Using Salience to Segment Desktop Activity into Projects

Daniel Lowd
Dept. of Computer Science & Engineering
University of Washington
Seattle, WA 98195-2350 – USA
lowd@cs.washington.edu

Nicholas Kushmerick
Decho Corporation
615 2nd Ave, Suite 280
Seattle, WA 98104-2245 – USA
nicholask@decho.com

ABSTRACT

Knowledge workers must manage large numbers of simultaneous, ongoing projects that collectively involve huge numbers of resources (documents, emails, web pages, calendar items, etc). An activity database that captures the relationships among projects, resources, and time can drive a variety of tools that save time and increase productivity. To maximize net time savings, we would prefer to build such a database automatically, or with as little user effort as possible. In this paper, we present several sets of features and algorithms for predicting the project associated with each action a user performs on the desktop. Key to our methods is *salience*, the notion that more recent activity is more informative. By developing novel features that represent salience, we were able to learn models that outperform both a simple benchmark and an expert system tuned specifically for this task on real-world data from five users.

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation

Author Keywords

Information workers, time tracking, interruption recovery, activity recognition, machine learning, logistic regression.

INTRODUCTION

Knowledge workers must deal with an overwhelming amount of information while multitasking many projects. When managing a large number of resources (such as desktop application files, web pages, email and calendar entries), the primary challenge is finding specific resources. For managing numerous simultaneous tasks or projects, two challenges are recovering from interruptions and tracking time.

Over the past two decades, there has been a wide variety of research addressed at various aspects of these problems, resulting in numerous software systems to help people manage information overload, enhance collaboration, and manage their tasks. Email in particular has received considerable attention; see for example [8, 9, 1, 5, 4]. Our work in this paper is part of Smart Desktop, a desktop information management application for information workers.

Smart Desktop was inspired by the TaskTracer project [12, 3]. TaskTracer treats tasks as virtual collections of related documents, webpages and email, and uses a variety of heuristic and machine learning techniques to link new information to their tasks. The TaskTracer user interface allows people to manage their information in terms of their logical tasks, instead of in terms of physical storage (email folders and servers, web and file servers, file system directories, etc). Both TaskTracer and Smart Desktop work by instrumenting the user's desktop applications. Each time a resource is opened, brought into or out of focus, closed, etc., we capture the action with a timestamp. Each of these actions is also associated with one of the user's defined projects. These associations form an activity database of projects, resources, and time segments that link the two together.

Such an activity database can power many time-saving and productivity-enhancing technologies. For example, by sorting one's list of resources by project (instead of by physical location), users can remember what they were working on and find information more easily. Also, by summing the time segments associated with each resource in a project, users can track how their time is spent.

Since it is too expensive for users to manually specify their tasks at every moment, Smart Desktop relies on an task predictor component that attempts to automatically infer the current project. This classification is based on two kinds of context: the characteristics of the resource being accessed at the given moment (words, path, file type, etc.) and aspects of the user's ongoing activities (recent projects, resources, etc.). Predictor accuracy is essential to the overall user experience: each wrong prediction is either one more error in the database (hence less accurate information retrieval, time tracking, etc.), or one more manual user correction.

Unlike traditional classification problems, the project labels associated with the time segments are not independent and identically distributed. First, the data drift rapidly: word-project correlations may arise and then disappear quickly, and many kinds of projects spring to life or are terminated very quickly. Second, due to people's limited multitasking ability, there is considerable inertia in the active projects that people are likely to switch between at any given time. Hidden Markov and similar models would help, but they only capture relationships between time segments that are very

IUI'09, February 8–11, 2009, Sanibel Island, Florida, USA.

close to each other. We solve this problem by developing novel sets of features to capture *salience*, the notion that recent activity is more informative. These features enable models that can represent both long- and short-range interactions, adapt quickly with limited data, and still be efficient enough to be unobtrusive for the user.

TASK PREDICTION AS A MACHINE LEARNING TASK

Problem Statement. Our goal is to predict the project the user is currently working on, given the current and previous actions and the user's project labels. More formally, we wish to find a function f that, given a sequence of actions $(a_0, \ldots, a_t)$ and project labels $(p_0, \ldots, p_{t-1})$, predicts the current project, $\hat{p_t} = f(a_0, \ldots, a_{t-1}, a_t, p_0, \ldots, p_{t-1})$.

Each action a_i corresponds to a desktop event such as opening a file, bringing focus to an application window, clicking a hyperlink, sending an email, etc. Each project label p_i corresponds to the project the user associates with that action. Note that different users may organize their work into projects in different ways. Furthermore, indentical actions could be associated with different projects in different contexts: opening a browser window for www.iuiconf.org could either be associated with the project of writing an IUI paper or the project of planning travel to the IUI conference.

Linear Classifiers. A linear classifier $c(x)$ uses a weighted set of feature values to discriminative between two or more classes. In the simplest, two-class case, $c(x) = \text{sign}(\mathbf{w} \cdot \psi(x))$, where $\Psi(x)$ is a vector of feature values for the instance x. Features may take on any real value, but are commonly 0/1-valued to indicate the presence or absense of particular words or other characteristics. In the multiclass case, different classes weigh the features differently. The simplest way to accomplish this would be to use one weight vector per class, or in our case, project:

$$c(x) = \arg\max_{p \in P} \mathbf{w}_p \cdot \Psi(x) \qquad (1)$$

Equivalently, we could use a single weight vector that encompasses all classes and make the feature vector a function of the class:

$$c(x) = \arg\max_{p \in P} \mathbf{w} \cdot \Psi(x, p) \qquad (2)$$

Thus far, we have assumed that each weight applies to exactly one class so the number of weights grows linearly with the classes. However, the second formulation allows for shared features that have different values for different labels.

For our particular problem, each instance x_t is an unlabeled action plus the history of all previous action/project pairs: $(a_0, \ldots, a_t, p_0, \ldots, p_{t-1})$. The set of labels is the set of user-defined projects, P.

FEATURES

Resource Features. Smart Desktop instruments a wide range of desktop applications such as web browsers, email clients and office document applications. Smart Desktop captures a wide variety of features from a user's resources. We refer to these as *resource features* to distinguish them from features that depend on the sequence of past actions and resources. Resource features are grouped into *feature types*.

The following feature types are general and apply to all resources: Full URI of the resource (e.g., http://www.iuiconf.-org/stats/index.html; file://c:/nick/budget.xls); Any meaningful sub-path of the URI (e.g., http://www.iuiconf.org/stats, http://iuiconf.org; file://c:/nick, file://c:); Any word in the title of the document, excluding stop words; Any of the first $\approx$ 100 words in the body of the document, excluding stop words; and Resource type (e.g., webpage, email, calendar entry, Excel document, etc.).

In addition, Smart Desktop collects the following feature types which are specific to email: The sender from the From: field (e.g., nick@smartdesktop.com); The sender's domain (e.g., smartdesktop.com); Any recipient (To:, Cc:, or Bcc: field); The mixture of sender and all recipients (sorted so that, for instance, the emails 'From: Barack; To: John' and 'From: John; To: Barack' have the same feature value); A unique identifier shared by all emails with a single thread; The subject line, cleaned of modifiers such as 'Re:' and 'Fwd:'; The name of each attached file; and Folder in which the email is stored.

Past Project Features. Users typically perform several actions in the same project before switching tasks. This makes the last project highly predictive of the next project. We can incorporate this information with an additional set of features. Specifically, for each project p we add a feature to represent "the last project was p." We can analogously define features for the projects of the past k time steps. Larger values of k give us more contextual information, but suffer from diminishing returns since projects from further in the past are less informative. We use $k = 4$ in our experiments; additional time steps were not helpful.

Salience Features. Recent activity within a project could be very different from the long-term trends. We expect a good model to recognize the similarity between current activity and recent activities, and to use this evidence in project prediction. We can represent this trend with a new set of features, each representing the fact that one of the resource features of a particular type was last seen with a particular project. This represents the fact that a feature in the current instance has a "salient" relationship to a particular project. Instead of having a separate salience feature for each of the original features, we use one salience feature for each feature type. If multiple features of a particular type last appeared with the same project, then we simply increase the value of the associated salience feature.

For example, suppose the user opens a web page with title words "international" and "conference," and with body words "call" and "papers." The last event with the title word "international" was in the Travel project, so we give the salience feature Title-Travel a value of 1. For "conference," "call," and "papers," the last events were in the Publications project, so the Title-Publications feature takes on a value of 1 and the Body-Publications fature takes on a value of 2. This informs the classifier that “papers” has recent associations with Publications, even if it has appeared more often with the Office Supplies project in the past.

Shared Salience Features. The previous feature sets were designed to be multiplied by the number of projects when constructing the full feature vector, $\Psi(x, p)$. This allows projects to weight different features differently. The downsides to such an approach are the large number of weights, which increases the risk of overfitting, and the strict dependence on the exact set of projects, preventing us from generalizing to new projects or different users.

We can avoid those problems using shared salience features. This allows us to represent complex, changing project descriptions with just a small number of weights that can be transferred to other users. We construct a shared salience feature for each type of resource feature. These features have different values for different projects but are weighted the same across all of them. The value of each shared salience feature is the number of observed features of a particular type that were seen most recently with a particular project.

Specifically, let the salience feature $s_i(a_t, p_t)$ be defined as the number of features f for which: f is resource feature present in a_t; f is of type i; and $p_t = p_j$, where j is the index of the most recent previous time segment for which f was present in a_j. If our feature function $\Psi(x, p)$ uses only these shared salience features, then we can write the prediction as follows:

$$f(x) = \arg\max_p \mathbf{w} \cdot \Psi(x, p) = \quad \arg\max_{\hat{p_t}} \sum_i w_i s_i(a_t, \hat{p_t})$$

ALGORITHMS

We now describe methods for learning the weight vector, $\mathbf{w}$, from data. With the online methods, naive Bayes and the passive-aggressive algorithm, we adjust the weights after seeing each new labeled example. With the offline methods, logistic regression and support vector machines, we learn weights using training data from other users and apply the weights to the target user. We also discuss a manually constructed expert system.

Naive Bayes. Naive Bayes (NB) is a statistical model that assumes all features $\Psi(x_t)$ in a classification problem are independent given the class:

$$P(\hat{p_t}|\Psi(x_t)) = \frac{1}{Z_t} P(\hat{p_t}) \prod_i P(\Psi_i(x_t)|\hat{p_t})$$

where Z_t is a normalization constant so that all probabilities sum to one, and Ψ_i refers to the ith feature of x_t. The marginal probabilities $P(\hat{p_t})$ indicate the relative frequency with which each project is seen, and the conditional probabilities $P(\Psi_i(x_t)|\hat{p_t})$ indicate the frequency for seeing each feature with each class. These are estimated from counts in the training data and can be updated in constant time as more data is seen. We used the multinomial model [11], in which the product $\prod_i$ ranges over only those features for which $\Psi_i(x_t)$ is true. This tends to work better than the alternative Bernoulli model for sparse features. The multinomial model also supports features that appear multiple times in a single resource by including the corresponding probability multiple times in the product. This allows us to handle salience features which may have values greater than one. We smoothed all estimated probabilities with a Laplace correction, adding one to each count.

To see that naive Bayes is equivalent to a linear model, consider the log probability[1], which is a sum over all true features weighted by log probability. Therefore, naive Bayes is a special case of our multiclass linear model, where the feature and weight vectors are defined as follows: Each feature in the vector $\Psi(x, p)$ is a conjunction $\Psi_i(x) \wedge (p = p_j)$ for some feature index i and project index j, and the matching weight in $\mathbf{w}$ is the log probability: $\log P(\Psi_i|p_j)$. We incorporate the class priors via “dummy” features, true$\wedge(p = p_j)$, with weight $\log P(p_j)$. Note that since naive Bayes is designed to work with a fixed feature vector, it is somewhat less flexible than our general multiclass formulation and does not naturally support shared salience features.

Passive-Aggressive Algorithm. The passive-aggressive algorithm (PA) [2] is an online max-margin method with a simple update rule reminiscent of the perceptron. For an instance (x_t, p_t), the *margin* m_t is the minimum difference between the score assigned to the true label p_t and the score assigned to any other label, p'_t:

$$m_t = \min_{p'_t \neq p_t} \mathbf{w}_t \cdot (\Psi(x_t, p_t) - \Psi(x_t, p'_t)) \tag{3}$$

Maximizing the margin is equivalent to minimizing the magnitude of the weight vector while maintaining a margin of 1. If an example falls within the margin, then PA incurs a loss proportional to how much the margin constraint is violated:

$$l_t = \begin{cases} 0 & m_t \geq 1 \\ 1 - m_t & \text{otherwise} \end{cases} \tag{4}$$

This is often referred to as the *hinge loss*.

The passive-aggressive algorithm is so named because, when the loss is zero, it passively leaves the weight vector unchanged; and, when the loss is greater than zero, it aggressively adjusts the weight vector so that p_t no longer violates the margin constraint this last instance (x_t, p_t) satisfies the margin constraint for p'_t. (In the multiclass case, it is possible for some other $p''_t \neq p'_t$ to still violate the margin constraint.) Of the many possible weight updates that would satisfy the margin, PA chooses the one that minimizes the distance from the old weight vector. This happens to have a simple, closed-form solution:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \tau_t(\Psi(x_t, p_t) - \Psi(x_t, p'_t)) \tag{5}$$

where,

$$\tau_t = \frac{l_t}{\|\Psi(x_t, p_t) - \Psi(x_t, p'_t)\|^2 + \frac{1}{2C}}$$

C is a parameter for limiting the rate at which the weights are updated. In the limit as C approaches infinity, the update

[1] Since log is a monotonic function, this has no affect on the $\arg\max$.

rule exactly satisfies the margin constraint. With smaller values of C, the weights will remain closer to their previous values, mitigating the effect of noise.

For a full description of the passive-aggressive algorithm, analysis, etc. see Crammer *et al.* [2].

Logistic Regression. Logistic regression (LR) is the discriminative analog of naive Bayes. Instead of modeling the full joint probability of the features and project label, it models the conditional probability of the project label given the features:

$$P(p|x_t) = \frac{1}{Z_t} \exp(\mathbf{w} \cdot \Psi(x_t, p))$$

As with naive Bayes, the log probability is a linear function:

$$\log P(p|x_t) = \mathbf{w} \cdot \Psi(x_t, p) - \log Z_t$$

When predicting the most likely project, the effect of the log partition function, $\log Z_t$, can be ignored because it is constant across all classes. Given a set of training examples, finding a weight vector to maximize their conditional likelihood is a convex optimization problem that can be solved with methods such as the quasi-Newton L-BFGS algorithm [10]. To avoid overfitting, we follow the common practice of applying a zero-mean Gaussian prior to each weight (effectively an L2 norm).

Support Vector Machines. Like PA, support vector machines (SVMs) [13] are a max-margin method. Let $h(\mathbf{w})$ be the hinge loss (Equations 3 and 4) summed over the entire set of training examples:

$$h(\mathbf{w}) = \sum_t \max_{p' \neq p_t} \max(0, 1 + \mathbf{w} \cdot (\Psi(x_t, p') - \Psi(x_t, p_t)))$$

Support vector machines (SVMs) optimize a weighted sum of this total hinge loss and the magnitude of the weight vector, where the parameter C trades off generalization with training set accuracy: $\|\mathbf{w}\| + Ch(\mathbf{w})$. Note that, unlike PA which updates with each new example, SVMs find a single weight vector $\mathbf{w}$ for all instances (x_t, p_t). There are many different algorithms for solving this optimization problem.

Expert System. A final approach is to construct a fully custom solution using a combination of expert knowledge and educated guesses. This has the potential advantage of providing a much better fit to the problem. However, such systems are also more complex and expensive to build. Furthermore, expert knowledge may be wrong or incomplete, failing to properly capture important patterns in the data. The advantage or disadvantage of choosing an expert system often comes down to the availability of knowledge versus data. Many of the people developing Smart Desktop are also regular users of the product and fall within the target demographic of knowledge workers. Therefore, experts were plentiful while data was sparse, and so the expert system approach made sense.

The hand-crafted predictor developed for Smart Desktop uses case-based reasoning to generate a prediction from a diverse set of features. The cases consider the file type, the context in which the action takes place, and whether or not enough information is available to make a reliable prediction from a single type of feature. Features believed to be more reliable are given higher priority. For some cases, predictions according to several different feature sets are combined.

EVALUATION

Data. The data Smart Desktop uses is, by its very nature, quite personal. Few persons or companies are comfortable with disclosing the text of every email read or written, or the URL of every website visited over a span of time. We dealt with this problem by obfuscating the data from five regular users of the Smart Desktop product, all employees of the company. The obfuscation replaces most words with a randomly generated replacement string of the same length. Words with special meaning, such as stop words, were left as-is. The goal was to provide as much anonymity as possible while still obtaining the exact same results that we would on non-anonymized data. In order to create a gold standard, each user cleaned the project labels for a complete two-week period. Each user had from 161–1181 resources distributed over 26–40 projects; these resources were about $\frac{2}{5}$ email, $\frac{2}{5}$ web pages, and $\frac{1}{5}$ files. We collected from 465–5036 time segments of activity data for each user.

Methodology. We make the simplifying assumption that users immediately correct all wrong predictions as they use the system. This allows us to use the labels from all previous time segments in predicting the current time segment.

We evaluated each algorithm on each user's data with many different feature combinations. For brevity, we associate sets of features with one-letter codes: R for resource features, P for past project features, S for salience features, s for shared salience features, and s' for shared salience features with infinite weight on the URI feature. The infinite weight forces the last known project to be predicted for resources that have been seen before, effectively assuming that each resource is associated with a single project. When using resource features (R), we make the same assumption, which greatly reduces the errors of NB and PA. The feature combinations we used are listed in Table 1.

For NB and PA, we started with an empty model and learned the parameters online for various feature set combinations. PA turned out to be very sensitive to the C parameter, so we tuned it separately for each feature set and user, using the value that worked best for the majority of the other users. For LR and SVM, we used a leave-one-out procedure in which we trained a different set of weights for each user using the databases of the other four users. LR and SVM used only the shared salience features, since the other features do not generalize across different users. We learned LR weights using the limited-memory quasi-Newton algorithm L-BFGS [10]. For the SVM weights, we used the SVM-Struct API by Joachims [6] to run the algorithm described in [7]. We have little danger of overfitting with so few weights, so we used a large value of $C = 100$ in order to focus on training-set accuracy more than generalization. The expert system ran on the same simulated events as our algorithms.

Table 1. Errors on each user's data.

Method	Features	Total	User 1	2	3	4	5
Baseline		1215	229	161	508	192	126
NB	R	1096	267	141	402	178	108
	R+P	1002	220	139	368	168	107
	R+S	941	184	143	328	170	116
	R+S+P	923	176	142	323	167	115
PA	R	914	187	131	321	168	107
	R+P	899	171	127	321	169	111
	R+S	1091	216	157	399	192	127
	R+S+P	1090	217	155	397	194	127
	s	1116	214	123	494	178	107
	s'	910	172	130	348	159	**101**
	R+s'	885	167	128	329	159	102
	R+P+s'	882	166	128	327	158	103
LR	s	1047	155	121	508	158	105
	s'	871	155	121	332	158	105
SVM	s	1011	**142**	**115**	501	**149**	104
	s'	**815**	**142**	**115**	**305**	**149**	104
Expert		910	149	147	335	167	112

We compared these algorithms to a a simple baseline algorithm. The baseline predicts the project last associated with the current URI if it was seen before. Otherwise, it predicts the last project for the current resource type, or the last project if we are seeing a resource type for the first time.

Results. In Table 1, we list the number of errors each algorithm made on each user's data, as well as the totals across all users. This represents the number of corrections users would have to make over a two-week period of time in order to maintain a fully accurate database.

For naive Bayes, the additional information in the past project and salience features clearly makes a difference: with the additional features, NB is competitive with the expert system. This is impressive for such a simple algorithm.

The passive-aggressive algorithm is more accurate than NB overall, using either the R or R+P feature sets. The many salience features (S), however, seem to "distract" PA from more informative features, significantly reducing accuracy. PA already captures some salience information indirectly, since its weights are adjusted to fit the most recent instance, not cumulative statistics as in naive Bayes. PA achieves its greatest accuracy with the help of shared salience features (s or s'), which have the advantage of being fewer and more informative than the those of S. Using a fixed large weight for the URI shared salience feature, as with s', seems to work much better than learning it online, as with s. With the feature sets R+s' or R+P+s', PA beats the expert system for four out of five users.

Logistic regression and support vector machines perform best overall, with SVMs showing a modest advantage. The probabilities produced by logistic regression might still be useful for decision-making applications, but SVMs excel at accuracy. The success of these algorithms suggests that shared salience features are effective and that their weights generalize well across different users.

Except for User 3, fixing a large weight for the URI shared salience feature has no effect on the results. This is what one might expect, since few resources are associated with more than one project. To determine what makes User 3 so different, we compared models learned on User 3's data to the models learned on the other users' combined data. We found that the weights for three of the 14 feature types ('From:' address, mail folder, and attachment filename) had opposite signs. This demonstrates that features that work well for one user may not work well for another. Giving a large weight to the URI shared salience feature overruled many of these mistakes. In a longer-term study, some automatic adaptation might be possible by training on the user's own data.

Overall, we can conclude that salience is important in segmenting desktop activities, and that surprisingly simple models can outperform even carefully engineered expert systems in this domain. Our best-performing method, SVM, made 10% fewer errors and was more accurate than the expert system for every user. We also observed that differences in how users organize their activity into projects can lead to large differences in the optimal feature weights. Given months of data, the best approach might be to train LR or SVM on the user's own data.

REFERENCES

1. W. Cohen, V. Carvalho, and T. Mitchell. Learning to classify email into speech acts. In *Proc. Conf. Empirical Methods in Natural Language Processing*, Barcelona, 2004.
2. K. Crammer, O. Dekel, S. Shalev-shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:2006, 2006.
3. A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker. Tasktracer: A desktop environment to support multi-tasking knowledge workers. In *Proc. Int. Conf. on Intelligent User Interfaces*, San Diego, CA, 2005.
4. M. Dredze and H. Wallach. User models for email activity management. In *Workshop on Ubiquitous User Modeling, Int. Conf. Intelligent User Interfaces*, 2008.
5. Y. Huang, D. Govindaraju, T. Mitchell, V. Carvalho, and W. Cohen. Inferring ongoing activities of workstation users by clustering email. In *Proc. Conf. Email and Anti-Spam*, Mountain View, CA, 2004.
6. T. Joachims. *Making large-scale SVM learning practical*, chapter 11. MIT Press, Cambridge, MA, 1999.
7. T. Joachims. Learning to align sequences: A maximum-margin approach. Technical report, Cornell University, 2003.
8. R. Khoussainov and N. Kushmerick. Email task management: An iterative relational learning approach. In *Proc. Conf. Email and Anti-Spam*, 2005.
9. N. Kushmerick and T. Lau. Automated email activity management: An unsupervised learning approach. In *Proc. Int. Conf. Intelligent User Interfaces*, 2005.
10. D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(3):503–528, 1989.
11. A. Mccallum. A comparison of event models for Naive Bayes text classification. In *In AAAI-98 Workshop on Learning for Text Categorization*, pages 41–48. AAAI Press, 1998.
12. J. Shen, L. Li, T. Dietterich, and J. Herlocker. A hybrid learning system for recognizing user tasks from desk activities and email messages. In *Proc. Int. Conf. Intelligent User Interfaces*, pages 86–92, Sydney, Australia, 2006.
13. V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, NY, 1995.