projects. To improve the performance of the classifiers in a next project, the classification models can be retrained using the new set of training projects.

5 List of Classification Tasks

In this section, we explain details of all classification tasks involved in the elicitation process. The order of sections reflects the actual sequence in which the classifiers are applied. Since the tasks form a Markov chain, their order is important. It is usual for the NER/RE methods to employ a two-phase approach – (i) decide which entities are related using a faster classifier, (ii) then for related entities classify the relation. Phases (i) and (ii) usually use distinct feature sets appropriate for their task.
5.1 Task: Identifying words forming a domain entity

The first task aims at identifying words that may represent domain entities. For every positively identified word we add an instance of a new domain entity to the specification model $M$. Some multi-word entities will be represented as multiple entities in $M$. For example words "User Account" will be represented as "User" and "Account" entities instead of a single one. Due to the fact that this is the first element of the Markov chain, we start by an empty domain model in $M$. Therefore we can only make use the "linguistic" features extracted from the text.

5.2 Task: Identifying multi-word entities

To further improve the quality of the generated domain model, the next phase’s goal is to merge selected candidate entities into a single entity. In this case the classifier predicts whether a pair of entities should be merged. We don’t want to classify all the $n^2$ pairs of entities in this task. We only check entities appearing together in a single sentence. Another improvement could be simply use the order of entities within a sentence and only check nearby entities.

5.3 Task: Identifying the aggregation relation

At this point, we should have a list of multi-word domain entity candidates. The next task is to derive aggregation relation among them. Also in this task, we group the entities by the sentence in which they appear. We use the classifier to predict which pairs of entities may be involved in the aggregation relation.

5.4 Task: Direction of the aggregation relation

A separate task is to derive the direction of the aggregation, i.e. which entity is the container and which is the containment.

5.5 Task: Identification of attributes

Now, after the aggregation relation has been derived, a human analyst would decide which of the containment entities are just attributes rather than separate classes. We run a classifier to do the same operation automatically.

5.6 Task: Merging duplicate entities

Last classification task is to identify duplicate entities by adding a binary "same as" relation. After that, we can merge those entities by producing the final version of our prototype domain model.

6 Evaluating the classification performance

Our method is a collection of supervised machine learning tasks. It is not possible to use a classifier without a proper evaluation of its performance. A classifier can perform badly even though its design may look reasonable. Moreover, we simply cannot be sure which combination of features performs better. Hence the Feature Selection phase depicted in Figure 1 and Formula (7). The evaluation should be automated and well understood.

In general, a supervised machine learning approach requires a representative corpus containing labeled relations between entities (usually labeled by hand) which is divided into (i) training set, (ii) development test set and (iii) test set. To achieve good performance, the classifier should be trained on a training set which sufficiently captures the variability of data expressed as "features". During the development process, we are trying to select the right set of features. For this task we use the development test set to assess the performance of the selected features. Finally, to avoid overfitting, the classifier’s performance is evaluated against the unseen test set.
6.1 Evaluation metrics

The standard approach is to compute precision, recall and balanced $F_1$ measure by constructing a contingency table for each classification problem as follows:

<table>
<thead>
<tr>
<th></th>
<th>correct</th>
<th>wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>selected</td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td>not selected</td>
<td>FN</td>
<td>TN</td>
</tr>
</tbody>
</table>

The table contains counts of true positives (TP) i.e. relations correctly extracted from the text, false positives (FP) i.e. relations incorrectly extracted from the text, false negatives (FN) i.e. relations incorrectly not extracted from the text, true negatives (TN) i.e. relations correctly not extracted from the text.

The evaluation metrics are then defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{12}$$

$$Recall = \frac{TP}{TP + FN} \tag{13}$$

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{14}$$

6.2 The Experiment

We will now focus on a single classifier $\lambda$ and show the design for an experiment which evaluates its performance. The experiment is depicted in Figure 6.

First, we choose all combinations of features (feature-sets) that we want to measure. Result of the experiment will determine the desired best feature-set $\phi_{\text{best}}^\lambda$ for a given classifier $\lambda$.

Another parameter required by the experiment is sample-set $S$ generated from the specification model $M$ as shown in Figure 5. Remember that $S$ is a matrix where columns represent the features and rows represent the samples. (see Formula (8)) Thus, $S$ has to contain columns for all features that we want to measure.

The remaining two parameters are split ratio and $k_{\text{dev}}$. Split ratio determines how the samples (rows of $S$) will be randomly divided into development test set and test set. By default we use 90% for development and 10% for testing. The parameter $k_{\text{dev}}$ drives the k-fold cross validation ((Figure 9) within the development phase (Figure 7). Finally, we take the $\phi_{\text{best}}^\lambda$ and run the evaluation phase (Figure 8) on the previously unseen test data.
6.2 The Experiment

Iterate over feature sets

Development Phase

Filtering of features

- k-fold cross-validation
- filtered samples
- samples

Figure 7: Development phase – selecting the best set of feature types.

Evaluation Phase

MaxEnt Training

- filtered samples
- samples

MaxEnt Model Evaluation

- filtered samples
- test samples
- train samples

Figure 8: Evaluation phase – using the previously unseen test data.
Figure 9: Using k-fold cross-validation for a given sample set, computing $F_1 +$ confidence.
6.2.1 Measuring a single model

The Figure 10 shows how a single model (here a trained MaxEnt model) is measured. The samples and the model are used by the classifier which tries to predict the outcome for each sample. Its guesses are compared to the gold values from the same samples and precision, recall and $F_1$ measure is computed.

7 Experimental Data

For the experiment we chose a specification describing a simple Library System. At the beginning we got a textual specification as a document written in English. From the text, we manually derived a domain model shown below.

7.1 Text of the specification

Here is an excerpt from the text of the Library System specification. It contains introductory chapters following by use-cases and non-functional requirements. The text is annotated by links to the domain model in the form: [original text][domain entity name]

1. Objective
   Some [Media] and [user] of a [library] are managed by the [library system].

2. Operational Area
   The [library system] is operated by the [library staff (librarian)] and [library user (user)] through [terminals].

3. Product overview
   The [library system] contains a [user administration] and a [media administration].

3.1 User administration
   The [user administration] contains a [user account] for each [user] which contains [all user data].
   A [librarian] is able to create a new [user account], to edit and to delete an existing [user account].
   A [user] is able to register at the [system] with his [user number], to manage his [user account],
   and to extend the [media’s rental period].
   A [password] is not necessary because the [user number] on the [identification card] is read with a [bar code scanner].

3.2 Media administration
   The [media administration] contains an entry for each [medium] in the [library].
   Several [instances of each medium] may be available, and they may have different [locations].
   A [librarian] is able to add a new [media instance] to the [media administration], change the [status] of an [instance]
   and [remove an instance] from the [media administration].
   A [user] is able to [search for a [medium]] by specifying one or more features of the [medium].
   [User] can choose and [reserve] a found [instance], and then it is added to his [account] and will only be [deleted] when it is returned.

4. Product function
   A [user account] must be available for each [user].
   A [librarian] has access to all [user account] of all [users].
   A [user] has only access to his own [user account] by using his [user number].
A [user] is able to:
- [search a medium] ([MediaAdministration.searchMedium])
- [reserve one or more instances] ([Instance.reserveInstance])
- [borrow one or more instances] ([Instance.borrowInstance])
- [extend the rental period] ([Instance.extendPeriod])
- [return one or more instances] ([Instance.returnInstance])

A [librarian] is able to:
- create, edit and delete a [user account] ([UserAccount])
- create, edit and delete a [medium] ([Medium])

4.1. Use Case: [Search medium] ([MediaAdministration.searchMedium])

The [user] specifies one or more [titles] and obtains a list of [media] which contain the [titles] and attributes.

Procedure:
1. The [user] specifies one or more [search criteria]
2. It is searched for matching [media] in the [MediaAdministration]
3. The [user] receives a list of [media] that match the search [criteria]

Requirement SYS-1.1:
Type: Security
Text: [User name] and [telephone number] are mandatory fields.

7.2 Summary of domain model entities

Here is a list of all domain entities (classes, attributes, references and methods) that are mentioned in the annotated text.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarCodeScanner</td>
<td>1x</td>
</tr>
<tr>
<td>BarCodeScanner.scanUserName</td>
<td>1x</td>
</tr>
<tr>
<td>IdentificationCard</td>
<td>2x</td>
</tr>
<tr>
<td>IdentificationCard.userNumber</td>
<td>2x</td>
</tr>
<tr>
<td>Instance</td>
<td>6x</td>
</tr>
<tr>
<td>Instance.borrowInstance</td>
<td>3x</td>
</tr>
<tr>
<td>Instance.deleteInstance</td>
<td>1x</td>
</tr>
<tr>
<td>Instance.extendPeriod</td>
<td>1x</td>
</tr>
<tr>
<td>Instance.location</td>
<td>1x</td>
</tr>
<tr>
<td>Instance.removeInstance</td>
<td>1x</td>
</tr>
<tr>
<td>Instance.reserveInstance</td>
<td>1x</td>
</tr>
<tr>
<td>Instance.returnInstance</td>
<td>1x</td>
</tr>
<tr>
<td>Instance.status</td>
<td>1x</td>
</tr>
<tr>
<td>Instance.extendPeriod</td>
<td>1x</td>
</tr>
<tr>
<td>Librarian</td>
<td>5x</td>
</tr>
<tr>
<td>Library</td>
<td>2x</td>
</tr>
<tr>
<td>MediaAdministration</td>
<td>6x</td>
</tr>
</tbody>
</table>

7.3 Manually created domain model

The domain model contains more information than the specification actually contained. This is because some relations were considered implicitly presumed by its authors.
8 Conclusions and Future work

We have described a method for deriving a domain model from textual specification using a supervised machine learning approach. We discussed main phases of the approach – feature selection, training, elicitation. Then we got into details of evaluating a performance of classifiers that is needed for the feature selection phase. We also provided an example of a specification (Library System) composed of the annotated text and a manually derived model serving an input for our experiment.

The next step is to finish our Java-based framework so that it can run experiments automatically to identify best feature sets and trained models for a given classification task. Finally, we need to chain all tasks to produce the automatically generated domain model which is the ultimate goal of our method.

References


