Predicting News Reader Preferences

Vinay Raj Hampapur, Helen Lu

***Index Terms*—machine learning, natural language processing, SVM, community detection, matrix completion, clustering,**

I. Introduction

WE We explore several techniques for predicting user reads on Pulse, a newsreader app for mobile devices. Accurately predicting reads could allow the application to personalize newsfeeds for each user by presenting stories most likely to be of interest.

Our approaches to building a classifier for predicting user reads include supervised learning with SVMs and several methods of clustering.

A. Dataset

Pulse provided a month of data for 1000 users. The data contains all stories published during August 2011 and all the user reads and clickthroughs in the same month. A *read* is defined as the user's first click on a story. Depending on the feed, this leads to a page showing anything from a short blurb to the full text of the story. The user then has the option for a *clickthrough*, which shows the original story in a browser.

The dataset contained the following information on each story, in a tab-delimited text file with one story per line:

- URL.
- Title.
- Feed name and feed URL.
- Timestamp of the story's first read.

The stories were keyed on (story URL, feed URL).

The files of user reads and clickthroughs contains of the above, plus the ID of the user doing the read or clickthrough. The timestamp refers to the date and time of the user's read.

II. Supervised Learning

In our supervised learning approach, we train on stories and reads in the first $i-1$ days, and use the model to predict for day i, where $i \in \{2 \dots 31\}$. Each user's prediction model is built only on the user's past reads, and does not incorporate other users' preferences.

The dataset contains about 210,000 stories in 946 feeds, but each user subscribes to a small number of feeds. Since the users have no way to read stories from other feeds, we remove those from consideration.

The simple separation of stories into 31 days is problematic, since users (a) do not open the Pulse app every day, (b) sometimes read stories from previous days, and (c) cannot see stories published between their last look at Pulse and the end of the day. We could simply not test on the days when a user did not open Pulse, but problems (b) and (c) still stand. Precision would be low, from including stories the user had no chance to read. Recall would also be low, from passing over older stories that the user does read.

Our solution defines day i as the period of time between the user's last read on day $i-1$ and the user's last read on day i. If the user does not read any stories on day $i-1$, the start time for day i reverts to the user's most recent read time on an earlier day, and day $i-1$ is skipped in testing. While this strategy might still omit the user's reads of even earlier stories, we do capture the stories that are "new" to the reader. In our results, recall is much better than precision, which suggests that including even earlier stories in the search space would hurt precision more than it would improve recall.

A. Features

The features for a given story are the tf-idf of the words in the title, the feed the story comes from, and time between the story's publication and the user's last read of the day.

Each row in matrix X represents a story available to the user, and each element in matrix Y represents whether the user read the corresponding story. Since each user has different feeds and different read times, X is unique to each user.

1) Story title: We expect that each user is interested in particular topics, and the title is the most easily accessible information about the story's topic. Also, the title (or the beginning of the title) is the only information visible to the user before the user clicks on the story. (Note: Perhaps we should truncate long titles for this reason, but titles are truncated inconsistently, depending on whether the feed contains pictures. Our dataset does not include information on the length of title visible, or whether pictures are included.)

We use tf-idf (term frequency - inverse document frequency) to weight the words depending on their importance within the title. Tf-idf is actually not very good at emphasizing the keywords in most sentences, since popular keywords appear in many titles. However, tf-idf is very easy to implement, and is useful for de-emphasizing words such as articles, pronouns, and common verbs.

We separated the titles into words and reduced the dimensionality of the data:

1) Delimit by space and hyphen.
2) Remove numbers and punctuation.
3) Change all letters to lowercase.
4) Apply Porter's stemmer.
5) Replace words with a total word count of 3 or less (across all titles) with the token UNK.

V. Hampapur is with the Department of Electrical Engineering, Stanford University, Stanford, CA, 94305 USA e-mail: vinayraj@stanford.edu.

H. Lu is with the Department of Computer Science, Stanford University, Stanford, CA, 94305 USA e-mail: helenlu@stanford.edu.

Report submitted December 16, 2011.

After the first 3 steps, there are over 82,000 unique words. Porter's algorithm and the elimination of rare words reduces the number of unique words to 18,029.

When computing tf-idf and eliminating rare words, we use the entire month of stories as the corpus, rather than restricting to earlier stories. This choice makes implementation simpler, and does not give an unrealistic performance result. In a real-world application, there would be sufficient past data to have a meaningfully large corpus.

2) Story feed: The next section of the X matrix consisted of 946 columns, each representing a feed. An element 1 in row i and column j indicates that story i came from feed j. Including feeds as a feature significantly improves performance, because users favor some of their feeds over others.

3) Story timing: The last column of X contains the length of time in minutes between the story's timestamp and the user's last read of the day. We concatenate multiple days of stories together when the user does not access Pulse for multiple consecutive days, so we also need a way to discount earlier stories. Pulse groups stories by feed, and orders stories within a feed chronologically, with most recent first. The number of displayed stories per feed is limited, so the user does not see old stories in prolific feeds.

We used time difference to favor newer stories because it was easier to implement correctly. Since the real determining factor of a story's visibility is the number of newer stories in the same feed, that would be a better feature.

Also, a user might access Pulse multiple times a day, and we use time difference to the last read. A more advanced implementation could instead calculate the time difference to the next read.

4) Other: We explored incorporating the clickthrough data by increasing the tf-idf of the words in the clicked-upon stories. The hope was to weight those stories more heavily, because the user showed greater interest in them. After testing on a small group of users, we saw little effect, so we did not include the clickthrough feature when running on the large dataset.

B. SVM learning algorithm

Our machine learning algorithm was the default SVM in liblinear (L2-regularized, L2-loss, dual problem). Logistic regression would also be suitable, but we found the SVM to be faster, and speed was a high priority. The test set contains 500 users and 31 days. To report results on every user and every day (after the first), we would need to train and test 15,000 models. Even after reducing the dimensionality of the data, the X matrix usually contains tens of thousands of rows and thousands of columns.

First, we dropped all-zero columns of X. These columns corresponded to words that did not appear in any stories in the user's feeds, and feeds that the user does not subscribe to.

We then scaled the elements to fall in the range [-1, 1]. Since the relative tf-idf values within one row are important, we scaled the entire block of tf-idf columns with the same value, and scaled the time column with another value.

There are generally far more negative examples than positive examples in the training set, so we include all the positive examples and a random selection of the negative examples. However, oftentimes there are so few negative examples that they do not provide a good representation of all negative examples. Therefore, we then test the model on the full (all negative examples included) training set, and add the higher-decision-value incorrectly-classified examples to the smaller training set. To keep the training set balanced, we duplicate an appropriate number of the positive training examples. We retrain on this more representative training set.

In a real-world application that extends over months and years, it becomes impractical to use all previous days as the training set. Furthermore, users' news preferences change over time. We compared using all training days with using up to 15 training days, but a more advanced system could learn the optimal number of training days for each user.

III. Clustering

The main idea that we pursue in this section is that if we can cluster either or both the stories and users into well defined groups, then we can leverage that information to make predictions.

A. Naive Clustering

Under this method, we group users who have accessed stories originating from the same base url into the same cluster of users.We build this set of clusters by using the data for a fixed number of days and then make predictions on the remaining days. This model allows for users to be grouped into multiple clusters. As the results will show, the key drawback in building this model stems from the over-simplifying assumptions made (hence, naive clustering) in building the clusters.

Consequently, we tried other techniques to build our clustering model.

B. Community detection

Our technique described here has been adapted from [1]. We begin by building our 'network' where each vertex represents a story and any two stories are "'connected"' if the sum of the overlapping tf-idf's of said stories is greater than a heuristically determined value. This determines the adjacency matrix A. Intuitively, we would expect a good demarcation of communities when the number of edges across different communities is significantly less than the average number of such edges for a randomly connected graph of similar dimensions. This metric is formalized by a quantity called the modularity. The modularity can either be positive or negative but positive values indicate the possible presence of community structures[1]. We model our network for all the stories, say n, for any one particular day. If we were to split the graph into only two communities, we denote $s_i = \pm 1$ if vertex i belongs to one community and $s_i = -1$ if vertex i belongs to a different community. Also, the expected number of edges between vertices i and j is $\frac{k_i k_j}{2m}$ where k_i denotes the degree of vertex i and m denotes the total number of edges in the graph.By noting that $\frac{1}{2}(s_i s_j + 1)$ is 1 if they are in the

same community and 0 otherwise, modularity can be written as:

$$Q = \frac{1}{4m}\sum_{i,j}\left(A_{ij} - \frac{k_i k_j}{2m}\right)(s_i s_j + 1) \quad (1)$$

$$\implies Q = \frac{1}{4m}\sum_{i,j}\left(A_{ij} - \frac{k_i k_j}{2m}\right)(s_i s_j) = \frac{1}{4m}\mathbf{s}^T\mathbf{B}\mathbf{s} \quad (2)$$

$$\mathbf{B}_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

Note that if we let u_i denote the normalized eigen vectors of $\mathbf{B}$, then $\mathbf{s} = \sum_{i=1}^{n}(\mathbf{s}^T u_i)u_i$ The goal now is to maximize the modularity by performing an eigen decomposition of the modularity matrix $\mathbf{B}$ and picking $\mathbf{s}$ which maximizes the modularity

$$Q = \frac{1}{4m}\sum_{i=1}^{n}\left((u_i^T\mathbf{s})^2\beta_i\right)$$

where β_iIt is to be noted that since s_i is constrained to be ± 1, in order to maximize the modularity. From the above equation it is clear that if we pick $s_i = 1$ when corresponding eigen component is positive and $s_i = -1$ when the corresponding eigen component is negative will result in the highest inner product. This is the core idea in the algorithm: we put all the vertices with $s_i = 1$ into one community and the remaining vertices into the other community.

In [1], the author provides additional details as to extend this idea to forming multiple communities; though, it should be quite believable that the above technique can be applied in a recursive fashion.For completeness, provided below are the set of equations for the change in modularity that (should) occur when breaking a group g of size n_g into 2

$$\delta Q = \frac{1}{2m}\left[\frac{1}{2}\sum_{i,j\in g}B_{ij}(s_i s_j + 1) - \sum_{i,j}B_{ij}\right]$$

$$\frac{1}{4m}\left[\frac{1}{2}\sum_{i,j\in g}B_{ij}s_i s_j - \sum_{i,j}B_{ij}\right]$$

IV. Matrix completion

In this technique, we explored other ways in which to make predictions on news reader preferences. Our motivation to begin this technique was to note that users, usually, have only a few common factors driving their tastes or preferences[2]. To this end, we model our matrix A to be completed in which the rows represent the users and columns represent the stories. An entry $A_{ij} = 1$ if user i has read story j and 0 otherwise.Of course, for this technique to at least have a chance at success, we must at least have one entry from each row and column. Then, the problem is reduced to finding a low-rank approximation to the matrix A since the data entries far outnumber the degrees of freedom of said matrix.

Initially, modeling our prediction problem seemed akin to the Netflix problem where users (rows of the data matrix) are given the opportunity to rate movies (columns of the data matrix), but users typically rate only very few movies so that there are very few scattered observed entries of this data matrix. A completed matrix of this sort would enable Netflix to recommend titles that any user might be willing to order. In order to test this idea, we built a sample matrix $A \in \mathcal{R}^{20\times 100}$ and picked only the eigen vectors corresponding to eigen values such that $|\lambda_i| >> 1$ to approximate this matrix, say G. We ended up with a rank 4 matrix corresponding to using the 4 largest eigen values. For all the entries in G, we set $G_{ij} = 0$ if $G_{ij} < 0.5$ and we set $G_{ij} = 1$ if $G_{ij} \geq 0.5$. It turned out that for the stories within the users' feed, our predictions were right roughly half the time. Also, G now invalidated some of the original data already used to build the matrix. Thus,a major draw-back in this technique is the requirement that all the stories be known ahead of time and that at least every user make his preference known for at least 1 story. In other words, it is impossible to make a prediction on a completely new brand of stories without first having at least some preferences for some users. Thus, we did not pursue this idea to the fullest extent possible.

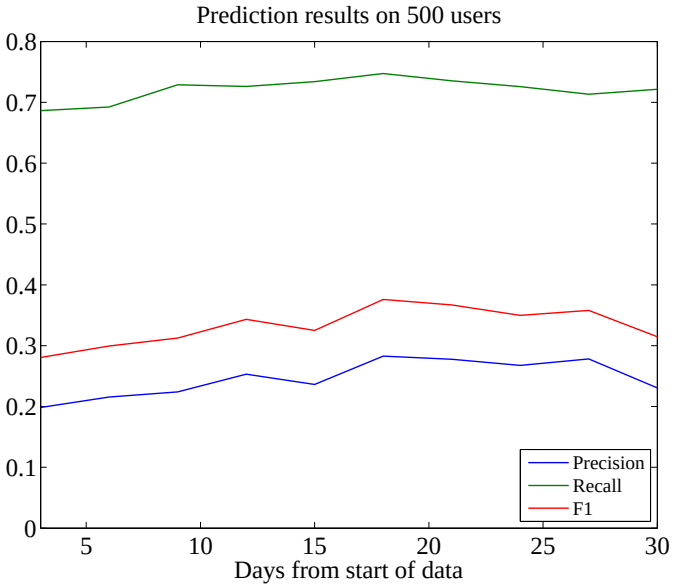

Fig. 1. Recall is consistently high, and precision is lower. We argue that recall is more important than precision in a news-prediction app, because the newsreader would like to present all the stories that are interesting to the user.

V. Results & Discussion

A. SVM

We tested over 500 users and every third day of the month. The average F-score is defined as the average of F-scores for each user.

When using all available training data, average F1 was 33.24%. When using the latest 15 days of training data, average F1 was 32.92%. This result indicates that eliminating older training data can be done without significantly hurting prediction accuracy.

The following plots(Fig.1 & Fig.2) are for the algorithm using all available data.

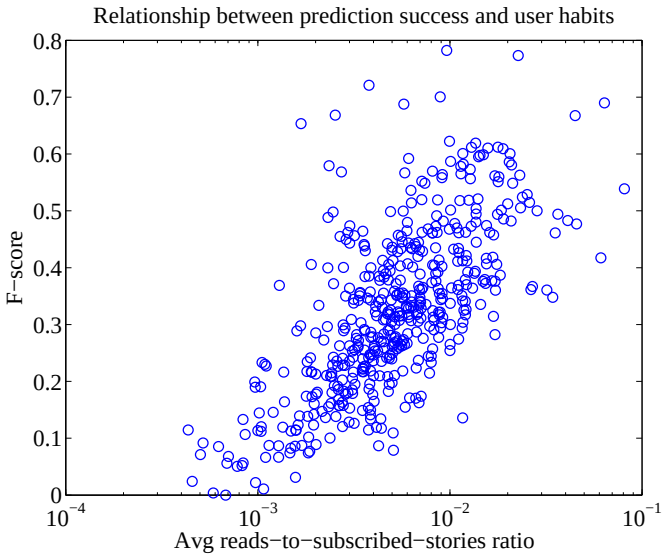

Fig. 2. Average F1 for a user is correlated with the proportion of stories the user reads. It is much harder to predict for a user who subscribes to many prolific feeds and yet reads few stories.

TABLE I
COMMUNITY STRUCTURES FOR STORIES ACROSS 3 DAYS.FROM THIS WE CAN INFER THAT IT POSSIBLE THAT A NATURAL DIVISION INTO COMMUNITY CLUSTERS EXISTS

Metrics	Day 1	Day 2	Day 3	Day 4
Avg. size	24	21.2	23.4	100
Max. members	46	36	37	135
# of communities	6	8	8	2

B. Community detection

We implemented this algorithm by placing a few constraints on the input to facilitate time, memory and processing constraints. Firstly, we capped the number of stories that went into the algorithm to 200. We then placed an edge between any two stories if they had at least 3 words in common and if their tf-idf was at least 1.2. The following table captures some of key metrics that resulted from this algorithm(first row gives us the average number of stories per community, the second row gives us the maximum number of stories any one community had and the last row gives us the number of communities into which the stories were divided into): Similar results follow for the remaining days. Ideally, we would have run this algorithm on the total set of stories and then, if any user accessed any particular story in one cluster, we would then predict that it would be very likely that the user would be interested in reading any of the remaining stories in the same cluster. In summary, community detection via maximization of spectral modularity is a solid technique for grouping stories into clusters.

A major draw-back with this technique though is the necessity to calculate eigen-vectors at each iteration of the process and consequently the inability to process more than 2000 connected stories at a time. Another draw-back of this process it the requirement of knowing all the stories before hand which cannot be the case for real-time news prediction.

VI. CONCLUSION

Our supervised learning approach yielded an average F-score of 33.24%, which seems low, but is actually reasonable given that less than 1% of stories are read. Furthermore, average recall is 72.44%, which means that our algorithm suggests most of the stories that the user wants to read. Perhaps many of our incorrect guesses would also be interesting to the user. We could test that hypothesis by looking at accessibility of incorrect guesses. If these stories mostly occurred a long time before the user opened the app, it could be that they were buried far back in the feed. Or, if the story was in a feed far down the user's list, perhaps the user did not have time to scroll down and see it.

The next step would be to merge the supervised learning and clustering algorithms. The clusters would be a valuable set of features to use in supervised learning. In addition to having a group of columns in X representing the feed of a given story, there would be another group of columns representing the cluster containing the story.

The clustering of stories via maximizing the spectral modularity turned out to be very effective in finding a network structure. Future work would be to make that algorithm work successfully in realizable run times for the entire dataset. Eventually, we would like to maintain two groups of clusters- one cluster for similar stories which would lend itself to predict a particular user's other story preference and another cluster for similar users wherein one user's reading of a story would lend itself to suggestion to other users in the same cluster.

ACKNOWLEDGMENT

The authors would like to thank Richard Socher for high-level guidance.We would also like to thank the other Pulse project groups for sharing experiences and tips.

REFERENCES

[1] M. E. J. Newman, *Modularity and community structure in networks*. National Academy of Sciences 2006
[2] E. J. Candes and B. Recht*Exact Matrix Completion via Convex Optimization*. Applied and Computational Mathematics, Caltech, Pasadena 2008
[3]
- Liblinear. http://www.csie.ntu.edu.tw/~cjlin/liblinear/
- Porter Stemming Algorithm. http://tartarus.org/martin/PorterStemmer/