Abstract—Reliably predicting software defects is one of the most significant tasks in software engineering. Two are the major components of modern software reliability modeling approaches: (i) Extraction of salient features for software system representation, based on appropriately designed software metrics; and (ii) development of intricate regression models for count data, to allow for effective software reliability data modeling and prediction. Surprisingly, research in the latter frontier of count data regression modeling has been rather limited. More specifically, a lack of simple and efficient algorithms for posterior computation has made Bayesian approaches appear unattractive and thus underdeveloped in the context of software reliability modeling. In this work, we try to address these issues by introducing a novel Bayesian regression model for count data, based on the concept of max-margin data modeling, effected in the context of a fully Bayesian model treatment with simple and efficient posterior distribution updates. Our novel approach yields a more discriminative learning technique, making more effective use of our training data during model inference. In addition, it allows for better handling uncertainty in the modeled data, which can be a significant problem when training data is limited. We derive elegant inference algorithms for our model under the mean-field paradigm, and exhibit its effectiveness using publicly available benchmark datasets.

Index Terms—Count data, max-margin, mean-field, Dirichlet process, software reliability.

I. INTRODUCTION

In this work, we focus on the problem of software fault prediction based on appropriate software metrics. In general, software reliability growth models can be divided into two types [1]: models based on measuring the times between successive software failures, usually referred to as type I (e.g., [2]), and models for the numbers of failures in a given execution time, usually referred to as type II (e.g., [3]). In both cases, software reliability models are typically based on two main assumptions: (i) the assumption that the software is possibly imperfectly corrected after each failure or after various fixed time periods; and (ii) the assumption that each time the software is corrected, we can collect a number of metrics providing sufficient information to allow for performing bug prediction using appropriate statistical models [4]. Indeed, there is a large corpus of published research works showing that software faults prediction using appropriate metrics is plausible, and can yield extremely reliable estimates if proper modeling algorithms are designed [4], [5], [6].

Metrics-based software reliability prediction systems comprise two main algorithmic components. The first one concerns extraction of appropriate software metrics data to facilitate software reliability modeling and prediction. The second one focuses on devising effective regression models for count data, allowing for generating accurate software reliability predictions using available training sets of metrics data.

Quite a lot of research work has been devoted to software metrics design and extraction. For example, in [7] relative code churn (i.e., the relative amount of change to the system) has been proposed as an effective software feature representation. In [8], Hassan introduced the entropy of changes, a measure of the complexity of code changes, and showed it yields better results than using the absolute amounts of code change. Further, a different feature representation paradigm is represented by the Chidamber and Kemerer (CK) metrics suite [9]: their approach does not require the history of the system, but analyzes its current state in more detail, using a variety of metrics, based on the assumption that the current design and behavior of the program is the factor that influences the presence of future defects.

On the other hand, work on the development of effective regression models for count data, allowing for obtaining increased prediction accuracy even under significant uncertainty (e.g., due to the lack of large training datasets) has been rather limited. Most works are based on the naive Poisson generalized linear model (pGLM) [10]. Although simple and efficient, pGLM suffers from significant limitations; specifically, being a log-linear model, it does not allow for capturing subtle underlying patterns and nonlinearities in the modeled data, or modeling data with multiple modes, which are quite common in real-world datasets. In addition, its maximum-likelihood treatment, only providing a point estimate of the model, does not allow for the incorporation of prior information and lacks robustness, exhibiting high proneness to yielding biased estimates and overfitting, especially when the training sample size is small [11] (which is usually the case in real-world applications).

In contrast, methods developed under the Bayesian inference paradigm are capable of modeling the uncertain estimation and incorporating available prior information. In regression analysis of counts, however, the lack of simple and efficient algorithms for posterior computation has seriously limited its application to software reliability modeling, making Bayesian analysis of counts appear unattractive and thus underdeveloped. Notable exceptions to this rule include the hierarchical Bayesian log-linear models of [12], [13], the method presented in [14], which introduces a proportional hazards type approach, the method presented in [15], which extends the seminal model of Jelinski-Moranda to the case of metrics-based reliability modeling, and the method recently introduced in [16], which utilized a doubly stochastic homo-
Poisson regression (MEDPR) model for software reliability

We dub our model as the maximum-entropy discrimination generalized from the familiar SVR-style margin constraints. On this basis, we derive a mean-field inference to the training data, and a measure of reconstruction error on an architected Bayesian model, which measures the goodness of fit considering both the (negative) log-likelihood of our hierarchical Bayesian model and support vector machines (SVMs). Adoption of the high-performance techniques for both hierarchical Bayesian computation in a computationally efficient fashion, allowing to leverage existing computational approaches. In Section II, we provide a brief review of the theoretical background of our model. Specifically, we briefly review the utility of the DP prior as a nonparametric prior for mixture models, and we provide a concise introduction to regression models formulated under the maximum entropy discrimination framework. In Section III, we introduce our method, and derive its inference algorithm. In Section IV, we conduct the experimental evaluation of our approach. Finally, in the last section, we summarize and discuss our results, and conclude this work.

II. THEORETICAL BACKGROUND

A. The Dirichlet process

Dirichlet process models were first introduced by Ferguson [30]. A DP is characterized by a base distribution $G_0$ and
a positive scalar $\alpha$, usually referred to as the innovation parameter, and is denoted as $\text{DP}(\alpha, G_0)$. Essentially, a DP is a distribution placed over a distribution. Let us suppose we randomly draw a sample distribution $G$ from a DP, and, subsequently, we independently draw $M$ random variables $\{\Theta^*_m\}_{m=1}^M$ from $G$:

$$G|\alpha, G_0 \sim \text{DP}(\alpha, G_0)$$

$$\Theta^*_m|G \sim G, \quad m = 1, \ldots, M$$

Integrating out $G$, the joint distribution of the variables $\{\Theta^*_m\}_{m=1}^M$ can be shown to exhibit a clustering effect. Specifically, given the first $M-1$ samples of $G$, $\{\Theta^*_m\}_{m=1}^{M-1}$, it can be shown that a new sample $\Theta^*_M$ is either (a) drawn from the base distribution $G_0$ with probability $\frac{\alpha}{\alpha + M-1}$, or (b) is selected from the existing draws, according to a multinomial allocation, with probabilities proportional to the number of the previous draws with the same allocation [31]. Let $\{\Theta^*_m\}_{c=1}^C$ be the set of distinct values taken by the variables $\{\Theta^*_m\}_{m=1}^M$. Denoting as $\nu^*_c$ the number of values in $\{\Theta^*_m\}_{m=1}^M$ that equal to $\Theta^*_c$, the distribution of $\Theta^*_M$ given $\{\Theta^*_m\}_{m=1}^M$ can be shown to be of the form [31]

$$p(\Theta^*_M|\{\Theta^*_m\}_{m=1}^M, \alpha, G_0) = \frac{\alpha}{\alpha + M - 1} G_0 + \sum_{c=1}^C \frac{\nu^*_c}{\alpha + M - 1} \delta_{\Theta^*_c}$$

where $\delta_{\Theta^*_c}$ denotes the distribution concentrated at a single point $\Theta^*_c$. These results illustrate two key properties of the DP scheme. First, the innovation parameter $\alpha$ plays a key role in determining the number of distinct parameter values. A larger $\alpha$ induces a higher tendency of drawing new parameters from the base distribution $G_0$; indeed, as $\alpha \to \infty$ we get $G \to G_0$. Conversely, as $\alpha \to 0$ all $\{\Theta^*_m\}_{m=1}^M$ tend to cluster to a single random variable. Second, the more often a parameter is shared, the more likely it will be shared in the future.

A characterization of the (unconditional) distribution of the random variable $G$ drawn from a DP, $\text{DP}(\alpha, G_0)$, is provided by the stick-breaking construction of Sethuraman [28]. Consider two infinite collections of independent random variables $v = \{v_c\}_{c=1}^\infty$, $\{\Theta^*_c\}_{c=1}^\infty$, where the $v_c$ are drawn from a Beta distribution, and the $\Theta^*_c$ are independently drawn from the base distribution $G_0$. The stick-breaking representation of $G$ is then given by

$$G = \sum_{c=1}^\infty \varpi_c(v) \delta_{\Theta^*_c}$$

where

$$p(v_c) = \text{Beta}(v_c|1, \alpha)$$

$$\varpi_c(v) = v_c \prod_{j=1}^{c-1} (1 - v_j) \quad \in [0, 1]$$

and

$$\sum_{c=1}^\infty \varpi_c(v) = 1$$

Under the stick-breaking representation of the DP, the atoms $\Theta^*_c$, drawn independently from the base distribution $G_0$, can be seen as the parameters of the component distributions of a mixture model comprising an unbounded number of component densities, with mixing proportions $\varpi_c(v)$.

### B. Maximum entropy discrimination

Max-margin methods, such as SVMs [19] and max-margin Markov networks (M3N) [20], have been successfully applied to a wide range of discriminative problems. Their power and popularity stem from their provision of strong generalization guarantees [19], [20], as well as the fact that their inference and training reduce to convex optimization problems, thus not suffering from the possibility of getting stuck to spurious local optima, which is often the case with alternative approaches. Depending on the nature of the response variable, the max-margin principle can be exploited in both classification and regression settings.

To unite the principles of Bayesian inference and max-margin learning, a formalism known as MED has been proposed in [22], [21], which learns a distribution of all possible regression/classification models that belong to a particular parametric family, subject to a set of margin-based constraints. In the particular case of a regression setting, the MED regression model learns an approximate posterior distribution $q(w)$ through solving the following optimization problem:

$$\min_{q(w), \xi} \text{KL} (q(w)||p(w)) + \gamma \sum_{d=1}^D (\xi_d + \xi_d^*)$$

$$\forall d, \text{ s.t.: } \begin{cases} y_d - \mathbb{E}[w^T x_d] \leq \varepsilon + \xi_d \\ -y_d + \mathbb{E}[w^T x_d] \leq \varepsilon + \xi_d^* \\ \xi_d, \xi_d^* \geq 0 \end{cases}$$

where $p(w)$ is a prior distribution over the model parameters, $\text{KL} (q||p)$ is the Kullback-Leibler (KL) divergence between the posterior and the prior, $x_d$ are the observed input variables (regressors), $y_d$ the observed output (dependent) variables, $\{\xi_d, \xi_d^*\}$ are SVR-style slack-variables, and $\varepsilon$ is an SVR-style precision parameter.

As shown in [21], inference under the MED paradigm yields an entropic-regularized posterior distribution $q(w)$ over the regression model parameters. Due to this fact, the resulting predictive distribution enjoys strong generalization guarantees, and subsumes SVR as a special case, reducing to it when the prior $p(w)$ is selected to be a standard normal distribution. On the other hand, alternative selections of the prior $p(w)$ allow for the resulting model (and obtained posterior $q(w)$) to exhibit a wide variety of desired characteristics. This renders it suitable for diverse modeling tasks, ranging from feature selection to learning complex non-linear discriminating functions, and acting as a suitable prior for complex hierarchical Bayesian models [32], [33].

As an aside, note that MED yields approximate (variational) posterior distributions. Exact treatment of our hierarchical large-margin model could be possibly performed following a data augmentation scheme, first presented in [34]. This framework consists in introducing auxiliary latent variables...
that enable derivation of an exact Gibbs sampling scheme for large-margin hierarchical models. However, this framework entails considerably more tedious derivations to obtain the model posteriors, it requires performing inference for much more (latent) variables, and is considerably less scalable, imposing significantly higher computational burden. For these reasons, we opt for adopting the MED framework in this work.

### III. Proposed Approach

#### A. Motivation

Several software reliability modeling approaches rely on the assumption that times between successive failures are exponentially distributed. For instance, the seminal system of [2] is based on this assumption, while introducing the extra hypothesis that after each failure, the fault which caused the failure is perfectly corrected. Similar exponential distribution assumptions do also postulate the methods in [35], [36], but allowing for the possibility of imperfect debugging, thus relaxing the optimistic assumptions of [2].

However, in real-world application scenarios, it is often the case that exact failure times are not available for modeling purposes. Instead, only the numbers of failures \( y_i \) over fixed time periods of length \( t_i \) are available. Under such a modeling setup, the most plausible and computationally convenient assumption is to assume that the software is only corrected at the end of each time period. This is, indeed, the most typical assumption employed in the literature, and is particularly appropriate in the case of beta software releases, where beta users report failure types and a new release is produced after some time [16].

Due to its strong plausibility and non-restrictive modeling assumptions, in this paper we adopt this latter software reliability modeling approach. In many cases, with the \( i \)th release of the software, software metrics \( x_i \) are also made available. Such metrics may reflect characteristics of the code (e.g., number of lines), measures of the amount of the correction work (e.g., number of hours), or the numbers of faults discovered in previous releases. In fact, most modern software reliability methods suppose that changes in the quality of the code will be reflected in changes in the values of the software metrics. As such, they attempt to perform failure count modeling and prediction using software metrics as the independent observed variables (regressors) of their models. We adopt this approach in our work.

As extensively discussed in the literature (e.g., [36], [14], [15], [16]), observed failure counts over fixed time periods, at the end of which (imperfect) debugging may take place, are distributed according to a piecewise homogeneous Poisson process, i.e.

\[
y_i \sim P(t_i, \lambda_i)
\]

where \( \lambda_i \) is the failure rate for the \( i \)th release of the software, and is considered to be a function of the observed metrics data \( x_i \). Under such a modeling assumption, the most important problem that remains to be addressed is to choose the most appropriate way of learning the unknown function that maps the observed metrics data \( x_i \) to the failure rate \( \lambda_i \) for the corresponding time window. Most existing approaches try to address this issue by assuming a log-linear mapping, e.g. [12]. More recently, non-linear modeling approaches have also been considered, e.g. [14], [16].

In this work, we aim to devise a software reliability modeling approach, relying on a piecewise homogeneous Poisson process likelihood, as in (1), and leveraging the strengths of max-margin modeling to perform learning of the mapping from the observed metrics data \( x_i \) to the failure rates \( \lambda_i \). We seek a Bayesian treatment for our model, to allow for better taking into account the uncertainty in the modeled data; this is a significant merit in our setting, since the availability of appropriate model training samples is limited. For this purpose, we resort to the MED inference paradigm, as described in Section II.B. In addition, to capture underlying mappings from the observed metrics data \( x_i \) to the failure rates \( \lambda_i \) with multiple modes, we consider that the expression of \( \lambda_i \) as a function of \( x_i \) is not uniquely described by one latent max-margin regression model, but rather by an infinite set of possible latent max-margin regression models with different parameters. To determine the association between input samples and latent regression functions, we impose a Dirichlet process prior over this set of functions, as discussed in Section II.A.

#### B. Model formulation

Let us consider a set of input/output observation pairs \( \{x_i, y_i\}_{i=1}^N \), comprising \( N \) samples, where \( x_i \) are the software metrics pertaining to the \( i \)th release, and \( y_i \) are the corresponding failure counts, with corresponding time durations \( \{t_i\}_{i=1}^N \). We denote as \( \lambda_i \) the failure rate for the \( i \)th release of the software. Let us also introduce the set of variables \( \{z_{ic}\}_{i,c=1}^{N,\infty} \), with \( z_{ic} = 1 \) if the function relating \( x_i \) to \( \lambda_i \) is considered to be expressed by the \( c \)th postulated max-margin regression model, \( z_{ic} = 0 \) otherwise. Based on the above description, and the description of the MED regression model as well as the DP.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda_i )</td>
<td>Failure rates.</td>
</tr>
<tr>
<td>( \eta_i )</td>
<td>Failure log-rates.</td>
</tr>
<tr>
<td>( \omega_i )</td>
<td>Parameters of the component Poisson regression models.</td>
</tr>
<tr>
<td>( \zeta_{ic} )</td>
<td>Latent indicators of component assignment.</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>Innovation parameter of the DP.</td>
</tr>
<tr>
<td>( v_c )</td>
<td>Stick-breaking variables of the DP.</td>
</tr>
<tr>
<td>( \xi_{ic}, \xi_j )</td>
<td>Slack-variables of the large-margin inference problem.</td>
</tr>
<tr>
<td>( \theta_1, \theta_2 )</td>
<td>Hyperparameters of the prior over ( \alpha ).</td>
</tr>
<tr>
<td>( \omega_{ic, 1}, \omega_{ic, 2} )</td>
<td>Hyperparameters of the posterior over ( v_c ).</td>
</tr>
<tr>
<td>( \mu_i )</td>
<td>Posterior means of the ( \omega_i ).</td>
</tr>
<tr>
<td>( \tau_i )</td>
<td>Truncation threshold of the DP.</td>
</tr>
<tr>
<td>( x_i )</td>
<td>Software metrics pertaining to the ( i )th release.</td>
</tr>
<tr>
<td>( y_i )</td>
<td>Failure counts pertaining to the ( i )th release.</td>
</tr>
<tr>
<td>( t_i )</td>
<td>Time durations pertaining to the ( i )th release.</td>
</tr>
<tr>
<td>( N )</td>
<td>Number of training data points.</td>
</tr>
<tr>
<td>( E {\cdot} )</td>
<td>Expectation of a quantity w.r.t. the posterior ( q(\cdot) ).</td>
</tr>
<tr>
<td>( \mathbb{E} {\cdot} )</td>
<td>Expectation of a quantity w.r.t. all the applicable posteriors.</td>
</tr>
</tbody>
</table>
prior, provided in Section II.A, the hierarchical configuration of our proposed model is formulated as follows:

\[ p(Y) = \prod_{i=1}^{N} P(y_i|t_i, \lambda_i) \]  

(2)

For mathematical convenience, and similar to related works in the recent literature (e.g., [16]), we elect to impose a prior on the log-rates \( \eta_i \) of the failure rate \( \lambda_i \), which we denote as \( \eta_i \sim \log \Lambda_i \). Specifically, following the preceding discussion, we consider a DP-based mixture prior of the form

\[ p(\eta_i|z_{ic} = 1) = N(\eta_i|w_{c}^T x_i, \sigma^2) \]  

(3)

with a spherical hyperprior on the \( w_c \)

\[ p(w_c) = N(w_c|0, I) \]  

(4)

where \( \sigma^2 \) is the variance of some white noise model, \( Y = \{y_i\}_{i=1}^{N} \), and, following the presentation of Section II.A, the DP-related hyperpriors yield

\[ p(z_{ic} = 1|v) = \varpi_c(v) \]  

(5)

\[ \varpi_c(v) = v_c \prod_{j=1}^{c-1} (1 - v_j) \in [0,1] \]  

(6)

with

\[ \sum_{c=1}^{\infty} \varpi_c(v) = 1 \]  

(7)

and

\[ p(v_c) = \text{Beta}(v_c|1, \alpha) \]  

(8)

We also impose a Gamma hyperprior over the innovation parameter \( \alpha \) of the DP, reading

\[ p(\alpha) = G(\alpha|\theta_1, \theta_2) \]  

(9)

In addition, we stipulate that the postulated (conditional) linear regression scheme (3), that connects the latent failure log-rates \( \eta_i \) and the observed metrics data \( x_i \), be subject to the following expected max-margin constraints:

\[
\begin{align*}
\mathbb{E}[\eta_i|z_{ic} = 1] - \mathbb{E}[w_c^T x_i] & \leq \varepsilon + \xi_{ic} \\
-\mathbb{E}[\eta_i|z_{ic} = 1] + \mathbb{E}[w_c^T x_i] & \leq \varepsilon + \xi_{ic}^* \quad \forall i, c \\
\xi_{ic}, \xi_{ic}^* & \geq 0
\end{align*}
\]  

(10)

Our imposed constraints are inspired from max-margin approaches, and especially, the literature on MED regression models [21, Chapter 4], as they are based on maximization of an expected margin, that takes into account the Bayesian nature of our model. Note that, in Eq. (10), \( \varepsilon \) is a precision parameter, functioning similar to the precision parameter in SVR, and \( \xi_{ic}, \xi_{ic}^* \) are some slack-variables, used in a way similar to SVR [19].

This concludes the formulation of our MEDPR model.

### C. Inference algorithm

To perform inference for our model, we adopt the MED inference framework, and extend it by introducing an additional (negative) likelihood term into the optimized objective function, which emanates from the assumption (2) of our model.

Specifically, conventional MED inference in the context of our model would comprise solution of the following minimization problem:

\[
\min_{q(Z,v,\eta,w), \xi, \sigma^2} \text{KL}(q(Z,v,\eta,w)|p(Z,v,\eta,w)) + \gamma \sum_{i=1}^{N} (\xi_{ic} + \xi_{ic}^*)
\]  

under the constraints (10), where \( Z = \{z_{ic}\}_{i,c}, v = \{v_c\}_{c=1}^{\infty}, \eta = \{\eta_i\}_{i=1}^{N}, \xi = \{\xi_{ic}\}_{i,c}, \) and \( \xi^* = \{\xi_{ic}^*\}_{i,c}. \) However, in our work, we follow a different approach, inspired from variational Bayesian inference [37]: We elect to optimize a composite objective function that takes into consideration both the expected (negative) log-likelihood of our hierarchical Bayesian model, which measures the goodness of fit to the training data, as well as the quality of the reconstruction of our training data, as is also performed in the context of conventional approaches. This way, inference for our model eventually reduces to solution of the following problem

\[
\min_{q(Z,v,\eta,w), \xi, \sigma^2} -\sum_{i=1}^{N} \mathbb{E}[\log p(y_i|t_i, \lambda_i)] + \gamma \sum_{c=1}^{N} (\xi_{ic} + \xi_{ic}^*)
\]  

+ \text{KL}(q(Z,v,\eta,w)|p(Z,v,\eta,w))

\[
\forall i, c, \text{ s.t. : } \begin{cases}
\mathbb{E}[\eta_i|z_{ic} = 1] - \mathbb{E}[w_c^T x_i] \leq \varepsilon + \xi_{ic} \\
-\mathbb{E}[\eta_i|z_{ic} = 1] + \mathbb{E}[w_c^T x_i] \leq \varepsilon + \xi_{ic}^* \\
\xi_{ic}, \xi_{ic}^* \geq 0
\end{cases}
\]  

(12)

Note that, in the above expressions, all the expectations \( \mathbb{E}[\cdot] \) are computed w.r.t. the posterior \( q(Z,v,\eta,w) \).

Our inference algorithm proceeds in an iterative fashion, under the mean-field principle [38]: On each iteration, we consecutively minimize (12) over each one of the factors of the sought posterior \( q(Z,v,\eta,w) \), as well as the noise variance \( \sigma^2 \), and the slack-variables \( \xi, \xi^* \), one at a time, holding the others fixed. It has been shown that such an iterative consecutive updating procedure is guaranteed to monotonically optimize the objective function of our problem [22]. An outline of the inference algorithm of our model presented next is depicted in Algorithm 1.

1) Posterior Distribution: Let us begin with deriving the expression of the posterior distribution of our model. Following the mean-field principle [38], we assume that the sought posterior distribution factorizes over the \( Z, v, \eta, w \), similar to the imposed prior. Then, attempting to minimize (12) w.r.t. \( q(v) \), we initially obtain an infeasible problem, due to the infinite-dimensional nature of the postulated set \( v \). For this reason, similar to [39], we resort to a truncation-based approximation; this is obtained by replacing the infinite-dimensional set \( v \) by a finite set defined over \( C \) atoms. As discussed in [39], it can be shown that the total variation...
Algorithm 1 Outline of the inference algorithm of our model.

**Input:** \( N \) input/output observation pairs \( \{x_i, y_i\}_{i=1}^{N} \), where \( x_i \) are the software metrics pertaining to the \( i \)th release, and \( y_i \) are the corresponding failure counts, with corresponding time durations \( \{t_i\}_{i=1}^{N} \).

**Output:** The model posteriors \( q(v_c) \) \( \forall c \), \( q(\alpha) \), \( q(z_{ic}) = 1 \) \( \forall i, c \), the mean parameters \( \mu_c \) of the posteriors \( q(w_c) \) \( \forall c \), a set of samples (approximately) drawn from the posteriors \( q(\eta_i | z_{ic} = 1) \) \( \forall i, c \), and estimates of the noise variance \( \sigma^2 \).

**Algorithm:** For a preset number of iterations MAXITER, or until convergence of (12), do:

1. Update the (hyperparameters of the) posterior over \( q(v_c) \) \( \forall c \), using (14)-(15).
2. Update the (hyperparameters of the) posterior over \( q(\alpha) \), using (17)-(18).
3. Update the posterior over the mixture component indicators \( q(z_{ic} = 1) \) \( \forall i, c \), using (20).
4. Update the posterior over the weight parameters \( w_c, \forall c \), by using (26) and solving the set of convex problems (27).
5. Draw conditional posterior samples of the log-rates \( \eta_i \), by means of HMC, using the expression of \(-\log q(\eta_i | z_{ic} = 1)\), given by (28), as the potential energy of the HMC algorithm.
6. Update noise variance, using (29).
7. Substitute the updates above into (12) to obtain the new value of our objective function, and check convergence.

In the same vein, the latent mixture component indicators \( Z = \{z_{ic}\}_{i,c} \) yield

\[
q(z_{ic} = 1) \propto \exp (\mathbb{E}[\log w_c(v)]) \exp (\varphi_{ic})
\]

where

\[
\mathbb{E}[\log w_c(v)] = \sum_{c' = 1}^{c-1} \mathbb{E}[\log(1 - v_{c'})] + \mathbb{E}[\log v_c]
\]

with (c.f., [40, Appendix B])

\[
\mathbb{E}[\log v_c] = \psi(\beta_{c,1}) - \psi(\beta_{c,1} + \beta_{c,2})
\]

and

\[
\varphi_{ic} = -\frac{1}{2\sigma^2} \left( \mathbb{E}[\eta_i^2 | z_{ic} = 1] + x_i^T \mathbb{E}[w_c w_c^T] x_i - 2 \mathbb{E}[\eta_i | z_{ic} = 1] \mathbb{E}[w_c^T] x_i \right)
\]

Subsequently, optimizing (12) over the posteriors of the parameters \( w_c \) of the component max-margin regression models, we yield

\[
q(w_c) = N(w_c | \mu_c, \Sigma_c)
\]

and the means \( \mu_c \) are obtained by solving (\( \forall c = 1, \ldots, C \)) the primal problem

\[
\min_{\mu_c, \xi_c} \frac{1}{2} \mu_c^T \Sigma_c^{-1} \mu_c + \gamma \sum_{i=1}^{N} (\xi_{ic} + \xi_{ic}^*)
\]

subject to:

\[
\begin{align*}
\mathbb{E}[\eta_i | z_{ic} = 1] - \mu_c^T x_i & \leq \varepsilon + \xi_{ic} \\
-\mathbb{E}[\eta_i | z_{ic} = 1] + \mu_c^T x_i & \leq \varepsilon + \xi_{ic}^* \\
\xi_{ic}, \xi_{ic}^* & \geq 0
\end{align*}
\]

where \( \xi_c = \{\xi_{ic}\}_{i=1}^{N}, \xi_c^* = \{\xi_{ic}^*\}_{i=1}^{N} \). The problem (27) comprises \( C \) subproblems, each one of which can be easily solved by leveraging highly-efficient solvers developed for quadratic programming problems. Indeed, in our work, we use CVX, a package for specifying and solving convex programs [41], [42]. In addition, the problems implied by (27) can be executed in parallel, by leveraging available parallel processing infrastructure. Note also that, from (25), we have

\[
\mathbb{E}[w_c] \Delta \mu_c
\]

\[
\mathbb{E}[w_c w_c^T] = \mu_c \mu_c^T + \Sigma_c
\]

Finally, regarding the posteriors over the latent failure log-rates \( \eta_i \), optimization of (12) yields

\[
\log q(\eta_i | z_{ic} = 1) \propto -e^{\eta_i} t_i + \eta_i (y_i - 1) - \frac{1}{2\sigma^2} (\eta_i^2 - 2\eta_i \mu_c^T x_i)
\]
From (28), it becomes apparent that model configuration is not conjugate when it comes to the latent variables $\eta_i$. As a consequence, the model does not yield a closed-form expression for the posteriors $q(\eta_i | z_{ic} = 1)$. A repercussion of this fact is that the entailed posterior expectations w.r.t. $q(\eta_i | z_{ic} = 1)$ cannot be computed in an analytical fashion, wherever they appear in the inference algorithm of our model (namely, the max-margin constraints (27), and the quantities $\varphi_{ic}$ in (24)). To resolve this issue, we can either resort to a deterministic approximation of the posteriors $q(\eta_i | z_{ic} = 1)$, e.g., by application of Laplace approximation, or perform sampling by means of MCMC. Our investigations have shown that Laplace approximation does not perform very well for our model. For this reason, in this paper we opt for the latter alternative.

Specifically, for this purpose we draw samples from the variational posterior (28) using hybrid Monte Carlo (HMC) [43]. HMC provides an efficient method to draw samples from the variational posterior distribution $q(\eta_i | z_{ic} = 1)$ by performing a physical simulation of an energy-conserving system to generate proposal moves. In detail, we add “kinetic energy” variables, $p_i$, for each latent dimension, while the expression of $-\log q(\eta_i | z_{ic} = 1)$ is used to specify a potential energy over the latent variables. Each HMC step involves sampling $p_i$ and carrying out a physics-based simulation using “leap-frog” discretization. The final state of the simulation is accepted or rejected based on the Metropolis-Hastings algorithm. HMC sampling of $\eta_i$ requires computation of the gradient of $q(\eta_i | z_{ic} = 1)$, which can be analytically performed in a straightforward manner.

2) Hyperparameter estimation: To conclude, let us now turn to the estimates of the parameters of our model over which we didn’t impose some prior distribution, that is the noise variance $\sigma^2$. Substituting (2)-(9) into (12) and taking derivatives w.r.t. $\sigma^2$, we directly obtain (after some simple algebra)

$$
\sigma^2 = \frac{1}{NC} \sum_{i=1}^{N} \sum_{c=1}^{C} q(z_{ic} = 1) \left( \mathbb{E}[\eta_i^2 | z_{ic} = 1] - 2 \mathbb{E}[\eta_i | z_{ic} = 1] \right)
$$

(29)

IV. EXPERIMENTS

We perform an extensive evaluation of our approach based on the benchmark dataset presented in [29], which offers a baseline against which bug prediction approaches can be compared. This benchmark is composed of datasets pertaining to several open-source object-oriented software systems; our experiments include Eclipse JDT Core, Equinox framework, and Mylyn. It provides: (i) process metrics on all the files of each system; (ii) system metrics on bi-weekly versions of each system; (iii) defect information related to each system file; (iv) bi-weekly models of each system version if new metrics need to be computed; and (v) measurements of change entropy [8] applied to source code metrics. An outline of the characteristics of the considered software systems is provided in Table II.

In all our experiments, comparison of different bug prediction approaches is performed in the following way: Given a release of a software system, the task is to predict, for each class of the system, the number of post-release defects, defined as the number of defects reported from the release date to six months later. Similar to [29], we use the last system release in the release period, and perform class-level defect prediction, rather than package- or subsystem-level defect prediction. In addition, we use post-release defects for validation (i.e., not all defects in the history) to emulate a real-world scenario. Following [29], we consider different combinations of extracted software metrics as the used independent variables, which comprise the following:

**Change Metrics.** These are sets of file-level change metrics, extracted as suggested in [44]. Specifically, we consider the case of using all the relevant metrics suggested in [44], henceforth referred to as MOSER setup, the number of previously detected defects, henceforth referred to as NFIX, the number of revisions, henceforth referred to as NR, and the combination of NFIX and NR, referred to as NFIX+NR. In addition, we also consider the BUG-CATEGORIES setup, which uses as metrics the numbers of previously reported bugs belonging to five categories, namely all bugs, non-trivial bugs, major bugs, critical bugs (critical or blocker severity), and high priority bugs.

**Source Code Metrics.** These features are based on the CK metrics suite [9], with the optional addition of some further object-oriented (OO) metrics. These latter metrics comprise: number of other classes that reference the class, number of other classes referenced by the class, number of attributes, number of public attributes, number of private attributes, number of attributes inherited, number of lines of code, number of methods, number of public methods, number of private methods, and number of methods inherited. In our experiments, we consider using only the CK metrics suite, henceforth referred to as CK setup, using only our object-oriented metrics, henceforth referred to as OO, using both of them, henceforth referred to as CK+OO, and using only the number of code lines as our metric, henceforth referred to as LOC.

**Entropy of Changes Metrics.** These features are based on measuring how distributed changes are in a system over a time interval. This is effected by computing the Shannon entropy of code change, following the approach presented in [8]. The more spread the changes are, the higher is the complexity. The intuition is that one change affecting one file only is simpler than one affecting many different files, as the developer who has to perform the change has to keep track of all of them. We refer to this experimental setup as HCM. We also consider the variants of HCM suggested in [8], namely WHCM, which weights the entropy of the system with the probability of each file being modified, EDHCM, which exponentially reduces the contribution of the entropies for earlier periods of time, as well as LDHCM, which applies a similar linear decay operator, instead of an exponential one.

**Entropy of Source Code Metrics.** These metrics, proposed in [29], extend the concept of code change entropy to the CK source code metrics outlined above. Similar to metrics
based on code change entropy measurements, apart from the main scenario (henceforth referred to as HH), we also consider a weighed variant (HWH), an exponentially decaying variant (EDHH), and a linearly decaying variant (LDHH), following the approach outlined in [29].

In a vein similar to [45], in cases of experimental setups using multiple software metrics as the independent variables (regressors), we postprocess our data means by principal component analysis (PCA) [11], and build the evaluated count data models using the resulting feature vectors. This way, we ameliorate the problem of multicollinearity among the independent variables, which could lead to inflated predictive variances.

Prediction using our model is performed by utilizing the posterior expectation of the predictive distribution of the model. This boils down to using the inferred posterior mean of the failure log-rate of our model at some time period \( i \) as the logarithm of the predicted number of defects \( \hat{y}_i \), yielding

\[
\hat{y}_i = \exp\left( \mathbb{E}[\eta_i] \right) \approx \sum_{c=1}^{C} \exp\left( \mathbb{E}[\log \omega_c(v)] \right) \exp\left( \mathbb{E}[\eta_i | z_{ic} = 1] \right)
\]

(considering unitary time durations, \( t_i = 1 \)). This is computed using samples of \( \eta_i \) drawn from the posteriors (28) by means of HMC sampling, similar to the inference algorithm of our model. HMC sampling of the failure log-rate of our model at some time period \( i \) is run for 100 iterations, with 25 leap-frog steps at each iteration, and with a step-length equal to 0.001\( \sqrt{N} \), where \( N \) is the number of modeled data points.

In all our experimental evaluations, we perform cross-validation in the following sense: we use 90\% of the dataset (i.e., 90\% of the available classes) to perform model training, and the rest for accuracy testing. To alleviate the effect of random training and test samples selection, we repeat our experiments 10 times, with different training-test set splits each time, and we report mean performances and standard deviations. Apart from our method, we also evaluate the baseline pGLM model [10], which is used as a key-component of a number of classical software reliability modeling approaches. Note that our method reduces to pGLM if one removes the MED-type constraints, and the associated prior assumptions. We also compare to the recently proposed, state-of-the-art Bayesian inference approach, namely the Gaussian process (GP)-based homogeneous Poisson process model [16] (henceforth denoted as GP), as well as a standard feedforward neural network-based modeling approach (henceforth referred to as NN [46]).

In Tables Va-Vd, we depict the obtained performance of the evaluated algorithms when using \textit{entropy of changes metrics} as the independent variables for model training and prediction generation. As we observe, our approach yields better average performance than pGLM in all cases, and better average performance than GP in most cases, with only two observed exceptions to this rule. In addition, the observed performance differences are deemed statistically significant in all scenarios, in the case of pGLM, and in most scenarios in the case of GP.

In Tables Va-Vd, we depict the obtained performance of the evaluated algorithms when using \textit{source code metrics} as the independent variables for model training and prediction generation. As we observe, our approach yields better average performance than pGLM in all cases, and better average performance than GP in most cases, with only two observed exceptions to this rule. In addition, the observed performance differences are deemed statistically significant in all scenarios, in the case of pGLM, and in most scenarios in the case of GP.

### Table II: Summary of characteristics of the considered benchmark systems.

<table>
<thead>
<tr>
<th>System</th>
<th>Prediction</th>
<th>Release</th>
<th>Time period</th>
<th>#Classes</th>
<th>#Versions</th>
<th>#Transactions</th>
<th>#Post-rel. defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse JDT Core</td>
<td>3.4</td>
<td>1.1.2005-6.17.2008</td>
<td>997</td>
<td>91</td>
<td>9,135</td>
<td>463</td>
<td></td>
</tr>
<tr>
<td>Equinox framework</td>
<td>3.4</td>
<td>1.1.2005-6.25.2008</td>
<td>439</td>
<td>91</td>
<td>1,616</td>
<td>279</td>
<td></td>
</tr>
<tr>
<td>Mylyn</td>
<td>3.4.1</td>
<td>1.17.2005-3.17.2009</td>
<td>2,196</td>
<td>98</td>
<td>9,189</td>
<td>677</td>
<td></td>
</tr>
</tbody>
</table>
Table III: Change metrics-based evaluation: Obtained performance of the evaluated methods: (a) MOSER setup. (b) NFIX setup. (c) NR setup. (d) NFIX+NR setup. (e) BUG-CATEGORIES setup.

<table>
<thead>
<tr>
<th>System</th>
<th>pGLM</th>
<th>GP</th>
<th>MEDPR</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse JDT Core</td>
<td>1.3686 (0.3338)</td>
<td>1.0754 (0.1254)</td>
<td>1.0028 (0.0896)</td>
<td>1.2081 (0.5617)</td>
</tr>
<tr>
<td>Equinox framework</td>
<td>1.3254 (0.5280)</td>
<td>1.2384 (0.3020)</td>
<td>0.9410 (0.2280)</td>
<td>1.7358 (0.5631)</td>
</tr>
<tr>
<td>Mylyn</td>
<td>1.1699 (0.1943)</td>
<td>1.1012 (0.1483)</td>
<td>0.9292 (0.0389)</td>
<td>1.2442 (0.3860)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse JDT Core</td>
<td>1.5259 (0.2986)</td>
<td>1.2963 (0.6992)</td>
<td>0.6557 (0.1486)</td>
<td>1.2777 (0.5194)</td>
</tr>
<tr>
<td>Equinox framework</td>
<td>1.1262 (0.3685)</td>
<td>1.0226 (0.2763)</td>
<td>0.9020 (0.3063)</td>
<td>1.9434 (0.3993)</td>
</tr>
<tr>
<td>Mylyn</td>
<td>1.1334 (0.1585)</td>
<td>0.6678 (0.2305)</td>
<td>0.5496 (0.1129)</td>
<td>1.0681 (0.3310)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse JDT Core</td>
<td>1.4749 (0.4259)</td>
<td>0.9389 (0.2412)</td>
<td>0.6535 (0.0740)</td>
<td>1.5813 (0.5619)</td>
</tr>
<tr>
<td>Equinox framework</td>
<td>1.1698 (0.2385)</td>
<td>1.1492 (0.2576)</td>
<td>0.8947 (0.1791)</td>
<td>1.2820 (0.5216)</td>
</tr>
<tr>
<td>Mylyn</td>
<td>1.0337 (0.2942)</td>
<td>0.5809 (0.2021)</td>
<td>0.5030 (0.1157)</td>
<td>1.0584 (0.3131)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse JDT Core</td>
<td>1.6517 (0.4501)</td>
<td>1.0266 (0.1357)</td>
<td>1.0251 (0.0967)</td>
<td>1.3163 (0.3754)</td>
</tr>
<tr>
<td>Equinox framework</td>
<td>1.3944 (0.5972)</td>
<td>1.3390 (0.4257)</td>
<td>1.1540 (0.4184)</td>
<td>1.8663 (0.5879)</td>
</tr>
<tr>
<td>Mylyn</td>
<td>1.3622 (0.3695)</td>
<td>1.1982 (0.5560)</td>
<td>0.6521 (0.1879)</td>
<td>1.0584 (0.5487)</td>
</tr>
</tbody>
</table>

Table IV: Source code metrics-based evaluation: Obtained performance of the evaluated methods: (a) CK setup. (b) OO setup. (c) CK+OO setup. (d) LOC setup.

<table>
<thead>
<tr>
<th>System</th>
<th>pGLM</th>
<th>GP</th>
<th>MEDPR</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse JDT Core</td>
<td>1.0968 (0.1868)</td>
<td>1.0293 (0.3987)</td>
<td>1.0027 (0.0740)</td>
<td>1.3557 (0.5532)</td>
</tr>
<tr>
<td>Equinox framework</td>
<td>1.0809 (0.1907)</td>
<td>0.9774 (0.2462)</td>
<td>1.0151 (0.0930)</td>
<td>1.4713 (0.4802)</td>
</tr>
<tr>
<td>Mylyn</td>
<td>1.1636 (0.3930)</td>
<td>1.1031 (0.1391)</td>
<td>1.0836 (0.0730)</td>
<td>1.3353 (0.2319)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse JDT Core</td>
<td>1.3111 (0.3556)</td>
<td>0.8875 (0.2707)</td>
<td>0.9991 (0.1243)</td>
<td>1.2818 (0.4879)</td>
</tr>
<tr>
<td>Equinox framework</td>
<td>1.1776 (0.2454)</td>
<td>0.9322 (0.1469)</td>
<td>0.9913 (0.0649)</td>
<td>1.2658 (0.1485)</td>
</tr>
<tr>
<td>Mylyn</td>
<td>1.1903 (0.2129)</td>
<td>1.1031 (0.1391)</td>
<td>1.0836 (0.770)</td>
<td>1.2765 (0.3963)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse JDT Core</td>
<td>1.2686 (0.3311)</td>
<td>1.1310 (0.3875)</td>
<td>1.0910 (0.1578)</td>
<td>1.2508 (0.3476)</td>
</tr>
<tr>
<td>Equinox framework</td>
<td>1.1176 (0.2454)</td>
<td>1.0595 (0.2084)</td>
<td>0.9915 (0.0649)</td>
<td>1.5281 (0.3751)</td>
</tr>
<tr>
<td>Mylyn</td>
<td>1.4903 (0.2038)</td>
<td>0.9793 (0.0967)</td>
<td>0.9604 (0.0913)</td>
<td>1.3843 (0.2107)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse JDT Core</td>
<td>1.4403 (0.4578)</td>
<td>0.8552 (0.1841)</td>
<td>0.5630 (0.0987)</td>
<td>1.4692 (0.4823)</td>
</tr>
<tr>
<td>Equinox framework</td>
<td>1.4948 (0.4753)</td>
<td>1.0907 (0.3609)</td>
<td>0.5908 (0.1494)</td>
<td>1.3475 (0.4650)</td>
</tr>
<tr>
<td>Mylyn</td>
<td>1.3622 (0.5360)</td>
<td>1.1982 (0.3699)</td>
<td>0.6521 (0.1879)</td>
<td>1.1042 (0.2585)</td>
</tr>
</tbody>
</table>
prediction generation. As we observe, our approach yields better average performance than pGLM in all cases, and better average performance than GP in most cases, with only two observed exceptions to this rule. In addition, the observed performance differences are deemed statistically significant in all scenarios, in the case of pGLM, and in most scenarios in the case of GP.

E. General observations

As we observed from the above presentation, our approach yields a statistically significant improvement over the competition in most cases, with the notable exception of the CK and OO metrics, where all methods yield disappointing results. Quite characteristic of this improvement is also Table VII, where we depict the best obtained performance of our approach for each one of the considered software systems, and how it compares to GP. It is apparent that our approach offers clear and consistent performance gains for all systems. Another interesting finding is that, on average, the NN-based approach is the worst performer in all the considered scenarios.

Finally, it is interesting to examine how our method compares to the competition in terms of computational costs. As we observed, in our unoptimized MATLAB implementation, average execution time of our MEDPR algorithm (until convergence) was 57.27 seconds, with GP taking 63.85 seconds, and pGLM requiring 16.09 seconds. Based on these results, we can claim that our approach offers a favorable performance-complexity trade-off compared to what the competition offers.

V. CONCLUSIONS AND FUTURE RESEARCH

In this paper, we addressed the problem of metrics-based software reliability modeling by introducing a hierarchical Bayesian model for regression modeling of counts. Our method uses max-margin principles to learn the function mapping software metrics to predicted counts of bugs. It is based on a doubly stochastic homogeneous Poisson process formulation, where the failure rate parameter at each time point is modeled through a mixture prior of max-margin regression models. Application of the max-margin learning principle allows for obtaining a more discriminative learning technique, making more effective use of our training data during inference. In addition, the utilization of a mixture of max-margin regression priors, each one modeling a different (latent) subspace of the parameter space, allows for learning multimodal underlying data distributions with increased flexibility compared to single-component models.

We imposed a truncated DP prior over the mixture components, to allow for data-driven determination of their optimal number in an elegant way. To introduce the max-margin learning principle in the context of our hierarchical Bayesian model, we performed inference under the MED framework. This way, we yielded a mean-field inference algorithm with SVR-style max-margin constraints, amenable to highly-efficient solvers developed for the standard SVR problem. We evaluated our model using a publicly available benchmark dataset, comprising a set of alternative metric vectors, extracted from several well-known open-source software systems. As we showed, our method yields a significant improvement over baseline alternatives and recently proposed state-of-the-art methods.

This research was initiated with the aim to better address the challenges relating to software reliability modeling. Failure prediction is one of the most significant problems in the field of software engineering, especially when it comes to the development of safety-critical systems, or systems with limited fault resilience. Clearly though, the introduced MEDPR model...
is a generic method for count regression, with its applicability not limited to software fault prediction.

Two of the application areas that could also benefit from our approach are financial modeling, and computational methods for insurance strategy development, to name just a few. For instance, our model could be used to perform prediction of the total number of claims for insured automobiles, based on suitable measurements pertaining to each user (independent variables). Investigation of the efficacy of our approach in such application areas constitutes part of our intended future research work.

Finally, we also consider application of our model to the case of a distributed system. In such a case, our model can be applied as it is currently formulated, with only a modification to its inference algorithm. For the latter purpose, one can resort to some MCMC sampling procedure, such as HMC (adopted here).

With these facts in mind, let us begin with the posterior over $v_c$, given by (13). Substituting (2)-(9) into the objective function (12), and retaining only the terms that contain $v_c$, (12) reduces to

$$
\mathcal{L} = E_{q(v_c)} \left[ \log q(v_c) - \log p(v_c) - \sum_{i=1}^{N} q(z_{ic} = 1) \log v_c 
\right]
\,$$

$$- \sum_{c'=c+1}^{C} \sum_{i=1}^{N} q(z_{ic'} = 1) \log (1 - v_c)$$

Then, taking the derivative of $\mathcal{L}$ w.r.t. $q(v_c)$, we have

$$\frac{\partial \mathcal{L}}{\partial q(v_c)} = 0 \Rightarrow$$

$$\log q(v_c) \propto E_{q(z_{ic} \mid \alpha)} [\log p(v_c)] + \sum_{i=1}^{N} q(z_{ic} = 1) \log v_c$$

$$+ \sum_{c'=c+1}^{C} \sum_{i=1}^{N} q(z_{ic'} = 1) \log (1 - v_c)$$

APPENDIX

Here, we provide a sketch of the derivations of the posterior distribution expressions of Section III.C. For more details on the technicalities of variational inference, the interested reader may refer to [40, Chapter 10]; for the analytical forms of common distributions such as Beta, Gamma, Gaussian, etc., refer to [40, Appendix B].

To begin with, note that the way we have selected the imposed priors results in a conjugate exponential formulation for our model, with the only exception being the log-rate parameters $\eta_i$. As such, the sought model posteriors take the same form as the imposed priors [38], except for the case of $\eta_i$. In this latter case, yielding an exact closed-form expression of the posterior in an analytical fashion is not possible, as we explained in Section III.C. For this reason, one can resort to some MCMC sampling procedure, such as HMC (adopted here).

Table VI: Source code metrics entropy-based evaluation: Obtained performance of the evaluated methods: (a) HH setup. (b) HWH setup. (c) EDHH setup. (d) LDHH setup.

<table>
<thead>
<tr>
<th>System</th>
<th>pGLM</th>
<th>GP</th>
<th>MEDPR</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse JDT Core</td>
<td>1.5051</td>
<td>0.7748</td>
<td>0.6321</td>
<td>1.2751</td>
</tr>
<tr>
<td>Equinox framework</td>
<td>1.0147</td>
<td>0.9253</td>
<td>0.9710</td>
<td>1.6836</td>
</tr>
<tr>
<td>Mylyn</td>
<td>1.2552</td>
<td>0.6036</td>
<td>0.4743</td>
<td>1.2468</td>
</tr>
<tr>
<td>(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse JDT Core</td>
<td>1.3538</td>
<td>0.8217</td>
<td>0.9258</td>
<td>1.3337</td>
</tr>
<tr>
<td>Equinox framework</td>
<td>1.3134</td>
<td>1.2234</td>
<td>1.0762</td>
<td>1.4172</td>
</tr>
<tr>
<td>Mylyn</td>
<td>0.9482</td>
<td>0.6185</td>
<td>0.6048</td>
<td>0.3957</td>
</tr>
<tr>
<td>(c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse JDT Core</td>
<td>1.2204</td>
<td>0.8773</td>
<td>0.8277</td>
<td>1.1447</td>
</tr>
<tr>
<td>Equinox framework</td>
<td>1.5710</td>
<td>1.5171</td>
<td>1.1597</td>
<td>2.8912</td>
</tr>
<tr>
<td>Mylyn</td>
<td>1.0895</td>
<td>0.5462</td>
<td>0.4046</td>
<td>1.3043</td>
</tr>
<tr>
<td>(d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse JDT Core</td>
<td>1.2291</td>
<td>0.9451</td>
<td>0.7067</td>
<td>1.3043</td>
</tr>
<tr>
<td>Equinox framework</td>
<td>1.2430</td>
<td>1.7676</td>
<td>1.1500</td>
<td>1.2421</td>
</tr>
<tr>
<td>Mylyn</td>
<td>1.0502</td>
<td>0.8817</td>
<td>0.4838</td>
<td>1.2492</td>
</tr>
</tbody>
</table>

Table VII: Best performances for the considered systems.

<table>
<thead>
<tr>
<th>System</th>
<th>GP</th>
<th>MEDPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse JDT Core</td>
<td>0.7748</td>
<td>0.5630</td>
</tr>
<tr>
<td>Equinox framework</td>
<td>0.9237</td>
<td>0.5908</td>
</tr>
<tr>
<td>Mylyn</td>
<td>0.5462</td>
<td>0.2125</td>
</tr>
</tbody>
</table>
\[ \Rightarrow \log q(v_c) \propto (E_{q(\alpha)}[\alpha] - 1) \log (1 - v_c) + \sum_{i=1}^{N} q(z_{ic} = 1) \log v_c + \sum_{c' = c+1}^{C} \sum_{i=1}^{N} q(z_{ic'} = 1) \log (1 - v_c) \]
from which we directly obtain expressions (13)-(15).

Proceeding to the posterior \( q(\alpha) \), substituting (2)-(9) into (12) and retaining only the terms that contain \( \alpha \), our objective function reduces to
\[ \mathcal{L} = E_{q(v_c), q(\alpha)} \left[ \log q(\alpha) - \log p(\alpha) - \sum_{c=1}^{C} \log p(v_c) \right] \]

Then, taking the derivative of \( \mathcal{L} \) w.r.t. \( q(\alpha) \), we have
\[ \frac{\partial \mathcal{L}}{\partial q(\alpha)} = 0 \Rightarrow \log q(\alpha) = -\sum_{c=1}^{C} E_{q(v_c)} \left[ \log q(v_c) \right] + \log p(\alpha) \]
\[ \Rightarrow \log q(\alpha) \propto \sum_{c=1}^{C} E_{q(v_c)} \left[ \log (1 - v_c) \right] + \sum_{c=1}^{C-1} \log q(z_{ic} = 1) \log \alpha + (\theta_1 - 1) \log \alpha - \theta_2 \alpha \]

where [40, Appendix B]
\[ E_{q(v_c)} \left[ \log (1 - v_c) \right] = \psi(\beta_{c,2}) - \psi(\beta_{c,1} + \beta_{c,2}) \]
from which we directly obtain (16)-(18).

Further, substituting (2)-(9) into (12) and retaining only the terms that contain the indicators \( z_{ic} \), our objective function reduces to
\[ \mathcal{L} = q(z_{ic} = 1) E_{q(v_c), q(w)} \left[ \log q(z_{ic} = 1) \right. \\
\left. - \log q(z_{ic} = 1) - \log N(\theta_{1} | w_c^T x_{i1}, \sigma^2) \right] \]

Then, taking derivatives w.r.t. \( q(z_{ic} = 1) \), similar to the previous derivations, we directly obtain (20). The expression of the posteriors \( \log q(\theta_i | z_{ic} = 1) \) can be obtained in the same vein.

To conclude, we turn to the mean parameters \( \mu_c \) of the posteriors \( q(w_c) \). These equations can be straightforwardly obtained from (12) by setting \( E[w_c] = \mu_c \), and discarding from the objective function any terms not pertaining to \( \mu_c \).

REFERENCES
Sotirios P. Chatzis received the M.Eng. (Hons.) degree in electrical and computer engineering and the Ph.D. degree in machine learning from the National Technical University of Athens, Athens, Greece, in 2005 and 2008, respectively. He was a Post-Doctoral Fellow with the University of Miami, Coral Gables, FL, USA, from 2009 to 2010. He was a Post-Doctoral Researcher with the Department of Electrical and Electronic Engineering, Imperial College London, London, U.K., from 2010 to 2012. He is currently an Assistant Professor with the Department of Electrical Engineering, Computer Engineering and Informatics, Cyprus University of Technology, Limassol, Cyprus. He has authored more than 50 papers in the most prestigious journals and conferences of the research field in his first seven years as a Researcher. His current research interests include machine learning theory and methodologies, specifically hierarchical Bayesian models, Bayesian nonparametrics, and deep hierarchical feature extractors, with a focus on modeling data with temporal dynamics. His Ph.D. research was supported by the Bodossaki Foundation, Greece, and the Greek Ministry for Economic Development. Dr. Chatzis was a recipient of the Dean’s scholarship for Ph.D. studies, being the best performing Ph.D. student of the class.

Andreas S. Andreou is an Associate Professor and Vice Chair of the Department of Electrical Engineering/Computer Engineering and Informatics of the Cyprus University of Technology (CUT). Prior to this appointment he was elected Lecturer and Assistant Professor at the Dept. of Computer Science of the University of Cyprus. He is also the director of the Software Engineering and Intelligent Information Systems Research Lab at CUT. He studied Computer Engineering and Informatics at the University of Patras, Greece (Diploma, 1993, Ph.D., 2000). Prior to joining the academia he worked in the industry at the posts of Programmer-Analyst, Director of Requirements Analysis, and IT Consultant in Banking Systems. He also served as Software Engineering and IT consultant in several major software projects in Cyprus, including the Integrated Software System for the New Nicosia General Hospital. His research interests include Software Engineering, Web Engineering, Electronic and Mobile Commerce, and Intelligent Information Systems.