Trade-offs of Heuristic Vs. Rigorous Algorithms in Text Mining

Andrew Matheny
LCSEE, West Virginia University
andrew.j.matheny@gmail.com

Abstract

This paper assesses the cost of using heuristic methods in the field of text mining. Previous research has shown many of the formal non-heuristic algorithms to be NP-hard which positive results only in small domains. Given the massive scale when dealing with unstructured textual data, these algorithms prove to be impossible with large datasets with many dimensions. Many heuristic approximations have been proposed that drastically improve run-times, but give a result of less accuracy than the rigorous methods.

In this paper, we evaluate the trade-offs of heuristic methods to their more computationally complex alternatives. We focus on algorithms for document clustering and dimensionality reduction, and provide results detailing the run-times and cluster validity of each method on its own along with combinations from each type (Rigorous and heuristic). Our findings indicate that the cost of these approximations vary when dealing with supervised vs unsupervised datasets, and provide recommended parameters and combinations for optimization.

1. Introduction

In the vast tombs of the National Archives and Records Administration, there lies a significant amount of information. Among all of the different categories of information from neatly organized databases, to cluttered hard drives, to post-it notes stuck to monitors; the most common type of information is unstructured text. Merrill Lynch estimates that this represents around 85 percent of all business information is stored in this manner [8]. This can be anything from e-mails, chat logs, web sites, white papers and proposals... the list goes on and on. Given such a large amount of scattered, seemingly useless data, is it possible to make sense of it all? Is it possible to make sense of it all in a reasonable amount of time?

This has been a significant question posed by scientists from computer science, statistics, linguistics, and mathematics since the mid 1980s and within the last 10 years has seen tremendous growth. Within the realm of making sense of unstructured text (text mining) there have been numerous methods proposed to help classify unknown documents and learn patterns from collections of many documents. These algorithms typically fall into two rough categories, heuristics and rigorous (NP-Hard in come cases). The rigorous approach is usually thought of as the most complete solution, but taking the longest time (by leaps and bounds). On the other hand the heuristic approach will typically complete in a much smaller amount of time, but can only do so by cutting corners and taking advantage of easy to compute approximations.

Given the massive scale of unstructured information, it is nearly always impractical to use the Rigorous approach. This has resulted in a wide spread adoption of heuristic approximations, simply out of necessity. Interestingly enough, these approaches have been shown to prove quite useful and any loss in quality of the result has come to be acceptable. The question we propose is; How much is lost by taking heuristic short cuts and what can be done to minimize this loss. In this paper we will focus on the tasks of dimensionality reduction and document clustering, with 3 algorithms for each task, with one from each category being rigorous and the other 2 heuristic. We hope to find an acceptable trade-off between run-time and quality of results and to offer ways to achieve the quality of the rigorous algorithms with the run-times of the heuristics.

2. Domain Specifics

2.1. Term and Document Matrix

The standard unit of classification within text mining is a document. A document can represent a number of things but most specifically, it is a collection of terms pertaining to some entity. When analyzing a collection of documents, this information is stored as a term document matrix. In this matrix, documents are rows and the terms that contain are the columns. Any place in the matrix where document d

contains term t some value is stored which can either be binary (in this case, we only need to know if it was in the document) or real-valued (when we care about the frequency of occurrence of term t in document d). For example, the phrase:

$$\begin{array}{c} The\ quick\ brown\ dog\ was\ very \\ quick,\ very\ brown,\ and\ very\ dog\ like. \end{array} \quad (1)$$

Will be turned into a vector which looks something like this:

$$Phrase = [1\ 2\ 2\ 2\ 1\ 3\ 1\ 1] \quad (2)$$

with each index of the above vector corresponding the a dimension which comes from the term list (in this case, the dimensions are the, quick, brown, dog, was, very, like).

Tf*Idf

Tf*Idf is shorthand for "term frequency times inverse document frequency." This calculation models the intuition that jargon usually contains technical words that appear a lot, but only in a small number of paragraphs. For example, in a document describing a space craft, the terminology relating to the power supply may appear frequently in the sections relating to power, but nowhere else in the document.

Calculating Tf*Idf is a relatively simple matter:

- Consider the frequency of occurrence of term i within document j, $tf_{i,j}$

$$tf_{i,j} = \frac{n_{i,j}}{\sum_{t \in T_j} n_{t,j}} \quad (3)$$

  Where $n_{i,j}$ is the number of times term i occurs in document j, and T_j the set of terms in document j.

- Now consider the frequency of occurrence of term i within the entire collection of documents.

$$idf_i = log \frac{|D|}{|\{d : t_i \in d\}|} \quad (4)$$

  Where D is the set of all documents and $\{\text{d: } t_i \in \text{d}\}$ is the set of all all documents that term i appears in.

- If we simply multiple eq. 3 by eq. 4 we have the the TfIdf value of term i in document j.

2.2. Dimensionality Reduction

Standard AI and machine learning methods work well for problems that are fully described using dozens (or fewer) attributes [?]. But a corpus of text documents must process thou- sands of unique words, and any particular document may only mention a few of them [?, ?]. Because of this high natural dimensionality, run-times suffer vastly when working with the entire set of terms. Additionally, the curse of dimensionality tells us that as the number of dimensions are increased, the volume of all possible values increases exponentially. Therefore, before we can apply learning to textual documents, we must intelligently reduce the number of dimensions to a manageable level, while still best preserving the information in the original space.

In addition to the dimension reduction algorithms analyzed in this paper, there are a number of other linear time methods pertaining solely to sparse textual datasets such as Porters' stemming algorithm, stoplists, tokenization, etc. While these methods have been shown to be useful in the past, this paper is focused solely on the tasks of clustering and dimesion reduction and they will not be examined.

2.3. Document Clustering

One of the primary goals in document classification, is to index large amounts of unstructured information and ask the question, "What documents are similar to this design?" In order be able to answer this question we must first find an appropriate way of locating the "structure" in these collections; including (a) the structure that exists within each document as part of the collection, and (b) the comparative structure of all the documents and how they relate to one other. Considering that each document type has a format that can vary from highly syntactic source code, to badly formed HTML, to unstructured text, there is not much common ground between the various document types. One common theme in all of these documents is exactly what you are looking at right now, natural language. The challenge of document is solving (b), or how to define the comparative structure of the documents and how they relate to each other.

2.4. Why Heuristics?

In the field of text mining, the sheer scope is one of the largest challenges. The table above (fig. 1) above shows the size of each dataset in our study by the number of terms and number of documents. Working with datasets of this magnitude at their originally dimensionality is certainly out of the question leaving dimensionality reduction as an absolute requirement. Using the traditional SVD-based methods

¡¡need to add chart here showing sizes of term document matrices ¿¿

Figure 1. Sizes of datasets used in this study. Notice the massive inherent dimensionality (large term counts).

require the T X T covariance matrix which takes an exponentially larger amount of time as the number of terms is increased. The same arguments can be used for arguing against clustering with K-Means. At each iteration, K-Means has a complexity of $O(k * n * t)$ where k is the number of clusters requested, n is the total number of documents, and t is the total number of terms. While these algorithms may be appropriate in toy domains, they are simply impossible in real-world applications such as program comprehension, news story aggregation, analysis of medical reports, fraud detection, etc. where new information is in constant flow and the existing data is already large enough.

Given the scale and scope of the data that is being used industry, heuristic methods are a necessity and as a result have been in use for some time. The algorithms being analyzed here are nothing that has not seen before. However, what has not been seen before is the analysis of what (if any) is lost when going to heuristic methods. Furthermore, what optimizations can be done to these algorithms to provide the most bang for the buck.

2.5. Rigorous Algorithms

2.5.1 Dimensionality Reduction

.

PCA

Numerous data mining methods check if the available features can be combined in useful ways. These methods offer two useful services:

1. Latent important structures within a data set can be discovered.
2. A large set of features can be mapped to a smaller set, then it becomes possible for users to manually browse complex data.

For example, principal components analysis (PCA) [3] has been widely applied to resolve problems with structural code measurements; e.g. [7]. PCA identifies the distinct orthogonal sources of variation in a data sets, while mapping the raw features onto a set of uncorrelated features that represent essentially the same information contained in the original data. For example, the data shown in two dimensions of fig. 2 (left-hand-side) could be approximated in a single latent feature (right-hand-side).

Since PCA combines many features into fewer latent features, the structure of PCA-based models may be very simple while still providing information pertinent to the original vector space.

PCA is one of the traditional methods of performing dimensionality reduction. It suffers from scale-up problems (for large data sets with many terms, the calculation of the correlation matrix between all terms is prohibitively computationally expensive).

2.5.2 Clustering

.

K-Means

K-Means is a clustering algorithm that, when given a dataset of unidentified objects, it will group those items into k groups based on some given similarity measure (cosine in our case). The algorithm is described in fig. 3. For an example of the algorithm in operation, see fig. 4.

2.6. Heuristic Algorithms

2.6.1 Dimensionality Reduction

.

FastMap

FastMap, proposed by Faloutsos circa 1995 [4], is similar in nature to PCA in that it synthesizes new features by combining all of the original features. Where FastMap different from PCA is in run-times (as the name may imply). The general goal of FastMap is to project items in a n dimensional to a d dimensional space, with $n > d$. FastMap works by recursively a new dimension, d times.

The basis of each reduction is using the cosine law on the triangle formed by an object in the feature space and the two objects that are furthest apart in the current (pre-reduction) space. These two objects are referred to as the pivot objects of that step in the reduction phase (n - d total pivot object sets). Finding the optimal solution of the problem of finding the two furthest apart points is a N^2 problem (where N is the total number of objects), but this is where the heuristic nature of FastMap comes into play.

Instead of finding the absolute furthest apart points, FastMap takes a shortcut by first randomly selecting an object from the set, and then finding the object that is furthest from it and setting this object as the first pivot point. After the first pivot point is selected, FastMap finds the points farthest from this and uses it as the second pivot point. The line formed by these two points becomes the line that all of the other points will be mapped to in the new n - 1 dimension

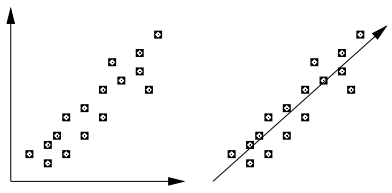


Figure 2. The two features in the left plot can be transferred to the right plot via one latent feature.

- i=0
- Partitioning the input points into k initial sets, either at random or using some heuristic data.
- Repeat unitl ($i \leq maxIterations$ or no point changes set membership)
  - Calculates the mean point, or centroid, of each set or cluster.
  - Constructs a new partition, by associating each point with the closest centroid.
  - Recalculate the centroids for the newly partitioned cluster
  - i = i + 1

Figure 3. K-Means algorithm. See fig. 4 for an example of this algorithm running in practice.

space.

FastMap uses fig. 5 to calculate x_i, or the position of object O_i in the reduced space. This technique can be visualized by imagining the hyperplane perpendicular to the line formed by pivot points, O_a and O_b, and projecting the new point onto this plane (fig. 6).

TfIdf Sorting

2.6.2 Clustering

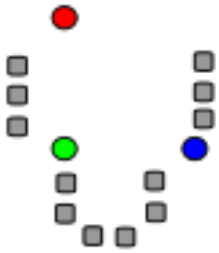
.

Canopy Clustering

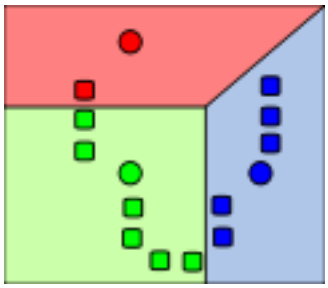
A naive clustering algorithm runs in $O(N^2)$ where N is the number of terms being clustered and all terms are assessed with respect to all other terms. For large archival collections, this is too slow. Various improvements over this naive strategy include ball trees, KD-trees and cover trees [1]. While all these methods are useful, their ability to scale to very large examples is an open question.

An alternative to traditional clustering methods is *canopy clustering* [2, 6]. It is intended to speed up clustering operations on large data sets, where using another algorithm directly may be impractical because of the size of the data set. In a standard clustering algorithms, two items are compared to determine some measure of how similar or different they are. There are several distance measures used for different domains (euclidean, cosine, manhattan, etc.), the draw back to all of these is that they are all relatively computationally expensive. The secret to canopy clustering's greater performance over conventional clustering techniques is it's

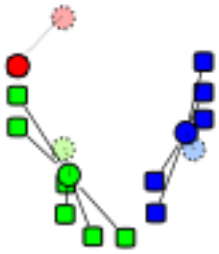
Step1: Here, we show some initial data points and the centroids generated based on random assignment



Step2: Points are associated with the nearest centroid:



Step3: Next, we recompute centroid using new associations and update the stored centroid:



Steps 2 & 3 are repeated until one of the two convergences criteria are reached.

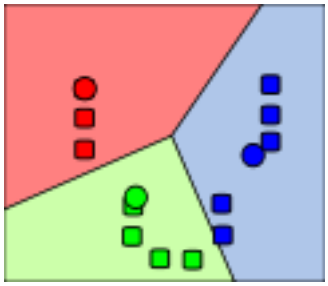


Figure 4. Example of K-means

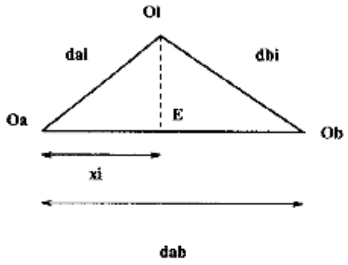


Figure 5. Example of using the cosine law to find the position of Oi in the dimension k

$$x_i = \frac{d_{a,i}^2 + d_{a,b}^2 - d_{b,i}^2}{2d_{a,b}} \tag{5}$$

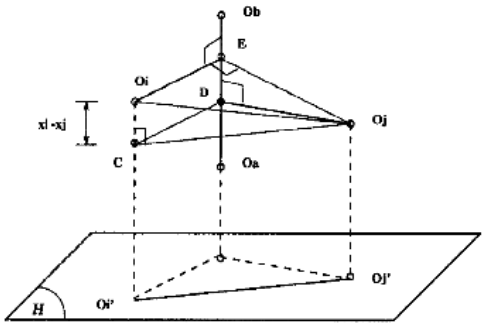


Figure 6. Projects of points O_i and O_j onto the hyperplane perpendicular to the line O_aO_b

use of two distance measures, one being approximately accurate but computationally cheap and the other being more accurate, however more expensive. To take advantage of the cheap distance metric, two passes are taken over the dataset. In the first pass, the cheap distance measure is used to determine *canopies*, which are groups of approximately close things. In the second pass, the more expensive distance measure is used. If any two items being compared do not share a canopy, then their distance is assumed to be infinite and no further comparison is done. By doing this, canopy clustering prevents having to perform n^2 comparisons at each step through the clustering algorithm.

The algorithm proceeds as follows:

- Cheaply partition the data into overlapping subsets, called 'canopies' (see fig. 7);
- Perform more expensive clustering, but only between these canopies.

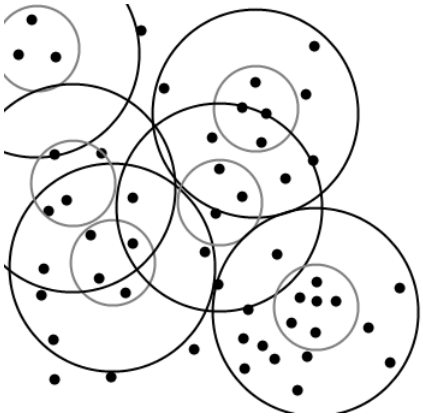


Figure 7. The darker circle represents all points in a given canopy, points in the smaller circles cannot be used as a new canopy center.

In the case of text mining applications like HAMLET, the initial cheap clustering method can be performed using an inverted index; i.e. a sparse matrix representation in which, for each word, we can directly access the list of documents containing that word. The great majority of the documents, which have no words in common with the partial design constructed by the engineering, need never be considered. Thus we can use an inverted index to efficiently calculate a distance metric that is based on (say) the number of words two documents have in common.

GenIc

GENIC is a generalized incremental clustering algorithm developed by Gupta and Grossman [5] that provides potentials for large improvements in scalability over K-Means. Since GENIC was designed with streaming data in mind, it only has a single pass through the data to work with. Because of this, it scales linearly, which is a requirement when dealing with large corpora. By using stochastic methods, GENIC can be given an initial k equal to the number of items (each item is its own clusters) and prune away unlikely clusters with each generation, giving a realistically estimated value for k after the last generation. Here is how GENIC works:

1. **Select parameters**
   - Fix the number of centers k.
   - Fix the number of initial points m.
   - Fix the size of a generation n.
2. **Initialize**
   - Select m points, $c_1, ..., c_m$ to be the initial candidate centers.
   - Assign a weight of w_i = 1 to each of these candidate centers.
3. **Incremental Clustering** For each subsequent data point p in the stream: do
   - $Count$ = $Count$ + 1
   - Find the nearest candidate center c_i to the point p
   - Move the nearest candidate center using the formula
   $$c_i = \frac{(w_i * c_i + p}{w_i + 1} \quad (6)$$
   - Increment the corresponding weight
   $$w_i = w_i + 1 \quad (7)$$
   - When $Count \ mod \ n$ = 0, goto Step 4
4. **Generational Update of Candidate Centers** When $Count$ equals $n, 2n, 3n$, ..., for every center c_i in the list L of centers, do:
   - Calculate its probability of survival using the formula
   $$p_i = \frac{w_i}{\sum_{i=1}^{n} w_i} \quad (8)$$
   - Select a random number δ uniformly from [0,1]. If p_i ¿ δ, retain the center c_i in the list L of centers and use it in the next generation to replace it as a center in the list L of centers.
   - Set the weight w_i = 1 back to one. Although some of the points in the stream will be implicitly assigned to other centers now, we do not use this information to update any of the other existing weights.
   - Goto step 3 and continue processing the input stream
5. **Calculate Final Clusters** The list L contains the m centers. These m centers can be grouped into the final k centers based on their Euclidean distances.

GENIC is of specific interest to HAMLET for two primary reasons, low expected run-times on large corpora and a potential ability at estimating the number of natural clusters in the collection.

- Scalability: Since GENIC was designed with streaming data in mind, it only has a single pass through the data to work with. Because of this, it scales linearly, which is a requirement if HAMLET is to scale to large corpora.
- An likely estimate for k: Because of GENIC's stochastic based method of removing unwanted or non-useful clusters, it has potential for use in correctly estimating a good value for k. By eliminating "bad stuff", GENIC can ideally identify the correct number of types of "good stuff".

2.7. Current Benchmarks

3. Analysis

3.0.1 Experiment Design

3.0.2 Datasets

3.0.3 Clustering Results

3.0.4 Dimension Reduction Results

3.0.5 Combinations

4. Conclusion

References

[1] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML'06*, 2006. Available from `http://hunch.net/~jl/projects/cover_tree/cover_tree.html`.

[2] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *KDD '02: Proceedings of the eighth ACM*

SIGKDD international conference on Knowledge discovery and data mining, pages 475–480, New York, NY, USA, 2002. ACM.

[3] W. Dillon and M. Goldstein. *Multivariate Analysis: Methods and Applications*. Wiley-Interscience, 1984.

[4] C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In M. J. Carey and D. A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174, San Jose, California, 22–25 1995.

[5] C. Gupta and R. Grossman. Genic: A single pass generalized incremental algorithm for clustering. In *In SIAM Int. Conf. on Data Mining*, pages 22–24. SIAM, 2004.

[6] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178, New York, NY, USA, 2000. ACM.

[7] J. C. Munson and T. M. Khoshgoftaar. The use of software complexity metrics in software reliability modeling. In *Proceedings of the International Symposium on Software Reliability Engineering, Austin, TX*, May 1991.

[8] C. C. Shilakes and J. Tylman. Enterprise information portals, 1998.