INFS3200 Sem 1, 2025

Prac3 Data Linkage

Introduction

Learning objectives:

- Learn how to use *psycopg2* to interact with *PostgreSQL*, including the operations of create table, update table, query table, etc.
- Learn how to measure the performance of data linkage.
- Understand various string similarity measures, e.g., edit distance and Jaccard coefficient, for field-level data linkage.

Part 1: Preparation of the Restaurant Data Set

In this part, our objective is to prepare data for subsequent tasks. You can choose one of the two options for data storage: (1) import data into PostgreSQL database, or (2) store data as CSV files. Both solutions are implemented and available in Python. Although both options are available, we recommend you to try the database option, as you can learn more about how to interact with database using Python code.

Step 0. Upload material files in "Prac3_material.zip" to UQ Zone.

Step 1. Database setup.

In your terminal, run following commands:

1. psql
2. CREATE DATABASE "Prac3";
3. \c "prac3"
4. CREATE USER "p3" WITH PASSWORD 'infs3200';
5. CREATE TABLE restaurant (id SERIAL, name VARCHAR(60), address VARCHAR(80), city VARCHAR(30));
6. \copy restaurant from '/YOUR/PATH/TO/restaurant.csv' with csv
7. GRANT SELECT ON TABLE "restaurant" TO "p3";

The above commands create a database called "prac3" and a user "p3" that has read privilege on it. The data in the provided CSV file has been loaded to the database.

In this prac, we use psycogp2 to read data from the database. Psycopg2 helps you to write Python applications that manage these three programming activities:

- Connect to a data source, like a database,
- Send queries and update statements to the database,
- Retrieve and process the results received from the database in answer to your query.

Now, let’s have a look at "DBconnect.py" file in "DataLinkage/src/psql". This code snippet contains a function for initializing database connection. To initialize the connection, we need to input the database detail and user detail in the function. The default DB and user detail is coded in the script. If you are using a different information for DB or user in the pervious step, please update the code accordingly.

<u>Task 1</u>: Read the code in *nested_loop_by_name.py* and focus on the data loading part. Understand how data are loaded into *restaurants* array and complete the following data statistics tasks **in** *data_statistics.py*:

- Count the number of restaurant records whose city is "new york" and "new york city", respectively.
- Count total number of distinct values in *city* attribute (**Hint**: use *set* in Python).

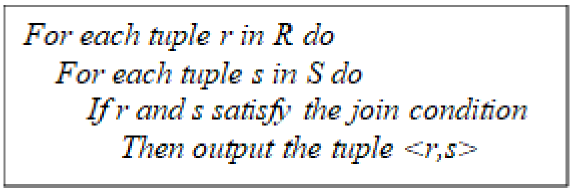
Part 2 Measure the Performance of Data Linkage

There are some duplications in the original *restaurant* dataset, and we will use *Data Linkage* techniques to detect these duplications, i.e., pairs of records that refer to the same real-world entity. For example, the following two records actually represent the same restaurant:

- 5, "hotel bel-air", "701 stone canyon rd.", "bel air"
- 6, "bel-air hotel", "701 stone canyon rd.", "bel air"

Nested-Loop Join for Data Linkage

The nested-loop join, a.k.a **nested iteration**, uses one join input as the outer input table and the other one as the inner input table. The outer loop consumes the outer input table row by row. The inner loop, executed for each outer row, searches for matching rows in the inner input table. The pseudo-code below shows its workflow.



```
For each tuple r in R do
    For each tuple s in S do
        If r and s satisfy the join condition
            Then output the tuple <r,s>
```

Figure 1. pseudo-code of nested-loop join.

In the simplest case, the search scans an entire table or index, which is called a naive nested loop join. In this case, the algorithm runs in *O(|R|*|S|)* time, where |R| and |S| are the number of tuples contained in tables *R* and *S* respectively and can easily be generalized to join any number of relations. Furthermore, the search can exploit an index as well, which is called an index nested loop join. A nested loop join is particularly effective if the outer input is small and the inner input is pre-indexed and large. In many small transactions, such as those affecting only a small set of rows, index nested loop joins are superior to both merge joins and hash joins. In large queries, however, nested loop joins are often not the optimal choice.

In this Prac, we adopt a nested-loop method to self-join the RESTAURANT table for data linkage. We first consider perfect matching on the "name" attribute. In other words, we link two restaurant records (i.e., they refer to the same entity) only if their names are identical, e.g.,

- 1, "arnie morton's of chicago", "435 s. la cienega blv.", "los angeles"
- 2, "arnie morton's of chicago", "435 s. la cienega blvd.", "los angeles"

Program *nested_loop_by_name.py* is the implementation of this algorithm. It first reads all restaurant records from the PostgreSQL database using psycopg2, and then self-joins the table by a nested-loop join. If two records have the same "name" value, the algorithm outputs this linking pair, i.e., ***id1_id2*** where *id1* and *id2* are the "**ID**"s of these records respectively. ***Please modify the path*** in the script accordingly and run the script.

Precision, Recall, and F-measure

We can regard the problem of data linkage as a classification task. Specifically, for each pair of records *r* and *s*, we predict a binary class label: "0" or "1", where "1" means we believe these two records refer to the same entity and hence can be linked. Naturally, a data linkage algorithm is perfect if and only if:

(1) **Precision**: all the linked pairs it predicts are correct, and

(2) **Recall**: all possible linked pairs are discovered.

We provide a file "**data\\restaurant_pair.csv**" which stores the gold-standard linking results, i.e., all the restaurant record pairs that refer to the same real-world entity. Suppose that ***D*** represents the set of linked pairs obtained by the data linkage algorithm, and ***D**** is the set of gold-standard linking results. The algorithm is regarded as perfect if the two sets are identical, i.e., ***D=D****.

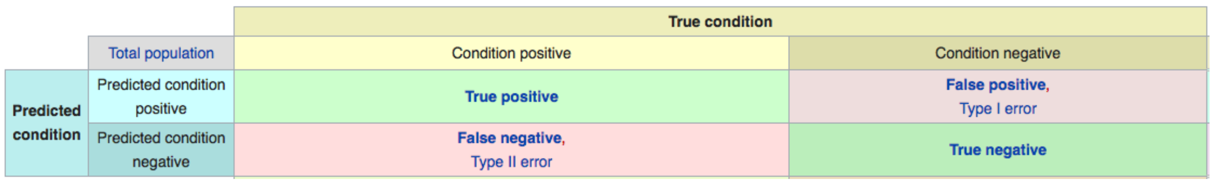
Precision and **Recall** are well-known performance measures that can capture the above intuitions. For classification tasks, the terms ***true positives***, ***true negatives***, ***false positives***, and ***false negatives*** are considered to compare the results of the classifier under test with trusted external judgments (i.e., **gold-standard**). The terms ***positive*** and ***negative*** refer to the classifier's prediction (sometimes known as the expectation), and the terms ***true*** and ***false*** refer to whether that prediction corresponds to the external judgment (sometimes known as the observation), as shown below. Given a linked pair ***id1_id2*** in this Prac, it is like:

True positive, if it belongs to both D and D^*

False positive, if it only belongs to D

False negative, if it only belongs to D^*

True negative, if it belongs to neither D nor D^*



		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error
	Predicted condition negative	**False negative**, Type II error	**True negative**

Figure 2. T*rue positives, true negatives, false positives,* and *false negatives*

Based on these terms, precision and recall are defined as follows, where ***tp, fp,*** and ***fn*** represent true positive, false positive, and false negative, respectively.

$$Precision = \frac{tp}{tp + fp}$$

$$Recall = \frac{tp}{tp + fn}$$

Often, there is an inverse relationship between precision and recall, where it is possible to increase one at the cost of reducing the other. For example, brain surgery can be used as an illustrative example of the trade-off. Consider a brain surgeon tasked with removing a cancerous tumour from a patient's brain. The surgeon needs to remove all of the tumour cells since any remaining cancer cells will regenerate the tumour. Conversely, the surgeon must not remove healthy brain cells since that would leave the patient with impaired brain function. The surgeon may be more liberal in the area of the brain she removes to ensure she has extracted all the cancer cells. This decision increases recall but reduces precision. On the other hand, the surgeon may be more conservative in the brain she removes to ensure she extracts only cancer cells. This decision increases precision but reduces recall. It can be seen that, greater recall increases the chances of removing healthy cells (negative outcome) and increases the chances of removing all cancer cells (positive outcome). Greater precision decreases the chances of removing healthy cells (positive outcome) but also decreases the chances of removing all cancer cells (negative outcome).

Therefore, another performance measure is used to consider the both precision and recall together, namely, F-measure, in which precision and recall are combined by their harmonic mean, as shown below.

$$F = 2 \times \frac{precision \times recall}{precision + recall}$$

Program *measurement.py* class implements above performance measures, namely precision, recall, and f-measure. Please read and understand the "***calc_measure***" function in *measurement.py*.

Part 3: Similarity Measures for Field-Level Data Linkage

In above Part 2, we link two restaurant records only if their names are identical. However, datasets, in practice, are always informal and noisy, full of abbreviations, spelling mistakes, various entity representations, etc. Therefore, a better solution would be to calculate the similarity between records, which is also the main issue of data linkage. We consider ***Jaccard coefficient*** in Task 3 to estimate the similarity between restaurant names, so to link corresponding restaurant records. You will see later how the similarity measure affect the performance of data linkage.

Jaccard Coefficient

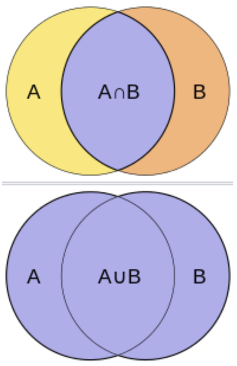


Figure 3. An illustration of the Jaccard Coefficient calculation

Jaccard coefficient, also known as Jaccard index or Intersection over Union, is a statistic method used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Here, A and B are two sample sets, and **J(A, B)=1** if A and B are both empty. As you can see, the Jaccard coefficient compares members for two sets to see which members are shared and which are distinct. It measures the similarity between two sets, with a range from 0% to 100%. The higher the percentage, the more similar the two sets. Consider the following simple example, how similar are these two sets?

- A = {0,1,2,5,6}
- B = {0,2,3,4,5,7,9}

Obviously, J(A,B) = |A∩B| / |A∪B| = |{0,2,5}| / |{0,1,2,3,4,5,6,7,9}| = 3/9 = 0.33.

In order to measure the similarity between two strings using Jaccard coefficient, the strings need to be converted into sets firstly. In this Prac, we consider ***q-gram*** representation of a string. Q-gram is a contiguous sequence of *q* items from a given string. The items can be phonemes, syllables, characters, or words according to the application. A q-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram"; size 3 is a "trigram" (or **3-gram**). Larger sizes are sometimes referred to by the value of *q* in modern language, e.g., "four-gram", "five-gram", and so on. Consider the string "University_of_Queensland", we can transform it as a set of 3-grams with each character as the item:

- "**University_of_Queensland**": {"##U", "#Un", Uni", "niv", "ive", "ver", "ers", "rsi", "sit", "ity", "ty_", "y_o", "_of", "of_", "f_Q", "_Qu", "Que", "uee", "een", "ens", "nsl", "sla", "lan", "and", "nd#", "d##"},

where using character "#" is optional to pad the beginning and ending of the string for the completeness of 3-gram. The Jaccard coefficient similarity measure based on q-grams has been implemented in the file *similarity.py*.

<u>Task 3</u>: In *nested_loop_by_name_jaccard.py,* we link two restaurant records if the Jaccard coefficient of corresponding restaurant names exceeds a predefined threshold. Run *nested_loop_by_name_jaccard.py* with different settings of "q" and "threshold" to see how these two parameters affect the output of the similarity measure, and therefore affect the performance of data linkage in terms of the output size and measurement result.

Test the influence of each parameter by altering its value to five different settings (up to your choice, but make sure your values are valid) as well as fixing the other parameter. Please modify the path in the script accordingly. Do such process to both parameters and screenshot all your precision, recall and f-measure results. Explain why the precision/recall increases/decreases based on your understanding. **Note** that, q=0 means we divide the original string into a bag-of-words.