Creating a better above-the-fold experience: Predicting news article preferences

Anil Das
Walter Mostowy

December 16, 2011

1 Background

News-article preferences provide a ripe area for learning. Readers have differing, and sometimes complicated, preferences. While it is reasonable to assume that a reader's indicated past history is correlated with future preferences, preferences are likely somewhat variable depending on, e.g., the time of day. Editorial responsibilities aside, a good algorithm would save users time by directing them to exactly what they want to read. Or perhaps an algorithm more tuned to editorial oversight could "pop the filter bubbles" by gently introducing readers to different, edifying topics.

2 Problem Domain

We would like to predict users' interest in arbitrary news articles. More specifically, presented with a particular article, we would like to predict whether a particular user will interact with—that is, read or click through to—that article. Metadata available to us include the articles' headlines and RSS source feeds. We also consider the timestamps at which the user interacts with the articles. In this paper, we describe a learning model used to make useful predictions given these metadata.

3 Method

We were provided with a full month's (August 2011) worth of data on 1000 users from the "Pulse" app for mobile devices[1], which presents the user with news stories he might be interested in. We trained and experimented on 500 of these users. We discuss the resulting scores on the remaining 500 users in the results section.

3.1 Preprocessing

After cleaning the data by removing corrupt records, we converted each story title to a vector of word-stem counts. Stemming was done using the Porter stemming algorithm[2], though we discarded stems with fewer than three characters. We also converted each stem-count vector to a tf-idf vector[3] using the tf-idf formula,

$$\text{tf}\,\text{idf} = \text{tf}\,\log\frac{|D|}{\text{idf}}$$

where $|D|$ is the total number of titles, idf is the number of titles in which the stem occurs, and tf is the number of times the stem occurs in the title in question.

3.2 Baseline

As a preliminary baseline for purposes of comparison, we constructed logistic regression models and support vector machines against our processed design and target matrices. If one were to naively measure the predictive power of these models without looking at the confusion matrices, one might conclude that their achieved 99% accuracy rates were very good. However, one must consider the fact that no user can ever interact with more than a very small fraction of possible stories. A trivial "model" which always guesses in the negative would do just as well! Therefore, in order to more meaningfully measure the success rates of our models, we used the F_1 score calculated from the confusion matrix:

$$F_1 = 2\left(\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}\right)$$

Our preliminary F_1 scores were measured to be very low, averaging about 0.05.

There were several reasons for this low score. Perhaps the most obvious reason is that the the number of negative instances in the target vector vastly exceeds the number of positive ones. Because the data is necessarily noisy, the "success" of correctly classifying many negative examples overwhelmed the noisy successes of correctly classifying the positive ones, resulting in a poor hypothesis fit. In addition, we only included stems from story titles and ignored other important signals. The remainder of this paper describes how we addressed these issues.

3.3 Algorithm and Model Selection

We tested logistic regression, linear SVMs using the liblinear library, and SVMs with a kernel of $K(u, v) = (\gamma u^T v + r)^d$ using the libsvm library. We implemented one using the limited-memory BFGS algorithm in the minFunc[4] optimization library. We empirically found the degree $d = 2$ to be optimal for libsvm models. However, we achieved our greatest F_1 scores using our logistic regression model.

Because we saw high variance when inputing normalized tf-idf features into logistic regression, we searched for a regularization parameter to decrease variance. To this end, we used the regularized logistic cost function

$$C(\theta) = \lambda||\theta|| - \sum \left(y \log g(\theta^T x) + (1 - y) \log(1 - g(\theta^T x))\right)$$

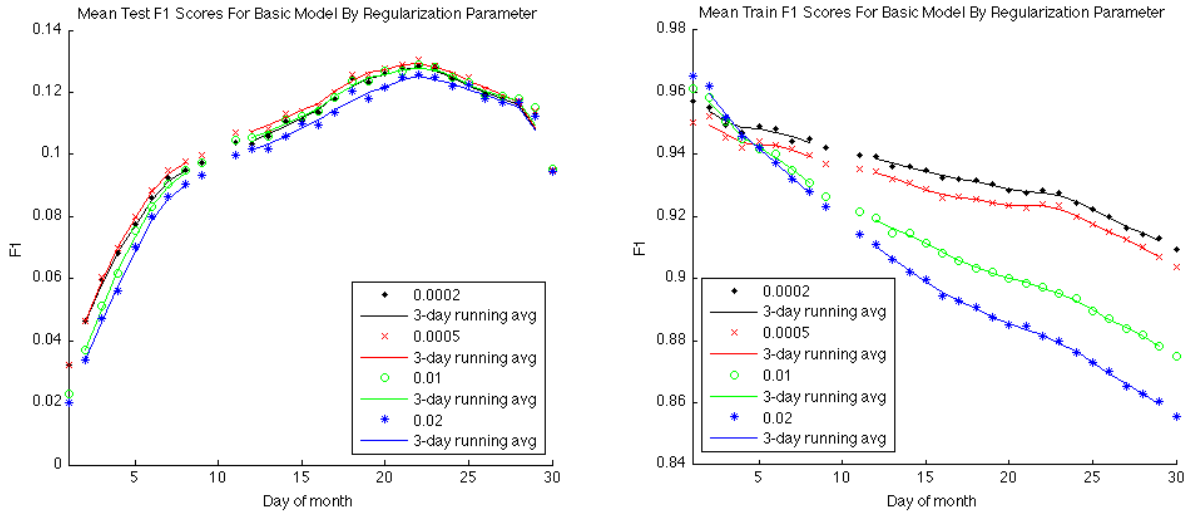
where λ is the regularization parameter, g is the sigmoid function, and y is an element of the target bit-vector. We empirically found $\lambda = 0.0005$ to be the best model.

However, the training and testing errors remained quite high due to the aforementioned overabundance of negative examples in our data. In order to alleviate this, we adopted an algorithm to "split" the negative training data into disjunct subsets. Iteratively, a logistic regression model would be trained against all of the positive examples and one subset of negative examples of about the same size, passing the resulting parameter θ as a guess for the next iteration. Once the algorithm iterates through all the negative examples in this way, a final learning phase would train on all the positive examples and an equal number of negative examples which were the most likely to be misclassified during iterated learning phases. The hypothesis obtained from this final phase would then be used for making the desired predictions.

Using this iterative logistic regression, we were able to fit a very good separating plane between the entire set of positive and negative training examples—reaching a mean F_1 above 0.9 (see Figure 1b). Testing F_1 scores improved, as well (see Figure 1a).

3.3.1 Stem vector representation

We also considered presenting the features as stem counts, tf-idf or normed tf-idf to our algorithms and models. We found the normalized tf-idf design matrices to be the form producing the best results.



(a) Test scores

(b) Train scores

Figure 1: Iterative logistic regression

3.3.2 Cross validation

We conducted 30-fold cross validation by training on the first d days and testing on the remainder. For this purpose, the preprocessed data were split into separate matrices, one for each day, so that for an arbitrary day d, we could more easily train on the first 1–d days and test on the remaining days. We also set aside all the data for a group of 500 users for final validation.

3.4 Feature Selection

3.4.1 Feeds

We found the RSS source feed for a story to be an important signal. Taken over the aggregate, we could approximate the set of feeds to which a user subscribed. This was useful to us because users are only presented articles from the feeds to which they are subscribed. We used this feed information to mark stories that would not be available to a user as negative (thus bypassing the logistic learning model). We then conducted training and testing only on the remaining stories.

We also added the feeds from which a story was available as a bit-vector feature to the design-matrix rows.

3.4.2 Time of Day

We conducted an error analysis on a selection of the users that were giving low prediction scores. From this analysis, we found that users had different temporal patterns of activity, and in particular, users tended to be wholly inactive for large parts of the day. Further, these patterns were distinct from the aggregate temporal pattern of activity. To take advantage of this data, we added as features to the design matrix the hours of the day at which users interacted with a story.

With these feed and hour data added (see Figure 2), our mean F_1 improved to 0.23.

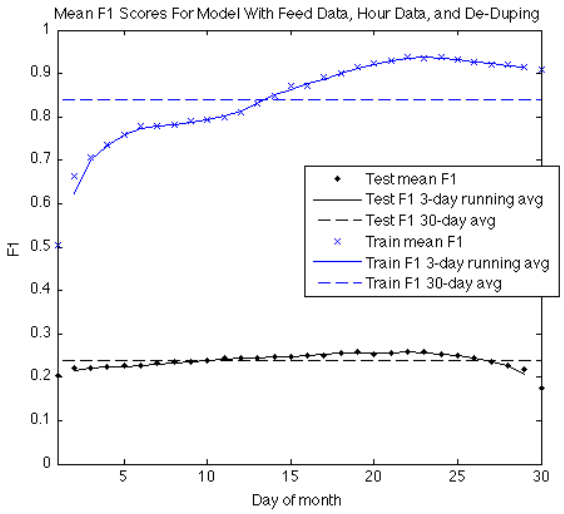


Figure 2: Scores for the model with feed data and hour data

3.4.3 Clustering of Users

We hypothesized that users can usefully be grouped into clusters of similar interest. If true, whether a peer group already interacted with a story would be an important signal. To convert this idea into a feature, we took the set of stories that the user read in the training set as a bit vector, and using Euclidean distance measurements, we clustered the users using K-means algorithm into 20 clusters. Then for each story in the design matrix, a feature was added showing what fraction of the people in a user's cluster interacted with the story on previous days. Data from the current day was not included, because such data might include data about the user himself, i.e. the event we are trying to predict.

Using this approach, our mean F_1 scores improved to 0.3, and training scores remained high at 0.9.

3.4.4 Feature Culling with Mutual Information

In order to reduce variance, we culled our word-stem vectors using mutual information measurements. Figure 3 shows, for 10 randomly selected users, the mutual information for all features, where the features have been ranked by mutual information.

One can see that the mutual information measurement rises from the baseline only for a very few features, typically less than 100. We used this fact to select only features that showed a high mutual information value for some user. This reduced our feature vector length from approximately 60,000 to around 1,500. However, the resulting improvement to mean F_1 scores was minimal, about 0.01.

4 Results

For each of the 30 days (excluding day 31) of data, we trained and tested a model for each of the 1000 users. The training design matrix consisted of all of the data for all days up to the current day. The resulting model was then tested against the data for the remaining days of the month. This resulted in a mean F_1 score of about 0.3. The F_1 scores for the holdout set of 500 users were very close to that of the initial set of 500 on which we developed our model.

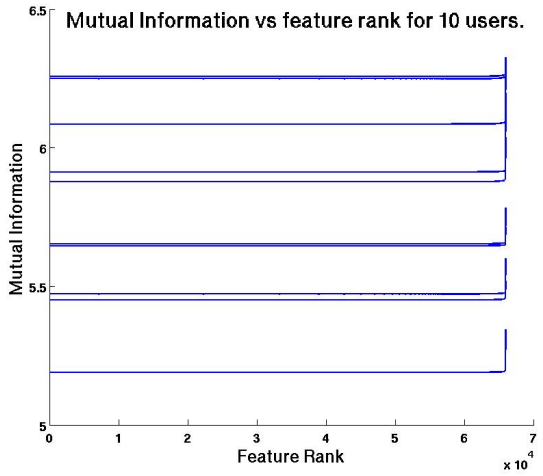


Figure 3: Mutual information

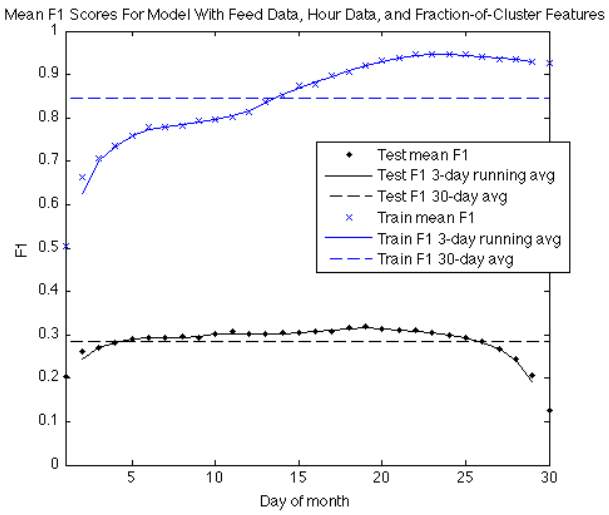


Figure 4: The scores resulting from our final learning model and algorithm

5 References

[1] http://pulse.me/
[2] http://tartarus.org/martin/PorterStemmer/
[3] Sparck Jones, Karen. "A statistical interpretation of term specificity and its application in retrieval" (1972).
[4] http//www.di.ens.fr/∼mschmidt/Software/minFunc.html