

CLASSiC

D2.1: Training Tools and Semantic Decoder for the TownInfo Task

François Mairesse

Distribution: Public

CLASSiC

Computational Learning in Adaptive Systems for Spoken Conversation
216594 Deliverable 2.1

Feb 2009



Project funded by the European Community
under the Seventh Framework Programme for
Research and Technological Development



The deliverable identification sheet is to be found on the reverse of this page.

Project ref. no.	216594
Project acronym	CLASSiC
Project full title	Computational Learning in Adaptive Systems for Spoken Conversation
Instrument	STREP
Thematic Priority	Cognitive Systems, Interaction, and Robotics
Start date / duration	01 March 2008 / 36 Months

Security	Public
Contractual date of delivery	M12 = Feb 2009
Actual date of delivery	Feb 2009
Deliverable number	2.1
Deliverable title	D2.1: Training Tools and Semantic Decoder for the Town-Info Task
Type	Prototype
Status & version	Final 1.0
Number of pages	11 (excluding front matter)
Contributing WP	2
WP/Task responsible	UCAM
Other contributors	M. Gašić, F. Jurčiček, S. Keizer, B. Thomson, K. Yu and S. Young at UCAM and J. Henderson at GENE
Author(s)	François Mairesse
EC Project Officer	Xavier Gros
Keywords	Semantic Decoder, Spoken Language Understanding

The partners in CLASSiC are:

University of Edinburgh HCRC	EDIN
University of Cambridge	UCAM
University of Geneva	GENE
Ecole Supérieure d'Electricité	SUPELEC
France Telecom/ Orange Labs	FT

For copies of reports, updates on project activities and other CLASSiC-related information, contact:

The CLASSiC Project Co-ordinator
Dr. Oliver Lemon
School of Informatics
Edinburgh University
EH8 9LW
United Kingdom
olemon@inf.ed.ac.uk
Phone +44 (131) 650 4443 - Fax +44 (131) 650 4587

Copies of reports and other material can also be accessed via the project's administration homepage, <http://www.classic-project.org>

©2009, The Individual Authors.

No part of this document may be reproduced or transmitted in any form, or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission from the copyright owner.

Contents

Executive Summary	1
1 Overview of the Semantic Decoder	2
1.1 Semantic tuple classifiers	2
1.2 Training the semantic decoder from unaligned data	2
1.3 Decoding user utterances	3
1.4 Probabilistic extension	4
1.5 Evaluation	5
1.6 Other semantic parsing techniques	6
2 Training Tools Manual	7
2.1 Requirements	7
2.2 Input data format	7
2.3 Training and evaluating the models	8
2.4 Using the trained models in the CLASSIC dialogue system architecture	9
A Publication Abstracts	11

Executive summary

This document describes the Prototype deliverable 2.1, due at month 12 of the CLASSiC project. It presents a brief overview of the trainable semantic decoder and details the training tools made available for the TownInfo task. The semantic decoder is a Spoken Language Understanding (SLU) component that can learn to predict the meaning of unseen user utterances from a set of training utterances labelled with CLASSiC project dialogue act annotations (derived from the Cambridge University dialogue act scheme). In order to address data sparsity issues, *semantic tuple classifiers* (STC) are trained to predict fragments of the semantic representation (e.g., FOOD→CHINESE). At decoding time, the predictions of the STC's are combined together to produce a user act hypothesis. As the main objective of the CLASSiC project is to build a system that optimises dialogue decisions by modelling the uncertainty over the nature of the user input, probabilistic STC's are used to output a weighted list of user act hypotheses. The document also describes the evaluation of this component against state-of-the-art approaches to SLU. A detailed paper presenting this work [1] was published at ICASSP 2009 (see abstract in Appendix A), and is available at www.classic-project.org.

1 Overview of the Semantic Decoder

This section presents an efficient yet simple technique that learns discriminative semantic concept classifiers whose output is used to recursively construct a semantic tree, without requiring any alignment information. While this deliverable focuses on the high level mechanism of the algorithm and the training tools, more details can be found in the conference paper presented at ICASSP 2009 [1].

1.1 Semantic tuple classifiers

In this prototype, we cast the spoken language understanding task as a classification problem in which each utterance is mapped to its semantic representation (e.g., ‘*I would like a Chinese restaurant*’ is mapped to the tree `INFORM(TYPE(RESTAURANT) FOOD(CHINESE))`). However, each possible semantic representation cannot be treated as a class because of data sparsity issues, i.e. there are not enough examples of individual semantic trees in the data. In order to reduce data sparsity, we split each semantic tree into *semantic tuples*—i.e., a sequence of contiguous nodes within a branch of the tree, such as `INFORM→TYPE` and `TYPE→RESTAURANT`—and learn individual classifiers for each tuple. When decoding a new utterance, each classifier is applied to a set of utterance features, and the semantic tree is reconstructed from the set of predicted tuples (see Figure 1 and Section 1.3).

1.2 Training the semantic decoder from unaligned data

The training process requires a dataset of utterances together with annotated semantic representations, as well as a database of entities of interest in the domain (e.g., specifying that ‘Main square’ is a valid place name). Concerning the learning algorithm of individual semantic tuple classifiers (STC’s), our experiments showed that Support Vector Machine (SVM) classifiers with a linear kernel produce the best results on our dataset. The utterance features used to discriminate between semantic concepts consist of the set of word n -grams counts in the utterance. The n -gram size n is optimised for each STC on the training data.

The maximum tuple length l parameter effectively controls the trade-off between (a) the accuracy of individual classifiers and (b) the non-ambiguity of the tree reconstruction process. For example, classifiers returning a full branch ($l = \infty$) make the reconstruction process trivial at the expense of classification accuracy, whereas classifiers returning individual semantic concepts ($l = 1$) produce ambiguous parses. Additionally, smaller tuples generalise better to branches that are not seen during training. For the Town-Info task, the best results were obtained with a tuple length of 2 concepts.

The full STC training algorithm can be described as follows:

- **Input:** a set of (utterance, semantic tree) pairs, a maximum tuple length l and a domain database.
 - **Output:** a set of semantic tuple classifiers and a domain grammar.
1. Replace database values in the training utterances with category labels (e.g., ‘*I want a restaurant near PLACE_NAME*’).
 2. Compute relevant lexico-syntactic features for each utterance (e.g., n -gram frequency counts with n from 1 to 3).

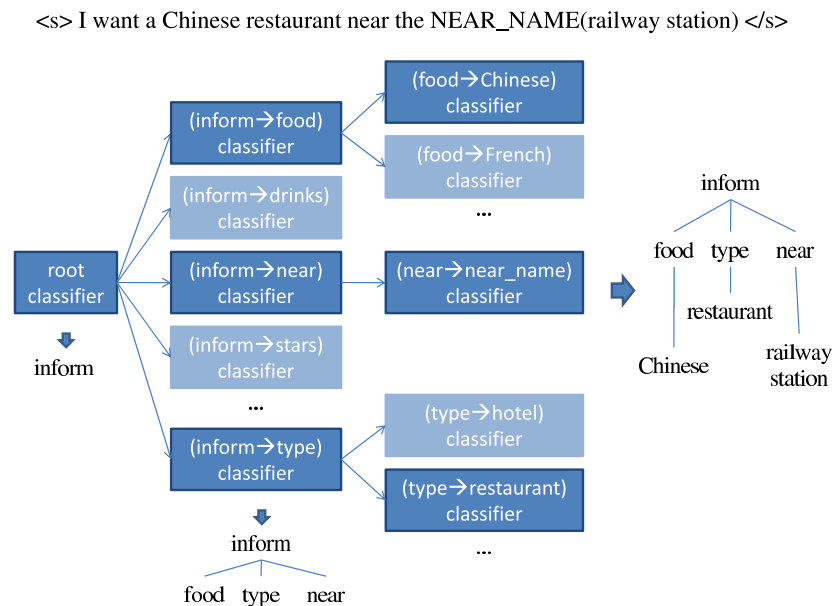


Figure 1: Semantic tree derivation for an utterance in the TownInfo dataset with a tuple length of 2, with positive concept tuple classifications in darker boxes.

3. For each distinct tuple of maximum l concepts in the semantic trees:
 - (a) Create a dataset associating each training utterance with a binary class representing whether the tuple occurs in the utterance's semantic tree.
 - (b) Train a binary semantic tuple classifier that predicts the class—i.e., the tuple—from the feature values. The root concept (e.g., dialogue act type) is predicted using a single multi-class classifier.
4. Construct a domain grammar that matches all trees in the training set.

1.3 Decoding user utterances

Once the STC models have been trained, they can be used to predict the meaning of unseen user utterances. Figure 1 illustrates the parsing process for the TownInfo domain, in which the semantic representation is reconstructed by combining positive semantic tuple classes based individual tuple constraints. The decoding algorithm consists of the following steps:

- **Input:** the user utterance, the semantic concept classifiers, the domain grammar and database.
- **Output:** the semantic tree of the input utterance.

1. Replace database values in the utterance with category labels.
2. Compute the utterance's n-gram features and filter out those not seen during training (e.g., unseen n-grams).
3. Run all semantic tuple classifiers (STC) on the utterance's features, set T equal to the set of positively classified tuples.
4. Initialise the output semantic tree as the predicted root, and a variable r pointing to the root concept. Then recursively do either:
 - (a) *High precision mode*: For each tuple t of T whose root has the same concept as r , append t 's non-root nodes to r in the semantic tree, and remove t from T . Start over recursively by setting r equal to t 's terminal node.
 - (b) *High recall mode*: Start with (a). Then, for each remaining tuple t of T that is dominated by the concept r in the domain grammar, append t to r in the semantic tree. Start over recursively by setting r equal to t 's terminal node.
5. Associate each of the tree's terminal nodes corresponding to a database category label with the corresponding value in the utterance (e.g., `NEAR_NAME` becomes `railway station`).

The output of this algorithm is a semantic tree of the utterance, with terminal concepts associated with database values. In *high precision mode*, the algorithm relies only on the outputs of the classifiers to expand the tree (e.g., `FOOD`→`CHINESE` can only be appended if the `FOOD` concept is already part of the tree), whereas in *high recall mode* the tree can be expanded as long as the result matches the domain grammar (e.g., `FOOD`→`CHINESE` can be appended to `INFORM` if this combination has been seen during training). Early experiments showed that the high recall mode performs better on the TownInfo task, it was thus used to obtain the evaluation results presented in Section 1.5.

1.4 Probabilistic extension

In order to produce a set of semantic hypotheses rather than a deterministic SLU decision, we extended the STC model to output an n-best list of dialogue acts. This is achieved by training probabilistic SVM classifiers that map the classification margin of a tuple t given an user utterance u to a probabilistic confidence score $P(t|u)$. The probability of a 2-level dialogue act da given the user utterance is evaluated by multiplying the probability of the root (i.e., the dialogue act type) with the probability of occurrence of each tuple in the act, together with the complement of the probability of non-occurring tuples:

$$P(da|u) = P(\text{root}|u) \cdot \prod_{t \in da} P(t|u) \cdot \prod_{t \notin da} (1 - P(t|u))$$

As evaluating this expression for every possible dialogue act would be intractable, the probabilistic STC model returns this probability for every dialogue act containing only tuples with a probability above a cut-off threshold (e.g., $P(t|u) > 0.1$). Once the STC semantic decoder is integrated in the CLASSiC architecture, the uncertainty from the automatic speech recognition (ASR) component is combined with

Parser	DA	Prec	Rec	F
TownInfo dataset with transcribed utterances:				
STC	94.92	97.39	94.05	95.69
Phoenix	94.82	96.33	94.20	95.26
TownInfo dataset with ASR output:				
STC	85.15	94.03	83.73	88.58
Phoenix	74.73	90.28	79.49	84.54
ATIS dataset with transcribed utterances:				
STC	92.63	96.73	92.37	94.50
He & Young (2006)	–	–	–	90.3
Z & Collins (2007)	–	95.11	96.71	95.9
Meza-Ruiz et al. (2008)	–	93.43	89.77	91.56

Table 1: Dialogue act classification accuracy (*DA*), slot/value precision (*Prec*), recall (*Rec*) and F-measure for the ATIS and TownInfo test datasets. Semantic tuple classifiers (STC) are compared with a handcrafted Phoenix parser, as well as results reported by He & Young, Zettlemoyer & Collins and Meza-Ruiz et al.

the uncertainty over the semantic decoding task. This is achieved by (a) decoding every candidate utterance from the n-best list returned by the ASR module, (b) multiplying the resulting confidence in each dialogue act hypothesis $P(da|u)$ by the probability of the input user utterance given the observed speech $P(u|o)$ (weighted by a scaling factor), and (c) merging identical candidate dialogue acts by summing their probabilities to produce the final n-best list of dialogue acts that is sent to the dialogue manager.

1.5 Evaluation

The STC semantic decoder was evaluated on two datasets: annotated user utterances in the tourist information (TownInfo) and air travel information (ATIS) domains [2]. Evaluation results for both datasets are presented in Table 1, in which STC’s are compared with existing state-of-the-art techniques on the ATIS data and a handcrafted Phoenix parser for the TownInfo data [3]. The model accuracy is measured in terms of the percentage of correctly classified dialogue act types, as well as the F-measure of the slot/value pairs. Only the first semantic hypothesis is considered, and both the slot and the value must be correct to count as a correct classification. The dialogue act type is the root of the output tree, whereas slot/value pairs are trivially extracted from the branches (e.g. FROMLOC→CITY→New York becomes FROMLOC.CITY = New York).

Results on the TownInfo domain show that STC’s trained and tested on transcribed utterances perform slightly better than the handcrafted Phoenix grammar. Classifiers trained and tested on ASR outputs show a large improvement over the handcrafted grammar, for both the dialogue act accuracy (10.42% improvement with 85.15% accuracy) and the F-measure (4.04% improvement with F=88.58%). These results imply that STC’s are capable of capturing patterns despite the noise in the ASR output, which makes them suitable for performing SLU in noisy environments.

Concerning the ATIS dataset, Table 1 shows that the STC algorithm produces a 92.63% dialogue act accuracy and a 94.50% F-measure, which represents a 4.2% improvement over He & Young’s Hidden Vector State model [4] and a 2.94% improvement over Meza-Ruiz et al.’s Markov logic approach [5], but 1.4%

lower compared with Zettlemoyer & Collins' PCCG model [6]. Note however that Zettlemoyer & Collins' complex iterative grammar induction technique requires handcrafting domain-independent parsing rules in the initial grammar lexicon (e.g., PCCG lexicon entries for wh-words).

1.6 Other semantic parsing techniques

We also investigated other statistical methods for semantic parsing throughout the first year of the CLASSiC project. Ongoing work reveals that a different approach based on *transformation-based learning* produces similar results as the STC method on the TownInfo and ATIS datasets (in preparation, see Appendix A).

2 Training Tools Manual

This section details how to use the training tools to re-train the STC semantic decoder on a new domain. All file and directory references are relative to the root of the STC training tools package.

2.1 Requirements

The training tools require the following external components, which are included in the training tools package:

- LibSVM jar file (file `lib/libsvm.jar`)
- Weka 3.5.8, with a modified LibSVM interface originally from Yasser EL-Manzalawy and Vasant Honavar (file `lib/weka-3-5-8.jar`)

The training tools have only been tested under Linux.

2.2 Input data format

The training tools have been evaluated on two datasets: the TownInfo dataset and the Air Travel Information System (ATIS) dataset, which are both included in the package archive. The input data is based on the following format, one utterance per line:

```
utterance <=> dialogue_act_type((slot_label=slot_value?)*)
```

The semantic representation on the right hand side is based on the CUED dialogue act representation. The transcribed utterances for the TownInfo task are located in the `data/towninfo-sem/* .sem` files, whereas the `data/towninfo-asr/* .asr` files contain the top speech recognition hypotheses. The transcribed ATIS data is located in the `data/atis-sem` directory.

In each directory, the training data must be split into a training, development and test set (e.g. `atis-dev.sem`). The `dialogAct.lst` in each dataset directory must contain all possible dialogue act types. The `db` directory contains the database associating lexical realisations of slot values to slot label categories (e.g., "Railway Station" is a value for the NEAR concept).

If some database values must be modelled independently of other values of the same category (e.g., to learn synonyms of specific values such as for `near="Railway Station"`), the file `db/non-enumerables.txt` should contain the list of category labels that should *not* be modelled explicitly, for which the value is retrieved by matching entries in the database against the utterance (e.g. `addr="Alexander Street"`). In order to reduce the number of classifiers, categories with a high cardinality should be included as non-enumerable. If the file is not found than all categories are modelled using a single generic classifier.

2.3 Training and evaluating the models

In order to train the semantic classifiers and evaluate their performance on held out data, one can run the `run_experiment.sh` script in the root directory to run the full experiment with either the ATIS or TownInfo dataset. E.g., to train and evaluate models on the TownInfo ASR outputs, run

```
./run_experiment.sh towninfo-asr
```

This script provides examples of experiments with different training and testing parameters, which are passed as arguments to the main shell script `run.sh` by using the following syntax:

```
./run.sh sem_file database_dir tuple_size output_dir (-test|-dev) (-prob|-det)
(-gridsearch|-nogridsearch) [-model_classes_in_dev] [-feat_ext file_suffix]
```

Arguments in square brackets are optional. The ordered arguments are:

- `sem_file`: the path to the training `.sem` file, must be of the form `[path]/[name]-train.(sem—asr)`
- `database_dir`: the path to the lexical database of slot values
- `tuple_size`: the maximum number of semantic nodes in each tuple
- `output_dir`: output directory of the experiment
- `-test|-dev`: either tests on the development test file ("`-dev`") or the test set ("`-test`")
- `-prob|-det`: either produce probabilistic SVM classifiers with n-best list output ("`-prob`") or non-probabilistic classifiers ("`-det`")
- `-gridsearch|-nogridsearch`: optimise the maximum n-gram size and cost SVM parameters by doing a gridsearch, i.e. by selecting parameter values through cross-validation (see Weka Grid-Search class). If disabled, the optimal parameter values must already be stored in the model files in the `dev-output/ngrams/*.txt` evaluation files.
- `-model_classes_in_dev`: only evaluate models for the classes in the development set (can increase precision, optional)
- `-feat_ext file_suffix`: replaces the n-gram features with the features provided in the files with extension `file_suffix`, in which each line matches the utterance in the dataset file and contains space separated features following the format `feature=value` (optional).

The `run.sh` script creates a new directory (e.g. `towninfo-asr`), in which the dataset directory contains a set of Weka `.arff` dataset files, the `models` directory contains the Weka models, and the output directory contain the evaluation results as well as the textual representation of the SVM models (i.e., support vectors). The `towninfo-asr/test-output/*.scoresheet` files contain the final evaluation in terms of dialogue act accuracy and attribute F-measure on the test data. The `towninfo-asr/test-output/*.sem` files contain the predicted top dialogue act hypotheses, and the `towninfo-asr/test-output/*.sem.nbest` files contain the predicted n-best dialogue acts for each utterance.

2.4 Using the trained models in the CLASSiC dialogue system architecture

The UCAM dialogue system contains a C++ implementation of the Semantic Tuple Classifier decoder, which can be launched using the command

```
CUED_SYSTEM_ROOT/demoSys/hub/Release/Hub -C hub-HIS-SEMCLASS.cfg -l mylogdir
```

The dialogue system is assumed to be installed in the CUED_SYSTEM_ROOT directory. In order to update the existing models, you need to modify the CUED_SYSTEM_ROOT/resources/SemClassifiers/models.cfg file to point to the new test-output/ngrams/indexed/*.txt ASCII model files corresponding to the new dataset. The models in the indexed directory—in which each n-gram feature is indexed according to a single n-gram list—must be used in order to limit the size of the n-gram table in memory. Additionally, the BINARYMODELS option in the configuration file must be set to 0 (ASCII).

Since support vectors in the ASCII model files can take a long time to load, it is recommended to convert them to a binary format by following these steps:

1. Convert the ASCII LibSVM models in the test-output/ngrams/indexed/*.txt files into binary format, by compiling and running the C++ STC decoder in the CUED_SYSTEM_ROOT/SemIO/SemClassDecode directory (under Linux):

```
make; chmod 755 SemClassDecode;
./SemClassDecode -C config_file -b bin_model_out_dir
```

in which the configuration file contains a MODELFILE option specifying a file containing the path to each ASCII model on each line, following the format "tuple = path_to_ascii_model" (see example file CUED_SYSTEM_ROOT/resources/SemClassifiers/models.cfg). The argument bin_model_out_dir is the directory in which the binary models will be output (.model file extension).

2. Replace the existing models in the CUED_SYSTEM_ROOT/resources/SemClassifiers/models/ngrams/bin/ directory with the new binary .model files
3. Update the CUED_SYSTEM_ROOT/resources/SemClassifiers/models.cfg if necessary, so that each tuple point to the new binary .model file, and set the BINARYMODELS option to 1 in the configuration file.

Bibliography

- [1] F. Mairesse, M. Gašić, F. Jurčićek, S. Keizer, B. Thomson, K. Yu, and S. Young. Spoken language understanding from unaligned data using discriminative classification models. In *Proceedings of ICASSP*, 2009.
- [2] D. A. Dahl, M. Bates, M. Brown, W. Fisher, K. Hunicke-Smith, D. Pallett, C. Pao, A. Rudnicky, and E. Shriberg. Expanding the scope of the ATIS task: The ATIS-3 corpus. In *Proceedings of the ARPA HLT Workshop*, 1994.
- [3] Wayne H. Ward. The Phoenix system: Understanding spontaneous speech. In *Proceedings of ICASSP*, 1991.
- [4] Yulan He and Steve Young. Spoken language understanding using the hidden vector state model. *Speech Communication*, 48(3-4):262–275, 2006.
- [5] I. V. Meza-Ruiz, S. Riedel, and O. Lemon. Accurate statistical spoken language understanding from limited development resources. In *Proceedings of ICASSP*, 2008.
- [6] Luke S. Zettlemoyer and Michael Collins. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of EMNLP-CoNLL*, 2007.

Appendix A

Publication Abstracts

Here are the abstracts of the publications related to this deliverable, the full publications are available at www.classic-project.org.

F. Mairesse, M. Gašić, F. Jurčiček, S. Keizer, B. Thomson, K. Yu, and S. Young. Spoken Language Understanding from Unaligned Data using Discriminative Classification Models. In *Proceedings of IEEE ICASSP*, April 2009.

Abstract: While data-driven methods for spoken language understanding reduce maintenance and portability costs compared with handcrafted parsers, the collection of word-level semantic annotations for training remains a time-consuming task. A recent line of research has focused on building generative models from unaligned semantic representations, using expectation-maximisation techniques to align semantic concepts. This paper presents an efficient, simple technique that parses a semantic tree by recursively calling discriminative semantic classification models. Results show that it outperforms methods based on the Hidden Vector State model and Markov Logic Networks, while performance is close to more complex grammar induction techniques. We also show that our method is robust to speech recognition errors, by improving over a handcrafted parser previously used for dialogue data collection.

F. Jurčiček, M. Gašić, S. Keizer, F. Mairesse, B. Thomson, K. Yu, and S. Young. Transformation-based Learning for Semantic Parsing. In preparation.

Abstract: This paper presents a semantic parser that transforms an initial semantic hypothesis into the correct semantics by applying an ordered list of transformation rules. These rules are learnt automatically from a training corpus with no prior linguistic knowledge and no alignment between words and semantic concepts. The learning algorithm produces a compact set of rules which enables the parser to be very efficient while retaining high accuracy. We show that this parser is competitive with respect to the state-of-the-art semantic parsers on the ATIS and TownInfo tasks.