Report - Surname Assignment

Oscar Galindo

Dr. Nigel Ward

Topics in Language Processing

September 3, 2020

Surname Assignment

Introduction:

For this assignment a set of rules known as *regular expressions* was implemented to map surnames to countries of origin. The rules implemented were derived from a file that contained samples of last names that were ordered by country of origin. The *regular expressions* are different that simple configurations of strings, because regular expressions match an architecture rather than pieces of the architecture as the substrings did.

Design and Implementation:

Rationale for *precision* increase: In order to increase the precision of the network my thinking was to create regular expressions that addressed particular word forms of the surnames. At the same time an attempt was made to create balanced regular expressions that tried to capture general patterns from the language not just a specific surname. So, in general the extraction of patterns is half based on substrings, but also on the structure of the strings of every language. Essentially, this decreases the false positives by better detecting the patterns of every language.

With this thinking the classifier achieved a 0.10 accuracy score. Knowing that the baseline probability of randomly choosing a category and being right is around 0.05 this is a good score.

Rationale for *recall* increase: In order to increase recall the regular expressions for every language were modified. The modifications applied were based on the idea that we want to avoid that patterns that some of the surnames show are matched in languages of which they are not part of. To do this there was a further review of the regular expression, elements like boundaries were included, regular expressions that contained dots were eliminated but for the two language that uses 3 character strings in this case Chinese and Korean. Formally speaking, "limits" were implemented into the rules to further adjust regular expression to follow the architecture of the language. Furthermore, the limitations were also added to more specific rules for every language, these rules also helped to increase the

With this thinking the classifier achieved a 0.17 accuracy score. Knowing that the previous score for precision was 0.10 achieving this accuracy was a great improvement over the previous iteration

Regular Expressions:

Comment: To create the set of regular expressions the examples in "**surnames-dev.csv**" were used to define the substrings for every language.

- Arabic:
    - Regular Expressions: r’\b(.+)ou.*\b',r'\bs.*f\b',r'\b(m|f|s).*(f|d|b)\b',r'\b.*abi\b',r'\b.*(sar|har).*\b',r'\b.*a.{1,2}a.*\b',r'\bg.{3,5}m\b',r'\b.*imi\b',r'\b.*tro.*\b'

- Chinese:
    - Regular Expressions: r’\b.+-.+\b',r'\b.+(a|e)ng\b',r'\b.iu\b',r'\b.+ze.+\b',r'\b.ei\b'

- Czech:
    - Regular Expressions: r’\b.*(j|z).*k\b',r'\bi.+r\b',r'\b.*(w|v).*sk(y|i)\b',r'\b.+ova\b'

- Dutch:
    - Regular Expressions: r'\b(s|r|p|a).+er\b',r'\bk.+n\b',r'\bn.*k\b'

- English:
    - Regular Expressions: r’\b.+kin.*\b',r'\b.+spoo.*\b',r'\b.*ear.*\b',r'\b.+rough.*\b',r'\b.+(i|e|a)ll\b',r'\b.+more.*\b',r'\b.+son.*\b',r'\b(c|s|r|m|f).+er.*\b',r'\b.+man(n)\b',r'\b.*eat.*\b'

- French:
    - Regular Expressions: r'\b.+onn.*\b',r'\b.*eaux.*\b',r'\ble.+\b',r'\b.+quet.*\b',r'\b.+amps\b',r'\b(b|p|c|t).+er\b',r'\b.+asso.+\b',r'\b.*agn.+\b'

- German:
    - Regular Expressions: ‘(r|t|g).+','(g|s|h|k|p|l).+l','st.+r','(b|f|h).+er','.*ott.*','.+eier','.+chard.*','.*omm.*','.+(echt|ach).*',".+ö.*"

- Greek:
    - Regular Expressions: r'\b.+(gas|ous|oulos)\b',r'\b.+os\b',r'\b.+is\b',r'\b.+os\b'

- Irish:
    - Regular Expressions: r'\bo.+',r'\b.+ha(n|nn)\b',r'\b.+ll.*\b',r'\b.*a.{1,3}a(n|nn)\b',r'\b.*owe.*\b'

- Italian:
    - Regular Expressions: r'\b.+icci\b',r'\b.+ieri\b',r'\b.+achi\b',r'\b(a|p|u|z|n).*(ri|i)\b',r'\b.*oia\b'

- Japanese:
    - Regular Expressions: r'\b.*i(c|s).*da\b',r'\b.*yama.*\b',r"\b.*i(s|k)i\b",r"\bs.+m",r".+a(w|h)a\b"

- Korean:
    - Regular Expressions: r'\b(..|...)\b',r'\bch.{3,5}\b',r'\bh.{2,3}ng*\b',

- Polish:
    - Regular Expressions: r'\bk.*ki\b',r'\bka.+a\b',r"\bg.*r.*k\b",r"\b.*ń","k.{0,1}ó\b"

- Portuguese:
  - Regular Expressions: r"\b.+veira",r'\b(p|c|a).*o\b'

- Russian:
  - Regular Expressions: r".*nov\b","\b(v|j|z|d|p|c).{5,9}sky\b"

- Scottish:
  - Regular Expressions: r".+son\b",r".+ant\b",".+ight.*"

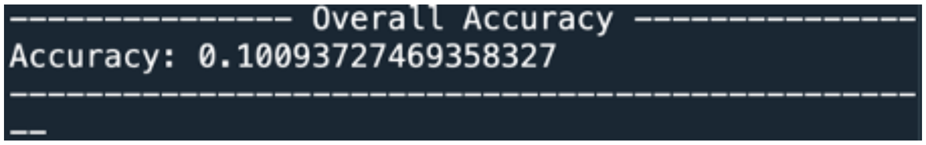
- Spanish:
  - Regular Expressions: '.*cal',r'.+e(r|rr)a\b',r'\b san.*','.*(á|ó|í|é|ú).*',r"\b(a|h|o).+a"

- Vietnamese:
  - Regular Expressions: '.*kim',r'\b.*(h|a).+h'

Implementation: Found in predictor.py

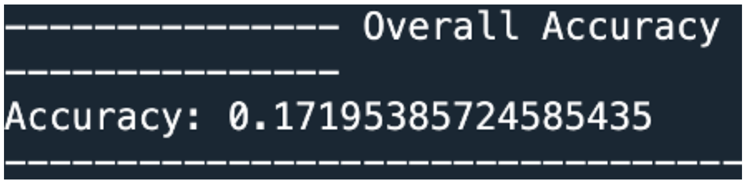
Experimental Results:

- Testing for precision optimization classification in only the provided samples in "**surnames-dev.csv**"
  - Results:
    - Accuracy = ~0.10



```
--------------- Overall Accuracy ---------------
Accuracy: 0.10093727469358327
------------------------------------------------
--
```

- Testing for recall optimization classification in only the provided samples in "**surnames-dev.csv**
  - Results:
    - Accuracy = ~0.17

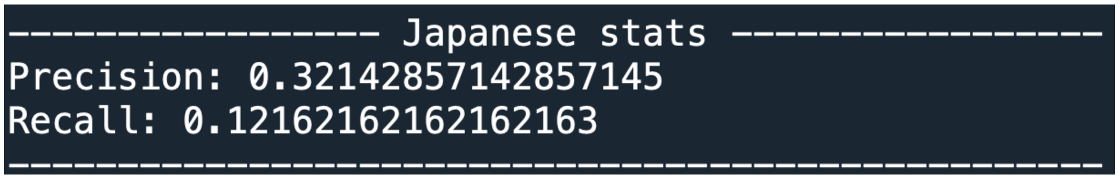


```
--------------- Overall Accuracy
---------------
Accuracy: 0.17195385724585435
----------------------------------------
```

Conclusions:

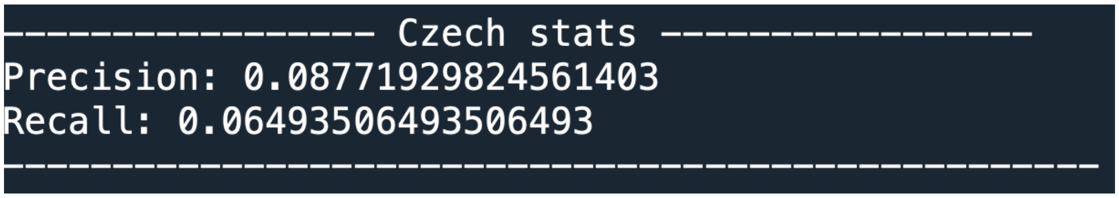
- Based on the data provided it seems the previous classifier, which was based on common substrings among surenames, actually performs better than the classifier based on general expressions.

- Good identifications, like those for Japanese could be based mainly on the fact that few general expressions exist but the samples have mostly one of two or three repeated "architectures".



```
---------------- Japanese stats ----------------
Precision: 0.32142857142857145
Recall: 0.12162162162162163
------------------------------------------------
```

- Bad Identifications might be a result of having a lot of variability between the structures of surenames, and since for increasing recall we eliminate rules that could be too general (e.g. by applying boundaries) some languages' regular expressions might lack any prediction power.



```
---------------- Czech stats ----------------
Precision: 0.08771929824561403
Recall: 0.06493506493506493
---------------------------------------------
```

- From the current perspective it seems that it could be an conclusion to say that we should select substrings to obtain a better classifier, but it seems that regular expression have more space to 'evolve' and become more complex and hence increase the prediction power.

- The accuracy of the predictor seems interesting as it is around 4 times as much as the probability of choosing the correct language for a surname if guessed by anybody.

Appendix (Python 3.8):

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 27 17:52:13 2020

@author: oscargalindo
"""

import random,re
```

```
def is_arabic(name):

strings=['fa','eif','ha','al','ba','ou','uo','ui','sar','dd','ad','ha','ni','nem','ki''zz','az','azz','am','sha
','ai','ouf']
    patterns = [r'\b(.+)ou.*\b',r'\bs.*f\b',r'\b(m|f|s).*(f|d|b)\b',r'\b.*abi\b',r'\b.*(sar|har).*\b',r'\b.*a.
{1,2}a.*\b',r'\bg.{3,5}m\b',r'\b.*imi\b',r'\b.*tro.*\b',]
    name = name.lower()
    for s in patterns:
        if re.search(s,name):
            return True
    # name = name.lower()
    # for s in strings:
    #     if s in name:
    #         return True
    return False

def is_chinese(name):
    strings=['ao','oa','wa','jin','qi','dai','Tz','ai','ia','an','xi','oo','ang','ni','in','sh']
    patterns = [r'\b.+-.+\b',r'\b.+(a|e)ng\b',r'\b.iu\b',r'\b.+ze.+\b',r'\b.ei\b']
    name = name.lower()
    for s in patterns:
        if re.search(s,name):
            return True
    return False

def is_czech(name):

strings=['ka','ff','sek','wa','sch','deh','ss','jj','ik','pp','vl','ski','ze','rich','cova','tak','ovsky','warz
','schm','ich','witz','pp']
    patterns = [r'\b.*(j|z).*k\b',r'\bi.+r\b',r'\b.*(w|v).*sk(y|i)\b',r'\b.+ova\b']
    name = name.lower()
    for s in patterns:
        if re.search(s,name):
            return True
    return False

def is_dutch(name):
    strings=['ee','jer','aye','oo','gg','ee','aa','ker','ss','nn','kk','ke']
    patterns = [r'\b(s|r|p|a).+er\b',r'\bk.+n\b',r'\bn.*k\b']
    name = name.lower()
    for s in patterns:
        if re.search(s,name):
            return True
    return False

def is_english(name):

strings=['ll','oo','wood','ers','bb','ss','way','ee','dow','ach','gg','van','tt','pp','down','ross','ver',
'ker','ren','ran','owe','mc']
    patterns = [r'\b.+kin.*\b',r'\b.+spoo.*\b',r'\b.*ear.*\b',r'\b.+rough.*\b',r'\b.+(i|e|a)ll\b',r'\b.
+more.*\b',r'\b.+son.*\b',r'\b(c|s|r|m|f).+er.*\b',r'\b.+man(n)\b',r'\b.*eat.*\b']
    name = name.lower()
    for s in patterns:
        if re.search(s,name):
            return True
    return False

def is_french(name):

strings=['le','ss','and','aux','oli','vage','eux','tit','ieu','nna','bon','boi','dú','neau','eau','oy','ias','
quet','reau','hur','agne']
    patterns = [r'\b.+onn.*\b',r'\b.*eaux.*\b',r'\ble.+\b',r'\b.+quet.*\b',r'\b.+amps\b',r'\b(b|p|c|t).
+er\b',r'\b.+asso.+\b',r'\b.*agn.+\b']
    name = name.lower()
    for s in patterns:
        if re.search(s,name):
            return True
    return False

def is_german(name):
    strings=['tt','ber','oh','eier','ich','altz','ieck','enz','hol','uh','aun','del','ō','eis','mm','kl','aub']
    patterns = ['(r|t|g).+','(g|s|h|k|p|l).+l','st.+r','(b|f|h).+er','.*ott.*','.+eier','.+chard.*','.*omm.*','.+
(echt|ach).*',".+ö.*"]
    name = name.lower()
    for s in patterns:
        if re.search(s,name):
            return True
    return False

def is_greek(name):
    strings=['ous','kos','nos','mos','los','is','as']
    patterns = [r'\b.+(gas|ous|oulos)\b',r'\b.+os\b',r'\b.+is\b',r'\b.+os\b']
    name = name.lower()
    for s in patterns:
        if re.search(s,name):
            return True
    return False

def is_irish(name):
    strings=["o'",'ach',"ley","lly",'all','an','mac','oi','inn','ald']
    patterns = [r'\bo.+',r'\b.+ha(n|nn)\b',r'\b.+ll.*\b',r'\b.*a.{1,3}a(n|nn)\b',r'\b.*owe.*\b']
    name = name.lower()
    for s in patterns:
        if re.search(s,name):
            return True
    return False

def is_italian(name):

strings=["eri","ari","icci","one","chi","ggi","gio","à","tti","di","gari","fin","lli","to","oli","dia","zzi",
"Scor","ini","ecce","acco","ani"]
    patterns = [r'\b.+icci\b',r'\b.+ieri\b',r'\b.+achi\b',r'\b(a|p|u|z|n).*(ri|i)\b',r'\b.*oia\b']
    name = name.lower()
    for s in patterns:
        if re.search(s,name):
            return True
    return False
```

```
def is_japanese(name):

strings=['shi','oku','ida','iko','aka','tsu','awa','aya','ishi','uno','ima','chi','uya','hara','zak','aki','ji
','uch']
   patterns = [r'\b.*i(c|s).*da\b',r'\b.*yama.*\b',r"\b.*i(s|k)i\b",r"\bs.+m",r".+a(w|h)a\b"]
   name = name.lower()
   for s in patterns:
      if re.search(s,name):
         return True
   return False

def is_korean(name):
   strings=["mo","ch","oo","se","ha","san"]
   patterns = [r'\b(..|...)\b',r'\bch.{3,5}\b',r'\bh.{2,3}ng*\b','']
   name = name.lower()
   for s in patterns:
      if re.search(s,name):
         return True
   return False

def is_polish(name):
   strings=["ń","ka","ski","ó","no","cz","ko","zga"]
   patterns = [r'\bk.*ki\b',r'\bka.+a\b',r"\bg.*r.*k\b",r"\b.*ń","k.{0,1}ó\b"]
   name = name.lower()
   for s in patterns:
      if re.search(s,name):
         return True
   return False

def is_portuguese(name):
   strings=["inho","ō","lho","ã","ro","é"]
   patterns = [r"\b.+veira",r'\b(p|c|a).*o\b']
   name = name.lower()
   for s in patterns:
      if re.search(s,name):
         return True
   return False

def is_russian(name):
   strings=["ov","nov","ev","ivch","sky","tov","iev","ich","cev","in","ff","hin",""]
   patterns = [r".*nov\b","\b(v|j|z|d|p|c).{5,9}sky\b",""]
   name = name.lower()
   for s in patterns:
      if re.search(s,name):
         return True
   return False

def is_scottish(name):
   strings=["aw","son","las","ald","ht","hy","ant","at"]
   patterns = [r".+son\b",r".+ant\b",".+ight.*"]
   name = name.lower()
   for s in patterns:
      if re.search(s,name):
         return True
```

```
    return False

def is_spanish(name):
    strings=["ó","lix","oj","é","rez","lla","cal","que","res","aya","ey","ez",'á','í']
    patterns = ['.*cal',r'.+e(r|rr)a\b',r'\b san.*','.*(á|ó|í|é|ú).*',r"\b(a|h|o).+a",""]
    name = name.lower()
    for s in patterns:
        if re.search(s,name):
            return True
    return False

def is_vietnamese(name):
    strings=["im","hao","han","ly","ach","ch"]
    patterns = ['.*kim',r'\b.*(h|a).+h']
    name = name.lower()
    for s in patterns:
        if re.search(s,name):
            return True
    return False

def get_origins(names):
    origin = []
    for name in names:
        options = []
        if is_arabic(name): options.append('Arabic')
        if is_chinese(name): options.append('Chinese')
        if is_czech(name): options.append('Czech')
        if is_dutch(name): options.append('Dutch')
        if is_english(name): options.append('English')
        if is_french(name): options.append('French')
        if is_german(name): options.append('German')
        if is_greek(name): options.append('Greek')
        if is_irish(name): options.append('Irish')
        if is_italian(name): options.append("Italian")
        if is_japanese(name): options.append("Japanese")
        if is_korean(name): options.append('Korean')
        if is_polish(name): options.append('Polish')
        if is_portuguese(name): options.append('Portuguese')
        if is_russian(name): options.append('Russian')
        if is_scottish(name): options.append('Scottish')
        if is_spanish(name): options.append("Spanish")
        if is_vietnamese(name): options.append("Vietnamese")
        if len(options) > 0:
            i = random.randint(0, len(options)-1)
            origin.append([name,options[i]])
        else:
            o = 
['Korean',"Japanese","Italian",'Irish','Greek','German','French','Dutch','Chinese','Russian','Eng
lish','Arabic','Polish','Portuguese','Scottish',"Spanish","Vietnamese"]
            i = random.randint(0,len(o)-1)
            origin.append([name,options[i]])
    return origin

def process_file(src):
    lines = open(src,encoding='utf-8').readlines()
    languages = dict()
    for line in lines:
        if 'To The First' in line: continue
        line = line.strip("\n")
        line = line.strip('"')
        line = line.strip('"')
        s = line.split(",")
        if "Jevolojnov" in line:
            s.remove('"')
        if not s[1] in languages:
            languages[s[1]] = []
        languages[s[1]].append(s[0])
    return languages

def get_keys(d):
    n = []
    for k in d.keys():
        n.append(k)
    return n

def process_file_names(src):
    lines = open(src,encoding='utf-8').readlines()
    names = dict()
    for line in lines:
        if 'To The First' in line: continue
        line = line.strip("\n")
        line = line.strip('"')
        line = line.strip('"')
        s = line.split(",")
        if "Jevolojnov" in line:
            s.remove('"')
        if not s[0] in names:
            names[s[0]] = []
        if s[1] in names[s[0]]:
            continue
        else:
            names[s[0]].append(s[1])
    return names

def calculate_accuracy(names,origins):
    correct = 0
    for e in origins:
        if e[1] in names[e[0]]:
            correct+=1
    print('---------------',"Overall Accuracy",'---------------')
    print("Accuracy:",correct/len(names))
    print("--------------------------------------------------")
def write_predictions(src,true,predictions):
    f = open(src+"prediction.txt","w")
    f.write("Name"+"\t"+"Actual"+"\t"+"Predicted\n")
    for e in predictions:
        # row = e[0]+"\t"+true[e[0]]"\t"+e[1]+"\n"
        f.write(e[0]+"\t"+true[e[0]][0]+"\t"+e[1]+"\n")
    f.close()

def get_stats_origins(names,origins,languages):
    for lang in languages:
        matrix = get_stats(names, origins, lang)
        print('-----------------',lang,"stats",'-----------------')
        print("Precision:",matrix[0] / (matrix[0] + matrix[1]))
        print("Recall:",matrix[0] / (matrix[0] + matrix[2]))
        print("------------------------------------------------")
def get_stats(names,origins,origin):
    #Assuming that origin is positive
    stats = [0,0,0,0] #TP,FP,FN,TN
    for e in origins:
        if e[1] == origin:
            if origin in names[e[0]]: #
                stats[0] += 1
            else:
                stats[1] += 1
        else:
            if origin in names[e[0]]:
                stats[2] += 1
            else:
                stats[3] += 1
    return stats

if __name__ == "__main__":
    src = './'
    countries = process_file(src+'surnames-dev.csv')
    names = process_file_names(src+'surnames-dev.csv')
    n = get_keys(names)
    origins = get_origins(n)
    calculate_accuracy(names, origins)
    get_stats_origins(names,origins,['Japanese','Arabic','Chinese'])
    ############Musicians##################
    musicians = process_file_names(src+'composers.txt')
    n = get_keys(musicians)
    origins = get_origins(n)
    calculate_accuracy(musicians, origins)
    write_predictions(src, musicians, origins)
```

Driver:

```
import predictor
src = './'
file = 'surnames-dev.csv'
countries = predictor.process_file(src+file)
names = predictor.process_file_names(src+file)
n = predictor.get_keys(names)
origins = predictor.get_origins(n)
predictor.calculate_accuracy(names, origins)
predictor.get_stats_origins(names,origins,['Russian'])
```