RECOMMENDATION SYSTEM FOR GOOGLE READER

NADAV SAMET

1. Introduction

Google Reader is a web-based content aggregator. Users can subscribe to content sources that interest them, such as blogs and news sites, and read the content in one central place. Users can also interact with the content in many ways: explicitly indicate they liked it, share with friends, post a comment, visit the source website or follow a link embedded in the content.

In this report, we explore two orthogonal recommendation problems that arise naturally from this setting. In the first problem we attempt to predict, given a content source and an unpublished news item in it, whether subscribers of that source will like the item. Using this information, Google Reader can advise content authors what kind of posts their readers enjoy.

The second problem we explore is interesting items discovery. We would like to provide users with personal recommendations of items they might like from streams they are not subscribed to.

2. Data model and terminology

We denote by $\mathcal{S}$ the set of all news streams users are subscribed to. Let $\mathcal{I} = \{I^{(1)}, I^{(2)}, \ldots, I^{(m)}\}$ denote the set of all the content items of the streams in $\mathcal{S}$. For each content item $I^{(k)}$ we store (in a Bigtable [1]) its title, body, author name, time published, and a back-reference to the stream $S \in \mathcal{S}$ the item originated from.

Each user $U \in \mathcal{U}$ has a set $S_U \subset \mathcal{S}$ of streams he is subscribed. For simplicity, in this report, we do not distinguish between the various ways a user may interact with an item. When a user performs any interaction with an item (clicking, sharing, emailing) we say that a user $U \in \mathcal{U}$ liked an item $I \in \mathcal{I}$. In our system, the collection of items continuously grows as new content is being crawled. From item recommendation perspective, it is desirable to prefer recommending recent items.

3. Predicting item popularity

3.1. **Problem statement.** For a fixed stream $S \in \mathcal{S}$, presented with items $\mathcal{I} = \{I^{(1)}, \ldots, I^{(m)}\}$ of S, and $y^{(1)}, \ldots, y^{(m)}$ indicating the number of users who liked the corresponding item, predict whether an unpublished item I will be popular if published on $\mathcal{S}$.

To pose this problem as a classification problem, we consider an item popular if the number of users who liked it is significantly more than average. Formally, if μ and σ are the mean and standard deviation of the $y^{(i)}$'s, then an item is $k-$popular if more than $\mu + k\sigma$ users liked it, for some parameter k.

Date: December 10, 2009.

3.2. **Naive Bayes approach.** I have experimented mainly with the popular blog LifeHacker (http://www.lifehacker.com). The data set included 6334 items which were randomly split in the ratio 90% : 10% train to test. To reduce the number of features, each word has been stemmed and words that appeared in less than 0.3% of the documents have been omitted. On this input, the Multivariate Naive Bayes algorithm has been applied.

3.3. **Results.** As k increases, less items are considered popular and hence the algorithm may do better by merely guessing that an item is not popular, therefore it is useful to consider the false positives and the false negatives separately. Setting $k = 2$ has been observed as an optimal point where the number of likes is significantly larger than average, hence the separation is meaningful, and still there were 140 popular items in the training set out of 5683 items.

The test set had 19 popular items and 632 non-popular items. Naive Bayes correctly classified 6 out of the 19 popular items and 580 of the non-popular items.

3.4. **Conclusion.** The success rate of the Naive Bayes approach for this problem is not satisfying as it missed 13 out of 19 of the popular items and yielded 8% of false positives. The terms that appeared to have high correlation with popularity appeared fairly random: mediawiki, wordpress, gizmo, bot, logon. Terms that have been highly correlated to non-popular items were: $0, bookmarket, coupon, opera, sale and tree.

Looking closer at the data, it appears that it is difficult a problem to predict item popularity based on intrinsic features. Very often two items that use similar words will vary a lot by popularity, for instance when one item announces a hot new product when it has just been released, while the other one reviews it few weeks after.

Perhaps this problem could be compared to the well-studied problem of predicting whether a song will become a hit. [5] suggests that algorithms based on intrinsic features (acoustic and lyrics, for instance) only do slightly better than a random classifier. In [7], the authors claim that the popularity of a song is not predictable beyond random using state-of-the-art machine learning algorithms.

4. Item recommendation

4.1. **Problem Statement.** Presented with a set of documents $\mathcal{I} = \{I^{(1)}, \ldots, I^{(m)}\}$, a set of users $\mathcal{U} = \{U^{(1)}, \ldots, U^{(n)}\}$, their subscriptions and liked items, and given a specific user U, recommend K items that the user might be interested in reading.

Given the scale of operation at Google, the algorithm has to be able to provide recommendations in an instant for any user. The recommendations have to be recalculated several times a day to keep the users interested.

4.2. **Our approach.** Given the results of the previous section, we would like the recommendation algorithm to be content agnostic and rely only on social signals. In addition, many popular items contain very few words or just embed an image or a video and therefore it is difficult to extract the right features from the content.

Our approach is to restrict the search space for recommended items to streams that are similar to the streams the user is currently subscribed to. The intuition behind this is that in most cases, users subscribe to a stream after they have read a part of its content via other means (on the content's source website) and their subscription is a 'vote' for their interest in the topic of that stream. Therefore, it

makes sense to examine similar streams which the user is not subscribed to and look in them for items the user may enjoy. The simplest way to find interesting content in this new set of streams is to take only k-popular items from each of those streams. Although this does not take into account the user's own preferences at the item level, this approach gives surprisingly good results.

4.3. **Finding similar streams using MinHash.** For a stream $S \in \mathcal{S}$, let $U_S \subset U$ denote the set of users subscribed to this streams. Given two streams, S_1 and S_2, we use the Jaccard coefficient $s(S_1, S_2)$ to measure their similarity:

$$s(S_1, S_2) = \frac{|U_{S_1} \cap U_{S_2}|}{|U_{S_1} \cup U_{S_2}|}$$

The intuition for this choice is that if two streams have a large overlap in their subscribers, they are likely to be related. For a fixed stream S we would like to find the streams that are most similar to it. Due to the large number of stream pairs, a clustering algorithm such as k-means is impractical and instead we use a probabilistic clustering approach called MinHash [6].

The algorithm starts by randomly choosing $p \in \mathbb{N}$ permutations of the set of all users $\mathcal{U}$. Let $f_i : \{1, \ldots, |\mathcal{U}|\} \to \mathcal{U}$ denote the i-th chosen permutation. For each $i \in \{1, \ldots, p\}$, we define the hash function $h_i : \mathcal{S} \to \mathbb{N}$ by $h_i(S) = \min_k\{f_i(k) \in U_s\}$. It is a well-known fact[2] that the probability that two streams are mapped by h_i to the same value equals their Jaccard similarity. For completeness, we include here its proof.

Claim. $\Pr(h_i(S_1) = h_i(S_2)) = s(S_1, S_2)$.

Proof.

$$\Pr(h_i(S_1) = h_i(S_2)) \quad = \sum_{u \in U_{S_1} \cap U_{S_2}} \Pr(f_i(h_i(S_1)) = u \text{ and } f_i(h_i(S_2) = u)$$

The event $(f_i(h_i(S_1)) = u$ and $f_i(h_i(S_2) = u)$ occurs only when the permutation f_i assigns u to a position left of all elements in $U_1 \cup U_2$. Since each element u has an equal chance to be the leftmost element in the permutation restricted to $U_1 \cup U_2$, the probability in the right hand side is $\frac{1}{|U_1 \cup U_2|}$. Since this quantity is independent of u the claim follows. □

Using the hash functions h_i, we map each stream S to a cluster $(h_1(S), \ldots, h_p(S)) \in \mathbb{N}^p$. The probability that two streams end up in the same cluster is $s(S_1, S_2)^p$. Increasing the value of p creates more clusters where the expected average similarity in each cluster is higher.

In practice, instead of choosing random permutations f_i of $\mathcal{U}$ over millions of users and implementing h_i, we only choose p random 64-bit unsigned integers r_i to be used as a seed and let $h_i(s) = \min_{u \in U_S} \text{Hash}(r_i, u)$, where Hash is a fixed hash function that returns a 64-bit unsigned integer. We think of $\text{Hash}(r_i, u)$ as the position of u in the i-th permutation.

To alleviate the problem that the clusters become smaller as we increase p (resulting in less recommendation opportunities), we repeat this process q times. The end result is that each stream belongs to q clusters, where each cluster is uniquely identified by p numbers.

As noted in [3] this construction is easy to implement efficiently in the MapReduce [4] model, where a cluster of machines processes the dataset in parallel. In

the map phase, each worker is iterating over a subset of the streams and for each steams it outputs q key-value pairs where each key is a cluster id (p numbers) and the value is a stream identifier. The data is then partitioned by the value of the key, and each partition is then sorted by the key. In the reduce phase, each worker scans through the key-value pairs ordered by key (which is the cluster id) and writes a mapping from a cluster id to the set of streams that it contains. The mapping is stored in Bigtable for efficient look-ups later. For this project, I experimented with $p \in \{2, 3\}$ and $q = 15$.

4.4. **Membership table.** For each user $u \in \mathcal{U}$, in order to find clusters of interest, we would like to be able to find the set of all clusters that contain a stream that the user is subscribed to. Fortunately, this can also be implemented efficiently in a MapReduce which is based on the output of the previous MapReduce (and the user's subscriptions data). This MapReduce outputs a mapping that maps each user to a list of (cluster, score) pairs where the score is the number of streams in the cluster the user is subscribed to divided by the size of the cluster. High score indicates that the user is subscribed to several streams from this cluster, and therefore the cluster represents a topic the user is likely to care about.

4.5. **Generating recommendations.** In order to generate recommendations for a user u, we obtain the set of clusters the user is a member of, along with their scores, from the membership table. Next, for each cluster we issue look-up requests to the cluster table to obtain the list of streams in this cluster. These requests occur in parallel. Finally, we issue a request to a Google Reader's backend server providing it with the set of streams we obtained, and get in return a ranked list of content items sorted by popularity originating from those streams (this per-stream data is precalculated).

4.6. **Measuring recommendations quality.** As this system is not currently available to the general public, I was only able to get feedback from a limited number of users. In general, it appears that most of the recommendations are very targeted at the user (topics are very close to topics of interest). At a second iteration I was able to improve the overall quality by pruning streams with less than 100 subscribers from the clustering algorithm. This resulted in a higher concentration of extremely popular items in the recommendation mix.

A strong indication that a user liked a recommendation is whether he interacted with the recommended item. The method I would use to get a quantitative evaluation of this recommendation system is to compare it to different algorithms. Whenever recommendations are requested for a user, a recommendation algorithm can be chosen at random, and we can keep track whether the recommended items were interacted with.

5. Conclusions

In the first part of this project I experimented with content-aware approaches in an attempt to predict the popularity of unpublished items. Having obtained poor classification results, I manually examined the data and concluded that popular items do not necessarily have different word distribution than less popular items. The reason for their popularity seem to be originating from new ideas or news conveyed on the same general topic of the stream, and hence the keywords are

indistinguishable from non-popular posts. That is to say, it is not the words; it is what they communicate.

The second part of the project was more successful and some of the recommendations it generated were actually useful. Although the recommendation algorithm is simple and is not based on a probabilistic model, the results look promising.

References

[1] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A Distributed Storage System for Structured Data. In *Proc. of the 7th Symposium on Operating System Design and Implementation,* (OSDI 2006).

[2] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. Ullman, and C. Yang. Finding Interesting Associations without Support Pruning. In *Proc. of the 16th Intl. Conf. on Data Engineering*, (ICDE 2000).

[3] Abhinandan S. Das, Mayur Datar, Ashutosh Garg and Shyam Rajaram. Google news personalization: Scalable online collaborative filtering. In *WWW '07: Proceedings of the 16th international conference on World Wide Web* (2007), pp. 271-280.

[4] J. Dean, and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters., In *Proc. of 6th Symposium on Operating Systems Design and Implementation (OSDI)*, San Francisco, 2004.

[5] Dhanaraj, R., Logan, B.: Automatic prediction of hit songs. In *6th International Conference on Music Information Retrieval* (ISMIR 2005), pp. 488–491 (2005)

[6] P. Indyk and R. Motwani. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. In: *Proc. of the 30th Annual ACM Symposium on Theory of Computing*, 1998, pp. 604–613.

[7] Pachet, F., Roy, P. Hit song science is not yet a science. In *9th International Conference on Music Information Retrieval* (ISMIR 2008) (2008)

E-mail address: `nadavs@google.com`
Current address: 1600 Amphitheatre Parkway, Mountain View, California 94043