KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INTELEKTIKOS PAGRINDAI 2019

Laboratorinio darbo 4 ataskaita

Darbą atliko:

IFF-6/6 gr. studentas

Ignas Jasonas

Priėmė:

Dėstytojas [Germanas](https://moodle.ktu.edu/course/view.php?id=2671) Budnikas

KAUNAS 2019

## Bajeso teorema

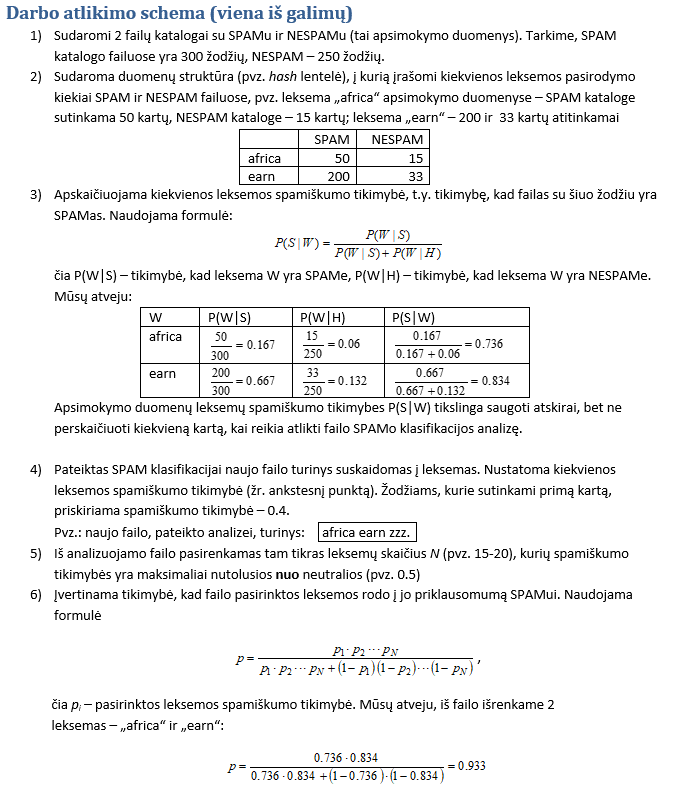
### Užduotis



### Darbo atikimas

Programa parašyta C# kalba. Mokymuisi programai skirta apie 90% duomenų rinkinių.

Darbas buvo atliktas vadovaujantis programos užduotyje aprašyta galima užduoties atlikimo schema:



### Programos kodas

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace L4\_ignjas\_IFF66

{

class Program

{

const string learningSpamPath = @"../../../learning/spam";

const string learningNotSpamPath = @"../../../learning/ne\_spam";

const string testingNotSpamPath = @"../../../testing/ne\_spam";

const string testingSpamPath = @"../../../testing/spam";

const int leksemuSkaicius = 1;

const double newWordSpamProbability = 0.4;

const double neutralProbability = 0.5;

static void Main(string[] args)

{

Dictionary<string, LearningCounter> learningTable = new Dictionary<string, LearningCounter>();

int learningSpamFilesCount = 0;

int learningSpamWordsCount = 0;

int learningNotSpamWordsCount = 0;

int learningNotSpamFilesCount = 0;

//LEARNING PHASE SPAM

foreach (string file in Directory.EnumerateFiles(learningSpamPath, "\*.txt"))

{

string contents = File.ReadAllText(file);

string word = "";

for (int i = 0; i < contents.Length; i++)

{

char character = contents[i];

//find words

if (character >= 'a' && character <= 'z' || character >= 'A' && character <= 'Z'

|| character >= '0' && character <= '9' || character == '\'' || character == '$' || character == '"')

{

word += character;

}

else if (word != "")

{

learningSpamWordsCount++;

if (!learningTable.ContainsKey(word))

{

learningTable.Add(word, new LearningCounter());

learningTable[word].increaseSpam();

}

else

{

learningTable[word].increaseSpam();

}

word = "";

}

}

learningSpamFilesCount++;

}

//LEARNING PHASE NOT SPAM

foreach (string file in Directory.EnumerateFiles(learningNotSpamPath, "\*.txt"))

{

string contents = File.ReadAllText(file);

string word = "";

for (int i = 0; i < contents.Length; i++)

{

char character = contents[i];

//find words

if (character >= 'a' && character <= 'z' || character >= 'A' && character <= 'Z'

|| character >= '0' && character <= '9' || character == '\'' || character == '$' || character == '"')

{

word += character;

}

else if (word != "")

{

learningNotSpamWordsCount++;

if (!learningTable.ContainsKey(word))

{

learningTable.Add(word, new LearningCounter());

learningTable[word].increaseNotSpam();

}

else

{

learningTable[word].increaseNotSpam();

}

word = "";

}

}

learningNotSpamFilesCount++;

}

Console.WriteLine(learningNotSpamWordsCount);

//CALCULATE ALL SPAM PROBABILITIES

foreach (KeyValuePair<string, LearningCounter> entry in learningTable)

{

// do something with entry.Value or entry.Key

entry.Value.calculateSpamProbability(learningSpamWordsCount, learningNotSpamWordsCount);

}

//TESTAVIMAS

double truePositiveSpam = 0;

double truePositiveSpamCount = 0;

double falsePositiveSpam = 0;

double falsePositiveSpamCount = 0;

double programAccuracy = 0;

double totalSpamAnalyzes = 0;

foreach (string file in Directory.EnumerateFiles(testingSpamPath, "\*.txt"))

{

List<AnalyzeHelper> analyzeList = new List<AnalyzeHelper>();

string contents = File.ReadAllText(file);

string word = "";

for (int i = 0; i < contents.Length; i++)

{

char character = contents[i];

//find words

if (character >= 'a' && character <= 'z' || character >= 'A' && character <= 'Z'

|| character >= '0' && character <= '9' || character == '\'' || character == '$' || character == '"')

{

word += character;

}

else if (word != "")

{

if (!learningTable.ContainsKey(word))

{

AnalyzeHelper newWord = new AnalyzeHelper(word, newWordSpamProbability, neutralProbability);

if (!analyzeList.Any(x => x.word == word)) analyzeList.Add(newWord);

}

else

{

LearningCounter wordInfo = learningTable[word];

AnalyzeHelper newWord = new AnalyzeHelper(word, wordInfo.spamProbability, neutralProbability);

if (!analyzeList.Any(x => x.word == word)) analyzeList.Add(newWord);

}

word = "";

}

}

//Sort by difference

analyzeList = analyzeList.OrderByDescending(x => x.probabilityDifference).ToList();

int wordsPicked = 0;

double finalSpamProbability = 0;

double upperFormula = 1;

double lowerFirstFormula = 1;

double lowerSecondFormula = 1;

foreach (AnalyzeHelper analyzedWord in analyzeList)

{

if (wordsPicked < leksemuSkaicius)

{

upperFormula \*= analyzedWord.spamProbability;

lowerFirstFormula \*= analyzedWord.spamProbability;

lowerSecondFormula \*= (1 - analyzedWord.spamProbability);

}

else

{

finalSpamProbability = (upperFormula / (lowerFirstFormula + lowerSecondFormula));

break;

}

wordsPicked++;

}

if (finalSpamProbability >= neutralProbability)

{

truePositiveSpam += finalSpamProbability;

truePositiveSpamCount++;

}

else

{

falsePositiveSpam += finalSpamProbability;

falsePositiveSpamCount++;

}

totalSpamAnalyzes++;

if (finalSpamProbability >= neutralProbability) Console.WriteLine(file + " probability : " + (int)(finalSpamProbability \* 100) + " %, Spamas");

else Console.WriteLine(file + " probability : " + (int)(finalSpamProbability \* 100) + " %, Ne spamas");

}

//NOT SPAM TESTING

double truePositiveNotSpam = 0;

double truePositiveNotSpamCount = 0;

double falsePositiveNotSpam = 0;

double falsePositiveNotSpamCount = 0;

int totalNotSpamAnalyzes = 0;

foreach (string file in Directory.EnumerateFiles(testingNotSpamPath, "\*.txt"))

{

List<AnalyzeHelper> analyzeList = new List<AnalyzeHelper>();

string contents = File.ReadAllText(file);

string word = "";

for (int i = 0; i < contents.Length; i++)

{

char character = contents[i];

//find words

if (character >= 'a' && character <= 'z' || character >= 'A' && character <= 'Z'

|| character >= '0' && character <= '9' || character == '\'' || character == '$' || character == '"')

{

word += character;

}

else if (word != "")

{

if (!learningTable.ContainsKey(word))

{

AnalyzeHelper newWord = new AnalyzeHelper(word, newWordSpamProbability, neutralProbability);

if (!analyzeList.Any(x => x.word == word)) analyzeList.Add(newWord);

}

else

{

LearningCounter wordInfo = learningTable[word];

AnalyzeHelper newWord = new AnalyzeHelper(word, wordInfo.spamProbability, neutralProbability);

if (!analyzeList.Any(x => x.word == word)) analyzeList.Add(newWord);

}

word = "";

}

}

//Sort by difference

analyzeList = analyzeList.OrderByDescending(x => x.probabilityDifference).ToList();

int wordsPicked = 0;

double finalSpamProbability = 0;

double upperFormula = 1;

double lowerFirstFormula = 1;

double lowerSecondFormula = 1;

foreach (AnalyzeHelper analyzedWord in analyzeList)

{

if (wordsPicked < leksemuSkaicius)

{

upperFormula \*= analyzedWord.spamProbability;

lowerFirstFormula \*= analyzedWord.spamProbability;

lowerSecondFormula \*= (1 - analyzedWord.spamProbability);

}

else

{

finalSpamProbability = (upperFormula / (lowerFirstFormula + lowerSecondFormula));

break;

}

wordsPicked++;

}

if (finalSpamProbability >= neutralProbability)

{

falsePositiveNotSpam += finalSpamProbability;

falsePositiveNotSpamCount++;

}

else

{

truePositiveNotSpam += finalSpamProbability;

truePositiveNotSpamCount++;

}

Console.WriteLine(finalSpamProbability);

totalNotSpamAnalyzes++;

if (finalSpamProbability >= neutralProbability) Console.WriteLine(file + " probability : " + (finalSpamProbability \* 100) + " %, Spamas");

else Console.WriteLine(file + " probability : " + string.Format("{0:00.00}", finalSpamProbability \* 100) + " %, Ne spamas");

}

if (falsePositiveNotSpamCount == 0) falsePositiveNotSpamCount = 1;

if (falsePositiveSpamCount == 0) falsePositiveSpamCount = 1;

if (truePositiveNotSpamCount == 0) truePositiveNotSpamCount = 1;

if (truePositiveSpamCount == 0) truePositiveSpamCount = 1;

double truePositive = (truePositiveSpamCount / totalSpamAnalyzes + truePositiveNotSpamCount / totalNotSpamAnalyzes) / 2;

double falsePositive = (falsePositiveSpamCount / totalSpamAnalyzes + falsePositiveNotSpamCount / totalNotSpamAnalyzes) / 2;

programAccuracy = (truePositive + falsePositive) / 2;

Console.WriteLine("True positive: " + truePositive);

Console.WriteLine("False positive: " + falsePositive);

Console.WriteLine("Tikslumas: " + truePositive \* 100 + " %");

Console.ReadKey();

}

}

public class LearningCounter

{

public int spam { get; set; }

public int notSpam { get; set; }

public double spamProbability { get; set; }

public LearningCounter()

{

spam = 0;

notSpam = 0;

spamProbability = 0;

}

public void increaseSpam()

{

spam++;

}

public void increaseNotSpam()

{

notSpam++;

}

public void calculateSpamProbability(int totalSpamWords, int totalNotSpamWords)

{

double PSW;

double PWS = spam / (double)totalSpamWords;

double PWH = notSpam / (double)totalNotSpamWords;

if (PWH == 0) PSW = 0.99;

else if (PWS == 0) PSW = 0.01;

else PSW = PWS / (PWS + PWH);

spamProbability = PSW;

}

}

public class AnalyzeHelper

{

public string word { get; set; }

public double spamProbability { get; set; }

public double probabilityDifference { get; set; }

public AnalyzeHelper(string word\_input, double spamProbability\_input, double neutralProbability\_input)

{

word = word\_input;

spamProbability = spamProbability\_input;

probabilityDifference = spamProbability - neutralProbability\_input;

if (probabilityDifference < 0) probabilityDifference = probabilityDifference \* -1;

}

}

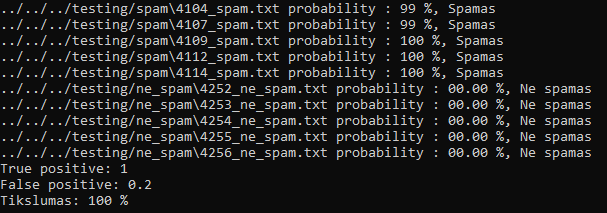
}

### Testavimas ir rezultatai

Testavimui buvo sukurti tu atskiri folderiai spam ir ne\_spam, kuriuose buvo po maždaug 50-75 duomenų failų.

### Kryžminė patikra

Segmenų skaičius N = 10. Buvo patikrinta 5 spam’ų ir 5 ne spam’ų failų (iš viso 10 eksperimentų). Rezultatai:

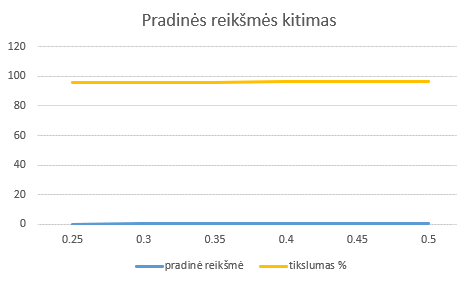


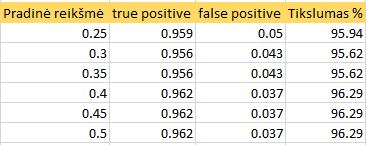
### Pradinės leksemos reikšmės keitimas

Šiame testavime buvo keičiama pradinė pirmą kartą sutinkamos leksemos spam tikimybė.

Leksemų skaičius – 10

Spamiškumo slenkstis – 0.5





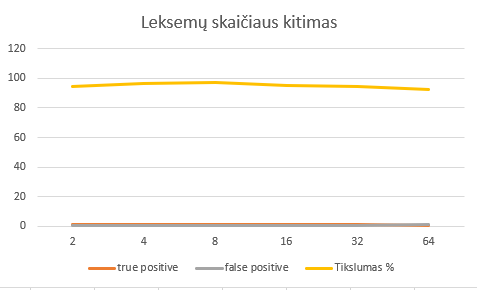
Iš rezultatų galime spręsti kad leksemų pradinė reikšmė turi nedidelę reikšmę tikslumui

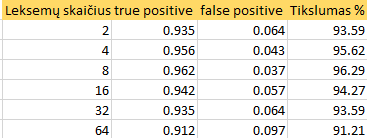
### Leksemų skaičiaus pokytis

Šiame testavime keičiama išrenkamų leksemų skaičius.

Pradinė leksemos reiškmė – 0.4

Spamiškumo slenkstis – 0.5





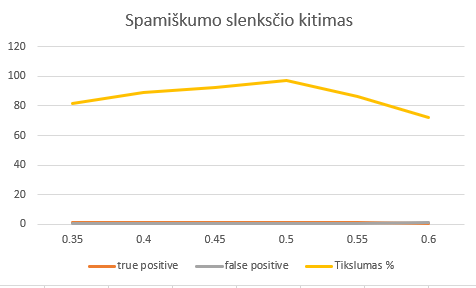
Iš rezultatų galime spręsti, kad didinant leksemų skaičių tikslumas mažėja, nes algoritmas pirmiausia tikrina žodžius kurie turi didžiausią skirtumą tarp neutralios reikšmės (0.5)

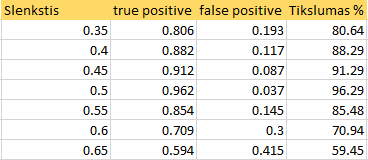
### Spamiškumo slenksčio kitimas

Šiame testavime keisime spamiškumo slenkstį. Jeigu failo spamo tikimybė perlipa šį slenkstį, jis užskaitomas kaip spamas.

Leksemų skaičius – 10

Pradinė leksemos reiškmė – 0.4





Iš šių rezutlatų galime matyti, kad spamiškumo slenkstis turi didelę įtaką klasifikavimo rezultatams. Slenkstis geriausius rezultatus duoda jei būna per vidurį – 0.5, nes tikimybės reiškmės rėžiai yra nuo 0 iki 1.

## Išvados

Atlikdamas laboratorinį darbą susipažinau su Bajeso teoremos panaudojimu klasifikavimo atvejuose ir praktiškai tai pritaikiau realiame gyvenimo atvejyje.