

Assignment 3: Duplicate Detection

Submitted By: **Group B**

Name	Matriculation Nummer
Abhishek Shrestha	390055
Jia Jia	389917
Syed Salman Ali	395898

1. Tasks Overview:

The main task is to detect all the duplicate tuples that refer to the same real world entities.

This involves two tasks:

1. Detect duplicate with brute-force
2. Partition based duplicate detection

Task 1: Brute-Force Duplicate Detection:

Setting up the environment:

For this part of the assignment we used C# since we already did some part of this in the first assignment so we modified parts of the same code.

Tasks

1. **Provide an algorithm. Specify the input, output, similarity function, and time complexity.**

Algorithm:

- Get the location of the csv file
- Read the sample size
- Read the csv file according to the sample size
- Loop through each row (tuple)

- Create a nested loop for the column value we want to compare with other tuples (in our case we selected “SSN”)
- Compare the tuple pairs
- If the column value being compared is empty then ignore the row and move ahead
- If match is found then write the “RecID” of both the tuple pairs to the output file.
- Move the matched tuple to a list to avoid selecting the tuple again.

Input:

Since we are comparing values for duplicates so we cleaned up the *InputDb* first.

We used OpenRefine (like in assignment 3) to clean-up the SSN value (the one we are comparing against). The rule we used was: the SSN should be a 6 digit value without any separators in between.

We therefore used the cleaned up table. The input file is attached with the submission bundle.

Sampling

We tried to use the whole input file and we waited for 1.5 hr to finish but the process was still continuing so we cancelled and took a sample data set instead.

The sample size in our case was: 50,000

Output:

Output.csv file which contained the detected tuple pairs

Similarity Function:

Since we are using SSN for comparison we are looking for an exact match.

Time Complexity:

Because of the nested loop the time complexity is: $O(n^2)$

2. Implement the algorithm and report the precision, recall, F1, and runtime.

Implementation:

It was not possible to read the whole csv file since it was taking too long to process so we ended up sampling the table. The sample size as mentioned above is 50,000.

Precision, recall and F1:

Below is screenshot with details on the results.

```
abhis@MandirVenio MINGW64 /d/TU/_SEM 3/800-6/Data Inte
$ cd "d:\TU\_SEM 3\800-6\Data Integration\DataIntegrat
n "c:\Users\abhis\.vscode\extensions\ms-python.python-
n\DataIntegration\Assignments\Assignment 3\WebClient"
Integration\DataIntegration\Assignments\Assignment 3\
Duplicate Detection Results:
Precision = 0.999956920691
Recall = 0.187780362018
F1 = 0.316184858897
```

Precision: ~ 100%

Recall: 18.78%

F1: 31.61%

Runtime:

We first tried on 50K data and the process took about 20 minutes to complete.

Then we tried to use the whole data set (~90K) results. We waited for more than an hour but the process did not complete. So we considered 50K as our final sample size.

```
file:///D:/TU/_SEM 3/800-6/Data Integration/DataIntegration/Assignments/
Start time: 28/06/2018 22:07:57
28/06/2018 22:25:09
The total time taken is: 17.2032630766667 Minutes.
```

3. What is the upsides and downsides of this method?

Advantage:

- Simple and easy to implement without any complexity.

Disadvantage:

- Not suitable for larger data set.

- The implementation was not using full CPU (and memory) even though there was available processing power, so it is a bit non efficient. We are not sure why this is happening since the windows thread manager should have handled this automatically, may be it is problem with our code but we could not increase the memory consumption or process consumption to decrease the processing time.

>	Google Chrome (13)	16.0%	760.9 MB	0.1 MB/s	0.1 Mbps	0%	
>	Steam Client Bootstrapper (32 bit) (5)	0.2%	132.9 MB	0 MB/s	0 Mbps	0%	
>	Microsoft Word	0%	77.1 MB	0 MB/s	0 Mbps	0%	
>	Antimalware Service Executable	0.9%	56.1 MB	0 MB/s	0 Mbps	0%	
>	vshost.exe (2)	26.8%	53.6 MB	0.1 MB/s	0 Mbps	0%	
>	Windows Shell Experience Host	0%	44.1 MB	0 MB/s	0 Mbps	0%	GPU 0 - 3D
	Desktop Window Manager	0.3%	34.1 MB	0 MB/s	0 Mbps	0.1%	GPU 0 - 3D
>	Windows Explorer	0%	32.0 MB	0 MB/s	0 Mbps	0%	
>	Service Host: Diagnostic Policy Service	0%	26.8 MB	0 MB/s	0 Mbps	0%	
>	Task Manager	0%	23.3 MB	0 MB/s	0 Mbps	0%	
>	Microsoft Visual Studio 2008 (32 bit) (3)	0%	22.7 MB	0 MB/s	0 Mbps	0%	
	SmartAudio (32 bit)	0%	18.1 MB	0 MB/s	0 Mbps	0%	
>	Microsoft Office Click-to-Run (SxS)	0%	13.5 MB	0 MB/s	0 Mbps	0%	
	Lenovo Photo Master Update (32 bit)	0%	11.6 MB	0 MB/s	0 Mbps	0%	
>	Microsoft Windows Search Indexer	0%	11.3 MB	0 MB/s	0 Mbps	0%	
	Lenovo.Modern.ImController.PluginHost	0%	9.1 MB	0 MB/s	0 Mbps	0%	
>	Service Host: DCOM Server Process Launcher (4)	0%	7.8 MB	0 MB/s	0 Mbps	0%	

Task 2: Partition-Based Duplicate Detection:

Setting up the environment:

We used python with entity matching API for this part of the assignment.

Setting up the entity matching consumed more time than we ever expected. Actually, coding this part of assignment took less time than setting up the environment.

Below are the steps we followed:

1. Uninstalled python and every dependency
(There was an error: "fatal error lnk1112 module machine type 'x86' conflicts with target"
We tried different setting with Linker but it did not work so we had to uninstall python)
2. Install Python x86 then pip and then *py_entitymatching* with:
install py_entitymatching
3. We then upgraded pip with
pip install --upgrade pip
(Tried to install *entitymatching* without the upgrade but there was an error saying "could not find a version that satisfies the requirement in python" so upgrade is necessary)
4. Install/ upgrade the following dependencies:
 - a. Upgrade "ipython":
pip install --upgrade IPython
 - b. Install "requests":
pip install requests
 - c. Install "scipy"
pip install scipy

Tasks

1. **Provide an algorithm. Specify the input, output, similarity function, and time complexity.**

Algorithm:

- Read the input csv file
- Entity matching assumes we have two data sets so for our case both the data sets are the InputDB provided.
- The input data is too long so we need to down sample.
"In *py_entitymatching*, you can use sample the input tables using *down_sample* command. This command samples the input tables intelligently that ensures a reasonable number of matches between them."

- Once the tables are loaded and downsampled, the next step is blocking. This is done to block a tuple pair from going through to the matching step. When applied to a tuple pair, a blocker returns *True* if the pair should be blocked.
We used attribute equivalence blocker in the column "SSN" in order to block the two tables to produce a candidate set of tuple pairs.
SSN is used because it is (should be unique) to all values and if this is same then chances are the pair are dupes. However, we did not consider possibilities of error in "SSN" column itself.
- Remove the same *RecID* matches (since we are using save csv files there are self-comparisons, remove them)
- The missing values in the blocking attribute are ignored and not handled separately including all possible tuple pairs with missing values can significantly increase the size of the candidate set.
- Once blocking is done we can then group together the possible de-dup pairs.

Input:

The cleaned up input file (same one which was used in task 1).

Output:

The output is the Output.csv file that contains the duplicate pairs.

Time Complexity:

Time complexity is $O(n)$.

Similarity Function:

We used the attribute equivalence blocker to block two tables to produce candidate set of tuple pairs.

For the given two tables attribute equivalence blocker takes one attribute and compares the tuples based on that attribute such that if the two attribute differ then the pairs cannot be duplicates at all.

2. Implement the algorithm and report the precision, recall, F1, and runtime.

Implementation:

Source is attached with the submission.

Precision and recall:

```
abhis@MandirVenio MINGW64 /d/TU/_SEM 3/800-6/Data Integration/DataInte
$ cd "d:\TU\_SEM 3\800-6\Data Integration\DataIntegration\Assignments\
n "c:\Users\abhis\.vscode\extensions\ms-python.python-2018.6.0\pythonF
n\DataIntegration\Assignments\Assignment 3\WebClient" 51043 34806ad9-8
Integration\DataIntegration\Assignments\Assignment 3\WebClient\web_cl
Duplicate Detection Results:
Precision = 1.0
Recall = 0.00310243705127
F1 = 0.00618568340914
```

Precision: 1

Recall: 0.003

F1: 0.00618

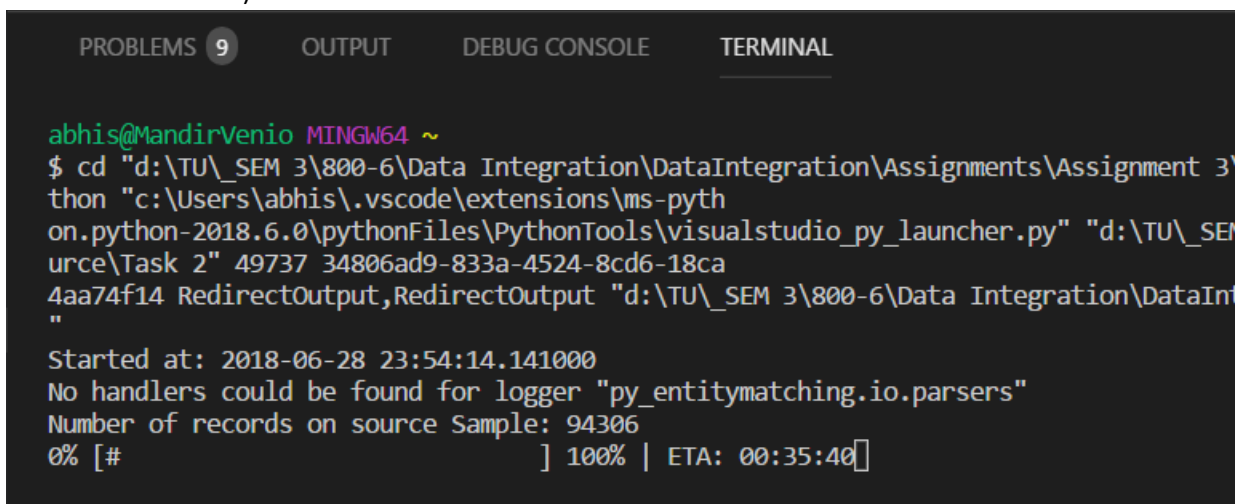
The very low recall in this case is due to low sample size (10,000) we tried to increase the sample size but this caused the time to increase drastically.

One of the improvements we can do is that instead of using the blocker to get candidate set of tuple pairs, we can use the blocker to only eliminate the records that cannot be dups (may be multiple blockers) then use matchers to get the final results. But we could not implement it since it was a bit complex and we have enough time to R&D into that.

Runtime:

Total Time: 46 mins (approx.)

- 35 min (approx.) for down-sampling with 10,000 sample size (for about 50,000 the ETA was 3 hrs!!)



```
PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL

abhis@MandirVenio MINGW64 ~
$ cd "d:\TU\_SEM 3\800-6\Data Integration\DataIntegration\Assignments\Assignment 3\
python "c:\Users\abhis\.vscode\extensions\ms-pyth
on.python-2018.6.0\pythonFiles\PythonTools\visualstudio_py_launcher.py" "d:\TU\_SEM
urce\Task 2" 49737 34806ad9-833a-4524-8cd6-18ca
4aa74f14 RedirectOutput,RedirectOutput "d:\TU\_SEM 3\800-6\Data Integration\DataInt
"

Started at: 2018-06-28 23:54:14.141000
No handlers could be found for logger "py_entitymatching.io.parsers"
Number of records on source Sample: 94306
0% [#                               ] 100% | ETA: 00:35:40[]
```

- ~ 10 min for blocking and writing csv.

```

Sample Head...
  RecID FirstName MiddleName LastName ... POBox POCityStateZip SSN DOB
32770 A937464 TRENT J REYNOLDS ... PO BOX 76972 TAMPA, FL 33675 65681937.0 NaN
32772 A937466 THOMAS NaN BERNAL ... NaN 3932779.0 30-Jul-23
49158 A953852 VUTHY A CROSS ... PO BOX 2503 PALATKA, FL 32178 103405572.0 01-11-34
16391 A921085 ESTHER NaN SPENCER ... NaN 7854971.0 NaN
65548 A970242 STEPHIE NaN VAZQUEZ ... UZON 780 WINCHESTER, CA 92596 NaN 19340801

[5 rows x 12 columns]

Sample Properties are:
id: 238434832
key: RecID
id: 238434352
key: RecID

After Blocking...
  _id L_RecID R_RecID L_SSN R_SSN
0 0 A937464 A937464 65681937.0 65681937.0
1 1 A937466 A937466 3932779.0 3932779.0
2 2 A953852 A953852 103405572.0 103405572.0
3 3 A921085 A921085 7854971.0 7854971.0
4 4 A921085 A995576 7854971.0 7854971.0
Ended at:2018-06-29 00:40:19.634000
Total time taken: 46.0

```

3. What is the upsides and downsides of this method?

Advantages:

- Easy to implement.
- Suitable for even larger data sets.
- It seems that py_entitymatching uses full CPU availability so it is ideal to use for large datasets and powerful machines.

Task Manager							
File Options View							
Processes Performance App history Startup Users Details Services							
Name	Status	100% CPU	46% Memory	2% Disk	0% Network	0% GPU	GPU Engine
> Visual Studio Code (161)		37.1%	1,392.4 MB	0 MB/s	0 Mbps	0%	GPU 0 - 3D
python.exe (32 bit)		27.1%	145.2 MB	0 MB/s	0 Mbps	0%	
> vshost.exe (2)		26.2%	83.4 MB	0.1 MB/s	0 Mbps	0%	
System		3.0%	0.1 MB	0.1 MB/s	0 Mbps	0%	
> Task Manager		1.8%	28.3 MB	0 MB/s	0 Mbps	0%	
> Antimalware Service Executable		0.9%	68.3 MB	0 MB/s	0 Mbps	0%	
WMI Provider Host		0.6%	5.4 MB	0 MB/s	0 Mbps	0%	
Client Server Runtime Process		0.6%	0.8 MB	0 MB/s	0 Mbps	0.1%	GPU 0 - 3D
> Service Host: Windows Management Instrumentation		0.5%	9.0 MB	0 MB/s	0 Mbps	0%	
TortoiseGit status cache		0.3%	2.3 MB	0 MB/s	0 Mbps	0%	
Adobe RdrCEF (32 bit)		0.3%	3.2 MB	0 MB/s	0 Mbps	0%	
Desktop Window Manager		0.3%	62.8 MB	0 MB/s	0 Mbps	0.1%	GPU 0 - 3D
System interrupts		0.2%	0 MB	0 MB/s	0 Mbps	0%	
Discord (32 bit)		0.2%	44.5 MB	0 MB/s	0 Mbps	0%	
> Microsoft Visual Studio 2008 (32 bit) (2)		0.2%	28.2 MB	0 MB/s	0 Mbps	0%	
> Windows Explorer (6)		0.2%	61.0 MB	0 MB/s	0 Mbps	0%	
Lenovo.Modern.ImController.PluginHost		0.2%	7.1 MB	0 MB/s	0 Mbps	0%	

^ Fewer details
End task

Disadvantages:

- Down sampling might cause less reliability.
- We did not observe any improvement in speed so not fast than brute force.

References:

Really nice read about the blockers:

- http://anhaidgroup.github.io/py_entitymatching/v0.1.x/user_manual/blocking.html
- <https://openproceedings.org/2018/conf/edbt/paper-82.pdf>

Everything about the Entity Matching:

- http://anhaidgroup.github.io/py_entitymatching/v0.3.x/index.html

Everything Else:

- <https://medium.com/district-data-labs/basics-of-entity-resolution-with-python-and-dedupe-bc87440b64d4>
- http://anhaidgroup.github.io/py_entitymatching/v0.1.x/user_manual/guides.html
- https://sites.google.com/site/anhaidgroup/projects/magellan/py_entitymatching