Task_1

May 20, 2018

```
In [37]: import pandas as pd
         import numpy as np

         import re
         from __future__ import division
```

Tomasz Tkaczyk (Group R) - 368998

0.1 1. Label Based Mapping

Here, we want to find the correspondences between the columns from the two datasets with the help of only schema headers. 1. Provide an algorithm. Specify the input, output, similarity function, and time complexity. 2. Implement the algorithm and report the results. Is there any parameter that affects the results? 3. What is the upsides and downsides of this method? When does it work and when not?

0.1.1 Solution:

Dice-distance on N-grams

get_mapping is the main function of the algortihm. It takes: - source and target data tables (padans DataFrame - n which adjusts how many letter dose the splic (n-gram) contains - treshold value for the dice score

The decription of the algorithm is in the comments.

It first takes two tuples of labes and splits each column name in n-letter long splits. Than for each of the column label computes score based on the dice-distance. All the possible matches that are beyond certain treshold are selected. Whenever there is more than one 'matching' column it takes the one with the greater dice score .

```
In [59]: im= pd.read_csv("./imdb.csv");
         rt= pd.read_csv("./rotten_tomatoes.csv");
         print('The imdb shape: %d x %d' % im.shape)
         print('The rt shape: %d x %d' % im.shape)
```

```
The imdb shape: 6407 x 13
The rt shape: 6407 x 13
```

0.2 General Truth

In [60]:
```
#Ibmd : RottenTomatoes
GT = {
      "Name":"Name",
      "ReleaseDate":"Release Date",
      "RatingValue":"RatingValue",
      "Director":"Director",
      "Creator":"Creator",
      "YearRange":"Year",
      "Genre":"Genre",
      "Duration":"Duration",
      "Cast":"Cast",
        "Description":"Description"
     }
```

In [61]:
```
#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)

    #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))
```

In [62]:
```
dice_coefficient("Name","Nameee")
```

Out[62]:
```
0.8571428571428571
```

In [63]:
```
def get_mapping(table_a,table_b,treshold=0.5,n=2):
    #print("Treshold: {}, {}-grams".format(treshold,n))
    A =[ str(a) for a in table_a.columns]
    B =[ str(b) for b in table_b.columns]
    #HardCoded skiping IDs
    A.remove('Id')
    B.remove('Id')
```

```
            #List of arrays containg header label and it's N-gram represatantion,
            rt_cl = [[ngrams(cl,n),cl]for cl in A]
            im_cl = [[ngrams(cl,n),cl] for cl in B]
            df = pd.DataFrame(columns=["tab_a_entry","tab_b_entry","coeff"])
            for r,r_ in rt_cl:
                #For each columns in rotten
                for i,i_ in im_cl:
                    #For each columns in IMDB
                    dc = dice_coefficient(r,i)
                    #If the score is above the treshold or the same exact word is
                    if (dc >= treshold ) or (r_ in i_):
                        #Append to the final table
                        df = df.append(pd.Series([r_,i_,dc],index=["tab_a_entry",'
            #Group by first column of sourcre and select the best results for eac
            df = df.groupby(['tab_a_entry'], sort=True)['tab_a_entry','tab_b_entry
            return dict(zip(df["tab_a_entry"],df['tab_b_entry']))
```

In [64]:
```
get_mapping(im,rt)
```

Out[64]:
```
{'Cast': 'Cast',
 'ContentRating': 'RatingCount',
 'Creator': 'Creator',
 'Description': 'Description',
 'Director': 'Director',
 'Duration': 'Duration',
 'Genre': 'Genre',
 'Name': 'Name',
 'RatingValue': 'RatingValue',
 'ReleaseDate': 'Release Date',
 'YearRange': 'Year'}
```

0.2.1 Evaluation Methods

In [65]:
```
def compute_recall(mapping):
    total_found =len(mapping.items())
    #print("Recall {}".format(float(total_found/len(GT.items()))))
    return float(total_found/len(GT.items()))

def compute_precision(mapping):
    precision = set(mapping.items()) & set(GT.items())
    true_matches = len(precision)
    total_found =len(mapping.items())
    #print("Precision {}".format(float(true_matches/total_found)))
    return float(true_matches/total_found)
```

In [66]:
```
def what_is_missing_or_wrong(mapping):
    #Helper method
    def _removekey(d, key):
        r = dict(d)
```

```
            del r[key]
            return r

        missing = list()
        for x in GT.items():
            try:
                mapping= _removekey(mapping,x[0])
            except KeyError as e:
                #print("Missing : "+x[0])
                missing.append(x)

        return missing, mapping
```

In [67]:
```
#For treshold between 0.001 and 1
treshold_range = [0.001,1.001];
#For 1- to 4-grams
ngrams_range= [1,5]


def benchmark_algo(tableA,tableB,to_csv=False):
    treshold_test_axis = np.arange(treshold_range[0],treshold_range[1] , .
    ngram_test_axis = np.arange(ngrams_range[0],ngrams_range[1])

    df = pd.DataFrame(columns=["treshold","ngrams","precision","recall","t
    for t in treshold_test_axis:
        for n in ngram_test_axis:

            mp =get_mapping(tableA,tableB,t,n)
            prec =compute_precision(mp)
            recall= compute_recall(mp)
            unique = set(mp.items())
            df = df.append(pd.Series([t,n,prec,recall,len(mp.items()),len
    df['precision_normalised']=(df['precision'] - df['precision'].mean())
    df['recall_normalised']=  (df['recall'] - df['recall'].mean()) / (df['
    if to_csv:
        df.to_csv('test.csv')
    return df
```

In [68]:
```
test_ = test_algo(im,rt,True)
```

In [50]:
```
#Perfect match !
test_[(test_['recall']==1.0)&(test_['precision']==1.0)]
```

Out[50]:

	treshold	ngrams	precision	recall	total_matches	uniqie_matches	\
82	0.401	3.0	1.0	1.0	10.0	10.0	
86	0.421	3.0	1.0	1.0	10.0	10.0	
90	0.441	3.0	1.0	1.0	10.0	10.0	

	precision_normalised	recall_normalised

```
82              0.267942            0.01125
86              0.267942            0.01125
90              0.267942            0.01125
```

1 Evaluation results:

Pros: - it found the perfect matching - performs well for 3- and 4-grams

Cons: - Hard coded skiping Ids

This method performs well in cases when the header row names in both tables include simmilar words. Diffrent order of these words or diffren sepatator does not affect the algorithm. Works best for 3-grams, with relatively low treshold of 0.4.

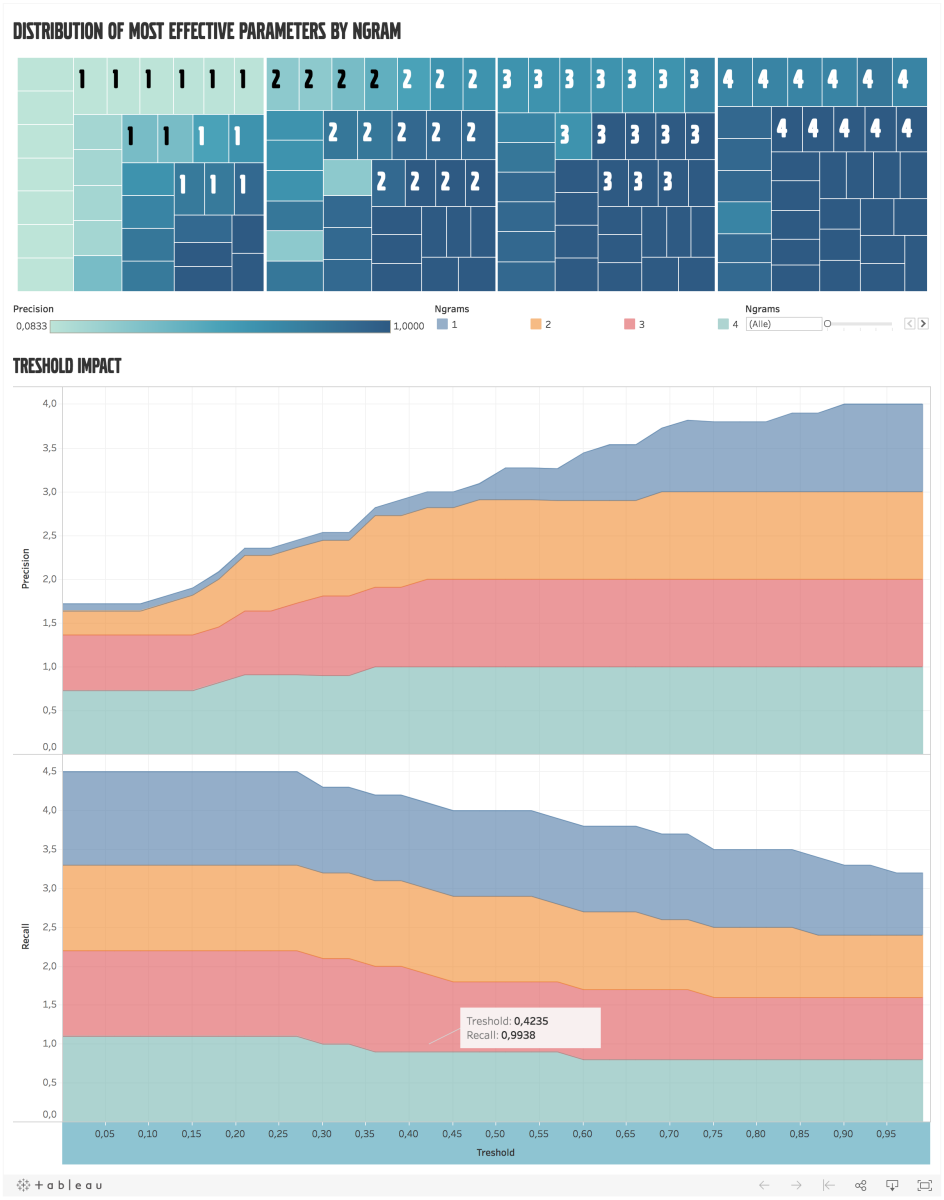
1.0.1 Vizualisatuon :

In order to see how does the algorithm perform for diffrent combination of parameters i've made a small benchmark. I've tested it out using 200 combinations of paramters and stored the results in csv. The visualisation can be found here:

https://public.tableau.com/views/Viz_41/Dashboard1?:embed=y&:display_count=yes

```
In [51]: from IPython.display import Image
         Image(filename='./screen.png')
```

Out[51]:



Viz

9 Ansichten | Tomasz Tkaczyk

Mehr Details

Zuletzt gespeichert: 19.05.2018
Arbeitsmappendetails: 6 Blätter
Ursprünglicher Autor: Tomasz Tkaczyk

Metadaten: Distribution of most effective parameters by NGRAM
Compare Two
Sheet 3
Dashboard 1
Best Configs
Treshold Impact

```
In [52]: #test_.to_csv("test.csv")
```

```
In [ ]:
```