Remarks on digits classification project

Wojciech Czaja

November 16, 2015

Remarks on data representation

- One of the main aspects of this project is the investigation of the role the data representation plays in applications. This happens in our project on two different levels:
  1. The images can be represented as vectors. This is conceptually a very non-trivial idea, but it is much easier in practice: since each digital image is formed by a finite collection of numbers, it is all about the order in which these coefficients are used. Please note that the usps.mat data already presents the images as vectors (reading the coefficients "column-wise")[1] . But there are many other ways to do it (e.g., continuously column-wise,"row-wise", spiral, etc).
  2. The data, understood as vectors, can be represented in different forms. This can be done in the same space in which the original data resides, by means of change of basis or other type of transformation. This can also be done by representing data vectors from $\mathbb{R}^D$ in some other space $\mathbb{R}^d$.

[1] This is essentially the content of part (2) of the project: all you have to do is to decipher the meaning of "column-wise" :)

Remarks on data representation

- For the first part, make sure you can access the data and view the individual digits. The following command should allow you to view the 5th digit 3 in the usps data:

  ```
  load('usps_all.mat');
  figure;
  imshow(reshape(data(:,5,3),[16 16]),[]);
  ```

- Remember that the whole dataset is not something that you can just "view" - it is 11K vectorized images.
- Other representations of the images as vectors can be simply identified as a permutation of the coordinates. But then you must remember that the reshape function will no longer produce an image of a digit. if your project analyzes different vectorization schemes (as described in part 1 on the previous slide), you must reverse that permutation before visually inspecting the digits.

Remarks on data representation

- Once images are vectorized (whether as is done already in the usps_all.mat data set, or using your own idea), we can use them directly in classification, or we can **transform** them, before we apply the classification. This is the main component of, and the difference between, parts (3) and (4):
  1. In part (3) you are applying the classification to the "original" vectorized data.
  2. In part (4) you are applying the classification to data which is transformed by eigenmaps of the graph Laplacian.
- The "original" vectorized data consists of 11K vectors in $\mathbb{R}^{256}$.
- The "transformed" data consists of 11K vectors in a possibly different space:
  1. If we compute D ($D \leq 11K$) eigenvectors of Δ, then we'll have 11K vectors in $\mathbb{R}^D$;
  2. If we happen to choose $D = 256$, then we'll have 11K vectors in $\mathbb{R}^{256}$;
  3. But if we choose $D = 11K$, then we'll have 11K vectors in $\mathbb{R}^{11K}$;

Remarks on parameter selection

- A parameter is a measurable factor that is used in defining any particular model or system.
- For the *kNN* classification problem, the parameters include: the number of nearest neighbors, the distance function, the voting method (if $k > 1$).
- The way we vectorize images can also be considered to be a parameter, because it is a function and, as such, it is a measurable factor in our scheme.
- In Graph Laplacian, parameters include: the number of neighbors chosen in the construction of the adjacency matrix, the weights (if we choose a weighted graph), the number D of eigenvectors chosen for representation.

Remarks on parameter selection

Please note that you should either theoretically explain why a specific choice of a parameter is the only one that makes sense, or you should test what happens when you change the parameters over a certain range. For example, for D, there is no ideal theoretical explanation why a certain number of parameters will perfectly describe our data. Clearly we should have $D \geq 10$, as we are trying to classify 10 digits. But it might be also larger numbers, as there are multiple ways of writing certain digits. So you could list how many different ways of writing 10 digits there are (illustrating them with examples from the data). Or you could test $D = 10, 20, 30$, etc. This is going to lead to higher computational costs, as you need to repeat all computations with multiple different choices for D. The choice is yours.

Remarks on computational costs

- There are two major cost generating components of our project:
  1. Sorting distances for kNN classification.
  2. Computing eigenvectors of Graph Laplacian.
- For these you should use existing matlab functions. If you are not careful, you can easily produce implementations which cost on the order of $O(n^3)$, or more. For $n = 11,000$, this would imply on the order of 1.3×10^{12} basic operations (or a constant multiple of it). Good laptops are capable of about 100 GFLOPs, i.e., 10^{11} basic operations. This indicates 13 seconds (or a multiple of) at peak efficiency, for just one aspect of our computations. It is OK for a single run, but sub-ideal for multiple iterations.

Remarks on computational costs

This is why any idea that can shed some unnecessary computations should be considered:

- When filling up a large adjacency or degree matrix, do not perform unnecessary computations for symmetric terms, i.e., set: A(i,j) = A(j,i) = distance (V(i);V(j)), for i = 1 to n, and for j=i +1 to n, rather than for the full $n \times n$ matrix.
- In the vectorization step, you can observe that a number of pixel positions across all images are significantly underutilized - one can get rid of them and reduce the computations without any significant impact on the results.
- Instead of computing all eigenvectors of Δ, compute only $D + 1$ of them, for a number D significantly smaller than 11K. Then use the following scheme: Discard the constant eigenvector of the 0 eigenvalue[2]. Utilize the remaining D eigenvectors to embed your data in a D-dimensional Euclidean space using the map $V(i) \mapsto (e_1(i), e_2(i), \ldots, e_D(i))$, where $e_j(i)$ represents ith coordinate of jth eigenvector.

[2] It is a good exercise to find out why a constant vector is an eigenvector of the 0 eigenvalue for graph Laplacian

Remarks on computational costs

- The last step of choosing only D eigenvectors for $D = 20, 30$, etc, significantly reduces your computational costs by:
  1. reducing the time needed to compute eigenvectors - it is approx. 500 times faster to computer 20 eigenvectors than to compute 10,000 eigenvectors;
  2. reducing the number of computations needed in classification - the distances are computed for much shorter vectors;
  3. simplifying the problem of training the optimal parameters.
- Additional savings can be achieved by sparsifying the graph: is we limit the number of edges connecting any node to the rest of the graph, then we will obtained an adjacency matrix with many zeros. (This is something that Matlab functions like very much.) For example, choosing a fixed number M of the shortest (most significant) edges for each node, and deleting all others, introduces 11,000 - 2M zeros in each row of the adjacency matrix.[3]

[3] Note that these connected nodes are called neighbors in literature - do not confuse with the neighbors in the kNN classification.

Remarks on classification

- You do not need to separate the dataset into two separate subsets in the training/testing split. The selection of the training set can be done by simply selecting the indices of the appropriate rows of the usps_all.mat dataset. Then, we can operate always on the same data matrix, without the need to use additional memory. Moreover, this trick helps in identifying the training vs testing points in the Laplacian representation: if you choose the vector with index i to be an element of the training set, then you will also use the ith vector $(e_1(i), e_2(i), \ldots, e_D(i)) \in \mathbb{R}^D$, to be an element of the training set.
- How well your classification performed is measured by the percentage of correctly labeled (classified) elements of the testing set. That is, if 3300 images out of 5500 were classified correctly, then we say that our classification accuracy is 60%. This is better than if the rate was, e.g., 40%. The goal is to come up with highest possible success rate for problems (3) and (4) separately, and then to compare these results, and draw conclusions.

Remarks on classification

- When you check the classification accuracy on the testing set, please use the earlier instructions to view examples of the corresponding digits which were correctly and incorrectly classified. If they were classified incorrectly, check what was the classification and try to see if you understand why was the digit misclassified. Try to use this information to improve your results.
- It is a good practice to run several different selections of the training set to ensure that our classification is not too dependent on the training (this is called overfitting the data).

Remarks on Laplacian Eigenmaps

One of the original sets of slides presented an algorithm called Laplacian Eigenmaps (LE). This algorithm is **NOT** a solution to your project. This means that you cannot download DR Toolbox, run it on digits data and include in your project. This is because, if you look carefully, you will notice that LE solves a different eigenvalue problem (called generalized eigenvalue problem). There are also other smaller differences: the graph weights are different, the graph Laplacian is defined as $D - A$, etc.
However, notwithstanding these differences, the algorithm and the code published for LE can be very helpful for your own construction!

Last but not least

Too many instructions can make the problem look more complicated than it really is.

This is a beautiful problem, with a goal that is very intuitive, and which allows you to **visually** inspect your results. This can help you understand what you are doing mathematically at each step. Be creative.

Norbert Wiener Center for Harmonic Analysis and Applications