Large-Scale Query Understanding

Khaled S. Refaat*
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095
krefaat@cs.ucla.edu

Sugato Basu
Google Research
Mountain View, CA 94043
sugato@google.com

Deirdre O'Brien
Google
Mountain View, CA 94043
deirdre@google.com

Liadan O'Callaghan
Google
Mountain View, CA 94043
liadan@google.com

Abstract

In this paper, we propose a large-scale multi-dimensional co-clustering framework for understanding queries in a search engine. To achieve this goal, the system simultaneously clusters queries along with attributes of results that were shown (and clicked) on these queries. In our application, we co-cluster queries along with advertisements (commercial results), advertisement keywords, and query terms — this gives us the ability to look at concepts that correspond to groups of queries and ads, as well as do better noise filtering in query clustering [1]. After getting query clusters, we identify representative queries for each cluster, in an attempt to explain what concept underlies a cluster. Our system extends the co-clustering MapReduce framework [2] to perform multi-dimensional co-clustering at scale.

1 Introduction

The goal of this project is to better understand queries. For example, having the query "celebrating christmas", we would like to consider what concept the user issuing the query might be thinking about — taking a Christmas vacation, Santa Claus, or buying a Christmas tree. Having a query like "boa", we would like to know that this could refer to the concept related to Bank of America or the heavy-bodied snake known as Boa constrictor. To achieve this goal, we would like to cluster queries such that each group of related queries are to be identified as a cluster. After that, representative queries (called *landmarks*) are discovered for each cluster and used to explain what the queries in a cluster are about. We next give the motivation for clustering queries using a method called co-clustering.

Suppose we have the two queries "flowers" and "send flowers to a friend". Since they have a common term "flowers", one can easily detect that they are similar and put them in the same cluster. On the other hand, the two queries "flowers" and "red roses" share no common terms, but suppose they were associated with the same advertisement (ad) with the URL www.suncoastflowers.com. In this case, we could still detect they are similar and assign them to the same cluster. This suggests using ads and terms as features for clustering queries. But what if "flowers" and "red roses" were associated with different but very similar ads: www.suncoastflowers.com and www.flowershopping.com? In this case, we still want to put them together in the same cluster. Therefore, we need to first determine that the two ads are similar and, accordingly, detect that the queries are also similar. This calls for clustering ads and terms simultaneously while clustering queries — which leads us to consider

*This work was done while the author was visiting Google Research.

co-clustering. The information obtained from clustering ads and terms is used in clustering queries, and vice versa.

After grouping related queries into clusters, we need to get a stable representation for each cluster that reveals what the cluster is about. Therefore, we discover the landmarks of a cluster by selecting queries that are close to the cluster centroid but far away from other centroids.

In this paper, we propose a framework for large-scale multi-dimensional co-clustering using the MapReduce model, generalizing the framework proposed in [2], initially proposed for binary relations. Every mapper takes a portion of the data, assigns a cluster label to every entity according to the closest centroid, and computes partial statistics. The reducers aggregate partial statistics related to every cluster to compute *cluster prototypes*, which act like cluster centroids. The clustering MapReduce is run iteratively, using the disk space for communication between consecutive iterations (which we will discuss in detail in following sections).

After clustering, another MapReduce is used to discover landmarks, where a mapper evaluates individual queries to see if they are suitable as landmarks, whereas a reducer uses this evaluation to discover the landmarks of a cluster.

The paper is organized as follows. In Section 2, we describe our notation. In Section 3, we propose a large-scale framework for multi-dimensional co-clustering. Next, Section 4 explains how landmarks are discovered in our system. We highlight some of our preliminary experimental results in Section 5. We review the related work in Section 6. The paper ends with a conclusion and a discussion of future work.

2 Notation

A random variable is denoted by an upper case letter X, whereas its value is denoted by a lower case letter x. The probability function of a random variable X is denoted by $p(x)$. A mapper's partial computation of $p(x)$ is denoted by $p'(x)$ — this means that $p'(x)$ is not the correct probability but is a partial quantity that will be used to compute $p(x)$, by aggregation. A cluster variable will be denoted by $\hat{X}$ and its value by $\hat{x}$.

3 Large-Scale Co-clustering

The reader is referred to [1] for a review on the sequential version of the used co-clustering algorithm. Suppose we are in the phase of clustering entities x using their relations with the so-far clustered entities y. For simplicity, suppose that the mapper takes one entity x_i as an input. It will then assign x_i to cluster $\hat{x}_i$ and will output the following statistics keyed by $\hat{x}_i$. Firstly, it outputs a sparse vector of $[p'(\hat{x}_i, \hat{y}_1)...p'(\hat{x}_i, \hat{y}_n)]$, where p' denotes the probability in the limited world seen by the mapper. Secondly, it outputs the summation $p'(\hat{x}_i) = p'(\hat{x}_i, \hat{y}_1) + ... + p'(\hat{x}_i, \hat{y}_n)$. As we will see, these statistics when aggregated will suffice to recover the correct cluster prototype for $\hat{x}_i$, assuming we have $p(Y|\hat{Y})$ computed. However, this phase should also compute $p(X|\hat{X})$ so that the next co-clustering phases can use it. Thus, the mapper finally outputs a vector of dynamic size: $[p(\hat{x}_i, x_i)]$, with the summation of this vector[1].

The reducers aggregate the output of the mappers by their key. For example, aggregating the outputs keyed by $\hat{x}_i$ will involve computing the summation of the vectors $[p'(\hat{x}_i, \hat{y}_1)...p'(\hat{x}_i, \hat{y}_n)]$ and the sums $p'(\hat{x}_i)$, to get $[p(\hat{x}_i, \hat{y}_1)...p(\hat{x}_i, \hat{y}_n)]$ and $p(\hat{x}_i)$. We do a further step and normalize the vector by its sum to get $[p(\hat{y}_1|\hat{x}_i)...p(\hat{y}_n|\hat{x}_i)]$. Aggregating the dynamic vector is, however, tricky. We choose an arbitrary base vector to aggregate all other vectors into. Suppose we chose the vector in the example above, $[p(\hat{x}_i, x_i)]$. We take every element, $p(\hat{x}_i, x_j)$, in every other dynamic vector, from other mappers. If there is an entry for x_j, we add the value of $p(\hat{x}_i, x_j)$ to it, otherwise, we add a new entry. In both cases, $p(\hat{x}_i, x_j)$ is added to the sum. We finally normalize the dynamic vector by its sum to get $[p(x_1|\hat{x}_i)....p(x_n|\hat{x}_i)]$.

[1]The reader might wonder why the summation is necessary for a one-element vector. However, in the general case, where the mapper takes more than one entity, the vector could have more than one element, each of which is in the form $p(\hat{x}_i, x_j)$.

Note that to compute the correct cluster prototype, we use:

$$p(y|\hat{x}) = p(y|\hat{y})p(\hat{y}|\hat{x}) \tag{1}$$

based on the formulation in Dhillon et al. [1], where $p(y|\hat{y})$ is ready, by assumption[2], and $p(\hat{y}|\hat{x})$ has also been computed.

In our application, however, we have multi-relations. For example, to cluster queries, we need to use the information gotten from clustering ads, ad keywords, and query terms. The generalization is, however, natural. For every query cluster, we compute a group of cluster prototypes (rather than a single cluster prototype), each of which is computed from the relation between queries and one of the other dimensions (ads, for example). Accordingly, when we compute the distance between a query and a query cluster centroid, we compute the average distance between the query and each of the query cluster prototypes.

4 Landmark Discovery

To discover the landmarks of a cluster, we propose computing a landmark measure for each member query in the cluster. The landmark measure is the ratio of the KL-divergence between the entity and the second closest cluster prototype to the KL-divergence between the entity and the closest cluster prototype. After computing the landmark measure, we select the n entities having the largest landmark measures as the landmarks of the cluster.

5 Initial Results

We ran the system on all queries/ads from Ireland over a week in August, considering queries that had a minimum threshold of ad clicks in aggregate. Our system was capable of discovering a cluster about banking which had the following landmarks: "ulsterbank", "aib ie internet banking", "aib internet banking", "aib online banking", and "ulster bank ireland". The system was also able to discover a cluster about hotels and houses, having the landmarks: "blarney golf resort", "ormonde hotel kilkenny", "dundrum house hotel", "dunbrody house", "hotels galway city", and "galway hotel deals". Furthermore, the system detected a cluster about games and matches with the landmarks: "www moshimember com", "match", "slender game download", "slender download", "world match play darts", "spiderman games", "www girlsgogames com", "morton games 2012", and "www lottery ie". The work is still in progress — we are currently running the system on more data.

6 Related Work

As a programming tool, MapReduce provides a powerful abstraction, hiding lots of details, and providing a framework for large-scale applications. Many data processing applications have used the MapReduce framework such as: [3], [4], [5]. Many machine learning applications have made use of MapReduce such as [6], for collaborative filtering, whereas others appealed to other frameworks such as [7], for deep learning.

Many versions of co-clustering have been proposed; see [1] or [8], for an example. In this paper, we propose a multi-dimensional large-scale co-clustering framework for [1], by generalizing [2], which was proposed for binary relations. We also go a step further and detect landmarks to help explain clusters, therefore achieving a goal similar to that proposed in [9] and [10], at scale and for multi-relations, however.

The problem of query understanding is not new. A recent user intent study was performed in [11]; one interesting finding was that it is usually hard to figure out the intent of the user from a single query. Other works tried to understand the query using lexicon modeling such as [12]. We view lexicon initiatives as an orthogonal addition to our approach, in case the queries are reliable. However, we observed that, in many cases, queries are just random words with little structure, if any. The reader is invited to think about some of the queries she uses while searching.

[2]There is a bootstrap MapReduce phase to compute $p(y|\hat{y})$, at the first iteration, given the initial labels.

7 Conclusion and Future Work

In this paper, we have proposed a novel framework for query understanding at scale. The initial results are encouraging — we are currently running the system on much larger data. In our future work, we plan to propose a new method of evaluation that is based on feeding the landmarks into downstream systems (e.g., classifiers in quality models).

References

[1] Dhillon, I. S. & Mallela, S. & Modha, D. S. (2003). Information Theoretic Co-clustering. *In Proceedings of ACM KDD, Washington, DC, 2003*.

[2] Papadimitriou, S. & Sun, J. (2008). Disco: Distributed co- clustering with map-reduce. *In Proceedings of ICDM, 2008*.

[3] Pike, R. & Dorward, S. & Griesemer, R. & Quinlan S. (2005). Interpreting the data: Parallel analysis with sawzall. *Sci. Prog. J., 13(4)*.

[4] Gedik, B. & Andrade, H. & Wu, K.-L. & Yu, P. S. & Doo, M. (2008). SPADE: The System S declarative stream processing engine. *In SIGMOD, 2008*.

[5] Isard, M. & Budiu, M. & Yu, Y. & Birrell, A. & Fetterly, D. (2007). Dryad: Distributed data-parallel programs from sequential building blocks. *In EuroSys, 2007*.

[6] Das, A. & Datar, M. & Garg, A. & Rajaram, S. (2007). Google news personalization: Scalable online collaborative filtering. *In WWW, 2007*.

[7] Le, Q. V. & Ranzato, M. A. & Monga, R. & Devin, M. & Chen, K. & Corrado, G. S. & Dean, J. & Ng, A. Y. (2012). Building high-level features using large scale unsupervised learning. *In ICML, 2012*.

[8] Dhillon, I. S. (2001). Co-clustering documents and words using Bipartite Spectral Graph Partitioning. *In KDD, 2001*.

[9] Tu, K. & Ouyang, X. & Han, D. & Yu, Y. & Honavar, V. (2011). Exemplar-based Robust Coherent Biclustering. *In SDM, 2011*.

[10] Lashkari, D. & Golland, P. (2007). Convex Clustering with Exemplar-Based Models. *In NIPS, 2007*.

[11] Donato, D. & Donmez, P. & Noronha, S. (2011). Toward a deeper understanding of user intent and query expressiveness. *In SIGIRI 2011 workshop on query representation and understanding*.

[12] Liu, J. & Li, X. & Acero, A. & Wang Y.-Y. (2011). Lexicon Modeling For Query Understanding. *In ICASSP, 2011*.