COMP8430 GCDE – Data Wrangling – 2020

Lab 3: Blocking and Comparison for Record Linkage Week 3

Overview and Objectives

Today's lab is the first in a series of three labs during which we will gradually build a complete record linkage system. We will provide you with basic Python skeleton modules, and today and over the next two labs you will be asked to complete the different components of the modules.

In today's lab we examine the blocking and comparison steps, which are covered in lectures 13 to 16. We have divided this lab into two parts. In the first part we cover the blocking step while the second part of this lab is allocated to the comparison step of the record linkage process.

The idea of blocking is to avoid having to compare every possible pair of records across two data sets. By using blocking we exclude from the comparison step those record pairs which are very unlikely to be matches. In the comparison step we compare the candidate record pairs that can be generated using the blocks we generate in the blocking step. We can use different comparison functions to compare attribute values. The basic idea of string comparison functions is to provide a numerical measurement of how similar two strings of characters are. In practice this is very important, because nicknames, typographical and data entry errors, variations, etc. are all common in many databases, and so only looking for exactly the same string (such as through a database join) may give very poor results.

Preliminaries

You should start this lab by downloading from Wattle the `comp8430-gcde-reclink-lab-3-4.zip` archive (in Week 3) that contains the lab materials. In there you will find the skeleton modules and data sets which we will be using to complete the record linkage system. Please create a folder for the program on your machine and extract the materials there.

Before you start, please also have a look through the Python skeleton modules to get a feel for how it is structured and what the different parts are. Today we are working with `blocking.py` and `comparison.py`, but it is good to have some understanding of how it all fits together. First look at `recordLinkage.py` since this is the module that runs the complete process.

Run `recordLinkage.py` as it is. It will use some of the provided data sets, and the functions already implemented. This will show you what the output for the different steps will look like. We recommend using the small data sets with no corruptions, named `clean-A-1000.csv` and `clean-B-1000.csv`, as a way to test whether your program is working. Once your program is working, apply it on the other, larger, data sets.

Part 1: Lab Questions on Blocking

Your first task today is to experiment with the blocking process and to implement two blocking functions. As a starting point, we have already implemented one blocking function, `simpleBlocking`, where records are placed into different blocks based on the value of a chosen blocking attribute (or attributes). For example, all the records with the same *Surname* will go in the same block.

We will first begin the lab by reviewing the aim of blocking, and how simple blocking, Soundex, and the SLK-581 methods (see below) work. The tasks are as follows:

1. Review and discuss the aim of blocking as part of the record linkage process. Discuss how simple blocking works.

   Now, calulate the Soundex codes for Brian Schmidt (our VC) and Queen Elizabeth II. Then calculate the SLK-581 for Brian Schmidt and Queen Elizabeth II (their required personal details are publicly available).

   Then calculate the Soundex code for your first name and your last name, and then calculate your SLK-581 code.

2. Next, start looking at `blocking.py` and explore how the blocking functions work (inputs, return values, etc.). Then run the blocking step on the two small data sets using both `noBlocking` and `simpleBlocking` (on attributes of your choice) and investigate how blocking affects the output. You can comment or uncomment different lines in `recordLinkage.py` to call different combinations of functions. Examine how these blocking techniques do affect the number of record pairs that are to be compared. Write down the number of blocks generated, as well as their minimum, average, and maximum sizes.

3. In the `blocking.py` module, implement the Soundex phonetic encoding, and use the Soundex value as the blocking value. In other words, all values with the same Soundex code should be placed in the same block. For a full description of Soundex, see lecture 14.

   Note: There are different specifications and implementations of Soundex, we expect you to follow the description given in lecture 14.

Three letters of Last name	Two letters of First name	Date of birth	Gender
XXX	XX	DDMMYYYY	N

4. The *Statistical Linkage Key* `SLK-581` is an identifier that can be used to identify records that belong to the same person if they have the same SLK-581. As shown in the figure on the next page, SLK-581 is made up of four elements, including three letters from family name (surname or last name), two letters from given name (first name), date of birth, and gender.

   In the `blocking.py` module, implement SLK-581 as a blocking key where all records with the same SLK-581 identifier should be added to the same block. We will cover SLK-581 in more detail in lecture 16, but you can also refer to the web page at https://www.aihw.gov.au/getmedia/e1d4d462-8efa-4efa-8831-fa84d6f5d8d9/aodts-nmds-2016-17-SLK-581-guide.pdf.aspx for more details on SLK-581.

After you have finished coding each blocking method, please repeat your experiments from task 2 on the other data sets, but using the new blocking functions. Consider both the output, i.e. the number of records per block, number of blocks, etc., and the performance information such as time taken. You may also want to print out some of the blocking keys to see what they look like, or build a frequency distribution of the blocking keys or block sizes. Think about the following questions:

- Which do you think are the best blocking functions and keys, and why?
- Can you use some of the attributes unsuitable for blocking in combination with others to improve blocking?
- Can you come up with a list of criteria for good blocking keys based on the experiments you conducted?

If time permits, there are plenty of other blocking techniques that you can look at and implement into the `blocking.py` module. Two that are commonly used in practice are canopy clustering and sorted neighbourhood blocking. If you have time please implement either of these techniques. Note that to implement sorted neighbourhood based blocking, you will need to modify the skeleton program somewhat in order to make it accept an index rather than a dictionary of blocks.

Part 2: Lab Questions on Comparison

In the second part of this lab, we focus on the comparison step. As with blocking, we provide you with a starting point `comparison.py` and two simple string comparison techniques, and we ask you to implement some additional ones yourself.

First have a look at the code already provided in `comparison.py` for the functions `exact_comp` and `jaro_comp` so you can see how the string comparison functions work (inputs, return values, etc.). You can experiment by running either function on some of the example data sets to see what the output looks like and how it performs. Maybe add some print statements to see what strings are being compared and what their corresponding calculated similarities are.

Now, manually calculate the Jaro similarities between the following pairs of names:

- jones / johnson
- michelle / michael
- shackleford / shackelford

Hint: See Section 5.5 in the *Data Matching* book for a description of this algorithm as well as example calculations.

Next, begin to implement the following three string comparison functions. Assume A and B are two list of values (such as q-grams extracted from Strings).

1. Jaccard similarity, which is calculated as, $\text{Sim}_{Jacc}(A,B) = \frac{|Set(A) \cap Set(B)|}{|Set(A) \cup Set(B)|}$
2. Dice similarity, which is calculated as, $\text{Sim}_{Dice}(A,B) = \frac{2*|Set(A) \cap Set(B)|}{|Set(A)|+|Set(B)|}$
3. Bag distance, which is calculated as, $\text{Dist}_{Bag}(A,B) = max(|Bag(A) - Bag(B)|, |Bag(B) - Bag(A)|)$, where the function $Bag()$ returns the multiset of values in an element (see lecture 16 for more details).

Discuss, as a group, what the functions $Set()$ and $Bag()$ should return, for example for the q-gram lists $[jo, oh, hn]$ (from 'john') and $[ba, ar, rn, na, ar, rd]$ (from 'barnard').

The basic idea of each of these string comparison techniques was outlined in lecture 16. You may find additional details on the Internet and there are several academic works that have compared different string comparison techniques (see for example the *Field and Record Comparison* chapter in the *Data Matching* book).

Once you have finished implementing each technique, please experiment with them. Some questions to explore are:

1. How does each comparison function compare to the others on different types of data (names, dates, postcodes, etc.)?
2. Are they all equally fast or slow? (You may need to run them on the larger data sets to get a good indication of this). You can use the Python *time.time()* function from the *time* module to measure how long some code is running.
3. Are there some problems / errors that none of these string comparison techniques can deal with?

If you finish the three techniques described above, please also have a look at the Winkler modifications to the Jaro string comparison technique, as well as the edit distance function discussed in lecture 16. Note that for edit distance, you will need to use a dynamic programming implementation.