KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Intelektikos Pagrindai (P176B101)

Pirmojo laboratorinio darbo ataskaita

Atliko:

IFF – 6/8 gr. studentas

Tadas Laurinaitis

2019 m. balandžio 29 d.

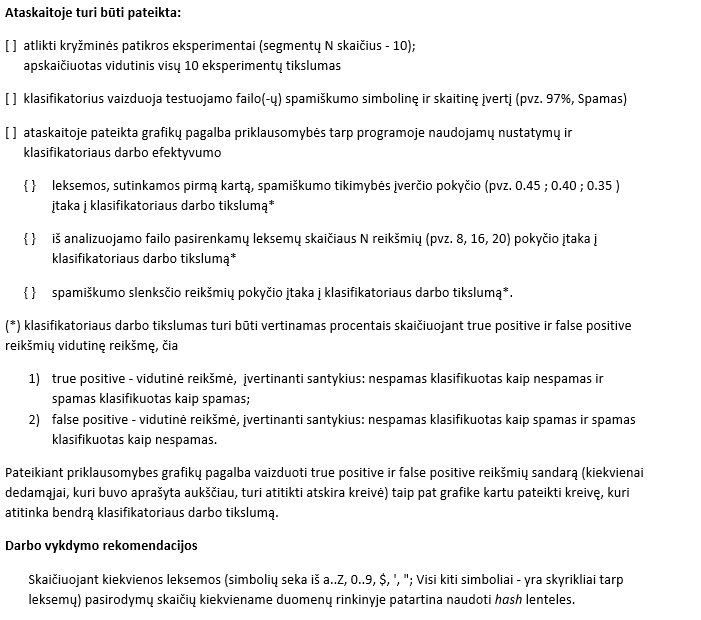
Priėmė:

Lekt. Germanas Budnikas

KAUNAS 2019

# Darbo užduotis

Sukurti programą SPAMui klasifikuoti panaudojant Bajeso teoremą. Ištirti priklausomybę tarp programoje naudojamų nustatymų ir klasifikatoriaus darbo efektyvumo (*žr.reikalavimus ataskaitai*). Programavimo kalba pasirenkama laisvai.

Ataskaitos reikalavimai:

# Darbo eiga ir tyrimai

Darbas buvo atliktas Python programavimo kalba.

Programos kodas:

import os

import re

class Lexeme:

def \_\_init\_\_(self, word, spamCount, hamCount, pWS, pWH, pSW):

self.word = word

self.spamCount = spamCount

self.hamCount = hamCount

self.pWS = pWS

self.pWH = pWH

self.pSW = pSW

#Note to self: no need for getters and setters

#function responsible for reading a number of files, defined by the fileLimit

#and getting lexemes from those files and then putting them into a lexicon/dictionary

#funcion returns a dictionary and two integer values - spam and ham word counts

#for later calculations

def readFilesIntoDictionary(spamDirectory, hamDirectory, fileLimit):

#initial values

lexicon = {}

spamCount = 0

hamCount = 0

fileCount = 0

#for loop responsible for spam lexeme placement into a dictionary

for file in os.listdir(spamDirectory):

if file.endswith(".txt") and fileCount <= fileLimit:

fileName = spamDirectory + "\\" + file

with open(fileName, encoding="Latin-1") as f:

for line in f:

for word in re.split('\W+', line):

if len(word) > 1:

if word in lexicon.keys():

lexicon[word].spamCount += 1

else:

lexema = Lexeme(word, 1, 0, 0, 0, 0)

lexicon[word] = lexema

spamCount += 1

fileCount += 1

fileCount = 0

#for loop responsible for ham lexeme placement into the same dictionary as spam

for file in os.listdir(hamDirectory): #all files in directory

if file.endswith(".txt") and fileCount <= fileLimit:

fileName = hamDirectory + "\\" + file

with open(fileName, encoding="Latin-1") as f:

for line in f:

for word in re.split('\W+', line):

if len(word) > 1:

if word in lexicon.keys():

lexicon[word].hamCount += 1

else:

lexema = Lexeme(word, 0, 1, 0, 0, 0)

lexicon[word] = lexema

hamCount += 1

fileCount += 1

return [lexicon, spamCount, hamCount]

#function responsible for counting the probabilities of lexemes

def probabilities(answerList):

#dictionary with all lexemes

lexicon = answerList[0]

#total word counts from all read files

totalSpamCount = answerList[1]

totalHamCount = answerList[2]

for key in lexicon.keys():

#probability that lexeme is IN Spam directory

lexicon[key].pWS = lexicon[key].spamCount / totalSpamCount

#probability that lexeme is IN Ham directory

lexicon[key].pWH = lexicon[key].hamCount / totalHamCount

#probability that lexeme is Spam

if lexicon[key].pWS <= 0: #if lexeme isnt found in any of the spam files

lexicon[key].pSW = 0.01

elif lexicon[key].pWH <= 0: #if lexeme isnt found in any of the ham files

lexicon[key].pSW = 0.99

else: #if lexeme is found in both spam and ham files

lexicon[key].pSW = lexicon[key].pWS / (lexicon[key].pWS + lexicon[key].pWH)

#print(lexicon[key].word, " Spam Count: ", lexicon[key].spamCount, " Ham count: ", lexicon[key].hamCount, " pWS ", lexicon[key].pWS, " pWH ", lexicon[key].pWH, " pSW ", lexicon[key].pSW)

#returns full lexicon with all probabilites

return lexicon

#function responsible for the analysis of whole folder of testing data files

def analyse(analyseDirectory, lexicon, isSpam):

results = []

#this part is almost the same as read function except this one doesn't make any Lexeme objects and instead

#just places the spam probabilities of lexemes into a list and then passes it to another function for further analysis

for file in os.listdir(analyseDirectory): #all files in directory

if file.endswith(".txt"):

fileName = analyseDirectory + "\\" + file

with open(fileName, encoding="Latin-1") as f:

list1 = []

for line in f:

for word in re.split('\W+', line):

if len(word) > 1:

if word in lexicon.keys():

list1.append(lexicon[word].pSW)

else:

list1.append(isSpam)

print(fileName)

tempResult = probabilityThatFileIsSpam(list1, 20) #after gathering all lexemes from file

results.append(tempResult)

#function that takes a number of lexemes that are farthest from neutral point

#(0.5 - neutral point, while 0 is min and 1 is max) from a file that needs to be analyzed

#and then based on the probability of spam on each of these lexemes,

#calculates the overall probability that file is spam

#numberOfMaxValues - a maximum number of lexemes to be taken into account when calculating overall probability

def probabilityThatFileIsSpam(list1, numberOfMaxValues):

maxValues = []

currentMaxDifference = 0

currentMax = 0

maxIndex = 0

for i in range(0, numberOfMaxValues):

#appends 0 to extend the lenght of list which allows accessing by index (NOT REALLY NECESSARY)

maxValues.append(0)

for x in range(0, len(list1)):

#difference calculation to find values that are farthest from neutral point - 0.5

difference = 0

if list1[x] >= 0.5:

difference = list1[x] - 0.5

elif list1[x] < 0.5:

difference = 0.5 - list1[x]

if currentMaxDifference < difference:

currentMaxDifference = difference

currentMax = list1[x]

maxIndex = x

maxValues[i] = currentMax

list1[maxIndex] = 0.5

maxIndex = 0

currentMax = 0

currentMaxDifference = 0

#calculates the overall probability p by using the following formula:

# p = (p1 \* p2 \* p3 ... \* pn)/((p1 \* p2 \* p3 ... \* pn) + ((1-p1) \* (1-p2) \* (1-p3) ... \* (1-pn))

topSide = 1

bottomSide = 1

for i in range(0, len(maxValues)):

topSide = topSide \* maxValues[i]

bottomSide = bottomSide \* (1 - maxValues[i])

spamProbability = (topSide / (topSide + bottomSide)) \* 100

#print(maxValues)

print(spamProbability, "%")

if spamProbability > 50:

print("SPAM")

elif spamProbability < 50:

print("NOT SPAM")

return spamProbability

#directories for learning purposes \*PATHS ARE NOT RELATIVE as ???apparently os.listdir doesn't accept relative paths???\*

spamDirectory = "D:\Tadas\KALBUTEORIJA\Python\spam"

hamDirectory = "D:\Tadas\KALBUTEORIJA\Python\ham"

#directory for files that need to be analyzed

analyseDirectory = "D:\Tadas\KALBUTEORIJA\Python\\analysis"

# answerList values: 0 - dictionary/lexicon, 1 - spam word count, 2 - ham word count

answerList = readFilesIntoDictionary(spamDirectory, hamDirectory, 150)

lexicon = probabilities(answerList)

analyse(analyseDirectory, lexicon, 0.4)

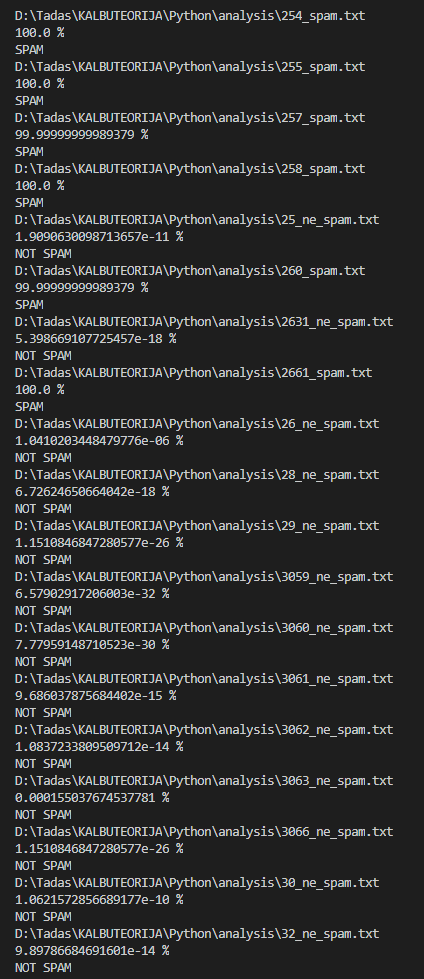
Tyrimai ir rezultatai:

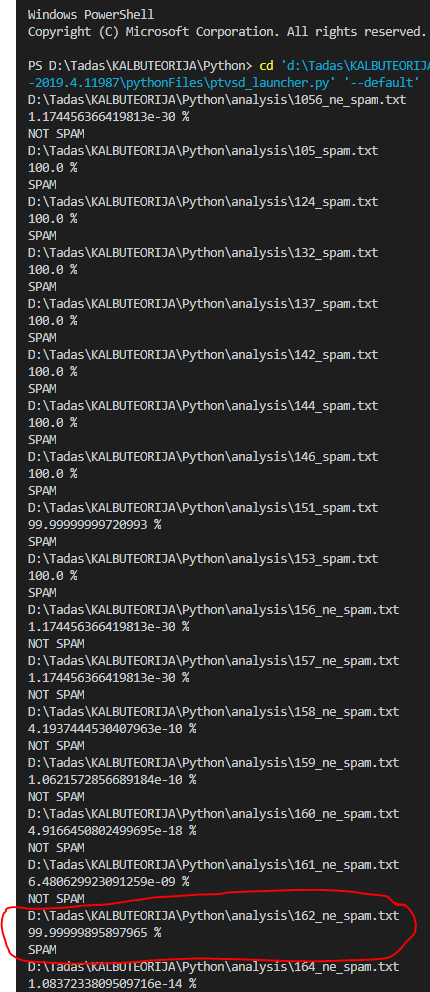
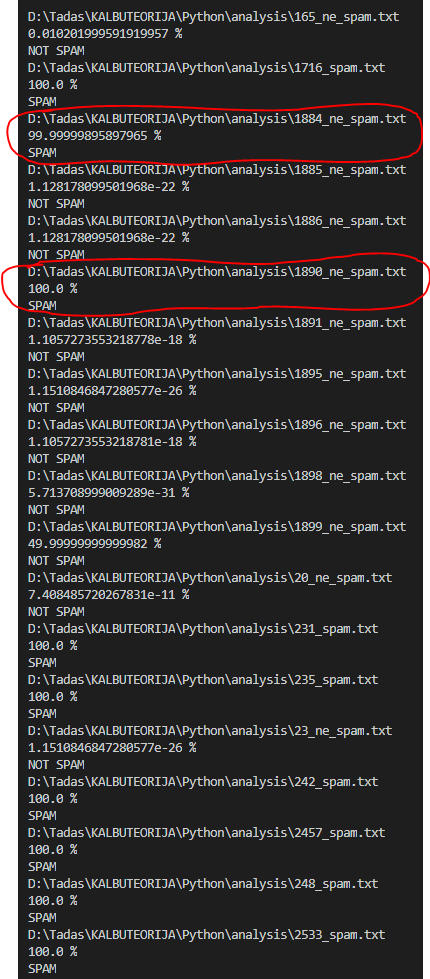
|  |  |
| --- | --- |
| Tikslumo priklausomybė nuo leksemos sutinkamos pirmą kartą spam tikimybės | |
| Leksemos tikymybė | Tikslumas |
| 0.45 | 98% |
| 0.4 | 97% |
| 0.35 | 96% |
| 0.3 | 96% |

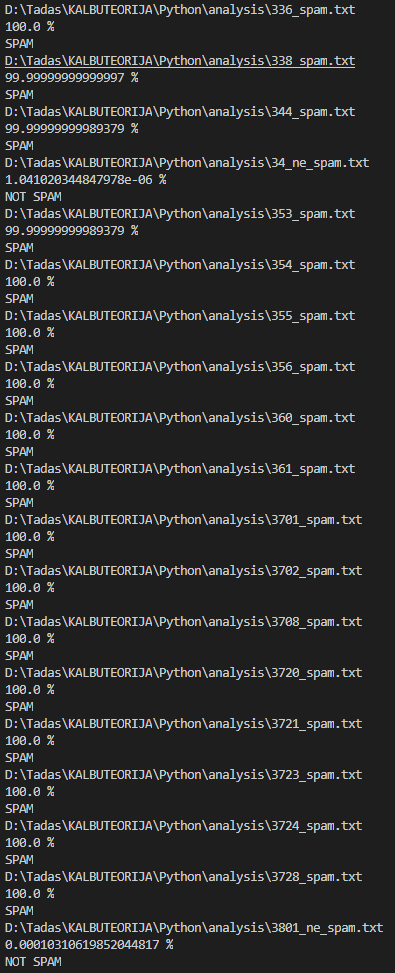
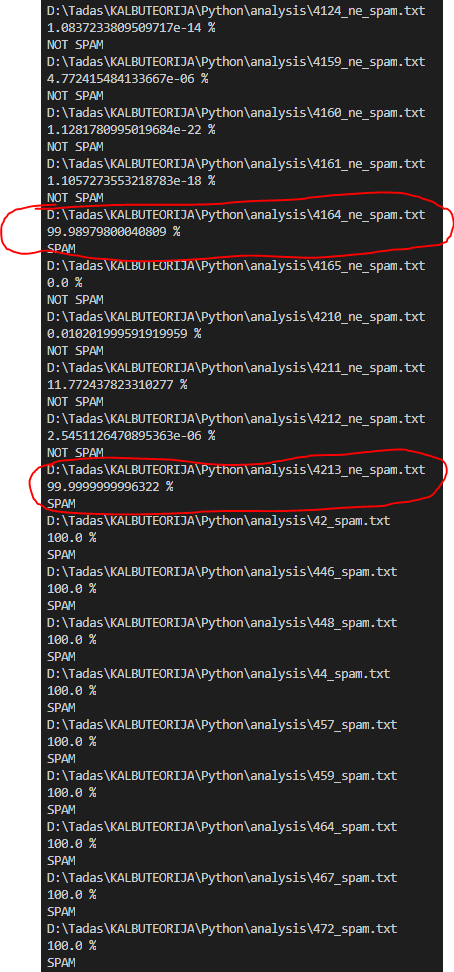
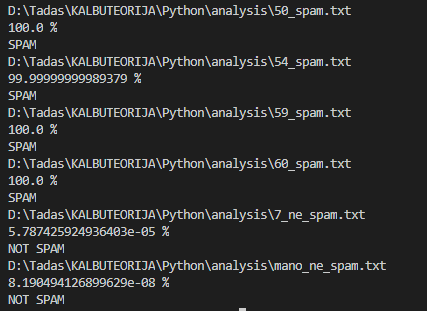
|  |  |
| --- | --- |
| Tikslumo priklausomybė nuo leksemų skaičiaus N | |
| Leksemų skaičius (N) | Tikslumas |
| 22 | 96% |
| 20 | 95% |
| 18 | 95% |
| 16 | 95% |
| 14 | 94% |

|  |  |
| --- | --- |
| Tikslumo priklausomybė nuo slenksčio dydžio | |
| Slenkstis | Tikslumas |
| 0.4 | 95% |
| 0.45 | 95% |
| 0.5 | 95% |
| 0.55 | 96% |
| 0.6 | 96% |

Rezultatų pavydys, naudojant N = 16, 0.4 dydžio slenkstį, 150 apsimokymo duomenų, 100 testavimo duomenų. Šiuo atveju tikslumas 96%. Raudonai apvesti false positive atvejai (true false atvejų nepasitaikė)







# Išvada

Naudojant Bajeso teoremą galima nesunkiai atrūšiuoti spamą nuo ne spamo, bei keičiant įvairius kintamuosius išgauti gana tikslų rezultatą. Iš tyrimo matyti, kad tikslingiausia naudoti apie 22 leksemas nustatinėjant bendrą tikimybę, apmokyti algoritmą su bent 150 skirtingų duomenų failų, taip pat naudoti 0.45-0.6 dydžio slenkstį bei 0.45 naujai sutiktos leksemos tikimybę.