Task 2

July 3, 2018

```
In [4]: import pandas as pd
        import numpy as np
        import requests
        import fuzzy
        import re
        Soundex =fuzzy.Soundex(1)
```

```
In [5]: def evaluate(csv):
            url = "http://gorilla.bigdama.tu-berlin.de:8001/DuplicateDetectionEvaluator/"
            output = open("output.csv", "r").read()
            r = requests.post(url, data={"value": output})
            if r.status_code != 200:
                print ("There is an error! The error code:", r.status_code)
            print (r.text)
```

0.1 Task 2: Partition-Based Duplicate Detection

Here, we want to reduce the time complexity of brute-force approach by dividing dataset into partitions. Thus, instead of comparing all the possible pairs of tuples in the whole dataset, we need to compare tuples just inside of their partition. 1. Provide an algorithm. Specify the input, output, similarity function, and time complexity. 2. Implement the algorithm and report the precision, recall, F1, and runtime. 3. What is the upsides and downsides of this method?

0.1.1 Algorithm Input :

- Treshold value for decision-making if the pair is semantically equal
- Size of the window

0.1.2 Output:

- Duplicates in a two column table

We've decided to use the "Sorted Neighborhood" algorithm by Hernandez Stolfo in which we: 1. Generate a key for sorting based on First, Last Name and SSN 3. Sort the whole table and capitalize all chars 2. Slide window over sorted tuples and compare all the rows within a window

Time Complexity Considering that each record in data table needs to be sorted first we need add "n*log(n)". Further we are doing the row comparision between every combinations (not permutations!) possible - 10!/(8! 2!) = 45 .

Let window size be k, we have then n*log(n)* + *k*n = n(log(n) + k) ### Results - Unfortunately the method did not as good as expected. The precision of the found results was quite high however the algorithm was able to find just a percentile of the duplicates, thus Recal is around "0,5%" - This methods enables to boost the computations and return quite resonable result at leas 10 times faster than Brute Force approach. However in order to work properly the developer must:

- try various parameter combinations ie. diffrent windowing stategies ()
- Tresholds
- Weights to adjust the similiarity between records

For the last point it can be defnitel beneficial to apply ML algorithm i.e Naive Baysian

```
In [121]: df = pd.read_csv("./inputDB.csv")
          df.head()
```

Out[121]:

	RecID	FirstName	MiddleName	LastName	Address	\
0	A904694	christine	NaN	urias	5404 ROCKLAND DR	
1	A904695	RICHARD	S	STOKES	2384 MOUNTAIN PINE RD	
2	A904696	michelle	NaN	daughtry	2301 plumeria lane	
3	A904697	A	NaN	GARFIO	1305 NORTH FOYD STR LOT 10	
4	A904698	MIHCAEL	L	VALENCIA	3 HOPE DR	

	City	State	ZIP	POBox	POCityStateZip	\
0	pearland	tx	77584	NaN	NaN	
1	Hot	Springs	Ar 7191	NaN	NaN	
2	Palmdale	Ca	93551	PO MBOX 901243	palmdale, ca 93590	
3	JONESBOROY	AR	72401	NaN	NaN	
4	WATSONVILLE	CA	95076	P Box 1353	watsonvile, calif. 95077	

	SSN	DOB
0	166-91-0767	NaN
1	NaN	04-12-72
2	086-71-1424	1956
3	22542011	NaN
4	157146562	NaN

```
In [66]: #Uppercase whole file
         df =df.apply(lambda x: x.astype(str).str.upper())


         # Add a Soundex generaed hashcode based on the name for sorting
         df['soudex_name'] = df.apply(lambda row: Soundex(str(row.FirstName))+" "+Soundex(str(ro
         df['sorting_key'] = df.apply(lambda row: "{} {}".format(row.soudex_name,str(row.SSN)),

         df = df.sort_values(by=["sorting_key","ZIP"])
         df.head()
```

Out[66]:

	RecID	FirstName	MiddleName	LastName	Address	City	State	ZIP	POBox	POCityStateZip	SSN	DOB	soudex_name	sorting_key
27668	A932362	ANNETTE	NAN	ALEMAN	960 22ND AVE N,	NAPLES	FLORIDA	3W4103	NAN	NAN	000-35-0599	NAN	A A	A A 000-35-0599
9827	A914521	AMERICA	NAN	ALCALA	2484 CILE LN	MARIANNA	FL	32446	PO BOX 6D27	MARIANA, FL 32447	005-63-4412	NAN	A A	A A 005-63-4412
35689	A940383	A	NAN	ALCALA	619 CHARLES SISE ST	LEHIGH	ACRES	FL 33974	CALLER 507	LEHIGH ACRES, FLORIDA 33970	005-63-4412	NAN	A A	A A 005-63-4412
2398	A907092	A	NAN	ALBORNOZ	420 NSHORE RD	VENICE	FL	34285	NAN	NAN	005-75-0057	NAN	A A	A A 005-75-0057
69848	A974542	ANA	NAN	ALVAREZ	1102 REENWOOD DRV	DENTON	TX	76209	NAN	NAN	006-38-1750	NAN	A A	A A 006-38-1750

In [8]:
```
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
```

In [9]:
```
def compare_rows(a,b):
    try:
        fn = [str(a['FirstName']),str(b['FirstName'])]
        ln = [str(a['LastName']),str(b['LastName'])]
        ZIP_ = [str(a['ZIP']),str(b['ZIP'])]
        SSN = [str(a['SSN']),str(b['SSN'])]
        Address = [str(a['Address']),str(b['Address'])]
        City = [str(a['City']),str(b['City'])]
        return [fuzz.ratio(pair[0],pair[1]) for pair in [fn,ln,ZIP_,SSN,Address,City]]
    except TypeError as E:
        print("Error : "+str(E))
        return [0,0,0,0,0];
```

In [56]:

Out[56]:

	RecID	FirstName	MiddleName	LastName	Address	City	State	ZIP	POBox	POCityStateZip
27668	A932362	ANNETTE	NAN	ALEMAN	960 22ND AVE N,	NAPLES	FLORIDA	3W4103	NAN	NAN
9827	A914521	AMERICA	NAN	ALCALA	2484 CILE LN	MARIANNA	FL	32446	PO BOX 6D27	MARIANA, FL 32447
35689	A940383	A	NAN	ALCALA	619 CHARLES SISE ST	LEHIGH	ACRES	FL 33974	CALLER 507	LEHIGH ACRES, FLORIDA 33970
2398	A907092	A	NAN	ALBORNOZ	420 NSHORE RD	VENICE	FL	34285	NAN	NAN
69848	A974542	ANA	NAN	ALVAREZ	1102 REENWOOD DRV	DENTON	TX	76209	NAN	NAN

```
35689    ACRES  FL 33974   CALLER 507  LEHIGH ACRES, FLORIDA 33970
2398        FL     34285          NAN                          NAN
69848       TX     76209          NAN                          NAN

                SSN  DOB soudex_name
27668  000-35-0599  NAN         A A
9827   005-63-4412  NAN         A A
35689  005-63-4412  NAN         A A
2398   005-75-0057  NAN         A A
69848  006-38-1750  NAN         A A
```

In [126]:
```
duplicates = pd.DataFrame(columns=['tuple_id_1','tuple_id_2'])
treshold = 0.52 * 100
# Each 10 Elements block
windowing_range = range((df.shape[0] + 9) // 10)
for i in windowing_range:
    X_subset = df.iloc[i * 10: (i + 1) * 10]
    ctr = 1
    if i % 200 == 199:
        duplicates = duplicates.drop_duplicates()
        print(str(round(100*(i/windowing_range[-1]),2))+"% Found:",duplicates.shape[0]
    for idx, candidate in X_subset.iterrows():
        #Remove self reference
        #id_vertex = np.array([x['RecID'] for idx,x in X_subset.iterrows()])
        #temp_set = X_subset.drop(X_subset.index[ctr])
        #print("###")
        for idy,y in X_subset[ctr:].iterrows():
            mean = np.array(compare_rows(candidate,y)).mean()
            # print (vec,y["RecID"],candidate['RecID'])
            if mean > treshold:
                #print(mean, treshold)
                duplicates = duplicates.append({'tuple_id_1': y["RecID"],'tuple_id_2':
        ctr += 1
        #print("###")

duplicates = duplicates.drop_duplicates()
evaluate(duplicates.to_csv(index=False) )
duplicates.to_csv("TEST_DUPLI_1.csv",index=False)
```

```
2.11% Found: 2
4.23% Found: 4
6.35% Found: 8
8.47% Found: 10
10.59% Found: 17
12.71% Found: 19
14.84% Found: 19
16.96% Found: 20
19.08% Found: 22
```

```
21.2% Found: 27
23.32% Found: 27
25.44% Found: 27
27.56% Found: 29
29.68% Found: 31
31.8% Found: 32
33.92% Found: 35
36.04% Found: 38
38.17% Found: 43
40.29% Found: 45
42.41% Found: 47
44.53% Found: 51
46.65% Found: 53
48.77% Found: 53
50.89% Found: 55
53.01% Found: 60
55.13% Found: 60
57.25% Found: 62
59.37% Found: 64
61.5% Found: 67
63.62% Found: 70
65.74% Found: 71
67.86% Found: 71
69.98% Found: 72
72.1% Found: 76
74.22% Found: 77
76.34% Found: 80
78.46% Found: 84
80.58% Found: 86
82.7% Found: 88
84.83% Found: 89
86.95% Found: 93
89.07% Found: 94
91.19% Found: 95
93.31% Found: 99
95.43% Found: 102
97.55% Found: 103
99.67% Found: 105
Duplicate Detection Results:
Precision = 1.0
Recall = 8.08979674386e-06
F1 = 1.61794625991e-05
```

```
In [ ]: duplicates.to_csv("TEST_DUPLI_1.csv",index=False)
```

0.1.3 Resolve all possible transitions

```
In [135]: def get_transitons(tuples):
              transitons = pd.DataFrame(columns=['tuple_id_1','tuple_id_2'])

              for A,B in tuples[['tuple_id_1','tuple_id_2']].values:
                  # E*-> A-> B -> C*
                  #where starts on B
                  C_= tuples.loc[tuples['tuple_id_1'] == B]['tuple_id_2'].values
                  #B_C = [[B,C] for C  in C_ ]
                  for x,y in [[B,C] for C  in C_ ]:
                       transitons=transitons.append({'tuple_id_1': x,'tuple_id_2': y}, ignore_in
                  #where where end on A
                  E_ =tuples.loc[tuples['tuple_id_2'] == A]['tuple_id_1'].values

                  for x,y in [[E,A] for E in E_ ]:
                      transitons=transitons.append({'tuple_id_1': x,'tuple_id_2': y}, ignore_ind

              #transitons = transitons.drop_duplicates()
              return transitons

In [136]: transitions = get_transitons(duplicates)
          print(str(duplicates.shape[0])+" added"+str(transitions.shape[0])+" transitons")

105 added0 transitons


In [133]: duplicates.append(transitions)
          evaluate(duplicates.to_csv(index=False))

Duplicate Detection Results:
Precision = 1.0
Recall = 8.08979674386e-06
F1 = 1.61794625991e-05
```

0.1.4 Backup Code

```
In [ ]: #Split words in the n-grams
        def ngrams(string, n=2):
            string = re.sub(r'[/]|\sBD',r'', string)
            ngrams = zip(*[string[i:] for i in range(n)])
            return [''.join(ngram) for ngram in ngrams]


        #compute dice for array or string
        def dice_coefficient(a, b,n=2):
            """dice coefficient 2nt/na + nb."""
```

```
#For comparing strings
if isinstance(a,str) & isinstance(b,str):
    a = ngrams(a,n)
    b = ngrams(b,n)
try:
#For comparing lists
    a_bigrams = set(a)
    b_bigrams = set(b)
    overlap = len(a_bigrams & b_bigrams)
    return overlap * 2.0/(len(a_bigrams) + len(b_bigrams))
except ZeroDivisionError as e:
    return 0.0
```