Unsupervised of Clustering Web Articles

Victor Ardulov

Fall 2016

Abstract

Recently web articles have become an increasingly popular medium to share innovation, and news across all forms of communication. Typically over time, most individuals accrue a set of articles they are interested in, and would want to go back to in the future. While various forms of bookmarking are available, most of the bookmarked articles have common overlapping themes and topics making regular queries miss important underlying subtleties in the text. One way to circumvent this problem is to manually label the data into topical categories, but can often prove to be tedious, again due to the overlapping topics, and the sheer volume of content.

This project explores the ability of techniques in modern AI and their ability to find underlying structure by analyzing article information and then storing the results as a vectors of indexed features and the link. The hope is that in the future this system can be used in conjunction to a search engine to make finding topically relevant content in the future easier.

1 Introduction

One of the many focuses within the realm of Artificial Intelligence(AI), and Machine Learning(ML) is the attempt to find underlying structure that is other wise missed in data. Typically though, in order to do this large repositories of data need to be available for the AI and ML to really be practically beneficial. The nuance lies in that finding structure in data that is labeled is easier, since relationships in the data are more clearly defined, however, finding large amounts of labeled data is difficult.

Unsupervised learning is the branch of ML which attempts to find structure in data without having a priori knowledge about the classes to which the data belongs. In essence unsupervised learning is a method of finding correlations in "input" data without knowing the "output", and assuming that similar "inputs" will have similar nature and "outputs".

One such example of large swaths of data are web articles, which have become a ubiquitous method to sharing interests and ideas between individuals. While key terms, and high level topics (e.g. economics, technology, news) are typically added, individuals will usually have many articles in one category with few from others. The issue is then keeping track of, and managing articles that all fall under a similar over-arching topic.

The discussion of this project will focus on the ability to use simple AI techniques to find underlying structure in large amounts of unlabeled data, as a means to identify subtle

trends, and topics in web articles, with similar over-arching themes. Below the discussion will cover different methods of data representation, from simple formulaic constructions, to learned structure, and compare the advantages and short-comings each. Finally, we will discuss unfinished components of the project that should be elaborated in the future, to improve the performance of the work done so far.

2 Background

Unsupervised clustering of text has been a widely studied field. The standard gauge on the performance for topic extraction in text, has been accuracy of sorting the Reuters classification data set. Both Apt. et al. [1] and Masand et al. [2] were early looks at the ability of machine learning to identify and categorize text data. A major component that grew from this were different methods of representing data and training models. This research was strongly tied to the belief that categorization of text should be done independently of the language. Instead they propose methods to represent text as a vector of features.

2.1 Text-Representations

There exist several different way to represent data, many of which are discussed in Trinkle [3]. Of those this project will compare strict term frequency (tf), term frequency inverse document frequency (tf-idf) and also introduce 2 more tf-idf based features: self-taught frequency features (stff), and deep self-taught frequency features (d-stff).

- **tf** - strict term frequency is often used by search engines to identify relevancy to a query. tf will convert the text into a vector where each entry represents a term and the value of the entry represents the raw number of times the term appears in the article.
- **tf-idf** - term frequency inverse document frequency is a method to penalize terms that are ubiquitous to the data. It does this by normalizing the value of a vector entry by the number of documents that term has been seen.
- **stff** - by attempting to approximate the identity function by compressing and decompressing the feature vector, we can then use the compression stage of our neural network to compress our data's feature space making the computation of the centroids in the lower dimensions faster[4].
- **d-stff** - this is an identical feature representation, except the training model has many more hidden layer and uses a deep learning neural network to compress and decompress through multiple layers.

The comparison comes down to show that simply using tf will lose meaningful contextual data that tf-idf will not because of the nature of the data, and vectorization methods. The self-taught features are an attempt to perform a Principle Component Analysis (PCA) using an autoencoder to find a dimensionally smaller mapping of the features, and reduce out the size of the data vectors. By training a autoencoder, and then clustering the dimensionally

reduced subspace, it will also reduce the amount of computation necessary during the execution of the actual clustering as there are less features that need to be compared and for which distances needed to be computed.

Another modification to the text-representation is that in Trinkle, n-gram representation is listed separately [3], however for this project the decision was made to incorporate the n-grams as terms in the feature space. The hope is that the n-grams will capture more structure that would be lost at just evaluating individual terms. For example a 2-gram phrase may appear less frequently than one or both of the words that appear separately, but the 2-gram phrase could prove to be more telling about the what cluster a document belongs to.

Much of the work in this project will be about the methods to represent the data. Throughout the progression of this project it has been clear that finding an appropriate feature space is great importance, and as such has been strongly emphasised in the work presented.

2.2 Clustering and Classification

Once the data is presentable there should be a means by which the to extract topics that link the data. The class of Unsupervised learning algorithms that perform this fall into the realm of clustering. One of the most common methods method presented in the literature was KMeans, which attempts to assign the data to clusters that a spatial correlated to one-another.

Another common branch of clustering algorithms are hierarchical-clustering algorithms, or agglomerative clustering, attempts to use similarity between smaller clusters to combine them until either there is one all encompassing cluster, or some minimum number of clusters.

Ultimately the idea was to move away from all encompassing terms, therefore many of the tests focus on KMeans for evaluating performance on text-representation.

It was found, that one of the benefits of the KMeans tf-idf, and KMeans tf clustering approaches was that it generated concrete "centroids", for which the arguments could be sorted. The sorting of the arguments, could then be used to generate the most "important" frequent features in the centroids. For the other forms of clustering and text representation, extra steps had to be taken to reconstruct and approximation. Namely the vectors of the clusters were averaged, and the components with the largest averages within the cluster were asserted as the clusters "key words". It should be noted that the reconstruction is an attempt to make sense of clustering and text representation which is not based on a central point or where the data is not mapped directly to the original features. Ultimately the hope is to make the results of the clustering human understandable.

3 Methods

This project was an attempt to compare and contrast different methods of Unsupervised Learning and data representation for text documents. As such the methods for the project represent a variety of tests that were utilized to represent the strengths and weaknesses of each method discussed.The discussion will follow the steps taken and rationales behind why certain methods and tests were run.

3.1 Data Acquisition and Cleaning

In order to run these tests and make conclusions on the results, a data set that was representative of real life situation was necessary. Namely, the constraints were that the articles should cover a variety of "subtopics" within a common domain.

In order to reflect this `lxer` was chosen as a repository of Linux news, source of web-articles. By utilizing the python library `BeautifulSoup4` and looking at the structure of the `lxer` website, 1000 link, and titles were scraped off of the website and stored in a JSON format.

Once there were links the links were processed using another Python library `Newpaper`. The library utilizes Natural-Language Processing (NLP) and a `BeautifulSoup4` back-end to extract information from news articles. It turned out that many of the articles linked, were in the style of a forum post, which for which the text and title was handled but extracting meta-data like authors proved to be difficult. The 1000 articles only 919 could actually be used to train and 82 were unreachable or "unscrapable". All of these articles can be found in `train_data.json`.

Once the data is aquired it needs to be constructed into a vector, for this the NLP toolkit Python library `nltk` was utilized. `Nltk` allows us to define stopwords (i.e. the words that do not add meaning to the text), like pronouns, conjugation of common verbs, etc. were defined and are removed from the text. Then n-grams are composed and the tf or tf-idf vector representation of the texts are computed and constructed in to a matrix.

In the case of stff and d-stff the tf-idf vectors are run through an autoencoder or deep autoencoder respectively. Using the `Keras` library with a `Tensorflow` backend, 2 nueral networks were trained hopin the weights weights on compression will approximate a PCA, and successfully reduce the dimensionality of your data, without compromising the clustering of your data. To train the autoencoders, the euclidean distance was minimized between the input and output. It was found that the convergence of the loss, occurs after approximately 250 epochs of training.

With the data cleaned and prepared, we are now ready to build our models.

3.2 Training

To train our models, first an ideal number of clusters needs to be chosen. The literature implies many different methods, and `lxer` inherently uses 21 different topics to sort their articles, but this runs the risk of being an over fit. Since the KMeans minimized over euclidean distance, the goal was to keep adding cluster sizes until the return on investment per cluster did not add significant improvement on the average distance between the data and its assigned cluster.

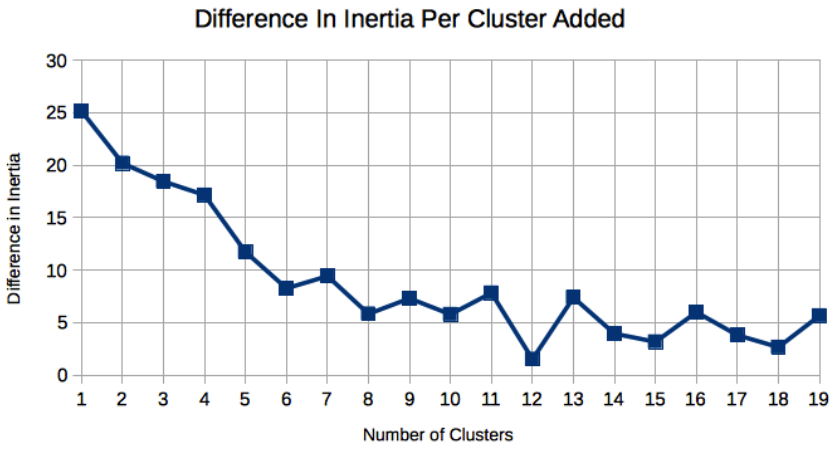

Figure 1: Change in inertia as result of adding another cluster centroid

As observed from the figure above, the reutnr on investment flattens out around 7 clusters. For that reason 7 was the number of clusters tested in all of the following simulations.

Reconstructing key terms for the other methods of clustering and the encoded feature vectors, requires some extra processing of the output, and was omitted for brevity of the discussion as time was constrained.

4 Results

4.1 Generating Cluster Key Terms

To make sense of the data that was clustered, we wanted some insight into the words and terms that were most important to the results. To do this for tf, and tf-idf the centroids that were generated by KMeans could be sorted in order of dominance, and then output. This produces human understandable outputs that make gauging the overall success of the clustering algorithm a simpler task.

In principle something similar could be done by remapping the centroids of the autoencoded vectors, which is done in `comparing_clustering.py` with varying degrees of success.

4.2 Example of tf Clusters

Cluster 0	linuxes using releasing new supporters installed development including update opens	Cluster 1	iot opens linuxes source supporters devices runs projection home
Cluster 2	using linuxes opens working projection likings development source secure users	Cluster 3	installed server using configurability file command creat users new directory
Cluster 4	using file command installed creat runs linuxes needed applicants makes	Cluster 5	opens source projection development using community softwares company working
Cluster 6	softwares freeing bugs using reportedly users opens source makes codes		

4.3 Example of tf-idf Clusters

Cluster 0	releasing update desktop new supporters linuxes version installed improvement ubuntu	Cluster 1	cloud iot red hat containment linuxes ubuntu enterprise supporters
Cluster 2	kernel linuxes patch releasing bugs update drivers secure using	Cluster 3	installed command file sudo using server configurability directory linuxes creat
Cluster 4	board port modules clicked images usbs supporters including interfacing offered	Cluster 5	using working secure times linuxes softwares likings needed makes years
Cluster 6	opens source projection community softwares development foundational linuxes using		

4.4 Cluster Correlation

As observed from the 2 tables above there is overlap between the 2 methods, however in general the tf key terms are much more broad, and get less at the subtleties of the data. For example the term "linux" appears highly ranked (top 3) in many of the tf key terms, while it only shows up top 3 for 1 cluster in the tf-idf cases.

Similarity tests were also run between all of the different models, that tested the correlation (in these tests average overlap) between clusters:

```
f cluster correlations to tf-idf:
{0: (0, 87.16216216216216), 1: (0, 64.51612903225806),
 2: (0, 57.03125), 3: (0, 67.12328767123287),
 4: (3, 36.36363636363637), 5: (6, 74.19354838709677),
 6: (5, 73.91304347826087)}
autoencoder correlation to tf-idf:
{0: (0, 25.0), 1: (2, 24.926686217008797),
 2: (4, 35.15625), 3: (4, 28.767123287671232),
 4: (3, 42.97520661157025), 5: (2, 50.0),
 6: (0, 58.69565217391305)}
deep autoencoder correlation to tf-idf:
{0: (0, 32.432432432432435), 1: (1, 26.686217008797655),
 2: (6, 50.0), 3: (4, 23.28767123287671),
 4: (5, 48.760330578512395), 5: (3, 62.903225806451616),
 6: (2, 60.869565217391305)}
agglomerative correlation to kmeans tf-idf:
{0: (1, 83.10810810810811), 1: (0, 79.17888563049853),
 2: (4, 81.25), 3: (2, 47.945205479452056),
 4: (0, 47.93388429752066), 5: (6, 90.3225806451613),
 6: (5, 86.95652173913044)}
```

The labels in the dictionaries above are the cluster number and the entries represent the maximum correlated cluster for the method compared, and the percent overlap (i.e. what percentage of the texts from that cluster ended up in the entry's cluster). Its clear from the data that the autoencoded representation have very weak correlation to the tf-idf mappings showing weak performance on the resulting clusters most likely. In general agglomerative and tf were respectively similar to the KMeans, but previous tests suggested that KMeans was not stable on the data, as there would often be articles that moved classifications between different fits of the data, this suggests that there are many "edge" cases and that 7 is not in fact the most optimal k value, or that the data has many outliers that do not fit well with the rest of the data.

5 Conclusion

It's clear that tf-idf and KMeans produce tangible results as observed by the cluster keywords. The conclusion is that it seems like the data is too scarce for a true manifolding technique to prove useful, but dense enough that simple tf vector representations are not enough to represent the output. This is seen in the overlap results seen in the correlation test, as the dimensionally reduced vectors map maintain clustering poorly, and the clustering resulting clusters are moved around a great deal.

6 Future Work

In the future it seems clear that larger models should be examined, it seems that 1000 articles is too scarce to truly understand the structure lying within the data. It would also be interesting take a look at some more modern clustering algorithms. Primarily they are:

6.1 Probabilistic Mixture Models

One innovation that warrants a closer look is the attempt to remove outliers, and find overlapping classifications for border cases. Since there are many edge cases, it would be better to look at a Gaussian Mixture Model to better understand the classification of the text examined.

Additionally adding something analogous to The Chinese Restaurant Method (CRM) might also find a more optimum solution to the classifications. CRM is an intelligent way using EM to determine whether or not a new cluster is necessary to fit the data to. This will converge on an optimal Km

6.2 Embedded Clustering Schemes

Namely Spectral Clustering, and Deep Embedded Clustering [5] are recent proven methods to improve Unsupervised Learning by clustering, yielding much stronger results over the KMeans and Agglomerative algorithms for

6.3 Down Stream Classification

Early methods were attempted however there some inabilities met when trying to vectorize and map terms previously unseen by the vectorizer and KMeans caused errors that were ultimately unresolved due to time constraints. In the future though a downstream classifier which can compare be used to compare methods, and see not only how they compare with one another but also how they compare with human intuition about the data.

Once the clusters were identified, the model can be used to get human input for another subset of articles to label them into the cluster categories that were generated by the KMeans model and then compare them to the predicted outputs of the model.

Acknowledgements

I extend my thanks to my TA Pujun for the consistent feedback and support through this project, Brandon Rose [6] for the inspiration behind much of the approach and methods in this project, and Stanford Center for Professional Development for granting me the opportunity learn by doing.

References

[1] C. Apt, F. Damerau, and S. M. Weiss, "Automated learning of decision rules for text categorization," *ACM Trans. Inf. Syst*, vol. 12, p. 233, 1994.

[2] B. Masand, G. Linoff, and D. Waltz, "Classifying news stories using memory based reasoning," in *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '92, (New York, NY, USA), pp. 59–65, ACM, 1992.

[3] P. Trinkle, "An introduction to unsupervised document classification," 2009.

[4] A. Ng, "Autoencoders and sparsity," 2016.

[5] J. Xie, R. B. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," *CoRR*, vol. abs/1511.06335, 2015.

[6] B. Rose, "Document clustering with python," 2016.