SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta chemickej a potravinárskej technológie

Evidenčné číslo: FCHPT-5415-61615

Použitie Support Vector Machine na klasifikáciu dát

Bakalárska práca

2015 **Natália Mikušová**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta chemickej a potravinárskej technológie

Evidenčné číslo: FCHPT-5415-61615

Použitie Support Vector Machine na klasifikáciu dát

Bakalárska práca

Študijný program: automatizácia, informatizácia a manažment v chémii a potravinárstve
Číslo študijného odboru: 2621
Názov študijného odboru: 5.2.14. automatizácia, 5.2.52. priemyselné inžinierstvo
Školiace pracovisko: Ústav informatizácie, automatizácie a matematiky
Vedúci záverečnej práce: doc. Ing. Michal Kvasnica, PhD.

Bratislava 2015 **Natália Mikušová**

Slovenská technická univerzita v Bratislave
Ústav informatizácie, automatizácie a matematiky

Fakulta chemickej a potravinárskej technológie
Akademický rok: 2014/2015
Evidenčné číslo: FCHPT-5415-61615

STU FCHPT

ZADANIE BAKALÁRSKEJ PRÁCE

Študentka:	**Natália Mikušová**
ID študenta:	61615
Študijný program:	automatizácia, informatizácia a manažment v chémii a potravinárstve
Kombinácia študijných odborov:	5.2.14. automatizácia, 5.2.52. priemyselné inžinierstvo
Vedúci práce:	doc. Ing. Michal Kvasnica, PhD.

Názov práce: **Použitie Support Vector Machine na klasifikáciu dát**

Špecifikácia zadania:

Cieľom práce je navrhnúť a implementovať systém na klasifikáciu dát založený na Support Vector Machine (SVM). Takýto systém môže byť použitý napríklad na biometrickú identifikáciu osôb na základe akcelerometrických dát, alebo na identifikáciu zaradenia produktov do jednotlivých tried na základe senzorických informácií.

Úlohy:
* výber vhodných dát na klasifikáciu
* implemtácia SVM v Matlabe
* návrh systému na zber akcelerometrických dát na účely biometrickej identifikácie
* zber akcelerometrických dát
* verifikácia presnosti a kvality klasifikácie

Rozsah práce: 40

Riešenie zadania práce od: 16. 02. 2015

Dátum odovzdania práce: 24. 05. 2015

L. S.

Natália Mikušová
študentka

prof. Ing. Miroslav Fikar, DrSc.
vedúci pracoviska

prof. Ing. Miroslav Fikar, DrSc.
garant študijného programu

Týmto by som rada poďakovala vedúcemu mojej bakalárskej práce doc. Ing. Michalovi Kvasnicovi, PhD. za jeho rady, pripomienky a pomoc poskytnutú pri vypracovávaní bakalárskej práce. Ďalej chcem poďakovať všetkým dobrovoľníkom, ktorí venovali vzorky svojich pohybov na účely identifikácie.

Natália Mikušová

Bratislava, 2015

Abstrakt

Cieľom tohto bakalárskeho projektu je biometrická identifikácia osôb na základe ich pohybov. V prvej časti práce sa zaoberáme získavaním reálnych vzoriek pohybov, ktoré sú potrebné na identifikáciu. Vzorky získavame meraním akcelerácii vznikajúcich pri ľudských pohyboch. V druhej časti namerané údaje spracovávame v Matlab-e. To zahŕňa použitie vybranej metódy extrakcie vlastností na transformáciu pôvodných akcelerácií na súbory vlastností. V tretej, poslednej časti práce vstupujú extrahované vlastnosti do procesu separácie. Separáciu vykonávame pomocou algoritmu Support Vector Machine a metódy klasifikačných stromov.

Kľúčové slová:

Biometrická identifikácia; akcelerácie; smartfón; extrakcia vlastností; Support Vector Machine; klasifikačný strom.

Abstract

The objective of this thesis is the biometric identification of persons based on their movements. In the first section, we will gather real motion samples, which are necessary to identify our subjects. Samples are obtained by measuring the accelerations generated during human movements. In the second section, we will process the obtained data in Matlab. This will also include utilizing the selected method of feature extraction to transform initial accelerations to a vector of features. In the third, last part, extracted features enter the separation process. The separation is performed using the Support Vector Machine algorithm and a classification trees method.

Keywords:

Biometric identification; accelerations; smartphone; feature extraction; Support Vector Machine; classification tree.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Smartfóny, tablety alebo notebooky sú významnou súčasťou života takmer každého z nás. Denne ich využívame na komunikáciu, zábavu, či prácu. Obsahujú informácie o nás, našich aktivitách, ľuďoch, s ktorými udržiavame kontakt. Preto sa stáva čoraz dôležitejším zabezpečovať všetky tieto zariadenia a chrániť tak svoje súkromie a osobné údaje pred nepovoleným prístupom.

Medzi najtrendovejšie spôsoby ochrany v súčasnosti patria systémy založené na biometrickej identifikácií osôb. Mnohé spoločnosti, napr. Apple, Samsung alebo Google majú tieto systémy, ktoré overujú identitu používateľa na základe niektorého unikátneho ľudského znaku, implementované vo svojich produktoch.

Spoločnosť Apple zaviedla do svojich zariadení iPhone a iPad bezpečnostný systém s názvom Touch ID založený na overovaní identity majiteľa zariadenia na základe odtlačku prstu. Odtlačky sa snímajú senzorom po priložení prsta na obrazovku. Porovnanie vzorového odtlačku s novým odtlačkom prebieha vždy pri odomykaní telefónu alebo tabletu, potvrdzovaní platieb alebo sprístupňovaní informácií [1]. Podobný systém môžeme nájsť tiež v niektorých notebookoch a smartfónoch od spoločnosti Samsung [2].

Podobný spôsob odomykania smartfónu ponúka aj operačný systém Android so svojou funkciou Face Unlock. Ako vyplýva z názvu funkcie, jedná sa o overovanie

identity na základe čŕt tváre. Systém funguje rovnako ako v predchádzajúcom prípade. Pri snahe odomknúť zariadenie fotoaparát vyhotoví fotografiu osoby držiacej smartfón alebo tablet. Ak sa osoba na fotografii zhoduje s majiteľom zariadenia, systém Face Unlock ho odomkne. V opačnom prípade nie je prístup povolený [3].

Posledným bezpečnostným systémom, ktorý spomenieme, je aplikácia On-Body Detection dostupná v operačnom systéme Android. Táto aplikácia využíva na ochranu zariadenia akcelerometer. Ak akcelerometer zaznamenáva pohyb, čo znamená, že so zariadením sa pracuje, aplikácia ho necháva odomknuté. V opačnom prípade, keď zariadenie nehybne leží, uzamkne ho a zabráni nepovolenému prístupu. Opätovné odomknutie sa musí vykonať pomocou hesla [4].

V tejto práci sa pokúsime vytvoriť podobný systém, aký je implementovaný v aplikácií On-Body Detection. Budeme sa teda zaoberať biometrickou identifikáciou osôb na základe pohybu. Nebudeme však vyhodnocovať, či je so zariadením manipulované, budeme sa snažiť presne identifikovať osobu, ktorá so zariadením narába.

Kapitola 1

Biometrická identifikácia osôb

Vo všeobecnosti sa biometria považuje za vedu, ktorá sa zaoberá štúdiom a analýzou merateľných biologických charakteristík. Medzi takéto merateľné charakteristiky ľudí zaraďujeme napr. odtlačky prstov, ryhovanie rúk, deoxyribonukleovú kyselinu (DNA), črty tváre, hlas, sietnice, dúhovky a ďalšie [5]. Pre všetky vyššie uvedené charakteristiky platí, že pre daného jedinca sú unikátne (výnimka nastáva iba v prípade DNA jednovaječných dvojčiat [6]). Práve pre túto jedinečnosť má biometria svoje zastúpenie v oblasti informačných technológií (IT), kde ju spájame najmä s bezpečnosťou údajov.

Biometrickú identifikáciu v IT môžeme definovať ako proces overovania identity porovnávaním unikátnych ľudských znakov. Bezpečnostné systémy fungujúce na princípe biometrickej identifikácie preto často slúžia na ochranu osobných údajov [5]. Daný bezpečnostný systém, ktorý môže byť implementovaný v rôznych druhoch zariadení obsahujúcich osobné údaje prečíta vybraný ľudský znak a informácie užívateľovi sprístupní až po overení jeho identity na základe tohto znaku. Týmto procesom identifikácie sa zabráni nepovolenému prístupu a prípadnému zneužitiu súkromných dát.

Obr. 1.1: Snímač odtlačkov prstov

Softvéry, ktoré identifikujú osoby na základe ich biometrie majú svoje veľké uplatnenie napr. aj vo forenzných vedách. Medzi najznámejšie a veľmi často používané metódy identifikácie páchateľov trestných činov patrí porovnávanie odtlačkov prstov alebo DNA [7]. Veľkú kategóriu tvoria tiež softvéry, ktoré dokážu identifikovať objekty alebo osoby na fotografiách a kamerových záznamoch.

1.1 Identifikácia na základe pohybu

Jednou z merateľných biologických charakteristík, ktorú sme zatiaľ nespomenuli je ľudský pohyb. Pohyb, rovnako ako ostatné uvedené charakteristiky, je špecifický a jedinečný pre každého človeka. Z tohto tvrdenia priamo vyplýva, že je možné osobu identifikovať sledovaním jej pohybov. Práve biometrickou identifikáciou na základe pohybu sa zaoberá táto práca.

1.1.1 Akcelerometre

V predchádzajúcej časti sme pohyb označili ako merateľnú charakteristiku. Prvým krokom preto bude zmeranie ľudských pohybov.

Každý pohyb, ktorý naše telo vykoná charakterizuje zrýchlenie alebo iným slovom akcelerácia. Akcelerácia je fyzikálna veličina, ktorú môžeme zjednodušene definovať ako zmenu rýchlosti za jednotku času [8]. Jednotkou zrýchlenia v sústave SI je $^{m}/_{s^2}$. Zariadenie, ktorým budeme akcelerácie vzniknuté pri pohyboch merať sa nazýva akcelerometer.

Akcelerometre sú považované za veľmi užitočné zariadenia v priemysle aj vede, kde majú mnoho aplikácii. V leteckom a automobilovom priemysle sa využívajú ako komponent pokročilých navigačných systémov. Sú súčasťou kamier a fotoaparátov, kde pomáhajú stabilizovať obraz. Medzi jeho významné úlohy patrí tiež meranie vibrácii budov a mostov alebo otrasov Zeme [9].

Dostupné sú v dnešnej dobe aj pre širokú verejnosť. Rovnako ako iné známe senzory (napr. gyroskop alebo GPS) sa nachádzajú vo vnútri moderných smartfónov alebo tabletov.

Obr. 1.2: Meranie akcelerácií

V tomto projekte budeme využívať smartfón. Ako naznačuje obrázok 1.2, akcelerácie meriame pozdĺž troch osí - X, Y a Z. Pri náklone smartfónu dopredu alebo dozadu sa zmení akcelerácia pozdĺž osi X. Otočením telefónu na pravú alebo ľavú stranu sa zmení akcelerácia pozdĺž osi Y. Zmenou z vertikálnej polohy (smartfón "stojí"na kratšej hrane) do horizontálnej (smartfón "leží"na dlhšej hrane) sa zmení akcelerácia pozdĺž osi Z.

Kapitola 2

Zber akcelerometrických údajov

Aby sme overili tvrdenie, že každý človek sa pohybuje jedinečným spôsobom, potrebujeme získať vzorky pohybov rôznych ľudí. V rámci projektu sme preto oslovili 6 dobrovoľníkov, ktorí tieto vzorky poskytli. Oslovení dobrovoľníci boli rôzneho veku (17 – 65 rokov) a pohlavia.

2.1 Priebeh merania

Ako bolo spomenuté v predchádzajúcej kapitole, akcelerácie meriame pomocou smartfónu vybaveného akcelerometrom. Smartfón sme ako ťažiskové zariadenie vybrali kvôli jeho vhodnej veľkosti a dostupnosti. Jednotnosť vzoriek sme dosiahli používaním jediného zariadenia, telefónu Samsung S6310 Galaxy Young. Počas celého zberu akcelerometrických dát sme dodržiavali identický priebeh merania.

Za jedno meranie (jednu vzorku) považujeme zdvihnutie telefónu k uchu. Takýto pohyb väčšina z nás vykonáva denne pri prijímaní alebo vytáčaní hovorov. Príprava na meranie spočíva vo vložení telefónu do vrecka nohavíc. Akcelerácie začíname merať vtedy, keď dobrovoľník siahne pre zariadenie, ktoré si umiestnil do vrecka. Meranie končí v okamihu priloženia telefónu k uchu.

Každý zo šiestich dobrovoľníkov opakoval tento postup niekoľkokrát. Prehľad počtu opakovaní udáva tabuľka 2.1.

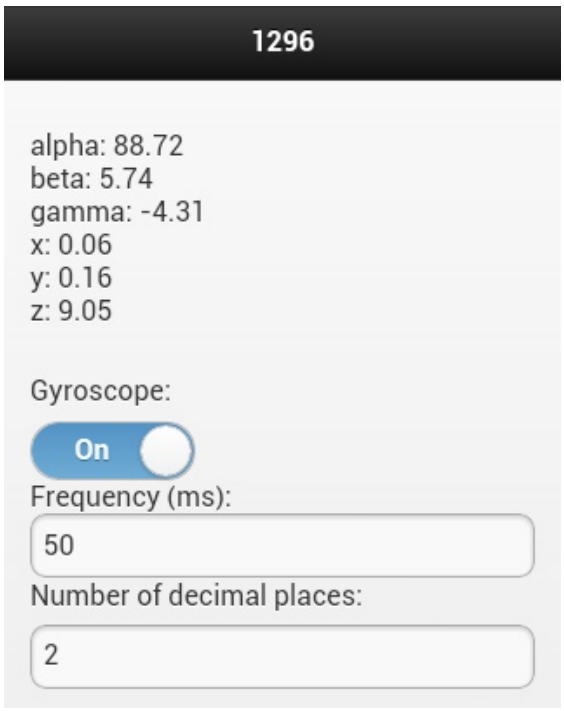
Tabuľka 2.1: Prehľad počtu nameraných údajov

Dobrovoľník č.	Vek	Pohlavie	Počet vykonaných meraní
1	22	žena	30
2	17	muž	31
3	22	muž	30
4	65	žena	32
5	47	muž	31
6	23	žena	31

Obr. 2.1: Samsung S6310 Galaxy Young

K akcelerometru, ktorý chápeme ako hardvér vytvoríme prístup pomocou webovej aplikácie GYRO. GYRO sprostredkováva alebo interpretuje údaje z akcelerometra užívateľovi. Vyobrazuje číselné hodnoty akcelerácii a navyše aj uhlov v reálnom čase. Aplikáciu môžeme vidieť na obrázku 2.2.

Nové meranie začíname zadaním webovej adresy aplikácie GYRO - http://kirp.chtf.stuba.sk:8025/html/gyro.html - do internetového prehliadača smartfónu.



Obr. 2.2: Aplikácia GYRO

Štvorčíslie nachádzajúce sa vo vrchnej časti aplikácie GYRO nazývame pin. Pin sa vygeneruje náhodne vždy pri opakovanom načítaní aplikácie. Pod pinom sa zobrazujú aktuálne hodnoty 6 veličín. alpha, beta a gamma sú uhly náklonu a x, y a z akcelerácie merané pozdĺž troch osí X, Y a Z. Uhly meriame v stupňoch a akcelerácie v metroch za sekundu na druhú.Ak sa modrý slider nachádza v polohe On, aplikácia ukazuje aktuálne hodnoty uhlov aj akcelerácii (v závislosti od vykonaného pohybu). V opačnom prípade, v polohe Off sa aktuálne hodnoty nesnímajú. Pri zmene z polohy On do polohy Off sa na obrazovke zachovajú posledné namerané hodnoty. Pole Frequency dovoľuje nastaviť frekvenciu merania v

milisekundách. Predvolená hodnota frekvencie je 50 milisekúnd. V poslednom poli s názvom Number of decimal places môžeme upraviť počet desatinných miest meraných veličín. Ak neuvedieme inak, uhly aj akcelerácie sa merajú na dve desatinné miesta.

Celý proces zdvihnutia telefónu k uchu zaznamená tzv. klient. Klient je prostredím, kde sledujeme priebeh jednotlivých meraní. Tu sa tvorí konečný výstup, matica nameraných uhlov a akcelerácii. Webovú adresu klienta - `http://kirp.chtf.stuba.sk:8025/html/client.html` - vpisujeme do internetového prehliadača počítača. Počítač je okrem smartfónu jediné ďalšie zariadenie, s ktorým pracujeme. Klient je znázornený na obrázku 2.3.

Obr. 2.3: Klient

Prvým krokom k zaznamenaniu merania je zadanie štvormiestneho pinu do prázdneho poľa Topic. Vložením štvorčíslia sa klient páruje s konkrétnym zariadením. Pod poľom Topic sa nachádzajú dve tlačidlá. Ľavým tlačidlom Start spustíme zaznamenávanie údajov. Po stisnutí tohto tlačidla sa za poľom Topic zobrazí slovo open. Následne je prítomné počas celého merania. Pravé tlačidlo Stop slúži na ukončenie zaznamenávania. Stisnutie tlačidla Stop indikuje slovo closed. Výsledkom zaznamenávania je matica skladajúca sa so siedmich stĺpcov. Prvý stĺpec s názvom Timestamp obsahuje časový údaj, druhý až štvrtý stĺpec namerané uhly alpha, beta a gamma, piaty až siedmy stĺpec namerané akcelerácie x, y a z. Príklad takejto matice je na obrázku 2.4.

Klientom vygenerovanú maticu môžeme uchovať stiahnutím do počítača.

Zvolením možnosti Export to Excel uložíme maticu vo formáte XML. Druhá možnosť, Export to CSV, uloží maticu vo formáte CSV. V tejto práci využívame druhú možnosť.

Topic: 1296 **closed**

Start Stop

Export to Excel Export to CSV

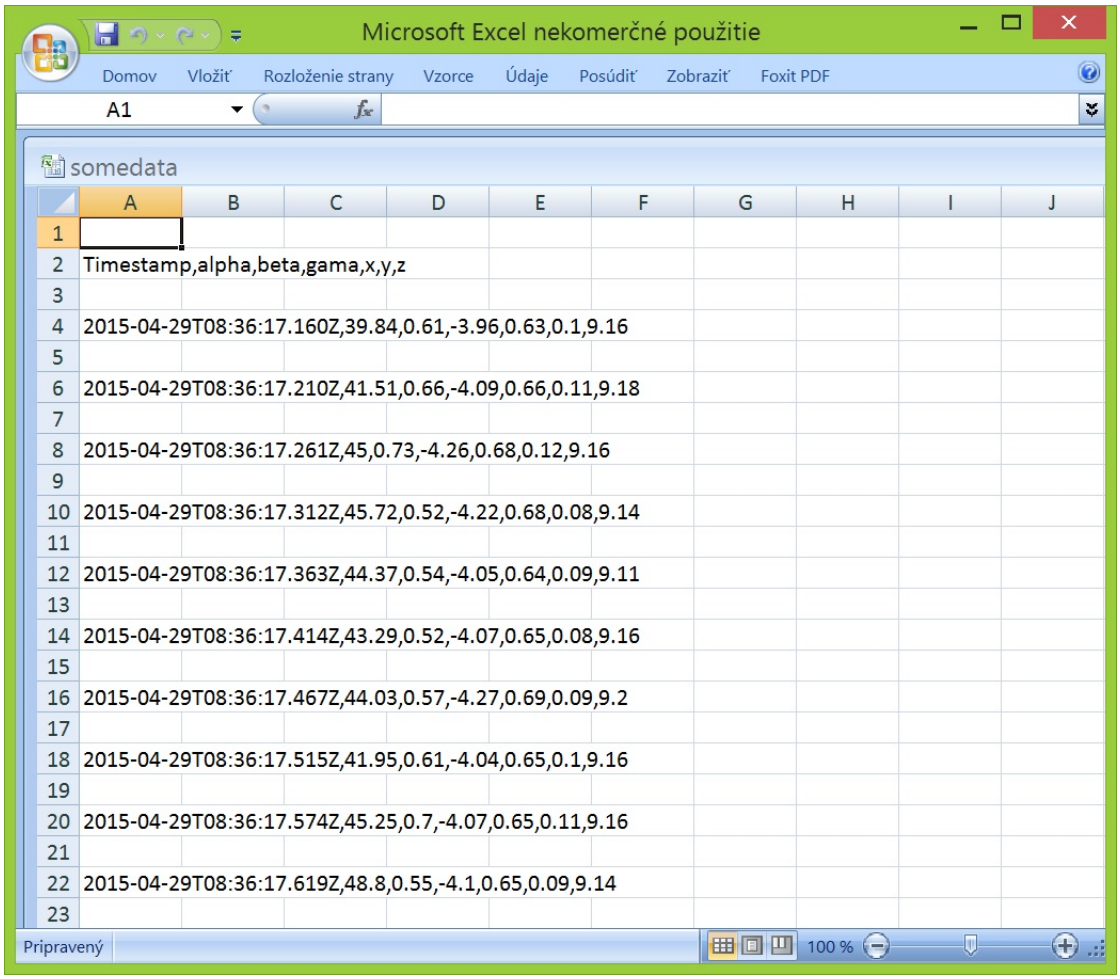
Timestamp	alpha	beta	gama	x	y	z
2015-04-29T08:36:17.160Z	39.84	0.61	-3.96	0.63	0.1	9.16
2015-04-29T08:36:17.210Z	41.51	0.66	-4.09	0.66	0.11	9.18
2015-04-29T08:36:17.261Z	45	0.73	-4.26	0.68	0.12	9.16
2015-04-29T08:36:17.312Z	45.72	0.52	-4.22	0.68	0.08	9.14
2015-04-29T08:36:17.363Z	44.37	0.54	-4.05	0.64	0.09	9.11
2015-04-29T08:36:17.414Z	43.29	0.52	-4.07	0.65	0.08	9.16
2015-04-29T08:36:17.467Z	44.03	0.57	-4.27	0.69	0.09	9.2
2015-04-29T08:36:17.515Z	41.95	0.61	-4.04	0.65	0.1	9.16
2015-04-29T08:36:17.574Z	45.25	0.7	-4.07	0.65	0.11	9.16
2015-04-29T08:36:17.619Z	48.8	0.55	-4.1	0.65	0.09	9.14
2015-04-29T08:36:17.674Z	47.04	0.61	-4.1	0.65	0.1	9.13
2015-04-29T08:36:17.758Z	46.02	0.63	-3.94	0.63	0.1	9.1
2015-04-29T08:36:17.789Z	49.46	0.54	-4.08	0.64	0.08	9.17
2015-04-29T08:36:17.842Z	48.27	0.61	-3.95	0.63	0.1	9.15
2015-04-29T08:36:17.898Z	47.12	0.66	-4.15	0.66	0.11	9.15
2015-04-29T08:36:17.955Z	46.58	0.64	-4.12	0.66	0.1	9.14

Obr. 2.4: Matica nameraných uhlov a akcelerácii

2.2 Súborový formát CSV

CSV alebo Comma-separated values je súborový formát, ktorý uchováva údaje z tabuliek vo forme čistého textu. Údaje sú rovnako ako v bežnej tabuľke zoradené do riadkov a stĺpcov. Riadky sú od seba oddelené znakom odriadkovania, stĺpce najčastejšie čiarkou (Comma-separated values znamená v slovenčine hodnoty oddelené čiarkami). Pretože formát CSV nemá presne zadefinovanú svoju štruktúru, je možné stĺpce oddeliť aj iným znakom, napr. tabulátorom alebo bodkočiarkou. Iný znak sa preferuje pri jazykoch (patrí sem aj slovenčina), ktoré používajú znak čiarky na oddelenie celej a desatinnej časti čísla [10]. V prípade, že iný znak využívať nechceme, máme možnosť uzavrieť hodnotu obsahujúcu čiarku do dvojitých úvodzoviek [11].

Súborový formát CSV sa pre svoju jednoduchosť a nenáročnosť často v praxi používa na prenos tabuľkových dát z programu do programu.



Obr. 2.5: Ukážka CSV súboru

Kapitola 3

Spracovanie nameraných údajov

Spracovanie nameraných údajov realizujeme v Matlab-e. Matlab je interaktívne prostredie určené na numerické výpočty, analýzu a vizualizáciu dát, modelovanie a simuláciu dynamických systémov, programovanie a návrh algoritmov, vývoj aplikácii a iné [12].

3.1 Export CSV súboru do Matlab-u

Namerané údaje uchované v jednotlivých CSV súboroch analyzujeme v Matlab-e. Obsah CSV súboru prevedieme do prostredia Matlab-u funkciou `csvRead`. Táto funkcia prečíta formátovaný text v CSV súbore a spracuje ho do podoby vhodnej pre Matlab. Matlab podporuje veľké množstvo dátových štruktúr. Jednou zo základných štruktúr je dvojrozmerné pole alebo matica. Práve matica nazvaná `data` je výstupom z funkcie `csvRead`. Jediným vstupom do funkcie je premenná `fname` - názov vybraného CSV súboru.

```
1 function data = csvRead(fname)
2
3 data = [];
```

```
4
5  fid = fopen(fname, 'r');
6
7  fgets(fid); % skip initial empty line
8  fgets(fid); % skip labels
9  fgets(fid); % skip next empty line
10
11 while ~feof(fid)
12
13     line = fgets(fid);
14     line = line(26:end);
15     parsed = textscan(line, '%f,%f,%f,%f,%f,%f');
16     data = [data; cat(2, parsed{:})];
17     fgets(fid);
18
19 end
20
21 fclose(fid);
22
23 end
```

Algoritmus, ktorým vytvárame maticu `data` vyplýva zo štruktúry CSV súboru. Ukážku CSV súboru vidíme na obrázku 2.5.

V úvode programu zadefinujeme premennú `data` ako prázdne dvojrozmerné pole. Funkciou `fopen` otvoríme CSV súbor (jeho meno je uložené v premennej `fname`) na čítanie (čítanie – 'r'). Trikrát zopakujeme príkaz `fgets(fid)`, čím sa presunieme na štvrtý riadok súboru, kde sa nachádzajú prvé namerané údaje. Prvý a tretí prázdny riadok ako aj hlavičku tabuľky v druhom riadku vynecháme.

Maticu `data` plníme pomocou cyklu while. Príkazy umiestnené v cykle sa opakujú dovtedy, kým program nezaznamená koniec súboru - `feof`. ~ je znak pre

negáciu. Prvý príkaz slučky načíta do premennej `line` jeden riadok súboru. Druhý príkaz modifikuje premennú tak, aby neobsahovala časový údaj pochádzajúci zo stĺpca s názvom Timestamp. Zvyšné hodnoty uhlov a akcelerácii rozseparuje funkcia `textscan`. Táto funkcia prečíta formátované údaje v textovom súbore [13]. Má nasledovnú syntax:

```
C = textscan(fileID, format)
```

Prvý vstup do funkcie, premenná `fileID` je identifikátor súboru. Jeho hodnota sa získa spustením funkcie `fopen`. Druhým vstupom do funkcie je premenná `format`. Táto premenná určuje formát, v akom sa údaj zo súboru zapíše do bunkového poľa `C`, výstupu z funkcie `textscan`. V našom prípade sa vykoná konverzia čistého textu na číselný údaj. %f je označenie dátového typu float.

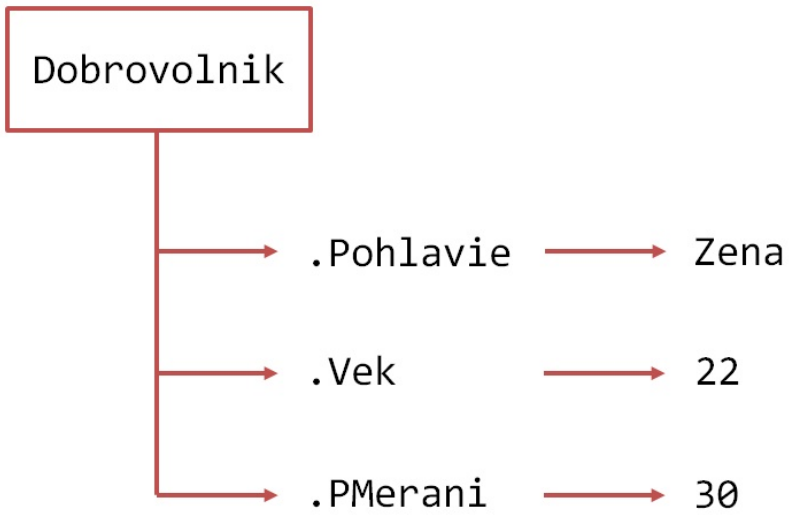
Predposledným príkazom v cykle vložíme hodnoty do premennej `data`. Matica `data` po každej iterácii zmení svoju veľkosť. V závere cyklu vynecháme prázdny riadok, ktorý sa nachádza za každým riadkom naplneným údajmi. Po vykonaní všetkých opakovaní program zavrie daný súbor pomocou funkcie `fclose`.

3.2 Práca so štruktúrami

Ďalším veľmi rozšíreným typom premennej v Matlab-e je štruktúra. Pretože sa tento typ premennej objavuje v tejto aj ďalších kapitolách práce, venujeme jej túto časť.

Štruktúra je premenná, ktorá sa skladá z jednej alebo viacerých položiek. Do jednotlivých položiek štruktúry zapisujeme údaje. Platí, že každá položka môže uchovávať údaje rôznych typov a veľkostí [14]. Vytvorenie štruktúry si ukážeme na príklade.

Chceme vytvoriť štruktúru s názvom `Dobrovolnik` (obrázok 3.1), ktorá bude obsahovať tri položky - `Pohlavie`, `Vek` a `PMerani`. Položky štruktúry `Dobrovolnik` majú uchovávať informácie o dobrovoľníkovi, ktorý sa zúčastnil zberu akcelerometrických údajov. Do položky `Pohlavie` sa zapíše pohlavie dobrovoľníka ako reťazec, do položky `Vek` vek dobrovoľníka ako celé číslo a do položky `PMerani` sa zapíše počet realizovaných meraní, znovu ako celé číslo. Na tomto príklade môžeme vidieť, že každá položka smie obsahovať premennú iného typu a veľkosti.



Obr. 3.1: Štruktúra Dobrovolnik

Jednotlivé položky štruktúry definujeme postupne pomocou operátora bodka. Cez tento operátor k nim tiež pristupujeme. Prvým príkazom vytvoríme štruktúru, jej prvú položku a zároveň tejto položke priradíme určenú hodnotu. Ďalšími príkazmi do štruktúry doplníme zostávajúce položky a ich hodnoty.

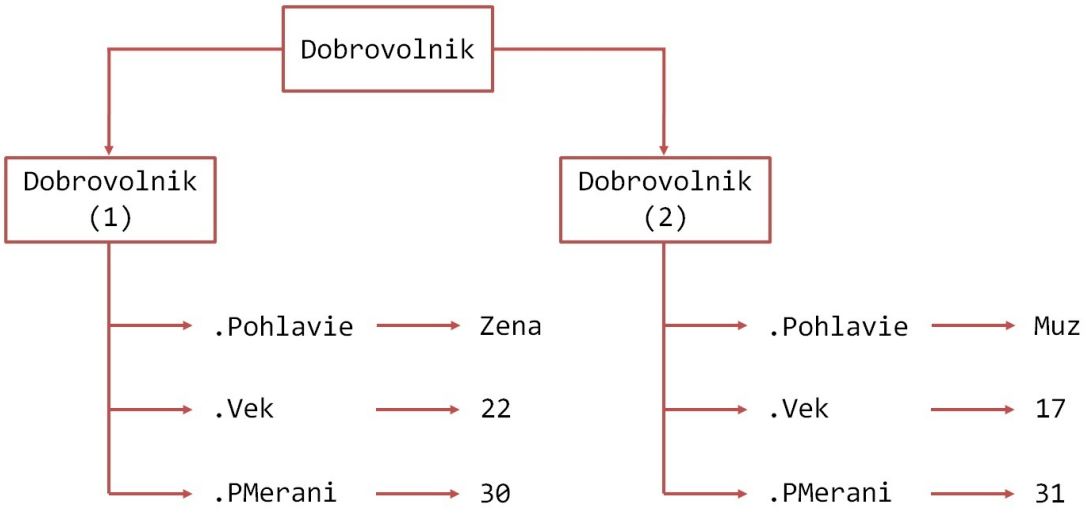
```
1 Dobrovolnik.Pohlavie = 'Zena';
2 Dobrovolnik.Vek = 22;
3 Dobrovolnik.PMerani = 30;
```

```
4 Dobrovolnik
```

```
Dobrovolnik =

    Pohlavie: 'Zena'
        Vek: 22
     PMerani: 30
```

Spojením dvoch a viacerých štruktúr s rovnakými položkami vznikne pole štruktúr (obrázok 3.2). Z existujúcej štruktúry tvoríme pole štruktúr indexovaním cez okrúhle zátvorky.



Obr. 3.2: Pole štruktúr Dobrovolnik

```
1 Dobrovolnik(2).Pohlavie = 'Muz';
2 Dobrovolnik(2).Vek = 17;
3 Dobrovolnik(2).PMerani = 31;
4 Dobrovolnik
```

```
Dobrovolnik =

1x2 struct array with fields:

    Pohlavie
    Vek
    PMerani
```

Pravidlá platné pre štruktúry a polia štruktúr [14]:

- Všetky štruktúry v poli štruktúr majú rovnaký počet položiek.
- Všetky štruktúry v poli štruktúr majú položky rovnakého názvu.
- Položka rovnakého názvu umiestnená v rôznych štruktúrach môže uchovávať údaje rôznych typov a veľkostí.

3.3 Hromadné spracovanie nameraných údajov

Funkcia `csvRead` predstavená v prvej časti tejto kapitoly umožňuje pri spustení z príkazového riadka exportovať práve jeden CSV súbor do Matlab-u. V tejto chvíli disponujeme 185 CSV súbormi pochádzajúcimi od 6 rôznych dobrovoľníkov. V podkapitole Hromadné spracovanie nameraných údajov si ukážeme, ako sa vyhnúť mnohonásobnému spúšťaniu funkcie `csvRead` a zároveň ako zefektívniť, teda automatizovať proces exportu údajov zo súborov.

Na úvod zvolíme organizáciu údajov vhodnú na hromadné spracovanie nameraných údajov. Údaje od jedného dobrovoľníka vo forme CSV súborov umiestnime do samostatného priečinka. Priečinok pomenujeme iniciálkami danej osoby (napr. NM). Názov CSV súboru vo vnútri priečinka sa skladá rovnako z iniciálok osoby a poradového čísla merania (napr. NM1). Priečinky všetkých dobrovoľníkov vložíme

do jedného spoločného priečinka. Do tohto hlavného priečinka vložíme tiež všetky funkcie, ktoré využijeme pri spracovaní údajov.

Aby sme dynamicky mohli spracovať ľubovoľný počet údajov, potrebujeme poznať hodnoty dvoch premenných - počet dobrovoľníkov, ktorí sa zúčastnili zberu dát a počet vzoriek každého dobrovoľníka. Pri zvolenej organizácii sa počet dobrovoľníkov rovná počtu podpriečinkov v hlavnom priečinku a počet vzoriek sa rovná počtu CSV súborov v podpriečinku. Tieto hodnoty je výhodné získavať dynamicky, napriek tomu, že v súčasnosti sú známe. Poskytne nám to možnosť manipulovať s počtami údajov počas celého trvania projektu.

Počet a zároveň mená dobrovoľníkov (meno dobrovoľníka je uložené v názve podpriečinka) zistí funkcia `SubfoldersList`. Jediným vstupom do funkcie je absolútna adresa hlavného priečinka v počítači. Adresu tohto priečinka využije funkcia `dir` v treťom riadku, ktorá uloží zoznam podpriečinkov a súborov nachádzajúcich sa v hlavnom priečinku do poľa štruktúr `dcontent`. Prehľad položiek štruktúry `dcontent` ponúka tabuľka 3.1 [15].

Tabuľka 3.1: Položky štruktúry vytvorenej funkciou dir

Názov položky	Popis	Dátový typ
name	názov priečinka alebo súboru	char
date	dátum a čas vytvorenia záznamu	char
bytes	veľkosť súboru v bytoch	double
isdir	rozlíšenie priečinka a súboru - 1 pre priečinok, 0 pre súbor	logical
datenum	modifikácia dátumu na sériové číslo	double

Druhým príkazom sa zapíše obsah položky `isdir` štruktúry `dcontent` do

premennej `isub`. Pomocou vektora `isub` typu logical a množinových zátvoriek vytvoríme bunkové pole `FolderNames`. Iba v prípade, keď sa v položke `isdir` resp. v prvku vektora `isub` nachádza logická hodnota 1 načítame do bunkového poľa hodnotu z položky `name` - meno podpriečinka vo forme reťazca. Premenná `FolderNames` - zoznam podpriečinkov - je výstupom z funkcie `SubfoldersList`.

```
1 function FolderNames = SubfoldersList(FolderPath)
2
3 dcontent = dir(FolderPath);
4 isub = [dcontent(:).isdir];
5 FolderNames = {dcontent(isub).name}';
6 FolderNames(ismember(FolderNames,{'.','..'})) = [];
7
8 end
```

Ďalej zistíme počet vzoriek od každého dobrovoľníka a v jednom kroku ich spracujeme.

Prenos údajov z CSV súborov nachádzajúcich sa v jednotlivých podpriečinkoch naraz vykoná funkcia `CollectRawData`. Vstupným argumentom do tejto funkcie je zoznam podpriečinkov uložený v premennej `FolderNames`. Výstupom je pole štruktúr `AllRawData`, ktoré bude mať pri súčasných podmienkach rozmer 1×185 (uskutočnili sme 185 meraní).

Premennú `AllRawData` vytvoríme na začiatku funkcie ako prázdne pole. Pole naplníme štruktúrami použitím for cyklu. Cyklus sa zopakuje pre všetky podpriečinky, pričom jeden podpriečinok spracuje vnorená funkcia `csvExport`. Obsah jedného podpriečinka sa vloží do pomocnej štruktúry `PersonRawData`. V závere zväčšíme pole štruktúr `AllRawData` o štruktúru `PersonRawData`.

Vstupným parametrom do vnorenej funkcie `csvExport` je meno podpriečinka vo forme reťazca, t.j. jeden prvok bunkového poľa `FolderNames`. Výstupným argumentom z funkcie je pole štruktúr `ExportData`. Premennú `ExportData` vytvoríme v úvode programu ako prázdne pole, rovnako ako v predchádzajúcom prípade. Potom sa príkazom `cd(FolderName)` presunieme z hlavného priečinka do podpriečinka určeného premennou `FolderName`. Pomocou funkcie `dir` nájdeme v podpriečinku všetky súbory s koncovkou .csv. Výstupnú premennú `ExportData` naplníme vo for cykle. Znovu používame pomocnú štruktúru - `Data`. Do jej položky `name` zapíšeme hodnotu premennej `FolderName` - meno podpriečinka (v každom opakovaní zapíšeme rovnakú hodnotu). Do položky `data` načítame údaje z CSV súboru pomocou funkcie `csvRead` prezentovanej na začiatku tejto kapitoly. Z algoritmu vyplýva, že pre funkčnosť programu je nevyhnutné funkciu `csvRead` nakopírovať do každého podpriečinka v hlavnom priečinku. Na záver pridáme štruktúru `Data` do poľa štruktúr `ExportData`.

```
1  function AllRawData = CollectRawData(FolderNames)
2
3  AllRawData = [];
4
5  for i = 1:length(FolderNames)
6      PersonRawData = csvExport(FolderNames{i});
7      AllRawData = [AllRawData, PersonRawData];
8  end
9
10 end
```

```
1  function ExportData = csvExport(FolderName)
2
```

```
3  cd(FolderName)
4
5  dir_files = dir('*.csv');
6  ExportData = [];
7
8  for i = 1:length(dir_files)
9      Data.name = value(FolderName);
10     Data.data = csvRead(dir_files(i).name);
11     ExportData = [ExportData, Data];
12 end
13
14 cd ..
15
16 end
```

Pretože sa projekt zaoberá biometrickou identifikáciou osôb na základe pohybu, z premennej `AllRawData` vyberieme namerané akcelerácie a uložíme ich do samostatnej premennej `Accelerations`. Uhly náklonu pri identifikácii neuvažujeme a nebudeme s nimi ďalej pracovať. Premenná `Accelerations` je výstupom z funkcie `CollectAccelerations`. Je typu pole štruktúr.

Algoritmus funkcie `CollectAccelerations` je veľmi podobný tomu vo funkcii `CollectRawData` a `csvExport`. Premennú `Accelerations` vytvoríme v úvode programu ako prázdne pole. Hodnoty jej pridávame využitím for cyklu a pomocnej premennej `Acc` typu štruktúra. Do položky `name` štruktúry `Acc` zapíšeme meno dobrovoľníka, od ktorého pochádzajú akcelerácie z premennej `AllRawData`. V ďalšom príkaze z položky `data` tej istej premennej vyberieme všetky riadky (znak dvojbodka) štvrtého až šiesteho stĺpca matice a vložíme ich do položky rovnakého názvu štruktúry `Acc`. Štvrtý stĺpec obsahoval akcelerácie merané pozdĺž osi X, piaty stĺpec pozdĺž osi Y a šiesty stĺpec pozdĺž osi Z. Matica akcelerácii v položke `data` je rozmeru $n \times 3$. V poslednom kroku pridáme štruktúru `Acc` do poľa štruktúr

`Accelerations`.

```
function Accelerations = CollectAccelerations(AllRawData)

Accelerations = [];

for i = 1:length(AllRawData)
    Acc.name = AllRawData(i).name;
    Acc.data = AllRawData(i).data(:,4:6);
    Accelerations = [Accelerations, Acc];
end

end
```

Kapitola 4

Extrakcia vlastností

Akcelerácie namerané pozdĺž všetkých troch osí X, Y a Z sa nachádzajú vo forme matíc v poli štruktúr `Accelerations`. Každá položka `data` tohto poľa štruktúr obsahuje záznam jedného merania - maticu akcelerácií rozmeru $n \times 3$. Z uvedeného rozmeru je zrejmé, že akcelerácie zaznamenané pozdĺž ľubovoľnej osi sú zoradené do stĺpca. Počet riadkov n sa v jednotlivých maticiach mení, pretože závisí od dĺžky trvania merania. Matice umiestnené v premennej `Accelerations` majú počet riadkov z intervalu $\langle 30, 135 \rangle$, pričom priemerná hodnota n je 61. Polia, teda matice s uvedenými počtami riadkov už považujeme za rozmerné.

Pri použití premenných veľkých rozmerov ako vstup do ľubovoľného algoritmu vzniká niekoľko neželaných efektov. S pribúdajúcou veľkosťou dát sa zvyšuje množstvo potrebnej operačnej pamäte, narastá tiež výpočtová zložitosť. Znižuje sa rýchlosť a naopak zvyšuje čas spracovania. Vyššie uvedené problémy sprevádzajúce analýzu však môžeme odstrániť vhodnou úpravou vstupných údajov. V tejto práci využijeme na úpravu akceleračných matíc metódu extrakcie vlastností [16].

Extrakcia vlastností je dej alebo proces, pri ktorom dochádza k transformácii pôvodných nameraných údajov na súbor tzv. vlastností. Tieto vlastnosti pred-

stavujú redukovanú formu pôvodných údajov, sú určitou reprezentáciu originálu. Napriek tomu, že extrahované vlastnosti priamo charakterizujú dáta, z ktorých pochádzajú, obsahujú z nich iba relevantné informácie. Práve preto sa používajú ako vstup do daného algoritmu namiesto pôvodných nameraných údajov. Výhoda spočíva najmä v tom, že zachováme žiadanú presnosť výsledkov a zároveň dosiahneme potrebnú rýchlosť analýzy. Je nevyhnutné dodať, že extrakcia vlastností nie je jediná konkrétna metóda. Pod extrakciu vlastností spadá veľké množstvo rôznych metód, z ktorých každá pracuje na inom výpočtovom princípe a extrahuje iný druh vlastností [16].

V tejto práci na matice akcelerácii aplikujeme dve rôzne metódy extrakcie vlastností. Prvá z metód je založená na výpočte autokorelačnej matice Fourierových vlastností. Druhá z metód je založená na výpočte energie a ďalších dvoch kvantitatívnych parametrov - mobility a zložitosti.

4.1 Extrakcia vlastností založená na výpočte autokorelačnej matice

Aplikovaním tejto metódy extrakcie na pôvodné trojdimenzionálne (3D) akcelerometrické signály dostávame sadu vlastností nemenných voči rotácii. To znamená, že vlastnosti nebudú závisieť od natočenia osoby v priestore. Pri výpočte vlastností sa zameriavame na frekvenčné informácie vo vnútri akcelerometrických signálov. Informácie o frekvenciách získame spracovaním vstupných signálov Rýchlou Fourierovou transformáciou (skrátene FFT od Fast Fourier Transform). Rýchla Fourierova transformácia je základným algoritmom tejto metódy.

Opíšme túto metódu tak, ako bola navrhnutá v práci [17].

Majme časovo závislý 3D akcelerometrický signál s označením $s(t)$ a vektor ω obsahujúci konkrétne, nami zvolené hodnoty frekvencií. Na vstupný akcelerometrický signál budeme opakovane aplikovať metódu FFT pre každú frekvenciu z vektora ω. Počet zvolených frekvencií a zároveň počet opakovaní integrálu je n.

$$f(\omega) = \int exp(-i\omega t)s(t)dt \in \mathbb{C}^3 \tag{4.1}$$

$$F = [f(\omega_1), f(\omega_2), \ldots, f(\omega_n)] \in \mathbb{C}^{3xn} \tag{4.2}$$

Aplikovaním FFT metódy transformujeme vstupný akcelerometrický signál s na frekvenčnú doménu f. Spojením n frekvenčných domén získame maticu komplexných Fourierových vlastností F. Rozmer matice F je $3 \times n$.

Autokorelačnú maticu R Fourierových vlastností F vypočítame podľa nasledujúceho vzťahu:

$$R = F^*F \in \mathbb{C}^{nxn} \tag{4.3}$$

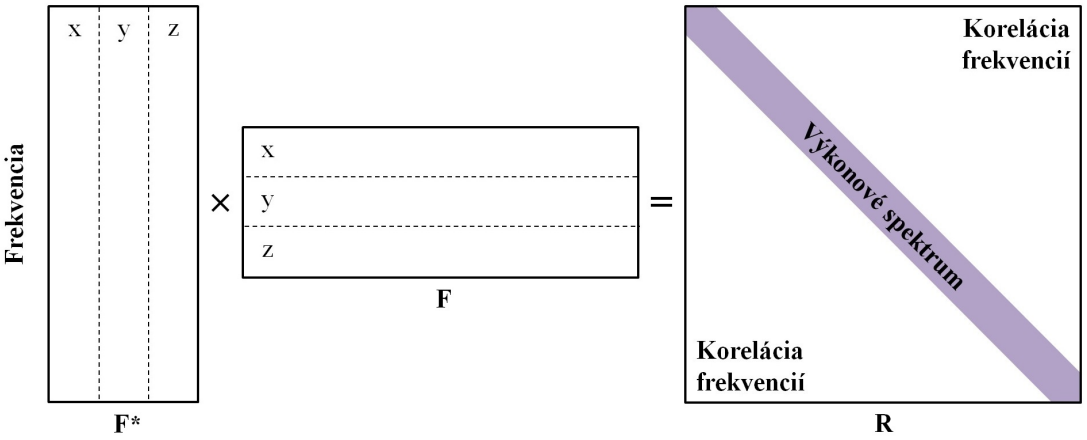
F je vo vzťahu matica komplexných Fourierových vlastností a F^* jej komplexná konjugovaná transpozícia. Vznik autokorelačnej matice R ilustruje tiež obrázok 4.1.

Autokorelačná matica R je symetrická s rozmerom $n \times n$.

Prvky na diagonále v matici R sú reálne čísla a korešpondujú s výkonovým spektrom.

$$R_{jj} = f(\omega_j)^* f(\omega_j) = \sum_{d \in \{x,y,z\}} |f_d(\omega_j)|^2 \in \mathbb{R} \tag{4.4}$$

Naopak, prvky nachádzajúce sa mimo diagonály sú komplexné čísla a predstavujú korelácie medzi všetkými frekvenčnými komponentami.



Obr. 4.1: Autokorelačná matica R Fourierových vlastností F

$$R_{jk} = f(\omega_j)^* f(\omega_k) = \sum_d f_d(\omega_j)^* f_d(\omega_k) \in \mathbb{C} \tag{4.5}$$

Z dôvodu, že autokorelačná matica R je symetrická, budeme uvažovať iba jej horný trojuholník vrátane diagonály. Horný trojuholník v absolútnej hodnote prevedieme do vektorového tvaru. Vektor z je potom vektorom extrahovaných, voči rotácii nemenných vlastností.

$$z = vector(upper(|R_{ij}|)) \tag{4.6}$$

4.1.1 Implementácia algoritmu v Matlab-e

Vektor extrahovaných vlastností z získame v Matlab-e spustením funkcie s názvom `FE_fft_correlation`. Funkcia má tri vstupné parametre, ktorých prehľad ponúka tabuľka 4.1.

Premenná `s` je matica rozmeru $n \times 3$, to znamená, že akcelerácie namerané pozdĺž jednej osi sú usporiadané do stĺpca (pre tri osi X, Y a Z existujú tri stĺpce).

Tabuľka 4.1: Prehľad vstupných argumentov funkcie FE_fft_correlation

Vstupný argument	Popis
`s`	3D akcelerometrický signál
`W`	vektor zvolených frekvencií
`Ts`	perióda vzorkovania

Vektor frekvencií `W` má rozmer $1 \times m$, kde m je počet zvolených frekvencií. Perióda vzorkovania `Ts` je skalár.

V tejto práci volíme nasledovné hodnoty premenných `W` a `Ts`:

- $W = [0.01, 0.1, 1, 10, 100]$;
- $Ts = 0.5$;

Pri použití piatich rôznych frekvencií bude vektor `z`, výstup z funkcie `FE_fft_correlation` obsahovať 15 vlastností.

V úvode programu uložíme do premennej `ns` počet sĺpcov premennej `s`, ktorá predstavuje 3D akcelerometrický signál. Ďalším príkazom vytvoríme nulovú maticu `F`. Počet riadkov tejto matice je `ns`, počet stĺpcov sa rovná počtu prvkov vektora `W` (príkaz `numel(W)`), teda počtu zvolených frekvencií.

Nuly v matici komplexných Fourierových vlastností `F` nahrádzame frekvenčnými doménami pomocou vnorených for cyklov. V prvom for cykle priradíme premennej `w` jeden prvok vektora `W` - jednu frekvenciu, s ktorou potom pracujeme. V druhom a treťom for cykle pre danú frekvenciu vypočítame integrál Rýchlej Fourierovej transformácie zjednodušene ako sumu. Výraz v integrále vyhodnotíme pre každý čas `t` a hodnoty všetkých výrazov sčítame. Algoritmus je navrhnutý tak,

že matica `F` sa plní po stĺpcoch. Najskôr prepíšeme nulu v prvom riadku prvého stĺpca, potom nulu v druhom a treťom riadku prvého stĺpca. Po naplnení celého stĺpca sa do premennej `w` načíta nová hodnota frekvencie a celý proces sa zopakuje.

Autokorelačnú maticu `R` Fourierových vlastností `F` dostávame vynásobením matice `F` a jej komplexnej konjugovanej transpozície. Príkazom `Ra = abs(R);` prevedieme všetky prvky matice `R` na ich absolútne hodnoty. V závere programu vytvoríme vektor `z` ako prázdne pole a naplníme ho horným trojuholníkom matice `R` (vrátane diagonály). Plnenie opäť prebieha cez vnorené for cykly.

```
1  function z = FE_fft_correlation(s, W, Ts)
2
3  ns = size(s, 2);
4  F = zeros(ns, numel(W));
5
6  for k = 1:length(W)
7      w = W(k);
8      for is = 1:ns
9          for t = 1:size(s, 1)
10             F(is, k) = F(is, k) + exp(-j*w*(t-1)*Ts)*s(t, is);
11         end
12     end
13 end
14
15 R = transpose(conj(F))*F;
16 Ra = abs(R);
17
18 z = [];
19 for ir = 1:size(Ra, 1)
20     for ic = ir:size(Ra, 2)
21         z = [z; Ra(ir, ic)];
```

```
22      end
23  end
24
25  end
```

4.2 Extrakcia vlastností založená na výpočte energie, mobility a zložitosti

Výsledkom druhej metódy extrakcie sú vlastnosti, ktoré sú kompenzované voči náklonom a gravitácii. Medzi tieto vlastnosti patrí energia a dva kvantitatívne parametre nazvané mobilita a zložitosť. Každú z týchto vlastností uvažujeme vo vertikálnom aj horizontálnom smere, čím získavame sadu 6 vlastností.

Postup, akým vypočítame uvedené vlastnosti je uvedený v práci [18].

Majme maticu a, ktorá predstavuje časovo závislý 3D akcelerometrický signál. Matica a je rozmeru $3 \times n$. Prvý riadok tejto matice obsahuje akcelerácie namerané pozdĺž osi X, druhý riadok pozdĺž osi Y a tretí riadok pozdĺž osi Z.

$$a(t) = [a_x(t), a_y(t), a_z(t)]^T \tag{4.7}$$

Keď spriemerujeme každý riadok matice a, získame vektor b rozmeru 3×1. V prvom riadku vektora b sa bude nachádzať priemerná hodnota akcelerácii nameraných pozdĺž osi X, v druhom pozdĺž osi Y a v treťom riadku pozdĺž osi Z.

$$b(t) = \frac{1}{N} \sum_{t=1}^{N} a(t) \tag{4.8}$$

Z prvkov vektora b vypočítame náklonové uhly θ_1 a θ_2 pomocou nasledovných vzťahov:

$$\theta_1 = \arctan\left(\frac{b_y}{b_z}\right) \tag{4.9}$$

$$\theta_2 = \arctan\left(\frac{b_x}{b_y \sin\theta_1 + b_z \cos\theta_1}\right) \tag{4.10}$$

Uhly θ_1 a θ_2 ako aj maticu akcelerácii a použijeme na výpočet akceleračnej matice a_1 kompenzovaného voči náklonu.

$$a_1(t) = \begin{bmatrix} \cos\theta_2 & -\sin\theta_1 \sin\theta_2 & -\cos\theta_1 \sin\theta_2 \\ 0 & \cos\theta_1 & -\sin\theta_1 \\ \sin\theta_2 & \sin\theta_1 \cos\theta_2 & cos\theta_1 \cos\theta_2 \end{bmatrix} \times a(t) \tag{4.11}$$

Akceleračná matica a_1 je kompenzovaná voči náklonu, stále však zahŕňa gravitačný efekt. Preto vykonáme pozdĺž osi Z korekciu tohto efektu. Dostávame tým okamžité akcelerácie kompenzované voči náklonu a gravitácii zároveň.

$$a_2(t) = \left[a_{1x}(t), a_{1y}(t), \left(a_{1z}(t) - \frac{1}{N}\sum_{t=1}^{N} a_{1z}(t)\right)\right]^T \tag{4.12}$$

Posledným krokom tejto časti je výpočet vertikálnej a horizontálnej akcelerácie.

$$a_v(t) = a_{2z}(t) \tag{4.13}$$

$$a_h(t) = \sqrt{a_{2x}(t)^2 + a_{2y}(t)^2} \tag{4.14}$$

Vertikálna a horizontálna akcelerácia sú východiskové hodnoty pre extrakciu 6 vlastností - vertikálnej energie, horizontálnej energie, vertikálnej mobility, horizontálnej mobility, vertikálnej zložitosti a horizontálnej zložitosti.

Vertikálnu energiu vypočítame ako určitý integrál z absolútnej hodnoty vertikálnej akcelerácie. Horizontálnu energiu vypočítame ako určitý integrál z absolútnej hodnoty horizontálnej akcelerácie.

$$e_v = \int_{t-t_0}^{t_0+\tau} |a_v| dt \tag{4.15}$$

$$e_h = \int_{t-t_0}^{t_0+\tau} |a_h| dt \tag{4.16}$$

Vlastnosti vertikálna a horizontálna energia vyjadrujú množstvo vykonanej aktivity. Je treba poznamenať, že hodnoty oboch týchto energií závisia od fyziologických faktorov ako napr. telesná hmotnosť, preto sa líšia od osoby k osobe.

Ďalšie dve, resp. štyri extrahované vlastnosti sú kvantitatívne parametre nazývané Hjorthova mobilita a zložitosť. Tieto parametre boli navrhnuté pre účely diagnostiky epilepsie. V tejto oblasti slúžia na vyhodnocovanie elektroencefalogramov (EEG), čo sú krivky (signály) reprezentujúce elektrickú aktivitu mozgu [19]. Dajú sa však použiť aj pri spracovaní iných typov signálov, v našom prípade akcelerometrických.

Hjorthova mobilita je druhá odmocnina z pomeru medzi varianciami prvej derivácie a amplitúdy. Opisuje tvar krivky meraním relatívneho priemerného náklonu.

Hjorthova zložitosť je pomer medzi mobilitou prvej derivácie signálu a mobilitou samotného signálu. Meria nepravidelnosť frekvenčnej domény.

4.2.1 Implementácia algoritmu v Matlab-e

Všetky vyššie uvedené vlastnosti - vertikálnu a horizontálnu energiu, mobilitu a zložitosť - nájde funkcia `FE_EMC`. Funkcia má dva vstupné parametre - premenné `a0` a `Ts`. Prehľad vstupných parametrov funkcie `FE_EMC` nájdeme v tabuľke 4.2

Tabuľka 4.2: Prehľad vstupných argumentov funkcie FE_EMC

Vstupný argument	Popis
`a0`	3D akcelerometrický signál
`Ts`	perióda vzorkovania

Premenná `a0` je matica rozmeru $3 \times n$, to znamená, že akcelerácie namerané pozdĺž jednej osi sú usporiadané do riadka (pre tri osi X, Y a Z existujú tri riadky). Perióda vzorkovania `Ts` je skalár.

Počas všetkých opakovaní funkcie sa perióda vzorkovania bude rovnať 0.5.

Algoritmus funkcie je založený na matematických vzťahoch uvedených v tejto podkapitole. Z 3D akcelerometrického signálu `a0` podľa rovnice (4.8) vytvoríme vektor `b`. Každý riadok matice `a0` spriemeruje funkcia `mean`. Náklonové uhly θ_1 a θ_2 vypočítame podľa vzťahov (4.9) a (4.10). Maticu `a1` kompenzovanú voči náklonu získame vynásobením matice `M` a matice `a0`. Maticu kompenzovanú voči náklonu aj gravitácii `a2` dostávame z matice `a1` podľa rovnice (4.12). Z matice `a1` prevezmeme prvý aj druhý riadok a vložíme ho do matice `a2`. Iba tretí riadok je rozdielny. Vznikne odčítaním priemernej hodnoty riadka od každého prvku riadka. Priemernú hodnotu opäť počítame cez funkciu `mean`. Matice `a1` a `a2` sú rovnako ako matica `a0` rozmeru $3 \times n$. Do premennej `av` vložíme tretí riadok matice `a2` a získame tak vertikálnu akceleráciu. Horizontálnu akceleráciu vypočítame podľa vzťahu (4.14). Každý prvok prvého aj druhého riadka matice `a2` umocníme na

druhú a sčítame medzi sebou. Z každého súčtu spravíme odmocninu. Premenné `av` a `ah` sú rozmeru $1 \times n$. Integrály použité na výpočet vertikálnej a horizontánej energie (rovnice (4.15) a (4.16)) nahradíme sumou. Absolútnu hodnotu každého prvka vektora `av` (`ah`) vynásobíme periódou vzorkovania `Ts` a súčiny spočítame. Hodnoty vertikálnej a horizontálnej mobility a zložitosti získame spustením vnorenej funkcie `HjorthParameters`. Konečným výstupom z funkcie `FE_EMC` je vektor `v` obsahujúci všetky vlastnosti.

```
1  function v = FE_EMC(a0,Ts)
2
3  b = mean(a0, 2);
4
5  O1 = atan(b(2)/b(3));
6  O2 = atan(b(1)/(b(2)*sin(O1)+b(3)*cos(O1)));
7
8  M = [cos(O2) -sin(O1)*sin(O2) -cos(O1)*sin(O2);
9       0 cos(O1) -sin(O1);
10      sin(O2) sin(O1)*cos(O2) cos(O1)*cos(O2)];
11
12 a1 = M*a0;
13
14 a2 = [a1(1,:);
15       a1(2,:);
16       a1(3,:) - mean(a1(3,:))];
17
18 av = a2(3,:); % vertical acceleration
19 ah = sqrt(a2(1,:).^2 + a2(2,:).^2); % horizontal acceleration
20
21 ev = sum(abs(av)*Ts); % vertical energy
22 eh = sum(abs(ah)*Ts); % horizontal energy
23
```

```
24 [mv,cv] = HjorthParameters(av'); % vertical mobility & complexity
25 [mh,ch] = HjorthParameters(ah'); % horizontal mobility & complexity
26
27 v = [ev;eh;mv;mh;cv;ch];
28
29 end
```

Jediným vstupným parametrom do vnorenej funkcie `HjorthParameters` je vektor vertikálnej alebo horizontálnej akcelerácie, premenná `xV`. Výstupom z funkcie je vertikálna alebo horizontálna mobilita a zložitosť (závisí od vstupného argumentu).

V prvých dvoch príkazoch programu využívame funkciu `diff`. Táto funkcia zisťuje rozdiely medzi prvkami vektorov nasledujúcimi za sebou [20]. Na začiatok vstupného vektora `xV` pridáme nulu a pomocou funkcie `diff` vypočítame rozdiely medzi jednotlivými prvkami tohto vektora. Tak vznikne vektor `dxV`. Tú istú operáciu vykonáme s vektorom `dxV`. Vznikne nový vektor `ddxV`. Každý prvok vektora `xV` umocníme na druhú a z druhých mocnín vypočítame priemer pomocou funkcie `mean`. Operáciu znovu opakujeme pre vektory `dxV` a `ddxV`. Premennú `mob` získame vydelením premenných `mdx2` a `mx2`. Zložitosť (premenná `complexity`) je potom druhá odmocnina výrazu v zátvorke, kde od pomeru premenných `mddx2` a `mdx2` odčítame premennú `mob`. Mobilita (premenná `mobility`) je druhou odmocninou z premennej `mob`.

```
1 function [mobility,complexity] = HjorthParameters(xV)
2
3 dxV = diff([0;xV]);
4 ddxV = diff([0;dxV]);
5 mx2 = mean(xV.^2);
6 mdx2 = mean(dxV.^2);
```

```
7  mddx2 = mean(ddxV.^2);
8
9  mob = mdx2 / mx2;
10 complexity = sqrt(mddx2 / mdx2 - mob);
11 mobility = sqrt(mob);
12
13 end
```

4.3 Hromadná extrakcia vlastností z akcelerácií

Extrakciu vlastností z akcelerácii oboma metódami zabezpečí v jednom kroku funkcia `FeatureExtraction`. Táto funkcia má 3 vstupné a 2 výstupné argumenty. Prehľad vstupných argumentov uvádza tabuľka 4.3.

Tabuľka 4.3: Prehľad vstupných argumentov funkcie FeatureExtraction

Vstupný argument	Popis
`Accelerations`	3D akcelerometrické signály
`freq`	vektor zvolených frekvencií
`Ts`	perióda vzorkovania

Pole štruktúr `Accelerations` sme vytvorili v závere podkapitoly 3.3. V položke `name` premennej `Accelerations` sa nachádza meno dobrovoľníka, ktorému patrí matica akcelerácií v položke `data`. Pole štruktúr je rozmeru $1 \times n$, kde n je počet všetkých realizovaných meraní (počet CSV súborov vo všetkých podpriečinkoch). Rozmery a hodnoty premenných `freq` a `Ts` sa nemenia, nájdeme ich v časti 4.1.1.

Výstupnými argumentami z funkcie `FeatureExtraction` sú premenné `FEemc` a `FEstat`. Obe tieto premenné sú typu pole štruktúr. Každá obsahuje dvojicu polo-

žiek `name` a `data`. Premenná `FEstat` bude po spustení funkcie uchovávať vlastnosti extrahované pomocou prvej metódy (podkapitola 4.1) a premenná `FEemc` vlastnosti extrahované pomocou druhej metódy (podkapitola 4.2). Vektory vlastností obsahuje položka `data`. Pomocou prvej metódy získame vektor 15 vlastností, pomocou druhej metódy vektor 6 vlastností.

Pri hromadnej extrakcii vlastností využívame funkcie `FE_fft_correlation` a `FE_EMC`.

V prvom kroku vytvoríme výstupné premenné `FEemc` a `FEstat` ako prázdne polia. Polia plníme osobitne v 2 for cykloch identickým algoritmom. Do položky `name` pomocnej štruktúry `v` aj `z` načítame z premennej `Accelerations` meno dobrovoľníka. Do premenných `a0` a `s` uložíme maticu akcelerácii - rovnako z premennej `Accelerations`. Keďže vnorená funkcia `FE_EMC` vyžaduje ako vstup maticu rozmeru $3 \times n$, v prvom for cykle využijeme transpozíciu. V ďalšom kroku načítame do položky `data` premennej `v` a `z` vektor extrahovaných vlastností. Extrakciu vykonáme pomocou vnorených funkcií `FE_fft_correlation` a `FE_EMC`. Hotovú štruktúru pridáme do poľa štruktúr `FEemc` a `FEstat`. Postup opakujeme pre každú akceleračnú maticu umiestnenú v premennej `Accelerations`.

```
1 function [FEemc, FEstat] = FeatureExtraction(Accelerations, Ts,
2 freq)
3
4 FEemc = [];
5 FEstat = [];
6
7 for i = 1:length(Accelerations)
8     v.name = Accelerations(i).name;
9     a0 = Accelerations(i).data';
```

```
10      v.data = FE_EMC(a0, Ts);
11      FEemc = [FEemc, v];
12  end
13
14  for i = 1:length(Accelerations)
15      z.name = Accelerations(i).name;
16      s = Accelerations(i).data;
17      z.data = FE_fft_correlation(s, freq, Ts);
18      FEstat = [FEstat, z];
19  end
20
21  end
```

Kapitola 5

Separácia

Postupom opísaným v kapitole 4 sme pôvodné akcelerometrické signály transformovali na vektory vlastností, ktoré majú vhodnú veľkosť na to, aby vstúpili do procesu identifikácie.

Identifikácia osoby na základe pohybu je založená na inom princípe ako napr. identifikácia na základe odtlačku prstu (prípadne DNA). Identifikácia na základe odtlačku prstu prebieha nasledovne. Danej osobe odoberieme vzorku odtlačku prstu. Následne porovnáme neznámy odtlačok s odobratou vzorkou. Vyhodnocujeme zhodu alebo naopak nezhodu. V prípade, že je neznámy odtlačok zhodný s odobratou vzorkou, môžeme povedať, že sme osobu identifikovali porovnaním odtlačkov prstov. Takýto postup je možný vtedy, keď sa daný znak počas života nemení.

Pri identifikácii na základe pohybu musíme zvoliť iný postup. Dôvodom je to, že človek nikdy dvakrát za sebou nevykoná identický pohyb. Na rozdiel od odtlačku prstu, pohyb sa stále mení, aj keď oko túto zmenu nemusí postrehnúť. Preto nie je jediná vzorka pohybu postačujúca, aj keď sa očakáva, že sa každý človek pohybuje jedinečným spôsobom. Identifikácia bude preto prebiehať použitím

všetkých nazbieraných vzoriek. Nástrojom na identifikáciu bude v tomto projekte separácia.

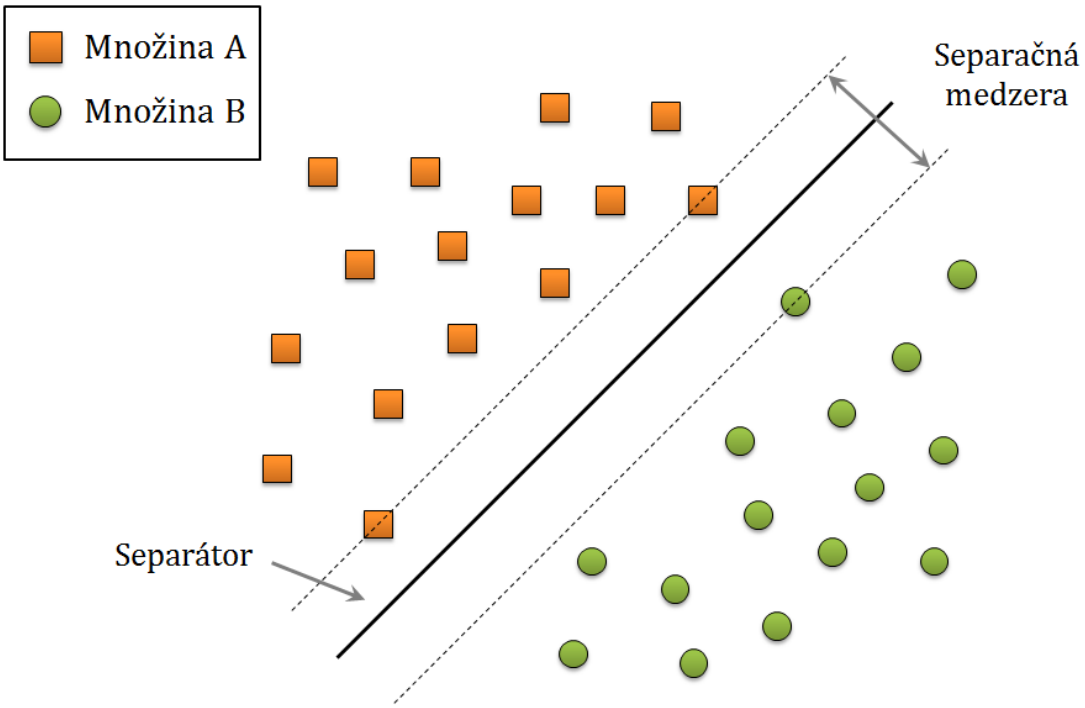
Najskôr uveďme, aký je vzťah medzi identifikáciou a separáciou. Majme jeden vektor vlastností ako výsledok jedného merania. Tento vektor obsahuje buď 6 alebo 15 vlastností, podľa toho, ktorá metóda bola využitá na extrakciu vlastností. Z matematického hľadiska predstavuje jeden vektor vlastností (jedno meranie) jeden bod v priestore. Potom sú jednotlivé vlastnosti ako prvky tohto vektora súradnice, ktoré daný bod definujú. Všetky vektory alebo merania pochádzajúce od jedného dobrovoľníka vytvárajú množinu bodov (v 6 alebo 15-rozmernom priestore). Od 6 dobrovoľníkov získavame 6 rôznych množín bodov. Prvým krokom k identifikácii osoby je oddelenie alebo odseparovanie jej množiny bodov od všetkých ostatných množín v danom priestore. Práve toto vykoná separácia.

Separácia je teda algoritmus, ktorý slúži na analýzu a binárnu klasifikáciu údajov. Je to proces, pri ktorom dochádza k oddeleniu práve dvoch množín bodov (binárna klasifikácia) tak, že medzi ne umiestnime lineárny separátor. Príklad separácie v 2-rozmernom priestore vidíme na obrázku 5.1. Na tomto príklade si vysvetlime separáciu aj identifikáciu.

Majme 2 konkrétne množiny bodov A a B. Rozdeľme každú z týchto množín na dve časti - na väčšiu, trénovaciu a menšiu, testovaciu. Predpokladajme, že body v menšej časti sú naozaj neznáme, t.j. nevieme, do ktorej množiny patria. Trénovacie časti množín sa použijú na vytvorenie alebo natrénovanie (odtiaľ pochádza výraz trénovacie) lineárneho separátora. Tieto časti nám povedia, kde v priestore sa bude nachádzať separátor oddeľujúci množiny A a B. To je krok separácie. Teraz umiestnime zvyšné body z testovacej časti do priestoru, kde sa nachádzajú trénovacie body aj lineárny separátor. Máme dve možnosti. Každý bod sa zaradí

na pravú alebo naopak ľavú stranu separátora. Ak sa bod zaradí na pravú stranu, vieme, že patrí do množiny B. Ak sa bod zaradí na ľavú stranu, patrí do množiny A. To je krok identifikácie. Identifikácia vždy prebieha na základe natrénovaného separátora.

Existuje viacero druhov separácii, všetky však fungujú na rovnakom princípe uvedenom vyššie. V ďalších podkapitolách sa budeme bližšie venovať lineárnej, robustnej, približnej separácii a Support Vector Machine [21]. Posledný druh separácie, Support Vector Machine potom využijeme na biometrickú identifikáciu osôb na základe pohybu.



Obr. 5.1: Príklad separácie dvoch množín bodov

5.1 Lineárna separácia

Lineárna separácia predstavuje najjednoduchší typ separácie. Práve ona položila základy pre vznik ďalších, zložitejších druhov sepáracií ako napr. robustná alebo približná.

Lineárna separovateľnosť je geometrická vlastnosť skupiny bodov. Demonštrujme túto vlastnosť na bodoch v dvojrozmernom priestore. Máme dve množiny bodov $\{x_1, x_2, \ldots, x_N\}$ a $\{y_1, y_2, \ldots, y_M\}$, pričom každý bod z oboch množín definujú práve dve súradnice. Body z prvej množiny si môžeme predstaviť ako oranžové štvorce a body z druhej množiny ako zelené kruhy (obrázok 5.1). Tieto množiny bodov sú lineárne separovateľné vtedy, ak existuje aspoň jedna priamka, ktorá v priestore oddelí oranžové štvorce a zelené kruhy tak, že všetky oranžové štvorce sa budú nachádzať na jednej strane priamky a všetky zelené kruhy na strane druhej [22]. Priamku oddeľujúcu dve množiny bodov nazývame lineárny separátor.

Lineárna separácia je optimalizačný problém, pri ktorom skúmame, či sú dve množiny bodov lineárne separovateľné. Ak je podmienka splnená, zároveň hľadáme také koeficienty lineárneho separátora $a \in \mathbb{R}^n$ a $b \in \mathbb{R}$, že budú spĺňať nasledujúce ohraničenia:

$$a^T x_i + b > 0, i = 1, \ldots, N \tag{5.1}$$

$$a^T y_j + b < 0, j = 1, \ldots, M \tag{5.2}$$

Koeficienty separátora a a b považujeme za optimalizované premenné (hľadáme ich hodnoty). Premenná a je stĺpcový vektor. Počet riadkov vektora a sa musí rovnať počtu súradníc jednotlivých bodov z oboch množín. Premenná b je skalár. x_i je i-ty bod z prvej množiny bodov a y_j j-ty bod z druhej množiny bodov.

Z rovníc (5.1) a (5.2) je zrejmé, že množiny môžu mať rozdielny počet bodov. Prvá množina má N a druhá M bodov.

Na ohraničeniach (5.1) a (5.2) si ďalej treba všimnúť, že obsahujú ostré nerovnosti, ktoré môžu spôsobovať problémy pri numerickom riešení optimalizačného problému. Z toho dôvodu preformulujeme pôvodné ostré ohraničenia na neostré. Vytvoríme si novú premennú epsilon, ktorej hodnotu zvolíme ako veľmi malé kladné číslo (napr. 10^{-9}). Budeme predpokladať, že keď je výraz vľavo väčší alebo rovný ako veľmi malé číslo, je zároveň ostro väčší ako nula. Podobný predpoklad zvolíme aj pre druhú nerovnosť.

$$a^T x_i + b \geq \varepsilon, i = 1, \ldots, N \tag{5.3}$$

$$a^T y_j + b \leq -\varepsilon, j = 1, \ldots, M \tag{5.4}$$

Epsilon, dosadený do pravých strán neostrých ohraničení chápeme ako toleranciu. V ďalšom kroku obe ohraničenia vydelíme epsilonom, čím na pravých stranách nerovníc dostávame hodnoty 1 a -1. Napriek tomu, že ľavá strana zostáva formálne nezmenená, v skutočnosti je po vydelení konštanta epsilon zahrnutá v premenných a a b.

$$a^T x_i + b \geq 1, i = 1, \ldots, N \tag{5.5}$$

$$a^T y_j + b \leq -1, j = 1, \ldots, M \tag{5.6}$$

Lineárna separácia je typická tým, že hľadáme ľubovoľný lineárny separátor vyhovujúci daným ohraničeniam. To znamená, že riešime optimalizačný problém bez účelovej funkcie. Po zavedení novej premennej z zapíšeme takýto problém

nasledovne:

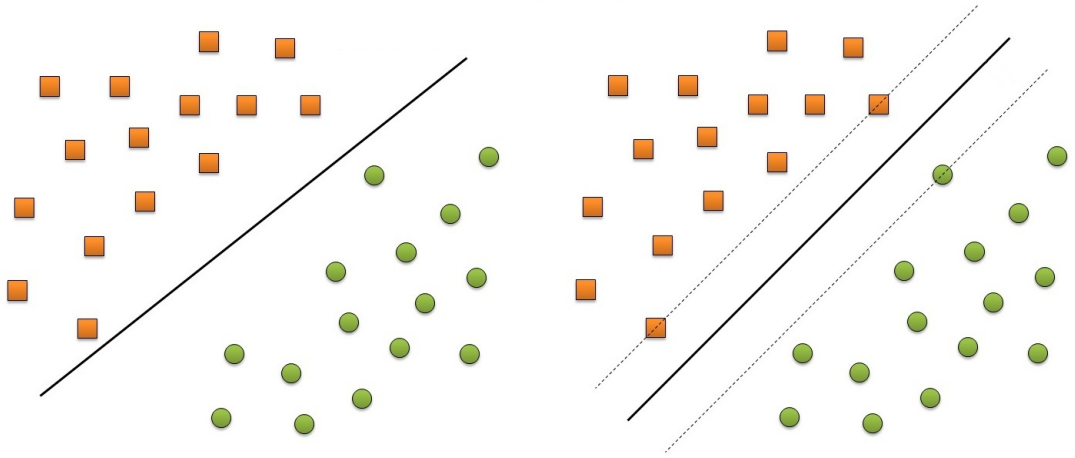
$$\min\ 0^T z \tag{5.7a}$$

$$s.t.\ -[x_i^T, 1]z \leq -1, j = 1, \ldots, N \tag{5.7b}$$

$$[y_j^T, 1]z \leq -1, j = 1, \ldots, M \tag{5.7c}$$

5.2 Robustná separácia

Robustná separácia je podobná lineárnej sepáracii. V lineárnej separácii hľadáme ľubovoľný separátor vyhovujúci zadaným ohraničeniam. V robustnej separácii hľadáme taký separátor, ktorý vyhovie zadaným ohraničeniam a súčasne vytvorí čo najširšiu separačnú medzeru medzi dvoma množinami bodov. Robustná separácia je teda optimalizačný problém, v ktorom maximalizujeme vzdialenosť separátora od bodov oboch množín.



Obr. 5.2: Lineárna vs. robustná separácia

Ohraničenia zostávajú zachované z lineárnej separácie:

$$a^T x_i + b > 0, i = 1, \ldots, N \tag{5.8}$$

$$a^T y_j + b < 0, j = 1, \ldots, M \tag{5.9}$$

Pri prevode ostrých ohraničení na neostré použijeme rovnaký postup ako v predchádzajúcej časti.

$$a^T x_i + b \geq 1, i = 1, \ldots, N \tag{5.10}$$

$$a^T y_j + b \leq -1, j = 1, \ldots, M \tag{5.11}$$

Maximalizáciu vzdialenosti separátora od bodov vyjadríme v matematike výrazom $^2/_{||a||_2}$. Člen $||a||_2$ voláme 2-norma. 2-norma alebo tiež Euklidovská norma meria dĺžku vektora v n-rozmernom Euklidovskom priestore $\mathbb{R}^n$. Dáva nám vzdialenosť od začiatku Euklidovského priestoru (začiatok vektora) po ľubovoľný bod v priestore (koniec vektora). 2-norma je dôsledok Pytagorovej vety [23]. Definujeme ju vzťahom:

$$||a||_2 = \sqrt{\sum_{i=1}^{n} |a_i|^2} = \left(\sum_{i=1}^{n} |a_i|^2 \right)^{\frac{1}{2}} \tag{5.12}$$

Výraz vyjadrujúci maximalizáciu vzdialenosti separátora od bodov, resp. minimalizáciu prevrátenej hodnoty, ktorá má tvar $^{||a||_2}/_2$ pridáme do účelovej funkcie. Vzniká nasledovný optimalizačný problém:

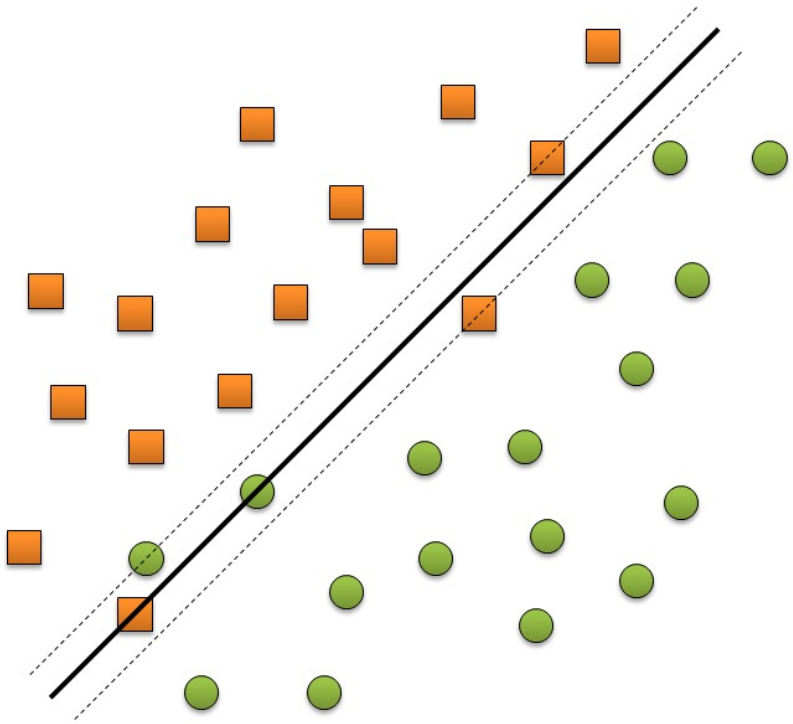
$$\min \frac{1}{2}||a||_2 \tag{5.13a}$$

$$s.t.\ a^T x_i + b \geq 1, i = 1, \ldots, N \tag{5.13b}$$

$$a^T y_j + b \leq -1, j = 1, \ldots, M \tag{5.13c}$$

5.3 Približná separácia

Ďalším typom separácie je približná separácia. Cieľom približnej separácie je nájsť taký lineárny separátor, ktorý minimalizuje počet zle klasifikovaných bodov.



Obr. 5.3: Približná separácia

Pri formulácii problému budeme vychádzať z ohraničení odvodených pri lineárnej separácii.

$$a^T x_i + b \geq 1, i = 1, \ldots, N \tag{5.14}$$

$$a^T y_j + b \leq -1, j = 1, \ldots, M \tag{5.15}$$

Tieto ohraničenia môžeme použiť v prípade, že množiny bodov sú lineárne separovateľné, t.j. dajú sa separátorom oddeliť tak, že jedna množina bodov sa bude nachádzať na jednej jeho strane a druhá na druhej. Ak množiny bodov nie sú lineárne separovateľné, optimalizačný problém je neriešiteľný, pretože neexistujú také koeficienty $a \in \mathbb{R}^n$ a $b \in \mathbb{R}$, ktoré by vyhovovali daným ohraničeniam.

Problém, kedy sa dve množiny bodov nedajú striktne oddeliť rieši práve približná separácia. V približnej separácii zmäkčíme pôvodné tvrdé ohraničenia. Zmäkčenie docielime pridaním zmäkčovacích premenných u a v do ohraničení. Tieto premenné budú optimalizované rovnako ako koeficienty separátora a a b.

$$a^T x_i + b \geq 1 - u_i, i = 1, \ldots, N \tag{5.16a}$$

$$a^T y_j + b \leq -1 + v_j, j = 1, \ldots, M \tag{5.16b}$$

$$u_i \geq 0, v_j \geq 0 \tag{5.16c}$$

Premenné $u \in \mathbb{R}^N$ a $v \in \mathbb{R}^M$ sú stĺpcové vektory. Platí, že každému bodu z prvej množiny prislúcha práve jeden prvok vektora u a každému bodu z druhej množiny prislúcha práve jeden prvok vektora v. Z tohto tvrdenia vyplýva, že počet prvkov vektora u je N a počet prvkov vektora v je M. Ďalej platí, že každý prvok vektora u aj v je číslo väčšie alebo rovné ako nula.

Vysvetlime si funkciu zmäkčovacích premenných v ohraničeniach. Predstavme si, že používame pôvodné ohraničenia z lineárnej separácie a výraz vľavo v prvej nerovnosti - $a^T x_i + b$ je rovný číslu 0.8. Výraz vľavo nie je väčší alebo rovný ako číslo 1, preto dostávame neriešiteľný problém. Teraz si predstavme, že sme zaviedli nové zmäkčené ohraničenia a výraz vľavo je znovu rovný číslu 0.8. Aby bola splnená podmienka, že ľavá strana nerovnice je väčšia alebo rovná ako pravá, nastavíme prvok vektora u prislúchajúci danému bodu na hodnotu 0.2. Zmenšením jednotky

na pravej strane o stanovenú hodnotu získavame riešiteľný problém.

Tento postup sa zopakuje pre všetky body prvej aj druhej množiny. V prípade, že je výraz vľavo väčší alebo rovný ako číslo 1, je splnené pôvodné tvrdé ohraničenie a hodnota prvku vektora u alebo v sa nastaví na nulu.

V účelovej funkcii potom penalizujeme použitie zmäkčovacích premenných, t.j. minimalizujeme počet nenulových prvkov vektorov u a v.

$$\min \sum u_i + \sum v_j \tag{5.17a}$$

$$s.t.\ a^T x_i + b \geq 1 - u_i, i = 1, \ldots, N \tag{5.17b}$$

$$a^T y_j + b \leq -1 + v_j, j = 1, \ldots, M \tag{5.17c}$$

$$u_i \geq 0, v_j \geq 0 \tag{5.17d}$$

5.4 Support Vector Machine

Support Vector Machine (SVM) je efektívnou kombináciou robustnej a približnej separácie. Pri tomto probléme hľadáme taký lineárny separátor, ktorý vytvorí medzi dvoma množinami bodov čo najširšiu separačnú medzeru pri čo najmenšom počte zle klasifikovaných bodov.

Keďže jedna z podmienok je čo najmenší počet zle klasifikovaných bodov, z približnej separácie preberáme zmäkčené ohraničenia aj účelovú funkciu, kde minimalizujeme použitie zmäkčovacích premenných u a v. Z robustnej separácie preberáme výraz vyjadrujúci maximalizáciu vzdialenosti separátora od bodov $^2/_{||a||_2}$.

Matematická formulácia potom vyzerá nasledovne:

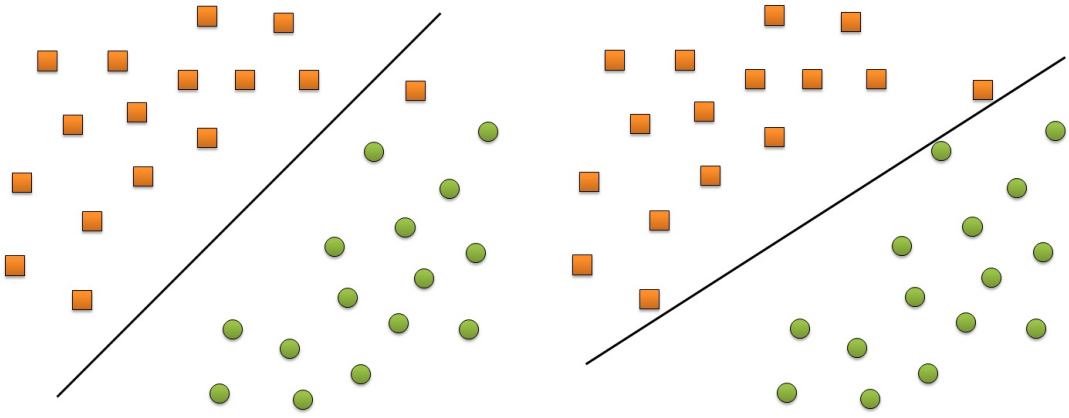
$$\min \frac{1}{2}||a||_2 + \gamma(1^T u + 1^T v) \tag{5.18a}$$

$$s.t.\ a^T x_i + b \geq 1 - u_i, i = 1, \ldots, N \tag{5.18b}$$

$$a^T y_j + b \leq -1 + v_j, j = 1, \ldots, M \tag{5.18c}$$

$$u_i \geq 0, v_j \geq 0 \tag{5.18d}$$

SVM algoritmus nám dovoľuje priradiť váhu jednotlivým členom účelovej funkcie. Pomer medzi šírkou separačnej medzery a aproximáciou počtu zle klasifikovaných bodov určuje parameter γ. γ môže nadobúdať hodnoty z intervalu $\langle 0, \infty)$.



Obr. 5.4: Veľmi malá vs. veľmi veľká hodnota parametra γ

Na záver umocníme výraz v účelovej funkcii obsahujúci 2-normu. Matema-

tická formulácia nadobudne nasledovný tvar:

$$\min \frac{1}{4}a^T a + \gamma(1^T u + 1^T v) \tag{5.19a}$$

$$s.t.\ a^T x_i + b \geq 1 - u_i, i = 1, \ldots, N \tag{5.19b}$$

$$a^T y_j + b \leq -1 + v_j, j = 1, \ldots, M \tag{5.19c}$$

$$u_i \geq 0, v_j \geq 0 \tag{5.19d}$$

5.4.1 Implementácia Support Vector Machine v Matlab-e

V tejto práci využijeme na biometrickú identifikáciu algoritmus Support Vector Machine, ktorý v sebe spája robustnú a približnú separáciu. Tento algoritmus implementujeme do Matlab-u pomocou Yalmip-u. Yalmip je modelovací jazyk určený na formuláciu a riešenie konvexných alebo nekonvexných optimalizačných problémov. Je dostupný vo forme voľne šíriteľného toolboxu [24].

Koeficienty lineárneho separátora, ktorý v n-rozmernom priestore oddelí dve množiny bodov nájde funkcia `yalmipSVM`. Funkcia má tri vstupné a dva výstupné parametre. Vstupné parametre sú zachytené v tabuľke 5.1.

Matica `M1` je rozmeru $n \times m$ a matica `M2` rozmeru $n \times l$. n je počet extrahovaných vlastností alebo tiež počet súradníc bodu, m je počet meraní alebo počet bodov patriacich do prvej množiny a l počet meraní alebo počet bodov patriacich do druhej množiny. Matica `M1` a matica `M2` musia mať rovnaký počet riadkov. V inom prípade by sme sa snažili separovať body nachádzajúce sa v rozličných priestoroch. Vstupný parameter `gamma` zvolíme ako ľubovoľné nezáporné číslo.

Výstupnými parametrami z funkcie `yalmipSVM` sú premenné `aopt` a `bopt`,ktoré predstavujú hľadané koeficienty lineárneho separátora. `aopt` je stĺpcový vektor roz-

Tabuľka 5.1: Prehľad vstupných argumentov funkcie yalmipSVM

Vstupný argument	Popis
`M1`	prvá trénovacia množina bodov, t.j. matica zložená z vektorov vlastností prináležiacich prvej osobe
`M2`	druhá trénovacia množina bodov, t.j. matica zložená z vektorov vlastností prináležiacich druhej osobe
`gamma`	parameter, ktorý predstavuje pomer medzi šírkou separačnej medzery a aproximáciou počtu zle klasifikovaných bodov

meru $n \times 1$ a `bopt` je skalár. Platí, že počet riadkov vektora `aopt` sa musí rovnať počtu riadkov v matici `M1` aj `M2`, teda počtu súradníc ľubovoľného bodu z oboch množín.

V úvode programu pomocou funkcie `size` zistíme rozmery matíc `M1` a `M2`. `n` je počet riadkov matice `M1`, `m` je počet stĺpcov matice `M1` a `l` počet stĺpcov matice `M2`. V ďalšom kroku porovnáme počet riadkov v matici `M1` a `M2`. Ak sa dané rozmery nerovnajú, na obrazovke sa zobrazí správa o chybe. Zvyšok funkcie je riešením optimalizačného problému opísaného v podkapitole 5.4. Na tomto mieste používame Yalmip.

Optimalizované premenné - `a`, `b`, `u` a `v` - zadefinujeme pomocou príkazu `sdpvar`. `sdpvar` je jedným z najdôležitejších príkazov Yalmip-u [25]. V prípade, že optimalizovaná premenná nie je skalár, do okrúhlych zátvoriek musíme uviesť jej rozmery. Yalmip uvažuje každú štvorcovú maticu ako symetrickú. Tretím argumentom povieme, že daná štvorcová matica symetrická nie je [26]. V našom prípade nie je nevyhnutné používať tretí argument funkcie, pretože pracujeme s

vektormi.

Ďalej zadefinujeme účelovú funkciu a ohraničenia. Postupujeme presne podľa matematickej formulácie SupportVector Machine. Účelovú funkciu reprezentuje premenná `objective_function`, ohraničenia premenná `constraints`. Ohraničenia uvádzame v hranatých zátvorkách, oddelené bodkočiarkami. Jednou z veľkých výhod Yalmip-u je kontrola rozmerov jednotlivých premenných. To nám umožňuje vynechať pri definícii účelovej funkcie jednotkové vektory.

Optimalizačný problém vyriešime pomocou funkcie `optimize`. Táto funkcia má dva povinné vstupné argumenty - ohraničenia a účelovú funkciu presne v tomto poradí. Tretí vstupný argument `sdpsettings` je voliteľný [27]. Slúži na komunikáciu medzi Yalmip-om a solverom, ktorý využívame pri riešení daného problému [28].

Na záver do výstupných premenných `aopt` a `bopt` načítame pomocou funkcie `value` optimalizované hodnoty koeficientov lineárneho separátora.

```
1  function [aopt, bopt] = yalmip_SVM(M1, M2, gamma)
2
3  n = size(M1,1); % n = size(M2,1);
4  m = size(M1,2);
5  l = size(M2,2);
6
7  if (size(M1,1) ~= size(M2,1))
8  error('Number of rows in M1 must be the same as in M2')
9  end
10
11 sdpvar b;
12 a = sdpvar(n,1,'full');
```

```
13 u = sdpvar(m,1,'full');
14 v = sdpvar(l,1,'full');
15
16 objective_function = 1/4*a'*a + gamma*(sum(u) + sum(v));
17
18 constraints = [u>=0; v>=0;
19                 a'*M1 + b >= 1 - u';
20                 a'*M2 + b <= -1 + v'];
21
22 optimize(constraints,objective_function,sdpsettings('solver',
23 'gurobi', 'gurobi.Method', 0))
24
25 aopt = value(a);
26 bopt = value(b);
```

Na identifikáciu neznámych vzoriek použijeme funkciu `sampleClassification`. Táto funkcia má 5 vstupných a 1 výstupný argument. Prehľad vstupných argumentov nájdeme v tabuľke 5.2.

Tabuľka 5.2: Prehľad vstupných argumentov funkcie sampleClassification

Vstupný argument	Popis
`aopt, bopt`	optimalizované hodnoty koeficientov lineárneho separátora
`sample`	matica neznámych vzoriek
`class1`	meno prvej množiny bodov
`class2`	meno druhej množiny bodov

Premenné `aopt` a `bopt` sú riešením optimalizačného problému. Ich rozmery sú uvedené vyššie v tejto časti. Premenná `sample` je matica rozmeru $n \times k$, ktorá sa skladá z vektorov vlastností prináležiacich obom osobám, t.j. z bodov oboch

množín. n je počet extrahovaných vlastností alebo počet súradníc definujúcich jeden bod v priestore (jedno meranie) a k je počet neznámych vzoriek (vektorov vlastností). Počet riadkov v matici sample musí byť zhodný s počtom riadkov v maticiach M1 a M2 vstupujúcich do funkcie yalmipSVM. Premenné class1 a class2 obsahujú meno prvej a druhej množiny bodov vo forme reťazca.

Výstupným argumentom z funkcie `sampleClassification` je bunkové pole `class` rozmeru $m \times 1$. Platí, že počet riadkov premennej `class` sa musí rovnať počtu stĺpcov premennej `sample`. Každý prvok bunkového poľa bude obsahovať meno množiny, do ktorej patrí neznáma vzorka. Premenná `class` je výsledkom klasifikácie.

Na začiatku funkcie `sampleClassification` zistíme pomocou funkcie `size` počet stĺpcov matice `sample`. Rozmer uložíme do premennej `m`, ktorú využijeme na vytvorenie prázdneho bunkového poľa `class`. Bunkové pole plníme vo for cykle reťazcami `class1` a `class2`. For cyklus opakujeme pre všetky vzorky (stĺpce) v matici `sample`. Ak je podmienka splnená, t.j. výraz v zátvorke je väčší alebo rovný ako nula, vzorka patrí do prvej množiny bodov a do bunkového poľa sa zapíše reťazec `class1`, meno prvej množiny. V opačnom prípade, ak je výraz menší ako nula, vzorka patrí do druhej množiny bodov a do bunkového poľa sa zapíše reťazec `class2`, meno druhej množiny.

```
1 function class = sampleClassification(aopt,bopt,sample,class1,
2 class2)
3
4 m = size(sample,2);
5 class = cell(m,1);
6
7 for i = 1:m
```

```
8       if ((aopt'*sample(:,i) + bopt) >= 0)
9           class{i} = class1;
10
11      else
12          class{i} = class2;
13      end
14  end
```

5.5 Príprava dát na separáciu

Ako bolo spomenuté na začiatku kapitoly 5, separácia je algoritmus, ktorý slúži na binárnu klasifikáciu údajov. To znamená, že separujeme dve množiny bodov jedným lineárnym separátorom. Body z množiny sú reprezentované vektormi extrahovaných vlastností. Všetky tieto vektory sú zoradené za sebou v premenných `FEstat` a `FEemc`. V premnnej `FEstat` sú uložené vektory pätnástich vlastností získané výpočtom autokorelačnej matice. V premennej `FEemc` sú uložené vektory šiestich vlastností získané výpočtom energie, mobility a zložitosti.

Jednu množinu bodov, ktorú budeme môcť využiť ako vstup do separácie vytvoríme spojením všetkých vektorov vlastností prináležiacich jednému dobrovoľníkovi do matice vlastností. Pre 6 dobrovoľníkov vznikne 6 rôznych matíc vlastností, t.j. 6 rôznych množín bodov.

Matice vlastností vytvorí funkcia `SeparateGroups`, ktorá má dva vstupy a jeden výstup. Prvým vstupom do funkcie je bunkové pole `FolderNames`, ktoré predstavuje zoznam všetkých podpriečinkov obsahujúcich CSV súbory alebo zoznam všetkých dobrovoľníkov, ktorí sa zúčastnili meraní. Druhým vstupom je pole štruktúr `FE`. V premennej `FE` sú uložené vektory vlastností. Vzhľadom na to, že máme dva typy vektorov vlastností, funkciu `SeparateGroups` je potrebné spustiť

dvakrát. Prvýkrát s premennou `FEstat` a druhýkrát s premennou `FEemc`. Výstupom z funkcie je premenná `Groups`, pole štruktúr rozmeru $1 \times n$.

V prvom rade zistíme dĺžku bunkového poľa `FolderNames` a poľa štruktúr `FE`. Ďalej vytvoríme nové prázdne pole `Groups`. Vektory spojíme do matíc vlastností pomocou vnorených for cyklov. V prvom cykle priradíme premennej `FolderName` jeden prvok bunkového poľa `FolderNames`. Zároveň vytvoríme ďalšie prázdne pole `PersonMatrix`. Toto pole plníme v druhom for cykle. Ak je reťazec uložený v premennej `FolderName` rovnaký ako reťazec v položke `name` premennej `FE`, teda sa jedná o jedného dobrovoľníka, do premennej `PersonData` načítavame vektory vlastností z položky `data` a postupne ich pridávame do matice `PersonMatrix`. Hotovú maticu vlastností `PersonMatrix` v zvyšnej časti prvého for cyklu pridávame do premennej `Groups`.

```
1  function Groups = SeparateGroups(FolderNames, FE)
2
3  n = length(FolderNames);
4  m = length(FE);
5
6  Groups = [];
7
8  for i = 1:n
9      FolderName = FolderNames{i};
10     PersonMatrix = [];
11
12     for j = 1:m
13         if FolderName == FE(j).name
14             PersonData = FE(j).data;
15             PersonMatrix = [PersonMatrix, PersonData];
16         end
```

```
17      end
18
19      G.name = value(FolderName);
20      G.data = PersonMatrix;
21      Groups = [Groups, G];
22  end
23
24  end
```

Celú maticu vlastností teraz rozdelíme na trénovaciu maticu, ktorá bude slúžiť na natrénovanie separátora a maticu neznámych vzoriek, ktorá je vstupom do procesu identifikácie. Delenie bude prebiehať vo funkcii s názvom `TrainSamples`. Funkcia `TrainSamples` vezme ako svoj vstup premennú `Groups` a v nej uložené matice vlastností rozdelí na trénovacie matice a matice neznámych vzoriek. Trénovacie matice vráti funkcia uložené v premennej `TrainStruct` a matice neznámych vzoriek v premennej `Samples`.

Pomocou funkcie `length` najskôr zistíme, akú dĺžku má premenná `Groups`. Potom vytvoríme obe výstupné premenné ako prázdne polia. Matice delíme vo for cykle. Do položky `name` pomocnej premennej `T` aj `S` vložíme meno dobrovoľníka, ktorému patrila matica vlastností. Funkciou `size` zistíme počet stĺpcov matice vlastností. Posledných 5 stĺpcov matice vložíme do položky `data` štruktúry `S` a zvyšok do položky `data` štruktúry `T`. Štruktúru `T` na záver pridáme do poľa štruktúr `TrainStruct` a štruktúru `S` do poľa štruktúr `Samples`.

```
1  function [TrainStruct, Samples] = TrainSamples(Groups)
2
3  n = length(Groups);
4
5  TrainStruct = [];
```

```
6  Samples = [];
7
8  for i = 1:n
9      T.name = Groups(i).name;
10     S.name = Groups(i).name;
11     m = size(Groups(i).data, 2);
12     T.data = Groups(i).data(:,1:(m-5));
13     S.data = Groups(i).data(:,(m-4):m);
14     TrainStruct = [TrainStruct, T];
15     Samples = [Samples, S];
16 end
17
18 end
```

Kapitola 6

Klasifikačné stromy

V úvode kapitoly 5 sme uviedli, že v procese separácie dokážeme v priestore oddeliť práve dve trénovacie množiny bodov tým, že medzi ne umiestnime lineárny separátor. Na základe tohto separátora potom vieme priradiť neznámy bod, vzorku, k jednej alebo druhej množine bodov, t.j. klasifikovať ju.

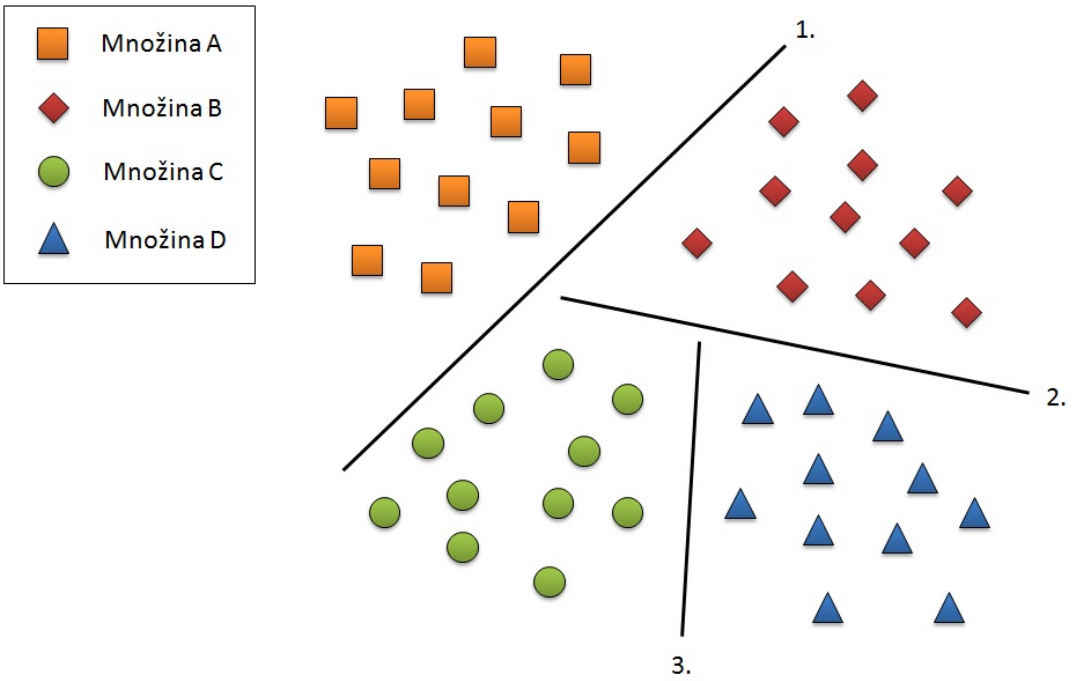
Veľmi výhodné by bolo vedieť separovať ľubovoľný počet množín bodov a potom priradiť neznámy bod k jednej z týchto množín. Na separovanie troch a viacerých množín a následné vyhodnocovanie neznámych vzoriek slúži metóda klasifikačných stromov. Táto metóda je založená na viacnásobnom opakovaní daného separačného algoritmu (v našom prípade Support Vector Machine) a postupnom delení jednotlivých množín bodov.

Vysvetlime si metódu klasifikačného stromu na príklade. Majme 4 rôzne množiny bodov A, B, C a D v 2-rozmernom priestore. Tieto množiny vidíme na obrázku 6.1. Na množiny budeme opakovane aplikovať Support Vector Machine algoritmus. Postup je nasledovný. Z množín B, C a D vytvoríme jednu množinu BCD. Prvým separátorom oddelíme v priestore množinu A od množiny BCD (prvé opakovanie). Množinu A si prestaneme všímať a množinu BCD rozdelíme na množiny B a CD.

Medzi množiny B a CD umiestnime druhý separátor (druhé opakovanie). Množinu A aj B si nevšímame, pracujeme iba s množinou CD. Rozdelíme ju na dve množiny, C a D a vytvoríme posledný, tretí, separátor (posledné opakovanie). Údaje o všetkých separátoroch pritom zapíšeme na jedno miesto, do klasifikačného stromu. Klasifikačný strom sa nachádza na obrázku 6.2.

Vyhodnocovanie neznámeho bodu prebieha na základe vytvoreného klasifikačného stromu. Neznámy bod reprezentuje symbol z, koeficienty separátora symboly a a b. Pri vyhodnocovaní postupujeme presne podľa obrázka. Pozrieme sa, či platí podmienka v prvom bielom bloku. Ak je výraz obsahujúci koeficienty prvého separátora väčší alebo rovný ako nula, bod patrí do množiny A. V opačnom prípade začneme vyhodnocovať podmienku v druhom bielom bloku. Ak je výraz obsahujúci koeficienty druhého separátora väčší alebo rovný ako nula, bod patrí do množiny B. Ak nie je, vyhodnocujeme podmienku uvedenú v treťom bielom bloku. V prípade, že je výraz obsahujúci koeficienty tretieho separátora väčší alebo rovný ako nula, bod prináleží množine C. Keď podmienka nie je splnená, bod zaradíme do poslednej množiny D. Takýto postup môžeme aplikovať pre ľubovoľne dlhý klasifikačný strom obsahujúci ľubovoľný počet množín a lineárnych separátorov.

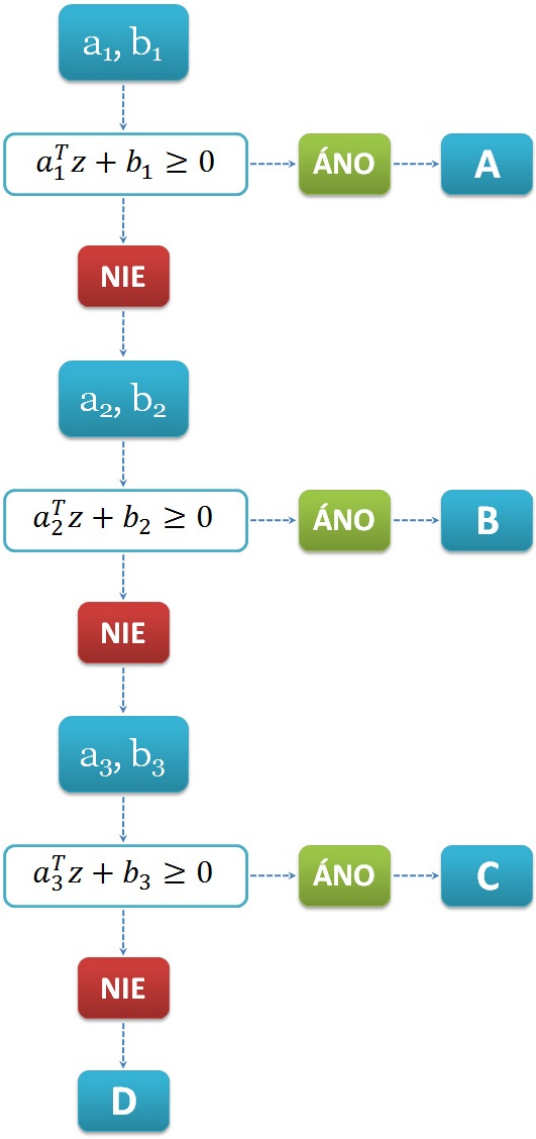
Je zrejmé, že klasifikačný strom použitý ako príklad sa vždy vetví smerom doprava. Je treba dodať, že to nie je jediný typ klasifikačného stromu. Existuje mnoho rôznych spôsobov, akými sa dá klasifikačný strom vytvoriť a mnoho rôznych organizácií uzlov. My použijeme takýto strom pre jeho jednoduchosť a ľahkú implementáciu v Matlab-e.



Obr. 6.1: Separácia 4 množín bodov

6.1 Implementácia metódy klasifikačných stromov v Matlab-e

Vybraný klasifikačný strom vetviaci sa smerom doprava vytvoríme v Matlab-e pomocou funkcie `CreateNodes`. Funkcia má dva vstupné a jeden výstupný parameter. Prvý vstupný parameter, pole štruktúr `TrainStruct` s rozmerom $1 \times n$ obsahuje jednotlivé trénovacie množiny bodov vo forme matíc extrahovaných vlastností. n vyjadruje počet trénovacích množín bodov. Premennú `gamma`, ktorá predstavuje pomer medzi šírkou separačnej medzery a aproximáciou počtu zle klasifikovaných bodov, rovnako ako trénovacie množiny vyžaduje ako svoje vstupy algoritmus Support Vector Machine. Výstupom z funkcie `CreateNodes` je premenná `Output`, pole štruktúr rozmeru $1 \times (n-1)$, v ktorom je uložený klasifikačný strom.



Obr. 6.2: Klasifikačný strom

V prvom rade zistíme pomocou funkcie `length` dĺžku premennej `TrainStruct`. Ďalej vytvoríme premennú `Output` ako prázdne pole. Koeficienty lineárnych separátov hľadáme a ukladáme do klasifikačného stromu pomocou dvoch vnorených for cyklov. V prvom for cykle vložíme do premennej `train_class1` prvú trénovaciu maticu vlastností. V druhom for cykle spojíme zvyšné trénovacie matice do jednej.

Spojená matica sa bude nachádzať v premennej `train_class2`. Koeficienty separátora získame spustením vnorenej funkcie `yalmipSVM`, ktorá bola vysvetlená v predchádzajúcej kapitole. Prvý for cyklus pokračuje plnením klasifikačného stromu. Jeden uzol stromu bude reprezentovať štruktúra `node`. Táto štruktúra má 4 položky - `a`, `b`, `right` a `down`. Do položky `a` zapíšeme hodnotu premennej `a_opt`, do položky `b` hodnotu `b_opt`. Položky `right` a `down` obsahujú informácie o dcérskych uzloch. Ako je vidno na obrázku 6.2, napravo od uzla obsahujúceho koeficienty separátora sa vždy nachádza meno trénovacej množiny. Pod týmto uzlom leží uzol s koeficientami ďalšieho separátora. Iba v prípade, keď sa nachádzame na konci stromu, je napravo aj dole uvedené meno trénovacej množiny. Toto je implementované v závere programu. Meno množiny bodov `train_class1` zapíšeme do položky `right`. Položka `down` zostáva prázdna. Hotový uzol pridáme do poľa štruktúr `Output`. V ďalšom opakovaní vložíme do premennej `train_class1` druhú trénovaciu maticu vlastností a do premennej `train_class2` tretiu až poslednú trénovaciu maticu. Opakujeme ten istý postup. Nájdeme koeficienty druhého separátora, vložíme ich spolu s informáciami o dcérskych uzloch do premennej `node` a tú pridáme do poľa štruktúr `Output`. Program končí vtedy, keď oddelíme predposlednú a poslednú trénovaciu maticu a zapíšeme meno poslednej matice do položky `down`.

```
1  function Output = CreateNodes(TrainStruct, gamma)
2
3  n = length(TrainStruct);
4
5  Output = [];
6
7  for i = 1:(n-1)
8      train_class1 = TrainStruct(i).data;
9      train_class2 = [];
10
```

```
11      for j = (i+1):n
12          train_rest = TrainStruct(j).data;
13          train_class2 = [train_class2, train_rest];
14      end
15
16      [a_opt, b_opt] = yalmipSVM(train_class1, train_class2, gamma);
17
18      node = [];
19
20      node.a = a_opt;
21      node.b = b_opt;
22      node.right = TrainStruct(i).name;
23
24      if (i == (n-1))
25          node.down = TrainStruct(end).name;
26      else
27          node.down = [];
28      end
29
30      Output = [Output, node];
31
32  end
33  end
```

Jednu neznámu vzorku klasifikujeme pomocou funkcie `treeClassify`. Prvým vstupným argumentom do funkcie je klasifikačný strom vo forme poľa štruktúr a druhým vstupným argumentom je neznáma vzorka `sample`. Premenná `sample` je v skutočnosti jeden vektor extrahovaných vlastností. Výstupným parametrom je reťazec `class_name` uvádzajúci meno množiny bodov, do ktorej vzorka patrí. Algoritmus je podobný ako vo funkcii `sampleClassification`. Pre každú dvojicu koeficientov separátora vyhodnocujeme platnosť alebo neplatnosť podmienky, ktorá sa objavuje aj na obrázku 6.2. Ak je podmienka platná, do premennej `class_name`

načítame hodnotu z položky `right` premennej `tree`. Ak podmienka neplatí a nachádzame sa na konci poľa štruktúr `tree`, teda vyhodnocujeme posledný separátor, do premennej `class_name` sa zapíše hodnota z položky `down`. Ak sa na konci zatiaľ nenachádzame, podmienka sa vyhodnocuje rovnakým spôsobom pre ďalšie separátory, pokým vzorku nezaradíme.

```
1  function class_name = treeClassify(tree, sample)
2
3  for i = 1:length(tree)
4
5      if (tree(i).a'*sample + tree(i).b >= 0)
6          class_name = tree(i).right;
7          return
8
9      elseif (i == length(tree))
10             class_name = tree(i).down;
11             return
12     end
13
14 end
15
16 end
```

Dve a viac neznámych vzoriek usporiadaných do matice klasifikujeme vo funkcii `classification`. Prvý vstupný parameter je opäť klasifikačný strom, druhým vstupným parametrom sú matice neznámych vzoriek v poli štruktúr `Samples`. Výstupom z programu je bunkové pole `Classes`, v ktorom nájdeme zaradenie jednotlivých vzoriek k množinám bodov. Bunkové pole má toľko prvkov, koľko je počet vzoriek v premennej `Samples`.

V úvode programu vytvoríme premennú `Classes` ako prázdne pole a zistíme

dĺžku poľa štruktúr `Samples`, teda počet matíc neznámych vzoriek. Bunkové pole `Classes` naplníme pomocou 2 for cyklov a pomocnej premennej `C`. V prvom for cykle zistíme počet stĺpcov jednej matice neznámych vzoriek a vytvoríme pomocnú premennú `C`. V druhom for cykle používame vnorenú funkciu `treeClassify`. Vstupom do tejto funkcie je klasifikačný strom a jeden stĺpec ľubovoľnej matice neznámych vzoriek. Výsledok klasifikácie sa uloží do pomocnej premennej `C`. Po poslednom opakovaní druhého for cyklu bude bunkové pole `C` obsahovať výsledky klasifikácie všetkých vzoriek z jednej matice. Takéto bunkové pole pridáme do výstupnej premennej `Classes` a pokračujeme so vzorkami nachádzajúcimi sa v ďalšej matici.

```
1  function Classes = classification(Tree, Samples)
2
3  Classes = [];
4
5  n = size(Samples, 2);
6
7  for i = 1:n
8      m = size(Samples(i).data, 2);
9      C = {};
10     for j = 1:m
11         ClassName = treeClassify(Tree, Samples(i).data(:,j));
12         C{j} = ClassName;
13     end
14     Classes = [Classes, C];
15 end
16
17 Classes = Classes';
18
19 end
```

Kapitola 7

Vyhodnotenie výsledkov

Na záver práce spojíme funkcie podieľajúce sa na biometrickej identifikácii osôb na základe pohybu do jednému celku, čím vznikne nová funkcia `RawToFinal`. V nej postupne dosahujeme všetky čiastkové ciele. Zistíme počet a mená dobrovoľníkov, ktorí sa zúčastnili zberu údajov. Nazberané údaje prevedieme do prostredia Matlab-u, kde sa budú ďalej spracovávať. Z pôvodných nameraných údajov vyberieme akcelerácie, na ktoré následne aplikujeme dve metódy extrakcie vlastností. Z vlastností vytvoríme trénovacie a testovacie množiny, ktoré využijeme ako vstup do metódy klasifikačných stromov založenej na viacnásobnom opakovaní separačného algoritmu Support Vector Machine. V poslednom kroku klasifikujeme neznáme vzorky a výsledky ukladáme do výstupu funkcie `RawToFinal`, premenných `emcC` a `statC`.

```
1 function [emcC, statC] = RawToFinal(FolderPath, Ts, freq, gamma)
2
3 FolderNames = SubfoldersList(FolderPath);
4 AllRawData = CollectRawData(FolderNames);
5 Accelerations = CollectAccelerations(AllRawData);
6 [FEemc, FEstat] = FeatureExtraction(Accelerations, Ts, freq);
```

```
7  emcGroups = SeparateGroups(FolderNames, FEemc);
8  statGroups = SeparateGroups(FolderNames, FEstat);
9  [emcTrainStruct, emcSamples] = TrainSamples(emcGroups);
10 [statTrainStruct, statSamples] = TrainSamples(statGroups);
11 emcTree = CreateNodes(emcTrainStruct, gamma);
12 statTree = CreateNodes(statTrainStruct, gamma);
13 emcC = classification(emcTree, emcSamples);
14 statC = classification(statTree, statSamples);
15
16 end
```

Napriek tomu, že sme vzorky v testovacích množinách označili za neznáme, v skutočnosti presne vieme, do ktorej z množín patria. Poznáme tiež poradie, v akom vstupovali do klasifikácie. To nám dovoľuje vykonať kontrolu výsledkov uložených v premenných `emcC` a `statC` a určiť úspešnosť klasifikácie v percentách. V procese kontroly porovnávame hodnotu zapísanú v prvku bunkového poľa s reálnou (známou) hodnotou. Zisťujeme, či program priradil vzorku do takej množiny, ku ktorej naozaj patrí.

Keďže neznáma vzorka, ktorú klasifikujeme nie je nič iné ako vektor vlastností, môžeme povedať, že úspešnosť klasifikácie v najväčšej miere ovplyvňuje práve použitá metóda extrakcie vlastností. V tejto práci sme na akcelerometrické signály aplikovali dve rozličné metódy extrakcie. Výsledky z prvej metódy, založenej na výpočte autokorelačnej matice Fourierových vlastností, sú uložené v premennej `statC`. Výsledky z druhej metódy, založenej na výpočte energie, mobility a zložitosti, sú uložené v premennej `emcC`. Pre každú premennú osobitne sme vyhodnotili úspešnosť klasifikácie, čím sme porovnali obe metódy extrakcie a posúdili kvalitu extrahovaných vlastností. Porovnanie zachytávajú tabuľky 7.1 a 7.2. Je zrejmé, že vyššiu úspešnosť klasifikácie dosahujeme aplikovaním prvej metódy extrakcie vlastností, preto je vhodnejšia na účely biometrickej identifikácie

na základe pohybu.

Na úspešnosť klasifikácie ďalej výrazné vplýva počet množín bodov vstupujúcich do separácie a hodnota parametra γ. Ako závisí úspešnosť klasifikácie od počtu množín bodov opäť uvádzajú tabuľky 7.1 a 7.2. Platí, že čím viac množín bodov chceme separovať, tým nižšia bude úspešnosť klasifikácie. Parameter γ určuje pomer medzi šírkou separačnej medzery a aproximáciou počtu zle klasifikovaných bodov. Ak chceme zvýšiť úspešnosť klasifikácie, musíme pridať váhu práve tomu výrazu v účelovej funkcii Support Vector Machine, ktorý zabezpečuje minimalizáciu počtu zle klasifikovaných bodov. Váhu mu pridávame zvyšovaním hodnoty parametra γ. Najlepšie výsledky sme dosiahli vtedy, keď bol tento parameter rovný číslu 5. Úspešnosti zapísané v tabuľkách 7.1 a 7.2 platia práve pre túto hodnotu.

Na záver môžeme dodať, že úspešnosť klasifikácie sa dá zvýšiť už v priebehu merania a pred samotným spracovaním údajov. Výsledky zlepšíme použitím presnejšieho a citlivejšieho akcelerometra alebo použitím vhodného filtra.

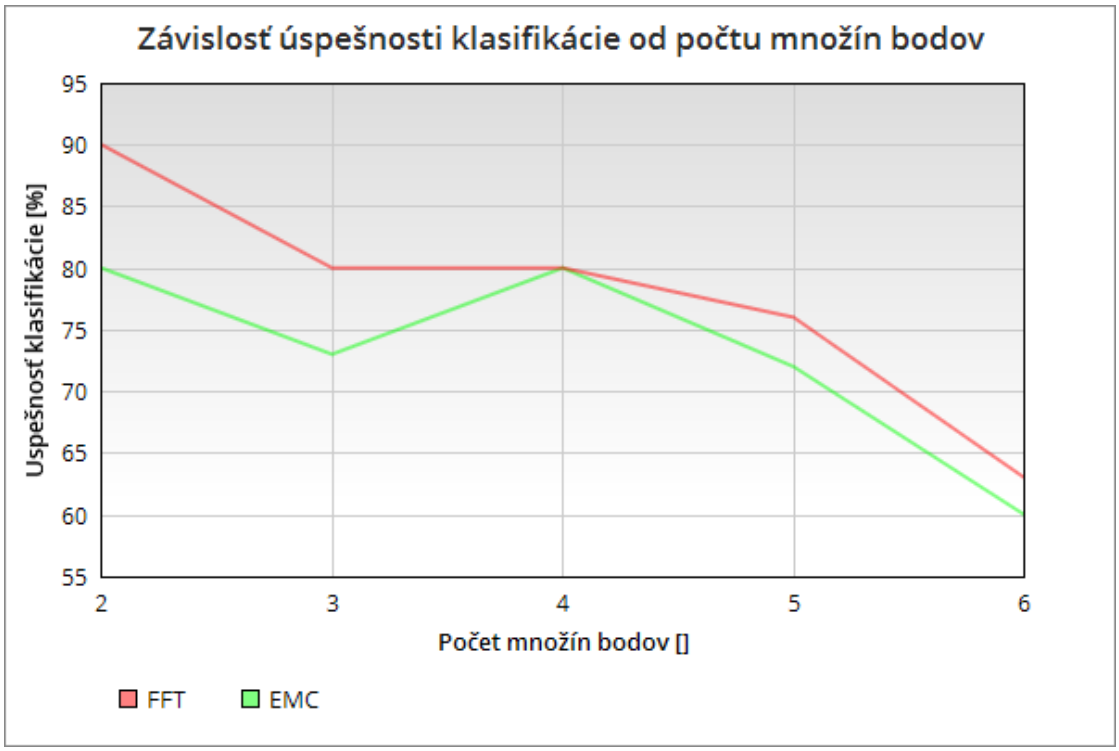
Tabuľka 7.1: Úspešnosť klasifikácie pri použití metódy extrakcie založenej na výpočte autokorelačnej matice

Počet množín bodov	Úspešnosť klasifikácie v %
6	63
5	76
4	80
3	80
2	90

Grafický priebeh výsledkov sa nachádza na obrázku 7.1.

Tabuľka 7.2: Úspešnosť klasifikácie pri použití metódy extrakcie založenej na výpočte energie, mobility a zložitosti

Počet množín bodov	Úspešnosť klasifikácie v %
6	60
5	72
4	80
3	73
2	80



Obr. 7.1: Závislosť úspešnosti klasifikácie od počtu množín bodov

Záver

V bakalárskom projekte sme sa zaoberali biometrickou identifikáciou osôb na základe pohybu.

Našou prvou úlohou bolo získať vzorky pohybov 6 dobrovoľníkov. Jednotlivé vzorky sme získavali meraním akcelerácií vznikajúcich pri pohyboch dobrovoľníkov. Za jedno meranie sme považovali jedno zdvihnutie telefónu k uchu. Zber údajov sme realizovali pomocou smartfónu vybaveného akcelerometrom a aplikácie GYRO. Výstupom zo zberu bolo 185 vzoriek pohybov.

V druhej fáze projektu sme namerané údaje previedli do prostredia Matlabu, kde prebiehalo ich ďalšie spracovanie. Najdôležitejšou časťou spracovania bolo aplikovanie extrakcie vlastností na pôvodné akcelerácie. V práci sme využili dve rôzne metódy extrakcie. Jedna bola založená na výpočte autokorelačnej matice Fourierových vlastností. Táto metóda produkovala vektor 15 vlastností. Druhá metóda sa zakladala na výpočte energie, mobility a zložitosti. Výsledkom tejto metódy bol vektor 6 vlastností.

Z vektorov vlastností sme vytvorili trénovacie a testovacie množiny bodov. Trénovacie množiny sme použili ako vstup do separácie. Oddeľovali sme ich pomocou metódy klasifikačných stromov založenej na viacnásobnom opakovaní separačného algoritmu Support Vector Machine. Výstupom z metódy bol klasifi-

kačný strom obsahujúci údaje o lineárnych separátoroch. Na základe vytvoreného klasifikačného stromu sme klasifikovali neznáme vzorky umiestnené v testovacích množinách. Na záver sme vyhodnotili úspešnosť klasifikácie a výsledky prehľadne spracovali do grafickej aj tabuľkovej podoby.

Môžeme povedať, že sme splnili cieľ tejto práce a dokázali, že je možné identifikovať osobu na základe jej pohybov.

Literatúra

[1] Z. Whittaker, "iPhone 5S fingerprint sensor: The end of passwords?." Dostupné na internete: http://www.cnet.com/news/iphone-5s-fingerprint-sensor-the-end-of-passwords/. Posledný prístup: máj, 2015.

[2] C. Gunther, "How to Use the Galaxy S5 Fingerprint Scanner." Dostupné na internete: http://www.gottabemobile.com/2014/07/14/how-to-use-the-galaxy-s5-fingerprint-scanner/. Posledný prístup: máj, 2015.

[3] G. Mazo, "How to set up Face Unlock on your Android phone." Dostupné na internete: http://www.androidcentral.com/how-set-face-unlock-your-htc-one-x-or-evo-4g-lte. Posledný prístup: máj, 2015.

[4] H. Arora, "Android 5.0 Lollipop Gets New 'On-Body Detection' Smart Lock." Dostupné na internete: http://gadgets.ndtv.com/apps/news/android-50-lollipop-gets-new-on-body-detection-smart-lock-674028. Posledný prístup: máj, 2015.

[5] A. K. Jain, R. Bolle, and S. Pankanti, *Biometrics: personal identification in networked society*. Springer Science & Business Media, 1999.

[6] U. ScienceLine, "Do twins have the same dna?." Dostupné na internete: http://scienceline.ucsb.edu/getkey.php?key=244. Posledný prístup: máj, 2015.

[7] K. Norrgard, "Forensics, dna fingerprinting, and codis," *Nature Education*, vol. 1, no. 1, p. 35, 2008.

[8] Wikipedia, "Zrýchlenie." Dostupné na internete: http://sk.wikipedia.org/wiki/Zr%C3%BDchlenie. Posledný prístup: máj, 2015.

[9] Wikipedia, "Accelerometer." Dostupné na internete: http://en.wikipedia.org/wiki/Accelerometer. Posledný prístup: máj, 2015.

[10] Wikipedia, "Comma-separated values." Dostupné na internete: http://sk.wikipedia.org/wiki/Comma-separated_values. Posledný prístup: máj, 2015.

[11] Y. Shafranovich, "Common format and mime type for comma-separated values (csv) files,"

[12] Mathworks, "Matlab." Dostupné na internete: http://www.mathworks.com/products/matlab/index-b.html. Posledný prístup: máj, 2015.

[13] Mathworks, "textscan." Dostupné na internete: http://www.mathworks.com/help/matlab/ref/textscan.html. Posledný prístup: máj, 2015.

[14] Mathworks, "Create a structure array." Dostupné na internete: http://www.mathworks.com/help/matlab/matlab_prog/create-a-structure-array.html. Posledný prístup: máj, 2015.

[15] Mathworks, "dir." Dostupné na internete: http://www.mathworks.com/help/matlab/ref/dir.html. Posledný prístup: máj, 2015.

[16] Wikipedia, "Feature extraction." Dostupné na internete: http://en.wikipedia.org/wiki/Feature_extraction. Posledný prístup: máj, 2015.

[17] T. Kobayashi, K. Hasida, and N. Otsu, "Rotation invariant feature extraction from 3-d acceleration signals," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pp. 3684–3687, IEEE, 2011.

[18] M. Khan, S. I. Ahamed, M. Rahman, and R. O. Smith, "A feature extraction method for realtime human activity recognition on cell phones," in *Proceedings of 3rd International Symposium on Quality of Life Technology (isQoLT 2011). Toronto, Canada*, 2011.

[19] Kardioklub, "Encefalografické vyšetrenie - eeg." Dostupné na internete: http://www.kardioklub.biznisweb.sk/info/o-vysetreniach/eeg-vysetrenie/. Posledný prístup: máj, 2015.

[20] Mathworks, "diff." Dostupné na internete: http://www.mathworks.com/help/matlab/ref/diff.html. Posledný prístup: máj, 2015.

[21] "Optimalizácia: on-line kurz pre FCHPT." Dostupné na internete: http://www.kirp.chtf.stuba.sk/moodle/course/view.php?id=65. Posledný prístup: máj, 2015.

[22] Wikipedia, "Linear separability." Dostupné na internete: http://en.wikipedia.org/wiki/Linear_separability. Posledný prístup: máj, 2015.

[23] Wikipedia, "Norm (mathematics)." Dostupné na internete: http://en.wikipedia.org/wiki/Norm_%28mathematics%29. Posledný prístup: máj, 2015.

[24] Y. Wiki, "What is yalmip." Dostupné na internete: http://users.isy.liu.se/johanl/yalmip/pmwiki.php?n=Main.WhatIsYALMIP. Posledný prístup: máj, 2015.

[25] Y. Wiki, "Basics." Dostupné na internete: http://users.isy.liu.se/johanl/yalmip/pmwiki.php?n=Tutorials.Basics. Posledný prístup: máj, 2015.

[26] Y. Wiki, "Sdpvar." Dostupné na internete: http://users.isy.liu.se/johanl/yalmip/pmwiki.php?n=Commands.Sdpvar. Posledný prístup: máj, 2015.

[27] Y. Wiki, "Optimize." Dostupné na internete: http://users.isy.liu.se/johanl/yalmip/pmwiki.php?n=Commands.Optimize. Posledný prístup: máj, 2015.

[28] Y. Wiki, "Sdpsettings." Dostupné na internete: http://users.isy.liu.se/johanl/yalmip/pmwiki.php?n=Commands.Sdpsettings. Posledný prístup: máj, 2015.