DD1418 projektarbete - Language Detection

Svetlana Schånberg
Filip Elander

December 2019

1 Introduktion

Allaboard är en applikation som skall lanseras i mars 2020 [1]. Syftet med applikationen är att knyta samman de Europeiska tågbokningssystemen till en och samma plattform. Användarvänlighet ligger i fokus och det är viktigt att tjänsten fungerar oavsett användarens språk-preferens. Användaren skall kunna söka efter destinationer på valfritt språk och tjänsten skall kunna hantera diverse exonym som finns mellan de Europeiska språken. Samt skall recensioner till diverse destinationer prioriteras till användare med samma språk och lokal platsinformation kan översättas åt användaren.

Allaboard värderar språkigenkänning högt och anser att det är viktigt att ge användaren friheten att använda sitt egna språk. För tillfället arbetar applikationen på det språk som webbläsaren är inställd på men exempelvis så har många svenska arbetsplatser engelska som grundinställning i sina webbläsare, vilket kan göra sökning av lokala svenska orter problematiska för tjänsten. Till en början vill *Allaboard* hantera engelska, tyska och svenska men på sikt skall applikationen vara tillgängliga på flera språk.

2 Bakgrund

Generativa metoder uppbyggda på statistik är idag vanligt för klassificering av språk. Då utnyttjas ofta sannolikheter för ord och bokstavsföljder i de olika språken som kan modelleras med hjälp av N-gram. Det finns även diskriminerande språkmodeller för klassificeringen som tar hjälp av olika features eller egenheter hos språk. I denna rapport kommer både den generativa samt den diskriminerande metoden att användas och sedan utvärderas mot varandra.

3 Hypotes

Huvudproblemet att lösa var att så effektivt som möjligt klassificera de av användaren givna sökorden. Klassificering går att göra både generativt och diskriminerande. För att jämföra de olika metoderna användes det ett N-gram för den generativa klassificeringen och logisk regression för att ta fram funktionernas viktning till den diskriminerande klassificeringen.

3.1 Bigram

Teorin om bigramsklassificeringen byggde på att bokstavskombinationer uppkommer med olika frekvenser för olika språk[4]. Genom att observera vilka bokstavspar som det angivna sökordet var uppbyggt av så skulle man med hjälp av de olika språkens bigramsannolikheter kunna se vilket språk som var mest sannolikt. Det krävdes ett antal olika antaganden för att kunna använda sig utav bigramsannolikheter. Det första antagandet var att man endast behövde kolla på den föregående bokstaven och inte alla föregående bokstäver (Markovs antagande) när man gjorde sannolikhetsberäkningen med Naive Bayes. Görs inte det antagandet skulle det snabbt bli många, långa och unika bokstavskombinationer som skulle kräva enorma träningskorpus som i alla fall skulle riskera att snedvrida sannolikheten[2]. Det andra antagandet som gjordes var att man kunde applicera kedjeregeln på sannolikheterna för alla individuella bokstavsföljder för att beräkna den totala sannolikheten för hela sökordet[2].

Tabell 1: De tre vanligast bokstavsbigrammen i Svenska, Engelska och Franska

Svenska	Engelska	Franska
EN: 2.44%	TH: 2.71%	ES: 2.91%
DE: 2.11%	HE: 2.33%	LE: 2.08%
ER: 2.10%	IN: 2.03%	DE: 2.02%

3.2 Regression

Genom att observera förekomsten(logisk 1:a) eller avsaknaden(logisk 0:a) av olika väl valda funktioner x_i s.k. Features hos sökordet så kan man väga förekomsten med en negativ eller positiv konstant θ_i för att öka eller minska språkets sannolikhet. Genom att spara de logiska ettorna och nollorna i en vektor och deras tillhörande vikt i en annan vektor så ger skalärprodukten mellan de två vektorerna ett siffervärde, där högre värde betyder högre sannolikhet och lägre värde mindre sannolikhet.

$$y = x_1\theta_1 + x_2\theta_2 + x_3\theta_3 \tag{1}$$

Viktingskonstanterna tas fram med hjälp av logisk regression. skalärprodukten kan transformeras med en Sigmoid funktion vilket översätter värdet på skalärprodukten till en logisk etta eller nolla (sant eller falskt). Den transformerade modellens

validitet kan sedan utvärderas genom att analysera den mot ett förutbestämt träningsset. Då man vet de korrekta logiska utfallen från träningsdatan så kan man bygga en funktion som beräknar hur mycket fel modellen har. Med logisk regressions så tar man fram vikternas medelfel-funktion, sedan genom att derivera fel-funktionen kan man med hjälp av gradienten stega mot derivatans nollvärde dvs. där felen planar av och blir så få som möjligt.

4 Metod

4.1 Bigram

Först skapades det ett gemensamt alfabet för det Europeiska alfabetet tillsammans med det grekiska samt det kyrilliska alfabetet. Varje språk tränades sedan på de olika översättningarna av bibeln , där texten skalats från Siffror och skiljetecken. Apostrofer behölls eftersom de är viktiga i vissa språk. valet av bibeln som träningskorpus gjorde så att alla språk tränades på enhetligt text. De olika bokstavsbigrammen och dess sannolikhet sparades i språkklasser. sannolikheten för varje bokstavsbigram togs fram med Naive Bayes modellering. När sannolikhetsbigrammen för varje språk var framtaget så gick det att stega igenom sökordet eller söktexten som man önskade klassificera och beräkna sannolikheterna för varje steg. Sannolikheterna beräknades logaritmiskt och summerades för varje steg. När hela sökargumentet hade bearbetats så valdes det språk som hade högst sannolikhet.

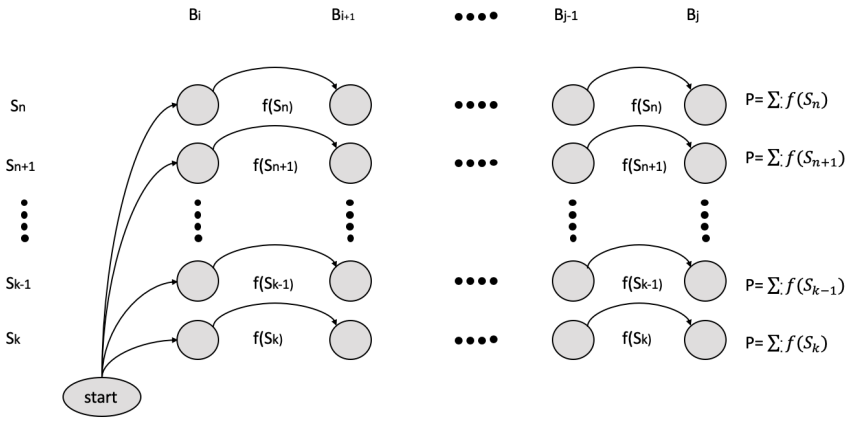

Figure 1: illustration av språkklassifikationen S_{n-k}bestäms av summeringen av sannolikhetsberäkningarna $f(S_{n-k})$ genom att stega genom bokstäverna B_{i-j}

4.2 Regression

För den diskriminerande metoden så togs det fram tre stycken features som skulle viktas med hjälp av logisk regression. Det togs även fram tre stycken features som skulle fungera som "bias-funktioner" som kunde utesluta alla andra klasser vid förekomst.

De klasser som skulle tränas var förekomsten av dubbelteckning f_1, förekomsten av andra tecken än a-z f_2 och förekomsten av apostrof f_3. Dessa valdes för att

de skulle kunna förekomma i flera språk och inte direkt kunde utesluta alla andra klasser, vilket gjorde deras viktning mer relevant för träning.

Vikterna för varje språkmodell fick ett träningsset på 1000 ord där 60 procent av utfallen var en logisk 1:a och 40 procent hade utfallen logisk 0:a. De "sanna" klassificeringarna var tagna från bibeln med rätt språk och de "falska" klassificeringarna var lika fördelade på de biblar med annat språk. Viktningskonstanterna tränades tills förändringen mellan varje iteration blev tillräckligt liten så att den kunde anses ha planat av.

De "bias-funktioner" som tagits fram var förekomsten av $å|ä|ö$ f_4, förekomsten av tyskt dubbel-s f_5 eller förekomsten av kyrilliska bokstäver f_6. Dessa funktioner är unika för ett enskilt språk i tjänsten och gavs manuellt en hög viktning för att med säkerhet ge störst sannolikhet vid förekomst.

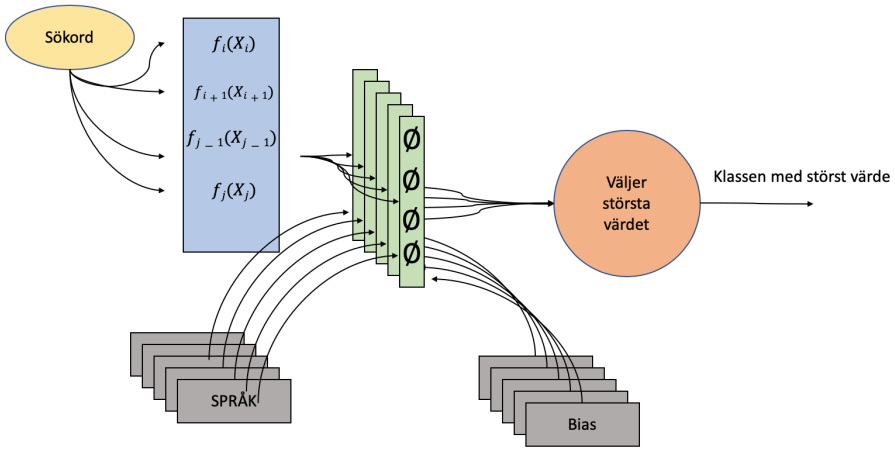

Figure 2: illustration av klassifikationen med vikter framtagna genom Logisk regression

5 Implementation

För att se hur programmet svarade på sökargument av endast ett sökord så testades klassificeringen på namnen av Europas huvudstäder. Sedan testades även längre sökargument bestående av tre slumpmässiga ordkombinationer på olika språk som skapades med hjälp av Random Word Generator [3]. Utvärderingset bestod av 50 argument vardera. Det skapades även en enklare interface för användare att skriva in ord eller meningar som sedan kunde väljas att klassificeras med genererande eller den diskriminerande modellen.

6 Resultat

6.1 Generativ

För endast ett sökord så blev medelresultatet ca 44 procent om man inte räknar med ryska. Resultatet för ryska var någorlunda irrelevant då det var det enda språket som använde kyrilliska bokstäver och kommer därmed inte vidare att analyseras. De enskilda resultaten blev någorlunda normalfördelad där spanska och tyska hade klart bäst säkerhet på ca 70 procent och sedan en fallande säkerhet ner till svenska med en säkerhet på ca 17 procent. När argumentet ökades till tre stycken sökord ökade säkerheten markant. Medelresultatet blev ca 78 procent och där spanska, svenska och tyska hade en säkerhet på över 80 procent.

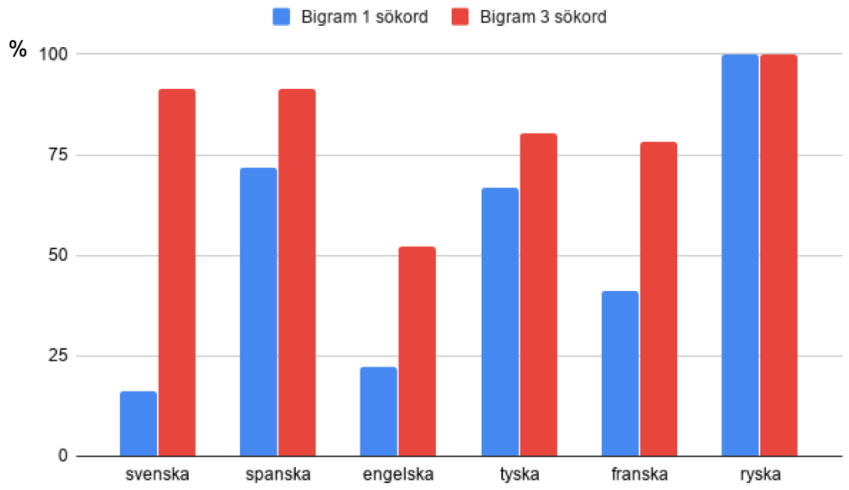

Figure 3: Diagram över den generativa klassificeringens säkerhet

6.2 Diskriminerande

De resulterande konstantvikterna visas i tabell[2]. Det blev tydligt att features som inte fanns med i vissa språk fick negativ vikt och en förekomst i sökordet drog ner sannolikheten att det skulle få den klassificeringen. De funktioner som blivit viktade iterativt med logisk regression sammanfördes med de egenvalda "bias-funktionerna" som vardera fick vikten 10. För varje skalärprodukt lades det även till en viktad "dummy"-term för att jämna ut summan och inte ge nollvärden ifall ingen funktion påträffades i sökargumenten.

Table 2: resulterande funktionssvikt

Språk	dubbelteckning	annat än a-z	apostrof
Svenska	0.6313	1.1377	-6.1142
Engelska	0.3824	-7.1369	0.6882
Franska	0.2885	0.1411	3.8094
Spanska	0.7824	0.5064	2.5941
Tyska	0.4194	0.6180	-3.9009

För endast ett sökord så gav den diskriminerande modellen endast resultat åt den tyska klassen och några få till svenska, annars misslyckades modellen helt att klassificera något annat språk rätt (förutom ryska). När man ökade till tre sökargument så gav den även några få korrekta klassificeringar åt franska men sjönk i säkerhet på tyskan och svenskan.

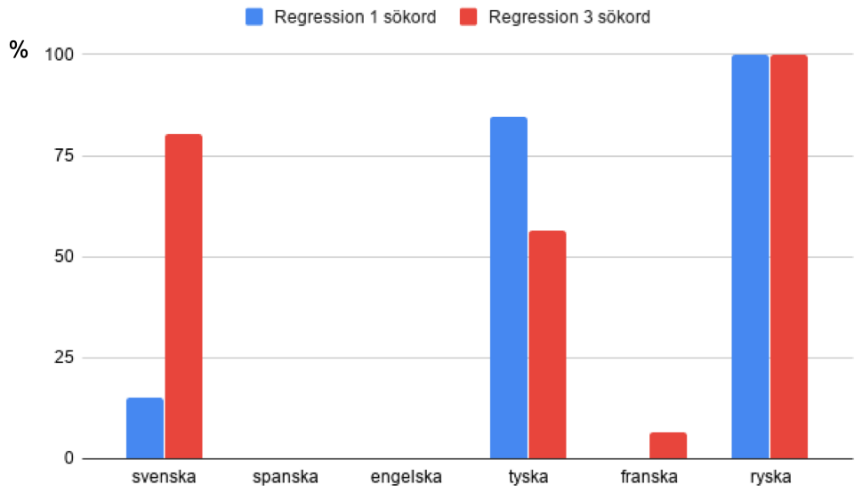

Figure 4: Diagram över den diskriminerande klassificeringens säkerhet

7 Slutsats

Försöker man att identifiera språk med hjälp av bigram så är det betydligt lättare med flera ord eller en mening som input. Att identifiera språk när man har ett ord är svårare, men fungerar bättre på vissa språk som spanska och tyska som eventuellt har mer unika bigram. Ryska har ett eget skriftspråk jämfört med de andra språken i våra modeller så resultatet blir 100 procent på grund av ett bias alfabet. Använder man regression så beror resultatet på hur bra funktioner som är valda. Man behöver kunskaper inom lingvistik för att välja bra funktioner. Har man det och om man har tillräckligt stora och bra träningsset så kan resultatet bli bra. Den metod är mycket flexibel då man man ändra flera parametrar i försöket att ta fram bättre resultat. Bigramsmetoden är däremot byggd på statistik, kräver mindre kunskap om olika språk och är mindre flexibel när det gäller ändring av parameter.

8 Diskussion

Språkklassificering av endast ett ord och speciellt namn på destinationer visade sig vara väldigt svårt och i många fall omöjligt då destinationer heter samma sak på flera olika språk. De fall då det har samma namn på flera språk går endast att kontrollera i test-set men blir avsevärt svårare att kontrollera för tjänsten när den används på marknaden. Ett sätt att hantera detta skulle kunna vara att mäta hur stor skillnad det är mellan de olika klassernas sannolikheter och endast klassificera ifall det är ett överväldigande värde för någon av klasserna, om sannolikheten är spridd över många klasser så skulle man kunna returner engelska tills fler argument har givits av användaren och en av klasserna är obestridligt mer sannolikt än de andra.

Den generativa funktionen fungerade relativt bra så fort som sökargumenten blev fler. För att öka precision mer skulle man kunna testa att utöka korpusen för varje språk, man skulle även kunna utöka bigrammen till längre N-gram vilket förmodligen skulle höja säkerheten då många språk har längre karaktäristiska bokstavskombinationer.

I fallet att man skulle vilja utöka tjänsten till fler språkklasser så finns det risk att precisionen skulle börja sjunka, då fler besläktade språk skulle introduceras och göra det svårare att skilja mellan dem. Introduceras fler språk så bör teoretiskt en fungerande diskriminerande metod ge bättre resultat då man kan ge fler funktioner som kan skilja dem åt.

Den Diskriminerande metoden i detta projekt fungerande inte alls, detta kan bero på en rad orsaker. Förmodligen så skulle man kunna välja mycket bättre funktioner att träna samt många fler, till exempel lägga till olika bigram som funktioner. Även fast man endast behöver träna funktionerna en gång så blir det väldigt hög komplexitet för programmen vid fler funktioner, då varje tillagd

dimension ökar komplexiteten exponentiellt, detta skulle dock kunna kringgås med en mindre effektiv regressionsteknik, till exempel en stokastisk.

En annan orsak till det dåliga resultatet för modellen kan vara dess träningsset som togs fram till den logiska regressionen. Längden för detta set bör utökas för att förhoppningsvis öka frekvensen av modellens funktioner. Argument i träningen som inte innehåller någon av de valda funktionerna är helt ointressanta och endast ökar komplexiteten för programmet.

References

[1] Allaboard *allaboard*.
Available: https://allaboard.eu/

[2] D. Jurafsky J.H. Martin, *Speech and Language Processing, 3rd ed. (draft)*.
Available: https://web.stanford.edu/ jurafsky/slp3/

[3] Random Word Generatorr.
Available: https://www.textfixer.com/tools/random-words.php (Hämtad 2019-12-09)

[4] Practical criptography .
Available: http://practicalcryptography.com//

Appendices

A Alfabets-genereraren

```
import re
# bildar ett alphabet fr n textfil med alla bokst ver i europa.
word_to_index = {}
index_to_word = {}
with open('bokstaver.txt','r') as f:
    counter = 0
    for i in f.readlines():
        stringen = re.sub(r'[0-9|.|,|:|(|)|\n|\s]','',i)
        s = list(stringen)
        for token in s:
            bokstav =token.lower()
            if word_to_index.get(bokstav) == None:
                word_to_index[bokstav] = counter
                index_to_word[counter] = bokstav
                counter += 1

# l gger till stora bokst ver
counter = len(word_to_index)
for i in range(len(word_to_index)):
    bokstav = index_to_word[i]
    stor_bokstav = bokstav.upper()
    word_to_index[stor_bokstav] = counter
    index_to_word[counter] = stor_bokstav
    counter +=1


with open('alphabet.txt','w') as wf:
    for ordet in word_to_index:
        index = word_to_index[ordet]
        index = str(index)
        line = "{} {}\n".format(index,ordet)
        wf.write(line)
    wf.write("328 '\n")
    wf.write("329 space")
```

B Alfabetsklassen

```
#klass f r alphabete
import sys
class alphabet(object):

    def __init__(self):
        self.alphabet_i2w = {}
        self.alphabet_w2i = {}

    def alphabetBuilder(self):    # bygger en dictionary i en klass
     med alphabetet
        with open('alphabet.txt','r') as v:
            for i in v.readlines():
                i = i.split()
```

```
                self.alphabet_i2w[i[0]] = i[1]
                self.alphabet_w2i[i[1]] = i[0]
```

C Bigrams-genereraren

```

# Kod f r att skriva en txt-fil med givet textsdokuments bigram sanolikheter
# det kr vs att man har klassen f r alphabetet!

# in arg -f "filnamnet.txt" som ineh ller tr ningscorpus"
# ut arg -d "filnamnet.txt " som skall ineh lla bigram sannolikheterna

from alphabet import alphabet
import argparse
import sys
import re
from collections import defaultdict
import math
parser = argparse.ArgumentParser(description='UnigramBuilder')
parser.add_argument('--file', '-f', type=str,  required=True, help='textfil')
parser.add_argument('--destination', '-d', type=str, help='sparad textfil')
argument = parser.parse_args()


#----------------- Bygger unigram ocg bigra COUNTER ---------------------------
alphabet = alphabet()
alphabet.alphabetBuilder()
index_2_bokstav = alphabet.alphabet_i2w
bokstav_2_index = alphabet.alphabet_w2i
unigram_count = {}
bigram_count = defaultdict(lambda: defaultdict(int))




for i in range(len(bokstav_2_index)+1):
    unigram_count[i] = 0

with open(argument.file, 'r') as t:
    text =t.read()
    text= re.sub(r'[\n]','',text)
    text = list(text)
    prev_alphabet_index = int(bokstav_2_index['space'])
    for bokstav in text:
        if bokstav == ' ':
            alphabet_index = int(bokstav_2_index['space'])
        else:
            alphabet_index = int(bokstav_2_index[bokstav])
        unigram_count[alphabet_index] += 1

        try:
            bigram_count[prev_alphabet_index][alphabet_index] += 1
```

```
        except KeyError:
            bigram_count[prev_alphabet_index][alphabet_index] = 1
        prev_alphabet_index = alphabet_index


# ---------------------- Skriver en fil med BigramProbs
    ---------------------------------------

with open(argument.destination,'w') as w:
    for first_ord in bigram_count:
        for secound_ord in bigram_count[first_ord]:
            p =format(math.log(bigram_count[first_ord][secound_ord]
     / unigram_count[first_ord]),'.15f')
            line = "{} {} {}\n".format(first_ord,secound_ord,p)
            w.write(line)
```

D Bigramsklassen

```
import sys
from collections import defaultdict

class Bigram(object):
    def __init__(self,model):
        self.Bigram = defaultdict(lambda: defaultdict())
        self.LanguageModel = model

    def BigramBuilder(self):    # bygger en klass f r bigramet
        with open(self.LanguageModel,'r') as t:
            for i in t.readlines():
                i = i.split()
                self.Bigram[i[0]][i[1]]= i[2]
```

E Bigrams modellen

```
# G r Bin rklassificieringen enligt bigram f r interface
    programmmet
# language labeler.

from alphabet import alphabet
from Bigram import Bigram
import argparse
import sys
import re
from collections import defaultdict
import math
import numpy as np

class Detection(object):

    def __init__(self,I2W,W2I):
        self.Lexikon = defaultdict(lambda:defaultdict(lambda:
    defaultdict()))
        self.antal_alphabet = 0
        self.sannolikhheter = []
        self.ind_2_word = I2W
```

```
        self.word_2_ind = W2I
        self.LanguageName = {}
        self.vikt = math.log(0.001)



    def Alfabeten(self,sprak_modellen,namn): # bygger en 4D dic med
     alphabeten
        self.Lexikon[self.antal_alphabet] = sprak_modellen
        self.LanguageName[self.antal_alphabet] = namn
        self.antal_alphabet +=1



    def Process(self,letters):
        letters = list(letters)
        # initierar den f rsta sannolikhter (alltid start till
    f rsta bokstaven)
        for i in range(self.antal_alphabet):
            try:
                P = float(self.Lexikon[i][self.word_2_ind['space'
    ]][self.word_2_ind[letters[0]]])
                self.sannolikhheter.append(P)
            except:
                P = self.vikt
                self.sannolikhheter.append(P)


        # forts tter p  bokstav 2 i in-argumentet
        prew = letters[0]
        for token_index in range(1,len(letters)):
            bokstav = letters[token_index]
            for sprak_index in range(len(self.Lexikon)): #g  r
    igenom varje spr k
                try:
                    P = float(self.Lexikon[sprak_index][self.
    word_2_ind[prew]][self.word_2_ind[bokstav]])
                    self.sannolikhheter[sprak_index] =  self.
    sannolikhheter[sprak_index] + P

                except:

                    P = self.vikt
                    self.sannolikhheter[sprak_index] =  self.
    sannolikhheter[sprak_index] + P


            prew = bokstav
        res = []
        for row in self.sannolikhheter:
            res.append(math.exp(row))

        #print(res)
        res_language = res.index(max(res))
        res_sannolikhet = res[res_language]
        print("Texten  r  med st rst sannolikhet skriven p  {}".
    format(self.LanguageName[res_language]))
```

```
69          return self.LanguageName[res_language]
70
71
72
73  def main(In_arg):
74
75      alfabetet = alphabet()
76      alfabetet.alphabetBuilder()
77
78      detection = Detection(alfabetet.alphabet_i2w,alfabetet.
        alphabet_w2i)
79
80
81      #------- Svenska ------------#
82      Svenska = Bigram('bigram/swedish_bigrams.txt')
83      Svenska.BigramBuilder()
84      SWE = Svenska.Bigram
85      detection.Alfabeten(SWE,'Svenska')
86      #------------------------#
87      #------- Engelska ------------#
88      Engelska = Bigram('bigram/english_bigrams.txt')
89      Engelska.BigramBuilder()
90      ENG = Engelska.Bigram
91      detection.Alfabeten(ENG,'Engelska')
92      #------------------------#
93      #------- Spanska ------------#
94      Spanska = Bigram('bigram/spanish_bigrams.txt')
95      Spanska.BigramBuilder()
96      SPE = Spanska.Bigram
97      detection.Alfabeten(SPE,'Spanska')
98      #------------------------#
99      #------- Tyska ------------#
100     Tyska = Bigram('bigram/german_bigrams.txt')
101     Tyska.BigramBuilder()
102     GER = Tyska.Bigram
103     detection.Alfabeten(GER,'Tyska')
104     #------------------------#
105     #------- Franska ------------#
106     Franska = Bigram('bigram/french_bigrams.txt')
107     Franska.BigramBuilder()
108     FRA = Franska.Bigram
109     detection.Alfabeten(FRA,'Franska')
110     #------------------------#
111     #------- Ryska ------------#
112     Ryska = Bigram('bigram/russian_bigrams.txt')
113     Ryska.BigramBuilder()
114     RUS = Ryska.Bigram
115     detection.Alfabeten(RUS,'Ryska')
116     #------------------------#
117
118
119     return detection.Process(In_arg)
120
121 if __name__ == "__main__":
122     main()
```

F Träningsset för Regression

```
# Genererar tr ningsset  f  r  logisk regression

import random
import nltk
import re
res = []
# byt till clean texter!
with open('clean/swedish_clean_2.txt','r') as c ,open('clean/
    english_clean.txt','r') as f1,open('clean/spanish_clean.txt','r
    ') as f2,open('clean/french_clean.txt','r') as f3,open('clean/
    german_clean.txt','r') as f4:
    correct_text = c.read()
    correct_text= nltk.word_tokenize(correct_text)
    false_text1 = f1.read()
    false_text1= nltk.word_tokenize(false_text1)
    false_text2 = f2.read()
    false_text2= nltk.word_tokenize(false_text2)
    false_text3 = f3.read()
    false_text3= nltk.word_tokenize(false_text3)
    false_text4 = f4.read()
    false_text4= nltk.word_tokenize(false_text4)

    cc = 1
    ff1 = ff2 = ff3 = ff4 = 1

for i in range(10000): ## kolla l ngden
        holder = []
        pekare = random.choices([0,1],[0.3,0.7])


        if pekare[0] == 1:
            holder.append(correct_text[cc])
            holder.append(pekare)
            cc = cc +1
        elif pekare[0] == 0:
            pek = random.choices([1,2,3,4],[0.25,0.25,0.25,0.25])
            if pek[0] ==1:
                holder.append(false_text1[cc])
                holder.append(pekare)
                cc = cc +1

            if pek[0] ==2:
                holder.append(false_text2[cc])
                holder.append(pekare)
                cc = cc +1

            if pek[0] ==3:
                holder.append(false_text3[cc])
                holder.append(pekare)
                cc = cc +1

            if pek[0] ==4:
                holder.append(false_text4[cc])
                holder.append(pekare)
                cc = cc +1

```

```
54
55
56          res.append(holder)
57  with open('train_swedish.txt','w') as w:
58      for rows in range(len(res)):
59          rad = res[rows]
60          s = str(rad[0]) + str(' ') + str(rad[1][0])+str('\n')
61          w.write(s)
```

G Regression

```
1   # kod f r regression p  svenska
2   from sklearn.model_selection import train_test_split
3   import nltk
4   import re
5   import random
6   import math
7   import numpy as np
8
9   #
        ------------------------------------------------------------------------------
10  class BinaryLogisticRegression(object):
11
12      def __init__(self, x=None, y=None):
13          self.number_of_func = 4
14          self.theta = np.random.uniform(-1, 1, self.number_of_func)
15          x_tr, y_tr, x_val, y_val = self.train_val_split(np.array(x)
        , np.array(y))
16          self.xtrain = x_tr
17          self.ytrain = y_tr
18          self.xval = x_val
19          self.yval= y_val
20          self.alpha = 0.6
21          self.gradient = np.zeros(self.number_of_func)
22  #
        ------------------------------------------------------------------------------
23      def train_val_split(self, x, y):
24          xtr,xval,ytr,yval = train_test_split(x,y,test_size=0.1)
25          return xtr, ytr, xval, yval
26  #
        ------------------------------------------------------------------------------
27      def sigmoid(self, z):
28          return 1.0 / ( 1 + math.exp(-z) )
29
30
31      def X(self,func,text):
32          #dummy
33          if func == 0:
34              x = 1
35          #f1
36          if func == 1:
37              match = re.search(r'(\w)\1+',text)
38              if match != None:
39                  x = 1
```

```
            else:
                x = 0
        #f2
        if func == 2:
            match = re.search(r'[^a-zA-Z]',text)
            if match != None:
                x = 1
            else:
                x = 0
        # f3
        if func == 3: #kryliska
            match = re.search(r"'",text)
            if match != None:
                x = 1
            else:
                x = 0
        return x

#-------------------------------------------------------------------------------

    def fit(self):
        i = 0
        for i in range(100000):
            i +=1
            sum = 0
            for k in range(self.number_of_func):
                x_sigH_Y_sum = 0
                for ord in range(len(self.xtrain)):
                    thetaTX = self.theta[0]*self.X(0,self.xtrain[
ord]) + self.theta[1]*self.X(1,self.xtrain[ord]) + self.theta
[2]*self.X(2,self.xtrain[ord])+self.theta[3]*self.X(3,self.
xtrain[ord])
                    x_sigH_Y_it = self.X(k,self.xtrain[ord])*(self.
sigmoid(thetaTX) - float(self.ytrain[ord]))
                    x_sigH_Y_sum = x_sigH_Y_sum + x_sigH_Y_it
                self.gradient[k] = x_sigH_Y_sum/len(self.xtrain)

            for k in range(self.number_of_func):
                self.theta[k] = self.theta[k] - self.alpha * self.
gradient[k]
            #loss = self.loss()

            if i%100 == 0:
                print(self.theta)
                #print(loss)



#-------------------------------------------------------------------------------


    def loss(self):
        x = self.xtrain
        y = self.ytrain
        sum = 0
        for m in range(len(self.xtrain)):
            thetaTX = self.theta[0]*self.X(0,self.xtrain[m]) + self
```

```
    .theta[1]*self.X(1,self.xtrain[m]) + self.theta[2]*self.X(2,
    self.xtrain[m])+self.theta[3]*self.X(3,self.xtrain[m])
90              loss_per_it = -int(y[m])*np.log(self.sigmoid(thetaTX))
    -(1-int(y[m]))*np.log(1-self.sigmoid(thetaTX))
91              sum = sum + loss_per_it
92              res = sum/len(self.xtrain)
93          return res
94 #-------------------------------------------------------------------------


95
96
97 #-------------------------------------------------------------------------

98 def main():
99     x = []
100    y = []
101    with open('train/train_spanish.txt','r') as f:
102        for i in f.readlines():
103            rad = nltk.word_tokenize(i)
104            x.append(rad[0])
105            y.append(rad[1])
106
107    b = BinaryLogisticRegression(x, y)
108    b.fit()
109
110
111 if __name__ == '__main__':
112    main()
```

H Regressions modellen

```
1 # G r Bin rklassificieringen enligt reggresed modell f r
    interface-programmmet
2 # language labeler.
3
4 from alphabet import alphabet
5 from Bigram import Bigram
6 import argparse
7 import sys
8 import re
9 from collections import defaultdict
10 import math
11 import numpy as np
12 import operator
13
14 def X(func,text):
15     #dummy
16     if func == 0:
17         x = 1
18     #f1
19     if func == 1:
20         match = re.search(r'(\w)\1+',text)
21         if match != None:
22             x = 1
23         else:
24             x = 0
25     #f2
```

```
    if func == 2:
        match = re.search(r'[^a-zA-Z]',text)
        if match != None:
            x = 1
        else:
            x = 0
    # f3
    if func == 3:
        match = re.search(r"'",text)
        if match != None:
            x = 1
        else:
            x = 0
    #f5
    if func == 4:
        match = re.search(r"[  |  |  ]",text)
        if match != None:
            x = 1
        else:
            x = 0
    #f6
    if func == 5:
        match = re.search(r"[\u0400-\u04FF]",text)
        if match != None:
            x = 1
        else:
            x = 0
    #f6
    if func == 6:
        match = re.search(r"[  |  ]",text)
        if match != None:
            x = 1
        else:
            x = 0
    return x

def main(In_arg):
    modeller = {}
    modeller['Svenska'] = [ 0.14601027, 0.93009565,  0.82639244,
    -0.03092643, 1000, 0, 0]
    modeller['Engelska'] = [ 0.92969424 ,-0.58307614, -4.59818166,
     0.19541981, 0,0,0]
    modeller['Tyska'] = [1.07453223,  0.33104029, -1.05536525 ,
    0.26345996,0,0,1000]
    modeller['Spanska'] = [ 0.45493518, -0.83723368,  0.722144,
    -0.8934274,0,0,0 ]
    modeller['Franska'] = [ 0.79979773, -0.48735955,  0.32604579,
    0.22527257,0,0,0]
    modeller['Ryska'] = [ 0.01, 0.1,  0.01, 0.01,0,1000,0]


    res = []
    for i in range(7): # antal funktioner
        holder = X(i,In_arg)
        res.append(holder)
    sannolikheter = {}
    for language in ["Svenska","Engelska","Tyska","Spanska","
```

```
    Franska","Ryska"]:
78          P = modeller[language][0]*res[0] + modeller[language][1]*
    res[1] + modeller[language][2]*res[2] +modeller[language][3]*
    res[3]+modeller[language][4]*res[4]+modeller[language][5]*res
    [5]+modeller[language][6]*res[6]
79          sannolikheter[language] = P
80
81      Keymax = max(sannolikheter, key=sannolikheter.get)
82      return Keymax
83
84  if __name__ == "__main__":
85      main()
```

I Interface

```
1   import sys
2   from tkinter import *
3   import LanguageLablerBigram
4   import LanguageLablerReg
5   # ansn tter roten
6   root = Tk()
7   root.title("Language Detection")
8   root.geometry("500x300")
9
10  #entering data
11  e = Entry(root,width=50)
12  e.grid(row=0,column=0,columnspan=2)
13
14  #funktion f r Bigram
15  def bigram():
16      mylabel2 = Label(root,text="mest sannolikt" + " " +
    LanguageLablerBigram.main(e.get()))
17      mylabel2.grid(row=2,column=0,columnspan=2)
18  #funktion f r regression
19  def regression():
20      mylabel2 = Label(root,text="mest sannolikt" + " " +
    LanguageLablerReg.main(e.get()))
21      mylabel2.grid(row=2,column=0,columnspan=2)
22  #knapp 1
23  mybutton = Button(root,text="Bigram",command = bigram,padx = 50,
    pady =25)
24  mybutton.grid(row=1,column=0)
25  #knapp2
26  mybutton2 = Button(root,text="Regressed",command=regression,padx
    =50,pady=25)
27  mybutton2.grid(row=1,column=1)
28
29  #loopen
30  root.mainloop()
```