Predicting User Purpose on BranchOut

Ben Holtz Ben Lasley Garrett Schlesinger

June 11, 2012

Abstract

In early 2012, BranchOut, a professional networking layer on top of Facebook, experienced a period of explosive growth, with multiple weeks of over 100,000 users joining each day. The BranchOut team hypothesized that there are distinct clusters of users on the site, including but not limited to job seekers, recruiters, and networkers. Through techniques such as frequent itemset mining, Markov Chains, and k-means clustering, we validated the existence of such user types. This paper details the process and provides semantic interpretation of these clusters. Further, through techniques such as Naïve Bayes classification and Support Vector Machine classification we determined that it is possible to identify with high precision and recall the type of user that is using the site within their first several uses of the application.

1 Introduction

1.1 Motivation

For consumer web services, knowing the intent of a new user can be essential to serving them content that will keep them engaged. However, getting a user to self-identify what they want to do is, at best, a user experience challenge and, at worst, a major barrier to entry. Even if they would self identify, some users don't know what they want out of the site. BranchOut is one such service which allows Facebook users to network in a professional capacity, post openings at their companies, and apply for jobs. Because BranchOut offers a multitude of services for users interested in different aspects of professional networking, being able to identify user intent quickly and serve them more appropriate content could dramatically increase user retention - the number of users who decide to stay more than a few clicks and become active members of the site. To try to investigate ways of predicting user intentions, BranchOut provided us with data they have logged about users and their actions on the site.

1.2 The BranchOut Dataset

BranchOut's data is generated starting when a user installs the BranchOut app to their Facebook profile. At that time, all of the user's profile information is copied, as well as their list of friends and any visible profile information from those friends. BranchOut also has data on the skills selected by users, the jobs they post, and the actions BranchOut users take on the BranchOut website.

BranchOut provided us with SQL dumps of many of their tables, with user ids obfuscated for privacy. Included in the data were basic user profile information, the Facebook friend graph as scraped by BranchOut, the log of all user actions from November through March, as well as users' identified skills and any uploaded resume data. We chose to focus on the user action data, as we believed this

data would be most helpful in trying to identify categories of users and in predicting what category a new user should belong to.

1.3 BranchOut User Types

In mid April, we met with several members of the BranchOut team to discuss project ideas and gain insight into the data we would be mining. They shared anecdotal reports of macro trends in the data, such as the (somewhat obvious) job seeker versus recruiter, the professional networkers who may be generating sales leads, the users interested mostly in making new connections, and the rare 'power-user' - the user who tries everything and spends substantial time on the site. However, these were only hypothesized, which suggested that we could try to confirm them through the action data. If, in fact, some sort of meaningful user type were present in the data, we could then move forward to try and identify the type of an individual user, using as little data as possible. This tool would aid BranchOut's efforts to retain new users as they try the site for the first time, and inform email campaigns to entice users back to the site.

1.4 Goals

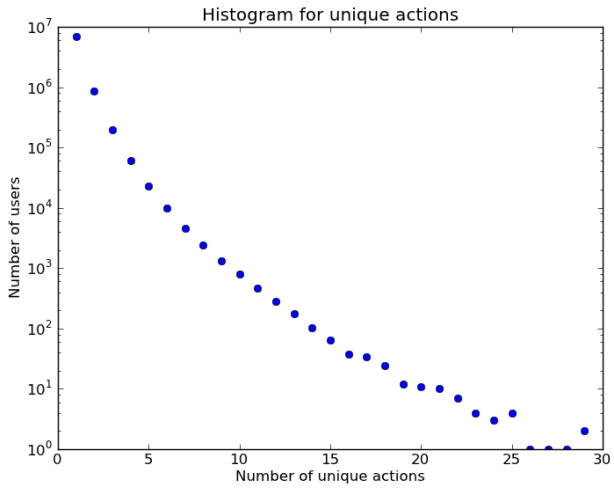
Armed with BranchOut's data and motivated by several data-driven questions from the BranchOut team, we set out to first identify what, if any, categories of users there are using the site, and if those categories had any meaning we could make sense of. If we were able to categorize the users, we would then examine strategies for predicting what category a particular user belonged to, limiting ourselves to only the data available when the user first joins the site.

2 Methods

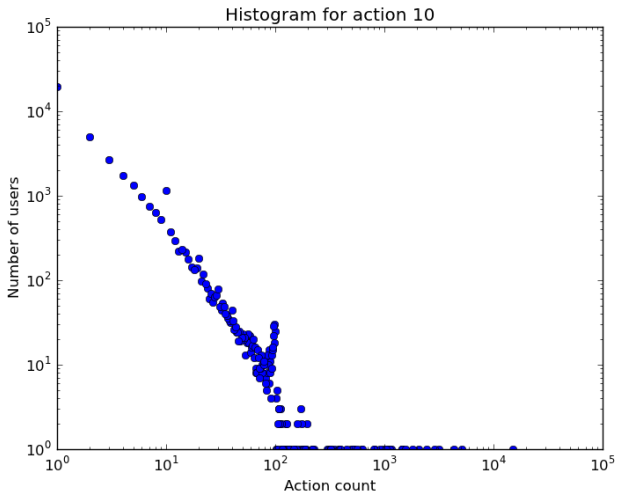
2.1 Data processing

First, we defined a feature space to represent users. To do so, we familiarized ourselves with the contents of the engagements table, which logs every action that occurs on the site. Each row of the engagements table represents a single action on BranchOut. Rows take the form [*user id, couple id, object id, action id, timestamp*], where *user id* is the unique id of the user performing the action (the "subject"), *couple id* is the unique id (possibly null) of the user receiving the action (the "indirect object"), *object id* is the unique id (possibly null) of the object (e.g. job posting) being acted on (the "direct object"), *action id* is an integer code from 0-80 describing the action (the "verb"), and *timestamp* is the YYYY-MM-DD H:M:S time at which the action occured. For example, *action id* 24 corresponds to a profile view, so the entry "1 2 0 24 2011-11-10 07:30:12" means that user 1 viewed the profile of user 2 at 7:30AM on November 10, 2011. To account for users being acted upon in our representation, we defined actions 81-161 to indicate that the active user was actually acted upon with the action 81 less than the given action. For example, if we say user 2 performed action 105, we mean that somebody performed action 24 on user 2, i.e. somebody viewed user 2's profile.

We then represented each user by his or her actions on BranchOut - that is, we associated to each user a vector of his or her actions from the engagements table. Depending on the type of analysis we were performing, we used one of two representations. The first representation considers each user as a 162-dimensional vector, where the value in the i^{th} entry was a raw count of the number of times the user had performed action i. In the second representation, each user was viewed as an in-order list of



(a) Histogram of the number of users who perform unique actions



(b) Count of times a user performs FriendRequestSent

Figure 1: Initial histograms

their actions. We will refer to the first representation as the "frequency" representation and the second as the "time-series" representation. To store user representations in these formats, we wrote simple map-reduce streaming scripts in Python to run on Amazon's Elastic MapReduce.

2.2 Preliminary examination

To begin our investigation of the BranchOut action space, we plotted basic statistics on the distribution of actions, generated frequent pairs and triples of actions, and trained a Markov model of user actions. These explorations supported our hypothesis of the exisitence of distinct user types and gave us further direction in clustering users.

2.2.1 Basic Statistics

We began by generating a number of histograms over the data, examining how frequently users would take various actions. Nearly all of the 81 actions was used, with varying degrees of popularity. We also split the data up by week, separating weeks at the weekly and daily low point for actions, Tuesday at 9:30am UTC, and looked for shifting trends over time. Importantly, we found that while every unique action type is performed, most users do not use more than one or two, and no user tried more than 29, as shown in Figure 1a.

We also found that many of the actions logged were created after the initial data collection, which could skew the clustering for users who were primarily active early in our data. However, the vast majority of our users joined very close to the end of our data set, (a result of apparent exponential growth), so this should have little effect.

We also examined the number of times users would perform each specific action. For nearly all actions, this resulted in a power-law distribution. However, for the 'FriendRequestSent' action, we found some strange features (shown in Figure 1b). There are two anomalies on the plot, which after examining other actions and talking with BranchOut, we determined are due to the user interface - the first at 10 actions, which shows a slight bump, likely because there are 10 friends showing on the screen

to invite, creating a barrier to adding more than 10. There is also a large deviation from power-law around the 100 action mark, which according to BranchOut is due to users being asked to invite friends in batches of 50 when they first join the site, and users tire of this feature after two uses.

2.2.2 Frequent Itemset Mining

From our basic analysis of the data, we knew that the action space is incredibly sparse, namely that the maximum number of unique actions performed by a single user was 29. However, 70 actions were performed by at least one user, so we suspected that there are statistically significant sets of actions that users frequently perform, i.e. a user that performs action x is lilkely to also perform action y. We computed such frequent pairs and triples of actions, using the "frequency" representation, as follows.

First, we counted the support for each action, i.e. for each action x, the number of users who performed action x. We set a support threshold of 5000. We then generated candidate frequent pairs by taking all combinations of actions with support over this threshold and counted the support for each of these candidates. Our "true" frequent pairs were the candidates with support over the threshold of 5000. We then sorted the frequent pairs in decreasing order of *lift*. The *lift* of a pair x, y is defined as

$$lift(x, y) = \frac{Support(x \cup y)}{Support(x) \times Support(y)}$$

and gives the ratio of support for a pair and the expected support if x and y were independent. The top 20 frequent pairs given by *lift* are as follows:

1	*MessageRead, MessageReply*	11	*FriendRequestSent, FriendRequestAccepted*
2	*MessageSent, MessageReply*	12	*JobCreated, JobViewed*
3	*MessageSent, MessageRead*	13	*FriendRequestReceived, FriendRequestIgnored*
4	*FriendRequestSent, GetIntroduced*	14	*MessageSent, FriendRequestSent*
5	*JobViewed, JobApplied*	15	*FriendRequestReceived, FriendRequestAccepted*
6	*MessageReceived, MessageReply*	16	*MessageSent, CreateEndorsment*
7	*MessageReceived, MessageRead*	17	*MessageReceived, FriendRequestAccepted*
8	*FriendRequestAccepted, SearchJob*	18	*MessageSent, MessageRead*
9	*MessageReceived, MessageSent*	19	*FriendRequestSent, CreateEndorsment*
10	*FriendRequestAccepted, MessageRead*	20	*FriendRequestAccepted, CreateEndorsment*

A few particular things are notable from these frequent pairs. First, as expected, actions which deal with sending messages (sending, receiving, replying, etc.) are all frequent pairs, as are pairs of items which deal with friend requests. Indeed, when we computed frequent triples, triples containing these actions were also frequent. More interesting is that both *JobViewed, JobApplied* and *JobCreated, JobViewed* are frequent pairs, but *JobCreated, JobApplied* is not a frequent pair. This preliminary result provides support for our hypothesis that distinct user types, in particular a job-seeking and a job-posting user type, exist. There is a set of users that posts jobs and views jobs (we learned through our Markov Chain model that they tend to view the same jobs they post) and a set of users that views jobs and applies for jobs, but there is little or no overlap between these two groups.

Another result of note was that *FriendRequestAccepted, CreateEndorsment, SearchJob* is a frequent triple. We found later that these actions are indicative of the Job Seeker cluster.

2.2.3 Markov Chains

We implemented a Markov Chain model to predict what actions a user is most likely to perform given a brief (one or two) action history. We used the "time-series" user representation to compute this model. We count the number of times each action x follows the single action y and the pair or actions y, z, i.e. we trained both a first and second order Markov model. We maintain a heap with the counts for each possible resulting action, and our model outputs the top k actions that could follow a given input. We also tracked whether the *couple id* and *object id* of the next action was the same as in the input. We did not do much useful prediction using this model, but we have it trained and ready to deploy, should the BranchOut team want to use it. Some interesting results we observed from the Markov Chains are summarized next.

We found that the most frequent followup to posting a job is to post another job. Second most frequent is to view a profile, and third most frequent is to view the job that was just posted. Viewing the job that was just posted is over five times as likely as viewing a different job, which is how we concluded in the previous section that there seems to be a job posting type of user that frequently posts jobs and then views the jobs that they post.

A somewhat surprising result was what happens right after a user applies for a job. The most frequent action after applying for a job is to view another job, but the second-to-most-frequent action is to apply again for the same job. It is possible that users frequently forget parts of their application and need to resubmit. We are unsure of the precise causality, but the result is interesting and perhaps useful to the BranchOut team.

2.3 Clustering

To verify our hypothesis of distinct user types, we performed clustering on the users in their action space representation, reweighted with tf-idf weighting with actions as terms and users as documents. As the size of this space is prohibitive for bottom-up clustering methods, we used k-means clustering on Hadoop. To generate initial seeds, we clustered a small sample set using Matlab's built-in hierarchical clustering function, starting with nine clusters, then allowed clusters with no data points to vanish. This yielded the five initial centers that were used for the rest of the clustering we performed.

Our k-means clustering was run with chained Hadoop jobs, with mappers finding the nearest cluster center for each point and reducers recomputing the cluster centers. Iterations of k-means were run until the sum of cosine distances between the two most recently generated sets of centers was less than 0.000001. This clustering was run on both the full set of actions and on the weekly slices of data in order to get a sense of how the clusters might have moved throughout the span of twenty weeks observed.

2.4 Classification

In order to attempt to classify users back into their clusters using limited data, we created a data set containing the user, their cluster, and up to the first twenty actions they took on the site. We then randomly sampled out 10% of these users, creating a 90% training set with a 10% held-out set for testing.

2.4.1 Naïve Bayes

In generating features from these first twenty actions, we tried several different methods. First and most importantly, we varied k, the number of first actions to store out of twenty. We also experimented with all combinaitons of decaying or growing the weight of later actions by a small exponential factor

(hereafter decay), considering n-grams of actions, and keeping track of action-position pairs. Once the weighted counts of the above features were generated, we experimented with three variants of Naïve Bayes classifier: basic Multinomial Naïve Bayes, Complement Naïve Bayes, and Weighted-Complement Naïve Bayes. Each model was trained by using the counts generated using the various parameters described above, and then run on the 10% held-out test set.

Naïve Bayes (MNB):

$$class_{predicted} = \text{argmax}_c \left[\log(\frac{N_c}{N}) + \sum_i f_i \log \frac{N_{ci} + \alpha_i}{N_c + \alpha} \right]$$

where N is the total number of users, N_c is the number of users in a class, and N_{ci} is the councount for the given action and this class.

Complement Naïve Bayes (CNB):

$$class_{predicted} = \text{argmax}_c \left[\log(\frac{N_c}{N}) - \sum_i f_i \log \frac{N_{\tilde{c}i} + \alpha_i}{N_{\tilde{c}} + \alpha} \right]$$

where $N_{\tilde{c}i}$ is the sum of N_{ki} for all classes except c, and $N_{\tilde{c}}$ is the sum of N_k for all classes except c. This algorithm avoids sample size bias by accounting for the size of other classes (the complement of the class in question).

Weighted Complement Naïve Bayes (WCNB) builds on CNB by taking the weight for each term,

$$\theta_{ci} = \frac{N_{\tilde{c}i} + \alpha_i}{N_{\tilde{c}} + \alpha}$$

and normalizing it.

$$class_{predicted} = \text{argmin}_c \left[\sum_i f_i \log \frac{\theta_{ci}}{\sum_k |\log \theta_{ck}|} \right]$$

Both CNB and WCNB are supposed to account for biases in the training data which the normal MNB algorithm may be subject to, due to uneven sample sizes, even after some downsampling of the monstrously overrepresented "bouncer" cluster.[1] However, these new algorithms are designed to account for bias in text classification - while our prediction setup is analogous, we experimented with all three methods to see which might give us the best performance.

2.4.2 SVMs

For comparison, we also implemented a linear Support Vector Machine model to classify users based on early actions. We used as features the first action, the count of each action type within the first k actions, and the set of pairs of actions occurring within the first k actions. We trained our SVM both on 10,000 random users and 100,000 random users, testing on the remaining 2,000,000 users.

[1] As described in Rennie, Shih, Teevan, Karger (2003): Tackling the Poor Assumptions of Naive Bayes Text Classifiers, http://www.aaai.org/Papers/ICML/2003/ICML03-081.pdf

3 Results

3.1 Clustering

3.1.1 Cluster interpretation

After twenty iterations of k-means, we identified five clusters and fit them to the narrative that we discussed with the BranchOut team. The clusters and top two associated actions are listed below. Each action is given with its strength in the cluster. An R subscript denotes that the action was received rather than actively performed.

0 Browser: 3,968,323 users - (ViewedProfile$_R$ 0.749), (ViewedProfile 0.625)

1 Job Seeker: 439,222 users - (CreateEndorsement 0.599), (JobViewed 0.517)

2 Networker: 333,282 users - (FriendRequestSent 0.510), (FriendRequestReceived$_R$ 0.510)

3 Job Creator: 17,928 users - (JobCreated 0.904), (JobViewed$_R$ 0.428)

4 "Bouncer": 12,237,322 users - (TotalConnections 0.998), (ViewedProfile$_R$ 0.033)

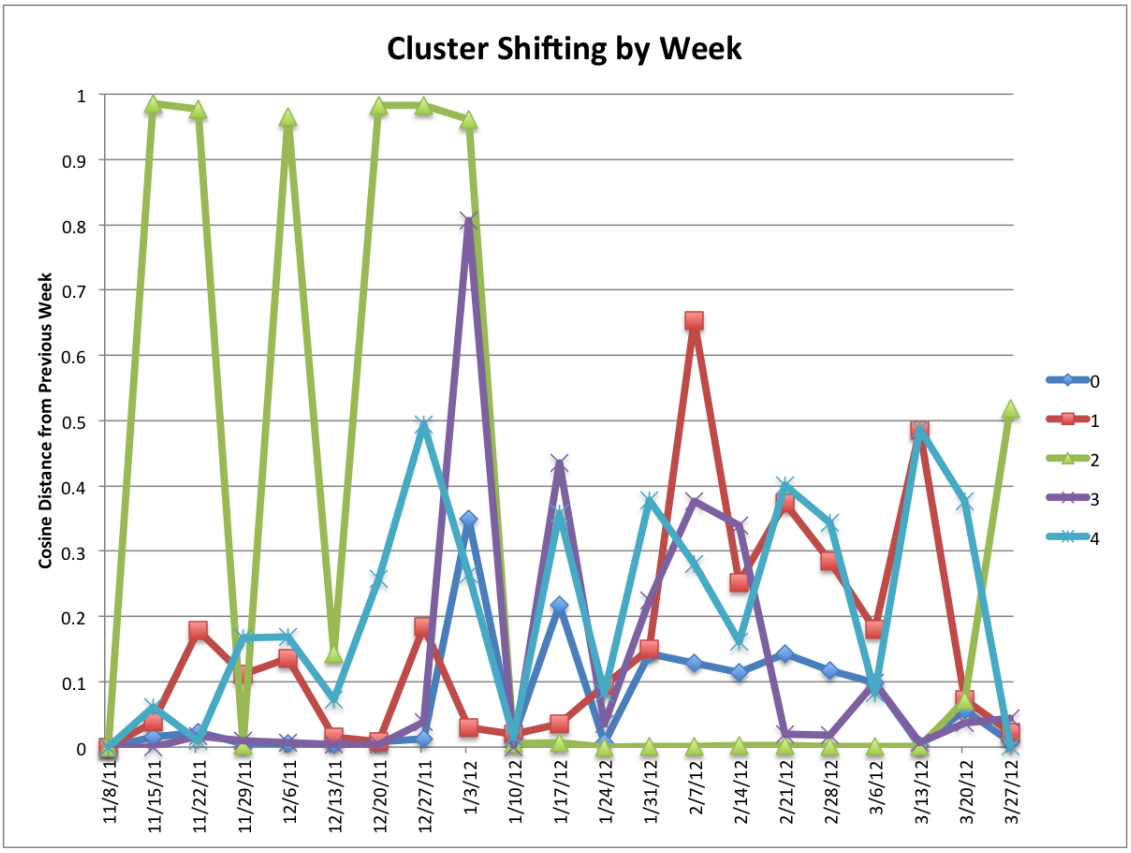


Figure 2: Cluster movement by week

	0	1	2	3	4
To nearest cluster	0.330	0.553	0.537	0.434	0.102
To next nearest cluster	0.767	0.880	0.798	0.830	0.889

Table 1: RMS distance to two nearest centers

3.1.2 Cluster stability

The clusters had fairly high variability through the weeks we observed, but this can be explained mostly by the influx of users in the last few weeks of the data. In these last few weeks, however, the clusters more closely matched the clusters over the full data set. We suspect that much of the noise is due to the changes in logging that began capturing more common actions at the tail end of the observed times. For example, from the weeks between January 17, 2012 to January 31, 2012, roughly the middle of our data collection period, about thirty actions were added. Among them was action 76, the most commonly performed action on the site.

3.1.3 Cluster coherence

For each user, we measured the (cosine) distance to the nearest two centers and found the average and root mean squared of these distances. As seen in Table 1, there is good separation, especially for the two largest clusters.

3.2 Classification

3.2.1 Naïve Bayes methods

We found generally that there were only minor differences between the Multinomial Naïve Bayes and the Complement Naïve Bayes, suggesting that the data was not too strongly biased from one cluster to another. MNB results for unigrams are shown in Table 2. Using bigrams of actions did not produce results that were substantially different, although the bigram tests that used unordered actions performed much better than their unigram counterparts, likely because the bigrams maintained some notion of ordering. For the WCNB model, for some reason we were not able to produce useful results, with abysmal accuracy below 10% with the exception of a few runs that managed to show about 60%. We triple-checked the algorithm, which has been shown to perform well on standard newsgroup data sets, and could not find any issues - for some reason, the WCNB model is not appropriate for our classification task.

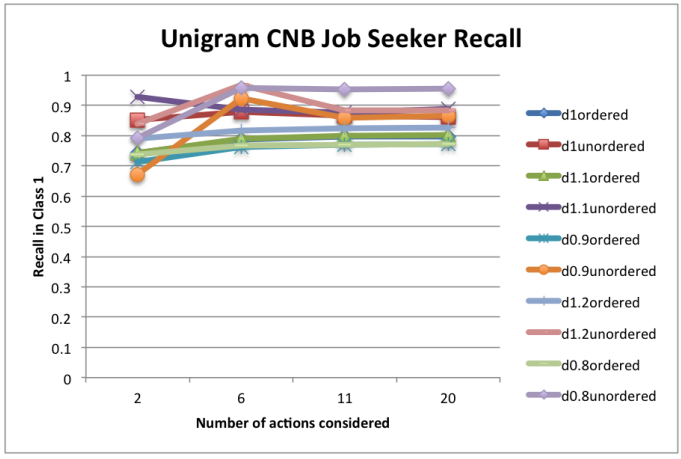


Figure 3: Job Seeker Recall

With regard to our other parameters, ordering seemed to have significant impact on our results - ordered test runs performed better overall. This, combined with the fact that unordered test runs performed best with small k, indicates that the first few actions are most significant in determining what cluster a user will belong to.

Table 3 shows the precision and recall for the test run from Table 2 that is highlighted as having he best overall accuracy. Precision numbers fare pretty well acros the board, and we get high recall for the Clusters 0-3. However, we have very poor recall for Class 4.

We believe this is actually a good thing, as Cluster 4 is our "Bouncer" cluster - users who don't have a general type, often because they don't perform many actions. Therefore, we think that the low recall in Cluster 4 indicates that members of that cluster are being classified into more meaningful categories, which can give BranchOut a better chance of engaging those users.

We considered that a company like BranchOut might also prefer to focus on the users who are likely to be job seekers (Cluster 1), as they are a resource which can help attract recruiters, and bolster the image that BranchOut is a good place to find a job. Therefore recall for Job Seekers would be the most important performance metric of a given model. We found that for this specific case, the CNB algorithm actually edges traditional MNB, as shown in Figure 3.

3.2.2 SVM methods

This is the effect of testing our SVM trained on 10,000 and 100,00 users, considering the first 20 actions:

	Cluster 0		C 1		C 2		C 3		C 4	
	Precision	Recall	P	R	P	R	P	R	P	R
10,000	0.892	0.957	0.878	0.851	0.829	0.857	0.925	0.712	0.747	0.379
100,000	0.895	0.954	0.879	0.848	0.801	0.886	0.842	0.863	0.756	0.378

We see that, at a certain point, more data does not significantly impact the model here, except for in Class 3 (Job Poster), where increasing the training data by tenfold causes a 15% jump in recall, which makes sense, since the Job Creator cluster is smaller than the others by an order of magnitude.

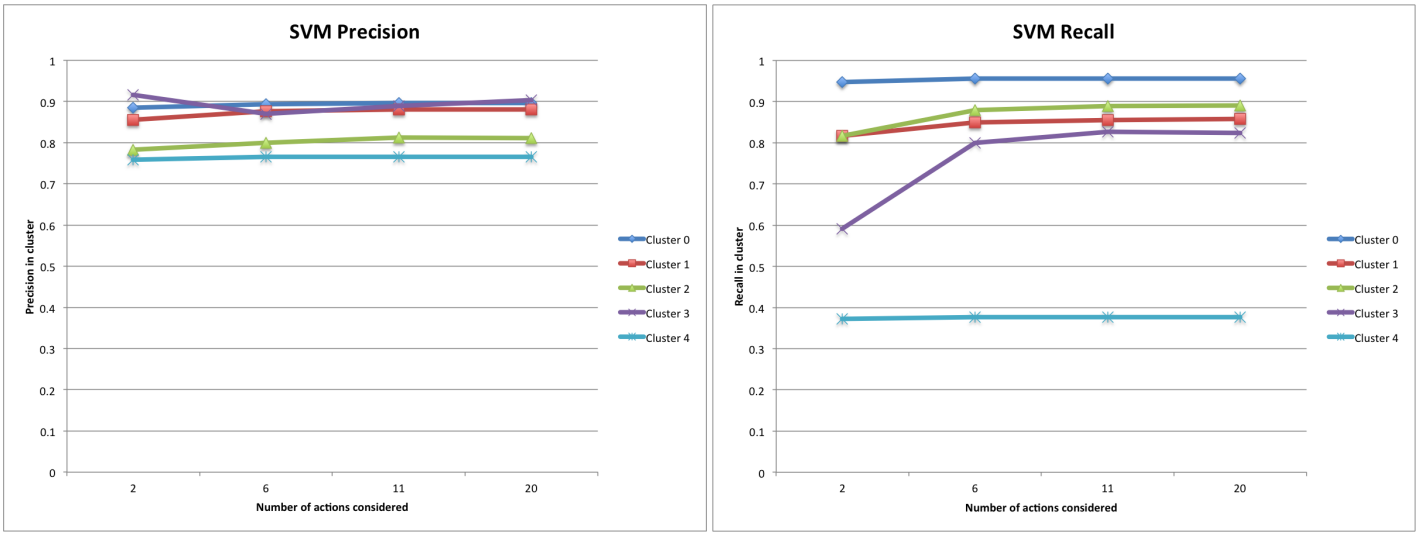
We then plotted precision and recall for the 100,000-user-trained SVM, considering the first 2, 6, 11, and 20 actions. The results are shown in Figure 4. We see that with an SVM, we enjoy both high precision and high recall in all clusters except for Cluster 4, the "Bouncer" cluster. We described in the previous section our hypothesis for low recall on this group, as well as why we do not find low recall on this group to be probelmatic.

	k=2	k=6	k=11	k=20
no decay unordered	0.830	0.718	0.742	0.735
no decay ordered	0.851	0.862	0.863	0.863
decay 1.1 unordered	0.794	0.783	0.794	0.796
decay 1.1 ordered	0.852	0.862	0.862	0.861
decay 1.2 unordered	0.849	0.796	0.796	0.832
decay 1.2 ordered	0.853	**0.864**	0.860	0.858
decay 0.9 unordered	0.818	0.667	0.795	0.796
decay 0.9 ordered	0.849	0.860	0.861	0.861
decay 0.8 unordered	0.846	0.719	0.795	0.792
decay 0.8 ordered	0.848	0.857	0.858	0.858

Table 2: Accuracy with MNB using Unigrams

	Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4
Precision	0.876	0.870	0.786	0.645	0.840
Recall	0.961	0.850	0.891	0.830	0.232

Table 3: Precision and Recall for MNB with Unigrams, Decay 1.2, ordered, k=6



(a) Precision of 100,000-user trained SVM considering variable number of first actions

(b) Recall of 100,000-user trained SVM considering variable number of first actions

Figure 4: SVM results

4 Conclusions

As was hypothesized, we found good evidence for clusters of distinctive user types within BranchOut. Not only are the discovered clusters cohesive, with relatively small in-cluster distance and comparably large distance to next largest cluster, but also the clusters are coherent, fitting nicely with both BranchOut's assumptions and our initial suspicions. We felt that this was sufficient evidence to move on with the assumption that these clusters were meaningful, albeit with some lingering concerns. The clustering is based only upon the action space representation of the users and there are biases from the extremely fast growth rate and changes in logging throughout the observed times.

Working with the assumptions that we made, we found that we could accurately classify users into the clusters. Even our baseline of Naive Bayes had high accuracy, and the SVM models granted us considerable improvement, especially in the smaller clusters, like the Job Creators.

Accuracy aside, the most promising result of our work was that the classifiers could work well with only a small number of initial actions seen. Ideally, BranchOut could use our work to take observations of the first few actions performed by a user and guess what content a user should be served next.