Parallelized Power Iteration Clustering of Nouns and Verbs using Subject-Verb and Verb-Object Pairs

Philip Gianfortoni PWG@CS.CMU.EDU
Mahesh Joshi MAHESHJ@CS.CMU.EDU
Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213 USA

Abstract

We explore the use of Power Iteration Clustering for large-scale clustering of nouns and verbs, using subject-verb-object triple relations. We have implemented a parallelized version of PIC, which can efficiently handle clustering of hundreds of thousands of sparse "documents." We tested our implementation on clustering of over 1 million noun phrases, and over 650K verb phrases. We also evaluate our verb clustering quantitatively using verb categories from VerbNet as our "gold standard" clusters.

1. Introduction

We investigate the problem of developing word clusters using distributional similarity statistics on massive amounts of data. Developing clusters of semantically similar words is an important task because these clusters can be used for exploratory data analysis as well as for smoothing in classification tasks.

In particular, we investigate clustering of predicate relations using their arguments as context. Using shallow parsing information or predicate relation information is attractive because assuming the parses and/or relation extractor is reasonably accurate, clusters produced using such information may be more informative than N-gram clusters.

Toward this goal of using large-scale data for verb clustering, we have implemented an efficient parallelized implementation of Power Iteration Clustering (PIC) (Lin & Cohen, 2010a). In our results we present PIC runtime performance results for verb as well as noun clustering, and quantitative accuracy evaluation of verb clusters using VerbNet verb categories.

This is a report for a class project done as a part of the *Machine Learning with Large Datasets* (10-605) course at Carnegie Mellon University, in the Spring 2012 semester.

2. Related Work

This project draws from three different lines of research. The first line of research is the development of using distributional similarity to perform clustering at the word level using shallow parsing information for distributional similarity (Pereira & Tishby, 1992). We use similar subject-verb-object (SVO) distributional statistics.

The second line of research that we draw from is that of iterative, graph-based clustering techniques. Here we use Power Iteration Clustering (PIC) algorithm (Lin & Cohen, 2010a), which is a very recent addition to the collection of clustering algorithms.

The third line of research that we draw from is that of scaling up iterative graph-based algorithms using cluster mangers such as Hadoop. Fairly recent research on a project called PEGASUS has yielded a generic way to write algorithms that use iterative matrix-vector multiplication at large scale. Part of PEGASUS, which is a toolkit for studying large graphs is a generalization of such multiplication called GIM-V that can run on Hadoop clusters (Kang et al., 2009). Although this project is highly optimized for such tasks, and although it is becoming more and more ubiquitous, we question whether Hadoop is potentially too constrained. We try an alternative approach by using a higher-level distributed framework and implementing a simple matrix-vector library on top of it instead of creating abstractions for GIM-V and tailoring it to Hadoop.

3. Approach

3.1. Dataset

We attempted clustering on two datasets: SVO triples from the Never-Ending Language Learning (NELL)

project Carlson et al. (2010) and the argument and predicate relations extracted from the the ClueWeb09 dataset by the Reverb systemFader et al. (2011).

We chose these datasets because they were both sufficiently large that they posed major challenges for naive implementations of the algorithms.

For our evaluation we use the Reverb dataset, which contains around 11 million subject-predicate pairs, around 10 million object-predicate pairs, with approximately 660,000 unique predicates. There are roughly 1.3 million unique subjects, and roughly 1.7 million unique objects. We performed a connected components analysis on the co-occurrence matrix and observed that the giant connected component contains most of the nodes. Hence we cluster the full set of predicates.

When clustering, we cluster all four combinations of links separately. Nouns in subject positions (NSUB) are clustered with the verb predicates as context. Similarly nouns in object position (NOBJ) are clustered with the verb predicates as context. The predicates themselves are clustered twice, once using NSUBs as context, and once using NOBJs as context.

3.2. PIC

PIC is a fast, iterative, dimensionality reduction algorithm that performs nearly as well as other state-of-the-art clustering algorithms, such as spectral clustering, but is much faster. PIC produces a lower-dimensional embedding of a dataset, which can then be used in a traditional clustering algorithm, such as K-means. The only requirement for running PIC is an affinity matrix between the items intended for clustering. Thus PIC may be seen as operating on a graph of nodes connected by weighted edges. If such a graph is bipartite, then a technique called path-folding can be used to avoid explicitly computing an affinity or similarity matrix. Since we are attempting to cluster noun phrases and verb phrases, using the different types of phrases as evidence for one another, we have a bipartite graph, so such a technique is ideal for our purposes.

At the heart of the algorithm is a series of matrix multiplications. The algorithm is initialized with a vector $\mathbf{v}^0$ containing a value for every node in the graph. At every iteration, the algorithm updates this vector by multiplying it by the affinity matrix (and possibly also normalization matrices), yielding an updated version of the vector. The main PIC update equation is as follows.

$$\mathbf{v}^{t+1} \leftarrow \frac{W\mathbf{v}^t}{\|W\mathbf{v}^t\|_1} \tag{1}$$

where v^t is the value of the PIC vector after iteration t, and W is the similarity or affinity matrix.

The final PIC vector gives an embedding of the data in a single dimension. Thus, if we run the algorithm multiple times with different starting values, we obtain a multi-dimensional embedding of the data, which can be fed straight to a clustering algorithm such as K-means.

Since we are clustering verbs using their arguments as context, we can use the path-folding trick described in Lin & Cohen (2010b) by splitting W into FF^T. This represents the same value of W except that F and $F^T$are both sparse and thus have efficient representations that require much less space to store than the entirety of W. In our case F and F^T represent the co-occurrence count of verb phrases with their arguments. Also following Lin & Cohen (2010b), we include a normalization vector $\mathbf{d} = FF^T\mathbf{1}$. Mathematically, this is represented by a diagonal matrix D, where $D_{i,i} = \mathbf{d}_i$. This matrix is then inverted and the final update equation becomes

$$\mathbf{v}^{t+1} \leftarrow \frac{D^{-1}(F(F^T\mathbf{v}^t))}{\|D^{-1}(F(F^T\mathbf{v}^t))\|_1} \tag{2}$$

We also maintain the following vector

$$\delta_{t+1} \leftarrow |\mathbf{v}_{t+1} - \mathbf{v}_t| \tag{3}$$

so that we can use the convergence criterion in Lin & Cohen (2010b), namely

$$|\delta_{t+1} - \delta_t| \leq \epsilon \tag{4}$$

3.3. Matrix by Vector Multiplication

As described earlier, PIC for bipartite data graphs is a series of matrix-by-vector multiplications. In order to abstract these matrix vector operations to work on large datasets in a scalable manner, we needed a distributed matrix vector multiplication library. To this end, we used the Spark framework (Zaharia et al., 2011) to implement a very simple vector and sparse matrix library including a representation that can be distributed across cluster nodes to increase performance. The simple library contains the following operations:

- sparse matrix by vector multiplication

  - column-partitioned
  - row-partitioned
- vector operations (L1 norm, scalar times, vector element-wise multiplication, and vector addition/subtraction)

3.4. Distributed Implementation

The Spark framework runs on the Mesos cluster manager. This is cluster manager that can be used to manage clusters for running distributed computation frameworks such as MPI, Hadoop, and Spark.

Spark is somewhat similar to Hadoop and indeed even may read and write to hdfs filesystems and interact with Hadoop sequence files. However, spark is structured in a much different way than Hadoop and uses a much higher level of abstraction. The extra abstractions makes it more difficult to optimize certain things such as data locality, but on the flip side offers the programmers a more intuitive view of the data.

3.4.1. Distributed Datasets

In essence, Spark provides a single abstract view of all data as a distributed dataset (RDD). The RDD is distributed using a partitioner much like the way a Hadoop partitioner distributes items. This abstraction hides all of the details of where the data is actually located, and in return offers a way to treat the dataset as a cohesive data structure without having to reason about where the parts of it will end up. In addition, none of the operations on distributed dataset actually destroy existing references to that dataset, instead returning a new dataset. Unlike Hadoop, these datasets do not require a strict key-value pairing (although there are several extra operations that are allowed when the RDD is a dataset of key-value pairs) and instead just represent a sequence of values like a table in a database.

We represent our sparse matrices by an RDD containing key-value pairs, where the key is the column id of a matrix and the value is a pair containing a row id and a numeric value. Likewise, vectors are just RDDs containing key-value pairs where the key is an index and the value is a numeric value. The reason for the column-major ordering is described below.

The Spark library provides many useful operations on RDDs but we only use a few of them. A few of the functions we use have a functional programming analogue, and couple have a database analogue. If we think about our data from a functional perspective, our data is simply a sequence of values, and if we think about it from a database perspective, it is a table of entries with several fields, one of which is a primary key. The functional methods that we use are the following:

- map: RDD[A]→(A→B)→RDD[B]
  Takes an RDD with values of type A, a function from $A \rightarrow B$ and turns it into an RDD with values of type B by applying the function to all elements in the RDD.
- flatMap: RDD[A]→(A→Seq[B])→RDD[B]
  Composes flatten and map, taking an RDD with values of type A and a function from $A \rightarrow Seq[B]$ and turns it into a an RDD with values of type B by applying the function to all elements to create a RDD with sequences of elements of type B.
- reduce: RDD[A]→((A,A)→A)→A
  Aggregates all elements in a dataset with an associative, commutative function.

The database-like methods that we use are the following

- join: RDD[(K,V1)]→RDD[(K,V2)]→RDD[(K,(V1,V2))]
  Joins values from two tables together that have the same key.
- groupByKey:RDD[(K,V)]→RDD[(K,Seq[V])]
  Collapses a sequence of key-value pairs with the same key into a single pair of a sequence of values.

Finally there are methods to convert to and from RDDs, as well as reduceByKey, a hybrid method that performs the functional reduce but only on the set of values belonging to each key.

- reduceByKey: RDD[(K,V)]→((V,V)→V)→RDD[(K,V)]
  Essentially the Hadoop reduce, except without the sorting. It takes an RDD of key-value pairs and a reduce function, and reduces all the values for each key, so that there is exactly one value for each key.

3.4.2. Implementation

Our implementation of matrix multiplication uses join, followed by a flatMap, followed by a reduceByKey. Since we store our matrices in column-major form, then we are able to partition the sparse matrices by columns. We also partition the vectors, by splitting them on their indices. Since vectors are keyed by these indices and matrices are keyed by their columns, as long as we are performing a matrix by vector multiplication, we can join vector with a sequence of elements

that it must be multiplied with, flatMap the multiplication, switching the indices in the process, and performing a reduceByKey that performs the summation with the reduceByKey operation.

Also, since PIC is an iterative algorithm, we needed fast matrix-vector multiplications. Spark provided us with a way to cache the sparse matrices used for these multiplications (via the RDD's cache method) across multiple iterations. Since the size of the matrices is inversely proportional to the number of computers used, we should always be able to guarantee that the matrix will fit in memory across all machines if we add enough machines.

- *join* matrix columns and vector elements:
$$\left(\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \\ a_{m1} & & a_{mn} \end{bmatrix}, \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}\right) = \left[\left(\begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix}, v_1\right) \dots \left(\begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix}, v_n\right)\right]$$
The key of a matrix is a column in the matrix, and the key of a vector is the row. Thus, when we join these, we get a sequence of matrix elements corresponding to a particular column in the matrix paired with a single vector element.
- $flatMap$ $:R_{p,i} = ([A_{p,1,i}, \dots, A_{p,m,i}], v_{p,i}) \rightarrow S_{q,j,i} = A_{p,j,i} * v_{p,i}$
Next we "flatMap" the sequence mapped to
- $reduceByKey$: $S_{q,j,i} = A_{p,j,i}v_{p,i} \rightarrow w_{q,j} = \sum_i S_{q,j,i}$

Similarly, L1 norm of a vector can be obtained with a reduce, scalar-vector operations can be performed with a map and element-wise vector operations can be obtained with join, followed by a map.

We were able to run this algorithm on the Amazon EC2 cluster, but we did not test its performance extensively. Furthermore, we did not employ any compression techniques, and we even left the indices as strings.

3.5. Parallel Implementation

In order to conduct experiments in a cheaper fashion than putting them on the cloud, we also developed a single computer parallel version of the algorithm. While we could run our distributed implementation on one machine, we found that there was a huge amount of overhead due to unnecessary scheduling and serialization.

To implement the matrix-vector operations library, we used Scala, which is a language that compiles to Java as an intermediate language and is thus intercompatible. Scala also happens to be the same language that Spark was written in, so we merely needed to supply another matrix-vector library without actually having to touch the PIC code.

One of Scala's features is a parallel collections library that uses "futures" on top of a thread pool to perform functional operations over datasets in parallel. For example, all of the functions listed above in the function section can also be applied to hash maps in Scala. If one of them is applied to a parallel hash map, then the operation will automatically be parallelized using the thread pool.

Thus, we were able to create a simple implementation of matrix-vector multiplication, using the map and fold (a more generic reduce) operations on parallel maps.

4. Experiments and Results

Since we had labeled data only for verbs, we focus only on the verb clustering results in this report.

We used the 272 verb categories available in VerbNet (Kipper et al., 2000) as cluster/category labels for the verb phrases in the ReVerb dataset. For every verb phrase, we simply attempted to match each word in the verb phrase with one of the list of words for a given verb category in VerbNet. When a given word had multiple categories in VerbNet, we randomly picked one of them as the gold standard category.

Out of the 664,746 unique verb phrases that we had in our data, we found category labels for 513,478 verb phrases.

4.1. PIC Settings

We ran PIC to cluster verbs using two types of co-occurrence information. First, we ran PIC to cluster verbs using the subject noun phrases (NSUB) that they appear with as the context. Second, we used the object noun phrases (NOBJ) that the verbs co-occur with as the context. For both experiments we ran PIC with 10 different initial random vectors. In all cases, we ran PIC to 50 iterations, saving the PIC vector at every 10 iterations.

The running time statistics for the noun as well as verb clustering are reported in Table 4.1. Each run used an 8-core 2GHz processor for parallelization of PIC, with

```
class ParallelVector(vector:ParMap[String,Double]){
  def data = vector
  ...
}
...
class ParallelMatrix(matrix:ParMap[String,Map[String,Double]]) {
...
  def vectorTimes(vec:ParallelVector):ParallelVector = {
    val vector = vec.data
    matrix.map{case (i,cols) => {
      i,cols.foldLeft(0.0){case (sum,(j,elem)) => {
            sum + elem * vector.get(j)
          }
        }
      }
    }
  }
...
}
```

Figure 1. The `ParallellMatrix` `vectorTimes` method

a maximum of 18GB heap size for the Java Virtual Machine.

4.2. K-Means Clustering

For running K-means clustering, we created a data matrix by stacking together the PIC vectors resulting from each of the 10 different random initializations. Such a data matrix was created at the end of every 10th iteration, so that we had a total of five datasets each for verb clustering using NSUB and NOBJ as contexts. In addition, we created a sixth dataset for each setting by stacking together the five different datasets that were created at every 10th iteration.

We used $K = 272$ (the actual number of categories in VerbNet) for running K-means clustering.

We evaluated the output of K-means clustering using the Adjusted Rand Index (ARI) measure (Hubert & Arabie, 1985), which is essentially equivalent to classification accuracy, but accounts for different possible permutations of the cluster labels. ARI also adjusts for random grouping of the elements. ARI ranges from 0 to 1, with 1 being the best possible ARI for a perfect clustering.

Results for the different runs of K-means clustering are shown in Table 4.2. Based on the ARI numbers, the clustering that we are getting is very inaccurate with respect to the "gold standard" clusters that we obtained using VerbNet. However, we are not sure if this necessarily means that the clustering we obtain is completely bad. VerbNet categories were created taking into account the linguistic aspects of verbs such as syntactic and semantic use. The limited co-occurrence information of verbs with subject or object nouns may not be able to capture this variation fully.

Context	Dataset	ARI
NSUB	iter-10	0.0005694
	iter-20	0.0001394
	iter-30	0.0001858
	iter-40	0.0000489
	iter-50	0.0001330
	iters-all	0.0009168
NOBJ	iter-10	0.0005433
	iter-20	0.0004592
	iter-30	0.0005332
	iter-40	0.0006333
	iter-50	0.0004473
	iters-all	0.0005107

Table 2. Results for K-means clustering on verb data, with NSUB and NOBJ as contexts. "iters-all" is the dataset that combines all the previous five datasets from every 10th iteration.

Based on our results it seems that co-occurrence with subject noun phrases is more useful for clustering verbs as per the VerbNet categories. However, given that the ARI numbers are very low, this is only suggestive evidence.

Data	Size (Rows)	Average Runtime (s)
NSUB, context=verbs	1,396,793	539
NOBJ, context=verbs	1,698,028	556
verbs, context=NSUB	664,746	437
verbs, context=NOBJ	664,746	388

Table 1. Average running time required for 50 iterations of PIC, across 10 different random initializations. **Size** indicates the number of nouns or verbs to be clustered.

4.3. Verb Category Classification using PIC Vectors

We also considered evaluating the quality of PIC vectors by using them as features for a supervised classification task of categorizing the verbs into the VerbNet classes. We limited the number of VerbNet classes to only those that had at least 10 verbs associated with them in the VerbNet lexicon — which gave us 164 VerbNet categories. This yielded a subset of 446,700 verbs belonging to one of the 164 categories. The datasets for these 446,700 verbs were created by taking the relevant subset of the datasets used for K-means clustering. We then divided each dataset into training, development and test subsets with 350,000, 46,700 and 50,000 verbs respectively. We then used a $L2$-regularized logistic regression model to predict verb categories using these datasets.

Unfortunately, the performance on the test sets was always the same as the performance of a simple majority class baseline — that is the logistic regression model[1] always predicted only the most frequent class present in the training data. We suspect this might have been due the very small numerical values of the features coming from the PIC vectors, as well as a fairly skewed distribution of the instances among the verb categories — only 17 verb categories had more than 5,000 instances each in the training set. Perhaps restricting the dataset to only consider a few major categories might be more useful as a next step.

5. Conclusions

Implementing PIC using the Spark framework and finding a reasonable method for evaluation provided interesting challenges. While writing the algorithm using Spark went much quicker than we thought, we were continually stymied by problems such as fighting with build systems, learning several new toolkits and/or APIs, trying to reasonably preprocess data, and adjusting PIC parameters. We learned a lot about trying to scale a problem up to large datasets, but we probably would have been more successful testing PIC on a task with a clearer evaluation. While PIC's ability to easily handle bipartite graphs was what inspired our desire to attempt using verbs as context for clustering nouns and vice versa, we needed to focus on a more targeted evaluation so that we could actually figure out what worked and what didn't. We leave a lot of questions unanswered and for future work. One thing we would have like to do was perform more speed comparisons to see just how well Spark works for this task. A second thing we would have liked to do is attempt better compression or at the very least map our phrases to numbers in order to reduce some of the high serialization overhead of using a distributed system. The original end goal was to use clusters as a way to smooth other word-level features, but we could not do a meaningful analysis of our clusters in order to use them appropriately. Finally one new direction that would have been interesting as far as clustering is concerned is to perform PIC at various stages and then try to obtain hierarchical clusters using the different vectors at different iterations. We think this would work because PIC converges locally before it converges globally meaning that some clusters would be more likely to converge before others.

[1]We tried several values of the regularization trade-off parameter.

Acknowledgments

We would like to thank the instructor of the course, Prof. William Cohen and the teaching assistants Alona Fyshe and Ni Lao for a well-designed course. We would also like to thank Frank Lin for his inputs on this project.

References

Carlson, Andrew, Betteridge, Justin, Kisiel, Bryan, Settles, Burr, Jr., Estevam R. Hruschka, and Mitchell, Tom M. Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI 2010)*, 2010.

Fader, Anthony, Soderland, Stephen, and Etzioni, Oren. Identifying Relations for Open Information

Extraction. In *EMNLP*, 2011.

Hubert, Lawrence and Arabie, Phipps. Comparing Partitions. *Journal of Classification*, 2(1), 1985.

Kang, U., Tsourakakis, Charalampos E., and Faloutsos, Christos. Pegasus: A peta-scale graph mining system implementation and observations. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ICDM '09, pp. 229–238, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3895-2. doi: 10.1109/ICDM.2009.14. URL http://dx.doi.org/10.1109/ICDM.2009.14.

Kipper, Karin, Dang, Hoa Trang, and Palmer, Martha. Class-Based Construction of a Verb Lexicon. In *Proceedings of AAAI 2000*, 2000.

Lin, Frank and Cohen, William W. Power iteration clustering. In *Proceedings of ICML 2010*, 2010a.

Lin, Frank and Cohen, William W. A very fast method for clustering big text datasets. In *Proceedings of ECAI 2010*, 2010b.

Pereira, Fernando and Tishby, Naftali. Distributional Similarity, Phase Transitions and Hierarchical Clustering. In *Working Notes, AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, pp. 108–112, 1992.

Zaharia, Matei, Chowdhury, Mosharaf, Das, Tathagata, Dave, Ankur, Ma, Justin, McCauley, Murphy, Franklin, Michael, Shenker, Scott, and Stoica, Ion. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. Technical Report UCB/EECS-2011-82, EECS Department, University of California, Berkeley, Jul 2011. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-82.html.