МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет информационных технологий и прикладной математики

Курсовая работа по курсу «Дискретный анализ»

студента группы М8О-208Б-17
Куликова Алексея Владимировича

Преподаватель
_____ А. А. Журавлев
«___» _____ 2019 г.

Москва, 2019 г.

Курсовой проект на тему «Автоматическая классификация документов»

Задание

Разработать программу для автоматической классификации статьей при помощи наивного Байесовского классификатора. Программа должна иметь два режима работы: режим обучения и режим классификации.

В режиме обучения программа должна собрать статистику из обучающей выборки и сохранить ее в файл для последующего использования в режиме классификации.

В режиме классификации программа для каждой статьи из входного документа должна вывести предполагаемые теги.

Формат входных файлов при обучении:

```
<Количество строк в вопросе [n]>
<Заголовок вопроса>
<Текст вопроса [n строк]>
<Тег 1>,<Тег 2>,...,<Тег m>
```

Формат входных файлов при запросах:

```
<Количество строк в вопросе [n]>
<Заголовок вопроса>
<Текст вопроса [n строк]>
```

Формат выходного файла: для каждого запроса в отдельной строке выводится предполагаемый набор тегов, через запятую.

Метод решения

В основе алгоритма классификации лежит теорема Байеса.

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

, где:

- $P(c|d)$ - вероятность того, что документ d принадлежит классу c. Это и есть вероятность, на основе которой и будет решаться стоит ли помечать статью d тегом c.;
- $P(d|c)$ - вероятность встречи документа d среди документов, имеющих класс c;
- $P(c)$ - безусловная вероятность встречи документа с классом c среди всех документов выборки;

- $P(d)$ - безусловная вероятность встречи именно документа d среди всех документов выборки.

Теорема Байеса по сути позволяет, зная вероятность с которой причина приводит к событию, узнать вероятность того, что событие произойдет именно по этой причине.

Суть классификации будет заключаться в определении нескольких наиболее вероятных классов для документа (о том как будут выбираться эти несколько классов и как понять сколько их будет написано ниже).

Для этого для классифицируемого документа найдем

$$c_{map} = \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

Для каждого конкретного документа знаменатель $P(d)$ одинаков, и поэтому никак не может повлиять на результат вычисления максимума, следовательно его можно опустить.

$$c_{map} = \arg\max_{c \in C} [P(d|c)P(c)]$$

В этом выражении

$$\begin{aligned} P(d|c)P(c) &= P(w_1, \cdots, w_n|c)P(c) \\ &= P(c, w_1, \cdots, w_n) \\ &= P(c)P(w_1|c)P(w_2|c, w_2) \cdots P(w_n|c, w_1, w_2, \cdots, w_{n-1}) \end{aligned}$$

Подобное выражение рассчитать либо очень сложно, либо невозможно вовсе, поэтому следует сделать предположение об условной независимости слов в тексте, т.е.

$$P(w_i|c, w_j) = P(w_i|c)$$

для любых слов.

Отсюда вытекает, что

$$\begin{aligned} P(d|c)P(c) &= P(w_1, \cdots, w_n|c)P(c) \\ &= P(c)P(w_1|c) \cdots P(w_n|c) \\ &= P(c) \prod_{i=1}^{n} P(w_i|c) \end{aligned}$$

В этом предположении и заключается наивность наивного Байесовского классификатора.

Т.о.

$$c_{map} = \arg\max_{c \in C}[P(c)\prod_{i=1}^{n} P(w_i|c)]$$

и есть решение задачи классификации.

Вероятность $P(c)$ можно вычислить так:

$$P(c) = \frac{D_c}{D}$$

, где D_c – количество документов класса C в выборке, а D – общее количество документов в выборке.

Теперь вычислим вероятности $P(w_i|c)$ встречи слова w_i в документе калсса C Это можно сделать так:

$$P(w_i|c) = \frac{W_{ic}}{\sum\limits_{w_k \in V} W_{kc}}$$

, где W_ic – частота слова w_i в документах класса C в выборке, а V – множество всех уникальных слов в выборке.

Т.е. просто как отношение частоты слова w_i к суммарному количеству слов во всех документах класса C.

И тут возникает вполне закономерная проблема – проблема слов с нулевой частотой. Если на этапе классификации в документе встретится слово, при использовании вышеуказанной формулы для вычисления вероятности $P(w_i|c)$, то для этого слова данная вероятность будет равна 0. Таким образом обнулиться все произведение. Поэтому данный документ нельзя будет корректно классифицировать.

Эту проблему можно решить используя аддитивное сглаживание Лапласа. По сути нужно вместо истинной частоты слова, использовать на единицу большую. Получаем:

$$P(w_i|c) = \frac{W_{ic} + 1}{|V| + \sum\limits_{w_k \in V} W_{kc}}$$

Таким образом оценка вероятности слова никогда не станет нулевой, а значит проблема решена. Это, конечно, слегка поменяет распределение оценок вероятностей принадлежности к конкретному классу (если точнее, сдвинет его в пользу менее вероятных классов), но это не может сильно изменить результат, потому что, как правило, если документ действительно принадлежит классу C, то оценка его вероятности значительно выше оценок для других классов.

Теоретически теперь все верно, но на практике может возникнуть еще одна проблема. При подсчете оценки возникает потребность в перемножении большого количества довольно малых чисел. А как известно в компьютерах действительные числа имеют конечную точность, поэтому может возникнуть проблема арифметического переполнения снизу. Т.е. произведение становится настолько малым числом, что его нельзя представить числом с плавующей точкой с конечной точностью.

Для того чтобы этого избежать можно воспользоваться свойством логарифма произведения $\log ab = \log a + \log b$.

К тому же логарифм от очень малого числа будет хоть и отрицательным, но по модулю значительно большим исходного числа, поэтому его можно использовать для устранения проблемы переполнения.

А в силу монотонности логарифма значения аргумента, при которых достигается максимум не изменятся.

Таким образом получаем:

$$c_{map} = \arg\max_{c \in C} [\log \frac{D_c}{D} + \sum_{i=1}^{n} \log \frac{W_{ic} + 1}{|V| + \sum\limits_{w_k \in V} W_{kc}}]$$

Обучение

Обучение модели – этап построения модели на котором собирется необходимая для классификации статистика из обучающей выборки.

В нашем случае необходимо собрать следующую статистику из выборки, которая будет использоваться на этапе классификации:

- общее количество различных классов;
- количества документов для каждого класса;
- общее количество документов;
- общее количество уникальных слов;
- для каждого класса общее количество слов ему принадлежащее;
- для каждого слова количество количество классов, которым оно принадлежит;
- для каждого слова для каждого класса, которому оно принадлежит имя класса и частоту этого класса для этого конкретного слова;

По мере сбора статистики она записывается в файл для последующего ее использования классификатором.

Совокупность этой информации и есть модель классификатора. Затем на этапе классификации собранная информация будет использовать для подсчета оценок вероятности принадлежности документа тому или иному классу.

Обработка данных

При обучении модели имеет смысл обрабатывать данные. На это есть масса причин. Во-первых данные, несущие одну и ту же смысловую нагрузку могут быть записаны несколькими способами.

В нашем случае, например, слово может быть написано полностью в нижнем регистре, если оно находится где-то в середине предложения. Но оно может писаться и с заглавной буквы, если оно стоит в начале предложения. Хотя это одно и то же слово, две его разных формы считались бы разными словами и поэтому, при оценка вероятности класса была бы не совсем верной, что может понизить качество классификации.

Для того чтобы оптимизировать потребление оперативной памяти, места на жестком диске, занимаемое собранной статистикой, а так же увеличить производительность программым полезно отбрасывать малозначимые данные.

1) приведение к нижнему регистру;

2) отбрасываение стоп-слов;

3) отбрасывание выражений вида `([A-Za-z]+[[:punct:]0-9]+[A-Za-z]+)`;

4) фильтрация знаков пунктуации;

5) отбрасывание слов длинее 12 символов.

Пункт 1 добавлен для приведения слов к единообразному виду и для облегчения последующей обработки.

Пункт 2 призван отбросить слова, являющиеся частью обычной человеческой речи и присущие практически любому тексту на английском языке. Это делается по той причине, что данные слова никак не могут повлиять на классификацию, но при этом занимают лишнюю память и время на их обработку, если их не отбросить.

Список стоп-слов был взят из библиотеки NLTK. Спасибо ей за это.

Пункт 3 фильтрации был добавлен для какого-никакого отсечения имен переменных, участвующих в выражениях индексации, образению к полю и т.п. Это имеет смысл т.к. имена переменных, вообще говоря, никак не зависят от языка, а лишь от программиста, который писал код и задал вопрос на StackOverflow. Имена переменных никак не помогут в классификации статьи по тегам, относящимся к конкретным языкам программирования, более того они могут "перетянуть"на себя распределение оценок вероятности от более значимых слов в статье тем самым помешав правильной классификации.

Пункт 4 добавлен для очистки слов, прошедших прошлые пункты, от знаков препинания и цифр.

Пункт 5 добавлен для отсеивания слов, получившихся из всяческих выражений языка программирования, которые прошли прошлые пункты, хотя вроде бы и не нужны. Это целесообразно потому что врядли подобное слово может встретится более одного раза в тексте.

Классификация

Теперь для классификации документа нужно всего лишь для каждого из возможных классов подсчитать выражение под оператором $argmax$ и выбрать несколько из имеющих максимальное значение.

Пороговым значением в данном случае можно выбрать значение, отличающееся от максимальной оценки вероятности (для наиболее вероятного класса) на 3%. Эксперименты показали что это пороговое значение более-менее оптимальным.

Будем считать, что наиболее вероятными классами являются те, оценка вероятности которых превышает пороговое значение (по модулю).

Исходный код

prog.cpp

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <string>
#include <unordered_map>
#include <unordered_set>
#include <vector>
#include <sstream>
#include <algorithm>
#include <set>
#include <cmath>

#include "helpers.h"

using namespace std;

typedef unordered_map<string, unsigned int> str_int_map;
typedef unordered_map<string, str_int_map>  str_mp_map ;

#define DIFF_MULT 1.030 // score of proper tag is different less than
                        // 3% from the most likely tag's score

void getArgs(int c, char **v, string &inp, string &stat, string &outp){
    for(int i = 1; i < c; ++i){
        if(!strcmp(v[i], "--input")){
            inp = v[i + 1];
        }
        else if(!strcmp(v[i], "--stats")){
            stat = v[i + 1];
        }
        else if(!strcmp(v[i], "--output")){
            outp = v[i + 1];
        }
    }
}

int main(int argc, char **argv){

    string inpFile, statFile, outpFile;
    getArgs(argc, argv, inpFile, statFile, outpFile);

    if(!strcmp(argv[1], "learn")){

        // training
        unordered_set<string> toks;      // set of tokens for unique tokens counting
        str_mp_map map;                  // main table token/class/count
```

```
str_int_map classDocsCount;      // count of documents for each class
str_int_map classToksCount;      // count of tokens for each class
unsigned int docCount = 0;       // total documents count

ifstream inp(inpFile);

cout << "Training" << endl;

while(!inp.eof()){
    // reading data

    // format:
    // <count of lines in document [n]>
    // <document title>
    // <document main part>
    // <tag 1>,<tag 2>,...,<tag m>

    stringstream ss;         // for easier retrieving tokens
                             // from text of document
    vector<string> tags;     // vector of tags for current document
    int lineCount;

    inp >> lineCount;
    inp.ignore((unsigned int)-1, '\n');

    if(inp.eof())
        break;

    string line;

    // reading of text
    for(int i = 0; i < lineCount; ++i){
        getline(inp, line);
        ss << line << endl;
    }

    // reading of tags
    getline(inp, line);

    tags = split(line, ',');

    // for each tag increment count of docs
    for(auto &el : tags){
        classDocsCount[el]++;
    }

    // filling the map with frequencies
    string tok;
    while(ss >> tok){
        if((tok = filter(tok)) != ""){
            toks.insert(tok);
            for(auto &cls : tags){
                classToksCount[cls]++;
                map[tok][cls]++;
            }
        }
    }
    docCount++;

    if(docCount % 100000 == 0){
        cout << "Processed: " << docCount << endl;
    }
}

cout << "Documents processed: " << docCount << endl;
cout << "Training completed" << endl;
```

```
    // serialization
    cout << "Serialization" << endl;
    ofstream stat(outpFile);

    if(stat){
        // count of classes
        cout << "Count of classes: " << classDocsCount.size() << endl;

        stat << classDocsCount.size() << endl;
        // total count of docs for each class
        for(auto &el : classDocsCount){
            stat << el.first << ' ' << el.second << endl;
        }
        // total count of docs
        cout << "Count of documents: " << docCount << endl;

        stat << docCount << endl;

        // total count of unique tokens
        cout << "Count of unique tokens: " << toks.size() << endl;

        stat << toks.size() << endl;

        // count of classes
        stat << classToksCount.size() << endl;

        for(auto &el : classToksCount){
            stat << el.first << ' ' << el.second << endl;
        }

        // count of tokens
        stat << map.size() << endl;

        // for each token
        for(auto &tok : map){
            stat << tok.first;
            stat << endl;

            // count of classes for that token
            stat << tok.second.size() << endl;
            // for each class
            for(auto &cls : tok.second){
                // class and its count for that token
                stat << cls.first << ' ' << cls.second << endl;
            }
        }
        cout << "Serialization completed" << endl;
    }
    else{
        cout << "Something went wrong" << endl;
        return 1;
    }
}
else{
    // deserialization
    str_mp_map map;
    unsigned int docCount = 0;
    unsigned int uniqueToksCount = 0;
    str_int_map classDocsCount;
    str_int_map classToksCount;

    vector<string> classes;

    ifstream stat(statFile);
```

```
if(stat){

    cout << "Deserelization" << endl;

    // count of classes
    unsigned int countToRead;
    stat >> countToRead;

    cout << "Count of classes: " << countToRead << endl;

    // total count of docs for each class
    classDocsCount.reserve(countToRead);
    classes.reserve(countToRead);
    for(unsigned int i = 0; i < countToRead; ++i){
        string cls;
        int count;
        stat >> cls >> count;
        classDocsCount[cls] = count;
        classes.push_back(cls);
    }
    // total count of docs
    stat >> docCount;
    cout << "Count of documents: " << docCount << endl;

    // total count of unique tokens
    stat >> uniqueToksCount;
    cout << "Count of unique tokens: " << uniqueToksCount << endl;

    // count of classes
    stat >> countToRead;
    map.reserve(countToRead);

    // count of tokens in documents for each class
    for(unsigned int i = 0; i < countToRead; ++i){
        string cls;
        int count;
        stat >> cls >> count;
        classToksCount[cls] = count;
    }

    // count of classes
    stat >> countToRead;
    map.reserve(countToRead);

    // fill the table
    for(unsigned int i = 0; i < countToRead; ++i){
        string tok;
        int clsCount;
        stat >> tok >> clsCount;
        str_int_map &ref = map[tok];
        ref.reserve(clsCount);
        for(int j = 0; j < clsCount; ++j){
            string cls;
            int count;
            stat >> cls >> count;
            ref[cls] = count;
        }
    }
    cout << "Deserelization completed" << endl;
}
else{
    cout << "Something went wrong" << endl;
    return 1;
}

ifstream inp(inpFile);
```

```
ofstream op(outpFile);

if(op && inp){
    cout << "Classification" << endl;
    int docsProcessed = 0;
    while(1){
        stringstream ss;
        vector<string> tags;
        int lineCount;
        inp >> lineCount;

        if(inp.eof())
            break;
        inp.ignore((unsigned int)-1, '\n');

        string line;

        // reading of text
        for(int i = 0; i < lineCount; ++i){
            getline(inp, line);
            ss << line << ' ';
        }

        vector<string> toks;

        string tok;

        // filter tokens
        while(ss >> tok){
            if((tok = filter(tok)) != ""){
                toks.push_back(tok);
            }
        }

        // classification
        unsigned int size = classes.size();
        vector<double> results(size, 0.0);

        // for each class compute rate

        // formula: log(D_c/D) + sum for each token of
        // log((W_ic + 1) / (|V| + L_c))

        // where:
        // D_c -- count of docs of C class in training set
        // D -- total count of docs in training set
        // W_ic -- count of i-th token occurance in documents of class C
        // |V| -- count of unique tokens
        // L_c -- total count of words of class C in training set
        // +1 -- laplace smoothing

        for(unsigned int i = 0; i < size; ++i){
            string &cls = classes[i];
            double &res = results[i];

            // log(D_c / D)
            res += log((double)classDocsCount[cls] / docCount);

            // |V| + L_c
            double denum = classToksCount[cls] + uniqueToksCount;

            // summation
            // for each token in main part
            for(auto &tok : toks){
                unsigned int count = 0;
```

```
                        auto externalIt = map.find(tok);
                        if(externalIt != map.end()){
                            auto nestedIt = externalIt->second.find(cls);
                            if(nestedIt != externalIt->second.end()){
                                count += nestedIt->second;
                            }
                        }

                        // W_ic +
                        double num = count + 1;
                        double toAdd = log(num / denum);
                        res += toAdd;
                    }
                }

                vector<pair<double, string>> pairs;
                pairs.reserve(size);

                for(unsigned int i = 0; i < size; ++i){
                    pairs.push_back(make_pair(results[i], classes[i]));
                }

                // sorting by rate in ascending order
                sort(pairs.begin(), pairs.end(), std::greater<pair<double, string>>());

                // show and write first 5 of them or which differ less than
                // 3% from maximum
                int lim = 5;

                for(int i = 0; i < lim; ++i){
                    if(pairs[i].first > pairs[0].first * DIFF_MULT){
                        if(i > 0)
                            op << ',';
                        op << pairs[i].second;
                    }
                }
                op << endl;
                docsProcessed++;
            }
            cout << "Documents proessed: " << docsProcessed << endl;
            cout << "Classification completed" << endl;
        }
    }
}
```

helpers.h

```
#ifndef HELPERS_H
#define HELPERS_H

#include <fstream>
#include <cstring>
#include <string>
#include <unordered_map>
#include <unordered_set>
#include <vector>
#include <sstream>
#include <algorithm>
#include <set>
#include <regex>

#include <iostream>
using namespace std;

// split string str to vector of strings by del delimeter
std::vector<std::string> split(const std::string &str, char del = ' '){
    size_t len = str.length();
```

```
    size_t ind = 0, pos = 0;
    std::vector<std::string> res;
    while(ind < len){
        pos = str.find(del, ind);
        if(pos == std::string::npos){
            // res.push_back(str.substr(ind, string::npos));
            // until the end of the string
            res.push_back(str.substr(ind, len - ind)); // until the end of the string
            break;
        }
        res.push_back(str.substr(ind, pos - ind));
        ind = pos + 1;
    }
    return res;
}

// check if str contains ([A-Za-z]+[[:punct:]0-9]+[A-Za-z]+) pattern
bool punctOrNumCheck(const string &str){
    int ind = 0;
    int len = str.length();
    bool res = false;
    while(ind < len && !isalpha(str[ind])){
        ind++;
    }
    ind++;
    while(ind < len && !(ispunct(str[ind]) || isdigit(str[ind]))){
        ind++;
    }
    ind++;
    while(ind < len && !isalpha(str[ind])){
        ind++;
    }
    if(ind < len){
        res = true;
    }
    return res;
}

// delete all punctuation signs and digits frome the str
void filterPunctNum(string &str){
    int ind = 0;
    int len = str.length();
    for(int i = 0; i < len; ++i){
        if(isalpha(str[i])){
            str[ind] = str[i];
            ind++;
        }
    }
    str.resize(ind);
}

// filter some unnecessary tokens
std::string filter(const std::string &str){
    // set of stop words to be filtered
    static const std::unordered_set<std::string> stops = {
    "a","about","above","after","again","against","ain","all","am","an","and","any",
    "are","aren","aren't","as","at","be","because","been","before","being","below",
    "between","both","but","by","can","couldn","couldn't","d","did","didn","didn't",
    "do","does","doesn","doesn't","doing","don","don't","down","during","each","few",
    "for","from","further","had","hadn","hadn't","has","hasn","hasn't","have",
    "haven","haven't","having","he","her","here","hers","herself","him","himself",
    "his","how","i","if","in","into","is","isn","isn't","it","it's","its","itself",
    "just","ll","m","ma","me","mightn","mightn't","more","most","mustn","mustn't",
    "my","myself","needn","needn't","no","nor","not","now","o","of","off","on",
    "once","only","or","other","our","ours","ourselves","out","over","own","re","s",
    "same","shan","shan't","she","she's","should","should've","shouldn","shouldn't",
```

"so","some","such","t","than","that","that'll","the","their","theirs","them",
"themselves","then","there","these","they","this","those","through","to","too",
"under","until","up","ve","very","was","wasn","wasn't","we","were","weren",
"weren't","what","when","where","which","while","who","whom","why","will","with",
"won","won't","wouldn","wouldn't","y","you","you'd","you'll","you're","you've",
"your","yours","yourself","yourselves","could","he'd","he'll","he's","here's",
"how's","i'd","i'll","i'm","i've","let's","ought","she'd","she'll","that's",
"there's","they'd","they'll","they're","they've","we'd","we'll","we're","we've",
"what's","when's","where's","who's","why's","would","able","abst","accordance",
"according","accordingly","across","act","actually","added","adj","affected",
"affecting","affects","afterwards","ah","almost","alone","along","already",
"also","although","always","among","amongst","announce","another","anybody",
"anyhow","anymore","anyone","anything","anyway","anyways","anywhere",
"apparently","approximately","arent","arise","around","aside","ask","asking",
"auth","available","away","awfully","b","back","became","become","becomes",
"becoming","beforehand","begin","beginning","beginnings","begins","behind",
"believe","beside","besides","beyond","biol","brief","briefly","c","ca","came",
"cannot","can't","cause","causes","certain","certainly","co","com","come",
"comes","contain","containing","contains","couldnt","date","different","done",
"downwards","due","e","ed","edu","effect","eg","eight","eighty","either","else",
"elsewhere","end","ending","enough","especially","et","etc","even","ever",
"every","everybody","everyone","everything","everywhere","ex","except","f",
"far","ff","fifth","first","five","fix","followed","following","follows",
"former","formerly","forth","found","four","furthermore","g","gave","get",
"gets","getting","give","given","gives","giving","go","goes","gone","got",
"gotten","h","happens","hardly","hed","hence","hereafter","hereby","herein",
"heres","hereupon","hes","hi","hid","hither","home","howbeit","however",
"hundred","id","ie","im","immediate","immediately","importance","important",
"inc","indeed","index","information","instead","invention","inward","itd",
"it'll","j","k","keep","keeps","kept","kg","km","know","known","knows","l",
"largely","last","lately","later","latter","latterly","least","less","lest",
"let","lets","like","liked","likely","line","little","'ll","look","looking",
"looks","ltd","made","mainly","make","makes","many","may","maybe","mean",
"means","meantime","meanwhile","merely","mg","might","million","miss","ml",
"moreover","mostly","mr","mrs","much","mug","must","n","na","name","namely",
"nay","nd","near","nearly","necessarily","necessary","need","needs","neither",
"never","nevertheless","new","next","nine","ninety","nobody","non","none",
"nonetheless","noone","normally","nos","noted","nothing","nowhere","obtain",
"obtained","obviously","often","oh","ok","okay","old","omitted","one","ones",
"onto","ord","others","otherwise","outside","overall","owing","p","page",
"pages","part","particular","particularly","past","per","perhaps","placed",
"please","plus","poorly","possible","possibly","potentially","pp",
"predominantly","present","previously","primarily","probably","promptly",
"proud","provides","put","q","que","quickly","quite","qv","r","ran","rather",
"rd","readily","really","recent","recently","ref","refs","regarding",
"regardless","regards","related","relatively","research","respectively",
"resulted","resulting","results","right","run","said","saw","say","saying",
"says","sec","section","see","seeing","seem","seemed","seeming","seems",
"seen","self","selves","sent","seven","several","shall","shed","shes","show",
"showed","shown","showns","shows","significant","significantly","similar",
"similarly","since","six","slightly","somebody","somehow","someone","somethan",
"something","sometime","sometimes","somewhat","somewhere","soon","sorry",
"specifically","specified","specify","specifying","still","stop","strongly",
"sub","substantially","successfully","sufficiently","suggest","sup","sure",
"take","taken","taking","tell","tends","th","thank","thanks","thanx","thats",
"that've","thence","thereafter","thereby","thered","therefore","therein",
"there'll","thereof","therere","theres","thereto","thereupon","there've","theyd",
"theyre","think","thou","though","thoughh","thousand","throug","throughout",
"thru","thus","til","tip","together","took","toward","towards","tried","tries",
"truly","try","trying","ts","twice","two","u","un","unfortunately","unless",
"unlike","unlikely","unto","upon","ups","us","use","used","useful","usefully",
"usefulness","uses","using","usually","v","value","various","'ve","via","viz",
"vol","vols","vs","w","want","wants","wasnt","way","wed","welcome","went",
"werent","whatever","what'll","whats","whence","whenever","whereafter","whereas",
"whereby","wherein","wheres","whereupon","wherever","whether","whim","whither",
"whod","whoever","whole","who'll","whomever","whos","whose","widely","willing",

```
    "wish","within","without","wont","words","world","wouldnt","www","x","yes",
    "yet","youd","youre","z","zero","a's","ain't","allow","allows","apart","appear",
    "appreciate","appropriate","associated","best","better","c'mon","c's","cant",
    "changes","clearly","concerning","consequently","consider","considering",
    "corresponding","course","currently","definitely","described","despite",
    "entirely","exactly","example","going","greetings","hello","help","hopefully",
    "ignored","inasmuch","indicate","indicated","indicates","inner","insofar",
    "it'd","keep","keeps","novel","presumably","reasonably","second","secondly",
    "sensible","serious","seriously","sure","t's","third","thorough","thoroughly",
    "three","well","wonder"};

    std::string res = str;

    // switch to lowercase only
    std::transform(res.begin(), res.end(), res.begin(), ::tolower);

    // switch to lowercase only
    if(punctOrNumCheck(res) || stops.find(res) != stops.end()){
        return "";
    }

    // delete all digits and punctuation if string has passed filter above
    filterPunctNum(res);

    // length filter after deleting unnecessary symbols
    if(res.empty() || res.size() > 12){
        return "";
    }

    return res;
}

#endif
```

Примеры классификации

Distribution of table in time I have a MySQL table with approximately 3000 rows per user. One of the columns is a datetime field, which is mutable, so the rows aren't in chronological order. I'd like to visualize the time distribution in a chart, so I need a number of individual datapoints. 20 datapoints would be enough. I could do this: select timefield from entries where uid = ? order by timefield; and look at every 150th row. Or I could do 20 separate queries and use 'limit 1' and 'offset'. But there must be a more efficient solution...	reference: sql,mysql prediction result: mysql,sql
27 Options for Google Maps over SSL We recently discovered that the Google Maps API does not play nicely with SSL. Fair enough, but what are some options for overcoming this that others have used effectively? >[Will the Maps API work over SSL (HTTPS)?](http://code.google.com/support/bin/answer.py?answer=65301&topic=10945) > >At this time, the Maps API is not available over a secure (SSL) connection. If you are running the Maps API on a secure site, the browser may warn the user about non-secure objects on the screen. We have considered the following options 1. Splitting the page so that credit card collection (the requirement for SSL) is not on the same page as the Google Map. 2. Switching to another map provider, such as Virtual Earth. Rumor has it that they support SSL. 3. Playing tricks with IFRAMEs. Sounds kludgy. 4. Proxying the calls to Google. Sounds like a lot of overhead. Are there other options, or does anyone have insight into the options that we have considered?	reference: google-maps,iframe,ssl,https prediction result: javascript,api,android,java,ios

Possible to "spin off"several GUI threads? (Not halting the system at Application.Run)
My Goal

I would like to have a main processing thread (non GUI), and be able to spin off GUIs in their own background threads as needed, and having my main non GUI thread keep working. Put another way, I want my main non GUI-thread to be the owner of the GUI-thread and not vice versa. I'm not sure this is even possible with Windows Forms(?)

(Sorry for the big post here. I've had complaints about previous shorter versions of this question just isn't comprehensible. I'm a lousy writer)

Background

I have a component based system in which a controller dynamically load assemblies and instantiates and run classes implementing a common _IComponent_ interface with a single method _DoStuff()_ .

Which components that gets loaded is configured via a xml configuration file and by adding new assemblies containing different implementations of _IComponent_ . The components provides utility functions to the main application. While the main program is doing it's thing, e.g. controlling a nuclear plant, the components might be performing utility tasks (in their own threads), e.g. cleaning the database, sending emails, printing funny jokes on the printer, what have you. What I would like, is to have one of these components be able to display a GUI, e.g. with status information for the said

email sending component.

...
Problem

When a component tries to fire up a GUI in _DoStuff()_ (the exact line of code is when the component runs Application.Run(theForm)) , the component and hence our system "hangs"at the Application.Run() line until the GUI is closed.
Well, the just fired up GUI works fine, as expected.

Example of components. One hasn't nothing to do with GUI, whilst the second

fires up a cute windows with pink fluffy bunnies in them.

```
public class MyComponent1: IComponent
{
public string DoStuff(...) { // write something to the database }
}

public class MyComponent2: IComponent
public void DoStuff()
{
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
Application.Run(new Form());

// I want the thread to immediately return after the GUI
// is fired up, so that my main thread can continue to work.
}
}
```

...
Is it possible to spin off a GUI and return after Application.Run()?

reference:
c#,.net,winforms

prediction result:
wpf

Из примеров видно, что классификация действительно работает. Возможно не настолько хорошо, как хотелось бы, но для этого и существуют другие более эффективные алгоритмы.

Можно заметить, что чем чаще встречается класс документа (в обучающей выборке), тем более точно предсказываются теги для документов подобных классов. (пример 1)

И наоборот, чем реже, тем хуже предсказания, которые довольно слабо согласуются с человеческим представлении о прочитанной статье. (пример 2)

Так же встречаются случаи, для которых классификатор «угадывает» смысл статьи, но точно предсказать теги оказывается не в состоянии. В таком случае на выход подаются теги, смежные с истинными. (пример 3)

Выводы

Задача классификации текста (вроде как) довольно часто встречается на практике, и поэтому важно иметь представление хотя бы о базовых алгоритмах ее решения. Одним из таких является наивный Байесовский классификатор.

Признаюсь, я до последнего не верил, что наивный Байесвоский классификатор сможет справиться с данной задачей т.к. во всех источниках в качестве приложения данного алгоритма приводится классификация всего на 2 класса (спам, не спам) простеньких текстов, не содержащих всяческих исходных кодов, формул, терминов. Это показалось каким-то несерьезным на фоне данной задачи. Но результат приятно удивил.

Данный алгоритм вполне можно использовать как отправную точку для задач классификации текста.

В решении данной задачи самым трудным оказалась правильная фильтрация данных. Т.к. в моем распоряжении был довольно слабенький компьютер с 8Гб оперативной памяти, то на этапе памяти просто не хватало (алгоритм на обучающей выборке без фильтрации потреблял порядка 10Гб) и я просто не мог обучить модель. В результате потребляюмую память удалось сократить до 6Гб, это позволило все-таки завершить обучение модели и запустить классификацию.

В результате выполнения проекта получен рабочий прототип наивного Байесовского классификатора для классификации статей с сайта StackOverflow.