KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Intelektikos pagrindai (P176B101)

4 laboratorinio darbo ataskaita

Atliko:
IFF-6/10 gr. studentas
Edvinas Deksnys
2019 m. gegužės 9 d.
Priėmė:
Lekt. Budnikas Germanas

KAUNAS 2019

TURINYS

1. Darbo aprašymas

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

2. Programos kodas

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
//using ExcelLibrary.SpreadSheet;
using Excel = Microsoft.Office.Interop.Excel;

namespace Spamo_tikrintojas
{
   class Word
   {
      public int spam = 0;
      public int notspam = 0;
      public float prob;

      public Word() { }

      public Word(float prob)
      {
         this.prob = prob;
      }
   }

   //class W
   //{
   //   public string word;
   //   public float prob;

   //   public W(string word, float prob)
   //   {
   //      this.word = word;
   //      this.prob = prob;
   //   }
   //}

   class Program
   {
      //static public List<Word> table = new List<Word>();
      static public Dictionary<string, Word> table = new Dictionary<string, Word>();
      static public List<float> slenkstis = new List<float>();
      const float DefSlenkstis = 0.2f; // ??
      static public List<int> leksemuSk = new List<int>();
      const int DefLeksemuSk = 90; // ??
      static public List<float> tikimybeNaujam = new List<float>();
      const float DefTikimybe = 0.4f; // ??
      static public int wordsInSpam = 0;
      static public int wordsInNotSpam = 0;
      const string excelFile = @"C:\Users\Edvinas\Desktop\Intelektikos pagrindai\L4\diagrams.xlsx"; //
@"..\..\..\..\diagrams.xlsx";
```

```
        static public Excel._Worksheet sheet;
        static public Dictionary<string, List<string>> Files;

        const int folds = 10;

        static void Main(string[] args)
        {
            //*
            Excel.Application xlApp = new Excel.Application();
            xlApp.DisplayAlerts = false;
            Excel.Workbook workbook = xlApp.Workbooks.Open(excelFile, true, false);
            sheet = workbook.Sheets[1];
            //*/
            DirectoryInfo d1 = new DirectoryInfo(@"..\..\..\..\laiskai\Spamas\");
            FileInfo[] SpamFiles = d1.GetFiles("*.txt");//.txt
            DirectoryInfo d2 = new DirectoryInfo(@"..\..\..\..\laiskai\Ne_spamas\");
            FileInfo[] NotSpamFiles = d2.GetFiles("*.txt");//.txt
            Files = new Dictionary<string, List<string>>();
            ReadAll(SpamFiles, NotSpamFiles);

            for (int i = 1; i < 10; i++)
            {
                slenkstis.Add(0.1f * i);
                Console.WriteLine(slenkstis[i-1]);
                tikimybeNaujam.Add(0.1f * i);
            }

            for (int i = 0; i < 20; i++)
            {
                //leksemuSk.Add(1 + 2 * i);
                leksemuSk.Add(20 + 15 * i);
            }

            Console.WriteLine(slenkstis.Count);
            Console.WriteLine(leksemuSk.Count);
            Console.WriteLine(tikimybeNaujam.Count);

            int trainingSpamCount = SpamFiles.Length / folds;
            int trainingNotSpamCount = NotSpamFiles.Length / folds;

            for (int i = 0; i < folds; i++)
            {
                Training(SpamFiles, NotSpamFiles, trainingSpamCount, trainingNotSpamCount, i);
                // case 1
                Console.WriteLine();
                Console.Write("Case1: ");
                for(int j = 0; j < slenkstis.Count; j++)
                {
                    TestCase(slenkstis[j], DefLeksemuSk, DefTikimybe, j + 1, i * 3 + 1, SpamFiles,
NotSpamFiles, trainingSpamCount, trainingNotSpamCount, i);
                    Console.Write(j + " ");
                }
                // case 2
                Console.WriteLine();
                Console.Write("Case2: ");
                for (int j = 0; j < leksemuSk.Count; j++)
                {
                    TestCase(DefSlenkstis, leksemuSk[j], DefTikimybe, j + slenkstis.Count + 2, i * 3 + 1,
SpamFiles, NotSpamFiles, trainingSpamCount, trainingNotSpamCount, i);
```

```
                    Console.Write(j + " ");
                }
                // case 3
                Console.WriteLine();
                Console.Write("Case3: ");
                for (int j = 0; j < tikimybeNaujam.Count; j++)
                {
                    TestCase(DefSlenkstis, DefLeksemuSk, tikimybeNaujam[j], j + slenkstis.Count + leksemuSk.Count + 4, i * 3 + 1, SpamFiles, NotSpamFiles, trainingSpamCount, trainingNotSpamCount, i);
                    Console.Write(j + " ");
                }
            }
            //*
            workbook.Save();
            workbook.Close();//*/
            Console.WriteLine("Finished!");
            Console.ReadKey();
        }

        static public void TestCase(float slenkstis, int leksemuSk, float tikimybeNaujam, int x, int y, FileInfo[] SpF, FileInfo[] NSpF, int Scount, int NScount, int step)
        {
            int SpamFound = 0;  // TP spam identified as spam
            int SpamMissed = 0; // FN spam identified as not spam
            int NSpamFound = 0; // TN not spam identified as not spam
            int NSpamMissed = 0;// FP not spam identified as spam

            for (int i = step * Scount; i < (step - 1 == folds ? SpF.Length : (step + 1) * Scount); i++)
            {
                bool res = CheckIfSpam(SpF[i].FullName, slenkstis, leksemuSk, tikimybeNaujam);
                if (res)
                {
                    SpamFound++;
                } else
                {
                    SpamMissed++;
                }
            }
            for (int i = step * NScount; i < (step - 1 == folds ? NSpF.Length : (step + 1) * NScount); i++)
            {
                bool res = CheckIfSpam(NSpF[i].FullName, slenkstis, leksemuSk, tikimybeNaujam);
                if (res)
                {
                    NSpamMissed++;
                }
                else
                {
                    NSpamFound++;
                }
            }

            //TPR = TP / (TP + FN)
            float TPR = (float) SpamFound / (SpamFound + SpamMissed);
            //FPR = FP / (FP + TN)
            float FPR = (float) NSpamMissed / (NSpamMissed + NSpamFound);
            //ACC = (TP + TN) / (TP + TN + FP + FN)
            float ACC = (float) (SpamFound + NSpamFound) / (SpamFound + SpamMissed + NSpamFound + NSpamMissed);
            //writing results
```

```
            //*
            sheet.Cells[x+1][y+1].Value = TPR;
            sheet.Cells[x+1][y+1+1].Value = FPR;
            sheet.Cells[x+1][y+2+1].Value = ACC;
            //*/
            Console.WriteLine("TP: " + SpamFound + " FN: " + SpamMissed + " TN: " + NSpamFound + " FP: " + NSpamMissed);
            //Console.WriteLine("TPR: " + TPR + " FPR: " + FPR + " ACC " + ACC);
            //Workbook book = Workbook.Load(excelFile);
            //Worksheet sheet = book.Worksheets[0];
            //book.Save(excelFile);
        }

        static public bool CheckIfSpam(string filename, float slenkstis, int leksemuSk, float tikimybeNaujam)
        {
            List<string> words = Files[filename];

            List<float> wordsProbs = words.Distinct().Select(x =>
            {
                if (table.ContainsKey(x))
                {
                    return table[x].prob;
                }
                else
                {
                    return tikimybeNaujam;
                }
            }).OrderByDescending(prob => prob).Take(leksemuSk).ToList<float>(); // .Where(word => word.prob > slenkstis)
              //      }).OrderByDescending(prob => prob).Where(prob => prob > slenkstis).Take(leksemuSk).ToList<float>();

            float P1 = wordsProbs.Aggregate(1f, (x, y) => x * y);
            float P2 = wordsProbs.Aggregate(1f, (x, y) => x * (1-y));
            Console.WriteLine("P1 " + P1 + " P2 " + P2);
            float P = (float)P1 / (P1 + P2);
            Console.WriteLine("P: " + P);
            return P > slenkstis;
        }

        static public void Training(FileInfo[] SpF, FileInfo[] NSpF, int Scount, int NScount, int step)
        {
            table = new Dictionary<string, Word>();
            Console.WriteLine("Experiment NR: " + step);
            for (int i = 0; i < step * Scount; i++) // ??
            {
                //Console.Write(i + " ");
                readFile(SpF[i].FullName, true);
            }
            for (int i = (step - 1 == folds ? SpF.Length : (step + 1) * Scount) + 1; i < SpF.Length; i++)  // ??
            {
                //Console.Write(i + " ");
                readFile(SpF[i].FullName, true);
            }
            //Console.WriteLine("////// ");
            for (int i = 0; i < step * NScount; i++)
            {
                //Console.Write(i + " ");
```

```
                readFile(NSpF[i].FullName, false);
            }
            for (int i = (step - 1 == folds ? NSpF.Length : (step + 1) * NScount) + 1; i < NSpF.Length; i++)
            {
                //Console.Write(i + " ");
                readFile(NSpF[i].FullName, false);
            }
            //Console.WriteLine();
            //List<W> disp = new List<W>();
            foreach (KeyValuePair<string, Word> entry in table) // ??
            {
                //Console.WriteLine(entry.Key + ": " + entry.Value.spam + "/" +wordsInSpam + " ;  "+ entry.Value.notspam +"/" +wordsInNotSpam);
                float WS = (float)entry.Value.spam / wordsInSpam;
                float WH = (float)entry.Value.notspam / wordsInNotSpam;
                float SW = WS / (WS + WH);
                if(WS == 0 || SW < 0.01)
                {
                    SW = 0.01f;
                } else if (SW == 1) {
                    SW = 0.99f;
                }
                //Console.WriteLine("WS: " + WS + " WH: " + WH + " SW: " + SW);
                table[entry.Key].prob = SW; // ??
                //disp.Add(new W(entry.Key, SW));
            }

           /* disp = disp.OrderByDescending(w => w.prob).ToList<W>();
            foreach(W w in disp)
            {
                Console.WriteLine(w.word + ": " + w.prob);
            }*/
        }

        static public void ReadAll(FileInfo[] SpF, FileInfo[] NSpF)
        {
            for (int i = 0; i < SpF.Length; i++)
            {
                Files[SpF[i].FullName] = Filter(SpF[i].FullName);
            }
            for (int i = 0; i < NSpF.Length; i++)
            {
                Files[NSpF[i].FullName] = Filter(NSpF[i].FullName);
            }
        }



        static public List<string> Filter(string filename)
        {
            string text = File.ReadAllText(filename);
            text = text.ToLower();

            List<string> words = Regex.Matches(text, "[a-z0-9]+|[\"\'$]").Cast<Match>().Select(m => m.Value).ToList<string>();
            return words;
        }

        static public void readFile(string filename, bool isSpam)
```

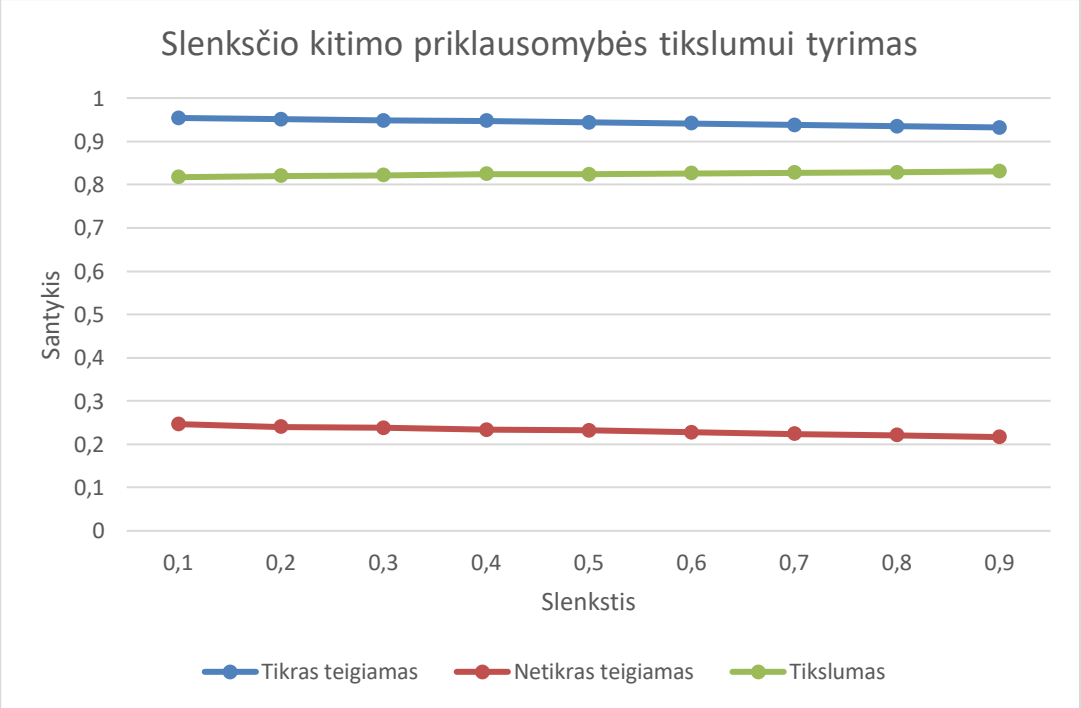
```
        {
            List<string> words = Files[filename];
            foreach (string word in words)
            {
                if (!table.ContainsKey(word))
                {
                    table[word] = new Word();
                }
                if (isSpam)
                {
                    wordsInSpam++;
                    table[word].spam++;
                }
                else
                {
                    wordsInNotSpam++;
                    table[word].notspam++;
                }
            }
        }
    }
}
```

3. Rezultatai

Atlikus 10 etapų kryžminę patikrą ir suskaičiavus vidutines reikšmes keičiant slenksčio, leksemų skaičiaus ir naujai rastų leksemų tikimybes, gavau tokius rezultatus:

3.1. Slenkčio kitimo priklausomybės tikslumui tyrimas

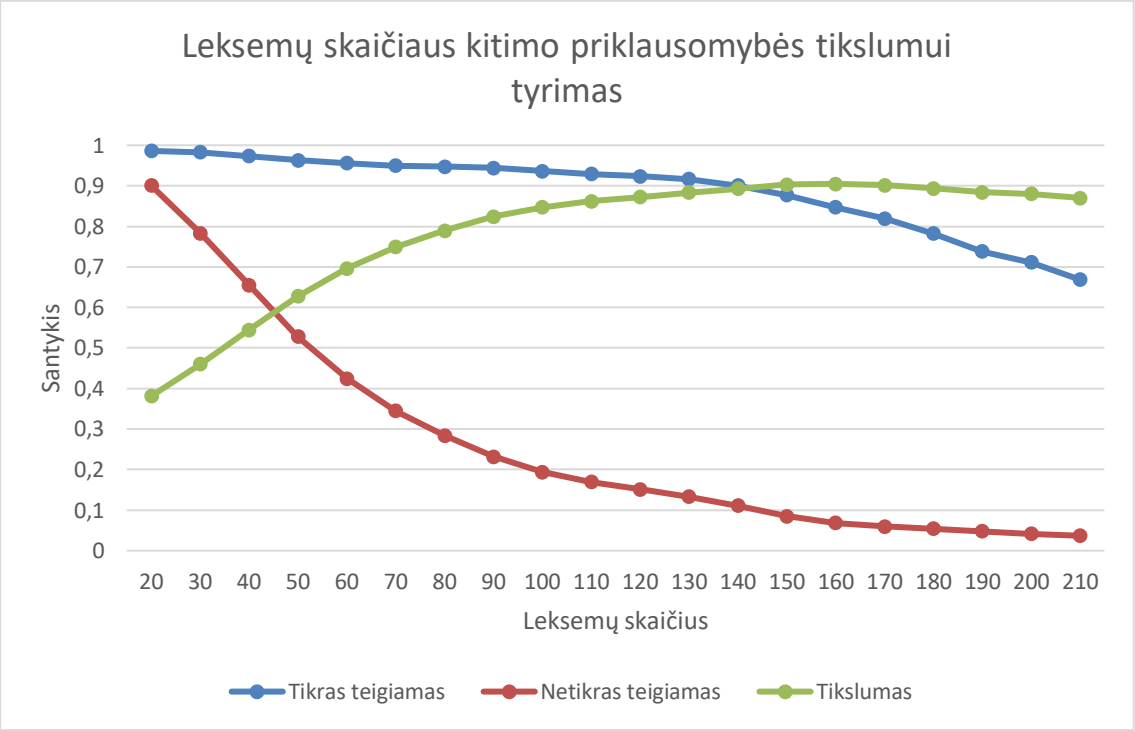
Leksemų skaičius yra 80, tikimybė naujai surastai leksemai yra 0.4



Slenksčio kitimo įtaka klasifikatoriaus tikslumui yra minimali. Didėjant slenksčiui tik šiek tiek geriau yra atskiriamas ne spamas. Kadangi gauti duomenys pakankamai gerai atskiriami kaip spamas arba ne spamas, tai slenksčio įtaka yra minimali.

3.2. Leksemų skaičiaus kitimo priklausomybės tikslumui tyrimas

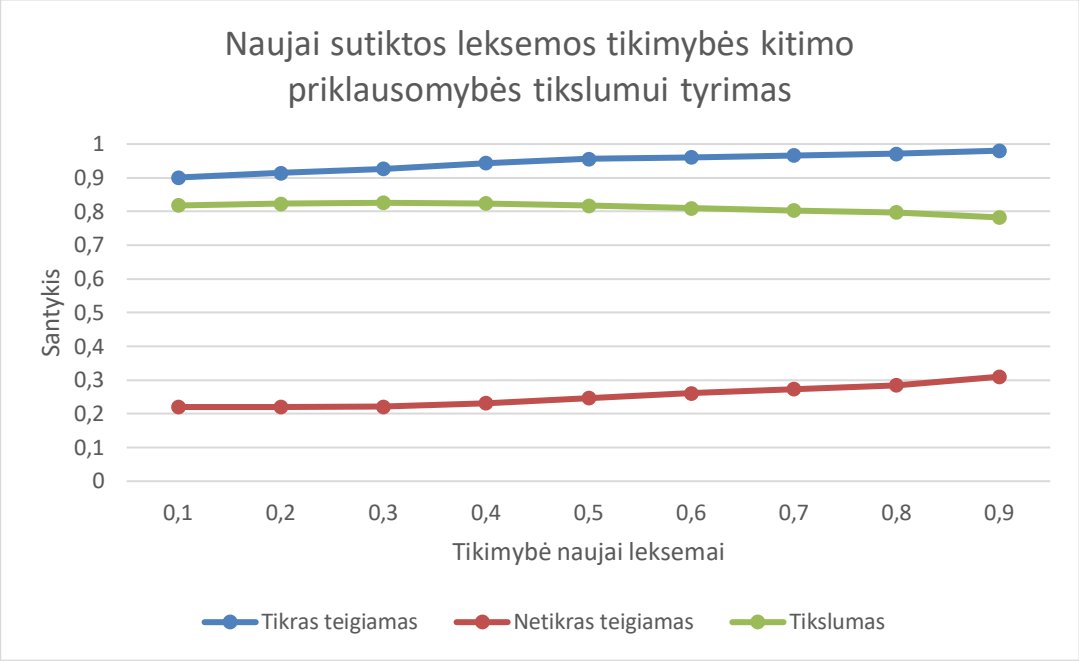
Slenkstis yra 0.5, tikimybė naujai surastai leksemai yra 0.4



Leksemų skaičius turi didelę įtaką klasifikatoriaus tikslumui. Kuo daugiau leksemų paimame tuo tiksliau klasifikatorius veikia, nes ne spamą tinkamai klasifikuoja kaip ne spamą. Tuo tarpu spamo atskyrimas praktiškai nekinta tik šiek tiek mažėja. Visa tai vyksta iki 140 leksemų skaičiaus. Pasirenkant daugiau leksemų klasifikavimui tikslumas sustoja ir pradeda mažėti, nes spamas vis klaidingiau atpažįstamas, nors ne spamas atpažįstamas geriau, tačiau jo atpažinimo santykis didėja lėčiau – klaidos atpažįstant ne spamą santykis mažėja lėčiau, nei spamo atpažinimo santykis.

3.3. Slenkčio kitimo priklausomybės tikslumui tyrimas

Leksemų skaičius yra 80, slenkstis yra 0.5



Naujai sutiktai leksemai priskiriamos tikimybės kitimas turi nedidelę įtaką klasifikatoriaus tikslumui. Didinant tikimybę tikslumas šiek tiek krenta, nes didėja klaida atpažįstant ne spamą greičiau nei gerėja ne spamo atpažinimas.

4. Išvados

Klasifikatorius parinkus tinkamus parametrus (slenkstį - 0.5, tikimybę naujai leksemai - 0.4, leksemų skaičių - 150) pasiekia klasifikavimo tikslumą 90%. Atsižvelgus, jog vertiname tik atskiras leksemas ir ne jų kombinacijas ar leksemų naudojimą atskirame laiške. Tai galime teigti, kad klasifikatoriu veikia gan gerai. Didžiausią įtaką klasifikatoriui turėjo leksemų skaičius, tuo tarpu slenkstis ir tikimybė naujai leksemai įtaka klasifikatoriaus tikslumui turi gan mažą.