Juraj Holas

Domáca úloha č. 3

**1. Vapnik-Červonenkisova dimenzia**

**a)** Skúsime ukázať, že VC dimenzia daného problému je aspoň 7. Potrebujeme teda nájsť takých 7 bodov v dvojrozmernom priestore, aby pre každé z možných označkovaní týchto bodov existoval konzistentný klasifikátor patriaci danej množine hypotéz .

Máme teraz 7 možností koľko bodov bude mať pozitívne ohodnotenie (na obrázkoch zelené body), rozoberieme si teda jednotlivé prípady:

1. Triviálny prípad – vytvoríme štvorec veľkosti okolo daného bodu
2. Vytvoríme „úzky“ obdĺžnik kopírujúci úsečku medzi bodmi a . Bude široký , a dlhý .
3. Tu môžeme odlišovať nasledujúce prípady ohodnotenia a výsledné hypotézy pre ne:

Všetky ďalšie možnosti ohodnotenia troch pozitívnych bodov vieme dostať rotáciou a zrkadlením týchto prípadov.

1. Tu môžeme odlišovať nasledujúce prípady ohodnotenia (inverzné ku predchádzajúcim) a výsledné hypotézy pre ne:

Ďalšie možnosti opäť dostaneme iba rotáciou a zrkadlením týchto prípadov.

1. Tu môžeme odlišovať nasledujúce prípady ohodnotenia a výsledné hypotézy pre ne:

Ďalšie možnosti opäť dostaneme iba rotáciou a zrkadlením týchto prípadov.

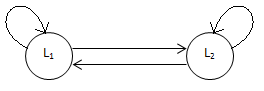
1. Tu, odhliadnuc od rotácie, môže nastať iba jediný prípad:
2. Triviálny prípad – vytvoríme dostatočne veľký štvorec, aby pokryl všetky body.

Dokázali sme teda, že VC dimenzia daného problému je aspoň 7. Keby sme chceli dokázať, že VC dimenzia je presne 7, museli by sme ukázať že nie je 8 (alebo viac). To znamená, že by sme museli dokázať, že pre *všetky* možné rozloženia ôsmich bodov v rovine vieme nájsť *aspoň jedno* také označkovanie, aby preňho neexistoval konzistentný klasifikátor z , t.j. neexistoval by obdĺžnik, ktorý by ohraničil práve všetky pozitívne príklady.

**2. Segmentácia textu**

Úlohu som implementoval v jazyku Java, všetky zdrojové kódy sú na konci dokumentu.

**a)** Stavový diagram pre HMM na segmentáciu textu:



Stavy tohto automatu predstavujú jazyky textu, emisie z každého stavu predstavujú jednotlivé znaky textu – písmená a medzeru.

Pri trénovaní potrebujeme určiť hodnoty prechodových pravdepodobností , počiatočných pravdepodobností a emisných pravdepodobností :

* → nevieme v akom jazyku text začne, preto rozložíme pravdepodobnosť rovnomerne – na polovicu
* → počty prechodov/výskytov sa samozrejme vzťahujú na značkovanie dát
* → tento pomer predstavuje vlastne frekvenciu znaku v jazyku . Počet všetkých znakov označkovaných je zároveň to isté, ako počet výskytov z minulého vzorca.

**b)** Implementáciu si môžete pozrieť v priložených zdrojových kódoch. Intuitívne, trénovanie prebieha v metóde train, ktorá počíta výskyty jednotlivých písmen v  jazykoch, celkový počet písmen v jazykoch a počty zmien jazykov. Následne predelí príslušné hodnoty a uloží prechodové pravdepodobnosti do poľa t[i][j], a emisné pravdepodobnosti do poľa e[i][x].

Obdobne, Viterbiho algoritmus implementuje metóda viterbi. V nej je použitá pomocná štruktúra reprezentujúca jedno políčko v tabuľke dyn. programovania A. V tejto štruktúre je uložená samotná hodnota daného políčka (pravdepodobnosť p), a referencia z ktorého predchádzajúceho stavu bola hodnota získaná (from).

Pri implementácií som pre prácu s veľmi malými číslami používal vlastnú triedu LogNum, ktorá číslo reprezentuje uložením hodnoty .

**c)** Úspešnosť algoritmu na testovacích dátach som vyhodnocoval ako percentuálnu zhodu so vzorovým výsledkom. Testovanie na daných štyroch príkladoch vrátilo nasledujúce výsledky:

Test sk-en-100:

Spravne urcenych 95,555% znakov

Test sk-en-1000:

Spravne urcenych 99,422% znakov

Test sk-cs-100:

Spravne urcenych 50,370% znakov

Test sk-cs-1000:

Spravne urcenych 59,278% znakov

Pri porovnávaní slovenčiny a angličtiny mal algoritmus veľmi dobré výsledky (nad moje očakávania). Dôvodom je, že tieto dva jazyky majú pomerne dosť odlišnú frekvenčnú tabuľku písmen. V slovenčine napr. s prehľadom vedie „a“ s frekvenciou cez 10%, v angličtine naopak podobnú hodnotu dosahuje jedine „e“. Navyše, niektoré znaky boli absolútne jednoznačne určiteľné – napr. „q“ sa v slovenčine nevyskytuje vôbec, preto ak naň Viterbiho algoritmus natrafil, vždy správne určil anglický jazyk. Podobne je to aj napr. s „w“.

Pri porovnávaní slovenčiny a češtiny naopak algoritmus dosahoval veľmi slabé výsledky. Pri kratšom zo vstupov úspešnosť iba nepatrne presiahla 50%, čo je priemerná úspešnosť aj náhodného generátora. Dôvodom je prílišná podobnosť frekvenčných tabuliek písmen oboch jazykov. Lepšiu úspešnosť by sme mohli docieliť v prípade, že by bola použitá diakritika („ô“ alebo „ř“ by sa správali podobne ako „q“ a „w“ v minulom prípade), prípadne tým, že by sme upravili samotný HMM na prácu s dvojicami písmen.

**Zdrojové kódy:**

TextSegment.java:

import java.io.BufferedReader;

import java.io.FileReader;

import java.text.DecimalFormat;

public class TextSegment {

public static void main(String[] args) {

TextSegment ts = new TextSegment();

System.out.println("Test sk-en-100:");

ts.train("res/sk-en-100-train.txt");

ts.test("res/sk-en-100-test.txt");

System.out.println("\nTest sk-en-1000:");

ts.train("res/sk-en-1000-train.txt");

ts.test("res/sk-en-1000-test.txt");

System.out.println("\nTest sk-cs-100:");

ts.train("res/sk-cs-100-train.txt");

ts.test("res/sk-cs-100-test.txt");

System.out.println("\nTest sk-cs-1000:");

ts.train("res/sk-cs-1000-train.txt");

ts.test("res/sk-cs-1000-test.txt");

}

char lang1, lang2;

LogNum[][] t;

LogNum[][] e;

private int cToIndex(char c){

if(c == ' '){ return 0; }

c = Character.toLowerCase(c);

return c - 'a' + 1;

}

public void train(String trainFile){

lang1 = '#';

lang2 = '#';

t = new LogNum[2][2];

e = new LogNum[2][27];

String chars = "", langs = "";

try {

BufferedReader in = new BufferedReader(new FileReader(trainFile));

chars = in.readLine();

langs = in.readLine();

in.close();

}

catch (Exception e) { e.printStackTrace(); }

double l1count = 0, l2count = 0, // Pocet pismen v jazyku 1/2

change12 = 0, change21 = 0; // Pocet zmien jazyka 1->2 / 2->1

double[] c1count = new double[27], // Pocty jednotlivych pismen v jazyku 1

c2count = new double[27]; // Pocty jednotlivych pismen v jazyku 2

lang1 = langs.charAt(0);

char lastLang = lang1;

// Pocitanie pismen, vyskytov jazyka, zmien jazyka

for(int i = 0; i < chars.length(); i++){

char c = chars.charAt(i),

l = langs.charAt(i);

if(l == lang1){

l1count++;

if(lastLang == lang2){ change21++; }

c1count[cToIndex(c)]++;

}

else{

if(lang2 == '#'){ lang2 = l; }

l2count++;

if(lastLang == lang1){ change12++; }

c2count[cToIndex(c)]++;

}

lastLang = l;

}

// Pocitanie emisnych pravdepodobnosti

for(int i = 0; i < 27; i++){

e[0][i] = new LogNum(c1count[i] / l1count);

e[1][i] = new LogNum(c2count[i] / l2count);

}

// Pocitanie prechodovych frekvencii

t[0][0] = new LogNum((l1count - change12) / l1count);

t[0][1] = new LogNum(change12 / l1count);

t[1][1] = new LogNum((l2count - change21) / l2count);

t[1][0] = new LogNum(change21 / l2count);

}

public String viterbi(String chars){

// Pomocna struktura pre dyn. prog.

class field{

LogNum p = new LogNum(); // Pravdepodobnost

int from = 0; // Predchadzajuce policko

public field(){}

public field(LogNum p, int from){

this.p = p;

this.from = from;

}

};

// Viterbiho algoritmus

field[][] A = new field[chars.length()][2];

A[0][0] = new field(LogNum.mul(0.5 , e[0][cToIndex(chars.charAt(0))]), -1);

A[0][1] = new field(LogNum.mul(0.5 , e[1][cToIndex(chars.charAt(0))]), -1);

for(int i = 1; i < chars.length(); i++){

char c = chars.charAt(i);

for(int j = 0; j < 2; j++){

LogNum

p0 = LogNum.mul(A[i-1][0].p , t[0][j] , e[j][cToIndex(c)]),

p1 = LogNum.mul(A[i-1][1].p , t[1][j] , e[j][cToIndex(c)]);

A[i][j] = new field();

if(p0.isGt(p1)){

A[i][j].p = p0;

A[i][j].from = 0;

}

else{

A[i][j].p = p1;

A[i][j].from = 1;

}

}

}

// Rekonstrukcia cesty

StringBuilder sb = new StringBuilder();

int from = 0;

if(A[A.length-1][1].p.isGt(A[A.length-1][0].p)){

from = 1;

}

for(int i = A.length-1; i >= 0; i--){

if(from == 0){

sb.append(lang1);

}else{

sb.append(lang2);

}

from = A[i][from].from;

}

return sb.reverse().toString();

}

public void test(String testFile){

String chars = "", langs = "";

try {

BufferedReader in = new BufferedReader(new FileReader(testFile));

chars = in.readLine();

langs = in.readLine();

in.close();

}

catch (Exception e) { e.printStackTrace(); }

// Spustenie Viterbiho algoritmu

String guess = viterbi(chars);

// Vyhodnotenie uspesnosti

int goodGuess = 0;

for(int i = 0; i < langs.length(); i++){

if(langs.charAt(i) == guess.charAt(i)){

goodGuess++;

}

}

System.out.println(new DecimalFormat("Spravne urcenych #0.000%

znakov").format((double)goodGuess / langs.length()));

}

}

LogNum.java:

public class LogNum {

private double logx = 0;

private boolean isZero = false;

public LogNum(double x){

if(x <= 0){

isZero = true;

}else{

logx = Math.log(x);

}

}

public LogNum(){

this(0);

}

public static LogNum mul(LogNum a, LogNum b){

LogNum result = new LogNum();

result.isZero = a.isZero || b.isZero;

result.logx = a.logx + b.logx;

return result;

}

public static LogNum mul(double a, LogNum b){

return mul(new LogNum(a), b);

}

public static LogNum mul(LogNum a, LogNum b, LogNum c){

return mul(mul(a, b), c);

}

public boolean isGt(LogNum b){

return !isZero && logx > b.logx;

}

}