

Command line completion: an illustration of learning and decision making using the imprecise Dirichlet model

Erik Quaeghebeur & Gert de Cooman

Abstract—A method of command line completion based on probabilistic models is described. The method supplements the existing deterministic ones. The probabilistic models are developed within the context of imprecise probabilities.

An imprecise Dirichlet model is used to represent the assessments about all possible completions and to allow for learning by observing the commands typed previously. Due to the use of imprecise probabilities a partial (instead of a linear) ordering of the possible completion actions will be constructed during decision making. Markov models can additionally be incorporated to take recurring sequences of commands into account.

Keywords—command line completion, imprecise Dirichlet model, learning, decision making, Markov model

I. INTRODUCTION

THE purpose of this paper is to illustrate how models using *imprecise probabilities* [1], which are a generalization of standard probabilities, can be used in a practical application. Command line completion is chosen because of its conceptual simplicity, because of its relevance (making daily computer use more user friendly) and because it allows us to show the important aspects of the models we're investigating.

The central model is the imprecise Dirichlet model or IDM [2]. It is a model for representing probabilistic information about multinomial processes, combined with a complementary updating scheme. The updating scheme allows for learning from observations. An IDM, together with utility specifications for completion actions can be used in decision making. The IDM has already been used for several other applications [3], [4].

Our research is concerned with the use of the IDM for learning in Markov models. This can be incorporated in the current illustration, as it allows the completion method to take preceding commands into account.

II. CONCEPTS AND MODELS

We start by defining the concepts we'll encounter and by separately describing the models we'll use for our command line completion method.

A. Command line completion

Command line completion is a feature that various operating systems (OS's) provide for users working in text-mode. Usually, the user invokes the feature by pressing

Erik Quaeghebeur and Prof. Gert de Cooman are members of the SYSTeMS research group of the department of Electrical Energy, Systems and Automation, Ghent University, Belgium.

E-mail: {Erik.Quaeghebeur,Gert.deCooman}@UGent.be.

a predefined key after having typed part of a command (used here as generic term for a command, a filename, etc.).¹ The user is then either presented with a list of n possible completions i from which he chooses one, or, if there is only one, this completion is given on the command line. An example is given in Fig. 1, where $n = 4$ and $i \in \{\text{logger, login, logname, logout}\}$.

```
command-prompt# log<TAB>
logger      login      logname    logout
command-prompt# logn<TAB>
command-prompt# logname_
```

Fig. 1. Example of command line completion. An underscore and <...> respectively denote the cursor and the pressing of a key.

Our objective is to add supplementary functionality to command line completion, as described above, in cases where multiple completions are possible. By modelling (and thus making assumptions about) the behavior of the user, we can order the *completion actions* the OS can take. For example, giving the 'best guess' completion on the command line or presenting all possible completions in an way that makes the best guesses the most visible ones. There has already been some previous work on applying probabilistic models in this context, e.g., as a testbed for techniques for web prefetching [6].

B. The imprecise Dirichlet model

We now assume that the user chooses an i according to an unknown, but fixed, (discrete) *multinomial distribution* \mathbf{c} over all possible completions. A component c_i gives the chance that the user chooses completion i . We call the set of all possible multinomial distributions

$$\mathcal{C}_n = \{\mathbf{c} \mid \sum_i c_i = 1; c_i \geq 0, \forall i\}$$

the *completion simplex*; an example is given in Fig. 2.

Ideally we would want to know \mathbf{c} exactly, but practically we only have partial, probabilistic, information about \mathbf{c} . The OS therefore uses an *imprecise Dirichlet model* to represent this probabilistic information. This model consists of a convex set of (continuous) *Dirichlet distributions* over the completion simplex \mathcal{C}_n . The expectation of \mathbf{c} under this model consists of a subset M_t of \mathcal{C}_n .

¹For example, the TAB-key is used for flavors of UNIX using the bash shell [5]. This shell also allows for context sensitive completions, e.g., only filenames are given as completions after a copying command.

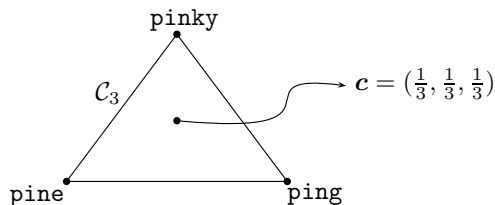


Fig. 2. The completion simplex for “command-prompt# pin<TAB>”.

Every time the user enters a new command, the OS updates the IDM.² This results in a new IDM with a modified, more precise expectation M_{t+1} , as is illustrated in Fig. 3. Initially, before the OS has observed any command entered by the user, M_0 is taken to be the whole interior of C_n .

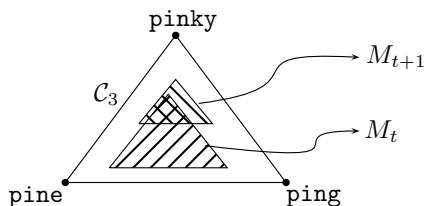


Fig. 3. The expectation under an IDM before and after observing “command-prompt# pinky<ENTER>”.

C. Markov models: extending the IDM

We can make the assumptions about the user’s behavior more complex by allowing the multinomial distribution c over all possible completions to depend on the command(s) typed on the preceding command line(s). This behavior corresponds to a *Markov model* with an unknown *transition matrix* T . The j^{th} row of T will correspond to the multinomial distribution, conditional on the preceding command(s). For example:

$$T = \begin{matrix} i = \text{pine} & \text{ping} & \text{pinky} & j = \\ \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1/2 & 1/4 & 1/4 \end{pmatrix} & \text{logname} \\ & & & \text{logger} \end{matrix}$$

The OS then uses a modified version of the IDM, consisting of one set of Dirichlet distributions per row, to represent the probabilistic information available about the transition probabilities of the Markov model (the elements of T). As before, this IDM can be updated after observing a transition (i.e., a couple of successive commands).

III. ACTIONS, UTILITY AND DECISION MAKING

Now that we have a model to represent the behavior of the user, and know how to keep it up-to-date, we are going to use it to decide what completion actions to take.

A. Completion actions and utility

As seen in Section II-A the completion actions can consist of (but are not necessarily limited to):

- action a_i , presenting completion i on the command line;
- giving a specially ordered list of n completions (other than the usual alphabetic ordering).

²The updating is done using a generalization of Bayes’ rule [1], [2].

These actions are illustrated in Fig. 4. We take the second to be the *default action* (why we need a default action will become clear in the next section).

command-prompt# pin<TAB>	command-prompt# log<TAB>
command-prompt# pine_	logout logger
	login
	logname

Fig. 4. Two examples of completion actions.

Independent of what action the OS takes, there is only one completion that the user is looking for. Each non-default action the OS takes thus has a certain *utility* for the user, which we suppose can be quantified. For example, action a_i gives a utility of +1 if i is the correct completion and -1 if it isn’t.

B. Expected utility and decision making

The OS now has a model both for what completion the user is looking for, the IDM, and for the utility of his actions. By combining these two models, the OS can calculate the *expected utility* for each of the completion actions. For the utility specifications given above, this amounts to calculating the expected utility for the actions a_i .

The expected utility for the a_i ’s can then be used to construct a *partial ordering* of these actions. If this ordering has one maximal element, the OS decides to present the corresponding completion. If there are multiple maximal elements, the default action is taken: showing the partial ordering to the user, such that the maximal elements are the most conspicuous ones.

IV. CONCLUSIONS

We’ve given a description of a model for command line completion. The focus was on the models used rather than on the details of the implementation. It is clear that these models can also be applied in conceptually similar situations such as the prefetching of web pages or word completion for the Short Message Service of mobile phones. An at first sight unrelated field as bio-informatics also contains possible applications (e.g., alignment of gene sequences), which we are going to look into in future research.

REFERENCES

- [1] Peter Walley, *Statistical reasoning with imprecise probabilities*, Chapman and Hall, London, 1991.
- [2] Peter Walley, “Inferences from multinomial data: learning about a bag of marbles,” *Journal of the Royal Statistical Society B*, vol. 58, no. 1, pp. 3–57, 1996.
- [3] Marco Zaffalon, “The naive credal classifier,” *Journal of Statistical Planning and Inference*, vol. 105, no. 1, pp. 5–21, 2002.
- [4] Erik Quaeghebeur and Gert de Cooman, “Game-theoretic learning using the imprecise Dirichlet model,” in *ISIPTA ’03 – Proceedings of the Third International Symposium on Imprecise Probabilities and Their Applications*, Bernard, Seidenfeld, and Zaffalon, Eds., 2003.
- [5] Free Software Foundation, <http://www.gnu.org/manual/bash/>, *The GNU Bash Reference Manual*.
- [6] Brian D. Davison, *The design and evaluation of web prefetching and caching techniques*, Ph.D. thesis, Rutgers University, 2002.