Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Rozpoznávanie gest pomocou strojového učenia

Bakalárska práca

2023
Matej Martinček

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Rozpoznávanie gest pomocou strojového učenia

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Ing. Viktor Kocur, PhD.

Bratislava, 2023
Matej Martinček

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Matej Martinček
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Rozpoznávanie gest pomocou strojového učenia
Gesture Recognition Using Machine Learning

Anotácia: Gestá môžu slúžiť ako vhodný prostriedok pri interakcií humanoidných robotov s človekom. Pre umožnenie takýchto interakcií je potrebné takéto gestá vhodným spôsobom rozpoznať.

Cieľ: Oboznámiť sa so systémom na detekciu pózy ruky LEAP. Vypracovať prehľad problematiky rozpoznávania gest pomocou metód strojového učenia. Navrhnúť, implementovať a vyhodnotiť možnosti rozpoznávania rôznych gest rúk s pomocou techník strojového učenia.

Vedúci: Ing. Viktor Kocur, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 31.05.2022

Dátum schválenia: 29.09.2022 doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Chcel by som poďakovať svojmu školiteľovi Viktorovi Kocurovi, ktorého skvelá komunikácia, podpora a pomoc s riešením problémov dopomohli k existencii tejto práce.

Abstrakt

Cieľom tejto práce je analyzovať a porovnať metódy, ktoré ponúka súčasný vedecký pokrok v oblasti učenia s učiteľom, použiteľné pre rozpoznávanie gest za účelom ich následnej aplikácie pre komunikáciu človeka s humanoidným robotom založenej na gestách.

Na klasifikáciu gest sú v našej práci použité a porovnané rekurentná neurónová sieť využívajúca GRU jednotky majúca výborné výsledky v oblasti rozpoznávania gest a všeobecne aplikovateľné a dlho známe metódy strojového učenia: metóda podporných vektorov a náhodný les.

Uvedené metódy sú v našej práci porovnávané na vybranom datasete, ktorý bol vytvorený pomocou technológie založenej na extrakcii dát o pozícii rúk v priestore zo záznamov infračervených kamier. Dataset obsahuje šesť typov záznamov gest.

Výsledkom našej práce, po trénovaní a testovaní použitia uvedených metód na našom datasete, je prekvapivá porovnateľnosť výkonnosti rekurentnej neurónovej siete a metódy podporných vektorov/náhodného lesa, pričom najlepšia dosiahnutá testovacia presnosť klasifikácie bola 80%.

Výsledky našej práce môžu byť priamo použité pri implementácii komunikácie s humanoidným robotom, prípadne na ne môže byť nadviazané ďalšou sadou experimentov.

Kľúčové slová: interakcia medzi človekom a robotom, rozpoznávanie gest, metódy učenia s učiteľom, predspracovanie dát, rekurentné neurónové siete, Leap Motion Controller

Abstract

The aim of this work is to analyze and compare the methods offered by the current scientific advances in supervised machine learning applicable to gesture recognition for the purpose of their subsequent application to gesture-based human-robot interaction.

In our work, a recurrent neural network using GRU units with excellent results in gesture recognition and generally applicable and well-known machine learning methods: support vector machine method, and random forest are used and compared for gesture classification.

The above methods are compared on a selected dataset, which was created using a technology based on extracting data of the hand's position from infrared camera recordings. The dataset contains six types of gesture records.

As a result of our work, after training and testing the application of the above methods on our dataset, the performance of the recurrent neural network and the support vector machines/random forest method is surprisingly comparable, with the best test classification accuracy achieved being 80%.

The results of our work can be directly used in the implementation of human-robot interaction or can be followed up with another set of experiments.

Keywords: human-robot interaction, gesture recognition, supervised learning models, data preprocessing, recurrent neural networks, Leap Motion Controller

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Hlavnou motiváciou tejto práce je umožnenie komunikácie pomocou gest s humanoidným robotom NICO [9] v rámci projektu prebiehajúceho na Katedre aplikovanej informatiky FMFI Univerzity Komenského v Bratislave, a to s využitím umelej inteligencie.

V našej práci nadväzujeme na pilotnú štúdiu [23] tohto projektu a využívame metódy strojového učenia pre rozpoznávanie sady gest: pýtať si (ask), uchopiť (grasp), mávať (move), ukázať (point) a ok. Ich zaznamenávanie bolo realizované pomocou prístroja Leap Motion Controller od spoločnosti Ultraleap [24], ktorý na základe údajov z dvoch infračervených kamier zaznamenáva pohyb ruky ako časovú postupnosť jej pozícií.

Keďže posledné desaťročia zaznamenali rýchly posun a veľký pokrok v oblasti umelej inteligencie a strojového učenia, ako cieľ tejto práce sme si zvolili porovnanie výkonnosti starších metód s nedávnym vývojom v oblasti hlbokého učenia. Ako klasifikačné modely pre rozpoznávanie gest sme si teda zvolili dlho známe a populárne algoritmy z oblasti strojového učenia, akými sú metóda podporných vektorov (angl. support vector machines skr. SVM) a náhodný les (angl. random forest). Pre porovnanie s modernejšími metódami strojového učenia sme si vybrali model rekurentnej neurónovej siete DeepGRU (DeepGRU: Deep Gesture Recognition Utility) [14], ktorá je inšpirovaná nedávnym pokrokom v oblasti hlbokého učenia pre rozpoznávanie gest a pohybu.

Rekurentné neurónové siete sú známe schopnosťou učenia sa závislostí v časovej sekvencii dát na vstupe. Keďže sú gestá v našom datasete zaznamenané ako postupnosť pozícií ruky v čase, je rekurentná neurónová sieť veľmi vhodným modelom pre naše účely. Motiváciou pre výber práve neurónovej siete DeepGRU boli najmä jej výborné výsledky prekonávajúce väčšinu doteraz použitých neurónových sietí v oblasti rozpoznávania gest a jej schopnosť pracovať s neupravenými hrubými vektorovými dátami v rôznych aplikáciách, ako sú gestá rúk, pohyb celého tela alebo dokonca interakcia viacerých ľudí.

V nasledujúcej kapitole (1) popíšeme súvisiace práce a pokračovať budeme kapitolou 2, v ktorej vysvetlíme princípy fungovania metód strojového učenia používaných v našej práci. V kapitole 3 potom popíšeme použité dáta, metódy predspracovania dát a klasifikačné modely spolu s ich použitou implementáciou a nakoniec popíšeme štruktúru

vykonaných experimentov. V 4. kapitole potom uvedieme výsledky týchto experimentov a v predposlednej kapitole (5) tieto výsledky rozoberieme a rozdiskutujeme.

Kapitola 1

Súvisiace práce

V tejto kapitole popisujeme pilotnú štúdiu vykonanú na rovnakých dátach, s akými pracujeme v našej práci a tiež prácu, v ktorej sa využíva iný prístup k zaznamenávaniu a rozpoznávaniu gest.

1.1 Pilotná štúdia

Pred napísaním tejto práce bola vykonaná pilotná štúdia [23] pre modul humanoidného robota majúci rozpoznávať ľudskú činnosť (a to v prvom rade gestá), na ktorú naša práca nadväzuje.

V rámci nej bola nahratá väčšina záznamov gest, ktoré sú v našej práci používané. Zavedené boli gestá: pýtať si (ask), uchopiť (grasp), mávať (move), ukázať (point), ok a tiež záznam, ktorý mal zodpovedať nevykonávaniu žiadneho z týchto gest (teda náhodnému pohybu) nazvaný nič (nothing). Pre účely ich rozpoznávania boli následne vykonané experimenty so **sieťou s echo stavom** (angl. echo state network skr. ESN) [10], ktorej vhodnosť bola pre riešenie tohto problému preverovaná.

Konkrétne boli v rámci tejto štúdie nahraté dve sady záznamov, každá z nich piatimi subjektami, pričom zámerom nahrávania druhej sady bolo získanie uniformnejších záznamov gest než v prvej sade. Následne boli dáta normalizované pomocou min-max škálovania. Ako popisované v kapitole 3, gestá boli zaznamenané ako časová sekvencia pozícií kĺbov ruky.

ESN sieť bola potom trénovaná na predikciu gesta v každom kroku jeho časovej sekvencie. Tieto predikcie boli v rámci jednej časovej sekvencie gesta agregované ako ich modus (najčastejšia hodnota). Pre trénovanie ESN bola použitá krížová validácia s vynechaním jedného (angl. leave-one-out cross-validation), a to vždy s použitím dát zaznamenaných jedným subjektom pre testovanie a dát zaznamenaných zvyšnými subjektami na trénovanie. Pri tom boli použité fixné hyperparametre ESN siete.

S takto natrénovanou sieťou v štúdii dosiahli priemernú presnosť 56% na prvej

sade a 69% na druhej (uniformnejšej) sade záznamov. Ďalej medzi výsledky tejto štúdie patrí zistenie, že pre korektné rozpoznanie týchto gest nie je potrebné odlišovanie pravej a ľavej ruky, keďže to malo minimálny dopad na finálnu presnosť. Minimálny dopad na presnosť malo tiež odstránenie gesta „nič“, ktoré bolo najčastejšie nesprávne interpretované. Na záver bol podotknutý fakt, že dataset obsahuje príliš malé množstvo záznamov gest, a preto dochádza k prílišnej variabilite záznamov medzi jednotlivými subjektami, čiže ich nedostatočnému zovšeobecneniu.

1.2 Iný prístup k rozpoznávaniu gest

Za zmienku stojí aj diplomová práca Mateja Králika [12] z FMFI UK, v ktorej k problematike rozpoznávania gest pristupujú odlišným spôsobom.

V tejto práci boli totižto skonštruované datasety gest (iných ako v našej práci a predchádzajúcej štúdii) pomocou rukavice (taktiež skonštruovanej v tejto práci) vybavenej IMU senzormi snímajúcimi uhlovú rýchlosť a zrýchlenie. Jeden záznam gesta je potom predstavovaný postupnosťou hodnôt (signálom) z jednotlivých kanálov senzorov, čo sa odlišuje od spôsobu zaznamenávania dát v našej práci (popísaného v kapitole 3). Okrem týchto datasetov boli v práci Mateja Králika použité aj iné dostupné datasety.

Na datasetoch nahratých s pomocou skonštruovanej rukavice boli v tejto práci dosiahnuté pozoruhodné výsledky použitím rôznych metód hlbokého učenia vrátane LSTM (long short-term memory) a konvolučných sietí. Ich presnosť na testovacích dátach po natrénovaní s použitím k-fázovej krížovej validácie presahovala 99%.

Táto presnosť však nie je porovnateľná s presnosťou dosiahnutou v našej práci (resp. predchádzajúcej pilotnej štúdii) kvôli rozdielnosti našich datasetov nielen čo sa týka odlišnosti gest, ale aj diametrálne odlišnému spôsobu ich zaznamenávania. Toho dôkazom je aj testovacia presnosť dosiahnutá v práci Mateja Králika pomocou metódy bootstrapovej agregácie na rozhodovacích stromoch (skr. bagging, popísaná v kapitole 2) na dátach z datasetu nahratého senzorovou rukavicou. Tá presahovala 96%. Pritom metóda náhodného lesa použitá v našej práci, ktorá je veľmi podobná baggingu, dosahovala na našom datasete najlepšiu testovaciu presnosť 80%.

Kapitola 2

Teoretické východiská

V tejto kapitole vysvetľujeme základné princípy fungovania metód strojového učenia, ktoré v našej práci aplikujeme. V celej tejto kapitole pracujeme primárne so stĺpcovými vektormi, ak nie je uvedené inak. Všetky modely sú zároveň popisované z pohľadu paradigmy učenia s učiteľom, keďže v našej práci je využívaná výhradne táto paradigma. Pripomenieme, že trénovacie príklady sa teda skladajú z dvoch častí, a to z príznakového vektora (angl. feature vector) a označenia/triedy (angl. label/class).

2.1 Metóda podporných vektorov

Metóda podporných vektorov [15] (angl. support vector machines, skr. SVM) je metóda používaná v oblasti strojového učenia s učiteľom pre klasifikáciu aj regresiu.

Jej základom je **binárna** klasifikácia, teda kategorizácia pozorovaní do dvoch tried. Na základe danej množiny trénovacích príkladov, sa toto dosahuje nájdením **optimálnej nadroviny** rozdeľujúcej príznakový priestor (angl. feature space) dát na dve časti tak, že príklady s opačným označením ležia na opačnej strane nadroviny.

Optimálnou nadrovinou je myslená taká nadrovina, ktorej vzdialenosť k najbližšiemu trénovaciemu príkladu, ako bodu určeného jeho príznakovým vektorom (angl. feature vector) v danom príznakovom priestore, je maximálna možná. Každá nadrovina v priestore $\mathbb{R}^n$ je daná parametrami $\vec{w} \in \mathbb{R}^n$ a $b \in \mathbb{R}$ a rovnicou $\vec{w}^\top \vec{x} + b = 0$ pre ľubovoľný bod $\vec{x} \in \mathbb{R}^n$. Dá sa teda reprezentovať ako dvojica parametrov $(\vec{w}, b)$. Trénovací príklad zas môže byť reprezentovaný ako dvojica $(\vec{x}, y)$, kde $\vec{x} \in \mathbb{R}^n$ je jeho príznakový vektor a $\vec{y} \in \{-1, 1\}$ je jeho označenie. Vzdialenosť trénovacieho príkladu $(\vec{x}, y)$ od nadroviny $(\vec{w}, b)$ je potom $D((\vec{x}, y), (\vec{w}, b)) = \frac{|\vec{w}^\top \vec{x} + b|}{||\vec{w}||}$ (vzdialenosť príznakového vektora daného príkladu ako bodu od danej nadroviny). Optimálna nadrovina je teda taká dvojica $(\vec{w}, b)$ maximalizujúca $V \in \mathbb{R}$ t.ž. $D((\vec{x}, y), (\vec{w}, b)) \geq V$ pre všetky trénovacie príklady $(\vec{x}, \vec{y})$. Predpokladajme, že príklady s označením 1 ležia v kladnom polopriestore určenom nadrovinou $(\vec{w}, b)$, t.j. príklady, pre ktorých príznakové vektory

$\vec{x}$ platí $(\vec{w}^\top \vec{x} + b) > 0$, a podobne príklady s označením -1 ležia v zápornom polopriestore určenom nadrovinou $(\vec{w}, b)$, t.j. príklady, pre ktorých príznakové vektory $\vec{x}$ platí $(\vec{w}^\top \vec{x} + b) < 0$. Potom aby sme zároveň splnili (ak je to možné) podmienku, že príklady s opačným označením majú ležať na opačnej strane nadroviny, môžeme maximalizovať miesto toho hodnotu V t.ž. $\frac{y(\vec{w}^\top \vec{x}+b)}{||\vec{w}||} \geq V$ pre všetky trénovacie príklady $(\vec{x}, \vec{y})$.

Takáto nadrovina sa zvykne nazývať aj **klasifikátorom s maximalizovanou hraničnou oblasťou** (angl. maximum margin classifier), kde hraničnou oblasťou je myslená oblasť tvorená všetkými bodmi príznakového priestoru, ktoré sa nachádzajú od nadroviny v kratšej vzdialenosti ako najbližší trénovací príklad, teda body spĺňajúce $\frac{|\vec{w}^\top \vec{x}+b|}{||\vec{w}||} < V$, kde $\vec{x} \in \mathbb{R}^n$ je vektor súradníc daného bodu.

Trénovacie príklady majúce najkratšiu vzdialenosť k nadrovine sa nazývajú **podporné vektory** (angl. support vectors), pretože v istom zmysle určujú (podopierajú) túto nadrovinu. Sú to teda príklady $(\vec{x}, y)$, pre ktoré platí $D((\vec{x}, y), (\vec{w}, b)) = V$.

Toto základné použitie SVM sa nazýva tiež lineárne SVM, keďže jeho použitie je vhodné len pre lineárne separovateľné dáta. Ako popíšeme v ďalšej časti, je možné ho rozšíriť pre použitie na nelineárnych dátach s využitím tzv. kernelovských funkcií alebo s použitím regularizácie.

2.1.1 Základný postup riešenia problému

Na základe predchádzajúceho popisu sa dá odvodiť, že tento problém je možné formulovať ako nasledujúci **optimalizačný problém** [15]:

$$\begin{aligned} \min_{\vec{w},b} \quad & \frac{1}{2}||\vec{w}||^2 \\ \text{pričom} \quad & y_i(\vec{w}^\top \vec{x_i} + b) \geq 1, \ i = 1, \ldots, m \end{aligned} \tag{2.1}$$

kde m je počet trénovacích príkladov a i-ty príklad je dvojica $(\vec{x_i}, y_i)$, pričom $y_i \in \{-1, 1\}$ je jeho označenie a $\vec{x_i} \in \mathbb{R}^n$ je jeho príznakový vektor.

Hoci tento problém je riešiteľný pomocou kvadratického programovania a jeho riešenie by bolo aj riešením nášho problému, zaujímavý bude pohľad na riešenie jeho duálneho optimalizačného problému najmä kvôli jeho rozšíriteľnosti pre použitie kernelovských funkcií. Silno **duálny** optimalizačný problém pre náš primálny problém je možné formulovať na základe princípu Lagrangeovej duality ako [15]:

$$\begin{aligned} \max_{\vec{\alpha}} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \vec{x_i}, \vec{x_j} \rangle \\ \text{pričom} \quad & \alpha_i \geq 0, \ i = 1, \ldots, m, \\ & \sum_{i=1}^{m} \alpha_i y_i = 0. \end{aligned} \tag{2.2}$$

pritom po nájdení optimálnej hodnoty $\vec{\alpha}$ nájdeme optimálne hodnoty $\vec{w}$ a b ako:

$$\vec{w}^* = \sum_{i=1}^{m} \alpha_i y_i \vec{x_i} \tag{2.3}$$

$$b^* = -\frac{\max_{(x_i,y_i)\in C_{neg}} (\vec{w}^*)^\top \vec{x_i} - \min_{(x_i,y_i)\in C_{poz}} (\vec{w}^*)^\top \vec{x_i}}{2}. \tag{2.4}$$

kde C_{poz} je množina tých trénovacích príkladov, ktorých označenie je 1 a analogicky C_{neg} je množina tých trénovacích príkladov, ktorých označenie je -1.

2.1.2 Kernelový trik

Môžeme si všimnúť, že duálny optimalizačný problém je zapísaný v tvare, v ktorom vektory príznakov trénovacích príkladov vystupujú len v skalárnom súčine ($\langle \vec{x_i}, \vec{x_j} \rangle = \vec{x_i}^\top \vec{x_j}$). Výmenou tohto skalárneho súčinu za inú funkciou týchto dvoch vektorov je možné simulovať transformáciu vstupných dát do viacrozmerného vektorového priestoru, v ktorom môžu byť lineárne separovateľné, aj keď pôvodne neboli a rozšíriť tak využitie SVM aj na niektoré pôvodne lineárne neseparovateľné dáta. Takáto funkcia, nazvime ju K, teda musí spĺňať $K(\vec{x_i}, \vec{x_j}) = f(\vec{x_i})^\top f(\vec{x_j}) = \langle f(\vec{x_i}), f(\vec{x_j}) \rangle$, kde $f(\vec{x})$ je nejaká funkcia transformujúca vektor $\vec{x}$ na vektor z iného vektorového priestoru príznakov. Funkciu K potom nazývame **kernelová funkcia** alebo **kernel** a táto výmena je známa ako **kernelový trik** (angl. kernel trick) [15]. Výhodou tohto prístupu je, že nevyžaduje, aby sme priamo počítali transformáciu dát $f(\vec{x})$, ale len vektorový súčin $\langle f(\vec{x_i}), f(\vec{x_j}) \rangle$, čo je častokrát omnoho efektívnejšie.

Medzi populárne kernelové funkcie patria [17]:

1. Lineárna kernelová funkcia: $K(\vec{x_i}, \vec{x_j}) = \langle \vec{x_i}, \vec{x_j} \rangle$
2. Polynomiálna kernelová funkcia: $K(\vec{x_i}, \vec{x_j}) = (\langle \vec{x_i}, \vec{x_j} \rangle + c)^d$
3. Gaussova RBF (radial basis function): $K(\vec{x_i}, \vec{x_j}) = e^{-\gamma ||\vec{x_i} - \vec{x_j}||^2}$ pre $\gamma > 0$
4. Sigmoidná kernelová funkcia: $K(\vec{x_i}, \vec{x_j}) = tanh(\gamma \langle \vec{x_i}, \vec{x_j} \rangle + r)$

2.1.3 Regularizácia

Ďalšou technikou, ktorou je možné upraviť algoritmus SVM tak, aby bol účinne aplikovateľný aj na nelineárne dáta a zároveň bol menej náchylný na pomýlenie šumom v dátach je **regularizácia** [15]. Tá umožňuje modelu dovoliť, aby sa niektoré trénovacie príklady nachádzali v hraničnej oblasti. Počet a vzdialenosť týchto bodov od nadroviny je však samozrejme tiež predmetom minimalizácie. Pridaním regularizácie teda

dostávame primálny problém:

$$\begin{aligned} \min_{\vec{w},b} \quad & \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{m}\xi_i \\ \text{pričom} \quad & y_i(\vec{w}^\top \vec{x_i} + b) \geq 1 - \xi_i,\ i = 1,\ldots,m \\ & \xi_i \geq 0,\ i = 1,\ldots,m \end{aligned} \tag{2.5}$$

kde ξ_i je miera prekročenia hranice hraničnej oblasti príznakového vektora $\vec{x_i}$ i-teho trénovacieho príkladu a C je parametrizácia veľkosti pokuty, ktorá je za toto prekročenie udelená. Jeho silno duálny problém určený pre optimalizáciu je potom:

$$\begin{aligned} \max_{\vec{\alpha}} \quad & \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \vec{x_i}, \vec{x_j} \rangle \\ \text{pričom} \quad & 0 \leq \alpha_i \leq C,\ i = 1,\ldots,m \\ & \sum_{i=1}^{m}\alpha_i y_i = 0 \end{aligned} \tag{2.6}$$

2.2 Rozhodovacie stromy

Rozhodovací strom (angl. decision tree) [20] je algoritmus používaný v rámci učenia s učiteľom pre klasifikáciu aj regresiu. My budeme v nasledujúcom texte najčastejšie popisovať vlastnosti/použitie klasifikačných rozhodovacích stromov, keďže problém riešený v rámci tejto práce je klasifikačný.

Rozhodovací strom má štruktúru zakoreneného orientovaného stromového grafu, ktorá zachytáva postupné rozdeľovanie vektorového priestoru príznakov trénovacích/testovacích príkladov na základe ich hodnôt. Jeho vnútorné uzly sa zvyknú nazývať **testovacie uzly** (angl. test nodes) a jeho listy sa zvyknú nazývať **rozhodovacie uzly** (angl. decision nodes). Každý testovací uzol predstavuje rozdelenie príznakového priestoru na niekoľko častí a hrany z neho vedúce predstavujú tieto časti. Každému rozhodovaciemu uzlu klasifikačného rozhodovacieho stromu je priradená jedna z rozlišovaných tried.

V prípade klasifikačného rozhodovacieho stromu klasifikácia nového príkladu zodpovedá ceste z koreňa do listov tvorenej na základe toho, do ktorej časti príznakového priestoru patria hodnoty príznakov daného príkladu. Trieda zodpovedajúca listu nachádzajúcemu sa na konci tejto cesty je potom predpovedanou triedou pre daný príklad.

Zložitosť stromu je najčastejšie určená počtom jeho uzlov alebo listov, jeho hĺbkou, alebo počtom príznakov, na základe ktorých sa príznakový priestor rozdeľuje. Táto zložitosť sa kontroluje technikami ako sú **kritériá pre zastavenie** rozdeľovania danej vetvy stromu a **orezávanie** (angl. pruning) stromu.

Existuje viacero algoritmov pre tvorbu rozhodovacích stromov na základe trénovacieho datasetu. Tieto je možné rozdeliť na algoritmy, ktoré vytvárajú strom **zhora-**

nadol a tie, ktoré ho vytvárajú **zdola-nahor**. V literatúre však zásadne prevažuje prvá skupina algoritmov. Medzi tie patria napríklad algoritmus **ID3** [18], **C4.5** [19], **CART** [5].

Algoritmus 1 [20] pre klasifikačné rozhodovacie stromy predstavuje základnú koncepciu používanú algoritmami typu „zhora-nadol“ rozhodujúcich o rozdelení príznakového priestoru vždy na základe práve jedného príznaku (čo sa používa vo veľkej väčšine aplikácií), pričom príznaky (ako náhodné premenné) nadobúdajú hodnoty z konečnej množiny. Je to pažravý algoritmus typu „rozdeľuj a panuj“ (angl. divide and conquer) a daný rozhodovací strom vytvára na základe danej trénovacej množiny rekurzívne. Pri každom rekurzívnom volaní vytvorí jeden uzol, pričom zároveň nájde taký príznak, na základe ktorého sa najviac vyplatí rozdeliť príznakový priestor, a to na základe tzv. **rozdeľovacieho kritéria**. Po zvolení najlepšieho možného rozdelenia sa pokračuje v rekurzívnom vytváraní nasledujúcich uzlov, až pokým nie sú splnené **kritériá pre zastavenie** rozdeľovania danej vetvy stromu alebo žiadny príznak nezíska dostatočnú hodnotu rozdeľovacieho kritéria pre pokračovanie delenia danej vetvy.

V nasledujúcom texte budeme označovať ako $dom(P)$ doménu hodnôt príznaku P (ako náhodnej premennej) a $\sigma_{P=c}(M)$ definujeme ako selekciu tých príkladov z množiny M, ktorých príznak P nadobúda hodnotu c.

2.2.1 Rozdeľovacie kritériá

Existuje viacero druhov rozdeľovacích kritérií. Najčastejšie používané sú však kritériá, ktoré rozhodujú vhodnosť rozdelenia na základe jedného príznaku. Medzi tie patria napríklad kritériá **informačný zisk** a **gini index**. Definujeme ich pre príznaky, ktoré nadobúdajú konečnú množinu hodnôt:

Informačný zisk

Informačný zisk (angl. information gain) [20] je kritérium, ktoré pre daný príznak P určí, aké výhodné je na ňom rozdeliť trénovaciu množinu M na základe zmeny entropie premennej Y (označení trénovacích príkladov z M) po tomto rozdelení. Formálne:

$$IZ(P, M) = H(Y, M) - \sum_{c \in dom(P)} \frac{|\sigma_{P=c}M|}{|M|} H(Y, \sigma_{P=c}M) \tag{2.7}$$

kde $H(Y, M)$ je entropia premennej Y v množine M. Formálne:

$$H(Y, M) = -\sum_{o \in dom(Y)} \frac{|\sigma_{Y=o}M|}{|M|} \log_2 \frac{|\sigma_{Y=o}M|}{|M|} \tag{2.8}$$

Algoritmus 1 Základná koncepcia používaná pre tvorbu klasifikačného rozhodovacieho stromu s príznakmi s konečnou množinou potenciálnych hodnôt

M je množina trénovacích príkladov
Π je množina príznakov trénovacích príkladov (ako náhodných premenných)
Y je označenie trénovacích príkladov (ako náhodnej premennej)
function VytvorStrom(M, Π, Y)
  Vytvor nový strom S obsahujúci len koreňový uzol u.
  if Je splnené niektoré z **kritérií pre zastavenie then**
    Označ koreň u za list a jeho hodnotu nastav na najčastejšie sa vyskytujúcu hodnotu premennej Y v M.
  else
    Nájdi príznak P z Π, ktorý dosahuje najlepšiu hodnotu **rozdeľovacieho kritéria** (skr. $nhrk$).
    if $nhrk >$ hraničná hodnota **then**
      Označ uzol u príznakom P.
      for all v_i z $dom(P)$ **do**
        $S_i =$ VytvorStrom($\sigma_{P=v_i} M, \Pi, Y$)
        Vytvor orientovanú hranu z uzla u do stromu S_i označenú ako v_i.
      end for
    else
      Označ koreň u za list a jeho hodnotu nastav na najčastejšie sa vyskytujúcu hodnotu Y v M.
    end if
  end if
  return S
end function

Gini index

Gini index [20] je kritérium, ktoré pre daný príznak P určí, aké výhodné je na ňom rozdeliť trénovaciu množinu M na základe rozdielnosti pravdepodobnostných rozdelení hodnôt premennej Y (označení trénovacích príkladov z M) po tomto rozdelení. Formálne:

$$GI(P, M) = GINI(Y, M) - \sum_{c \in dom(P)} \frac{|\sigma_{P=c} M|}{|M|} GINI(Y, \sigma_{P=c} M) \tag{2.9}$$

kde $GINI(Y, M)$ je definovaná ako:

$$GINI(Y, M) = 1 - \sum_{o \in dom(Y)} \left(\frac{|\sigma_{Y=o} M|}{|M|} \right)^2 \tag{2.10}$$

2.2.2 Kritériá pre zastavenie rozdeľovania danej vetvy stromu a orezávanie stromu

Ukončenie rastu danej vetvy stromu je dôležité pre zabránenie nárastu zložitosti daného stromu a následnému preučeniu. Medzi najčastejšie používané kritériá patria:

1. Dosiahnutie maximálnej hĺbky
2. Všetky trénovacie príklady z tejto časti príznakového priestoru patria do rovnakej triedy
3. Najlepšia hodnota rozdeľovacieho kritéria je nižšia než hraničná hodnota
4. Ďalšie rozdelenie priestoru by znamenalo zmenšenie počtu príkladov v rozdelenom priestore pod hraničnú hodnotu

Ďalšou technikou pre zmenšenie veľkosti stromu je tzv. **orezávanie** stromu. Táto technika je založená na použití miernych kritérií pre zastavenie rozdeľovania vetvy stromu, čo by malo za normálnych okolností za následok preučenie stromu na danej trénovacej množine. Orezávanie stromu však veľkosť tohto stromu zmenší odstránením niektorých jeho vetiev, ktoré neprispievajú k jeho zovšeobecneniu [20].

2.2.3 Algoritmy pre tvorbu rozhodovacích stromov

ID3

ID3 [20] je jednoduchý typ algoritmu tvoriaci rozhodovací strom na základe kritéria informačného zisku. Zastavenie rozdeľovania danej vetvy stromu nastáva, keď všetky

príklady z danej časti príznakového priestoru patria do rovnakej triedy alebo keď najlepší možný informačný zisk je menší alebo rovný nule. ID3 nepoužíva žiadnu techniku orezávania stromu a nedokáže spracovať príznaky, ktorých hodnoty sú z množiny reálnych čísel.

C4.5

C4.5 [20] je vylepšením ID3 algoritmu. Používa pokročilejšie rozdeľovacie kritérium. Rast danej vetvy stromu je ukončený, keď by počet trénovacích príkladov patriacich do novovzniknutej časti rozdelenia klesol pod hraničnú hodnotu. Výhodou je tiež, že používa orezávanie stromu a dokáže spracovať aj príznaky, ktorých hodnoty sú z množiny reálnych čísel.

CART

CART (classification and regression trees) [20] je algoritmus tvoriaci binárne rozhodovacie stromy. Jeho hlavnou výhodou je jeho schopnosť tvoriť okrem klasifikačných stromov aj regresné stromy. Rovnako ako C4.5 algoritmus, používa orezávanie stromov.

2.2.4 Náhodný les

Náhodný les (angl. random forest) [13] je algoritmus založený na metóde bootstrapovej agregácie (angl. bootstrap aggregation skr. **bagging**) [4] použitej na rozhodovacie stromy (angl. decision trees).

Bagging funguje na základe trénovania skupiny rozhodovacích stromov na náhodných bootstrapových vzorkách z trénovacích dát, čiže náhodného výberu trénovacích príkladov s návratom, čo znamená, že jeden príklad môže byť vybratý viackrát. Pre klasifikáciu potom používa systém **väčšinového hlasovania** (angl. majority vote), t.j. vstupný príklad je klasifikovaný všetkými rozhodovacími stromami z danej skupiny, pričom predpovedaná trieda je tá, ktorá bola zvolená väčšinou stromov [4].

Náhodný les rozširuje bagging o ďalšiu vrstvu náhodnosti tým, že pri každom rozdeľovaní príznakového priestoru sa berie do úvahy len náhodná podmnožina príznakov, ako ukazuje nasledujúci algoritmus [13]:

Algoritmus náhodný les:

1. Vyber n bootstrapových vzoriek z trénovacích dát.

2. Pre každú zo vzoriek vytvor rozhodovací strom **bez** použitia orezávania s tým, že pri tvorbe každého z testovacích uzlov sa rozhoduj o rozdelení na základe náhodne zvolenej podmnožiny m príznakov

3. Predikuj nové dáta agregáciou predikcií jednotlivých stromov (buďto väčšinovým hlasovaním pri klasifikácii alebo spriemerovaním predikcií pri regresii)

2.3 Analýza hlavných komponentov

Cieľom analýzy hlavných komponentov [3] je redukcia zložitosti (dimenzionality) dát so zachovaním čo najväčšieho množstva informácie.

Ak sa na príznaky dát pozeráme ako na náhodné premenné, potom analýza hlavných komponentov je vlastne konštrukcia nových náhodných premenných ako lineárnej kombinácie počiatočných. Tieto potom nazývame hlavné komponenty, pričom ale musí platiť, že prvý komponent má najväčší možný rozptyl (ktorý sa v tomto prípade používa ako miera množstva popisovanej informácie), druhý komponent má druhý najväčší možný rozptyl a tak ďalej. Prípadným vynechaním komponentov počínajúc od tých s najmenším rozptylom sa potom dosahuje redukcia dimenzionality dát s minimalizovanou stratou informácie.

Algoritmus analýzy hlavných komponentov:

1. Vstup je matica X dát veľkosti $I \times J$, kde I je počet príkladov a J je dimenzionalita dát (počet príznakov každého príkladu), pričom predpokladáme, že dáta sú normalizované tak, že priemer každého stĺpca (zodpovedajúceho jednému príznaku) je nulový.

2. Vypočítame kovariančnú maticu C príznakov trénovacích príkladov ako $C = \frac{1}{I} X^\top X$.

3. Keďže C je štvorcová symetrická matica, existujú (a dajú sa vypočítať) ortogonálna matica P a diagonálna matica D, také že $C = PDP^\top$. Pritom stĺpce matice P sú vlastné vektory matice C, ktoré sú po dvojiciach kolmé a čísla na diagonále matice D sú im prislúchajúce vlastné čísla.

4. Každý vlastný vektor a vlastné číslo zodpovedá jednému hlavnému komponentu. Platí pritom, že usporiadaním vektorov podľa im prislúchajúcich vlastných čísel od najväčšieho po najmenšie získame vektory v poradí, v akom prislúchajú komponentu s najväčším, druhým najväčším atď. rozptylom. Zostrojíme teda maticu $\bar{P}$ usporiadaním stĺpcových vektorov matice P podľa ich vlastných čísel od najväčšieho po najmenšie s prípadným vynechaním niekoľkých posledných vlastných vektorov (stĺpcov) pre redukciu dimenzionality.

5. Nové dáta vyjadrené v premenných (príznakoch) prislúchajúcich ponechaným hlavným komponentom potom získame ako maticu $\bar{X} = X\bar{P}$.

Medzi najpodstatnejšie využitie PCA patrí redukcia výpočtových nárokov, ktoré sú pri spracovaní mnohorozmerných dát vysoké. Ďalším významným využitím je možnosť grafického znázornenia dát menšej dimenzie a zjednodušenie pozorovania trendov alebo výchyliek v dátach.

2.4 Neurónové siete

2.4.1 Dopredné neurónové siete

V prípade paradigmy učenia s učiteľom je úlohou dopredných neurónových sietí [7] aproximácia nejakej funkcie f na základe vstupu vo forme trénovacích príkladov, ktoré pre každý príznakový vektor $\vec{x}$ obsahujú očakávaný výstup funkcie $f(\vec{x})$. Dopredná neurónová sieť definuje funkciu $d(\vec{x}, \vec{\theta})$ aproximujúcu $f(\vec{x})$, pričom výsledné správanie funkcie je možné upraviť modifikáciou jej parametrov $\vec{\theta}$. Toto je úlohou tzv. učiaceho algoritmu (angl. learning algorithm), ktorého cieľom je nájsť také $\vec{\theta}$, pre ktoré funkcia d najlepšie aproximuje funkciu f. Tento využíva tzv. účelovú funkciu (angl. loss function) ako mieru toho, ako moc sa funkcia d odlišuje od funkcie f.

Základným stavebným prvkom neurónovej siete sú tzv. **umelé neuróny**. Výpočet neurónu je daný jeho **aktivačnou funkciou** σ, **odchýlkou** b (angl. bias) a **ohodnotením** vstupov váhami $\vec{w} = (w_1, \ldots, w_k)^\top$. Jeho vstupom je vektor hodnôt $\vec{h} = (h_1, \ldots, h_k)^\top$. Výstupom neurónu je potom hodnota $y = \sigma(\vec{h}^\top \vec{w} + b)$ [8].

Samotná sieť sa skladá z postupnosti vrstiev neurónov $(V_1, \ldots, V_n)$, pričom platí, že vstupom pre neuróny vrstvy V_{l+1} môže byť len podmnožina výstupov neurónov vrstvy V_l. Vrstvu V_1 nazývame **vstupná vrstva** a vrstvu V_n nazývame **výstupná vrstva**. Zvyšné vrstvy sa nazývajú **skryté** [16].

Označme $w^l_{j,k}$ pre každé $l \in \{2, \ldots, n\}$ ako ohodnotenie hrany vedúcej z k-teho neurónu vo vrstve V_{l-1} do j-teho neurónu vo vrstve V_l. Potom nech b^l_j označuje výchylku j-teho neurónu vrstvy V_l. Definujme maticu W^l ako maticu, ktorej prvok v j-tom riadku a k-tom stĺpci je $w^l_{j,k}$, skrátene označíme ako $W^l = (w^l_{j,k})$. Obdobne budú definované aj matice váh v nasledujúcich častiach, kde už podobnú definíciu nebudeme spomínať. Definujeme vektor $\vec{b}^l$ ako $\vec{b}^l = (b^l_1, \ldots, b^l_k)^\top$, kde k je počet neurónov vrstvy V_l. Ak σ je používaná aktivačná funkcia, tak pre každé $l \in \{2, \ldots, n\}$ platí, že výstupom neurónov vrstvy V_l je vektor $\vec{y}^l = \sigma(W^l \vec{y}^{l-1} + \vec{b}^l)$.[1] Výstupom neurónov vstupnej vrstvy je vektor vstupu neurónovej siete, teda vektor príznakov $\vec{x}$ nejakého trénovacieho/testovacieho príkladu.[2] Platí teda $\vec{y}^1 = \vec{x}$. Výstup poslednej vrstvy $\vec{y}^n$ by mal aproximovať výstup funkcie f, teda $f(\vec{x})$ [16].

[1] Funkciu σ definujeme vo vektorizovanej podobe t.j. pre ľubovoľný vektor $\vec{v} = (v_1, \ldots, v_n)^\top$ definujeme $\sigma(\vec{v}) = (\sigma(v_1), \ldots, \sigma(v_n))^\top$.

[2] Neuróny vrstvy V_1 teda nezodpovedajú definícii neurónu vyššie.

Z uvedeného vyplýva, že dopredná neurónová sieť môže byť popísaná ohodnoteným acyklickým orientovaným grafom, ktorého vrcholy predstavujú jednotlivé neuróny a ohodnotené hrany predstavujú ohodnotené vstupy/výstupy jednotlivých neurónov.

Návrh modelu neurónovej siete vo všeobecnosti zahrnuje výber viacerých parametrov, a to najmä výber učiaceho algoritmu, účelovej funkcie, aktivačnej funkcie neurónov, počtu vrstiev/neurónov a prepojení medzi jednotlivými vrstvami.

Ako učiaci algoritmus sa najčastejšie používa algoritmus gradientného zostupu (angl. gradient descent), pričom pre výpočet gradientu sa používa algoritmus spätného šírenia chyby (angl. backpropagation).

Gradientný zostup

Gradientný zostup [21] je algoritmus pre minimalizáciu účelovej funkcie $J(\vec{\theta})$ vzhľadom na parametre modelu $\vec{\theta}$ zmenou hodnôt týchto parametrov v smere opačnom ku gradientu účelovej funkcie vzhľadom na parametre $\vec{\theta}$. Formálne:

$$\vec{\theta}^{n+1} = \vec{\theta}^{n} - \eta \cdot \nabla_{\vec{\theta}} J(\vec{\theta}^{n}) \tag{2.11}$$

kde parameter η udáva veľkosť kroku učenia a θ^i sú hľadané parametre v i-tom kroku. Na výpočet gradientu sa pritom v každom kroku učenia častokrát využíva nejaká (v každom kroku iná) náhodne zvolená podmnožina trénovacích dát (angl. batch).

Účelová funkcia krížovej entropie

Ako ukážku účelovej funkcie sme v našej práci vybrali účelovú funkciu krížovej entropie (angl. cross-entropy loss), keďže je to funkcia používaná aj neurónovou sieťou DeepGRU používanej v našej práci. Je používaná najmä v oblasti logistickej regresie a neurónových sietí.

Pre binárnu klasifikáciu môže byť účelová funkcia krížovej entropie [15] odvodená z funkcie:

$$L(\vec{\theta}) = \prod_{i=1}^{n} \sigma_{f_{\vec{\theta}}}(\vec{x}_i)^{y_i} (1 - \sigma_{f_{\vec{\theta}}}(\vec{x}_i))^{1-y_i} \tag{2.12}$$

kde n je počet trénovacích príkladov a pre každé $i \in \{1, \ldots, n\}$ je (x_i, y_i) trénovací príklad, kde x_i je jeho príznakový vektor a y_i je jeho značenie. Funkcia $\sigma_{f_{\vec{\theta}}}$ označuje hypotézu, čo je funkcia priraďujúca príznakovému vektoru predikované označenie. V tomto prípade ide o zloženie sigmoidnej funkcie a danej funkcie $f_{\vec{\theta}}$ na príznakoch trénovacích príkladov parametrizovanou parametrami $\vec{\theta}$. Teda $\sigma_{f_{\vec{\theta}}} = \frac{1}{1+e^{-f_{\vec{\theta}}(x)}}$ (funkcia $f_{\vec{\theta}}$ býva často lineárna funkcia) a obor hodnôt funkcie $\sigma_{f_{\vec{\theta}}}$ je interval $[0, 1]$ a nie diskrétna množina $\{0, 1\}$. Na základe jej výstupu však môžeme podľa zvoleného prístupu jej predpovedané diskrétne označenie určiť (napríklad pre hodnoty menšie ako $0,5$ predikovať 0 a pre hodnoty $\geq 0,5$ predikovať 1). Maximalizáciou funkcie (2.12) teda maximalizujeme

počet prípadov, kedy sa očakávaný a predikovaný výstup rovnajú. Logaritmizáciou a následným otočením znamienka tejto funkcie dostaneme účelovú funkciu krížovej entropie určenú pre minimalizáciu ako:

$$l(\vec{\theta}) = -\sum_{i=1}^{n} \left(y_i \ln(\sigma_{f_{\vec{\theta}}}(\vec{x}_i)) + (1 - y_i) \ln(1 - \sigma_{f_{\vec{\theta}}}(\vec{x}_i)) \right) \tag{2.13}$$

Algoritmus spätného šírenia chyby

Algoritmus spätného šírenia chyby je algoritmus využívaný v neurónových sieťach na výpočet gradientu účelovej funkcie používanej v danej neurónovej sieti vzhľadom na parametre tejto siete.

Nech $\vec{x}$ je vektor vstupných hodnôt siete. Nech $(V_1, \ldots, V_n)$ sú vrstvy neurónovej siete. Nech σ je používaná aktivačná funkcia všetkých neurónov a C je používaná účelová funkcia. Pre každé $l \in \{2, \ldots, n\}$ sú matica W^l a vektory b^l a y^l definované ako v predchádzajúcom texte. Nech pre každé $l \in \{2, \ldots, n\}$ vektor $\vec{z}^l = W^l \vec{y}^{l-1} + \vec{b}^l$. Potom výstup neurónov vrstvy V_l môžeme zapísať ako $\vec{y}^l = \sigma(\vec{z}^l)$. Očividne $y_j^l = \sigma(\sum_k w_{j,k}^l y_k^{l-1} + b_j^l)$ a podobne pre z_j^l.

Označme $\delta_j^l = \frac{\partial C}{\partial z_j^l}$ a nazvime to chybou j-teho neurónu vrstvy V_l. Vektor $\vec{\delta}^l$ definujeme analogicky k prechádzajúcim definíciám. Označme ako $\odot$ Hadamardov súčin matíc. Algoritmus spätného šírenia chyby je potom založený na nasledujúcich platných rovnostiach, ktorých dôkaz však v tejto práci neuvádzame:[16]

$$\vec{\delta}^n = \nabla_y C \odot \sigma'(\vec{z}^n), \tag{2.14}$$

$$\vec{\delta}^l = ((W^{l+1})^\top \vec{\delta}^{l+1}) \odot \sigma'(\vec{z}^l), \tag{2.15}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l, \tag{2.16}$$

$$\frac{\partial C}{\partial w_{j,k}^l} = y_k^{l-1} \delta_j^l, \tag{2.17}$$

kde $\nabla_y C$ je stĺpcový vektor, ktorého komponenty sú prvky $\frac{\partial C}{\partial y_j^n}$.

Majme daný algoritmus gradientného zostupu, ktorý v každom kroku učenia na výpočet gradientu účelovej funkcie používa m trénovacích príkladov (jeden batch). Tento algoritmus potom s využitím algoritmu spätného šírenia chyby na výpočet gradientu funguje nasledovne [16]:

1. Pre každý trénovací príklad $\vec{x}_i$ vykonaj **algoritmus spätného šírenia chyby** nasledovne:
   (a) Nastavenie výstupu $\vec{y}^{1,i}$ neurónov vrstvy V_1 na vektor vstupu $\vec{x}_i$.
   (b) Výpočet výstupov zvyšných vrstiev $V_2, \ldots, V_n$:
   Pre každé $l \in \{2, \ldots, n\}$: $\vec{z}^{l,i} = W^l \vec{y}^{l-1,i} + \vec{b}^l$ a $\vec{y}^{l,i} = \sigma(\vec{z}^{l,i})$

(c) Výpočet chyby výstupnej vrstvy $\vec{\delta}^{n,i}$ (podľa 2.14): $\vec{\delta}^{n,i} = \nabla_{y,i} C \odot \sigma'(\vec{z}^{n,i})$

(d) Spätné šírenie chyby (podľa 2.15):
Pre každé $l \in \{n-1, n-2, \ldots, 2\}$: $\vec{\delta}^{l,i} = ((W^{l+1})^\top \vec{\delta}^{l+1,i}) \odot \sigma'(\vec{z}^{l,i})$

(e) Výpočet hodnoty gradientu pre trénovací príklad $\vec{x}_i$. Pre každé $l \in \{n, n-1, \ldots, 2\}$, a pre každé $j \in \{1, \ldots, p\}$; $k \in \{1, \ldots, q\}$, t.ž. matica W^l je typu $p \times q$: $\frac{\partial C}{\partial w^l_{j,k}} = y^{l-1,i}_k \delta^{l,i}_j$ (podľa 2.17) a $\frac{\partial C}{\partial b^l_j} = \delta^{l,i}_j$ (podľa 2.16)

2. Vypočítaj hodnotu každej časti gradientu ako priemer za všetky trénovacie príklady:
Pre každé $l \in \{n, n-1, \ldots, 2\}$, a pre každé $j \in \{1, \ldots, p\}$; $k \in \{1, \ldots, q\}$, t.ž. matica W^l je typu $p \times q$: $\frac{\partial C}{\partial w^l_{j,k}} = \frac{1}{m} \sum_i y^{l-1,i}_k \delta^{l,i}_j$ a $\frac{\partial C}{\partial b^l_j} = \frac{1}{m} \sum_i \delta^{l,i}_j$

3. Aktualizuj hodnoty parametrov pomocou gradientu:
Pre každé $l \in \{n, n-1, \ldots, 2\}$, a pre každé $j \in \{1, \ldots, p\}$; $k \in \{1, \ldots, q\}$, t.ž. matica W^l je typu $p \times q$: $w^l_{j,k} \leftarrow w^l_{j,k} - \frac{\eta}{m} \sum_i y^{l-1,i}_k \delta^{l,i}_j$ a $b^l_j \leftarrow b^l_j - \frac{\eta}{m} \sum_i \delta^{l,i}_j$

2.4.2 Rekurentné neurónové siete

Rekurentné neurónové siete [22] sa od dopredných neurónových sietí odlišujú v tom, že v nich existuje prepojenie neurónov tvoriace časový alebo sekvenčný cyklus. Preto sú tieto siete schopné učiť sa dlhodobé závislosti medzi prvkami nejakej (časovej) sekvencie dát (trénovacích/testovacích príkladov). Pre jednoduchosť túto sekvenciu budeme považovať za časovú.

Fungovanie rekurentnej neurónovej siete vysvetlíme na zjednodušenom modeli, ktorý obsahuje 3 vrstvy: vstupnú, rekurentnú skrytú a výstupnú.

Vstupná vrstva obsahuje n neurónov, ktorých výstupom je postupne časová sekvencia $(\ldots, \vec{x}^{t-1}, \vec{x}^t, \vec{x}^{t+1}, \ldots)$, pričom $\vec{x}^t = (x^t_1, \ldots, x^t_n)^\top$. Tieto neuróny sú prepojené s neurónmi skrytej vrstvy hranami ohodnotenými maticou váh W^{vs}.

Skrytá vrstva obsahuje m neurónov, ktorých výstupná hodnota je v každom čase t vektor $\vec{s}^t = (s^t_1, \ldots, s^t_m)^\top$, ktorý je definovaný ako:

$$\vec{s}^t = f_s(\vec{z}^t), \tag{2.18}$$

kde f_s je aktivačná funkcia neurónov skrytej vrstvy a $\vec{z}^t = W^{vs}\vec{x}^t + W^{ss}\vec{s}^{t-1} + \vec{b}^s$, pričom $\vec{b}^s$ je vektor výchyliek neurónov skrytej vrstvy a $\vec{s}^{t-1}$ je vektor výstupov neurónov skrytej vrstvy v čase $t-1$. Tieto neuróny sú teda rekurentne prepojené v čase, pričom tieto prepojenia sú ohodnotené maticou váh W^{ss}.

Sieť obsahuje p neurónov výstupnej vrstvy, ktoré sú prepojené s neurónmi skrytej vrstvy, a ktorých výstupom je v každom čase t vektor $\vec{y}^t = (y_1, \ldots, y_p)^\top$, ktorý je definovaný ako:

$$\vec{y}^t = f_v(W^{sv}\vec{s}^t + \vec{b}^v) \tag{2.19}$$

kde $\vec{b}^v$ je výchylka neurónov výstupnej vrstvy a f_v je ich aktivačná funkcia a matica váh W^{sv} je matica ohodnotení prepojení medzi skrytou a výstupnou vrstvou.

Tieto výpočty sa opakujú pre každý čas t a jemu zodpovedajúci vstup $\vec{x}^t$, ktorému zas zodpovedá výstup siete $\vec{p}^t$.

Rekurentná neurónová sieť teda dokáže v každom čase obsahovať informáciu o stave siete v množstve predchádzajúcich časov. Táto dôležitá informácia dokáže pomôcť neurónovej sieti robiť presnejšie výstupné predikcie.

Gradient chybovej funkcie vzhľadom na parametre tejto siete sa počíta pomocou priamočiareho rozšírenia algoritmu spätného šírenia chyby pre dopredné neurónové siete. V tejto práci ho ale neuvádzame.

Bránové rekurentné jednotky

Siete bránových rekurentných jednotiek (angl. gated recurrent units skr. GRU) [6] sú typom rekurentných neurónových sietí, ktorý vznikol ako jedno z riešení tzv. problému miznúceho gradientu. Takto je označovaný problém toho, že rekurentná sieť po istom čase stratí schopnosť uchovávať informáciu o dlhodobých závislostiach v postupnosti dát, čo je spôsobené zmenšovaním veľkosti gradientu pri spätnom šírení chyby v čase [22].

Sieť je definovaná výstupom jej bránových jednotiek $\vec{s}^t$ v každom čase t, ktorý je určený nasledujúcou sadou rovníc aplikovaných v čase t na ich vstup $\vec{x}^t$ a ich výstup z času $t-1$ [6, 14]:

$$\text{aktualizačná brána (angl. update gate): } \vec{a}^t = \sigma(W^a\vec{x}^t + U^a\vec{s}^{t-1} + \vec{b}^a) \tag{2.20}$$

$$\text{obnovovacia brána (angl. reset gate): } \vec{r}^t = \sigma(W^r\vec{x}^t + U^r\vec{s}^{t-1} + \vec{b}^r) \tag{2.21}$$

$$\text{potenciálny výstup (angl. candidate gate): } \vec{p}^t = tanh(W^p\vec{x}^t + U^p(\vec{r}^t \odot \vec{s}^{t-1}) + \vec{b}^p) \tag{2.22}$$

$$\text{výstup: } \vec{s}^t = \vec{s}^{t-1} \odot (1 - \vec{a}^t) + \vec{p}^t \odot (\vec{a}^t) \tag{2.23}$$

Označenie σ v tomto prípade predstavuje sigmoidnú funkciu a W^a, U^a, W^r, U^r, W^p, U^p, b^a, b^r a b^p označujú matice ohodnotení prepojení neurónov a ich výchylky pre každé použitie vstupu. Brána (2.21) umožňuje modelu optimalizovať, aké množstvo informácie z predchádzajúceho výstupu jednotky sa oplatí zabudnúť pri výpočte potenciálneho nového výstupu jednotky (2.22). Brána (2.20) zas umožňuje modelu optimalizovať pomer informácie z potenciálneho nového a predchádzajúceho výstupu vo výstupe jednotky (2.23).

2.4.3 Deep Gesture Recognition Utility skr. DeepGRU

Model neurónovej siete *DeepGRU* [14] používaný v našej práci sa skladá z troch hlavných komponentov: kódovacej siete (angl. encoder network), modulu pozornosti (angl.

attention module) a dvoch úplne prepojených vrstiev.

Vstupom DeepGRU sú záznamy gesta reprezentované ako časová sekvencia snímok tohto gesta. Model podporuje variabilnú dĺžku týchto sekvencií. V čase t je vstupom kódovacej siete príznakový vektor $\vec{x}^t \in \mathbb{R}^n$. Dáta celej časovej sekvencie záznamu gesta na vstupe teda popisuje matica $X \in \mathbb{R}^{n \times l}$, kde l je dĺžka tejto časovej sekvencie v počte snímok.

Úlohou kódovacej siete je extrakcia čŕt/príznakov (feature extraction) vstupných dát a teda zníženie ich dimenzionality. Skladá sa z piatich jednotiek GRU, ktorých výstupom je v čase t vektor $\vec{s}^t \in \mathbb{R}^{128}$.

Vstupom modulu pozornosti je vektor $\vec{s}_0$, ktorý je zreťazením výstupných vektorov kódovacej siete v každom čase t počnúc vektorom $\vec{s}^0$ a končiac vektorom $\vec{s}^{l-1}$. Dimenzionalita vektora $\vec{s}_0$ je potom modulom pozornosti ešte ďalej zredukovaná. Jeho výstupom je vektor $\vec{v}_{poz}$, ktorý je zreťazením vektorov $\vec{c} \in \mathbb{R}^{128}$ a $\vec{c}_0$, pričom vektor $\vec{c}$ je definovaný ako:

$$\vec{c} = \frac{exp\left(\left(\vec{s}^{l-1}\right)^\top W^c \vec{s}_0\right)}{\sum_{t=0}^{l-1} exp\left(\left(\vec{s}^{l-1}\right)^\top W^c \vec{s}^t\right)} \vec{s}_0 \tag{2.24}$$

a vektor $\vec{c}_0$ je výstupom GRU jednotky, ktorej vstupom je vektor $\vec{c}$ pričom pre jej výpočet je za prechádzajúci stav považovaný vektor $\vec{s}^{l-1}$.

Posledným komponentom modelu sú dve úplne prepojené vrstvy neurónov s *ReLU* aktivačnými funkciami. Ich vstupom je vektor $\vec{v}_{poz}$, pričom výstupom celého modelu je potom $\vec{y} = softmax(F_2(ReLU(F_1(\vec{v}_{poz}))))$, kde softmax označuje softmax klasifikátor a F_1 a F_2 označuje aplikáciu jednotlivých úplne prepojených vrstiev.

Za účelovú funkciu modelu je zvolená funkcia krížovej entropie (angl. cross-entropy loss) a za učiaci algoritmus je zvolený algoritmus *Adam* [11], ktorý je populárnou modifikáciou algoritmu gradientného zostupu.

Kapitola 3

Použité dáta a implementácia

V tejto kapitole popisujeme tvorbu a štruktúru datasetu gest, s ktorým v našej práci pracujeme, implementáciu experimentov vykonaných v rámci našej práce pre porovnanie klasifikátorov SVM, náhodný les a DeepGRU na tomto datasete a spomíname aj použité metódy predspracovania a ich implementáciu.

3.1 Použité dáta

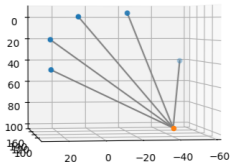
Leap Motion Controller (skr. LMC) [24] je zariadenie, ktoré sa používa na zaznamenanie pohybu rúk. S pomocou infračervených kamier a infračerveného LED svetla sú vytvorené záznamy, z ktorých LMC rozpoznáva pozície kĺbov ruky relatívne k pozícii tohto prístroja a pohyb ruky potom ukladá ako sekvenciu pozícií kĺbov ruky v čase.

S použitím LMC bolo v rámci pilotnej štúdie [23] nahratých niekoľko záznamov šiestich typov s pomocou viacerých osôb. Z týchto šiestich typov záznamov zodpovedá päť z nich gestám pýtať si (ask, obr. 3.1), uchopiť (grasp, obr. 3.2), mávať (move, obr. 3.3), ukázať (point, obr. 3.4) a ok (obr. 3.5) a posledný typ záznamu s názvom nič (nothing, obr. 3.6) zodpovedá náhodnému pohybu resp. žiadnemu z predchádzajúcich gest. Pomocou LMC bolo každé z týchto gest zaznamenané ako sekvencia pozícií prstov a stredu dlane ruky v čase. Tento dataset bol neskôr doplnený ďalšími záznamami týchto istých typov gest, čoho výsledkom je dataset použitý v našej práci.

Jeden záznam gesta je tvorený niekoľkými opakovaniami tohto gesta. Celkový počet záznamov v našom datasete je 192 a z každého typu záznamu obsahuje dataset približne rovnaký počet. Každý typ záznamu/gesta je teda zastúpený približne 32 záznamami. Dĺžky záznamov sú rôzne a to od 60 do 200 snímok pozície ruky a celkovo náš dataset obsahuje odhadom okolo 22 000 snímok pozície ruky.

Keďže každá snímka pozície ruky obsahuje informáciu o pozícii koncov prstov a stredu dlane ruky v priestore $\mathbb{R}^3$, dá sa reprezentovať ako vektor dĺžky 18 (viď. obr. 3.7).

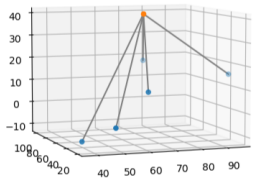
(a)



(b)

Obr. 3.1: Ukážka gesta „pýtať si“(a) a vizualizácia vektora obsahujúceho informáciu o pozícii prstov a dlane ruky prislúchajúceho gestu „pýtať si“(b).

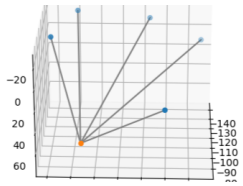
(a)



(b)

Obr. 3.2: Ukážka gesta „uchopiť“(a) a vizualizácia vektora obsahujúceho informáciu o pozícii prstov a dlane ruky prislúchajúceho gestu „uchopiť“(b).

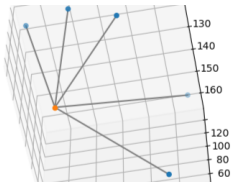
(a)



(b)

Obr. 3.3: Ukážka gesta „mávať“(a) a vizualizácia vektora obsahujúceho informáciu o pozícii prstov a dlane ruky prislúchajúceho gestu „mávať“(b).

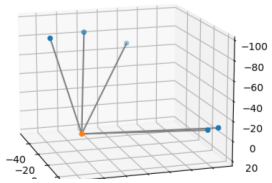
(a)



(b)

Obr. 3.4: Ukážka gesta „ukázať“(a) a vizualizácia vektora obsahujúceho informáciu o pozícii prstov a dlane ruky prislúchajúceho gestu „ukázať“(b).

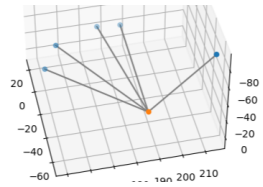
(a)



(b)

Obr. 3.5: Ukážka gesta „ok“(a) a vizualizácia vektora obsahujúceho informáciu o pozícii prstov a dlane ruky prislúchajúceho gestu „ok“(b).

(a)



(b)

Obr. 3.6: Ukážka gesta „nič“(a) a vizualizácia vektora obsahujúceho informáciu o pozícii prstov a dlane ruky prislúchajúceho gestu „nič“(b).

Obr. 3.7: Typy gest z nášho datasetu a ich vizualizácia.

3.2 Predspracovanie a klasifikátory

Na predspracovanie dát využívame analýzu hlavných komponentov (Principal Component Analysis skr. PCA) spolu s rôznymi metódami škálovania dát a tiež vlastnú metódu predspracovania založenú na centrovaní a normalizácii dát, skrátene nazvanú *center-norm*. Pre implementáciu škálovania dát a analýzu hlavných komponentov používame knižnicu scikit-learn [2].

Prvá časť predspracovania center-norm pozostáva z odpočítania súradníc stredu ruky od súradníc prstov, čím sa docieli relativizácia pozícií prstov k pozícii stredu ruky. Samotnú pozíciu stredu ruky ale nechávame nezmenenú, aby mohla aj naďalej niesť informáciu o pozícii ruky ako takej. Na druhú stranu pozície prstov sú teraz zaznamenané voči tejto pozícii relatívne. Druhou časťou tohto predspracovania je tzv. normalizácia, ktorá je založená na vydelení súradníc všetkých prstov vzdialenosťou medzi stredom ruky a prostredníkom s cieľom normalizovať tieto vzdialenosti tak, aby vzdialenosť medzi stredom ruky a prostredníkom bola 1.

Z bežne používaných metód škálovania dát sme vybrali metódu štandardného škálovania a min-max škálovania.

Metóda štandardného škálovania (angl. standard scaling) využíva klasické škálovanie, ktoré vznikne modifikáciou každého príznaku každého príznakového vektora od-

čítaním priemeru daného príznaku a vydelením jeho štandardnou odchýlkou, pričom odchýlka aj priemer každého príznaku sú počítané naprieč všetkými vektormi vstupných dát. Takto sa docieli nulový priemer a jednotková variancia každého príznaku.

Ako častá alternatíva k štandardnému škálovaniu sa využíva metóda min-max škálovania (angl. min-max scaling), ktorá je založená na škálovaní každého príznaku vstupu do určitého rozsahu. V našom prípade bude tento rozsah interval $[0, 1]$.

V našej práci porovnávame klasifikáciu pomocou metódy podporných vektorov (angl. support vector machines alebo SVM), algoritmu náhodný les (angl. random forest) a modelu neurónovej siete DeepGRU [14]. Pre implementáciu klasifikátorov SVM a náhodný les používame knižnicu scikit-learn [2]. Návrh modelu neurónovej siete aj jeho implementácia je prevzatá z práce „DeepGRU: Deep gesture recognition utility“ od Mehrana Maghoumiho a Josepha J LaViolu [14].

3.3 Implementácia experimentov

Cieľom experimentov je výber najoptimálnejšieho z klasifikátorov SVM, náhodný les a DeepGRU pre rozpoznávanie našej sady gest.

Na začiatku z nahratých záznamov gest vyčleníme trénovaciu (a validačnú) a testovaciu množinu. Do testovacej množiny boli vyčlenené záznamy od subjektov *Zu*, *St* a *Pa*. Zvyšné záznamy boli použité na trénovanie a validáciu. Na testovaciu množinu teda pripadá 36 záznamov gest a na trénovaciu a validačnú množinu pripadá zvyšných 156 záznamov. Z toho približne 132 záznamov sa používa na trénovanie pri každej fáze 5-fázovej krížovej validácie, ktorú používame na trénovanie.

Na testovacej množine na záver otestujeme najlepší z klasifikátorov spolu s preň najlepším predspracovaním dát.

3.3.1 Dĺžka sekvencie snímok reprezentujúcej gesto

Pre klasifikátory SVM a náhodný les v experimentoch testujeme najvhodnejšiu dĺžku sekvencie reprezentujúcej jedno gesto. Zvolili sme si dĺžky sekvencií $1, 10, 20$ a 40. Tie sú zvolené takto preto, lebo rovnomerne pokrývajú priestor všetkých možných dĺžok sekvencií, keďže najdlhšia možná dĺžka sekvencie je 60, čo vyplýva z počtu snímok najkratšieho záznamu gesta. Pre každú dĺžku sekvencie je zo záznamov gest vytvorená nová množina príkladov. Pre dĺžku sekvencie n je vytvorená tak, že pre každý záznam gesta je pre každú jeho súvislú podpostupnosť snímok pozícií ruky dĺžky n vytvorený príznakový vektor dĺžky $n \cdot 18$ ako zreťazenie snímok pozícií ruky tejto podpostupnosti (ktoré majú dĺžku 18). K tomuto vektoru je potom priradené patričné označenie (typ gesta). Týmto spôsobom vzniknú všetky nové príklady tejto množiny.

V prípade modelu DeepGRU sme sa rozhodli netestovať najvhodnejšiu dĺžku sekvencie majúcej reprezentovať jedno gesto. Miesto toho postupujeme tak, že sekvenciu zodpovedajúcu jednému gestu vytvárame vždy rozdelením záznamu gesta na pätiny. Z jedného záznamu teda máme päť sekvencií pätinovej dĺžky, ktorých zreťazenie je celý záznam. V rámci jedného záznamu už teda nepracujeme s takým počtom sekvencií, koľko je súvislých podpostupností danej dĺžky, čo znamená, že vzhľadom na počet záznamov pracujeme s lineárnym a nie kvadratickým nárastom počtu spracovávaných dát. Príčinou tejto zmeny prístupu je časová náročnosť trénovania neurónovej siete DeepGRU. Ďalej potom tieto sekvencie nereprezentujeme ako vektor zreťazenia snímok pozície ruky, ale len ako časovú postupnosť snímok pozície ruky (teda vektorov dĺžky 18) kvôli snahe využiť potenciál neurónovej siete DeepGRU, ktorá dokáže vďaka rekurencii zachytávať dlhodobé závislosti v časovej postupnosti dát.

3.3.2 Výber kernelu SVM

Pre SVM na začiatku vykonávame výber kernelu, a to z možností lineárneho, polynomiálneho a gaussovského RBF kernelu. Tento experiment vykonávame na dátach z množiny vytvorenej pre dĺžku sekvencie 20 a pre fixné hyperparametre C a $gamma$ klasifikátora SVM, a to $C = 8$ a $gamma =$ "$scale$" z implementácie v knižnici scikit-learn. Kernely sú porovnávané na dátach predspracovaných všetkými možnými kombináciami z nasledujúcich možností predspracovania:

1. škálovanie: štandardné, min-max
2. center-norm predspracovanie: s použitím/bez
3. PCA: bez / (s použitím s počtom ponechaných komponentov po PCA 42 (zachovávajúcich 95% variancie dát) a 360 (zachovávajúcich 100% variancie dát))

Pre prehľadávanie kernelov a počtu komponentov využívame mriežkové prehľadávanie (angl. gridsearch) a pre trénovanie a validáciu používame metódu 5-fázovej krížovej validácie (angl. 5-fold cross-validation). Pre ich implementáciu používame triedu GridSearchCV z knižnice scikit-learn [2].

3.3.3 Výber optimálnej dĺžky sekvencie a predspracovania pre SVM a náhodný les

Následne postupujeme rovnako pre klasifikátory SVM a náhodný les. Na začiatku si zafixujeme ich hyperparametre. Pre SVM to sú $C = 8$ a $gamma =$ "$scale$", pričom používame najvhodnejší kernel zvolený po predchádzajúcom experimente. Pre náhodný les to sú maximálna hĺbka stromu 50 a počet stromov 100. Zvyšné parametre týchto modelov ponechávame ako východzie pre ich implementáciu v knižnici scikit-learn [2].

Pre daný klasifikátor realizujeme výber optimálnej dĺžky sekvencie majúcej popisovať jedno gesto porovnaním výkonnosti tohto klasifikátora na množinách príkladov vytvorených pre jednotlivé dĺžky sekvencie. Zároveň s tým vyberieme aj najlepšiu kombináciu predspracovania dát pre daný klasifikátor.

Konkrétne pre každú z dĺžok sekvencií vykonáme experiment na dátach z jej prislúchajúcej množiny. Tieto predspracujeme (takmer) každou kombináciou z nasledujúcich možností predspracovania[1]:

1. škálovanie: bez, štandardné, min-max

2. center-norm predspracovanie: s použitím/bez

3. PCA: bez / (s použitím + počet ponechaných komponentov zachovávajúcich 50%, 80%, 90%, 95%, 98%, 99% až 100% variancie dát)

Počty ponechaných komponentov po PCA zachovávajúce dané množstvá variancie dát sa pre každú dĺžku sekvencie môžu odlišovať a sú preto počítané pre každú zvlášť.

Pre najlepšie nájdené kombinácie potom vyhľadáme čo najoptimálnejšie hodnoty hyperparametrov C a *gamma* modelu SVM a maximálnej hĺbky a počtu stromov modelu náhodný les.

Rovnako ako v predchádzajúcom experimente pre prehľadávanie počtu ponechaných komponentov a hodnôt hyperparametrov SVM a náhodného lesa využívame mriežkové prehľadávanie a pre trénovanie a validáciu používame metódu 5-fázovej krížovej validácie. Pre ich implementáciu používame rovnako triedu GridSearchCV z knižnice scikit-learn [2].

3.3.4 Výber optimálneho predspracovania pre DeepGRU

V prípade neurónovej siete DeepGRU postupujeme obdobne. Na začiatku zafixujeme hyperparametre krok učenia (angl. learning rate) a úbytok váh (angl. weight decay) algoritmu *Adam* [11]. Ich hodnoty sú $learning\_rate = 0.001$ a $weight\_decay = 0$ a zvyšné hyperparametre algoritmu *Adam* sú ponechané ako východzie. V práci je použitá implementácia algoritmu *Adam* z knižnice PyTorch [1]. Zafixujeme taktiež počet epoch (teda koľko krát aktualizujeme parametre neurónovej siete použitím celého datasetu) na 15 a veľkosť množiny, na základe ktorej v jednom kroku aktualizujeme parametre (angl. batch size) na 64. Zafixované hodnoty kroku učenia a úbytku váh pre algoritmus *Adam* a tiež počet epoch sa nakoniec po testovaní iných možných hodnôt ukázali ako najoptimálnejšie.

[1] Vynechané sú tie kombinácie, v ktorých nepoužívame žiadne škálovanie a používame PCA, pretože pred použitím PCA je vo väčšine prípadov výhodnejšie ho použiť

Rozdiel je potom v tom, že už nevyberáme optimálnu dĺžku sekvencie. Výber optimálneho predspracovania realizujeme na príslušne označených časových postupnostiach snímok pozície ruky, ktoré sú vstupom neurónovej siete DeepGRU. Tieto predspracujeme (takmer) každou kombináciou z nasledujúcich možností predspracovania[2]:

1. škálovanie: bez, štandardné, min-max
2. center-norm predspracovanie: s použitím/bez
3. PCA: bez / (s použitím + počet ponechaných komponentov zachovávajúcich 50%, 80%, 90%, 95%, 98%, 99% až 100% variancie dát)

V prípade DeepGRU sú vždy používané vektory dĺžky 18 a počty ponechaných komponentov po PCA sú teda postupne 2, 4, 5, 8, 11, 12 a 18.

Trénovanie a validáciu realizujeme v tomto prípade vlastnou implementáciou 5-fázovej krížovej validácie a prehľadávanie počtov ponechaných komponentov po PCA realizujeme spolu s prehľadávaním kombinácií predspracovania použitím vlastnej implementácie mriežkového prehľadávania.

[2]Vynechané sú znova tie kombinácie, v ktorých nepoužívame žiadne škálovanie a používame PCA

Kapitola 4

Výsledky

V tejto kapitole popisujeme výsledky všetkých experimentov, ktoré boli v našej práci vykonané za účelom porovnania klasifikátorov SVM, náhodný les a DeepGRU pre účely rozpoznávania gest z nášho datasetu. Výsledky týchto experimentov vyhodnocujeme v kapitole 5.

4.1 Metóda podporných vektorov

V prípade modelu SVM sa ukazuje, že najbližšie jeho optimálnym hyperparametrom sú pre každú dĺžku sekvencie východzie hyperparametre $C = 8$ a $gamma = "scale"$.

4.1.1 Experiment 0 (výber kernelu)

V tejto časti práce sme sa rozhodli vhodný kernel vyberať z troch možností: lineárneho, polynomického a gaussovského rbf kernelu. Ako vyjadruje tabuľka 4.1, ako najúspešnejší sa v každom prípade ukázal gaussovský rbf kernel.

4.1.2 Experiment 1 (pre dĺžku sekvencie 1)

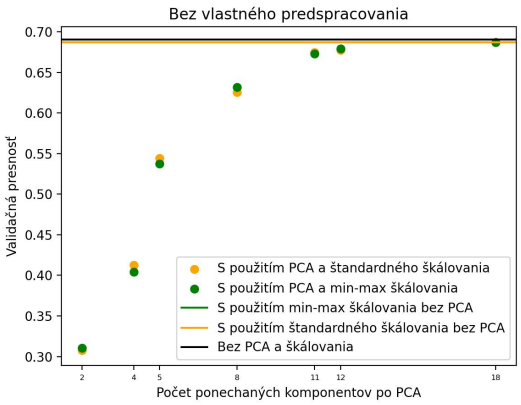
V prípade dĺžky sekvencie 1 máme podľa množstva popisovanej variancie na výber z nasledujúcich počtov ponechaných komponentov po PCA: 2, 4, 5, 8, 11, 12 a 18.

Grafy 4.3 vyjadrujú vývoj dosiahnutej validačnej presnosti pre testované kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA.

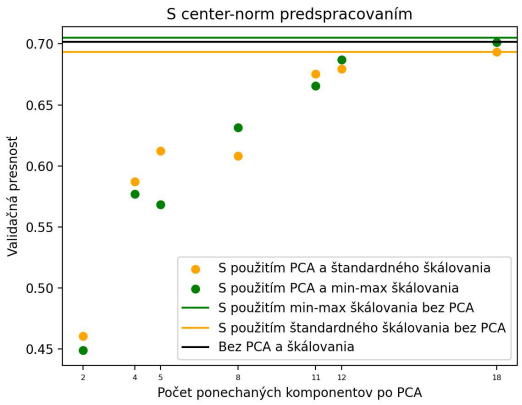
Na základe týchto výsledkov môžeme usudzovať, že najvhodnejšou kombináciou pre túto dĺžku sekvencie je použitie center-norm predspracovania, min-max škálovania a vynechanie PCA. Dosiahnutá validačná presnosť tejto kombinácie a zároveň aj najlepšia dosiahnutá validačná presnosť pre model SVM a dĺžku sekvencie 1 je približne $70,49\%$.

Tabuľka 4.1: Tabuľka vyjadrujúca najlepšiu dosiahnutú validačnú presnosť pre lineárny, polynomický a gaussovský rbf kernel vzhľadom na použitie rôznych kombinácií center-norm predspracovania, škálovania a počtu komponentov ponechaných po použití PCA resp. bez použitia PCA pre dĺžku sekvencie 20. Pritom boli použité východzie hyperparametre SVM $C = 8$ a $gamma =$ "$scale$".

	Bez predspracovania		Center-norm predspracovanie	
	MinMaxScaler	StandardScaler	MinMaxScaler	StandardScaler
	Počet komponentov = 42			
linear	0.717087	0.712366	0.737352	0.73027
poly	0.719512	0.72338	0.745276	0.740422
rbf	**0.78829**	**0.790323**	**0.784227**	**0.780293**
	Počet komponentov = 360			
linear	0.727511	0.729805	0.732103	0.716563
poly	0.725936	0.728625	0.747965	0.749996
rbf	**0.80127**	**0.80468**	**0.789995**	**0.785602**
	Bez PCA			
linear	0.727577	0.729805	0.732037	0.716563
poly	0.772885	0.728625	0.770589	0.749996
rbf	**0.797992**	**0.80468**	**0.795765**	**0.785602**

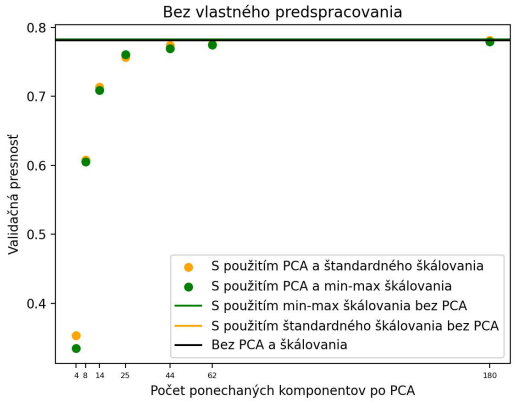


Obr. 4.1: Bez vlastného predspracovania

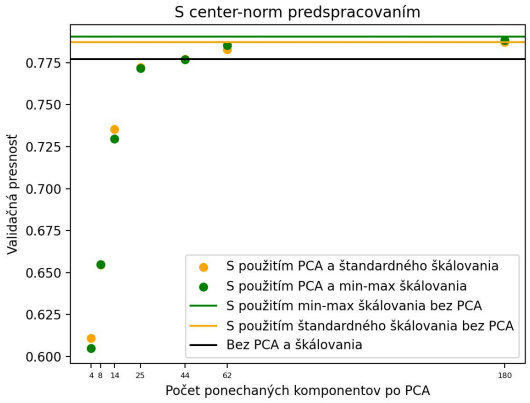


Obr. 4.2: S center-norm predspracovaním

Obr. 4.3: Grafy vyjadrujúce vývoj dosiahnutej validačnej presnosti pre klasifikátor SVM a rôzne kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA pre dĺžku sekvencie 1.



Obr. 4.4: Bez vlastného predspracovania



Obr. 4.5: S center-norm predspracovaním

Obr. 4.6: Grafy vyjadrujúce vývoj dosiahnutej validačnej presnosti pre klasifikátor SVM a rôzne kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA pre dĺžku sekvencie 10.

4.1.3 Experiment 2 (pre dĺžku sekvencie 10)

V prípade dĺžky sekvencie 10 máme podľa množstva popisovanej variancie na výber z nasledujúcich počtov ponechaných komponentov po PCA: 4, 8, 14, 25, 44, 62 a 180.

Grafy 4.6 vyjadrujú vývoj dosiahnutej validačnej presnosti pre testované kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA.

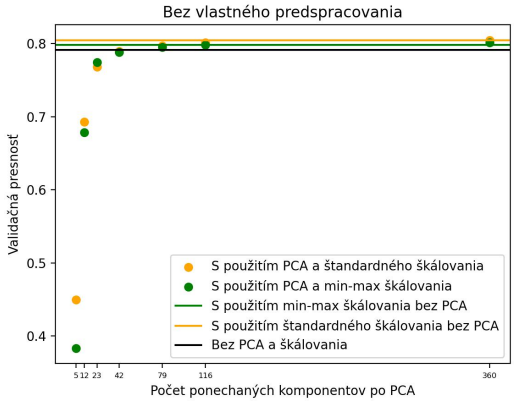
Na základe týchto výsledkov môžeme usudzovať, že najvhodnejšou kombináciou pre túto dĺžku sekvencie je rovnako ako pri dĺžke sekvencie 1 použitie center-norm predspracovania, min-max škálovania a vynechanie PCA. Dosiahnutá validačná presnosť tejto kombinácie a súčasne najlepšia dosiahnutá presnosť pre model SVM a dĺžku sekvencie 10 je ale až približne $79,045\%$.

4.1.4 Experiment 3 (pre dĺžku sekvencie 20)

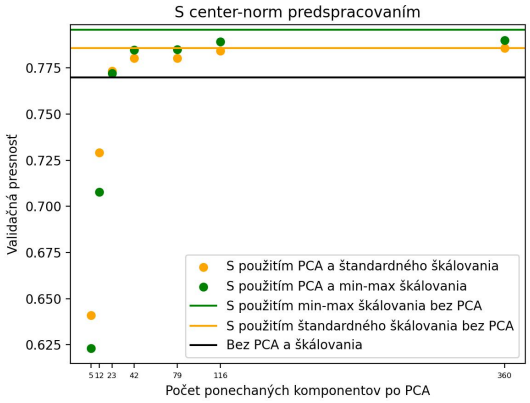
V prípade dĺžky sekvencie 20 máme podľa množstva popisovanej variancie na výber z nasledujúcich počtov ponechaných komponentov po PCA: 5, 12, 23, 42, 79, 116 a 360.

Grafy 4.9 vyjadrujú vývoj dosiahnutej validačnej presnosti pre testované kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA.

Na základe týchto výsledkov môžeme usudzovať, že najvhodnejšou kombináciou pre túto dĺžku sekvencie je na rozdiel od predchádzajúcich použitie štandardného škálovania bez center-norm predspracovania. Na zvolení alebo nezvolení použitia PCA v prípade použitia všetkých komponentov po PCA v tomto prípade nezáleží, keďže obe



Obr. 4.7: Bez vlastného predspracovania



Obr. 4.8: S center-norm predspracovaním

Obr. 4.9: Grafy vyjadrujúce vývoj dosiahnutej validačnej presnosti pre klasifikátor SVM a rôzne kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA pre dĺžku sekvencie 20.

tieto možnosti majú v tomto prípade rovnakú a zároveň najlepšiu presnosť $80,468\%$.

4.1.5 Experiment 4 (pre dĺžku sekvencie 40)

V prípade dĺžky sekvencie 40 máme podľa množstva popisovanej variancie na výber z nasledujúcich počtov ponechaných komponentov po PCA: 6, 21, 39, 75, 150, 223 a 720.

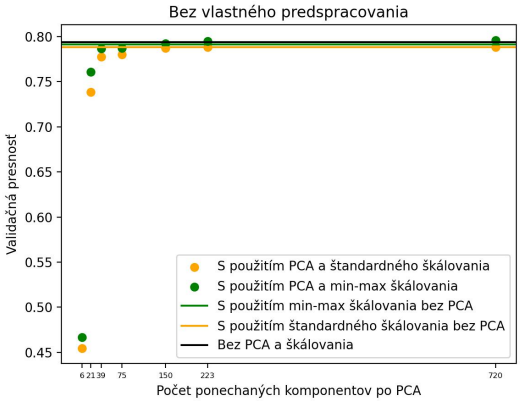
Grafy 4.12 vyjadrujú vývoj dosiahnutej validačnej presnosti pre testované kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA.

Na základe týchto výsledkov môžeme usudzovať, že najvhodnejšou kombináciou pre túto dĺžku sekvencie je kombinácia s použitím min-max škálovania bez použitia center-norm predspracovania a s použitím PCA a následným ponechaním všetkých komponentov. Dosiahnutá validačná presnosť tejto kombinácie a súčasne najlepšia dosiahnutá presnosť pre model SVM a dĺžku sekvencie 40 je približne $79,608\%$, čo je horšie ako pre dĺžku sekvencie 20.

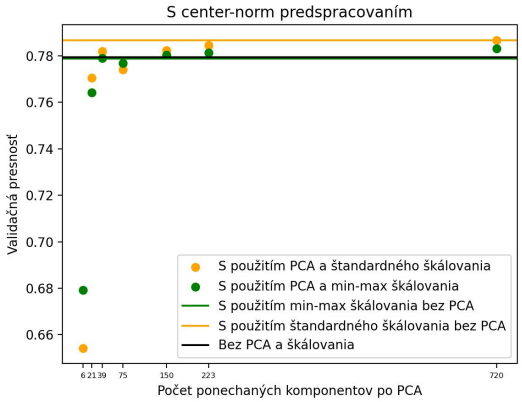
4.2 Náhodný les

4.2.1 Experiment 1 (pre dĺžku sekvencie 1)

V prípade dĺžky sekvencie 1 máme podľa množstva popisovanej variancie na výber z rovnakých počtov ponechaných komponentov po PCA ako v analogickom prípade pri testovaní SVM.



Obr. 4.10: Bez vlastného predspracovania



Obr. 4.11: S center-norm predspracovaním

Obr. 4.12: Grafy vyjadrujúce vývoj dosiahnutej validačnej presnosti pre klasifikátor SVM a rôzne kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA pre dĺžku sekvencie 40.

Grafy 4.15 vyjadrujú vývoj dosiahnutej validačnej presnosti pre testované kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA.

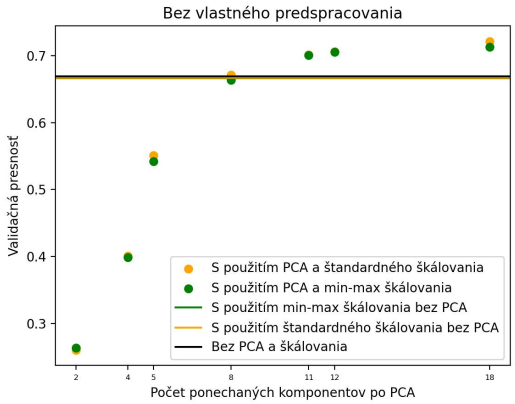
Na základe týchto výsledkov môžeme usudzovať, že najvhodnejšou kombináciou pre túto dĺžku sekvencie je kombinácia s použitím štandardného škálovania, bez použitia center-norm predspracovania, s použitím PCA a následným ponechaním všetkých komponentov. Dosiahnutá validačná presnosť tejto kombinácie je približne $72,13\%$, čo je viac ako v analogickom prípade pre SVM. Najlepšia dosiahnutá presnosť pre model náhodný les a dĺžku sekvencie 1 je však po optimalizácii hyperparametrov až $72,44\%$, a to pre počet stromov (n_estimators) 300 a maximálnu hĺbku jedného stromu (max_depth) 100.

4.2.2 Experiment 2 (pre dĺžku sekvencie 10)

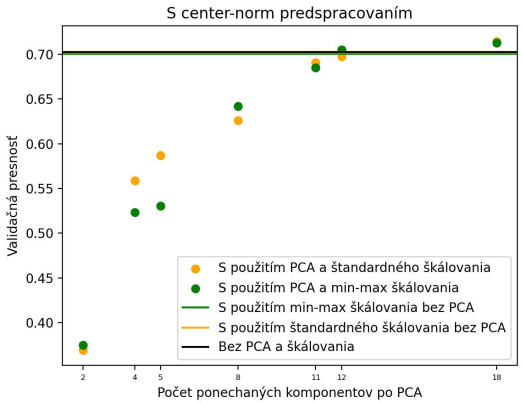
V prípade dĺžky sekvencie 10 máme podľa množstva popisovanej variancie na výber z rovnakých počtov ponechaných komponentov po PCA ako v analogickom prípade pri testovaní SVM.

Grafy 4.18 vyjadrujú vývoj dosiahnutej validačnej presnosti pre testované kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA.

Na základe týchto výsledkov môžeme usudzovať, že najvhodnejšou kombináciou pre túto dĺžku sekvencie je takmer rovnaká kombinácia ako v prípade dĺžky sekvencie 1 s tým rozdielom, že v tomto prípade je o niečo málo výhodnejšie použitie min-max škálovania. Dosiahnutá validačná presnosť tejto kombinácie je približne $81,15\%$, čo je

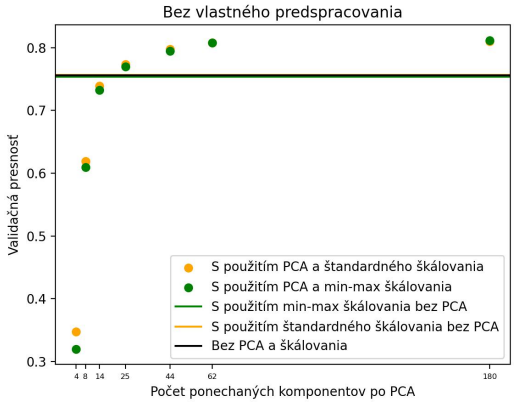


Obr. 4.13: Bez vlastného predspracovania

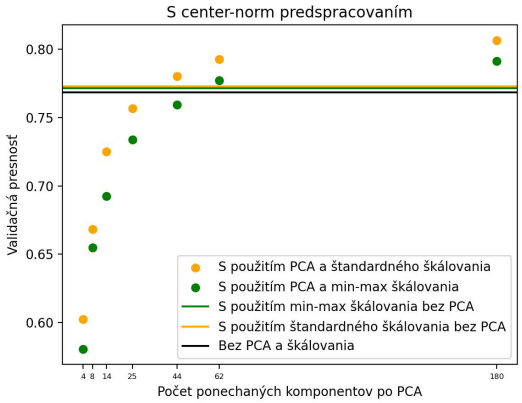


Obr. 4.14: S center-norm predspracovaním

Obr. 4.15: Grafy vyjadrujúce vývoj dosiahnutej validačnej presnosti pre klasifikátor náhodný les a rôzne kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA pre dĺžku sekvencie 1.

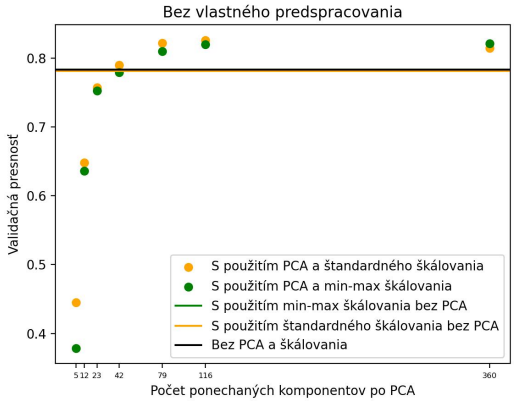


Obr. 4.16: Bez vlastného predspracovania

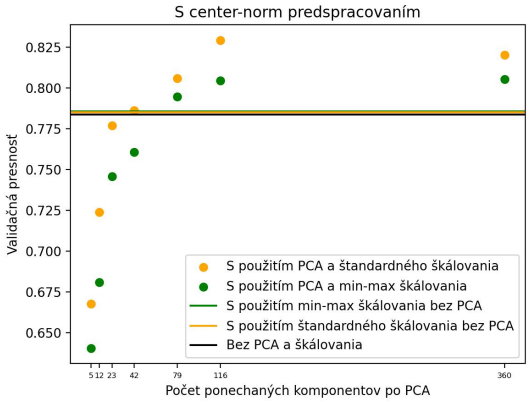


Obr. 4.17: S center-norm predspracovaním

Obr. 4.18: Grafy vyjadrujúce vývoj dosiahnutej validačnej presnosti pre klasifikátor náhodný les a rôzne kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA pre dĺžku sekvencie 10.



Obr. 4.19: Bez vlastného predspracovania



Obr. 4.20: S center-norm predspracovaním

Obr. 4.21: Grafy vyjadrujúce vývoj dosiahnutej validačnej presnosti pre klasifikátor náhodný les a rôzne kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA pre dĺžku sekvencie 20.

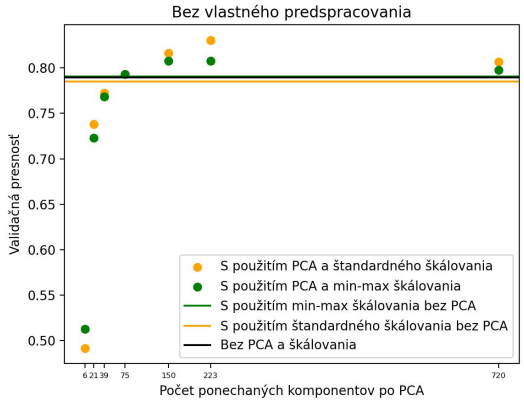
viac ako pre dĺžku sekvencie 1 a taktiež viac ako v analogickom prípade pre SVM. Najlepšia dosiahnutá presnosť pre model náhodný les a dĺžku sekvencie 10 je však po optimalizácii hyperparametrov až $82,55\%$, a to pre počet stromov (n_estimators) 500 a maximálnu hĺbku jedného stromu (max_depth) 100.

4.2.3 Experiment 3 (pre dĺžku sekvencie 20)

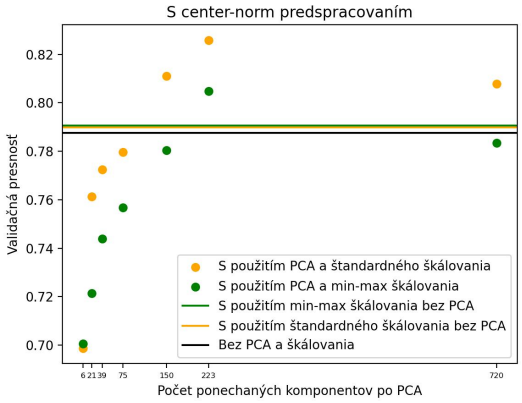
V prípade dĺžky sekvencie 20 máme podľa množstva popisovanej variancie na výber z rovnakých počtov ponechaných komponentov po PCA ako v analogickom prípade pri testovaní SVM.

Grafy 4.21 vyjadrujú vývoj dosiahnutej validačnej presnosti pre testované kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA.

Na základe týchto výsledkov môžeme usudzovať, že najvhodnejšou kombináciou pre túto dĺžku sekvencie je kombinácia s použitím center-norm predspracovania, štandardného škálovania a PCA s ponechaním 116 komponentov, čo prislúcha ponechaniu komponentov popisujúcich 99% variancie dát. Dosiahnutá validačná presnosť tejto kombinácie je približne $82,92\%$, čo je viac ako pre dĺžku sekvencie 10 a taktiež viac ako v analogickom prípade pre SVM. Najlepšia dosiahnutá presnosť pre model náhodný les a dĺžku sekvencie 20 je však po optimalizácii hyperparametrov až $84,15\%$, a to pre počet stromov (n_estimators) 500 a maximálnu hĺbku jedného stromu (max_depth) 300.



Obr. 4.22: Bez vlastného predspracovania



Obr. 4.23: S center-norm predspracovaním

Obr. 4.24: Grafy vyjadrujúce vývoj dosiahnutej validačnej presnosti pre klasifikátor náhodný les a rôzne kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA pre dĺžku sekvencie 40.

4.2.4 Experiment 4 (pre dĺžku sekvencie 40)

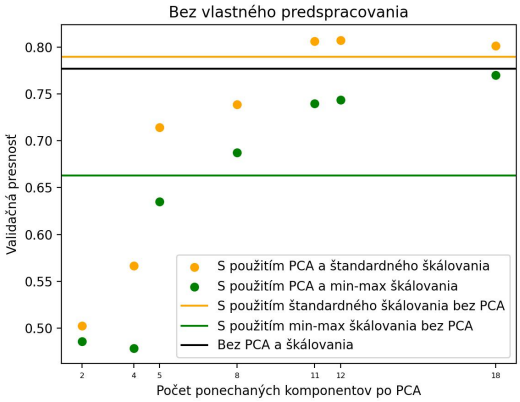
V prípade dĺžky sekvencie 40 máme podľa množstva popisovanej variancie na výber z rovnakých počtov ponechaných komponentov po PCA ako v analogickom prípade pri testovaní SVM.

Grafy 4.24 vyjadrujú vývoj dosiahnutej validačnej presnosti pre testované kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA.

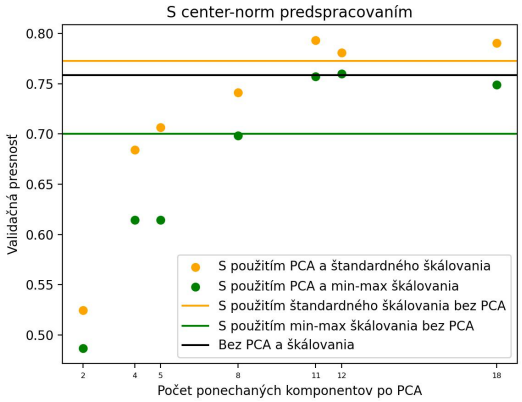
Na základe týchto výsledkov môžeme usudzovať, že najvhodnejšou kombináciou pre túto dĺžku sekvencie je kombinácia s použitím štandardného škálovania bez použitia center-norm predspracovania a s použitím PCA s následným ponechaním 223 komponentov, čo rovnako zodpovedá ponechaniu komponentov popisujúcich 99% variancie dát. Dosiahnutá validačná presnosť tejto kombinácie je približne $83,06\%$, čo je viac ako v analogickom prípade pre SVM, ale menej ako po optimalizácii hyperparametrov v prípade dĺžky sekvencie 20. Ani po optimalizácii hyperparametrov nie je v prípade dĺžky sekvencie 40 výsledná validačná presnosť väčšia, ako v prípade dĺžky sekvencie 20, a síce $83,93\%$, a to pre počet stromov (n_estimators) 500 a maximálnu hĺbku jedného stromu (max_depth) 50.

4.3 DeepGRU

Keďže sú v tomto prípade používané vektory dĺžky 18, na výber máme z nasledujúcich počtov ponechaných komponentov po PCA: 2, 4, 5, 8, 11, 12 a 18.



Obr. 4.25: Bez vlastného predspracovania



Obr. 4.26: S center-norm predspracovaním

Obr. 4.27: Grafy vyjadrujúce vývoj dosiahnutej validačnej presnosti pre model neurónovej siete DeepGRU a rôzne kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA

Grafy 4.27 vyjadrujú vývoj dosiahnutej validačnej presnosti pre testované kombinácie škálovania, center-norm predspracovania a počtu ponechaných komponentov po PCA a rovnako bez použitia PCA.

Na základe týchto výsledkov môžeme usudzovať, že najvhodnejšia kombinácia predspracovania vstupných dát pre neurónovú sieť DeepGRU neobsahuje center-norm predspracovanie, ale používa štandardné škálovanie dát a PCA s ponechaním 12 komponentov, ktoré popisujú 99% variancie v dátach. Dosiahnutá validačná presnosť tejto kombinácie je približne $80,75\%$.

Kapitola 5

Diskusia

V tejto kapitole sumarizujeme dosiahnuté výsledky a porovnávame výkonnosť modelov SVM, náhodný les a neurónovej siete DeepGRU na našom datasete gest.

Po optimalizácii dĺžky sekvencie popisujúcej jedno gesto, predspracovania dát a hyperparametrov klasifikačných modelov sa model náhodný les ukázal ako najoptimálnejší klasifikačný model pre riešenie problému rozpoznávania gest na našom datasete. Na grafe 5.1 možno vidieť, že algoritmus náhodný les je pre každú dĺžku sekvencie optimálnejší než SVM a pre dĺžky sekvencií 10, 20 a 40 je dokonca optimálnejší než model neurónovej siete DeepGRU, čo je prinajmenšom prekvapivý výsledok. Na druhú stranu algoritmus SVM sa ukazuje byť menej výkonný ako algoritmus náhodný les aj neurónová sieť DeepGRU.

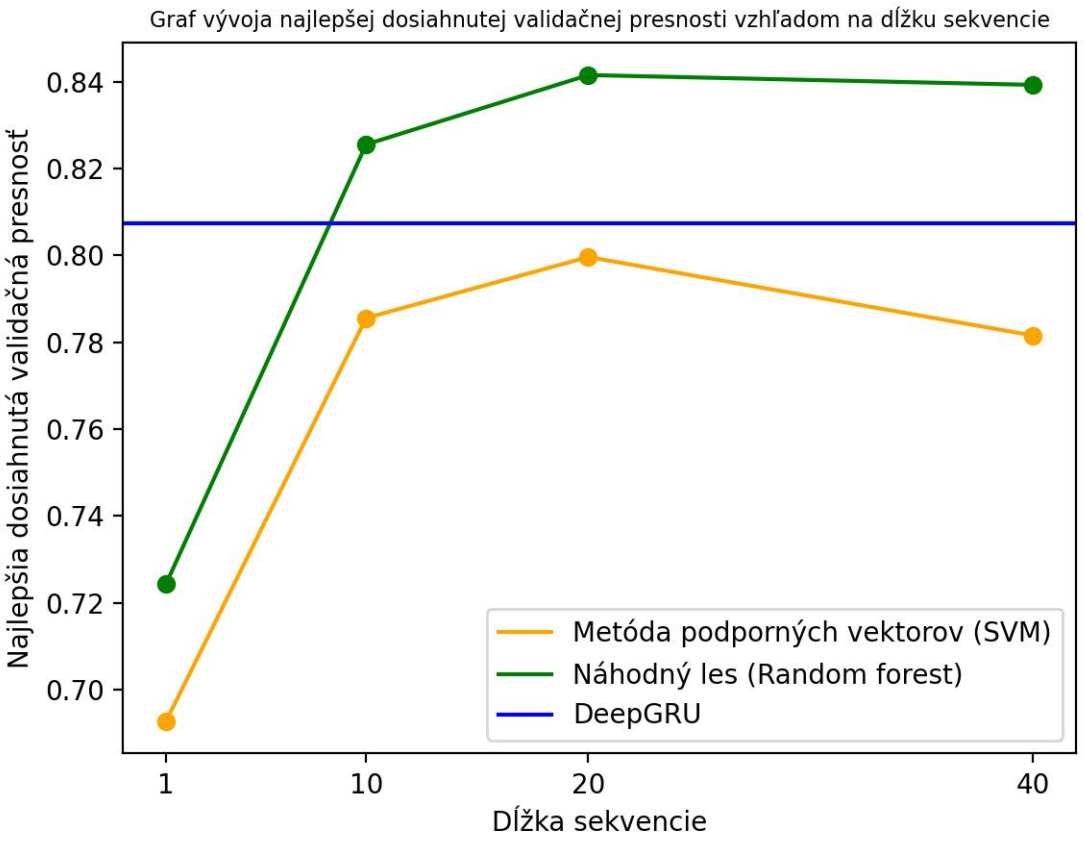
Najlepší výsledok dosahuje algoritmus náhodný les pre dĺžku sekvencie predstavujúcej jedno gest zvolenú na 20 snímok a jej celkovú optimálnosť podporuje aj rovnako najlepší výsledok modelu SVM práve na sekvencii dĺžky 20 (viď. graf 5.1).

Očividne najvhodnejším kernelom pre použitie modelu SVM na našom datasete je gaussovský RBF kernel dosahujúci o $2 - 8\%$ lepšiu presnosť ako ostatné kernely.

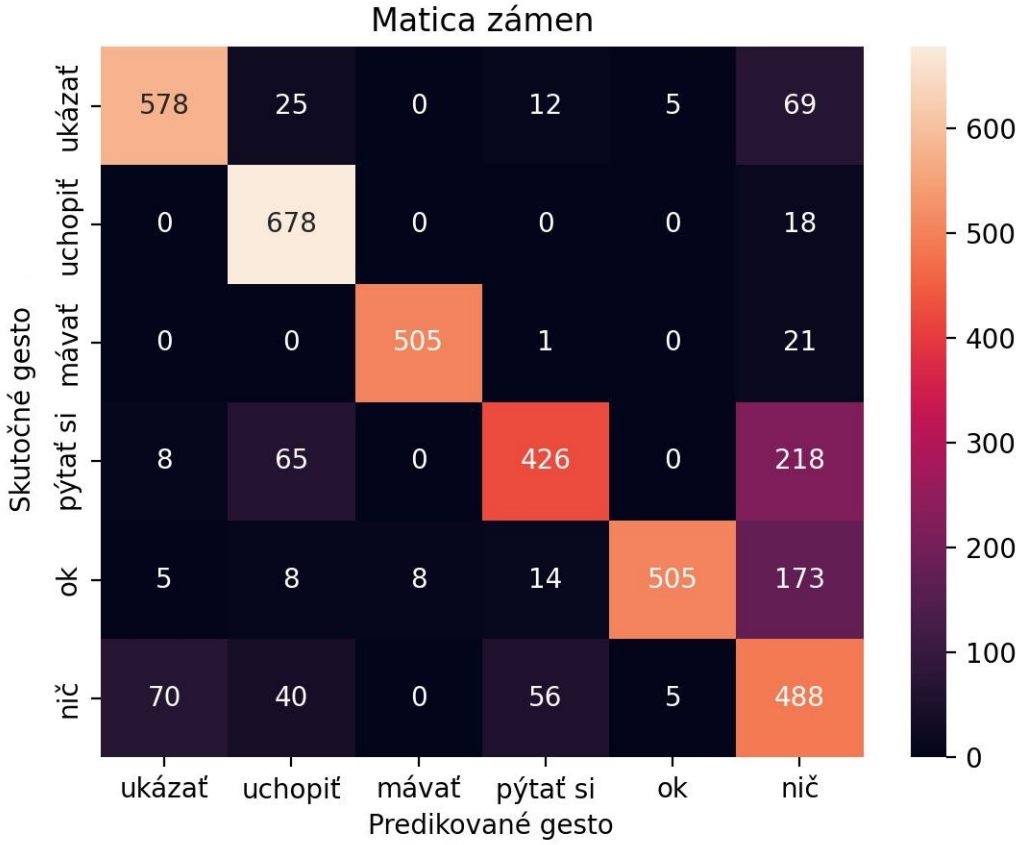
Nami navrhnuté predspracovanie center-norm preukazuje v niektorých prípadoch zlepšenie a v niektorých prípadoch zhoršenie presnosti klasifikácie. Obe tieto zmeny sú však len nepatrné.

Za najoptimálnejšiu kombináciu môžeme určiť výber dĺžky sekvencie predstavujúcej jedno gesto ako 20, použitie nami navrhnutého center-norm predspracovania, štandardného škálovania a PCA s ponechaním 116 komponentov, čo zodpovedá ponechaniu komponentov zachovávajúcich 99% variancie v dátach, pričom použijeme klasifikátor náhodný les s počtom stromov 500 a maximálnou hĺbkou jedného stromu 300. Táto kombinácia má validačnú presnosť $84,15\%$ a výslednú testovaciu presnosť $79,18\%$.

Tento výsledok, približne 80% je v súlade s očakávaniami. Z matice zámen (5.2) po záverečnom teste možno totižto vidieť, že mnoho prípadov zámeny nastáva v prípade zlého vyhodnotenia gest ako žiadne gesto (označené ako „nič“). Kvôli nejasnej



Obr. 5.1: Graf popisujúci vývoj najlepšej dosiahnutej validačnej presnosti pre každú testovanú dĺžku sekvencie pre klasifikátory SVM a náhodný les a najlepšej dosiahnutej validačnej presnosti pre model neurónovej siete DeepGRU



Obr. 5.2: Matica zámen popisujúca počet vzájomných zámen jednotlivých gest pri testovaní najlepšej konfigurácie s klasifikátorom náhodný les. Posledný stĺpec popisuje počet vyhodnotení jednotlivých gest ako záznam typu „nič“.

charakteristike záznamu typu „nič“ nie je prekvapením, že náš model pri trénovaní aj následnom testovaní dokáže pomýliť. Keby sme teda nebrali do úvahy záznam typu „nič“, potom by výsledná testovacia presnosť modelu náhodný les dosť pravdepodobne presahovala 80%.

5.1 Miera možného skreslenia výsledkov

Model DeepGRU nebol trénovaný na úplne rovnako predspracovaných dátach ako modely SVM a náhodný les, kvôli čomu mal voči nim istú nevýhodu. Tieto klasifikačné modely boli trénované na kvadraticky väčšom množstve dát spracovaných do štyroch rôznych dĺžok sekvencií pokrývajúcich všetky podpostupnosti záznamu gesta. Zároveň bola dĺžka vstupu jedného trénovaného klasifikátora SVM/náhodný les zakaždým rovnakej dĺžky, pričom v prípade DeepGRU sa tieto dĺžky líšili. Napriek tomu veríme, že naše porovnanie klasifikátorov nie je príliš skreslené.

Aj keď boli SVM a náhodný les trénované na kvadraticky väčšom množstve dát,

je dosť pravdepodobné, že to nie je významným činiteľom v ich konečnej presnosti, keďže množstvo trénovacích dát je už tak dosť veľké (pri každej fáze krížovej validácie približne 132 záznamov). Navyše boli pre trénovanie DeepGRU využité všetky snímky datasetu použité aj pri SVM/náhodnom lese, pričom v prípade SVM/náhodného lesa vznikli nové príklady oproti prípadu DeepGRU len posunom/zmenou dĺžky o niekoľko snímok v rámci záznamu gesta. Vďaka rekurentnosti DeepGRU by ale adaptácia na tento posun/zmenu dĺžky, zaistená pri SVM/náhodnom lese novými príkladmi, nemusela byť problémom. Sieť DeepGRU by teda mala byť do istej miery odolná aj voči variabilite dĺžky časovej sekvencie na vstupe.

Z pozorovaní tiež vyplýva, že modely SVM a náhodný les preukazujú značné zvýšenie presnosti už pri dĺžke sekvencie 10 (pri náhodnom lese už vtedy väčšiu presnosť ako DeepGRU). Pritom keďže minimálna dĺžka jedného záznamu je 60 snímok, dĺžka jednej sekvencie použitej pri DeepGRU nikdy neklesla pod 10 snímok.

Na základe týchto pozorovaní si dovoľujeme tvrdiť, že miera skreslenia našich výsledkov je minimálna.

Záver

Cieľom tejto bakalárskej práce bolo porovnanie klasifikátorov SVM, náhodný les a DeepGRU pre použitie na rozpoznávanie danej sady gest z nášho datasetu s motiváciou ďalšieho použitia pre komunikáciu s humanoidným robotom NICO.

Toto sme v našej práci úspešne splnili. Dosiahli sme to vďaka úspešnému návrhu, implementácii a vykonaniu experimentov. Tie zahŕňali najmä výber správnej dĺžky sekvencie snímok ruky reprezentujúcej jedno gesto, výber optimálneho škálovania dát a predspracovania pomocou PCA alebo center-norm predspracovania navrhnutého v rámci tejto práce. Okrem toho zahŕňali aj výber najvhodnejšieho kernelu pre SVM a iných hyperparametrov.

Zistili sme, že algoritmus náhodný les je najvhodnejší pre rozpoznávanie sady gest z nášho datasetu. Všetky tri metódy sa však v presnosti klasifikácie odlišovali len minimálne. Každopádne sa použitie modelov SVM a náhodný les v našom prípade ukázalo ako ekvivalentné použitiu aktuálneho modelu neurónovej siete DeepGRU navrhnutého pre rozpoznávanie gest.

Medzi ďalšie získané výsledky patrí napríklad zistenie, že najvhodnejší kernel pre použitie SVM na našom datasete je gaussovský RBF kernel a najoptimálnejšia dĺžka sekvencie snímok pozície ruky zodpovedajúca jednému gestu je 20.

Táto práca zároveň predstavuje významný posun oproti spomínanej pilotnej štúdii [23] pre modul humanoidného robota majúci rozpoznávať ľudskú činnosť, keďže najlepšia testovacia presnosť dosiahnutá v rámci tejto práce bola 80%, čo je nárast o 10% oproti najlepšiemu výsledku približne 70% dosiahnutom v rámci pilotnej štúdie. Tento výsledok bol dosiahnutý na pôvodnom datasete z pilotnej štúdie rozšírenom o niekoľko ďalších záznamov gest.

Na túto pracú možno nadviazať napríklad zväčšením rozpoznávaného spektra gest alebo návrhom vlastnej architektúry neurónovej siete pre rozpoznávanie nášho/rozšíreného datasetu gest. Nakoniec je samozrejme očakávaným rozšírením najmä aplikovanie natrénovaných klasifikátorov pre klasifikáciu v reálnom čase a jej využitie pre komunikáciu s humanoidným robotom NICO.

Literatúra

[1] PyTorch: An imperative style, high-performance deep learning library. `https://pytorch.org/`. Accessed on: 20.5.2023.

[2] scikit-learn: Machine learning in Python. `https://scikit-learn.org/`. Accessed on 21.2.2023.

[3] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[4] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

[5] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[6] Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1998.

[9] IEEE. *NICO – Neuro-Inspired COmpanion: A Developmental Humanoid Robot Platform for Multimodal Interaction*, Lisbon, Portugal, 2017.

[10] Herbert Jaeger. Echo state network. *scholarpedia*, 2(9):2330, 2007.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Matej Králik. Multi-sensor accelerometer-based gesture recognition. Master's thesis, Comenius University in Bratislava, 2020.

[13] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[14] Mehran Maghoumi and Joseph J LaViola. Deepgru: Deep gesture recognition utility. In *Advances in Visual Computing: 14th International Symposium on Visual Computing, ISVC 2019, Lake Tahoe, NV, USA, October 7–9, 2019, Proceedings, Part I 14*, pages 16–31. Springer, 2019.

[15] Andrew Ng. cs229 - machine learning. `https://see.stanford.edu/course/cs229`. Accesed on: 19.05.2023.

[16] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. `http://neuralnetworksanddeeplearning.com/`.

[17] Arti Patle and Deepak Singh Chouhan. Svm kernel functions for classification. In *2013 International Conference on Advances in Technology and Engineering (ICATE)*, pages 1–9. IEEE, 2013.

[18] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.

[19] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[20] Lior Rokach and Oded Maimon. Decision trees. *Data mining and knowledge discovery handbook*, pages 165–192, 2005.

[21] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[22] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.

[23] Clara Swaboda. Gesture recognition using echo state networks. Mobility Project Report, Comenius University in Bratislava, 2022.

[24] Ultraleap. World-leading hand tracking: Small. fast. accurate. https://www.ultraleap.com/tracking/. Accessed on: 11.03.2023.

Príloha A: obsah elektronickej prílohy

V elektronickej prílohe priloženej k práci sa nachádza zdrojový kód programu napísaného za účelom vykonania experimentov a súbory s výsledkami týchto experimentov. Taktiež sa v nej nachádzajú dáta (nahraté záznamy gest), na ktorých boli výsledky získané. Tie sa nachádzajú aj na stránke: `http://cogsci.dai.fmph.uniba.sk/~kocur/gestures/`

Zdrojový kód je rozdelený na dve časti.

Prvá časť zodpovedá experimentom na klasifikátoroch SVM a náhodný les a okrem elektronickej prílohy je zverejnená aj na stránke `https://github.com/matomarss/gesture_recognition_thesis_SVM_RF.git`.

Druhá časť zodpovedá experimentom na neurónovej sieti DeepGRU a okrem elektronickej prílohy je zverejnená aj na stránke `https://github.com/matomarss/gesture_recognition_thesis_deepGRU.git`.