LATENT TARGETING SEGMENTS VIA FREQUENT DIRECTIONS

Jonathan Malkin, Alexander Saydakov, Lee Rhodes

{jmalkin,saydakov,lrhodes}@oath.com

ABSTRACT

We present our implementation of Frequent Directions, a streaming algorithm for approximate Singular Value Decomposition (SVD), to uncover latent advertising segments. We demonstrate that our system can efficiently scale to large scale data and provide meaningful vectors characterizing the ad targeting segment space. This represents a first step in being able to use segment data for user modeling, rather than just qualifying users to receive an ad, and holds the promise of allowing fixed-size segment profiles to improve storage predictability.

1 Introduction

Data is at the heart of everything we do at Oath, whether personalizing content or targeting advertisements; the entire promise of the Oath brand lies in our ability to reach a vast audience. Although we have large amounts of data, not all of it has been as useful as we might hope.

Ad targeting relies on what are known as *segments*. Advertising segments are how advertisers define an audience. A segment may indicate a user age bucket, an interest in celebrity news, a household income bracket, the presence of an app on a device, or some combination of other segments. Some segments are defined based on internal Oath data, and some are received from 3rd parties. While there may be a score associated with each segment when initially determining a user's qualified segments, at serving time they are generally treated as binary indicators. In total, there are hundreds of thousands of distinct segments.

With so many segments from which to choose, campaign managers setting up new advertising campaigns cannot know more than a tiny portion of all possible segments. When trying to target a combination of dimensions, it is often easier to define a new segment than to search for an existing one. In addition, by having our own data as well data from 3rd parties, if the segments are at all meaningful then we expect substantial redundancy.

Due to the rich data sets, one might assume that segment data is an important part of a variety of data-driven systems. That has not been the case: Attempts to use segment data for tasks such as predicting ad clickthrough rates have been abandoned because there was no obvious benefit. We believe this is largely due to very high cardinality of the segment space.

With so many input dimensions, some sort of dimensionality reduction seems desirable. Beyond improved modeling, such a reduction also allows the possibility of generating fixed-size user profiles which would allow for greater predictability in capacity planning. For a variety of reasons we also want to retain interpretability of the resulting dimensions, to be able to handle tasks such as revenue sharing and attribution modeling when using 3rd party targeting segments or providing users access to intelligible information about their own profiles. Such requirements argue against deep learning methods, which perform very well but are known for being inscrutable.

Latent Semantic Analysis (LSA) has been successfully used for tasks such as word clustering in natural language processing [3]. The same principle is also heavily used in collaborative filtering systems, especially the earlier systems [8, 10]. LSA is based on Singular Value Decomposition (SVD), one of the basic matrix decompositions. The method finds the primary directions within a matrix and provides rotation and scaling matrices that can be used to project data into the uncovered latent dimensions. In doing so, it clusters dimensions that carry values in a similar context. Because SVD is purely a linear algebra technique, it does so without assigning meaning to those clusters. For a given matrix $A, SVD(A) = U\Sigma V^T$ where U is a matrix of the eigenvectors of AA^T, Σ is a diagonal matrix with the singular values sorted in decreasing order, and V^T is the eigenvectors of A^TA.

Frequent Directions (FD) [7, 5] was proposed several years ago as a streaming solution for low rank approximation of the SVD of an input matrix. FD is a single-pass algorithm that can be efficiently parallelized, and also scales linearly (amortized) in the number of input points, number of input dimensions, and output dimensionality. Although data sketches for operations such as unique counting or frequent items are more well-known, FD is also a sketching algorithm, in this case providing an approximation of the k-rank decomposition of a matrix.

In this paper, we describe our prototype system for finding latent targeting dimensions. We begin with a brief overview of Frequent Directions and some needed background on ad targeting segments. We also describe some practical issues we encountered while ensuring that FD would be able to scale appropriately. While we have not yet been able to bucket test

our results, analysis of the resulting matrices suggests we are uncovering meaningful dimensions that may help us gain control of segment data, allowing for improvements in user modeling.

2 Frequent Directions

As mentioned earlier, Frequent Directions is a streaming algorithm for computing a low rank approximation of an input matrix with provable, deterministic error bounds. As an approximate streaming algorithm, FD is also an example of a sketching algorithm. Sketch algorithms [2] for efficient and approximate processing of large-scale data have been widely used at Yahoo and many other companies to substantially speed up data processing for tasks such as unique user computations and quantile estimation. Although FD seems a departure from those other classes of sketches, the algorithm has deep theoretical connections to the Heavy Hitters sketch originally proposed by [9] and represented in [2] as the Frequent Items sketch.

The sketches of most interest for very large scale data processing have several properties. The first is that they are *single-pass* algorithms, meaning we need to make only a single pass through the input data. The second is that they are *mergeable*, meaning we can take two sketches computed over independent data sets and efficiently merge them to produce a sketch with the same accuracy as if we had concatenated the two input data sets and send them to a single sketch. These two properties allow us to take advantage of the parallelism offered by distributed compute clusters, or even to process data in an online streaming manner.

2.1 The Algorithm

The Heavy Hitters approach of [9] tracks the most common items in a stream by maintaining a list of size k of items seen, along with a counter for each item. There are a number of slight variations for handling updates, but the basic idea is that when an item is added to the sketch we will either increment an existing counter, or else we decrement all counters (purging any that fall below one) and add the new item. By tracking the total amount decremented, we can obtain good frequency estimates of an item where the true frequency of that item is within provable error bounds with high confidence.

Frequent Directions takes inspiration from that idea, but using directions in the matrix data rather than item counts. The algorithm performs singular value decomposition incrementally, tracking the directions with the highest singular values, decrementing those values to prune away the less major directions. See Algorithm 1 for a formal description.

As we can see from the algorithm, whenever B fills its last zero row, we perform SVD on the matrix and keep only the top k directions; the remaining rows are left as zero, awaiting more data. SVD runs in $O(dk^2)$, but by performing the expensive computation only when rows of B are full we apply SVD only $n/2k$ times which gives an amortized run time of $O(nkd)$ using $O(kd)$ space.

Algorithm 1 Frequent Directions

Input: $k, A \in R^{n \times d}$
  $B \leftarrow$ all zeros matrix $\in R^{2k \times d}$
  for $i \in 1$ **to** n **do**
    Insert a_i into a zero-valued row of B
    if B has no zero valued rows **then**
      $[U, \Sigma, V] \leftarrow \text{svd}(B)$
      $\delta \leftarrow \sigma_k^2$
      $B \leftarrow \sqrt{\max(\Sigma^2 - I_k\delta, 0)} \cdot V^T$
    end if
  end for
  return B

2.2 Error Bounds

Frequent Directions has several proven, deterministic error bounds. Matrix approximation proofs generally focus on bounding the covariance error or the projection error. One well-known linear algebra result that we will use is that the optimal rank k approximation of a matrix A, which we denote as A_k, is the truncated singular value decomposition using the k largest singular values and their corresponding directions.

There are two related covariance error bounds for FD. The first is in terms of the Frobenius norm[1] of the input matrix, from [7]:

$$||A^T A - B^T B||_2 \leq 2||A||_F^2/\ell$$

where $\ell > k$ is the number of rows used to construct B. If we instead bound the reconstruction error, we have from [5] that:

$$||A^T A - B^T B||_2^2 \leq 2||A - A_k||_F^2/(\ell - k)$$

Alternatively, also from [5], there is a reconstruction error bound on the rank k matrix resulting from projecting A on the row span of B:

$$||A^T A - \pi_{B_k}(A)||_F^2 \leq (1 + \frac{k}{\ell - k})||A - A_k||_F^2.$$

The only drawback to these bounds is that, while strong results about the overall quality of the approximation, they give us little actionable information about specific directions beyond their aggregate behavior.

By following Algorithm 1 as written, we implicitly use $\ell = 2k$. Use of a larger value would decrease the approximation error, but would also require a more expensive incremental SVD calculation.

[1] The Frobenius norm of matrix A is computed as $||A||_F = \sqrt{\sum_{i,j} A_{i,j}^2}$; the norm $||A||_2 = \sigma_{max}(A)$ where $\sigma_{max}(A)$ is the largest singular value of A.

3 User Profile Data

User profiles in Yahoo's ad systems are stored in the Grid User Profile (GUP), and can range from tiny to rather large. The profiles contain a variety of information, from search keywords to inferred demographics. As described in 1, for this work we focused exclusively on ad targeting segments.

GUP's segment data contains roughly 280,000 distinct segments ids, although we have a total of nearly double that defined and active; some segments, such as technical information about a user's current device, are defined but added on the fly in real time.

As mentioned earlier, we believe there is a high degree of redundancy in the segment data, arising from both highly correlated segments and having multiple views of a user from different data providers. As an example of correlated segments, consider a segment hierarchy: if one branch in a hierarchy is substantially more prevalent than the other branches, then the parent and its child will be highly correlated. As an example of duplication across data providers, if we have determined that a user likes following football then there is a reasonable chance that one or more 3rd party data providers have also discovered the same thing. In such cases, each data source would provide a separate segment indicating an affinity for the sport, again resulting in effectively redundant targeting segments. Finally, despite the huge number of segments available, most ad campaign managers are familiar with only a very small number of segments, defining a new ones as needed.

The total number of user profiles in GUP is also quite large. Even restricting ourselves to the set of profiles defined as active, we have nearly 9B total profiles.

4 Experiments

For these proof-of-concept experiments, we considered the profiles for registered accounts, which tend to be the richest profiles with the highest mean number of segments, and seemed likely to give the best training data for a low dimensional projection. This yielded a total of about 780M total user profiles. We constructed our input matrix A by treating each user record as a row and each column as a different segment.

Even using this subset of user profile data, a matrix of $780M \times 280k$ would still be challenging to handle efficiently because the decomposed matrix is dense even if the input is sparse. In order to experiment with how the algorithm scales and to limit ourselves to the most meaningful segments in these initial experiments, we created input features by taking the top 20k, 50k, or 100k most frequent segments; the least common segments reached 1.4%, 0.3%, and 0.02% of profiles, respectively, indicating a very long tail of segments with little audience reach. We tested two values of k, 50 and 100.

We implemented Frequent Directions and created Pig UDF wrappers to enable easy deployment on our compute clusters, using the ojAlgo [1] linear algebra package for SVD and matrix multiply.

Our raw segment data consists of binary features: A segment has a value of 1.0 if present and 0.0 otherwise. Since this work was inspired by Latent Semantic Analysis, we primarily focused on IDF (inverse document frequency) features [11]; term frequencies per record were only 0 or 1 so term frequency weighting was inherently binary.

The code used for these experiemnts is available at `https://git.corp.yahoo.com/jmalkin/frequent-directions`.

5 Results

Because we building a full system using these new features was beyond the scope of this work, our evaluation focused on two components: Execution time and a brief qualitative look at the singular values and dimensions.

5.1 Run Time

Our initial attempts at testing the model were useful for learning how to scale the system. Because the algorithm parallelizes very efficiently, our first attempt built FD models map-side. While that is conceptually a viable model, the correct place to build them is in the combiner (a map-side pre-reducer) to avoid instantiating a new model for each record. But when using the combiner alongside the data loading job, we ran into memory issues — even when the full size of the model should have been only about 16Mb.

Due to the memory issues, we inserted an extra step between data loading and funneling data to Frequent Direction. And to avoid all the data, even the reduced-scale matrices, flowing to a single node we set up a 2-tier merge model. The first stage generates a random integer and splits the raw data into $75^2 = 6525$ partitions. We then partition by 75 to speed the merge before finally sending all results to a single node. We believe there is significant room for further improvement in this process, but the overall performance was sufficient to run further tests.

Table 1. Frequent Directions execution time, both CPU core time and wall clock time. All times are hours and used $n = 780M$ samples.

k	d	CPU Time (hrs)	Wall Clock Time (hrs)
50	20,000	10.6K	3.2
50	50,000	25.0K	8.2
50	100,000	51.5K	14.1
100	20,000	19.5K	4.9
100	50,000	49.0K	12.9

In Table 1 we can see the execution times for Frequent Directions. The first thing to note is that the overall execution time scaling lines up very well with the expected results from theory. Wall clock time for distributed jobs tends to be fairly noisy due to cluster variability, and even that follows the predicted pattern.

We believe that we can improve the run time with a better job partitioning scheme, and that we will be able to address some of the memory challenges our initial code experienced. The use of ojAlgo meant that we had a high quality implementation of SVD, but that came at the cost of some suboptimal memory use. We will return to this point in Section 6.

5.2 Qualitative Analysis

With $k = 50$, the singular values fell off very rapidly, with the ratio of the first to last singular value on the order of 10^5. When expanding to $k = 100$ we found a very similar ratio between the first and last values, while the ratio of the first to the 50th singular value had decreased to 10^3. This suggests that the singular values do fall off fairly quickly, but that the values are also distorted by the subtraction process.

The biggest caveat to the analysis of the singular values is that the FD algorithm iteratively subtracts from all the singular values. As noted in [6], the algorithm as presented in Section 2.1 tends to underestimate the Frobenius norm of the original data A, although still within the deterministic error bounds. [4] proposed a modification called Compensative FD that, like with the original heavy hitters algorithm of [9], restores the subtracted weight before returning a final result.

By making the restoration optional, we were able to compare results with and without compensation. When compensating, the last singular value was between 20-30% of the first singular value when $k = 50$; with $k = 100$, that ratio was in the 15-20% range. The compensation process provides a sort of "noise floor" for the singular values. While compensation will help avoid under-estimating the Frobenius norm, we have not yet found a robust way to determine whether the resulting vectors are a meaningful improvement in practice.

Examining the segments associated with the largest magnitude weights, both positive an negative, the system appears to be working as intended. LSA groups together items that appear in a similar semantic context using a bag of words model, so the method will identify correlated segments without assigning meaning to them. As examples, we found directions associated with high household incomes and more expensive spending items; others clustering parenting and children; Spanish language speakers; lookalike segments; and even one that appeared to group together a number of non-US English language users.

We also noticed some questionable segment groupings, assuming the names are meaningful (which is far from certain): A set of segments seeming to rate the likelihood that a user is "underbanked" included the full range from most likely to least likely, all with identical weights from exactly identical user counts per segment. Such overlap between segments, even if not expected for segment names that suggest disjoint sets, is exactly what motivated this project in the first place.

Although the directions discovered do appear meaningful and are quite consistent within a day across values of k and d, there is noticeable variation between days, although similar sorts of patterns are present. GUP is a complete daily snapshot, so this inter-day variation happens despite despite substantial overlap between data sets.

6 Future Work

Having established that Frequent Directions can scale to handle our advertising profiles, there is still quite a bit left to do. The obvious missing step is a bucket test using the new features to train one of our online models, ideally around ad click prediction.

Beyond that we can improve the memory footprint of the SVD process, possibly at some cost of speed. In order to scale up to very large matrices, on the order of $k = 1000$ with a similar number of dimensions yielding a working model size on the order of 2Gb, an emphasis on the memory footprint will be vital.

While the vectors within a day were largely self-consistent across different values of k and d, the increased level of variation between days is something to address. From a theoretician's perspective, the latent segments are the result of the algorithm and any consumers should be prepared to accept them. From a business perspective, we generally prefer to associate the dimensions with a more intuitively meaningful description, and to have stable associations between descriptions and dimensions. A dimension alignment process, perhaps on the orthonormal vectors in V^T, would help improve the stability of these features over time and remains an open problem. At the same time, one useful aspect of FD that is that its mergeability means we can incrementally add new data to an existing sketch without reprocessing old data, allowing the results to change slowly over time as determined by the data.

There are also a number of extensions to SVD for use in collaborative filtering [10]. Finding ways to apply some of them within the frequent directions framework would provide novel theoretical results in addition to potentially improving the resulting features.

Finally, we would like to expand the scope of FD to new problem areas, for instance to content personalization models or to building latent user models. We believe this will be a useful tool for efficient, meaningful dimensionality reduction and allow us to make better use of the data we have already collected.

7 References

[1] oj! Algorithms. `http://ojalgo.org`.

[2] Sketches library from Yahoo. `http://datasketches.github.io`.

[3] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990.

[4] A. Desai, M. Ghashami, and P. Jeff M. Improved practical matrix sketching with guarantees. *IEEE Trans. on Knowledge and Data Engineering*, 28(7):1678–1690, 2016.

[5] M. Ghashami, E. Liberty, J. M. Phillips, and D. P. Woodruff. Frequent directions: Simple and detrministic matrix sketching. *SIAM J. of Computing (SICOMP)*, 45(5):1762–1792, 2016.

[6] M. Ghashami and J. M. Phillips. Relative errors for deterministic low-rank matrix approximations. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 707–717. Society for Industrial and Applied Mathematics, 2014.

[7] E. Liberty. Simple and detrministic matrix sketching. In *KDD'13*, Aug. 2013.

[8] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[9] J. Misra and D. Gries. Finding repeated elements. Technical report, Ithaca, NY, 1982.

[10] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, volume 2007, pages 5–8, 2007.

[11] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.