SAPIENZA
UNIVERSITÀ DI ROMA

Capturing expert domain knowledge and decision strategies via artificial intelligence

Faculty of Information Engineering, Computer Science and Statistics

Corso di Laurea in Informatica

Candidate

Lorenzo Levantesi

ID number 1785694

Thesis Advisor

Prof. Toni Mancini

Academic Year 2019/2020

Capturing expert domain knowledge and decision strategies via artificial intelligence

Bachelor thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: levantesi.1785694@studenti.uniroma1.it

Contents

Chapter 1

Introduction

Active learning is a subfield of machine learning which aims to improve the learning process of a machine by intelligently choosing the elements of the training set. The algorithms of active learning are mainly used when the availability of labeled data is scarce and the obtaining of them is too expensive.

In this thesis, an active learning method is studied to obtain an algorithm that intelligently generates cases to propose to an expert for the labeling. This method has been implemented in the research project *Virtual Doctor Generator*, coordinated by professor Toni Mancini, on which I worked during my internship.

The second chapter is a project that I presented in the course of Artificial Intelligence, and it is requested that it is not evaluated for the final grade. The project describes the various methods used in active learning, to obtain a high level of accuracy in the model learnt using a small number of labeled data with respect to other methods used in machine learning.

The third chapter describes the project *Virtual Doctor Generator*. The project aims to study and implement methods to acquire knowledge and decision strategies of experts, using active learning strategies. Initially, a general description is made, for later diving into the parts of the project in which I worked.

The fourth chapter describes the active learning methods studied in the project, providing a formal description of the techniques used. The results of the tests of these methods are shown in chapter five, where is described the software testing that I made and the considerations on the results.

During the internship, I also designed and implemented a backend system, necessary for the acquiring of medical records, in chapter six the used technologies are briefly described. The system documentation of the backend is placed in the appendices.

Chapter 2

Active Learning

The content of this chapter is the project presented and evaluated in the Artificial Intelligence course.

It is requested to not evaluate this chapter for the decision of the degree mark.

2.1 Introduction

Machine learning algorithms use a large amount of data to produce a decision model that performs well. Almost all machine learning techniques rely on the fact that labeled data are easily available.

For certain domains, these methods can not be used due to the expensive acquiring or limited disponibility of labeled data. Such domains are for example the problem of *classification and filtering of documents* (*e.g.*, web pages), where it is necessary a person, in literature called *Oracle*, that have to *label* each document with its correct classification. With the number of documents available nowadays this is a tedious and even redundant method.

These techniques use a "passive" approach to learn, where the learning algorithm elaborates the labeled data in batch mode, prone to the elaboration of redundant data that not increase the correctness and efficiency of the decision model. That is where an active learning approach can be useful.

Given a small set of labeled data, the ease of obtaining/synthesize unlabelled data, and an oracle capable to label an unlabelled instance, an active learning algorithm can produce an accurate predictive model as one built by any other machine learning approach, using a reduced number of data.

Active learning [9] is a subfield of machine learning where the learner has an "active" role, more specifically the learner has the possibility to asks an oracle to label an unlabelled instance that it believes is the most useful in its learning process. Various active learning methods have a different meaning of "most useful", given the set of unlabelled instances some use a sampling based on uncertainty and others use a sampling based on correlation within the unlabelled instances.

In a typical active learning cycle, the algorithm selects the most valuable instance and requests its label. Then the new labeled instance is added to the training set and the model is retrained.

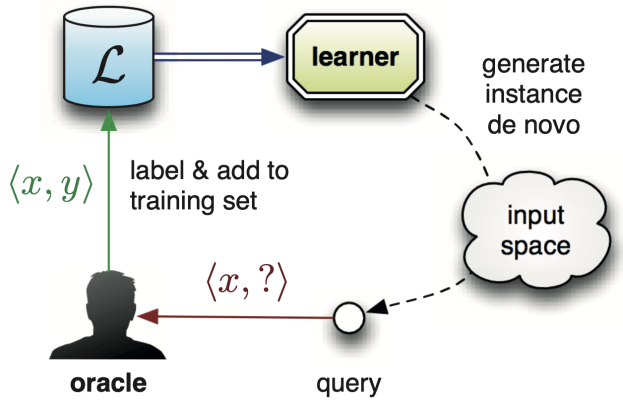


Figure 2.1. Query Synthesis (image taken from [9])

The next sections are composed of an initial general description of the main scenarios of active learning, for later diving into a description of the most used techniques in works and problems where active learning has the main role.

2.2 Sampling scenarios

Active learning is appropriate when the unlabelled data can be easily collected or synthesized and the labeling of an instance comes with a high cost, so the learner can choose which instance to be labeled by an oracle is more useful to improve its decision model.

The main scenarios of sampling instances to ask for labeling are described below.

2.2.1 Query Synthesis

In this scenario (Figure 2.1) the learner may query the labeling of any unlabeled data instance in the input space, including those synthesized by itself. In this case, it's important that the learner has a definition of the input space available to it.

Query synthesis is efficient since the learner can generate cases, given a small set of labeled data. The problem of this approach is that the synthesized instances can be unrecognizable to a human oracle.

For example, Lang and Baum [7] trained a neural network to classify handwritten characters, in that, case many of the characters generated by the learner were without meaning, letting the human oracle unable to label them.

On the other hand, query synthesis has the advantage that the time necessary to learning is independent of the size of the pool of unlabeled data.

To address the problem of synthesizing an unrecognizable instance, a hybrid approach is used. It consists to use both query synthesis and pool-based sampling (explained later), by synthesizing the most informative instance and choose from the pool an instance that is as similar as possible to the synthesized one, allowing the oracle to understand and label it.

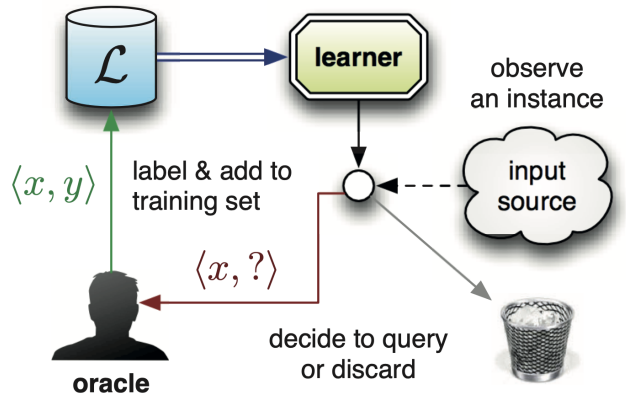


Figure 2.2. Stream-Based Selective Sampling (image taken from [9])

2.2.2 Stream-Based Selective Sampling

In a stream-based selective sampling scenario (Figure 2.2) an instance is sampled from the actual distribution, and then the learner has to choose to query for its label to the oracle or not. This scenario is called stream-based as each sampled instance is drawn one at a time from the input source.

Exist various policies to choose whether to keep or discard a sampled instance. One approach is to define a measure of utility or information content and then make a biased random decision. Another approach is to compute a region of uncertainty [2] which represent the part of the instance space that is ambiguous to the learner and then asking instances that fall within that region.

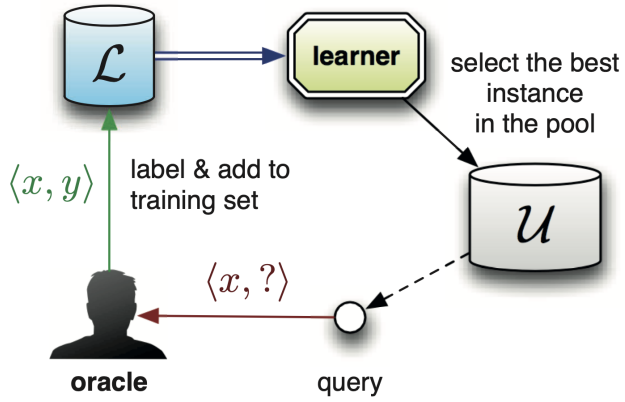


Figure 2.3. Pool-Based Sampling (image taken from [9])

2.2.3 Pool-Based Sampling

A pool-based sampling (Figure 2.3) is one of the most popular scenarios for applied research in active learning. As in many problems, a large set of unlabeled data is available, it would be efficient to ask the instance that is the most informative in the set.

That is what a pool-based sampling approach does. Instances are selected from the pool of unlabelled data and chosen according to a utility measure, labeled by an oracle, and then added to the set of labeled instances.

A pool-based sampling scenario, as said above, is one of the most employed, but there can be cases in which a stream-based selective sampling is more suitable, for example, if memory or processing power is limited (*e.g.*, embedded devices).

In fact, a problem of the pool-based sampling is that the performance depends on the size of the pool, as we compute the information measure of each instance in the pool and compare them by choosing the best one.

2.3 Uncertainty sampling

Uncertainty sampling [8] is one of the most popular strategies in active learning. The basic idea is that a learner queries the labeling of an instance when it finds it confusing to label, avoiding instances it is already confident about.

For example, in a binary classification problem, a learner may ask the labeling of an instance x, which is the closest to the decision boundary θ.
In a probabilistic form, we can say that given a probabilistic classifier that calculate the posterior distribution $P_\theta(Y \mid x)$ over the label class Y, and given the learned model θ, a learner want to find the instance x for which $P_\theta(\hat{y} \mid x)$ is closest to a uniform distribution (*e.g.*, in case of a binary classifier, the one closest to 0.5), where $\hat{y}$ refers to the classifier's most likely prediction for x.

An uncertainty sampling approach requires a classifier that is cheap to build and to use since, at each iteration of the learning process, a new classifier is built and used for an estimate of the uncertainty of classifications.

2.3.1 Measures of uncertainty

```
1 U = the pool of unlabaled instances x^u, u ∈ [1, |U|]
2 L = the set of inital labeled instances (x, y)^l, l ∈ [1, |L|]
3 for t = 1, 2, ... do
4   θ = train(L)
5   select x* ∈ U, the most uncertain instance according to model θ
6   query x* to obtain its label y*
7   add (x*, y*) to L
8   remove x* from U
```

Algorithm 1: Uncertainty sampling pseudo-algorithm

Given the algorithm in Algorithm 1, we initially have U and L, respectively the set of unlabeled instances and the set of labeled instances. Then, while the learnt model θ doesn't satisfy our requirements, train θ with L and choose an instance x^*, according to a measure of uncertainty A of the instances in U, to query and obtain its label y^*. Finally, add (x^*, y^*) to L and remove x from U.

Let be x_A^* the most uncertain instance that the learner would choose depending on the utility measure A (*i.e.*, the one that the model θ find the most difficult to label), the following sections will illustrate various of these utility measures.

Least Confident

This is a basic strategy that consists to query the instance whose predicted label is the least confident:

$$x_{LC}^* = \underset{x}{\operatorname{argmin}}\, P_\theta(\hat{y} \mid x)$$

where

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, P_\theta(y \mid x)$$

is the label prediction on x with the highest posterior probability, according to θ. So the learner queries for the label of the instance whose most likely labeling is the least likely among all the other unlabelled instances. This is a greedy approach, all the information regarding the parts of the prior distribution which isn't the most likely are indeed not considered.

Margin

A margin strategy is based on the output margin:

$$x_M^* = \underset{x}{\operatorname{argmin}}[P_\theta(\hat{y}_1 \mid x) - P_\theta(\hat{y}_2 \mid x)]$$

Here $\hat{y}_1$ and $\hat{y}_2$ are the respective first and second label on x with the highest probability in the posterior distribution $P_\theta(Y \mid x)$ over the label class Y.

The margin sampling chooses the instance whose first two most likely labels has the smallest margin. A small margin is considered ambiguous, so asking for its label would help the learner to improve its model.

The margin approach tries to mitigate the problem in the least confident strategy, by using not only the class $\hat{y}$ with the highest probability value, but including also the second most probable label (*e.g.*, $\hat{y}_2$ in the formula above). However this is still a greedy approach, as for large label sets it still doesn't consider the rest of prior distribution, but only the two most likely.

Entropy

The most general and common strategy is the measure of entropy [11]:

$$\begin{aligned} x_H^* &= \underset{x}{\operatorname{argmax}}\, H_\theta(Y \mid x) \\ &= \underset{x}{\operatorname{argmax}} - \sum_y P_\theta(y \mid x) \log_2 P_\theta(y \mid x) \end{aligned}$$

Here the entropy is denoted with H and in the second form of the formula y ranges over all possible labelings of x. In machine learning, entropy is used as an uncertainty or impurity measure, another interpretation is to use it as the expected log-loss, the needed number of bits necessary to "encode" the model's posterior label distribution.

With entropy we want to calculate the "disorder" of each posterior distribution $P_\theta(y \mid x)$ over the label class Y, obtaining a value $\in [0, 1]$. The higher the entropy is, the more the label of x is informative.

So, a learner may want to ask the instance with the highest entropy, which would represent the instance with the highest uncertainty on its real label, according to the model θ.

2.4 Searching Through the Hypothesis Space

In machine learning, a hypothesis is a model that attempts to generalize or explain the training data and make predictions on new data instances.

Given H the set of all possible hypotheses of a particular problem (*e.g.*, in case of the use of decision tree as a model, H is composed by all the possible combination of decision tree built upon the input space), $V \subseteq H$ is the version space.

The version space is the subset of hypotheses which are consistent with the training data (*i.e.*, those which label correctly the labelled data), so if we suppose that the underlying function to be learned is one and can be expressed by one of these hypotheses, we can set ourselves the objective to minimize V as fast as possible.

In the following sections will be illustrated the most used techniques to find the right hypothesis using the fewest possible queries.

2.4.1 Query by disagreement

Query by disagreement [2] is one of the earliest active learning algorithms that have the objective to reduce the version space. It assumes the stream-based selective sampling scenario and the main idea is that a learner maintains the working version space V of consistent hypotheses and for each instance x that arrives from the stream, the label of x is asked only if any two hypotheses in V disagree in its labeling. On the other hand, if all the hypotheses in V do agree, then x is discarded. The pseudo-algorithm is illustrated in Algorithm 2.

```
1 V ⊆ H is the set of consistent hypotheses
2 for t = 1, 2, ... do
3   receive an instance x from the stream
4   if h1(x) ≠ h2(x) for any h1, h2 ∈ V then
5     query for the label of x to the oracle
6     L = L ∪ (x, y)
7     V = {h : h(x') = y' for each (x', y') ∈ L)}
8   else
9     discard x
```

Algorithm 2: The query by disagreement algorithm (QBD)

One problem with this algorithm is that the version space V can be infinite, so we can not maintain V in memory during the for loop in line 2 of Algorithm 2.

In this case, a solution can be to not store V entirely, but to have $S, G \subseteq V$, where S is a set of hypotheses most "specific" (*i.e.*, hypotheses that label few instances as positive) and G is a set of hypotheses more "liberal" (*i.e.*, hypotheses that label more instances as positive than those in S). So, in Algorithm 2 we would take $h_1 \in S$ and $h_2 \in G$ for the comparison in line 4, and if $h_1(x) \neq h_2(x)$ and the label of x is positive, then the hypothesis in S are made more general, otherwise those in G are made more specific.

Unfortunately, according to [5], S and G can grow exponentially in the size of L, making their maintenance in memory expensive.

A different solution can be that one in which we have only two hypotheses $h_S, h_G \in V$, where h_S is the most "conservative" hypothesis trained with artificial cases with the $\ominus$ label added in L (*i.e.*, hypothesis that label few instances as positive), and h_G is the most "liberal" hypothesis trained with artificial cases with the $\oplus$ label added in L (*i.e.*, hypothesis that label more instances as positive than h_S). So, an instance x is queried only if these two hypotheses disagree. This approach is called SG-based QBD.

2.4.2 Query by committee

The query by committee algorithm [10] works in a pool-based scenario, and overcome cases where measure the disagreement among all the hypotheses in the version space is problematic, even if only two extremes like h_S and h_G are used.

The algorithm uses a "committee" C which can be built using various techniques. A basic one is to run a learning algorithm on a set of labeled instances L several times, to obtain the hypothesis for the committee.

Another approach is called *query by boosting* [3], where a boosting algorithm is used to construct a sequence of hypotheses that gradually become more precise by focusing them on the erroneous instances in L. More in detail, given the set L of instances, the algorithm builds a hypothesis using the labeled set $L^{'}$, created according to a probability distribution on L. For each iteration of the algorithm a hypothesis h is built upon a distribution D on L, then the error rate of h is calculated and the distribution D is properly modified to create a more "precise" hypothesis in the next iteration.

Query by bagging is another method that uses the bagging algorithm [1] to train a committee of ensembles based on re-samples of L with replacement. It operates by creating subsets of labeled instances $L_i \subseteq L$, where L is the set of instances, according to a fixed distribution on L.
This method is based on the idea that prediction error consists of the "bias" in the data, which is the estimation error caused by the input data size, and the "variance" which depends on the statistical variation existing in the data.
So, the method aims to minimize the variance component of the error.

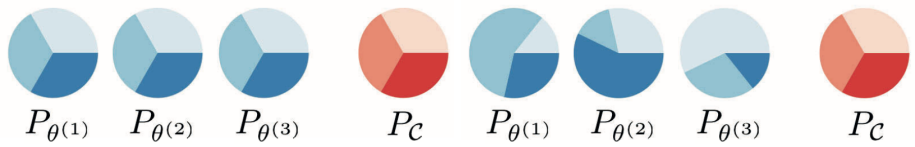
There isn't the right number of committee members to use, in literature a committee from five to fifteen hypotheses is quite common.

Once built the committee it is important to choose an heuristic for measuring the disagreement among the hypotheses in the committee. An example is a committee-based generalization of uncertainty sampling. For example, *vote entropy* is defined as:

$$x^*_{VE} = \underset{x}{\operatorname{argmax}} - \sum_y P_C(y \mid x) \log P_C(y \mid x)$$

where $P_C(y \mid x) = \frac{1}{|C|} \sum_{\theta \in C} P_\theta(y \mid x)$ is the average probability that y is the correct label according to the committee. Instead of entropy, least-confident and margin heuristics can be used as well.

A different type of measure of disagreement is based on Kullback-Leibler (KL) divergence [6], that is an information-theoric measure of the difference between two probability distribution. In this case, the interest is to calculate the average divergence of each committee member θ's prediction from that of the consensus C:



(a) uncertain but in agreement **(b)** uncertain and in disagreement

Figure 2.4. Examples of committee and consensus distributions. $P_{\theta^{(i)}}$ refers to the output distribution of the ith hypothesis, and P_C represents the consensus across all committee members. (image taken from [9])

$$x^*_{KL} = \underset{x}{\operatorname{argmax}} \frac{1}{|C|} \sum_{\theta \in C} KL(\ P_\theta(Y \mid x) \ ||\ P_C(Y \mid x))$$

where KL divergence is defined as:

$$KL(\ P_\theta(Y \mid x) \ ||\ P_C(Y \mid x)) = \sum_y P_\theta(y \mid x) \log \frac{P_\theta(y \mid x)}{P_C(y \mid x)}$$

An example of the difference between vote entropy and KL divergence is in Figure 2.4. In Figure 2.4a the probability distributions of the hypotheses $\theta^{(i)}$ are relatively uniform, so the consensus output distribution P_C is also uniform. In Figure 2.4b the probability distributions of the hypotheses are not uniform, where each distribution prefers a different label. But the consensus output distribution is uniform as in Figure 2.4a. Vote entropy only considers P_C and cannot distinguish among the two examples since they both have high entropy in the consensus. So, querying an instance with the distribution like in Figure 2.4a is not really the idea behind QBC, the consensus label is uncertain but all the committee agree that it is uncertain. KL divergence would query an instance with predictions like in Figure 2.4b, where the consensus is uncertain but the individual committee members vary widely in their predictions, which is more coherent with the notion of a disagreement.

2.5 Minimizing Expected Error and Variance

2.5.1 Expected error reduction

In a case where we need a machine that learns to make correct predictions in the future, we may want that each instance queried will reduce future errors as much as possible. Unfortunately, a learner doesn't know what its error will be after an answered query, so it is necessary to make decisions under uncertainty, trying to minimize the error as an expected value. The idea is to identify the best expected outcome v among the others for each action, with the following summation $\mathbb{E}[V] = \sum_v P(v)v$. In our case, we would choose the one with the lower expected future error. To compute the expected error, two probability distributions are needed.

One is the probability of the oracle's label y in answer to query x, and the second is the probability that the learner will make an error on some other instance $x^{'}$ once the answer is known. These probability distributions are not really known, but they can be approximated with the model's posterior distribution. Assuming that we have a large unlabeled data U, the utility measure is:

$$x^*_{ER} = \underset{x}{\operatorname{argmin}} \sum_{y} P_\theta(y \mid x) \left[\sum_{x' \in U} 1 - P_{\theta^+}(\hat{y} \mid x^{'}) \right]$$

where θ^+ refers to the new model re-trained using a new labeled set $L \cup (x, y)$. Here we want the instance x that, once obtained its label, our new model θ^+ expect to make fewer errors on labeling the remaining instances in the set of unlabeled instances U. So, the objective is to reduce the expected total number of incorrect predictions.

In most cases expected error reduction is significantly computationally expensive. Not only it requires estimating the expected future error over U for each query, but a new model must be re-trained for every possible labeling of every possible instance in the pool.

2.5.2 Variance reduction

To overcome the computational cost of the expected error reduction, the error can be inderictly reduced by minimizing the output variance, according to [4]. So, we can attempt to reduce the error by labelling instances that are expected to most reduce the model's output variance over the unlabaled instances U:

$$x^*_{VR} = \underset{x}{\operatorname{argmin}} \sum_{x' \in U} Var_{\theta^+}(Y \mid x^{'})$$

where Var_{θ^+} represent the model's output variance after it has been re-trained with $L \cup (x, y)$, where y is the expected label of x according to the model θ.

Chapter 3

Research project

During my internship, I worked in a research project coordinated by professor Toni Mancini, regarding the field of artificial intelligence. The project aims to design general methods to acquire, model, and formalize decision strategies in any kind of domain, creating a customizable knowledge acquisition software suitable for any working field.

3.1 Description

The project is focused on the healthcare domain, where the objective is to build general methods and software to model and formalize treatment strategies, and then later use the acquired knowledge to support the practitioners via Clinical Decision Support Systems (CDSSs).

From a high point of view, the processes in the project to acquire, handle, and validate decision strategies consist of:

- Learn core decision strategies from retrospective data from Electronic Health Records
- Acquire and continuously peer-review medical expert knowledge through a user-engaging serious game
- Improve the acquired strategies by actively learning from intelligently-chosen synthetic medical cases provided to experts for evaluation

Retrospective data (*e.g.*, hospital Electronic Health Records) are necessary to bootstrap the overall process, but they are not enough to correctly acquire decision strategies to treat medical cases. This is because retrospective data are usually limited, not treating all the types of cases present in a domain such as the healthcare one. Another reason is that often they do not represent all the factors of a patient that impacted the clinical decisions of the physician.

Given these problems, the need for a serious game combined with an active learning strategy is necessary to learn these strategies, it allows the learning algorithm to choose and learn any medical case it retains difficult or ambiguous to treat, depending on its knowledge acquired up to that moment.

The main methodologies used in the project are described below:

1. **Serious game:** Eliciting knowledge from domain experts using standard engineering approaches and methods scale poorly in the healthcare domain, where a physician is in great difficulty when asked to provide general rules, but is more confident in answering questions on actual cases.

   The objective of using a serious game is to engage the expert while the learning algorithm acquires its decision strategy providing cases as informative for it as possible.

   In this serious game, the expert medical staff will be involved as players, and they will be asked to virtually treat synthetic medical cases and to validate or refute the knowledge acquired by the system via other players.

2. **Active learning of clinical decision strategies:** The learning algorithm will acquire decision strategies exploiting both retrospectives clinical data and the serious gaming platform. Retrospective data will be used as a bootstrap to let our "learner" to acquire core decision strategies and allow it to synthesize cases for the serious game.

   The "learner" will create synthetic cases by intelligently generate a high-level model of a virtual patient, and will submit it to the players for the treatment, with the aim of discovering the strategies that the expert would take.

3. **Evaluation of the methods:** The methods will be evaluated in two areas of psychiatry: mood disorders during pregnancy and forensic psychiatric evaluations.

   The formalization of strategies and techniques in these two areas are fundamental, for example, in forensic psychiatric a lack of standardized procedures and the presence of unintentional or intentional biases would lead to an erroneous decision by the expert (*i.e.*, the statement of a "not guilty for reason of insanity" for a defendant).

Given the early stages of the project, the major work has been done in creating an active learning algorithm that would generate the most informative, and efficient for the learning process, case to be treated by an expert via the serious game. The implementation of the serious game and the evaluation of the methods in the two areas of psychiatric are left for the next stages of the project.

3.2 Serious game

Serious games are games whose primary purpose is not entertainment but they are usually used for educational/training purpose. In a serious game, the engagement of the player is still a fundamental requirement that allows the player to play while another purpose is achieved. An important characteristic of serious games is that the player acts in a "safe" simulated environment, making the experience of the player more comfortable in situations where, in real life, it may be faced with less calm, prone to actions not spontaneous.

The objective of our serious game is to obtain the decision strategies of the experts in an easy to use and engaging game.

Each player has a private account, with which he can start or resume a case that represents a virtual patient to treat. Each case is made up of visits. In each visit, the player can ask for the value, current or of a previous visit, of a *feature*. A *feature* is a variable that represents a characteristic of the virtual patient, and each *feature* has its own domain of values. When the player is ready to take a decision he declares one or more *actions* on the patient, that are represented as variables with values associated. Then, the player can decide to continue or terminate the visit. Furthermore, in a visit, the player can also ask what action was taken in a previous visit and what was its value. When the player queries for the actual value of a feature, the active learning algorithm responds to the query with the most informative value, creating intelligently a case important for its learning task. Once the player has received the value he can claim the value inconsistent with the previously observed *feature* and *actions* and declare a new constraint on the values. So, the learning algorithm has to provide values consistent with these constraints.

The pseudo-algorithm 3 describes the phases of the game.

One thing to add is that the query for the value of a feature in a previous visit, or for the action and its value, can be inconsistent, in the sense that the player may ask for the value of a visit precedent to the visits seen in the case (this phase is in line 11 of 3). In this case, an error message is returned and the player is asked to make another query, while our learning algorithm takes into account this request in its learning model.

To acquire, model, and represent the decision strategies of the experts, the decision tree model is used.

3.3 Decision tree

A decision tree is a learning model frequently used in machine learning. It is a tree structure that represents a function that takes as input a vector of features values and returns in output a decision.

Each internal node in the tree is a "test" on the value of a feature and each branch from the node is labeled with the possible values of the feature. The leaves of the tree are the final decisions taken, then, given a vector of features, the corresponding decision is that one on the leaf of the path of the tree root-leaf, where the vector satisfies all the tests present in the path.

Decision trees are usually used in machine learning when a set of examples is available and the objective is to learn a function that approximates as much as possible the "real" function that controls the domain in which we are trying to learn.

An important property of the decision trees is that it is a model easily readable for a human and the reason for a decision output is understandable compared to other types of model in machine learning (*e.g.*, Neural Networks).

3.4 Active learning of clinical decision strategies

Active learning has a main role in the serious game. The overall objective is to learn, as much as possible from the cases, the various decision strategies of the

```
1  while the player doesn't exit from the game do
2      The player start or resume a case
3      while the case is not finished or the player doesn't exit from the case
        do
4          if the player is ready to take an action then
5              Declare one or more actions and their values
6              if the player want to terminate the visit then
7                  Terminate the visit
8              end
9          else
10             if the player asks for the value of a feature/action in a
                previous visit then
11                 while the query is inconsitent do
12                     Return an error message
13                     Ask to the player to make another query
14                 end
15                 Respond to the query with the value, consistent with
                    the constraints declared by the player
16             else if the player asks for the current value of a feature then
17                 Return a value consistent with the constraints
18                 while the returned value is inconsistent for the player
                    do
19                     The player declares a constraint
20                     The set of constraints is updated
21                     Return a new value consistent with the constraints
22                 end
23             end
24         end
25     end
26 end
```

Algorithm 3: Pseudo-algorithm of the serious game

experts. Given the infeasibility of learning from all the possible cases, our learning algorithm has to choose the most informative cases to propose to the experts for the treatment.

The serious game tries to simulate a realistic context, where the physician is faced with multiple cases simultaneously so that at the beginning of each visit, the doctor will remember as little as possible about the facing case.

The main difficulty in acquiring and modeling these strategies is that physicians do not always operate in the same way in each medical case they treat, where they may respond differently to cases of the same type, making the formalization of the decision-strategy difficult. Furthermore, in a decision tree optic, the sequence of queries asked by the physician does not always follow the same order, where, for the same type of visits, queries are asked in a different order, with the problem of recognizing what type of strategy it is.

As the project is in the early stages, the active learning method studied is based on some preconditions:

- Doctors are not noisy.
- All the values returned by the algorithm are consistent with the constraints declared by the doctor.
- The doctor can declare only one action, and when he does it the visit is concluded.

The next chapter will describe the active learning algorithm of the serious game used to return, at each query of the physician, the most informative value.

Chapter 4

Methodology

4.1 Introduction

The objective of the active learning method studied is to propose to an oracle the most informative case and request for its labelling. For the case is meant any type of problem, which can be solved using a strategy (*e.g.*, in the healthcare domain, the terms can be compared to the medical case).

The peculiarity of this method is that the "learner" does not have a set of unlabeled cases, among which he can choose which one to query to the oracle and then obtain the correct strategy, but he has to generate the case, building it from what he has seen up to that moment.

Query synthesis (*i.e.*, the generating of an unlabeled instance by the learner) is a scenario in active learning not very widespread.

Most of the solutions in literature, not always propose a full query synthesis approach, proposing a solution between the query synthesis and the pool-based sampling, as described for example in *e.g.*, [12]. This is because, in certain types of domain, is very difficult guarantees that the learner will always generate unlabeled instances understandable for a human oracle.

The proposed algorithm will be tested in the context of the serious game.

In the serious game, the domain of a case can be seen as a set of variables,*i.e.*, features and actions, where the features describe the characteristic of the case and the actions describe the decisions made after checking the value of a certain subset of features. Given that the features and actions are variables, they have also associated a domain. This domain can be numeric, enumerate and boolean, allowing to represent every aspect of a case.

Given this point of view, the decision strategy of an oracle can be viewed as a decision tree, where each non-leaf node represents a feature and its children represent the values of its domain and the leaves of tree correspond to the actions.

So, the objective of the active learning algorithm is to learn as fast as possible the decision tree of the oracle.

The main challenge is, once obtained a query by the oracle, which value the active learning algorithm has to return?

4.2 The informativeness of a value

Given the domain of the problem described in the previous section (*i.e.*, a case made up of features and actions), an analysis of the characteristics of a strategy can be made.

For example, given a feature f with a domain of 5 numbers, ranging from 1 to 5, one may think that values near between them in the domain may have the same outcomes.

e.g., If an action a is taken for $f = 1$ and $f = 3$, with high probability, a will be executed even for $f = 2$.

This is valid also for enumerate domains, if order on the values is established.

Starting from this idea, we could compute the informativeness of a value by the neighbour values in the domain. This level of informativeness can be achieved declaring a probability distribution, for each value of the domain, on the possible decisions for that value (*i.e.*, actions or the query of another feature).

Given these probabilities distribution on the values of a domain, we can compute the level of "disorder" and "uncertainty" of these distributions, using a suitable variable.

4.3 Entropy

In information theory, the entropy of a random variable is a value of "information" or "uncertainty" on the variable's possible outcomes. Given an event E, its information content is a function which decreases as the probability of the event E increases, namely $I(E) = -\log_2(p(E))$. Entropy computes the expected amount of information contained on the outcomes of a random variable.

Formally, given a discrete random variable X, with possible outcomes $x_1, \ldots, x_n$, the entropy in bits of X is:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)$$

The basic idea is that the information to know the outcome of an event E is higher when E is less likely to happen. In a random variable, a high entropy represents a high level of uncertainty on the outcomes of the random variable.

4.4 Probabilities

Given the value of the domain of a feature, the entropy of its probability distribution gives us an information value in knowing the decision taken by the oracle for the value.

So, when the oracle queries for the value of a feature, our active learning algorithm has to compute the probabilities distribution of each value of the domain of the feature and then return the value with the highest entropy on its probability distribution.

To compute the probability distribution of a value it is necessary to define how is defined this probability and which are the elements in the probability distribution.

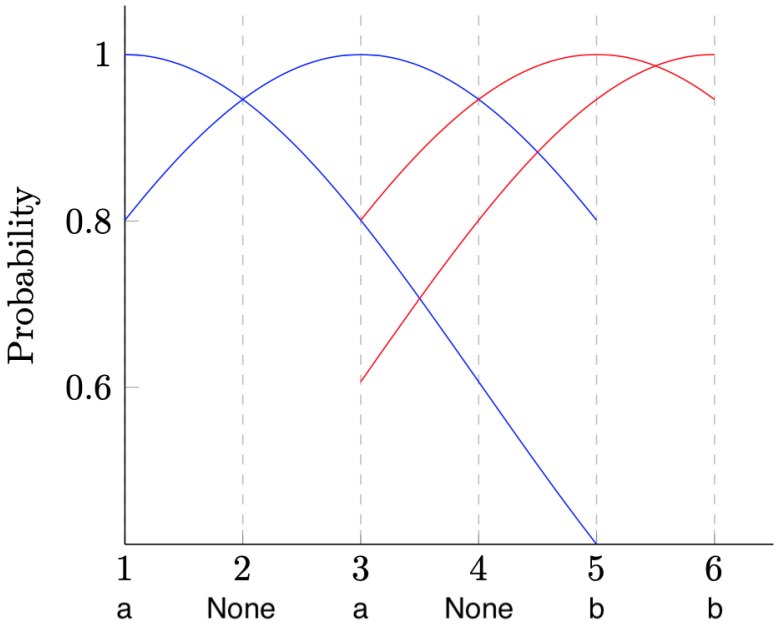


Figure 4.1. Example of probabilities in a domain of a feature

For the computation of the probabilities, a symmetric bell-shaped function is necessary, as given a value of the feature, it is mainly conditioned by the near values around it. The symmetric bell-shaped function used is the Gaussian function:

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$$

The parameter a is the height of the curve's peak, in our case $a = 1$ as the probability of a decision in its own position is equals to 1, b is the centre of the peak and c controls the width of the bell.

To compute the probabilities, each decision taken in the domain of feature has its own bell, and then, for each value v without a decision in the same domain, the probability distribution of v is computed based on the values of the bells on its domain.

Example 4.1 figures a domain of a feature. a and b are actions, and the "None" values represent values never been proposed to the oracle. From now on the term label will be used to refers to features and actions.

In a situation like this, we are interested in the value of the entropy of the values 2 and 4. To compute these entropies, we define the probability distribution of the labels that are near the values. These labels are chosen only if they are the first label encountered by the "None" value on its left side or on its right side, we define these values "consistent labels".

For example, the consistent labels of the value 4 are the labels a and b, as starting from the value 4, they are the first label encountered. On the other hand, the consistent label of the value 2 is only a, this is because on the right of the value 2, the first label encountered is a.

So, to compute the probability distribution of each "None" value, we need to compute the probabilities of the consistent values. For each label, its probability in a "None" node is the probability that at least one "consistent node" of the label is "realized" in the "None" node.

For example, in the upper graph, the probability of the label a in the "None" node 2 is:

$$P(a \mid x = 2) = 1 - (1 - P(1 = a))(1 - P(3 = a))$$

where $P(1 = a)$ and $P(3 = a)$ are the probabilities computed by the bell function of the label a of the value 1 and 3 respectively. In this case the probability of b is 0, as it is not the first label encountered on the left or on the right of the "None" node 2.

A hyperparameter Z is added in the probability distribution of a "None" node to represent the probability that the "None" node will be labeled with a different label than those of the neighbours computed in the distribution.

Once computed the probabilities of the labels of the "consistent values" and of Z a normalization is made, adding up all the probabilities and dividing them by the total obtained.

Once obtained the normalized distribution, the entropy is computed for each "None" node, obtaining a high value for nodes that are more difficult to predict for their label and therefore the most informative values.

4.5 Heuristic

Each node in the tree has a utility function u associated. The u of a node labeled with a feature is the maximum entropy of the "None" nodes of its domain. In case that among the "None" nodes children of a node labeled with a feature, there is a node labeled with a feature that has the u higher than the maximum entropy of the "None" nodes, the u of the child node labeled with a feature is passed to the u of the parent. The nodes labeled with an action have $u = 0$.

Then, when a physician query for the value of a feature, the value with the highest u is returned. When a new feature is added to a node and then in the domain, there are only "None" nodes, the value returned of that feature is the one in the middle of the domain. This because returning the value in the middle in the next visit in the same node I will have the "None" nodes with the highest entropy at the extremes of the domain, performing a high level of exploration in the domain of the feature queried.

When the query of the physician is inconsistent (*i.e.*, when is requested the value of a feature or action in a visit happened before the first visit of the case) a node is still made but a branch with the value N.A. (Not Assigned) is created, representing the fact that the physician will make a certain type of visit when facing this problem.

Finally, the u of the node representing the request for the value of a feature or action in a previous visit, is computed representing the expected value:

$$u = \mathbb{E}_{\lambda}(min_{v}\ (max_{v' \neq v}(u(v^{'}, v, \lambda))))$$

where λ is the set of all the features, v and $v^{'}$ are the values in the domain of a certain feature in λ, and $u(v^{'}, v, \lambda)$ is the function that compute the u of $v^{'}$ given v. So, the u of the node is the expected value on the minimum entropies of each feature $f \in \lambda$, which are obtained among the maximum entropies of each value v in the domain of f.

Chapter 5

Evaluation

5.1 Testing software

For the evaluation of the performance of the method proposed in chapter four, I implemented a testing software written in Python.

The main objective is to test the efficiency of the active learning algorithm with respect to a "random" learning algorithm, which it has not a strategy and returns random values for each query received.

For the evaluation, the context of the testing process has to be similar to the one in the serious game, then the generation of an oracle, and of its decision tree, that tries to replicate a "plausible" decision strategy, is necessary.

The generator of random trees has been implemented in the test. It builds a decision tree based on some parameters that are passed in input to the software (*e.g.*, number of features/actions, etc...). This is a crucial step, considering that the decision tree generated declares the complexity of strategy that it represents. A very complex strategy is characterized by a decision tree that does not have homogeneity in the decisions, based on values in the domain of the features (*i.e.*, for certain consequent values of a feature with a numeric domain, the same action is done).

The procedures in the test are very similar to those of the serious game. Once generated an oracle and its decision tree, a case is started. A case is basically a set of "paths" (*e.g.*, in a healthcare domain they can be described as visits on a patient) of the oracle's decision tree, each path of a case represents a set of queries made by the oracle to the learner, concluded with an action (*i.e.*, a leaf in the decision tree of the oracle).

From now on we will use "label" as a reference for feature or action.

The testing software takes in input the following parameters:

- The number of features to use
- The minimum and maximum size of the domains of the features
- The number of actions to use
- The minimum and maximum size of the domains of the actions

- The rate, on the number of leaves of the oracle's decision tree, of paths to perform
- The probability value of the probability distributions on the size of the expansion of a label on the neighbour values of the domain of a feature (this will be described more in detail later)

The testing software is mainly divided into two modules, the "oracle module" and the "learner module".

In the next sections, these modules will be described.

5.2 Oracle module

The oracle module contains all the functionalities necessary to generate a random decision tree and to simulate the exchange of queries-responses with the learners.

When a random tree is generated, first of all, the sets of features and actions are created.

The domains of the labels used in the tests are numeric. Despite the presence of only numeric domains, this does not influence the performance of the method in contexts with other types of domains, as the same heuristic principles described in the previous chapter are used as well with the other types of domain.

Furthermore, the queries regarding the values of labels in previous visits are not handled for a reason of simplification. Adding this concept to the testing software will not affect the results.

The size of the sets is in agreement with that is given in input. Then, when the size of the sets has to be chosen, the binomial distribution is used.

In probability theory, the binomial distribution is a discrete probability distribution that represents the number of "successes" in a Bernoulli process.

A Bernoulli process is usually associated with a coin toss sequence, where the number of possible outcomes is only two, success or failure. So, the binomial distribution describes the probabilities of success of exactly k times on N conditionally independent trials.

Formally, the binomial distribution is defined as $B(N, p)$, where:

- N is the number of the trials
- p is the probability of success

The probability distribution is computed as $P(k) = \binom{N}{k} p^k (1-p)^{n-k}$, where k is the number of successes, and the number of all the possible combinations is computed with the binomial coefficient $\binom{N}{k} = \frac{n!}{k!(n-k)!}$.

A binomial distribution, with $p = 0,15$ and N equals to the number of elements in the range provided in input, is used to choose the size of the domains of the labels, this because, setting $p = 0,15$, a symmetric bell shaped probability distribution is obtained, favouring the intermediate values of the ranges given in input for the size of the domains.

Created the sets of the features and actions, the tree is started to be built. For the root, a feature has to be chosen, then it is extracted from the set of features using a uniform distribution.

Once created the root, a branch is made for each value in its domain. Starting from the first value, a choice is made to whether to select a feature or an action.

This choice is made following a "dynamic" probability. At choices made in a higher level of the tree, this dynamic probability favours the selection of a feature rather than an action. Instead, when the choice is made in lower levels of the tree (the lower limit is given by the number of features, as, in the longest path of the tree, all the features are present, without repetition), an action is selected with a very high probability. This guarantee a "consistency" with the structure of a real decision tree, where the actions are more likely to be taken after a large number of queries.

Two scenarios can happen:

- If a feature f is selected, the size of the "expansion" of f on the neighbour values in the domain of the root is computed. This is an important factor, as this determinates the complexity of the strategy, given that with a lower size of expansion for the labels, an inference process on the label of the values in the domain is complicated.

  For the size of the "expansion", it is intended the number of times that f is propagated in the neighbour nodes of the values of the root.

  For example, if the domain of the root is made up of 5 values and f is selected for the value 1, with the size of the expansion equals to 2, f is assigned also for the value 2 and 3.

  The size of the expansion is selected using a binomial distribution with a parameter p given in input to the software. This parameter is fundamental to declare the complexity of the generated decision tree, as for lower values of p, small-sized expansions are made, filling the domain of a feature of a high number of different labels.

  Finally, the steps are repeated starting from the choice to whether to select a feature or an action.

- Instead, if an action is selected, its value is selected randomly from its domain. Then, the same process used in the above scenario to select the size of the "expansion" is made.

Once the decision tree is generated, a "simulation" of the serious game is performed.

The oracle creates a new case and then starts to make queries to the learner, following its decision tree after the responses to the queries. This process continues until the oracle reaches a leaf of its, returning the action associated with the node to the learner.

For each case, several "paths" are performed. By paths, we mean the number of sequence queries-action done in a case. This factor affects the accuracy of the final tree obtained by the learner.

5.3 Learner module

In the learner module is implemented the method described in chapter four.

A learner has to provide the most informative value at each query of the oracle and then learn its strategy as accurately as possible using the minimum possible number of paths. The sets of features and actions generated in the oracle module are shared with the learner module.

Two types of learner are implemented for the testing: the active learner and the random learner.

The active learner has initially an empty decision tree. Then, when the oracle starts with the first query, the learner returns the value in the middle of the numeric domain. This is done because it guarantees a maximum entropy on the borders of the domain for the second value that will be chosen in the next path. In fact, choosing a suitable value for the Z hyperparameter, we can choose which part of the domain has the highest entropy during this initial phase. In the tests, Z is equals to the value of the gaussian function, with the bell centred in the middle of the domain and the value computed on the last value of the domain, this guarantees a higher entropy on the extremes values in the domain.

This step is done also when a new node is created in the tree and its label is a feature.

Instead, when the node associated with a feature queried by the oracle exists, the value with the highest heuristic u is returned.

Once terminated a path, and an action is obtained, for each node, from the leaf to the path, the heuristic u is computed.

Given a node x, for each value t children of x with the label "None" (*i.e.*, not yet proposed to the oracle), the probability of each "consistent label" (*i.e.*, label differents from "None" that appears as first on the left or on the right of t) is computed.

Given Y the set of the positions, on the domain of x, of a "consistent label" c, the probability of c in t is:

$$1 - \prod_{y \in Y} (1 - P(y))$$

where $P(y)$ is equals to the gaussian function centered in y and computed in t.

Given the gaussian function:

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$$

The parameters used for the gaussian function are:

- $a = 1$, as the probability of a label in its position is equals to 1
- $b =$ position of the "consistent label" y
- $c = len^{1/3}$, where len is the lenght of the domain of x, this is an arbitrary value as it does not affect performance much

Once computed these probabilities, the normalization is made using the sum of the probabilities of each "consistent label" of t and of the Z hyperparameter, and then the entropy of t is computed on this probabilities.

Computed the entropies of each value in the domain of x, the maximum is taken, and if exists a feature label in the children nodes of x with a higher u, that one is passed to the u of x, otherwise, the maximum entropy becomes the u of x.

Once the number of paths is terminated, a final decision tree of the learner is made by choosing for each "None" children of each node of the tree, the label with the highest probability, computed as described above.

For the random learner, the proceedings is basically the same, with the only difference that instead of returning the value with the highest u at each query of the oracle, a random value is returned, among the values without an action as label. The final tree of the random learner is obtained with the same procedure as the active learner.

5.4 Results

The tests have been done with these parameter given in input to the random generator of the decision tree:

- Number of features = 4
- Minimum size of the domain of the features = 10
- Maximum size of the domain of the features = 30
- Number of actions = 4
- Minimum size of the domain of the actions = 1
- Maximum size of the domain of the actions = 10

The test is executed on 3 types of tree generated:

- A decision tree representing a complex strategy, where the binomial distribution on the number of repetitions for a single choice has the probability of success $p = 0,10$. The number of the nodes in these types of trees generated varies from 150 to 800, and the number of actions (leaves of the tree) varies from 150 to 270.
- A decision tree representing a medium complexity strategy, where the binomial distribution on the number of repetitions for a single choice has the probability of success $p = 0,20$. The number of the nodes in these types of trees generated varies from 400 to 1500, and the number of actions (leaves of the tree) varies from 150 to 550.
- A decision tree representing an easy strategy to learn, where the binomial distribution on the number of repetitions for a single choice has the probability of success $p = 0,30$. The number of the nodes in these types of trees generated varies from 700 to 19000, and the number of actions (leaves of the tree) varies from 200 to 500.

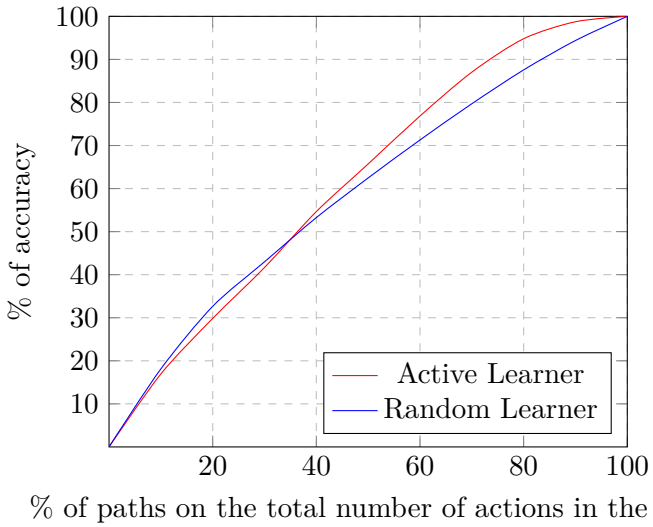


Figure 5.1. Test using a decision tree with high complexity

The Key Performance Indicator chosen for the test is the mean of the accuracy (*i.e.*, the number of correct nodes with respect to the tree of the oracle) of the active learner and random learner decision tree.

For each type of decision tree generated, 9 cases are performed, increasing at each case the rate, on the number of leaves of the oracle's decision tree, of paths to perform.

The below graphs shows the results of the tests.

In the graph 5.1 is shown the results of the tests, using an oracle decision tree with high complexity. It is aspected that in cases where the percentage of paths are lower, the random learner performs better. This happens because the random learner is more inclined to do exploration on the values of the features, instead of the active learner which tries first to find the boundaries of the "sequences" of labels in the domain of the features.

In fact, increasing the number of paths, the active learner performs better, achieving a 90% of accuracy with the 70% of paths.

In the graph 5.2 is shown the results of the tests, using an oracle decision tree with medium complexity. Also, in this case, the random learner initially performs better than the active learner. Further, the active learner has arrived at 85% of accuracy with the 60% of paths.

The final test 5.3 uses an oracle decision tree with low complexity. This is the ideal case for the active learner, in fact with only the 40% of visits he can reach the 71% of accuracy.

The source code of the testing software is available to the GitHub repo.

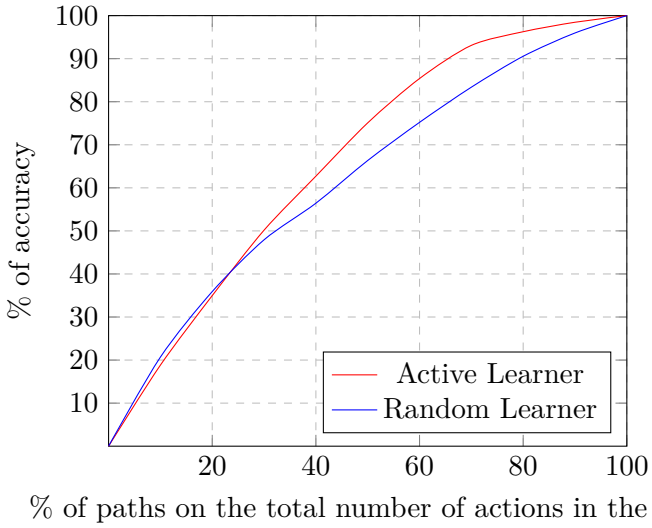


Figure 5.2. Test using a decision tree with medium complexity

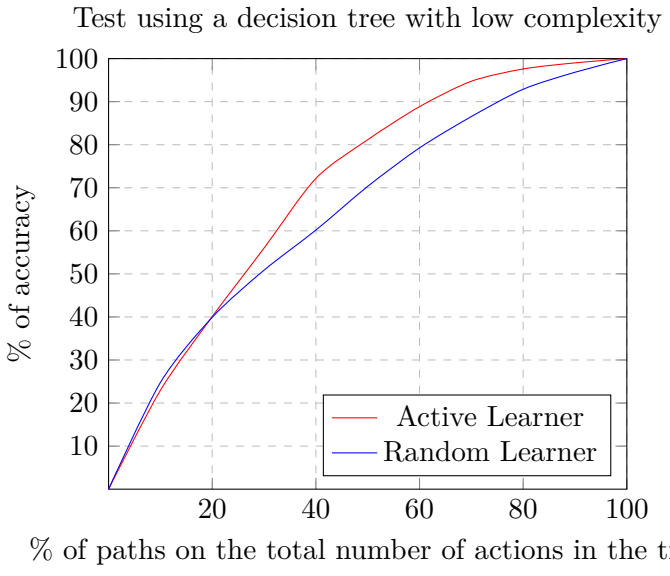


Figure 5.3. Test using a decision tree with low complexity

Chapter 6

Backend

During my internship, I also designed a RESTful backend for the acquisition of retrospective data in the field of mood disorders during pregnancy. The main functionalities of the backend are those related to the insertion of medical records in the form of features and actions so that the data are ready for the bootstrap of the active learning algorithm. Each physician has an account associated, with which he can create a medical case. Each medical case is composed of states, which represents the medical visits made on the patient.

In each state, the user has to be able to define new features and actions, and, in a state, assign values of the feature and declare actions.

The backend has been implemented using the PostgreSQL database and the Python micro-framework Flask:

- PostgreSQL is an open source object-relational database system that uses and extends the SQL language. For the communication with the python APIs, the library Psycopg2 is used.
- For the RESTful APIs the micro-framework Flask is used.

REST (Representational State Transfer) is an architectural style in which each resource on the web is represented by an URL. Each resource can have multiple URLs associated, but each URL represents only a resource. REST exploit the HTTP methods to perform CRUD operations on the resources.

Flask is defined as a micro-framework because it provides few functionalities but essentials, allowing to easily create an API RESTful in python.

The documentation related to the design of the backed is reported in the appendices.

The docker is available on the GitHub repo.

Appendices

Appendix A

System Design Document

Virtual Doctor: System Design Document

Lorenzo Levantesi

Version of 24/11/2020 20:38:57+01:00

Contents

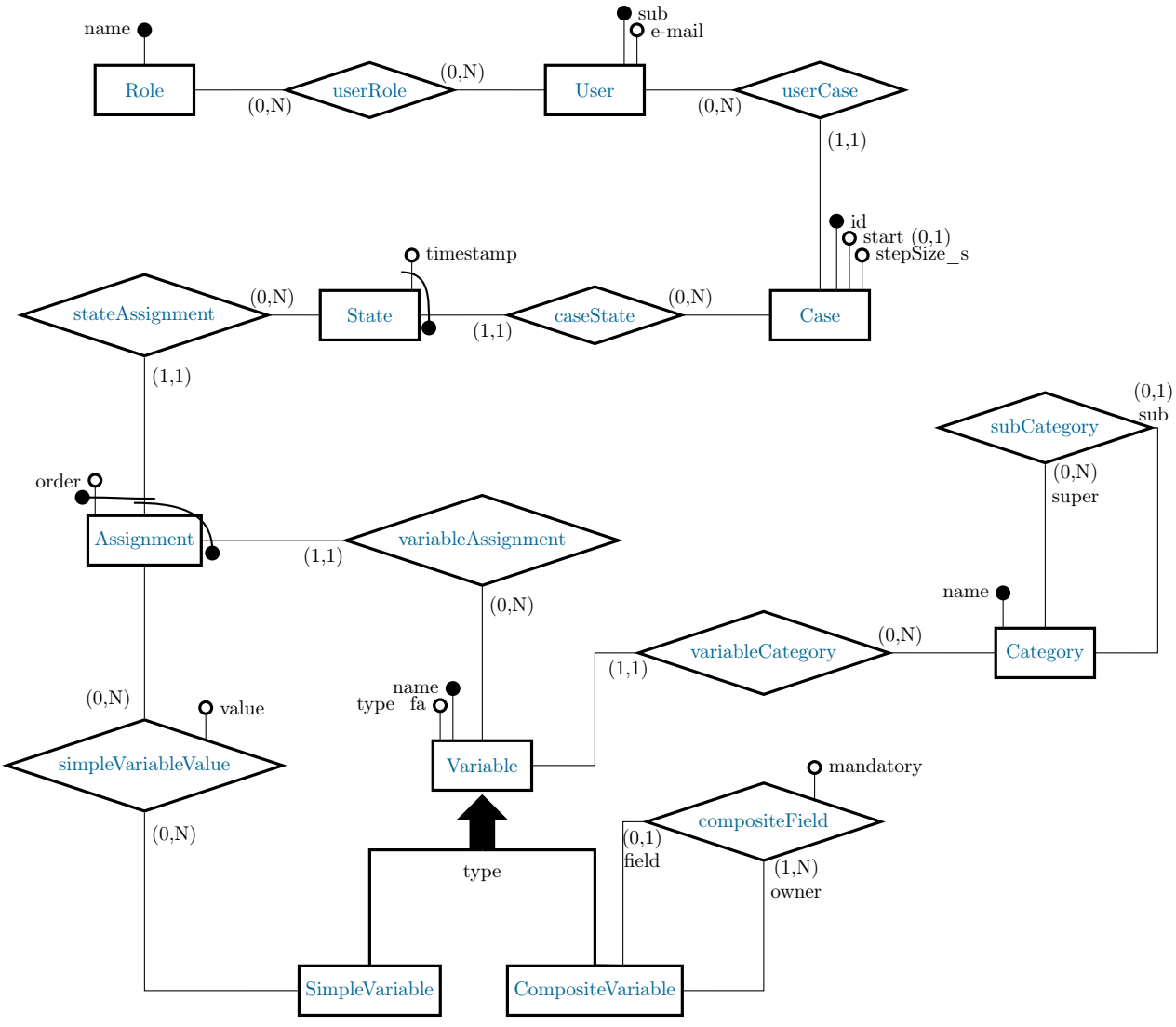
Part I

Conceptual Analysis

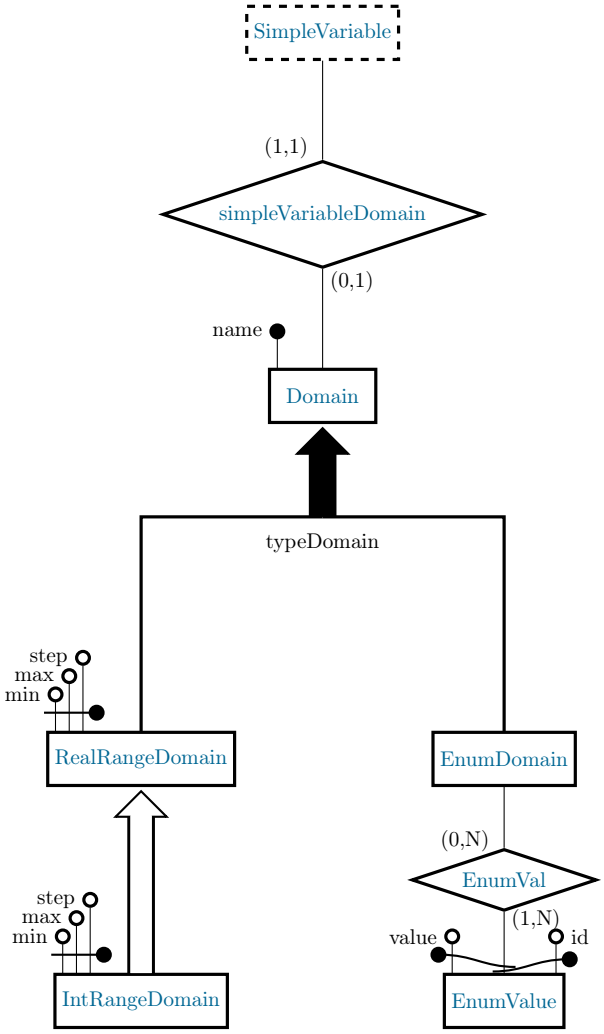
Chapter 1

Conceptual Analysis of Data - Entity Relationship Model

1.1 Entity Relationship Diagram: "Case"



1.2 Entity Relationship Diagram: "Domain"



1.3 Data Dictionary

Entity User
Each instance of this entity represents a user.

attribute	domain	cardinality	description
e-mail	email		The email address of the user
sub	integer		The unique id of the user

Entity Role
Each instance of this entity represents a user role.

attribute	domain	cardinality	description
name	string		The name of the role

Entity Case
Each instance of this entity represents a case.

attribute	domain	cardinality	description
stepSize_s	integer>0		The unit in seconds of the case
start	timestamp	(0,1)	The timestamp of the start of the case
id	integer>0		The unique id of the case

Entity State
Each instance of this entity represents a state.

attribute	domain	cardinality	description
timestamp	integer≥0		The number of case's stepsizes since the start

Entity Assignment
Each instance of this entity represents an assignment.

attribute	domain	cardinality	description
order	integer>0		The order number of the assignment in the state

Entity Variable
Each instance of this entity represents a variable.

attribute	domain	cardinality	description
type_fa	{feature, action}		the type of the variable
name	string		The name of the variable

Entity Category
Each instance of this entity represents a category.

attribute	domain	cardinality	description
name	string		The name of the category

Entity SimpleVariable
Each instance of this entity represents a simple variable.
Attributes: None

Entity CompositeVariable
Each instance of this entity represents a simple variable.
Attributes: None

Entity Domain
Each instance of this entity represents a domain.

attribute	domain	cardinality	description
name	string		The name of the domain

Entity IntRangeDomain
Each instance of this entity represents an integer domain.

attribute	domain	cardinality	description
min	integer		The minimum value of the domain
max	integer		The maximum value of the domain
step	integer > 0		The step value in the range of the domain

Constraints:

[C.IntRangeDomain.minmax] $min < max$.

Formally:

$$\forall ird, min, max\ IntRangeDomain(ird) \wedge min(ird, min) \wedge max(ird, max) \rightarrow min < max$$

Entity RealRangeDomain
Each instance of this entity represents a real domain.

attribute	domain	cardinality	description
min	real		The minimum value of the domain
max	real		The maximum value of the domain
step	real > 0		The step value in the range of the domain

Constraints:

[C.RealRangeDomain.minmax] $min < max$.

Formally:

$$\forall rrd, min, max\ RealRangeDomain(rrd) \wedge min(rrd, min) \wedge max(rrd, max) \rightarrow min < max$$

Entity EnumDomain
Each instance of this entity represents an enumerate domain.
Attributes: None

Entity EnumValue
Each instance of this entity represents an enumerate value.

attribute	domain	cardinality	description
value	string		A value of the domain
id	integer > 0		The id of the value

Relationship userRole
Attributes: None

Relationship userCase
Attributes: None

Relationship caseState
Attributes: None

Relationship stateAssignment
Attributes: None

Relationship variableAssignment
Attributes: None

Constraints:

[C.variableAssignment.variable.mandatory] If a composite variable cv is assigned, each mandatory field has a value.
mandatory_tree(field, root) is a function that returns "true" if the variable field is mandatory when its composite variable root is assigned.

Formally:

$$\forall a, cv, sv\ CompositeVariable(cv) \wedge variableAssignment(a, cv) \wedge SimpleVariable(sv) \wedge mandatory\_tree(cv, sv) = \text{“}true\text{”} \rightarrow \exists val\ simpleVariableValue(a, sv, val)$$

[C.variableAssignment.variable.composite] If a composite variable cv is assigned in a and a simple variable sv has a value in a, then sv must be contained in cv.
tree_root_composite(field, composite) is a function that returns "true" if field has as root composite in the composite fields tree.

Formally:

$$\forall a, cv, sv, val\ CompositeVariable(cv) \wedge SimpleVariable(sv) \wedge variableAssignment(a, cv) \wedge simpleVariableValue(a, sv, val) \rightarrow \text{tree\_root\_composite(sv, cv)} = \text{“true”}$$

[C.variableAssignment.variable.simple] If a simple variable sv is assigned in a, then sv is the only simple varibale that has a value in a.

Formally:

$$\forall a, sv, sv', val\ SimpleVariable(sv) \wedge SimpleVariable(sv') \wedge variableAssignment(a, sv) \wedge simpleVariableValue(a, sv', val) \rightarrow \text{sv} = \text{sv'}$$

Relationship subCategory
Attributes: None

Constraints:

[C.subCategory.subcategories.different] A category can't contain itself in its subcategories.
path_category_tree(sub, super) is a function that returns "true" if the subcategory sub is contained in the category tree of super.

Formally:

$$\forall c, c'\ Category(c) \wedge \text{path\_category\_tree(c',c)} = \text{“true”} \rightarrow\ c' \neq c$$

Relationship variableCategory
Attributes: None

Relationship compositeField

attribute	domain	cardinality	description
mandatory	boolean		If the field's value is mandatory

Constraints:

[C.compositeField.field.type] A composite variable can contain only variables of its type.
path_composite_tree(field, root) is a function that returns "true" if the variable field is contained in the composite variable tree of root.

Formally:

$$\forall cv, f, t\ CompositeVariable(cv) \wedge Variable(f) \wedge \text{path\_composite\_tree(f,cv)} = \text{“true”} \wedge type\_fa(cv, t) \rightarrow\ type\_fa(f, t)$$

[C.compositeField.field.different] A composite can't contain itself.

Formally:

$$\forall cv, f\ CompositeVariable(cv) \wedge path\_composite\_tree(f, cv) \rightarrow\ f \neq cv$$

Relationship simpleVariableValue

attribute	domain	cardinality	description
value	value		Value of the simple variable in the assignment

Constraints:

[C.simpleVariableValue.domain.value] For each value val assigned to a simple variable `sv`, val is of the same domain of sv. value_domain(val, dom) is a function that returns "true" if val is of the dom domain.

Formally:

$$\forall a, sv, val, dom\ [Assignment(a) \wedge SimpleVariable(sv) \wedge simpleVariableValue(a, sv, val) \wedge simpleVariableDomain(sv, dom)]\ \rightarrow\ [\text{value\_domain(val,dom)} = \text{“true”}]$$

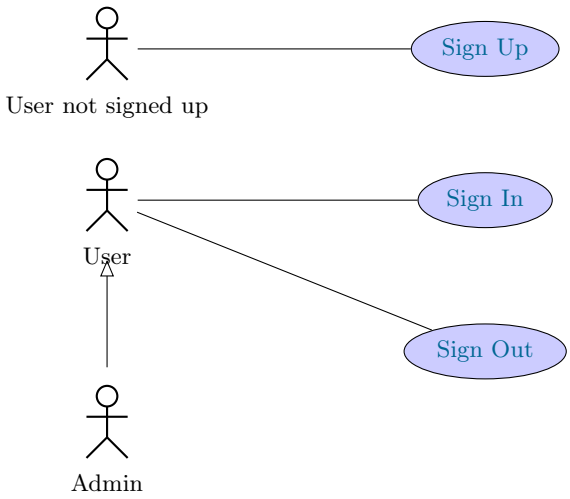
Relationship simpleVariableDomain
Attributes: None

Relationship EnumVal
Attributes: None

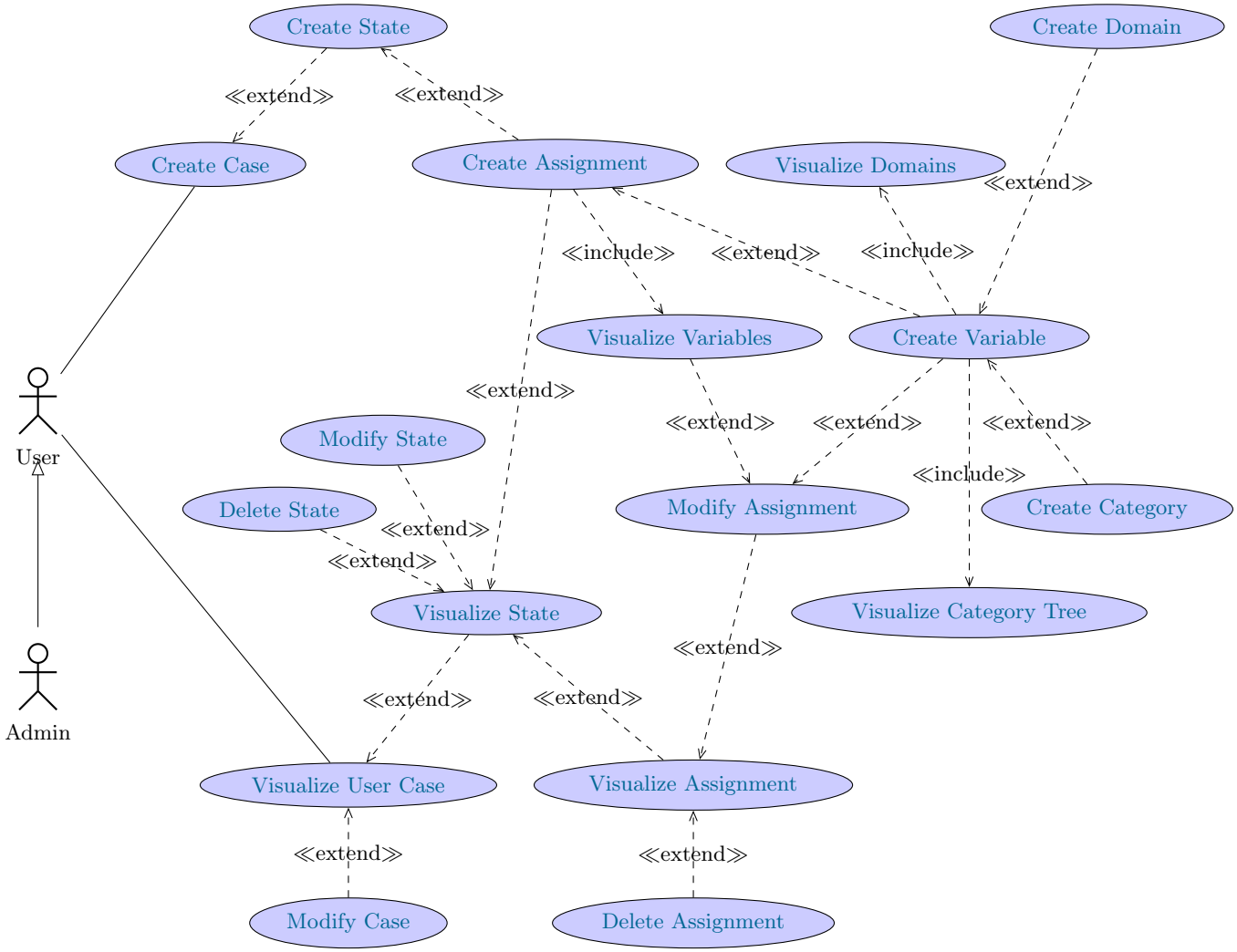
Chapter 2

Conceptual Analysis of Functionalities - Use Case Model

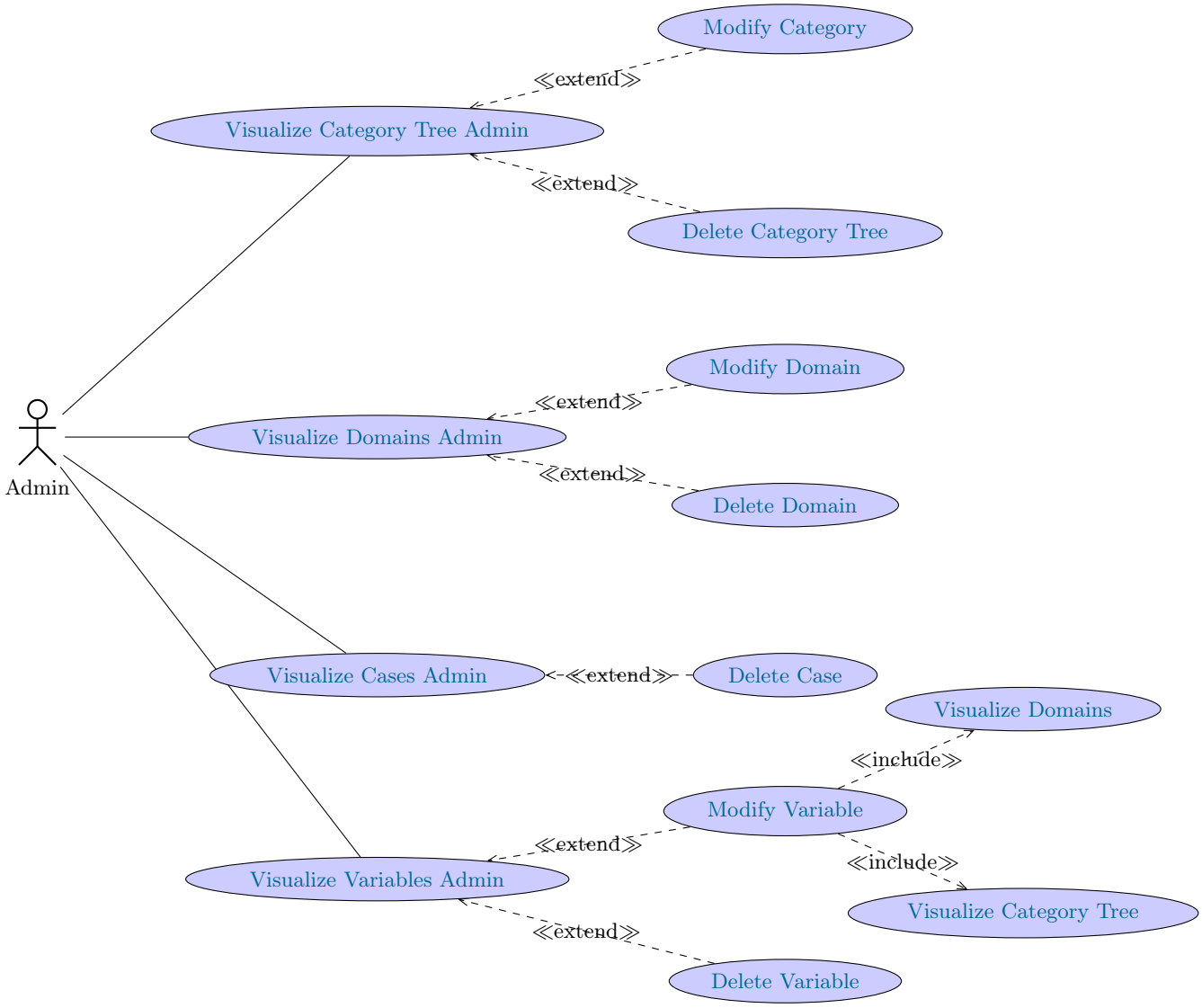
2.1 Use Case Diagram: "Access"



2.2 Use Case Diagram: "User"



2.3 Use Case Diagram: "Admin"



2.4 Specification: "Access"

2.4.1 Use Case Specification "Sign Up"

ID	1
Name	Sign Up
Actors	User not signed up
Arguments	email : string, password : string
Preconditions	The user is visualizing a login page.
Main Flow	1. The user requests to register into the system; 2. the system starts the signup procedure; 3. the user gives to the system his credentials; 4. if the user is not already registered, then the system registers the user.
Postconditions	The user is correctly registered into the system.

2.4.2 Use Case Specification "Sign In"

ID	2
Name	Sign In
Actors	User
Arguments	email : string, password : string
Preconditions	The user is visualizing a login page.
Main Flow	1. The user requests to sign in into the system; 2. the system starts the sign in procedure; 3. the user gives to the system his credentials; 4. if the User is registered and it is approved, then the user logs in into the system as a User or an Admin (according to his user type). Otherwise, the system rejects the sign in procedure.
Postconditions	The user is authenticated.

2.4.3 Use Case Specification "Sign Out"

ID	3
Name	Sign Out
Actors	User
Arguments	
Preconditions	The user is signed in.
Main Flow	1. The user requests to sign out of the system; 2. the system log out the user;
Postconditions	The user is signed out.

2.5 Specification: "User"

2.5.1 Use Case Specification "Create Case"

ID	4
Name	Create Case
Actors	User
Arguments	start : timestamp, stepsize : integer > 0
Preconditions	The application is ready to use.
Main Flow	1. The user requests to create a case; 2. the system requests to insert the following data: a) start: a non mandatory timestamp of the start of the case b) stepsize: the number of seconds that pass from a state to another 3. the user inserts the data; 4. if the data are correct the system creates the case, otherwise, it returns an error message
Postconditions	The case is created. Otherwise, the operation is canceled

2.5.2 Use Case Specification "Visualize User Case"

ID	5
Name	Visualize User Case
Actors	User
Arguments	case : integer > 0
Preconditions	The application is ready to use.
Main Flow	1. The user requests to view its cases; 2. the system shows the cases of the user; 3. the user chooses one case; 4. the system shows the data of the case.
Postconditions	The system is showing the case.

2.5.3 Use Case Specification "Modify Case"

ID	6
Name	Modify Case
Actors	User
Arguments	case : integer > 0, start : timestamp, stepsize : integer > 0
Preconditions	The User is visualizing a case.
Main Flow	1. The user requests to modify the case; 2. the system gives to the user the capability of modify the sequent data: a) start b) stepsize 3. if the data are correct the system modifies the case, otherwise, it returns an error message
Postconditions	The case is modified. Otherwise, the operation is canceled

2.5.4 Use Case Specification "Create State"

ID	7
Name	Create State
Actors	User
Arguments	case : integer > 0, timestamp : integer ≥ 0
Preconditions	The User is visualizing a case or has just created one.
Main Flow	1. The user requests to creates a state; 2. the system requests to insert the sequent data: a) timestamp: the unit value that represents the number of case stepsize passed since the start 3. if the data are correct, the system creates the state, otherwise, it returns an error message
Postconditions	The state is created. Otherwise, the operation is canceled

2.5.5 Use Case Specification "Visualize State"

ID	8
Name	Visualize State
Actors	User
Arguments	case : integer > 0, timestamp : integer ≥ 0
Preconditions	The User is visualizing a case.
Main Flow	1. The user requests to visualize a state; 2. the system shows a list of states of the case; 3. the user chooses one state 4. the system shows the data of the state.
Postconditions	The system is showing the data of the state

2.5.6 Use Case Specification "Modify State"

ID	9
Name	Modify State
Actors	User
Arguments	case : integer > 0, timestamp : integer ≥ 0, new_timestamp : integer ≥ 0
Preconditions	The User is visualizing a state.
Main Flow	1. The user requests to modify the state; 2. the system gives to the user the capability of modify the sequent data: a) timestamp 3. if the data are correct, the system modifies the state, otherwise, it returns an error message
Postconditions	The state is modified. Otherwise, the operation is canceled

2.5.7 Use Case Specification "Delete State"

ID	10
Name	Delete State
Actors	User
Arguments	case : integer > 0, timestamp : integer ≥ 0
Preconditions	The User is visualizing a state.
Main Flow	1. The user requests to delete the state; 2. the system deletes the state.
Postconditions	The state is deleted

2.5.8 Use Case Specification "Create Assignment"

ID	11
Name	Create Assignment
Actors	User
Arguments	case : integer > 0, timestamp : integer ≥ 0, variable : string, values : [string, ...]
Preconditions	The User is visualizing a state.
Main Flow	1. The user requests to create an assignment; 2. the system requests to insert the sequent data: a) a variable chosen from one in the list of all variables b) if the variable is simple, then the value of the variable, otherwise, the values of the fields of the variable 3. if the data are correct the system creates the assignment, otherwise, it returns an error message
Postconditions	The assignment is created, otherwise, the operation is canceled

2.5.9 Use Case Specification "Visualize Assignment"

ID	12
Name	Visualize Assignment
Actors	User
Arguments	case : integer > 0, timestamp : integer ≥ 0, variable : string
Preconditions	The User is visualizing a state.
Main Flow	1. The user requests to visualize an assignment; 2. the system shows the data of the assignment.
Postconditions	The system is showing the data of the assignment

2.5.10 Use Case Specification "Delete Assignment"

ID	13
Name	Delete Assignment
Actors	User
Arguments	case : integer > 0, timestamp : integer ≥ 0, variable : string
Preconditions	The User is visualizing an assignment.
Main Flow	1. The user requests to delete the assignment; 2. the system deletes the assignment.
Postconditions	The system has deleted the assignment. In case of error, a message is shown

2.5.11 Use Case Specification "Modify Assignment"

ID	14
Name	Modify Assignment
Actors	User
Arguments	case : integer > 0, timestamp : integer ≥ 0, variable : string, new_variable : string, value : string
Preconditions	The User is visualizing an assignment.
Main Flow	1. The user requests to modify the assignment; 2. the system gives to the user the capability of modify the sequent data: a) variable b) value of the variable in case it is simple, otherwise the fields and their values 3. if the data are correct the system modifies the assignment, otherwise it returns an error message
Postconditions	The system has modified the assignment. Otherwise, the operation is canceled

2.5.12 Use Case Specification "Visualize Variables"

ID	15
Name	Visualize Variables
Actors	
Arguments	
Preconditions	The application is ready to use.
Main Flow	1. The system retrieves all the variables
Postconditions	The system has retrieved all the variables

2.5.13 Use Case Specification "Create Variable"

ID	16
Name	Create Variable
Actors	User
Arguments	name : string, type : {feature,action}, category : string, domain : string, fields : [string, ...]
Preconditions	The User is creating or modifying an assignment.
Main Flow	1. The user requests to create a variable; 2. the system requests to insert the following data: a) name: the name of the variable b) type: if the variable is a feature or an action c) category: a category chosen from one in the list of all categories d) domain: a domain chosen from one in the list of all domains e) fields: the fields of the variable if the variable is composite 3. if the data are correct the system creates the variable, otherwise, it returns an error message
Postconditions	The system has created the variable. Otherwise, the operation is canceled

2.5.14 Use Case Specification "Create Domain"

ID	17
Name	Create Domain
Actors	User
Arguments	name : string, type : string, attributes : [string, ...]
Preconditions	The User is creating a variable.
Main Flow	1. The user requests to create a domain: 2. the system requests to inserts the following data: a) name: the name of the domain b) type: the type of the domain c) attributes: the attributes of the domain depending on the type 3. if the data are correct the system creates the domain, otherwise, it returns an error message
Postconditions	The system has created the domain. Otherwise, the operation is canceled

2.5.15 Use Case Specification "Visualize Domains"

ID	18
Name	Visualize Domains
Actors	
Arguments	
Preconditions	The User is creating a variable.
Main Flow	1. The system retrieves all the domains
Postconditions	The system has retrieved all the domains

2.5.16 Use Case Specification "Create Category"

ID	19
Name	Create Category
Actors	
Arguments	supercategory : string, category : string
Preconditions	The User is creating a variable.
Main Flow	1. The user requests to create a category: 2. the system requests to insert the following data: a) supercategory: the root of the category to create b) category: the category to create 3. if the data are correct the system creates the categories, otherwise it returns an error message
Postconditions	The system has created all the categories. Otherwise, the operation is canceled

2.5.17 Use Case Specification "Visualize Category Tree"

ID	20
Name	Visualize Category Tree
Actors	
Arguments	
Preconditions	The User is creating a variable.
Main Flow	1. The system retrieves the category tree;
Postconditions	The system has recovered the category tree.

2.6 Specification: "Admin"

2.6.1 Use Case Specification "Visualize Categories Tree Admin"

ID	21
Name	Visualize Categories Tree Admin
Arguments	
Actors	Admin
Preconditions	The application is ready to use.
Main Flow	1. The Admin requests to visualize the categories tree; 2. the system shows the categories tree.
Postconditions	The system is showing the category tree.

2.6.2 Use Case Specification "Modify Category"

ID	22
Name	Modify Category
Actors	Admin
Arguments	category : string, supercategory : string
Preconditions	The Admin is visualizing the category tree.
Main Flow	1. The Admin chooses a category and requests to modify it; 2. the system gives to the Admin the capability to modify the following data: a) name of the category b) supercategory 3. if the data are correct the system modifies the subtree, otherwise, it returns an error message.
Postconditions	The system has modified the category tree. Otherwise the operation is canceled

2.6.3 Use Case Specification "Delete Category Tree"

ID	23
Name	Delete Category Tree
Actors	Admin
Arguments	category : string
Preconditions	The Admin is visualizing the category tree.
Main Flow	1. The Admin chooses a category and requests to delete it; 2. the system deletes the category and its subtree, otherwise, it returns an error message.
Postconditions	The system has deleted the category and its subtree. Otherwise, the operation is canceled

2.6.4 Use Case Specification "Visualize Domains Admin"

ID	24
Name	Visualize Domains Admin
Actors	Admin
Arguments	
Preconditions	The application is ready to use.
Main Flow	1. The Admin requests to visualize the domains; 2. the system shows the domains.
Postconditions	The system is showing the domains

2.6.5 Use Case Specification "Modify Domain"

ID	25
Name	Modify Domain
Actors	Admin
Arguments	name : string, attributes : [string, ...]
Preconditions	The Admin is visualizing the domains.
Main Flow	1. The Admin chooses a domain and requests to modify it; 2. the system gives to the Admin the capability to modify the following data: a) name b) attributes 3. if the data are correct the system modifies the domain, otherwise, it returns an error message.
Postconditions	The system has modified the domain. Otherwise, the operation is canceled

2.6.6 Use Case Specification "Delete Domain"

ID	26
Name	Delete Domain
Actors	Admin
Arguments	domain : string
Preconditions	The Admin is visualizing the domains.
Main Flow	1. The Admin chooses a domain and requests to delete it; 2. the system deletes the domain, otherwise, it returns an error message.
Postconditions	The system has deleted the domain. Otherwise, the operation is canceled

2.6.7 Use Case Specification "Visualize Cases Admin"

ID	27
Name	Visualize Cases Admin
Actors	Admin
Arguments	
Preconditions	The application is ready to use.
Main Flow	1. The Admin requests to visualize the cases 2. the system show a list of cases 3. the Admin chooses a case 4. the system shows the data of the case
Postconditions	The system is showing the data of the case

2.6.8 Use Case Specification "Delete Case"

ID	28
Name	Delete Case
Actors	Admin
Arguments	case : string
Preconditions	The Admin is visualizing a case.
Main Flow	1. The Admin requests to delete the case; 2. the systems deletes the case.
Postconditions	The system is showing the data of the case

2.6.9 Use Case Specification "Visualize Variables Admin"

ID	29
Name	Visualize Variables Admin
Actors	Admin
Preconditions	The application is ready to use.
Main Flow	1. The Admin requests to visualize the variables; 2. the system shows the variables; 3. the Admin chooses a variable; 4. the system shows the data of the variable
Postconditions	The system is showing the data of the variable

2.6.10 Use Case Specification "Modify Variable"

ID	30
Name	Modify Variable
Actors	Admin
Arguments	variable : string, name : string, type : {feature,action}, category : string, domain : string, fields : [string, ...]
Preconditions	The Admin is visualizing a variable.
Main Flow	1. The Admin requests to modify the variable; 2. the system gives to the Admin the capability to modify the following data: a) name b) type c) category d) domain e) fields 3. if the data are correct the system modifies the variable, otherwise, it returns an error message.
Postconditions	The system is showing the data of the variable

2.6.11 Use Case Specification "Delete Variable"

ID	31
Name	Delete Variable
Actors	Admin
Arguments	variable : string
Preconditions	The Admin is visualizing a variable.
Main Flow	1. The Admin requests to delete the variable; 2. the system deletes the variable. Otherwise, a message error is returned.
Postconditions	The system has deleted the variable. Otherwise, the operation is canceled

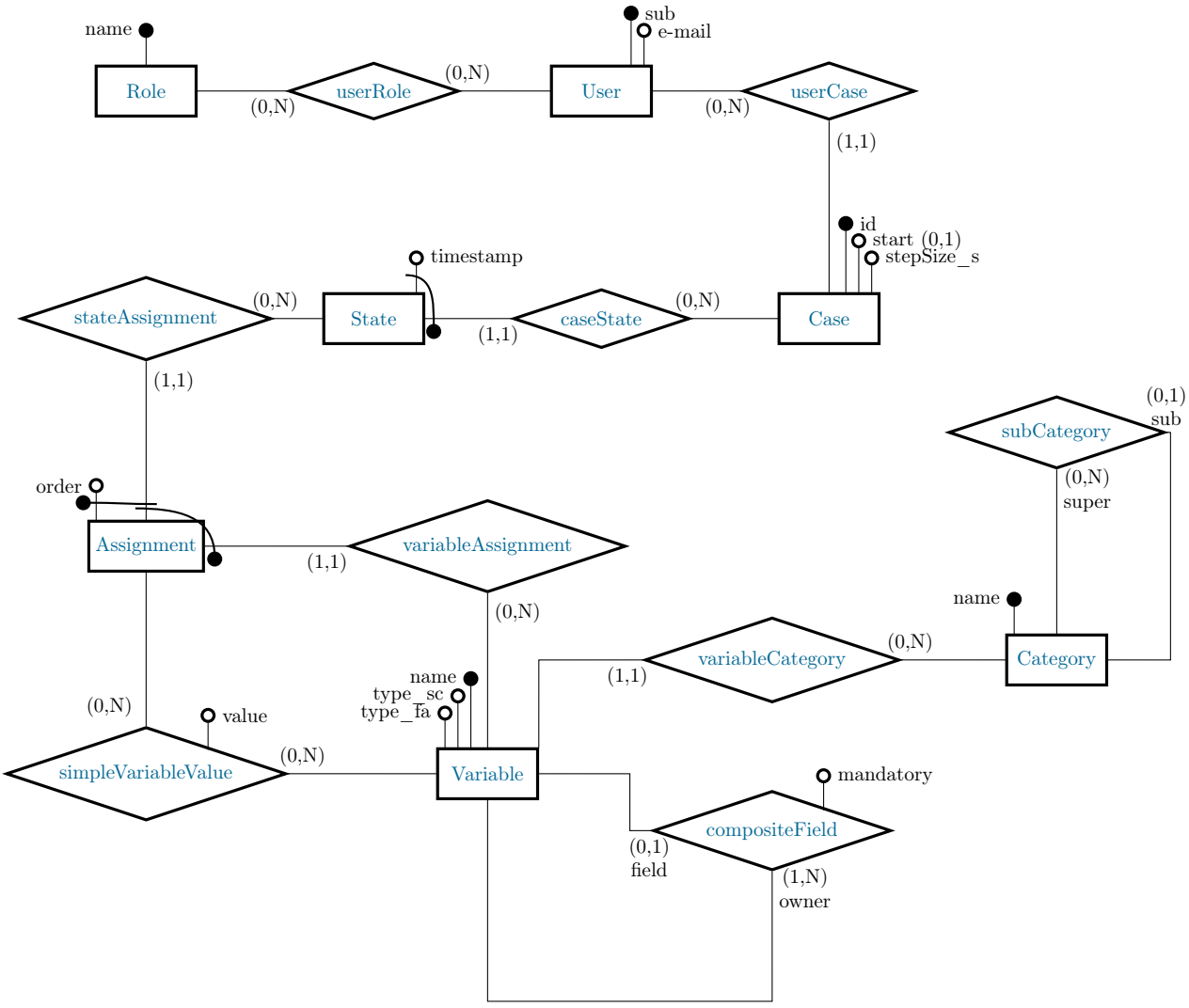
Part II

Database and Application Design

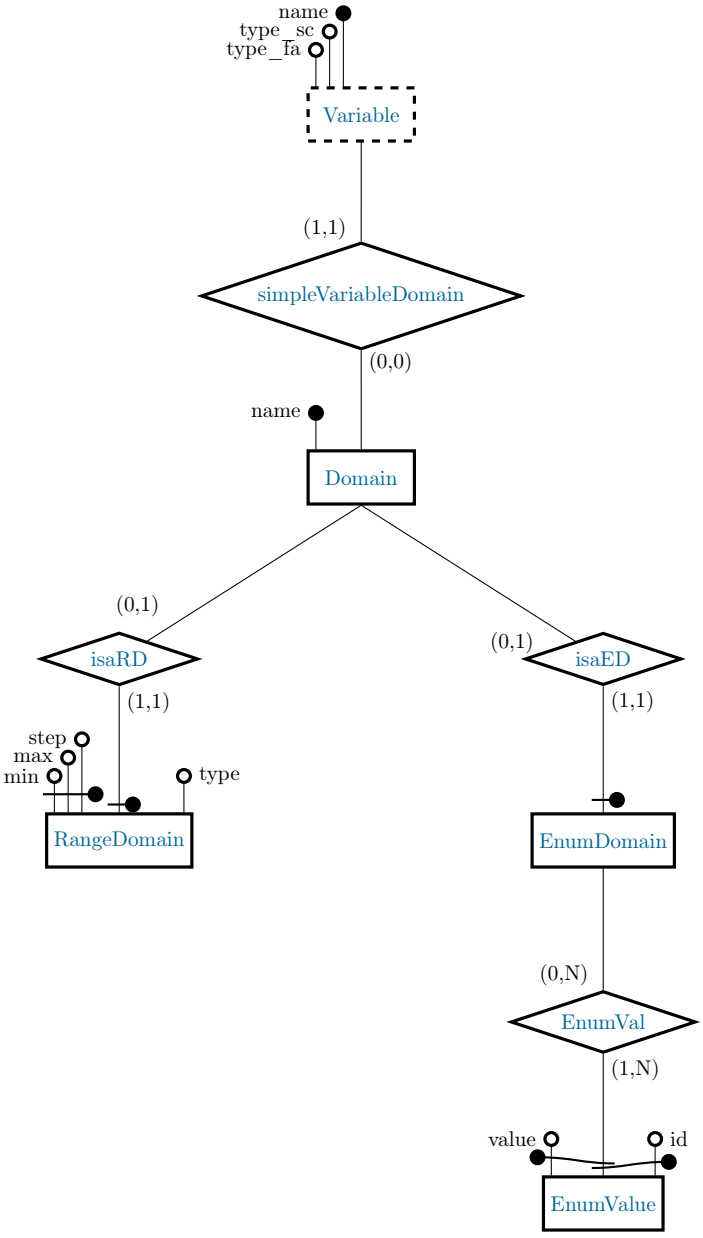
Chapter 3

Database Design - Entity Relation Restructuring

3.1 Entity Relationship Diagram: "Case"



3.2 Entity Relationship Diagram: "Domain"



3.3 Data Dictionary

Entity User
Each instance of this entity represents a user.

attribute	domain	cardinality	description
e-mail	longString		The email address of the user
sub	integer		The unique id of the user

Entity Role
Each instance of this entity represents a user role.

attribute	domain	cardinality	description
name	smallString		The name of the role

Entity Case
Each instance of this entity represents a case.

attribute	domain	cardinality	description
stepSize_s	intGEZ		The unit in seconds of the case
start	timestamp	(0,1)	The timestamp of the start of the case
id	integer		The unique id of the case

Entity State
Each instance of this entity represents a state.

attribute	domain	cardinality	description
timestamp	intGZ		The number of case stepsize since the start

Entity Assignment
Each instance of this entity represents an assignment.

attribute	domain	cardinality	description
order	intGZ		The order number of the assignment in the state

Entity Variable
Each instance of this entity represents a variable.

attribute	domain	cardinality	description
type_fa	val_fa		the type of the variable
type_sc	val_sc		the structure of the variable
name	smallString		The name of the variable

Constraints:

[C.Variable.completeanddisjoint] Formally:

$$\forall v\ Variable(v)\ \rightarrow$$
$$[type\_sc(v, \text{“}composite''\text{)} \leftrightarrow \exists cm\ compositeField(v, c, m)] \wedge$$
$$[\exists a, val\ simpleVariableValue(a, v, val)\ \rightarrow\ type\_sc(v, \text{“}simple''\text{)}] \wedge$$
$$[type\_sc(v, \text{“}simple''\text{)}\ \leftrightarrow\ \exists dom\ simpleVariableDomain(v, dom)]$$

Entity Category
Each instance of this entity represents a category.

attribute	domain	cardinality	description
name	smallString		The name of the category

Entity Domain
Each instance of this entity represents a domain.

attribute	domain	cardinality	description
name	smallString		The name of the domain

Entity RangeDomain
Each instance of this entity represents a range domain.

attribute	domain	cardinality	description
min	real		The minimum value of the domain
max	real		The maximum value of the domain
step	realGZ		The step value in the range of the domain
type	domType		The type of the domain

Constraints:

[C.RangeDomain.completeanddisjoint] Formally:

$$\forall dom\ Domain(dom)\ \rightarrow$$
$$\{\exists rd\ isaRD(dom, rd)\ \rightarrow\ [\nexists ed\ isaED(dom, ed)]\}$$

[C.RangeDomain.fusion] isaInt(int) is a function that returns “true" if int is an integer.
Formally:

$$\forall dom\ Domain(dom)\ \rightarrow$$
$$\{type(dom, \text{“}int\text{”})\ \leftrightarrow\ [\exists min, max, step\ min(dom, min) \wedge max(dom, max) \wedge step(dom, step) \wedge$$
$$isaInt(min)=\text{“}true\text{”} \wedge isaInt(max)=\text{“}true\text{”} \wedge isaInt(step)=\text{“}true\text{”}]\}$$

[C.RangeDomain.minmax] $min < max$.
Formally:

$$\forall rd, min, max\ RangeDomain(rd) \wedge min(rd, min) \wedge max(rd, max)\ \rightarrow\ min < max$$

Entity EnumDomain
Each instance of this entity represents an enumerate domain.
Attributes: None

Constraints:

[C.EnumDomain.completeanddisjoint] Formally:

$$\forall dom\ Domain(dom) \rightarrow \{\exists ed\ isaED(dom, ed) \rightarrow \nexists rd\ isaRD(dom, rd)]\}$$

Entity EnumValue
Each instance of this entity represents an enumerate value.

attribute	domain	cardinality	description
value	smallString		A value of the domain
id	intGZ		The id of the value

Relationship userRole
Attributes: None

Relationship userCase
Attributes: None

Relationship caseState
Attributes: None

Relationship stateAssignment
Attributes: None

Relationship variableAssignment
Attributes: None

Constraints:

[C.variableAssignment.variable.mandatory] If a composite variable cv is assigned, each mandatory field has a value.
mandatory_tree(field, root) is a function that returns "true" if the variable field is mandatory when its composite variable root is assigned.

Formally:

$$\forall a, cv, sv\ Variable(cv) \wedge type\_sc(cv, \text{“}composite''\text{)} \wedge variableAssignment(a, cv) \wedge Variable(sv) \wedge type\_sc(sv, \text{“}simple''\text{)} \wedge mandatory\_tree(cv, sv) = \text{“}true'' \rightarrow \exists val\ simpleVariableValue(a, sv, val)$$

[C.variableAssignment.variable.composite] If a composite variable cv is assigned in a and a simple variable sv has a value in a, then sv must be contained in cv.
tree_root_composite(field, composite) is a function that returns "true" if field has as root composite in the composite fields tree.

Formally:

$$\forall a, cv, sv, val\ Variable(cv) \wedge type\_sc(cv, \text{“}composite''\text{)} \wedge Variable(sv) \wedge type\_sc(sv, \text{“}simple''\text{)} \wedge variableAssignment(a, cv) \wedge simpleVariableValue(a, sv, val) \rightarrow \text{tree\_root\_composite(sv, cv)} = \text{“true”}$$

[C.variableAssignment.variable.simple] If a simple variable sv is assigned in a, then sv is the only simple varibale that has a value in a.

Formally:

$$\forall a, sv, sv', val\ Variable(sv) \wedge type\_sc(sv, \text{“}simple''\text{)} \wedge Variable(sv') \wedge type\_sc(sv', \text{“}simple''\text{)} \wedge variableAssignment(a, sv) \wedge simpleVariableValue(a, sv', val) \rightarrow \text{sv} = \text{sv'}$$

Relationship subCategory
Attributes: None

Constraints:

[C.subCategory.subcategories.different] A category can't contain itself in its subcategories.
path_category_tree(sub, super) is a function that returns "true" if the subcategory sub is contained in the category tree of super.

Formally:

$$\forall c, c'\ Category(c) \wedge \text{path\_category\_tree(c',c)} = \text{“true”} \rightarrow\ c' \neq c$$

Relationship variableCategory
Attributes: None

Relationship compositeField

attribute	domain	cardinality	description
mandatory	boolean		If the field's value is mandatory

Constraints:

[C.compositeField.field.type] A composite variable can contain only variables of its type.
path_composite_tree(field, root) is a function that returns "true" if the variable field is contained in the composite variable tree of root.

Formally:

$$\forall cv, f, t\ Variable(cv) \wedge type\_sc(cv, \text{“}composite''\text{)} \wedge Variable(f) \wedge \text{path\_composite\_tree(f,cv)} = \text{“true”} \wedge type\_fa(cv, t) \rightarrow\ type\_fa(f, t)$$

[C.compositeField.field.different] A composite can't contain itself.

Formally:

$$\forall cv, f\ Variable(cv) \wedge type\_sc(cv, \text{“}composite''\text{)} \wedge path\_composite\_tree(f, cv) \rightarrow\ f \neq cv$$

Relationship simpleVariableValue

attribute	domain	cardinality	description
value	real		Value of the simple variable in the assignment

Constraints:

[C.simpleVariableValue.domain.value] For each value val assigned to a simple variable sv, val is of the same domain of sv.
value_domain(val, dom) is a function that returns "true" if val is of the dom domain.

Formally:

$$\forall a, sv, val, dom\ [Assignment(a) \wedge Variable(sv) \wedge type\_sc(sv, \text{“}simple''\text{)} \wedge simpleVariableValue(a, sv, val) \wedge simpleVariableDomain(sv, dom)]\ \rightarrow\ [\text{value\_domain(val,dom)} = \text{“true”}]$$

Relationship simpleVariableDomain
Attributes: None

Relationship EnumVal
Attributes: None

Relationship isaRD
Attributes: None

Relationship isaED
Attributes: None

Domain intGZ
The domain is the subset of the Integer domain where values are >0.

Domain intGEZ
The domain is the subset of the Integer domain where values are ≥0.

Domain realGZ
The domain is the subset of the Real domain where values are >0.

Domain realGEZ
The domain is the subset of the Real domain where values are ≥0.

Domain smallString
The domain is composed of the strings with maximum 50 characters.

Domain longString
The domain is composed of the strings with maximum 254 characters.

Domain val_fa
The domain is composed of the following elements:

$$\{feature, action\}$$

Domain val_sc
The domain is composed of the following elements:

$$\{simple, composite\}$$

Domain domType
The domain is composed of the following elements:

$$\{int, real\}$$

Chapter 4

Database Design - Relational Schema

4.1 SQL Definition of Domains

4.1.1 smallString

```
CREATE DOMAIN smallString AS varchar(50)
```

4.1.2 longString

```
CREATE DOMAIN users.longString AS varchar(254);
```

4.1.3 var_type_fa

```
CREATE TYPE var_type_fa AS ENUM('action','feature');
```

4.1.4 var_type_sc

```
CREATE TYPE var_type_sc AS ENUM('simple','composite');
```

4.1.5 domType

```
CREATE TYPE domType AS ENUM('int','real');
```

4.1.6 intGZ

```
CREATE DOMAIN intGZ AS INTEGER CHECK (value > 0);
```

4.1.7 intGEZ

```
CREATE DOMAIN intGEZ AS INTEGER CHECK (value >= 0);
```

4.1.8 real

```
CREATE DOMAIN realGZ AS NUMERIC(1015,15);
```

4.1.9 realGZ

```
CREATE DOMAIN realGZ AS NUMERIC(1015,15) CHECK (value > 0);
```

4.2 Relational Schema: "Users"

The following DB relations will be created within DB schema "User".
Columns denoted with * (asterisk) might host NULL values.

Users.Role (name: smallString)

Users.User (sub: Integer, email: longString)
[DBConstraint.1] *unique:* email

Users.UserRole (role: smallString, sub: Integer)
[DBConstraint.2] *foreign key:* role references Users.Role(name)
[DBConstraint.3] *foreign key:* sub references Users.User(sub)

4.3 Relational Schema: "public"

The following DB relations will be created within DB schema "public".
Columns denoted with * (asterisk) might host NULL values.

public.Case (id: Integer, start*: timestamp, stepsize_s: intGZ, sub: Integer)
foreign keysub references Users.User(sub)

public.State (case: Integer, timestamp: intGEZ)
[DBConstraint.4] *foreign key:* case references public.Case(id)

public.Category (name: smallString)

public.subCategory (super: smallString, sub: smallString)
[DBConstraint.5] *foreign key:* super references public.Category(name)
[DBConstraint.6] *foreign key:* sub references public.Category(name)

public.Domain (name: smallString)

public.Variable (name: smallString, type_fa: var_type_fa, type_sc: var_type_sc, category: smallString, domain: smallString)
[DBConstraint.7] *foreign key:* category references public.Category(name)
[DBConstraint.8] *foreign key:* domain references public.Domain(name)

public.compositeField (owner: smallString, field: smallString, mandatory: boolean)
[DBConstraint.9] *foreign key:* owner references public.Variable(name)
[DBConstraint.10] *foreign key:* field references public.Variable(name)

public.Assignment (case: Integer, timestamp: intGEZ, ordernum: intGZ, variable: smallString)
[DBConstraint.11] *foreign key:* (case,timestamp) references public.State(case, timestamp)
[DBConstraint.12] *foreign key:* variable references public.Variable(name)
[DBConstraint.13] *unique:* case,timestamp,variable

public.SimpleVariableValue (case: Integer, timestamp: intGEZ, ordernum: intGZ, variable: smallString, value: Real)
[DBConstraint.14] *foreign key:* (case,timestamp,ordernum) references public.Assignment(case, timestamp, ordernum)
[DBConstraint.15] *foreign key:* variable references public.Variable(name)

public.RangeDomain (domain: smallString, min: Real, max: Real, step: realGZ, type: domType)
[DBConstraint.16] *foreign key:* domain references public.Domain(name)
[DBConstraint.17] *unique:* min,max,step

public.EnumerateDomain (domain: smallString)
[DBConstraint.18] *foreign key:* domain references public.Domain(name)

public.EnumerateValue (domain: smallString, value: smallString, id : intGZ)
[DBConstraint.19] *foreign key:* domain references public.Domain(name)
[DBConstraint.20] *unique:* domain,id

Chapter 5

Design of Functionalities

5.1 User

5.1.1 REST Layer

The data type in JSON format refers to the effective data type of the data transferred.

5.1.1.1 Create case

- Request:
  - URL: `/case`
  - HTTP Method: `POST`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content:

```
{
    "timestamp": timestamp,
    "stepsize": integer
}
```

- Response:
  - Success:
    - Code: `201`
    - Body:

```
{
    "id": integer
}
```

  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.2 Get cases of the user

- Request:
  - URL: `/cases`
  - HTTP Method: `GET`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None
- Response:
  - Success:
    - Code: `200`
    - Body:

```
{
    "cases": [
        {
            "id" : integer,
            "timestamp" : timestamp,
            "stepsize" : string,
            "user" : integer
        },
        ...
    ]
}
```

  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.3 Modify case

- Request:
  - URL: `/case/{id}`
  - HTTP Method: `PUT`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content:

```
{
    "timestamp": timestamp,
    "stepsize": integer
}
```

- Response:
  - Success:
    - Code: `200`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - not authorized:
    - Code: `403`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.4 Delete case

- Request:
  - URL: `/case/{id}`
  - HTTP Method: `DELETE`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None
- Response:
  - Success:
    - Code: `200`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.5 Get states

- Request:
  - URL: `/case/{id}/states`
  - HTTP Method: `GET`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None
- Response:
  - Success:
    - Code: `200`
    - Body:

```
{
    "states" : [
        {
        "case": integer
        "timestamp" : integer
        },
        ...
    ]
}
```

  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.6 Create state

- Request:
  - URL: `/case/{id}/state`
  - HTTP Method: `POST`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None

```
{
    "timestamp": integer
}
```

- Response:
  - Success:
    - Code: `201`
    - Body:
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.7 Delete state

- Request:
  - URL: `/case/{id}/state/{timestamp}`
  - HTTP Method: `DELETE`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None
- Response:
  - Success:
    - Code: `200`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.8 Create assignment

- Request:
  - URL: `/case/{id}/state/{timestamp}/assignment`
  - HTTP Method: `POST`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content:

```
{
    "variable": string,
    "simplevariables": [
```

If variable is simple:

```
            {
                "simplevariable" : string
                "value" : real
            }
```

If variable is composite:

```
            {
                "simplevariable" : string
                "value" : real
            },
            ...
    ]
}
```

- Response:
  - Success:
    - Code: `201`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.9 Get Assignments

- Request:
  - URL: `/case/{id}/state/{timestamp}/assignments`
  - HTTP Method: `GET`

  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None

- Response:
  - Success:
    - Code: `200`
    - Body:

```
{
    "case": integer,
    "timestamp": integer,
    [
        {
        "order" : integer,
        "variable" : string,
        "simplevariables": [
            {
            "simplevariable" : string,
            "value" : real
            },
            ...
        ]
        },
        ...
    ]
}
```

  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.10 Modify assignment

- Request:
  - URL: `/case/{id}/state/{timestamp}/assignment/{variable}`
  - HTTP Method: `PUT`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content:

```
{
    "variable": string,
    "simplevariables" : [
        {
        "simplevariable" : string,
        "value" : real
        },
        ...
    ]
}
```

- Response:
  - Success:
    - Code: `200`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.11 Delete assignment

- Request:
  - URL: `/case/{id}/state/{timestamp}/assignment/{order}`
  - HTTP Method: `DELETE`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None
- Response:
  - Success:
    - Code: `200`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.12 Create domain

- Request:
  - URL: `/domain`
  - HTTP Method: `POST`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content:

```
{
    "type": enum("boolean","enumerate","intrange","realrange
       "),
    "attributes": {
```

If type=="range":

```
        {
            "min": real,
            "max": real,
            "step": real
        }
```

If type=="enumerate":

```
        [
            value1: string,
            value2 : string,
            ...
        ]
}
```

- Response:
  - Success:
    - Code: `201`
    - Body:
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.13 Get domains

- Request:
  - URL: `/domains`
  - HTTP Method: `GET`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None
- Response:
  - Success:
    - Code: `200`
    - Body:

```
{
    "domains" :
    {
        domain1 : [ attribute1, attribute2, ...],
        ...
    }
}
```

  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.14 Delete domain

- Request:
  - URL: `/domain/{name}`
  - HTTP Method: `DELETE`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content:
- Response:
  - Success:
    - Code: `200`
    - Body: None
  - Failure - bad request:

    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.15 Create category

- Request:
  - URL: `/category`
  - HTTP Method: `POST`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content:

```
{
    "supercategory" : string,
    "category" : string
}
```

- Response:
  - Success:
    - Code: `201`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.16 Get categories

- Request:
  - URL: `/categories`
  - HTTP Method: `GET`
  - Headers:
    - `X-User-Token`: token

  - Content type: `JSON`
  - Content: None

- Response:
  - Success:
    - Code: `200`
    - Body:

```
{
    "category": [
        root :
            children1_height1 : [
                children1,1_height2,
                ...
            ],
            children2_height1 : [
                children2,1_height2,
                ...
            ],
            ...
    ]
}
```

  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.17 Get subcategories

- Request:
  - URL: `/category/{name}/subcategories`
  - HTTP Method: `POST`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None

- Response:
  - Success:
    - Code: `201`
    - Body:

```
{
    "subcategories" :
    [
        children1_level1 : [
            children1,1_level2,
            ...
        ],
        children2_level2 : [
            ...
        ],
        ...
    ]
}
```

- Failure - bad request:
  - Code: `400`
- Failure - not authenticated:
  - Code: `401`
- Failure - internal server error:
  - Code: `500`
  - Body:

```
{
    "msg" : string
}
```

5.1.1.18 Create variable

- Request:
  - URL: `/variable`
  - HTTP Method: `POST`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content:

```
{
    "name": string,
    "type_fa": enum("feature","action"),
    "category": string,
```

If type_sc == "simple":

```

    "domain": string
```

If type_sc == "composite":

```
    "fields" : [
        variable1 : string,
        variable2 : string,
        ...
    ]
}
```

- Response:
  - Success:
    - Code: `201`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.1.1.19 Get variables

- Request:
  - URL: `/variables`
  - HTTP Method: `GET`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None
- Response:
  - Success:
    - Code: `200`
    - Body:

```
{
    "variables": [
        {
            "name" : string,
            "type_fa" : enum("feature","action"),
            "type_sc" : enum("simple","composite"),
            "category" : string,
            "domain" : string,
            "fields" : [
                field1 : string,
                field2 : string,
                ...
            ]
        },
        ...
    ]
}
```

  - Failure - bad request:
    - Code: `400`

  - Failure - not authenticated:
    * Code: `401`
  - Failure - internal server error:
    * Code: `500`
    * Body:

```
{
    "msg" : string
}
```

5.1.1.20 Modify variable

- Request:
  - URL: `/variable/{name}`
  - HTTP Method: `PUT`
  - Headers:
    * `X-User-Token`: token
  - Content type: `JSON`
  - Content:

```
{
    "name": string,
    "category": string
}
```

- Response:
  - Success:
    * Code: `200`
    * Body: None
  - Failure - bad request:
    * Code: `400`
  - Failure - not authenticated:
    * Code: `401`
  - Failure - internal server error:
    * Code: `500`
    * Body:

```
{
    "msg" : string
}
```

5.2 Admin

5.2.1 REST Layer

The data type in JSON format refers to the effective data type of the data transferred.

5.2.1.1 Get cases

- Request:
  - URL: `/cases`
  - HTTP Method: `GET`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None
- Response:
  - Success:
    - Code: `200`
    - Body:

```
{
    "cases": {
            "id" : integer,
            "start" : timestamp,
            "stepsize" : string,
            "user" : integer
            },
            ...
}
```

  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.2.1.2 Modify category

- Request:
  - URL: `/category/{name}`
  - HTTP Method: `PUT`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content:

```
{
    "category" : string,
    "supercategory" : string
}
```

- Response:
  - Success:
    - Code: `200`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.2.1.3 Delete category

- Request:
  - URL: `/category/{name}`
  - HTTP Method: `DELETE`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None
- Response:
  - Success:
    - Code: `200`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.2.1.4 Modify domain

- Request:
  - URL: `/domain/{name}`
  - HTTP Method: `PUT`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content:

```
{
    "domain": {
            "type": string,
    "attributes": {
```

If type=="range":

```
        {
        "min": real,
        "max": real,
        "step": real
        }
```

If type=="enumerate":

```
        [
            value1: string,
            value2 : string,
            ...
        ]
            }
        }
}
```

- Response:
  - Success:
    - Code: 200
    - Body: None
  - Failure - bad request:
    - Code: 400
  - Failure - not authenticated:
    - Code: 401
  - Failure - internal server error:
    - Code: 500
    - Body:

```
{
    "msg" : string
}
```

5.2.1.5 Delete domain

- Request:
  - URL: `/domain/{name}`
  - HTTP Method: `DELETE`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None
- Response:
  - Success:
    - Code: `200`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.2.1.6 Delete case

- Request:
  - URL: `/case/{id}`
  - HTTP Method: `DELETE`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content: None
- Response:
  - Success:
    - Code: `200`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.2.1.7 Modify variable

- Request:
  - URL: `/variable`
  - HTTP Method: `PUT`
  - Headers:
    - `X-User-Token`: token
  - Content type: `JSON`
  - Content:

```
{
    "name": string,
    "type_fa": enum("feature","action"),
    "category": string,
```

If type_sc == "simple":

```
    "domain": string
```

If type_sc == "composite":

```
    "fields" : [
        variable1 : string,
        variable2 : string,
        ...
    ]
}
```

- Response:
  - Success:
    - Code: `200`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

5.2.1.8 Delete variable

- Request:
  - URL: `/variable/{name}`
  - HTTP Method: `DELETE`
  - Headers:
    - `X-User-Token`: token

  - Content type: `JSON`
  - Content: None

- Response:
  - Success:
    - Code: `200`
    - Body: None
  - Failure - bad request:
    - Code: `400`
  - Failure - not authenticated:
    - Code: `401`
  - Failure - internal server error:
    - Code: `500`
    - Body:

```
{
    "msg" : string
}
```

Bibliography

[1] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[2] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[3] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997.

[4] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[5] David Haussler. Learning conjunctive concepts in structural domains. *Machine learning*, 4(1):7–40, 1989.

[6] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[7] K. Lang and E. Baum. Query learning can work poorly when a human oracle is used. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 335–340. IEEE Press, 1992. Cited on page(s) 6,7.

[8] David D. Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In William W. Cohen and Haym Hirsh, editors, *Machine Learning Proceedings 1994*, pages 148 – 156. Morgan Kaufmann, San Francisco (CA), 1994.

[9] B. Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[10] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, 1992.

[11] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[12] Liantao Wang, Xuelei Hu, Bo Yuan, and Jianfeng Lu. Active learning via query synthesis and nearest neighbour search. *Neurocomputing*, 147:426–434, 2015.