Clustering of handwritten words based on structural features extraction

Lorenzo Cioni, Francesco Santoni

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

17 February 2015

Abstract

In this report we discuss our application for the extraction of primitive features from images of handwritten words and the generation of clusters of similar elements. In this case the words, compared to the names of American States and other countries, are extracted from forms of the 1930' U.S. census.

Contents

1 Introduction

Segmentation and clustering of large amounts of data is one important research field of artificial intelligence.

In recent years, with the expansion and diffusion of the digitalization of physical archives, both in the private and public sector, these techniques have become increasingly prominent for economic and financial reasons since they can easily shrink the amount of man-hours needed to achieve the tasks involved.

Related to these circumstances, the application developed in this work presents an approach to group together handwritten words with similar visual shape. The intent of the application is to make possible the grouping of images, representing the same word, contained in handwritten scanned documents in sets with a predefined minimum precision. From these sets a human operator can thus categorize all the words through the apposition of a single label per set, rather than per element, with obvious gains in time and costs.

The basis of this work is the collection of U.S. Census data of the year 1930. The goal is to divide enrollees by state: this is done through the clustering of the handwritten words contained in the "State" field of the forms.

Document segmentation and extraction of the words corresponding to the states was developed previously by two of our colleagues in the course of *Technology of Databases* for documents with a structure similar to the one of our dataset. Their project is thus the starting point of this work: the main contribution of this project to the preexisting work is in the definition of a different, more relevant, valuation method through which determine the distance between different images, in hope that the results may have a greater precision. The method proposed in this work is based on the use of *structural features*, directly connected to the stroke with which the words were written. The main problem on which we have worked is thus the extraction of features from handwritten words with the goal of creating clusters of similar elements so as to facilitate the recognition by a human agent.

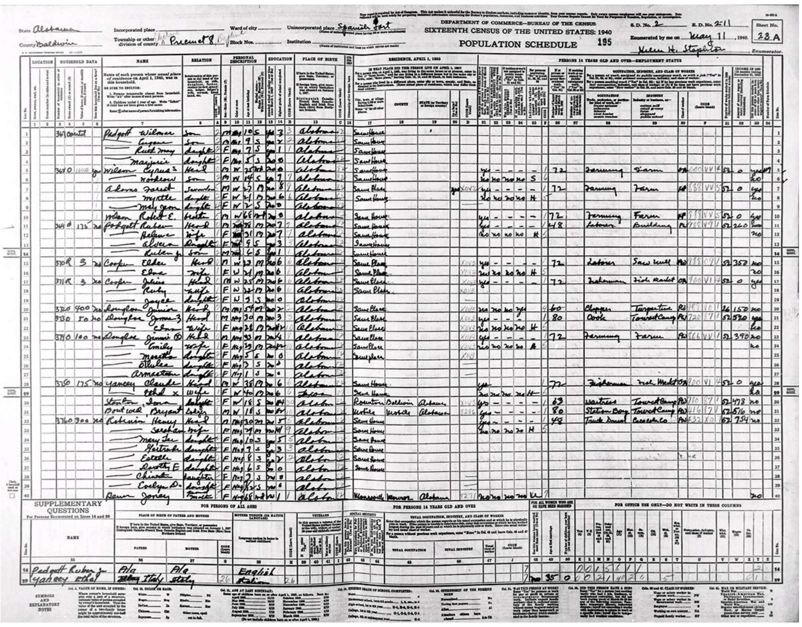
2 The pre-existing project

The census page that contains the data 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 information related to the censor or supplemental 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



State Alabama County Baldwin

Unincorporated place Spanish Fort

Township or other division of county Precinct 8

DEPARTMENT OF COMMERCE—BUREAU OF THE CENSUS

SIXTEENTH CENSUS OF THE UNITED STATES: 1940

POPULATION SCHEDULE 195

S. D. No. 2 E. D. No. 2-11 Sheet No. 23 A

Enumerated by me on May 11, 1940. Helen H. Stapleton, Enumerator.

Figure 1: An example of a table containing census data

contain cut-outs of the same State.

The goal of [1] 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 clustering.

2.1 Processing 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 and clustering

The last step, corresponding to the feature extraction and clustering of images, is the focus of this work: while the previous work was focused on the localization and retrieval of the samples, not much thought was put on the features to be extracted. It is this very point that gives our work a reason d'être: starting from the preceding, already implemented, two steps, we proceed by providing an improved clustering method through the use of new features, these steps will be discussed in Sections 3 and 4.

2.1.1 Localization of the area of interest

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

First of all the black border of the document is removed since it is an artefact resulting from the physical scanning. Then, through a vertical projection of the pixels, the first grid column is located. From this first column, utilizing the known offset of the interested state column from the first, the interested column's borders are located. The borders are searched in a given area based on the offset, through them just the concerned area of the image is extracted.

2.1.2 Row segmentation

After the extraction of the concerned column from the document, our colleagues [1] proceed with row segmentation.

Similar to the previous step, but using an horizontal projection, the rows of the extracted column can be determined with some accuracy. In this case, however, it is 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 and clustering

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.

This operation of "deletion" consists in setting the related black pixels to the value of 255, pure white. In case the spurious rows are intersected by the word, additional steps are taken to leave the intersecting pixels to the original value. We find an example of this in Figure 2.2.

This allows us to extract the most significant features in the next steps of the process. To extract the features each image is divided in windows with width of 1 pixel, on these windows are then applied the functions utilized for the extraction.

The preexisting project focus was the extraction of the words, placing less importance on the clustering and thus the features themselves. Moreover the project philosophy is not to create excessive computational burden therefore the features are relatively simplistic. For each window the features considered are: the height of the first and last black pixels and the number of transitions of the pixels from black to white and vice versa. The distance between images is computed through the Euclidean Distance where the considered axis are the number of transitions and the stroke height calculated through a simple subtraction of the first and last black pixels height. The distance matrix is then fed to the Affinity Propagation algorithm to create the desired clusters.

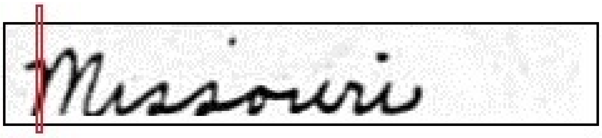


Figure 2: An image sample extracted from document, the red box delineates the 1 pixel wide window.

A sample of extracted image is shown in Figure 2, the word is centered and the bottom presents a white line: the bottom border row of the grid was intersecting the word and was extracted with it, in the pre-processing phase it was thus removed to not hinder the feature extraction.

2.2 Problems

The localization of words and their segmentation has inherent difficulties.

A first error source is represented by a significant document skew. The skew may be due to a not perfectly horizontal scan of the original document, its presence reduces the efficacy in the search for rows and columns of the grid, making the segmentation impossible with the given implementation of the project.

Another source of errors is the presence in some images of other census forms behind the main one. Black columns from the lower document are interpreted as belonging to the main one, making the application recognize them as the first column of the form.

This erroneous association renders the scan unusable since the targeted state area is found from the first black column through an hard-coded distance in pixel.

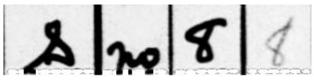


Figure 3: The resulting erroneous images

The identification and removal of touching lines is another source of difficulties: since this happens very often the correct recognition is extremely hard, for it is not always possible to *clean* the images and this can cause errors in the clustering phase.

Figure 4: Cut-out with intersecting lower line

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

Regarding the previous source of errors many have been corrected in the new implementation: through a more precise and thorough identification of the first black column the erroneous segmentation resulting in samples similar to Figure 3 is unlikely to occur; the problems stemmed from the difficult phase of cleaning the segmented images have been skirted through the removal of the steps to clean intersecting lines altogether. As a consequence the presence of a black line doesn't generate problems for the new features' detection.

3 Features extraction and implementation

In this work we want to improve upon the previous project by providing an alternate method to calculate the distances, the focus isn't any more speed and simplicity but rather even at a major cost we want to find characteristics more deeply related to the handwritten words nature in hope they may be better suited for the recognition of similar handwritten words.

The idea is thus 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.

In particular due to the way the code was implemented the feature extraction follows immediately the segmentation of the image and can thus be executed in sequence within the same threads without adding great complexity to the process.

The feature extraction and implementation step can be summarized in:

- Operate a sliding window to scan a cut-out image
- Find the structural features belonging to the given window
- Associate the strings corresponding to the features to the window, creating a bigger string
- Create the string corresponding to the whole cut-out image from the fusion of the strings associated to the windows

3.1 Structural features

The features are identified by the study of a sub-portion, or window, of the images correspondent to the entries of the "State" field of the census document on which the procedure is applied.

The features that we search are intrinsically characteristics of the stroke, they are located through the study of the area (the window), with which they are implicitly associated; this poses a problem: there is the possibility that an area may be characterized by multiple features, in this case there is no clear order of which of the multiple features 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 not directly applicable: this means that a sample must necessarily be cut in separate windows, which are allowed to overlap, with the result 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 ordered identifiers of the windows in the sample, each window's string is itself made from the identifier strings of the features recognized in the sample. The features that convey more information about a word or are more precisely identified, for example loops and dots, are associated to longer strings so that their presence may be better recognised.

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 the samples(the cut-out images) are cut in windows of 40 pixels each, with the exception of the last segment of the sample that, deemed irrelevant to the end of characterization since it contains almost always white space, is simply ignored.

These windows are spaced 7 pixels from the preceding and succeeding ones, with the intent of creating overlap and lengthening the string that characterizes the samples: the lengthening allows more precision in the clustering phase increasing the distance between strings. (Both the dimension of the window and the spacing between them was decided through tests to obtain the optimal efficacy of the program)

While the simplest way to create the windows in the code would be simply cutting the original image in pieces through specialized functions of the *Leptonica* library, these steps require a great amount of memory and time. We have thus preferred to increase the complexity of the functions that search for the features, to whom we pass as a variable

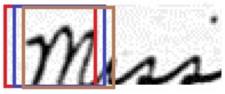


Figure 5: Overlapping windows 40 pixels wide, spaced 7 pixels

the original images 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 due to the fact that different features are associated with areas of the image with varying dimensions: for example it is not sensible to 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.

In particular, the features with less space requirements are searched in both 20 pixels halves of the windows and their strings are appended to the image string, the whole 40 pixels window is then searched for the space-hungry features, only then their identifiers are added to the general string.

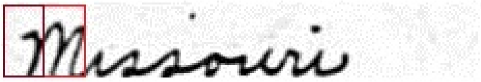


Figure 6: A window (40px) and its first half(20px)

The height of the windows used is never considered as a parameter since for all the features searching functions the height is customarily the height of the sample.

3.1.1 Whitespace

The first feature that is searched in the windows is the presence of white space. If a window is identified as blank there will not be a need to proceed with the search of the other features, saving time.

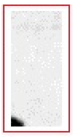


Figure 7: A *Whitespace*

A window is identified as blank if the average pixel values is below a pre-set threshold.

The string associated with the *Whitespace* is " ". This string is not given much consideration when calculating distances since it may bring problems if images with the same word are not properly centered in the segmented samples, while differentiating words through the presence of spaces does not bring significant improvement.

3.1.2 Loop

The feature that represents a loop requires the search of the whole 40 pixels window. The localization of a loop starts from the search of a black pixel. The idea behind this approach is that a point on the border of a loop is such that by looking in a pre-defined direction (in this case the y axis) we find two pixel's value transitions first from black to white and then from white to black. Between the two transitions for a loop to be recognised there must be a minimum number of white pixels.

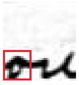


Figure 8: A loop

To recognise a loop the above condition must also be verified in the other cardinal axis. To do so, starting from the center of the supposed loop (the median value of the segment described above) we check that moving either right or left we find a transition from white to black after a suitable number of white pixels.

The implemented method has numerous problems related to the difficulty of finding the best white spaces threshold: an excessively little minimum number of white pixels makes us recognise as loops white noise that happens while scanning images, a threshold too high forces us to discard some small but real loops. The method also does not take into account the thickness of the stroke, rendering the overall recognition harder, with appropriate threshold values tough we can achieve good results.

The string associated with the *Loop* feature is "LL".

3.1.3 Dot

To search for *Dots* within the segment we first proceed locating the *connected components* inside of it. The individual components are extracted and inserted into a *box* of which we know the size and the relative position to the segment.

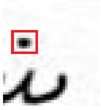


Figure 9: A dot

At this point we search for those boxes with dimensions between two pre-set values (minimum and maximum radius), those that meet this condition are boxes that most likely contain points. Using the connected components we can thus retrieve dots in a simple way.

The string associated with the Dot feature is ".....".

This string is especially long because the dot helps distinguish the words that contain the "i" letter and a dot is easily recognised with little error.

3.1.4 Diagonal line

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

In the case of an upward facing diagonal, given a black pixel we consider it connected to another black pixel if this second one is in one of 3 different position that are illustrated in Figure 10. The lines recognized are thus all those with gradient between 30 and 60%.

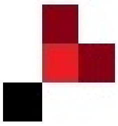
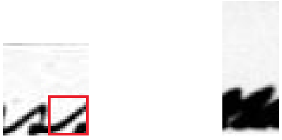


Figure 10: Given the black pixel we search for other black pixels in the 3 positions, the closer red one is given priority

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



(a) (b)

Figure 11: proper *Diagonal* feature(a) and a formless blob that introduces errors(b), in this second both upward and downward diagonal lines are recognised

A distinction in the associated string is introduced for the diagonal features that appear in the lower or upper bottom of the window, moreover the searches for the two kinds of lines are separate. 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. The lines that are found are further categorised on their length if medium or long.

Many different strings are associated with the *Diagonal* feature, they are properly listed in Table 1. In particular the string associated with the "long" version of a feature is made from the string associated with the "medium" version of the feature prefixed with an additional character: this allows to distinguish features based on their length while at the same time maintaining similarity with a shorter version of themselves to account for variation in the stroke used to write the same word.

3.1.5 Cross

The function, having found at most a lower and upper case for upward and downward diagonal lines, proceeds to compare 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, priority is given to the bottom part if for example a bottom upward diagonal intersects an upper downward diagonal.

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.

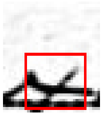


Figure 12: A cross

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.

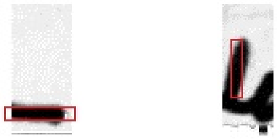
The string associated, depending if the cross is found in the upper or lower half, is "X" and "x" respectively. While it may seem that a crossing is distinctive of a word and thus deserving of more "characterizing force", that is the length of the associated strings, the difficulty in recognising the feature and it's uncertainty renders an excessive weight given to its presence dangerous for clustering purposes. The feature is thus maintained for future improvements but doesn't bring immediate contributions to the application.

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 require 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 additional threshold. In particular, as obvious difference from the diagonal line, no play is left to recognise inclined lines as horizontal or vertical.

Once a fitting line has been found the function keeps searching for more until the exhaustion of the assigned window. The scansion of the window is continued after having moved a certain distance from the last line found in order not to confuse a particularly thick stroke as different separate lines. The corresponding string as thus no limitation in length other than the ones posed by the maximum number of lines that can fit in a window.



(a) (b)

Figure 13: Horizontal(a) and Vertical(b) features

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.

Recapitulation

Follows a recapitulation of the various features and their associated strings in Table 1. In the *Simplified* column are contained the corresponding strings utilized in the example of features' extraction of Section 3.3.

Feature	**Associated string**	**Simplified string**
Whitespace	" "	" "
Loop	"LL"	"L"
Dot	"....."	"."
Upward facing diagonal lower half, medium	"s"	"s"
Upward facing diagonal lower half, long	"bs"	"s"
Upward facing diagonal upper half, medium	"S"	"S"
Upward facing diagonal upper half, long	"BS"	"S"
Downward facing diagonal lower half, medium	"u"	"u"
Downward facing diagonal lower half, long	"vu"	"u"
Downward facing diagonal upper half, medium	"U"	"U"
Downward facing diagonal upper half, long	"VU"	"U"
Cross upper	"X"	"X"
Cross lower	"x"	"x"
Horizontal line lower half, medium	"-"	"H"
Horizontal line lower half, long	"h-"	"H"
Horizontal line upper half, medium	"H-"	"H"
Horizontal line upper half, long	"Hh-"	"H"
Vertical line lower half, medium	"ii"	"V"
Vertical line lower half, long	"IIii"	"V"
Vertical line upper half, medium	"Vii"	"V"
Vertical line upper half, long	"VIIii"	"V"

Table 1: Recapitulation of the strings associated to each feature

3.2 Dimensional features

While the already proposed structural features help in categorizing and recognising words to cluster, they are not completely able to distinguish a word from similar others, we have thus searched for ways to improve the accuracy of the application: while many optimizations have been added to the defined structural features we have found that adding another layer of features of a different kind, in this case *dimensional*, helps greatly. Moreover the complexity added is extremely limited: as can be seen later on in the Table 3 the required calculation time does not increase in a particularly meaningful way: the majority of the process complexity is not located in the features' extraction but rather in the construction of the LCS-distance matrix and in the clustering. Thus in order to improve word clustering we combine structural features with *dimensional features* for each word, after constructing the distance matrices for both we combine them through a factor determined from testing.

These dimensional features are exactly the ones previously utilized by our colleagues in the pre-existing work: height of the stroke, calculated through the difference between highest and lowest black pixel, and number of transition of the pixels' value from black to white and vice versa.

A simple explanation on how they are extracted can be found in section 2.1.3.

3.3 Features extraction example

Follows an example of features extraction from a word sample. In this particular example are utilized simplified strings associated to each feature (for the relations consult Table 1).

We initially create, as described in the section above, a sliding window that scrolls along the image by a fixed step. Then, for each section, we extract features and generate a substring.

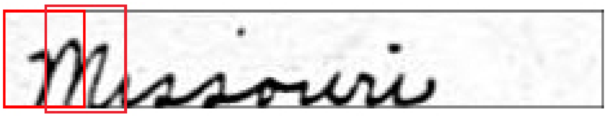
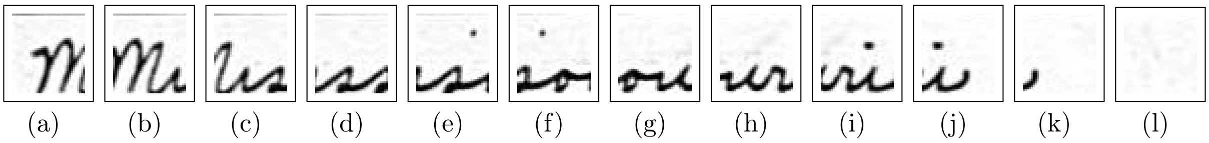


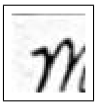
Figure 14: Feature extraction from *Missouri* word



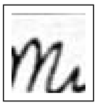
(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l)

Figure 15: Sliding window segmentation

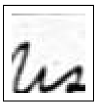
(a) Some **diagonal lines** (ascending (s) and descending (u)), at the top (S) and at the bottom (s) of the image. Generated string: ”*sSUusSUSu*”.



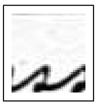
(b) As the previous image and two more diagonal lines representing the "*i*". An **horizonal line** (H). Generated string: ”*sSUusSUSuHsu*”.



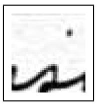
(c) Some diagonal lines and, at the end of the window, an horizontal line. Generated string: ”*ssSusSsH*”.



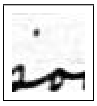
(d) Two consecutive similar character represented by diagonal lines and **vertical lines** in the middle. Generated string: ”*sVHsV*”.



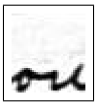
(e) In that window we can find a **dot** (.). Generated string: ”*ssVHs.*”



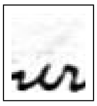
(f) The same previous dot and a little **loop** (L) at the bottom. Generated string: ”*ssVs.LssH*”.



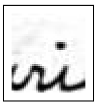
(g) The same loop as previous, an horizontal line a vertical line and two diagonal. Generated string: ”*LVHssu*”.



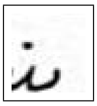
(h) Generated string: ”*ssVHus*”.



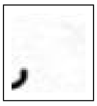
(i) The other dot here. Generated string ”*ssus.H*”.



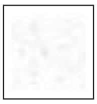
(j) An i. Generated string: ”*ssuu.*”.



(k) Only a diagonal line. Generated string: ”*s*”.



(l) Empty window. Generated string: ” ”.



The resulting string is:
"sSUusSUSusSUusSUSuHsussSusSsHsVHsVssVHsssVs.LssHLVHssussVHusssus.Hssuu.s"

Once those strings are generated they are combined together to generate the *structure string* associated with the word sample.

4 Evaluating distances

After the extraction of good features from the words our aim is to construct a similarity matrix between different samples that can be used as input to the clustering algorithm. To do that we compute the distance between pairs of words through the use of the *Longest Common Subsequence* algorithm in which the strings are constructed by appending of conventional identifiers associated with the features found in the word in a consistent order.

4.1 Longest Common Subsequence

In order to compare the generated strings 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 aims at extracting from a set of sequences (in this case only two) the longest common subsequence, that is a sequence that is obtainable from both 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 insertions 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.

4.2 Euclidean Distance

The Longest Common Subsequence is only used with strings. In order to improve the correctness of the words' distance matrix we combine LCS distance with the Euclidean Distance obtained through the use of the *dimensional features.*

The Euclidean Distance between two samples, a and b, is evaluated through the formula:

$$d_{a,b} = \sum_{i=1}^{W} \sqrt{[(t_a^{(i)} - b_a^{(i)}) - (t_b^{(i)} - b_b^{(i)})]^2 - (n_a^{(i)} - n_b^{(i)})^2}$$

where:

- W is the word image width.
- $t_a^{(i)}$ is the top black pixel in i column in word a.
- $t_b^{(i)}$ is the top black pixel in i column in in word b.
- $b_a^{(i)}$ is the bottom black pixel in i column in in word a.
- $b_b^{(i)}$ is the bottom black pixel in i column in in word b.
- $n_a^{(i)}$ is the number of transitions in i column in in word a.
- $n_b^{(i)}$ is the number of transitions in i column in in word a.

5 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 only words corresponding to the same state of birth.

To do so the previously built distance matrices, obtained by the processing of the structural and dimensional features, are utilized: the matrices are combined in a single one through a simple weighted sum.

$$ClusteringMatrix[i,j] = StructuralMatrix[i,j] + 0.5 * DimensionalMatrix[i,j]$$

The factor of 0.5 was determined through extensive tests.

To utilize the vast majority of the clustering algorithms we need to know the number of desired final clusters. Since our objective isn't having a particular number of clusters but rather having clusters with elements with certain properties we need to use an algorithm that hasn't the requirement: we have thus used the *Affinity Propagation* algorithm.

5.1 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[2], $s[i,j] \quad \forall i = 1,\ldots,n;\ j = 1,\ldots,n$. Any type of similarities is acceptable thus Affinity Propagation is widely applicable.

Given the 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

[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.

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:

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

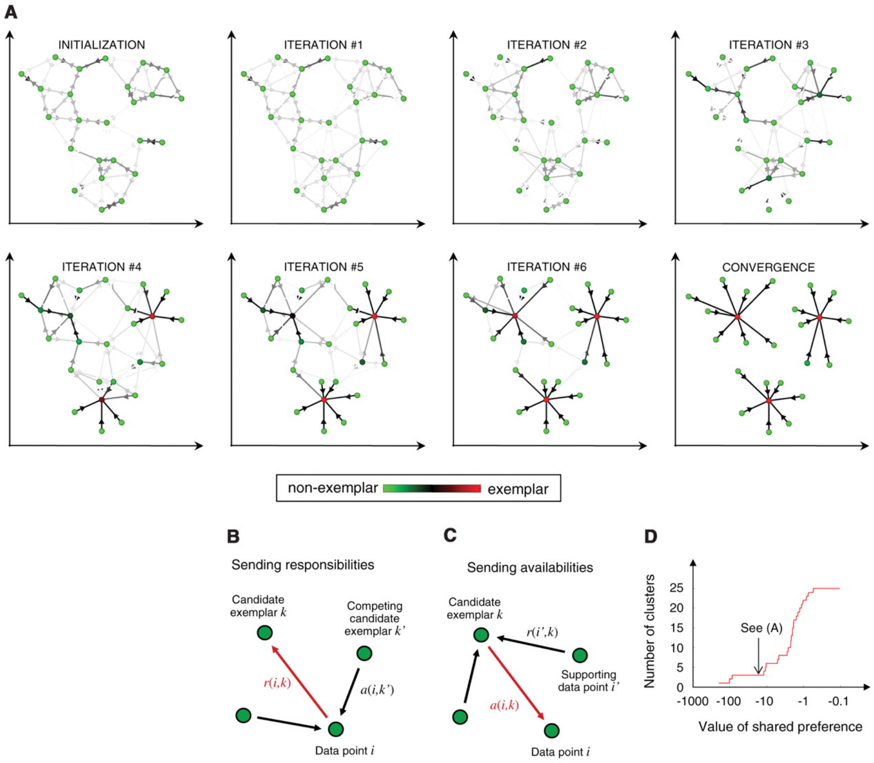


Figure 16: How affinity propagation works

As described above, the original algorithm requires $O(n^2t)$ 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 16 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 words. 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.

6 Results

Initially we ran debug tests on our personal PCs in order to easily modify the code, these starting tests and the tests utilized to determine the right value for the many defined constants are not shown here.

All the tests shown below were performed on a single more powerful machine that has enabled us to work with much larger data in less time. The machine used is composed of two Xeon processors for a total of 16 cores 2.80 Ghz and 48 Gb of RAM.

The tests were performed on a growing number of forms, each containing a maximum of 50 lines from which we extracted the words that represent the states. For each set of forms we present three possible distances calculation: only LCS, only L1 (Euclidean distance) and LCS and L1 combined together through the formula presented in Section 5.

- **Estimated words (E)**: the number of words estimated on the basis of the number of forms(50x).
- **Extracted words (W)**: the number of words actually mined and processed, as well as the corresponding percentage.
- **Number of clusters (C)**: the number of clusters created in process.
- **Average words per cluster (AWC)**: the average number of words in each cluster.
- **Running time (T)**: the total execution time, in seconds.
- **Average precision (AP)**: the accuracy of the results, the average accuracy of clusters.

  $$Precision_{average} = \frac{\sum_i P_i}{N_c}$$

  where P_i is the precision of cluster i and N_c is the number of clusters.
- **Precision (P)**: the accuracy of the results, the average accuracy of individual clusters weighted with the number of words.

  $$Precision = \frac{\sum_i P_i * n_i}{N_w}$$

[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

where P_i is the precision of cluster i, n_i is the number of words in cluster i and N_w is the number of words.

In the next table are shown the main results of the tests.

	E	**W**	**C**	**AWC**	**T**	**AP**	**P**
16 forms (L1)	800	550	55	11.00	15.52	65.06	58.00
16 forms (LCS)	800	550	71	7.74	44.39	76.56	60.00
16 forms (LCS and L1)	800	550	66	8.33	47.41	74.37	62.91
32 forms (L1)	1600	800	67	11.94	38.58	59.85	52.87
32 forms (LCS)	1600	800	92	8.69	94.54	72.97	55.50
32 forms (LCS and L1)	1600	800	86	9.30	112.90	70.19	56.25
75 forms (L1)	3750	2350	173	13.58	221.03	69.03	63.65
75 forms (LCS)	3750	2350	189	12.43	1169.72	75.09	67.49
75 forms (LCS and L1)	3750	2350	199	11.81	1203.28	78.01	71.16
130 forms (L1)	6500	5050	425	11.88	5444.43	76.66	71.12
130 forms (LCS)	6500	5050	441	11.45	5629.71	82.11	74.41
130 forms (LCS and L1)	6500	5050	463	10.91	6729.62	84.87	77.94
500 forms (L1)	25000	18146	3957	4.58	85545.11	90.66	77.89
500 forms (LCS)	25000	18146	3316	5.47	76324.63	91.02	81.22
500 forms (LCS and L1)	25000	18146	3941	4.60	138309.85[5]	92.27	82.14

Table 2: Main results

As can be seen in Table 2 the number of extracted words is much lower than the estimated number of words. This is mainly due to the fact that not all census tables are completely filled: in some cases there are only a few lines or the states column was purposefully left blank. In other cases the absence of the state samples is due to errors occurring in the segmentation phase due to a wrong interpretation of the rows or the columns.

As one can see in Figure 17 the accuracy of the cluster grows with the amount of words extracted. This phenomenon is due to the fact that Affinity Propagation works best with a large number of available data: the greater the number of words, the greater the chances of finding words similar between them, and then combine them within a single cluster.

[5] During this test, the machine used was concurrently executing other tasks creating a bottleneck in the allotted memory, in all similar tests the time was in the order of 90k seconds.

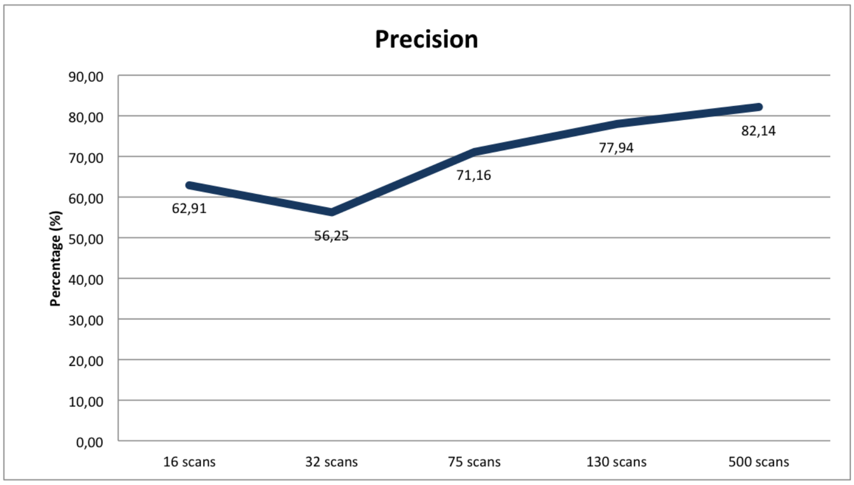


Figure 17: Clustering precision relative to the number of forms

In the next table are presented the results regarding the time needed for the calculations. The running time is divided mainly into three distinct categories corresponding to different phases of the process: time needed for the extraction of features, time needed for the creation of similarity matrix (calculation of distances) and time needed for the clustering.

- **Features extraction time**: the time, in seconds, required for features extraction from the words.
- **Evaluating distances time**: the time, in seconds, required to generate the similarity matrix with the distances between all the words extracted.
- **Clustering time**: the time, in seconds, required for the creation of clusters.
- **Running time**: the total execution time, in seconds.

	Features extraction time (s)	**Evaluating distances time (s)**	**Clustering time (s)**	**Total (s)**
16 forms (L1)	6.81	5.02	3.69	15.52
16 forms (LCS)	6.84	34.40	3.15	44.39
16 forms (LCS and L1)	6.84	37.32	2.85	47.41
32 forms (L1)	11.54	7.22	19.82	38.58
32 forms (LCS)	11.27	74.72	8.55	94.54
32 forms (LCS and L1)	11.34	89.96	11.60	112.90
75 forms (L1)	33.65	69.00	118.38	221.03
75 forms (LCS)	35.13	1002.63	131.96	1169.72
75 forms (LCS and L1)	34.70	1070.92	97.66	1203.28
130 forms (L1)	70.80	313.70	5059.93	5444.43
130 forms (LCS)	68.17	4353.85	1207.69	5629.71
130 forms (LCS and L1)	73.10	5608.82	1047.70	6729.62
500 forms (L1)	289.33	4482.02	80773.76	85545.11
500 forms (LCS)	323.66	57699.51	18301.46	76324.63
500 forms (LCS and L1)	403.41	61424.57	76481.87[5]	138309.85[5]

Table 3: Running time

As can be seen in Table 3 the complexity is mainly due to the phase of construction of the similarity matrix, that is the calculation of distances. This high operative cost occurs especially in calculating the LCS distance due to the fact that the structural strings of our words are very long and the cost of the algorithm is $O(nm)$, with n and m the lengths of the two strings, cost that must be considered in each comparison between couples of samples.

Since long structural strings were sought to obtain a better characterization of the words (and therefore make more accurate clustering) the calculation time necessarily increased.

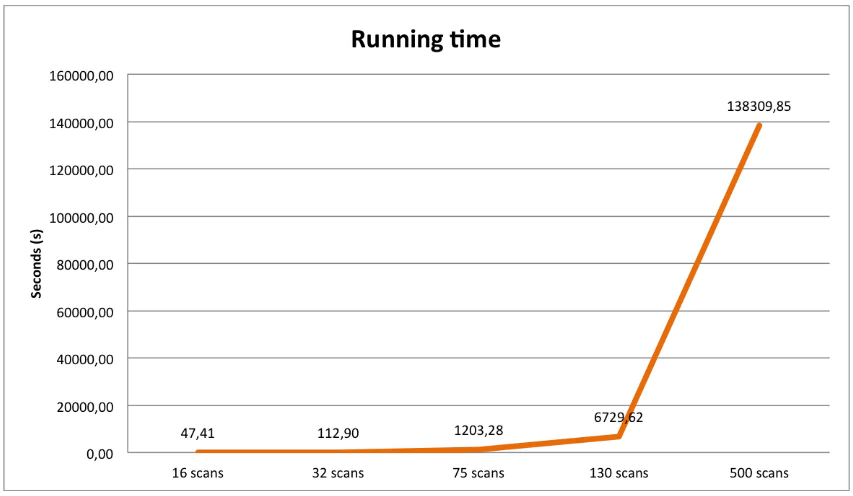


Figure 18: Running time relative to the number of forms

A parameter to evaluate the goodness of this application relatively to the task can be the number of clusters that have absolute precision, that is contain only similar elements that are properly classified. Proper classification is important since due to human error sometimes the words in the debug files are wrongly categorized.

In the following table are presented the number of clusters which respect the above property and the number of items that they contain.

- **Correct clusters**: the number of clusters with absolute precision, therefore containing only words equal to each other.
- **Correct words**: the number of words within the Correct clusters and their corresponding percentage respect to all the elements.
- **Single clusters**: clusters containing only one word.
- **Average words per correct non-single cluster** (**AWCC**): number of words in average in the correct clusters that do not contain a single element.

	Correct clusters	**Correct words**	**Single clusters**	**AWCC**
16 forms (L1)	12	18	10	4.00
16 forms (LCS)	33	44	29	3.75
16 forms (LCS and L1)	25	45	18	3.86
32 forms (L1)	11	11	11	–
32 forms (LCS)	39	48	36	4.00
32 forms (LCS and L1)	32	43	28	3.75
75 forms (L1)	39	172	11	5.75
75 forms (LCS)	78	265	43	6.34
75 forms (LCS and L1)	77	316	43	8.03
130 forms (L1)	138	651	23	5.46
130 forms (LCS)	208	713	107	6.00
130 forms (LCS and L1)	215	860	89	6.12
500 forms (L1)	2854	7354	151	2.66
500 forms (LCS)	2902	7423	163	2.65
500 forms (LCS and L1)	3020	7616	145	2.60

Table 4: Correct clusters

7 Compiling and running notes

This software was developed entirely in C++, and uses an open source library specialized in image processing and analysis, *Leptonica*[6], version 1.70. Due to technical reasons the newest *Leptonica* version 1.71 could not be used. In this new version even .j2k files are supported, utilizing the older version meant manually converting the files to the already supported .jpg extension to elaborate them.

Leptonica provides many functions for manipulating images pixel by pixel using a high-level approach. Thanks to this library for example, one can draw up a diagram of projections, crop images, or find the connected components in a portion of the image.

To compile the code, once included the library described above, it is necessary, in the case of version of *GCC/G++* less than 4.7, to compile with version 11 of C++ that introduces support for threads.

To run this program the following parameters are used:

- **-d** to specify the images directory, from there the program will automatically load all the files with the specified extension (in our case .jpg) and the corresponding .txt files utilized in the determination of the precision.
- **-t** to specify the number of threads. Default is 2.
- To determine the distance matrix used (default lcs+l1) one can use:
  - **--lcs** to use only LCS distance for building similarity matrix.
  - **--l1** to use only Euclidean distance for building similarity matrix.

[6] Leptonica, *a pedagogically-oriented open source site containing software that is broadly useful for image processing and image analysis applications* -http://www.leptonica.org/

8 Conclusion

In this project we wrote an application that after accepting as input U.S. Census documents returns clusters containing words that corresponds to the same state of birth. Starting from a pre-existing work that already located and segmented the samples corresponding to the citizens' state, we added a more thorough search for the targeted area, and with the intent of improving the application by providing an alternate method of calculating the distance matrix we developed a different kind of features that we feel are more closely associated to the way a word is written through different strokes. Through the use of these *primitive* features and the already defined dimensional features we are able to generate distance matrices based only on one of the features' kind or on both after a reasoned combination. We utilize then the Affinity Propagation algorithm to obtain the desired clusters.

Based on the results obtained we can note that the accuracy of the cluster grows with the amount of words extracted. This phenomenon is due to the fact that Affinity Propagation works best with a large number of available data: the greater the number of words, the greater the chances of finding words similar between them, and then combine them within a single cluster. At the same time, with the increase of the processed words there's also an increase in the rate of correct clusters.

We note that the time needed to complete the calculations increases quadratically with the number of elements. This, as shown in Table 4, mainly due to the time needed by LCS to calculate the distance between each pair of words. Using other calculation methods one might reduce the necessary computing time.

From the tests the new features clearly provide an improvement on the older dimensional features, this improvement can further be expanded through the use of the mixed distance matrix. The improvement is sharp when low numbers of scans are considered but decreases at higher numbers where the precision of the 3 methods tend to converge.

The application overall succeeds in our intent although with not incredible improvements on the preceding work, possible avenues of optimization are in the definition of the strings associated to the features, taking in consideration that an increase in the string length is directly proportional to an increase in LCS distance's calculation time.

Other possible and obvious improvement, although that would require an overall rewriting of the project from it's foundation, would be the association of each segmented word with its coordinates in the general census image: this cannot be done at the moment because the coordinates of a segment are ignored since the preexisting inherited segmentation steps that simply cuts the words without any reference to their place in the general image. This improvement would confer an effective practical utility to the application; at the same time it would permit a betterment of the debug and testing phase through the possible use of the same coordinates in the search of the word corresponding to the segment, rather than having to rely on each segment's row number.

References

[1] Alessio Melani, Moreno Niccolai, *Segmentazione e Clustering di Stati del censimento americano del 1940.* Database Technology course, University of Florence, 2014.

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