Application of Clustering for Unsupervised Language Learning

Jeremy Hoffman and Omkar Mate

Abstract

We describe a method for automatically learning word similarity from a corpus. We constructed feature vectors for words according to their appearance in different dependency paths in parse trees of corpus sentences. Clustering the huge amount of raw data costs too much time and memory, so we devised techniques to make the problem tractable. We used PCA to reduce the dimensionality of the feature space, and we devised a partitioned hierarchical clustering approach where we split the data set and gradually cluster and recombine the partitions. We succeeded in clustering a huge amount of word data with very reasonable time and memory cost.

Motivation

Fully automated learning of similar words and dependency paths is extremely pertinent to many natural language processing (NLP) applications. Similarity estimation can help with problems of data sparseness in statistical NLP [4], and clustering could automatically generate similarity estimates. Word similarity estimates also can be used in question answering and machine translation, major areas of current NLP research.

Overview of Process

Our process is to build a large input matrix that describes words from their appearances in various dependency paths, cluster the words in this feature space, then estimate that words that ended up in the same cluster are semantically similar. Since language is so varied with a vast vocabulary, we must build a huge matrix to infer anything useful from our clusters. However, clustering may not be tractable for huge matrices due to time and computer memory constraints, so we took a more complex approach. In the following sections, we discuss how principle component analysis and a multi-tiered form of hierarchical clustering solved this problem and allowed a large matrix to be clustered.

Input Data

Our input data was a corpus of six million newswire articles, parsed using MINIPAR into dependency path triplets, as in [6]. The corpus contained about 750,000 unique noun pairs and about 70,000 unique dependency paths. From these data, we

constructed a matrix of training examples. Our input matrix has *m* rows corresponding to different nouns, and *n* columns corresponding to different dependency paths. An entry (*i*, *j*) is the number of times that noun *i* appeared in dependency path *j*. Any given word is likely to only appear in a very small number of sentences, even in a corpus of six million articles, so the input matrix is very sparse.

Principle Component Analysis

To cluster nouns, we need to reduce the column dimension of the input matrix. In order to do this, we use Principal Components Analysis (PCA). PCA was implemented using Simple Lanczos algorithm for Singular Value Decomposition [3]. (See Figure 1)

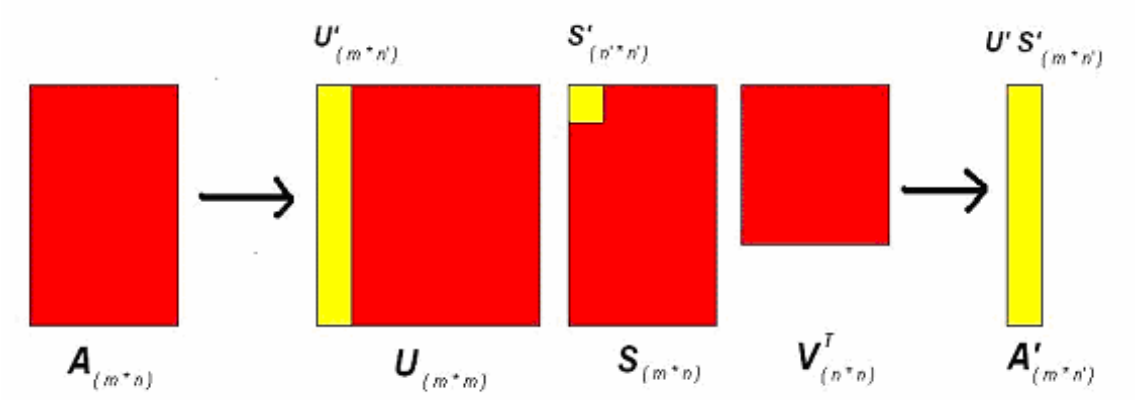

Figure 1: PCA using Singular Value Decomposition

The input matrix A is decomposed as

$$\mathbf{A} = \mathbf{U}\,\mathbf{S}\,\mathbf{V}^{\mathbf{T}}.$$

S has nonzero entries only along the diagonal, representing singular values of **A**, or eigenvalues of $\mathbf{AA}^{\mathbf{T}}$. Since $\mathbf{AA}^{\mathbf{T}}$ represents covariance of rows of **A**, (i.e. nouns), its maximum eigenvalues represent eigenvectors directions in which covariance is maximized. We choose the first *k* (which happen to be *k* largest) entries of **S**, and the first *k* columns of **U**, to get **U'**. (The choice of *k* is based on the size of the eigenvalues and the desired computational efficiency for clustering.) We multiply **U'** by **S'** to get **A'**, which is the desired output matrix with reduced column dimensions.

Following are the results we obtained by running PCA on input matrices of various sizes:

No. of Rows (i/p)	No. of Columns (i/p)	No. of Columns (o/p)
7000	3000	20
14000	5000	100
21000	5000	119
30000	10000	138
100000	20000	144

Partitioned Hierarchical Clustering

Hierarchical clustering (HC) is well-suited to the task of grouping similar nouns. In contrast to "hard assignment" clustering algorithms such as K-means, HC builds a tree, or dendrogram, of closest points deterministically. The basic HC procedure to cluster m points into k clusters is as follows. First, start with m clusters of one point each. Then, find and merge the two closest clusters, and repeat m-k times.

We computed the distance between two points (i.e. nouns) as their cosine, computed by dot product of their feature vectors. The distance between two clusters can be defined as the minimum, maximum, or average distance between points in the clusters. Minimum distance clustering tends to produce long chains, while "spherical" clusters more intuitively match word similarity. Maximum distance clustering is susceptible to outliers, which makes it unsuitable for this problem because the data is noisy (the corpus could contain a few "bizarre" sentences). Thus, average-link clustering was the most suitable approach. Specifically, we maintain along with each cluster the mean of the feature vectors of its points, and compute the similarity of two clusters as the cosine of their mean vectors.

HC is computationally expensive. In particular, average-link clustering on m points in d-dimensional space takes $O(dm^2 \log m)$ time and $O(dm+m^2)$ memory [5]. Even if cluster mean vector and cosine values are discretized to 4-byte integers, storing m clusters and their pairwise cosines takes $4(dm+m^2)$ bytes; for m=50,000, this is 10^{10} bytes = 10 GB, more than the memory of most computers.

To make HC tractable, we devised an approach that we call partitioned hierarchical clustering. In this approach, the m points are split into k partitions of size m/k such that HC on m/k points can be executed on a single computer. For each partition, HC is used to reduce the number of clusters by 50% by making $m/(2k)$ mergers, so that the m/k points are combined into $m/(2k)$ clusters. Then pairs of partitions are concatenated to create $k/2$ partitions each containing m/k clusters (which contain a total of $2m/k$ points). HC is again used to reduce the size of each partition by 50%, and pairs of partitions are again combined, until all partitions are eventually recombined in the $\log_2 k$'th step. (See Figure 2.)

Since partitioned HC requires running HC on at most m/k clusters at a time, its space requirement is $O(dm/k+(m/k)^2)$, an improvement by a factor of about k^2. Thus partitioned HC can be run on a computer that otherwise would not have enough memory to cluster the data. Partitioned HC is also fast; it requires $\log_2 k$ steps, and the ith step entails i instances of running HC on m/k clusters. Thus partitioned HC has a time cost of $O(d(\log k)^2(m/k)^2 \log(m/k))$, an asymptotic improvement over normal HC of a factor

$(k/\log_2 k)^2$. Furthermore, the process can be parallelized on several computers, while this was not possible with normal HC.

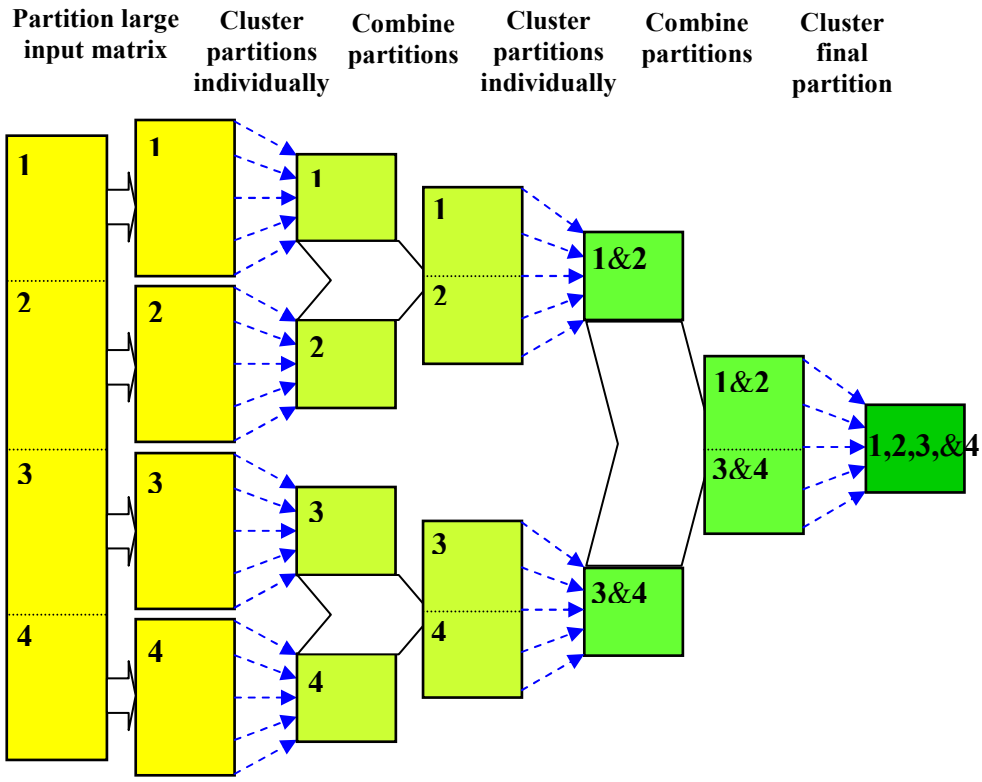

Figure 2: Partitioned Hierarchical Clustering with k=4 initial partitions.

The problem introduced by partitioned HC is that the guarantee of the closest points being merged is lost. If two very close points are in separate partitions that only meet in the final step, they may not be merged at all if they belong to clusters whose means are farther apart. This is unlikely to happen except in borderline cases, because close points should end up in close clusters that will eventually be merged, but a further investigation into partitioned HC's deviation from "normal" HC clusters would be worthwhile. A possible solution to this problem is to perform partitioned HC multiple times, randomly picking a different initial partitioning of the data each time, and then averaging the word similarity results.

Implementation and Testing

To evaluate the feasibility of our approach, we implemented PCA with SVD using the code from the SVDPACKC package [1] and implemented partitioned hierarchical clustering from scratch. For our trial run's input matrix, we took 65,000 of the most frequently occurring nouns and 20,000 of the most frequently occurring dependency paths from the corpus. Running PCA on the data reduced the column

dimension of the matrix from 20,000 to 144 (in about four minutes on a Windows XP PC with 256MB RAM). Dividing the 65,000 rows into four partitions of 16,250 and performing partitioned HC, successively reducing the 16,250 clusters by half, we generated 8,125 clusters (in about 35 minutes on a Dual 3 GHz Xeon shared Linux machine with 2 GB RAM). When hierarchical clustering was run without partitioning, an "out of memory" error occurred.

Thus we conclude that our approach is fundamentally sound, and can allow clustering of matrices with a very large number of both rows and columns.

Further Work

To continue this work, we would try running our program on even larger input matrices. To evaluate the semantic significance of our cluster output, we would compare the word similarity suggested by our cluster output to a "gold standard" such as WordNet [6], Latent Semantic Analysis [2], or human-tagged data. We would compute the pairwise similarity of all of the words in our input data, and compare the average similarity of words that were clustered together in our output to the average similarity of words that were not.

Acknowledgements

We acknowledge the valuable guidance provided by Rion Snow, Prof. Andrew Ng, Prof. Dan Jurafsky, and Prof. Gene Golub.

References

[1]Berry. "SVDPACKC" ftp://cs.utk.edu/pub/berry

[2] Laham, D. (1998) "Latent Semantic Analysis @ CU Boulder." Last updated Oct 1998. Accessed Nov 2005. http://lsa.colorado.edu/

[3] Golub, G., and Loan, C. (1996) Matrix Computations. Baltimore, MD: Johns Hopkins U Press.

[4] Pereira, F., Tishby, N., and L. Lee. (1993) "Distributional clustering of English words", 30th Annual Meeting of the ACL, pp183-190.

[5] Schütze, H. "Single-Link, Complete-Link & Average-Link Clustering." No date given. Accessed Nov 2005. http://www-csli.stanford.edu/~schuetze/completelink.html

[6] Snow, R., Jurafsky, D., and Ng, A. (2004) "Learning syntactic patterns for automatic hypernym discovery", Advances in Neural Information Processing Systems 17, 2004.