Property-Based Testing for Parameter Learning of Probabilistic Graphical Models

Anna Saranti, Behnam Taraghi, Martin Ebner, Andreas Holzinger

Property-Based Testing for Parameter Learning of Probabilistic Graphical Models

Anna Saranti[1], Behnam Taraghi[3], Martin Ebner[3], Andreas Holzinger[1,2][0000−0002−6786−5194]

[1] Medical University Graz, Auenbruggerplatz 2, A-8036 Graz, Austria
<forename.familyname>@medunigraz.at
[2] xAI Lab, Alberta Machine Intelligence Institute, T6G 2H1, Edmonton, Canada
[3] Department Educational Technology, Graz University of Technology
Münzgrabenstrasse 36/I, 8010, Graz, Austria
b.taraghi@tugraz.at
martin.ebner@tugraz.at

Abstract. Code quality is a requirement for successful and sustainable software development. The emergence of Artificial Intelligence and data driven Machine Learning in current applications makes customized solutions for both data as well as code quality a requirement. The diversity and the stochastic nature of Machine Learning algorithms require different test methods, each of which is suitable for a particular method. Conventional unit tests in test-automation environments provide the common, well-studied approach to tackle code quality issues, but Machine Learning applications pose new challenges and have different requirements, mostly as far the numerical computations are concerned. In this research work, a concrete use of property-based testing for quality assurance in the parameter learning algorithm of a probabilistic graphical model is described. The necessity and effectiveness of this method in comparison to unit tests is analyzed with concrete code examples for enhanced retraceability and interpretability, thus highly relevant for what is called explainable AI.

Keywords: Machine Learning, Probabilistic Graphical Models, Property-Based Testing

1 Introduction

Most Machine Learning (ML) approaches are stochastic. Consequently, most existing testing techniques are inadequate for ML code implementations. Consequently, the ML community uses numerical testing, metamorphic testing, mutation testing, coverage-guided fuzzing testing, proof-based testing, and especially property-based testing to detect problems in ML code implementations as early as possible [1]. Because these ML models are increasingly used for decision support, e.g. in the medical domain, there is an urgent need for quality assurance - particularly with a focus on domain-dependent properties. On such is monotonicity and specifies a software as learned by an ML model to provide a prediction.

Interestingly, approaches for checking monotonicity of the generated model, in particular of black-box models, are lacking [13].

The concept of property-based testing (PBT) relies on randomly generated test cases and it is a very relevant extension for unit tests. Defining concrete test cases is a central task when developing unit tests. However, it is very time-consuming and still often incomplete. Therefore methods for automatic generation of a variety of test-cases with only one specification, are more effective and profitable. Test developers don't have to define all possible edge cases anymore; those are automatically discovered by the corresponding frameworks [9]. The task of the developer shifts from listing and programming a lot of use-cases, to analyzing the constraints and the properties of the software under test and let the framework randomly generate values that fulfil the constraints and explore the relevant edge-cases automatically.

Since Artificial Intelligence applications have become prevalent, the need of corresponding quality management tools is rising. Frameworks like ProbFuzz [4], tailored for the needs of probabilistic programming systems and generate lots of different probability distributions. Several examples of machine learning programs involving neural networks are described in [3] and one of the most popular and useful ones, the Markov Chain Monte Carlo (MCMC) is indeed exercised with property-based techniques [5]. An extensive study is provided by [15]. This research work focuses on a particular probabilistic graphical model with a defined structure, where the parameters need to be learned with the expectation-maximization algorithm. The testing of the implementation follows the paradigm of property-based testing.

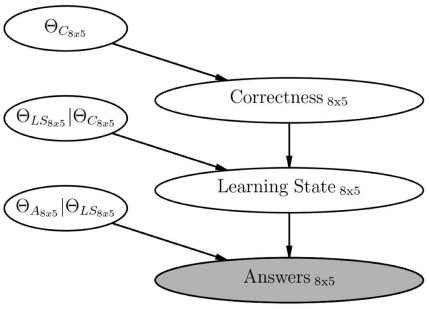
2 Previous Work

Previous research work was based on data of the learning analytics application "1x1 trainer" [4], developed by the department Educational Technology of Graz University of Technology, Austria. Users answer 1-digit multiplication questions that are posed to them sequentially. Detailed information about the gathered data, student modelling and analysis can be found in [14], [12]. The student model that was designed and provided valuable insights, is used in the forthcoming sections and its structure is depicted in figure 1. Each question has its own probabilistic graphical model; the structure of all models is basically the same.

In Bayesian parameter estimation the computation of the posterior distribution with regard to the prior is computed with the Bayes rule 1:

$$\overbrace{P(\Theta|\mathcal{D})}^{posterior} = \frac{\overbrace{P(\mathcal{D}|\Theta)}^{likelihood}\overbrace{P(\Theta)}^{prior}}{\underbrace{P(\mathcal{D})}_{marginal\ likelihood}} \tag{1}$$

[4] `https://schule.learninglab.tugraz.at/einmaleins/`, Last accessed 26 April 2020



wrong	**correct**
0.5	0.5

	operand	···	**unclassified**	**correct**
wrong	1/7	···	1/7	0.0
correct	0.0	···	0.0	1.0

	···	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	···
operand	···	0.083	0.0	0.0	0.0	0.0	0.0	0.083	0.0	···
intrusion	···	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	···
consistency	···	0.0	0.0	0.0	0.0	0.0	0.058	0.058	0.058	···
off-by	···	0.0	0.0	0.25	0.25	0.0	0.25	0.25	0.0	···
add/sub/div	···	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	···
pattern	···	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	···
unclassified	···	0.0	0.018	0.0	0.0	0.0	0.0	0.0	0.0	···
correct	···	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	···

Fig. 1: Parameters of the Learning Competence Probabilistic Graphical Model

The parameters of the student model in figure 1 are the ones corresponding to the uniform (uninformative) prior. The data that were collected from the students by their interaction with the learning application, are used to update the parameters, as it will be described in the following sections. The equation of the joint distribution 2 will be very valuable in the following sections for the update of the parameters.

$$\begin{aligned} P(\mathbf{Correctness_q}, \mathbf{Learning\ State_q}, \mathbf{Answers_q}) = \\ P(\mathbf{Correctness_q})\ P(\mathbf{Learning\ State_q}|\mathbf{Correctness_q}) \\ P(\mathbf{Answers_q}|\mathbf{Learning\ State_q}) \end{aligned} \tag{2}$$

3 Learning the Model's Parameters with batch Expectation-Maximization (EM)

Bayesian parameter learning is applicable when all variables are visible [7]; in that case all components of the equation 1 are computable. The model of the learning competence contains hidden variables, therefore the computation of the posterior of each random variable cannot be made directly. In this case the method that is used is expectation-maximization (abbreviated by EM). The concepts of prior, likelihood and posterior that were described in the previous section 2 are used in the description of this method.

3.1 Notation

The entities that are necessary for the analytical solution for the computation of the posterior distributions of all model's variables are the following:

N : Number of all samples in dataset

n : One sample of N

M : Number of $\mathbf{Correctness_q}$ possible outcomes ($M = 2$)

μ : Index of $\mathbf{Correctness_q}$ outcome

w_μ : Parameters of $\mathbf{Correctness_q}$

K : Number of $\mathbf{Learning\ State_q}$ possible error types and correct outcome ($K = 8$)

k : Index of $\mathbf{Learning\ State_q}$ outcome (one error type out of $K - 1$ or correct)

$\pi_{k|\mu}$: Parameters of the $\mathbf{Learning\ State_q}$ variable

Q : Number of $\mathbf{Answers_q}$ random variables (90 in total)

q : Index of $\mathbf{Answers_q}$ (one question of Q)

X : Number of all possible answers of each question (columns of conditional probability tables of $\mathbf{Answers_{1\times1}}$ to $\mathbf{Answers_{10\times9}}$)

x : one answer out of X

x_n : the answer of the n-th sample

$\theta_{x|k}$: The parameters of the $\mathbf{Correctness_q}$ random variable

Θ : All current parameters of the model : (set of all w_μ, $\pi_{k|\mu}$, $\theta_{x|k}$).

Θ_{old} : All parameters of the previous EM iteration.

$\mathbf{X}$: The set of all visible variables. In this model, they are all $\mathbf{Answers_q}$ variables.

$\mathbf{Z}$: The set of all latent variables or hidden causes. In this model, they are all $\mathbf{Correctness_q}$ and $\mathbf{Learning\ State_q}$ variable.

3.2 Expectation-Maximization (EM) algorithm

The goal of the EM-Algorithm is to find appropriate values for all parameters Θ. In general a better model will fit the data better, although it must not overfit. The latent variables $\mathbf{Correctness_q}$ and $\mathbf{Learning\ State_q}$ are not observed, so the direct maximization of the likelihood $P(\mathbf{X};\Theta)$ of the data according to this model is not possible; the observed data $\mathbf{X}$ (not to be confused with the number of possible answers of each question X) is incomplete. Each iteration of the EM-algorithm computes a different instantiation of the table CPDs.

By using marginalization:

$$P(\boldsymbol{X};\Theta) = \sum_{\boldsymbol{Z}} P(\boldsymbol{X},\boldsymbol{Z};\Theta) \tag{3}$$

$$\ln P(\boldsymbol{X};\Theta) = \ln \left\{ \sum_{\boldsymbol{Z}} P(\boldsymbol{X},\boldsymbol{Z};\Theta) \right\} \tag{4}$$

If the complete data set $\{\boldsymbol{X},\boldsymbol{Z}\}$ were known, then it would be straightforward to try to maximize the complete data log-likelihood. To avoid multiplication of very small floating point numbers that can lead to zero, one can equivalently maximize the log-likelihood function $\ln P(\boldsymbol{X};\Theta)$.

The EM-Algorithm works iteratively and consists of four steps:

1. Initialization of all parameters to Θ_0 of the complete dataset $\{\boldsymbol{X},\boldsymbol{Z}\}$ and set $\Theta_0 = \Theta_{old}$.
2. E-Step: Computation of the posterior distribution $P(\boldsymbol{Z}|\boldsymbol{X};\Theta_{old})$ of $\mathbf{Z}$ given the visible variables and the previous parameters.
3. M-Step: Compute new Θ parameters by trying to maximize 6 the expected value of the posterior distribution 5 over the latent variables $\boldsymbol{Z}$:

$$\mathcal{Q}(\Theta,\Theta_{old}) = \sum_{\boldsymbol{Z}} P(\boldsymbol{Z}|\boldsymbol{X};\Theta_{old}) \ln P(\boldsymbol{X},\boldsymbol{Z};\Theta) \tag{5}$$

$$\Theta = \underset{\Theta}{\operatorname{argmax}}\ \mathcal{Q}(\Theta,\Theta_{old}) \tag{6}$$

4. Compute the incomplete data likelihood $P(\boldsymbol{X};\Theta)$ or equivalently the log-likelihood $\ln P(\boldsymbol{X};\Theta)$ 4. If the log-likelihood's increase or the Θ parameters' change is not significant compared to the previous iteration, then stop. Else, set current Θ with the values computed in M-Step and return to E-Step. The EM-algorithm is a "meta-algorithm" since it contains an inference in the E-Step ([11]). The iterative process is depicted in figure 2.

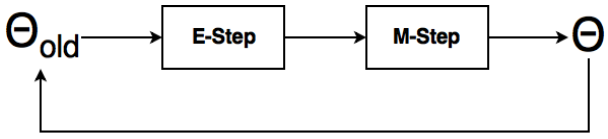


Fig. 2: The iteration loop of the EM-algorithm

3.3 Analytical Solution of Expectation-Maximization (EM) for the model of Learning Competence

The steps of the EM-algorithm are applied to the model of the Learning Competence for the derivation of the analytical solution for the update of the parameters. We apply those steps to the model of Learning Competence of each

question q separately. The equations in this subsection omit the subscript q; they apply to the model of each question independently.

The equation 7 expresses the joint probability distribution equation 8 is derived from:

$$P(\boldsymbol{X}, \boldsymbol{Z}; \Theta) = \prod_{n} \prod_{\mu} \prod_{k} w_{\mu} \pi_{k|\mu} \theta_{x_n|k} \tag{7}$$

$$\ln P(\boldsymbol{X}, \boldsymbol{Z}; \Theta) = \sum_{n=1}^{N} \ln \left(\sum_{\mu=1}^{M} \sum_{k=1}^{K} w_{\mu} \pi_{k|\mu} \theta_{x_n|k} \right) \tag{8}$$

The expected value of the complete log-likelihood $\mathcal{Q}(\Theta, \Theta_{old})$ is:

$$\begin{aligned} \mathcal{Q}(\Theta, \Theta_{old}) = \mathbb{E}_{P(\boldsymbol{Z}|\boldsymbol{X}; \Theta_{old})} \left[\ln P(\boldsymbol{X}, \boldsymbol{Z}; \Theta) \right] = \\ \sum_{\boldsymbol{Z}} P(\boldsymbol{Z}|\boldsymbol{X}; \Theta_{old}) \ln P(\boldsymbol{X}, \boldsymbol{Z}; \Theta) = \\ \sum_{n} \sum_{\mu} \sum_{k} \gamma(z_{\mu k}^{(n)}) \left(\ln w_{\mu} + \ln \pi_{k|\mu} + \ln \theta_{x_n|k} \right) \end{aligned} \tag{9}$$

The responsibility $\gamma(z_{\mu k}^{(n)})$ of the hidden error cause or correct k for $n-$th sample coupled with the probability of the answer being answered correctly, can be computed by using the Bayes rule and the factorization of the joint probability distribution from equation 2:

$$\gamma(z_{\mu k}^{(n)}) = P(z_{\mu k}^{(n)}|x^{(n)}; \Theta_{old}) \propto P(x^{(n)}|z_{\mu k}^{(n)}; \Theta_{old}) P(z_{\mu k}^{(n)}; \Theta_{old}) \tag{10}$$

The values of all $\gamma(z_{\mu k}^{(n)})$ values in equation 10 are provided up to a normalization factor. Since $\gamma(z_{\mu k}^{(n)})$ depends only on Θ_{old}, it can be considered a constant in the process of maximization of $\mathcal{Q}$. At the same time following constraints that reflect the conditional probability rules must be fulfilled:

$$\sum_{\mu=1}^{M} w_{\mu} = 1 \tag{11}$$

$$\sum_{k=1}^{K} \pi_{k|\mu} = 1 \tag{12}$$

$$\sum_{x=1}^{X} \theta_{x|k} = 1 \tag{13}$$

The maximization of the complete log-likelihood $\mathcal{Q}(\Theta, \Theta_{old})$ leads to the parameters of the model. The maximization process must also fulfil the constraints

in equations 11, 12, 13, which can be made with the use of Lagrange Multipliers. The maximum of the following expression must be found:

$$\mathcal{Q}(\Theta, \Theta_{old}) + \lambda\left(\sum_{\mu} w_{\mu} - 1\right) + \sum_{\mu} \lambda_{\mu}\left(\sum_{k} \pi_{k|\mu} - 1\right) + \sum_{k} \lambda_{k}\left(\sum_{x} \theta_{x|k} - 1\right) \quad (14)$$

First, the update of the parameters of a particular $\mathbf{Correctness_q}$ value $w_{m|q}$, is made from the data samples n' that have as question $q = q_{n'} \in [q_1 \cdots q_Q]$, and answer m being either correct or wrong:

$$\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) \frac{1}{w_{m|q}} + \lambda \stackrel{!}{=} 0 \quad \| \cdot w_{m|q} \quad (15)$$

$$\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) + \lambda w_{m|q} \stackrel{!}{=} 0 \quad \| \sum_{m=1}^{M} \quad (16)$$

$$\sum_{m=1}^{M} \sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) + \lambda \sum_{m=1}^{M} w_{m|q} \stackrel{!}{=} 0 \quad (17)$$

$$\lambda = -\sum_{m=1}^{M} \sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) = -N' \quad (18)$$

because :

$$\sum_{m=1}^{M} \gamma(z_{\mu k}^{(n')}) = 1 \quad (19)$$

$$w_{m|q} = \frac{\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')})}{N} \quad (20)$$

Secondly, the maximization with respect to a particular $\pi_l \in [\pi_1 \cdots \pi_K]$ is computed. The derivative must be set to 0 and all parameters of the expression 14 not related to π_l can be eliminated as constants. If the number of samples that are answered wrongly is N', the following steps provide the analytical solution for the update rule for any π_k:

$$\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) \frac{1}{\pi_l} + \lambda_{\mu} \stackrel{!}{=} 0 \quad \| \cdot \pi_l \quad (21)$$

$$\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) + \lambda_{\mu} \pi_l \stackrel{!}{=} 0 \quad \| \sum_{k=1}^{K} \quad (22)$$

$$\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) + \lambda_{\mu} \sum_{k=1}^{K} \pi_l \stackrel{!}{=} 0 \quad (23)$$

$$\lambda_\mu = -\sum_{k=1}^{K}\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) \tag{24}$$

$$\pi_k = \frac{\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')})}{\sum_{k=1}^{K}\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')})} \tag{25}$$

Thirdly, the maximization with respect to $\theta_{x|q,k,}$ is performed in a similar manner. The update of the parameters of a particular $\mathbf{Answers_q}$ value $\theta_{x|q,k}$, is made from the data samples n' that have as question $q = q_{n'} \in [q_1 \cdots q_Q]$, and answer $x = x_{n'} \in [x_1 \cdots x_X]$:

$$\sum_{n'=1}^{N'} \frac{\gamma(z_{\mu k}^{(n')})}{\theta_{x|k}} + \lambda_k \stackrel{!}{=} 0 \quad \| \cdot \theta_{x|k} \tag{26}$$

$$\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) + \lambda_k \theta_{x|k} \stackrel{!}{=} 0 \quad \| \sum_{x=1}^{X} \tag{27}$$

$$\sum_{x=1}^{X}\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) + \lambda_k \sum_{x=1}^{X} \theta_{x|k} \stackrel{!}{=} 0 \tag{28}$$

$$\lambda_k = -\sum_{x=1}^{X}\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')}) \tag{29}$$

$$\theta_{x|k} = \frac{\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')})}{\sum_{x=1}^{X}\sum_{n'=1}^{N'} \gamma(z_{\mu k}^{(n')})} \tag{30}$$

The steps of the EM-Algorithm for updating the parameters of this Bayesian Model are as follows:

1. Initialization of all parameters Θ_0. In this case that is the uniform prior.
2. E-Step: Computation of $\gamma(z_{\mu k}^{(n)})$ using equation 10
3. M-Step: Compute new Θ parameters $w_{\mu|q}$, π_k and $\theta_{x|k}$ using equations 20, 25 and 30
4. Compute the likelihood $P(\boldsymbol{X};\Theta)$ or log-likelihood $\ln P(\boldsymbol{X};\Theta)$:

$$\begin{aligned} P(\mathbf{Correctness_q}, \mathbf{Answers_q}; \Theta) = \\ \frac{P(\mathbf{Correctness_q}, \mathbf{LearningState_q}, \mathbf{Answers_q}; \Theta)}{P(\mathbf{LearningState_q}|\mathbf{Correctness_q}, \mathbf{Answers_q}; \Theta)} \\ P(\mathbf{Correctness_q}; \Theta) P(\mathbf{Correctness_q}|\mathbf{LearningState_q}; \Theta) \end{aligned} \tag{31}$$

If the likelihood or the parameters values do not converge, then set current Θ with the values computed in M-Step and goto E-Step.

Figure 3 depicts the steps of the EM-algorithm updating procedure. The dataset used for training is called training set.

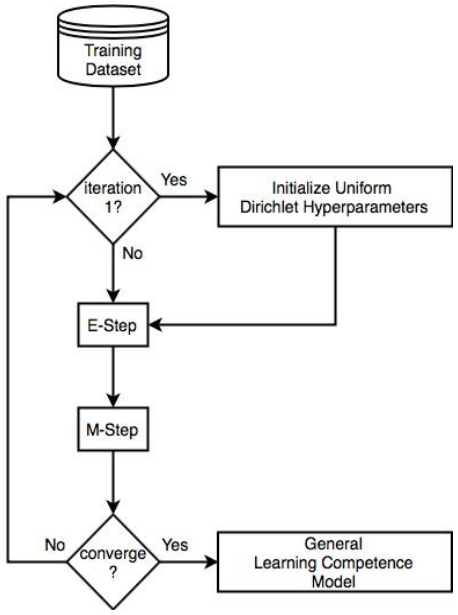


Fig. 3: Expectation-maximization training algorithm applied on the training set. In the first step the uniform Dirichlet hyperparameters are used. Then alternating E- and M-steps bring the parameters/log-likelihood convergence.

It is proven that the EM-algorithm increases the log-likelihood of the observed data X at each iteration ([2]):

$$\ln P(\boldsymbol{X};\Theta) \geq \ln P(\boldsymbol{X};\Theta_{old}) \tag{32}$$

The procedure of updating the log-likelihood in this manner is shown to guarantee convergence to a stationary point, which can be a local minimum, local maximum or saddle point. Fortunately, by initializing the iterations from different starting Θ_0 and injecting small changes to the parameters, the local minima and saddle points can be avoided ([7]).

4 Fractional Updating

Since the learning application proposes questions continuously, it is important to update the beliefs about the learning competence of the student as soon as an answer is present. As new evidence is observed - in the form of answered questions - the model shifts the value of the parameters to reflect the fact that the belief about the learning competence of the user is changed.

With fractional updating ([6]), the initialization and updating of the parameters is made by means of the Dirichlet pseudocounts. The starting pseudocount number is set to 1.0 to express a weak belief about the learning competence of the student. In this application, for each question-answer pair, only one probabilistical graphical model needs to be updated. The data sample only contains the value of the corresponding observed variable $\mathbf{Answers_q}$. The update of the pseudocounts α is provided by equation:

$$\alpha_{ijk}^{l+1} = \alpha_{ijk}^{l} + P(\boldsymbol{X_i} = k, Parents_G(\boldsymbol{X_i}) = j|\mathcal{D}) \tag{33}$$

where the current joint probability of the updated variable and the value of its parents are used to update the value of the pseudocounts, which may no longer be an integer. $\mathcal{D}$ denotes the dataset of samples.

The fractional updating procedure can be explained by an example where a student provides the wrong answer 42 to the question 8×5. The probabilities start with the following values:

wrong	**correct**
$\frac{1}{2}$	$\frac{1}{2}$

operand	**intrusion**	**consistency**	**off-by**	**add/sub/div**	**pattern**	**unclassified**
$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$

	$\cdots$	**42**	$\cdots$
operand	$\cdots$	$\frac{1}{12}$	$\cdots$
intrusion	$\cdots$	0	$\cdots$
consistency	$\cdots$	$\frac{1}{17}$	$\cdots$
off-by	$\cdots$	$\frac{1}{4}$	$\cdots$
add/sub/div	$\cdots$	0	$\cdots$
pattern	$\cdots$	0	$\cdots$
unclassified	$\cdots$	0	$\cdots$

Table 1: Probabilistic graphical model of the question 8×5 where all conditional probabilities (all rows of the conditional probability tables) are set uniformly.

The corresponding pseudocounts start with the following values:

wrong	**correct**
1	1

operand	**intrusion**	**consistency**	**off-by**	**add/sub/div**	**pattern**	**unclassified**
1	1	1	1	1	1	1

	···	**26**	···
operand	···	1	···
intrusion	···	0	···
consistency	···	1	···
off-by	···	1	···
add/sub/div	···	0	···
pattern	···	0	···
unclassified	···	0	···

Table 2: Probabilistic graphical model of the question 8×5 where all pseudocounts are set to value 1.

There are three error types that can cause the answer 42. The weights for each case, corresponding to the entity $P(\boldsymbol{X_i} = k, Parents_G(\boldsymbol{X_i}) = j|\mathcal{D})$ of the equation 33, are computed as follows:
$\frac{1}{2}\frac{1}{7}\frac{1}{12}/(\frac{1}{2}\frac{1}{7}\frac{1}{12} + \frac{1}{2}\frac{1}{7}\frac{1}{17} + \frac{1}{2}\frac{1}{7}\frac{1}{4}) = 0.21259$
$\frac{1}{2}\frac{1}{7}\frac{1}{17}/(\frac{1}{2}\frac{1}{7}\frac{1}{12} + \frac{1}{2}\frac{1}{7}\frac{1}{17} + \frac{1}{2}\frac{1}{7}\frac{1}{4}) = 0.15006$
$\frac{1}{2}\frac{1}{7}\frac{1}{4}//(\frac{1}{2}\frac{1}{7}\frac{1}{12} + \frac{1}{2}\frac{1}{7}\frac{1}{17} + \frac{1}{2}\frac{1}{7}\frac{1}{4}) = 0.63776$

The value of the updated pseudocounts of the **operand**, **intrusion**, and **consistency** error types, are presented in the following table:

	$\mathbf{CQ_{8\times5}}$	$\mathbf{LS_{8\times5}}$	$\mathbf{AS_{8\times5}}$	pseudocounts	probability
$D_{8\times5,\ 42,\ operand}$	wrong	operand	42	$1 + 0.21259$	0.151
$D_{8\times5,\ 42,\ consistency}$	wrong	consistency	42	$1 + 0.15006$	0.144
$D_{8\times5,\ 42,\ off-by}$	wrong	off-by	42	$1 + 0.63776$	0.205

The pseudocounts of the rest of the error types will remain to the value 1, so the total sum of pseudocounts that will be used as normalization value is: $4 \times 1 + 1.21259 + 1.15006 + 1.63776 = 8.0004$. The actual probabilities of the involved error types are listed in the table above; the rest have the value 0.12499. The pseudocounts of "wrong" are increased by +1, setting the probability of "wrong" to $\frac{2}{3}$ and "correct" to $\frac{1}{3}$. The values of the $\mathbf{Answers_{8x5}}$ random variable are computed accordingly.

This comprises a full iteration of the online EM-algorithm. In the next iteration, the newly computed pseudocounts will play the role of the prior that needs updating. Drawbacks and extensions of the fractional updating algorithm can be explained in the work of [6].

5 Evaluation of Parameter Learning

The evaluation of the parameter learning EM-algorithm is firstly made by computing the likelihood of a training set at each iteration. It is expected that the likelihood is increasing monotonically and converging with increasing number of

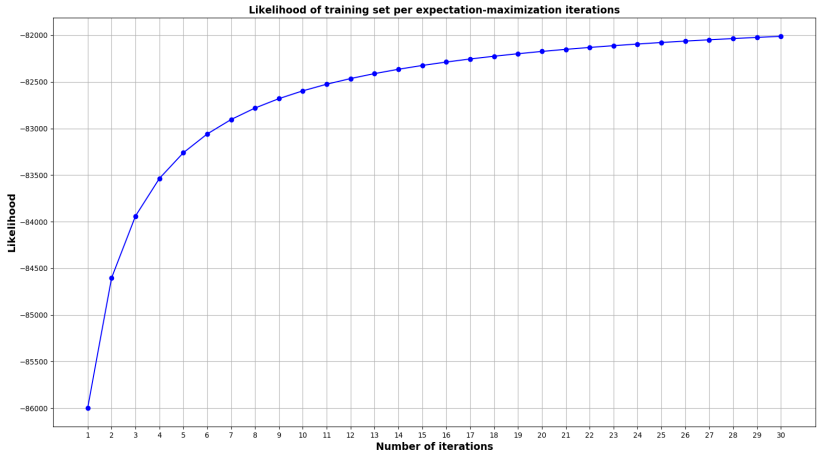


Fig. 4: Evolution of the likelihood of the training set with respect to the number of EM-iterations in the training set.

iterations. As seen in figure 4, this also applies for the likelihood of the models as a whole.

If the EM-algorithm is repeated for many iterations, the values of the parameters will be adjusted too much to the training set, without being able to generalize to the properties of the test set; an unseen user's learning competence must be also modelled sufficiently by the model. Figure 5 depicts the evolution of the likelihood of the test set, with respect to the number of EM-iterations of the training set. The likelihood of the test set decreases after 4 iterations; this is an indication of overfitting [2].

The expectation-maximization algorithm bases on the fact that all possible outcomes of all questions present at least one sample of the dataset. This was not the case in this application; the sufficient statistics condition was not fullfiled ([7], [8]). Nevertheless, it has been shown that the models performance is sufficient for practical purposes and that as new data are gathered, this problem might be solved.

6 Testing

Software testing in python is made with the use of the pytest framework [5] [10]. The expectation-maximization update rules were tested with some examples and compared to numerically computed results. But since the number of possible cases that must be tried out is very large, property-based testing was used [9].

[5] `https://docs.pytest.org/`, Last accessed 10 March 2020

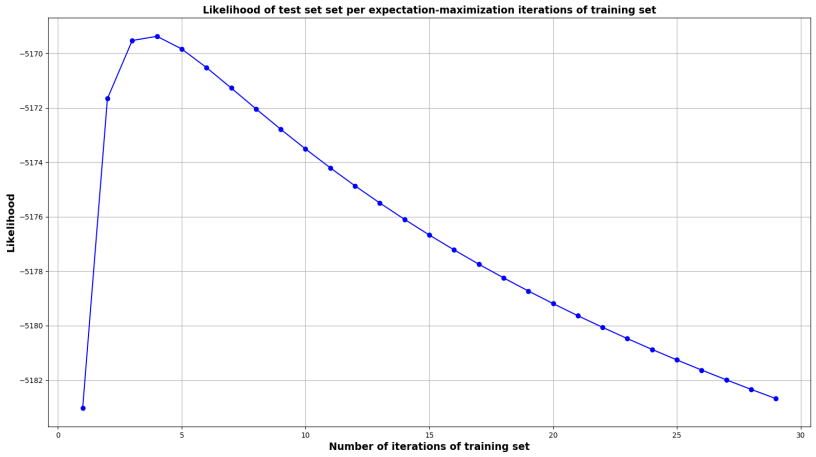


Fig. 5: Evolution of the likelihood of the test set with respect to the number of EM-iterations in the training set. The test set contains 1000 samples.

The hypothesis framework [6] gives the ability to write tests in an abstract manner, where the concrete numerical values are generated automatically.

An extended description of property-based testing is out of scope of this work, but the main idea is describing the properties of the function that is tested. By that means, generating several numerical examples was proven to be very effective. The fractional expectation-maximization update rule 4 has the following properties:

- The sum of each row of the conditional probability tables must equal to 1.0 after the update.
- Only one of the $\mathbf{Answers_q}$ (namely the one that corresponds to the posed question), as well as the corresponding $\mathbf{Learning\ State_q}$ and $\mathbf{Correctness_q}$ will change.
- A sampled answer can belong to one or more error types. The conditional probability of this answer will be increased in the $\mathbf{Answers_q}$ conditional probability table. Because the sum must remain 1.0, the conditional probabilities of the other answers will be decreased after the update. Similarly, the error types that could be responsible for this answer will have an increased conditional probability in the $\mathbf{Learning\ State_q}$, whereas the rest conditional probabilities will decrease. If the answer is wrong, then the belief that the student's ability to answer a question correctly will fall, and the $\mathbf{Correctness_q}$ will change by the same means.

[6] `https://hypothesis.readthedocs.io/en/latest/`, Last accessed 10 March 2020

The following code in listing 1.1, is the definition of one pytest test-case which covers several valid scenarios at once. The test-case exercises the fractional expectation-maximization learner and updater algorithm. During each scenario, one of the questions is randomly sampled, as well a valid corresponding answer between a minimum and a maximum value. The number of iterations of expectation-maximization also varies. Up to 10 tests are run at each execution (controlled by the parameter `max_examples`) and the execution can have unlimited time. The test's fixture that defines all those values, is passed as a parameter.

Listing 1.1: Property-based test-case definition

```
@given(question=sampled_from(
    elements=['1x1', '1x2', '1x3', '1x4', '1x5',
              '1x6', '1x7', '1x8', '1x9',
              '2x1', '2x2', '2x3', '2x4', '2x5',
              '2x6', '2x7', '2x8', '2x9',
              '3x1', '3x2', '3x3', '3x4', '3x5',
              '3x6', '3x7', '3x8', '3x9',
              '4x1', '4x2', '4x3', '4x4', '4x5',
              '4x6', '4x7', '4x8', '4x9',
              '5x1', '5x2', '5x3', '5x4', '5x5',
              '5x6', '5x7', '5x8', '5x9',
              '6x1', '6x2', '6x3', '6x4', '6x5',
              '6x6', '6x7', '6x8', '6x9',
              '7x1', '7x2', '7x3', '7x4', '7x5',
              '7x6', '7x7', '7x8', '7x9',
              '8x1', '8x2', '8x3', '8x4', '8x5',
              '8x6', '8x7', '8x8', '8x9',
              '9x1', '9x2', '9x3', '9x4', '9x5',
              '9x6', '9x7', '9x8', '9x9',
              '10x1', '10x2', '10x3', '10x4', '10x5',
              '10x6', '10x7', '10x8', '10x9']),
    answer=integers(min_value=0, max_value=99),
    number_of_em_iterations=integers(min_value=1,
                                     max_value=5))
@settings(max_examples=10, timeout=unlimited)
def test_fractional_em(
    question: str,
    answer: int,
    number_of_em_iterations: int,
    init_fractional_em_fixture: typing.Tuple):
```

Part of the implementation properties in the test-case are listed in 1.2. After the fractional updater is applied, the properties are checked one by one. The

probability distributions at each row of the Conditional Probability Tables must sum up to one all the time. Depending on the answer the "Correct" or "InCorrect" proportion must increase, as well as the corresponding error types that could generate it. The probabilities of the other error types that could not have generated this answer must decrease correspondingly, while the sum remains equal to 1. The likelihood of the training set data must increase, as described by the equation 32.

Listing 1.2: Properties of the test-case

```
for index in range(0, number_of_em_iterations):

    em_fractional_updater.update_alpha(users_questions,
                                       users_answers)

    # Returned parameters after update
    w_prob_dist = em_fractional_updater.get_w_prob()
    pi_prob_dist = em_fractional_updater.get_pi_prob()
    theta_prob_dist = em_fractional_updater.get_theta_prob()

    # Constraint: Probability distributions must sum up to 1
    learning_state_types = init_fractional_em_fixture[2]
    constraint_prob_dist_sum_one(w_prob_dist,
                                 pi_prob_dist,
                                 theta_prob_dist,
                                 question,
                                 learning_state_types)

    # Probabilities that must increase or decrease
    is_correct_answer = em_utils.is_correct_answer(question,
                                                   answer)

    if index >= 1:
        if is_correct_answer:
            assert w_prob_dist[question]["Correct"] >
                   w_prob_dist_prev[question]["Correct"], \
                   "The proportion of correct must increase"
            assert w_prob_dist[question]["InCorrect"] <
                   w_prob_dist_prev[question]["InCorrect"], \
                   "The proportion of incorrect must decrease"
        else:
            assert w_prob_dist[question]["Correct"] <
                   w_prob_dist_prev[question]["Correct"], \
                   "The proportion of correct must decrease"
            assert w_prob_dist[question]["InCorrect"] >
                   w_prob_dist_prev[question]["InCorrect"], \
                   "The proportion of incorrect must increase"
```

```
        learning_state_types = init_fractional_em_fixture[2]
        for learning_state_type in learning_state_types:
            if answer in \
            theta_prob_dist[question][learning_state_type].keys():

                # pi_prob_dist
                assert pi_prob_dist[question][learning_state_type] > \
                    pi_prob_dist_prev[question][learning_state_type], \
                    "The proportion of error type " +
                    learning_state_type + " must increase"

                assert theta_prob_dist[question]
                                      [learning_state_type]
                                      [answer] > \
                    theta_prob_dist_prev[question]
                                        [learning_state_type]
                                        [answer], \
                    "The probability of the given answer must increase"

                # theta_prob_dist
                for other_answer in \
                    theta_prob_dist[question][learning_state_type].keys():
                    if other_answer != answer:
                        assert theta_prob_dist[question]
                                              [learning_state_type]
                                              [other_answer] < \
                            theta_prob_dist_prev[question]
                                                [learning_state_type]
                                                [other_answer], \
                            "The probability other answers must decrease"
            else:
                assert pi_prob_dist[question][learning_state_type] < \
                    pi_prob_dist_prev[question][learning_state_type], \
                    "The proportion of error type " +
                    learning_state_type + " must decrease"

    # Copy
    likelihood = em_fractional_updater.
                 compute_log_likelihood(users_questions,
                                        users_answers)
    likelihood_training_set_array.append(likelihood)

    w_prob_dist_prev = copy.deepcopy(w_prob_dist)
    pi_prob_dist_prev = copy.deepcopy(pi_prob_dist)
    theta_prob_dist_prev = copy.deepcopy(theta_prob_dist)

# Likelihood must increase because it is the training set
likelihood_decreasing = em_utils.\
    check_decreasing_likelihood(likelihood_training_set_array)
assert not likelihood_decreasing, \
```

```
"The likelihood must not decrease : " +
str(likelihood_training_set_array)
```

7 Conclusion

Property-based testing is an effective method to ensure the code quality of probabilistic graphical models parameter learning. The necessity and effectiveness of this method has justified its use and was beneficial for the quality management of a learning-aware application. This work provides a concrete paradigm, that can be used by other similar applications that use probabilistic programming and analytical solutions of their learned parameter updating rules.

References

1. On testing machine learning programs. Journal of Systems and Software **164**, 110542 (2020). https://doi.org/10.1016/j.jss.2020.110542
2. Bishop, C.: Pattern recognition and machine learning. Springer (2006)
3. Braiek, H.B., Khomh, F.: On testing machine learning programs. Journal of Systems and Software **164**, 110542 (2020)
4. Dutta, S., Legunsen, O., Huang, Z., Misailovic, S.: Testing probabilistic programming systems. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 574–586 (2018)
5. Grosse, R.B., Duvenaud, D.K.: Testing mcmc code. arXiv preprint arXiv:1412.5218 (2014)
6. Jensen, F.V., Nielsen, T.D.: Bayesian Networks and Decision Graphs (Second Edition). Springer (2007)
7. Koller, D., Friedman, N.: Probabilistic graphical models: principles and techniques. MIT Press (2009)
8. Murphy, K.P.: Machine learning: a probabilistic perspective. MIT press (2012)
9. Nilsson, R.: ScalaCheck: the definitive guide. Artima (2014)
10. Okken, B.: Python Testing with Pytest: Simple, Rapid, Effective, and Scalable. Pragmatic Bookshelf (2017)
11. Pfeffer, A.: Practical Probabilistic Programming. Manning Publications (2016)
12. Saranti, A., Taraghi, B., Ebner, M., Holzinger, A.: Insights into learning competence through probabilistic graphical models. In: Lecture Notes in Computer Science LNCS 11713, pp. 250–271. Springer/Nature, Cham (2019). https://doi.org/10.1007/978-3-030-29726-8-16
13. Sharma, A., Wehrheim, H.: Testing monotonicity of machine learning models. arXiv:2002.12278 (2020)
14. Taraghi, B., Saranti, A., Legenstein, R., Ebner, M.: Bayesian modelling of student misconceptions in the one-digit multiplication with probabilistic programming. In: Proceedings of the Sixth International Conference on Learning Analytics & Knowledge. pp. 449–453 (2016)
15. Zhang, J.M., Harman, M., Ma, L., Liu, Y.: Machine learning testing: Survey, landscapes and horizons. arXiv preprint arXiv:1906.10742 (2019)