Tree kernel and Feature Vector Methods for Formal Semantic Requests Classification

François Bouchet and Jean-Paul Sansonnet

LIMSI-CNRS
Université Paris-Sud XI
BP 133, 91403 Orsay Cedex, France
bouchet@limsi.fr

Abstract. In this paper, we're interested in the classification of natural language requests converted into a formal representation, where the classes represent the conversational activity of those requests. This study is based on a corpus of requests collected using an assisting conversational agent, in which we identified four different classes (control, chat, direct and indirect assistance requests). The objective would be to take over from the rule-based system when it fails. First representing formal requests as a tree, we show it is possible to adapt tree kernel methods to our problematic. A second approach consisting in ignoring the request structure to focus on its components (*i.e.* considering it as a feature vector) gives better results a priori. We finally consider combining several of the previous classifiers, thus reaching a performance rate of 76.1%, which could be enough for using it as a complementary system.

1 Introduction

Providing a relevant and motivating assistance to novice users of computer applications has been a key issue for software designers for a long time, and it remains an open and topical issue as the population of new computer users keeps increasing. It has been shown that natural language (NL) modality is an ideal way to capture users' needs as this is a modality spontaneously used to express their frustration (similarly to the 'thinking aloud effect' [1]) and also because using mutimodality allows a clear cognitive separation between the application and the help system (with graphical [2] or dialogical [3] modalities). In this context, we have chosen to develop an assistance system relying on the use of Embodied Conversational Agents (ECA) [4] to take moreover advantage of the increased agreeability and believability a human-like presence brings to the system (cf. the 'Persona Effect' described in [5]).

In order to answer the requests issued by the user, regardless of the ability to ultimately build a correct answer (either from text mining a written help database like in TREC [6] or through an analysis of a dynamic model of the application [7]), the agent needs to understand *the purpose* of the request. Typically, this is achieved in two steps: first, through the transformation of an input request given as a string into a structured grammaticaly-based representation

like in HPSG [8], and in a second time, with a semantic analysis in order to extract higher level structures (*e.g.* an order to modify an element of the application, a question about the possibility to do some action, *etc*). The automation of this process is a challenging task that we have been addressing so far mainly through a set of association rules between lemmas or grammatical structures on one side, and semantic structures on the other. Nonetheless, as it has been shown for instance in the case of machine translation [9], machine learning approaches are often very complementary to rule-based ones, which leads us to investigate in this paper the possibility to apply automatic classification methods that would enable the agent to be able to at least identify the conversational activity of a request when the rule-based methods fail. Being able to provide even an only very partially relevant answer is always better than no answer at all, as that kind of situations immediately leads to a very negative 'Clippy Effect' [10], where the disappointment and the feeling to have been misleaded equal user's original high expectations in the agent generated by the embodiment. For instance, when asked "Click the red button?", its correct identification as a control request could lead the agent to answer something like "Please explain the action you're expecting me to do" intead of a very frustrating "I don't understand".

If automatic document classification has been largely studied, it is less the case for simple requests which often consists in a single utterance. Most attempts on single utterances focused on classification for question answering (QA) like in [11] where the authors particularly use a tree kernel approach (similarly to what we'll see in section 3) but which is based on the grammatical representation of the request (instead of a semantic one in our case). [12] also tries to take into account the semantic content of the requests but focuses only on "WH-" questions (what, when, why, *etc*) that expect a simple fact as answer, whereas the requests we have to deal with have much more variety (cf. section 2.2).

After an introduction to the considered conversational activities and to the formalism used to represent the requests, we first regard the formal semantic requests as trees and show the possibility to apply a tree kernel approach. In a second time, we forget the structure of requests and see them as feature vectors to which we apply classical classification methods (KNN, multilayer perceptron, decision tree and naive Bayes classifier). We finally try to exploit the complementarity of the approaches through a combination of the previous classifiers.

2 Resources: requests, classes and formal representation

2.1 Corpus of user requests

This study relies on real novice users requests that have been collected for two years and gathered into a corpus of 11,000 requests, in order to get an overview of the kind of requests an assisting agent in situ would have to face. However, since the whole corpus hasn't been manually annotated in terms of conversational activities (cf. section 2.2), the work presented in this paper has been achieved on a subset of 1,070 sentences. This subset has been shown to be representative of the full corpus since it is a collection of two independent subsets of randomly

chosen sentences without repetition, and the comparison of those two subsets according to various parameters hasn't revealed any significant difference between them [13].

2.2 Classes: four conversational activities

In a preliminary study of the aforementioned subset of the corpus, we have shown that it can be divided into three main categories of conversational activities:

1. **Control** requests (15%), in which the user expects the agent to act as a mediator interacting with the application.
2. **Assistance** requests (45%), the ones we're the most interested in and among which we can distinguish:
   - **Direct assistance** requests (36%), where the need for assistance is explicit at the linguistic level – *e.g.* "how can I register?"
   - **Indirect assistance** requests (9%), requiring a pragmatic analysis since a low level analysis could lead to another reaction than assistance – *e.g.* "pity, I can't change the color of the table".
3. **Chat** requests (40%), made of feedbacks and requests where the agent replaces the assisted application as the user's center of attention.

2.3 The DAFT formal representation language

As a preparation to the work presented here, we have defined a formal request language, called the DAFT language, in order to capture the main semantics of the assistance requests from the corpus. We have also created a natural language processing tool, called DIG (for DAFT Interpretation Generator) which can automatically translate a user request into a formal request represented in DAFT. Note that the 1,070 formal requests used for classifications done in this article are issued from DIG, and hence shouldn't be considered as error-proof as it would have been the case if we had been working on their manual transcription. However, since our objective is ultimately to use the classification in real assistance cases, we consider it would be useless to have a system efficient with a correct representation of the request when we are actually not able to automatically generate it. This means that shall be DIG improved in the future, the performances obtained with the classification methods below might change too.

Request basic elements: entities The basic element for the structure of the formal language is the *entity*. An entity E is used to reify any actual semantic notion. It is formally defined and represented as:

$$\begin{aligned} E &= t_s : s\,[g, t_1 : a_1 = v_1, \ldots, t_n : a_n = v_n] \\ &= t_s : s\left[g, \bigcup_{i=1}^{n} t_i : a_i = v_i\right] \end{aligned}$$

where: $t_s \in \mathbb{T}_E$ is the *type* of the entity E,
$s \in \mathbb{S}$ is the *identifier* of the entity E,
g is a string representing the *gloss* defining, in natural language, the meaning of the entity, possibly through examples (like for synsets in WordNet [14]),
$t_i \in \mathbb{T}$ is the i$^{\text{th}}$ attribute type of the entity E,
a_i is the i^{th} attribute identifier (a symbol),
$v_i \in \delta_{t_i}$ is the value of the i$^{\text{th}}$ attribute, within the domain of validity δ_{t_i} explicitly associated to t_i.

We call $\mathbb{E}$ the (infinite) set of entities.
To refer to the identifier s of an entity E, we'll write it as $id[E]$. Same goes for the type written as $type[E]$ and the gloss written as $gloss[E]$. We also refer to the number of triples (t_i, a_i, v_i) of an entity E as its cardinal written as $|E|$.

Example 1: an entity E_{ex1} corresponding to the verb "to click":

E_{ex1} = act : Click[
    ‘‘A person is clicking on an object in a certain manner’’,
    person : clicker = ∅,
    object : clicked = ∅,
    manner : manner = ∅
]

where $id\,[E_{ex1}] = \texttt{Click}$, $type[E_{ex1}] = \{\texttt{act}\}$ and the three attributes are 'clicker', 'clicked' and 'manner', so we have $|E_{ex1}| = 3$.

Identifier: The identifier, used as the head of an entity, is what defines other parameters except values, which can be expressed as:

$$\begin{aligned}\forall (E_1, E_2) \in \mathbb{E}^2, \mathrm{id}[E_1] = \mathrm{id}[E_2] \Rightarrow (&\mathrm{gloss}[E_1] = \mathrm{gloss}[E_2] \\ &\wedge\ \mathrm{type}[E_1] = \mathrm{type}[E_2] \wedge |E_1| = |E_2| \\ &\wedge\ \forall i \in [\![1, |E_1|]\!], t_i[E_1] = t_i[E_2] \\ &\wedge\ \forall i \in [\![1, |E_1|]\!], a_i[E_1] = a_i[E_2])\end{aligned}$$

So if E_1 and E_2 have the same identifier, only their values can be different.

Schemes: The association between on one side an identifier, and on the other side a type, a gloss, a set of attributes identifiers and their associated types, is done through the definition of a scheme for each identifier. The scheme of an entity could then be defined as the invariant elements between two entities with the same identifier. There are 237 different schemes[1], so $|\mathbb{S}| = 237$.

[1] Depending on their semantic role (*i.e.* the meaning they have over their attributes), identifiers belong to one of four different classes (namely "Modalities", "Actions", "References" or "Properties") but this notion is not used for the classification and thus will not be discussed it in this paper – see [15] for more details. It explains however the different font cases used for identifiers' names.

Types: Types are a restriction over the values that can be associated to the corresponding attribute. An attribute type t_i can be:

1. a symbol in $\mathbb{T}_E$ representing a subset of the 237 existing schemes,
2. a value chosen in a finite set written as $[\![val_1, val_2, \ldots, val_n]\!]$,
3. a value chosen in an interval written as $[bound_{inf}, bound_{sup}]$,
4. a classical type: String or Boolean.

Attributes: Each attribute identifier a_i has a unique name in the context of the entity E to which it belongs:

$$\forall E \in \mathbb{E}, \forall i, j \in [\![1, |E|]\!], i \neq j \Rightarrow a_i \neq a_j$$

Conversely, that means two entities with a different identifier can have some or all attributes identifiers in common:

$$\forall((E_1, E_2) \in \mathbb{E}^2, (\forall i \in \mathbb{N}, a_i\,[E_1] = a_i\,[E_2])) \not\Rightarrow id\,[E_1] = id\,[E_2]$$

Moreover, there is no bijection between an attribute identifier and a type:

$$\forall\left((E_1, E_2) \in \mathbb{E}^2, \mathrm{id}\,[E_1] \neq \mathrm{id}\,[E_2]\right), (i, j) \in \mathbb{N}^2, a_i[E_1] = a_j[E_2] \not\Rightarrow t_i[E_1] = t_j[E_2]$$

The attributes associated to the scheme of an entity have been defined accordingly to what has been found in the Daft corpus. That means attributes of an entity are the statistically most frequent parameters associated to the notion described by the gloss, but we're clearly not pretending to be exhaustive.

Values: A value v_i can be:

1. an entity, if its type t_i is a symbol,
2. a terminal value, if its type t_i is anything but a symbol,
3. empty (written as $\emptyset$), in any case, when no value is associated to the attribute a_i. By default, values are empty so an entity doesn't need a value associated to each attribute in order to be valid (cf. example 1).

Requests as an association of entities A user request is represented as a tree of entities. Normally, if the request is correctly represented, the set should have a size of 1 (unique root), which is not the case practically as we'll see on figure 2.

Example 2: representation of the request "Click the red button" (using the entity Click[...] introduced in example 1)

R_{ex2} =
```
act : Click[
 person : clicker = ∅,
 object : clicked = object[
     ppt* : property = {
         type : type[{[window, button, field...]} : val = button],
```

```
        quantity : quantity[{[0, 1, 1+, N, Some...]} : val = 1],
        color : color[{[black, white, red, blue...]} : val = red]
    },
    act : doing = ∅,
    act : subject-of = ∅
],
manner : manner = ∅
]
```

The 'property' attribute type, ppt, is suffixed with a '*' to mean that the attribute can be associated to a list of entities having 'ppt' as their type or in the ancestors of their type. Otherwise by default only a single value can be associated to the attribute.
For readability reasons, DIG can produce a representation of the nested lists as nested boxes. A box represents an entity, and hence is made of an identifier (in the top left-hand corner) and a list of triples (t_i, a_i, v_i) shown as $[t_i]a_i = v_i$. If the value is another entity, it appears as another box.
A graphical version of example 2 can be seen on figure 1 (attributes with an associated empty value are not represented).

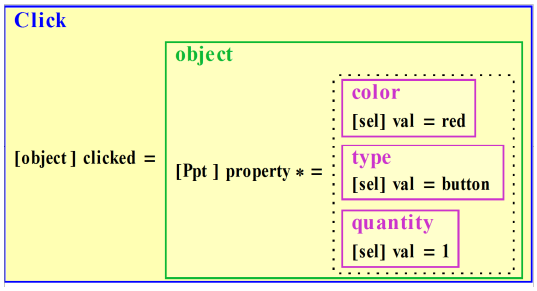

Fig. 1. DIG output (nested boxes) representation of the request "Click the red button"

3 Structure-based classification: requests as trees

Now that we have introduced the formalism used to represent the semantics of the requests and the four classes into which we want to classify them, we can focus on the different possible classification methodologies, starting by considering the request as a tree.

3.1 Methodology: tree kernel approach

General principles of tree kernel methods The idea behind kernel based approaches is to be able to use a linear classification method to solve a non-linear

problem. It is based on Mercer's theorem stating that any continuous, symmetric, positive semi-definite kernel function can be expressed as a dot product in a high dimension space. So knowing the definition of a kernel function, any algorithm only relying on the dot product of two vectors can be transformed into a non-linear version by replacing this dot product by the kernel function. In our case, we need to define a kernel function based on the distance between two elements (*i.e.* two nested lists representing the formal semantic analysis of user's requests). To illustrate the notions introduced below, we use again the example 2 (R_{ex2}) introduced in 2.3 and represented on figure 1. Let R_1 and R_2 be two requests. Let's call:

s_i or $s(R_i)$	the identifier of R_i	($s(R_{ex})$ = Click)
$f_{i,j}$	the j$^{\text{th}}$ field of R_i	($f_{ex,2}$ = "clicked")
$\|f_i\|$	the number of fields of R_i	($\|f_{ex}\| = 3$)
$t(f_{i,j})$	the type of $f_{i,j}$	($t(f_{ex,2})$ = object)
$v(f_{i,j})$	the value of $f_{i,j}$	
	(a request (*e.g.* clicked = object[...]), a list of requests	
	(*e.g.* property field of 'object') or a terminal value in a list	
	or into an interval (*e.g.* val = button))	
$\|v(f_{i,j})\|$	the number of values of $f_{i,j}$	
	(1, except in the cases of multiple values fields	
	like property* for the references)	
$\mathcal{R}_i$	the set of all subtrees of requests of the request R_i	
	($\mathcal{R}_{ex}$=Click[...], object[...], type[...], quantity[...], color[...])	

We wish to measure the distance between R_1 and R_2. We can consider those requests as two trees, and hence use a kernel function like the one described in [16] (and reminded in [17]) to measure the distance between two labeled ordered (children are in a constant order) directed trees, which key idea is to consider all subtrees from a parse tree.
We define the kernel function k as:

$$k(R_1, R_2) = \sum_i h_i(R_1) h_i(R_2)$$

where $h_i(R)$ is the number of times the i$^{\text{th}}$ subtree appears among all the possible subtrees of the request R.

The problem is that the number of all possible subtrees is very high (the number of subtrees of a tree being exponential relatively to its size), and the complexity of the algorithm would be dependent of it. To get back to a polynomial complexity, if we consider the couple of requests subtrees r_1 and r_2 belonging respectively to $\mathcal{R}_1$ and $\mathcal{R}_2$, we can rewrite the kernel function as:

$$k(R_1, R_2) = \sum_{r_1 \in \mathcal{R}_1, r_2 \in \mathcal{R}_2} S(r_1, r_2)$$

where $S(r_1, r_2)$ is the number of isomorphic subtrees of r_1 and r_2.

If we define:

$\|\delta(r)\|$	the number of children of the entity r ($\neq \|f_i\|$, as the fields can contain a terminal value which is not a child - *e.g.* type[val="button"] has a field filled with a value, but no child request)
$\delta(r, j)$	the j^{th} child of the entity r.

We can get the value of $S(r_1, r_2)$ as:

$$
\begin{aligned}
S(r_1, r_2) &= 0 \text{ if } s(r_1) \neq s(r_2) \\
S(r_1, r_2) &= 1 \text{ if } s(r_1) = s(r_2) \wedge |\delta(r_1)| = |\delta(r_2)| = 0 \\
S(r_1, r_2) &= \prod_{j=1}^{|\delta(r_1)|} (1 + S(\delta(r_1, j), \delta(r_2, j))) \text{ otherwise}
\end{aligned}
$$

It is possible since $|\delta(r_1)| = |\delta(r_2)|$ as the number of children is linked to the identifier). Note that this recursive algorithm has a time complexity of $\mathcal{O}(|R_1||R_2|)$.

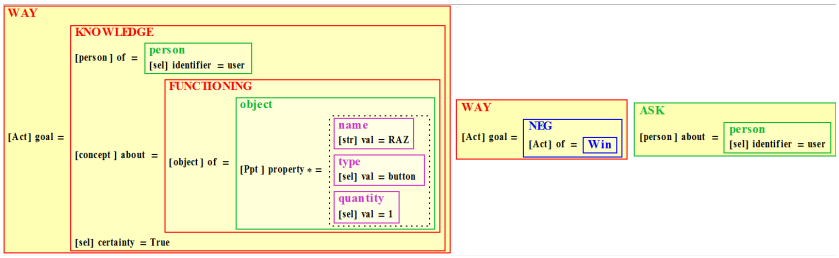

Fig. 2. Example of representation of a partially analyzed user request ("I can't figure out how the RAZ button is working")

Possibility of application to DAFT requests We need to check if the constraints reminded in 3.1 can be satisfied in our particular case:

- As any request follows the order of definition given by schemas, the fields of a request are always in the same order, and hence it is not necessary to consider their permutations among the subtrees. So the children of an entity are indeed in a constant order.
- The number of fields is the same for any given head (*e.g.* Click), so we indeed have $|f_i| = |f_j|$ if $s_i = s_j$.
- If a request hasn't been completely analyzed by DIG, it can be made of several trees (cf. figure 2). However, a set of N trees can be treated as a single tree which root has N children. The order of elements is linked to the order of words in the original user request: we'll consider it doesn't need to be changed.

– A field can sometimes contain a list of values (instead of a single one - cf. the field property* of object on figure 2): it means that a field can have more than a child in the tree. From a representation point of view, it's not a problem: property* shall have three children, just like KNOWLEDGE. However, there is a real difference between multiple fields and entities. As said above, an entity always has a known number of fields given in a defined order. On the contrary, the number of entities into a field containing a list of values is not always the same and their order can be different too (as it is based on the order of appearance of elements in the natural language request).

The necessary conditions to apply the kernel defined above to DIG requests are filled. However the last point concerning multiple fields shows a modification in the calculation of S will be needed.

Modifications to the kernel function The kernel function used to measure the distance between two requests isn't fully appropriate for several reasons, related to the specificities of the data used (problems 1 and 2) and to the chosen kernel function itself (problems 3 and 4):

1. The kernel function only takes into account entities' heads, and not the fields values when they are not an entity (*i.e.* when they are leaves of the tree, that is a terminal value like "button" or "user" in figure 2).
2. The lack of handling of multiple values fields (*cf.* last point in 3.1).
3. The kernel function values are dependent of the tree size. For instance, the proximity between two identical short requests will be lower than between two identical requests having more entities.
4. The kernel is said to be very peaked: very small modifications in a subtree can have an important impact on the value of the distance function. For instance, we can easily have $k(R_1, R_1) \geq 10^2 \ \ k(R_1, R_2)$. That means it will give a very high weight to the closest tree, which can be problematic for some classification algorithms.

To solve problems 1 and 2 linked to the field values (a list in the first case, a terminal value in the second) requires modifications to the calculation of S:

1. Handling terminal values in subtrees, led to the following changes:

$$
\begin{aligned}
S(r_1, r_2) &= 0 \ \text{ if } \ s(r_1) \neq s(r_2) \\
S(r_1, r_2) &= 1 \ \text{ if } \ s(r_1) = s(r_2) \wedge |f_1| = |f_2| = 0 \\
S(r_1, r_2) &= \frac{1 + |v(f_1, j) = v(f_2, j), j \in [\![1, |f_1|]\!]|}{1 + |f_1|} \\
&\quad \text{if } \ s(r_1) = s(r_2) \wedge |\delta(r_1)| = |\delta(r_2)| = 0 \\
S(r_1, r_2) &= \prod_{j=1}^{|\delta(r_1)|} (1 + S(\delta(r_1, j), \delta(r_2, j))) \ \text{ otherwise}
\end{aligned}
$$

The second line handles the case where r_1 and r_2 are identical terminal values (*e.g.* "button" and "button"), whereas the third line is used when the head of r_1 and the head of r_2 are identical but not necessarily the content of all their fields (*e.g.* $a[b, c]$ and $a[b, d]$ have the same head and one field in common). So instead of considering two requests can just be identical (1) or different (0), we say that their proximity value is dependent on the similarity of the values of their fields: if all the fields are filled with the same value, they are identical (1), whereas if they are all different, the proximity is low but they are not completely different since at least their heads were the same ($1/(1 + |f1|)$).

2. To handle multiple values fields, the problem is double as it means:

- Dealing with the fact the order of fields can be different. For example:
  `object[ppt*=color[val=‘‘red’’],size[val=‘‘small’’]]`
  `object[ppt*=size[val=‘‘small’’],color[val=‘‘red’’]]`
  We can get back to an ordered situation (as suggested in [18]), by using the requests heads to sort them alphabetically during the data preprocessing.
- To handle the variability of the number of fields is trickier, because we can't keep comparing elements two by two. However, the algorithm can be adapted to test instead the presence of values from a multiple values field r_1 into another multiple values field r_2 (and conversely). This is done through a normalization based on the total number of values in this field for both r_1 and r_2. Formally it gives:

$$S(r_1, r_2) = \frac{|\{v(f_{1,j}) \in f_{2,j}\}| + |\{v(f_{2,j}) \in f_{1,j}\}|}{|\{v(f_{1,j})\} \cup \{v(f_{2,j})\}|} \quad \text{if } s(r_1) = s(r_2) \wedge |v(f_{i,j})| \neq 1$$

As for problems 3 and 4, a solution to them is suggested in [16] that can be applied here too:

3. To prevent the raise of the k with the complexity of the request, it can be normalized:

$$k'(R_1, R_2) = \frac{k(R_1, R_2)}{\sqrt{k(R_1, R_1) \times k(R_2, R_2)}}$$

4. Finally to get rid of the peak effect, a solution is to add a coefficient λ such as $0 < \lambda \leq 1$ in the calculation of S, which would give a weight dependent on the number of subtrees, hence modifying the formula of the kernel function:

$$k(R_1, R_2) = \lambda^{\text{size}_i} \sum_i h_i(R_1) h_i(R_2)$$

The value of λ can be changed to optimize the classification.

3.2 Classification results

Now that we have defined an appropriate distance measure between the requests, any classical data classification method could be tried, by using the distance between the request to classify (in one of the four requests categories) and the set of requests from the training set. We have been using a 10-fold cross-validation, so the performance results of each classifier given below are a mean of the 10 results.

We have been applying K-Nearest neighbors method for K varying from 1 to 15 and λ from 0.05 to 1 with a step of 0.05. In every case the optimum was found for $\lambda \in [0.7, 0.9]$, and the performance was globally decreasing as K was increasing, the best result being obtained for K=1 and $\lambda = 0.85$, which classifies correctly 65.4% of the requests. Figure 3 shows the variation of λ for K=1 and variation of K for the optimum λ.

When more than one class is possible (for example, with 4-nearest neighbors, 2 close neighbors are in the control category and 2 others in the chat one), the class is picked randomly among the possibilities. Changing this strategy to pick the category of the closest neighbor instead hasn't revealed any significant difference in the results (note that in that case, for K=2 the result is then identical to K=1, which explains the plateau for K=2 on figure 3).

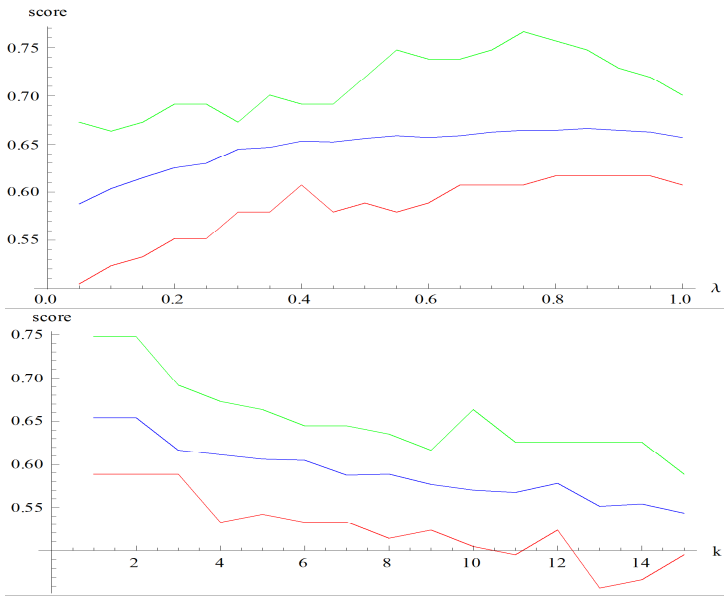

Fig. 3. Performance of the NN and KNN methods for $\lambda = 0.85$ (top = best score, middle = mean score, bottom = minimum score)

4 Semantic-based classification: requests as vectors

4.1 Methodology: schemes vectors definition

For each request, we can consider its *scheme vector*, made of the number of each of the 237 schemes used in a request. Doing so, we do not take into account the structure of a request but only the nature of its elements.
From the 1,070 requests automatically transcribed in DAFT by DIG, we have thus obtained a matrix of 1,070 scheme vectors of 237 integer values corresponding to the number of occurrences of each scheme in the request. For example, if the first element of the vector represents the number of ASK schemes and the second the number of NEG schemes, a request with no ASK and two NEG entities would be associated to a scheme vector $v = 0, 2, \ldots$ (with $|v| = 237$). From there, we can use any classical classifier to identify the subcorpus of a request according to its scheme vector.

4.2 Classification results

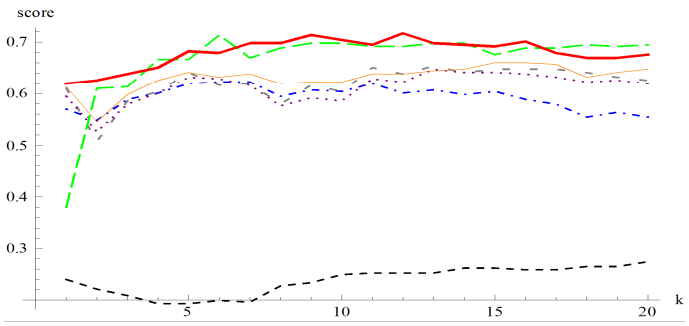

Fig. 4. KNN method ($K \in [\![1, 20]\!]$) applied to schemes vectors with different distances: Euclidian (gray dashed), Chebyshev (bottom dashed), Hamming (dot dashed), Manhattan (dotted), Canberra (thin), Correlation (long dashed), Bray-Curtis (thick)

As in 3, we can use a KNN approach. But this time, the distance function hasn't been defined beforehand; we have first tried seven classical distance functions: Euclidian, Chebyshev, Hamming, Manhattan, Canberra, Correlation and Bray-Curtis. They have been tested with a training set made of $7/10^{\text{th}}$ of the corpus (749 requests) and a test set made of the remaining $3/10^{\text{th}}$ (321 requests), with the number of neighbors K varying from 1 to 20 (cf. figure 4).
Bray-Curtis and correlation distance appear to be the most promising ones and thus have also been tested with a 10-fold cross-validation in order to compare the system overall performance with what we had in previous sections.

Among other classical classifications methods tried with Weka (which main results are displayed in table 1), the best results were obtained with a Naive Bayes classifier with a kernel estimator.

Table 1. Performance of classification methods with a vector approach (leave-one-out)

Classification method	Overall score
Baseline	*40.2%*
Multilayer perceptron	55.7%
K Nearest Neighbours (K=10, Bray-Curtis)	69.2%
Decision tree (C4.5)	70.4%
Adaboost with C4.5	71.8%
Naive Bayes (with kernel estimator)	**74.1%**

5 Combination of classifiers

At this point, we can consider the possibility that previous classifiers from both approaches could be combined into multiple classifier system to improve the overall performance [19] and we have chosen a parametric approach (*i.e.* addition of classifier taking as input the output of the previous ones).

Table 2. Summary of the performance of all the classifiers (with leave-one-out)

ID	**Classification method**	**Overall score**
-	Baseline	*40.2%*
sem1	Naive Bayes (with kernel estimator)	74.4%
sem2	Decision tree (C4.5)	71.1%
sem3	K Nearest Neighbours (K=9)	63.7%
sem4	Adaboost Naive Bayes (with kernel est.)	71.1%
sem5	Bayes Net	64.9%
sem6	Decision Table	71.0%
sem7	Adaboost C4.5	71.7%
ker	Nearest Neighbours	65.4%
sem1+ker	Decision Table	74.4%
sem1+sem2	C4.5	75.2%
sem[1-7]	Decision Table	**76.1**%
sem[1-7]+ker	Decision Table	76.1%

Table 2 recalls the results of each classifier taken individually (results may vary a bit from the ones given in previous sections since each classification is done with a leave-one-out method instead of the 10-folds):

- The ID is an arbitrary name used to name combinations of classifiers.
- The classification method is the name of the method used, possibly with a parameter.

– The overall score is the performance achieved by the classifier.

In the worst case, when we have N first level classifiers, the addition of a $(N+1)^{th}$ first level classifier may not increase the overall score of the global classifier but can't decrease it as there is always a way with a decision table or a decision tree to keep only the results of the N first level classifiers, if the $(N+1)^{th}$ would decrease the overall score. We see the best results are obtained with a decision table applied to the output of different semantic-based classifiers: the addition of the kernel-based classifier doesn't help to improve the overall performance.

6 Conclusion and perspectives

We have seen that identifying conversational activities from the semantic formal representation of a user request could be achieved using machine learning methods in at least two different ways: first, using a kernel-based method where requests were taken with a tree representation (*i.e.* taking into account both the structure and the semantics of the request), secondly by considering them as feature vectors where each feature is corresponding to the number of a given entity in the semantic formal representation (*i.e.* taking into account only the semantics of the request). Although providing more information, the first approach has given lower results (65.1% with a NN method) than the second (74.1% with Naive Bayes). Combining different classifiers using the feature vectors approach, we manage (with a decision table) to reach a performance of 76.1% correctly identified conversational activities. Those results, without being outstanding, are enough to qualify this method as a good counterpart to the rule-based approach used in DIG, in order to be able to at least identify the general nature of the user's request whenever the accurate meaning of the request fails to be understood by the assisting agent. Note that to evaluate precisely this sinergy would require focusing only on cases where rules fail, since they are mainly the cases where we would need the machine learning approaches exposed in this paper.

The fact that the approach based only on semantics provides better results seems to mean the structure might be less relevant in conversational activity identification. Nonetheless, to confirm this, whenever the choice discussed in the preamble of section 2.3 to deal with real data instead of theoretically ideal representation of requests remain valid, it could be interesting to try it to see if the second method remains the best one when the structure of the request is always correct. Indeed, the main problems in the requests produced by DIG are more related to the construction of the correct global structure from individual entities than to the identification of those entities themselves. Besides, the importance of the performance improvement would help in evaluating how crucial an enhancement of DIG would be. Finally, we might also consider a finer level of granularity for the conversational activity, for example by trying to identify "assistance question about the possibility to do something" in the subcase of the "direct assistance" activity, to increase the relevance of reactions.

References

1. Ummelen, N., Neutelings, R.: Measuring reading behavior in policy documents: a comparison of two instruments. IEEE Transactions on Professional Communication **43**(3) (2000) 292–301
2. Morrell, R.W., Park, D.C.: The effects of age, illustrations, and task variables on the performance of procedural assembly tasks. Psychology and Aging **8**(3) (September 1993) 389–99 PMID: 8216959.
3. Amalberti, R.: La conduite des systèmes à risques. PUF (1996)
4. Cassell, J., Sullivan, J., Prevost, S., Churchill, E., eds.: Embodied Conversational Agents. MIT Press (April 2000)
5. Lester, J.C., Converse, S.A., Kahler, S.E., Barlow, S.T., Stone, B.A., Bhogal, R.S.: The persona effect: affective impact of animated pedagogical agents. In: Proceedings of the SIGCHI conference on Human factors in computing systems, Atlanta, Georgia, United States, ACM (March 1997) 359–366
6. Voorhees, E.M.: Overview of TREC 2007. In: The Sixteenth Text REtrieval Conference Proceedings, Gaithersburg, Maryland (November 2007) 1–16
7. Leray, D., Sansonnet, J.: Ordinary user oriented model construction for assisting conversational agents. In: CHAA'06 at IEEE-WIC-ACM Conference on Intelligent Agent Technology. (2006)
8. Pollard, C., Sag, I.A.: Head-Driven Phrase Structure Grammar. University Of Chicago Press (August 1994)
9. Brown, P.F., Cocke, J., Pietra, S.A.D., Pietra, V.J.D., Jelinek, F., Lafferty, J.D., Mercer, R.L., Roossin, P.S.: A statistical approach to machine translation. Comput. Linguist. **16**(2) (1990) 79–85
10. Randall, N., Pedersen, I.: Who exactly is trying to help us? the ethos of help systems in popular computer applications. In: Proceedings of the 16th annual international conference on Computer documentation, Quebec, Quebec, Canada, ACM (1998) 63–69
11. Zhang, D., Lee, W.S.: Question classification using support vector machines. In: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, Toronto, Canada, ACM (2003) 26–32
12. Li, X., Roth, D.: Learning question classifiers: The role of semantic information. Natural Language Engineering **12**(03) (2006) 229–249
13. Bouchet, F., Sansonnet, J.: Etude d'un corpus de requêtes en langue naturelle pour des agents assistants. In: WACA 2006 : Actes du Second Workshop Francophone sur les Agents Conversationnels Animés. (October 2006)
14. Fellbaum, C.: WordNet: An Electronic Lexical Database. MIT Press, Cambridge, MA (1998)
15. Bouchet, F., Sansonnet, J.: Caractérisation de requêtes d'Assistance à partir de corpus. In: Actes de MFI'07, Paris, France (May 2007)
16. Collins, M., Duffy, N.: Convolution kernels for natural language. In Dietterich, T.G., Becker, S., Ghahramani, Z., eds.: Advances in Neural Information Processing Systems 14, Cambridge, MA, MIT Press (2002)
17. Gaertner, T.: A survey of kernels for structured data. SIGKDD Explor. Newsl. **5**(1) (2003) 49–58
18. Vishwanathan, S., Smola, A.J. In: Fast Kernels for String and Tree Matching. The MIT Press (August 2004) 113–130
19. Kittler, J.: Combining classifiers: A theoretical framework. Pattern Analysis & Applications **1**(1) (March 1998) 18–27