Collaborative Topic Modeling for Recommending GitHub Repositories

Naoki Orii
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
norii@cs.cmu.edu

ABSTRACT

The rise of distributed version control systems has led to a significant increase in the number of open source projects available online. As a consequence, finding relevant projects has become more difficult for programmers. Item recommendation provides a way to solve this problem. In this paper, we utilize a recently proposed algorithm that combines traditional collaborative filtering and probabilistic topic modeling. We study a large dataset from GitHub, a social networking and open source hosting site for programmers, and compare the method against traditional methods. We also provide interpretations on the latent structure for users and repositories.

1. INTRODUCTION

Since the mid-2000's, there has been an increased adoption in distributed version control systems among software developers. Several mature systems such as Bazaar, Mercurial, and Git have appeared, alongside with hosting sites such as Launchpad, Bitbucket, and GitHub. Combined with these hosting sites, the distributed version control paradigm has led to an adoption of collaborative software development, with a significant increase in the number of open source software projects. In particular, GitHub (`https://github.com/`) has attracted the largest user base of these websites with 2.7 million users and hosting over 4.5 million projects[1]. The site hosts a wide array of projects, ranging from the Linux kernel to famous web application frameworks such as Ruby on Rails, and non-software related projects such as the German Federal Law.

In addition to its role as a hosting site, GitHub also functions as a social network for programmers. Projects on GitHub, also known as *repositories*, have profile pages. Users can download, fork[2], and commit to any repository. A key feature of GitHub is that it allows users to *watch* repositories that they find intreresting and want to keep track of[3].

While an increase in the number of availble open source software projects certainly benefits the open source ecosystem, it has become more difficult for programmers to find projects of interest. In 2009, GitHub hosted a recommendation contest to recommend repositories to users. The training dataset was based on 440,000 user-watches-repository relationships given by over 56,000 users to nearly 121,000 repositories. The test dataset consisted of 4,800 users (which all of them are in the training data), and the goal is to recommend up to 10 repositories for each of the test users.

The problem of item recommendation has been studied extensively, especially involving the Netflix Prize. However, there is a distinct difference between the Netflix Prize and GitHub's recommendation contest. While in both contests we have the user-item matrix, we can also consider source code data in the GitHub contest. The source code of a software library can tell us rich information about the content of the repository, and by exploiting this we expect to improve recommendation results. A recent paper formulated this idea, combining traditional collaborative filtering on the user-item matrix and probabilistic topic models on text corpora. In this paper, we apply this method on a large dataset from GitHub.

2. PROBLEM DEFINITION

We assume there are I users and J items. Let $R = \{r_{ij}\}_{I \times J}$ denote the user-item matrix, where each element $r_{ij} \in \{0, 1\}$ represents whether or not user i "favorited" item j. While $r_{ij} = 1$ represents that user i is interested in item j, note that $r_{ij} = 0$ does not necessarily mean that the user is not interested in the item: it can also be the case the user i does not know about item j.

In this paper, we consider two tasks: (i) in-matrix prediction and (ii) out-of-matrix prediction. For in-matrix prediction, the task is to estimate the missing values in R based on the known values. For out-of-matrix prediction, the task is to predict user interest for items that are not included in R. In this paper, this amounts to recommending new repositories that have never been watched by a single user.

3. METHODS

We first give brief explanations of probabilistic matrix factorization and probabilistic topic models. We then de-

[1] As of December 2012.

[2] Forking is a feature on GitHub that allows a users to copy another user's project in order to contribute to it, or to use it as a starting point for his/her own project.

[3] Note that the semantics of watching repositories in GitHub has changed in August 2012 (`https://github.com/blog/1204-notifications`). GitHub has now introduced *stars*, which function similarly as the previous version of watched repositories. In the new watch functionality, once a user watches a repository, he/she will receive notifications for updates in discussions (project issues, pull requests, and comments).

scribe Collaborative Topic Regression, which combines the two methods.

3.1 Probabilistic Matrix Factorization

The basic idea behind latent factor models is that user preference is determined by a small number of unobserved, latent factors. The goal is to uncover these latent user and item features that explain the observed ratings R. User i is represented by a latent vector $u_i \in \mathbb{R}^K$, and item j is represented by a latent vector $v_j \in \mathbb{R}^K$. K is typically chosen such that $K \ll I, J$. The predicted rating $\hat{r}_{ij}$ is given by the inner product of the two latent vectors:

$$\hat{r}_{ij} = u_i^T v_j \tag{1}$$

Thus, given R, the problem is to compute the latent feature vectors u and v. We commonly do this by minimizing the following regularized square error:

$$\min_{U,V} \sum_{i,j} \left(r_{ij} - u_i^T v_j\right)^2 + \lambda_u \|u_i\|^2 + \lambda_v \|v_j\|^2 \tag{2}$$

λ_u and λ_v are regularization parameters.

It is possible to adopt a probabilistic approach for matrix factorization [9]. We can imagine a simple generative model using a probabilistic linear model with Gaussian observation noise as follows:

1. For each user i, draw user latent vector $u_i \sim \mathcal{N}(0, \lambda_u^{-1} I_K)$
2. For each item j, draw item latent vector $v_j \sim \mathcal{N}(0, \lambda_v^{-1} I_K)$
3. For each user-item pair (i, j), draw the response

$$r_{ij} \sim \mathcal{N}\left(u_i^T v_j, c_{ij}^{-1}\right) \tag{3}$$

where I_K is a K-dimensional identity matrix, and c_{ij} measures our confidence in observing r_{ij}. As discussed earlier, we are confident that user i is interested in item j when $r_{ij} = 1$, but we are not as confident that i is not interested in j when $r_{ij} = 0$. Accordingly, we use different values for c_{ij} depending on the value of r_{ij}, as follows:

$$c_{ij} = \begin{cases} a & \text{if } r_{ij} = 1 \\ b & \text{if } r_{ij} = 0 \end{cases} \tag{4}$$

where $a > b > 0$.

3.2 Probabilistic Topic Models

Topic modeling algorithms can be used to automatically extract *topics* from a corpus of text documents. Each topic represents a distribution of terms, and gives high probability to a group of tightly co-occuring words. A document can be represented by a small set of topics.

Source code in software projects can also be thought of text documents. Typically, programmers give meaningful names to variables, types, and functions. Unless they purposely try to obfuscate, minimize, or compress the code, programmers adhere to naming conventions that improve its readability. Thus, groups of tightly co-occuring words appear in source code, much similar to the way in that of text documents written in natural language. In the context for software repositories, we expect to see topics such as "database connection", "encoding", and "networking." For example, the topic "database" may contain the terms {`table`, `column`, `select`, `connection`}, and the topic "user interface" may contain the terms {`click`, `top`, `width`, `button`, `hidden`}. In this paper, we use these discovered topics to improve recommendation, as described in the following section.

The simplest form of a probabilistic topic model is Latent Dirichlet Allocation (LDA) [3]. The generative process for LDA is formulated as follows:

1. Draw topic proportions $\theta_j \sim \text{Dirichlet}(\alpha)$ for document $\boldsymbol{w_j}$
2. For each term n in $\boldsymbol{w_j}$,
   (a) Draw topic assignment $z_{jn} \sim \text{Mult}(\theta_j)$
   (b) Draw word $w_{jn} \sim \text{Mult}(\beta_{z_{jn}})$

3.3 Collaborative Topic Regression

The collaborative topic regression (CTR) model combines traditional collaborative filtering with topic modeling [10]. Note that the term "item" used in collaborative filtering and the term "document" used in LDA both refer to the same thing. Unless otherwise noted, from now on we will use these two terms interchangeably.

Similarly to LDA, in CTR each item j is assigned a topic proportion θ_j that is used to generate the words. A naïve approach is to directly use θ_j to represent the item latent vector in equation 3:

$$r_{ij} \sim \mathcal{N}\left(u_i^T \theta_j, c_{ij}^{-1}\right) \tag{5}$$

Instead of taking this approach, CTR exploits the user data to get an "adjusted" item latent vector v_j. The generative process for CTR is formulated as follows:

1. For each user i, draw user latent vector $u_i \sim \mathcal{N}(0, \lambda_u^{-1} I_K)$
2. For each item j,
   (a) Draw topic proportions $\theta_j \sim \text{Dirichlet}(\alpha)$
   (b) Draw item latent offset $\epsilon_j \sim \mathcal{N}(0, \lambda_v^{-1} I_K)$ and the item latent vector as $v_j = \epsilon_j + \theta_j$
   (c) For each word w_{jn},
      i. Draw topic assignment $z_{jn} \sim \text{Mult}(\theta_j)$
      ii. Draw word $w_{jn} \sim \text{Mult}(\beta_{z_{jn}})$
3. For each user-item pair (i, j), draw the rating

$$r_{ij} \sim \mathcal{N}\left(u_i^T v_j, c_{ij}^{-1}\right) \tag{6}$$

Note that in Step 2(b), ϵ_j is added to item j's topic proportion θ_j in order to obtain the adjusted item latent vector v_j.

The graphical representation of the model is given in Figure 1. The top part of the model represents the repository content, and is essentially an LDA model. The bottom half of the model deals with user-repository data.

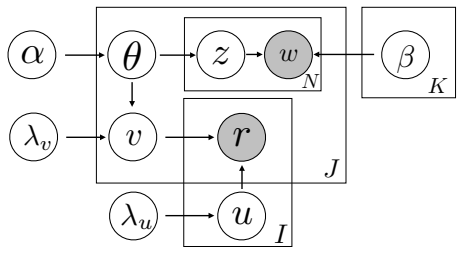

Figure 1: The graphical model for the CTR model

4. EXPERIMENTAL STUDY

4.1 Dataset

The original GitHub recommendation contest dataset contains 440,237 user-watches-repository relationships among 56,159 users and 120,867 repositories. The metadata for repositories consists of repository ID, repository name, date of creation, and (if applicable) the repository ID that it was forked off of. Example lines of the repository metadata file is given in Figure 2.

```
1382:mojombo/grit,2007-10-29
...
1449:schacon/grit,2008-04-18,1382
1450:tokuhirom/http-mobileattribute,2009-03-24
1451:pjhyett/github-services,2008-04-28
```

Figure 2: Example lines of the repository metadata file. Note that repository 1449 is forked from repository 1382.

As the original dataset contains only the metadata for each repository and not the actual source code files themselves, we crawled git repositories from GitHub using the repository names[4]. Once we crawl the repositories, using git, we revert the repositories to the condition they were in July 2009, when the contest was held.

Prior to applying the topic model on the repositories, we first separate the repositories by their programming language. As each repository can contain files with various languages, we classify a given repository into a particular language category if that language's proportion (in lines of code with respect to the entire repository) exceeds 50%. We ignore non source code files (e.g. `README.txt`, `LICENSE.txt`) and vendored files (e.g. commonly bundled files, such as jQuery for web application frameworks). We preprocess the source files by using an appropriate lexer to extract tokens. The types of allowed tokens are given in Table 1. Note that

Type	Example in Java
Literal.String.Single	`'foobar'`
Literal.String.Double	`"foobar"`
Name.Class	`FileSystem`
Name.Constant	`RESOURCE_PROCESSOR`
Name.Decorator	`@Cacheable`
Name.Exception	`RuntimeException`
Name.Function	`main`
Name.Namespace	`java.util`
Name.Variable	`this.name`

Table 1: Allowed token types

comments are removed. In addition to limiting the types of tokens, we also split tokens with CamelCase (`fooBar`) and underscores (`foo_bar`). After the above preprocessing, each repository is seen as an individual document represented with its bag-of-words. Finally, words that occur in more than 80% of the documents and in less than 2% of the documents are removed.

Statistics about the repositories by programming language are presented in Table 2. As GitHub is a Ruby-centric community, Ruby is the dominant language in the dataset. It is also interesting to note that while many scripting languages (Ruby, Python, and Perl) have a smaller vocabulary size compared to "heavyweight" languages (C, C++, and Java), PHP has the largest vocabulary size with over 10,000 terms.

[4]The crawling was done between October 25st and November 1st of 2012.

4.2 Evaluation

As we do not know whether $r_{ij} = 0$ represent that user i is interested in item j or not, we do not use precision but instead use recall to evaluate all of our experiments. Given M recommendations, for each user, we calculate recall@M as follows:

$$\text{recall@}M = \frac{\text{number of repos the user watches in the top } M}{\text{total number of repos the user watches}}$$

This measure is averaged over all users to obtain a global metric.

We perform model evaluation using two different tasks: in-matrix recommendation and out-of-matrix recommendation:

In-matrix recommendation

This is the case where we recommend items that have been watched by at least one user.

We divide up the dataset into a training set and test set, making sure that all items in the test set have appeared at least once in the training set. We perform a 5-fold cross-validation. For repositories that have been watched by 3 or more users, their user-repository pairs will be evenly split into 5 folds. Repositories that have been watches by less than 3 users are always put into the training data.

User preferences are known to drift over time [5], and thus, ideally, we should consider the temporal aspects, ensuring that user-repository pairs in the test data occur later than those in the training data. However, although the original dataset contains the date of creation for repositories, it does not contain the date where users watch repositories. Therefore in this paper, we split the user-repository pairs into folds irrespectively of their (unknown) date. We similarly ignore temporal aspects for out-of-matrix recommendation.

Out-of-matrix recommendation

This is the case where we recommend new repositories that have not been watched previously. As we do not have user data for these repositories, we recommend these based solely on their source code content.

Similarly to in-matrix recommendation, we perform a 5-fold cross-validation. We first evenly split the repositories into 5 folds, and train a model using user-repository pairs that correspond to repositories in 4 of the 5 folds. We test on user-repository pairs that correspond to repositories in the remaining fold.

4.3 Experimental Setting

For matrix factorization (denoted MF), we used the following paramter settings: $K = 200$, $\lambda_u = \lambda_v = 0.01$, $a = 1$, and $b = 0.01$. For collaborative topic regression (denoted CTR), we used $K = 200$, $\lambda_u = 0.01$, $\lambda_v = 10$, $a = 1$, and $b = 0.01$. These values were chosen using grid search on held-out data. In addition to MF and CTR, we also compare against a model that only uses text content, which is based on equation 5 (denoted LDA). This is equivalent to a CTR model where we fix $v_j = \theta_j$.

	Ruby	Python	Perl	PHP	C	C++	Java
# of user-watches-repo	160,205	15,192	6,548	6,667	8,271	3,414	4,677
# of users	19,263	5,780	1,806	3,664	5,038	2,651	3,054
# of repos	14,298	3,943	2,483	2,036	2,248	1,388	1,820
# of vocabulary terms	2,057	4,223	2,866	10,243	7,706	6,131	4,826
# of total words	23,189,461	17,500,898	11,998,887	51,828,603	59,022,110	17,329,607	25,559,017

Table 2: Repository statistics by language.

Figure 3 shows the overall performance for in-matrix and out-of-matrix prediction, when we vary the number of recommended articles from $M = 10, 30, ...150$. For in-matrix-recommendation, CTR slightly underperforms MF. Note that MF is not able to recommend any repositories in the out-of-matrix setting, as the task is to predict repositories that have never been watched before. CTR and LDA perform similarly for out-of-matrix recommendation.

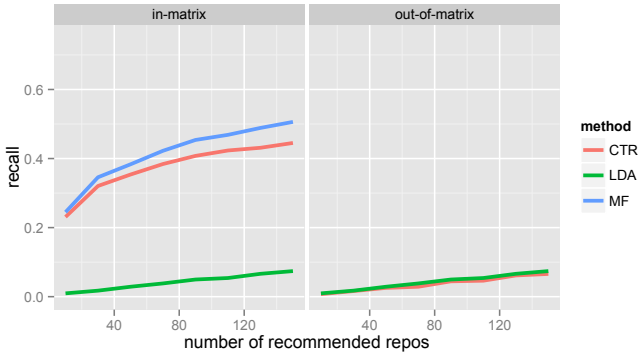

Figure 3: Recall comparison for in-matrix and out-of-matrix prediction tasks for the Ruby language. Error bars are too small to show. CTR shows a slight drop in performance compared against MF. Similar patterns were observed for other languages.

5. EXPLORATORY STUDY

In this section, we conduct an exploratory study, looking at topics discovered from data, which repositories are associated with a given topic, which repositories and topics a given user is interested in, and which topics are associated with a given repository. Most of these analyses are not possible with classical matrix factorization.

Table 3 shows some example topics and their corresponding words learned from the Ruby data and Java data. We are able to uncover topics such as "database access", "user interface", and "encoding." Hadoop, Android, and Clojure are so influential that they form their own topics.

We next examine the item latent space by looking at topic distributions of specific repositories. In particular, we look at Ruby on Rails (RoR)[5], one of the most famous web application framework, and the most watched Ruby repository in our dataset. Figure 4 shows the topic proportion θ_j and adjusted item latent vector v_j for RoR. Using source code content, we are able to identify topics such as "active record"[6] {`active`, `record`, `models`}, "view" {`show`, `render`, `partial`}, and "testing" {`test`, `default`, `mock`}. With user data, we are able to discover further topics such as "Merb"[7] {`merb`, `mongrel`, `rack`} and "jRuby" {`jruby`, `org`, `java`}. Merb is another web application framework that was once considered as a rival to RoR. However, on December 2008, it was announced that Merb would get merged into RoR[8], and the actual merge took place on the relase of Rails 3 on August 2010. Considering that the original GitHub dataset was released on August 2009, it is not surprising that the topic "Merb" does not come up using source code alone (as RoR and Merb had yet to merge), but only comes up using user-watches-repository data.

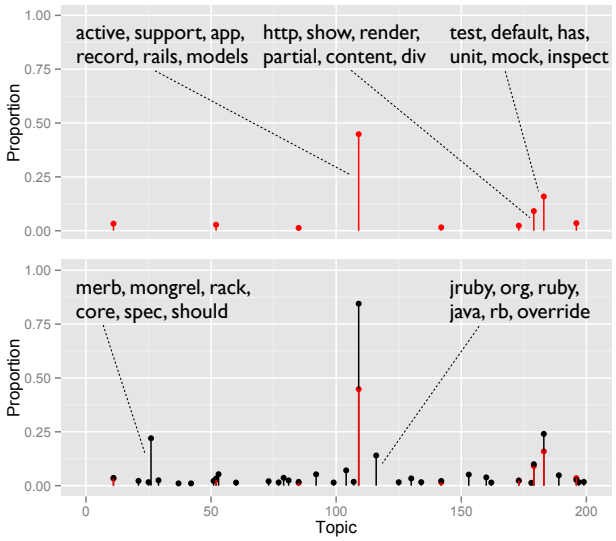

Figure 4: Topic proportion θ_j (top and bottom, in red) and item latent vector v_j (bottom, in black) for Ruby on Rails. Most probable words for the prominent topics are shown for interpretability.

Next, we examine user profiles. Table 4 shows two example users and their top 3 topics along with their top 10 repositories recommended by the CTR model. We see that user I is interested in the topics "git" and "search." User II is interested in the topics "RoR", "images", and "testing."

We can also inspect repositories that have a large item latent offset ϵ_j, or the deviation between the item latent vector v_j and topic proportions θ_j. Table 5 shows the results, with the deviation measured by the (squared) norm

[5] `https://github.com/rails/rails`

[6] Active Record is a design pattern used for accessing data in a database. By default, RoR employs this patten for its database access. Note that `ActiveRecord` is also the name of the implementation of the Active Record pattern in RoR.

[7] `https://github.com/wycats/merb`

[8] `http://weblog.rubyonrails.org/2008/12/23/merb-gets-merged-into-rails-3/`

"datetime"	"database"	"web"	"UI"	"packaging"	"encoding"	"process"	"filesystem"	"mail"
time	table	com	ui	ruby	utf	start	file	to
date	column	http	li	config	iso	pid	dir	content
day	name	www	click	install	name	run	path	net
month	db	google	top	path	rb	stop	directory	from
year	database	url	left	exec	jp	process	files	subject
en	sql	example	selected	for	length	running	filename	text
us	create	with	show	dir	encoding	server	name	mail
am	from	domain	width	ext	char	log	tmp	header
in	adapter	uri	button	no	cp	master	error	utf
locale	select	org	position	the	ascii	daemon	exist	multipart

"HTTP"	"database"	"UI"	"testing"	"hadoop"	"android"	"clojure"	"git"	"audio"
http	sql	image	test	hadoop	android	lang	revision	frame
org	id	awt	junit	org	content	clojure	git	audio
apache	name	color	not	apache	view	invoke	commit	text
request	table	javax	framework	io	override	seq	project	album
io	select	event	case	file	text	meta	idea	sync
response	foo	mouse	should	test	widget	fn	version	artist
connection	from	data	is	output	database	symbol	branch	track
protocol	null	io	up	status	app	lazy	file	media
not	column	point	run	write	cursor	bindings	operation	information
client	insert	file	value	input	create	nil	unsupported	tag

Table 3: Example topics discovered from the Ruby data and Java data.

	User I	watches repo?
Top 3 topics	1. git, tree, refs, pack, io, object, repository, override, lib, file, commit, ref, id 2. search, field, query, index, lucene, override, time, term, value, data, result, string 3. apache, org, io, cos, pd, stream, row, ts, name, dictionary, array, scanner, base, list	
Top 10 repos	1. `Fudge/gitidea` 2. `we4tech/semantic-repository` 3. `imyousuf/jgit-usage` 4. `we4tech/folder-content-guard` 5. `imyousuf/smart-dao` 6. `tjake/thrudb` 7. `j16sdiz/egit-freenet` 8. `sonatype/JGit` 9. `we4tech/bangla-dictionary-based-on-lucene-proximity-search` 10. `we4tech/java-open-search-servlet`	✓ ✓ ✓ ✓ ✓ ✓ ✓
	User II	**watches repo?**
Top 3 topics	1. rails, application, data, helper, config, gemfile, bundler, data, create, gems 2. image, png, jpg, attachment, public, gif, size, file, jpeg, thumbnail, upload, content 3. returns, the, spec, helper, to, when, if, raises, passed, self, error, fixtures, is, method	
Top 10 repos	1. `rails/exception_notification` 2. `technoweenie/attachment_fu` 3. `activescaffold/active_scaffold` 4. `technoweenie/restful-authentication` 5. `dchelimsky/rspec-rails` 6. `rails/rails` 7. `mislav/will_paginate` 8. `dchelimsky/rspec` 9. `drnic/ruby-on-rails-tmbundle` 10. `thoughtbot/paperclip`	 ✓ ✓ ✓ ✓ ✓ ✓

Table 4: Two example users, one from the Java dataset and another from the Ruby dataset. We show their position in latent space via their highest weighted topics in u_i. We also list the top 10 repositories as predicted by CTR. The last column shows whether each repository is actually watched by the user.

of ϵ_j, $\epsilon_j^T \epsilon_j = (v_j - \theta_j)^T (v_j - \theta_j)$. A repository with a v_j that is drastically different from θ_j suggests that the repository is being watched by a wide range of users, as the table indicates.

Finally, extending the idea of looking at the difference between θ_j and v_j, we can find:

- widely watched repositories in a topic
  (in-topic, watched in topic)
- repositories in a topic that are widely read in other topics
  (in-topic, watched in other topics)
- repositories from other topics that are widely read in a topic
  (out-of-topic, watched in topic)

Let θ_j^i represent the topic proportion for item j with respect to topic i, and similarly, v_j^i represent the ith element of the jth item's latent vector, corresponding to topic i. If we want to find a repository from other topics that are widely read in a topic i, we can look for item j that has a low θ_j^i value and high v_j^i value. A schematic of this is shown in Figure 5. Table 6 shows the resulting repositories for the topics "Active Record" and "Hadoop (Distributed File System)." RoR, probability due to its fame, is watched by users both who are primarily interested in "Active Record" and those who aren't, while `docrails` is watched only by those who are primarily interested in "Active Record." This makes sense, considering that `docrails` is a specific branch of RoR where users can make documentation fixes (which is used for fixing typos and factual errors, adding examples, and complementing existing documentation). Thus, we expect users who are deeply involved with RoR development to be interested in `docrails`, whereas casual users might be interested in RoR itself but not `docrails`. We also see that users interested in "Active Record" is also interested in `will_paginate`, `rspec`, `merb`, and `sinatra`, all of which are famous libraries in Ruby's ecosystem.

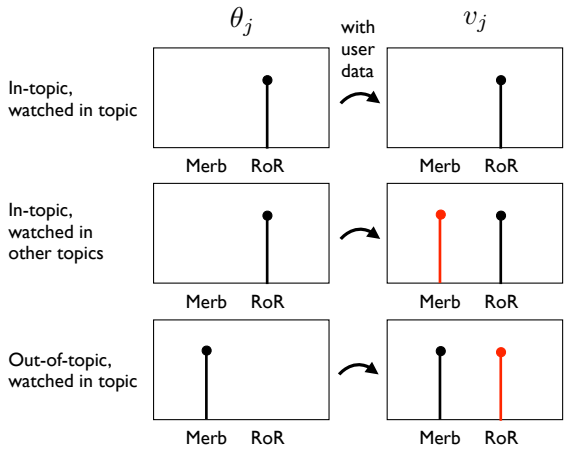

Figure 5: Schematic of the 3 different cases for the relationship between θ_j and v_j.

6. RELATED WORK

Much of this paper is influenced by a recent paper that proposed collaborative topic regression [10]. This paper is an application of collaborative topic regression on an alternative dataset. Another relevant work is fLDA [1], which is a generalization of the supervised topic model for collaborative filtering.

Applying probabilistic topic models to source code is itself not a new idea. Researchers have recently applied topic models to various aspects of software development, including both source code [2; 6] and documentation [4]. There has been work on extracting the relationship between developers (authors) and source code topics [7] using the Author-Topic model [8].

Topic	`active, record, rails, models`
In-topic,watched in topic	`lifo/docrails` `rails/rails`
In-topic,watched in other topic	`rails/rails`

Out-of-topic,watched in topic	`mislav/will_paginate`
	`dchelimsky/rspec`
	`wycats/merb`
	`bmizerany/sinatra`
	`chneukirchen/rack`

Topic	`hadoop, io, fs, output`

In-topic,watched in topic	`apache/hadoop`
	`apache/mahout`
	`apache/pig`
	`apache/hive`

In-topic,watched in other topic	`apache/nutch` `apache/hadoop`
Out-of-topic,watched in topic	`myabc/nbgit` `talios/clojure-maven-plugin`

Table 6: Topic Exploration for "Active Record" and "Hadoop (Distributed File System)."

7. CONCLUSION AND FUTURE WORK

In this paper, we applied collaborative topic regression, a method that combines collaborative filtering and topic modeling, on a large dataset from GitHub. While this method does not outperform existing methods, it produces highly interpretable latent structures for users and items.

One possible future area of work is to to exploit the structure of repositories. Complex software libraries almost always divide their functionality in a logical structure to make it easier for developers to navigate easily and better comprehend the library. For example, modern web application frameworks such as RoR employ the Model-View-Controller design pattern, where the *model* takes care of the data representation, and the *controller* mediates between the model and the *view*, which is responsible for generating output that is visible to the user. Many web application frameworks, including RoR, have directory structures that reflect this architecture.

8. REFERENCES

[1] Deepak Agarwal and Bee-Chung Chen. flda: matrix factorization through latent dirichlet allocation. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 91–100, New York, NY, USA, 2010. ACM.

[2] Pierre F. Baldi, Cristina V. Lopes, Erik J. Linstead, and Sushil K. Bajracharya. A theory of aspects as latent

Repository	# dataset	# stars	# fork
joshuaclayton/blueprint-css	4,200	5,038	480
rails/rails	10,793	16,739	4,391
technoweenie/restful-authentication	2,301	1,659	260
binarylogic/authlogic	1,680	3,397	457
insoshi/insoshi	1,311	1,480	436
thoughtbot/factory_girl	2,517	2,509	370
mislav/will_paginate	2,051	3,410	512
chriseppstein/compass	1,099	4,022	566
thoughtbot/paperclip	3,882	4,656	988
mojombo/jekyll	1,088	8,368	1,338

Table 5: Top 10 Ruby repositories with the largest deviation between the item latent vector v_j and topic proportions θ_j, measured by $(v_j - \theta_j)^T (v_j - \theta_j)$. Column 2 shows the number of users watching the repository in the original dataset. Columns 3 and 4 show the number of stars and number of forks (retrieved from GitHub on Dec 10, 2012). All of these repositories are extremely popular.

topics. In *Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, OOPSLA '08, pages 543–562, New York, NY, USA, 2008. ACM.

[3] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[4] Abram Hindle, Michael W. Godfrey, and Richard C. Holt. What's hot and what's not: Windowed developer topic analysis. In *ICSM*, pages 339–348. IEEE, 2009.

[5] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 447–456, New York, NY, USA, 2009. ACM.

[6] Erik Linstead, Paul Rigor, Sushil Bajracharya, Cristina Lopes, and Pierre Baldi. Mining concepts from code with probabilistic topic models. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ASE '07, pages 461–464, New York, NY, USA, 2007. ACM.

[7] Erik Linstead, Paul Rigor, Sushil Bajracharya, Cristina Lopes, and Pierre Baldi. Mining eclipse developer contributions via author-topic models. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pages 30–, Washington, DC, USA, 2007. IEEE Computer Society.

[8] Michal Rosen-Zvi, Thomas Griffiths, Mark Steyvers, and Padhraic Smyth. The author-topic model for authors and documents. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, UAI '04, pages 487–494, Arlington, Virginia, United States, 2004. AUAI Press.

[9] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, volume 20, 2008.

[10] Chong Wang and David M. Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 448–456, New York, NY, USA, 2011. ACM.