

Exploiting Relations, Sojourn-Times, and Joint Conditional Probabilities for Automated Commit Classification

Sebastian Hönel^a

Department of Computer Science and Media Technology, Linnaeus University, Växjö, Sweden

Keywords: Software Maintenance, Repository Mining, Maintenance Activities.

Abstract: The automatic classification of commits can be exploited for numerous applications, such as fault prediction, or determining maintenance activities. Additional properties, such as parent-child relations or sojourn-times between commits, were not previously considered for this task. However, such data cannot be leveraged well using traditional machine learning models, such as Random forests. Suitable models are, e.g., Conditional Random Fields or recurrent neural networks. We reason about the Markovian nature of the problem and propose models to address it. The first model is a generalized dependent mixture model, facilitating the Forward algorithm for 1st- and 2nd-order processes, using maximum likelihood estimation. We then propose a second, non-parametric model, that uses Bayesian segmentation and kernel density estimation, which can be effortlessly adapted to work with nth-order processes. Using an existing dataset with labeled commits as ground truth, we extend this dataset with relations between and sojourn-times of commits, by re-engineering the labeling rules first and meeting a high agreement between labelers. We show the strengths and weaknesses of either kind of model and demonstrate their ability to outperform the state-of-the-art in automated commit classification.


1 INTRODUCTION

The automated classification of changes, as occurring during software evolution, is the subject of numerous studies, e.g., (Purushothaman and Perry, 2005; Fluri et al., 2007). Change is an omnipresent concept in the lifespan of almost any software. Furthermore, software must change, otherwise, it will lose its value over time, it *rots* or *decays* (Eick et al., 2001). If technical debt is only accumulated but not paid back, the decay will eventually result in obsolescence. Lehman's first law of software evolution—*Continuing Change*—is thus essential for proper software maintenance.

Changes provide valuable insight into, e.g., the development process itself (Hindle et al., 2009). Others exploit it for, e.g., fault localization or -prediction (Purushothaman and Perry, 2005; Bell et al., 2011; Shivaji et al., 2013). Change is rarely documented well, if at all. The rationale for and the reason behind changes, if not accessible, is hardly comprehensible. Change may be obtained from a number of sources, such as issue trackers, technical documentation, or source code versioning sys-

tems. We attempt the automatic classification of *commits* added to *software repositories*, by using commit metadata and features mined from the underlying source code. Others have attempted to detect change by, e.g., inspecting the commit message (Fu et al., 2015; Levin and Yehudai, 2016). These features, as well as other developer-level information, are often unreliable or subjective. Therefore, we rely on objective (i.e., not subject to human error) features only. We take the size of the underlying source code, the time between consecutive commits with direct parent-child relations (*sojourn-time*, cf. Fig. 1), and some mostly Boolean features describing further relations (e.g., whether a commit was the first or last on its branch), into account.

Software maintenance was suggested to be categorized into *adaptive*, *corrective*, and *perfective* activities (Swanson, 1976). Adaptive changes comprise the addition of new features. Corrective changes are necessary to fix faults, correct behavior, or remediate vulnerabilities. The most frequent occurring change, however, is of perfective nature. Such changes are made to, e.g., optimize the software, or to prepare for the accommodation of future features. Beyond these three categories, other subdivisions and exten-

^a  <https://orcid.org/0000-0001-7937-1645>

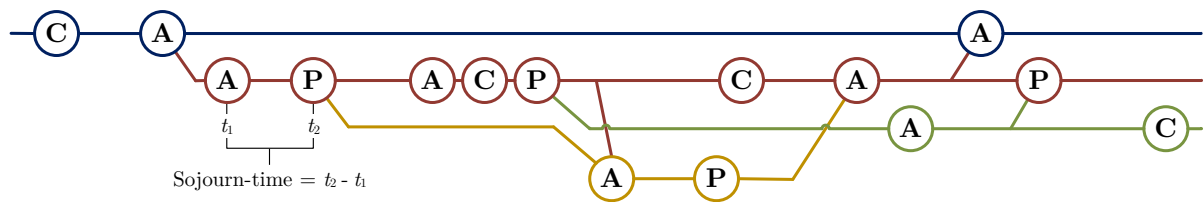


Figure 1: Exemplary Git branching, involving various purpose-bound branches. Sojourn-times are computed between consecutive commits on a branch. Merge commits are ignored in our study.

sions have been proposed (Lin and Gustafson, 1988). However, most of the other literature seems to adapt these three principal categories (Mockus and Votta, 2000; Levin and Yehudai, 2017b).

Furthermore, the focus of this paper is the efficient exploitation of effortlessly accessible and objective features, using models able to facilitate the relationships of commits and sojourn-times between those, for the purpose of commit classification into the three principal maintenance activities. We reason about and introduce various models that can be fitted to this kind of problem. We show their theoretical and practical limitations, their advantages and disadvantages, and provide an overview of suitable models. Finally, we demonstrate a solution to exploit relational datasets using traditional, *stateless* classifiers, such as Random forest (Breiman, 2001).

Notions

Before proceeding, we lay out the notions that we will use from here on and after. Some sections use additional notions, and those are given there.

Symbols: We use a number of symbols, and these are: a (scalar value), \mathbf{a} (vector), \mathbf{A} (two-dimensional matrix), \mathbf{A} (three- or higher-dimensional Tensor), f_i^j or \mathbf{f}_i^j (j -th order (vector of) estimator(s) fixed on/segmented over state i).

Order: Refers to the depth of the relations in a model or dataset. A first-order model is one that computes a probability of some observation O at time t using the preceding observation $t - 1$, e.g., $P(O_t | O_{t-1})$.

2 PROBLEM STATEMENT

Commits are sets of changes that are added as patches to a software repository. These sets can comprise any number of changes, such as added, modified, or deleted lines or files. These differences can be quantified, and models using such features represent the current state of the art (Hönel et al., 2020). Since two consecutive commits C_t, C_{t+1} represent the software repository at the two states $t, t + 1$, we have measured

the difference in size between these two states, i.e., how many lines or files were affected, and by how much. A commit’s maintenance activity is its associated state, that is, a commit labeled *adaptive* is said to be in the state with the same name. The class of a new commit (the state it transitioned into) is latent. However, we can observe its features, as well as the features and classes (states) of its preceding commits.

The main underlying conjecture for this paper is, that taking into account relations, sojourn-times between commits, as well as mostly size-based features significantly affect posterior distributions for some or all of a commit’s features. The goal is to classify a commit C at time t , that is, to assign it to one of the three possible *discrete states* $S_i \in \mathcal{S} = \{a, c, p\}$ (adaptive, corrective, perfective). We hypothesize that developer behavior is not purely random, that is, it is more likely to observe certain successions of activities than others. For example, a relatively small commit that quickly follows a large adaptive commit is more likely to be of corrective or perfective nature. Mathematically, we reason that the conditional posterior probability for a chain of k preceding commits (Eq. 1) is significantly different from $P(C_t = S_i)$.

$$\begin{aligned} &\theta \dots \text{feature vector of a commit,} \\ &P(C_t = S_i | \theta_t, \dots, \theta_{t-k} \wedge C_{t-1} = S_j, \dots, C_{t-k} = S_n). \end{aligned} \tag{1}$$

In other words, we hypothesize that commit classification models can benefit from additionally exploiting preceding commits’ features $\theta_t, \dots, \theta_{t-k}$, as well as their known classes $C_{t-1} = S_j, \dots, C_{t-k} = S_n$. Given the stateful nature of this problem, we argue that models that capture transitions and (conditional) transition probabilities are suitable for this task.

2.1 Scope

Developers tend to *bulk-commit* unrelated changes, resulting in *tangled* commits (Kirinuki et al., 2014). However, our own labeling has shown that most of the commits are still assignable to a single principal maintenance activity. The existence of tangled commits is further somewhat exacerbated by merge

commits, which were expressly avoided in our study. Therefore, we only consider single-parent commits and assign each commit purely to one of the three discrete maintenance activities.

2.2 Formulation and Transposition

Given a labeled dataset of commits, e.g., (Levin and Yehudai, 2017a; Hönel, 2019), we gather observations of measurements for any of the three maintenance activities. This kind of dataset is applicable to stateless classification problems, as we neither have information about the transitions between states, nor how long these took. Previously, it was demonstrated that utilizing observations about the source code changes can be leveraged for automatic commit classification (Levin and Yehudai, 2017b) and that adding additional attributes about the size of a commit and the size of preceding commits can be integrated into such models (Hönel et al., 2020) to achieve state-of-the-art classification rates, both cross-project and for individual projects.

Traditionally, capturing transitions between states, given continuous or discrete state spaces, requires models that exhibit temporal dynamic behavior. Some non-exhaustive examples of these kinds of models are, e.g., Hidden (semi-) Markov Models (HsMM) (Ramage, 2007; Helseke and Helseke, 2019), Maximum Entropy Markov Models (MEMM) (McCallum et al., 2000), Conditional Random Fields (CRF) (Sutton and McCallum, 2012), recurrent neural networks (RNN), or Long short-term models (LSTM). HsMMs, MEMMs, and CRFs can be generalized into *dependent mixture* models. Such models also allow fitting transition models with covariates, meaning that some posterior distribution may depend on the current or previous state, or any other desired property (Visser and Speekenbrink, 2010). Some Bayesian models and HsMMs with finite and discrete state spaces are quite similar in the sense that both use segmented data to build a posterior distribution of some random variable. We discuss applicable models later, Bayesian and Markovian in Section 3.2, and Dependent Mixture Models in Section 3.3.

The data we gathered is likely too scarce to be fed into LSTM models. And while a few implementations exist for the other models, those are often limited in some way; e.g., some only support 1st-order transitions (often), so that we can only capture the immediate child of a commit, but not its grandchild. Another frequent occurring limitation is the restriction to univariate models, or, in case there is support for multivariate models, these cannot be of mixed type or are restricted to a few specific probability distributions.

In other words, to capture the problem as we have defined it, one would need a general-, multiple observation sequence-, *n*th-order multivariate (semi) dependent mixture model. A semi-model allows for arbitrary sojourn-time distributions in each state, instead of just fixed discrete-time transitions (O’Connell and Højsgaard, 2011).

We propose transposing the problem by leveraging existing parent-child relations. Each instance is horizontally concatenated with its predecessor (or successor). This has several advantages. First, we can generate plain datasets of arbitrary order. In each order, we have the same kind of features available. Second, we may fix the state of any instance at some $t \in T$, and examine the distributions of the random variables of any of its related instances in any arbitrary order. Third, this segmentation can be exploited to assess the *variable importance* of some order. We make use of this in our empirical evaluations. Last, transposing the problem in this way makes it compatible with stateless *and* stateful classification models alike. This is especially useful for another class of models, that are based on joint conditional probability distributions. With the suggested transposition, the problem is now eligible for *segmentation*, and we may estimate joint conditional probabilities over any set of random variables, from any order, effectively allowing us to encapsulate transitions between states by capturing the posterior probability distributions of some of the random variables. Using kernel density estimation, we can estimate individual densities, i.e., it is not required to estimate one multidimensional density. We introduce a number of these models more formally in Section 3.4, as we achieve a new state of the art with them.

2.3 Research Questions

With the problem transposed as in Section 2.2, we are now capable of attempting to fit stateful and stateless models similarly. Our research questions are, therefore, driven by the striving for answering the question, of whether either type of model is suitable for the problem.

RQ1: Using common state-of-the-art stateless models, such as Random forest or Gradient Boosting Machines, what is the best achievable classification rate?

Answering this question will not only set a baseline for achievable classification rates but also allow us to compare the effects, especially of joint density models, where some densities are estimated over earlier orders.

RQ2: We have plenty of features available, are univariate HMMs with discrete state-space apt for clas-

sification?

We conjecture that no univariate HMM can adequately represent a commit’s state. This question is therefore mostly approached for the sake of justification of more sophisticated Markov models.

RQ3: Do 1st- and 2nd-order dependent mixture models for multiple observation sequences perform better than stateless classifiers?

We have implemented a classical 1st-order model, that would likely be the most recommended for the problem at hand. Since no 2nd-order implementation was available, we provide our own and compare with a focus on the different orders.

RQ4: Do 1st-, 2nd-, or 3rd-order joint conditional density models perform better than stateless classifiers, given that these models do not require ordered sequences for inferencing?

Since any marginal density that is part of the joint density can be estimated over the variables of any available arbitrary order, it allows some degrees of freedom with regard to which dimensions we segment over a specific state.

3 BACKGROUND

In this section, we introduce Bayesian and Markovian models (Section 3.2), before we show and generalize dependent mixture models in Section 3.3. Models based on joint conditional densities are introduced in Section 3.4.

3.1 Data Used

We have enriched a dataset (Hönel, 2019) by first adding parent-child relations and then by manually labeling series of consecutive (non-merge) commits. Thirdly, we gathered additional attributes about the relation and the sojourn-time for each commit. The resulting dataset (Hönel, 2023) contains more than 50 independent variable-length sequences of consecutive commits and almost 300 commits in total. Commits were taken from the four Java applications elasticsearch, OrientDB, RxJava, and Spring-FW. Sojourn-times appear to resemble a log-normal distribution. The average sojourn time is ≈ 26.2 hours. The 10/25/50/75/95th percentiles are $\approx 0.7, 9, 34, 453, 5810$ minutes.

3.2 Bayesian and Markovian Models

Bayesian networks and Markovian models share some similarities, as both are directed acyclic graphs, that compute the probability or likelihood of some

variable given some conditional probability, which is usually estimated over a *posterior* probability distribution. In Bayesian networks, these conditions are usually derived from some probability tables, so that the distribution of some variable depends on another variable assuming a specific value. Given the classical Rain – Grass wet – Sprinkler example, the probability of the grass being wet is conditional on whether it rained, the sprinkler ran, or both. The process of imposing these conditions, in order to derive a conditional probability density or mass, is called *segmenting*. Furthermore, we facilitate this mechanic to segment one or more variables at the same time. This is used when segmenting over more than one order in the dataset of the transposed problem.

Markov models that make some observations at any discrete point in time, without being able to observe the actual state, are called latent or hidden. Similar to Bayesian networks, however, we can exploit this in 1st-order HMMs by building densities that are conditional on the current transition, i.e., conditional of having come from state $t - 1$, and now being in state t . This concept can be extended to any arbitrary order. Building and exploiting conditional densities this way, however, is more common in dependent mixture models. In this paper, classical 1st-order HMMs were only used to address RQ1.

3.3 Dependent Mixture Models

Dependent mixture models are by themselves a generalization of standard Markov models, latent/hidden Markov models, and latent class and finite mixture distribution models. Like traditional HMMs, dependent mixture models make use of a discrete and finite state space, initial probabilities for all these states, transition probabilities between them, and emission probabilities. The latter, however, are generalized and represented as a vector of density functions for each feature, thus effectively making these models multivariate. In our implementation, we estimate the empirical probability mass for discrete features, and the empirical probability density or cumulative probability, alternatively, for continuous features.

We use the notions and definitions of (Visser and Speekenbrink, 2010) to define and implement 1st- and 2nd-order models. However, we waive using additional symbols for segmenting random variables using some covariates. The data is considered to have the form $\mathbf{O}_{1:T} = (O_1^1, \dots, O_1^m, \dots, O_T^1, \dots, O_T^m)$, for an ordered sequence of m -variate observations. We usually do not denote m to indicate each random variable and instead just use O_t or \mathbf{O}_t as a shorthand notation.

1. S_t is an element of $\mathcal{S} = \{1, \dots, n\}$, the discrete

and finite state space,

2. $\pi_i = P(S_1 = i)$, a vector of initial state probabilities, giving the probability for starting in state i ,
3. $\mathbf{A}_{i,j} = P(S_{t+1} = j | S_t = i)$, a two-dimensional matrix providing transition probabilities for going from $S_t = i$ to $S_{t+1} = j$,
4. \mathbf{b}_{S_t} , a vector of conditional density functions for state S_t , one for each random variable; $b_j(m) = P(O_t^m | S_t = j)$.

The first-order dependent mixture model, which assigns a likelihood of being in state j for observation O_t , is defined in (2). This is a recursive definition and constitutes the *Forward* algorithm, that requires the likelihood of the preceding state. The initial observation is estimated using (3).

$$\phi_t^1(j) = \sum_{i=1}^N [\phi_{t-1}^1(i) \mathbf{A}_{ij} \mathbf{b}_j(O_t)] \times \left(\sum_{i=1}^N \phi_{t-1}^1(i) \right)^{-1}, \quad (2)$$

$$\phi_1^1(j) = \pi_j \mathbf{b}_j(O_1). \quad (3)$$

Equivalently, we define the 2nd-order dependent mixture model as (4) with an estimator for the second observation, that is the same as the generic estimator for 1st-order models, i.e., (5) is equal to (2). Beyond that, the 2nd-order model uses the same $\phi_1^1(j)$ for estimating the likelihood of the first observation. This is true for any n -th order dependent mixture model. Generally, any n -th order model uses the estimator ϕ_t^{n-1} for observation O_{t-1} , $\forall t < \min(n, T-1)$.

$$\phi_t^2(j) = \frac{\sum_{h=1}^N \sum_{i=1}^N [\phi_{t-2}^2(h) \phi_{t-1}^2(i) \mathbf{A}_{hij} \mathbf{b}_j(O_t)]}{\sum_{h=1}^N \sum_{i=1}^N \phi_{t-2}^2(h) \phi_{t-1}^2(i)}, \quad (4)$$

$$\phi_2^2(j) = \phi_1^1(j). \quad (5)$$

We have implemented and evaluated 1st- and 2nd-order dependent mixture models. Additionally, we add the following generalization for arbitrary-order dependent mixture models (6). This model generalizes by having a vector of previous states, \mathbf{i} , and the tensor $\mathbf{A}_{\mathbf{i},j}$, that has the dimensions $\mathbf{card}(\mathbf{i}) + 1$. That tensor can be thought of as folding $n = \mathbf{card}(\mathbf{i})$ tensors, each of which sums to 1. The initial n -th order model, for $t > 1$, is chosen by $\phi_t^n = \Theta_{t,n}$ (7).

$$\phi_t^n(j) = \frac{\sum_{\mathbf{i} \in \mathbf{i}} [\theta_n(\mathbf{i}) \mathbf{A}_{\mathbf{i},j} \mathbf{b}_j(O_t)]}{\sum_{\mathbf{i} \in \mathbf{i}} [\theta_n(\mathbf{i})]}, \quad \forall t > 1, \quad (6)$$

$$\Theta_{t,n}(j) = \begin{cases} \phi_1^1(j), & \forall t < 0, \\ \phi_t^{\min(t-1, n)}(j), & \text{otherwise.} \end{cases} \quad (7)$$

Table 1: Comparison of conditional density functions b_j , c_j , d_j , e_{ij} , and e_{hij} .

Function	Order	S_t	S_{t-1}	S_{t-2}	X_t	X_{t-1}	X_{t-2}
$b_j(O_t)$	-	✓	-	-	✓	-	-
$c_j^1(O_t)$	1	-	✓	-	✓	-	-
$c_j^2(O_t)$	2	-	-	✓	✓	-	-
$d_j^1(O_{t-1})$	1	✓	-	-	-	✓	-
$d_j^2(O_{t-2})$	2	✓	-	-	-	-	✓
$e_{ij}^1(O_t)$	1	✓	✓	-	✓	-	-
$e_{hij}^2(O_t)$	2	✓	✓	✓	✓	-	-

3.4 Joint Conditional Probability Models

In this section, we introduce models based on joint conditional densities. We distinguish two sub-types of models, (i) models that require input from parent observations, and (ii) those that do not. None of these models requires transition probabilities, as those are encapsulated by the joint densities. Models of the first sub-type are therefore stateful, and models of the second sub-type are not, and hence only require the current observation as input. We implement and evaluate one model of the first sub-type, and suggest another one. For the second sub-type, we suggest, implement, and evaluate two models.

An overview of the conditional density functions is found in Table 1. In this table, X is the set of all random variables X associated with each observation. X_t refers to the set of random variables belonging to the t -th predecessor of O_t . Hence, any S_t designates which instances were selected before densities were estimated. As an example, the function $e_{ij}^1(O_t)$ selects those instances, that were of type j ($S_t = j$) and had a parent of type i ($S_{t-1} = i$). It then estimates densities over X_t , which is the set of random variables associated with O_t .

We suggest four models A through D, and provide implementations for the first three models. All of the models use $b_j(O_t)$, as it expresses a likelihood for $O_t = j$ (is O_t a “ j ”?).

Model A uses $d_j^n(O_{t-n})$ as estimator. Additionally, to inquiring about a likelihood for the current observation, models using d_j^n test for the likelihood of the entire transition, as they estimate densities over the predecessors while having had previously segmented over a specific j . Informally speaking, these models ask for whether the parent O_{t-1} as given is one that *usually* precedes the kind j currently presumed to be O_t . Model A in 1st- and 2nd-order is given by Equations (8) and (9), respectively.

$$\phi_t^1(j) = d_j^1(O_{t-1}) b_j(O_t), \quad (8)$$

$$\phi_t^2(j) = d_j^2(O_{t-2}) d_j^1(O_{t-1}) b_j(O_t). \quad (9)$$

Model B uses $c_j^n(O_t)$. This model is peculiar as it neither requires to use an extra estimator for the first states nor does it require previous observations for inferencing. Models using c_j^n had previously segmented on some predecessor of j , while estimating densities over the observations at point $S_t = j$. The transition test in these functions can more informally be understood as asking, how likely O_t is, having come from some j . The underlying conjecture is that variables are differently distributed if they had a specific parent and that densities over these distributions will yield a somewhat different likelihood if this O_t is a typical successor to the current j in question. In other words, is it plausible that this O_t usually follows a j ? The 1st- and 2nd-order models for B are given by Equations (10) and (11), respectively.

$$\phi_t^1(j) = \sum_{i=1}^N [c_i^1(O_t) b_j(O_t)] \times (N \times b_j(O_t))^{-1}, \quad (10)$$

$$\phi_t^2(j) = \sum_{i=1}^N [c_i^2(O_t) c_i^1(O_t) b_j(O_t)] \times (N \times b_j(O_t))^{-1}. \quad (11)$$

Model C uses $e_i^1(O_t)$ (where i is a vector of previous states). Similar to model B, this model does not require extra estimators for the first states, and it infereces using only the current observation. It is somewhat similar to B, but we are asking about $O_t = j$, given a specific set of parents i . This is the transition test for this model. Again, the underlying conjecture is that $O_t = j$ is more likely if we have segmented over predecessors that usually precede that j . The model in 1st- and 2nd-order is specified in (12) and (13).

$$\phi_t^1(j) = \sum_{i=1}^N [e_{ij}(O_t)], \quad (12)$$

$$\phi_t^2(j) = \sum_{h=1}^N \sum_{i=1}^N [e_{hij}(O_t)]. \quad (13)$$

As can be seen in Table 1, we can almost arbitrarily define joint conditional density functions that estimate the likelihood of the current or previous observations, while encapsulating all transition probabilities, thus lifting the need for transition tensors entirely. Additionally, some models do not require initial state probabilities either and become effectively stateless.

Model D is only formally defined but was not evaluated in any way. It is a hybrid of a dependent

mixture- and a joint conditional density model. It is similar to model B, but we are factoring in the likelihood of any potential parent state. Since this is maximized, it may affect the prediction. The 1st order of that model is shown in (14).

$$\phi_t^1(j) = \frac{\sum_{i=1}^N [\phi_{t-1}(i) c_i^1(O_t) b_j(O_t)]}{\sum_{i=1}^N [\phi_{t-1}(i) c_i^1(O_t)]}. \quad (14)$$

3.5 Other Models

Other models that are at least partially suitable to the described problem exist. However, none of these was fulfilling all the criteria or there was simply no implementation available that could be used off the shelf, and an own implementation was not considered feasible. The models we have described until now are all implemented for the R statistical software (R Core Team, 2019) and were made freely available. They are compatible with generic data that fits the same problem as we have tried to solve here so that we are certain that other researchers and practitioners can reuse them.

During early exploration, we examined Conditional random fields (Sutton and McCallum, 2012) closer, as they seemed to be the closest candidate. However, those models are undirected and effectively not using the transition properties that we especially obtained.

4 METHOD

In the previous section, we outlined a number of concrete models suitable to the transposed problem as formulated earlier. Identifying appropriate models was an initial step in approaching this study. Models based on joint conditional density were selected and assembled only after carefully evaluating their theoretical capabilities to encompass the problem.

Another issue was obtaining a suitable dataset, that has a sufficient amount of instances and reliable attributes, and also comes with relational properties. There was no such dataset for commits. We previously released a tool to analyze Git repositories, which we extended to also extract additional relational properties (Hönel, 2022). We then used an existing commit dataset (Levin and Yehudai, 2017a; Hönel, 2019) that had labels across more than 1,150 commits, which were unfortunately not labeled consecutively. It was necessary to reverse-engineer an extensive ruleset for how to label commits. When we were done with the rules, we used them to label a few hundred of the other, already labeled commits.

We achieved an almost perfect agreement. Only after that, we labeled additional adjacent commits. The extended ruleset, dataset, and all software and tools are published (Hönel, 2023). The final dataset has about 50 features per observation, and we use about half of them during our empirical evaluation.

When concatenating datasets horizontally, based on the parent-child relationships in the data (cf. 1), the width and amount of features increase quickly. While for some models this is not problematic, it is for others. We, therefore, apply a Recursive Feature Elimination (RFE) (Guyon et al., 2002) before we train some of the models. As an outer resampling method, at least three times repeated five- or ten-fold cross-validation is used, with a constant holdout of 20% across all tests. Cross-validation behaves as a regularizer when fitting models, and is an effective tool for preventing the over-fitting of models. Over-fitting is probably the largest threat to validity for some models.

5 EMPIRICAL EVALUATION

In this section, we present some results of the evaluation of the dependent mixture- and joint conditional density models. We are checking classification accuracy and Cohen’s Kappa (Cohen, 1960, cf. Eq. 15). Accuracy and Kappa are the two metrics found in most of the related work, and they make for a great team since neither metric has enough explanatory power on its own. If both metrics are reported at the same time, they can mitigate shortcomings of other metrics, such as the F1-score.

$$Kappa = \frac{Accuracy_{total} - Accuracy_{random}}{1 - Accuracy_{random}} \quad (15)$$

As stated in Section 3.4, all of our models are divided into two sub-types, (i) models that require parent observations as input, and those that do not (ii). For an overview, we list models of either sub-type:

- (i) Dependent Mixture models, Joint conditional density model A,
- (ii) Joint conditional density models B,C; 3rd-party models (the models used are: Random forest (Liaw and Wiener, 2002; Wright and Ziegler, 2017; Breiman, 2001), Gradient Boosting Machines (Greenwell et al., 2019), Naïve Bayes (Majka, 2019), and C5.0 (Kuhn and Quinlan, 2020)).

It is worth noting that all training and inferencing were done cross-project, as the data we have spans four Java projects. We have previously shown that fitting a model to an individual project can result in

Table 2: Results of evaluating the stateful models.

Model	Order	Top Acc.	Top Kappa	∅ Acc.	∅ Kappa
DepMix	1	0.75	0.00	0.44	-0.02
DepMix	2	0.66	0.00	0.53	-0.01
JCD: A	1	0.35	-0.01	0.23	-0.05
JCD: A	2	0.40	0.06	0.27	-0.04

higher accuracy, but here, it would result in the data becoming too scarce. For stateless models, we will do a classic cross-validation. For stateful models we require one or more lists of consecutive observations, i.e., we cannot just randomly sample some fraction from the data and then inference. Training on the other hand does not require sequences, as data is selected by segmentation. To alleviate this problem, we do the following:

1. Deterministically shuffle the available sequences of observations in each resample.
2. Take sequences of commits out of the available data until the ratio between commits left in the data and commits present on any chain is 0.8 or less.
3. Each sequence is then predicted separately, and all predictions and ground truth are concatenated. Then, a confusion matrix is created, yielding the metrics’ accuracy and Kappa.
4. Repeat at least 15 times and shuffle the list of available sequences in every pass. Finally, average the results.

5.1 Stateful Models

Almost always is the achieved Kappa for these models zero, or even negative, which means the agreement is worse than it would be by chance. By inspecting some of the results it becomes clear, that these models often predict long sequences without alternating between states much or at all. That explains also the high accuracy. Given that most of the commits are of perfective nature, just predicting that type without changing to another type will deliver exactly the results as we have gotten them. It seems as if the dependent mixture models get stuck in absorbing states as the univariate HMMs did. The results are to be found in Table 2.

5.2 Stateless Models

All 3rd-party stateless models were given data that was previously reduced by RFE to the most significant predictors. The stateless joint conditional density models, however, were evaluated once for all features, and once over the reduced set. Recall the definition

Table 3: Results of evaluating the stateless models.

Model	Order	Top Acc.	Top Kappa	∅ Acc.	∅ Kappa
<i>ZeroR</i>		0.547	0.000	0.515	0.000
C5.0		0.596	0.341	0.592	0.309
GBM		0.619	0.369	0.599	0.321
Naive B.	n/a	0.384	0.171	0.358	0.144
Ranger		0.650	0.421	0.622	0.341
Random F.		0.641	0.409	0.618	0.330
JCD: B	1	0.423	0.199	0.255	0.010
JCD: B	2	0.461	0.266	0.251	0.000
JCD: B	3	0.432	0.237	0.248	0.000
JCD: C	1	0.757	0.612	0.467	0.182
JCD: C	2	0.717	0.519	0.450	0.122

of these models from Section 3.4. It becomes apparent, that the functions build sums and not products. If it were a product, we should have probably introduced *Laplacian/Lidstone smoothing* (Manning et al., 2008), to avoid zero- or near-zero factors. However, since sums are built, additional features, even if they result in such low likelihoods, do not spoil the result. Indeed, the joint conditional density models using all available variables can outperform the best 3rd-party classifier (Ranger) by double digits, with a boost of 10.7% in accuracy and an improvement of 0.19 for Kappa, which improves it from *moderate* to *substantial* (Landis and Koch, 1977). These models thereby also outperform the state-of-the-art models that use the same feature sets. All results are to be found in Table 3.

6 SUMMARY AND CONCLUSIONS

In this paper, we have presented the problem of commit classification, using relational properties. These were not considered previously. Classical attempts that come to mind for solving the presented problem proved not to be of value so appropriate models were suggested.

Our answer to **RQ1** is, that current models, such as Random forest or Gradient Boosting Machines, deliver a similar performance on our data. No model can seem to surpass a certain threshold. All of the 3rd-party models are about 5–10% better than the baseline (ZeroR).

While not evaluated in detail here, the answer to **RQ2** is, that not only can univariate HMMs not be used for automatic classification of changes, but those kinds of models are also prone to *absorbing states*. If the data is skewed, resulting in some states occurring more frequently than others, then this will affect the initial state probabilities, the transition probabilities, and the emission probabilities in a way that these models cannot effectively leave the state they

are stuck in anymore for the predictive scenarios we described here.

RQ3 can only be answered negatively. It seems that dependent mixture models, regardless of their order, are also subject to absorbing states. Since we get consistently almost zero for the Kappa, the accuracy reported by these models is worthless.

However, for **RQ4**, we can report that some of our joint conditional density models can outperform the best 3rd-party models by double digits. We can see an improvement in Kappa by almost an entire class.

It appears that the joint conditional density models capture developer behavior well. Our intuition behind these models was that maintenance activities in typical software projects are not carried out at random but rather follow a logical sequence. Considering a single direct preceding commit, i.e., its features and associated maintenance activity, can be exploited for estimating precise conditional densities. To some degree, this confirms that developer behavior is deterministic.

6.1 Limitations

While we found sufficient corroboration to confirm our hypothesis, the external validity of the trained models cannot be guaranteed. While we used to three times repeated five- or ten-fold cross-validation in order to obtain the results, their robustness is limited.

7 REPRODUCIBILITY

All of our work was done with reproducibility in mind. All tools, data, experiments, implemented methods and reverse-engineered classification rules can be obtained from (Hönel, 2023).

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback that helped us to improve our work.

REFERENCES

- Bell, R. M., Ostrand, T. J., and Weyuker, E. J. (2011). Does measuring code change improve fault prediction? In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*. ACM.
- Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

- Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46.
- Eick, S. G., Graves, T. L., Karr, A. F., Marron, J. S., and Mockus, A. (2001). Does code decay? assessing the evidence from change management data. *IEEE Transactions on Software Engineering*, 27(1):1–12.
- Fluri, B., Würsch, M., Pinzger, M., and Gall, H. C. (2007). Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Transactions Software Engineering*, 33(11):725–743.
- Fu, Y., Yan, M., Zhang, X., Xu, L., Yang, D., and Kymer, J. D. (2015). Automated classification of software change messages by semi-supervised latent dirichlet allocation. *Information and Software Technology*, 57:369–377.
- Greenwell, B., Boehmke, B., Cunningham, J., and Developers, G. (2019). *gbm: Generalized Boosted Regression Models*. R package version 2.1.5.
- Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning*, 46(1/3):389–422.
- Helske, S. and Helske, J. (2019). Mixture hidden markov models for sequence data: The seqHMM package in R. *Journal of Statistical Software*, 88(3):1–32.
- Hindle, A., Germán, D. M., Godfrey, M. W., and Holt, R. C. (2009). Automatic classification of large changes into maintenance categories. In *The 17th IEEE International Conference on Program Comprehension*, pages 30–39. IEEE Computer Society.
- Hönel, S., Ericsson, M., Löwe, W., and Wingkvist, A. (2020). Using source code density to improve the accuracy of automatic commit classification into maintenance activities. *Journal of Systems and Software*, 168:110673.
- Hönel, S. (2019). 359,569 commits with source code density; 1,149 commits of which have software maintenance activity labels (adaptive, corrective, perfective).
- Hönel, S. (2019–2022). Git Density 2022.10: Analyze git repositories to extract the Source Code Density and other Commit Properties.
- Hönel, S. (2023). GitHub Repository: Datasets, Experimental Setups, and Code for Exploiting Relations Between Commits. DOI: 10.5281/zenodo.7715007.
- Kirinuki, H., Higo, Y., Hotta, K., and Kusumoto, S. (2014). Hey! are you committing tangled changes? In *22nd International Conference on Program Comprehension*, pages 262–265. ACM.
- Kuhn, M. and Quinlan, R. (2020). *C50: C5.0 Decision Trees and Rule-Based Models*. R package version 0.1.3.
- Landis, J. R. and Koch, G. G. (1977). An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers. *Biometrics*, 33(2):363–374. PMID:884196.
- Levin, S. and Yehudai, A. (2016). Using temporal and semantic developer-level information to predict maintenance activity profiles. In *2016 IEEE International Conference on Software Maintenance and Evolution*.
- Levin, S. and Yehudai, A. (2017a). 1151 commits with software maintenance activity labels (corrective, perfective, adaptive).
- Levin, S. and Yehudai, A. (2017b). Boosting automatic commit classification into maintenance activities by utilizing source code changes. In Turhan, B., Bowes, D., and Shihab, E., editors, *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*.
- Liaw, A. and Wiener, M. (2002). Classification and regression by randomforest. *R News*, 2(3):18–22.
- Lin, I. and Gustafson, D. A. (1988). Classifying software maintenance. In *Proceedings of the Conference on Software Maintenance*, pages 241–247. IEEE.
- Majka, M. (2019). *naivebayes: High Performance Implementation of the Naive Bayes Algorithm in R*. R package version 0.9.7.
- Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press.
- McCallum, A., Freitag, D., and Pereira, F. C. N. (2000). Maximum entropy markov models for information extraction and segmentation. In Langley, P., editor, *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 591–598.
- Mockus, A. and Votta, L. G. (2000). Identifying reasons for software changes using historic databases. In *2000 International Conference on Software Maintenance*, pages 120–130. IEEE Computer Society.
- O’Connell, J. and Højsgaard, S. (2011). Hidden Semi Markov Models for Multiple Observation Sequences: The mhsmm Package for R. *Journal of Statistical Software*, 39(4):1–22.
- Purushothaman, R. and Perry, D. E. (2005). Toward understanding the rhetoric of small source code changes. *IEEE Transactions on Software Engineering*, 31(6):511–526.
- R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.
- Ramage, D. (2007). Hidden markov models fundamentals. *CS229 Section Notes*.
- Shivaji, S., Jr., E. J. W., Akella, R., and Kim, S. (2013). Reducing features to improve code change-based bug prediction. *IEEE Transactions on Software Engineering*, 39(4):552–569.
- Sutton, C. and McCallum, A. (2012). An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373.
- Swanson, E. B. (1976). The dimensions of maintenance. In Yeh, R. T. and Ramamoorthy, C. V., editors, *Proceedings of the 2nd International Conference on Software Engineering*, pages 492–497.
- Visser, I. and Speekenbrink, M. (2010). depmixs4: an r package for hidden markov models. *Journal of Statistical Software*, 36(7):1–21.
- Wright, M. N. and Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17.