Written Mutual Intelligibility in Europe

Jacek Pardyak

5 september 2018

Motivation

```
#![](https://upload.wikimedia.org/wikipedia/commons/c/c6/Europe_%28orthographic_projection%29.svg)
```

Europe is a continent with approximately 750 million people who communicate using approximately 100 languages. Five of the languages have more than 50 million native speakers: French, Italian, German, English, and Russian. Twenty four languageses are official languages of the EU: Bulgarian, Croatian, Czech, Danish, Dutch, English, Estonian, Finnish, French, German, Greek, Hungarian, Irish, Italian, Latvian, Lithuanian, Maltese, Polish, Portuguese, Romanian, Slovak, Slovene, Spanish, and Swedish. In this notebook we try to see the similarities and differences between the most important languages in Europe. We will analyze the relationships between the dictionaries used for spell checking in Libre Office.

We believe that the results may be helpful in verifying theses put forward by Mutual Intelligibity researchers. For more information, see https://en.wikipedia.org/wiki/Mutual_intelligibility.

Set and vector similarity measures

In statistics and related fields, a similarity measure or similarity function is a real-valued function that quantifies the similarity between two objects.

In the context of this work we require:

$$0 \leq similarity(X,Y) \leq 1$$

$$similarity(X,Y) = 0 \Leftrightarrow X = Y$$

$$similarity(X,Y) = 1 \Leftrightarrow X \cap Y = \emptyset$$

In this notebook we will use Tversky, Sørensen–Dice, Jacard and Overlap similarity measures.

Tversky index

For sets X and Y the Tversky index is a number between 0 and 1 given by

$$S_{\alpha,\beta}(X,Y) = 1 - \frac{|X \cap Y|}{|X \cap Y| + \alpha|X - Y| + \beta|Y - X|}$$

Tversky index is not a proper distance metric as it does not satisfy the symmetry principle. For more information, see https://en.wikipedia.org/wiki/Tversky_index .

Sørensen–Dice index

Sørensen–Dice index is special case of Tversky similarity index for $\alpha = \beta = 0.5$. The calculation formula simplifies to:

$$S_{\alpha=\beta=0.5}(X, Y) = 1 - \frac{2|X \cap Y|}{|X| + |Y|}$$

Sørensen–Dice index is not a proper distance metric as it does not satisfy the triangle inequality. The simplest counterexample of this is given by the three sets $\{a\}$, $\{b\}$, and $\{a, b\}$ and indices of all pairs

$$S_{\alpha=\beta=0.5}(\{a\}, \{b\}) = 1$$

$$S_{\alpha=\beta=0.5}(\{a\}, \{a, b\}) = \frac{1}{3}$$

To satisfy the triangle inequality, the sum of any two of these three sides must be greater than or equal to the remaining side. However \$1/3 +1/3 <1 \$. For more information, see https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient

Jacard index

Jacard index is special case of Tversky similarity index for $\alpha = \beta = 1$. The calculation formula simplifies to:

$$S_{\alpha=\beta=1}(X, Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}$$

Jacard index is a proper distance metric as it satisfies identity of indiscernibles, symmetry and triangle inequality. For more information, see https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient

Overlap index

The overlap index] or Szymkiewicz–Simpson coefficient, is related to the Jaccard index and is defined as follows:

$$overlap(X, Y) = 1 - \frac{|X \cap Y|}{min(|X|, |Y|)}$$

For more information, see https://en.wikipedia.org/wiki/Overlap_coefficient .

similarities between pairs of dictionaries

We start from getting the dictionaries from Libre Office Github repository with following snippet:

```
import pandas as pd
import numpy as np
languages = pd.DataFrame(data = np.matrix([

['Bulgarian' , 'bg_BG', 'bg_BG', 'bg_BG'      , 'utf_8'],
['Croatian'  , 'hr_HR', 'hr_HR', 'hr_HR'      , 'iso-8859-2'],
['Czech'     , 'cs_CZ', 'cs_CZ', 'cs_CZ'      , 'iso-8859-2'],
['Danish'    , 'da_DK', 'da_DK', 'da_DK'      , 'iso-8859-1'],
['Dutch'     , 'nl_NL', 'nl_NL', 'nl_NL'      , 'iso-8859-1'],
```

```
['English'   , 'en_GB', 'en'   , 'en_GB'      , 'iso-8859-1'],
['Estonian'  , 'et_EE', 'et_EE', 'et_EE'      , 'iso-8859-15'],
['Finnish'   , pd.NA  ,  pd.NA ,  pd.NA       ,  pd.NA],
['French'    , 'fr_FR', 'fr_FR', 'fr'         , 'iso-8859-15'],
['German'    , 'de_DE', 'de'   , 'de_DE_frami', 'iso-8859-1'],
['Greek'     , 'el_GR', 'el_GR', 'el_GR'      , 'iso-8859-7'],
['Hungarian' , 'hu_HU', 'hu_HU', 'hu_HU'      , 'utf_8'],
['Irish'     , pd.NA  ,  pd.NA ,  pd.NA       ,  pd.NA],
['Italian'   , 'it_IT', 'it_IT', 'it_IT'      , 'iso-8859-15'],
['Latvian'   , 'lv_LV', 'lv_LV', 'lv_LV'      , 'iso-8859-13'],
['Lithuanian', 'lt_LT', 'lt_LT', 'lt'         , 'iso-8859-13'],
['Maltese'   , pd.NA  , pd.NA  ,  pd.NA       ,  pd.NA],
['Polish'    , 'pl_PL', 'pl_PL', 'pl_PL'      , 'iso-8859-2'],
['Portuguese', 'pt_PT', 'pt_PT', 'pt_PT'      , 'iso-8859-1'],
['Romanian'  , 'ro_RO', 'ro'   , 'ro_RO'      , 'iso-8859-2'],
['Slovak'    , 'sk_SK', 'sk_SK', 'sk_SK'      , 'iso-8859-2'],
['Slovenian' , 'sl_SI', 'sl_SI', 'sl_SI'      , 'iso-8859-2'],
['Spanish'   , 'es_ES', 'es'   , 'es_ES'      , 'iso-8859-1'],
['Swedish'   , 'sv_SE', 'sv_SE', 'sv_SE'      , 'iso-8859-1'],
['Russian'   , 'ru_RU', 'ru_RU', 'ru_RU'      , 'utf_8'],
['Ukrainian' , 'uk_UA', 'uk_UA', 'uk_UA'      , 'utf_8']]),

columns = ['language', 'code', 'folder', 'sub_folder', 'encoding'])

languages = languages.dropna().reset_index()
languages
```

```
##     index    language   code folder   sub_folder     encoding
## 0       0   Bulgarian  bg_BG  bg_BG        bg_BG        utf_8
## 1       1    Croatian  hr_HR  hr_HR        hr_HR   iso-8859-2
## 2       2       Czech  cs_CZ  cs_CZ        cs_CZ   iso-8859-2
## 3       3      Danish  da_DK  da_DK        da_DK   iso-8859-1
## 4       4       Dutch  nl_NL  nl_NL        nl_NL   iso-8859-1
## 5       5     English  en_GB     en        en_GB   iso-8859-1
## 6       6    Estonian  et_EE  et_EE        et_EE  iso-8859-15
## 7       8      French  fr_FR  fr_FR           fr  iso-8859-15
## 8       9      German  de_DE     de  de_DE_frami   iso-8859-1
## 9      10       Greek  el_GR  el_GR        el_GR   iso-8859-7
## 10     11   Hungarian  hu_HU  hu_HU        hu_HU        utf_8
## 11     13     Italian  it_IT  it_IT        it_IT  iso-8859-15
## 12     14     Latvian  lv_LV  lv_LV        lv_LV  iso-8859-13
## 13     15  Lithuanian  lt_LT  lt_LT           lt  iso-8859-13
## 14     17      Polish  pl_PL  pl_PL        pl_PL   iso-8859-2
## 15     18  Portuguese  pt_PT  pt_PT        pt_PT   iso-8859-1
## 16     19    Romanian  ro_RO     ro        ro_RO   iso-8859-2
## 17     20      Slovak  sk_SK  sk_SK        sk_SK   iso-8859-2
## 18     21   Slovenian  sl_SI  sl_SI        sl_SI   iso-8859-2
## 19     22     Spanish  es_ES     es        es_ES   iso-8859-1
## 20     23     Swedish  sv_SE  sv_SE        sv_SE   iso-8859-1
## 21     24     Russian  ru_RU  ru_RU        ru_RU        utf_8
## 22     25   Ukrainian  uk_UA  uk_UA        uk_UA        utf_8
```

```
for i in range(21,23) : #  range(15,16) languages.shape[0]
  if i == 15:
```

```
    continue
#  print(i)
  url = "https://raw.githubusercontent.com/LibreOffice/dictionaries/master/" + languages['folder'][i] +
  encoding = languages['encoding'][i]
  if languages['folder'][i] == 'en':
    data = pd.read_csv(filepath_or_buffer = url, encoding = encoding, comment = '/', skiprows = [21066,
  else:
    if languages['folder'][i] == 'de':
      data = pd.read_csv(filepath_or_buffer = url, encoding = encoding, comment = '/', skiprows = [1, 2
    else:
      if languages['folder'][i] == 'hu_HU':
        data = pd.read_csv(filepath_or_buffer = url, encoding = encoding, comment = '/', skiprows = [935
      else:
        if languages['folder'][i] == 'sv_SE':
          data = pd.read_csv(filepath_or_buffer = url, encoding = encoding, comment = '/', skiprows = [4
        else:
          data = pd.read_csv(filepath_or_buffer = url, encoding = encoding, comment = '/'  )
  data.to_csv('./data_in/' + languages['code'][i] + '.csv', index = False, encoding = encoding, header =
```

```
# Portuguese
from io import StringIO
import requests
import re

i = 15
url = "https://raw.githubusercontent.com/LibreOffice/dictionaries/master/" + languages['folder'][i] + "/
encoding = languages['encoding'][i]

f = requests.get(url)
str = f.text
str_out = str[200:len(str)]
str_out = re.sub(r'\[.*\]', '', str_out)
url = StringIO(str_out)

df = data = pd.read_csv(filepath_or_buffer = url, encoding = encoding, comment = '/', skiprows = [11, 30

data.to_csv('./data_in/' + languages['code'][i] + '.csv', index = False, encoding = encoding, header = T
```

Now we can load the clean data:

```
# load the data
files = languages['code'].tolist()

for file in files:
  encoding = languages.loc[languages['code'] == file]['encoding'].to_list()[0]
  url = './data_in/' + file + '.csv'
  df = pd.read_csv(url, encoding = encoding, header = None)
  df = df[df.columns.values[0]].tolist()
  globals()[file] = df
```

We count number of elements in set differences and intersection of each language pair:

```
import itertools
pairs = list(itertools.combinations(files, 2))
```

```
df = pd.DataFrame()

for pair in pairs: # pairs[0:3]
  left_count = list(set(globals()[pair[0]]).difference((globals()[pair[1]])))
  intersect_count = list(set(globals()[pair[0]]).intersection(globals()[pair[1]]))
  right_count  = list(set(globals()[pair[1]]).difference(globals()[pair[0]]))
  res = {'lang_l' : [pair[0]], 'lang_r' : [pair[1]], 'count_l' : [len(left_count)], 'count_i' : [len(in
  df = df.append(pd.DataFrame(data = res), ignore_index=True)

df.head()
```

```
##   lang_l lang_r  count_l  count_i  count_r
## 0  bg_BG  hr_HR    78238        0    53597
## 1  bg_BG  cs_CZ    78238        0   165084
## 2  bg_BG  da_DK    78238        0   139053
## 3  bg_BG  nl_NL    78238        0   135254
## 4  bg_BG  en_GB    78238        0    89790
```

We calculate `overlap`, `sørensen`, and `jacard` index of each language pair:

```
# calculate measures
tmp = pd.DataFrame()
for i in range(df.shape[0]):
  res = {
  'index_overlap' : [1 - (df.iloc[i]['count_i'])/max(df.iloc[i]['count_l'], df.iloc[i]['count_r'])],
  'index_sorensen' : [1 - (2*df.iloc[i]['count_i'])/(df.iloc[i]['count_l'] + 2*df.iloc[i]['count_i'] + 
  'index_jacard'   : [1 - (df.iloc[i]['count_i'])/(df.iloc[i]['count_l'] + df.iloc[i]['count_i'] + df.il
  }
  tmp = tmp.append(pd.DataFrame(data = res), ignore_index=True)

df_ind = pd.concat([df.reset_index(drop=True), tmp], axis = 1)

df_ind.to_csv('./data_out/indices.csv', index = False, header = True)
```

Hierarchical Clustering of Languages

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). In data mining and statistics, hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. To illustrate the arrangement of the clusters produced by the corresponding analyses we dendrograms.

For more information, see https://en.wikipedia.org/wiki/Cluster_analysis and https://en.wikipedia.org/wiki/Hierarchical_clustering.

```
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
import matplotlib
matplotlib.rc('font', family='Arial')
from matplotlib import pyplot as plt

labelList = list(languages.code.values)
indices = pd.read_csv('./data_out/indices.csv',)

linkage_matrix = linkage(y = indices['index_overlap'].values)
```

```
plt.figure(figsize=(15, 15))
dendrogram(Z = linkage_matrix,
            orientation = 'right',
            labels = labelList,
            distance_sort = 'descending',
            show_leaf_counts = False)
```

```
## {'icoord': [[25.0, 25.0, 35.0, 35.0], [15.0, 15.0, 30.0, 30.0], [75.0, 75.0, 85.0, 85.0], [105.0, 105
```

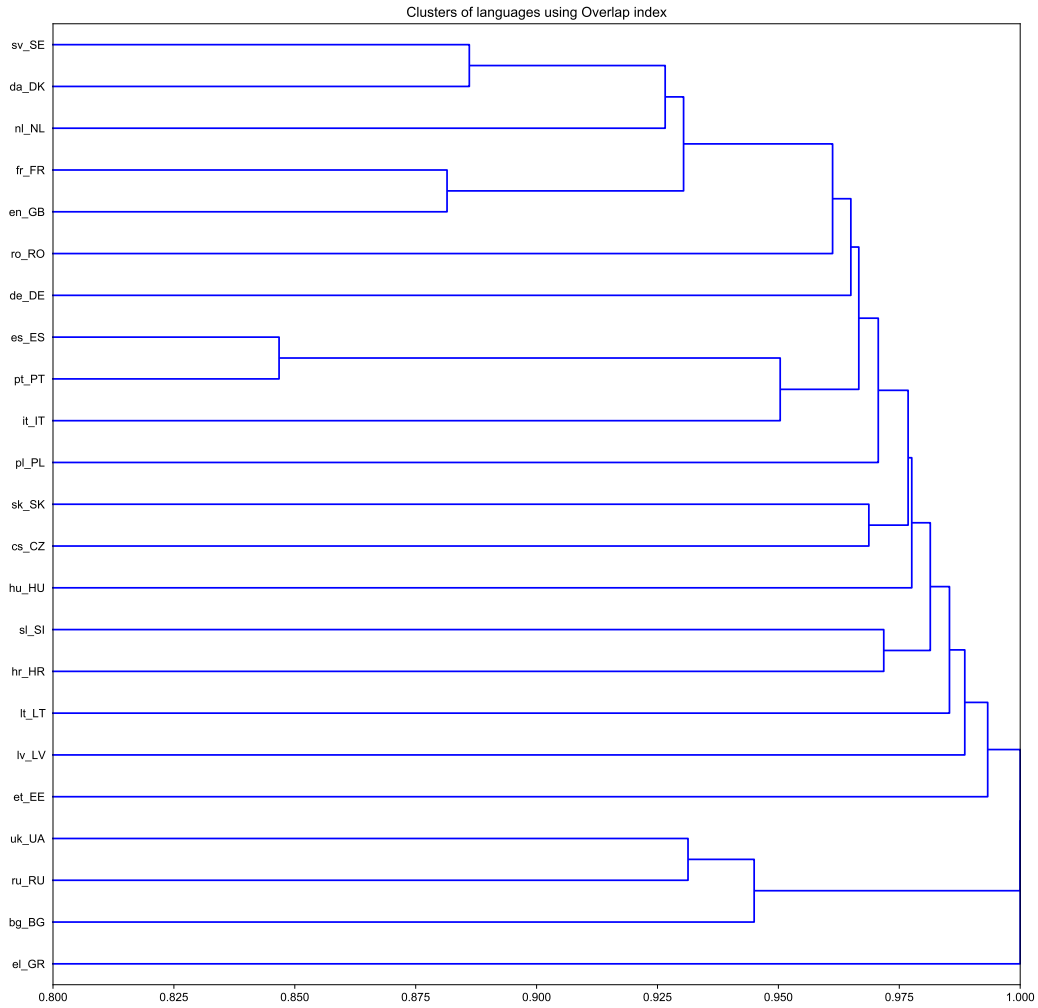
```
plt.title(label = u"Clusters of languages using Overlap index")
plt.xlim(left = 0.8)
```

```
## (0.8, 1.05)
```

```
plt.xlim(right = 1.0)
```

```
## (0.8, 1.0)
```

```
plt.show()
```




```
plt.close()
```

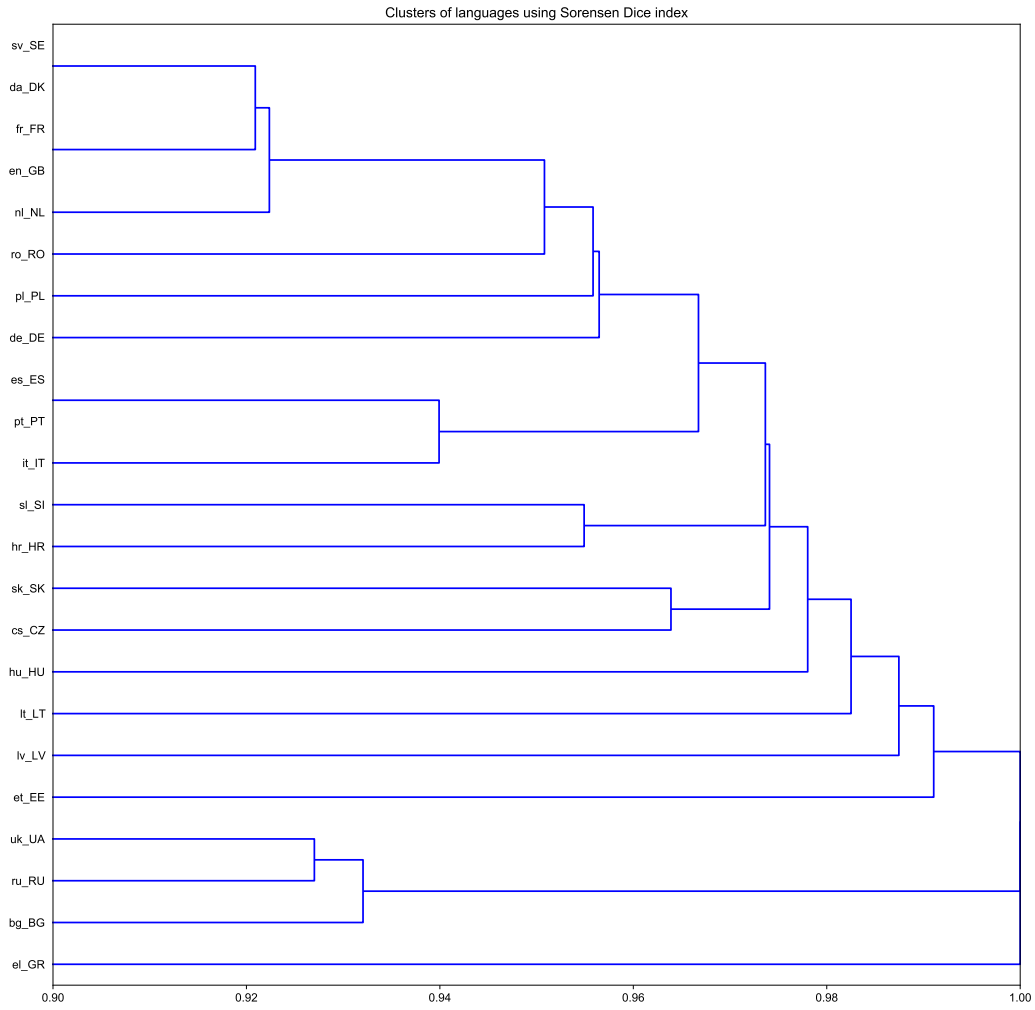
```
linkage_matrix = linkage(y = indices['index_sorensen'].values)
plt.figure(figsize=(15, 15))
dendrogram(Z = linkage_matrix,
           orientation = 'right',
           labels = labelList,
           distance_sort = 'descending',
           show_leaf_counts = False)
```

```
## {'icoord': [[25.0, 25.0, 35.0, 35.0], [15.0, 15.0, 30.0, 30.0], [85.0, 85.0, 95.0, 95.0], [105.0, 105
```

```
plt.title(label = 'Clusters of languages using Sorensen Dice index')
plt.xlim(left = 0.9)
```

```
## (0.9, 1.05)
plt.xlim(right = 1.0)
```

```
## (0.9, 1.0)
plt.show()
```




```
plt.close()

# # r'Sorensen
```

```
linkage_matrix = linkage(y = indices['index_jacard'].values)
plt.figure(figsize=(15, 15))
dendrogram(Z = linkage_matrix,
           orientation = 'right',
```

```
            labels = labelList,
            distance_sort = 'descending',
            show_leaf_counts = False)
```

```
## {'icoord': [[25.0, 25.0, 35.0, 35.0], [15.0, 15.0, 30.0, 30.0], [85.0, 85.0, 95.0, 95.0], [105.0, 105
```

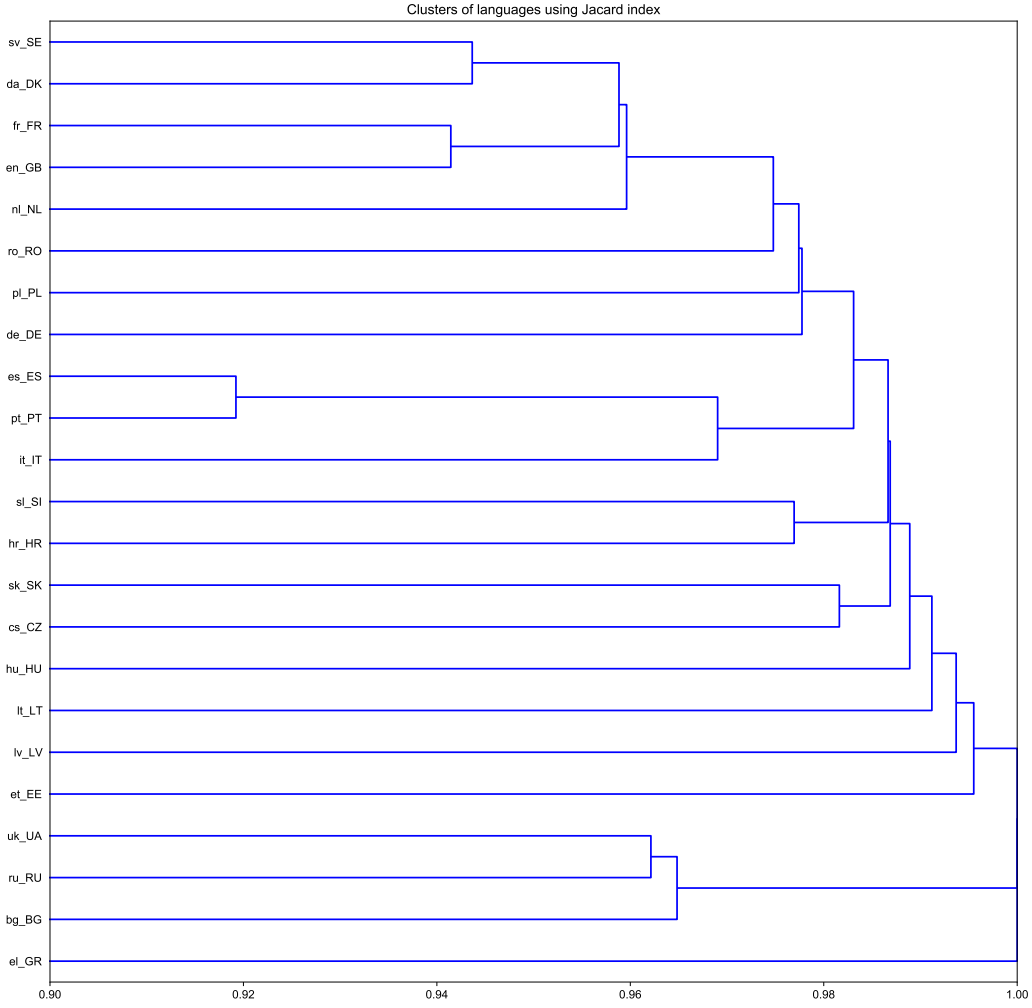
```
plt.title(label = "Clusters of languages using Jacard index")
plt.xlim(left = 0.9)
```

```
## (0.9, 1.05)
```

```
plt.xlim(right = 1.0)
```

```
## (0.9, 1.0)
```

```
plt.show()
```



```
plt.close()
```

Summary

Some interesting insights from HCA:

- Spanish and Portugese pair share the largest in the EU number of common words,
- Slavic languages family (Polish, Czech and Slovak) indeed share common words,
- Saying that Dutch is just a mixture between English and German is inaccurate.
- Saying 'That's greek to me' is really justified :)

```
import requests
from requests.auth import HTTPDigestAuth
import json

# Replace with the correct URL
url = "https://webgate.ec.europa.eu/inspire-sdi/srv/eng/catalog.search#/metadata/6d5e861e-d50c-4e8c-854
url = "https://webgate.ec.europa.eu/inspire-sdi/srv/eng//resources.get?uuid=41fc901e-1524-47fe-9162-4bc
# It is a good practice not to hardcode the credentials. So ask the user to enter credentials at runtim
myResponse = requests.get(url)
myResponse
#jData = json.loads(myResponse.content)
#jData
```

```
## <Response [200]>
```

```
import requests
r = requests.get('https://ec.europa.eu/eurostat/cache/GISCO/distribution/v1/nuts-2016.json').json()
r
```

```
## {'csv': {'NUTS_AT_2016': '../v2/nuts/csv/NUTS_AT_2016.csv', 'NUTS_RG_BN_01M_2016': '../v2/nuts/csv/NU
```

```
mDF <- as.matrix(mDist)
pl_PL <- mDF[, 'pl_PL']
pl_PL[order(pl_PL, decreasing = FALSE)]
```

```
nl_NL <- mDF[, 'nl_NL']
nl_NL[order(nl_NL, decreasing = FALSE)]
```

```
en_GB <- mDF[, 'en_GB']
en_GB[order(en_GB, decreasing = FALSE)]
```

There are some limitations of this approach:

- we count words in two vocabularies that look the same, but differ significantly in meaning (false friends). For example: 'hak' (pl_PL) - 'hook' (en_GB) but 'hak' (nl_NL) - 'heel' (en_GB)
- we don't count words in two vocabularies that look similar and mean the same. For example: 'banan'(pl_PL) - 'banaan'(nl_NL) - 'banana'(en_GB)

In the next article I will present hierarchical cluster analysis using 'partial word match' instead of 'exact word match'.

Part II

Fuzzy set intersection

$$x \in V_1 \overline{\cap} V_2 : \forall x \in V_1 \exists y \in V_2 \quad d(x,y) \leq \epsilon$$

Definitions

alphabet - set of letters

word (w) - sequence of elements from an alphabet

vocabulary (V) - set of words

words similarity - two words x and y are similar if $d(x,y) \leq \epsilon$ with regard to predetermined distance $d(x,y)$ and threshold ϵ.

common vocabulary - for two vocabularies V_1 and V_2 elements of two subsets $\overline{V_1} \subseteq V_1$ and $\overline{V_2} \subseteq V_2$ meet criterion:

$$\forall x \in \overline{V_1} \exists y \in \overline{V_2} : d(x,y) \leq \epsilon$$

distance between vocabularies - ratio of cardinality $\overline{V}_1 \cup \overline{V}_2$ and cardinality of $V_1 \cup V_2$. More precisely:

$$d(V_1, V_2) = 1 - \frac{|\overline{V_1} \cup \overline{V_2}|}{|V_1 \cup V_2|}$$

In the special case of distance defined as $d(x,y) = 0$ if $x = y$, and 1 otherwise and threshold $\epsilon = 0$, (perfect match) we get

$$d(V_1, V_2) = 1 - \frac{|V_1 \cap V_2|}{|V_1 \cup V_2|}$$

The data

The EU has 24 official languages used by people within 28 member states. Vocabularies were retrieved from ftp://ftp.snt.utwente.nl/pub/software/openoffice/contrib/dictionaries/, part of Hunspell project http://hunspell.github.io/. Two of 24 official EU languages, Finnish and Maltese don't have available vocabularies. We use ISO 639-2 codes of those `length(names)` languages: `names`.

Clustering vocabularies using exact word matching

```
library(rgdal)
library(raster)
library(rgeos)
#sjer_aoi_WGS84 <- spTransform(sjer_aoi,
#                              crs(nycounties))

# what is the CRS of the new object
crs(nycounties)
## CRS arguments:
##  +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
# does the extent look like decimal degrees?
extent(sjer_aoi_WGS84)
```

```
mat <- as.matrix(mDist)
#mat[lower.tri(mat)] <- ''
knitr::kable(mat[,2:8], caption = "Vocabularies distance using exact word matching")
knitr::kable(mat[,9:15], caption = "Vocabularies distance using exact word matching cntd")
knitr::kable(mat[,16:22], caption = "Vocabularies distance using exact word matching cntd")
```