A2DI 2015/2016
Unsupervised text classification using spectral clustering algorithms

Rok Ivanšek

February 24, 2016

Abstract

For a project in the A2DI course I took in Lille 1 university I looked at how spectral clustering performs when applied to the domain of unsupervised text classification. In the first part of the report I will provide a quick summary of the spectral clustering algorithms presented in the article "A Tutorial on Spectral Clustering" by Ulrike von Luxburg. I implemented the algorithms in the python programming language and in the second part I will present the results of the testing I did. I used different "famous" text classification datasets and looked at how different variants of spectral clustering perform on them. I will also show some comparisons of efficiency with the traditional clustering algorithms, such as the K-means and hierarchical clustering.

Spectral clustering algorithms

For a more detailed presentation of algorithms used I suggest reading the original article by Luxburg as this is just a quick summary and a lot of details will be left out.

Introduction

In essence spectral clustering algorithms work by transforming the given object representations (samples) into some other representation that enhance the clustering properties of the data. Traditional clustering algorithms like the K-means are than used to cluster this new object representations. This performs much better than if, for instance, the K-means would be used directly on the original representations. The procedure of the transformation works the same for all the clustering algorithms.

- First we construct a similarity graph $G(V, E)$ of our choosing using some metric of similarity for our n samples in the dataset. We present the graph in a standard way with a degree matrix D and a weight matrix W. Both of the matrices are square $D, W \in \mathbb{R}^{n \times n}$. D is a diagonal matrix.
- Using matrices W and D we build one of the variants of the Laplacian matrix L.
- We than calculate k first eigenvectors of L, where k is the number of classes/clusters we wish to construct.
- We construct a matrix U that has the k calculated eigenvectors as columns.
- The rows of matrix U are now the new representations of original samples. Depending on the Laplacian matrix chosen, normalization of rows in matrix U is required before we use the clustering algorithm like K-means on the new representations.

The procedure differs only depending on which similarity graph and Laplacian matrix we choose and of course on the similarity metric.

All of the algorithms and different similarity graphs are implemented in the module "spectral.py". The spectral object is constructed by the call "spectral(X, labels, s, d)". Where X contains the attributes of the samples in the data set chosen. labels are ground truth labels for the samples, s is a similarity function and d is the distance function. The latter two should always be chosen in accordance with each other. If for instance the cosine similarity function is chosen then one should choose the cosine distance as well, anything else can lead to unexpected results.

Similarity graphs and Laplacian matrices

Given some samples from a dataset $x_1, \ldots, x_n$, a similarity graph $G(V, E)$ is a weighted graph, so a set of vertices and a set of edges with weights, where the latter are determined by some similarity metric $s(x_i, xj) - > s_{i,j}$, where $s \geq 0$. The graph is presented by a weight matrix W and a degree matrix D. The weights of the edges are stored in the matrix W. If $W[i, j] = w_{ij}$ is a non-negative value it means that the points x_i and x_j are connected and if its value is 0 they are not. A degree $D[i, i] = d_i$ of a data point x_i is defined as $d_i = \sum_{j=1}^{n} w_{ij}$. I tested the spectral clustering algorithms for the 4 following similarity graphs:

Fully connected graph

Here all data points (vertices) are connected with each other. The weights of the edges correspond to some similarity metric chosen. The construction is initialized by the method "spectral.full_graph(metric)" where metric is the chosen similarity metric. It can take "euclidean" and "cosine" as arguments for metric it than calculates the similarity values for all possible pairs of data points. For the argument "euclidean" the algorithm will use the Gaussian similarity as the similarity function and for argument "cosine" the cosine similarity.

The ϵ-neighborhood graph

Here only the points whose similarity is bigger than a certain threshold ϵ are connected. The construction is initialized by the method "spectral.eps_graph(eps)", where eps is the threshold.

k-nearest neighbour (non-mutual)

Here the vertex v_i is connected with the vertices that are k-nearest neighbours to v_j. According to the chosen similarity matrix. The construction of this graph is initialized by the method "spectral.kNN_graph(k, metric, False)", where k implies k-nearest neighbours, metric is a chosen distance metric according to which the nearest neighbours are calculated and False indicates that this is not a Mutual k-nearest neighbour graph.

Mutual k-nearest neighbour

Here the vertex v_i is connected with the vertex v_j only if v_i is among k-nearest neighbours to v_j and v_j is among k-nearest neighbours to v_i. The construction of this graph is initialized by the method "spectral.kNN_graph(k, metric, True)", where k implies k-nearest neighbours, metric is a chosen distance metric according to which the nearest neighbours are calculated and True indicates that this is a Mutual k-nearest neighbour graph.

Variants of Laplacian matrices

Once we construct one of the above mentioned similarity graphs and obtain the matrices W and D, we have to calculate the Laplacian matrix. The choice of the Laplacian matrix determines the variant of spectral clustering we are using. The original article mentions three Laplacian matrices.

- $L = D - W$
- $L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$
- $L_{rw} = D^{-1} L = I - D^{-1} W$

The first one is the so called unnormalized Laplacian matrix. The other two are the normalized Laplacian matrices. All of this matrices share convenient algebraic properties that have some nice implications in terms of clustering. This makes them particularly attractive. I will not explain and discus those here. For explanation refer to the original article by Luxburg.

Unnormalized spectral clustering

The algorithms presented in pseudocode in the next three sections are exactly as they are described in the original article by Luxburg.

This is the algorithm that uses the unnormalized version of the Laplacian matrix

$$L = D - W.$$

It follows all of the steps described in the beginning of this report except that it does not normalize the samples in matrix U. Also as said before the clustering algorithm used in the last step of the algorithm, before the actual output is arbitrary (we could choose another clustering algorithm instead of k-means.

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters to construct.

- Construct a similarity graph by one of the ways described in the last section. Let W be its weighted adjacency matrix.
- Compute the unnormalized Laplacian L.
- Compute the first k eigenvectors $u_1, \cdots, u_k$ of L.
- Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, \cdots, u_k$ as columns.
- For $i = 1, \cdots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
- Cluster the points $(y_i)_{i=1,\cdots,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \cdots, C_k$.

Output: Clusters $A_1, \cdots, A_k$ with $A_i = \{j | \, y_j \in C_i\}$.

The algorithm is run with the method "spectral.unnorm_sc(k)", where k is the number of clusters we wish to construct.

Normalized spectral clustering, random walk Laplacian

This is the algorithm that uses the normalized version of the Laplacian matrix

$$L_{rw} = D^{-1}L = I - D^{-1}W.$$

It does use the unnormalized version of the Laplacian matrix, but in fact uses the generalized eigenvectors of it which correspond to the eigenvectors of L_{rw}.

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters to construct.

- Construct a similarity graph by one of the ways described in the last section. Let W be its weighted adjacency matrix.
- Compute the unnormalized Laplacian L.
- Compute the first k generalized eigenvectors $u_1, \cdots, u_k$ of the generalized eigenproblem $Lu = \lambda Du$.
- Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, \cdots, u_k$ as columns.
- For $i = 1, \cdots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
- Cluster the points $(y_i)_{i=1,\cdots,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \cdots, C_k$.

Output: Clusters $A_1, \cdots, A_k$ with $A_i = \{j | \, y_j \in C_i\}$.

It is run with the method "spectral.norm_rw_sc(k)", where k is the number of clusters.

Normalized spectral clustering, symetric Laplacian

This is the algorithm that uses the normalized version of the Laplacian matrix

$$L_{sym} = D^{-1/2}LD^{-1/2} = I - D^{-1/2}WD^{-1/2}.$$

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters to construct.

- Construct a similarity graph by one of the ways described in the last section. Let W be its weighted adjacency matrix.
- Compute the normalized Laplacian L_{sym}.
- Compute the first k eigenvectors $u_1, \cdots, u_k$ of L.
- Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, \cdots, u_k$ as columns.
- Form the matrix $T \in \mathbb{R}^{n \times k}$ from U by normalizing the rows to norm 1, that is set $t_{ij} = u_{ij}/(\sum_k u_{ik}^2)^{1/2}$
- For $i = 1, \cdots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of T.
- Cluster the points $(y_i)_{i=1,\cdots,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \cdots, C_k$.

Output: Clusters $A_1, \cdots, A_k$ with $A_i = \{j | \, y_j \in C_i\}$.

As we see this is the only one out of the four algorithms that introduces the normalisation step. It is run with the method "spectral.norm_sym_sc(k)", where k is the number of clusters.

Application to text classification

The datasets

I tested the performance of before described algorithms on different text clustering/classification datasets. While browsing the internet in search of the appropriate datasets I stumbled upon Ana Cardoso Cachopo's site Datasets for single-label text categorisation that serves as a complement for her PHD thesis titled "Improving Methods for Single-label Text Categorization". She took some of datasets commonly used in the area of text clustering/classification and pre-processed them in a really neat way. All the details about the datasets and the exact pre-processing done can be found on the link provided.

For my testing I used the r8, r52 and webkb from Ana's site. I used the data that was intended as a test set for these datasets. I did this because the training datasets were too big to do the extended testing I had in mind in reasonable time. For all of these datasets I used the entirely processed "stemmed" sets versions.

Besides these three datasets I also used the 20newsgroups dataset. I downloaded the "bydate" version and did the proper parsing/preprocessing myself. Since the dataset if very large, I only used some of the samples for the set. I took:

- 100 samples from the comp.sys.ibm.pc.hardware category
- 100 samples from the comp.sys.mac.hardware category
- 100 samples from the rec.sport.hockey category
- 30 samples from the sci.med category
- 150 samples from the sci.space category
- 100 samples from the soc.religion.christian category
- 100 samples from the talk.politics.guns category

As we can see all of the categories are quite different accept the first two that would appear to use similar vocabulary. I also tried to use different number of samples for some categories. I tried to construct a very representative dataset, because this was the only dataset where I fine tuned the parameters of the algorithms to achieve a high score of prediction (more about this in the setup section). I also constructed another dataset containing the same number of different samples from the same categories to use as a test set.

Processing and construction of the tf-idf matrix

For each dataset I represented each document as a tf-idf vector for all the words that appear in a dataset. I used the raw frequency - inverse document frequency variant. So the i-th element of the vector v_j that represents the j-th sample (text document - d_j) from the dataset D with N samples is defined as

$$v_j[i] = tf(t_i, d_j) \cdot idf(t_i, D).$$

$tf(t_i, d_j)$ tells us how many times term t_i appears in the document d_j and
$idf(t_i, D) = \log \frac{N}{|d \epsilon D : t_i \epsilon d|}$ tells us the logarithm of the number of all documents divided by the number of documents that contain the term t_i. In this way i constructed the tf-idf matrix that had the tf-idf vectors as rows.

The procesing and tf-idf matrix generation was done with module "process.py" for the r8, r52 and webkb datasets and with the module "newsgroups20.py" for the newsgroups20 dataset.

Testing set-up

For the testing measured the score of prediction with three different metrics. The adjusted random index, the adjusted mutual information and the normalized mutual information.
I tested all of the 12 possible combinations of similarity graphs and spectral clustering algorithms for two different similarity functions (cosine and gaussian) with fixed parameters relative to the dataset size. I did this for the r8, r52 and webkb datasets to observe how well this combinations of similarity graphs and spectral clustering stack against each other. The goal was to find out which combination works best for which dataset. The algorithms were not tuned in any way and were only following some basic rules of thumbs for the parameters. For the 20newsgroups dataset I tried to obtain a high score of prediction by fine tuning the parameters on the train set and than evaluating it on the test set. As a benchmark I also evaluated some of the other traditional clustering methods on these dataset.

The parameters

These are the parameters I used when running the different combinations of algorithms on the r8, r52 and webkb datasets.

- Gaussian similarity: $\sigma = 1$
- Epsilon graph: ϵ chosen as the length of the shortest edge in a minimal spanning tree of the fully connected graph (this guarantees that the graph is connected)
- kNN graph (non-mutual): $k = \frac{n}{\log n}$ when I was using Gaussian similarity and $k = \frac{2n}{\log n}$ when using cosine similarity, where n is the number of samples
- mutual kNN graph: $k = \frac{2n}{\log n}$ when I was using Gaussian similarity and $k = \frac{3n}{\log n}$ when using cosine similarity, where n is the number of samples

By choosing this set-up, I had some way of uniformly setting the parameters across all three dataset.

Results

r8 dataset

- Gaussian similarity: n=2189, k=8, eps=0.4931, k(mutual kNN)= 569, k(kNN)= 284

	norm_sym_sc	norm_rw_sc	unnorm_sc
full graph	ARI: 0.2582 AMI: 0.4000 NMI: 0.4485	ARI: 0.1016 AMI: 0.3346 NMI: 0.3416	ARI: 0.0032 AMI: 0.0028 NMI: 0.0413
eps graph	ARI: 0.2575 AMI: 0.3988 NMI: 0.4473	ARI: 0.1016 AMI: 0.3346 NMI: 0.3416	ARI: 0.0032 AMI: 0.0028 NMI: 0.0413
mutual kNN	ARI: 0.3445 AMI: 0.4413 NMI: 0.5476	ARI: 0.3950 AMI: 0.4776 NMI: 0.5463	ARI: -0.0028 AMI: -0.0015 NMI: 0.0117
kNN graph	ARI: 0.2624 AMI: 0.3941 NMI: 0.4927	ARI: 0.2191 AMI: 0.3799 NMI: 0.4612	ARI: 0.2234 AMI: 0.3811 NMI: 0.4637

- Cosine similarity: n=2189, k=8, eps=8.5444e-07, k(mutual kNN)= 853, k(kNN)= 569

	norm_sym_sc	norm_rw_sc	unnorm_sc
full graph	ARI: 0.5691 AMI: 0.5487 NMI: 0.6341	ARI: 0.7548 AMI: 0.6654 NMI: 0.6801	ARI: 0.0032 AMI: 0.0028 NMI: 0.0413
eps graph	ARI: 0.5691 AMI: 0.5487 NMI: 0.6341	ARI: 0.7567 AMI: 0.6657 NMI: 0.6803	ARI: 0.0032 AMI: 0.0028 NMI: 0.0413
mutual kNN	ARI: 0.2697 AMI: 0.4084 NMI: 0.4547	ARI: 0.4197 AMI: 0.3931 NMI: 0.4716	ARI: -0.0004 AMI: 0.0004 NMI: 0.0251
kNN graph	ARI: 0.4320 AMI: 0.4596 NMI: 0.5537	ARI: 0.5069 AMI: 0.5121 NMI: 0.5951	ARI: 0.4570 AMI: 0.3416 NMI: 0.4719

- Using just kMeans directly on the tf_idf matrix (sklearn implementation of the algorithm, default parameters): ARI: 0.4029, AMI: 0.4861, NMI: 0.5418

r52 dataset

- Gaussian similarity: n=2568, k=52, eps=0.4931, k(mutual kNN)= 654, k(kNN)= 327

	norm_sym_sc	norm_rw_sc	unnorm_sc
full graph	ARI: 0.1121 AMI: 0.3425 NMI: 0.5358	ARI: 0.1140 AMI: 0.3290 NMI: 0.4872	ARI: 0.0091 AMI: 0.0038 NMI: 0.0813
eps graph	ARI: 0.0923 AMI: 0.3295 NMI: 0.5225	ARI: 0.1255 AMI: 0.3374 NMI: 0.4977	ARI: 0.0091 AMI: 0.0038 NMI: 0.0813
mutual kNN	ARI: 0.0717 AMI: 0.3033 NMI: 0.4947	ARI: 0.2393 AMI: 0.3453 NMI: 0.4840	ARI: -0.0033 AMI: -0.0013 NMI: 0.0644
kNN graph	ARI: 0.0702 AMI: 0.2911 NMI: 0.4784	ARI: 0.0734 AMI: 0.2785 NMI: 0.4600	ARI: 0.0871 AMI: 0.2917 NMI: 0.4743

- Cosine similarity: n=2568, k=52, eps=6.5576e-07, k(mutual kNN)= 981, k(kNN)= 654

	norm_sym_sc	norm_rw_sc	unnorm_sc
full graph	ARI: 0.1230 AMI: 0.3530 NMI: 0.5445	ARI: 0.3150 AMI: 0.4208 NMI: 0.4827	ARI: 0.0097 AMI: 0.0044 NMI: 0.0835
eps graph	ARI: 0.1229 AMI: 0.3573 NMI: 0.5500	ARI: 0.4369 AMI: 0.4175 NMI: 0.4913	ARI: 0.0097 AMI: 0.0044 NMI: 0.0835
mutual kNN	ARI: 0.0924 AMI: 0.3351 NMI: 0.5288	ARI: 0.2930 AMI: 0.4096 NMI: 0.4653	ARI: 0.0096 AMI: 0.0055 NMI: 0.0872
kNN graph	ARI: 0.1022 AMI: 0.3456 NMI: 0.5407	ARI: 0.4666 AMI: 0.4622 NMI: 0.5357	ARI: 0.2777 AMI: 0.2093 NMI: 0.3638

- Using just kMeans directly on the tf_idf matrix (sklearn implementation of the algorithm, default parameters): ARI: 0.2304, AMI: 0.3409, NMI: 0.4228

webkb dataset

- Gaussian similarity: n=1396, k=4, eps=0.4931, k(mutual kNN)= 385, k(kNN)= 192

	norm_sym_sc	norm_rw_sc	unnorm_sc
full graph	ARI: 0.2950 AMI: 0.2948 NMI: 0.3013	ARI: 0.2656 AMI: 0.2597 NMI: 0.2950	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056
eps graph	ARI: 0.2949 AMI: 0.2941 NMI: 0.3006	ARI: 0.2669 AMI: 0.2610 NMI: 0.2963	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056
mutual kNN	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056
kNN graph	ARI: 0.3274 AMI: 0.3119 NMI: 0.3200	ARI: 0.3055 AMI: 0.2857 NMI: 0.2880	ARI: 0.3114 AMI: 0.2896 NMI: 0.2928

- Cosine similarity: n=1396, k=4, eps=03.7741e-06, k(mutual kNN)= 578, k(kNN)= 385

	norm_sym_sc	norm_rw_sc	unnorm_sc
full graph	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056
eps graph	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056
mutual kNN	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056
kNN graph	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056	ARI: 0.0001 AMI: -0.0011 NMI: 0.0056

- Using just kMeans directly on the tf_idf matrix (sklearn implementation of the algorithm, default parameters): ARI: 0.3174, AMI: 0.3094, NMI: 0.3447

Newsgroups20 dataset

This is the only dataset where I fined tuned the parameters on the train set to obtain a better score on the test set. I ran the standard tests for standard parameters as with all other datasets. I then looked which combinations look promising and tried tuning the parameters for that combinations to obtain an even better score. The combination that turned out to work the best for this dataset (the scores below are for this combination) was the 60-NN (mutual) similarity graph paired with the symetric Laplacian using Gaussian similarity with $\sigma = 1$. I did not tune the parameters of other two methods used. They serve only as a benchmark.

- Train set
  - spectral clustering with some fine tuning: ARI: 0.5521, AMI: 0.6360, NMI: 0.6485
  - kMeans clustering (default values): ARI: 0.2264, AMI: 0.3737, NMI: 0.4093
  - hierarchical clustering (default values): ARI: 0.2373, AMI: 0.3518, NMI: 0.4517
- Test set
  - spectral clustering with some fine tuning: ARI: 0.4952, AMI: 0.5654, NMI: 0.5729
  - kMeans clustering (default values): ARI: 0.1177, AMI: 0.3079, NMI: 0.3812
  - hierarchical clustering (default values): ARI: 0.0925, AMI: 0.2413, NMI: 0.3386

Comments

As is evident from results of the testing there is no one general rule that can be deducted, concerning which method of spectral clustering works best on the domain of document classification/clustering. Cosine similarity usually performs better than Gaussian, although as we can see on the webkb dataset fails completely. In all cases the epsilon similarity graph produces good results and is only bested by some variant of kNN graphs in particular cases. As for the Laplacian matrices, the algorithms that use any of the two normalized variants of the Laplacian always perform better than the ones using the unnormalized variant. The norm_sym algorithm seams to work nicely with the Gaussian similarity as the norm_rw algorithm produces really good results when paired with the cosine distance, like in the case of r8 dataset where the highest score was obtained for this metric using eps graph combined with norm_rw algorithm. One thing that is evident from the tests though, is that spectral clustering performs well on the document classification/clustering problem and outperforms some of the traditional methods that were tested here.

As is the case with the majority of machine learning problems the choosing of the algorithm really depends on the problem itself and sometimes the trial-and-error seams the best way to go about solving it. Hopefully this report can serve as some kind of guide on where to start, to someone interested in using spectral clustering in the field of document classification/clustering.

References

[1] Ulrike von Luxburg, *A Tutorial on Spectral Clustering*, Statistics and Computing, 17 (4), 2007, available at http://www.kyb.mpg.de/fileadmin/user_upload/files/publications/attachments/luxburg06_TR_v2_4139%5B1%5D.pdf.

[2] Ana Cardoso Cachope, *Improving Methods for Single-label Text Categorization*, 2007, available at http://ana.cachopo.org/datasets-for-single-label-text-categorization

[3] *Scikit-learn library*, available at http://scikit-learn.org/stable/.