Lab 4: Dimensionality Reduction

March 19th, 2014

Objectives

The topic of this lab is *dimensionality reduction*. More specifically, the objectives are as follows.

1. Get a feeling of dimensionality: how can we understand and interpret the underlying dimensionality of a high-dimensional dataset?
2. Understand how dimensionality reduction techniques based on singular value decomposition work, and how to apply them.

We will not make use of the Hadoop cluster; however later labs will eventually use some of the techniques presented here on Hadoop, so make sure you understand how they work.

The lab is divided into parts. The first part will explore a synthetic dataset of computer-generated faces. The second part will let you analyze real-world data consisting of political opinions collected before the Swiss federal elections in 2011.

Deliverables

The following deliverables, to be handed in before the start of the next lab, should be bundled in a single `zip` file.

- For Section 1, a plot of the magnitude of the components found by the PCA on the faces dataset,
- a text file with the correspondence between principal components and facial features,
- a text file containing the visual description of the face represented by row number 3797.
- For Section 2, a plot of the projection of the candidates' answers matrix onto the two first principal components,
- the text of the questions that contribute the most to the three first principal components of the candidates' answers matrix.

1 Faces

We are interested in analyzing a database of computer-generated faces. A thousand of them is provided in the archive `faces.tar.gz` that you can download from the course's page on Moodle; feel free to take a look already.

These images have been processed by a (fictitious ⌣̈) program called FaceAnalyzr; this program takes an image as input and outputs a vector of measurements (floating point values.) In order to force users to buy the program (which knows how to interpret the measurements), the output is obfuscated in a special way. We are going to analyze the output vectors and uncover the relation with the corresponding images.

1.1 Dimensionality of the Dataset

The output of FaceAnalyzr is stored in `data/faces.txt`. Take a look at it: each line represents the vector of measurements associated with a single face. The components of each vector are separated with a vertical bar (|). In order to process the data in Java, we will work with matrices[1]. The dataset will be represented as an $N \times M$ matrix of `double` values; each row vector contains the measurements associated with a face[2].

1. How many faces are there in the dataset, and how many measurements are provided for each face? That is, what are the values of N and M? In `Faces.java`, complete the top of the function `main()`. Hint: although we provide the images behind the first 1000 faces only, there are many more in the dataset.

2. What is the variance[3] of the data along each dimension? Complete the function `variance()`, run `Faces` and enter `variance` when prompted for an action. Take advantage of the unit tests in `FacesTest` to check your implementation.

Now take a look at the images in `data/faces.tar.gz` again. They share some similarities: for one, they all show a face, they all have a hat, etc. However they do differ on several characteristics.

3. Simply by looking at the images, can you identify the different sources of variability between the faces? Phrased differently: how many numbers would you need to precisely describe the variable part of a face?

4. How does this compare to the dimension of the measurement vector outputted by FaceAnalyzr for each face?

5. By looking at the images and their corresponding row in the dataset (e.g. `face0000.png` corresponds to the 1st line in `data/faces.txt`), are you able to identify any correlation between facial features and vector values?

1.2 Principal Component Analysis

Although it is far from evident just by looking at the data, you are told that the output of FaceAnalyzr does indeed capture the main variability behind the faces. Because the dimension of the measurement vector is much bigger than the apparent number of variables, you decide to apply a dimensionality reduction technique: Principal Component Analysis.

6. Complete the function `pca()`. The function should return an object of type `PCAResult`, a simple container for the direction of each component (`rotation`) as well as its magnitude (`values`). Use the unit tests again to check your implementation.

[1] We will make use of a third party Java library, JAMA: `http://math.nist.gov/javanumerics/jama/`.

[2] Each one of the N faces can then be thought of as a vector in $\mathbb{R}^M$.

[3] Do not use the unbiased estimator for the variance. That is: divide the sum of the squared deviations by N, not $N-1$.

7. Run `Face` with the action `pca`, and look at the plot of the variances. How many components stand out? How does this relate to your intuition about the variability of the faces?

8. Project the dataset onto the new basis formed by the principal component analysis, by completing the function `project()`. By running `Faces` with the action `project` you can display the projection of any row in the dataset.

9. Using the action `extremes` you can see which of the 1000 first faces have extremal values along any dimension[4] in the new space. By visually comparing faces with low values to faces with high values for the first few dimensions, map each principal component to a variable facial feature.

10. There is no image attached to the item at row 3797 of the dataset. Can you give a description of the main facial features? You should be able to do it with the machinery implemented so far.

Note: there are multiple algorithmic approaches to perform PCA on a dataset. For example, you can either compute the covariance matrix and its eigendecomposition, or work directly on the data matrix and compute its singular value decomposition. The relationship between these two approaches will be looked at during an exercise session.

For the purposes of this lab, you are free to use whatever method you want. You can use JAMA's functionalities for eigendecomposition or SVD, or use the provided class `util.EigenDecomposition`, that respects the conventions you saw in class (eigenvalues in decreasing order.)

Now that you have a good intuition about the ins and outs of dimensionality reduction, let us move on to a real-world dataset.

2 SMARTVOTE

In the second part of the lab, you will work with a dataset from smartvote, a Swiss on-line voting advice application. It helps voters finding parties or candidates which are close to their personal political opinions.

This application was widely used during the Swiss federal election of 2011. A majority of the candidates running for the election filled in a questionnaire, which consisted of 75 questions on different political and economical matters[5]. A sample question is shown in Fig. 1.

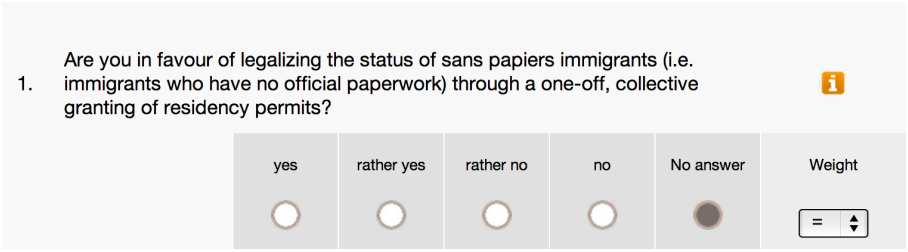

Figure 1: Sample question of the smartvote questionnaire for the 2011 Swiss federal election.

[4]The dimension is 1-indexed: dimension 1 corresponds to column 0.

[5]The questionnaire is still online at: `http://www.smartvote.ch/11_ch_nr/questionnaire/deluxe?lang=en_GB`.

All the questions follow a similar structure, and we can encode the answers with real numbers ranging from 0 to 1:

- no $\rightarrow$ 0
- rather no $\rightarrow$ 0.25
- no answer $\rightarrow$ 0.5
- rather yes $\rightarrow$ 0.75
- yes $\rightarrow$ 1.0

We provide you with the answers of 2985 candidates to each of the 75 questions. In addition, we have included the candidates political affiliation, i.e., to which party he or she belongs. This data is available in the file `data/sv-candidates.txt`. I addition, we have also included a file that contains the text of all the questions, `data/sv-questions.txt`.

By considering that each question corresponds to a dimension, we can represent candidates as points in a 75-dimensional space. The question we ask in this setting is whether or not it is possible to reduce the number of dimensions we need to represent the data. For example, if almost every candidate who chose the answer "no" to the question q_1 also answered "yes" to the question q_2, then we can somehow group the questions q_1 and q_2 together to form a common feature, and hopefully a handful of these features is enough to capture most of the variablity between candidates.

2.1 Analyzing the Candidates' Answers

In this part, you will more or less perform the same steps as you did in Section 1. This time, you will have bit less guidance; go back to the Section 1 if you feel unsure about how to proceed.

Write you code in `SmartVote.java`, and run the function `main()` to execute the different subtasks.

1. First, complete the `main()` function so as to display the number of rows (candidates) and columns (questions) in the dataset. Check that it matches with the description above.
2. Inside of the function `variance()`, compute the PCA[6] on the candidates' answers matrix, and plot the eigenvalues (i.e. the variance explained by each principal component.) Several utility functions (including plotting) are provided by `util.Common`.
3. How much of the variance (in percentage) does the first principal component explain? The two first ones? The three first ones? Make the function `variance()` print out the respective values.
4. Inside of the function `project()`, compute again the PCA of the answers matrix and project the candidates on the principal components. Plot the canidates using the coordinates along the *first* and *second* principal directions, using `SmartVoteUtils.plotProjection()`.
5. On the plot that you obtain, each color represents a different political party. How do you intepret this plot? What interpretation do you give to the first two principal directions? If you are unfamiliar with Swiss political parties, take a little time to read the following pages:
   - `http://en.wikipedia.org/wiki/List_of_political_parties_in_Switzerland`
   - `http://swiss-government-politics.all-about-switzerland.info/`
6. Plot the coordinates along the *third* principal direction (against either the first or second.) Are you able to give a meaningful interpretation to the third direction as well?

[6] You might want to write a separate function that computes the PCA and reuse it for all the subtasks.

2.2 Finding the Most Important Questions

In the last points, we used our knowledge about the main features of the Swiss political landscape to interpret the principal components, i.e., the directions of highest variance. Alternatively, we can also look at the text of the questions that contribute most to each principal component.

7. Inside of the function `questions`, compute once more the PCA of the answers matrix. Extract the first three principal components, and for each of them, print out the text of the three most relevant questions. How do you find them? What is your operational definition of *most relevant question* for a given prncipal component?

8. Interpret the questions for each principal component. How do they relate to your previous interpretation of the candidates' plot?