Variational methods - Course project

Dan Ben-David, Alexander Balabanov

April 2023

Contents

1 Introduction

In this report we discuss a paper [3] on the topic of graph clustering of data. Graph clustering can preform well with a small subset of the data labeled.

1.1 Spectral clustering

Spectral clustering is a powerful technique for clustering data points based on their similarity. The basic idea behind spectral clustering is to transform the data into a new space, where the clusters become more separable and it become easier to use clustering algorithm on the data. This transformation can be represented by a Laplacian graph, when the connection between the node correspond to similarity between the data points. then a similarity matrix is created base on the graph, and by extracting the the n must dominant eigenvectors of the matrix we can use them to define a low-dimensional embedding of the data, which is used as input to the clustering algorithm.

Let $X = \{x_i\}$; $i \in [1, N]$ be a set of N data points, with a known dimension $x_i \in \mathbb{R}^d$. Data point can be compared to each other with a weight function to create a $N \times N$ symmetric matrix that hold the relations between all data points.

$$W_{ij} = f(x_i, x_j) = W_{ji} \tag{1}$$

Function $f(x_i, x_j)$ in equation 1 can be any equation that takes 2 vector inputs $i, j \in \mathbb{R}^d$ and maps it to a none negative scalar $f(x_i, x_j) \geq 0$. Classically the weight function is a euclidean distance and a gaussian kernel, more on it in section 2.3.

Given a weighted adjacency matrix W [1] of an undirected graph, the Laplacian matrix L can be computed, where D is the degree matrix of the graph, defined as a diagonal matrix with D_{ii} equal to the sum of the weights of the edges incident to vertex i.

$$L = D - W \tag{2}$$

$$D_{ii} = \sum_{j=1}^{N} W_{ij} \tag{3}$$

After creating L we want to extract the n most dominant eigenvectors, when $n << N$ is the number of clusters we expected to have. each vector represent a low-dimensional embedding of the data points that preserves the pairwise similarity between them, and by taking the n most dominant one we can use them to cluster the data point.

Those eigenvector can help us to spectral clustering the data point, for a given data point x_{ij} we will pick the eigenvector $v_k, k \in [1, n]$ that is closest to the data point

$$\text{cluster}\,(x_i) = \text{argmin}_j \left\| v_j - x_i \right\|_2 \tag{4}$$

1.2 SSL - Semi Supervised Laplacian

Similar to the spectral clustering discussed in the previous sections we will add a small supervised element to the affinity matrix.

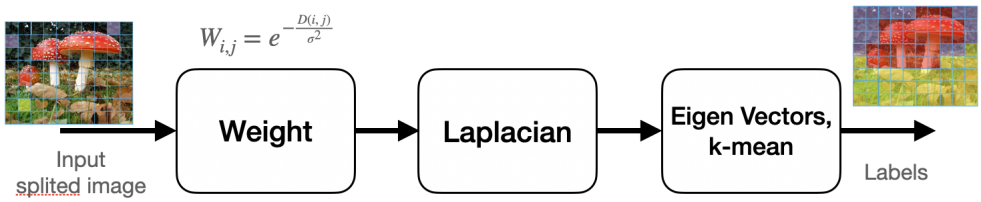


Figure 1: Graph Laplacian Diagram

Given an affinity matrix W that is calculated without the labeled information, we will use the labeled info to create a second matrix W_{lables}. This matrix will be based on the labeled portion of the data. Let there be a subset of the data S the labeled data in this group can belong to K mutually exclusive groups

$$\begin{aligned} \bigcup_{k=1}^{K} &= S \subset X \\ \bigcap_{k=1}^{K} S_K &= \{\} \end{aligned} \tag{5}$$

Now each data point will get a value according to the group it belongs (or doesn't belong)

$$W_{lables}(i,j) = \begin{cases} max(W) & x_i, x_j \in S_k \\ -\frac{2}{a} W_{ij} & x_i \in S_k; x_j \in S_l; k \neq l \\ W_{ij} & x_i \in S_k; x_j \in X \backslash S \\ W_{ij} & x_j \in S_k; x_i \in X \backslash S \\ 0 & x_i, x_j \in X \backslash S \end{cases} \tag{6}$$

Where a is defined as the ration between the groups

$$a = \frac{|X|}{|S|} - 1 \tag{7}$$

We will combine both matrices into the W_{SLL} matrix

$$W_{SLL} = 2W + a \cdot W_{labels} \tag{8}$$

Here a is the same at defined in equation 7.

Now the matrix is increasing the weights associated with the same sub group, and completely eliminates the affinity weight between data points of different subgroups.

With this W_{SLL} we will calculate the laplacian as seen in equations (2, 3) and solve the spectral clustering.

2 SLL Patch work

SLL can be used to take a single photo. Separating the image into patches and with a small subset of the patches tagged we can perform a basic segmentation task with no other priors or training data.

2.1 Image patching

Breaking the image to smaller patches allows us to compare small parts of the image to each other. In our experimental results we tried a wide range of sizes of patches, with or without overlap between patches.

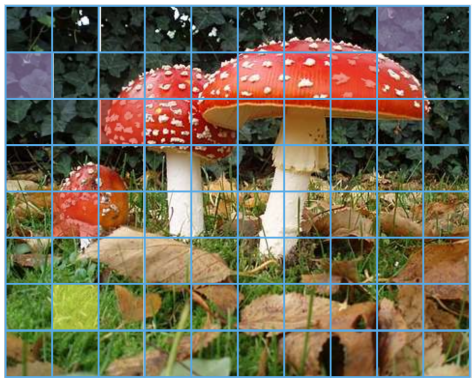


Figure 2: Example of patches and overlap

To show the proposed process view figure 3

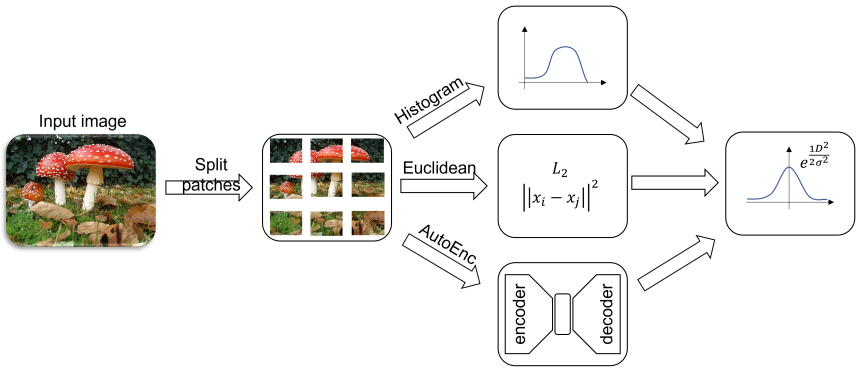


Figure 3: workflow diagram

2.2 Labels

We use fully tagged images as reference to the algorithm. The algorithm get only a small subset of a tagged information. the relative size of the subset varies and we try to find the minimal size that gives us good results.

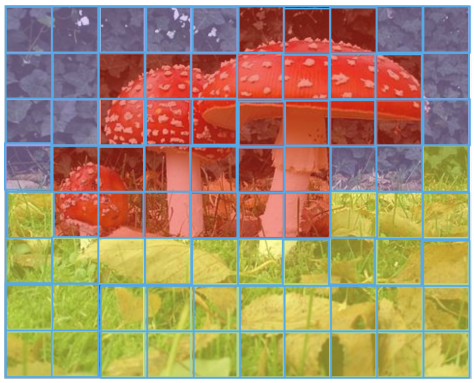


Figure 4: Example of tagging the image

2.3 Weight function

Weight function it is the way we compare the patches between themselves. The affinity matrix is constructed by comparing each patch to all of the other patches (including itself). All the patches are of the same size and multi dimensional. The input might be the RGB values of the pixels, or other preprocessed values. For example gray levels or latent space representation. In any case the input will be 2 arrays each of size $\mathbb{R}^d$

$$W_{ij} = f(i,j) \in \mathbb{R}^+ \tag{9}$$

Part of our goal is to see how different weight functions will perform.

2.3.1 Gaussian kernel

A gaussian kernel is the method to find affinity between 2 patches. Using a non negative distance function $D(i,j)$ that calculates the distance of patch p_i from p_j, we get a wide scale result were 0 represents no distance at all, and high positive values to represent high distances.The guassian is caling it to a $[0-1]$ scale with 1 representing complete affinity is the better variation.

$$W_{i,j} = e^{-\frac{D(i,j)^2}{\sigma^2}} \tag{10}$$

Scaling can be controlled with the σ value. The possible distance functions will be discussed next.

2.3.2 Euclidean (L2) distance

First method to calculate the affinity of 2 patches is simple L2 norm.

$$D_{L2}(i,j) = \sqrt{||p_i - p_j||^2} = \sqrt{\sum_{k=1}^{d} \big(p_i[k] - p_j[k]\big)^2} \tag{11}$$

Usually RGB or gray values are given in a single vector. It is the widespread distance function to use, it shows good results in previous work. However L2 is flawed. For example, 2 patches with a big overlap can have a big distance between them since the same image features will not align and the similarity is lost. There for 2 other methods we are proposing.

2.4 Preprocess

Here we will discuss some suggested functions.

2.4.1 Histogram

The main drawback from L2 distance is the alignment issue, high frequency features with even a 1 pixel shift will see high distance even if the general image is very similar. Therefore we propose taking the patch and creating a histogram of its' pixel values.

$$D_H(i,j) = \sqrt{\sum_{l=1}^{d}\sum_{k=1}^{M} \big(\mathbb{I}(p_i[l] == k) - \mathbb{I}(p_j[k] == M)\big)^2} \tag{12}$$

Where $p_i[l]$ represents the l-th pixel of patch p_i and d stands for the patch dimension, M stand for the number of pixel values we're going to inspect. The distance function will go through pixel pixel in each patch and see if it matches the a value and build an histogram. Eventually we will take the L2 distance of 2 histograms belonging to different patches.

2.4.2 Latent space

Acknowledging the fact we have a weakly supervised task on our hands we try to use an unsupervised method for calculating distance. Using an autoencoder we can train a model to recognize and reconstruct patches from the image. Autoencoder has a latent space vector that represents the patch in a lower dimension. The motivation to use this technique is to harness the non-local attributes of the autoencoder to alleviate the alignment issue. Moreover, using a lower dimension representation will get rid of small insignificant details from the patch and can be a source of additional decreased affinity.

$$D_{AE}(i,j) = \sqrt{\sum_{k=1}^{r} \big(g(p_i) - g(p_j)\big)^2} \tag{13}$$

Where $g(\cdot)$ is the autoencoder model and r is the latent space dimension.

$$g : \mathbb{R}^d \to \mathbb{R}^r \ ; \ d \gg r \tag{14}$$

Secondary method can be taking the reconstructed image as the input, It will keep most of the information but will be much more smooth. Effectively the autoencoder is a sophisticated low-pass filter for the image making the alignment problem much less prominent.

2.5 Weighted Nonlocal Laplacian

In this work we realize the semi-supervised learning of weighted nonlocal laplacian, base on [2]. we want to label a given patch group $P = \{p_1, p_2, ..., p_n\}$, base on a small known subgroup of labels for each label.

$$S = \bigcup_{i=1}^{l} S_i \tag{15}$$

When l is the number of labels, $S \subset P$ is a subgroup of the label patchs and $S_i \subset S$ is the subset of a specific label. For each label i, we want to find a function ϕ_i that will rate us how close the a given patch is to the this label, base on the weight between 2 patchs, and the known label group S. ϕ_i is being computed by solving the following problem.

$$\begin{aligned} &\sum_{y \in P} (w(x,y) + w(y,x)) \left(\phi_i(x) - \phi_i(y)\right) \\ &+ \left(\frac{|P|}{|S|} - 1\right) \sum_{y \in S} w(y,x) \left(\phi_i(x) - \phi_i(y)\right) = 0, \quad x \in P \backslash S \\ &\phi_i(x) = 1, \quad x \in S_i, \quad \phi_i(x) = 0, \quad x \in S \backslash S_i. \end{aligned} \tag{16}$$

When the weight function resemble a Gaussian kernel with Euclidean distance, but with a deviation that is base on the distance between x and it 20th nearest neighbor.

$$w(x,y) = \exp\left(-\frac{\|x-y\|^2}{\sigma(x)^2}\right) \tag{17}$$

Then the weight is truncated to the nearest 50 neighbors in order to sparse the matrix. this problem is not a convex problem, and the solution for this equation can vary depend on how mush free space we going to give to ϕ. this is way this part is solve with the pytorch library, in order to find a solution that can suit the problem.

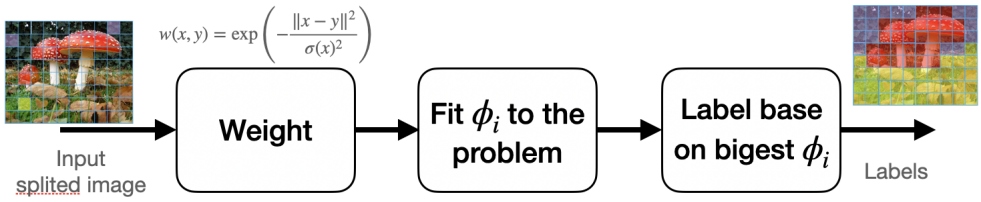


Figure 5: Nonlocal Laplacian Diagram

3 Results

To show our results we took different images and classified them into patches, and use a pre-label patches to label the splited image with semi-supervised learning. we will show the 3 different methods we had talked about, and show the pro and cons of each one.

3.1 Graph-Laplacian

A well known classification tool in the field is the Graph-Laplacian [1.2], who tries to sort the image according to how close the algorithm thinks the patches are. In the fallowing example [Figure 6] we can see an image of a car on a beach, splited into patches of size 28x28, and with a shift between every patch of 14x14, so that there will be an overlap between them. We wanted to classified the sky, beach and car.

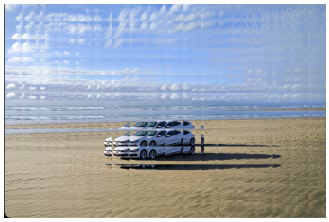
We can see that there is an improvement [Figure 6] of when we try to use pre-label patches in order to classified the patches over the non-pre-label one. but for both cases, the algorithm was not able to classified the entire car.

A big problem we had working with this method, was with try and set the parameters of the method in order to get a acceptable results. more then once the method give a huge bias to one classified color over the other. but on the other hand, it's the most simple of all the method we will show here.

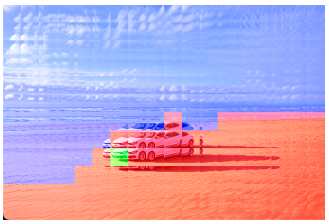
3.2 Weighted-nonlocal-Laplacian results

Next we have the Weighted Nonlocal Laplacian method [2.5], that try to fit a function of labels to a non convex problem. this one give as better result [Figure 7], could label images that the normal Laplacian method could not solve and even was way more flexible with the free parameters we could have inset it. overhaul, it was easier to work with. but now the problem is more complex, with a fitted function that can be solve in a several of ways, with different result.

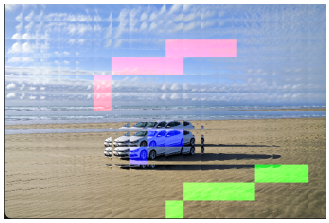
An recurring problem we still have is that both of the method could not label the car in an acceptable way. even when he pre-label the entire car, the method



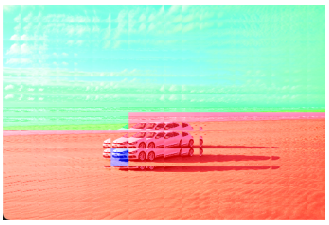
(a) Pre-label Patches



(b) Labels Patches



(c) Pre-label Patches



(d) Labels Patches

Figure 6: Graph-Laplacian classification on splinted image with patches at size 28x28, and shift of 14x14. (a, c) are Prelabel patches for the Semi supervised learning, (b, d) are label patches of the algorithm

refuse to label it as a diffident object. we can also see it with the sunflower seeds [Figure 7], even though there was pre-label the method force them to be label along the green part of the flower.

Another good example of the limitation of this method is **??**], when we wanted a classification of giraffes, lion and background, and got classification of smooth land, animals/bushes and sky/water.

(a) Pre-label Patches

(b) Labels Patches

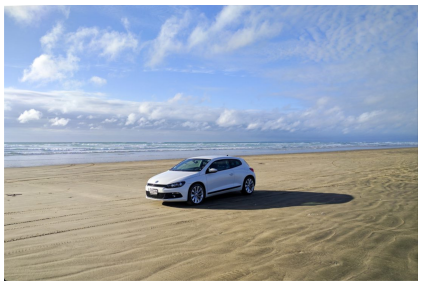
(c) Pre-label Patches

(d) Labels Patches

(e) Pre-label Patches

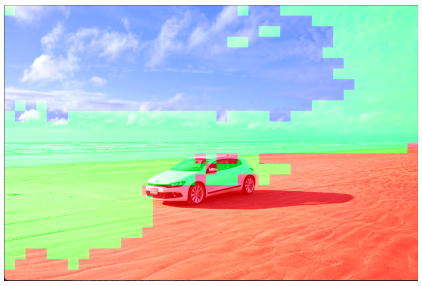
(f) Labels Patches

Figure 7: Weighted Nonlocal Laplacian classification on splinted image with patches at size 28x28, and shift of 28x28. (a, c, e) are Pre-label patches for the Semi supervised learning, (b, d, f) are label patches of the algorithm

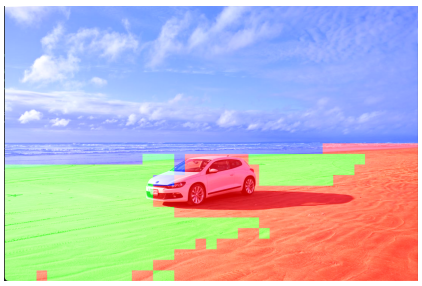


(a) Original image

(b) Histogram distance result



(c) Latent space distance result



(d) Reconstructed L2 result

Figure 8: Segmentation results for different weight functions

3.3 Preprocessed patches

We separate to image to $28 \times 28 \times 3$ patches with no overlap, resulting in 26×39 patches for the image. After that we preprocess the patches with the different processing function before we find the affinity matrix.

For the histogram distance we bin the patch into 32 bins. We chose 32 to avoid sparse histograms that we results in very high distances.

For the autoencoder we get a latent space vector space of $\mathbb{R}^7$, training was done on a bigger set of patches, by using overlapped patches.

As seen in figure 8b we don't get good results. So is true for both autoencoder techniques, latent space and reconstruction as seen in figures 8c,8d respectively.

The car repeatedly proved to be a tough object to segment correctly, light sand and the car blend well together in histogram space and the darker sand stands out. Latent space shows the worse results, probably it has a complicated manifold. The reconstructed image has mixed results. Training set was probably too small to give good results.

4 Conclusions

In this work we compare out method to the vanilla Graph Laplacian Classification, and we found that in same part in perferm better then the old method. on the one hand, our method got the patches labeled more correctly without touching the parameters of the problem too much. but on the other hand, they both fail to rely on the pre-label patches the user provide. the pre-label did help the semi-supervise problem to understand better what are we trying to label, but both methods still ignore same of the know pre-label. We think the inherent problem accrue because the both methods prefers to separate a smooth surface from an exposed one. We try to solve this problem with Segmentation, but it not seem to create the result we desire. more over, when working with patches is that big patches hold a lot of information each so it is easier to match them together. But the big patch creates a rough segmentation. Also using a autoencoder needs more weights to be studied but with little number of patches to train the model. On the other hand small patches create huge weight matrices making the solving extremely resource costly.

References

[1] Xin-Ye Li and Li jie Guo. Constructing affinity matrix in spectral clustering based on neighbor propagation. *Neurocomputing*, 97:125–130, 2012.

[2] Zuoqiang Shi, Stanley Osher, and Wei Zhu. Weighted nonlocal laplacian on interpolation from sparse data. *Journal of Scientific Computing*, 73(2):1164–1177, 2017.

[3] Or Streicher and Guy Gilboa. Graph laplacian for semi-supervised learning, 2023.