UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

IDENTIFIKÁCIA ANOMÁLIÍ V DÁTACH

DIPLOMOVÁ PRÁCA

2020 Matúš BAGIN

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

IDENTIFIKÁCIA ANOMÁLIÍ V DÁTACH

DIPLOMOVÁ PRÁCA

Študijný program:	Ekonomicko-finančná matematika a modelovanie
Študijný odbor:	1114 Aplikovaná matematika
Školiace pracovisko:	Katedra aplikovanej matematiky a štatistiky
Vedúci práce:	doc. Mgr. Radoslav Harman, PhD.

Bratislava 2020 **Matúš BAGIN**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Matúš Bagin
Študijný program: ekonomicko-finančná matematika a modelovanie (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: aplikovaná matematika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Identifikácia anomálií v dátach
Detection of anomalies in data

Anotácia: Vektor pozorovaných čŕt objektov považujeme za anomálny, ak sa výrazne odlišuje od vektorov čŕt ostatných objektov. Anomálny vektor čŕt môže byť spôsobený nezámernou chybou (napríklad zlyhaním meracieho zariadenia), ale aj špecifickým zámerným konaním (napríklad podvodným správaním sa zákazníka, ktorému tento vektor čŕt zodpovedá). Anomálne objekty by preto mali byť podrobnejšie individuálne preskúmané. Väčšina existujúcich postupov detekcie anomálií v dátach sa zakladá na technikách strojového učenia, napríklad na analýze zhlukov a na metódach oporných bodov.

Vedúci: doc. Mgr. Radoslav Harman, PhD.
Katedra: FMFI.KAMŠ - Katedra aplikovanej matematiky a štatistiky
Vedúci katedry: prof. RNDr. Daniel Ševčovič, DrSc.

Dátum zadania: 07.01.2019

Dátum schválenia: 08.01.2019 prof. RNDr. Daniel Ševčovič, DrSc.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie Touto cestou by som sa rád poďakoval svojmu vedúcemu diplomovej práce doc. Mgr. Radoslavovi Harmanovi, Phd. za ochotu a ústretovosť pri konzultáciach, za cenné pripomienky, návrhy a odborné rady, ktoré mi veľmi pomohli pri písaní tejto práce.

Abstrakt v štátnom jazyku

BAGIN, Matúš: IDENTIFIKÁCIA ANOMÁLIÍ V DÁTACH [Diplomová práca], Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, Katedra aplikovanej matematiky a štatistiky; školiteľ: doc. Mgr. Radoslav Harman, PhD., Bratislava, 2020, 65 s.

V tejto práci sme navrhli nový algoritmus REX-DEPTH na identifikáciu a zoradenie anomálií v dátach. Základom algoritmu je nedávno publikovaný algoritmus REX, ktorý je okrem iného možné použiť na efektívnu konštrukciu elipsodiu minimálneho objemu obsahujúceho dátové body (MVEE; minimal volume enclosing ellipsoid). Algoritmus REX-DEPTH je založený na postupnom odstraňovaní vrstiev dátových bodov ležiacich na hranici MVEE. V práci ukazujeme, že navrhnutý algoritmus je výpočtovo efektívny aj pre veľké dátové súbory a spôsob, ktorým identifikuje odľahlé pozorovania, má niektoré vhodné teoretické vlastnosti, ktoré iné algoritmy na detekciu odľahlých pozorovaní nemajú.

Kľúčové slová: dátová analýza, Minimum Volume Enclosing Elipsoid (MVEE), identifikácia anomálií, návrh experimentu

Abstract

BAGIN, Matúš: Identification of anomalies in the data set [Diploma Thesis], Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics, Department of Applied Mathematics and Statistics; Supervisor: doc. Mgr. Radoslav Harman, PhD., Bratislava, 2020, 65 p.

In this work we proposed a new algorithm REX-DEPTH for identification and sorting of anomalies in data. The key component of this algorithm is the recently published algorithm REX, which, among other applications, can be used for an effective construction of the minimum volume enclosing ellipsoid (MVEE). Algorithm REX-DEPTH is based on the gradual removal of layers of data points lying on the boundary of the MVEE. In this work we demonstrate that our proposed algorithm is computationally efficient also for big data sets and and the resulting method for the oultier identification has many convenient properties that other methods for outlier detection lack.

Keywords: Data Science, Minimum Volume Enclosing Elipsoid (MVEE), identification of anomalies, design of experiment

Obsah

Úvod

Dátová veda zaznamenala v posledných rokoch následkom nástupu výkonnejších technológií a najmä oveľa výraznejšej dostupnosti dát veľký pokrok. Táto disciplína sa už aktívne využíva takmer v každej oblasti bežného života. Pomocou analýzy dát vieme zefektívniť produkciu, optimalizovať výrobu, cieliť reklamu, predpovedať počasie, odhaľovať nové choroby alebo spôsoby ich liečenia,...

Samostatnou oblasťou v rámci dátovej vedy je odhaľovanie odľahlých pozorovaní, tzv. detekcia anomálií. Odľahlé pozorovanie je pojem označujúci dátový bod výrazne sa odlišujúci od ostatných. Algoritmov ako tieto body identifikovať je množstvo, líšia sa prístupom k problému alebo aj samotným kritériom, podľa ktorého určujeme či je dátový bod už dostatočne odlišný. Detekcia anomálií má opäť v praxi širokospektrálne využitie. Vieme pomocou nej odhaľovať podvody, identifikovať a následne predchádzať hackerské útoky, pomocou výrazne odchýlených meraní identifikovať chybné meracie prístroje a veľa ďalšieho.

V našej diplomovej práci sa budeme zaoberať detekciou anomálií pomocou tzv. *Minimum volume enclosing ellipsoid* (MVEE). Ide o metódu, v ktorej sa "mrak" dát v Euklidovskom priestore obkolesí elipsoidom s čo najmenším objemom a dátové body na hranici tohto elipsoidu sú potom prehlásené za najsilnejších kandidátov na anomálie.

Tento prístup úzko súvisí s úlohou na odhad optimálneho návrhu (dizajnu) experimentu. Táto úloha bude pre nás zaujímavá najmä kvôli tomu, že na jej výpočet bol prednedávnom vypracovaný úplne nový *Random Exchange Algorithm* (REX), ktorý budeme aj my používať na numerické výpočty.

Hlavným cieľom našej práce bude jednak popísať podrobnejšie vlastnosti algoritmu REX a porovnať prístup cez MVEE s inými známymi metódami, ale aj vypracovať preň modifikáciu, resp. nadstavbu. Konkrétne by sme chceli, aby vedel REX nie len určiť, ktoré dátové body sú pre náš prípad považované za odľahlé pozorovania, ale ich aj zoradiť vzhľadom na mieru ich odľahlosti, ktorú pre potreby tejto práce budeme stručne nazývať "hĺbka".Výslednú modifikáciu algoritmu REX sme nazvali REX-DEPTH.

Naša práca sa skladá zo šiestich kapitol. Prvá časť bude zameraná na prehľad doteraz známych a najčastejšie používaných metód na detekciu odľahlých pozorovaní, predstavíme stručný popis jednotlivých metód a popíšeme ich vlastnosti. V druhej časti sa budeme

venovať našej vybranej metóde MVEE a jej duálnej úlohe hľadania optimálneho dizajnu experimentu. V tretej časti si predstavíme podrobne samotný algoritmus REX a základný princíp, na ktorom pracuje. Vo štvrtej kapitole sa zameriame na nadstavbu algoritmu REX s názvom REX-DEPTH, predstavíme jeho myšlienku, spôsob merania hĺbky dátových bodov a prehľad samotného algoritmu. V ďalšej kapitole porovnáme tento algoritmus s ostatnými známymi metódami, prehľad porovnávaných vlastností zhrnieme v záverečnej tabuľke. V poslednej šiestej kapitole aplikujeme algoritmus REX-DEPTH na dátový súbor opisujúci rôzne vlastnosti a kvalitu vín a zhodnotíme výsledky.

1 Prehľad vybraných známych metód

Vo všeobecnosti poznáme veľké množstvo rôznych metód na odhaľovanie odľahlých pozorovaní v dátových súboroch. Líšia sa najmä prístupom a tiež kritériom určujúcim odlišnosť jednotlivých dátových bodov. Poznáme napr. metódy založené na hustote bodov, kde sa hľadajú body s malým počtom iných bodov v rámci ich okolia. Ďalej poznáme metódy založené na zhlukovaní, pri ktorých sa dáta rozdelia do zhlukov a buď vzniknú priamo zhluky týchto odľahlých pozorovaní, alebo sú za ne prehlásené body nepatriace do žiadneho zhluku. Niekoľko ďalších metód už nespadá do určitej skupiny, ale sú to metódy vychádzajúce z konkrétneho algoritmu prípadne aj samostatnej disciplíny.

My sme si v tejto kapitole vybrali niekoľko známych metód, ktoré si predstavíme bližšie, popíšeme podrobne princípy na ktorých pracujú a porovnáme výhody a nevýhody týchto metód.

1.1 Priestorové zhlukovanie založené na hustote (DBScan)

Tento algoritmus bol prvýkrát predstavený v [5], pričom jeho názov vychádza zo skratky jeho anglického názvu *Density-based spatial clustering of applications with noise*. Je to algoritmus založený na hustote, čo znamená, že zoskupuje spolu body, ktoré sú dostatočne blízko pri sebe a za odľahlé označuje body s nízkou hustotou vo svojom okolí. Ide o jeden z najpoužívanejších algoritmov na zhlukovú analýzu.

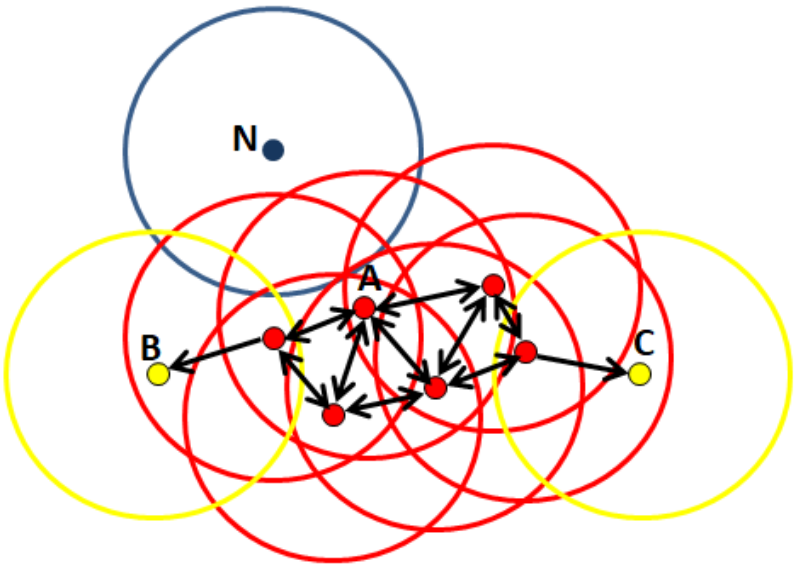
Uvažujme množinu bodov v priestore $\mathbb{R}^m$, ktoré chceme rozdeliť na zhluky. Parametrom ε označíme polomer "susedstva" vzhľadom na určitý bod. Druhým parametrom *minPts* budeme označovať minimálny počet bodov na formovanie zhluku.

DBScan klasifikuje body na jadrové, dosiahnuteľné alebo šum na základe nasledovných vlastností:

- Bod p je *jadrovým* bodom ak aspoň *minPts* bodov je v jeho ε -okolí (vrátane bodu p).
- Bod q je *priamo dosiahnuteľný* z bodu p ak bod q je v ε -okolí bodu p pričom p je jadrovým bodom.

- Bod q je *dosiahnuteľný* z bodu p, ak existuje cesta $p_1, p_2, ..., p_n$, kde $p_1 = p$ a $p_n = q$ a kde p_{i+1} je priamo dosiahnuteľný z bodu p_i.

Ak je bod p jadrovým bodom, potom tvorí zhluk so všetkými bodmi (jadrovými alebo nejadrovými), ktoré sú z neho dosiahnuteľné. Každý zhluk obsahuje aspoň jeden jadrový bod; nejadrové body môžu byť sučasťou zhluku, ale formujú len jeho "hranicu", keďže z nich už nie sú dosiahnuteľné iné body.



Obr. 1: Ukážka algoritmu DBScan

Na tejto ukážke má parameter *minPts* hodnotu 4. Červené body sú jadrové, nakoľko sú v ich ε -okolí aspoň 4 body (vrátane samotného bodu). Body B a C nie sú jadrové, ale sú dosiahnuteľné napr. z jadrového bodu A, a teda spolu formujú jeden zhluk. Bod N je odľahlý (šum), keďže nie je ani jadrový bod a ani dosiahnuteľný z iného jadrového bodu.

Dosiahnuteľnosť je nesymetrická relácia, nakoľko nejadrový bod môže byť dosiahnuteľný z nejakého jadrového, ale z neho už nie je žiadny iný bod dosiahnuteľný. Formálne preto dodefinujeme vlastnosť *prepojenosti*. Dva body p a q sú vzájomne *prepojené*, ak existuje taký bod o, že oba body p a q sú dosiahnuteľné z bodu o. Relácia prepojenosti je symetrická.

Každý zhluk potom bude spĺnať dve vlastnosti:

- Všetky body v jednom zhluku sú vzájomne prepojené.
- Ak je nejaký bod dosiahnuteľný ľubovoľným bodom zo zhluku, je takisto súčasťou daného zhluku.

1.1.1 Popis algoritmu

DBScan požaduje ako vstup dva parametre ε a *minPts*. Algoritmus začína v ľubovoľnom počiatočnom bode. Vytvorí sa ε-okolie tohto bodu a pokiaľ obsahuje dostatočný počet bodov, sformuje sa zhluk. V opačnom prípade je tento bod prehlásený za šum. Všimnime si, že tento bod môže byť neskôr v ε-okolí iného jadrového bodu a teda súčasťou zhluku.

Ak sa jadrový bod stane súčasťou zhluku, pridá sa doňho aj ε-okolie tohto bodu, rovnako ako aj ε-okolia týchto susedných bodov, pokiaľ je nejaký z nich tiež jadrový. Tento proces pokračuje, až kým je daný zhluk naplnený všetkými prepojenými bodmi. Potom sa vyberie a podobne spracuje nový počiatočný bod, (ktorý ešte nebol navštívený) čím dostaneme nový zhluk alebo šum.

DBScan môže byť použitý s ľubovoľnou normou na meranie vzdialenosti medzi bodmi (štandardne je používaná Euklidovská norma). Táto norma môže byť pridaná do algoritmu ako dodatočný parameter.

Priebeh algoritmu popíšeme v nasledovnom pseudokóde:

```
DBSCAN(DB, eps, minPts, norma) {
    C = 0                                   #pocitadlo clusterov
    for every point P in database DB {
        if label(P) is not null then break
        N = Neighbours(DB,P, eps, norma)    #najdi susedov
        if |N| < minPts then {              #kontrola hustoty
            label(P) = Sum
            break
        }
        C = C + 1
        label(P) = C
```

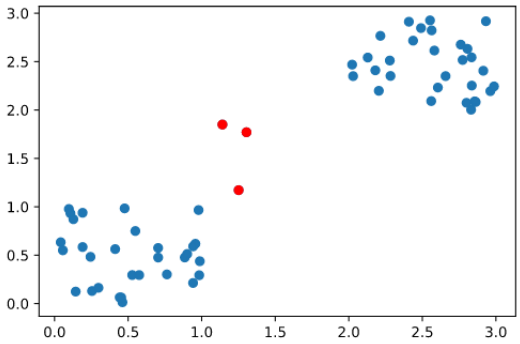
```
        S = N - {P}                                  #mnozina susedov
        for every point Q from S {
            if label(Q) = Sum then label(Q) = C
            if label(Q) is not null then break
            label(Q) = C
            N =  Neighbours(DB,Q,eps,norma) #najdi susedov
            if |N| >= minPts then{
                S = S + N                            #pridaj novych susedov
            }
        }
    }
}

Neighbours(DB, Q, eps, norma){
    neighbour = empty list
    for every point P in database DB {      #prejde vsetky body
        ak norma(Q,P) <= eps then {         #kontrola vzdialenosti
            neighbour = neighbour + {P}     #pridaj do mnoz. susedov
        }
    }
    return neighbour
}
```

1.1.2 Vlastnosti

Časová zložitosť algoritmu DBScan bola dlho považovaná ako $O(n \log(n))$. Až v [6] autori ukázali, že zatiaľ čo v 2-rozmernom priestore je naozaj rýchlosť algoritmu $O(n \log(n))$, vo väčšej dimenzii môže v najhoršom prípade algoritmus vykazovať kvadratickú zložitosť, teda $O(n^2)$. Táto zložitosť sa dá ale pri nedegenerovaných prípadoch a pri vhodnej implementácii znížiť až na lineárnu, $O(n)$.

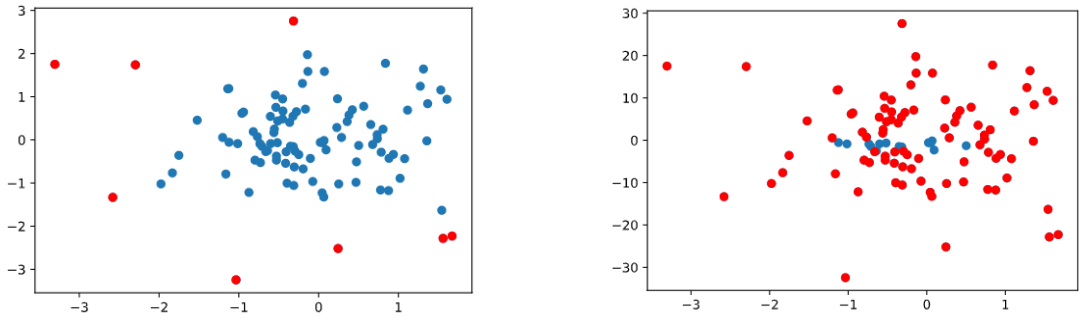
Hlavnou výhodou algoritmu DBScan je, že dokáže identifikovať anomálie aj pri zložitejšom a štrukturovanejšom rozložení dátových bodov. Vďaka tomu, že sa nesnaží všetky body nejakým spôsobom obaliť do jednej skupiny, ale naopak vie sám rozpoznať aj viacero rôzne tvarovaných zhlukov, dokáže nájsť anomálie medzi dvoma zhlukmi, dokonca aj kompletne obkolesených inými dátovými bodmi.



Obr. 2: Vizuálna 2D demonštrácia schopnosti algoritmu DBScan rozpoznať odľahlé pozorovania aj medzi zhlukmi. Tieto odľahlé pozorovania sú na obrázku vyznačené červenou farbou.

Ďalšou z výhod tohto algoritmu je možnosť zvoliť si hodnoty parametrov *minPts* a ε. Užitočné to je, najmä ak vopred poznáme charakter dát v ktorých chceme anomálie hľadať.

Táto vlastnosť je zároveň aj nevýhodou, vzhľadom na to, že tieto parametre sú po zvolení rovnaké pre celý dátový súbor. V prípade, že máme dáta, v ktorých sa výraznejšie mení hustota, táto metóda by nebola vhodná na použitie. Rovnako tak pokiaľ dáta nepoznáme, môže byť problémové vhodne zvoliť tieto parametre, keďže metóda je veľmi citlivá na ich voľbu. Táto metóda takisto nie je invariantná na zmenu jednotiek, čo si ukážeme aj na jednoduchom príklade. Na obr.3a je štandardne aplikovaná metóda, pričom parametre boli zvolené ako $minPts = n/10, \varepsilon = 1/5$. Na obrázku obr.3b je ten istý dátový set, ale druhá súradnica bodov, tj. x_2 je desaťnásobná.



(a) **(b)**

Obr. 3: DBScan nie je invariantný na zmenu jednotiek. Dáta označené ako odľahlé pozorovania sú vyznačené červenou farbou.

Môžeme pozorovať, že pri natiahnutí druhej súradnice sa súčasne aj adekvátne k tomu zmenili body vyhodnotené touto metódou ako odľahlé. Podľa očakávaní bude DBScan pri rovnakých parametroch $minPts$ a ε za odľahlé pozorovania označovať oveľa viac bodov, keďže sa natiahnutím súradnice aj výrazne zväčšili vzdialenosti medzi bodmi.

1.2 Metóda oporného bodu jednej triedy

Táto metóda je špeciálnym prípadom štandardnej *Metódy oporného bodu.* Metóda oporného bodu (skrátene SVM z anglického *Support Vector Machine*) sa používa typicky pri separácií dát na dve skupiny, pričom hranica medzi týmito skupinami je v priestore definovaná vhodne zvolenou nadrovinou. Pre lepšie porozumenie si preto najskôr predstavíme túto štandardnú metódu.

1.2.1 Metóda oporného bodu (SVM)

Metóda SVM bola prvýkrát predstavená v [4], avšak v tej dobe bola menej používaná vzhľadom na výpočtovú náročnosť. S nástupom modernej techniky nabrala SVM na popularite a začala sa využívať na rôzne aplikácie strojového učenia. Predstavíme si základný princíp tejto metódy na lineárne separovateľnom prípade.

Majme množinu X s bodmi $x_i \in \mathbb{R}^n$, $i = 1, 2, ..., N$ pričom ku každému x_i prislúcha y_i s hodnotou 1 alebo -1. Úloha separácie je nájsť takú funkciu $f : \mathbb{R}^n \to \mathbb{R}$, ktorá bude dávať kladné hodnoty pre x_i s prisluchajúcim y_i s hodnotou 1 a záporné pre x_j s prislúchajúcim y_j s hodnotou -1.

Túto funkciu zapíšeme v tvare:

$$f_\beta(x) = \beta_0 + \beta_1^T x, \tag{1}$$

pričom sa budeme snažiť vhodne nájsť parametre $\beta = (\beta_0, \beta_1^T)^T \in \mathbb{R}^{p+1}$. Túto funkciu budeme tiež nazývať aj tzv. *rozhodovacia funkcia*.

Pri lineárne separovateľnom prípade by sme vedeli túto funkciu zapísať ako:

$$\begin{aligned} f(x_i) &= \beta_0 + \beta_1^T x_i > 0 \quad \forall i : y_i = 1, \\ f(x_i) &= \beta_0 + \beta_1^T x_i < 0 \quad \forall i : y_i = -1. \end{aligned} \tag{2}$$

Nadrovinu $\{x|\beta_1^T x = -\beta_0\}$ potom nazývame oddeľujúcou nadrovinou. V prípade, že je takýchto nadrovín viac, vyriešením nasledovnej úlohy vyberieme tú, ktorá je najďalej od jednotlivých bodov oboch množín.

Celé odvodenie tejto úlohy nájdeme v [4], my si napíšeme jej finálny tvar, pričom si môžme všimnúť, že ide o úlohu kvadratického programovania:

$$\min_{\beta_0,\beta_1} \frac{1}{2}||\beta_1||^2 \tag{3}$$

$$s.t. \quad y_i(\beta_0 + \beta_1^T x_i) \geq 1, \qquad i = 1, ..., n. \tag{4}$$

V prípade, že úloha nie je lineárne separovateľná, pridá sa premenná ξ, ktorá predstavuje akúsi "penalizáciu" za prekročenie hranice oddeľujúcej nadroviny. Úloha potom dostane tvar:

$$\begin{aligned} \min_{\beta_0,\beta_1,\xi\in\mathbb{R}^n} \ & \frac{1}{2}||\beta_1||^2 + C\sum_{i=1}^{n}\xi_i \\ s.t. \quad & y_i(\beta_0 + \beta_1^T x_i) \geq 1 - \xi_i, \qquad i = 1, ..., n, \\ & \xi_i \geq 0, \qquad i = 1, ..., n, \end{aligned} \tag{5}$$

kde konštanta C predstavuje váhu, akou je pre nás dôležité, aby boli body správne rozdelené. Treba ale poznamenať, že pre vysoké C hrozí tzv. "pretrénovanosť" modelu. Pretrénovanie modelu nastáva, keď je model až príliš naviazaný na špecifiká danej trénovacej množiny, čím stráca univerzálnosť použitia na iných dátových súboroch.

Dáta nemusia byť typicky separovateľné len štandardnou nadrovinou v priestore. V prípade, že tvoria zhluky rôznych tvarov, vieme si pomôcť technikou využívajúcou tzv. *kernelové funkcie*. Idea tejto metódy je založená na zdvihnutí dát do priestoru s vyšším rozmerom, v ktorom už bude možné nájsť nadrovinu, ktorá bude naše body lepšie oddelovať. Poznáme niekoľko tvarov týchto kernelových funkcií, najčastejšie používané je *Gaussovo jadro*, ktoré má tvar:

$$K(x,y) = exp\left(-\frac{||x-y||^2}{2z^2}\right), \tag{6}$$

kde z je tzv. hyperparameter. Podrobnosti o kernelových funkciách je možné nájsť napr. v [14]

1.2.2 SVM jednej triedy

Pri klasickom SVM predpokladáme, že naše body x_i sú z dvoch rôznych množín vzhľadom na y_i ku ktorému prislúchajú. Pri One Class SVM máme body len z jednej množiny, pričom sa snažíme nájsť model reprezentujúci naše dáta, aby sme identifikovali oblasť v ktorej je veľká hustota bodov, čím zároveň aj separujeme odľahlé body, teda anomálie.

Predpokladajme, že máme body $x_i \in X$, ktoré sú generované z nejakého rozdelenia P. Funkciu $\phi : \mathbb{R}^n \to F$ zadefinujeme ako tzv. "mapovaciu" funkciu, projektujúcu naše dáta na priestor F s vyššou dimenziou. Presný predpis tejto funkcie je často neznámy, používa sa len jej implicitné vyjadrenie.

Kernelová funkcia má potom tvar:

$$K(x,x_i) = \phi(x)^T\phi(x_i). \tag{7}$$

Našou úlohou je nájsť takú podmnožinu S, že pravdepodobnosť že bod generovaný rozdelením P sa bude nachádzať mimo podmnožiny S sa rovná hodnote vopred zvoleného parametra ν. Úloha potom podľa [11] bude mať tvar:

$$\begin{aligned} \min_{\beta_1,\xi\in\mathbb{R}^n,\rho} \ & \frac{1}{2}||\beta_1||^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho \\ s.t. \quad & \beta_1^T\phi(x_i) \geq \rho - \xi_i \qquad , i = 1,...,n, \\ & \xi_i \geq 0 \qquad\quad , i = 1,...,n. \end{aligned} \tag{8}$$

Môžeme si všimnúť, že opäť ide o úlohu kvadratického programovania.

Tento tvar úlohy je veľmi závislý od parametra ν, ktorý priamo určuje, aký podiel dátových bodov bude označený za odľahlé pozorovania. Preto sa zvykne označovať nami uvedený tvar tejto metódy aj ako ν-SVM metóda.

Riešením tejto úlohy dostaneme hodnoty parametrov β_1 a ρ. Rozhodovacia funkcia by mala potom tvar:

$$f(x) = sgn(\beta_1^T \phi(x) - \rho), \tag{9}$$

teda dáva hodnotu 1 pre body x_i vnútri našej ohraničenej podmnožiny S, hodnotu 0 pre body na hranici a hodnotu -1 pre anomálie, teda body mimo podmnožiny S.

Pomocou Lagrangeovej funkcie a kernelovej funkcie vieme dostať duálnu úlohu:

$$\begin{aligned} &\min_{\alpha} \frac{1}{2} \sum_{ij} \alpha_i \alpha_j K(x_i, x_j) \\ &s.t. \quad 0 \leq \alpha_i \leq \frac{1}{\nu n}, \qquad i = 1, ..., n, \\ &\sum_i \alpha_i = 1. \end{aligned} \tag{10}$$

Riešením duálnej úlohy by sme dostali hodnoty parametra α, čím by sme dostali nový tvar rozhodovacej funkcie:

$$f(x) = sgn(\sum_i \alpha_i K(x_i, x) - \rho). \tag{11}$$

Táto metóda nám teda vytvorí separujúcu nadrovinu charakterizovanú parametrami β_1, ν, ktorá oddeľuje podmnožinu S s veľkou hustotou bodov a odľahlé pozorovania mimo nej, pričom podiel týchto anomálií je vopred definovaný parametrom ν.

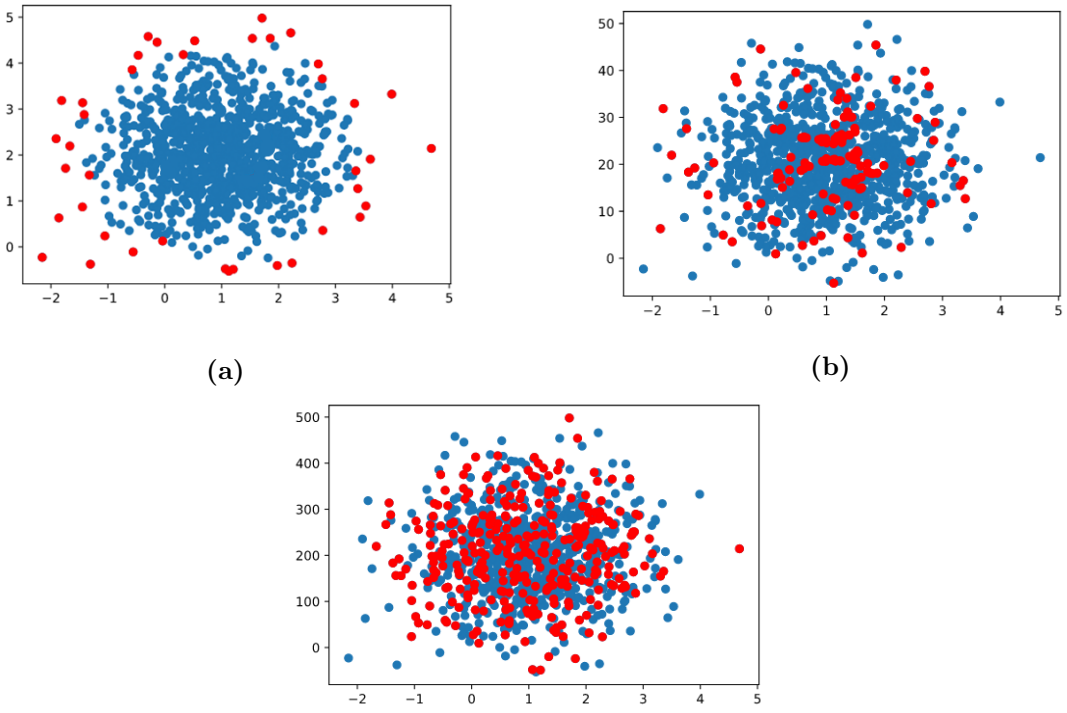
1.2.3 Vlastnosti

Veľkou výhodou metódy One Class SVM je jej rýchlosť, nakoľko je na detekciu odľahlých pozorovaní nutné vyrátať len jednu úlohu kvadratického programovania, čo s dostupnou technikou nie je časovo náročné. Táto metóda takisto veľmi dobre funguje aj vo veľkej dimenzii, čo býva častá komplikácia pri iných metódach. Za ďalšiu z výhodu tejto metódy môžeme považovať jej závislosť na parametri ν, keďže si vieme sami zvoliť, akú časť pozorovaní chceme vyhodnotiť ako odľahlé. Táto vlastnosť je ale výhodná vtedy, ak poznáme náš dátový súbor a máme voči nemu isté očakávania. Pokiaľ spracuvávame neznáme dáta, nutnosť voľby hodnoty parametra ν sa môže stať naopak nevýhodou. Táto metóda funguje na princípe lineárnej separácie, pomocou kernelových funkcií ale vieme zachytiť aj

nelineárne prípady. Dokáže preto odhaliť aj anomálie, ktoré sú vnútri zhluku, alebo v okolí rôznych neštandardných tvarov.

Nevýhodou je, už ako z názvu vychádza, že táto metóda vie zachytiť len nanajvýš jeden zhluk dát s veľkou hustotou. Preto v prípadoch, kde sa môže vyskytovať väčší počet zhlukov, je táto metóda nevhodná na použitie.

Na jednoduchom 2D príklade si ukážeme, či je metóda invariantná na zmenu jednotiek. Aplikujeme túto metódu trikrát s fixným ν na tom istom dátovom súbore, pričom ale súradnicu x_2 postupne natiahneme a budeme pozorovať zmenu vo výstupoch. Výslednú klasifikáciu odľahlých pozorovaní môžeme porovnať v nasledovných obrázkoch:



(a) **(b)**

(c)

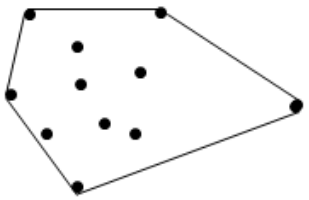
Obr. 4: One class SVM

Na obr. 4a je aplikovaná metóda One Class SVM s hodnotou parametra $\nu = 0.05$. Na obr. 4b sú použité rovnaké dáta, ale súradnica x_2 je natiahnutá 10-krát a na obr.4c je natiahnutá 100-krát. Vidíme, že výsledky sa výrazne líšia a s rastúcou disproporciou klesá schopnosť metódy správne vyhodnotiť odľahlé pozorovania.

1.3 Metóda konvexného obalu

Poslednou zo štandardných metód, ktoré si predstavíme, je metóda hľadania anomálií pomocou konvexného obalu. Hlavná myšlienka tejto metódy je založená na konštrukcii konvexného obalu okolo všetkých dát a nájdeniu tých bodov, ktoré sú na hranici tohto obalu. Vychádzať budeme z článku [2], v ktorom je predstavená nadstavba tejto metódy v podobe postupného odstraňovania jednotlivých vrstiev odľahlých pozorovaní, tzv. *onion peeling*. Táto myšlienka bola motiváciou aj pre našu nadstavbu algoritmu REX.

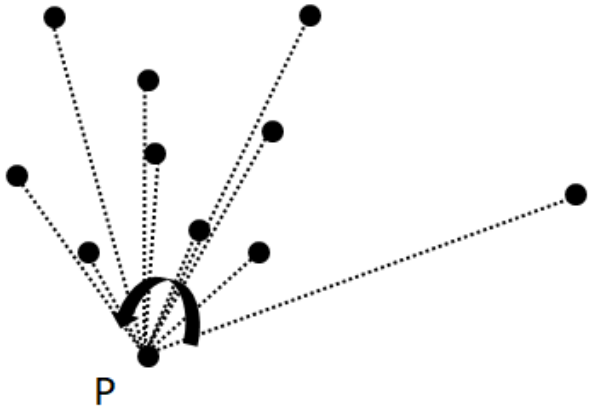
Najskôr si zadefinujeme pojem *konvexný obal*. Poznáme niekoľko rôznych definícii, môžeme ho definovať napr. tak, že je to najmenšia konvexná množina obsahujúca všetky body danej množiny.



Obr. 5: Konvexný obal množiny bodov

Skonštruovať tento konvexný obal je technicky pomerne zložité a výpočtovo náročné, najmä vo väčších dimenziách. Preto si konkrétny algoritmus na jeho konštrukciu predstavíme len na jednoduchom dvojrozmernom prípade.

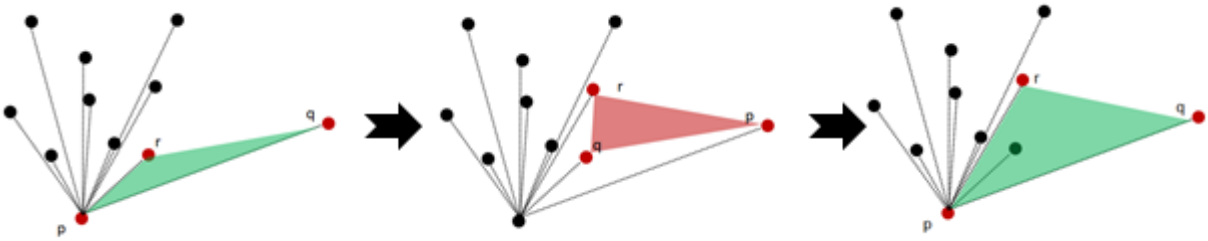
Algoritmus, ktorý použijeme, podobne ako v [2], sa nazýva *Graham scan*. Prvým krokom je najskôr si dáta vhodne zoradiť. Vyberieme si jeden bod, ktorý bude určite na hranici výsledného obalu, typicky to je bod s najnižšiou y-súradnicou. Tento bod označíme P a bude pre nás predstavovať **pivota**. Následne všetky ostatné body zoradíme v smere rastúceho uhla voči pivotovi.



Obr. 6: Grahamov scan

Následne sa aplikuje tzv. *three-penny algorithm*, ktorým sa hľadajú body tvoriace daný konvexný obal. Tento algoritmus funguje tak, že počnúc bodom P vyberie vždy tri body v danom zoradenom poradí a aplikuje na ne jedno z dvoch nasledovných pravidiel:

- Ak sú body p, q, r zoradené proti smeru hodinových ručičiek, posuň sa o jeden bod v poradí na ďalšiu trojicu.
- Ak sú body p, q, r zoradené v smere hodinových ručičiek, vyhoď bod q z polygónu a s bodom p sa posuň o jeden naspäť.



Obr. 7: Three-penny algorithm

Týmto algoritmom teda dostaneme množinu bodov, ktoré tvoria hranicu konvexného obalu našich dát. Tieto body potom budú pre nás predstavovať množinu odľahlých pozorovaní.

Táto metóda je žiaľ veľmi pomalá, najmä kvôli výpočtu konvexného obalu, ktorý je takisto aj týmto uvedeným spôsobom možný len v dvojrozmere. V práci [9] je predstavený al-

goritmus na výpočet približného konvexného obalu s časovou zložitosťou $O(K^{3/2}N^2 \log(K))$, kde N je počet bodov a K je číslo blízko počtu vrcholov vo výslednom približnom konvexnom obale. Tento algoritmus je časovo nezávislý na veľkosti dimenzie.

V neskoršej kapitole porovnáme túto metódu s našou nadstavbou REX-DEPTH a predstavíme, v čom je naša metóda rýchlejšia a výrazne efektívnejšia ako metóda uvedená v tejto podkapitole.

1.3.1 Vlastnosti

Metóda konvexného obalu je veľmi intuitívnou metódou, ktorá je v malom rozmere ľahko aplikovateľná a intepretovateľná. Ako sme už ale spomenuli, vyrátať konvexný obal vo väčšej dimenzii je časovo a výpočtovo zložité a tým sa táto metóda stáva v takýchto prípadoch neefektívnou.

Rýchlosť uvedeného algoritmu vieme popísať podrobne, pričom vychádzať budeme z [2]. Algoritmus sa skladá z dvoch častí, prvá je zoradenie bodov vzhľadom na pivota, druhou je konštrukcia samotného obalu pomocou *three-penny algorithm* resp. algoritmu "troch mincí". Na zoradenie je sa dá použiť ľubovoľný štandardný algoritmus, ktorého časová náročnosť je $O(n \log(n))$. Na algoritmus troch mincí sa pozrieme podrobne.

Vždy keď sa minca pohne vpred, posunie sa na vrchol, na ktorom minca doteraz nebola (okrem posledného posunu), takže prvé pravidlo sa aplikuje $n - 2$ krát. Vždy keď sa minca posunie vzad, jeden vrchol sa odstráni, takže druhé pravidlo sa aplikuje $n - h$ krát, kde h je počet vrcholov výsledného konvexného obalu. Keďže každý test proti smeru hodinových ručičiek zaberá konštantný čas, celý algoritmus troch mincí zaberie dokopy čas $O(n)$.

Celková časová náročnosť teda bude:

- Zoradenie zaberie $O(n \log(n))$
- Algoritmus troch mincí zaberie $O(n)$
- Celková časová naročnosť teda bude $O(n) + O(n \log(n)) + O(n) = O(n \log(n))$

Metóda je invariantná na zmenu jednotiek, čo vysvetlíme jednoduchou úvahou. Platí, že konvexný obal je prienikom všetkých konvexných množín obsahujúcich dané dáta. Tiež platí, že ak nejakú konvexnú množinu lineárne transformujeme regulárnou transformáciou

L, tak obrazom bude konvexná množina obsahujúca rovnaké dáta, ale tiež transformované transformáciou L. Zároveň prienik systému konvexných množín po transformácií transformáciou L je rovný prieniku rovnakých konvexných množín jednotlivo transformovaných transformáciou L.

Táto metóda skonštruuje jeden konvexný obal pre celý dátový súbor, očividne teda neodhalí anomálie, ktoré sú medzi dvoma zhlukmi, pokiaľ nie sú súčasťou tohto konvexného obalu. Na rozdiel od predošlých metód do tejto nevstupujú žiadne parametre, čo môže byť výhodou pokiaľ analyzujeme pre nás neznáme dáta.

1.4 Data Envelopment Analysis (DEA)

Ako príklad jednej z neštandardných metód na detekciu odľahlých pozorovaní je použitie DEA modelov. DEA je matematická disciplína zaoberajúca sa porovnavaním objektov v rámci danej skupiny a následne vyhodnocovaniu miery efektivity jednotlivých objektov. Je založená na aplikovaní vedomostí z lineárneho programovania na špeciálnych prípadoch. V praxi sa primárne používa práve na porovnávanie efektivity objektov a odhaľovaniu tých z nich, ktoré sú výrazne efektívne príp. neefektívne. Vo vybraných prípadoch sa dá táto disciplína vhodnou zmenou interpretácie použiť tiež aj ako metóda na detekciu anomálií.

Najprv si zadefinujeme niekoľko základných pojmov.

Ako už bolo spomenuté, DEA modelovanie sa zaoberá efektivitou. Táto efektivita sa viaže k vykonávaniu nejakej určitej činnosti, ktorú budeme nazývať *technológia*. Technológia T je charakterizovaná m vstupmi, ktoré predstavujú napr. rôzne materiálne alebo finančné náklady a s výstupmi, ktoré zase predstavujú napr. služby alebo vyrobené produkty.

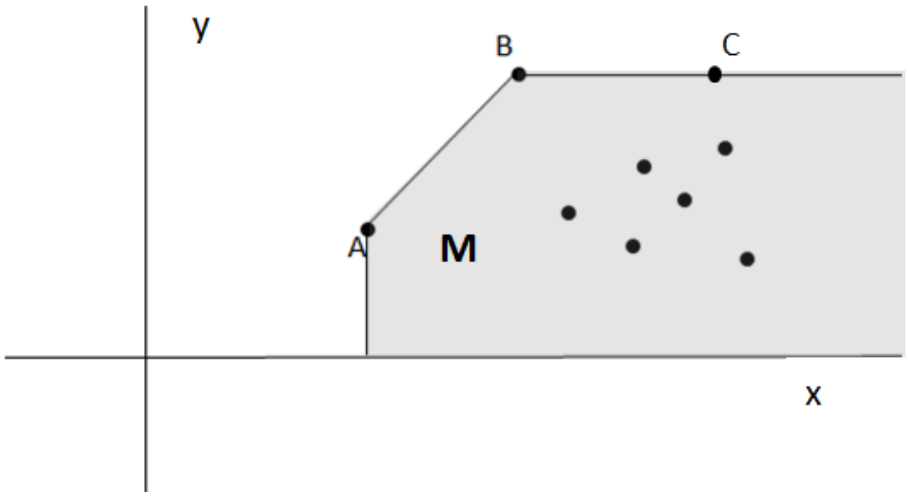
Samotné hodnoty vstupov budeme označovať vektorom $x \in \mathbb{R}^m$ a hodnoty výstupov vektorom $y \in \mathbb{R}^s$. Ak bude pre dvojicu (x, y) platiť, že zo vstupu x vieme pomocou technológie T vyprodukovať hodnotu výstupu y, tak hovoríme, že pre technológiu T je dvojica (x, y) prípustná. Množinu všetkých takýchto prípustných dvojíc budeme označovať G a volať ju *množina produkčných možností*.

Túto množinu ale nevieme presne popísať, preto sa štandardne pracuje len s jej aproximáciou M, pričom táto množina má dva základné tvary. Rozlišujeme konštantné výnosy z rozsahu, vtedy pôjde o M_{CRS} alebo variabilné výnosy z rozsahu s označením M_{VRS}. My

budeme pre naše účely pracovať len s variabilnými výnosmi z rozsahu, množina M_{VRS} má potom tvar:

$$M_{VRS} = \{(x, y) \in R^{m+s} \quad | \quad \sum_{j=1}^{n} \lambda_j x_j \leq x, \quad \sum_{j=1}^{n} \lambda_j y_j \geq y, \quad \lambda \geq 0, \quad \sum_{j=1}^{n} \lambda_j = 1\}.$$

Množina M potom môže vyzerať nasledovne:



Obr. 8: Množina produkčných možností pre variabilné výnosy z rozsahu

V terminológii DEA modelovanie rozlišujeme *efektívne* a *pseudoefektívne* útvary. Rozlišujú sa podľa toho, na akej časti hranice množiny M (označovanej ako δM) sa náchadzajú. Tieto dve časti δM si teraz zadefinujeme:

- **Hranica efektívnosti** H_E zložená z efektívnych bodov na hranici, t.j.

$$H_E := \{(x, y) \in \delta M \quad | \quad \nexists (x', y') \in M, (x', y') \neq (x, y) : x' \leq x, y' \geq y\}.$$

- **Hranica pseudoefektívnosti** H_P zložená z neefektívnych bodov na hranici, t.j.

$$H_P := \{(x, y) \in \delta M \quad | \quad \exists (x', y') \in M, (x', y') \neq (x, y) : x' \leq x, y' \geq y\}.$$

Ako môžeme pozorovať na obr. 8, body A a B sú na efektívnej hranici a teda sú to efektívne útvary, bod C je pseudoefektívnym útvarom.

Poznáme mnoho rôznych modelov a prístupov, ako tieto efektívne (pseudoefektívne) útvary nájsť. Primárne sa rozlišuje najmä, či chceme pri niektorom zo štandardných modelov použiť jeho vstupný alebo výstupný tvar. Voľba správneho modelu býva často rozhodujúca a je závislá od vlastností daného dátového súboru ako aj našich očakávaní a výrobných možností.

My si na ukážku vyberieme obálkový prístup modelu pri variabilných výnosoch z rozsahu a predstavíme jeho vstupný ako aj výstupný tvar. Celé odvodenie týchto modelov nájdeme v [7].

Vstupný model si vyberáme v prípade, keď sme spokojní s úrovňou našej produkcie (teda s výstupmi), ale chceli by sme znížiť náklady (vstupy) spotrebované na dosiahnutie tejto úrovne. Všetky útvary projektujeme na hranicu množiny M vo vstupnom smere a porovnávame ich vzdialenosť od tejto hranice.

Model má nasledovný tvar:

$$
\begin{aligned}
\text{BCC-I-OM} \quad & \min_{\theta,\lambda} \theta \\
& s.t. \quad \sum_{j=1}^{n} \lambda_j x_{ij} \leq \theta x_{io}, \qquad i = 1, ..., m, \\
& \qquad \sum_{j=1}^{n} \lambda_j y_{rj} \geq y_{ro}, \qquad r = 1, ..., s, \\
& \qquad \sum_{j=1}^{n} \lambda_j = 1, \\
& \qquad \lambda_j \geq 0, \qquad j = 1, ..., n.
\end{aligned}
$$

Hodnota parametra θ predstavuje koeficient, ktorým môžeme prenásobiť vstupy daného útvary, čím sa daný útvar sprojektuje na efektívnu (pseudoefektívnu) hranicu množiny M. Je to hodnota z intervalu $(0, 1\rangle$ a zároveň aj priamo udáva vstupnú efektivitu daného útvaru. Tie z nich, pre ktoré je $\theta = 1$ sú už na hranici množiny M, teda nevieme ich vstupy už nijako znížiť a teda pre nás predstavujú efektívne (príp. pseudoefektívne) útvary.

Výstupný model má podobný tvar. Používame ho naopak v prípadoch, keď sme spokojní so vstupmi, ktoré spotrebuvávame, ale radi by sme pri ich zachovaní zvýšili úroveň produkcie.

Výstupne orientovaný obálkový model pri variabilných výnosoch z rozsahu má tvar:

$$
\begin{aligned}
\text{BCC-O-OM} \quad & \max_{\psi,\lambda} \psi \\
& s.t. \quad \sum_{j=1}^{n} \lambda_j x_{ij} \leq x_{io}, \qquad i = 1, ..., m, \\
& \sum_{j=1}^{n} \lambda_j y_{rj} \geq \psi y_{ro}, \qquad r = 1, ..., s, \\
& \sum_{j=1}^{n} \lambda_j = 1, \\
& \lambda_j \geq 0, \qquad j = 1, ..., n.
\end{aligned}
$$

Parameter ψ teraz predstavuje maximálnu hodnotu koeficientu, ktorým by sme prenásobili výstupy útvaru tak, aby bol útvar ešte stále v množine produkčných možností. Inými slovami by sme teda pre každý útvar vedeli pri zachovaní všetkých vstupov vedieť vyprodukovať až ψ-krát väčšie množstvo. Hodnota tohto parametra je z intervalu $\langle 1, \infty)$, pričom jeho prevrátená hodnota, teda $\frac{1}{\psi}$ nám pre daný útvar určuje hodnotu jeho výstupnej efektivity.

Opäť platí, že pokiaľ pre daný útvar je parameter $\psi = 1$, produkujeme už na maximálnej možnej úrovni a preto tento útvar považujeme za efektívny (príp. pseudoefektívny).

Ako sme spomenuli, táto metóda sa v praxi používa na porovnávanie efektivity jednotlivých útvarov, na identifikáciu tých efektívnych z nich a odhaľovanie priestoru na zvýšenie efektivity či už na strane vstupov alebo výstupov.

V špeciálnych prípadoch sa DEA modelovanie dá využiť aj ako alternatívna metóda na identifikáciu anomálií. Pokiaľ analyzujeme dáta, v ktorých vieme určiť parametre predstavujúce vstupy a výstupy, anomálie by v takomto prípade pre nás mohli predstavovať práve tie útvary, ktoré by naš model identifikoval ako efektívne.

Ak by sme napríklad analyzovali efektivitu produkcie určitého množstva firiem, firmy, ktorých efektivita by vyšla výrazne vysoká, by mohli byť kandidáti na bližšiu analýzu z dôvodu podozrenia na podvod, teda tzv.*fraud detection*. Rovnako tak by sme takýmto spôsobom vedeli identifikovať novinky na trhu (*novelty detection*), keďže by mohli byť výrazne efektívnejšie ako doterajší priemer trhu.

Táto metóda je výrazne odlišná od všetkých metód, ktoré sme zatiaľ spomenuli. Nedá

sa použiť v typických úlohách na detekciu odľahlých pozorovaní, nakoľko nie je rotačne symetrická. Vieme pomocou nej odhaliť odľahlé pozorovania len v niektorých určených smeroch.

Táto vlastnosť môže byť ale naopak aj veľkou výhodou, najmä keď analyzujeme napr. vyššie spomenuté ekonomické prípady. Ak by sme takýto súbor dát analyzovali niektorou z predošlých metód, za odľahlé pozorovania by boli považované nielen firmy, ktoré pri nízkych vstupoch produkujú vysoké výstupy, ale aj iné (v tomto prípade nežiadúce) pozorovania. Mohli by to byť napríklad aj firmy, ktoré majú pri vysokých vstupoch príliš nízke výstupy, teda veľmi neefektívne firmy, ale aj jednoducho také, čo majú buď len veľmi vysoké vstupy alebo veľmi nízke výstupy (bez ohľadu na druhú premennú). V praxi by pre nás takéto výsledky analýzy nemali veľký význam.

Hlavnou výhodou metódy pomocou DEA modelovania je, že hladá netypické pozorovania len vo vybranom "podozrivom"smere. Teda napr. v tomto prípade by naozaj našla len tie firmy, ktoré majú pri nízkych vstupoch výrazne vysokú produkciu.

2 Minimum Volume Enclosing Elipsoid

2.1 Optimálny návrh experimentu

Predtým, ako prejdeme priamo k úlohe MVEE, si predstavíme úlohu hľadania optimálneho návrhu experimentu a súvis týchto dvoch úloh. Vhodný návrh zohráva pri vedeckých experimentoch kľúčovú rolu. Vďaka nemu dokážeme vykonávať pokusy, ktoré nám dodajú maximálne množstvo informácií o daných neznámych parametroch modelu.

Predpokladajme, že chceme vykonať experiment pozostávajúci z niekoľkých pokusov. Tieto pokusy budeme vykonávať na jednotlivých návrhových bodoch x patriacich množine $X = \{1, 2, 3, ..., n\}$. Pre návrhové body $x \in X$ sa predpokladá, že pozorovaná hodnota $Y(x)$ spĺňa lineárny regresný model

$$Y(x) = f(x)'\beta + \varepsilon(x), \tag{12}$$

kde $f(x) \in \mathbb{R}^m$, $m \geq 2$, je známy regresor generujúci priestor $\mathbb{R}^m$ a $\varepsilon(x) \sim (0, \sigma^2)$ je náhodný šum.

Definujme návrh na priestore X ako n-rozmerný vektor w s nezápornými zložkami so sumou 1. V praxi potom člen w_x reprezentuje podiel pokusov vykonaných v bode x.

Informačná matica je potom definovaná ako

$$M(w) := \sum_{x \in X} w_x f(x) f(x)'. \tag{13}$$

Našou snahou je nájsť také w, aby bola táto informáčná matica čo najväčšia, vzhľadom na nami zvolené kritérium optimality.

Definícia 2.1. *Nech S_+^m je množina $m \times m$ kladne semidefinitných matíc a nech Φ: $S_+^m \to \mathbb{R}$ je kritériom optimality, teda funkcia merajúca "veľkosť" informačnej matice. Hovoríme potom, že návrh w^* je Φ-optimálnym návrhom, ak maximalizuje $\Phi(M(w))$ v triede Ξ všetkých návrhov:*

$$w^* \in \operatorname{argmax} \{\Phi(M(w)) : w \in \Xi\}. \tag{14}$$

Kritérií optimality poznáme niekoľko, najpoužívanejšie z nich sú A-optimalita, kde $\Phi_A(M) = (tr(M^{-1}))^{-1}$ a D-optimalita, kde $\Phi_D(M) = (\det(M))^{1/m}$, pričom toto kritérium

budeme používať aj my. V prípade singularity matice M je kritérium A-optimality formálne definované ako 0.

Dostávame sa teda k formulovaniu úlohy D-optimálneho návrhu

$$\begin{aligned} max_{w\geq 0} \quad & \log(\Phi_D(M)) \\ subject\ to \ & \sum w_x = 1, \\ & w \geq 0. \end{aligned} \tag{15}$$

Pozn.: $\log(0) = -\infty$

Hľadanie MVEE je ako sme povedali úzko späté s úlohou hľadania D-optimálneho návrhu. Pokúsme sa k úlohe (15) odvodiť *Lagrangeovu duálnu úlohu*:

Úlohu (15) prevedieme na minimalizačnú a využitím vlastností logaritmu a determinantu dostávame:

$$\begin{aligned} min_{w\geq 0} \quad & \log(\det(M^{-1})) \\ M(w) := & \sum_{i=1}^{n} w_i v_i v_i' \\ subject\ to \ & \sum_{i=1}^{n} w_i = 1, \\ & w \geq 0. \end{aligned}$$

Zostrojme si Lagrangeovu funkciu:

$$\begin{aligned} \mathcal{L}(M, w; \mu, A) &= \log(\det(M^{-1})) + \mu(\sum_{i=1}^{n} w_i - 1) + tr(A(M - \sum_{i=1}^{n} w_i v_i v_i^T)) \\ &= \log(\det(M^{-1})) + tr(AM) + \mu \sum_{i=1}^{n} w_i - tr(A \sum_{i=1}^{n} w_i v_i v_i^T)) - \mu \\ &= (\log(\det(M^{-1})) + tr(AM)) + \sum_{i=1}^{n} w_i(\mu - v_i^T A v_i) - \mu. \end{aligned}$$

Lagrangeovu funkciu máme v separovanom tvare, teda môžeme hľadať infimum zvlášť cez každú premennú.

$$\begin{aligned} \bigtriangledown_M \mathcal{L} &= \bigtriangledown_M \log(\det(M^{-1})) + \bigtriangledown_M (AM) \\ &= -M^{-1} + A \Rightarrow A = M^{-1} \end{aligned} .$$

Nakoľko je druhý člen Lagrangeovej funkcie lineárny a w je nezáporné, tak platí nasledovná nerovnosť (pričom pre rovnosť nastáva optimalita):

$$\mu - v_i^T A v_i \geq 0, \quad i = 1, ..., n.$$

Ďalej si teda vieme napísať predpis funkcie G, ktorá do duálnej úlohy vstupuje ako účelová funkcia:

$$G(\mu, A) = inf_{M,w}\mathcal{L}(M, w; \mu, A) = \{\log(\det(A)) + n - \mu \mid \mu - v_i^T A v_i \geq 0, \quad i = 1, ..., n\}.$$

Zavedieme si transformáciu $A = \mu W$. Potom:

$$\begin{aligned}\log(\det(A)) + n - \mu &= \log(\det(\mu W)) + n - \mu \\ &= \log(\mu^n \det(W)) + n - \mu \\ &= n\log(\mu) + \log(\det(W)) + n - \mu.\end{aligned}$$

Takisto prepíšeme aj ohraničenie:

$$\begin{aligned}\mu - v_i^T A v_i \geq 0, \\ \mu - v_i^T \mu W v_i \geq 0, \\ 1 - v_i^T W v_i \geq 0.\end{aligned}$$

Celkovo teda dostávame:

$$G(\mu, W) = \{n\log(\mu) + \log(\det(W)) + n - \mu \mid 1 - v_i^T W v_i \geq 0, \quad i = 1, ..., n\}.$$

Účelová funkcia je opäť v separovanom tvare, vieme si teda znížiť počet optimalizačných premenných pomocou nájdenia optima cez premennú μ:

$$\bigtriangledown_\mu G(\mu, W) = \frac{n}{\mu} - 1 = 0 \Rightarrow n = \mu$$

Po doplnení do funkcie G dostávame konečnú formuláciu duálnej úlohy:

$$\begin{aligned}max \quad & \log(\det(W)) + n\log(n) \\ s.t. \quad & v_i^T W v_i \leq 1, \quad i = 1, ..., n.\end{aligned}$$

Môžeme pozorovať, že dostávame priamo formuláciu MVEE úlohy pre elipsoid centralizovaný v nulovom vektore, a teda tieto dve úlohy sú ekvivalentné.

2.2 Úloha MVEE

V predošlej podkapitole sme predstavili úlohu hľadania optimálneho návrhu experimentu. K tejto úlohe sme odvodili duálnu úlohu, čím sme dostali priamo predpis úlohy MVEE. V tejto podkapitole sa na ňu pozrieme bližšie a odvodíme niektoré jej vlastnosti.

Pozrime sa ešte raz na predpis tejto úlohy:

$$\begin{aligned} max \quad & \log(\det(W)) \\ s.t. \quad & v_i^T W v_i \leq 1, \quad i = 1, ..., n. \end{aligned} \tag{16}$$

Člen $n \log(n)$ z účelovej funkcie môžeme vynechať, nakoľko neovplyvní jej výsledok, len ho posunie o konštantu. Matica W pochádza zo substitúcie $W = \frac{1}{n} M^{-1}$.

Veľmi užitočnou vlastnosťou ľubovoľnej metódy na odhalovanie odľahlých pozorovaní je jej invariantnosť na zmenu jednotiek. Umožňuje nám identifikovať takéto pozorovania aj v prípadoch, keď sú jednotlivé atribúty namerané v rôznych jednotkách bez nutnosti vhodného škálovania.

Zmena jednotiek predstavuje prenásobenie vektorov s nameranými hodnotami v_i regulárnou maticou A. Pozrime sa, ako to ovplyvní maticu M:

$$M^* = \sum_{i=1}^{n} w_i A v_i v_i^T A^T = A \left(\sum_{i=1}^{n} w_i v_i v_i^T \right) A^T = A M A^T,$$

pričom hviezdičkou budeme označovať matice po transformácii.

Pozrime sa teraz na účelovú funkciu našej úlohy:

$$\begin{aligned} \log(\det(W^*)) &= \log(\det\left(\frac{1}{n} M^{*-1}\right)) = \log\left(\frac{1}{n}\right)^n \det(M^{*-1}) = \\ &= -n \log(n) - \log(\det(M^*)) \\ &= -n \log(n) - \log(\det(A M A^T)) = \\ &= -n \log(n) - \log(\det(M)) - \log(\det(A A^T)) = \\ &= \log\left(\frac{1}{n}\right)^n + \log(\det(M^{-1})) - \log(\det(A)\det(A^T) = \\ &= \log(\det\left(\frac{1}{n} M^{-1}\right)) - \log(\det(A)^2) = \\ &= \log(\det(W)) - \log(\det(A)^2) \end{aligned}$$

Môžeme pozorovať, že účelová funkcia sa nezmení, iba sa posunie o konštantu.

Ďalej sa pozrieme, ako transformácia ovplyvní podmienku v našej úlohe:

$$\begin{aligned} v_i^T A^T W^* A v_i &\leq 1, \quad i = 1, ..., n, \\ \frac{1}{n} v_i^T A^T M^{*-1} A v_i &\leq 1, \quad i = 1, ..., n, \\ \frac{1}{n} v_i^T A^T A^{-T} M^{-1} A^{-1} A v_i &\leq 1, \quad i = 1, ..., n, \\ \frac{1}{n} v_i^T I M^{-1} I v_i &\leq 1, \quad i = 1, ..., n, \\ v_i^T W^* v_i &\leq 1, \quad i = 1, ..., n. \end{aligned}$$

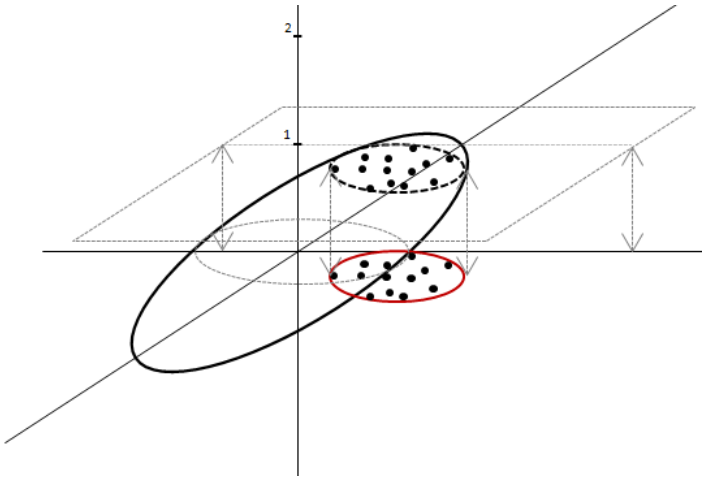
Vidíme, že podmienka v našej úlohe sa danou transformáciou vôbec nezmení.

Môžeme preto konštatovať, že úloha (16) je ekvivalentná s transformovanou úlohou po prenásobení vektorov v_i regulárnou transformačnou maticou A. Úloha MVEE je teda invariantná na zmenu jednotiek.

Doteraz sme v našej úlohe vždy uvažovali elipsoid centralizovaný v nulovom vektore. Otázkou preto je, ako by sme riešili prípad, keď tomu tak nebude.

Podrobné vysvetlenie úpravy úlohy pre tento prípad nájdeme v [13], my budeme vychádzať z [8]. Postup riešenia takejto úlohy je založený na idei, že daný mrak dát (vychýlený od nulového vektora) zdvihneme do vyššieho priestoru o jednu dimenziu, pričom hodnota každého vektora v tejto novej dimenzii bude 1. Teda vektory $\{v_1, v_2, ..., v_n\} \subset \mathbb{R}^m$ tranformujeme na $\{(v_1, 1), (v_2, 1,), ..., (v_n, 1)\} \subset \mathbb{R}^{m+1}$. V tomto väčšom priestore potom vypočítame štandardnú úlohu MVEE, čím dostaneme elipsoid E^* centralizovaný v nulovom vektore. Výsledným riešením potom bude potom elipsoid E, ktorý vznikne rezom pôvodného vypočítaného elipsoidu E^* rovinou všetkých bodov tvaru $(v, 1)$. Tento výsledný elipsoid E už len kolmo premietneme do nášho pôvodného priestoru $\mathbb{R}^m$.

Na obrázku 9 môžeme pozorovať tento postup pre $m = 2$:



Obr. 9: Spôsob výpočtu úlohy MVEE pre elipsoid bez centralizovaného stredu.

Formálne môžeme túto úlohu zapísať nasledovne:

Nech $\{v_1, v_2, ..., v_n\} \subset \mathbb{R}^m, 1 \leq m \leq n-1$ sú vektory neležiace na spoločnej nadrovine. Elipsoid s najmenším objemom, obsahujúci $\{v_1, v_2, ..., v_n\}$ je potom:

$$E(\overline{W}, \overline{v}) := \{v \in \mathbb{R}^m : (v - \overline{v})^T \overline{W} (v - \overline{v}) \leq 1\},$$

kde $\overline{v} = \sum_{i=1}^{n} w_i^* v_i$, $\overline{W} = \frac{1}{m+1} \left[\sum_{i=1}^{n} w_i^* (v_i - \overline{v})(v_i - \overline{v})^T\right]^{-1}$ a w^* je D-optimálny návrh pre lineárny regresný model s $(m+1)$ dimenzionálnymi regresormi $f(i) = (v_i^T, 1)^T, i = 1, ..., n$.

3 REX (Randomized Exchange Algorithm)

3.1 Definovanie základných pojmov

V tejto kapitole sa podrobnejšie pozrieme na algoritmus REX, ktorým vieme numericky vyrátať optimálny dizajn experimentu, pričom budeme vychádzať z [8]. Predtým ako prejdeme k samotnému algoritmu si uvedieme niekoľko pojmov.

Nosič dizajnu w je definovaný ako $supp(w) = \{x \in X : w_x > 0\}$. Tento nosič má pre optimálny dizajn tendenciu byť pomerne riedky, výrazne menší ako n. Dokonca, vychádzajúc z *Proposition III.8* v [10], poznáme tvrdenie, že vždy existuje optimálny dizajn, ktorého nosič nemá veľkosť väčšiu ako $1 + m(m+1)/2$.

Za *regulárny dizajn* budeme považovať také dizajny, pre ktoré bude informačná matica $M(w)$ nesingulárna. Množinu všetkých regulárnych dizajnov budeme potom označovať Ξ_R a budeme ju zatiaľ považovať za fixnú .

Vektor disperzie dizajnu w budeme označovať $g(w)$ a jeho zložky sú počítané ako

$$g_x(w) = f'(x)M^{-1}(w)f(x), \quad x \in X.$$

Tento vektor disperzie sa dá takisto vyrátať aj ako

$$g_x(w) = \lim_{\alpha \to 0_+} \frac{\Psi_D\left[(1-\alpha)M(w) + \alpha M(e_x)\right] - \Psi_D(M(w))}{\alpha} + m,$$

kde $\Psi_D(M) = log(det(M))$ je verzia D-optimálneho kritéria a e_x je singulárny dizajn v bode x (t.j. x-ová zložka normalizovaného jednotkového vektora). Vychádzajúc z tohto zápisu nám potom hodnota $g_x(w)$ indikuje, aká je lokálka miera zlepšenia, ak by sme preniesli infinitezimálnu váhu návrhu (pomer počtu pozorovaní) do bodu x a z ostatných bodov túto váhu ubrali. Teda inými slovami pre nás vektor $g(w)$ predstavuje akýsi vektor "pravdepodobnosti" jednotlivých zložiek dizajnu, že budú aj vo výslednom optimálnom dizajne. Ďalším využitím vektora $g(w)$ je, že ho hodnotu $max_x g_x(w)$ budeme neskôr používať ako kritérium na zastavenie algoritmu.

Efektivitu dizajnu w budeme merať vždy vzhľadom na iný regulárny dizajn $\overline{w}$ ($\overline{w} \in \Xi_R$), konkrétne

$$\text{eff}(w|\overline{w}) := \frac{\Phi_D(M(w))}{\Phi_D(M(\overline{w}))}.$$

Táto efektivita môže byť interpretovaná ako pomer vykonaných pokusov potrebných na dosiahnutie požadovanej hodnoty daného kritéria. Nech napr. $\text{eff}(w|\overline{w}) = 0.85$, to zna-

mená, že na dosiahnutie rovnakého množstva informácií by nám stačilo len 85% pokusov použitím dizajnu $\overline{w}$ v porovnaní s počtom pokusov, ktoré by bolo treba pri použití dizajnu w. Nech $w \in \Xi_R$, potom ak w^* je optimálny dizajn, tak podľa [12] platí

$$\text{eff}(w|w^*) \geq \frac{m}{max_{x \in X} g_x(w)}.$$

Pokiaľ je teda w náš aktuálny dizajn a jeho efektivita vzhľadom na efektívny dizajn je približne rovná 1, nie je potrebné už ďalej pokračovať vo výpočte.

3.2 Prehľad algoritmu

Nech je vektor w regulárny dizajn a nech $g(w)$ je n-rozmerný vektor disperzie jednotlivých zložiek dizajnu w. Hodnota $g_x(w)$ potom ako sme už spomenuli odhaduje pravdepodobnosť, že bod x bude v nosiči optimálneho dizajnu.

Základná myšlienka algoritmu REX je opakovane vyberať skupinu párov dizajnových bodov a vykonať medzi nimi optimálnu výmenu váh. Počet týchto párov nie je fixne určený a môže sa líšiť medzi jednotlivými iteráciami. Výber týchto bodov závisí od vektora $g(w)$.

Algoritmus by sa dal schématicky popísať podobne ako v [8] v nasledovných krokoch:

1. **LBE krok.** Pri danom aktuálnom dizajne w vyrátame $g(w)$ a následne vykonáme *leading Böhning exchange* (LBE) ako:

$$w \leftarrow w + \alpha^*_{k,l}(w)(e_l - e_k), \tag{17}$$

kde

$$\begin{aligned} k &\in \text{argmin}\left\{g_u(w) :\ u \in \text{supp}(w)\right\}, \\ l &\in \text{argmax}\left\{g_u(w) :\ u \in X\right\}, \\ \alpha^*_{k,l}(w) &\in \text{argmax}\{\Phi(M(w + \alpha(e_l - e_k))) \ :\ \alpha \in [-w_l, w_k]\} \end{aligned}$$

Optimálny krok $\alpha^*_{k,l}(w)$ sa nazýva *nulujúci* ak je rovný $-w_l$ alebo w_k.

Vidíme, že hľadanie optimálneho kroku $\alpha^*_{k,l}(w)$ predstavuje pre každú iteráciu samostatný optimalizačný problém. Veľkou výhodou zabezpečujúcou vysokú rýchlosť tohto algoritmu je, že pre kritérium D-optimality vieme túto hodnotu vyrátať explicitne. Presný vzorec nájdeme v [8].

2. **Výber z podpriestoru.** Podpriestor S z X na ktorom bude metóda pracovať je definovaný ako zjednotenie dvoch podmnožín. Jedna z nich je vytvorená takzvaným "pažravým" procesom (S_{greedy}) a druhá je identická s nosičom aktuálneho dizajnu w (S_{supp}).

   - **Greedy.** Položíme $L = min(\lceil \gamma m \rceil, n)$, a vyberieme body

   $$S_{greedy} = \{l_1^*, ..., l_L^*\} \subseteq X,$$

   kde l_i^* prislúcha k i-tej najväčšej zložke vektora $g(w)$.

   - **Nosič.** Položíme

   $$S_{supp} = supp(w).$$

   Nech K je veľkosť nosiča supp(w): $K = |supp(w)|$.

   - **Aktívny podpriestor.** Aktívnu množinu S pre body dizajnu potom definujeme ako

   $$S = S_{greedy} \cup S_{supp}.$$

   V jednotlivých iteráciach potom už pracujeme výhradne len s týmito bodmi dizajnu. Všimnime si, že ak by $\gamma m \geq n$ alebo ak by $K = n$, tak S by bola celá množina X, teda všetky body dizajnu. Typicky ale dochádza k situácii, kedy $|S| \ll n$ kedy metóda pracuje na výrazne menšom podpriestore ako je celé X.

3. **Výmeny váh v rámci podpriestoru.** Ďalším krokom je špecifická výmena váh medzi pármi v rámci podpriestoru S. Dizajnové body w_v také, kde $v \notin S$ ostanú bez zmeny.

   - **Vytvorenie párov.** Nech $(k_1, ..., k_K)$ je rovnomerná náhodná permutácia S_{supp} a nech $(l_1, ..., l_L)$ je rovnomerná náhodná permutácia S_{greedy}. Vytvoríme postupnosť

   $$(k_1, l_1), (k_2, l_1), ..., (k_K, l_1), ..., (k_1, l_L), (k_2, l_L), ..., (k_K, l_L) \tag{18}$$

   tvorenú $K \times L$ pármi z aktívnych dizajnových bodov.

   - **Výmeny.** Ak LBE krok nebol nulujúci v danej iterácii, postupne vykonáme všetky Φ-optimálne výmeny medzi pármi bodov z 18 a následne dopočítame w a $M(w)$.

Ak bol LBE krok nulujúci, postupne vykonáme všetky *nulujúce* Φ-optimálne výmeny medzi pármi bodov z 18 a následne dopočítame w a $M(w)$.

4. **Pravidlo zastavenia.** Po dopočítaní vektora $g(w)$ skontrolujeme, či je hodnota $max_x g_x(w)$ dostatočne blízko 1 a ak áno, algoritmus zastavíme. Pokiaľ nie, bude nasledovať ďalšia iterácia počínajúc krokom 1.

4 REX-DEPTH ako modifikácia algoritmu REX

4.1 Motivácia a základná myšlienka

V predošlej kapitole sme podrobne opísali, ako pomocou algoritmu REX identifikovať jednotlivé body predstavujúce šum. V praxi je často podstatnejšie tieto body nielen odhaliť, ale vedieť ich aj zoradiť vzhľadom na to, ako veľmi sú odlišné od zvyšku dátového setu.

Nech dátový set predstavuje napr. poistné udalosti a odľahlé pozorovania sú podozrivé udalosti resp. potencionálne podvody. Špeciálny tím poisťovne, kontrolujúci tieto podozrivé udalosti, nemá kapacitu prešetriť jednotlivo každú z týchto udalostí. Oveľa užitočnejšie by bolo, ak by mal k dispozícii zoznam týchto udalostí ale zoradený postupne od najviac podozrivej (teda najodľahlejšej) a už vzhľadom na svoje možnosti prešetrí vybraný počet.

V kapitole 1 sme sa pri metóde konvexného obalu stretli s pojmom *onion peeling*. Ide o postupné odstraňovanie jednotlivých vrstiev odľahlých pozorovaní z dátového súboru. V danej metóde to bolo aplikované tak, že sme vyrátali daný konvexný obal a pre každý bod z hranice tohto obalu sme vypočítali jeho naškálovanú Euklidovskú vzdialenosť od centra. Táto hodnota pre nás v tomto prípade predstavovala už spomenutú *hĺbku*.

Presne táto myšlienka postupného odstraňovania dátových bodov vzhľadom na ich hĺbku nás motivovala k modifikácii algoritmu REX.

Majme dátový súbor, na ktorom raz aplikujeme algoritmus REX. Výstupom bude vektor návrhu w s nosičom dlžky p ako aj vektor g, z ktorého vieme určiť body na hranici MVEE. Tieto body na hranici elipsoidu môžeme pripodobniť k bodom na hranici konvexného obalu pri predošlej metóde, čiže rovnako tak sú to kandidáti na odľahlé pozorovania a budeme ich bližšie analyzovať.

Pojem *excentricita* si zadefinujeme ako akúsi vzdialenosť daného bodu, resp. jeho "odľahlosť" od zvyšku dátového súboru. Čím má bod menšiu hĺbku, teda nachádza sa ďalej od centra, tým má väčšiu excentricitu.

Našou snahou bude teraz zoradiť tieto body podľa nami zvolenej excentricity. Zadefinujeme ju ako veľkosť zmeny objemu MVEE po odstránení daného bodu z hranice elipsoidu (pozn.: táto hodnota je nezáporná). Postupne odstránime vždy jeden bod z hranice a zaznamenáme si jeho prislúchajúcu excentricitu. Bod, ktorý bude mať túto

hodnotu najväčšiu, teda po jeho odstránení sa objem MVEE zmenšil najviac, budeme potom považovať za najodľahlejší.

Zaznamenáme si ho do vektora odľahlých pozorovaní, odstránime z nosiča, vhodne preusporiadame váhy na zvyšných bodoch nosiča (tak aby $\sum w_i = 1$) a celý postup zopakujeme. Takýmto spôsobom budeme postupne "rozbaľovať" vektor w až po nami požadovanú úroveň.

4.2 Prehľad algoritmu

Predtým ako prejdeme ku samotnému modifikovanému algoritmu sa zamyslime ako ho spraviť čo najúspornejšie. Bolo by neefektívne, ak by sme po každom vyhodení jedného bodu spúštali pôvodný algoritmus REX na náhodnom počiatočnom návrhu. Miesto toho by sme chceli využiť informáciu z predchádzajúcej iterácie, keďže už optimálny návrh poznáme.

Naprogramujeme teda nový algoritmus REX_repeat, ktorý bude fungovať veľmi podobne ako pôvodný REX, len ako vstup dostane $n-1$ rozmerný vektor návrhu a $(n-1)\times m$ rozmernú maticu F. Nebudeme musieť preto pri opakovanom spúštaní algoritmu vždy začínať na náhodnom vektore návrhu, ale využitím predošlej informácie a očakávania, že nový návrh sa nebude veľmi líšiť od predošlého, dostaneme výsledok už po výrazne nižšom počte iterácií.

Algoritmus bude teda vyzerať nasledovne:

1. **Iniciálne spustenie.** Spustíme prvýkrát základný algoritmus REX, pričom ako vstup je $n \times m$ matica F a náhodný počiatočný vektor návrhu w. Ako výstup dostaneme optimálny návrh w^*, ktorého nosič má rozmer p, a aktuálny objem MVEE V.

2. **Odstránenie bodu návrhu.** Následne budeme pre každý bod z hranice MVEE aplikovať nasledovný postup. Dočasne odstránime daný bod z vektora návrhu a aj z matice F. Vektor w^* bude mať teraz rozmer len $n-1$ ale hlavne suma jeho členov bude menšia ako 1. Váhu, ktorá bola na tomto návrhovom bode preto vhodne preusporiadame. Možností je niekoľko, my sme zvolili takú, že celú váhu daného bodu premiestnime na bod návrhu, ktorý nebol v pôvodnom nosiči, ale má spomedzi

ostatných bodov najvyššiu hodnotu $g(x)$. Tým vytvoríme nový vektor w'.

3. **Nájdenie najodľahlejšieho bodu.** Tento nový vektor w' spolu s upravenou maticou F' zvolíme ako vstup do algoritmu REX_ repeat. Ako výstup dostaneme opäť nový optimálny návrh (pozn.: predpokladáme, že bude podobný tomu vo vstupe), ale najmä hodnotu zmenšeného MVEE, ktorú označíme V'. Tento postup zopakujeme pre všetky body na hranici pôvodného elipsoidu, pričom si vždy zapamätáme hodnotu V'. Za najodľahlejší bod budeme potom považovať ten, pre ktorý bude hodnota V' najmenšia.

4. **Odstraňovanie vrstiev.** Tento najodľahlejší bod si zaznamenáme do výsledného vektora odľahlých pozorovaní a natrvalo ho odstránime z dátového setu. Iterujeme znova od kroku 1, pričom už ale nespustíme základný REX s náhodným počiatočným návrhom, ale REX_ repeat s návrhom prislúchajúcim k danému odstránenému bodu z kroku 2. Iterujeme až po nami zvolenú úroveň.

Veľkou výhodou aplikovania myšlienky postupného odstraňovania vrstiev dát v algoritme REX-DEPTH je najmä výrazne vyššia rýchlosť celého algoritmu. Zatiaľ čo v metóde konvexného obalu bolo treba v každej iterácii rátať konvexný obal mnohorozmernej množiny bodov čo je vypočtovo zložité, v našej modifikácii bude stačiť zrátať samotný elipsoid čo je pomocou upraveného algoritmu REX_ repeat oveľa efektívnejšie. Rovnako tak počítanie excentricity jednotlivých bodov ako zmena objemu elipsoidu je výpočtovo nenáročná.

Presné vyjadrenie časovej náročnosti algoritmu REX-DEPTH nepoznáme, nakoľko nie je známa ani časová náročnosť samotného algoritmu REX. Celú modifikáciu sme sa ale snažili robiť čo najviac efektívne. Zakomponovanie pridruženého algoritmu REX_ repeat nám umožňuje vo výpočte využívať informáciu o predošlom optimálnom návrhu. Takisto voľba spôsobu preusporiadania návrhu po odstránení jedného jeho bodu bola zvolená za účelom dosiahnutia čo najvyššej výpočtovej efektivity.

5 Porovnanie metód

V tejto kapitole spravíme záverečné porovnanie vybraných metód spomenutých v kapitole 1 aj spolu s našim algoritmom REX-DEPTH. Porovnáme a zosumarizujeme niektoré vhodné vlastnosti týchto metód, a každej z nich spravíme a popíšeme jednoduchú nadstavbu, ktorá pridá metóde schopnosť zoraďovať odľahlé pozorovania podľa excentricity. Následne aplikujeme metódy na dátových súboroch z rôznych rozdelení a porovnáme výsledky.

V závere kapitoly zosumarizujeme všetky poznatky o týchto metódach v prehľadnej tabuľke. Porovnáme jednak vlastnosti už popísané v kapitole 1, ako aj tie, ktoré ešte popíšeme v tejto kapitole.

Skúsme najskôr navrhnúť jednotlivé nadstavby pre metódy spomenuté v kapitole 1. Nadstavbu chceme spraviť tak, aby sme si mohli vopred zvoliť, koľko odľahlých pozorovaní chceme identifikovať a tiež, aby boli vo výsledku zoradené vzhľadom na ich excentricitu.

V metóde **DBScan** používame dva parametre a to *minPts* a ε. My budeme v našej jednoduchej nadstavbe hýbať iba parametrom ε a to tak, že ho budeme postupne zmenšovať. Pri veľkej hodnote ε odhalíme len pozorovania, ktoré sú výrazne odľahlé, pri zväčšujúcej sa hodnote tohto parametra bude ich počet narastať. Teda čím dlhšie necháme algoritmus bežať a čím viac pozorovaní sa budeme snažiť odhaliť, tým menšie bude ε a teda tým budú ďalšie odhalené pozorovania menej odľahlé.

Je zjavné, že pri fixnom parametri *minPts* bude pre epsilony $\varepsilon_1 < \varepsilon_2$ a množiny odľahlých pozorovaní $S(\varepsilon_1)$ a $S(\varepsilon_2)$ platiť: $S(\varepsilon_1) \supseteq S(\varepsilon_2)$. Ak pre každý bod v množine $S(\varepsilon_2)$ platí, že v jeho ε_2-okolí nie je žiaden iný bod, tak určite nebude žiaden iný bod ani v jeho zmenšenom ε_1-okolí, teda bude patriť aj do množiny $S(\varepsilon_1)$.

Presnú myšlienku algoritmu popíšeme v nasledovnom pseudokóde:

```
DBSCAN_modify(DB, minPts ,N){
    vystup = zeros(N)        #vysledny vektor zoradenych anomalii
    T = 0                    #pocet doteraz odhalenych anomalii
    it = 1                   #cislo iteracie
    M = max(DB)-min(DB)      #rozsah DB
    while T<N:
        anomalie = DBSCAN(DB, eps = M/it , minPts , norma)
```

```
#najdeme vektor anomalii pri aktualnom eps a fixnom minPts
for every points in anomalie:
    if bod not in vystup:
        #ak sa dany bod este nenachadza vo vektore vystupu
        vystup[T] = bod      #pridaj bod do vektora vystupu
        if T > N-1:
            break        #skonci, po danom pocte anomalii
it +=1    }
```

Pre metódu **One Class SVM** žiaľ takáto nadstavba nebude možná, nakoľko metóda nedokáže odhaliť len veľmi malý počet odľahlých pozorovaní (pri veľmi nízkom parametri ν), čo je nevyhnutné pre správne zoraďovanie. Takisto nemáme pri tejto metóde zaručené, že pri rastúcom ν bude nová množina odľahlých pozorovaní nadmnožinou tej predošlej.

Preto nadstavbu tejto metódy vynecháme a nebudeme ju ani v tejto kapitole porovnávať s ostatnými metódami.

Poslednou metódou, pre ktorú spravíme nadstavbu je **metóda konvexného obalu** . Pri tejto metóde nemáme žiaden vstupný parameter, preto bude princíp mierne odlišný ako pri metóde DBScan.

Na začiatku algoritmu vyrátame priemer dát, ktorý nazveme *centrum* a bude nezmenený počas celého výpočtu. V prvej iterácii vyrátame konvexný obal, a pre každý bod z tohto obalu zrátame naškálovanú Euklidovskú vzdialenosť od centra, teda jeho relatívnu vzdialenosť vzhľadom na rozptyl bodov v danom smere. Bod s najväčšou vzdialenosťou pridáme do výsledného vektora anomálií a odstránime z našich dát. Znova vyrátame nový konvexný obal a tento proces opakujeme až po dosiahnutie požadovaného počtu bodov vo vektore anomálií.

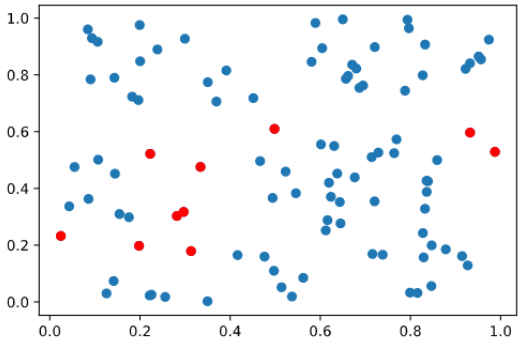
Princíp metódy je popísaný aj nasledovne:

```
ConvexHull_modify(DB,N){
    anomalie = zeros(N)
    centrum = average(DB)
    for k in (1:N):
        obal = ConvexHull(DB)          #vyratame kvx obal DB
        i=0
```

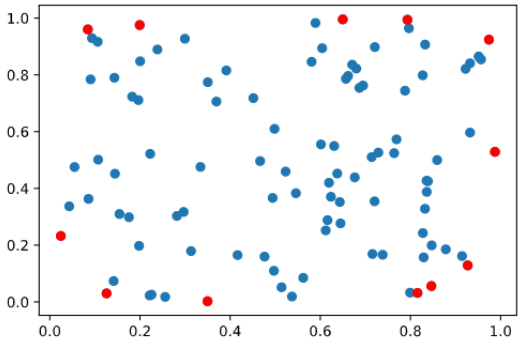
```
for p in obal:
    dist[i] = distance(p,centrum)    #vzdial. od centra
    i += 1
anomalie[k] = DB[max(dist)]      #bod s najvacsou vzdial.
delete point from DB}
```

Pozrieme sa teda podrobnejšie na samotné porovnanie. Budeme porovnávať metódy DBScan, metódu konvexného obalu a REX-DEPTH. Metódy budeme postupne aplikovať na rôzne súbory 2D dát a graficky porovnáme výsledky.

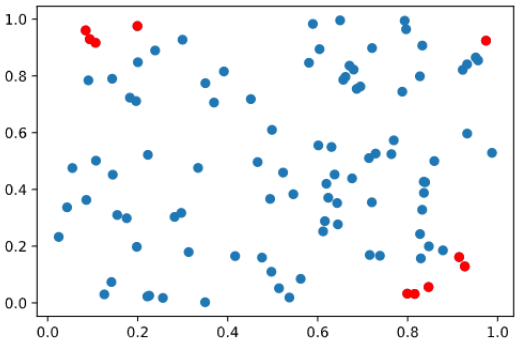
V prvom kroku sa pozrieme na základnú funkcionalitu metód pri ich aplikácii na rovnomerne rozdelených 100 bodoch na intervale $(0, 1) \times (0, 1)$. Odľahlé pozorovania budeme vždy označovať červenou farbou.



Obr. 10: DBScan, $minPts = 3, \varepsilon = 0.1$.



Obr. 11: Metóda konvexného obalu.



Obr. 12: REX-DEPTH s počtom odľ. pozorovaní $N = 10$.

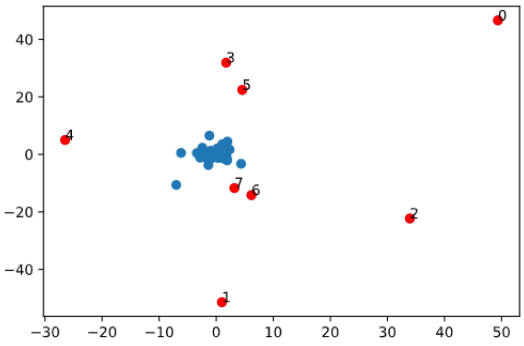
Všímame si rozdiel medzi metódami, DBScan odhalí pozorovania na miestach s nízkou hustotou, metóda konvexného obalu a REX-DEPTH naopak nájdu pozorovania na okraji dátového súboru. V každom prípade, aplikácia metód na takéto rovnomerne rozdelené

dáta nemá veľký význam, nakoľko sa v takomto súbore skutočne odľahlé pozorovania ani nenachádzajú.

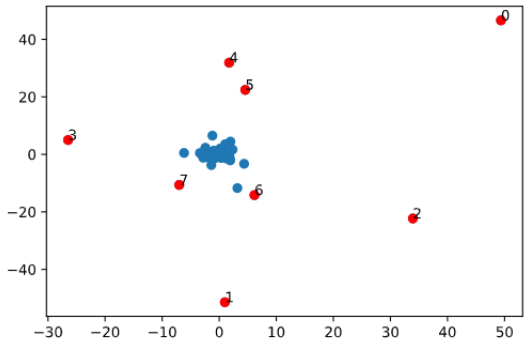
V druhom kroku preto porovnáme všetky tri metódy pri ich aplikácii na 100 bodoch zo samostatne navrhnutého rozdelenia. Body sú generované nasledovne:

$$x_i = r_i cos(\varphi_i), \quad i = 1, ..., n,$$
$$y_i = r_i sin(\varphi_i), \quad i = 1, ..., n,$$

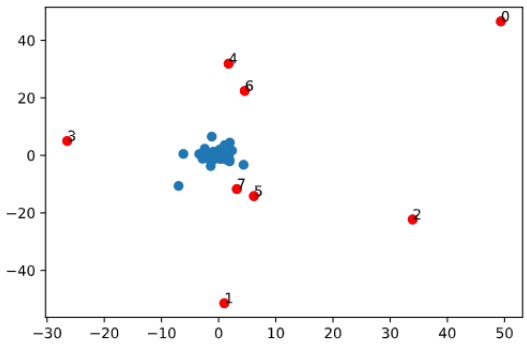
kde $r_i \sim Cauchy(0, 1)$ a $\varphi_i \sim R(0, 2\pi)$. V tomto dátovom súbore sú už výraznejšie odľahlé pozorovania, preto použijeme nadstavby metód a porovnáme, ako zoradia tieto pozorovania jednotlivé metódy. Vopred si zvolíme fixný počet, koľko odľahlých pozorovaní chceme odhaliť $(N = 8)$.



Obr. 13: Nadstavba metódy DBScan aplikovaná na dátach zo štandardného Cauchyho rozdelenia



Obr. 14: Nadstavba metódy konvexného obalu aplikovaná na dátach zo štandardného Cauchyho rozdelenia.

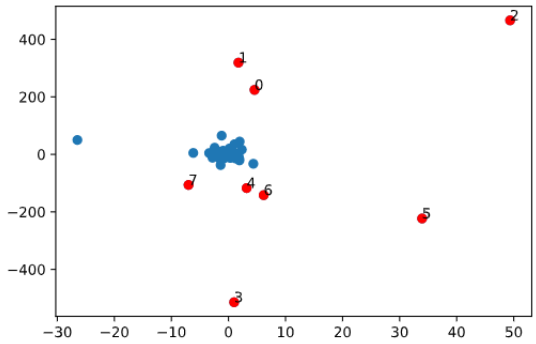


Obr. 15: REX-DEPTH aplikovaný na dátach zo štandardného Cauchyho rozdelenia.

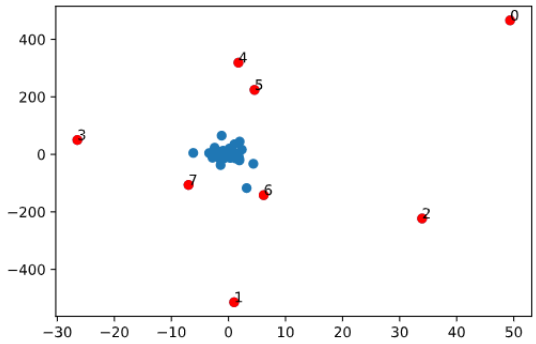
Môžeme si na obrázkoch všimnúť, že každá z metód zoradila odľahlé pozorovania iným spôsobom. Nedá sa objektívne rozhodnúť, ktoré zoradenie je správne, vždy to závisí najmä

od charakteru spracovávaných dát, prípadne aj o našej hlbšej vedomosti o nich. Môžeme ale konštatovať, že každá z metód nám vrátila rozumné výsledky.

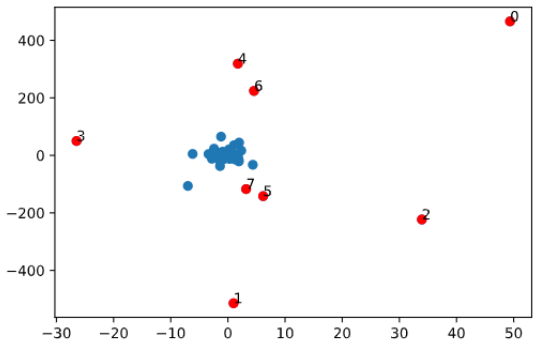
Zaujímavejšie to bude, ked rovnaké dáta lineárne transformujeme, a to opäť tak, že druhu súradnicu natiahneme 10-krát. Overíme si, ktoré z metód sú invariantné na zmenu jednotiek, a ako to ovplyvní výsledné zoradenie.



Obr. 16: Nadstavba metódy DBScan aplikovaná na transformovaných dátach.



Obr. 17: Nadstavba metódy konvexného obalu aplikovaná na transformovaných dátach.



Obr. 18: REX-DEPTH aplikovaný na transformovaných dátach.

Na tejto ukážke sa nám potvrdilo, čo sme popísali v kapitole 1 a 2.2. Metóda DBScan nie je invariantná na zmenu jednotiek, výsledne zoradenie sa líši od predošlého, zatiaľ čo metóda konvexného obalu a REX-DEPTH dávajú rovnaké výsledky aj po transformácii.

Všetky doterajšie poznatky o daných metódach zosumarizujeme v nasledovnej tabuľke:

	DBScan	OC SVM	Convex Hull	REX-DEPTH
Časová zložitosť	$O(nlog(n))$	$O(n^3)$	$O(nlog(n))*$	neznáma
Efektívne aj vo veľkej dimenzií	áno ($O(n^2)$)	áno	nie	áno
Odhalenie poz. medzi zhlukmi	áno	nie	nie	nie
Invariantnosť na zmenu jednotiek	nie	nie	áno	áno
Nezávislosť na vstupnom parametri	nie	nie	áno	áno

Tabuľka 1: Porovnanie vlastností metód.

**takúto časovú zložitosť má metóda len v dvojrozmernom prípade*

Z tabuľky môžeme konštatovať nasledovné závery.

Metóda DBScan je pomerne rýchla aj vo vyššej dimenzii a ako jediná z rozoberaných metód vie odhaliť odľahlé pozorovania aj medzi zhlukmi. Nevýhodou je ale, že dáta musia byť normované a metóda je veľmi závislá na voľbe vstupných parametrov.

Metóda oporného bodu jednej triedy (One Class SVM) je o niečo pomalšia, najmä kvôli nutnosti riešenia úlohy kvadratického programovania. Výhodou ale je, že časovú náročnosť výpočtu neovplyvňuje počet atribútov, teda metóda je rovnako efektívna aj vo vysokej dimenzii. Nie je ale invariantná na zmenu jednotiek a takisto sú jej výsledky závislé na vstupnom parametri.

Ďalšia v poradí, metóda konvexného obalu, je veľmi intuitívna a v dvojrozmere aj rýchla. Je invariantná na zmenu jednotiek a nie je nutné jej zadávať vstupné parametre. Jej najväčšou nevýhodou je veľmi náročná aplikácia vo vysokej dimenzii, nakoľko zrátať konvexný obal množine bodov vo vysokej dimenzii je výpočtovo zložité.

Pozrime sa na našu metódu REX-DEPTH. Jej jedinou nevýhodou v porovnaní s ostatnými je, že sa nám aplikovaním tejto metódy nepodarí odhaliť odľahlé pozorovania, ktoré sa nachádzajú medzi zhlukmi. V prípade takýchto dát by sme odporučili použiť metódu DBScan. V ostatných vlastnostiach ale REX-DEPTH ostatné metódy prevyšuje. Je veľmi rýchly a efektívny aj vo vysokej dimenzii a nie je závislý na vstupných parametroch, čo je výhoda najmä pri spracovávaní pre nás menej známych dátových súboroch.

Táto metóda je takisto invariantná aj na zmenu jednotiek a ako sme vysvetlili v kapitole 2.2, vie si poradiť aj s posunom. Je teda invariantná na ľubovoľnu afinnú transformáciu.

V kombinácii s nízkou časovou efektivitou vieme preto algoritmus REX-DEPTH odporučiť ako veľmi praktickú a efektívnu metódu vhodnú na odhalovanie odľahlých pozorovaní aj v rozsiahlych dátových súboroch.

6 Aplikácia

V záverečnej kapitole našej práce aplikujeme algoritmus REX-DEPTH na konkrétnom dátovom súbore. Graficky a slovne popíšeme všetky výsledky a vyvedieme z nich závery analýzy.

Dáta, ktoré budeme analyzovať, pochadzájú z *UCI Machine Learning Repository* a ide o rozbor fyzikálno-chemických vlastností $n = 6497$ rôznych portugalských vín "Vinho Verde" a vplyv týchto vlastností na kvalitu daných vín.

Dátový súbor obsahuje nasledovné stĺpce:

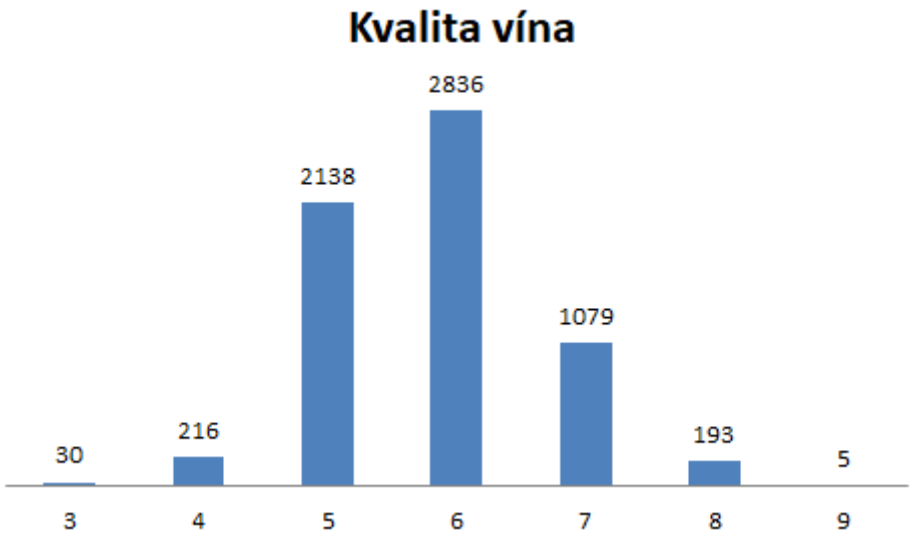
- **Celková kyslosť.** Tvoria ju organické kyseliny, ako kyselina vínna, jablčná a mliečna. $[0 - 15, 9]$
- **Prchavé kyseliny.** Vyjadrené vo víne ako kyselina octová. Vplýva na arómu vína. $[0 - 1, 9]$
- **Kyselina citrónová.** Hladina kyseliny citrónovej, jedno z jej využití je na dodanie sviežosti. $[0 - 1, 1]$
- **Zvyškový cukor.** Hladina neprekvaseného cukru, výška jeho koncentrácie určuje *sladkosť* alebo *suchosť* daného vína. $[0 - 65, 8]$
- **Chloridy.** Množstvo chloridov vo víne. $[0 - 0, 611]$
- **Voľný oxid siričitý**. Množstvo voľného SO_2 používaného najmä na ochranu vína pred oxidáciou. $[1 - 289]$
- **Celkový oxid siričitý.** Celkové množstvo SO_2, pri jeho vyššej hodnote sa zvyšuje stabilita voľného SO_2. $[6 - 440]$
- **Hustota.** Ovplyvňujú ju všetky zložky vína. Sacharidy a kyseliny ju zvyšujú, alkohol a ine prchavé látky hustotu znižujú. $[0, 99 - 1, 04]$
- **pH.** Číslo udáva pH-hodnotu vína, víno s nízkym pH chutí kyslejšie a sviežejšie, vysoké pH podporuje bakteriálny rast vo víne. $[0 - 4, 01]$
- **Sírany.** Množstvo síranov vo víne. $[0 - 2]$

- **Alkohol.** Percentuálne množstvo alkoholu vo víne. $[1,1 - 14,9]$
- **Kvalita.** Znalecky určená kvalita daného vína vzhľadom na chuť, vôňu, konzistenciu, ... Čísla sú z rozsahu $[3 - 9]$, pričom hodnotenie 3 je pre najmenej kvalitné víno.

Pozrime sa aj na ukážku prvých 10 riadkov našich dát:

FK	PK	$C_6H_8O_7$	ZC	Cl	V SO_2	C SO_2	hustota	pH	SO_4^2	Alc.	kvalita
7,7	0,41	0,76	1,8	0,611	8	45	0,9968	3,6	1,1	9,4	5
9,2	0,52	1	3,4	0,61	32	69	0,9996	1,2	2	9,4	4
7,8	0,41	0,68	1,7	0,467	18	69	0,9973	3,8	1,1	9,3	5
7,8	0,43	0,7	1,9	0,464	22	67	0,9974	1,3	1,1	9,4	5
8,6	0,49	0,51	2	0,422	16	62	0,9979	3,3	1,1	9	5
7,8	0,48	0,68	1,7	0,415	14	32	0,99656	3,9	1,6	9,1	6
8,7	0,78	0,51	1,7	0,415	12	66	0,99623	3	1,1	9,2	5
8,7	0,78	0,51	1,7	0,415	12	66	0,99623	3	1,1	9,2	5
8,5	0,46	0,59	1,4	0,414	16	45	0,99702	3,3	1,1	9,2	5
9,1	0,76	0,68	1,7	0,414	18	64	0,99652	2,9	1,1	9,1	6

Takisto sa ešte pred začiatkom analýzy pozrime na histogram vín z hľadiska kvality:



Obr. 19: Histogram vín vzhľadom na ich ohodnotenú kvalitu.

Cieľom našej analýzy bude pomocou uvedených vlastností odhaliť vína, ktoré sú v nejakom zmysle odľahlé voči "štandardu" a vyhodnotiť, či táto odľahlosť nejako vplýva aj na ich celkovú kvalitu. Dočasne preto odstránime stĺpec určujúci kvalitu vína a na zvyšných dátach aplikujeme algoritmus REX-DEPTH ako metódu na detekciu odľahlých pozorovaní bez učiteľa. Po identifikovaní nami vopred zvoleného počtu pozorovaní priradíme týmto vínam ich skutočnú kvalitu a zhodnotíme výsledky analýzy.

Zvoľme si počet odľahlých pozorovaní, ktoré chceme odhaliť $N = 10$. Na dátach teraz spustíme algoritmus REX-DEPTH, pričom ako vstup mu dáme maticu s dátami o vínach a počiatočný náhodný vektor návrhu.

$$REX\_DEPTH(data = vino, pociatocny\_navrh = random)$$

Pozrime sa na jednotlivé výstupy nášho algoritmu:

- Vektor indexov dátových bodov *ind_out* dlžky N vyhodnotených ako odľahlé, zoradené na základe ich excentricity.
- Vektor *timer* dĺžky N, obsahujúci trvanie (v sekundách) každej iterácie, teda kroky 1-4. z kapitoly 4.2.
- Vektor *body_na_hranici* dĺžky N, ktorý v každej iterácii udáva aktuálny počet bodov na hranici MVEE. Toto číslo zároveň udáva, koľkokrát budeme v danej iterácii spúšťať algoritmus REX_repeat.

K samotným bodom sa potom pomocou vektora *ind_out* dostaneme jednoducho zavolaním príkazu *vino[ind_out]*, čím dostaneme len identifikované odľahlé dátové body aj v poradí, v akom ich zoradil náš algoritmus.

Všimnime si, že ak by sme prvky vektora *timer* predelili prvakmi vektora *body_na_hranici*, dostali by sme priemerný čas, za ktorý zbehne odstránenie jedného bodu z hranice MVEE a vyrátanie nového (typicky zmenšeného) elipsoidu na zvyšných bodoch. Táto hodnota sa pohybuje približne v rozmedzí $1.5 - 3$ sekundy.

Všetky tieto tri vektory zobrazíme v nasledovnej tabuľke, pričom pridáme aj prislúchajúcu hodnotu kvality daného vína:

ind_out	timer	body_na_hranici	kvalita
2781	66.9	34	6
4745	76.6	28	3
5049	59.7	32	4
5156	83.9	31	5
5097	68.9	35	4
5018	80.3	32	5
4886	77.2	31	7
1417	73.7	27	3
2127	63.3	29	5
3152	77.5	32	6

Radi by sme si naše výsledky aj graficky vizualizovali. Keďže pracujeme s 11-rozmernými dátami, na grafické znázornenie v 2D použijeme techniku *Metódy hlavných komponentov* (Principal Component Analysis, skr. PCA).

Ide o metódu primárne využívanú práve na takúto grafickú vizualizáciu, pri ktorej hľadáme takú množinu lineárnych kombinácií pôvodných premenných (pozorovaní), ktorá zachováva čo najväčšie množstvo informácií o pôvodných premenných (pozorovaniach) a zároveň jej dimenzia bude menšia. Viac o tejto metóde nájdeme napr. v [3].

My by sme radi znázornili naše výsledky tak, aby sme mohli pozorovať, že pozorovania identifikované našou metódou sú naozaj odľahlé. Pokiaľ by sme len štandardne aplikovali PCA, dostali by sme síce prvé dva hlavné komponenty najlepšie opisujúce celý dátový súbor, ale po vyznačení identifikovaných N pozorovaní by sme nemali zaručené, že po ich projektovaní do danej nadroviny definovanej týmito prvými dvoma komponentami by bola takáto vizualizácia prínosná.

V tomto prípade by bolo vhodnejšie, ak by sme aplikovali PCA len na daných N pozorovaniach, a do nadroviny definovanej prvými dvoma hlavnými komponentami premietneme celý dátový súbor.

Touto technikou budeme mať zaručené, že naše vybrané pozorovania sú v tomto roz-

mere naozaj odľahlé. Inými slovami, budeme sa na naše dáta pozerať "v tom správnom smere".

Pozrime sa na samotný výpočet. Matica X rozmeru $p \times N$ označuje dáta pre odľahlé pozorovania (kde $p = 11$), matica $\overline{X}$ rozmeru $p \times n$ označuje celý dátový súbor. Podobne vektory μ a $\overline{\mu}$ označujú príslušné vektory stredných hodnôt.. Výpočet bude potom prebiehať nasledovne:

$\Sigma = cov(X)$, kovariančná matica pre X

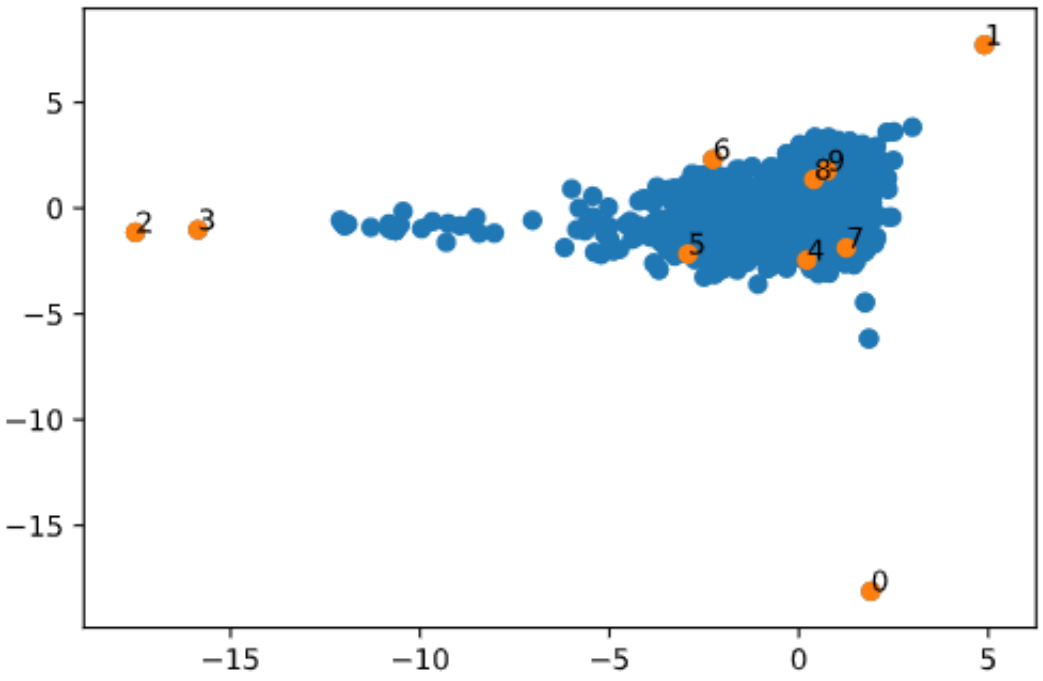
$\Sigma = U\Lambda U^T$ Schurov rozklad kovariančnej matice

$Y = U^T(X - \mu)$ matica hlavných komponentov pre odľahlé pozorovania

$\overline{Y} = U^T(\overline{X} - \overline{\mu})$ matica hlavných komponentov pre všetky body

$\overline{y_1}, \overline{y_2}$ prvé dva hlavné komponenty

Tieto body si teraz znázorníme v 2D rovine, pričom farebne vyznačíme N bodov identifikovaných algoritmom REX-DEPTH. Vieme, že algoritmus tieto body aj zoradí, priradíme preto k nim aj prislúchajúce čísla:



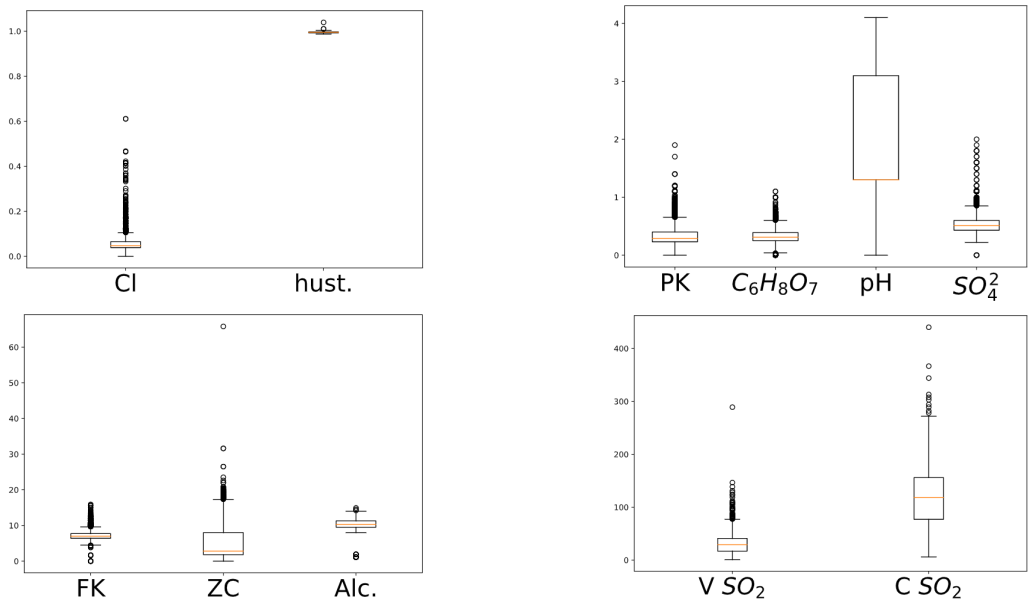
Obr. 20: 2D vizualizácia výsledkov algoritmu REX-DEPTH pomocou metódy PCA.

Do nasledovnej tabuľky si teraz vypíšme samotné hodnoty pozorovaných premenných pre týchto N identifikovaných vín aj vo výslednom poradí (od vrchu nadol).

Zvýrazníme pri tom niektoré vybrané hodnoty. Podčiarknuté a hrubím písmom sú tie hodnoty, ktoré sú najväčšie (najmenšie) pre danú premennú v celom dátovom súbore, iba hrubím písmom budú tie, ktoré sú druhé najväčšie (najmenšie) v celom súbore.

FK	PK	$C_6H_8O_7$	ZC	Cl	V SO_2	C SO_2	hust.	pH	SO_4^2	Alc.	kvalita
7,8	0,965	0,6	**65,8**	0,074	8	160	1,038	1,3	0,69	11,7	6
6,1	0,26	0,25	2,9	0,047	**289**	**440**	0,993	1,3	0,64	10,5	3
9,2	0,52	**1**	3,4	**0,61**	32	69	0,999	1,2	**2**	9,4	4
7,7	0,41	0,76	1,8	**0,611**	8	45	0,996	3,6	1,1	9,4	5
6,9	**1,9**	0,06	2,1	0,061	12	31	0,994	1,3	0,43	11,4	4
7,3	**1,7**	0,09	1,7	0,178	10	89	0,996	3,3	0,57	9	5
6,2	0,21	0,28	5,7	0,028	45	121	0,991	1,3	**1,8**	**1,12**	7
8,6	0,55	0,35	15,55	0,057	35,5	**366,5**	1,000	3,4	0,63	11	3
9,1	0,33	0,38	1,7	0,062	50,5	**344**	0,995	3,1	0,7	9,5	5
7,6	0,25	**1,1**	4,6	0,035	51	294	0,990	3,3	0,43	13,1	6

Aby sme ešte o niečo lepšie porozumeli výsledkom, zobrazme si *boxploty* pre jednotlivé premenné. Zoskupené sú podľa rozsahov daných premenných.



Obr. 21: Grafická vizualizácia dátového súboru pomocou boxplotov.

Môžeme konštatovať, že sme naozaj odhalili pozorovania, ktoré sú extremálne v nejakej premennej. Pomocou boxplotov tiež pozorujeme, že naše zoradenie je v súlade s výškou excentricity v daných premenných. Takouto analýzou môžeme následne aj vyhodnotiť, ako takéto extrémne hodnoty ovplyvňujú výslednú kvalitu vína.

Na prvý pohľad si už môžeme všimnúť, že z prvých 10 vín sú až 4 nízkej kvality (kvalita 3 alebo 4), pričom v celom dátovom súbore bolo takýchto nekvalitných vín menej ako 4% (podľa obr. 19). Vína s extremálnymi hodnotami pozorovaných premenných majú teda tendenciu byť menej kvalitné.

Pozrime sa, ako ovplyvňujú kvalitu extremálne hodnoty niektorých konkrétnych vlastností. Môžeme si napríklad všimnúť, že vysoké hodnoty zvyškového cukru alebo kyseliny citrónovej nemajú negatívny dopad na kvalitu. Takisto aj víno s veľmi nízkym obsahom alkoholu má stále dobré hodnotenie.

Naopak veľmi vysoké hodnoty oxidu siričitého, či už voľného alebo celkového, indikujú nízku kvalitu vína. Takisto negatívny dopad môže mať príliš vysoká hladina prchavej kyslosti.

Záver

Hlavným cieľom tejto diplomovej práce bolo predstavenie algoritmu REX a najmä konštrukcia jeho nadstavby REX-DEPTH dodávajúcej schopnosť zoradiť odľahlé pozorovania podľa ich excentricity a jej porovnanie s inými štandarnými metódami na detekciu takýchto pozorovaní.

V prvej časti sme predstavili niekoľko známych metód. Vysvetlili sme základný princíp ich fungovania a tiež popísali a graficky znázornili niektoré ich vlastnoti. Konkrétne metóda konvexného obalu so svojou nadstavbou *onion peelingu* bola motiváciou pre náš algoritmus REX-DEPTH.

V druhej kapitole sme predstavili úlohu MVEE a tiež úlohu hľadania optimálneho návrhu experimentu. K úlohe (15) sme odvodili Lagrangeovu duálnu úlohu, čím sme ukázali dualitu týchto úloh. Ďalej sme sa pozreli na samotnú úlohu MVEE a dokázali jej invariantnosť na zmenu jednotiek ako aj poukázali na spôsob riešenia pri necentralizovaných dátach.

Tretia kapitola bola už zameraná na samotný algoritmus REX na výpočet tejto úlohy. Definovali sme základné pojmy a následne vysvetlili jednotlivé kroky algoritmu.

V ďalšej kapitole sme skonštruovali nadstavbu REX-DEPTH. Ukázali sme motiváciu a základnú myšlienku pre tento algoritmus a spísali jeho jednotlivé kroky.

V piatej kapitole sme porovnávali náš algoritmus s metódami predstavenými v kapitole 1. Každej z vybraných metód sme spravili jednoduchú nadstavbu na zoraďovanie bodov a porovnali a graficky vizualizovali dosiahnuté výsledky. V závere kapitoly sme všetky porovnávané vlastnosti zhrnuli v tabuľke 1.

V poslednej kapitole sme náš algoritmus aplikovali na konkrétnom dátovom súbore, kde sme hľadali odľahlé vína na základe chemicko-fyzikálnych vlastností týchto vín. Takisto sme analyzovali, či takáto netypickosť v jednotlivých parametroch má vplyv na kvalitu daného vína. Výsledky sme graficky znázornili využitím metódy PCA a zhodnotili dosiahnuté výsledky.

Za hlavný prínos tejto práce považujeme popísanie algoritmu REX a najmä predstavenie vlastnej modifikácie REX-DEPTH schopnej odhaliť odľahlé pozorovania a zároveň ich aj zoradiť na základe ich excentricity. Takisto sme porovnali tento algoritmus s vybranými známymi metódami a poukázali na výhodné vlastnosti nášho algoritmu ako aj vhodné

spôsoby jeho použitia. Do budúcna by sa dalo experimentovať napríklad s voľbou preusporiadania váh na jednotlivých zložkách vektora návrhu pri prechode na ďalšiu iteráciu v modifikovanom algoritme, alebo inom efektívnom spôsobe odstraňovania vrstiev dát a opakovaného spušťania algoritmu s využívaním spätnej informácie.

Zoznam použitej literatúry

[1] Atkinson, A.C., Donev, A.N., Tobias, R.D.:*Optimum Experimental Designs, With SAS*, Oxford University Press, Oxford, 2007

[2] Ball, J. E., Harsh, A., Wei, P.:*Onion-Peeling Outlier Detection in 2-D data Sets*, International Journal of Computer Applications, Vol.139 No.3(2016), 26-31

[3] Chapman, K. W., Lapidus, S.H., Chupas, P.J.: *Applications of principal component analysis to pair distribution function data.* Journal of Applied Crystallography Vol.48 No.6 (2015), 1619-1626

[4] Cortes, C., Vapnik, V.: *Support-vector network*, Machine Learning,1-25.s, 1995

[5] Ester, M., Kriegel, H.P., Sander, J., Xu, X. :*A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, Institute for Computer Science, University of Munich, Vol.96 No.34(1996), 226-231

[6] Gen, J., Tao, Y: *Dbscan revisited: mis-claim, un-fixability, and approximation*, Chinese University od Hong Kong New Territories (2015), 519-530

[7] Halická, M.: DEA modely, učebné texty, FMFI UK, Bratislava, 2015, dostupné na internete (6.12.2017): http://www.iam.fmph.uniba.sk/institute/halicka/text/TextDEA40.pdf

[8] Harman, R., Filová, L., Richtárik, P.: *Randomized Algorithm for Computing Optimal Approximate Designs of Experiments*, Journal of the American Statistical Association, Vol.115 No.529 (2020), 348-361

[9] Hossein, S., Tyrone, L.V.,: *Computing the Approximate Convex Hull in High Dimensions*, Electrical Engineering and Computer Science, Colorado School of Mines, 2016

[10] Pázman, A.:*Foundations of Optimum Experimental Design*, Springer Netherlands, Heidelberg, 1986

[11] Platt, J., Shawe-Taylor, J., Schölkopf, B., Smola, A., Williamson, R.:*Support Vector Method for Novelty Detection*, NIPS, 1999

[12] Pukelsheim, F.:*Optimal Design of Experiments*, Society for Industrial and Applied Mathematics, Philadelphia, 2006

[13] Titterington, D.M. *Optimal design: Some geometrical aspects of D-optimality.*, Biometrika, Vol.62 No.2(1975), 313-320

[14] Yekkehkhany, B., Safari, A.,Homayouni, S., Hasanlou, M.:*A Comparison Study of Different Kernel Functions for SVM-based Classification of Multi-temporal Polarimetry SAR Data*,The International Archives of the Photogrammetry, Vol.XL-2/W3 (2014), 281-286

Príloha A

Zdrojové kódy naprogramovaných funkcií

A.0.1 REX

```
1  def REX_run(Fx, supp_ini, ver=1, gamma=4, eff=1-1e-9, it_max=
       float("inf"), t_max=30):
2      f = Fx
3      start = time.time()
4      [n,m] = f.shape
5      eps = 1.e-24
6      eff_inv = 1/eff
7      delta = 1.e-14
8
9      L = min(math.ceil(gamma*m),n)
10     iteracia = 0
11     supp = supp_ini
12     f_supp = f[supp]
13
14     w = np.zeros(n)
15     w[supp] = 1/len(supp)
16     w_supp = w[supp]
17     M = np.sqrt(w_supp)*f_supp.T@(np.sqrt(w_supp)*f_supp.T).T
18     g = np.diag(f@np.linalg.solve(M, f.T))/m
19     isort = np.argsort(-g)
20     lx_vec = np.random.permutation(isort[0:L])
21     K = len(supp)
22     kx_vec = np.random.permutation(supp)
23     df=pd.DataFrame(g)
24
25     while True:
```

```
26          k_ind = np.argmin(g[supp])
27          kb = supp[k_ind]
28          lb = np.argmax(g)
29          vec = [kb,lb]
30          g_kl = f[vec]@np.linalg.solve(M,f[vec].T)
31          al_optLBE = min(w[kb],(g_kl[1,1]-g_kl[0,0])/(2*(g_kl
                [0,0]*g_kl[1,1]-g_kl[0,1]**2+eps)))
32          w[kb] = w[kb] - al_optLBE
33          w[lb] = w[lb] + al_optLBE
34          M = M + al_optLBE*(f[[lb]].T@f[[lb]] - f[[kb]].T@f[[kb
                ]])
35          if (w[kb] < delta or w[lb] < delta):
36              for l in range(L):
37                  lx = lx_vec[l]
38                  for k in range(K):
39                      kx = kx_vec[k]
40                      vec = [kx,lx]
41                      g_kl = f[vec]@np.linalg.solve(M,f[vec].T)
42                      al_opt = min(w[kx],max(-w[lx],(g_kl[1,1]-
                            g_kl[0,0])/(2*(g_kl[0,0]*g_kl[1,1]-g_kl
                            [0,1]**2+eps))))
43                      w_tempk = w[kx] - al_opt
44                      w_templ = w[lx] + al_opt
45                      if (w_tempk<delta) or (w_templ<delta):
46                          w[kx] = w_tempk
47                          w[lx] = w_templ
48                          M = M + al_opt*(f[[lx]].T@f[[lx]] - f[[
                                kx]].T@f[[kx]])
49          else:
50              for l in range(L):
51                  lx = lx_vec[l]
```

```
52                  for k in range(K):
53                      kx = kx_vec[k]
54                      vec = [kx,lx]
55                      g_kl = f[vec]@np.linalg.solve(M,f[vec].T)
56                      al_opt = min(w[kx],max(-w[lx],(g_kl[1,1]-
                            g_kl[0,0])/(2*(g_kl[0,0]*g_kl[1,1]-g_kl
                            [0,1]**2+eps))))
57                      w[kx] = w[kx] - al_opt
58                      w[lx] = w[lx] + al_opt
59                      M = M + al_opt*(f[[lx]].T@f[[lx]] - f[[kx]].
                            T@f[[kx]])
60          supp = np.squeeze(np.argwhere(w>0).T, axis=(0,))
61          K = len(supp)
62          w_supp = w[supp]
63          g = np.diag(f@np.linalg.solve(M,f.T))/m
64          isort = np.argsort(-g)
65          lx_vec = np.random.permutation(isort[0:L])
66          kx_vec = np.random.permutation(supp)
67          df[iteracia]=g
68          iteracia = iteracia + 1
69          end = time.time()
70          if (iteracia > it_max) or (max(g)<eff_inv) or (end -
               start > t_max):
71              break
72
73      wsupp_best = w_supp
74      phi = np.linalg.det(M)**(1/m)
75      timer1 = end - start
76      df['w']=w
77      return timer1, wsupp_best, iteracia, w, df,g,phi,f
```

A.0.2 REX_ rep

```
1  def REX_rep(Fx, w, supp_ini, ver=1, gamma=4, eff=1-1e-9, it_max=
       float("inf"), t_max=30):
2      f = Fx
3      start = time.time()
4      [n,m] = f.shape
5      eps = 1.e-24
6      eff_inv = 1/eff
7      delta = 1.e-14
8
9      L = min(math.ceil(gamma*m),n)
10     iteracia = 0
11     supp = supp_ini
12     f_supp = f[supp]
13     w_supp = w[supp]
14     M = np.sqrt(w_supp)*f_supp.T@(np.sqrt(w_supp)*f_supp.T).T
15     g = np.diag(f@inv(M)@f.T)/m
16     isort = np.argsort(-g)
17     lx_vec = np.random.permutation(isort[0:L])
18     K = len(supp)
19     kx_vec = np.random.permutation(supp)
20     df=pd.DataFrame(g)
21
22     while True:
23         k_ind = np.argmin(g[supp])
24         kb = supp[k_ind]
25         lb = np.argmax(g)
26         vec = [kb,lb]
27         g_kl = f[vec]@inv(M)@f[vec].T
28         al_optLBE = min(w[kb],(g_kl[1,1]-g_kl[0,0])/(2*(g_kl
               [0,0]*g_kl[1,1]-g_kl[0,1]**2+eps)))
```

```
29          w[kb] = w[kb] - al_optLBE
30          w[lb] = w[lb] + al_optLBE
31          M = M + al_optLBE*(f[[lb]].T@f[[lb]] - f[[kb]].T@f[[kb
                ]])
32          if (w[kb] < delta or w[lb] < delta):
33              for l in range(L):
34                  lx = lx_vec[l]
35                  for k in range(K):
36                      kx = kx_vec[k]
37                      vec = [kx,lx]
38                      g_kl = f[vec]@inv(M)@f[vec].T
39                      al_opt = min(w[kx],max(-w[lx],(g_kl[1,1]-
                            g_kl[0,0])/(2*(g_kl[0,0]*g_kl[1,1]-g_kl
                            [0,1]**2+eps))))
40                      w_tempk = w[kx] - al_opt
41                      w_templ = w[lx] + al_opt
42                      if (w_tempk<delta) or (w_templ<delta):
43                          w[kx] = w_tempk
44                          w[lx] = w_templ
45                          M = M + al_opt*(f[[lx]].T@f[[lx]] - f[[
                                kx]].T@f[[kx]])
46          else:
47              for l in range(L):
48                  lx = lx_vec[l]
49                  for k in range(K):
50                      kx = kx_vec[k]
51                      vec = [kx,lx]
52                      g_kl = f[vec]@inv(M)@f[vec].T
53                      al_opt = min(w[kx],max(-w[lx],(g_kl[1,1]-
                            g_kl[0,0])/(2*(g_kl[0,0]*g_kl[1,1]-g_kl
                            [0,1]**2+eps))))
```

```
54                          w[kx] = w[kx] - al_opt
55                          w[lx] = w[lx] + al_opt
56                          M = M + al_opt*(f[[lx]].T@f[[lx]] - f[[kx]].
                                T@f[[kx]])
57          supp = np.squeeze(np.argwhere(w>0).T, axis=(0,))
58          K = len(supp)
59          w_supp = w[supp]
60          g = np.diag(f@inv(M)@f.T)/m
61          isort = np.argsort(-g)
62          lx_vec = np.random.permutation(isort[0:L])
63          kx_vec = np.random.permutation(supp)
64          df[iteracia]=g
65          iteracia = iteracia + 1
66          end = time.time()
67          if (iteracia > it_max) or (max(g)<eff_inv) or (end -
                start > t_max):
68              break
69      wsupp_best = w_supp
70      phi = np.linalg.det(M)**(1/m)
71      timer1 = end - start
72      df['w']=w
73      return timer1, phi, iteracia, w, g
```

A.0.3 Hľadanie najodľahlejšieho bodu

```
1  def REX_find(w,f,g, phi_origin):
2      ind = np.squeeze(np.argwhere(g>1-1e-8).T, axis=(0,))
3      volume = np.zeros(len(ind))
4      Timer = np.zeros(len(ind))
5      Iter = np.zeros(len(ind))
6      j = 0
7      for i in ind:
```

```
8           w_temp = copy(w)
9           maxin = np.argmax(np.where(g<1-1e-8,g,0))
10          w_temp[maxin] += w_temp[i]
11          w_temp[i] = 0
12          f_temp = copy(f)
13          f_temp[i] = np.mean(f, axis=0)
14          (timer1, phi, iteracia, w_new, g_new) = REX_rep(f_temp,
                w_temp,np.squeeze(np.argwhere(w_temp>0).T, axis=(0,))
                )
15          volume[j] = phi
16          Timer[j] = timer1
17          Iter[j] = iteracia
18          j += 1
19          if phi < phi_origin:
20              w_out = copy(w_new)
21              f_out = copy(f_temp)
22              g_out = copy(g_new)
23              phi_origin = copy(phi)
24              ind_out = copy(i)
25      return ind_out, w_out, f_out, g_out, phi_origin, volume
```