Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Biometrická autentizace

Diplomová práce

Autor: Bc. Jan Kozlovský

Studijní obor: AI2 – P

Vedoucí práce: Ing. Karel Petránek

Hradec Králové Duben 2016

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Pardubicích dne: 25.4.2016 Jan Kozlovský

Poděkování

Tímto bych chtěl poděkovat mému vedoucímu práce za odborné vedení při implementaci metod a tvorbě samotné práce.

Anotace

Díky vyššímu počítačovému výkonu je nyní možno pro počítačovou bezpečnost využívat technologie strojového učení. Tato diplomová práce představuje tři algoritmy strojového učení, které autentizují uživatele na základě jejich biometrických znaků: neuronové sítě, algoritmus k-nejbližších sousedů a Bayesovský klasifikátor. Biometrickým znakem uživatele, který je zkoumán v této práci, je rytmus psaní hesla, tzv. dynamika psaní. Práce se zabývá vhodným návrhem reprezentace vstupních a výstupních dat, prozkoumáním uvedených metod strojového učení a jejich implementací pro biometrickou autentizaci uživatele s využitím dynamiky psaní. V závěru práce jsou použité metody srovnány z pohledu složitosti implementace, rychlosti učení se, přesnosti predikce, bezpečnosti a možností reálného nasazení.

Annotation

Title: Biometric Authentication

Higher computer performance brings a possibility to use machine learning technology for computer security. This diploma thesis introduces three machine learning algorithms which authenticate users based on their biometric identifiers. The following machine learning algorithms are demonstrated: neural network, k-nearest neighbour algorithm and Bayessian classifier. Keystroke dynamics (the typing rhythm of a written password) is used as the user's biometric identifier. This diploma thesis describes the representation of input and output data, explores the above mentioned machine learning methods and their suitability for biometric authentication using keystroke dynamics. All methods are compared with respect to the complexity of an implementation, the ability to learn, prediction accuracy, safety and the suitability for a real-world application.

Obsah

Úvod

Snadný přístup k informacím skrze moderní počítačové technologie a rozsáhlou síť internetu se v dnešní době stal téměř samozřejmým požadavkem. Neustále se vylepšující počítačové technologie nám usnadňují každodenní život, zatímco se na této technologii stáváme čím dál tím více závislí. Technologický pokrok nastal zejména v oblasti výkonu, spolehlivosti a dostupnosti a významně snížil operační náklady díky své vyšší efektivitě. Využívání počítačových technologií tak neustále vzrůstá. Nicméně tento vzrůstající celosvětový zájem o přístup k informacím má také své hrozby, na které je třeba myslet. Konkrétně počítačová bezpečnost a ochrana dat.

S vylepšujícími se dostupnými počítačovými technologiemi dochází k pokroku nejen v oblasti dostupnosti dat, ale také v oblasti počítačové bezpečnosti. Díky vyššímu počítačovému výkonu je nyní možno řešit počítačovou bezpečnost na takové úrovni, na které to dříve nebylo možné. Je nyní možno využívat technologie, jako je např. strojové učení. Strojové učení spadá do oblasti umělé inteligence. Ke své činnosti potřebuje výkon, který dříve nebyl dostupný, a může pomoci proti falešným neautorizovaným přístupům a krádeži dat. Tato diplomová práce představuje několik algoritmů strojového učení, které autentizují uživatele na základě jejich biometrických znaků.

Biometrie, tedy fyziologické znaky a behaviorální charakteristiky, které dělají člověka unikátním, je nejvhodnějším způsobem verifikace člověka, neboť oproti klíčům nebo heslům, biometrie nemůže být ztracena nebo ukradena a nabízí téměř neoklamatelný způsob určení identity. Fyziologické biometrické znaky, např. otisky prstů, jsou skvělým kandidátem k autentizaci, neboť jsou unikátní skrze rozsáhlý výběr populace.

Biometrickým znakem, který je zkoumán a popisován v této práci, je rytmus psaní hesla (tzv. dynamika psaní). Dynamika psaní analyzuje způsob, jakým uživatel píše na klávesnici, a snaží se jej na základě stylu psaní identifikovat. Použití právě dynamiky psaní, tj. rytmu psaní hesla pro biometrickou autentizaci, je přirozenou volbou v počítačových technologiích. Vychází z pozorování, že podobné neuro-fyziologické faktory, které tvoří ručně psaný podpis unikátním, jsou pozorovány také v rytmu psaní na klávesnici. Latence mezi stisky kláves, doba stisku kláves, poloha prstů a tlak aplikovaný na klávesu mohou být použity pro zkonstruování unikátního podpisu (profilu) daného uživatele. Pro předem známý pravidelně psaný řetězec (kterým je např. heslo) je pak takto vytvořený podpis při každém pokusu jednoho člověka shodný [1].

Cílem práce je prozkoumat a popsat existující metody strojového učení a implementovat je pro biometrickou autentizaci uživatele s využitím dynamiky psaní. Existují práce, které popisují a případně srovnávají statistické metody a metody rozpoznávání vzorů [1]. Jedná se však stále o úzké portfolio metod umělé inteligence. Tato práce rozšiřuje použitou škálu metod o využití neuronových sítí. Soustředí se podrobně na každou z metod a důkladně je porovnává.

Práce v první kapitole nejprve teoreticky popisuje použité metody pro biometrickou autentizaci. Za účelem této práce byly zvoleny tři nejčastější metody strojového učení –

algoritmus k-nejbližších sousedů, tedy metoda patřící do oblasti rozpoznávání vzorů, Bayesovský klasifikátor, tedy statistická metoda a neuronové sítě.

V kapitole druhé je popsáno samotné využití vybraných metod. To sestává ze sběru dat mezi třemi reálnými uživateli, návrh a implementace samotných metod, jejich učení pomocí tréninkových dat a testování reálnými uživateli jak těmi, pro které probíhalo učení, tak uživateli neznámými.

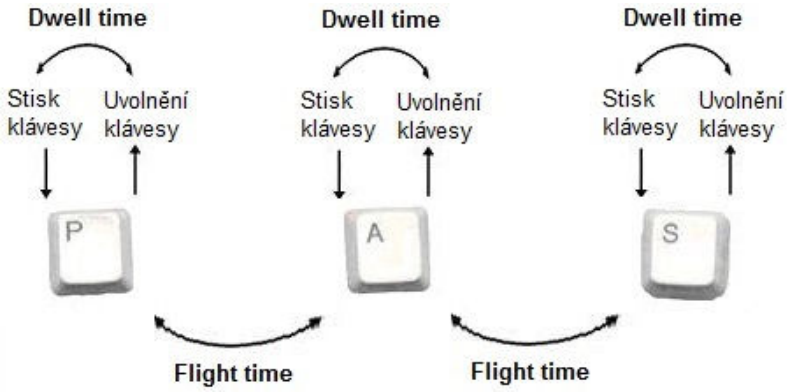
Kapitola třetí shrnuje dosažené výsledky a v kapitole čtvrté jsou použité metody porovnány a vyhodnoceny. Zmíněny jsou výhody a nevýhody použitých algoritmů a dosažené úspěšnosti. Závěr práce a doporučení k dalšímu výzkumu se nachází v kapitole páté.

1 Teorie dynamiky psaní pro biometrickou autentizaci

Dynamika psaní (keystroke dynamics) popisuje automatickou metodu pro identifikaci nebo potvrzování totožnosti uživatele na základě povahy a rytmu psaní na klávesnici. Dynamika psaní je biometrie chování, tzn. biometrický faktor je zde „něco, co děláme." Rytmus psaní na klávesnici je dobrým znakem identity a na rozdíl od jiných metod biometrické autentizace je také levný k implementaci - pro jeho použití postačuje pouze klávesnice [1].

Biometrickou předlohou pro identifikaci uživatele je napsaný řetězec, rytmus a rychlost jak byl napsán. Samotné měření používané v dynamice psaní sleduje dvě veličiny:

- Dwell time – časový úsek, po který je klávesa stisknuta,
- Flight time – časový úsek mezi uvolněním klávesy jedné a stiskem klávesy druhé (viz obrázek 1).



Obrázek 1 - Znázornění extrakce znaků identity z napsaného hesla

Během psaní textového řetězce je čas, který uživatel potřebuje k nalezení správné klávesy (flight time) a čas, po který danou klávesu drží (dwell time), specifický pro každého uživatele. Tyto časové intervaly posbírané během psaní hesla sestávajícího z více písmen jsou vhodným vzorkem k analýze uživatele a mohou být použity k biometrické autentizaci. Tato práce se zabývá možnostmi použití metod strojového učení (machine learning) pro autentizaci uživatele na základě těchto časových intervalů.

1.1 Strojové učení

Strojové učení je podoblastí umělé inteligence a v tomto kontextu znamená „obor studia, který dává počítačům schopnost učit se, aniž by byly explicitně naprogramovány" [2]. Strojové učení hledá vhodnou funkci, která co nejpřesněji mapuje vstup či sekvenci vstupů na odpovídající výstup. Má mnoho metod, pomocí kterých je možno toto mapování implementovat. Mezi často používané patří rozhodovací stromy [3], algoritmus k-nejbližších sousedů [4], diskriminační analýzy [5], neuronové sítě [6], Bayesovské klasifikátory [7, 8] a genetické algoritmy [9].

Jako příklad strojového učení může být implementace programu, který má za úkol rozpoznat požadovaný objekt v komplikovanější scéně. Jedná se o běžnou věc našeho

života, lidský mozek je zaměstnán rozpoznáváním denně. Protože však není známo, jak konkrétně rozpoznávání uvnitř našeho mozku funguje, není možno určit, jak sestavit program, který by byl schopen rozpoznat požadovaný objekt. I kdyby bylo známo, jak program sestavit, takový program by byl pravděpodobně extrémně komplikovaný.

Namísto ručního psaní komplikovaného programu pro každou specifickou úlohu je shromážděno mnoho tréninkových příkladů určujících odpovídající výsledek (výstup) pro dané zadání (vstup). Algoritmus strojového učení poté zpracuje tyto příklady a požadovaný program aproximuje. Výsledný program může vypadat velice rozdílně od typického ručně psaného programu. Obvykle sestává z milionů čísel reprezentujících parametry, pomocí kterých lze vyjádřit funkci pro mapování daných vstupů na odpovídající výstupy. Pokud je program vytvořen správně, měl by pracovat pro nové, neznámé příklady stejně dobře jako pro ty, na kterých byl naučen. Pokud se vstupní data změní, je možno změnit také program předložením těchto změněných dat, na kterých se program přeučí [10].

Tento přístup znamená veliké množství výpočtů, avšak v současné době, kdy počítače disponují obrovskou výpočetní rychlostí, je tento přístup často levnější, než ruční tvoření programů.

1.1.1 Typy učení

Ve strojovém učení existují tři typy učícího algoritmu: učení s učitelem, učení bez učitele a učení se zpětnou vazbou. V této práci budou popsány první dva typy, které jsou nejpoužívanější [10].

1.1.1.1 Učení s učitelem

Rozpoznávací algoritmus se učí predikovat výstup z předloženého vstupního vektoru. Každý tréninkový případ se skládá ze vstupního vektoru x a odpovídajícího výstupu y. U regresního typu úloh je výstupem obvykle reálné číslo, nebo vektor reálných čísel (např. cena akcií za 6 měsíců). U klasifikačních úloh je výstupem označení příslušnosti k určité třídě (např. rozpoznávání ručně psaných číslic 0 až 9). S těmito daty poté nastává proces učení.

Učení s učitelem je mapování vstupu na odpovídající výstup. Lze jej vyjádřit vztahem:

$$y = f(x; W)$$

kde f je funkce, která mapuje vstupní vektor x a numerické parametry (váhy) W na výstup y. V procesu učení jsou tyto numerické parametry W upravovány tak, aby mapování co nejvíce odpovídalo vzorovým datům (snaha o minimalizaci chyby).

1.1.1.2 Učení bez učitele

U tohoto typu učení má rozpoznávací algoritmus za cíl nalézt vhodnou interní reprezentaci vstupního vektoru. Je tzv. „samoorganizovaný“ a sám rozhodne, která repezentace je pro daný vstup nejlepší a podle toho se přizpůsobí. Používá se například u úloh shlukové analýzy. V případě klasifikace, čímž se zabývá i tato práce, není možno učení bez učitele použít, neboť v případě učení bez učitele by byla poskytnuta pouze vstupní data, tedy časy dwell time a flight time, a není jednoznačně určeno, ke kterému uživateli daný vstup patří. Protože je algoritmus samoorganizovaný, mohl by si v datech nalézt nevhodné rozlišovací

znaky a v takovém případě by i vstupní data byla nevhodně roztříděna k jednotlivým uživatelům. Mohl by také např. určit, že uživatelů je více (nebo méně), než je skutečný počet. Například konkrétně u jednoho uživatele v této práci existuje několik vzorků jednoho hesla, kde uživatel stiskne další klávesu dříve, než předchozí klávesu pustil. Tedy „flight time“ mezi těmito klávesami má hodnotu 0. Neplatí to však pro všechny vzorky od tohoto uživatele, resp. existují i vzorky, kde je tato hodnota větší jak 0. Nejedná se proto o vhodný rozlišovací znak. Algoritmus učení bez učitele však tuto informaci nemá a mohl by všechny vzorky, kde tato hodnota je rovna 0 považovat za jednoho uživatele, a tam, kde je hodnota větší jak 0, za uživatele jiného. Pokud jsou k dispozici správné výstupy k daným vstupům, je vhodnější použití učení s učitelem, neboť algoritmus má pro naučení k dispozici více informací.

1.1.2 Typy úloh strojového učení

Typy úloh, pro které jsou využívány učící algoritmy, jsou: klasifikace, regresní analýza a shluková analýza.

Klasifikace je druh úlohy, kdy je cílem určit, do které z kategorií výstupních dat daný vstup patří. Například rozpoznávání ručně psaných číslic 0 až 9 [11], nebo téma této práce - rozpoznávání osob, tedy autentizace.

Regresní analýza je úloha, kde je aproximována hodnota jisté veličiny (výstupu) na základě znalosti jiných veličin (vstupních dat) a cílem je mezi nimi nalézt funkční vztah. Např. odhadování ceny nemovitosti na základě jejích parametrů.

Na rozdíl od dvou předchozích typů úloh spadá shluková analýza pod učení bez učitele. U shlukové analýzy jsou vstupní data rozdělena do několika shluků (clusterů). Např. pokud jsou zkoumané prvky různé automobily, výstupem je přesný počet shluků symbolizující skupiny automobilů podle daných vstupů (např. podle spotřeby paliva, roku výroby, apod).

V rámci této diplomové práce budou v následujících kapitolách teoreticky popsány použité metody strojového učení z pohledu klasifikace.

1.2 Algoritmus k-nejbližších sousedů

Algoritmus k-nejbližších sousedů je metodou strojového učení spadající do rozpoznávání vzorů (instance based learning) jehož základem je vykonat co možná nejméně výpočtů během fáze učení. Většina výpočtů se děje až při vyhodnocování (rozpoznávání). Pokud algoritmus obdrží trénovací data $\langle x_i, f(x_i)\rangle$, kde x_i je vektor vstupních dat a $f(x_i)$ je třída odpovídající daným vstupním datům, algoritmus učení data pouze uloží. Jedná se o nejrychlejší trénovací algoritmus ze všech metod strojového učení. Trénovací data jsou pouze uložena do databáze [2].

Ve fázi rozpoznávání, pokud je algoritmu předložena nová instance vstupních dat x_q, v databázi trénovacích případů je nalezen nejpodobnější trénovací příklad x_n, poté je odhadnuta výstupní třída $\hat{f}(x_q)$ na základě třídy nejbližšího trénovacího příkladu (nejbližšího souseda):

$$\hat{f}(x_q) \leftarrow f(x_n)$$

Příkladem může být nový pacient vykazující určité symptomy nemoci, bez známé diagnózy. Přičemž již existuje celá databáze mnoha pacientů s jejich vlastními symptomy a již známými diagnózami. V této chvíli algoritmus projde celou databázi pacientů, kde již jsou známy diagnózy a nalezne nejvíce se podobající případ, resp. pacienta, jehož symptomy se nejvíce blíží příznakům pacienta nového. Je-li databáze pacientů dostatečně rozsáhlá, je velká pravděpodobnost, že diagnóza, která byla určena pro pacienta v databázi, bude platit i pro pacienta nového.

Ve výše uvedeném příkladu je vyhledáván právě jeden nejbližší soused (tj. k = 1). Jedná se o speciální případ algoritmu k-nejbližších sousedů, který nepracuje příliš spolehlivě, protože je náchylný na šum v trénovacích datech. Aby algoritmus pracoval spolehlivěji, je hledáno více, obecně k nejbližších sousedů, namísto pouze jednoho. Pokud je předložen nový vektor vstupních dat x_q, je nalezeno k nejbližších sousedů (např. 3 až 5). Každý ze sousedů má svou odpovídající výstupní třídu, mezi nimiž je hlasováno.

V případě, že je odhadována výstupní hodnota pro spojitou veličinu, je možno výstupní veličinu vypočítat jako průměr, případně vážený průměr z výstupních hodnot nalezených sousedů.

1.2.1 Metrika pro nalezení nejbližšího souseda

Nejbližší soused mezi vzorky dat je nalezen na základě měření vzdálenosti. Existuje několik metrik, pomocí kterých je možno tyto vzdálenosti měřit. Za účely této diplomové práce, kdy jsou porovnávány spojité veličiny, tedy dwell time a flight time, je zde popsána pouze metrika euklidovské vzdálenosti. Toto je metrika, která se v algoritmu k-nejbližších sousedů používá právě pro spojité veličiny. V případě diskrétních veličin je možno použít např. metriku překrývání (overlap metric) nebo Hammingovu vzdálenost [2].

Metrika euklidovské vzdálenosti je popsána vztahem:

$$d_e(x_q, x_n) = \sqrt{\sum_{i=1}^{m} (x_{q,i} - x_{n,i})^2}$$

kde x_q je vstupní vektor dat, která algoritmus rozpoznává, x_n je jeden z vektorů trénovacích dat z databáze. Veličina $d_e(x_q, x_n)$ je pak euklidovská vzdálenost mezi těmito dvěma vektory. Jako nejbližší vektor je zvolen takový vektor x_n, který je svými vlastnostmi nejblíže vektoru x_q (resp. vektor s nejmenší euklidovskou vzdáleností). Vektoru x_q je poté přiřazena výstupní třída vektoru x_n, tj.:

$$\hat{f}(x_q) \leftarrow f(x_n)$$

Hranice mezi vektory příslušejících k jednotlivým výstupním třídám lze reprezentovat v rovině pomocí tzv. Voroného diagramu.

1.2.2 Voroného diagram

Definice Voroného diagramu zní [12]: Nechť $\mathbb{P} = \{P_1, \dots, P_n\}$ je množina n různých bodů v rovině, které nazýváme generující body. Voroného diagram množiny bodů $\mathbb{P}$ je rozdělení

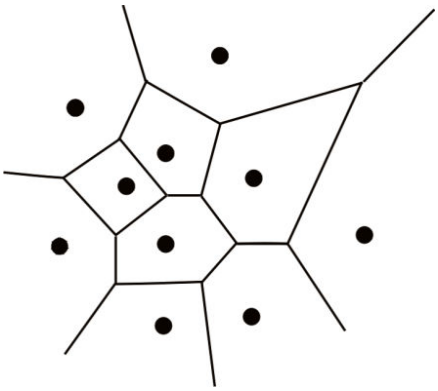
roviny na n buněk příslušných k jednotlivým bodům P_i takových, že libovolný bod Q leží v buňce příslušné k bodu P_i právě tehdy, když:

$$|QP_i| < |QP_j|,$$

$$\forall P_j \in \mathbb{P}, j \neq i.$$

Z definice je zřejmé, že Voroného diagram dělí prostor na několik buněk. Každá pak obsahuje všechny body, které mají nejblíže k jednomu bodu z množiny $\mathbb{P}$. Všechny Voroného buňky jsou konvexní mnohoúhelníky. Hranicí mezi dvěma sousedními oblastmi je úsečka (případně polopřímka, nebo přímka), která leží na ose úsečky spojující odpovídající body sousedních oblastí [13].

Ukázka Voroného diagramu je znázorněna na obrázku 2.



Obrázek 2 - Ukázka Voroného diagramu

Každá z buněk reprezentuje danou výstupní třídu. Třída $\hat{f}(x_q)$ predikovaná algoritmem k-nejbližších sousedů je třída té buňky, do které spadá také vektor x_q.

1.2.3 Teoretické výhody a nevýhody

Výhodami algoritmu k-nejbližších sousedů a obecně metodiky rozpoznávání vzorů (instance based learning) jsou:

- rychlý proces učení,
- schopnost naučit se i velmi komplexní funkce díky tomu, že se algoritmus nesnaží nalézt komplikovaný funkční vztah, ale pouze hledá nejblíže odpovídající vzorek,
- fakt, že při učení nedochází ke ztrátě učících dat (je zachována celá databáze učících dat).

Nevýhodami naopak jsou:

- fáze rozpoznávání může být velmi pomalá, resp. je tím pomalejší, čím více je učících dat. Proto je tento algoritmus nevhodný tam, kde je třeba rozpoznávání aplikovat v reálném čase (robotika, apod.) a nad rozsáhlými databázemi,
- databáze může vyžadovat velké množství kapacity,
- algoritmus může být oklamán nerelevantními atributy, na základě kterých může dát nesprávný výstup.

Poslední zmíněná nevýhoda je nejpodstatnější. Například pokud ve vstupním vektoru jsou informace o osobách, z nichž většina je irelevantních a pouze na několika z nich opravdu záleží, proces hledání nejbližšího souseda je ovlivněn i nerelevantními atributy a může být určen nesprávný výstup.

1.3 Neuronové sítě

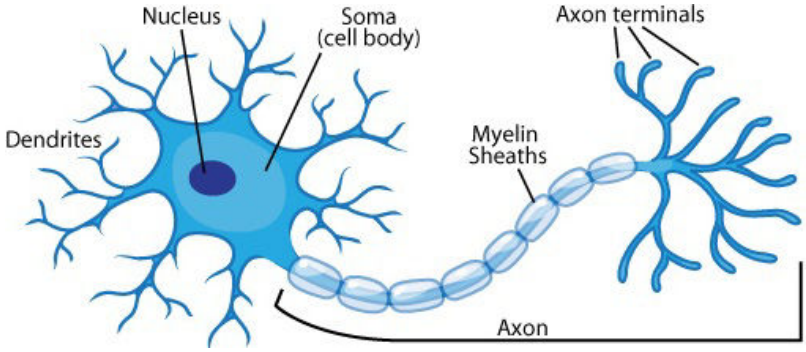
Umělá neuronová síť je jeden z algoritmů strojového učení inspirovaných biologickými systémy. Mezi výhody neuronové sítě patří snadné distribuované paralelní zpracování dat. Historie modelování neuronových sítí sahá až do poloviny 20. století. Jde o výpočetní model, který je odvozen od chování biologických neuronů uvnitř mozku.

1.3.1 Konstrukce biologické a umělé neuronové sítě

Termín neuronová síť, který je běžně používán, je ve skutečnosti nevhodný. Počítače se snaží simulovat funkci biologických neuronových sítí implementací umělé neuronové sítě, která je však velmi zjednodušeným modelem. Nicméně většina publikací používá termín „neuronová síť“ namísto pojmu „umělá neuronová síť“ (ANN – artificial neural network). Pokud není před pojmem neuronová síť určeno, že se jedná o „biologickou“ či „umělou“, téměř jistě se jedná o umělou neuronovou síť [14]. Aby bylo možno porozumět její funkci, je nutno se podívat, jak funguje biologická neuronová síť.

Pro zkonstruování počítačového programu, který by uměl myslet podobně jako člověk, byl použit jediný dostupný pracující model – lidský mozek. Nicméně lidský mozek jako celek je příliš složitý, než aby bylo možno simulovat jeho komplexní funkčnost. Proto byla pozornost zaměřena na studium samotných nervových buněk – neuronů. Jedná se o základní stavební prvky lidského mozku. Umělé neuronové sítě se snaží simulovat chování těchto buněk (neuronů).

Znázornění neuronu je možno vidět na obrázku 3. Neuron se skládá z dendritů (dendrites), které shromažďují vstupní informace z ostatních neuronů, těla neuronu (cell body), axonu a terminály (axon terminals), která je větvena a napojena na další neurony. Jakmile neuron pomocí dendritů přijme dostatečně velký signál, vyšle vzruch, který je přenesen pomocí axonu a terminály dále na další neurony [15].



Obrázek 3 - Znázornění stavby neuronu [41]

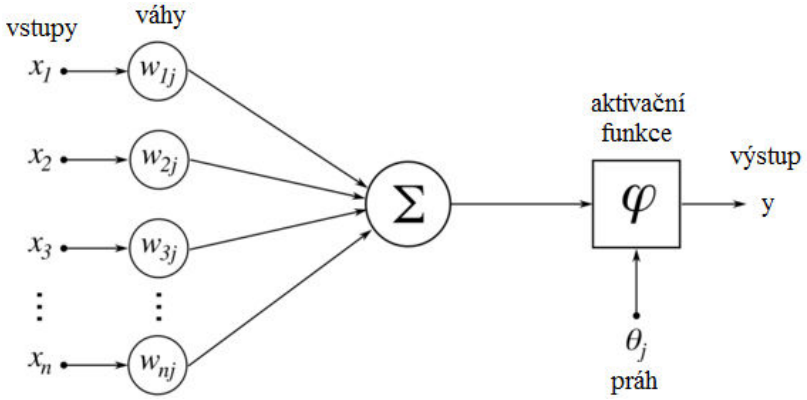
Místo, kde axon jednoho neuronu je napojen na dendrit dalšího neuronu, je nazýváno synapse. Jedná se tedy o spojení dvou neuronů sloužící k předávání vzruchů. Neurony se přímo nedotýkají, je mezi nimi mezera. K přenosu vzruchu dojde pomocí chemické látky –

neurotransmitteru, která je vyloučena z nervového zakončení jednoho neuronu a přenesena na následující neuron, kde způsobí vzruch.

Synapse mají schopnost přizpůsobovat se a měnit svoji efektivitu tím, že mění množství neurotransmitteru, který je uvolněn při přenosu vzruchu, nebo citlivost dalšího neuronu na tento přenos. Mění tedy sílu resp. váhu přenosu. Toto je z velké části podstata schopnosti učení se. Tím, že synaptické váhy mohou být přizpůsobovány, celá síť se může učit úkoly jako rozpoznávání objektů, porozumění řeči, kontrola pohybu těla, apod.

Až 100 trilionů[16] synaptických spojení, resp. synaptických vah, kterými lidský mozek disponuje, může ovlivnit probíhající procesy uvnitř mozku ve velmi krátkém čase v řádu milisekund. V porovnání s vlastností bandwidth moderních počítačových stanic jde stále o řádově vyšší výkon [10].

Uspořádání biologické neuronové sítě je vzorem pro ty umělé. V umělých neuronových sítích je součástí neuronu vstupní část, která přijímá vstupy z ostatních neuronů. Na základě vah mohou být jednotlivé vstupy potlačeny, nebo naopak zvýhodněny. Funkční část neuronu zpracuje informace ze vstupů a vygeneruje výstup. Výstupní část potom přivede výslednou informaci na vstup jiného neuronu. Znázornění umělého neuronu je vidět na obrázku 4.



Obrázek 4 - Znázornění stavby umělého neuronu

Výstup neuronu y je aktivován ve chvíli, když suma vstupů do neuronu x_i vynásobených jejich konkrétními vahami w_i překročí určitou prahovou hodnotu Θ. Neuron lze popsat následujícím vztahem:

$$y = f\left(\sum_{i=1}^{n} x_i w_i - \Theta\right)$$

kde x_i je konkrétní hodnota na i-tém vstupu, w_i je váha tohoto vstupu, Θ je prahová hodnota, n je celkový počet vstupů, f je aktivační funkce a y je hodnota na výstupu neuronu.

1.3.2 Typy neuronů

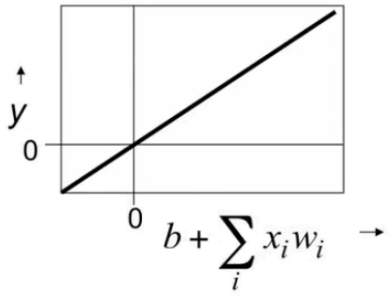
Neurony lze rozdělit na několik typů podle jejich aktivační funkce. V této práci jsou popsány nejpoužívanější typy aktivačních funkcí.

1.3.2.1 Lineární neuron

Neuron s lineární aktivační funkcí je jednoduchý, ale limitovaný co do jeho schopností, neboť síť složená z těchto neuronů není schopna aproximovat nelineární funkce. Je tedy vhodný k úlohám týkajících se lineárních závislostí [17]. Je dán vztahem:

$$y = b + \sum_{i=1}^{n} x_i w_i$$

kde y je výstup neuronu, b je bias (číselná konstanta, která umožňuje posunout aktivační funkci nahoru či dolů a ovlivnit tak výstupní hodnoty neuronu) a $\sum_{i=1}^{n} x_i w_i$ je součet vstupních hodnot neuronu vynásobený jejich vahami. Průběh funkce lineárního neuronu je vidět na obrázku 5 [10].



Obrázek 5 – Průběh funkce lineárního neuronu [10]

1.3.2.2 Binární prahový neuron

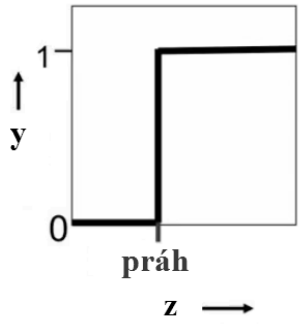
V první řadě je spočítána suma vstupních hodnot vynásobená jejich vahami $\sum_{i=1}^{n} x_i w_i$. Poté neuron vyšle konstantní výstupní signál („vzruch“) pokud $\sum_{i=1}^{n} x_i w_i + b$ přesáhne daný práh Θ. Každý neuron kombinuje signály na vstupu, aby určil svůj vlastní výstup.

Binární prahová aktivační funkce je dána vztahy:

$$z = b + \sum_{i=1}^{n} x_i w_i$$

$$y = \begin{cases} 1 \text{ pokud } z \geq \Theta \\ 0 \end{cases}$$

Průběh funkce je znázorněn na obrázku 6.



Obrázek 6 - Průběh funkce binárního prahového neuronu [10]

1.3.2.3 Rektifikovaný lineární neuron

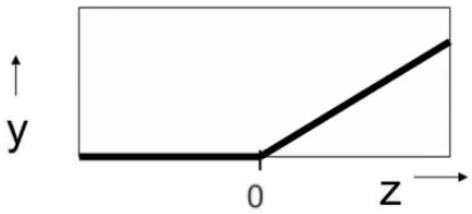
Někdy je také nazýván lineární prahový neuron. Vypočítává lineární vážený součet vstupů stejně jako lineární neuron, avšak výstupní signál vyšle jen tehdy, je-li vypočtený součet větší než 0. Výstupem je nelineární funkce celkového vstupu [18, 19].

Rektifikovaný lineární neuron je dán vztahy [10]:

$$z = b + \sum_{i=1}^{n} x_i w_i$$

$$y = \begin{cases} z \text{ pokud } z \geq 0 \\ 0 \end{cases}$$

Průběh funkce rektifikovaného lineárního neuronu je znázorněn na obrázku 7.



Obrázek 7 - Průběh funkce rektifikovaného lineárního neuronu [10]

Tyto neurony umožňují získat chování lineárních systémů, pokud z je větší než 0 a možnost ignorovat vstup, pokud z je menší než 0.

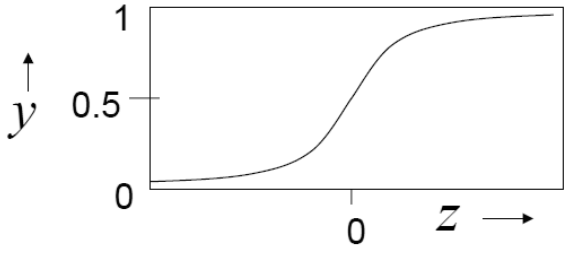
1.3.2.4 Sigmoid neuron

Tento typ neuronu vrací plynulou funkční hodnotu jeho celkového vstupu. Je definován vztahem [10]:

$$z = b + \sum_{i=1}^{n} x_i w_i$$

$$y = \frac{1}{1 + e^{-z}}$$

Průběh funkce Sigmoid neuronu je znázorněn na obrázku 8.



Obrázek 8 - Průběh funkce Sigmoid neuronu [10]

Výhodou tohoto neuronu je, že díky „hladkému“ funkčnímu průběhu je první derivace aktivační funkce spojitá, což umožňuje učení neuronové sítě složené z tohoto typu neuronů pomocí gradientních metod. Díky tvaru funkčního průběhu se také kontinuálně mění první derivace, což umožňuje snadné učení [10, 20].

1.3.2.5 Stochastické binární neurony

Používají stejné rovnice, jako Sigmoid neurony:

$$z = b + \sum_{i=1}^{n} x_i w_i$$

Průběh funkce je znázorněn na obrázku 8. Avšak celkový výsledek y není brán jako výstupní hodnota, ale jako pravděpodobnost vyslání výstupního signálu. Výstupem tohoto neuronu je hodnota 0 nebo 1.

$$p(s = 1) = \frac{1}{1 + e^{-z}}$$

Pokud p je číslo blížící se k hodnotě +1, téměř vždy neuron vyšle vzruch ($y = 1$). Pokud p je číslo blížící se k hodnotě -1, výstupem bude téměř vždy $y = 0$ [21].

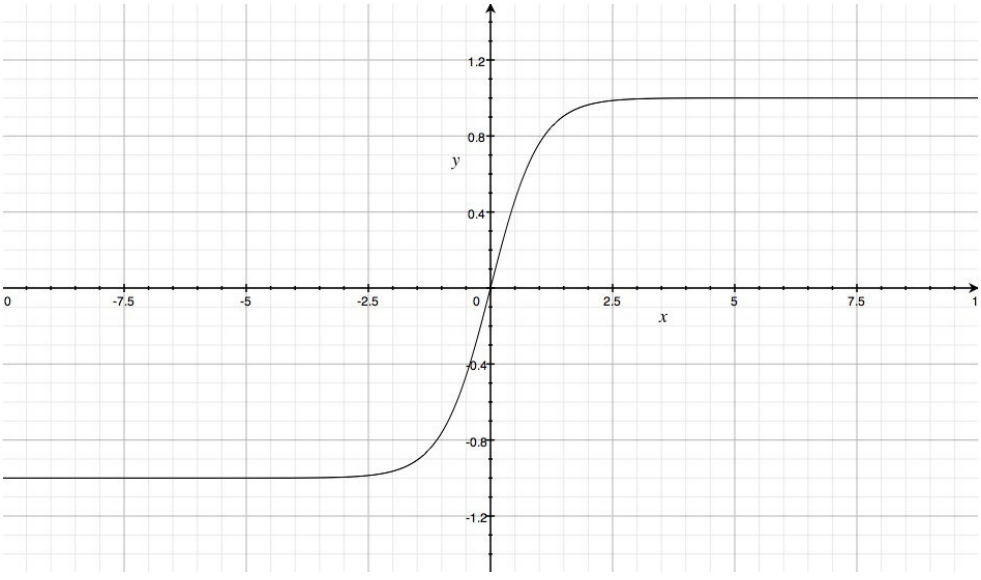
1.3.2.6 Neuron s aktivační funkcí hyperbolického tangentu

Tento typ neuronu vrací taktéž plynulou funkční hodnotu jeho celkového vstupu, jako u neuronu s aktivační funkcí Sigmoid. Aktivační funkce je dána vztahem:

$$z = b + \sum_{i=1}^{n} x_i w_i$$

$$y = \frac{e^{2z} - 1}{e^{2z} + 1}$$

Průběh funkce hyperbolického tangentu je znázorněn na obrázku 9.



Obrázek 9 - Průběh funkce hyperbolického tangentu [22]

Z průběhu aktivační funkce je vidět, že má podobný charakter, jako aktivační funkce Sigmoid, tedy má i stejné výhody (spojitá první derivace, snadné učení). Podstatný rozdíl těchto dvou funkcí je v tom, že aktivační funkce hyperbolického tangentu pracuje s výstupními hodnotami v intervalu $\langle -1; 1 \rangle$, zatímco aktivační funkce Sigmoid v intervalu $\langle 0; 1 \rangle$ [22].

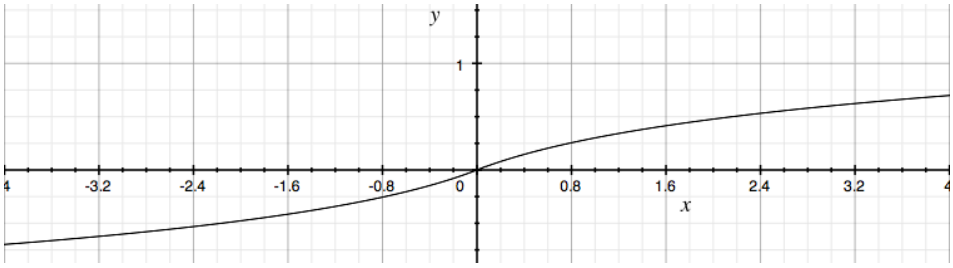
1.3.2.7 Neuron s logaritmickou aktivační funkcí

Jako předchozí dva typy neuronů, i tento vrací spojitou funkční hodnotu. Aktivační funkce je dána vztahy:

$$z = b + \sum_{i=1}^{n} x_i w_i$$

$$y = \begin{cases} \log(1+z), z \geq 0 \\ \log(1-z), z < 0 \end{cases}$$

Průběh logaritmické aktivační funkce je znázorněn na obrázku 10.



Obrázek 10 - Průběh logaritmické aktivační funkce [22]

Logaritmická aktivační funkce je užitečná k prevenci proti saturaci. Neuron je tzv. saturovaný, pokud na daný vstup vrací ve většině případů výstup přibližně 1 nebo -1. Saturace může nastat u aktivační funkce Sigmoid i hyperbolického tangentu, kdy na krajích funkce, kde obě funkce vracejí právě hodnoty blížící se 1 nebo -1, je první derivace funkce příliš malá a v takovém případě se síť hůře učí. Logaritmická aktivační funkce je možným řešením díky strmějšímu průběhu funkce na koncích grafu. Logaritmická aktivační funkce přijímá i vrací hodnoty v intervalu $\langle -1; 1 \rangle$ [22].

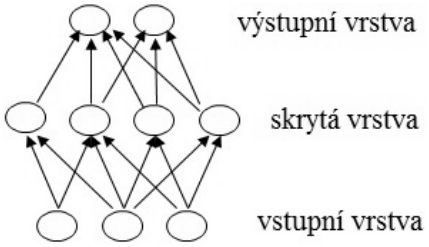
1.3.3 Typy architektur neuronových sítí

Pojmem architektura neuronových sítí je myšlen způsob, jakým jsou jednotlivé neurony v síti spolu propojeny. Nejčastějším typem architektury neuronové sítě je tzv. dopředná (feedforward) architektura, kde signál přichází do vstupních neuronů a prochází jedním směrem skrze skryté vrstvy neuronů do výstupní vrstvy. Tento typ architektury je použit pro účely této práce.

Druhým typem architektury je rekurentní neuronová síť, ve které signál může procházet v cyklech. Tyto neuronové sítě si mohou pamatovat informace dlouhou dobu a mohou projevovat zajímavé oscilace, ale je mnohem náročnější je naučit v porovnání s dopřednými neuronovými sítěmi právě proto, že jsou mnohem komplikovanější.

1.3.3.1 Dopředná neuronová síť

Dopředná neuronová síť je pro svou jednoduchost nejpoužívanějším typem architektury. Neurony jsou organizovány do vrstev, kde výstupy předcházejících vrstev jsou použity jako vstupy do vrstev následujících. První vrstvu tvoří vstupní neurony, výstupem sítě jsou hodnoty neuronů v poslední výstupní vrstvě. Pokud je v neuronové síti více jak jedna skrytá vrstva, pak se jedná o tzv. hluboké neuronové sítě (deep neural networks). Architektura takové sítě je znázorněna na obrázku 11.

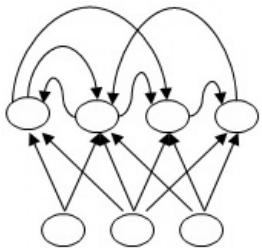


Obrázek 11 - Náznak struktury dopředné neuronové sítě [10]

Tyto sítě počítají sérii transformací mezi jejich vstupem a výstupem. S každou další vrstvou vzniká nová reprezentace vstupních dat, kde vlastnosti, které byly v předchozí vrstvě podobné, se mohou stát méně podobné ve vrstvě další, nebo naopak vlastnosti, které si nebyly podobné, se mohou stát podobnými. Konkrétně v případě rozpoznávání hesel pro biometrickou autentizaci osob - žádný člověk nenapíše dvakrát po sobě své heslo stejným způsobem, vždy se způsob napsání odlišuje. V takovém případě je žádáno, aby pokusy, kdy jedno heslo je napsané vícekrát jednou osobou, byly vyhodnocovány jako podobné a naopak pokusy, kdy toto heslo je napsané osobami různými, byly vyhodnoceny jako rozdílné. Aby tohoto stavu bylo dosaženo, je třeba, aby aktivita neuronů v každé vrstvě byla nelineární funkcí aktivit vrstvy předchozí [10].

1.3.3.2 Rekurentní neuronové sítě

Jsou schopnější, než dopředné neuronové sítě. Mají řízené cykly ve své struktuře. To znamená, že pokud informace začíná u jednoho neuronu a následuje cestu propojení skrze neurony další, může se nakonec dostat zpět do toho samého neuronu, u kterého začala. Toto je však může činit složité k natrénování. Architektura tohoto typu sítě je znázorněna na obrázku 12.



Obrázek 12 - Náznak struktury rekurentní neuronové sítě [10]

Jak již bylo zmíněno, tento typ neuronové sítě je schopen si ve svých skrytých vrstvách zapamatovat informaci po velice dlouhou dobu, avšak je velice složité je natrénovat, aby tuto schopnost dokázaly využít. Z biologického pohledu je tento typ neuronové sítě více realistický.

Tento typ sítě je například používán pro predikci dalšího znaku ve slově. Konkrétně např. pokus z roku 2011, kdy Ilya Sutskever použil pro tento pokus speciální typ rekurentní neuronové sítě. Pro naučení byl použit text z Wikipedie [10].

1.3.4 Backpropagation učící algoritmus

Jak již bylo uvedeno v předchozích kapitolách, proces učení je upravování vah vedoucí k minimalizaci chyby, resp. minimalizaci celkové chyby v součtu skrze všechny

tréninkové případy. Nejjednodušší příklad je lineární neuron. Zde k úpravě vah může být použito např. čtvercové odchylky mezi aktuálním výstupem neuronu a chtěným výstupem. Lineární neuronové sítě jsou však velice omezené svými schopnostmi. Ve většině případů jsou použity nelineární neuronové sítě, které jsou navíc vícevrstvé (mohou mít i více než 1 skrytou vrstvu). Bylo tedy třeba vytvořit obecnou metodu, která bude umožňovat přizpůsobování vah také mezi nelineárními neurony i ve vícevrstvých sítích.

Backpropagation je jedna z nejstarších a nejpoužívanějších metod pro dopředné neuronové sítě, která tuto schopnost má. Cílem algoritmu backpropagation je výpočet derivací mezi neurony uvnitř struktury neuronové sítě na základě změny celkové chyby. Tyto derivace se pak dále používají pro přizpůsobování vah mezi jednotlivými neurony. Tím je možno automaticky nacházet ve vstupních datech vhodné rozlišovací znaky (features) a přizpůsobovat jim váhy uvnitř neuronové sítě tak, aby predikovaný výstup generoval co nejmenší chybu. Celý tento proces je možno nazvat „učením“ sítě. Děje se automaticky strojově, tedy bez nutnosti zásahů člověka do struktury sítě.

Jednou z možností, jak toto zajistit, je náhodně měnit váhy spojení mezi jednotlivými neurony a sledovat vliv těchto změn na celkovou chybu. Tento přístup je však velice neefektivní, neboť spojení je v síti velmi mnoho (mohou jich být i miliony) a pro změnu každé váhy je třeba udělat několik průchodů sítí, aby byl zjištěn vliv změny na celkovou chybu. Namísto toho se nabízí možnost náhodně měnit výstupy neuronů, namísto váhy jejich spojení. Tím, že neuronů je řádově méně, než jejich spojení, je algoritmus efektivnější. I přesto je však méně efektivní než algoritmus backpropagation.

Myšlenka algoritmu backpropagation spočívá v tom, že sice nelze v obecném případě určit, co skryté vrstvy neuronů v síti aproximují (proto se nazývají skryté vrstvy), ale lze vypočítat, jak rychle se mění celková chyba pro daný tréninkový případ s tím, jak se mění výstupy skrytých neuronů. Vztah pro výpočet chyby neuronů výstupní vrstvy v jednom tréninkovém případu je dán jako polovina součtu čtvercových odchylek výstupů všech neuronů výstupní vrstvy:

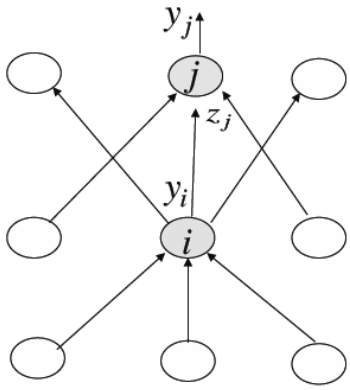
$$E = \frac{1}{2}\sum_{j}\left(t_j - y_j\right)^2$$

kde t_j je správná (očekávaná) hodnota výstupu neuronu, a y_j je aktuální hodnota výstupu neuronu (hodnota určená neuronovou sítí). První derivace tohoto vztahu podle proměnné y_j (výstupní hodnota neuronu) je dána:

$$\frac{\partial E}{\partial y_j} = -\left(t_j - y_j\right)$$

Tímto způsobem je vypočítána chyba v každé vrstvě neuronové sítě. Zjednodušeně řečeno, algoritmus použije velikost změny chyby ve vrstvě jedné a z ní vypočte velikost změny chyby ve vrstvě předchozí.

Na obrázku 13 je naznačena zjednodušená struktura sítě. Konkrétně výstupní neuron j výstupní vrstvy a neuron i skryté vrstvy. Výstup neuronu i je symbolizován jako y_i, výstup neuronu j je y_j a celkový vstup přijatý výstupním neuronem j je značen jako z_j.



Obrázek 13 - Zjednodušený nákres neuronové sítě s naznačením vrstev [10]

Jakmile je známa velikost změny chyby ku změně výstupu výstupního neuronu j, tedy $\frac{\partial E}{\partial y_j}$, je nutno převést tento vztah na velikost chyby ku změně celkového vstupu (z_j) do neuronu j:

$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial E}{\partial y_j} = y_j(1 - y_j)\frac{\partial E}{\partial y_j}$$

Nyní je možno vypočítat velikost změny chyby ku změně výstupu neuronu i, tedy y_i:

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial z_j}$$

Vztah $\frac{dz_j}{dy_i}$ je roven váze spojení neuronu i a j. Ta je značena w_{ij}. Jedná se tedy o součet hodnot všech výstupních spojení neuronu i dané jako součin váhy spojení w_{ij} a velikostí změny chyby, která již byla vypočítána, tj. $\frac{\partial E}{\partial z_j}$.

A konečně je také možno určit velikost změny chyby ku změně váhy spojení $\frac{\partial E}{\partial w_{ij}}$:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_i\frac{\partial E}{\partial z_j}$$

Tedy pravidlo pro změnu hodnoty váhy spojení mezi neurony v algoritmu backpropagation je součin změny chyby ku změně celkového vstupu z_j neuronu j, tedy $\frac{\partial E}{\partial z_j}$, a výstupu neuronu i, tedy y_i.

Algoritmus prochází v opačném směru architekturou sítě a upravuje hodnoty vah dle tohoto pravidla. Tento postup je prováděn pro všechny vrstvy neuronové sítě zpětně [10].

1.3.5 Návrh struktury neuronové sítě

Velikost sítě se netýká pouze počtu vrstev, ale také počtu neuronů každé jednotlivé vrstvě a počtu spojení mezi nimi. Pro danou skupinu dat může být nekonečně mnoho struktur relevantních k tomu, aby se naučily charakteristiky z daných dat. Otázkou je, jak velká síť bude nejvhodnější pro danou skupinu dat. Bohužel tuto otázku není jednoduché zodpovědět. Architektura neuronové sítě, která bude dávat nejlepší výsledek pro zkoumaný

případ, může být určena pouze experimentálně. Kvalita řešení nalezená neuronovou sítí je silně závislá na velikosti použité sítě. Obecně řečeno, velikost sítě ovlivňuje komplexnost sítě a čas potřebný pro naučení, ale hlavně ovlivňuje schopnost generalizace. Generalizace je schopnost sítě interpretovat a klasifikovat data, která síť před tím neviděla. Síla neuronové sítě závisí na tom, jak dobře umí generalizovat z daných tréninkových dat. Velikost a charakteristika tréninkových dat dohromady s počtem iterací jsou další faktory ovlivňující schopnost generalizace neuronové sítě [23]. Podrobný popis návrhu struktury sítě je popsán v implementační části v kapitole 2.3.3.

1.3.6 Normalizace vstupních dat

Data připravená jako vstup pro neuronovou síť často nelze neuronové síti přímo předložit k vyhodnocení. Neuronové sítě umí aproximovat libovolné funkce [24], avšak pro praktické využití není vhodné, aby přijímala jakákoliv data. Pro získání co nejsmysluplnějších výsledků musí být data nejprve normalizována. Normalizací je omezen definiční obor a obor hodnot. Gradientní metody v průběhu učení lépe konvergují, pokud jsou hodnoty vhodně namapovány na funkční průběh aktivačních funkcí [25].

Neuronové sítě přijímají vstup ve formě vektoru desetinných čísel. Obvykle by tato vstupní čísla měla být pro maximální přesnost výstupu v intervalu $\langle -1; 1 \rangle$ nebo $\langle 0; 1 \rangle$. Volba správného rozsahu je závislá na zvolené aktivační funkci. Aktivační funkce mají rozsah oboru hodnot kladný, nebo kladný a záporný.

Například aktivační funkce Sigmoid má rozsah oboru hodnot pouze v kladných číslech. Naopak aktivační funkce v podobě hyperbolického tangentu má obor hodnot kladných i záporných čísel. Nejčastější je použít aktivační funkci hyperbolický tangent a normalizovat data do intervalu $\langle -1; 1 \rangle$.

Normalizace numerické řady hodnot je mapování existujících číselných hodnot na rozsah v intervalu $\langle -1; 1 \rangle$, resp $\langle 0; 1 \rangle$. Normalizace způsobí, že všechny atributy numerické řady (tréninkové sady) budou ve stejném rozsahu. Pro normalizaci je použito vztahu:

$$f(x) = \frac{(\mathrm{x} - \mathrm{d_L})(\mathrm{n_H} - \mathrm{n_L})}{(\mathrm{d_H} - \mathrm{d_L})} + \mathrm{n_L}$$

kde x je normalizovaný atribut řady, d_L je nejnižší hodnota řady, d_H je nejvyšší hodnota řady, n_L je dolní mez požadovaného rozsahu (-1 nebo 0), n_H je horní mez požadovaného rozsahu (tedy 1) [22]. V případě, že se v souboru dat nacházejí anomální hodnoty, které jsou příliš vzdáleny ostatním hodnotám a zbytečně by zvětšovaly rozsah, používá se k normalizaci průměr (či medián) a rozptyl. Například pokud jsou hodnoty normalizovány na interval $\langle -1; 1 \rangle$ je průměr použit jako prostřední hodnota intervalu, tj. 0, a rozptyl jako krajní hodnoty intervalu, tedy -1 a 1.

1.3.7 Normalizace nominálních hodnot

Nominální hodnoty jsou používány pro pojmenování věcí. Jeden velmi častý příklad jednoduché nominální hodnoty je pohlaví. Případně příklady, které jsou typu Boolean (pravda / nepravda). Nicméně ne všechny nominální hodnoty mají pouze dvě hodnoty.

Nominální hodnoty mohou být také využity pro popsání vlastnosti daného předmětu, např. barva. Neuronové sítě umí nejlépe pracovat s nominálními hodnotami, jejichž množina je fixní a nemění se. Např. rozpoznávání druhů kosatce (anglicky „iris“) – pokud je neuronová síť naučena rozpoznávat pouze 3 druhy (Setosa, Versicolor, Virginica), není možno očekávat, že rozpozná 5 druhů.

Nominální hodnoty je třeba neuronové síti také normalizovat a reprezentovat je číselně. Nejjednodušší způsob reprezentace nominálních hodnot je nazýván „One-of-n“ normalizace [22]. Tento způsob však bývá složitý pro naučení neuronové sítě obzvláště pokud se pracuje s více nominálními hodnotami. Výhodnější volbou je proto rovnostranná (equilateral) normalizace, viz dále.

One-of-n normalizace

Jedná se o jednoduchý způsob normalizace, kdy neuronová síť má tolik výstupních neuronů, kolik je výstupních nominálních hodnot. Predikovaná nominální hodnota je reprezentována neuronem s nejvyšší aktivační hodnotou [22].

Například rozpoznávání druhů kosatce. Vstupní data pro neuronovou síť jsou číselné parametry k dané individuální rostlině. Výstup naznačuje, který druh kosatce posuzujeme, tedy jedná se o nominální hodnoty. Neuronová síť je učena pro 3 druhy kosatce: Setosa, Versicolor, Virginica. Za použití one-of-n normalizace má tedy neuronová síť 3 výstupní neurony. Každý z nich reprezentuje daný druh kosatce. Predikovaný druh kosatce neuronovou sítí odpovídá výstupnímu neuronu s nejvyšší aktivační hodnotou.

Výstupní data při normalizaci one-of-n jsou generována tak, že neuronu, který odpovídá správnému druhu kosatce, je přiřazena hodnota +1, zbývajícím dvěma neuronům je přiřazena hodnota -1. Výstup pro všechny tři druhy rostliny pak vypadá následovně:

- Setosa: 1,-1,-1
- Versicolor: -1,1,-1
- Virginica: -1,-1,1

Ve stádiu trénování neuronové sítě jsou výstupní neurony a jejich hodnoty neustále kontrolovány vzhledem ke správnému ideálnímu výstupu, který je poskytnut. Chyba mezi aktuálním výstupem a ideálním výstupem je reprezentována procentem. Toto může způsobit komplikaci pro metodu one-of-n. Například pokud neuronová síť chybně predikovala druh Versicolor namísto správného druhu Virginica:

- Ideální výstup: -1,-1,1
- Aktuální výstup: -1,1,-1

Komplikace je v tom, že pouze dva ze tří neuronů vykazují chybné hodnoty. Přičemž nejvhodnější je rozprostřít „vinu“ za tuto chybu na větší procento neuronů. Aby to bylo možné, musí být pro každý výstup (druh kosatce) určen unikátní soubor výstupních hodnot, čehož je docíleno v tzv. rovnostranné normalizaci.

Rovnostranná normalizace (equilateral normalization)

V rovnostranné normalizaci má každý soubor hodnot stejnou euklidovskou vzdálenost od ostatních [22]. Stejná vzdálenost zajistí, že chybné zvolení druhu Setosa namísto Versicolor bude mít stejnou váhu, jako chybné zvolení druhu Setosa namísto Virginica.

Na příkladu tří druhů rostlin, jejichž výstupní hodnoty byly normalizovány na hodnoty v rozsahu -1 až 1, výsledkem rovnostranné normalizace je soubor výstupních hodnot:

- Setosa: 0.866, 0.5
- Versicolor: -0.866, 0.5
- Virginica: 0.0, -1.0

Oproti normalizaci one-of-n je u tohoto typu normalizace vždy o jeden výstupní neuron méně. Rovnostranná normalizace se používá pro 3 a více nominálních hodnot.

V tomto případě, pokud by neuronová síť chybně predikovala druh Versicolor namísto Virginica:

- Ideální výstup: 0.0, -1.0
- Aktuální výstup: -0.866, 0.5

Zde jsou pouze dva neurony, přičemž oba dva produkují chybné hodnoty. Navíc jde pouze o dva neurony, které učíme, což nepatrně snižuje složitost neuronové sítě [22].

1.4 Bayesovský přístup

Klasifikátor vycházející z Bayesovského přístupu (tzv. Bayesovský klasifikátor) na rozdíl od algoritmu k-nejbližších sousedů a neuronových sítí pracuje na základě Bayesovy věty z teorie pravděpodobnosti.

Bayesovský přístup hraje dvě významné role v oblasti strojového učení [2]. První role je jeho využití v učících algoritmech (tj. Bayesovský klasifikátor, Bayesovské sítě, apod.). Druhou rolí je jeho využití pro vyhodnocování ostatních klasifikačních algoritmů a jejich porovnávání. V případě Bayesovského přístupu nestačí poskytnout pouze trénovací data jako u algoritmu k-nejbližších sousedů a neuronových sítí. Protože pracuje na bázi pravděpodobnosti, je nutno Bayesovskému klasifikátoru poskytnout také počáteční znalosti (pravděpodobnosti) o problematice.

Jádrem Bayesovského přístupu je tzv. Bayesova věta.

1.4.1 Bayesova věta

Bayesova věta popisuje pravděpodobnost toho, že nastane určitý jev na základě podmínek, které mohou souviset s tímto jevem. Matematicky je Bayesova věta popsána následujícím vztahem [26]:

$$P(h|D) = \frac{P(h)P(D|h)}{P(D)}$$

kde h je zkoumaný jev a D je vektor vstupních dat. Bayesova věta určuje pravděpodobnost $P(h|D)$ což je pravděpodobnost, že nastane jev h na základě předložených dat D. Dále:

- $P(h)$ = počáteční pravděpodobnost jevu h.

- $P(D)$ = počáteční pravděpodobnost trénovacích dat D,
- $P(D|h)$ = pravděpodobnost trénovacích dat D na základě pozorování jevu h.

Názornější interpretaci výše uvedeného vztahu popisuje následující příklad pacienta, u kterého je zkoumána pravděpodobnost rakoviny. Opět je uvažován výše uvedený vzorec Bayesovy věty:

- h je jev, že zkoumaný pacient má rakovinu,
- D je soubor vstupních dat, na základě kterých je rozhodováno, tj. v tomto případě např. výsledky testů, příznaky, apod.,
- $P(h|D)$ je pravděpodobnost, že pacient má rakovinu na základě předložených výsledků testů,
- $P(h)$ obecná pravděpodobnost výskytu rakoviny v lidské populaci,
- $P(D|h)$ pravděpodobnost toho, že testy na rakovinu (tj. data D) prokáží rakovinu u pacienta, který skutečně rakovinu má (jev h),
- $P(D)$ pravděpodobnost, že test na rakovinu bude u pacienta pozitivní bez ohledu na to, zda rakovinu skutečně má, nebo ne.

1.4.2 Bayesovský klasifikátor

Ve vztahu k této diplomové práci a zkoumané problematice dynamiky psaní je Bayesova věta vyjádřena takto [27, 28]:

$$P(y|x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n|y)}{P(x_1, \dots, x_n)}$$

kde x je vektor vstupních dat. Vstupními daty se rozumí vektor hodnot dwell time a flight time jednoho napsaného hesla. Výraz y symbolizuje náležitost vektoru k dané třídě, tedy uživateli, který heslo píše. Výraz $P(y|x_1, \dots, x_n)$ je pravděpodobnost, že daný vektor vstupních dat x náleží k třídě (uživateli) y.

Za předpokladu, že vektory vstupních dat jsou nezávislé, platí:

$$P(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i|y)$$

pro všechna i je výše uvedený výraz zjednodušen na:

$$P(y|x_1, \dots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, \dots, x_n)}$$

Protože hodnota pravděpodobnosti vstupního vektoru $P(x_1, \dots, x_n)$ je konstantní, je možno použít následující klasifikační pravidlo:

$$P(y|x_1, \dots, x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

Pomocí tohoto klasifikačního pravidla je možno nyní odhadnout výslednou třídu pro vektor vstupních dat. K tomuto je použito hypotézy „Maximum a posteriori“, neboli h_{MAP}.

1.4.3 Hypotéza Maximum a posteriori

Pomocí hypotézy h_{MAP} je proveden odhad třídní příslušnosti vstupního vektoru x. Ta maximalizuje pravděpodobnost $P(y|x_1, \dots, x_n)$ a odhaduje třídu $\hat{y}$ s tzv. nejvyšší aposteriorní pravděpodobností [29]:

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

$P(y)$ je obecná pravděpodobnost toho, že vstupní vektor bude náležet ke třídě y. Jedná se o počáteční pravděpodobnost vycházející z určité předchozí zkušenosti či znalosti (např. obecná pravděpodobnost výskytu rakoviny v lidské populaci). V tomto je Bayesovský klasifikátor rozdílný od ostatních metod (viz odstavec 1.4). Tato pravděpodobnost může být vyjádřena například jako relativní četnost třídy y v trénovacích datech.

V případě této diplomové práce je uživatel rozpoznáván pomocí napsaných hesel a tedy pravděpodobnost $P(y)$ by byla relativní četnost vzorků třídy (resp. uživatele) y ve vstupních datech. Toto není vhodný způsob, protože počet vzorků od jednotlivých uživatelů se může lišit. A menší počet vzorků uživatele y_1 než počet vzorků uživatele y_2 neznamená, že pravděpodobnost uživatele y_1 je také menší. Ve skutečnosti pravděpodobnost $P(y)$ na počtu vzorků od jednotlivých uživatelů zcela nezáleží.

Proto je možno výraz ještě zjednodušit a nastavit předpoklad, že pravděpodobnosti $P(y_1) = P(y_2) = P(y_3) = \cdots = P(y_n)$. Za tohoto předpokladu platí [29]:

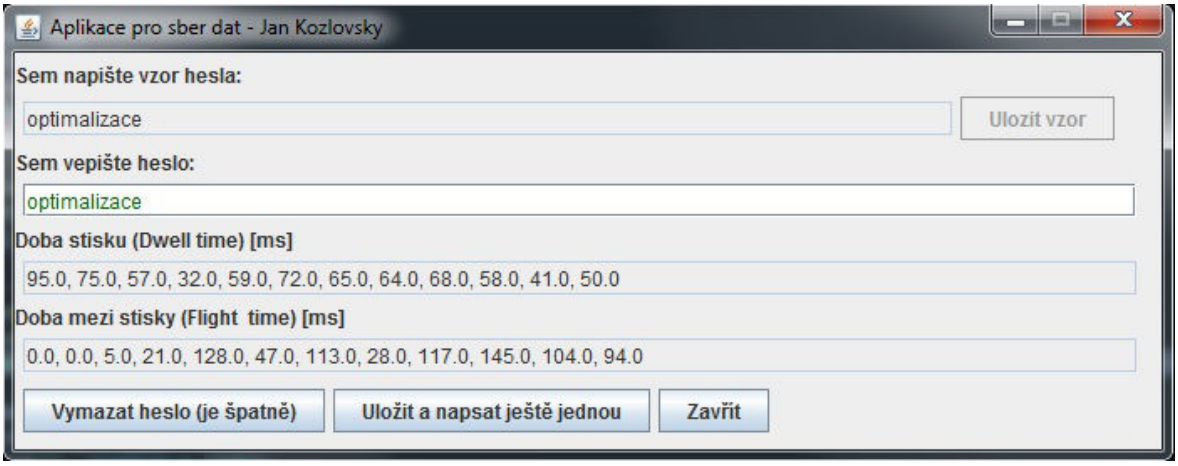
$$\hat{y} = \arg\max_y \prod_{i=1}^{n} P(x_i|y)$$

2 Vlastní implementace

Za účelem této diplomové práce byla vlastní implementace testována za pomocí vzorků od třech různých uživatelů pro dvě různá hesla. Prvním testovacím heslem bylo slovo „optimalizace“, druhým testovacím heslem bylo slovo „velikonocnizajic“.

2.1 Sběr dat a jejich reprezentace

Pro sběr dat byla vytvořena samostatná aplikace s grafickým rozhraním v programovacím jazyce Java (viz obrázek 14). Tato aplikace neměla za potřebí využití metod strojového učení, byla pouze navržena za účelem získání správných znaků (features) z napsaného hesla a jejich export do souboru CSV. Těmito znaky byly časy dwell time a flight time pro jednotlivá písmena hesla.



Obrázek 14 - Ukázka aplikace pro sběr dat

Do aplikace uživatel nejprve napíše vzor hesla, např. „optimalizace“ a uloží jej. Do pole pro vepsání hesla poté vepíše heslo svým obvyklým způsobem. Zde se již sledují zmíněné charakteristiky flight time a dwell time a zapisují se do paměti (taktéž je aplikace zobrazuje ve svém okně). Správnost napsání hesla je poté zkontrolována oproti vzoru. Pokud je napsané dobře, heslo zezelená. Pokud je špatně, heslo zčervená. Jako vzorky jsou použita pouze správně napsaná hesla bez překlepů. Barva hesla rychle uživateli řekne, zda je heslo správně. Ušetří se tak uživateli čas pro jeho případnou kontrolu. Pokud je heslo správně, klikne uživatel na tlačítko „Uložit a napsat ještě jednou“.

Sledované charakteristiky flight time a dwell time jsou časy v milisekundách.

Pro jeden napsaný znak jsou ukládány do třídy `MyCharacter`:

```
public class MyCharacter {
	double flightTime;
	double dwellTime;
}
```

Heslo je poté kolekcí těchto znaků, resp. kolekcí instancí třídy `MyCharacter`:

```
ArrayList<MyCharacter> password;
```

Skupina napsaných hesel je poté kolekce těchto kolekcí:

```
ArrayList<ArrayList<MyCharacter>> listOfPasswords
```

Na konci psaní ve chvíli, kdy uživatel ukončí aplikaci, je aplikací před svým ukončením vygenerován soubor CSV s napsanými hodnotami. Hodnoty napsaných hesel aplikace ukládá do CSV tak, že hodnoty každého jednoho pokusu (hesla) ukládá na samostatný řádek. Sledované charakteristiky flight time a dwell time ukládá pro každý znak za sebe, nejprve hodnotu proměnné flightTime, poté hodnotu proměnné dwellTime. Tedy např. pro heslo, které je vidět na obrázku 14, by hodnoty v CSV souboru vypadaly tímto způsobem:

```
0.0,95.0,0.0,75.0,5.0,57.0,21.0,32.0,128.0,59.0,...,94.0,50.0
```

Ukázku zdrojového CSV souboru vygenerovaného aplikací po jednom ze sběrů dat od jednoho uživatele pro heslo „optimalizace“ je možno vidět na obrázku 15.

```
FT0,DT0,FT1,DT1,FT2,DT2,FT3,DT3,FT4,DT4,FT5,DT5,FT6,DT6,FT7,DT7,FT8,DT8,FT9,DT9,FT10,DT10,FT11,DT11,User
0.0,124.0,0.0,95.0,37.0,81.0,50.0,65.0,120.0,63.0,43.0,108.0,129.0,85.0,14.0,73.0,135.0,104.0,203.0,76.0,98.0,49.0,80.0,90.0,UserA
0.0,101.0,0.0,89.0,29.0,69.0,33.0,69.0,129.0,71.0,42.0,82.0,23.0,77.0,49.0,88.0,164.0,100.0,250.0,54.0,121.0,52.0,121.0,74.0,UserA
0.0,99.0,0.0,107.0,53.0,86.0,40.0,66.0,125.0,74.0,42.0,99.0,3.0,113.0,53.0,96.0,160.0,94.0,241.0,64.0,113.0,48.0,89.0,81.0,UserA
0.0,103.0,0.0,96.0,36.0,78.0,38.0,66.0,137.0,70.0,18.0,99.0,112.0,107.0,50.0,77.0,147.0,102.0,245.0,66.0,130.0,56.0,118.0,75.0,UserA
0.0,87.0,0.0,115.0,56.0,89.0,44.0,72.0,140.0,76.0,59.0,71.0,99.0,133.0,42.0,88.0,155.0,93.0,7.0,66.0,112.0,49.0,94.0,81.0,UserA
0.0,99.0,0.0,99.0,44.0,90.0,40.0,73.0,142.0,84.0,48.0,79.0,97.0,111.0,40.0,66.0,124.0,100.0,203.0,66.0,116.0,53.0,114.0,73.0,UserA
0.0,113.0,0.0,98.0,30.0,78.0,20.0,62.0,133.0,65.0,72.0,58.0,105.0,110.0,33.0,86.0,128.0,86.0,6.0,71.0,120.0,51.0,103.0,69.0,UserA
0.0,111.0,0.0,96.0,2.0,88.0,48.0,77.0,123.0,78.0,110.0,91.0,93.0,114.0,147.0,110.0,154.0,95.0,238.0,62.0,104.0,47.0,99.0,62.0,UserA
0.0,101.0,0.0,96.0,15.0,83.0,54.0,76.0,126.0,69.0,282.0,110.0,15.0,105.0,74.0,78.0,137.0,98.0,224.0,78.0,95.0,49.0,97.0,59.0,UserA
0.0,105.0,0.0,97.0,22.0,89.0,49.0,68.0,128.0,70.0,43.0,91.0,127.0,82.0,31.0,79.0,125.0,97.0,178.0,61.0,114.0,50.0,103.0,65.0,UserA
0.0,112.0,0.0,107.0,21.0,78.0,29.0,61.0,128.0,63.0,43.0,82.0,117.0,94.0,39.0,61.0,126.0,96.0,188.0,51.0,108.0,46.0,107.0,56.0,UserA
0.0,101.0,0.0,84.0,49.0,86.0,35.0,62.0,119.0,66.0,57.0,71.0,95.0,114.0,39.0,51.0,118.0,88.0,180.0,55.0,89.0,56.0,100.0,46.0,UserA
0.0,112.0,0.0,95.0,50.0,81.0,37.0,68.0,117.0,70.0,35.0,65.0,4.0,85.0,57.0,65.0,132.0,94.0,208.0,79.0,94.0,60.0,87.0,63.0,UserA
0.0,86.0,0.0,90.0,16.0,58.0,21.0,46.0,149.0,66.0,79.0,97.0,163.0,101.0,58.0,72.0,131.0,66.0,11.0,57.0,116.0,42.0,101.0,90.0,UserA
0.0,114.0,0.0,76.0,33.0,71.0,23.0,41.0,126.0,66.0,20.0,92.0,95.0,94.0,66.0,62.0,121.0,58.0,162.0,56.0,108.0,45.0,100.0,73.0,UserA
0.0,89.0,0.0,79.0,33.0,69.0,12.0,45.0,119.0,72.0,41.0,75.0,69.0,101.0,41.0,83.0,137.0,69.0,16.0,55.0,127.0,55.0,109.0,81.0,UserA
0.0,92.0,0.0,85.0,42.0,64.0,20.0,54.0,122.0,62.0,79.0,42.0,90.0,98.0,30.0,101.0,182.0,68.0,245.0,62.0,118.0,47.0,101.0,78.0,UserA
0.0,116.0,0.0,96.0,12.0,77.0,41.0,53.0,113.0,69.0,68.0,44.0,79.0,84.0,9.0,88.0,149.0,58.0,134.0,76.0,103.0,46.0,90.0,68.0,UserA
0.0,84.0,0.0,81.0,9.0,55.0,35.0,41.0,120.0,74.0,13.0,67.0,3.0,88.0,68.0,75.0,112.0,62.0,139.0,57.0,112.0,39.0,92.0,71.0,UserA
0.0,101.0,0.0,84.0,13.0,62.0,21.0,54.0,116.0,71.0,27.0,71.0,80.0,72.0,42.0,57.0,121.0,69.0,161.0,58.0,133.0,50.0,102.0,68.0,UserA
0.0,90.0,0.0,75.0,35.0,69.0,20.0,49.0,118.0,68.0,64.0,75.0,93.0,95.0,21.0,50.0,120.0,77.0,160.0,58.0,107.0,53.0,95.0,67.0,UserA
0.0,88.0,0.0,95.0,4.0,63.0,47.0,52.0,112.0,70.0,50.0,65.0,75.0,98.0,24.0,71.0,123.0,71.0,148.0,57.0,116.0,47.0,97.0,75.0,UserA
0.0,109.0,0.0,57.0,27.0,68.0,69.0,25.0,134.0,68.0,67.0,68.0,105.0,114.0,49.0,70.0,125.0,75.0,148.0,55.0,125.0,59.0,111.0,69.0,UserA
0.0,105.0,0.0,95.0,4.0,73.0,35.0,57.0,114.0,74.0,62.0,57.0,72.0,118.0,39.0,76.0,149.0,109.0,243.0,69.0,119.0,50.0,102.0,75.0,UserA
0.0,130.0,0.0,92.0,15.0,84.0,57.0,49.0,127.0,78.0,77.0,99.0,136.0,102.0,24.0,82.0,129.0,88.0,186.0,57.0,121.0,47.0,97.0,68.0,UserA
```

Obrázek 15 - ukázka zdrojového CSV

Jak je vidět na obrázku 15, do zdrojového CSV aplikace vygeneruje sloupce, do kterých uloží popis k jednotlivým hodnotám. Zde se střídají charakteristiky flight time („FT0“, „FT1“, „FT2“,...) a dwell time („DT0“, „DT1“, „DT2“,...) pro jednotlivé znaky jdoucí za sebou. Na konec řádku je uložena textová nominální hodnota o uživateli, se kterou je dále pracováno při normalizaci. Aplikace pro sběr dat nerozlišuje, kdo ve skutečnosti heslo píše, pouze vygeneruje charakteristky hesel do CSV. Členění dat ke kterému uživateli vygenerovaný CSV soubor patří muselo být zajištěno manuálně. Tedy od uživatele A byly soubory CSV ukládány jinam, než CSV soubory uživatele B a C. Z těchto tří zdrojů již nadále čerpá aplikace, která soubory vstupních dat normalizuje. Při generování CSV byla z důvodu dalšího zpracování CSV použita jako desetinný oddělovač tečka a pro oddělovač textu byla použita čárka.

Data byla od uživatelů sbírána průběžně po dávkách s odstupem několika dní. Časový odstup zajistí, že data od jednoho uživatele jsou rozmanitá a tedy algoritmus strojového učení bude mít vzorky, které více odpovídají realitě. Pokud by data od jednoho uživatele byla získána ve velkém objemu najednou, nebyla by tak realistická. Mozek si během neustále opakovaného psaní hesla navykne stejného postupu a uživatel si tak vytvoří

takový styl psaní, že hodnoty dwell time a flight time se budou neustále zmenšovat. Tento styl uživatel s odstupem času jen stěží zopakuje. Je tedy nutné data sbírat dlouhodobě.

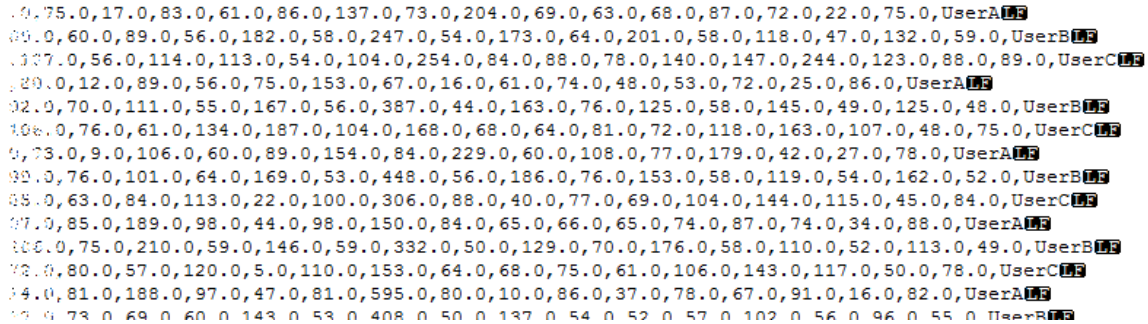
Například neuronová síť na různorodých datech, která jsou získána v rámci dlouhodobého sběru od jednoho uživatele, si díky algoritmu backpropagation najde v datech takové znaky (features), podle kterých bude schopna uživatele rozpoznat i přes to, že jsou data rozmanitá. Nalezení správných znaků je klíčové pro spolehlivost rozpoznávání. I přesto, že jsou v datech od jednoho uživatele drobné rozdíly vzniklé tím, že jde o několik sběrů po určité době, existují jisté znaky, které tyto vzorky společně vykazují právě proto, že jsou od jednoho uživatele. Pokud neuronová síť tyto znaky najde, je spolehlivější, než kdyby nalezla korelaci mezi daty sebranými najednou, které se od sebe jen velmi málo liší, a uživatel s odstupem času nebude schopen se k těmto datům přiblížit. Takový uživatel by se pak mohl pro neuronovou síť jevit jako zcela nový.

Pro implementaci neuronové sítě, kterou podrobně popisuje kapitola 2.3, bylo použito frameworku Encog [31]. Tento framework umožňuje snadný návrh neuronové sítě, její trénink, testování, uložení i následné nasazení na reálná data. Další výhodou frameworku také je, že obsahuje vestavěné funkce pro normalizaci vstupních dat. Před tím, než jsou získaná data použita v jakékoliv ze tří metod strojového učení, je nutno je nejprve normalizovat. K této normalizaci bylo použito právě frameworku Encog a jeho vestavěných funkcí. Následující kapitola popisuje normalizaci vstupních dat pomocí frameworku Encog.

2.2 Normalizace dat

Prvním krokem je vytvoření aplikace využívající framework Encog za účelem normalizace získaných dat pro jednotlivé metody strojového učení. Framework je navržen pro programovací jazyk Java a C#. Pro normalizační aplikaci byl vytvořen Maven projekt v jazyce Java s vloženou závislostí pro framework Encog.

Jak již bylo řečeno, existují celkem tři CSV soubory se zdrojovými daty, pro každého uživatele jeden zdrojový soubor. Ukázku takového CSV souboru je možno vidět na obrázku 15. Data z těchto tří zdrojových souborů jsou normalizační aplikací smíchána do jednoho CSV souboru. Hesla jsou ukládána po jednom od každého uživatele v postupném pořadí: uživatel A, uživatel B, uživatel C, uživatel A, uživatel B, uživatel C, atd. Každý záznam, tj. heslo je na konci opatřeno nominálním identifikátorem uživatele, kterému patří, tedy „UserA“, „UserB“, nebo „UserC“. Část výsledného souboru je možno vidět na obrázku 16).



Obrázek 16 - Ukázka části CSV smíchaných dat

Tyto identifikátory jsou hodnoty, které bude později predikovat použitá metoda strojového učení. Jedná se o výstupní hodnoty, které slouží jako vzor pro naučení a poté k testování dané metody.

Nyní je tedy pouze jeden zdrojový CSV soubor se všemi daty od všech uživatelů. Dalším krokem je samotná normalizace. K tomuto kroku je již použit Encog, resp. jeho vestavěné nástroje pro analýzu a normalizaci CSV.

Data jsou normalizována po sloupcích, tedy první sloupec tvoří první skupinu k normalizaci, druhý sloupec druhou, atd. Je tedy třeba nejprve projít celý vstupní soubor a pro každý sloupec určit maximální a minimální hodnotu, aby bylo poté možno hodnoty v tomto sloupci normalizovat.

Na obrázku 17 je znázorněn výstup z analýzy vstupního souboru. Jak je na tomto obrázku vidět, poslední zjištěný sloupec s názvem „user“ (tento název byl vyčten ze záhlaví CSV souboru) je normalizován pomocí rovnostranné normalizace (action=Equilateral). To je z toho důvodu, že normalizační metoda frameworku Encog zjistila, že tento sloupec obsahuje nominální hodnoty („UserA“, „UserB“, „UserC“) a sama určila způsob normalizace. Ostatní předešlé sloupce jsou normalizovány klasickým způsobem jako numerické hodnoty.

```
Fields found in file:
ft0,action=Normalize,min=-1.0E-4,max=1.0E-4
dt0,action=Normalize,min=47.0,max=130.0
ft1,action=Normalize,min=0.0,max=199.808
dt1,action=Normalize,min=39.946,max=115.0
ft2,action=Normalize,min=2.0,max=359.0
dt2,action=Normalize,min=49.0,max=90.0
ft3,action=Normalize,min=12.0,max=170.222
dt3,action=Normalize,min=25.0,max=79.0
ft4,action=Normalize,min=112.0,max=411.388
dt4,action=Normalize,min=46.5,max=93.0
ft5,action=Normalize,min=13.0,max=282.0
dt5,action=Normalize,min=42.0,max=110.0
ft6,action=Normalize,min=3.0,max=192.174
dt6,action=Normalize,min=46.0,max=133.0
ft7,action=Normalize,min=9.0,max=219.0
dt7,action=Normalize,min=36.771,max=110.0
ft8,action=Normalize,min=112.0,max=453.689
dt8,action=Normalize,min=32.0,max=109.0
ft9,action=Normalize,min=6.0,max=417.0
dt9,action=Normalize,min=44.765,max=97.0
ft10,action=Normalize,min=76.0,max=297.539
dt10,action=Normalize,min=35.112,max=88.0
ft11,action=Normalize,min=54.0,max=177.0
dt11,action=Normalize,min=44.0,max=99.978
user,action=Equilateral,min=0.0,max=0.0
```

Obrázek 17 - ukázka výstupu analýzy

Data je možno normalizovat na interval $\langle -1; 1\rangle$ nebo $\langle 0; 1\rangle$ dle zvoleného druhu aktivační funkce neuronů v neuronové síti. Zároveň však použitý normalizační interval nemusí být pro vstupní hodnoty stejný jako pro hodnoty výstupní. Vše záleží na použité aktivační funkci v neuronech neuronové sítě. Jak již bylo řečeno, aktivační funkce pracují jak s kladnými, tak i se zápornými hodnotami. Vstupní hodnoty jsou vždy normalizovány na interval $\langle -1; 1\rangle$ kvůli lepším konvergencím pomocí gradientních metod během učení. Na druhou stranu však některé z aktivačních funkcí vrací výstupní hodnoty pouze v intervalu $\langle 0; 1\rangle$, např. funkce Sigmoid. V případě funkce Sigmoid jsou tedy vstupní data normalizována na interval $\langle -1; 1\rangle$ zatímco výstupní data jsou normalizována na interval $\langle 0; 1\rangle$. Stejně tak při použití rektifikovaného lineárního neuronu. Naopak pří použití

aktivační funkce hyperbolického tangentu, nebo logaritmické aktivační funkce jsou díky jejich vlastnostem všechna data, tedy vstupní i výstupní, normalizována na interval $\langle -1; 1 \rangle$.

V této fázi jsou data normalizována a uložena ve výstupním souboru a je také k dispozici statistika o vstupních datech (zejména minimální a maximální hodnoty jedotlivých sloupců). Tyto statistiky je třeba také uchovat. Důvodem je, že poté, co budou metody strojového učení připraveny k použití, budou jim předkládána nová data, resp. nové případy napsaných hesel, pro které budou predikovat výstup, resp. uživatele, který je napsal. Tato nová data je třeba také normalizovat. K jejich normalizaci je třeba nastavit stejné podmínky, jako pro data tréninková. Budou tedy použity statistiky z trénovacích dat. Minimální a maximální hodnoty jednotlivých sloupců definující rozsah časových intervalů pro každé písmeno hesla určí správné hodnoty nového pokusu po normalizaci. Při této normalizaci může dojít k tomu, že tato nová data budou mimo daný rozsah $\langle -1; 1 \rangle$. Toto však není na škodu, neboť se jedná o autentizaci a autentizovaná osoba by se měla co nejvěrohodněji prokázat. Její vzorky dat by proto měly být co nejpodobnější těm tréninkovým. Nová data jsou tak normalizována dle stejných podmínek jako data trénovací.

```
","dt10","ft11","dt11","user(p0)","user(p1)"CRLF
,0.479,0.597,0.099,0.262,0.211,0.821,0.933,0.25CRLF
0.317,0.247,0.449,0.130,0.345,0.669,1,0.066,0.25CRLF
.178,0.415,0.277,0.693,0.275,0.829,0.715,0.446,0.5,1CRLF
```

Obrázek 18 - Ukázka části normalizovaného CSV

Na obrázku 18 je vidět část CSV, které obsahuje normalizované hodnoty. Konkrétně na tomto příkladu jsou vstupní i výstupní hodnoty v intervalu $\langle 0; 1 \rangle$. Poslední dva sloupce vzniklé rovnostrannou normalizací symbolizují výstupní hodnoty, tedy uživatele. Výsledkem rovnostranné normalizace je počet prvků (v případě neuronové sítě je to počet neuronů), který je o jeden menší, než počet uživatelů. Protože jsou uživatelé tři, jsou v CSV vidět dvě výstupní hodnoty *user(p0)* a *user(p1)*.

2.3 Implementace neuronové sítě

Jako první je z použitých metod strojového učení popsána metoda neuronové sítě k jejíž implementaci byl využit framework Encog. Ten načte data z normalizovaného CSV a přímo je použije k naučení sítě. Ve frameworku je možno síť navrhnout vlastním způsobem tak, že je možno zvolit:

- počet vrstev sítě,
- počet neuronů v každé vrstvě sítě,
- aktivační funkci neuronů v každé vrstvě sítě (je možno volit z: Sigmoid neuron, binární prahový neuron, lineární neuron, neuron hyperbolického tangentu, logaritmický neuron a rektifikovaný lineární neuron – framework tento poslední typ neuronu standardně neobsahuje, byl doprogramován v rámci této diplomové práce),
- zda má pro danou vrstvu sítě existovat konstantní prahová hodnota bias.

Počet vrstev a počet neuronů v jednotlivých vrstvách sítě je klíčovým faktorem pro sestavení nejvhodnějšího modelu vykazující co nejlepší výsledky a co nejmenší chybu. Určením podoby jednotlivých vrstev sítě se zabývají následující kapitoly.

2.3.1 Vstupní vrstva

Počet neuronů vstupní vrstvy je roven počtu hodnot vstupního vektoru do neuronové sítě. Každý vstupní neuron přijímá jednu hodnotu. Pro heslo „optimalizace“, které má celkem 12 písmen, tedy dohromady 24 hodnot dwell time a flight time, je třeba neuronová síť s 24 neurony ve vstupní vrstvě. Pro heslo „velikonocnizajic“, které má 16 písmen a dohromady 32 hodnot, je třeba neuronová síť s 32 vstupními neurony.

2.3.2 Výstupní vrstva

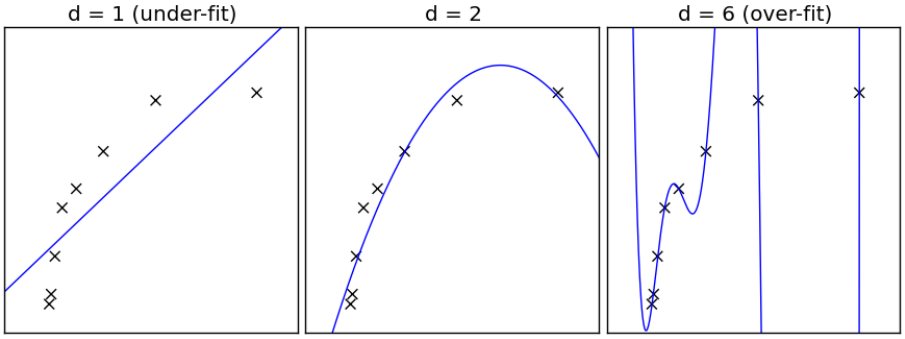
Počet výstupních neuronů je u obou hesel stejný – tím, že jsou pro účely této diplomové práce použiti tři uživatelé, se kterými byl prováděn sběr dat a testování, je díky rovnostranné normalizaci výstup generován do dvou výstupních neuronů. Pokud by byl počet uživatelů vyšší, bude díky rovnostranné normalizaci počet výstupních neuronů roven číslu o 1 menší, než počet uživatelů.

2.3.3 Skryté vrstvy

Obecně, velké neuronové sítě potřebují veliké množství času, aby se potřebné charakteristiky naučily, zatímco u malých sítí hrozí riziko uvíznutí v lokálních minimech chybové funkce a tím pádem se z tréninkových dat nenaučí. Existuje mnoho literatury popisující otázku vhodného návrhu struktury neuronových sítí [30, 32, 33, 34]. Navržená architektura sítě vždy sleduje, zda funkční model sítě není příliš jednoduchý, podhodnocený (tzv. underfitting), nebo zda již není příliš komplikovaný (tzv. overfitting).

Podhodnocená, nedostatečná funkční složitost sítě, tzv. underfitting, se projevuje tak, že síť není schopna se na trénovacích datech naučit a dosáhnout uspokojivě nízké chyby. Model není schopen zachytit vztahy mezi vstupními a výstupními daty.

Naopak stav přetrénování, tzv. overfitting, vzniká ve chvíli, kdy síť je již tak komplexní, že na trénovacích datech si vede velice dobře a generuje velice malou chybu, ale na nových datech vykazuje naopak chybu velkou a není schopna nová data správně rozpoznat. Síť v takovém případě ztrácí schopnost generalizace [35].



Obrázek 19 - Příklad nedostatečného modelu sítě a přetrénování sítě [35]

Příklad stavů podhodnocení, přetrénování a stavu optimálního je vidět na obrázku 19. Ten ukazuje různé příklady tří funkcí s různými stupni volnosti d. Pro d = 1 je funkční model

podhodnocen. Lineární funkce není vhodným modelem pro naučení na daných datech. Tento podhodnocený model má tzv. vysoký bias[1]. Druhým extrémním příkladem je naopak případ kdy d = 6. Model je zde přetrénován. Má příliš mnoho stupňů volnosti, které mohou být přizpůsobeny tak, že model bude perfektně odpovídat trénovacím datům. Pokud však bude do vstupních dat přidán nový bod, šance, že se tento bod přiblíží přetrénovanému funkčnímu předpisu, je velice malá. Tento přetrénovaný model má velký rozptyl.

Uprostřed na obrázku 19 je uveden příklad pro d = 2. Tento model má dva stupně volnosti a předepsaná křivka nejlépe odpovídá datům v grafu. Model netrpí ani velkým rozptylem, ani vysokou hodnotou bias. Při určování nejvhodnější struktury neuronové sítě je snaha optimalizovat obě hodnoty bias a rozptylu. Toto u neuronové sítě ovlivňuje právě počet skrytých vrstev a počet neuronů v nich a jejich počty je nutno nalézt experimentálně během implementace.

Dle [30] se teorie v této problematice různí. Některé např. doporučují, že počet neuronů ve skrytých vrstvách, by měl být roven $2/3$ počtu neuronů ve vstupní vrstvě. Jiný zase, že by tento počet měl být v rozsahu 70% až 90% počtu neuronů ve vstupní vrstvě. Existují ale také zdroje doporučující, že počet neuronů ve skrytých vrstvách by měl být menší než dvojnásobek počtu neuronů ve vstupní vrstvě, apod. Doporučení skrze zdroje se tedy výrazně liší. Avšak dle závěru z [30] toto není vhodný způsob určení struktury neuronové sítě. Je velice složité určit vhodnou topologii sítě pouze z počtu vstupních neuronů. Vhodná struktura sítě by měla být závislá také na množství trénovacích dat a komplexnosti celkové klasifikační problematiky daného problému. Mohou existovat extrémní situace, kdy je k dispozici jeden vstup a síť vyžaduje miliony skrytých neuronů, nebo situace, kdy jsou k dispozici miliony vstupních a výstupních případů, které pro dostatečné naučení sítě vyžadují pouze jeden skrytý neuron.

Ze závěru [30] vyplývá, že co do počtu skrytých vrstev jedna, maximálně dvě skryté vrstvy jsou dostatečné pro uspokojivé řešení většiny nelineárních komplexních problémů, přičemž pokud má síť dvě skryté vrstvy, měly by se počty neuronů v těchto dvou vrstvách držet alespoň přibližně na stejné úrovni, aby se síť dala snadno natrénovat. Celkový počet neuronů ve skrytých vrstvách je však nutno nalézt experimentálně, neboť závisí i na samotných datech, nebo typu zvolené aktivační funkce (typu neuronu).

2.3.3.1 Experimentální určení nejvhodnější neuronové sítě

Experimenty byly prováděny v kombinacích s následujícími vlastnostmi:

- Typ neuronu (aktivační funkce): Sigmoid, hyperbolický tangent, rektifikovaný lineární neuron, logaritmický neuron.
- Bias: zapnutý, vypnutý (je možno zvolit, zda u každé vrstvy neuronové sítě bude přítomná bias hodnota).
- Počet skrytých vrstev: 1 nebo 2
- Celkový počet neuronů ve skrytých vrstvách: 10 až 40

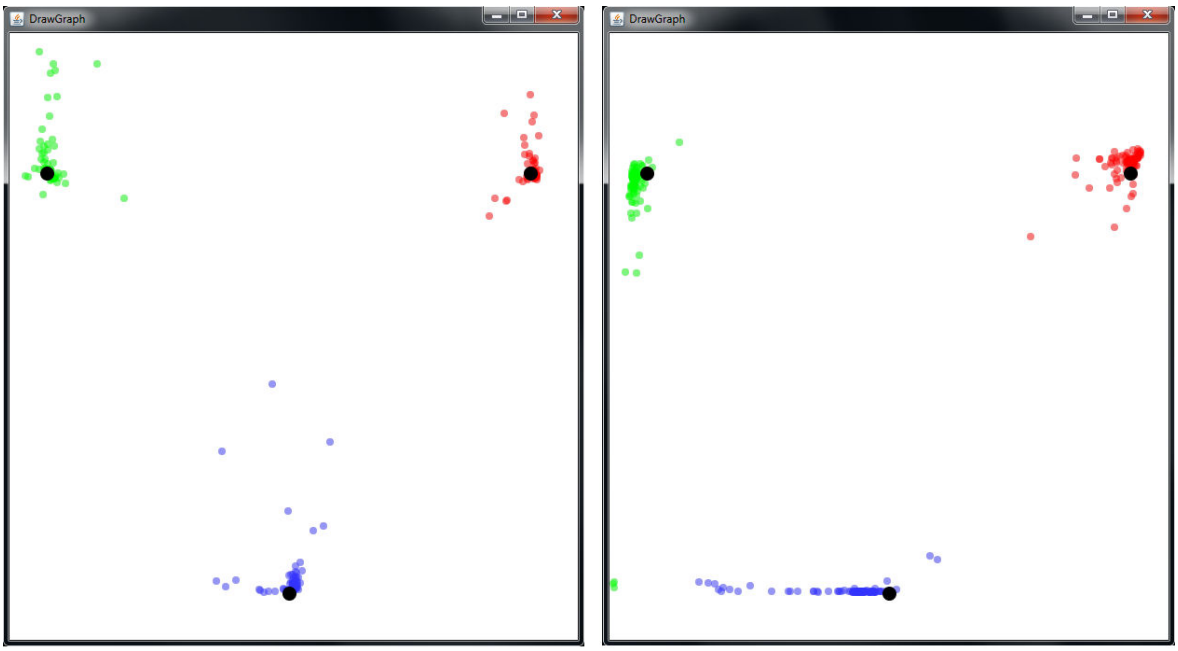
[1] Bias ve smyslu podhodnocené funkce je systematická chyba ve sběru, analýze a interpretaci dat, která vede ke zkreslení konečných výsledků. Nejedná se o stejný bias, který se používá u aktivační funkce neuronů v neuronové síti pro posun aktivační funkce (viz kapitola 1.3.2)

Pro obě hesla byly vytvořeny dva soubory vzorkových dat od 3 uživatelů, učící soubor a testovací soubor. Každý obsahoval cca 100 vzorků od každého uživatele (celkem cca 300 hesel v učícím i testovacím souboru). Síť byla učena pomocí učícího souboru až do dosažení chybovosti 0,05%. Po naučení sítě byl zkoumán dosažený výsledek na testovacím souboru hesel. Pro testování za účelem co nejvhodnější interpretace výsledků byla vytvořena aplikace, která hodnoty predikované pro testovací soubor zobrazuje formou grafu. Díky rovnostranné normalizaci pro tři uživatele jsou výstupem neuronové sítě dvě hodnoty. Tyto hodnoty lze zobrazit v souřadnicovém systému. Vzorky od jednotlivých uživatelů jsou barevně rozlišeny a lze tedy sledovat jejich grafický rozptyl. Každá konfigurace sítě byla vyzkoušena několikrát a vybráno bylo vždy takové zobrazení, které se pravidelně opakovalo. Učení a testování probíhalo pro každé heslo zvlášť.

Heslo „optimalizace“:

Síť s aktivační funkcí hyperbolického tangentu (dále TanH) vykazovala napříč všemi kombinacemi konfigurace (různý počet neuronů a skrytých vrstev, přítomnost hodnoty bias) schopnost rychlejšího učení, než funkce Sigmoid (dále Sig). Počet nutných iterací k dosažení uspokojivé chyby na testovacím souboru dat, která byla stanovena na 0,05%, byl u neuronů s funkcí Sig řádově o několik tisíc vyšší, než u neuronů s funkcí TanH). Po otestování několika konfigurací sítě s těmito aktivačními funkcemi se jako nejlepší jevila síť v konfiguraci:

- Skryté vrstvy: 2 po 10 neuronech,
- Bias: zapnutý



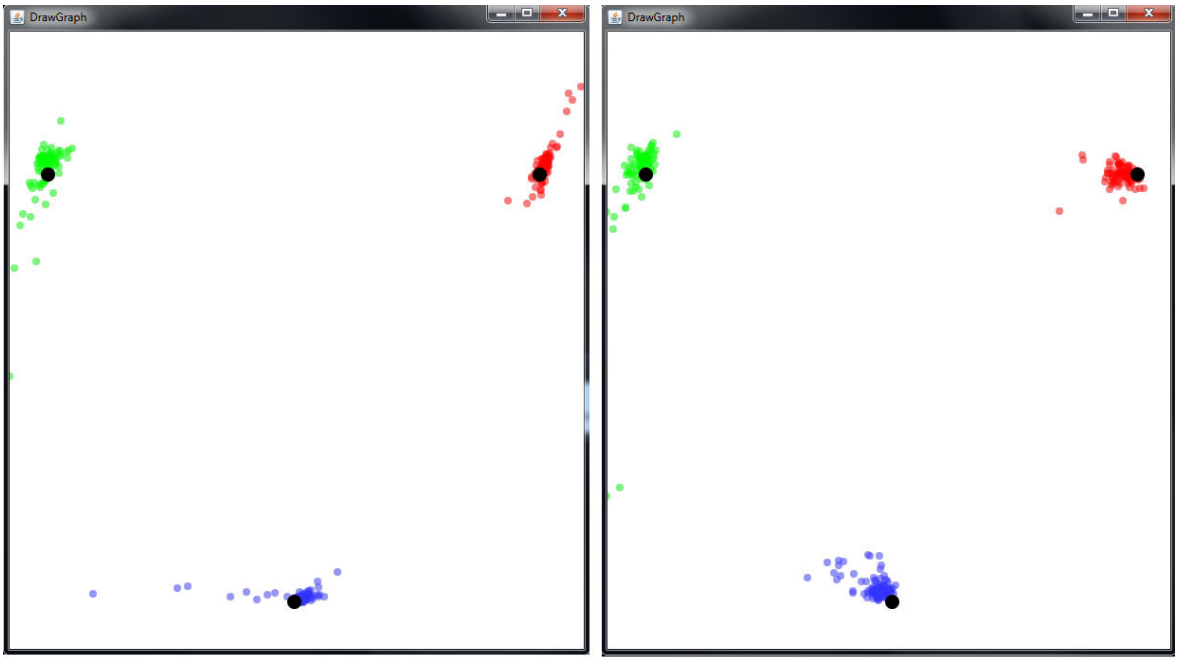
TanH, 2 x 10, iter = 1 324 Sig, 2 x 10, iter = 7 108

Obrázek 20 - Grafické vyjádření predikovaných výsledků aktivačních funkcí TanH a Sigmoid

Na obrázku 20 je naznačen grafický výsledek sítě ve výše uvedené konfiguraci při použití obou aktivačních funkcí. Síť s aktivační funkcí TanH (vlevo) se naučila predikovat výsledky s chybou 0,05% za 1 324 iterací, síť s aktivační funkcí Sig potřebovala pro stejný

výsledek 7 108 iterací. Také výsledné rozložení predikovaných výsledků má nižší entropii[2] u sítě TanH. U sítě Sig většinou vzniká u jednoho z uživatelů velký rozptyl na jednom neuronu (viz modré vzorky na grafu vpravo, kde souřadnice y je predikována poměrně přesně, avšak souřadnice x má velké rozpětí).

Síť s logaritmickou aktivační funkcí (dále Log) vykazovala nejlepší chování v ohledu grafické reprezentace rozptylu (vzorky jsou nejblíže své ideální hodnotě). Naopak se tato síť učí pomaleji, než sítě s aktivačními funkcemi TanH a Sig. Na obrázku 21 je vidět grafické srovnání dvou různých konfigurací Log neuronové sítě. Vlevo je konfigurace jedné skryté vrstvy o 5 neuronech, vpravo je výše uvedená konfigurace 2 skrytých vrstev, každá po 10 neuronech. Grafické znázornění je velice podobné, avšak síť vlevo pro své naučení s chybovostí 0,05% potřebovala cca 120 tisíc iterací, naproti tomu síť vpravo cca 10 tis.



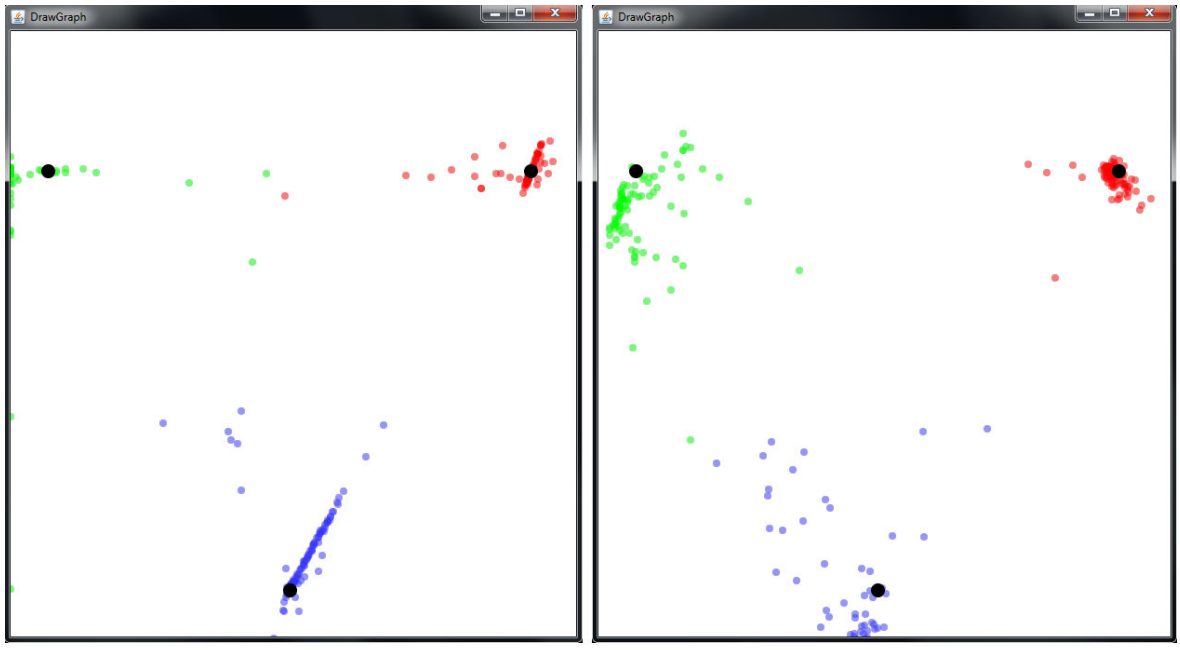
Obrázek 21 - Grafické znázornění rozptylu logaritmické aktivační funkce

Pomalejší schopnost učení může být přikládána hladšímu průběhu aktivační funkce u počátku souřadnicového systému (viz kapitola 1.3.2.7) ve srovnání s funkcemi Sig (1.3.2.4) a TanH (1.3.2.6), kde je funkční průběh v počátku souřadnicového systému strmější. Naopak na koncích funkce u hodnot x blížících se -1 a 1 má strmější průběh (a tím i vyšší hodnotu první derivace) Log aktivační funkce a i zde se síť učí pomocí gradientních metod lépe, než síť s funkcemi Sig a TanH, kde průběh je na koncích téměř plochý, tedy první derivace funkce v těchto částech je téměř nulová a dochází tak v procesu učení k minimální změně. Pokud se hodnoty dostanou do těchto částí funkce, síť s neurony TanH a Sig se již učí velmi pomalu. Díky tomu je pravděpodobně možno

[2] Entropii lze chápat jako míru neuspořádanosti systému. V kontextu výše nejde o formální definici, protože pojem „neuspořádanost“ není v práci přesně definován, jde spíše o intuitivně snazší uchopení pojmu. Čím je entropie vyšší, tím více bude graf neuspořádán, bude se jevit jako chaotický a naopak. [36]

vysvětlit fakt, že u Log aktivační funkce jsou všechny testovací vzorky blíže své ideální hodnotě oproti funkcím Sig a TanH.

Jako poslední byla otestována síť s rektifikovanými lineárními neurony (dále ReLU). Tato síť ve srovnatelné konfiguraci s předchozími představenými konfiguracemi vykazovala nejpomalejší schopnost učit se (až několikanásobně pomalejší). Neboť součástí funkčního průběhu je lineární funkce (resp. buď je neuron neaktivní a vrací výstupní hodnotu 0 pro hodnoty $x < 0$, nebo je funkce lineární pro hodnoty $x > 0$), pro naučení složitějších, nelineárních příkladů je nutné použití vícevrstvých sítí. Na obrázku 22 je vidět srovnání dvou konfigurací ReLU sítí. Vlevo je znázorněn výstup sítě v konfiguraci dvou skrytých vrstev po 10 neuronech. Vpravo je konfigurace sítě s pěti skrytými vrstvami o počtu 20, 10, 10, 10, 8 neuronů. Na zobrazení vlevo je vidět lineární závislost výstupu na vstupu. Většina zeleně zbarvených hodnot je také mimo ideální hodnotu u kraje grafu. To značí, že složitost sítě není dostatečná k tomu, aby byla schopna mapovat nelineární příklad. Také proces učení celkem o 38 685 iteracích je dlouhý. Naproti tomu složitá síť vpravo je již schopna se poměrně rychle naučit (2 200 iterací), avšak mapování stále není tak přesné, jako u předchozích aktivačních funkcí Sig, TanH a Log. U některých testovacích případů se dokonce stává, že jsou predikovány téměř stejné hodnoty pro dva různé uživatele (dvě různě barevné tečky jsou v grafu blízko sebe). ReLU sítě tedy nejsou v tomto případě vhodným kandidátem.



Obrázek 22 - Grafické znázornění rozptylu dvou sítí s lineárními rektifikovanými neurony

Přítomnost hodnoty bias neměla skrze všechny vyzkoušené kombinace téměř žádný, nebo jen nepatrný vliv na rychlost učení. Ať už s hodnotou bias, nebo bez ní, vždy jinak stejně nakonfigurovaná síť byla naučena téměř stejně rychle jen s nepatrným rozdílem ± několika stovek iterací (v kontextu toho, že síť potřebuje až několik tisíc iterací se jedná o nepatrný rozdíl).

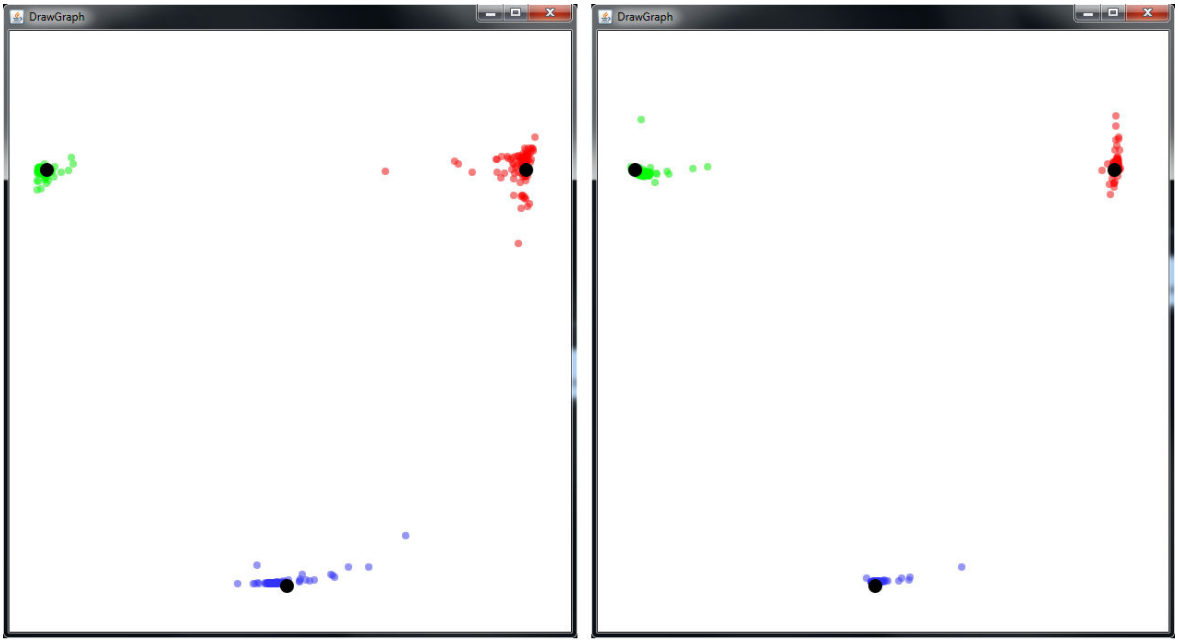
Pro heslo „optimalizace“ byla na základě výše uvedených testů jako optimální síť zvolena síť v následující konfiguraci:

- Aktivační funkce: Log,
- Struktura sítě: 2 skryté vrstvy po 10 neuronech,
- Bias: zapnutý,
- Počet iterací: 10 825,
- Chyba na učících datech: 0,05%,

Síť v této konfiguraci vykazovala na testovacím souboru dat pravidelně nejmenší chybu, pohybující se mezi 5 až 7%.

Heslo „velikonocnizajic“:

U druhého hesla, stejně tak jako u hesla „optimalizace“, aktivační funkce TanH vykazuje rychlejší schopnost učení než funkce Sig, nyní však až několikanásobně vyšší. Na obrázku 23 je znázorněn výstup sítě v konfiguraci jedné skryté vrstvy o 5 neuronech za použití aktivační funkce Sig (vlevo) a TanH (vpravo). Síť s funkcí TanH byla schopna se naučit s chybou 0,05% za 1 498 iterací. S funkcí Sig tento proces trval 5 509 iterací.



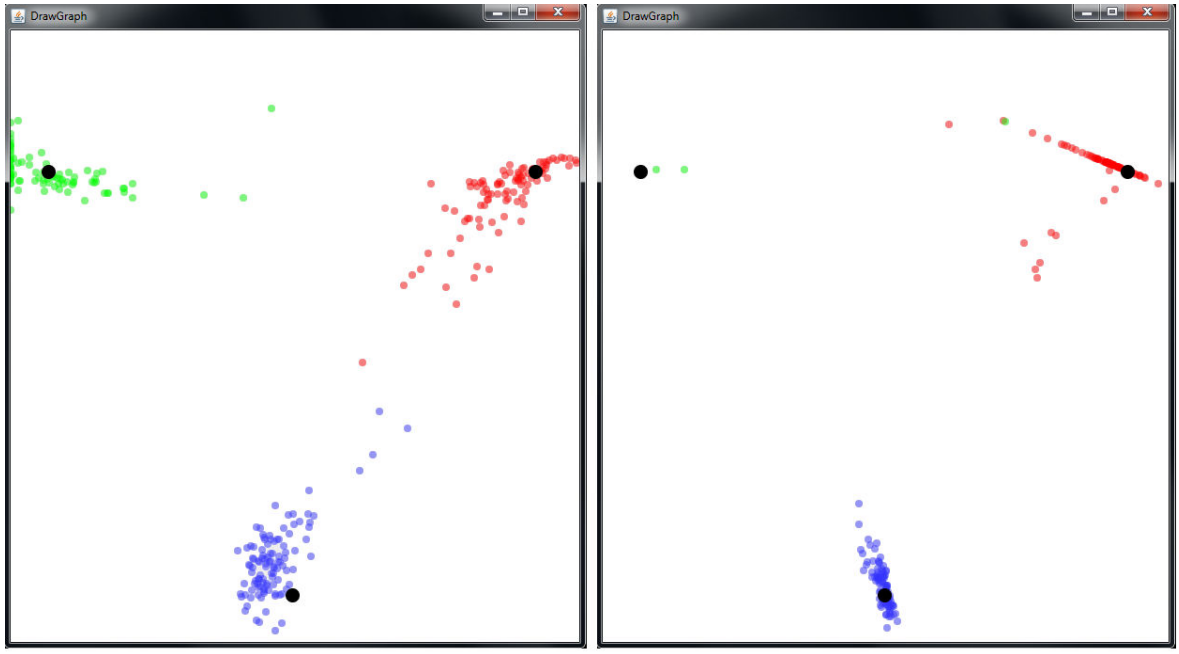
Sig, 1 x 5, iter = 5 509 TanH, 1 x 5, iter = 1 498

Obrázek 23 - Grafické znázornění rozptylu funkcí Sig a TanH

Přidávání počtu neuronů či skrytých vrstev u TanH a Sig funkcí nijak nezlepšovalo výstup a spíše navyšovalo počet nutných iterací (zejména u Sig). Bylo tedy bezpředmětné.

Pro srovnání byly otestovány také různé konfigurace sítě složené z neuronů ReLU. Při testování struktury o dvou skrytých vrstvách po 10 neuronech nebyla síť schopna docílit nižší chybovosti než 0,2%. Zobrazení predikovaných výsledků pro testovací soubor dat ani není zcela optimální (viz obrázek 24 vlevo). Složitější síť o pěti skrytých vrstvách v počtu 20, 10, 10, 8 neuronů již dosáhla chybovosti 0,05% avšak opět zobrazení predikovaných

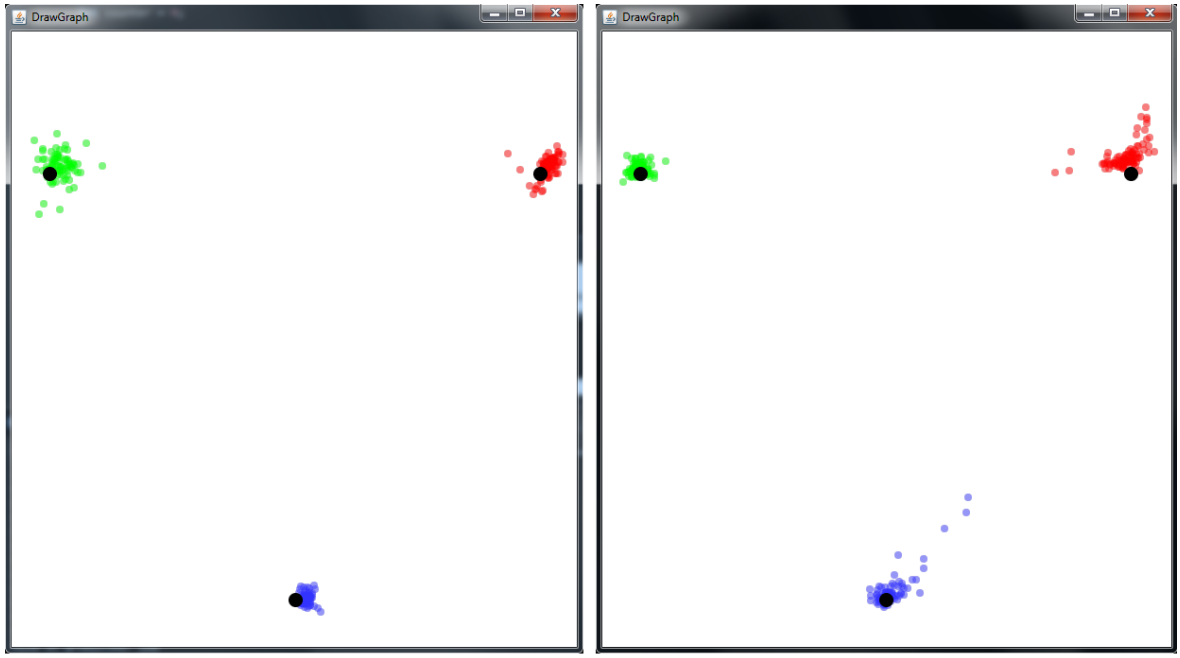
hodnot není přesné, jako u předchozích funkcí (viz obrázek 24 vpravo). Síť ReLU tedy ani v tomto případě není vhodným kandidátem.



ReLU, 2 x 10, iter = 1 086 ReLU, 20,10,10,8, iter = 2 779

Obrázek 24 - Grafické znázornění rozptylu sítí s ReLU neurony

Síť s Log aktivační funkcí vykazovala horší výsledek co do rychlosti učení. Oproti předchozím funkcím TanH a Sig, u funkce Log se počet nutných iterací zmenšoval spolu s tím, jak rostla složitost sítě. U funkcí TanH a Sig tomu bylo naopak.



Log, 1 x 5, iter = 53 021 Log, 2 x 10, iter = 7 185

Obrázek 25 - Grafické znázornění rozptylu sítí s Log neurony

S funkcí Log trvalo síti v konfiguraci jedné skryté vrstvy o 5 neuronech 53 021 iterací, než dosáhla požadované chybovosti 0,05%. Pokud však síť byla nakonfigurována na počet dvou skrytých vrstev o 10 neuronech, dokázala se naučit za 7 185 iterací (viz obrázek 25).

Za ideální konfiguraci sítě byla na základě výše uvedených experimentů pro heslo „velikonocnizajic“ zvolena následující konfigurace:

- Aktivační funkce: TanH,
- Struktura sítě: 1 skrytá vrstva o 5 neuronech,
- Bias: zapnutý,
- Počet iterací: 1 498,
- Chyba na učících datech: 0,05%,

Tato síť predikovala hodnoty na testovacím souboru dat s chybou pod 1%.

Jak je možno si povšimnout, u hesla „velikonocnizajic“ ve srovnání s předchozím heslem „optimalizace“ je k vytvoření sítě predikující uspokojivě přesné výsledky, které mají mnohem menší chybovost (pod 1%) než je chybovost zvolené sítě pro heslo „optimalizace“ (5 až 7%), zapotřebí mnohem méně neuronů. To je pravděpodobně dáno délkou samotného hesla. Tím, že heslo „velikonocnizajic“ obsahuje více znaků, má síť více vstupů a tím tedy více možností rozpoznat daného uživatele. V takovém vzorku si síť najde snáze rozlišovací znaky, pomocí kterých uživatele rozpozná s mnohem menším úsilím.

2.4 Implementace algoritmu k-nejbližších sousedů

Pro implementaci dvou následujících metod, tedy algoritmu k-nejbližších sousedů (dále k-NN) i Bayesovského klasifikátoru (dále BK) byl použit programovací jazyk Python. Konkrétně vývojové prostředí Python (x,y) [37] s vestavěným matematickým modulem scikit-learn [27]. Tento modul umožňuje jednoduchou implementaci obou klasifikačních metod, postačuje pouze poskytnout učící a testovací data.

Klasifikace je v případě algoritmu k-NN vypočtena pomocí jednoduchého většinového hlasování pro každý dotazovaný bod. Dotazovanému bodu je přiřazena třída (uživatel) která má největší počet zastoupení mezi nejbližšími body [38]. Nástroj Scikit Learn implementuje také metodu nalezení nejbližšího souseda uvnitř pevně daného poloměru. Tato metoda může být vhodnější tam, kde vzorová data pro jednotlivé třídy nejsou rovnoměrně rozmístěna a jsou „prořídlá“. Pro vícedimenzionální vstupní data, jako je v tomto případě, se však metoda může stát méně efektivní právě díky vysokému počtu dimenzí (tzv. „Curse of dimensionality“[3]). Proto je v tomto případě účelnější použití klasického algoritmu k-NN.

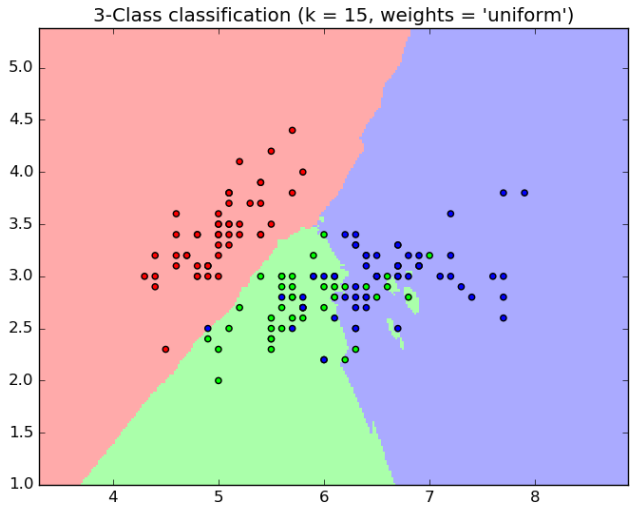
Optimální volba hodnoty k velmi záleží na vstupních datech. Obecně, vyšší k potlačuje efekt šumu, ale klasifikační hranice mezi třídami díky tomu není tak zřetelná. Pro účely této diplomové práce bylo voleno mezi jedním až pěti sousedy ($1 \leq k \leq 5$). Metoda k-NN

[3] Termín popisuje skutečnost, že použití velkého počtu parametrů (vysoké dimenze) může způsobit tzv. přetrénování a v konečném důsledku zhoršit následnou analýzu, resp. klasifikaci. Tento problém u klasického počtu dimenzí (dvou-, tří-dimenzionální prostory) nenastává. Velikost prostoru se se vzrůstajícím počtem dimenzí zvětšuje tak, že dostupná data se stávají „prořídlá“ [2, 39].

používá jednotné váhy pro nalezené sousedy. Je však také možno použít váhy rozdílné – bližší soused má vyšší váhu, než soused vzdálenější. Byly vyzkoušeny oba způsoby, přičemž jejich přepnutí nečinilo žádný rozdíl ve výstupních datech.

Znázornění výstupu metody k-NN je samo o sobě v tomto případě komplikované, protože vstupní data jsou vícedimenzionální. Ideální pro zobrazení je dvou, maximálně třídimenzionální prostor (viz zobrazení výstupu neuronové sítě v kapitole 2.3). Tato data bylo možno zobrazit, neboť na výstupu se nachází dva neurony, tedy dvě vypočtené hodnoty zobrazitelné do kartézských souřadnic. Algoritmus k-NN však žádné výstupní hodnoty neposkytuje, pouze přiřadí třídu (uživatele) vstupnímu vzorku hodnot. Vstupní vektory nelze v grafu zobrazit, neboť pro heslo optimalizace jde o 24 dimenzionální vektor. Pro heslo velikonocnizajic jde o 32 dimenzí.

Pro alespoň částečné naznačení rozmístění hodnot lze případně omezit vstupní vícedimenzionální vektory pouze na dvě hodnoty (dvě dimenze) a ty zobrazit, zbylé hodnoty ignorovat. Avšak v tomto případě není možné přesně vybrat ty správné dvě reprezentující hodnoty, na kterých bude nejvíce zřejmá identifikace uživatele. U vektorů vstupních hesel jde o drobné rozlišnosti mezi celkem 24 až 32 hodnotami. Neuronová síť pro nalezení těch správných rozlišujících charakteristik potřebuje až několik tisíc iterací. Tedy v tomto případě není toto zjednodušení, aby o něčem věrohodně vypovídalo, možné. Na obrázku 26 je znázorněna částečná reprezentace pro jednodušší případ. Je zde vidět grafická reprezentace rozmístění učících dat u klasifikace problému rozpoznání druhu kosatce na základě jejich rozměrů. V tomto konkrétním příkladu jsou vstupní data celkem 4 rozměry. Vybrány byly pouze 2 reprezentující rozměry a ty byly zobrazeny v grafu. Příslušný druh kosatce (třída) je pak barevně rozlišena v grafu.

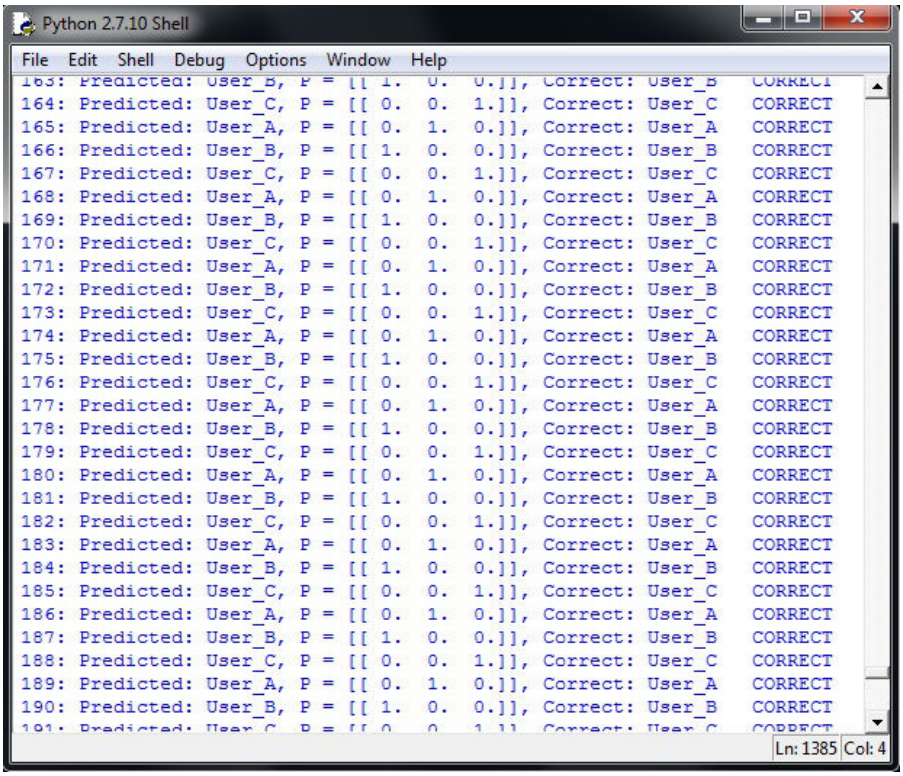


Obrázek 26 - Grafická reprezentace rozmístění dat u klasifikace pomocí algoritmu k-NN

Jak je vidět na obrázku 26, v modré části se nachází také zelené „ostrůvky“ patřící k jiné třídě. To je zřejmě proto, že 2 vybrané parametry dostatečně nepopisují rozdíl mezi oběma druhy kosatce a částečně zkreslily výsledek. To samé by se stalo při znázornění

trénovacích dat pro zkoumaná hesla, avšak zkreslení by bylo mnohem větší a jednotlivé třídy by od sebe nebyly zřetelně odlišitelné.

Protože výstup není graficky reprezentovatelný, je k dispozici pouze textový výstup v podobě výpisu predikovaných a správných hodnot. U algoritmu k-NN u predikování uživatelů pro obě hesla nezávisle na volbě počtu sousedů *k* nebyla zaznamenána žádná chyba, tedy na testovacích vzorcích algoritmus v obou případech nalezl odpovídající uživatele správně. Jedná se tedy o 100% přesnost. Výstup je znázorněn na obrázku 27.



Obrázek 27 - Výstup algoritmu k-NN v aplikaci Python(x,y)

Jak je vidět na výstupních datech, algoritmus vrací výstupní hodnoty s přesnou pravděpodobností *P*, tedy vždy jeden z uživatelů má pravděpodobnost 1, ostatní dva mají pravděpodobnost 0, viz např.:

```
1: Predicted: User_B, P = [[ 1.  0.  0.]], Correct: User_B   CORRECT
2: Predicted: User_C, P = [[ 0.  0.  1.]], Correct: User_C   CORRECT
3: Predicted: User_A, P = [[ 0.  1.  0.]], Correct: User_A   CORRECT
```

Tedy algoritmus si je vždy „jistý“ svým predikovaným výstupem.

2.5 Implementace Bayesovského klasifikátoru

Bayesovský klasifikátor je dle [27] možno implementovat třemi způsoby, vždy dle odpovídajícího rozhodovacího pravidla, které je použito.

2.5.1 Gaussův Bayesovský klasifikátor

Tento typ klasifikátoru je založen na Gaussově naivním Bayesovském algoritmu. Algoritmus v tomto případě předpokládá Gaussovo rozdělení pravděpodobnosti vzorků a je dán vztahem:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Parametry σ_y a μ_y jsou odhadnuty s využitím maximální pravděpodobnosti [27].

Gaussův Bayesovský klasifikátor v případě hesla optimalizace určil na testovacích datech uživatele chybně pouze ve 3 případech. V ostatních případech byl uživatel zvolen správně. Z celkového počtu 266 testovacích případů jde o chybovost $\frac{3}{266} * 100 = 1{,}13\%$.

V případě hesla velikonocnizajic pro testovací data určil chybně uživatele pouze ve dvou případech. Z celkového počtu 258 testovacích příkladů jde o chybovost $\frac{2}{258} * 100 = 0{,}78\%$. Ukázka výstupu Gaussova Bayesovského klasifikátoru:

```
0: Predicted: User_A, P = [[   1.56905951e-44     1.00000000e+00     8.15559702e-26]],
Correct: User_A   CORRECT
1: Predicted: User_B, P = [[   1.00000000e+00     8.72365158e-59     1.41109572e-39]],
Correct: User_B   CORRECT
2: Predicted: User_C, P = [[   9.04635969e-09     5.76921724e-10     9.99999990e-01]],
Correct: User_C   CORRECT
```

Na výstupech je vidět, že algoritmus také poskytuje informaci o pravděpodobnosti, přičemž predikovaný uživatel má pravděpodobnost rovnu 1 a pravděpodobnost zbylých uživatelů je téměř rovna 0, stejně jako u algoritmu k-NN.

2.5.2 Multinomický Bayesovský klasifikátor

Tento klasifikátor implementuje naivní Bayesovský algoritmus pro data, která mají multinomické rozdělení a je jedním ze dvou klasických naivních Bayesovských variant používaných v klasifikaci textu. Rozdělení je parametrizováno vektory $\Theta_y = (\Theta_{y1}, \dots, \Theta_{yn})$ pro každou třídu y, kde n je počet vstupních parametrů a Θ_{yi} je pravděpodobnost $P(x_i|y)$ parametru i, že se objeví v příkladu (vektoru) patřícím do třídy y.

Parametry Θ_y jsou vypočteny pomocí upravené funkce maximální pravděpodobnosti využívající relativní četnosti:

$$\widehat{\Theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

kde $N_{yi} = \sum_{x \in T} x_i$ je četnost parametru i v příkladu třídy y v trénovací množině dat T a $N_y = \sum_{i=1}^{|T|} N_{yi}$ je celkový počet parametrů pro třídu y [27].

Volba upravujícího (vyhlazovacího) parametru $\alpha \geq 0$ má význam pro parametry nevyskytující se v trénovacích datech a zabraňuje tomu, aby bylo s těmito hodnotami nulových pravděpodobností počítáno. Nastavení $\alpha = 1$ je nazýváno Laplaceovo vyhlazení a $\alpha < 1$ se nazývá Lidstonovo vyhlazení. Nastavení $\alpha = 0$ znamená, že data nejsou vyhlazena [27].

Klasifikátor nezávisle na zvolené hodnotě parametru α (tedy $\alpha < 1$, $\alpha = 1$ a $\alpha > 1$) predikoval uživatele vždy se stejnou přesností. V případě hesla optimalizace určil chybně uživatele pouze u 7 případů (chybovost $\frac{7}{266} * 100 = 2{,}63\%$). V případě hesla

velikonocnizajic určil uživatele chybně pouze ve 2 případech (chybovost $\frac{2}{258} * 100 = 0{,}78\%$).

Pravděpodobnosti v případě tohoto Bayesovského klasifikátoru jsou více rozprostřeny mezi jednotlivé uživatele, např.:

```
0: Predicted: User_A, P = [[ 0.08379632   0.58123613   0.33496755]], Correct: User_A
CORRECT
2: Predicted: User_C, P = [[ 0.33567735   0.25801299   0.40630966]], Correct: User_C
CORRECT
237: Predicted: User_C, P = [[ 0.05335822   0.46609264   0.48054913]], Correct: User_A
WRONG!!!
```

Tento klasifikátor si je méně „jistý“ svou volbou, než předchozí Bayesovský klasifikátor, či algoritmus k-NN. Vybraný uživatel má největší pravděpodobnost, avšak pravděpodobnost ostatních uživatelů je také vysoká (někdy v řádu desítek procent). Tento výstup se nejvíce podobá výstupu neuronové sítě.

2.5.3 Bernoulliho Bayesovský klasifikátor

Tento Bayesovský klasifikátor implementuje algoritmus pro data, která mají binomické rozdělení. Používá se například jsou-li parametry vstupního vektoru binární (hodnoty boolean).

Rozhodovací pravidlo pro Bernoulliho Bayesovský klasifikátor je založeno na vztahu:

$$P(x_i|y) = P(i|y)x_i + \big(1 - P(i|y)\big)(1 - x_i)$$

Tento vztah se odlišuje od vztahu multinomického naivního Bayesovského klasifikátoru v tom, že explicitně penalizuje ty parametry, které se pro danou třídu y neobjevily v trénovacích datech, zatímco multinomický klasifikátor by tyto parametry jednoduše ignoroval [27].

Bernoulliho Bayesovský klasifikátor vykazoval největší chybu. V případě hesla optimalizace byl uživatel predikován chybně celkem v 98 případech (chybovost $\frac{98}{266} * 100 = 36{,}84\%$).

V případě hesla velikonocnizajic byl uživatel predikován špatně ve 144 případech, což je chybovost $\frac{144}{258} * 100 = 55{,}81\%$. Pravděpodobnost je zde taktéž více rozprostřena mezi jednotlivé uživatele, jako v předchozím případě:

```
4: Predicted: User_C, P = [[ 0.28457757 0.34561286 0.36980957]], Correct: User_B
WRONG!!!
5: Predicted: User_C, P = [[ 0.28457757 0.34561286 0.36980957]], Correct: User_C
CORRECT
6: Predicted: User_C, P = [[ 0.28457757 0.34561286 0.36980957]], Correct: User_A
WRONG!!!
```

Tento typ Bayesovského klasifikátoru je tedy pro účely klasifikace uživatele na základě dynamiky psaní nepoužitelný z důvodu vysoké chybovosti.

2.5.4 Shrnutí Bayesovských klasifikátorů

Z výše uvedených výsledků všech tří Bayesovských klasifikátorů vyplývá následující srovnání chybovostí:

Heslo	Gaussův BK	Multinomický BK	Bernoulliho BK
optimalizace	1,13%	2,63%	36,84%
velikonocnizajic	0,78%	0,78%	55,81%

Tabulka 1 - Srovnání Bayesovských klasifikátorů

Z tohoto srovnání se jako nejlepší Bayesovský klasifikátor jeví Gaussův, jehož výstup v podobě pravděpodobností je téměř stejný, jako u algoritmu k-NN (predikovaný uživatel má pravděpodobnost 1, ostatní mají pravděpodobnost téměř nulovou), případně klasifikátor multinomický, který se svým výstupem více blíží neuronové síti.

2.6 Ověření algoritmů na neznámém uživateli

Doposud se předchozí kapitoly zabývaly autentizací uživatele na základě predikce jedné ze tří možných variant. Tedy bylo jisté, že se jedná o jednoho ze zkoumaných uživatelů, pro kterého mají algoritmy k dispozici trénovací data. Predikci daného uživatele ze tří možných variant vybrané metody zvládly poměrně dobře vždy s chybovostí v řádu maximálně jednotek procent. Tato práce však zkoumá autentizaci, kde je nutno předpokládat, že bude docházet také k pokusům o tzv. neautentizované přístupy, tedy bude se pokoušet přihlašovat uživatel, který nesmí být autentizován. Algoritmy je tedy nutno otestovat také na tuto možnost. Toto otestování je provedeno tak, že „naučeným" algoritmům strojového učení jsou předloženy testovací případy (tedy vzorky napsaných hesel) od neznámých uživatelů, přičemž jsou pozorovány predikované výsledky. Za účelem tohoto testování byly získány vzorky hesel od 10 dalších uživatelů. Testování probíhalo za použití celkem cca 450 vzorků pro každé heslo.

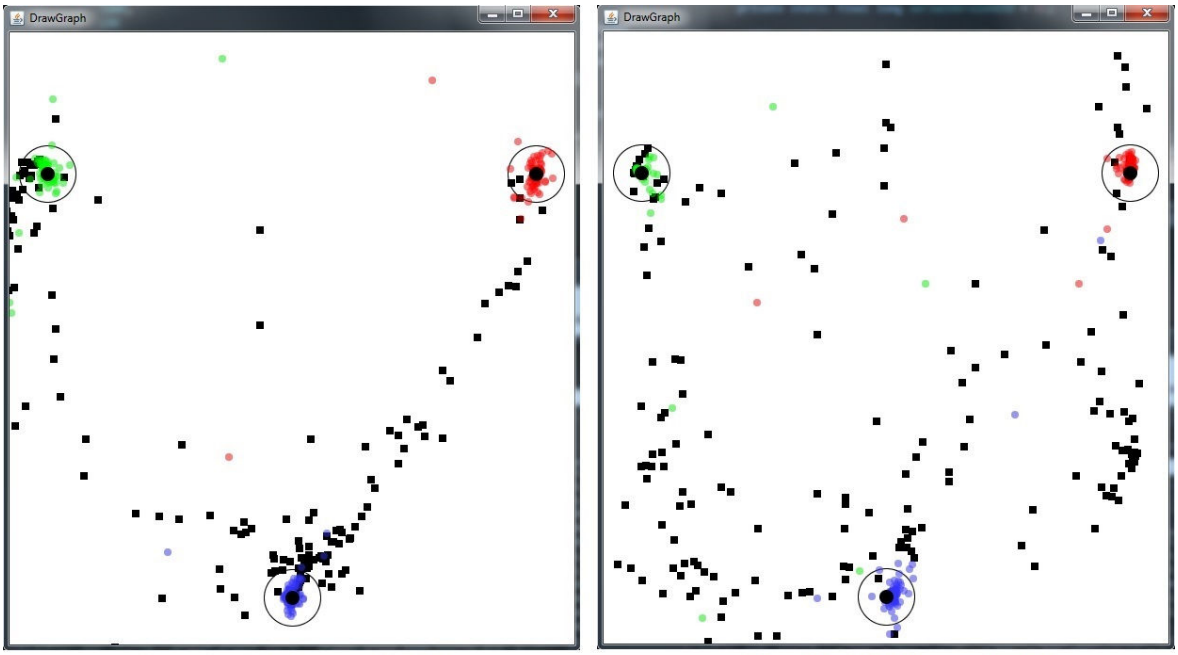
2.6.1 Ověření na neuronové síti

Neuronové sítě produkují výstup v podobě normalizovaného vícesložkového vektoru dle počtu uživatelů. Hodnoty tohoto vektoru lze brát jako souřadnice v prostoru. Čím více se predikovaný bod v prostoru blíží své ideální hodnotě, tím větší je pravděpodobnost, že je predikován správně. V případě této práce jsou to 3 uživatelé, pro které jsou výsledky zobrazovány ve dvojrozměrném prostoru. Do tohoto prostoru je možno promítnout také výsledky (resp. souřadnice) predikované pro data neznámých uživatelů.

Při ověřování práce neuronových sítí pro příklady hesel neznámých uživatelů (dále neautorizovaná hesla) prokázala svou významnost celková složitost architektury sítě. Z předchozího zkoumání lze říci, že jednodušší architektury byly úspěšnější (kromě neuronu ReLU) co do klasifikace známých uživatelů. Avšak poté, co jim byla předložena také neautorizovaná hesla, síť se snaží díky své jednoduchosti je přiřadit ke vzorovým uživatelům. Neautorizovaná hesla se tak více přibližují těmto třem ideálním hodnotám. Pro schopnost lépe odlišit neautorizovaná hesla od těch správných (ideálních) bylo nutno použít komplexnější architekturu (větší počet skrytých vrstev a neuronů v těchto skrytých vrstvách).

Neautorizovaná hesla byla předložena nejprve nejúspěšnějšímu modelu sítě pro heslo optimalizace: logaritmická aktivační funkce v konfiguraci vrstev 2 x 10 neuronů. Jak je

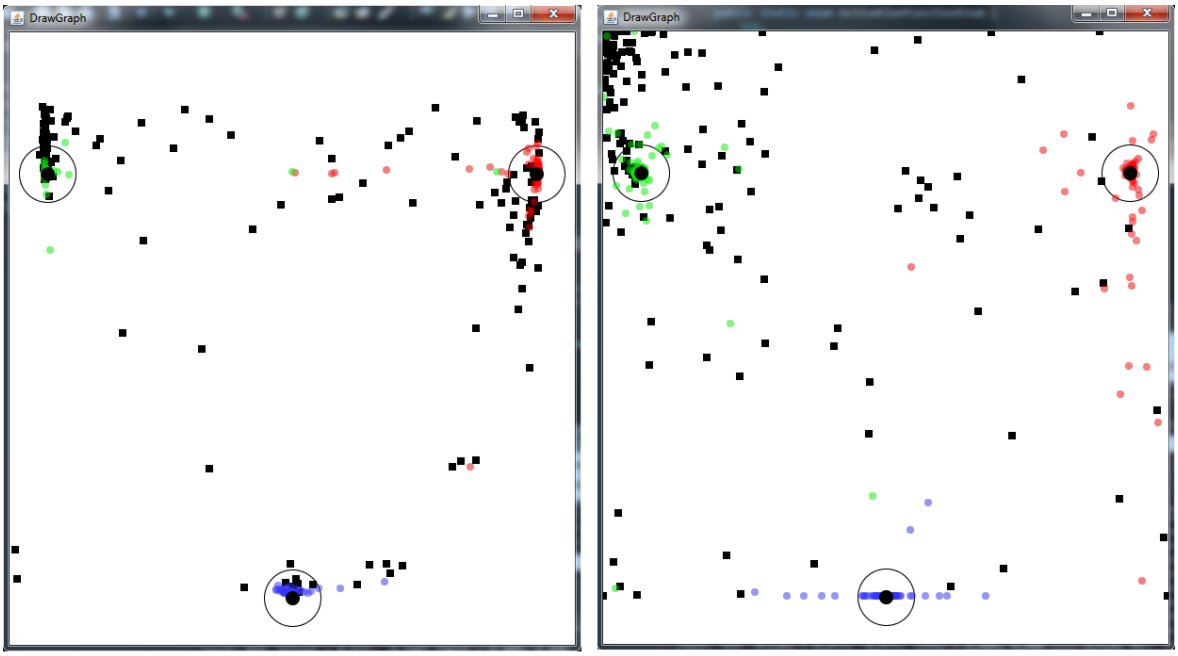
možno vidět na obrázku 28, jsou predikované hodnoty touto sítí pro neautorizovaná hesla rozřazeny blíže k ideálním hodnotám tří uživatelů (na obrázku jsou neautorizovaná hesla prezentována černými čtverečky).



Log 2 x 10, iter = 11 000 **Log 8 x 30, iter = 1 200**

Obrázek 28 - Srovnání výstupu jednoduché a složité architektury sítě Log pro heslo optimalizace

V případě složitější architektury 8 x 30 neuronů jsou hodnoty více rozprostřeny v prostoru. I tak jsou však některá neautorizovaná hesla stále blízko hodnotám 3 uživatelů, což je nežádoucí. Na obrázku je také v grafu u vzorových uživatelů naznačena kružnicí prahová hranice 5% odchylky. Několik hodnot neautorizovaných hesel je uvnitř této hranice.



TanH, 1 x 5, iter = 1 200 **TanH, 3 x 30, iter = 1 300**

Obrázek 29 - Srovnání výstupu jednoduché a složité architektury sítě TanH pro heslo velikonocnizajic

Dále byla otestována druhá nejúspěšnější architektura sítě pro heslo velikonocnizajic, tj. TanH v konfiguraci 1 x 5 neuronů, viz obrázek 29. V případě jednoduché architektury 1 x 5 neuronů jsou opět neautorizovaná hesla rozřazena blíže ideálním hodnotám 3 uživatelů. Architekturu bylo nutno rozšířit. Ke srovnání byla použita rozšířená architektura 3 x 30 neuronů, která vykazovala lepší schopnost odlišit neautorizovaná hesla. Spolu s rozptýlením neautorizovaných hesel do prostoru se však rozptylují i hesla správná, což není ideální stav, avšak z hlediska bezpečnosti je přijatelnější se potýkat s vyšším počtem chybně odmítnutých správných hesel, než s vyšším počtem chybně přijatých neautorizovaných hesel.

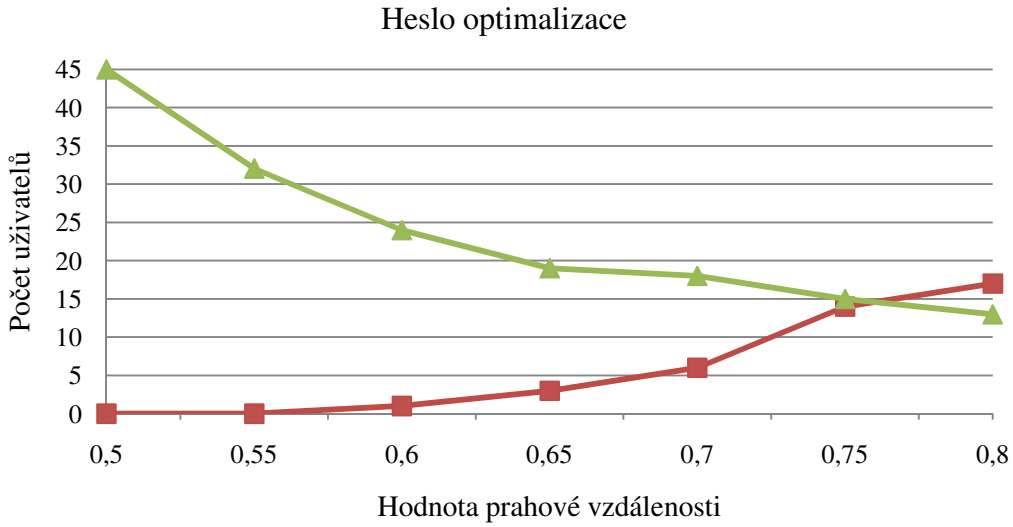
Přesný počet chybně přijatých neautorizovaných hesel však nezávisí na vlastní konfiguraci sítě tolik, jako na ní závisí jejich vizuální rozptyl, resp. entropie. Při každém novém tréninku sítě je počet chybně přijatých neautorizovaných hesel jiný i přesto, že byla ponechána stejná konfigurace sítě, a pohybuje se v rozmezí 20 až 100 hesel. Tento fakt je možno vidět také na obrázcích 28 a 29, kde přesto, že neautorizovaná hesla jsou v případě složitějších sítí více rozptýlena a tedy pravděpodobnost chybné autentizace je menší, stále se uvnitř kružnic hranice 5% odchylky nachází několik málo hesel, která by měla být odmítnuta. Bohužel se nepodařilo nalézt žádnou konfiguraci sítě pro obě hesla, která by jednoznačně odlišila neautorizovaná hesla od správných tak, aby počet chybně přijatých neautorizovaných hesel byl nulový. Vždy se jednalo o několik desítek neautorizovaných hesel, která byla chybně přijata. Počet chybně odmítnutých správných hesel se pohyboval přibližně na stejné úrovni (správná hesla, která se pohybovala za hranicí 5% odchylky a byla tudíž odmítnuta, přičemž měla být správně přijata).

2.6.2 Ověření na algoritmu k-NN

Po předložení testovacích vzorků od neznámého uživatele algoritmu k-NN došlo k tomu, že algoritmus stejně tak jako pro vzorky známých uživatelů vybral nejbližšího souseda reprezentovaného jedním ze tří vzorových uživatelů. Algoritmus poskytuje informaci o tom, v jaké vzdálenosti se nejblíže nalezený vzorový uživatel nachází od testovacího vzorku uživatele neznámého. Pomocí této vzdálenosti lze také určit míru věrohodnosti provedeného výběru. Na základě testovacích vzorků, mezi kterými se nacházely také vzorky neznámého uživatele, byla experimentálním způsobem určena prahová vzdálenost, kterou pokud testovací vzorek překročí (tedy je dále než tato prahová vzdálenost), je odmítnut (tj. takový uživatel nebude autentizován). Vzdálenosti vzorků od ideálních hodnot pro obě hesla se pohybují v rozmezí od 0,19 do 6,04. Cílem bylo nalézt optimální hodnotu prahové vzdálenosti, pro kterou platí:

- co nejvíce správných vzorků má ke svému nejbližšímu sousedu vzdálenost menší – bude tedy co nejméně správných vzorků chybně odmítnuto i přesto, že mají být přijaty,
- co nejvíce špatných vzorků má vzdálenost větší – co nejméně špatných vzorků bude chybně přijato i přesto, že mají být odmítnuty. Protože jde o autentizaci osob, kde je předpokládán důraz na bezpečnost, měla by se hodnota chybně přijatých cizích uživatelů limitně blížit nule.

Pro obě dvě hesla byla tímto způsobem určena hodnota prahové vzdálenosti, která nejvíce vyhovovala tomuto předpokladu. Vývoj počtu chybně odmítnutých správných vzorků a chybně přijatých cizích vzorků v závislosti na hodnotě prahové vzdálenosti je možno vidět na následujících grafech:



Obrázek 30 - Graf závislosti počtu chybně akceptovaných a odmítnutých uživatelů na hodnotě prahové vzdálenosti pro heslo optimalizace

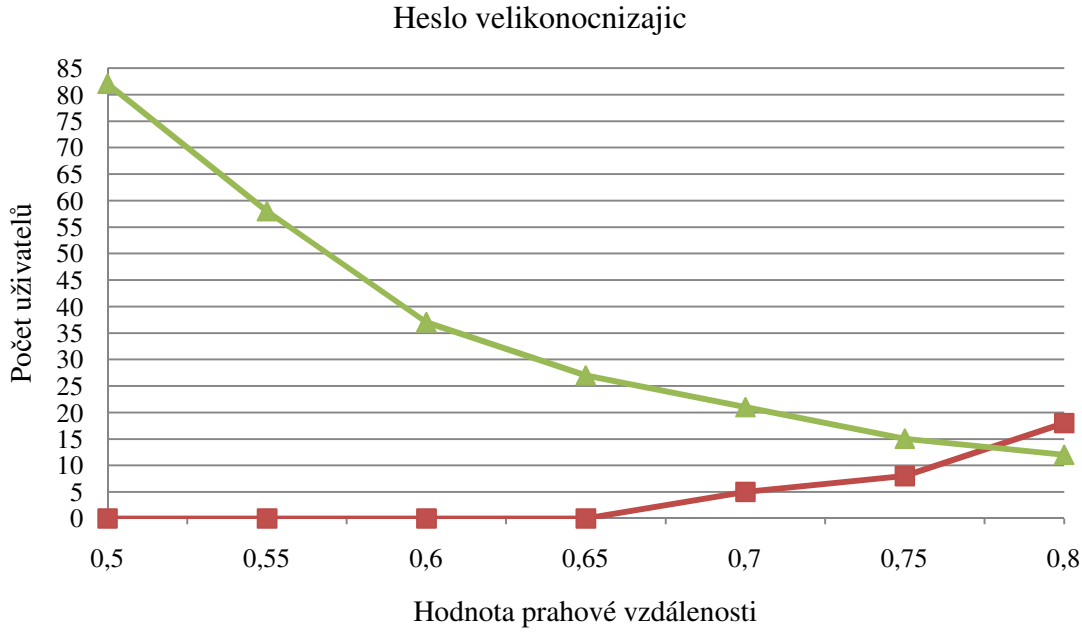
Pro heslo optimalizace byla experimentálním způsobem nalezena hodnota prahové vzdálenosti 0,55 pro kterou platí nulový počet chybně akceptovaných cizích uživatelů a co nejmenší počet chybně odmítnutých správných uživatelů (viz graf na obrázku 30, kde zelená linka znázorňuje počet chybně odmítnutých správných uživatelů a červená linka znázorňuje počet chybně přijatých cizích uživatelů).

Při optimální hodnotě prahové vzdálenosti 0,55 bylo pro heslo optimalizace z celkového počtu 452 vzorků:

- chybně přijato 0 cizích vzorků z celkového počtu 185 cizích vzorků,
- chybně odmítnuto 32 správných vzorků z celkového počtu 267 správných vzorků (cca 12%).

Pro heslo velikonocnizajic byla stejným experimentálním způsobem nalezena hodnota prahové vzdálenosti 0,65 (viz graf na obrázku 31). Při této optimální hodnotě bylo pro heslo velikonocnizajic z celkového počtu 428 vzorků:

- chybně přijato 0 cizích vzorků z celkového počtu 169 cizích vzorků,
- chybně odmítnuto 27 správných vzorků z celkového počtu 259 správných vzorků (cca 10%).



Obrázek 31 - Graf závislosti počtu chybně akceptovaných a odmítnutých uživatelů na hodnotě prahové vzdálenosti pro heslo velikonocnizajic

Algoritmus k-NN tedy umí poměrně přesně a rychle bez potřeby učení predikovat uživatele a zároveň odfiltrovat uživatele cizí, přičemž úroveň chybně odmítnutých správných uživatelů se pohybuje na úrovni 10 – 12%.

2.6.3 Ověření na Bayesovském klasifikátoru

Ze tří prezentovaných Bayesovských klasifikátorů, tedy Gaussova, multinomického a Bernoulliho, vyplynuly z předchozího zkoumání pouze dva, které jsou použitelné v případě dynamiky psaní - Gaussův a multinomický. V případě Gaussova Bayesovského klasifikátoru vypadá výstup predikovaný pro příklady neznámého uživatele následovně:

```
267: Predicted: User_B, P = [[  1.00000000e+000   1.30614929e-070   5.89701893e-181]],
Correct: FalseUser
268: Predicted: User_B, P = [[  1.00000000e+000   0.00000000e+000   2.86303904e-191]],
Correct: FalseUser
269: Predicted: User_B, P = [[  1.00000000e+000   1.09897342e-183   6.07488669e-188]],
Correct: FalseUser
270: Predicted: User_B, P = [[  1.00000000e+000   3.25726209e-257   9.50377830e-146]],
Correct: FalseUser
271: Predicted: User_B, P = [[  1.00000000e+000   4.32403916e-166   2.89018925e-099]],
Correct: FalseUser
272: Predicted: User_B, P = [[  1.00000000e+000   9.72659872e-142   5.12801416e-097]],
Correct: FalseUser
```

Pravděpodobnostní vektor má podobu stejnou, jakou má v případě uživatelů správných (tedy jeden z členů pravděpodobnostního vektoru má vždy hodnotu 1). Predikovaný výsledek pro neznámého uživatele se tak jeví jako správný, tj. algoritmus si je „jistý“ svou volbou. Implementace tohoto Bayesovského klasifikátoru bohužel neposkytuje informaci o věrohodnosti zvoleného výstupu v podobě vzdálenosti, jako tomu bylo například u algoritmu k-NN. Není tak možno algoritmu říci, aby přijímal výsledky pouze s přesností do určitého prahového limitu. Tedy tento klasifikátor není vhodný pro filtraci neznámých

uživatelů ze souboru vzorků dat, protože není možno věrohodnost na základě ničeho ověřit.

Multinomický Bayesovský klasifikátor, jak již bylo řečeno, poskytuje výstup, který se nejvíce podobá neuronovým sítím. Tento výstup obsahuje pravděpodobnostní vektor, kde pravděpodobnost je více rozprostřena mezi jednotlivé uživatele. Toto je základní předpoklad, aby algoritmus mohl také případně odfiltrovat neznámé uživatele tím, že pravděpodobnost takového uživatele bude nízká. Výstup multinomického Bayesovského klasifikátoru pro předložené příklady od neznámého uživatele je (nezávisle na volbě velikosti parametru α):

```
267: Predicted: User_B, P = [[ 0.98505971  0.00466605  0.01027424]], Correct: FalseUser
268: Predicted: User_B, P = [[ 0.98163177  0.01100484  0.00736339]], Correct: FalseUser
269: Predicted: User_B, P = [[ 0.91682632  0.04660693  0.03656675]], Correct: FalseUser
270: Predicted: User_B, P = [[ 0.95230595  0.0235961   0.02409795]], Correct: FalseUser
271: Predicted: User_B, P = [[ 0.90302085  0.0518756   0.04510354]], Correct: FalseUser
272: Predicted: User_B, P = [[ 0.86398778  0.08990557  0.04610665]], Correct: FalseUser
```

I přesto, že algoritmus poskytuje pravděpodobnosti jednotlivých uživatelů podobně jako neuronová síť, je ve všech případech vzorků od neznámého uživatele predikován uživatel User_B. Pravděpodobnost, že neznámý uživatel je uživatel User_B, je až příliš vysoká (někdy až 98%) a není možno ji použít pro rozhodnutí, zda se jedná o neznámého nebo správného uživatele. Implementace tohoto Bayesovského klasifikátoru také neposkytuje jinou informaci, podle které by bylo možno rozlišit věrohodnost, jako např. vzdálenost u algoritmu k-NN. Tedy tento klasifikátor také není vhodný pro filtraci neznámých uživatelů ze souboru vzorků dat.

Bayesovské klasifikátory jsou skvělým prostředkem pro klasifikaci na již odfiltrovaných datech, kde poměrně přesně dovedou predikovat výstup. Avšak pro odfiltrování vzorků neznámých uživatelů jsou nevhodné. Toto zjištění bylo objeveno až po implementaci těchto klasifikátorů v rámci této diplomové práce.

3 Shrnutí výsledků

V případě všech klasifikátorů bylo testování úmyslně rozděleno do dvou částí, první část pojednávala čistě o klasifikaci známých uživatelů, aby bylo možno porovnat přesnost klasifikace pro „známá“ data, druhá část poté samostatně ověřovala přesnost a důvěryhodnost algoritmů na datech od „neznámých uživatelů“.

Přesnost predikce pro známé uživatele měla nejvyšší úroveň u algoritmu k-NN a neuronové sítě, kde byly uživatelé predikováni se 100% přesností, tedy ani u jednoho uživatele nedošlo k záměně. U neuronové sítě se ukázalo vhodnější použití jednodušších architektur (Log 2 x 10 pro heslo optimalizace, TanH 1 x 5 pro heslo velikonocnizajic) na rozdíl od složitějších, které vyžadovaly více času pro natrénování. V případě Gaussova a multinomického Bayesovského klasifikátoru byli uživatelé predikováni s chybou do 3%. Jako nevyhovující se ukázal Bernoulliho Bayesovský klasifikátor, který predikoval chybný výsledek i pro více jak polovinu vzorků.

Ověření algoritmu k-NN na neautorizovaných heslech ukázalo, že pro tato data algoritmus stejně tak, jako pro známá data vybere jeden ze tří nejbližších ideálních výstupů. Avšak díky vzdálenosti, kterou algoritmus poskytuje jako jeden z výstupních parametrů, je možno rozhodnout, zda je tato predikce věrohodná. Experimentálním způsobem bylo možno nalézt takovou limitní hodnotu prahové vzdálenosti, která zajistila spolehlivé odfiltrování neautorizovaných hesel, tj. nebyl chybně autentizován žádný neznámý uživatel. Díky stanovenému prahu byli chybně odmítnuti někteří známí uživatelé, avšak jejich počet se pohyboval na úrovni přijatelných 10 – 12%. Volba hodnoty prahové vzdálenosti nastavuje „přísnost“ algoritmu a je možno ji dle potřeby modifikovat.

V případě ověřování neuronové sítě opět výstup závisel na volbě architektury sítě a bylo nutno otestovat řadu kombinací. Vhodnou volbou architektury bylo možno kontrolovat rozptyl výstupů. Ukázalo se, že složitější architektury dovedou lépe rozpoznat neautorizovaná hesla, která tím byla více rozprostřena mezi ideálními hodnotami správných uživatelů (jako nejefektivnější architektury byly experimentálně zvoleny Log 8 x 30 neuronů pro heslo optimalizace a TanH 3 x 30 pro heslo velikonocnizajic). Avšak i přesto se několik desítek neautorizovaných hesel vždy pohybovalo velice blízko ideálním hodnotám s odchylkou menší než 5%. Jejich počet byl variabilní po každém jednotlivém tréninku sítě a pohyboval se mezi 20 až 100 hesly (tj. chybovost až 20%). Bohužel se nepodařilo nalézt žádnou architekturu takovou, která by spolehlivě s maximální přesností odfiltrovala neautorizovaná hesla od těch správných a přiblížila se tak přesnosti algoritmu k-NN.

Bayesovské klasifikátory poskytují jako svůj výstup pravděpodobnostní vektor, který však v případě neautorizovaných hesel není nijak rozlišitelný od vektoru pro hesla známá. Pravděpodobnost predikovaného výsledku pro neautorizovaná hesla je v mnoha případech téměř na 100% úrovni i přesto, že predikovaný výstup není správný. Jejich implementace v použité knihovně Scikit-Learn neposkytuje navíc takovou informaci, na základě které by bylo možno vyhodnotit spolehlivost predikované hodnoty. Bayesovské klasifikátory se

tedy v tomto případě ukázaly jako nevhodné pro spolehlivé rozpoznání neautorizovaných hesel.

Tabulka 2 názorně srovnává chybovosti jednotlivých algoritmů z pohledu chybně přijatých neautorizovaných hesel (α), chybně odmítnutých správných hesel (β) a chybné záměny jednoho predikovaného uživatele za druhého (γ).

Klasifikátor	**α**	**β**	**γ**
Neuronové sítě	5 – 20%	5 – 20%	0%
k-NN	0%	10 – 12%	0%
Gaussův BK	není k dispozici	není k dispozici	1,13%
Multinomický BK	není k dispozici	není k dispozici	2,63%
Bernoulliho BK	není k dispozici	není k dispozici	až 56%

Tabulka 2 - Srovnání implementovaných klasifikátorů z pohledu zkoumaných veličin

Co do přesnosti predikovaných výsledků a bezpečnosti se na základě popsaných aspektů jako optimální jeví algoritmus k-nejbližších sousedů (k-NN).

4 Závěry a doporučení

Implementované klasifikátory mají tyto porovnatelné vlastnosti: složitost implementace algoritmu, rychlost učení se, přesnost predikce, bezpečnost a složitost reálného nasazení. Jednotlivým tématům jsou věnovány následující kapitoly. Klasifikátory jsou porovnávány čistě z pohledu problematiky klasifikace uživatelů na základě dynamiky psaní (keystroke recognition).

4.1 Složitost implementace a rychlost učení se

Neuronové sítě co do své rozsáhlé teorie a možností implementace jsou nejsložitějším implementovaným algoritmem. Nalezení optimální architektury sítě a navržení vhodné interpretace vstupu a výstupu sítě je stěžejní pro její správnou funkci. Architekturu je nutno hledat experimentálně, tedy je nutno otestovat velké množství kombinací, které by nejlépe vyhovovaly dané problematice. Nevýhodou neuronových sítí je to, že do volby vhodné architektury mluví mnoho aspektů: počet vstupních neuronů, počet výstupních neuronů, množství trénovacích dat, složitost trénovacích dat, apod. Jak bylo možno vidět v kapitole 2.3 (Implementace neuronové sítě), pro obě hesla se hodila jiná architektura. Volba architektury je tedy velmi variabilní. Při každé změně trénovacích dat (např. přidání uživatele v případě klasifikace), je nutno síť přetrénovat, případně přehodnotit celou architekturu, což klade vysoké časové nároky jak na implementaci, tak na přeučení sítě.

Algoritmy k-NN a Bayesovské klasifikátory není třeba učit tak, jako neuronové sítě. Podstatou jejich funkčnosti je vyhledávání a práce nad uloženými daty. Tyto algoritmy je tedy možno nad trénovacími daty aplikovat velmi rychle. Jsou také lepší volbou co do škálovatelnosti dat, neboť při přidání nového uživatele není nutno algoritmy přetrénovat. Na rozdíl od neuronových sítí však kladou vyšší nároky na úložný prostor. Avšak v dnešní době, kdy jsou paměťové kapacity počítačů neustále posouvány kupředu, jsou kapacitní nároky pro algoritmy k-NN a Bayesovské klasifikátory obvykle zanedbatelné.

Z pohledu složitosti algoritmu a rychlosti učení je vhodnější použít algoritmy k-NN a Bayesovské klasifikátory.

4.2 Přesnost predikce a bezpečnost

Aby bylo možné lépe předpovědět chování jednotlivých klasifikátorů při praktickém nasazení, je vhodné algoritmy otestovat na více jak třech uživatelích. Díky charakteru zkoumané problematiky, kdy je klasifikován uživatel na základě dynamiky psaní svého napsaného hesla, je nepravděpodobné, že bude více než několik málo uživatelů, mezi kterými bude probíhat klasifikace. Nejpravděpodobnější uvažovaný scénář počítá s jedním uživatelem, pro kterého je přesně znám jeho vzor dynamiky psaní (např. uživatel běžného PC). Proto nebyla klasifikace algoritmů testována pro více jak 3 uživatele. Tři uživatelé byli zvoleni rovněž z důvodu optimální možnosti interpretace výstupů neuronové sítě ve 2D prostoru. Oproti tomu může existovat celá řada uživatelů neznámých, kteří se snaží heslo napodobit a kteří by měli být rozpoznáni a odmítnuti. Algoritmy byly podrobeny testování za použití tzv. neautorizovaných hesel od neznámých uživatelů (celkem cca 450 vzorků hesel od 10 uživatelů).

Z pohledu přesnosti predikce a bezpečnosti je optimální použít algoritmus k-NN, který spolehlivě dovedl odfiltrovat neznámé uživatele, zatímco přesně klasifikoval uživatele známé, viz shrnutí výsledků v kapitole 3 .

4.3 Složitost reálného nasazení

Nasazení výše uvedených algoritmů do reálného provozu závisí zejména na jejich složitosti implementace a spolehlivosti. Jak ukázaly předchozí kapitoly, s důrazem na jednoduchost, škálovatelnost, rychlost a přesnost se jako nejlepší algoritmus projevil algoritmus k-nejbližších sousedů. Použití tohoto algoritmu k vyhodnocení dynamiky psaní jako jeden z několika biometrických znaků člověka může být vhodné např. na pracovních stanicích či jiných pracovních zařízeních, kde je vyžadována vyšší úroveň zabezpečení. Díky vlastnostem dynamiky psaní, kdy vyhodnocovaná osoba musí být v klidu, musí psát oběma rukama, nesmí být ve stresu či pod vlivem návykových látek a ve většině případů by měla mít k dispozici také stále stejnou klávesnici, není pravděpodobné, že by algoritmy byly použitelné v běžných PC pro domácí použití. Reálné použití může být např. v bankách a jim podobných institucích pro administrátorské přístupy, přístupy k citlivým údajům, apod., kde je v zájmu zkoumaného uživatele psát heslo vždy dle stejného nenapodobitelného vzoru. Implementace algoritmu sice vyžaduje určitý čas pro sběr vzorků hesla, toto však může být sbíráno průběžně, než je algoritmus „aktivován“ do reálného použití, kdy bude chybné vzory hesel již odmítat.

5 Závěr

Oproti očekávanému výsledku, kdy byly neuronové sítě pro jejich široké použití v oblasti klasifikace předpokládány jako spolehlivá metoda, se ukázalo, že naopak pro případ klasifikace na základě dynamiky psaní vhodné nejsou. Důvodem je existence takových vzorků hesel ve vstupním souboru, které by neměly být přiřazeny k ani jednomu známému uživateli, ale naopak by měly být odmítnuty (tzv. neautorizovaná hesla). Neuronová síť není bezpečným nástrojem na jejich spolehlivé rozpoznání. Chybná autentizace těchto hesel se pohybovala na úrovni až 20%. Pokud by byl k dispozici pouze soubor vzorků hesel, který je již zbaven neautorizovaných hesel, neuronová síť by si v klasifikaci uživatelů vedla velmi dobře. Během zkoumání v rámci této diplomové práce klasifikovala uživatele se 100% přesností.

Bayesovské klasifikátory a algoritmus k-nejbližších sousedů jsou ve srovnání s neuronovými sítěmi jednodušší v ohledech: výpočetní složitosti, implementační složitosti, škálovatelnosti. Bayesovské klasifikátory se však ukázaly jako nevhodné z toho důvodu, že stejně jako neuronové sítě neumí spolehlivě odfiltrovat neautorizovaná hesla. K neautorizovanému heslu je Bayesovským klasifikátorem přiřazen vždy vzorový uživatel, jako by se jednalo o heslo uživatele známého. Není pak k dispozici takový parametr, na základě kterého by bylo možno rozhodnout o spolehlivosti výstupu.

Jako optimální pro případ klasifikace uživatelů na základě jejich dynamiky psaní se ukázal algoritmus k-nejbližších sousedů, který vyhledá ke zkoumanému vzorku hesla nejbližší vzor a na základě limitní prahové vzdálenosti zhodnotí, zda je tato volba dostatečně jistá, tj. zda daného uživatele autentizovat. Nejen že tento algoritmus dokázal klasifikovat známé uživatele se 100% přesností stejně, jako neuronová síť, ale také na základě experimentálně zvolené prahové vzdálenosti správně odmítl všechny neznámé uživatele, jejichž hesla byla algoritmu předložena (chybně neautorizoval ani jednoho neznámého uživatele), přičemž chybně odmítl pouhých 10 – 12% správných uživatelů. Tedy cca 1 z deseti pokusů správného uživatele byl algoritmem chybně odmítnut.

Tématem dalšího výzkumu se nabízí hlouběji prozkoumat implementaci neuronových sítí a nalézt takovou strukturu sítě, která by spolehlivě dokázala filtrovat neautorizovaná hesla. Například prověřit, zda tohoto stavu může být dosaženo kombinací různých aktivačních funkcí v jednotlivých vrstvách sítě. Nabízí se použití rektifikovaných lineárních neuronů, které jsou díky svému funkčnímu průběhu schopny se aktivovat či deaktivovat a za jistých podmínek tak zcela zamezit nebo obnovit tok informací v jejich části sítě [10, 18, 19]. Dále otestovat, zda bude přínosem neautorizovaná hesla považovat za dalšího uživatele a tedy tomuto přizpůsobit počet výstupních neuronů a normalizovat dle toho výstupní hodnoty. Síť pak podrobit pokročilejšímu tréninku s použitím metody cross-validation [40], optimalizovat ji pro rozsáhlejší data [33] apod. Stejnému zkoumání mohou být podrobeny Bayesovské klasifikátory a může být zkoumáno nalezení takového pravidla v jejich implementaci, na základě kterého bude možno vypočtené výstupy klasifikátoru vyhodnotit jako spolehlivé, či nespolehlivé. Neméně důležitý je také výzkum reálného nasazení, tedy jak nejlépe vybranou metodu uvést do praktického použití, optimalizovat ji a otestovat její

spolehlivost v praxi na více uživatelích s řádově několikanásobně více vzorky dat. Implementovány a srovnány mohou být případně další algoritmy strojového učení [42].

6 Seznam použité literatury

[1] MONROSE, Fabian a Aviel D. RUBIN. Keystroke dynamics as a biometric for authentication. Future Generation Computer Systems 16 (2000) 351–359 [online]. a Courant Institute of Mathematical Science, New York University, New York, NY, USA, 1999 [cit. 2016-04-24]. Dostupné z: http://www.cs.columbia.edu/~hgs/teaching/security/hw/keystroke.pdf

[2] DOMINGOS, Pedro. University of Washington: Machine Learning. Coursera [online]. [cit. 2016-04-24]. Dostupné z: https://www.coursera.org/course/machlearning

[3] QUINLAN, J.R,. Induction of Decision Trees. Machine Learning 1. Kluwer Academic Publishers, Boston, 1986, 81-106.

[4] COVER, T. a P. HART. Nearest neighbor pattern classification. IEEE Transactions on Information Theory [online]. 1967, 13(1), 21-27 [cit. 2016-04-22]. DOI: 10.1109/TIT.1967.1053964. ISSN 0018-9448. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1053964

[5] MCLACHLAN, Geoffrey J. Discriminant analysis and statistical pattern recognition. 1st ed. Hoboken: Wiley-Interscience, 2004. Wiley series in probability and statistics. ISBN 0-471-69115-1.

[6] GROSSBERG, Stephen. Nonlinear neural networks: Principles, mechanisms, and architectures. Neural Networks [online]. 1988, 1(1), 17-61 [cit. 2016-04-22]. DOI: 10.1016/0893-6080(88)90021-4. ISSN 08936080. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/0893608088900214

[7] RENNIE, Jason D. M., Lawrence SHIH, Jaime TEEVAN a David R. KARGER. Tackling the Poor Assumptions of Naive Bayes Text Classifiers. Artificial Intelligence Laboratory; Massachusetts Institute of Technology; Cambridge, MA 02139, 2003.

[8] RISH, Irina. An empirical study of the naive Bayes classifier. Yorktown Heights, NY 10598: IBM Research Division, Thomas J. Watson Research Center, 2001.

[9] BANZHAF, Wolfgang. Genetic programming: an introduction on the automatic evolution of computer programs and its applications. Heidelburg: Dpunkt-verlag, c1998. ISBN 3920993586.

[10] HINTON, Geoffrey. University of Toronto: Neural Networks for Machine Learning. Coursera [online]. 30.09.2012 [cit. 2016-01-03]. Dostupné z: https://class.coursera.org/neuralnets-2012-001.

[11] The MNIST database of handwritten digits [online]. Yann LeCun, Corinna Cortes, Christopher J.C. Burges [cit. 2016-04-23]. Dostupné z: http://yann.lecun.com/exdb/mnist/

[12] KAPLAN, Craig S. VORONOI DIAGRAMS AND ORNAMENTAL DESIGN [online]. University of Washington: Department of Computer Science & Engineering,

1999 [cit. 2016-04-24]. Dostupné z: http://www.cgl.uwaterloo.ca/csk/washington/tile/papers/kaplan_isama1999.pdf

[13] Petřík Oldřich, Západočeská univerzita v Plzni, Matematické modelování, Voroného diagramy, 7.3.2008. Dostupné z: http://home.zcu.cz/~mikaMM/Galerie%20studentskych%20praci%20MM/2008/Pet%C5%99%C3%ADk%20Old%C5%99ich%20-%20Voron%C3%A9ho%20diagramy.pdf

[14] HEATON, Jeff. Introduction to neural networks with Java. 2nd ed. St. Louis: Heaton Research, 2008, xxxviii, s. 39-439. ISBN 1-60439-008-5.

[15] CS231n Convolutional Neural Networks for Visual Recognition [online]. [Cit. 24.4.2016]. Dostupné z: http://cs231n.github.io/neural-networks-1/

[16] Nguyen, Thai (2010) "Total Number of Synapses in the Adult Human Neocortex," Undergraduate Journal of Mathematical Modeling: One + Two: Vol. 3: Iss. 1, Article 26. DOI: http://dx.doi.org/10.5038/2326-3652.3.1.26 Available at: http://scholarcommons.usf.edu/ujmm/vol3/iss1/26.

[17] CS-449: Neural Networks [online]. Willamette University: Instructor: Genevieve Orr, 1999 [cit. 2016-04-23]. Dostupné z: http://www.willamette.edu/~gorr/classes/cs449/multilayer.html

[18] GLOROT, Xavier, Antoine BORDES a Yoshua BENGIO. Deep Sparse Rectifier Neural Networks [online]. DIRO, Université de Montréal [cit. 2016-04-23]. Dostupné z: http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf

[19] NAIR, Vinod a Geoffrey E. HINTON. Rectified Linear Units Improve Restricted Boltzmann Machines. Appearing in Proceedings of the 27 th International Conference on Machine Learning, Haifa, Israel. Department of Computer Science, University of Toronto, Toronto, ON M5S 2G4, Canada, 2010. Dostupné z: http://machinelearning.wustl.edu/mlpapers/paper_files/icml2010_NairH10.pdf

[20] ROJAS, Raúl. Neural Networks: A Systematic Introduction. Berlin: Springer, 1996.

[21] TURCHETTI, Claudio. Stochastic models of neural networks. Tokyo: Ohmsha, c2004. Frontiers in artificial intelligence and applications, v. 102. ISBN 4274906264.

[22] JEFF HEATON. Programming neural networks with Encog3 in Java. 2011. ISBN 9781604390216.

[23] KAVZOGLU, Taskin. Determining Optimum Structure for Artificial Neural Networks. In Proceedings of the 25th Annual Technical Conference and Exhibition of the Remote Sensing Society, Cardiff, UK, pp. 675-682, 8-10 September 1999 [online]. The University of Nottingham, NG7 2RD, Nottingham, 1999 [cit. 2016-04-24]. Dostupné z: http://www.academia.edu/13017783/Determining_Optimum_Structure_for_Artificial_Neural_Networks

[24] HORNIK, Kurt. Approximation capabilities of multilayer feedforward networks. Neural Networks [online]. 1991, 4(2), 251-257 [cit. 2016-04-23]. DOI: 10.1016/0893-

6080(91)90009-T. ISSN 08936080. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/089360809190009T

[25] PRIDDY, Kevin L a Paul E KELLER. Artificial neural networks: an introduction. Bellingham, Wash.: SPIE Press, c2005. ISBN 0819459879.

[26] JACKMAN, Simon. Bayesian analysis for the social sciences. 1st ed. Chichester: Wiley, 2009. Wiley series in probability and statistics. ISBN 978-0-470-01154-6.

[27] Naive Bayes — scikit-learn 0.17.1 documentation [online]. [Cit. 24.4.2016]. Dostupné z: http://scikit-learn.org/stable/modules/naive_bayes.html

[28] Naive Bayes Classifier: Naive Bayes Classifier Introductory Overview. Dell Software: Technical Documentation [online]. Dell Inc., 2015 [cit. 2016-04-24]. Dostupné z: http://documents.software.dell.com/Statistics/Textbook/Naive-Bayes-Classifier

[29] ZHANG, Harry. The Optimality of Naive Bayes [online]. Faculty of Computer Science, University of New Brunswick, Fredericton, New Brunswick, Canada, 2004 [cit. 2016-04-23]. Dostupné z: http://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf

[30] KARSOLIYA, Saurabh. Approximating Number of Hidden layer neurons in Multiple Hidden Layer BPNN Architecture. International Journal of Engineering Trends and Technology- Volume3Issue6- 2012 [online]. Maulana Azad National Institute of Technology, Bhopal, India, 2012 [cit. 2016-04-27]. ISSN 2231-5381. Dostupné z: http://www.ijettjournal.com/volume-3/issue-6/IJETT-V3I6P206.pdf

[31] HEATON, Jeff. Encog Machine Learning Framework [online]. [Cit. 27.4.2016]. Dostupné z: http://www.heatonresearch.com/encog/.

[32] RAFIQ, M.Y, G BUGMANN a D.J EASTERBROOK. Neural network design for engineering applications. Computers & Structures [online]. 2001, 79(17), 1541-1552 [cit. 2016-04-24]. DOI: 10.1016/S0045-7949(01)00039-6. ISSN 00457949. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0045794901000396

[33] YANG, Jie, Jun MA, Matthew BERRYMAN a Pascal PEREZ. A structure optimization algorithm of neural networks for large-scale data sets. In: 2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE) [online]. IEEE, 2014, s. 956-961 [cit. 2016-04-24]. DOI: 10.1109/FUZZ-IEEE.2014.6891662. ISBN 978-1-4799-2072-3. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6891662

[34] NAKAMA, Takehiko. Comparisons of Single- and Multiple-Hidden-Layer Neural Networks [online]. s. 270 [cit. 2016-04-24]. DOI: 10.1007/978-3-642-21105-8_32. Dostupné z: http://link.springer.com/10.1007/978-3-642-21105-8_32

[35] Underfitting vs. Overfitting — scikit-learn 0.17.1 documentation [online]. [Cit. 27.4.2016]. Dostupné z: http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

[36] Entropie – WikiSkripta [online]. [Cit. 27.4.2016]. Dostupné z: http://www.wikiskripta.eu/index.php/Entropie

[37] Python-xy.GitHub.io by python-xy [online]. [Cit. 27.4.2016]. Dostupné z: http://python-xy.github.io/.

[38] Nearest Neighbors — scikit-learn 0.17.1 documentation [online]. [Cit. 24.4.2016]. Dostupné z: http://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-classification.

[39] BELLMAN, Richard. Dynamic programming. 6. print. Princeton: Princeton University Press, 1972. ISBN 9780691079516.

[40] 3. Machine Learning 102: Practical Advice — Machine Learning for Astronomy with Scikit-learn [online]. [Cit. 27.4.2016]. Dostupné z: http://www.astroml.org/sklearn_tutorial/practical.html

[41] Neuron Diagram | ASU - Ask A Biologist [online]. 2011. [Cit. 27.4.2016]. Dostupné z: https://askabiologist.asu.edu/neuron-anatomy

[42] MURPHY, Kevin P. Machine learning: a probabilistic perspective. Cambridge, Mass.: MIT Press, c2012. Adaptive computation and machine learning. ISBN 978-0-262-01802-9.

Zadání práce

Práce přidělena dne:	23.1.2015
Přiděleno studentovi:	Kozlovský Jan (I1300422)
Přidělené téma:	Biometrická autentizace
Vedoucí:	Ing. Karel Petránek
Konečný název práce:	Biometrická autentizace
Anglický název práce:	Biometric Authentication
Cíl práce	Cílem práce je prozkoumat existující metody strojového učení za účelem použití pro biometrickou autentizaci, navrhnout vlastní implementaci prozkoumaných metod, srovnat dosažené výsledky na testovacích datech a shrnout možnosti využití zkoumaných metod.
Osnova práce:	1. Prozkoumat existující metody biometrické autentizace, 2. Návrh a popis vlastního řešení, 3. Implementace metod, 4. Testování a porovnání implementovaných metod.
Datum oficiálního zadání práce:	23.1.2015