CSE 515: Literature Survey - Google Code Jam Project

Jijiang Yan, Daniel Graf

May 17, 2013

1 Detecting outsourced Student Programming Assignments

by Bruce S. Elenbogen and Naeem Seliya in Journal of Computing Sciences in Colleges 23.3 (2008): 50-57. (Link to the PDF)

Background In this paper data mining techniques are applied to detect if a student's programming assignment is likely to have been coded by some third-party. According to the paper such 'ghost-coder' companies have experienced a large growth over the past years, due to the rise of the internet and a wide difference in wages around the world.

In detecting such outsourced assignments they faced the following challenges:

- Traditional plagiarism detection schemes can not be applied, as the outsourced solutions are not just copies of other solutions. Much like ghost-writers, someone gets paid to solve and implement a students homework on their own, ensuring that it is different from all other submissions.
- The course was a lower-level CS course, meaning that most students were improving their programming skills throughout the term. So it is likely that their programming style evolves and hopefully improves over time, which makes it harder to identify a student based on his programming style. The authors suppose that such changes in style would be highly correlated to other students. For example, if a student starts writing more comments this might be caused by the feedback he got for previous assignments or by seeing code samples in class.
- They only had a relatively small number of programs per student (seven, resp. six), as it is only based on submissions for a single course.
- The instructor of the course is a common influence to all students. He might have given hints or pointers on how the student code should look like. While this makes the distinction within the class harder, this might help in differentiating outsourced code. An outsourced program would be missing the elements outlined in class.

Method and Results They extracted high-level features (number of lines of code, number of comments, average length of variable names, number of variables, relative number of for loops and the number of bits in the zipped program) to capture the students programming style. Note that none of these features capture the functionality of the program. Based on these features they used the C4.5 algorithm to build a decision tree to distinguish the students.

At this point we want to highlight the two main limitations of the paper, from our point of view. First, they only used a very small code base of 12 students and a total of 83 programs. Also, they considered only a few seemingly arbitrary features. They found that the best students were easier to identify than bad students. Given that the best students are unlikely to outsource their assignments, it remains unclear how effective the approach would be to detect outsourced code in practice.

Despite these limitations they have some interesting results. Their decision tree was able to successfully assign 75% of the programs to the author. They found that the 3 most important features were number of

lines of codes, number of comments and variable length and that the relative number of for loops was less significant than the other features.

Discussion Of course, a bigger dataset would be favourable. For larger class sizes it would make sense to cluster the students into groups of coders with similar programming styles. Since the more struggling students were found harder to distinguish, they propose to investigate features that specifically address these students.

For our Code Jam dataset we hope to achieve useful classifiers also by just using such high-level features. Our dataset has much more students, but only very few lines of code per student. In our project we do not want to identify the coder, but we want to group them (respectively their programs) into slow vs. fast buckets. Our objective is different in the way that our focus includes the functionality of the program. As we are not trying to identify the coder but the efficiency of the implementation, the high-level features that they used might prove to be less effective in our case. In our dataset we know a priori that all programs are solving the same task and that they are correct (but maybe inefficient). This allows us to isolate the efficiency of the implementation, where static high-level features might prove to be effective as well.

2 A Hierarchical Approach for Multi-task Logistic Regression

by Lapedriza, Àgata, David Masip, and Jordi Vitrià in Pattern Recognition and Image Analysis. Springer Berlin Heidelberg, 2007. 258-265. (Link to the PDF)

Background This paper presents a technique for automatic pattern classification for a set of related tasks. On an abstract level a classification task asks for assigning a single object to one out of a set of predefined classes, when given a set of features that describe this object. The task of single letter optical character recognition, which we saw in our first homework, is a good example for such a task.

Now, one often wants to train several classifiers for several tasks, that might share some common structure. Like if we want to train handwriting classifiers for person's unqiue style of writing, we want to make use of the fact that all of them use the english alphabet. Also we might want to get a generic classifier that is able to perform well for a new handwriting, that was not included in the training data.

The authors illustrate the problem using facial verification as we will see later on. But first we want to summarize the mathematical steps they proposed to extend logistic regression to multi-task learning. We will focus on the binary classification task, as this is what we want to use in our project.

Problem setup We are given a set of related binary tasks $T_1, \ldots, T_M$ with corresponding training data $D = \{S_1, \ldots, S_M\}$. Each dataset consists of some feature vector x of a fixed length d and a binary class y, so $S_i = \{(x_n^i, y_n^i)\}_{n=1,\ldots,N(i)}$ with $x_n^i \in \mathbb{R}^d$ and $y_n^i \in \{-1, 1\}$.

If we now train separate logistic regression classifiers f_i for each task T_i we would learn separate weight vectors $\mathbf{w}^{(i)}$ leading to a weight matrix $W = (\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(M)})$. Each classifier f_i uses the same logistic regression formula we saw in class, so

$$f_i(x) = P(y = 1 \mid x, T_i) = \frac{1}{1 + exp(-\mathbf{w}^{(i)} x^T)}.$$

They denote the negated log-likelihood estimator over all tasks as $L(D, W)$ and add a Gaussian prior to regularize the weights. If we just take the norm of the weight matrix we end up with a loss function $H(W)$ that corresponds to what we saw in class

$$H(W) = L(D, W) + \frac{1}{\sigma^2}\|W\|_2, \text{ where } L(D, W) = -\sum_{i=1}^{M}\sum_{n=1}^{N(i)} log(P(y_i^n \mid x_i^n, W)).$$

This would keep the different tasks completely unrelated. To couple the parameters they impose a prior distribution on the rows of W. So instead of the L2-norm of W as the regularization term, they take the norm of the *row-mean vector* $\bar{\mathbf{w}}$ of W into account, where $\bar{\mathbf{w}}_j = \sum_{i=1}^{M} w_j^{(i)}/M$. By adding the deviations of each $\mathbf{w}_{(i)}$ from $\bar{\mathbf{w}}$ to the regularization term, the goal is to get as $\bar{\mathbf{w}}$ the best possible fit across all tasks. They call the model hierarchical as now $\bar{\mathbf{w}}$ is the best fit across all $\mathbf{w}^{(i)}$ and each $\mathbf{w}^{(i)}$ is the best fit for its task. So they propose the loss function

$$G(W) = L(D, W) + R(W), \text{ where } R(W) = \frac{1}{\sigma_1^2}\|\bar{\mathbf{w}}\|_2 + \frac{1}{\sigma_2^2}\sum_{i=1}^{M}\|\mathbf{w}^{(i)} - \bar{\mathbf{w}}\|_2.$$

Note that we now have two variances σ_1^2 and σ_2^2 that are parameters of the model.

Training Algorithm To train the model, so to minimize the loss function, they applied the BFGS gradient descent method. Compared to what we did in our homework, the derivation of the gradient gets more complicated, as the regularization part of the gradient depends on $\bar{\mathbf{w}}$ now. They derive

$$\frac{\partial R(W)}{\partial w_k^{(s)}} = \frac{2w_k}{\sigma_1^2}\frac{\partial \bar{\mathbf{w}}_k}{\partial w_k^{(s)}} + \frac{2}{\sigma_2^2}\sum_{i=1}^{M}((w_k^{(i)} - \bar{\mathbf{w}}_k)\frac{\partial \bar{\mathbf{w}}_k}{\partial w_k^{(s)}}), \text{ where } \frac{\partial \bar{\mathbf{w}}_j(W)}{\partial w_k^{(s)}} = \begin{cases} \frac{\sigma_1^2}{\sigma_2^2 + M\sigma_1^2} & \text{if } j = k \\ 0 & \text{if } j \neq k \end{cases}.$$

Unfortunately the authors do not mention whether this extended model has an effect on the number of iterations needed until the gradient descent method converges. Also they do not state how to choose the model parameters σ_1 and σ_2.

Experiments To show the effectiveness of the approach they apply the model on a face verification task. This binary task is the decision whether a new image belongs to the learned subject or not. They used very few images to learn from (only 2 positive and 4 negative samples) and then trained up to ten tasks simultaneously. They found that with four or more tasks the new method outperforms standard logistic regression. With all ten tasks at once the multi-task solution achieves an error rate of 15% instead of 30%. While this sounds very promising they unfortunately do not provide any information on what the features were that they trained the classifier with.

Discussion Compared with standard logistic regression their approach has the main advantage that they can work with only very few samples per task. They found a quite generic model that is not targeted to any specific application and is reasonable for many multi-task problems. Also it is not considerably harder to optimize than training separate classifiers.

On the side of limitations one has to note that the method makes an assumption on how the tasks are related by the particular choice of loss function. This statistical prior restricts the way information between features for different tasks can be shared.

With respect to our project the authors note that multi-task learning is most effective (in comparison with single task learning) if there are only a few samples per class for training. As we collected several hundred submissions for each programming task for the dataset in our project, this might lead to only very limited improvement by multi-task logistic regression. However, we might consider training classifiers with only a few samples as well. This would correspond to the situation an organizer of a smaller contest would face or the situation during the beginning of a larger contest.

3 From Uncertainty to Belief: Inferring the Specification Within

by Kremenek, Ted, et al. in Proceedings of the 7th symposium on Operating systems design and implementation. USENIX Association, 2006. (Link to the PDF)

Background Traditional software error detecting tools follows the idiom: *if they do not know what to check, they cannot find bugs.* Hence those tools require a set of specifications that document what a program should do in order for the tool to discern good program behavior from bad. This paper presents a novel framework based on factor graphs for automatically inferring specifications directly from programs, following a different idiom: *the more something behaves like an X, the more probable it is an X.* In particular, the paper analyzed the problem of inferring what functions in C programs allocate and release resources.

Many C programs use the ownership idiom: a resource has at any time exactly one owning pointer, which must release the resource. Ownership can be transferred from a pointer by storing it into a data structure or by passing it to a function that claims it. Functions that allocate and release resources (such as file handles and memory) vary widely in their implementations, but their interfaces are used nearly identically.

A function that returns an owning pointer has the annotation *ro* (returns ownership) associated with its interface. A function that claims a pointer passed as an argument has the property *co* (claims ownership) associated with the corresponding formal parameter. In this model, allocators are *ro* functions, while deallocators are *co* functions. Given the following code fragment, we want to determine if *fopen* is an *ro* and if either *fread* or *fclose* are *co*'s.

```
FILE *fp = fopen("myfile.txt", "r");
fread(buffer, n, 1000, fp);
fclose(fp);
```

If *fopen* is an *ro*, then to avoid a resource-leak, either *fread* or *fclose* must be a *co*. To avoid a use-after-release error, however, *fread* cannot be a *co*. This leaves *fclose* being a *co*. It's easy to check that other assignments will lead to a bug in the code. Assuming that the code fragment is likely to be bug-free, we are not certain of the assignment, but we believe that some conclusions are more likely than others. The rules for ownership is summarized by the DFA(Deterministic finite automata) in Figure 1.

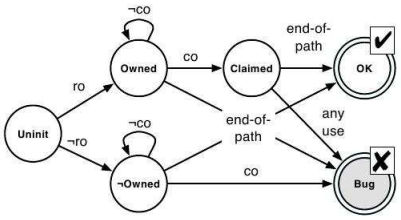

Figure 1: DFA summarizing basic ownership rules.

Now let's define *annotation variables* for the example: *fopen:ret*, *fread:4* and *fclose:1*. The variable *fopen:ret* corresponds to the possible annotations for the return value of *fopen*, and has the domain $\{ro, \neg ro\}$. The variables *fread:4* and *fclose:1*(where :i denotes the ith formal parameter) have the domain $\{co, \neg co\}$. We denote the set of annotation variables as $\mathbf{A}$. In our example, $\mathbf{A} = \{$*fopen:ret*, *fread:4*, *fclose:1*$\}$. Then we define factors in annotation factor graphs. Each factor f_i is a map from the possible values of a set of variables $\mathbf{A}_i (\mathbf{A}_i \subseteq \mathbf{A})$ to $[0, \infty)$. Factors "score" an assignment of values to a group of related annotation variables, with higher values indicating more belief in an assignment. We could define a *check* factor for our example, $f_{\langle check \rangle}$(*fopen:ret*, *fread:4*, *fclose:1*):

$$f_{\langle check \rangle}(\dots) = \begin{cases} \theta_{\langle ok \rangle} & \text{if DFA} = \text{OK} \\ \theta_{\langle bug \rangle} & \text{if DFA} = \text{Bug} \end{cases} .$$

Once we represent individual beliefs with factors, we combine a group of factors $\{f_j\}_{j=1}^{J}$ into a single probability model by multiplying their values together and normalizing:

$$P(\mathbf{A}) = \frac{1}{Z} \prod_{f_i \in \{f_j\}_{j=1}^{J}} f_i(\mathbf{A}_i).$$

The normalizing constant Z causes the scores to define a probability distribution over the values of A, which directly translates into a distribution over specifications.

By using a more complicated DFA and introducing prior belief factors to indicate initial preferences, like $f_{\langle ro \rangle}$, they evaluated effectiveness of the framework on five codebases: SDL, OpenSSH, GIMP, and the OS kernels for Linux and Mac OS X(XNU). For each codebase, starting with zero initially provided annotations, they observed an inferred annotation accuracy of 80-90%, with often near perfect accuracy for functions called as little as five times.

Strength The key strength of the approach is that it can incorporate many disparate forms of evidence. For instance, besides behavioral signatures and prior beliefs, we often have a volume of valuable, non-numeric ad hoc information such as domain-specific naming conventions (e.g., a function name containing the word "alloc" often implies the function is an allocator). Also the framework is pragmatic even for rapidly evolving codebases: the process of inferring annotations and using those annotations to check code with automated tools can be integrated into a nightly regression run.

Weakness and Relation to Our Project The paper only applied their approach to the ownership problem. We don't know if it's also good for other specifications, like the interesting code efficiency problem in our project. Moreover essentially the paper is discussing the syntactical errors. However we are more interested in the semantical errors, that is whether the code could solve a particular problem. Although the syntactical error checking could rule out a lot of programms, it still remains unclear if the program could solve the particular problem or not.