Juraj Holas

Domáca úloha č. 2 – úloha 2

Úloha je implementovaná v jazyku Octave s využitím knižnice LibSMV. Zdrojové kódy k programu nájdete na konci tohto dokumentu.

**b)**

Trénovanie aj testovanie SVM som vykonal pomocou štandardných funkcií knižnice LibSVM: svmtrain a svmpredict. Úspešnosť bola hodnotená percentuálnym podielom správne klasifikovaných testovacích príkladov. Po vyskúšaní oboch typov kernelov som dostal nasledujúce výsledky:

Linear kernel:

Accuracy = 79.4788% (244/307) (classification)

Gaussian kernel:

Accuracy = 68.0782% (209/307) (classification)

**c)**

Pri škálovaní som si vypočítal posun a koeficient (pre každý atribút) z trénovacej množiny, pričom rovnakými parametrami som následne škáloval aj testovacie dáta. Po opätovnom vykonaní experimentov som dostal nasledujúce výsledky:

Linear kernel:

Accuracy = 78.8274% (242/307) (classification)

Gaussian kernel:

Accuracy = 67.101% (206/307) (classification)

Normalization & linear kernel:

Accuracy = 79.8046% (245/307) (classification)

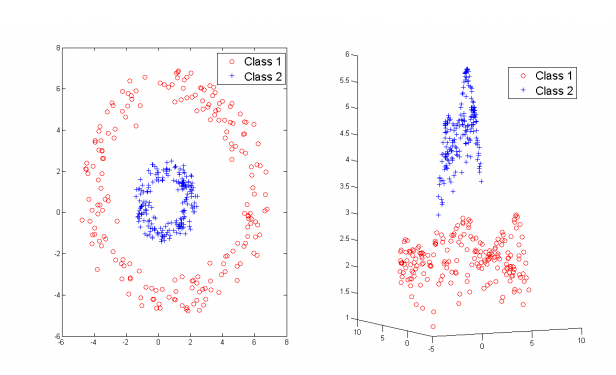
Normalization & gaussian kernel:

Accuracy = 77.5244% (238/307) (classification)

Improvement on linear kernel: 0.977199%

Improvement on gaussian kernel: 10.423453%

Zlepšenie lineárneho kernela sa pohybovalo v rádoch nanajvýš niekoľko percent, niekedy sa dokonca o niekoľko desatín aj zhoršil. Veľkosť týchto zmien bola minimálna najmä kvôli tomu, že normalizácia atribútov bola afinná transformácia (posunutie a škálovanie) vzhľadom na reprezentovaný vektorový priestor, a teda aj oddeľujúcu nadrovinu stačilo iba transformovať rovnakým spôsobom. Fakt, že výsledky nie sú úplne zhodný je daný už iba zaokrúhľovaním desatinných čísel v priebehu výpočtu.



Obrázok 1 - Ilustrácia transformácie Gausovskou funkciou (zdroj: <http://i.stack.imgur.com/7yM2K.png>)

Na druhej strane, Gausovský kernel zaznamenal pri každom testovaní vyššiu úspešnosť v priemere asi o 10%. V tomto prípade normalizácia už nepredstavuje v reprezentovanom priestore afinnú transformáciu, a tak sa výsledky líšia v podstatne väčšej miere. Posun smerom k lepšiemu zaručilo najmä škálovanie hodnôt tak, aby bola smerodajná odchýlka rovná jednej. Vyplýva to z vlastností Gausovskej funkcie, ktorá práve v okolí ±1 najrýchlejšie mení svoje hodnoty, a teda je jednoduchšie vodorovnou nadrovinou oddeliť prípady „hore“ od tých „dole“.

Ilustrované je to na obrázku č. 1, kde sa hodnoty z 2-dinenzionálneho priestoru rozmiestnili na povrch Gausovskej funkcie. Viac-menej vodorovným rezom cez stred „kopčeka“ sa teraz jednoducho oddelia jednotlivé triedy klasifikácie.

**e)**

Knižnica obsahuje v prvom rade niekoľko rôznych typov kernelov: okrem lineárneho a Gausovského, s ktorými sme pracovali pri úlohe, je možné vybrať si polynomiálny kernel alebo sigmoid, prípadne poskytnúť predpočítanú kernelovú funkciu podľa vlastného gusta.

Samotným kernelovým funkciám je následne možné nastaviť aj parametre. Vo väčšine prípadov sú však prednastavené hodnoty blízke ku ideálnym, takže nie je potrebné ich meniť. Výber z nastavovateľných parametrov:

* stupeň polynómu pri polynomiálnom kerneli
* parametre a pri polynomiálnom kerneli
* parameter pri Gausovskom kerneli

Úpravou stupňa polynomiálneho kernela a parametrov a  sa zvyšuje zložitosť polynómov a tiež aj spôsob ich rozvíjania. Vyššie polynómy vedia zväčša lepšie oddeliť dáta, avšak napr. pri zlom nastavení môžeme obmedziť rozvoj polynómov nižších stupňov, čo môže mať opäť negatívny výsledok.

Pri nastavovaní parametra zas určujeme, ako „prísny“ má byť náš Gausovský kernel. Najlepšie výsledky sa mi podarilo dosiahnuť s prednastavenou hodnotou, teda 1/počet atribútov. Pri prílišnom zväčšovaní alebo zmenšovaní tejto hodnoty bol kernel buď príliš „voľný“, čím sa následne zmenšila jeho úspešnosť, alebo naopak príliš prísne kopíroval trénovacie dáta, čím pri testovaní opäť iba znížilo úspešnosť.

**f)**

Pokiaľ prípady nie sú striktne oddeliteľné, tak metóda SVM nájde takú separujúcu nadrovinu, ktorá minimalizuje zle klasifikované prípady, t.j. odstup ich podporných vektorov od nadroviny. Snaží sa teda docieliť, aby podporné vektory mali čo najmenší „prešľap“.

|  |  |
| --- | --- |
| C:\Users\Juraj\Dokumenty\Škola\FMFI\4. ročník\Strojové učenie\svm1.png  Obrázok - vyrovnaný počet prípadov | C:\Users\Juraj\Dokumenty\Škola\FMFI\4. ročník\Strojové učenie\svm2.png  Obrázok - prevaha jednej triedy |

Pri pridávaní negatívnych prípadov získame taktiež veľa negatívnych podporných vektorov. Keby nadrovina ostala na mieste, zväčšil by sa aj počet zle klasifikovaných negatívnych prípadov, a teda aj celkový „prešľap“. Nadrovina sa teda posúva smerom ďalej od negatívnych, bližšie ku pozitívnym príkladom. Tento trend môžeme vidieť na obrázkoch 2 a 3.

Výsledok je taký, že získaný model lepšie klasifikuje negatívne prípady, avšak podstatne horšie tie pozitívne. V našom prípade to znamená, že ak pacient nemá cukrovku, tak SVM to potvrdí, ak je však diabetik, tak je väčšia pravdepodobnosť že SVM sa zmýli, a bude tvrdiť že pacient je zdravý. V krajnom prípade by SVM označil všetkých za zdravých.

Zdrojové kódy k úlohe

*Všetky zdrojové súbory sú dostupné aj online:* http://davinci.fmph.uniba.sk/~holas3/\_etc/HolasDU2.zip

Kvôli prehľadnosti kódu bol program rozdelený do dvoch súborov. Hlavný program:

source('functions.m');

% PODULOHA A

% nacitanie dat

data = csvread('pima-indians-diabetes.data.txt');

% prehodenie predpovedanej hodnoty do prveho stlpca, a zmena z {0,1} na {-1,1}

data = [((data(:, 9) .\*2) .-1), data(:, 1:8)];

% nahodny vyber trenovacej a testovacej mnoziny v pomere cca 6:4

[rows, cols] = size(data);

randOrder = randperm(rows);

splitIndex = round(rows \* 0.6);

train\_data = data(randOrder(1 : splitIndex), 2:end); % trenovacie atributy

train\_label = data(randOrder(1 : splitIndex), 1); % trenovacie vysledky

test\_data = data(randOrder(splitIndex+1 : end), 2:end); % testovacie atributy

test\_label = data(randOrder(splitIndex+1 : end), 1); % testovacie vysledky (pouzite iba na

% zistovanie uspesnosti)

% PODULOHA B

disp('Linear kernel:');

accL = train\_and\_test(train\_label, train\_data, test\_label, test\_data, 'linear');

disp('Gaussian kernel:');

accG = train\_and\_test(train\_label, train\_data, test\_label, test\_data, 'gaussian');

% PODULOHA C

% zisteneie normalizacnych parametrov

[shift, coef] = normParams(train\_data);

% normalizacia

train\_data = normalize(train\_data, shift, coef);

test\_data = normalize(test\_data, shift, coef);

% opatovne trenovanie a testovanie

disp('Normalization & linear kernel:');

accNL = train\_and\_test(train\_label, train\_data, test\_label, test\_data, 'linear');

disp('Normalization & gaussian kernel:');

accNG = train\_and\_test(train\_label, train\_data, test\_label, test\_data, 'gaussian');

% porovnanie

printf('Improvement on linear kernel: %f%%\n', (accNL(1) - accL(1)));

printf('Improvement on gaussian kernel: %f%%\n', (accNG(1) - accG(1)));

Súbor functions.m:

1;

% PODULOHA B

function accuracy = train\_and\_test(train\_label, train\_data, test\_label, test\_data, kernel)

opts = '-q'; % default kernel = gaussian

if strcmp(kernel, 'linear')

opts = '-t 0 -q';

endif

% trenovanie

model = svmtrain(train\_label, train\_data, opts);

% testovanie

[predict\_label, accuracy, dec\_values] = svmpredict(test\_label, test\_data, model);

Endfunction

% PODULOHA C

function [shift, coef] = normParams(X)

shift = []; coef = [];

[rows, cols] = size(X);

for i = (1:cols)

col = X(:, i);

% kazdy atribut sa ma posunut (zmensit) o priemer hodnot daneho atributu

shift = [shift; sum(col) / rows];

% kazdy atribut sa zoskaluje (predeli) podla st. odchylky hodnot daneho atributu

coef = [coef; std(col)];

endfor

endfunction

function result = normalize(X, shift, coef)

result = [];

[rows, cols] = size(X);

for i = (1:cols)

col = X(:, i);

col = (col .- shift(i)) ./ coef(i);

result = [result, col];

endfor

endfunction