LATENT VARIABLE MODELS FOR PREDICTING FILE DEPENDENCIES IN LARGE-SCALE SOFTWARE DEVELOPMENT

{Diane Hu, Youngmin Cho, Lawrence Saul, Sorin Lerner} @ University of California, San Diego {Laurens van der Maaten} @ Delft University of Tech.

Problem Overview

When modifying code in large software systems, software developers must know: *What other dependent files need to be modified as a result of this change?*

Dependencies can be difficult to detect. This is especially true when the code base is large or contains files of different formats (e.g. code in different languages, meta data, web documents, etc.). Given any set of *starter files* that the developer wishes to modify, we want to automatically recommend possible dependent files that the developer must update to ensure correctness of code.

Mining Development Histories

Mining development histories involves:

(1) Obtain log from control versioning system (e.g. SVN) for code base
(2) Each check-in (or *transaction*) denotes sets of jointly modified files
(3) Use transactions to find patterns of jointly modified files and reveal dependencies
(4) Recommend dependent files for starter files ***s***

The development history is represented by an N (# of transactions) by D (# of files) binary matrix. Non-zero elements in a row indicate files that were checked-in together and thus, jointly modified at some point in time.

Related Methods

Impact Analysis[4]

- Analyzes semantic content of code using call graphs, dynamic/static slicing, etc.
- Fails in identifying cross-language or cross-program dependencies.

Frequent Itemset Mining (FIM)[5,6]

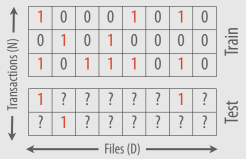
- Mines development history
- Efficiently stores the number of times each possible file set has been jointly modified
- Recommends files that have been jointly modified with *s* more than *t* times

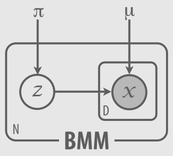
Each starter set is an *incomplete transaction*; finding dependent files is done by solving a problem in *binary matrix completion.* We explore four latent variable models (LVMs) for this task.

Latent Variable Models

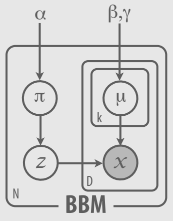
The problem set-up is pictured on the right. All models treat the binary elements of each transaction as observed variables ***x*** for training. Given a set of starter files ***s***, each model recommends files whose posterior probability $Pr(\mathbf{x}=1|\mathbf{s}=1)$ exceeds some threshold.

The binary matrix completion problem: Fill in ?s at test time





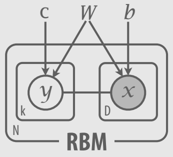
Bernoulli Mixture Model (BMM)
EM is used to learn parameters π, μ of hidden clusters.



Bayesian Bernoulli Mixture Model (BBM)
BBM is a Bayesian treatment for BMM, where

$$Pr(\pi|\alpha) \sim Dirichlet(\alpha/k, \ldots, \alpha/k)$$
$$Pr(\mu_j|\beta,\gamma) \sim Beta(\beta,\gamma)$$

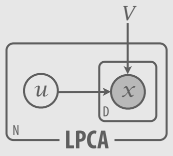
Collapsed Gibbs sampling is used to estimate the expected values of $Pr(\mathbf{x}|\mathbf{s})$ under the posterior.



Restricted Boltzmann Machines (RBM)
RBMs model the joint distribution over data $\mathbf{x}$ and binary hidden units $\mathbf{y}$:

$$Pr(\mathbf{x},\mathbf{y}) = \frac{1}{Z}\exp(-\mathbf{x}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{x} - \mathbf{c}^T\mathbf{y})$$

Here, $\mathbf{W}$ stores weight matrix between layers. The RBMs are trained using contrastive divergence.



Logistic PCA (LPCA)
LPCA finds the low-rank log-odds matrix Θ that maximizes the log-likelihood of the observed data, $\mathbf{X}$:

$$\mathcal{L}_{\mathbf{X}} = \sum_{nd}[X_{nd}\log\sigma(\Theta_{nd}) + (1-X_{nd})\log\sigma(-\Theta_{nd})]$$

The low-rank factorization $\Theta = UV$ is used to complete the missing binary data.

Datasets & Results

Datasets were constructed from check-in records of three large, open source systems. Test sets were constructed by randomly choosing starter files from each transaction; remaining files became the "ground truth" that our models try to predict. We report our results with varying values of *support*[3] and *start* (# of starter files) below. The *f-measure*[1] (in gray) and *correct prediction ratio*[2] (in blue) are used as evaluation metrics.

	Mozilla Firefox			Eclipse Subversive			Gimp		
Dates	Mar 2007 - Nov 2007			Dec 2006 - May 2010			Nov 2007 - May 2010		
Support	Train	Test	# Files	Train	Test	# Files	Train	Test	# Files
15	9,015	2,266	778	316	92	38	5,084	3,436	899
25	8,021	1,771	411	233	59	25	4,469	3,012	447

		Mozilla Firefox				Eclipse Subversive				Gimp			
Model	**Support**	**Start = 1**		**Start = 3**		**Start = 1**		**Start = 3**		**Start = 1**		**Start = 3**	
FIM	15	0.129	0.144	0.127	0.194	0.141	0.461	0.319	0.632	0.014	0.091	0.016	0.159
	25	0.124	0.135	0.110	0.195	0.227	0.616	0.360	0.637	0.006	0.057	0.010	0.095
BMM	15	0.160	0.202	0.110	0.141	0.181	0.486	0.350	0.489	**0.134**	0.205	0.085	0.143
	25	0.177	0.218	0.130	0.160	0.251	0.566	0.382	0.482	0.117	**0.212**	0.010	0.131
BBM	15	0.192	0.340	**0.180**	0.376	0.202	0.607	0.374	**0.769**	0.114	0.200	0.107	0.183
	25	**0.197**	**0.360**	0.175	**0.391**	**0.262**	**0.694**	**0.418**	0.756	0.110	0.206	0.103	0.179
RBM	15	0.156	0.246	0.063	0.310	0.157	0.238	0.138	0.423	0.080	0.148	0.024	0.205
	25	0.172	0.269	0.088	0.340	0.200	0.426	0.259	0.524	0.062	0.143	0.025	0.230
LPCA	15	0.182	0.254	0.157	0.295	0.138	0.452	0.281	0.615	0.124	0.200	**0.145**	**0.288**
	25	0.174	0.277	0.162	0.325	0.247	0.605	0.344	0.625	0.100	0.205	0.131	0.230

LVMs outperform the popular FIM approach on all datasets. While FIM simply looks at co-occurence data, LVMs can exploit higher-order information by discovering underlying structure in data, leading to better predictions. LVMs also have an advantage over traditional impact analysis in its ability to find dependent files that span different languages and file formats.

Footnotes & References

[1] *f-measure:* harmonic mean of precision and recall
[2] *correct prediction ratio:* fraction of correct predictions, assuming # of files to predict are given (alleviates thresholding problem)
[3] *support*: number of transactions each file must appear in in order to be considered (a method of pruning to reduce noisy file data)

[4] Robert Arnold, et. al. Software Change Impact Analysis. IEEE Computer Society, 1996
[5] Annie Ying, et. al. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering,* 2004
[6] Thomas Zimmerman, et. al. Mining version histories to guide software changes. *International Conf. on Software Engineering,* 2004