Features extraction and image clustering

Lorenzo Cioni, Francesco Santoni

lore.cioni@gmail.com, fsanto92@hotmail.it

19 January 2015

Abstract

In this report we will discuss a method of extracting features from manostritti characters in order of clustering similar words that represent the American States from census of 1970.

Contents

1 Introduction

Segmentation and clustering of large amounts of data is one of the main research fields of modern artificial intelligence.

The basis of our work is the collection of u.s. Census data in the year 1940 and fits into the process of digitalizing of handwritten documents that characterizes our time. In this case we have scans of census register and the main problem is to divide enrollees for each American State.

Document segmentation and extraction of the word corresponding to the State was developed previously by two of our colleagues in the course of *Technology of Databases* and is the base from which our work started.So the problem on which we have worked is the extraction of features from handwritten characters to cluster similar words in order to facilitate the recognition by a human agent.

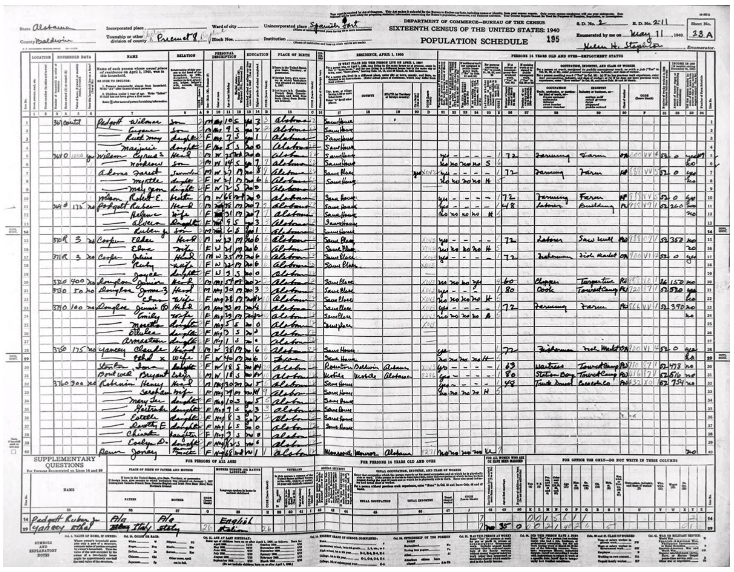
DEPARTMENT OF COMMERCE—BUREAU OF THE CENSUS
SIXTEENTH CENSUS OF THE UNITED STATES: 1940
POPULATION SCHEDULE

Figure 1: An example of a table containing census data

2 The project

The page that contains the data in digital form, contains a wealth of information about each person surveyed including name, gender, membership status and some secondary features such as work or the breed.

All these data are housed in a moulded grid composed of numbered rows and columns for each field to conduct a census. In addition to the data of persons are reported on each archive also additional data related to censor or data relating to samples of the population.

The objective of the project was concerned with finding a particular grid area, the State of each person, from which extract handwritten text associated with the correct name of the State.

Following the data extraction, the project plans to group visually similar elements to form a collection of homogenous States. So each set and then will contain grid theoretically corresponding cutouts of the same State.

2.1 Process steps

We can summarize the project in three basic steps:

1. Search for the text area containing the words of interest
2. Row segmentation
3. Postprocessing of each cut image

The last step, the clustering of images, is our point of interest: it has been rewritten and improved using new features extraction method and a new method for evaluating similarity and will be discussed in detail later.

2.1.1 Word searching

In this first phase the aim is to identify the region of the grid within which we find the State words.

First of all we remove the black border of the document resulting from scanning. Then, using the technique of vertical histogram, we can locate the grid columns and, knowing the correct column's offset relative to the beginning of the document, just the one concerned is extracted is extracted.

2.1.2 Row segmentation

After the extraction of the column concerned from the document, we proceed with row segmentation.

Similar to the previous step, but using the histogram horizontal this time, we are able to determine with some accuracy the rows in the grid. In this case, however, was necessary to center the word in the extracted image: this is done by correcting the height of each row by the analysis of black spikes on the histogram.

At the end of this phase, each word is contained in a single image.

2.1.3 Postprocessing

In this postprocessing phase the individual images extracted are reworked in order to remove any vertical or horizontal residual lines from the previous cut.

Delete row/column strokes can be about as bring back the black pixels related to the residual value of pure white (255 as grayscale).

This allows us to extract the most significant features in the next steps of processing.

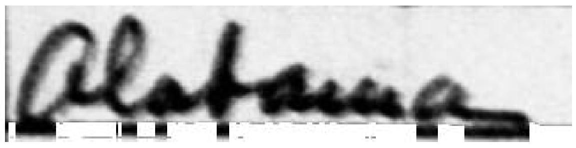

Figure 2: An image extracted from document.

As you can see in Figure 2, the word is centered and the extra line have been removed (there is a line set to white). For each picture we proceed with feature extraction for handwritten characters and clustering.

2.2 Troubles

The identification of words and their segmentation and has some strong issues.

A first problem is represented by the document skew. In this case, this may be due to a not perfectly horizontal scan of the original document. The presence of skew reduces the performance in searching for rows and columns of the grid, thus making impossible segmentation. In some cases, for example, is interpreted incorrectly the first column and this causes the extraction of the wrong column in the first stage of the process.

The main problems of the second phase are mainly related to the identification of extraneous lines. Because of the large number of dashed lines present isn't always possible to *clean* the images and this can cause errors in clustering.

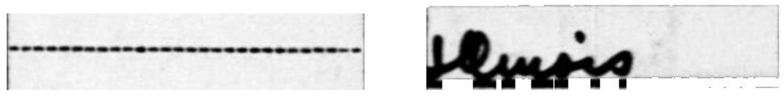
(a) An image with a dashed line (b) The line intersects the word

Figure 3: Segmentation and postprocessing issues

In other cases the word intersects directly gridlines. Because of this we may experience a loss of information about words by removing the line using the method described above.

3 Clustering

The clustering phase provides for the categorization of the various segments in homogeneous groups, so that, at the end of the process, each of them have the same words.

At this stage the main goal is to be able to extract some good features from images that will be used for comparison. The aim is to construct a similarity matrix between different segments so that they can be used as input to the clustering algorithm.

Unable to establish a priori the final number of desired clusters is used *Affinity Propagation.*

3.1 Features

In this work we want to find *primitive* features, resembling the possible types of strokes used to write a word, to characterize the sample from which they are extracted in order to perform clustering on their set.

The features are identified by the study of a sub-portion, or window, of the images extracted by the paper on which this work is based.

The features are located through a study of an area, the window, to which they are implicitly associated: as such there is no clear order of which of the features found in the area comes first. Due to the absence of such *native* order the string that defines a window is maintained consistent with the other strings trough the convention of generating the string with the features' identifiers always taking the same order, if present.

The chosen way to resolve the issue however presents the problem of making sliding windows inapplicable: this means that a sample must necessarily be cut in separate windows, which are allowed to overlap, with the corresponding effect that due to the random cut some features like *loops* may not be recognised in an instance and recognised in another.

Windows Currently we have chosen to cut the samples in windows of 32 pixels each with the exception of the last part of the sample that, deemed irrelevant to the end of characterization due to containing almost always white space, is simply ignored.

These windows are spaced only 16 pixels from the preceding and succeeding one, with the intent of creating overlap and lengthening the string that characterizes the samples.

While the simplest way to create the windows is simply creating a new PIX with the required coordinates it requires unnecessary passages and time in the creation of new objects. We have thus preferred to increase the complexity of the functions that search for the features, to whom we pass as a variable the original sample PIX with information about the *offset* at which to start the search and the *width* of the window.

3.1.1 Whitespace

The first feature that is search in the windows is the white space. This way if a window is identified as blank will not be necessary to proceed with the investigation of other features, saving time.

A window is identified as blank if the average pixel values is below a preset white threshold.

3.1.2 Loop

The feature that represents a loop is recognized only with maximum window size. To locate a loop is initially found a black pixel. The idea behind is to imagine that we had found a point on the border and meet two transitions going in the same direction, first from black to white then from white to black. Between the two transitions there must be a minimum step above a preset threshold where the pixels are white.

If the above condition occurs then we verify that it is real in the same way along the horizontal axis. We then sit in the center of the loop (the median value of the segment described above) and check that, moving both right or left you get a transition from white to black after a suitable number of white pixels.

The implemented method has numerous problems related mainly to the identification of the center of the loop in the segment. With a value of threshold too high also might have to discard some loops too small. The method also does not take into account the thickness of the stroke. However, with appropriate threshold values you can get good results and to identify obvious loops of various segments.

3.1.3 Dot

To find the points in the segment we proceed as in the previous case, seeking a first black pixel.

At that point, for it to be a Dot, there must be a circle of a given radius (whose value is preset and adjustable) in which the pixels are black and outside of that, for some white space.

Even in this case you have the problem of determining a good value of the radius of the circle, having to remember even the size of the stroke.

3.1.4 Diagonal line

The feature representing a diagonal line, both upward and downward facing, is extracted through a simple exhaustive scansion of the window for lines of connected black pixels that have an incline in a range of values and are sufficiently long.

At the moment the function doesn't account for the width of the lines found, thus diagonal features can be recognised in a shapeless blob of black pixel that is sufficiently big.

There's a distinction for diagonal features that appear in the lower and upper bottom of the window. At most in a window the function identifies a couple of upward and a couple of downward facing lines (lower and upper parts), once one has been found it stops searching for the same type.

3.1.5 Cross

The function, having found at most a lower and upper case for upward and downward diagonal lines, proceeds to confront the edge points of these lines: if they possibly intersects it extracts a *crossing* feature with distinction if the crossing happens in the lower or upper part of the window.

The problems with this sub-feature are that intersections of bottom and upper lines of the same type (e.g. both upward facing) with different inclines are not recognised, in the same way there's no consideration for intersections made from lines that are not the "primary" bottom and upper diagonals: if in a single windows are present more lower upwards diagonals and one of them other than the first intersects the recognised downward diagonal, such a crossing is ignored.

The crossing feature may also not necessarily be a complete intersection, in fact for recognition there just needs to be a merging of a downward and upward line.

3.1.6 Horizontal and Vertical line

Both horizontal and vertical lines' features are extracted through a simple scan of the window. The horizontal lines requires a double-window for their implicit characteristic. The function identifies a line if it finds a connected row or column of black pixels that has sufficient length, in particular it distinguishes the lines found in normal or long through an ulterior threshold. Once a fitting line has been found the function keeps searching for more, continuing the scansion of the window after having moved a certain distance from the last line found in order not to confuse a particularly thick stroke as different separate lines. There still persists the problem that the stroke width is not fully considered so a big blob of black pixels is seen as a series of horizontal and vertical lines, at the same time such an occurrence is rare so the presence of many horizontal and vertical lines ends up distinguishing the word in itself.

3.2 LCS

In order to compare the generated strings each others we use the *Longest Common Subsequence* (LCS) algorithm for evaluating distances.

3.3 Affinity Propagation

Affinity Propagation is a clustering algorithm that identifies a set of *exemplars* that represents the dataset[1]. The input of Affinity Propagation is the pair-wise similarities between

[1]Brendan J. Frey, Delbert Dueck, *Clustering by Passing Messages Between Data Points*, http://www.sciencemag.org/, 2007

each pair of data points, $s[i,j] \forall i,j = 1,\ldots,n$[2]. Any type of similarities is acceptable thus Affinity Propagation is widely applicable.

Given similarity matrix $s[i,j]$, Affinity Propagation attempts to find the exemplars that maximize the net similarity, i.e. the overall sum of similarities between all exemplars and their member data points. The process of Affinity Propagation can be viewed as a message passing process with two kinds of messages exchanged among data points: *responsibility* and *availability.*

Responsibility, $r[i,j]$, is a message from data point i to j that reflects the accumulated evidence for how well-suited data point j is to serve as the exemplar for data point i.

Availability, $a[i,j]$, is a message from data point j to i that reflects the accumulated evidence for how appropriate it would be for data point i to choose data point j as its exemplar. All responsibilities and availabilities are set to 0 initially, and their values are iteratively updated as follows to compute convergence values:

$$r[i,j] = (1-\lambda)\rho[i,j] + \lambda r[i,j]$$

$$a[i,j] = (1-\lambda)\alpha[i,j] + \lambda a[i,j]$$

where λ is a damping factor introduced to avoid numerical oscillations, and $\rho[i,j]$ and $\alpha[i,j]$ are, we call, *propagating responsibility* and *propagating availability*, respectively. $\rho[i,j]$ and $\alpha[i,j]$ are computed by the following equations:

$$\rho[i,j] = \begin{cases} s[i,j] \max_{k=j} a[i,k] + s[i,k] & i \neq j \\ s[i,j] \max_{k=j} s[i,k] & i = j \end{cases}$$

$$\alpha[i,j] = \begin{cases} min 0, r[j,j] + \sum_{k \neq i,j} \max 0, r[k,j] & i \neq j \\ \sum_{k \neq j} \max 0, r[k,j] & i = j \end{cases}$$

That is, messages between data points are computed from the corresponding propagating messages. The exemplar of data point i is finally defined as:

$$arg \max r[i,j] + a[i,j] : \forall\, j = 1, 2, \ldots, n$$

As described above, the original algorithm requires $O(n^2 t)$ time to update massages, where n and t are the number of data points and the number of iterations, respectively. This incurs excessive CPU time, especially when the number of data points is large[3]. The Figure 4 shows how affinity propagation works[4].

The clustering process described is then applied to the array of similarity calculated previously on Features extracted from each crop. Once you have run the calculation of clusters, segments are organized into individual folders to provide a Visual result of the proceedings just completed. Each group is identified by a segment that represents the centroid of the cluster, which is the element to which all others in the group are closer.

4 Results

5 Conclusion

[2] $s[i,j]$ for each data point is called preference and impacts the number of clusters.

[3] Yasuhiro Fujiwara, Go Irie, Tomoe Kitahara, *Fast Algorithm for Affinity Propagation*, 2009

[4] Brendan J. Frey, Delbert Dueck, *op. cit.*, Figure 1, p. 974

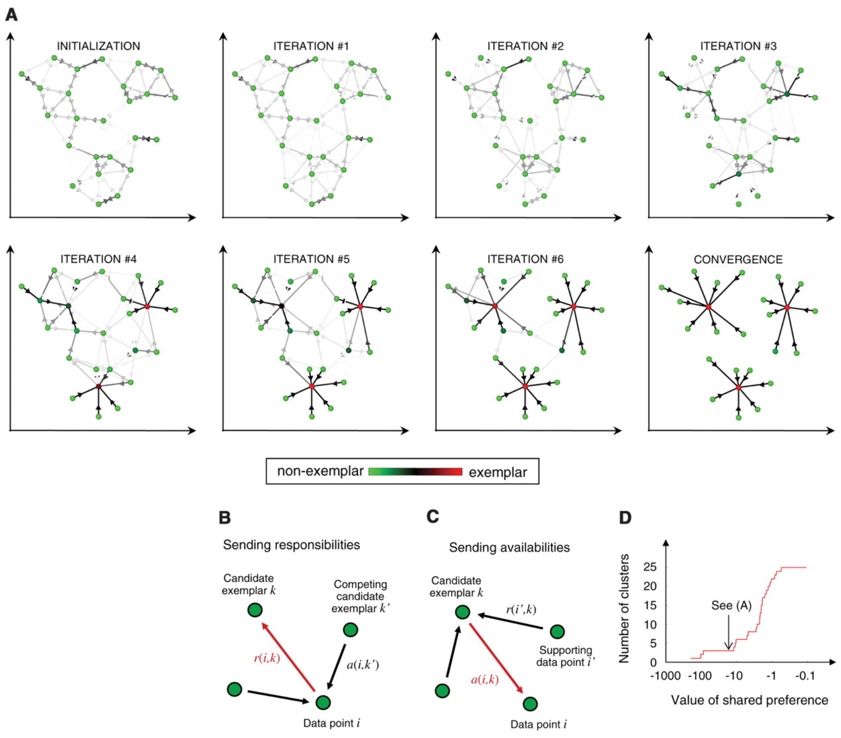

Figure 4: How affinity propagation works