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 primitive features from handwritten characters to generate clusters of similar words, in our case the names of American States found in the 1970' census.

Contents

1 Introduction **1**

2 The Pre-existing project **2**
2.1 Process steps . . . 2
2.1.1 Word localization . . . 3
2.1.2 Row segmentation . . . 3
2.1.3 Post-processing . . . 3
2.2 Troubles . . . 4

3 Clustering **4**
3.1 Features . . . 4
3.1.1 Whitespace . . . 5
3.1.2 Loop . . . 6
3.1.3 Dot . . . 6
3.1.4 Diagonal line . . . 6
3.1.5 Cross . . . 6
3.1.6 Horizontal and Vertical line . . . 7
3.2 LCS . . . 7
3.3 Affinity Propagation . . . 8

4 Results **9**

5 Conclusion **9**

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 digitalization of handwritten documents that characterizes our time. In this case we have scans of the census register and the main goal is to divide enrollees by 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 basis from which our work started. The problem on which we have worked is the extraction of features from handwritten characters with the goal of creating clusters of similar words in order to facilitate the recognition by a human agent.

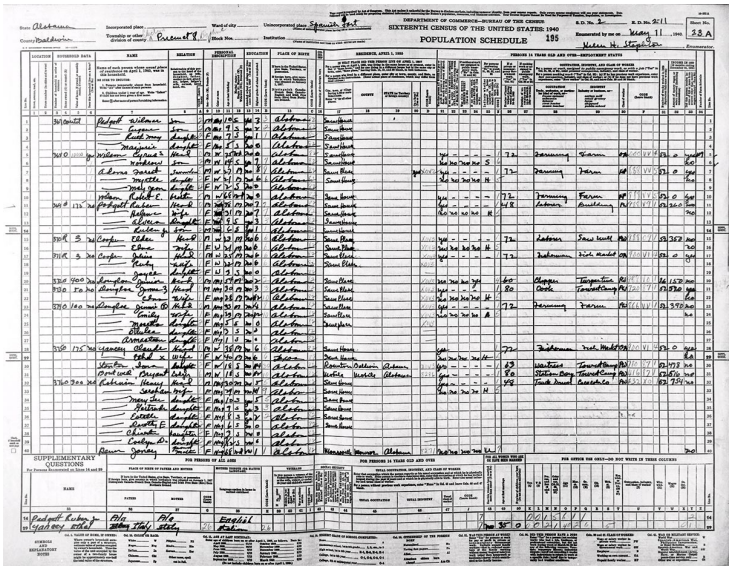


Figure 1: An example of a table containing census data

2 The Pre-existing project

The census page that contains the data in digital form holds a wealth of information about each person surveyed: 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. In addition to the citizens data, each archive also contains additional information related to the censor or data relating to samples of the population.

The project's objective was finding a particular grid area, corresponding to the area containing the registered State of each person, from which extract the handwritten text.

Following the words localization, the existing work proceeds by grouping visually similar elements to form a collection of homogeneous samples, in hope that each set will then contain cut outs of the same State.

The goal of the work is not to group all the words corresponding to the same state in a single cluster but rather to generate clusters that contain words that belong to a single state, the major requirement of the work is thus precision in the retrieval steps.

2.1 Process steps

We can summarize the project in three basic steps:

1. Localization of the text area containing the words of interest

2. Row segmentation

3. Post-processing of each cut image

//questo dovrebbe essere dopo

The last step, corresponding to the clustering of images, is the point of interest of our paper: while the past work was focused on the localization and retrieval not much thought was put on the features to be extracted preferring simplicity and efficiency to effectiveness. It is this very point that gives our paper a reason d'etre: starting from the preceding, already implemented, two steps, we proceed by providing a rewritten and improved clustering method through the use of new features, these steps will be discussed later in the section 3.

2.1.1 Word localization

//direi di togliere "we" visto che non siamo noi

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

First of all we remove the black border of the document, an artefact resulting from the physical scanning. Then, through a vertical projection of the pixels and the creation of an histogram, we locate the grid columns and, knowing the correct column's offset regarding the beginning of the document, just the one concerned is extracted.

2.1.2 Row segmentation

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

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

At the end of this phase, ideally each word is contained in an individual image.

2.1.3 Post-processing

In the post-processing phase the individual images extracted are reworked in order to remove any vertical or horizontal residual lines left from an imperfect 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.

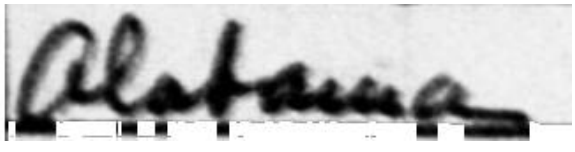


Figure 2: An image extracted from document.

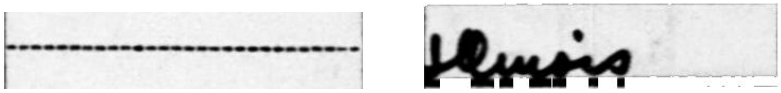
As you can see in Figure 2, the word is centered and the extra line have been removed (there is a line set to white). For each picture we proceed with feature extraction for handwritten characters and clustering.

2.2 Troubles

The localization of words and their segmentation has inherent strong difficulties.

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

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 it isn't always possible to *clean* the images and this can cause errors in the clustering phase.



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

Figure 3: Segmentation and post-processing issues

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

3 Clustering

The clustering phase consists in the categorization of the various segments in homogeneous groups, so that, at the end of the process, each cluster contains words corresponding to the same State.

At this stage the main goal is the extraction of good features representative of the images to 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. To do so we generate a measure of distance between couples of samples through the use of the *Longest Common Subsequence* algorithm in which the handled strings, representative of the corresponding samples, are constructed through the appending of conventional identifiers associated with the features found in the samples in a consistent order.

Unable to establish a priori the optimal number of desired clusters makes the use of the *Affinity Propagation* algorithm a necessity.

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, with which they are implicitly associated: as such there is no clear order of which of the possibly multiple 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.

Each sample is associated to a characterizing string of characters made from the identifiers of the features recognized in the sample. The string is constructed modularly by appending in order the strings associated to each window the sample is divided in. After having recognised the presence of a feature in a given window the corresponding identifier is added to the end of the string correspondent to the window; the order in which the features are searched and thus added to the queue is decided a priori and maintained consistent during the construction of the strings.

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. The width of the window passed has actually a non banal meaning duo to the fact that different features are linked to areas of the image with varying dimensions: for example we can't reasonably search for an horizontal line and a vertical line utilizing windows with the same width due to the fact that horizontal lines realistically will require windows with great width but will not have requisites on the height. The height of the windows used is never considered as a parameter since for all features' functions the height is customarily the whole height of the sample.

//possibilmente sezione stringa/caratteri?

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 one another we must define a distance on the samples. The distance used in our work is based on the *Longest Common Subsequence* (LCS) algorithm.

The LCS algorithm has the aim of extracting from a set of sequences (in this case only two) the longest common subsequence, that is a sequence that is obtainable from both the starting sequences by deleting some elements without changing the order of the remaining ones.

LCS is a particular case of the *Edit Distance* algorithm where the only allowed operations are insertion and deletion. The distance associated with LCS in our current work is the number of insertion and deletions that must be applied to obtain the longest subsequence, in accordance with the Edit distance where the distance is calculated with the number of operations needed to morph a string in the other (in Edit Distance it's possible moreover to confer customizable costs to the substitutions).

While the LCS algorithm may require high costs when applied concurrently to high numbers of sequences, in our case there exists an easy and light implementation that exploits *dynamic programming*, in this type of implementation the cost ends up being $O(n * m)$ where n and m are the length of the compared strings.

$$LCS(X_i, Y_j) = \begin{cases} 0 & if\ i = 0\ or\ j = 0 \\ LCS(X_{i-1}, Y_{j-1}) \cup x_i & if\ x_i = y_j \\ longest(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)) & if\ x_i \neq y_j \end{cases}$$

Obtained the Longest Common Subsequence between two strings the distance between them is thus:

$$x.length() + y.length() - 2 * LcsLength$$

where x and y are the strings and the $length()$ function returns the length of a string.

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 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:

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

$$\alpha[i,j] = \begin{cases} min0, r[j,j] + \sum_{k \neq i,j} \max 0, r[k,j] & i \neq j \\ \sum_{k \neq j} \max 0, r[k,j] & i = j \end{cases}$$

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.

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

[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

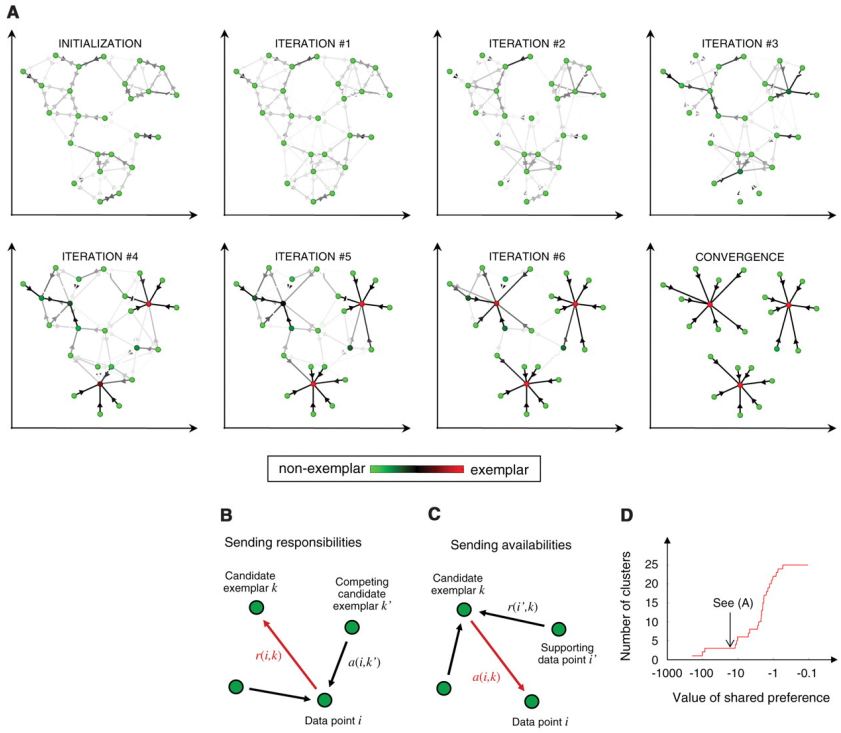


Figure 4: How affinity propagation works

4 Results

5 Conclusion