UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences
Département d'Informatique

Natural Language Processing - Assignment 1

Chatri Alaaedine

1 Introduction

In this project, we will use language modeling to detect in which language variety a document is written. We will build trigram letter language models for three varieties, generate random output, score test sentences with each LM, and determine the language variety based on perplexity.
We will write a program in Python (3.x) that does the following :

— builds the Language model : reads in a text file, collects counts for all letter 3-grams, estimates probabilities, and writes out the model into a file
— generates random output strings of a given length k ($3 < k < 300$), according to its probabilistic model
— reads in a test document, applies the language model to it, and outputs its perplexity

We will make the following simplifications on documents (training) :

1. lowercase all characters
2. remove all characters that are not ASCII alphabetic (i.e. apostrophes digits, commas, periods, accented characters etc.)
3. replace single whitespaces between words by a double underscore.

More details in the next sections.

2 Data

2.1 Data collection

First of all, we had to retrieve the 4 files :

1. training.gb – British English training data
2. training.us - American English training data
3. training.au – Australian English training data
4. test – labelled test sentences

I do it in the constructor of the class (see section 6) :

```
def __init__(self, fileN=None)
```

2.2 Data modification

The data had to be modified as described in the section 1.
To do this, I have a function 'parseFile' (see section 6) :

```
def parseFile(self, dataInFile)
```

3 Methods

3.1 Probability

Once the input corpus has been decoded and transformed without spaces and no ascii symbols, the first three symbols are grouped together in order to form the first trigram. While processing, trigrams will be formed by discarding the left-most symbol of the previous trigram, shifting it one position to the left, and adding the new symbol to it. Exploring "natural" will therefore produce trigrams nat, atu, tur, ura and ral. When a trigram is produced, its count is incremented. This allows to count how many times each trigram appears. And we do the same for bigram.
After counting the number of trigrams and bigrams, the following formula is used to find the probability of each trigram :

$$P(w_i \mid w_{i-2}, w_{i-1}) = \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-1}, w_i)} \tag{1}$$

3.2 Generating random text

Generating random text starts by looking for the most probable bigram of the model. This is done by computing the probability that any bigram appears in the document using the following formula, then the bigram which maximizes it is emitted.

$$P(w_1, w_2, ..., w_n) = \prod_i P(w_i \mid w_1, w_2, ..., w_{i-1})(\text{chain rule}) \tag{2}$$

Once the first bigram has been emitted, we'll take the third symbol to form the trigram from the probability table. After, to have the next trigram, we take the last two symbols of this generated one and the two are used in order to construct a bigram $(w_{i-2}; w_{i-1})$, then w_i is chosen such that :

$$w_i = s \text{ with probability } P(s \mid w_{i-2}, w_{i-1}) \tag{3}$$

This symbol is appended to the generated text, then the loop starts over.

3.3 Perplexity

The last part of the project consists of computing the perplexity of an input text when matched against the model.

4 Results

The results presented here are computed from training files and a testing file (see 2.1). When the code is executed the probability files for all trigrams and for "iza, izo, ..." are created. The perplexity and random text is displayed on the console.

4.1 Excerpt of the language model

An excerpt of the language model for British English and another excerpt for American English model, displaying all n-grams and their probability with the two-letter history i z (E.g. iza, ize, izo etc.).

4.1.1 American English

iz.	
a	0.22262773722627738
b	0.0005213764337851929
e	0.6115745568300313
h	0.0005213764337851929
i	0.07559958289885298
m	0.0015641293013555788
n	0.0005213764337851929
o	0.032846715328467155
t	0.0005213764337851929
u	0.013034410844629822
z	0.02346193952033368
_	0.017205422314911366

TABLE 1 – iz-Table

4.1.2 British English

iz.	
a	0.19662921348314608
d	0.0022471910112359553
e	0.6112359550561798
g	0.0011235955056179776
h	0.0022471910112359553
i	0.05056179775280899
o	0.04157303370786517
s	0.0011235955056179776
u	0.015730337078651686
y	0.0011235955056179776
z	0.0449438202247191
_	0.03146067415730337

TABLE 2 – iz-Table

4.2 Random Text

4.2.1 American English

giblession and ine lotbale inateet saimuliuses laroter m of andat witalifyinese the throd oup ons youricaudirl itingreenisinds fre aelvir the dit ner i al itese buy nan st hand wast arthrounny ore ories say to on a forchavithe and thersh me for seen

4.2.2 British English

gran all sang bef swooldraftery ad trat the orm ords to ink wille ne sace iturnereatu-racto nothat we in tood got bers drand warc tions size able and mencomith pod it and to ho i of abes layabonecie youtilition ot re bout sect the wo the prory ton

4.2.3 Australian English

i the us derapplawas mand stas shour lial fill las cone kiny ok onagovegrosed mal thes werand co wiscre of usisimpands bach eve the a ofir nards faimprocklystrain sted whe vables suesologesourece of gons thisso efarelon abon anks squill nal ativel load

4.3 Perplexity

Here is a table that lists the perplexities obtained when matching the 9 test sentences against the three language models. The lowest perplexities are highlighted in bold :

	AU	GB	US
Sentence 1 : AU	**6.098110847134138**	6.33513823096318	6.283886294268185
Sentence 2 : AU	**6.29221470955582**	6.544314424697557	6.514849108417449
Sentence 3 : AU	**7.097974953122491**	7.383432183649177	7.186229813025471
Sentence 4 : GB	5.562703269599351	**5.404311366720503**	5.953777264056316
Sentence 5 : GB	6.2026794219836106	6.050527602402864	**6.020915791237737**
Sentence 6 : GB	5.802231563807819	**5.776970442911992**	5.817718139834288
Sentence 7 : US	5.637165848103404	5.357513573869093	**5.300900254863111**
Sentence 8 : US	6.011998311003168	6.017574573000815	**5.99353712162236**
Sentence 9 : US	5.404945679716553	**5.166160375433095**	5.19171760136564

TABLE 3 – Perplexity table

We can observe that the minimum perplexity is achieved when a test sentence is matched against the model of the language it is written into(green). This means that finding in which language a sentence has been written is possible by matching this sentence against the three models, and then selecting the language of the model that had the least perplexity. But two sentences are in red and this means that the sentences do not correspond to the model language.

4.3.1 Precision across all test sentences

As we can see in the table 3, 7 sentences were correctly predicted (in green) on 9 test sentences.

5 Discussion

Do you need to run the test set on all three language models or is the score from a single variety model sufficient ?
Yes we need to run the test set on all three language models to compare the perplexity. We used language models to compare the variety of language in which a text is written. To do it we calculated a lot of probabilities to build the model for a specific variety. Then, we apply the model to test sentences for computing their perplexity (see table 3).

Would a unigram or bigram language model work as well ? Explain why (not).
A unigram or bigram model will decrease the number of letters that we consider. So, the model will try to generate words that are not necessarily understandable because the "scope" of the model is too small. We will have less information to be able to create sentences that are close to the basic language.

Do the language models show anything about similarity of the varieties ? Why (not) ?
If you look at the section 4.1, the probabilities for US and GB are really close :

	US	GB
iza	0.22262773722627738	0.19662921348314608
ize	0.6115745568300313	0.6112359550561798
izu	0.013034410844629822	0.015730337078651686

TABLE 4 – iz-Proba-Example

We can look the table of perplexity (see table 3) and we see that perplexities for the three languages are close too. This shows us that there are similarities between the languages.

Why replacing spaces between word by double underscore is useful ?
To not have trigrams with letters of 2 different words.
Can you think of a better way to make a language variety guesser ?
We can use Deep Neural Networks and for example :

A Neural Probabilistic Language Model[1]

1. http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

6 Appendix

```python
import re
import json
from io import StringIO
from numpy import random
from collections import OrderedDict


class LanguageModel:
    def __init__(self, fileN=None):
        if fileN:
            #Open file in param (fileN)
            #Read file and put data into dataInFile
            #Call the function parseFile
            with open(fileN, encoding='utf8') as fileOpened:
                dataInFile = fileOpened.read()
            dataParsed = self.parseFile(dataInFile)

            #Call function to calculate probability
            self.probaTable = self.probaDataParsed(dataParsed)
        else:
            self.probaTable = {}

    def parseFile(self, dataInFile):

        #Replace all characters with lowercase letters
        dataInFile = dataInFile.lower()
        #Regex to keep only 26 alphabetic letters
        dataInFile = re.sub('[^a-z ]', "", dataInFile)
        #Replace spaces and no ascii by '__'
        dataInFile = dataInFile.replace(" ", "__")
        dataInFile = dataInFile.replace('$', "__")
        dataInFile = dataInFile.replace('^', "__")
        dataInFile = dataInFile.replace('__+', "__")
        #Return the modified data
        return dataInFile

    def probaDataParsed(self, dataInFile):

        probaTable = {}
        dicoBigram = {}
        dicoTrigram = {}

        for index in range(len(dataInFile) - 3 + 1):
            #First trigram with first 3 symbols
            trigram = dataInFile[index:index + 3]
            #First bigram with first 2 symbols
            bigram = dataInFile[index:index + 2]
            #If trigram don't exist in my trigram's dictonnary
            #the trigram takes 1
            #Else I'll do +1
            if trigram not in dicoTrigram:
                dicoTrigram[trigram] = 1
            else:
                dicoTrigram[trigram] += 1
            if bigram not in dicoBigram:
                dicoBigram[bigram] = 1
            else:
                dicoBigram[bigram] += 1
        #Calculate the proba with the formula P(w-2, w-1, w) / P(w-1,w)
        for bigram in dicoBigram:
            dico = {}
            for trigram in dicoTrigram:
                if trigram[0] + trigram[1] == bigram:
                    dico[trigram[2]] = dicoTrigram[trigram] / float(dicoBigram[bigram])
            probaTable[bigram] = dico
        return probaTable

    #This function writes probability table into a file
    #It writes in another file the probability for trigrams iza,ize...
    def probaInFileView(self, newFile, izFile):
        proTableOrdered = OrderedDict(sorted(self.probaTable.items()))
        out = open(newFile, 'w+')
        outIz = open(izFile, 'w+')
        for item in proTableOrdered:
            out.write("%s %s  \n" % (item, proTableOrdered[item]))
            if item[0] == 'i' and item[1] == 'z':
                outIz.write("%s %s  \n" % (item, proTableOrdered[item]))

    #This function writes probability table into a file using json
    def probaInFileJson(self, fileIn):
        s = OrderedDict(sorted(self.probaTable.items()))
        with open(fileIn, 'w') as newFile:
            json.dump(s, newFile)

    #This function reads probability table from a file using json
    def probaFromFileJson(self, fileIn):
        with open(fileIn, 'r') as fileOpened:
            self.probaTable = json.load(fileOpened)


    #This function takes bigrams in params and return a random char
    #using probabilites
    def randomInProbaTable(self, bigram):
        var = 0
        inBigramProba = self.probaTable[bigram]
        randomly = random.uniform(0, 1)
        for char, proba in inBigramProba.items():
            if var + proba >= randomly:
                return char
            var += proba
        assert False, "Error here !"

    #This function generates a random text
    #using the function randomInProbaTable to take a random letter
    def generateRandomData(self, rank):
        data = '__'
        for i in range(2, rank):
            bigram = data[i - 2] + data[i - 1]
            randomOne = self.randomInProbaTable(bigram)
            data += randomOne
        data = data.replace('__', ' ')
        data = data.replace('^__', '',)
        data = data.replace('_+$', '')
        data = data.replace('_','')
        print(data)

    #This function computes the perplexity
    def perplexityFromData(self, data):
        buffer = StringIO(data)
        perplexiteTable = []
        for oneLine in buffer:
            oneLine = oneLine[3:]
            oneLine = self.parseFile(oneLine)
            thePerplexity = 1
            lineSize = len(oneLine)
            for index in range(2, lineSize):
                bigram = oneLine[index - 2] + oneLine[index - 1]
                thePerplexity *= pow((1 / self.probaTable[bigram][oneLine[index]]),
                                     1 / float(lineSize - 2))
            perplexiteTable.append(thePerplexity)

        tableSize = len(perplexiteTable)
        for index in range(tableSize):
            print('Sentence '+ str(index + 1) + ' : ' + str(perplexiteTable[index]))

if __name__ == "__main__":
    fileTest = "test"
    with open(fileTest, encoding='utf8') as fileOpened:
        dataInFileTest = fileOpened.read()
    print("------------------------------------------------")
    print("TRAINING.US")
    print("------------------------------------------------")
    filen = "training.US"
    #-----CREATE LM-----
    LM = LanguageModel(filen)
    LM.probaInFileView("probaFile-"+filen, "izProbaFile"+filen)
    #-----RANDOM TEXT-----
    LM.generateRandomData(200)
    #-----PERPLEX-----
    LM.probaInFileJson("probaJson"+filen)
    LM.perplexityFromData(dataInFileTest)

    print("------------------------------------------------")
    print("TRAINING.AU")
    print("------------------------------------------------")
    filen = "training.AU"
    #-----CREATE LM-----
    LM = LanguageModel(filen)
    LM.probaInFileView("probaFile-"+filen, "izProbaFile"+filen)
    #-----RANDOM TEXT-----
    LM.generateRandomData(200)
    #-----PERPLEX------
    LM.probaInFileJson("probaJson"+filen)
    LM.perplexityFromData(dataInFileTest)

    print("------------------------------------------------")
    print("TRAINING.GB")
    print("------------------------------------------------")
    filen = "training.GB"
    #-----CREATE LM-----
    LM = LanguageModel(filen)
    LM.probaInFileView("probaFile-"+filen, "izProbaFile"+filen)
    #-----RANDOM TEXT-----
    LM.generateRandomData(200)
    #-----PERPLEX------
    LM.probaInFileJson("probaJson"+filen)
    LM.perplexityFromData(dataInFileTest)
```