Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

Detekce tématu dokumentu

BAKALÁŘSKÁ PRÁCE

Plzeň, 2019

Jan Kandyba

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne

.....

Poděkování

Mé vřelé poděkování patří mému vedoucímu bakalářské práce panu Ing. Zbyňkovi Zajíci, Ph.D. za čas strávený konzultacemi, odborné vedení práce, pomoc při zpracování doporučením vhodné odborné literatury a kvalitních zdrojů. Dále chci poděkovat panu Ing. Jaromíru Novotnému za poskytnutí informací ohledně použití lemmatizace a TFIDF parametrizace. Další poděkování patří panu Doc. Ing. Pavlu Ircingovi, Ph.D. za poskytnutí lematizátoru pro český jazyk. Děkuji také zakladatelům stránek *edx.org* a *coursera.org* za poskytnutí obecných informací o klasifikaci a neuronových sítí. V neposlední řadě děkuji za podporu mým přátelům a mé rodině.

Anotace

Tato práce se zabývá problematikou zpracování přirozeného jazyka, konkrétně klasifikací dokumentů a s použitím knihoven Pythonu testuje jejich úspěšnost pro úlohu detekce tématu na reálných datech od jazykové poradny Ústavu pro jazyk český Akademie věd ČR (projekt DG16P02B009). Mezi vybrané algoritmy patří Support Vector Classification, lineární diskriminační analýza a především aplikace neuronových sítí.

Klíčová slova: klasifikace, neuronové sítě, učení s učitelem, lineární diskriminační analýza, podpůrné vektory

Abstract

The aim of this work is to investigate the methods of document classification and with the support of Python libraries test their efficiency rate on the real data from Czech Language Institute of the Czech Academy of Sciences (project DG16P02B009). The specified algorithms are Support Vector Classification, Linear Discriminant Analysis and primarily the application of the neural networks.

Keywords: classification, neural networks, supervised learning, linear discriminant analysis, support vectors

Obsah

Úvod

1. Cíl bakalářské práce

Cílem práce je nastudovat metody klasifikace dokumentů a s použitím knihoven Pythonu otestovat jejich úspěšnost pro úlohu detekce tématu na reálných datech z jazykové poradny Ústavu pro jazyk český Akademie věd ČR (projekt DG16P02B009). Mezi vybrané algoritmy patří Support Vector Classification, lineární diskriminační analýza a především aplikace neuronových sítí. Jelikož se jedná o učení s učitelem, byly dopředu dány třídy, do kterých bylo následně klasifikováno – třídy byly odvozené na základě dat z jazykové poradny. Práce obsahuje teoretický úvod do dané problematiky, kde jsou rozebrány úlohy zpracování přirozeného jazyka, mezi které patří i téma této bakalářské práce – detekce tématu dokumentu. Dále jsou zde popsány principy klasifikace obecně i použité klasifikační algoritmy. V neposlední řadě práce obsahuje experimentální část, kde byly tyto algoritmy řádně otestovány, vyhodnoceny a porovnány.

2. Motivace

Práce je založena na projektu NAKI, který se zabývá klasifikací otázek kladených na Ústav pro jazyk český [3]. Ten nabízí možnost konzultace různých záležitostí týkající se českého jazyka – např. gramatických správností určitých slov, vysvětlování nově zavedených pravidel v českém jazyce a další. Cílem je vytvořit semi-automatický systém pro zpracovávání a uchovávání dotazů, který usnadní vyhledávání a kategorizaci dotazů jazykovým poradcům i uživatelům. Tento proces zahrnuje několik úloh zpracování přirozeného jazyka, jako jsou automatické rozpoznávání řeči, detekce slov a klasifikace tématu. Každá z těchto úloh s sebou přináší určité výzvy – automatické rozpoznávání řeči se musí zabývat problémem detekce cizích slov, neboť v poslední době přibývá spousta cizích slov do českého jazyka (nejčastěji přechýlené anglické výrazy). Na základě detekovaných slov systém klasifikuje položenou otázku do určité skupiny, avšak finální klasifikaci musí potvrdit samotný jazykový poradce – klasifikace se v tomto případě musí zabývat skutečností, že v 1 otázce se může vyskytovat více témat najednou – musí tedy ve správném okamžiku detekovat aktuální téma. Tento plán je však do budoucna, nyní je pouze implementována offline klasifikace, která detekuje téma dotazu na základě dat poskytnutých ÚJČ.

Data poskytnutá ÚJČ jsou ručně anotované rozhovory mezi dotazujícím se a jazykovým poradcem. Tato data jsou zároveň rozdělena do 7 tříd a jsou určeny pro prvotní natrénování klasifikátorů. V projektu NAKI byly zprvu provedeny experimenty na neklasifikovaných datech, avšak zde se ukázalo, že pokud budou data nejprve ručně anotována, dosáhne se značně lepších výsledků. Při anotování se narazilo na problém, že pro některé kategorie klasifikace bylo příliš malé množství dat. Proto se musely některé třídy slučovat – při výstupu klasifikátoru je tedy nejprve jazykovému poradci nabídnuta tato vyšší třída, on pak následně může klasifikovat do konkrétní podtřídy.

Teoretická část

3. Zpracování přirozeného jazyka

Zpracování přirozeného jazyka (Natural Language Processing) [12] je rozsáhlé téma počítačových věd, strojového učení a umělé inteligence. Mezi oblasti zpracování přirozeného jazyka patří zejména rozpoznávání mluvené řeči, porozumění přirozenému jazyku nebo generování přirozeného jazyka. Přirozeným jazykem je myšlena běžná mluva, psaní běžných textů, včetně chyb obsažených jak v řeči, tak v textu.

3.1. Historie

Zpracováním přirozeného jazyka se začalo zabývat na počátku vzniku prvních počítačů. Již tehdy se zabývalo otázkou, jak zpracovat psaný text, aby mu počítač porozuměl a uměl na něj inteligentně zareagovat. V roce 1950 vydal Alan Turing článek zvaný „Intelligence“, ve kterém navrhl kritérium inteligence, později nazvané jako Turingův test [11].

Turingův test je ověření, zda se stroj zachová stejně, jako by se v dané situaci zachoval člověk. Turing ve svém testu navrhuje, že v tomto testu bude probíhat konverzace mezi počítačem a člověkem skrze pouhý textový vstup – odděluje se problematika převodu řeči na text. Pokud nástroj, který vyhodnocuje úspěšnost testu, nebude schopen rozeznat člověka od počítače, pak vyhlásí test za úspěšný. V případě úspěšného testu je stroj podle Turinga inteligentní.

Ani v dnešní době (zatím) neexistují algoritmy, které by byly natolik inteligentní, že by zvládly bez problému vést plynulou konverzaci – existuje spousta hlasových asistentů, avšak jejich schopnosti jsou stále značně omezené, ačkoliv se neustále zlepšují. Zpracování přirozeného jazyka je velice komplexní oblast a z tohoto důvodu zpočátku vznikaly pouze velice primitivní nástroje – např. stroj byl schopen odpovědět „ano“ nebo „ne“ a to pouze na limitované množství otázek, kterým byl schopen porozumět a zpracovat je. Tyto věty navíc člověk musel říct přesně, jak je byl algoritmus schopen zpracovat – pokud je řekl lehce jinak, algoritmus měl již problém s rozpoznáním. I s tímto se musí v oblasti porozumění přirozeního jazyka počítat.

Až do osmdesátých let 20. století využívala většina algoritmů ručně vypsaná pravidla, jimiž se řídila. Později však přišla revoluce v podobě strojového učení, kde se přestala vypisovat jednotlivá pravidla. Stroj je schopen se sám učit na základě určitých pravidel, případně vytvářet a odvozovat pravidla další.

Tato revoluce byla možná díky zvyšující se výpočetní síle a změně v logice zpracování samotného jazyka – byly objeveny skryté Markovovy modely a statistické modely, které dělají rozhodnutí založená na vypočtené pravděpodobnosti. Tento přístup přinesl značné zlepšení při zpracování textu – zejména pro text obsahující chyby, které jsou běžné pro data z reálného světa.

V poslední době se výzkum zaměřil zejména na samo učící se algoritmy (unsupervised a semi-supervised algoritmy). Tyto algoritmy jsou schopny se učit z dat, které nebyly ručně anotovány. Tento

přístup je značně komplikovanější než přístup učení s učitelem a často poskytuje horší výsledky. Avšak na světě existuje obrovské množství neanotovaných dat (např. celý internet), které nám dokáže tyto horší výsledky vynahradit – sice nebude vysoká přesnost, ale bude vysoké množství správně klasifikovaných prvků.

3.2. Úlohy zpracování přirozeného jazyka

Existuje obrovské množství úloh zpracování přirozeného jazyka. Reálné aplikace obvykle zahrnují několik těchto úloh. Do úloh zpracování přirozeného jazyka patří veškeré úlohy týkající se lidského jazyka – rozpoznávání řeči, textu, klasifikace témat, porozumění textu a spousta dalších. V seznamu níže jsou rozepsány některé úlohy, rozdělené do 3 kategorií: mluvená řeč, syntaxe a sémantika.

3.2.1. Mluvená řeč

Rozpoznávání mluvené řeči

Proces rozpoznávání mluvené řeči se zabývá převodem mluveného slova na psaný text. Celá tato problematika je velice komplexní, protože s sebou nese řadu problémů. Jedním z problémů je rozpoznání začátku a konce jednoho slova – v reálném světě mezi slovy neděláme téměř žádné pauzy, s čímž musí algoritmus umět počítat (dříve se musely uměle dělat pomlky mezi jednotlivými slovy). Dalšími problémy, které do této problematiky spadají je například extrakce šumu, problém překrývání hlásek, jiná výslovnost určitých písmen podle kontextu, rozpoznání začátku a konce řeči a další.

Syntéza řeči

Proces syntézy řeči převádí psaný text na zvukovou podobu. S tímto procesem se pojí například i tvorba barvy hlasu – dřívejší stroje sice uměly vyprodukovat zvuk podobný člověku, avšak zněl příliš roboticky; tyto stroje vyslovaly pouze písmeno po písmenu a nyní již dobrý syntetizér zvládne správně vyslovovat celá slova i věty.

3.2.2. Syntaxe

Značení části řeči (part-of-speech tagging)

Značení části řeči, označované také jako PoS tagging, je proces označování slova z korpusu k části mluvené řeči. Při tomto procesu se zohledňuje kontext, v jakém se dané slovo nachází. Tento proces je však značně složitější, neboť některá slova mohou reprezentovat více částí věty, mají různé významy a některé části věty bývají nevyřčené – získat tak jasný kontext daného slova bývá velmi komplikované. V běžné řeči se na některá slova, která již byla řečena, odkazuje pomocí zájmen – on, ona, ono; další části, které jsou obtížnější z textu či řeči vyextrahovat, jsou nevyřčené axiomy, které společnost zná a může se na ně nepřímo odkazovat.

Syntaktická analýza

Jedná se o proces analýzy posloupnosti formálních prvků s cílem určit jejich gramatickou strukturu vůči předem dané formální gramatice.

Generování gramatiky

Generování gramatiky je proces, který se snaží naučit formální gramatiku z množiny příznaků. Formální gramatika je množina pravidel pro tvorbu řetězců ve formálním jazyce s ohledem na syntaxi daného jazyka. Gramatika však nepopisuje význam těchto řetězců jako takových, ani nepopisuje, jak tyto slova použít v kontextu.

Lemmatizace

Lemmatizace je proces převodu slov na jejich základní tvar. (př.: slovo „lepší“ se zlemmatizuje na „dobrý“). Tomuto základnímu tvaru se říká lemma. Tento proces bývá hojně využíván při předzpracování textu pro následnou klasifikaci a byl také využíván v této bakalářské práci.

Morfologická segmentace

Jak již název napovídá, morfologická segmentace je proces rozdělení slov na jednotlivé morfémy s následnou definicí třídy těchto morfémů (morfém je nejmenší oddělitelná část slova nesoucí věcný nebo gramatický význam). Složitost této úlohy je velice závislá na jazyce, pro který je aplikována. Například angličtina má poměrně jednoduchou morfologii – často je možné tento proces přeskočit a namodelovat všechny formy nějakého slova. Čeština je oproti angličtině výrazně složitější – máme spoustu forem jednotlivých slov, avšak existují i jiné jazyky (např. turečtina), kde je forem jednoho slova ještě značně víc než v češtině.

Rozdělování vět

Tato problematika se zabývá procesem rozdělení textu na jednotlivé věty – věty často rozdělují interpunkční znaménka, ale například tečka nemusí vždy rozdělovat větu (může označovat např. zkratku). I takovýto zdánlivě jednoduchý proces je složitější, než na první pohled vypadá.

Stemming

Výsledkem stemování jsou převedená slova do jejich kořenového tvaru – např. slovo „běžím“ se převede na „běžet“.

Segmentace slov

Proces zabývající se rozdělením textu na slova. V případě angličtiny i češtiny je tento proces relativně jednoduchý, neboť slova z těchto jazyků jsou většinou rozděleny mezerou. Problém nastává, když se jedno slovo skládá například z pomlčky či jiných znaků.

3.2.3. Sémantika

Generování přirozeného jazyka

Proces, který generuje přirozený jazyk z počítačové databáze a sémantických pravidel daného jazyka.

Lexikální sémantika

Zabývá se otázkou významu jednotlivých slov v kontextu.

Strojový překlad

Strojovým překladem je myšlen automatický překlad jednoho jazyka do jiného. Automatický překlad je jedním z nejsložitějších problémů, neboť tato problematika zahrnuje několik již zmíněných oblastí – zabývá se gramatikou, sémantikou, významem jednotlivých slov atd. a na základě kombinace všech těchto oblastí se snaží dělat správný překlad.

Rozpoznávání jmenných entit

Tato problematika se zabývá rozpoznáváním jmen a názvů z textu. V jazycích jako je angličtina a čeština pomáhá např. velké první písmeno, avšak to není podmínka postačující. V češtině máme například názvy skládající se z více slov, kde pouze první slovo má velké první písmeno a tento proces by měl rozpoznat celý název.

Porozumění přirozenému jazyku

Zabývá se významem jazyka jako takového – hledá vztahy mezi slovy a snaží se jim přiřadit význam tak, aby jim počítač byl schopen porozumět. Tohoto se využívá například při strojovém překladu.

Optical character recognition

Tento proces se zabývá rozpoznáváním textu z obrazu. Na tento proces se následně mohou napojovat další procesy jako je porozumění psanému textu, automatický překlad apod.

Odpovídání na otázky

Položíme-li stroji otázku, snaží se na ní odpovědět – tato problematika se zabývá jak otázkami, které mají přesně stanovenou odpověď, tak i otázkami otevřenými, na které odpověď není známá nebo neexistuje.

Sentimentální analýza

Zabývá se problematikou sentimentu jednotlivých slov – slova každý člověk vnímá s určitým sentimentem, některá negativně, některá kladně. Počítač však vnímá všechny stejně, neutrálně. Snaží se přiřadit každému slovu určitý stupeň sentimentu.

Detekce a segmentace tématu dokumentu

Téma této bakalářské práce spadá právě do této kategorie úloh zpracování přirozeného jazyka. Detekcí tématu dokumentu rozumíme klasifikaci s předem známým počtem tříd. Segmentací témat je naopak myšleno učení bez učitele, kde z daného dokumentu se snažíme detekovat téma, resp. hledáme n slov, které nejlépe charakterizují daný dokument.

Detekce tématu dokumentu zahrnuje několik dalších úloh – stemming, lemmatizace, morfologická segmentace, segmentace slov, porozumění přirozenému jazyku a další. Dokumenty, na které může být aplikována klasifikace, jsou textové dokumenty, obrázky, zvukové či hudební soubory apod. Každý z těchto dokumentů se potýká se svými unikátními problémy. Tato práce se zabývá klasifikací textových dokumentů, které vznikly na základě anotace telefonních hovorů.

Existují 2 základní přístupy ke klasifikaci dokumentů – přístup na základě obsahu a přístup orientovaný na požadavky. Klasifikace na základě obsahu je klasifikace, kde váhy přiřazené určitým tématům určují, do jaké třídy bude dokument klasifikován. V automatické klasifikaci můžou být těmito tématy například počet určitých slov a na jejich základě bude určena klasifikace. Klasifikace orientovaná na uživatelské požadavky je klasifikace, která bere v potaz, jak by uživatel hledal daný dokument – jakou otázku/požadavek by systému poslal. Tomuto přístupu se někdy říká indexování dokumentů.

Detekce tématu má široké použití v dnešních aplikacích. Zejména:

- Používá se při filtrování spamu, kde se na základě obsahu, adresáta a dalších vlastností klasifikuje, že jde o spam a uživatele tak nezahrnují nechtěné zprávy – stále jde o klasifikaci, která není nikdy přesná a stává se, že někdy klasifikuje špatně,
- Další oblastí je identifikace jazyka, která automaticky přiřadí jazyk k danému dokumentu na základě jeho obsahu,
- Klasifikace žánru dokumentu,
- Sentimentální analýza – klasifikuje se „nálada“ řečníka či autora dokumentu na základě používaných slov,
- Mnoho dalších.

4. Klasifikace

Klasifikace jako taková je sloučení většího množství objektů do jednoho celku podle jejich společných vlastností [12]. Jako příklad můžeme uvést například rozdělení živočišné říše – zde rozlišujeme např.: obratlovce, bezobratlé, plazy, ptáky, savce apod. Pokud bychom měli klasifikovat následující 3 „objekty“: kočka, pes, kachna, tak je můžeme buď všechny klasifikovat do 1. třídy: obratlovci, nebo je také můžeme klasifikovat do dvou tříd: kočku a psa klasifikujeme spolu do jedné třídy savci a kachnu klasifikujeme do třídy ptáci.

Do jaké třídy jednotlivé objekty klasifikujeme, resp. do kolika tříd chceme klasifikovat, záleží na typu úlohy. V nějaké úloze by bylo potřeba zjistit, zda se jedná o obratlovce či nikoliv. V takovémto případě by nám postačila pouze třída obratlovci, v jiných úlohách bychom naopak potřebovali přímo zjistit, zda se jedná o savce či ptáka.

Aby však bylo vůbec možné nějak klasifikovat, musíme mít k dispozici nějaké měřitelné informace, tzv. příznaky (features). Tedy v úloze klasifikace si musíme nejprve určit, jaké vlastnosti jsou měřitelné, jestli nám tyto vlastnosti přináší podstatnou informaci o rozlišnosti tříd, jestli je výhodné tyto vlastnosti měřit – některé bývají složitěji měřitelné a přináší dobře rozlišitelný rozdíl mezi třídami, ale pořídit daný vzorkovač je drahé, tak se měří jiné vlastnosti, díky kterým se zlevní nároky na snímače za cenu zesložitění úlohy klasifikace. V případě této úlohy bychom mohli například rozlišovat, jestli dané zvíře má či nemá křídla, pro rozlišení, jestli se jedná o ptáka či savce. V případě této úlohy by nám tato jediná vlastnost stačila, avšak pokud bychom měli rázem klasifikovat ještě netopýra, který má křídla a zároveň je savec, pak by nám tato vlastnost nestačila a museli bychom mít k dispozici nějaké další.

U této úlohy byla předem známá informace o existenci tříd: savci, ptáci, obratlovci. Existují však úlohy, kde tuto informaci nemusíme mít. Tento případ může nastat, když na vstup dostaneme množinu dat, o kterých předem nic nevíme. Klasifikace se v tomto případě dělá tak, že se všechna data mezi sebou porovnávají a hledá se „rozumný“ práh, kdy řekneme, že porovnávané objekty jsou z různých tříd. Tomuto druhu klasifikace se říká unsupervised čili učení bez učitele.

V případu strojového učení chceme klasifikovat automaticky. Typické úlohy ve strojovém učení jsou takové, že dostaneme množinu dat, podle kterých se snažíme natrénovat klasifikátor, který v budoucnu bude automaticky umět správně klasifikovat podobná vstupní data. V případě této bakalářské práce jsou na vstupu otázky a odpovědi mezi Ústavem pro jazyk český a dotazujícími v podobě telefonních nahrávek – tyto otázky jsou rozřazené do několika tříd a na jejich základě bude natrénovaný klasifikátor umět klasifikovat i nové, zatím neviděné vstupy. Při procesu trénování se snažíme najít takový klasifikátor, který bude klasifikovat s co nejmenší chybou. Hledání optimálního klasifikátoru bývá velice často iterativní proces, při kterém se algoritmus postupně zlepšuje, učí se a tím se zmenšuje chyba klasifikace. Proces učení upravuje vnitřní parametry daného algoritmu.

Algoritmů, které realizují klasifikátory je celá řada a některé z nich budou rozebrány níže. Obecně lze rozdělit učení klasifikátorů na 2 druhy – učení s učitelem a učení bez učitele.

4.1. Učení bez učitele

Učením bez učitele nazýváme proces, kdy algoritmus na vstup dostane neznámá data a snaží se je roztřídit. Tomuto procesu se také říká shluková analýza – snažíme se ze získaných dat dostat informace o tom, jak jsou rozložené v prostoru.

Algoritmy, které spadají do kategorie učení bez učitele jsou například k-means, metoda maximin, hierarchické klastrování nebo například z neuronových sítí autoenkodéry.

4.2. Učení s učitelem

Učení s učitelem probíhá tak, že klasifikátoru předem řekneme, do kolika tříd budeme chtít klasifikovat. Klasifikátor následně dostane na vstup označená data (o těchto datech ví, do jaké třídy mají patřit), na kterých si natrénuje své parametry. Po natrénování dostane klasifikátor na vstup neznámá data, která se pokusí klasifikovat a na těchto datech bude vyhodnocena jeho percentuální úspěšnost.

Algoritmy implementující učení s učitelem jsou například Support Vector Classification (SVC), lineární diskriminační analýza (LDA) a Neuronové Sítě. Tyto algoritmy budou podrobně rozebrány v dalších kapitolách.

4.3. Maximalizace úspěšnosti klasifikace

Volba klasifikátoru je zásadní pro dosažení co nejlepší úspěšnosti klasifikace. Volba správného klasifikátoru se odvíjí od typu úlohy a neexistuje konkrétní návod, jak určit, který klasifikátor bude poskytovat nejlepší výsledky. Nejčastěji je volba klasifikátoru expertní záležitost, kdy expert na základě svých předchozích zkušeností řekne, který klasifikátor by měl poskytovat nejlepší výsledky. Ve spoustě úlohách se také zkouší více klasifikátorů pro různá nastavení parametrů a porovnávají se jejich výsledky.

Doba trénování klasifikátoru je další velice důležitá složka. Pokud bychom nechali klasifikátor trénovat se příliš krátko, pravděpodobně bude mít špatně nastavené parametry a pokud bychom ho takto použili, bude poskytovat špatné výsledky. Tomuto problému se říká podtrénování (underfitting). Také nemůžeme nechat trénovat klasifikátor příliš dlouho, neboť ačkoliv by mohl klasifikovat vzorky z trénovací množiny s celkovou úspěšností blížící se 100 %, jakmile bychom mu dali na vstup nová data, která nikdy neviděl, začne dávat špatné výsledky. Tento problém nazýváme přetrénování (overfitting).

Pro trénování klasifikátoru se obvykle rozdělují data na 3 množiny: trénovací, testovací a development množinu. Trénování, resp. prvotní nastavení vnitřních proměnných se provádí na trénovací množině, která musí být z těchto tří množin nejobsáhlejší. Trénování obvykle probíhá v několika cyklech. Development množina se používá pro vyhodnocení úspěšnosti jednotlivých cyklů a pro adekvátní upravení vah. Po natrénování proběhne konečné vyhodnocení úspěšnosti klasifikátoru na testovací množině dat, které daný klasifikátor ještě neviděl.

Při rozdělení na trénovací, testovací a development množinu se musí dávat pozor, aby nikdy nenastal případ, že testovací množina bude obsahovat nikdy neviděnou třídu. Pokud by se v testovací množině dat vyskytovala třída, pro kterou klasifikátor nebyl natrénován, klasifikátor bude vyhodnocen jako velice špatný, tedy s nízkou úspěšností klasifikace. Musí být zajištěno, že data budou rozdělena rovnoměrně, ve stejném poměru pro každou třídu. Každá třída tak musí obsahovat přibližně stejné množství dat. Třídy s nedostatkem dat bychom měli vyřadit a prohlásit o nich, že neobsahují dostatek dat pro natrénování klasifikátoru. U tříd s velkým množstvím dat by se měla zredukovat, aby třídy byly vzájemně vyvážené. Pokud bychom měli třídu, která by obsahovala řádově více dat, klasifikátor by byl schopen správně klasifikovat převážně pouze do této třídy.

Vyhodnocení klasifikátoru se provádí takzvanou křížovou validací (cross-validation) – data se náhodně rozdělí na trénovací a testovací množinu a na testovací množině se vyhodnocuje úspěšnost klasifikacce.

Mezi další způsoby vyhodnocení patří například k-fold. Tento způsob je obdobný křížové validaci, avšak budeme mít k podmnožin z jedné množiny dat, tedy proces křížové validace je k-krát opakován. Pro dosažení přesnějších výsledků se tento proces provádí několikrát (jak pro křížovou validaci tak pro k-fold) a výsledky se následně průměrují klasickým aritmetickým průměrem.

4.4. Míry vyhodnocení klasifikace

Úspěšnost klasifikace též závisí na zvolené míře, kterou se buď snažíme minimalizovat (např. hodnotu ztrátové funkce – loss) nebo maximalizovat (např. přesnost klasifikace – accuracy) [17]. Zvolení míry závisí na typu úlohy. Například pokud bychom měli úlohu, kde chceme klasifikovat, zda člověk má rakovinu či nikoliv, musíme zvolit jinou míru, než přesnost klasifikace. To je z důvodu, že máme velice nevyváženou trénovací množinu dat – je značně vyšší množství lidí bez rakoviny než s rakovinou.

Tabulka 4.1: Matice záměn

Třída	**Klasifikace: Negativní**	**Klasifikace: Pozitivní**
Negativní	TN	FN
Pozitivní	FP	TP

TN – True Negative, počet správně klasifikovaných do třídy „Negativní“
FN – False Negative, počet špatně klasifikovaných do třídy „Negativní“
FP – False Positive, počet špatně klasifikovaných do třídy „Pozitivní“
TP – True Positive, počet správně klasifikovaných do třídy „Pozitivní“

Většina měr vyhodnocení vychází z tabulky 4.1.

Při klasifikaci se obvykle používají následující 4 míry vyhodnocení:

Přesnost:

$$accuracy = \frac{TN + TP}{TN + TP + FN + FP} \quad (1)$$

Preciznost:

$$precision = \frac{TP}{TP + FP} \quad (2)$$

Úplnost:

$$recall = \frac{TP}{TP + FN} \quad (3)$$

F1 skóre:

$$F1 = 2 \cdot \frac{precision * recall}{precision + recall} \quad (4)$$

Preciznost udává, jak přesně se klasifikovali pozitivní jedinci, tedy v případě příkladu s rakovinou ti, kteří ji skutečně měli. Tato míra je dobrá, pokud je vysoká cena špatně klasifikovaných do pozitivní třídy – tedy pokud by člověk, který rakovinu nemá byl přesto klasifikován, že ji má.

Úplnost klasifikace se naopak používá, pokud je vysoká cena FP – tedy těch, co byly klasifikováni, že rakovinu nemají, přestože ji ve skutečnosti mají. Právě tato míra by byla vhodná pro případ s rakovinou.

V některých případech se hledá vyvážení mezi precizností a úplností klasifikace. V takovém případě se používá míra zvaná F1 skóre.

Při klasifikaci se také může sledovat hodnota ztrátové funkce (loss). V některých případech se může stát, že ačkoliv hodnota ztrátové funkce se stále zmenšuje, přesnost klasifikace se nijak nezlepšuje.

5. Předzpracování vstupních dat

Předtím, než klasifikátoru poskytneme data [10], je nutno je nejprve předzpracovat. Předzpracováním připravíme vstupní data pro klasifikátor, neboť ten dostává jako vstup pouze číselné hodnoty. Proto musíme nejprve vstupní hodnoty parametrizovat.

5.1. Odstranění šumu

Odstraněním šumu obvykle myslíme například odstranění metadat ze souboru, získání čistého textu z HTML, XML či JSON formátu, odstranění hlavičky/patičky souboru, apod. Při zpracování řeči se odstraňují zbytečné informace jako například hluk v pozadí – například šum větráku, hluk z ulice atd.

5.2. Tokenizace

První krok, který se při předzpracování textu dělá je tzv. tokenizace. Tokenizace je krok, který rozděluje větší celky textu na menší části, tokeny. Větší části jsou rozděleny na věty, věty dále na slova, apod. Tokenizaci se někdy říká také segmentace textu či lexikální analýza. Avšak segementací textu se spíše rozumí rozdělení dlouhého textu na menší celky, kdežto u tokenizace vyžadujeme rozdělení až na samotná slova.

5.3. Normalizace

Před dalším zpracováním textu musí být text normalizován. Normalizací textu se obvykle rozumí několik následujících kroků:

- Převedení textu na stejná písmena (velká či malá)
- Odstranění interpunkce, bílých mezer
- Převedení čísel na jejich slovní ekvivalent
- Převedení slov na jejich kořenový tvar (odstranění sufixů, prefixů, apod.)
- Lemmatizace a odstranění stop words

Lemmatizace převede jednotlivá slova na jejich základní tvar. Například slovo „lepší“ by se zlemmatizovalo na „dobrý“. Lemmatizace se provádí za účelem zlepšení výsledků klasifikátoru. Nicméně ne vždy přináší lepší výsledky, takže se někdy tento krok předzpracování vypouští.

Stop words definují seznam slov, která se při parametrizaci TFIDF úplně vynechají. Mezi tato slova patří zejména nějaká obvyklá, často se vyskytující slova. Již z definice TFIDF by slova s nízkým výskytem měla velice nízkou váhu, nicméně pokud jsou tato slova vyjmenována ve stop words, pak ve výstupu vůbec nejsou zahrnuta, což redukuje paměťové nároky na výpočet.

5.4. Parametrizace

V tomto bodě máme upravený text do podoby, kdy jej můžeme transformovat na číselné vektory. Např. pomocí TFIDF transformace, word2vec či doc2vec přístupů.

5.4.1. Batoh slov

Batoh slov (Bag of Words - BoW) [6] je základní typ parametrizace dokumentů. Tento typ parametrizace pouze spočte četnost každého slova v daném dokumentu. Výsledkem pro jeden dokument bude tedy vektor, obsahující četnost každého slova. Z principu BoW vychází většina dnešních metod – například u parametrizace TFIDF je BoW součástí parametru TF.

5.4.2. Parametrizace TFIDF

Termín TFIDF označuje Term Frequency – Inverse Document Frequency [6]. Z názvu metody je patrné, že se tato metoda skládá ze 2 složek, TF a IDF. TF složka vyjadřuje, jak často se výraz vyskytuje v 1 dokumentu. Hodnota TF se často normalizuje na počet slov vyskytujících se v daném dokumentu, tedy:

$$TF = \frac{n}{N} \tag{5}$$

n – celkový výskyt daného slova v jednom dokumentu
N – celkový počet všech slov v daném dokumentu

IDF složka vyjadřuje, jak moc velkou má slovo váhu napříč všemi dokumenty. Čím častěji se dané slovo vyskytuje ve všech dokumentech, tím menší má váhu, neboť často vyskytující se slova ve všech dokumentech nám neposkytují žádnou podstatnou informaci o tématu daného dokumentu. IDF vypočteme podle následujícího vzorce:

$$IDF = \log\left(\frac{|DOC|}{N_{DOC}}\right) \tag{6}$$

$|DOC|$ - celkový počet všech dokumentů
N_{DOC} – celkový počet dokumentů, kde se vyskytuje dané slovo

Celková „hodnota“ slova podle TF-IDF bude tedy:

$$TFIDF = TF * IDF \tag{7}$$

Vzorce pro výpočet TFIDF bývají někdy různě obměňovány. Jedna z obměn je, že se upravuje složka TF do binární podoby, tedy TF bude mít hodnotu rovno 1, pokud se slovo vyskytuje v dokumentu. 0 pokud ne. Další možností obměny TF je čistý počet výskytu slova v dokumentu, tedy nenormalizuje se na počet všech slov. Někdy se IDF složka úplně vypouští, a to tak, že se její hodnota zafixuje na 1.

Při implementaci TFIDF jsou zde i další parametry, které ovlivňují výsledný vektor TFIDF. Mezi tyto patří například definování, jak často se slovo mezi dokumenty může vyskytovat a to jak maximálně, tak minimálně. Pokud se tedy nějaké slovo vyskytuje mezi všemi dokumenty více, než je předem

definovaná hodnota, pak toto slovo ve výstupu TFIDF nebude zahrnuto – vyskytuje se příliš často na to, aby nám o dokumentu prozradilo jeho téma. Obdobně to platí pro málo se vyskytující slova – pokud se v dokumentech vyskytuje méně, než nějaká definovaná hodnota, pak opět nepřináší žádnou podstatnou informaci o textu a můžeme jej vyřadit. Tato úprava navíc značně snižuje paměťovou náročnost celého procesu. Při definici maximálního a minimálního počtu výskytu slov musíme zajistit, abychom skutečně neodstranili z textu slova s podstatnou informační hodnotou. Pokud by se tak stalo, projeví se to na úspěšnosti klasifikace. Ideální případ je takový, že máme ohodnocené co největší množství slov, avšak při obrovském množství dat narážíme na technická omezení výpočetní paměti zařízení, na kterém algoritmus běží.

5.4.3. Parametrizace word2vec

Většina parametrizačních metod využívá princip BoW [6], avšak často poskytují pouze průměrné výsledky, jelikož tento princip nezohledňuje například pořadí slov. Reprezentace pomocí word2vec vychází z 2 hlavních algoritmů: CBoW (Continuous Bag of Words) a Skip gram modelu. Při tomto přístupu se navíc zohledňuje význam jednotlivých slov – čili synonyma nebo analogie slov si jsou při reprezentaci blíže.

CBoW

CBoW vytváří posuvné okno okolo každého slova, čímž zohledňuje kontext tohoto slova. Každé slovo je reprezentováno jako vektor příznaků. Při tomto přístupu se snažíme z kontextu slov predikovat hledané slovo.

Skip gram

Skip gram je opakem CBoW. Při vstupu 1 slova se algoritmus snaží predikovat jeho okolí. Tento přístup je mnohem pomalejší než CBoW, avšak bývá přesnější pro méně se vyskytující slova.

5.4.4. Parametrizace doc2vec

Cílem doc2vec [6] je vytvořit číselnou reprezentaci dokumentu a vychází z principů word2vec. Algoritmus doc2vec modeluje číslenou reprezentaci všech dokomentů v daném korpusu, zatímco word2vec modeluje reprezentaci každého slova v daném korpusu. Doc2vec vychází ze 2 přístupů:

Distributed Memory version of Paragraph Vector (PV-DM)

Metoda doc2vec založená na PV-DM je rozšířený CBOW o 1 další příznak, kterým je ID dokumentu, ve kterém se dané slovo nachází. Tento model nám navíc dává informaci o zbytku, který chybí v aktuálním kontextu slova – vektory slov reprezentují koncept těchto slov, zatímco přidané vektory ID dokumentů, reprezentují jejich koncept.

Distributed Bag of Words version of Paragraph Vector (PV-DBOW)

Obdobně jako u word2vec i zde existuje druhý přístup, který je obdobný skip gramu. Tento algoritmus narozdíl od word2vec je rychlejší a má menší paměťové nároky, jelikož zde nemusíme uchovávat vektory slov. Rozdílem od word2vec je, že na vstupu je vektor paragrafu, ze kterého algoritmus predikuje slova v malém okně.

Ve většině úloh postačuje použít PV-DM přístup, avšak kombinace PV-DM s PV-DBOW dává lepší výsledky pro složitější úlohy.

Velkou nevýhodou doc2vec parametrizace je, že při procesu parametrizace musí vidět veškerá data, tedy jak trénovací tak testovací množinu dat. Pokud tedy přijdou na vstup úplně nová data, doc2vec se musí podívat na úplně všechna data, aby parametrizoval správně.

6. Klasifikátory

Klasifikátor je algoritmus, který pro daný vstup určí jeho třídu – klasifikuje. Ve strojovém učení existuje celá řada klasifikátorů. Tato práce se zabývá použitím třech klasifikátorů založených na podpůrných vektorech, lineární diskriminační analýze a neuronových sítí.

6.1. Support Vector Machines

Support Vector Machines (SVM) je metoda učení s učitelem používaná zejména v klasifikaci a regresní analýze [8]. Metoda SVM má za cíl nalézt optimální nadrovinu rozdělující prostor příznaků, tedy takovou nadrovinu, která má maximální pásmo necitlivosti. Maximální pásmo necitlivosti reprezentuje vzdálenost mezi nejbližšími body separovaných tříd. Čím větší bude maximální pásmo necitlivosti, tím nižší je šance špatné klasifikace. Abychom splnili tuto podmínku, postačí nám pouze body ležící na okrajích pásem - tzv. podpůrné vektory (support vectors). Díky použití značně menšího množství bodů pro odhad rozdělující nadroviny klesá výpočetní náročnost pro veškeré algoritmy využívající metodu SVM.

Původem jde o metodu rozdělující 2 třídy, avšak metoda lze zobecnit pro n tříd tak, že porovnáváme každou třídu s každou – v takovémto případě budeme mít $\frac{n\cdot(n-1)}{2}$ rozdělujících nadrovin. Klasifikátor následně zvolí tu třídu, do které byl bod přiřazen nejčastěji.

Nadrovina rozděluje prostor lineárně, avšak většina souborů dat není přirozeně lineárně separabilní. Proto je důležitou součástí SVM tzv. jádrová transformace, která přemění lineárně neseparovatelnou množinu dat na lineárně separabilní, typicky tím, že transformuje data do vyšší dimenze příznaků. Optimální funkčnost SVM tedy velice záleží na volbě jádrové transformace.

6.1.1. Lineární SVM

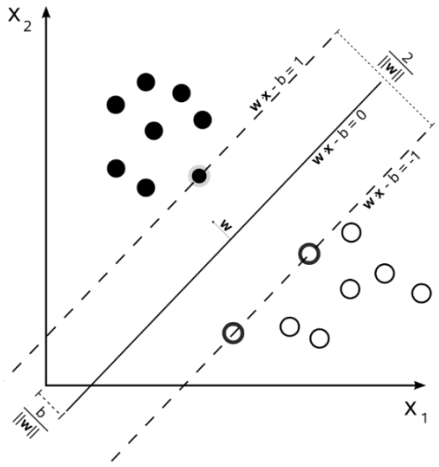
V lineárně separabilním případě se nabízí nekonečně mnoho řešení, jak lineárně oddělit dvě třídy. Abychom našli optimální nadrovinu, využijeme následující rovnice, kde se snažíme minimalizovat $\|w\|$:

$$w \cdot x - b = 0 \tag{8}$$

x – množina všech bodů (podpůrných vektorů) definovaných jako dvojice (x_i, y_i), kde x_i jsou podpůrné vektory a y_i je informace od učitele – nabývá hodnot 1 nebo -1 v závislosti na tom, do které třídy bod skutečně patří
w – normálový vektor rozdělující nadroviny
b – posun nadroviny od středu souřadnic

Šířku pásma dostaneme po dosazení:

$$\frac{2}{\|w\|} \tag{9}$$



Obr. 1) Lineárně separabilní případ SVM[1]

Pro všechny body musí tedy platit:

$$w \cdot x_i - b \geq +1, \text{ pro } y_i = +1 \quad (10)$$

$$w \cdot x_i - b \leq -1, \text{ pro } y_i = -1 \quad (11)$$

V neseparabilním případě do rovnice zavádíme proměnnou, která nám určuje chybu klasifikace. Tuto chybu se snažíme co nejvíce zmenšit.

Rovnice pro neseparabilní případ budou tedy vypadat následovně:

$$w \cdot x_i - b \geq +1 - \xi_i, \text{ pro } y_i = +1 \quad (12)$$

$$w \cdot x_i - b \leq -1 - \xi_i, \text{ pro } y_i = -1 \quad (13)$$

$$\xi_i \geq 0, \text{ pro } \forall i$$

6.1.2. Nelineární SVM

Jelikož většina dat není lineárně separabilní, využívá se zejména nelineární SVM. Nelineární SVM využívá nelineární jádrovou transformaci, která převede množinu bodů do vyššího prostoru, kde již lineárně separabilní budou.

Existuje několik druhů jádrových transformací. Mezi nejčastěji používané patří například:

- Polynom stupně p
- Radiální bázová funkce (Gausovská či Laplaceovo)
- Hyperbolický tangens
- Sigmoid
- Funkce po částech lineární

[1] Wikimedia Commons contributors, 'File:Svm max sep hyperplane with margin.png', *Wikimedia Commons, the free media repository*, 4 December 2016, 22:22 UTC, <https://commons.wikimedia.org/w/index.php?title=File:Svm_max_sep_hyperplane_with_margin.png&oldid=225162099> [accessed 15 May 2019]

Při klasifikaci textu se často používá funkce po částech lineární, neboť si dokáže dobře poradit s velkými vektory [8], jak ukazují i experimenty v této práci (viz kapitola 7.7).

Předpis po částech lineární funkce:

$$k(x,y) = 1 + x \cdot y + x \cdot y \cdot \min(x,y) - \frac{x+y}{2} \cdot \min(x,y)^2 + \frac{1}{3} \cdot \min(x,y)^3 \quad (14)$$

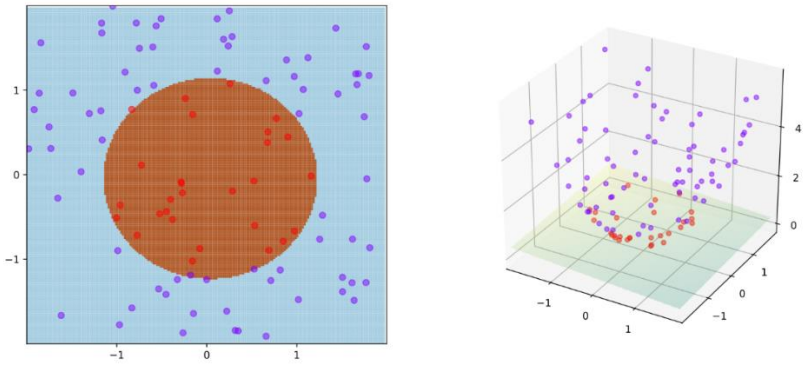
$k(x,y)$ – jádrová funkce (transformace)
x – podpůrné vektory
y – informace od učitele o původu vektoru

6.1.3. Kernel trick

Další vhodnou jádrovou transformací je tzv. kernel trick (jádrový trik). Jádrový trik spočívá v tom, že všechna data mezi sebou vynásobí. Příklad jádrového triku:

$$k(x,y) = x \cdot y + x^2 \cdot y^2 \quad (15)$$

Vhodnou ukázkou je příklad, kdy v 2-dimenzionálním prostoru máme 2 třídy, avšak tyto 2 třídy nelze lineárně rozdělit (třídy by rozdělovala kružnice, viz. obrázek 2). Jádrový trik tato data převede do 3-dimenzionálního prostoru, kde data již lze snadno lineárně rozdělit.



Obr. 2) Příklad jádrového triku (vlevo původní data, vpravo po jádrovém triku)[2]

[2] Wikimedia Commons contributors, 'File:Kernel trick idea.svg', *Wikimedia Commons, the free media repository*, 11 August 2017, 18:03 UTC, <https://commons.wikimedia.org/w/index.php?title=File:Kernel_trick_idea.svg&oldid=255062995> [accessed 15 May 2019]

Původní formulace problému (15) se častěji převádí na Lagrangeovskou formulaci úlohy. Lagrangeova funkca má následující předpis:

$$L_D = \sum_i \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j \cdot y_i y_j \cdot x_i x_j \tag{16}$$

λ_i – nezáporné Lagrangeovy multiplikátory (jeden pro každou nerovnost (12), (13))

x – podpůrné vektory

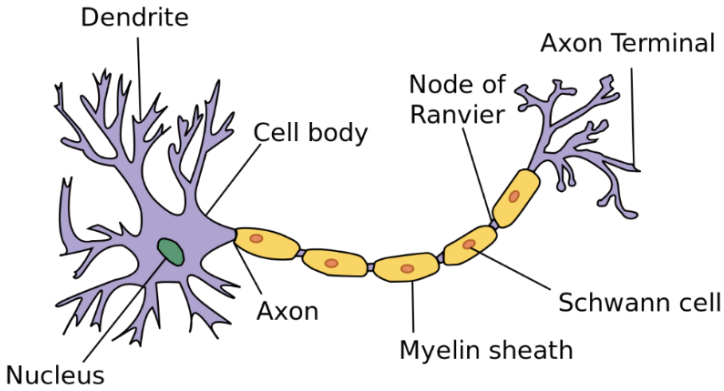
y – informace od učitele o původu vektoru

Trénování SVM probíhá hledáním maxima L_D – určení parametrů λ_i.

6.2. Neuronové sítě

6.2.1. Úvod

Umělé neuronové sítě (ANN) jsou v dnešní době jeden z nejpopulárnějších způsobů řešení problémů strojového učení [13]. Jsou inspirovány poznatky o schopnostech neuronů a nervových sítí živých organismů. Mezi tyto schopnosti patří zejména: extrakce a reprezentace závislostí v datech, které nejsou na první pohled zřejmé; řešení silně nelineárních úloh; schopnost učit se a zevšeobecňovat.



Obr. 3) Lidský neuron[3]

Pro ilustraci je zde přiložen obrázek příkladu lidského neuronu. Dendrity představují vstupní vrstvu neuronu, skrz ně přichází do neuronu veškeré informace. Neuron dále obsahuje Axon, který tvoří výstupní vrstvu neuronu. Ta zajišťuje odeslání zpracované informace do dalšího neuronu.

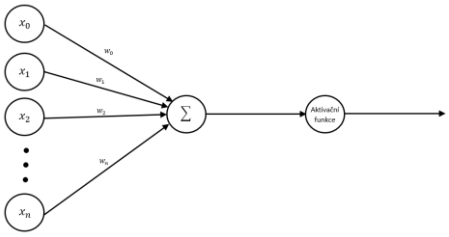
Neuronová síť je výsledkem spojení jednotek nazývaných umělý neuron, resp. perceptron. Propojením si neurony mohou vzájemně předávat a zpracovávat informace. Neuronové sítě jsou typicky organizované do vrstev neuronů. Různé vrstvy mohou vykonávat různé transformace vstupů, mohou používat jiné aktivační funkce čili nesou různé významy. Signál je pak postupně přenášen od vstupní vrstvy, přes veškeré další až k poslední, výstupní vrstvě. V některých případech se signál opakovaně vrací zpět, než neuronová síť poskytne výstup.

V dnešním době jsou trendem především hluboké neuronové sítě (DNN). Tyto sítě jsou prosté neuronové sítě se značným množstvem vrstev mezi vstupní a výstupní vrstvou. DNN umožňují modelování komplexních nelineárních vztahů mezi objekty. DNN jsou sítě typu „feedforward“ – vstupní data putují od vstupní vrstvy až po výstupní bez vracení se zpět.

[3] Wikimedia Commons contributors, 'File:Neuron.svg', *Wikimedia Commons, the free media repository*, 17 March 2019, 18:45 UTC, <https://commons.wikimedia.org/w/index.php?title=File:Neuron.svg&oldid=343028396> [accessed 15 May 2019]

6.2.2. Perceptron

Základem neuronových sítí ve strojovém učení je perceptron.



Obr. 4) Perceptron

$X\ (x_0, x_1, x_2 \dots x_n)$ – vstupní vektor
$W\ (w_0, w_1, w_2 \dots w_n)$ – váhový vektor

Do každého perceptronu vstupuje vektor o velikosti n rozšířený o x_0, tzv. bias. Každá ze složek vektoru je přenásobena svou vlastní vahou, kterou má přiřazenou perceptronem. Na přenásobené vstupy je následně aplikována aktivační funkce. Výstup neuronu se spočte podle následující rovnice:

$$y = \varphi\left(\sum_{i=0}^{n} x_i \cdot w_i\right) \tag{17}$$

φ – aktivační funkce

6.2.3. Aktivační funkce

Aktivační funkce představuje potenciál spuštění dané buňky, resp. daného neuronu. V nejednodušší formě reprezentuje pouze binární hodnotu, zda byl neuron aktivován. Aktivace či deaktivace jednoho neuronu na určitý vstup následně ovlivňuje spuštění dalších neuronů. Je zřejmé, že v případě pouhého binárního výstupu ztrácíme značnou hodnotu informace poskytovanou dalším neuronům. Proto existuje řada funkcí, které hodnotu spuštění neuronu definují na nějakém intervalu.

Každá z aktivačních funkcí má své výhody i nevýhody. Základním rozdělením aktivačních funkcí je: linearita, konečnost oboru hodnot dané funkce, existence spojité derivace, monotónnost a další.

Mezi nejčastěji používané aktivační funkce patří zejména:

- Sigmoid
- Hyperbolický tangens
- ReLU
- Softmax

Neuronová síť bez aktivační funkce je pouhým regresním modelem.

Sigmoid

Aktivační funkce sigmoid bývala a stále je velice populární aktivační funkcí. Mezi její výhody patří, že její derivace je definována na celé množině reálných čísel, funkční hodnoty se nachází v rozsahu $(0, 1)$. Mezi nevýhody patří, že při vyšších vstupních hodnotách se funkční hodnota velice málo mění – dochází k problému zvanému Vanishing Gradient (viz. kapitola 6.2.5) při kterém se značně prodlužuje doba trénování celé neuronové sítě.

Používá se v úlohách, kde výstupem je pravděpodobnost či binární rozhodnutí. (např. pravděpodobnost, zda má pacient rakovinu; zda se na obrázku nachází zvíře či nikoliv).

Předpis funkce:

$$\varphi(z) = \frac{1}{1 + e^{-z}} \tag{18}$$

Hyperbolický tangens

Hyperbolický tangens je aktivační funkce velice podobná funkci sigmoid. Opět je definována na celém intervalu, všude je derivovatelná a při vyšších vstupních hodnotách se funkční hodnoty velice málo mění. Funkčních hodnot nabývá hyperbolický tangens v rozsahu $(-1, 1)$, tím pádem je tato aktivační funkce souměrná a v nule nabývá funkční hodnoty 0. Používá se zejména v úlohách klasifikace mezi 2 třídy.

Předpis funkce:

$$\varphi(z) = \frac{2}{1 + e^{-2z}} - 1 \tag{19}$$

Rectified Liner Unit (ReLU)

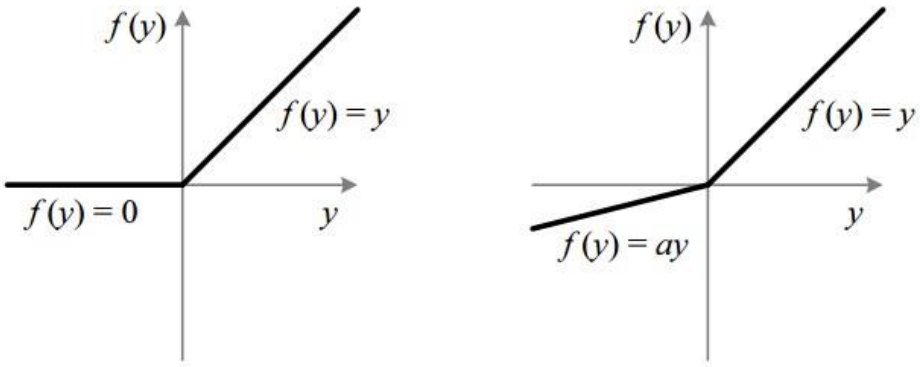
Aktivační funkce ReLU je v dnešní době nejpoužívanější aktivační funkcí z důvodu, že je používána ve většině konvolučních (CNN) či hlubokých (DNN) neuronových sítí [13]. Tato aktivační funkce nemá definovanou derivaci v nule a nabývá funkčních hodnot na intervalu $< 0, +\infty)$.

Nevýhodou této aktivační funkce je, že všechny vstupní hodnoty nižší nebo rovny 0 jsou automaticky namapována na 0. Z tohoto důvodu se nemůže neuron správně naučit, jak reagovat na záporný vstup.

Předpis ReLU funkce:

$$\begin{aligned} \varphi(z) &= 0 \quad \text{pro } z < 0 \\ \varphi(z) &= z \quad \text{pro } z \geq 0 \end{aligned} \tag{20}$$

Existují modifikované verze ReLU funkce (například tzv. Leaky ReLU), které se snaží tento problém vyřešit tím, že funkci na intervalu $(-\infty, 0 >$ změní například na $\varphi(z) = a \cdot z$, kde a se obvykle volí rovno $0{,}01$.



Obr. 5) ReLU vs Leaky ReLU funkce (nalevo ReLU, napravo Leaky ReLU)

$f(y)$ – hodnota výstupu neuronu
y – vstupní hodnota neuronu
a – volená proměnná pro Leaky ReLU

Softmax

Aktivační funkce softmax se často používá v poslední vrstvě neuronové sítě při klasifikačních úlohách do více tříd. Tato vrstva obsahuje takový počet neuronů, jaký je počet tříd, do kterých chceme klasifikovat. Pro každou třídu je následně vypočtena pravděpodobnost a klasifikátor následně vybere tu nejvyšší.

6.2.4. Učení neuronové sítě

Právě díky možnosti učení si získaly neuronové sítě tolik zájmu. Učení zahrnuje definice ztrátové funkce, kterou se snažíme minimalizovat. Ztrátová funkce nám poskytuje informaci, jak je nastavení neuronové sítě daleko od optimálního nastavení. Učící algoritmy prohledávají prostor výsledků, aby nalezly nejnižší možnou hodnotu ztrátové funkce. Volba konkrétní ztrátové funkce se odvíjí od typu úlohy, zda se jedná o učení s učitelem či bez učitele a na typu zvolené aktivační funkce.

Aby neuronová síť poskytovala dobré výsledky, musí dojít k její natrénování. Natrénováním neuronové sítě je myšleno nastavení vah W pro každý neuron z celé neuronové sítě. Způsobů natrénování neuronové sítě je několik. Mezi často používané patří algoritmus Gradient Descent či optimizační algoritmus zvaný *adam*.

Backpropagation

Neuronové sítě mohou být natrénovány skrze standartní zpětnou propagaci. Algoritmus zpětné propagace je definován jako výpočet gradientu ztrátové funkce a následným upravením vah. Úprava vah může být uskutečněna skrze algoritmus stochastický klesající gradient (SGD) definované následovně:

$$w_{i,j}(t+1) = w_{i,j}(t) - \mu \frac{\partial C}{\partial w_{i,j}} \tag{21}$$

μ – parametr učení
C – ztrátová funkce
$w_{i,j}(t)$ – váha j-tého neuronu v i-té vrstvě v cyklu t

Při implementaci neuronové sítě ovlivňuje zpětnou propagaci parametr *batch_size*. Tento parametr definuje počet prvků, které budou propagované v síti. Pokud bychom měli počet trénovacích dat vyšší než zvolený parametr, neuronová síť bude postupně brát data z trénovací množiny o velikosti *batch_size*. Úprava parametrů neuronové sítě tedy proběhne vícekrát, avšak po menších hodnotách. Pokud bychom očekávali veliký vstup, tímto parametrem omezíme počet najednou zpracovávaných dat.

Algoritmus trénování neuronové sítě

1. Dopředná propagace je prvním krokem algoritmu trénování neuronové sítě. Neuronová síť na vstup dostane zparametrizovaná data a každý neuron na ně nějakým způsobem reaguje podle zvolené aktivační funkce. Hodnota výstupu neuronu se následně propaguje dalším neuronům.
2. Následně je vypočtena hodnota ztrátové funkce.
3. Dále se zpětně propagují výstupy aktivačních funkcí napříč celou neuronovou sítí.
4. Podle vstupů ze zpětné propagace jsou adekvátně upravené váhy. Upravení vah se navíc řídí nastavitelným parametrem μ, který ovlivňuje rychlost konvergence k optimálním hodnotám.

Optimální nastavení učícího parametru

Nastavení učícího parametru μ je velice důležitou součástí procesu učení. Pokud by hodnota μ byla příliš vysoká, docházelo by k příliš velkým změnám nastavení parametru, které by vedly k možnému minutí optimální hodnoty. Algoritmus by tak divergoval a optimální hodnotu by nenašel. Pokud by naopak byl učící parametr nastaven na příliš nízkou hodnotu, doba konvergence by se značně prodloužila a celý proces se značně zpomalil, což je opět nežádoucí.

Jednou z možností, která se pro řešení tohoto problému využívá, je adaptivní nastavení učícího parametru. Při tomto nastavení se zohledňuje několik faktorů, zejména změna nastavení v předchozím kroce.

6.2.5. Problémy s ANN

Upravení vah spočívá ve vypočtení gradientu dané chyby. Pokud byla zvolena aktivační funkce sigmoid či hyperbolický tangens, bude pravděpodobně docházet k problému zvaném Vanishing Gradient. Při velkých vstupních hodnotách dochází k malým změnám funkčních hodnot, tím pádem je gradient nízký až téměř nulový, takže dochází k velice malé změně v nastavení váhy, tím pádem k velice malému zlepšení neuronové sítě. Tento problém se zhoršuje, čím dále se tato hodnota propaguje – tedy vrstvy až úplně na začátku neuronové sítě se budou trénovat nejdéle. Vanishing Gradient tak velice prodlužuje dobu trénování neurnové sítě. Pokud však zvolíme jinou aktivační funkci, urychlíme tak celý proces trénování. I přes existenci tohoto problému existují případy, kdy se stále vyplatí tyto aktivační funkce používat, neboť dosahují stále velice dobých výsledků.

Velké a efektivní neuronové sítě vyžadují značné výpočetní zdroje [13]. Pokud na vstupu dostane neuronová síť velké množství dat, doba trénování se značně prodlužuje. Aby mohla být neuronová síť aplikována v realtime aplikacích, je zapotřebí zajistit dostatečně výkonný hardware.

Dalším problémem neuronových sítí je, že nejsme schopni říct proč mají neurony zrovna takové parametry – nevíme, který neuron, co a jak hodnotí; neuronová síť se chová jako black-box model, resp. gray box, jelikož definujeme počet vrstev, neuronů, volíme aktivační funkce atd. Kvůli tomuto problému je velice obtížné debuggovat neuronovou síť.

Neuronové sítě se také potýkají s problémy existencí lokálních minim či sedlových bodů [13]. Pokud je počáteční nastavení zvoleno nevhodně, neuronová síť může dokonvergovat pouze k lokálnímu minimu, které může být značně vyšší než globální minimum. Také se může stát, že se při počátečním nastavení „trefí“ do sedlového bodu – v tomto bodě se při iteracích nastavení neuronové sítě vůbec nemění (tento případ je však velice nepravděpodobný v mnoha-dimenzionálních prostorech).

6.2.6. Druhy neuronových sítí

Existuje řada neuronových sítí. Hlavním rozdělením neuronových sítí je na konvoluční a rekurentní neuronové sítě. Další druhy jsou obměnou těchto sítí.

Konvoluční neuronové sítě (CNN)

Konvoluční neuronové sítě jsou třída hlubokých neuronových sítí složené z jedné nebo více konvolučních vrstev, které jsou plně propojené [13]. CNN jsou vhodné pro zpracování vizuálních či jiných dvoudimenzionálních dat (např. rozpoznávání řeči). Tyto sítě mohou být natrénovány skrze zpětnou propagaci a jsou snadnější na natrénování než jiné DNN z důvodu nízkého počtu odhadovaných parametrů.

Dekonvoluční síť

Tento druh sítí je inverzí konvolučních sítí. Zatímco konvoluční sítě přijímají na vstup rozsáhlý vektor a snaží se ho klasifikovat (př. obrázek kočky je vstup, výstupem je, že je rozpoznána), tak

dekonvoluční sítě dostanou na vstupu omezený vektor a snaží se z něj vygenerovat adekvátní výsledek (př. vstupem je „kočka“ a výstupem je její obrázek).

Auto enkodéry

Auto enkodéry jsou typem konvolučních neuronových sítí a používají se v klasifikaci, klastrování a redukci příznaků. Princip auto enkodérů spočívá v hledání optimální reprezentace vstupních dat na jejímž základě lze vygenerovat původní vstup.

Variační auto enkodéry

Na rozdíl od běžných auto enkodérů, VAE komprimují pravděpodobnosti místo příznaků. Rozdíl by se dal popsat tak, že auto enkodéry jsou odpovědí na otázku „jak můžeme data generalizovat?“, zatímco VAE odpovídají na „jak silné je propojení mezi dvoumi událostmi; jsou kompletně nezávislé, nebo je mezi nimi nějaká spojitost?“.

Odšumující auto enkodéry

U auto enkodérů se někdy stává, že místo aby nalezly co nejrobustnější příznaky, adaptují se na původní data, což vede k problému přetrénování. Odšumující autoenkodéry přidávají na vstup šum, náhodně prohazují data apod. Tímto zajišťují, aby auto enkodér více generalizoval vstupy, neboť se nikdy nemůže natrénovat přesně na náhodný šum.

Rekurentní neuronové sítě (RNN)

RNN jsou druhem DNN, kde data mohou téct libovolným směrem, tj. mohou se vracet libovolně zpět. Tento druh sítí bývá zpravidla používán pro jazykové modelování.

- Long short-term memory (LSTM)

  Neuronové sítě LSTM jsou druhem rekurentních neuronových sítí, které eliminují Vanishing Gradient [15]. Běžný neuron v LSTM je obohacen o vstupní, výstupní a zapomínací bránu. Neuron si pamatuje hodnoty v časových intervalech a tyto brány umožňují kontrolovat jejich tok. Vstupní brána tak určuje, kdy neuron začne přijímat data; výstupní po jaké době je začne předávat dále a zapomínací po jaké době začne uchovávaná data zapomínat.

  Původní motivací k LSTM byl Vanishing Gradient problém. Při používání zpětné propagace, gradient, který byl zpětně propagován mohl „mizet“ (blížit se k nule) nebo „explodovat“ (blížit se k nekonečnu) z důvodu navrhnutých výpočtů v tomto procesu. Rekurentní neuronové sítě, které využívají princip LSTM, tento problém částečně eliminují, neboť dovolují, aby se gradienty propagovaly nezměněné. LSTM však mohou stále trpět problémem explodování gradientu.

- Hopfieldovy sítě

  Tyto sítě jsou trénovány na limitovaném množství dat tak, že korespondují známému prvku stejným prvkem. Každá buňka slouží jako vstupní před trénováním, jako skrytá při trénování a jako výstupní, když je použita. Hopfieldovy sítě mohou být použity pro odšumování a obnovování vstupů. Pokud tedy na vstup dostane polovinu obrázku, na kterém se natrénovala a na vstup dostane pouze polovinu, vrátí obrázek celý [12].

- Boltzmannův stroj

  Boltzmannův stroj je typ neuronových sítí, který je velice obdobný Hopfieldovým sítím – některé buňky jsou zde však označeny jako vstupní a zůstávají pouze skryté. Ostantí buňky se stanou výstupními poté, co každá ze skrytých změní svůj stav (při trénování Boltzmannova stroje či Hopfieldových sítí se buňky trénují postupně, nikoliv paralelně). Existují různé obměny tohoto přístupu, kde se například umožňuje v Boltzmannově stroji používat zpětná propagace [16].

Existuje řada dalších druhů neuronových sítí, kde se různě obměňují propojení jednotlivých neuronů – náhodné propojení, ob n vrstev apod. Mezi ně patří například Liquid State Machine, Extreme Learning Machine, Echo State Network, Deep Residual Network, Kohonen Network či Neural Turing machine.

6.3. Lineární diskriminační analýza

Lineární diskriminační analýza (LDA) je metoda používaná ve statistice, rozpoznávání obrazů a strojovém učení k nalezení lineární kombinace příznaků, které charakterizují dva nebo více objektů [9]. Jde o metodu, která se používá pro redukci dimenze příznaků před následnou klasifikací, ale může být použita jako samostatný lineární klasifikátor.

6.3.1. Klasifikace do dvou tříd

LDA předpokládá, že pravděpodobnostní hustotní funkce mají normální rozložení se středními hodnotami μ_i a kovariančními maticemi Σ_i. Uvažujme nyní pouhé rozdělení dvou tříd. Za těchto zmíněných předpokladů Bayesovo optimální řešení přiřazuje body do druhé třídy, pokud je logaritmus míry pravděpodobnosti vyšší než nějaká hranice T.

$$(x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0)+\ln|\Sigma_0|-(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)-\ln|\Sigma_1|>T \tag{22}$$

x – měřené příznaky, pozorování

Dále LDA předpokládá, že kovarianční matice jsou identické, tedy $\Sigma_0=\Sigma_1=\Sigma$ a že kovarinční matice mají plnou hodnost. Tím pádem lze vzorec (22) upravit do následující podoby:

$$w\cdot x>c \tag{23}$$

$w=\Sigma^{-1}(\mu_1-\mu_0)$

$c=\frac{1}{2}(T-\mu_0\Sigma^{-1}\mu_0+\mu_1\Sigma^{-1}\mu_1)$

Kritérium c je tedy pouhou lineární kombinací měřených příznaků a hranice T.

6.3.2. Klasifikace do více tříd

V případě klasifikace do více tříd je LDA rozšířena o nalezení podprostoru, který by měl obsahovat celou vnitřní variabilitu dané třídy. Předpokládáme-li, že každá třída má svou vlastní střední hodnotu μ_i a všechny třídy mají stejnou kovarianční matici Σ, pak definujeme rozptýlení mezi třídní variabilitou následovně:

$$\Sigma_b=\frac{1}{C}\sum_{i=1}^{C}(\mu_i-\mu)(\mu_i-\mu)^T \tag{24}$$

μ – zprůměrovaná hodnota středních hodnot

Rozdělení tříd tedy bude dáno:

$$\mathrm{S}=\frac{\mathrm{w}^T\Sigma_b\mathrm{w}}{\mathrm{w}^T\Sigma\mathrm{w}} \tag{25}$$

Klasifikátor zařadí bod do té třídy, pro kterou bude hodnota S nejvyšší.

7. Experimentální část

7.1. Struktura dat

Původní struktura dat byl strom se značným množstvím listů, pro které však nebylo dostatek dat, aby se na nich dal natrénovat klasifikátor. Třídy tedy bylo nutno poshlukovat, aby obsahovaly dostatečné množství dat pro natrénování klasifikátoru. Některé uzly, které obsahovaly málo dat a nebylo možné je sloučit s jinými byly kompletně vyřazeny. Průběh shlukování tříd lze vidět na obrázku 10.1.

Data, na kterých byly prováděny experimenty, byly rozděleny do 7 tříd. Každá z těchto tříd obsahovala řádově do tisíce dokumentů, kromě nejpočetnější třídy „balast“, která obsahovala přes 25 tisíc dokumentů.

Každý z těchto dokumentů představuje ručně anotované rozhovory mezi dotazujícím se a Ústavem pro jazyk český – přeskočil se proces automatického rozpoznávání řeči, neboť pak by se muselo počítat s chybami automatického přepisu. V každém dokumentu tedy máme již oddělená slova, nebylo tedy třeba se zabývat otázkou segmentací slov.

7.2. Vyvážení tříd

Při načítání dat nastal problém, kdy třída „balast“ byla značně objemnější než všechny ostatní třídy dohromady. Natrénování na těchto datech pak vedlo k problému, že klasifikátor se snažil klasifikovat většinu dat do třídy balast a protože těchto dat bylo velké množství, měly klasifikátory zdánlivě obrovskou procentuální úspěšnost.

Z tohoto důvodu bylo nutno provést vyvážení tříd, kdy celkový počet dat z jedné třídy byl omezen na tisíc. Tím, že bylo provedeno vyvážení tříd, byly získány více přesné výsledky. Vyvážení dat bylo provedené náhodně, tedy při každém testovacím cyklu byly použity jiné hodnoty.

7.3. Předzpracování dat

Při předzpracování dat bylo prováděno několik procesů. Experimenty byly zkoušeny s následujícím nastavením parametrů předzpracování dat:

- Použití balastu
- Použití lemmatizace (s dalším parametrem, jestli se mají používat přidané informace)
- Použít vyvážení tříd

Pokud byl nastaven parametr pro použití balastu na *false*, celá třída s balastem se kompletně vynechala. Celkový počet tříd v tomto případě byl 6. Dále pokud byl tento parametr vypnut, nebylo nutné dělat vyvážení tříd, neboť třídy samy o sobě byly vyvážené.

Dalším parametrem bylo použití lemmatizace [14]. Při tomto nastavení zde byl ještě jeden parametr a to, zda výsledek lemmatizace má poskytovat přídavné informace či nikoliv. Přídavnými informacemi je myšlen například význam slova – např. když se lemmatizovalo sloveso „stalo“ na „stát“,

tak se k němu připojila přídavná informace „něco se přihodilo“, neboť sloveso stát může mít význam stát na místě.

7.4. Parametrizace

Po veškerém předzpracování, byla data načtena do operační paměti. Soubory, resp. dokumenty, byly načteny včetně informace o jejich třídě. Pro rychlejší načítání dat při dalším běhu byla vyexportována do souboru s koncovkou plk (export pomocí knihovny pickle), který umožňuje několikanásobné zrychlení při načítání dat – tento proces zrychlení nebyl možný v případě, kdy se provádělo vyvážení tříd, neboť bylo dělané náhodně.

V tomto bodě přichází na řadu parametrizace dat. Parametrizace byla provedena pomocí TFIDF vektorizace. Tento bod je velice významný pro veškeré další kroky, protože určuje, s jakými konkrétními slovy bude klasifikátor počítat. Bylo tedy nutné zvolit adekvátní nastavení parametrů.

Zprvu byly vyzkoušeny pokusy s nízkým omezením na minimální a maximální DF. Tyto experimenty se ukázaly jako naprosto nepoužitelné, neboť v takovémto případě jsme narazili na limit výpočetní síly – do klasifikátorů tak vstupoval extrémně rozsáhlý vektor při jehož použití byla dosažena hranice operační paměti. Program tak ani nedoběhl do konce a přerušil se z důvodu nedostatku paměti.

Z tohoto důvodu bylo nutno počet dat značně omezit. Hledání optimálních parametrů je iterativní proces. Na základě heuristického přístupu se omezovalo DF postupně shora i zdola a bylo pozorováno chování programu – důraz byl kladen hlavně na rozměr vektoru TFIDF a úspěšnost klasifikátorů. Bylo zjištěno, že omezení DF shora (postupně z 1 až na 0,5 po jedné desetině) nijak zvlášť nesnižuje rozměr TFIDF vektoru. Avšak snížení zdola (z 0 na 0,1) zredukovalo dimenzi TFIDF až příliš. Z tohoto chování můžeme říct, že data neobsahují příliš slov, která by se vyskytovali často mezi všemi dokumenty a obsahuje značné množství specifických slov. Dále byl zkoušen měněn parametr ngram_range, avšak změna n-gramů nepřinesla značné zlepšení, proto byl parametr ponechán na defaultní hodnotě – tj. bez n-gramů.

Při experimentech jsem došel k závěru, že je vhodné nastavení parametrů rozdělit pro případ, zda se používá vyvážení tříd či nikoliv. Pro případ bez vyvážení dat bylo zvoleno následující nastavení parametrů TFIDF:

vectorizer = TfidfVectorizer(max_df=0.5, min_df=0.0005, use_idf=True)

A pro případ s vyvážením dat:

vectorizer = TfidfVectorizer(max_df=0.5, min_df=0.01, use_idf=True)

Tímto se inicializuje TfidfVectorizer a výsledné TFIDF dostaneme po zavolání funkce fit_transform, které jako vstupní parametr pošleme veškeré načtené dokumenty.

TfidfVectorizer je Pythonovská funkce z knihovny scikit-learn. Scikit-learn je knihovna určená primárně pro data mining a analýzu dat a je postavená primárně na knihovnách NumPy, SciPy a matplotlib. Tato knihovna byla v bakalářské práci hojně použita, neboť poskytuje i další funkce jako jsou LinearDiscriminantAnalysis, train_test_split či SVC.

Funkce TfidfVectorizer poskytuje další užitečné parametry jako jsou stop_words, ngram_range, max_features či vocabulary. Tyto byly ponechány na defaultních hodnotách. Stop_words umožňuje definovat stopová slova, která budou při parametrizaci vynechána; ngram_range určuje jaký počet n-gramů bude vytvořen (tj. bude nahlíženo na kontext jednotlivých slov); max_features umožňuje nastavení limitu maximálního počtu příznaků a parametr vocabulary umožňuje definovat si vlastní slovník, která slova budou při parametrizaci brána v potaz – vlastní slovník byl použit v projektu NAKI, proto se výsledky této bakalářské práce mírně liší.

7.4.1. Parametrizace doc2vec

Při experimentech byla testována i parametrizace doc2vec a po ní následná klasfikace. Bylo zjištěno, že při této parametrizaci dosahují klasifikátory horších výsledků. To je dáno malým množstvím dat, jelikož doc2vec zajišťuje lepší výsledky pro big data.

7.5. Klasifikační metody

Bylo použito několik klasifikačních metod – metoda založená na Support Vector Machines, SVC: Support Vector Classification, dále lineární diskriminační analýza, kombinace LDA a SVC – pomocí LDA byla redukována dimenze a SVC dále klasfikoval a v neposlední řadě klasifikace pomocí neuronových sítí. Každé z těchto metod bylo nutné najít takové parametry se kterými produkoval klasifikátor nejlepší výsledky. Je nutné zmínit, že při změně nastavení parametrů TFIDF klasifikátory poskytovaly jiné výsledky a někdy poskytovaly o něco lepší výsledky, pokud jejich nastavení bylo opět modifikováno.

Před natrénováním klasifikátorů zde proběhlo rozdělení na trénovací a testovací množinu pomocí funkce train_test_split opět poskytnuté knihovnou scikit-learn. Data byla rozdělena v poměru 9:1 (90% celkových dat TFIDF vectorizeru bylo použito jako trénovací a 10% jako testovací). Před rozdělením jsou data nejprve náhodně zamíchána a následně poměrně rozdělena. Příkaz pro poměrné rozdělení:

X_train, X_test, y_train, y_test = train_test_split(TFIDF, target, test_size=0.1, stratify=target)

7.5.1. Support Vector Classification

První klasifikace, která byla testována byla pomocí SVC – metody založené na Support Vector Machines. Pro SVC byly nastaveny následující parametry:

clf = SVC(kernel='linear')

Jediný parametr, který byl změněn, je jádro SVM. Defaultní parametr jádra je RBF – Radial Basis Function, avšak po vyzkoušení různých kombinací bylo zjištěno, že linární jádro poskytuje nejlepší výsledky.

Další jádra, která algoritmus SVC od scikit-learn poskytuje, jsou poly, sigmoid a precomputed. Samotná metoda má několik nastavitelných parametrů, například umožňuje nastavit vlastní hodnotu parametru C, který určuje váhu chyby; dále umožňuje omezit počet iterací algoritmu či nastavení rozhodovací funkce. Při klasifikaci do více tříd je vždy uplatňován přístup one-vs-one.

Natrénování klasifikátoru probíhá přes zavolání funkce *clf.fit()*, které jako vstupní parametr posíláme trénovací množinu dat včetně informace o správné klasifikaci. Po natrénování je již klasifikátor schopen klasifikovat na neznámých datech.

7.5.2. Lineární diskriminační analýza

Další klasifikátor, který byl testován je LDA. Knihovna scikit-learn opět nabízí značné množství volitelných parametrů, nicméně jediný parametr, který byl nastaven, je n_components, který určuje dimenzi, ve které LDA pracuje. Pro klasifikaci čistě pomocí LDA byl tento parametr ponechán na počet příznaků obsažených v trénovací množině (*X_train.shape[1]*).

*clf = LDA(n_components=int(X_train.shape[1]*1))*

Následné natrénování klasifikátoru bylo vykonáno obdobně jako u SVC pomocí funkce fit, které jako vstupní parametry posíláme trénovací množinu dat, včetně informace o správné klasifikaci.

7.5.3. LDA-SVC Klasifikace

LDA-SVC klasifikace je kombinace předchozích dvou klasifikací. Pomocí LDA snižujeme dimenzi vstupních dat a pomocí SVC je prováděna následná klasifikace. Bylo testováno různé nastavení redukce dimenze – od naprosto minimální (což je počet tříd, do kterých se klasifikuje) až po žádnou redukci.

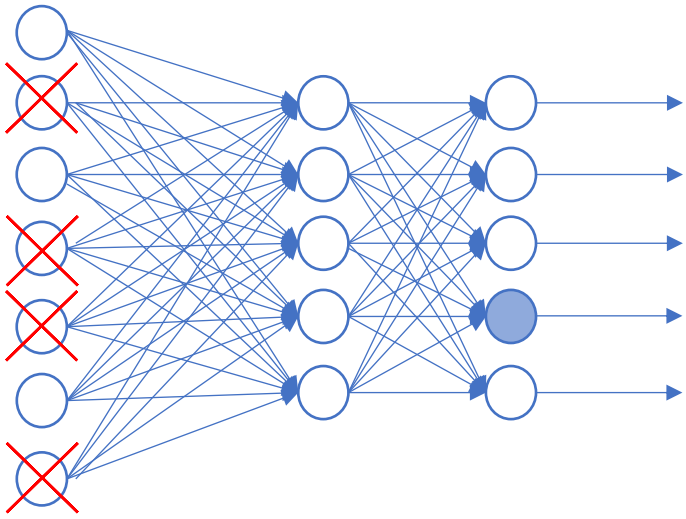
U SVC byla v tomto případě také testována změna jádra, avšak lineární jádro se opět ukázalo jako nejlepší.

7.5.4. Neuronové Sítě

Poslední klasifikace byla uskutečněna pomocí neuronových sítí. Neuronové sítě jako takové nabízí extrémní možnosti kombinací parametrů. I v těchto experimentech bylo vyzkoušeno značné množství různých kombinací. Při takovém množství kombinací se nabízelo tento proces zautomatizovat – navrhl se systém, který postupně měnil různé parametry. Tento systém zároveň sledoval i dobu cyklu pro

jedno nastavení parametrů a po několika cyklech bylo zjištěno, že není možné otestovat veškeré kombinace, neboť odhadovaná doba otestování všech možných kombinací byla několik let. Bylo nutno specifikovat značně omezené množství kombinací, které bylo následně otestováno.

Výsledná neuronová síť je sekvenční model skládající se z plně propojené vrstvy na vstupu s následnou dropout vrstvou, další plně propojenou vrstvou, u které byla měněna většina parametrů a poslední, výstupní, plně propojenou vrstvou.



Obr. 6) Architektura použité neuronové sítě: 3 plně propojené vrstvy, 1. vstupní vrstva obsahuje n neuronů, kde n je počet příznaků; 2. vrstva obsahuje pouze 7 neuronů a nahlíží pouze na 10 % neuronů z předchozí vrstvy (vyřazené neurony jsou označeny červeným křížem); 3. výstupní vrstva obsahuje též 7 neuronů, kde při trénování je aktivován pouze 1

Tabulka 7.1: Tabulka k obrázku 7, popisující jednotlivé vrstvy

Typ metody	**1. vrstva**	**2. vrstva**	**3. vrstva**
Typ vrstvy	vstupní	skrytá	výstupní
Počet neuronů	n	7	7
Aktivační funkce	-	sigmoid	softmax
Propojení	90 % neuronů je náhodně vyřazeno, tj. propojeno pouze 10 %	plně propojená	plně propojená

Vstupní vrstva na začátku neuronové sítě obsahuje takový počet neuronů jako je dimenze vstupních parametrů. Každá z plně propojených vrstev obsahuje i aktivační funkci. Bylo zkoušeno několik různých aktivačních funkcí, avšak nakonec byla ponechána funkce sigmoid, protože poskytovala nejuspokojivější výsledky.

Následná vrstva je dropout, která dělá to, že náhodná data vyháže, čímž nutí následující vrstvu predikovat bez veškeré znalosti o vstupních datech. Tato „vrstva“ úplně vrstvou není, pouze v kódu je označována jako vrstva – ve skutečnosti pouze redukuje počet použitých neuronů ve vstupní vrstvě. Tento proces značně zlepšuje následné predikování neuronové sítě, nicméně celý proces trénování mírně zpomaluje, neboť se neuronová síť musí učit déle. Parametr dropout vrstvy byl nastaven na 0,9 čili 90 % vstupních dat bylo náhodně vymazáno před vstupem do další vrstvy. Opět zde byly testovány různé hodnoty a výsledné nastavení s poměrně velkým výmazem dat poskytovalo nejlepší výsledky.

Další částí neuronové sítě byla plně propojená vrstva. V této části bylo testováno použití několika vrstev s různým počtem neuronů i různými nastaveními aktivačních funkcí, nicméně žádné z testovaných nastavení nepřinoslo značné zlepšení. Nakonec v tomto bloku byla ponechána pouze 1 vrstva, která obsahovala pouze počet neuronů rovný počtu klasifikovaných tříd.

Poslední vrstvou byla již výstupní, klasifikační vrstva. Tato vrstva obsahuje počet neuronů rovný počtu tříd, do kterých bylo klasifikováno. Výstupní vrstva používá pro klasifikaci aktiviační funkci softmax.

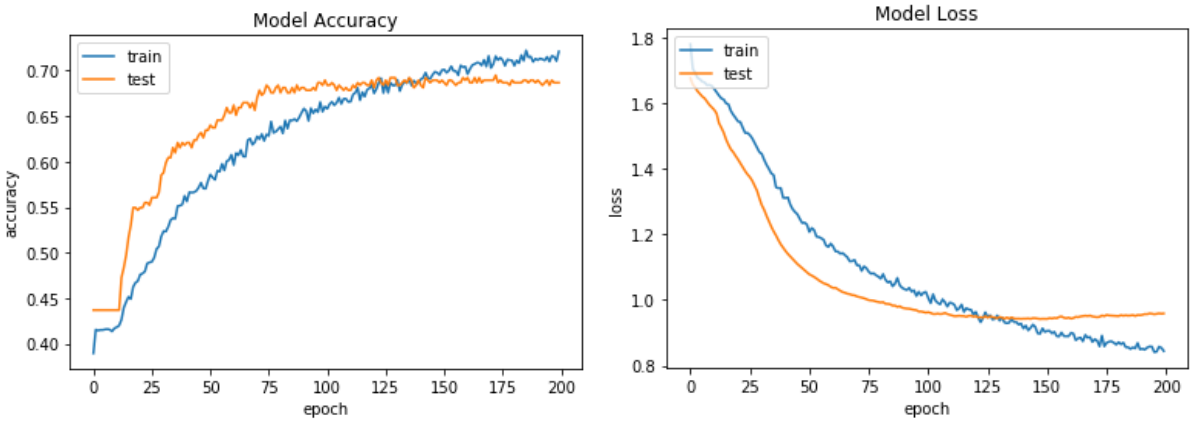
Po nadefinování celé neuronové sítě je potřeba ji zkompilovat a natrénovat. Při kompilaci bylo nutno nadefinovat ztrátovou funkci, optimalizační funkci a metriku. Ztrátová funkce byla nastavena na kategorickou křížovou entropii (categorical_crossentropy), která se zdála být nejvhodnější volbou. Dalšími možnostmi ztrátové funkce pro klasifikaci do více tříd byly řídká křížová entropie (Sparse Cross-Entropy) a Kullback Leibner Divergence, avšak tyto ztrátové funkce jsou používány pro více komplexní problémy klasifikace. Optimalizační funkce byla zvolena na funkci *adam*, která sama predikuje na základě výpočtů optimální nastavení parametrů, není tedy třeba iterovat jako u *SGD.* Dalšími možnostmi nastavení byly například různé obměny algoritmu *adam*, například Nadam či Adamax. Používaná metrika byla zvolena accuracy čili přesnost klasifikace.

Posledním krokem je již samotné trénování sítě. Při trénování sítě je nutno zvolit dobu trénování čili počet cyklů neboli epoch – tuto dobu je potřeba správně odhadnout, nebo vypozorovat na chování neuronové sítě, opět je zde stejný problém, pokud bude trénována na stejných datech příliš dlouho, dojde k jejímu přetrénování.

Při trénování neuronové sítě bylo potřeba rozdělit dataset na 3 části – train, test a dev. Dev část dat byla použita pro vyhodnocení jednotlivých epoch trénování neuronové sítě kdežto testovací část dat byla použita pro finální vyhodnocení natrénované neuronové sítě.

Dalším podstatným parametrem je *batch_size*, který určuje, jaké množství vstupních dat se bude používat při jednom cyklu. Pokud by na vstupu byl příliš velký počet dat, mohlo by dojít k problému přetečení paměti. Snížením tohoto parametru se zvyšuje potřebný počet iterací k natrénování sítě, neboť její parametry se upravují pomaleji. Finální hodnota tohoto parametru byla nastavena na hodnotu 128.

Po každém natrénování neuronové sítě pro každou kombinaci parametrů byl vykreslen graf průběhu učení neuronové sítě.



Obr. 7) Graf průběhu přesnosti a ztráty trénování modelu neuronové sítě

Na grafu vlevo je zobrazena přesnost modelu oproti trénovacím a testovacím datům. Z grafu lze vypozorovat, že přesnost modelu oproti testovacím datům, resp. dev množině, se po cca 120 cyklech ustálí a při dalším trénování se pouze zlepšuje přesnost klasifikace trénovacích dat. Obdobnou situaci

vidíme na grafu vpravo, kde je znázorněna ztráta. Na tomto grafu lze vidět, že po cca 120 cyklech se ztráta mírně začíná zvyšovat pro testovací data, zatímco pro trénovací se stále snižuje. Z těchto grafů můžeme říci, že optimální počet iterací trénování této neuronové sítě je právě oněch 120 epoch – při značně menším počtu by neuronová síť byla podtrénovaná; a naopak při značně vyšším počtu dochází k problému přetrénování.

7.6. Míra vyhodnocení úspěšnosti klasifikace

Úspěšnost klasifikátorů byla vyhodnocena metodou *score* obsaženou v knihovně scikit-learn, která vrací střední hodnotu přesnosti klasifikace. Vzorec pro výpočet skóre je následující:

$$accuracy(y, \hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1} 1 \cdot (y_i = \hat{y}_i) \tag{26}$$

y – vektor obsahující informaci o správné klasifikaci jednotlivých dokumentů
$\hat{y}$ – vektor obsahující predikovanou třídu jednotlivých dokumentů
n – celkový počet testovaných dokumentů, tj. testovací množina dat

Tato metoda byla použita pro všechny klasifikátory. Další možnou mírou vyhodnocení je F1 skóre. F1 míra se však využívá, když hledáme rovnováhu mezi podílem preciznosti a úplnosti klasifikátoru. Preciznost klasifikátoru udává, jak moc se dá věřit naší klasifikaci a úplnost udává, jaký podíl všech správných klasifikací odhalíme. F1 míra může být lepší mírou, pokud hledáme právě tuto rovnováhu a zároveň je zde nerovnoměrné rozdělení tříd. Zde však bylo požadováno, aby měl klasifikátor co největší přesnost, proto byla zvolena míra přesnosti klasifikace.

Z přiložených tabulek je patrné, že metoda SVC s lineárním jádrem ve většině případů poskytuje nejlepší výsledky, nehledě na ostatní nastavené parametry.

7.7. Vyhodnocení výsledků

První experimenty byly vykonány pro nevyvážené třídy. Bylo dosaženo až 90% přesnosti klasifikace – tyto výsledky však byly špatně, neboť klasifikátor uměl správně klasifikovat pouze do třídy balast, která obsahovala řádově více dat než ostatní třídy. Všechny následující experimenty byly učiněny pro vyvážené třídy.

V tabulkách 10.1 až 10.6 lze vidět, že při použití vyvážených tříd přítomnost třídy balast nijak zásadně neovlivňuje výsledky klasifikátorů. V těchto tabulkách je porovnána i přítomnost lemmatizace, a to jak s přídavnými informacemi, tak bez přídavných informací. Bylo zjištěno, že přítomnost lemmatizace bez přídavných informací mírně zvyšuje přesnost klasifikace, proto ve všech ostatních experimentech je vždy používána. Lemmatizace s přídavnými informacemi žádné zásadní zlepšení nepřinesla.

Tabulky 10.7, 10.8, 10.9 zachycují úspěšnost klasifikátorů pro různá nastavení parametrů TFIDF. Poměrně velké navýšení parametru max_df nijak zásadně nezměnilo výsledky experimentů, kdežto

snížení parametru min_df způsobilo značné zhoršení klasifikace – tento jev je dán tím, že TFIDF zpracovává slova příliš specifická pro jednotlivé dokumenty a klasifikátory pak nejsou schopny správně klasifikovat neviděné dokumenty z testovací množiny. Je zde také vidět, že SVC zvládá stále klasifikovat obstojně, kdežto LDA ani neuronové sítě tolik ne. Pro zbytek experimentů bylo použito nastavení TFIDF shodné s tabulkou 10.7.

Následující oddíl tabulek (10.10 až 10.14) znázorňuje vyhodnocení úspěšnosti klasifikátorů pro různá rozdělení trénovací a testovací množiny. Bylo zjištěno, že se výsledky zhoršují při nastavení poměru 1:1, avšak pouze mírně. To je způsobeno již tak ne příliš úspěšnou klasifikací, která se pohybuje okolo 67 % u klasifikátoru SVC.

V tabulce 10.15 je vidět, že bylo dosaženo velice dobrých výsledků (okolo 90% přesnosti), avšak to je zapříčiněno nevyváženými třídami. Většina dat pochází ze třídy balast, proto je tato třída dobře natrénovaná a většina dat je klasifikována právě do této třídy. Ostatní třídy jsou však natrénovány špatně čili je klasifikace do nich velice neúspěšná. Proto tyto výsledky příliš nevypovídají skutečnou realitu.

Tabulka 10.16 znázorňuje výsledky klasifikátorů při použití parametrizace doc2vec. Tyto výsledky jsou značně horší oproti použití TFIDF, neboť doc2vec je úspěšný zejména při použití velkých dokumentů, kdežto v této práci bylo pracováno pouze se soubory malými.

Další experimenty testovaly úspěšnost klasifikátoru SVC pro různé nastavení jader (10.17). Bylo zjištěno, že linární jádro poskytuje nejlepší výsledky.

V neposlední řadě byla testována změna redukce dimenze pomocí LDA s následnou SVC klasifikací využívající jádro RBF. Bylo zjištěno, že změna redukce dimenze zásadně neovlivňuje úspěšnost klasifikace – všechny případy měli přibližně stejné průměrné výsledky.

Na závěr byly prováděny experimenty pro různá nastavení neuronových sítí. Jelikož neuronové sítě nabízí téměř neomezené množství způsobů jejího sestrojení, bylo nutno některá nastavení zvolit experimentálně. Experimenty probíhali tak, že se zafixovaly všechny parametry kromě jednoho, který se měnil a sledoval se jeho vliv na neuronovou síť. Těmito parametry je myšleno zejména: počet neuronů, počet skrytých vrstev, způsob jejich propojení, volba aktivační funkce nebo také počet cyklů trénování sítě.

První neuronová síť byl sekvenční model skládající se pouze ze vstupní a výstupní vrstvy. Výstupní vrstva používá aktivační funkci *softmax*, ztrátová funkce byla zvolena křížová entropie, algoritmus pro nalezení optimálních parametrů byl použit *adam* a metrika byla nastavena na přesnost modelu.

Zprvu byl počet cyklů nastaven na velice nízké číslo, což způsobilo, že neuronová síť nebyla dotrénovaná a klasifikátor tak poskytoval špatné výsledky. Když byl počet cyklů navýšen, tak přesnost modelu vždy konvergovala k nějakému číslu (tento jev lze vidět na obrázku 6).

První experimenty s neuronovými sítěmi zahrnovali upravování počtů neuronů ve skrytých vrstvách. Experimenty ukázaly, že počet neuronů neměl zásadní vliv na úspěšnost klasifikace. Nejlepší výsledky poskytovala neuronová síť s počtem 200 neuronů bez ohledu na počet vrstev, toto nastavení pak bylo zvoleno pro další experimenty. Výsledky odráží tabulky 10.23 a 10.24.

V předchozích experimentech byly vrstvy vždy plně propojené, proto v následujících testech bylo vyzkoušeno přidání další vrstvy dropout. Tato vrstva byla přidána hned za vstupní vrstvu a její vliv byl ten, že určité procento vstupů před neuronovou sítí skryl – tak se tak mohla naučit pracovat s omezeným vstupem. Experimenty ukázaly, že čím vyšší procento vrstvy dropout je, tím lépe neuronová síť pracuje s novými vstupy. Výsledky lze vidět v tabulce 10.19.

Další experimenty zachycuje tabulka 10.20, kde bylo testováno nastavení objemu várky (batch_size). Tento parametr udává, jaké množství vstupních dat se bude zpracovávat při jednom cyklu neuronové sítě. Ukázalo se, že omezení těchto dat zlepšuje výsledky klasifikace.

V dalších experimentech (tabulka 10.21) se zkoumal vliv aktivačních funkcí. Ukázalo se, že přestože funkce sigmoid trpí problémy jako je klesající gradient, ukázalo se, že v tomto případě je nejvhodnější funkcí. V případě ostatních aktivačních funkcí poskytoval klasifikátor velice špatné výsledky.

Na závěr byl otestován optimizátor SGD. Ukázalo se, že v tomto případě se neuronová síť nenatrénuje tak dobře jako v případě algoritmu *adam*.

8. Závěr

V této práci bylo zkoumáno chování klasifikačních metod na textových dokumentech představujících anotované rozhovory mezi dotazujícím a jazykovým poradcem od Ústavu pro jazyk český. Tento proces obsáhl několik oblastí úloh zpracování přirozeného jazyka – od úloh předzpracování dat (převážně lemmatizace), přes úlohy parametrizace dat (TFIDF, doc2vec), až po následné klasifikační metody (SVC, LDA, LDA-SVC, ANN). Před poskytnutím dat byly rozhovory ručně anotovány a byla jim ručně přiřazena témata – práce se tedy zabývá zpracováním textu, čímž navazuje na předchozí zpracování řeči.

Bylo provedeno několik experimetů pro různá nastavení parametrů metod. První problém, který se musel v této práci řešit, bylo značné nevyvážení tříd – třída balast obsahovala řádově více dat než ostatní třídy. Při klasifikaci byly výsledky zdánlivě nadměrně úspěšné, avšak tyto vysoce úspěšné výsledky byly dány tím, že většina dat pocházela právě ze třídy balast, proto bylo velké množství dat klasifikováno úspěšně. Z tohoto důvodu se muselo provést vyvážení tříd, aby všechny obsahovaly řádově stejné množství dat.

Po úspěšném načtení dat ve smysluplném celku byla řešena parametrizace – zprvu TFIDF, následně doc2vec. Při parametrizace TFIDF bylo nutno zvolit správné parametry minimálního a maximálního počtu výskytů prvků v dokumentech. Pokud tyto byly nastaveny nevhodně, podepsaly se na tom i výsledky klasifikátorů – klasifikátor nemůže správně klasifikovat, pokud dostane špatně parametrizovaná data. Je důležité zmínit, že nebylo možné použít všechny parametry (resp. všechna slova obsažené v dokumentech), neboť jinak byly atakovány hardwarové omezení počítače, na kterém běžely tyto experimenty. V tomto kroce bylo tedy úkolem zvolit správné nastavení parametrů. Zjistilo se, že při omezování maximálního výskytu se dimenze vektoru zmenšovala velice nepatrně, avšak při omezení výskytu zdola se dimenze vektoru snižovala velice rychle. Z tohoto chování je možné o datech říci, že se v nich vyskytuje malé množství často se opakujících stejných slov, ale obsahuje mnoho specifických slov pro jednotlivé dokumenty. Parametrizace TFIDF byla následně porovnána s parametrizací doc2vec. Bylo zjištěno, že výsledky po parametrizaci doc2vec byly značně horší, což bylo dáno zejména nízkým množstvím trénovacích dat.

Následně bylo v této práci řešeno rozdělení dat na trénovací, testovací a development množinu. Počáteční natrénování všech klasifikátorů bylo provedeno na trénovací množině. Všechny klasifikátory si měly následně upravovat nastavení parametrů pomocí development množiny, avšak to bylo vykonáno pouze u neuronových sítí, kde se bez development množiny nedalo obejít. Ostatní klasifikátory byly pouze natrénovány na trénovací množině z důvodu malého množství dat.

Dále byly aplikovány algoritmy pro klasifikaci pro různá nastavení parametrů a byly reprodukovány výsledky projektu NAKI. Výsledky se liší z důvodu různého použití nastavení – primárně kvůli různému nastavení parametrů TFIDF, kde v projektu NAKI byl použit vlastní slovník, kdežto v této práci bylo TFIDF vytvořeno pomocí omezení min a max document frequency. Při nejlepším zvolení parametrů s

použitím vyvážení tříd, bylo dosaženo přesnosti klasifikace 70 % (projekt NAKI 77 %). Nejlepší klasifikátor se ukázal SVC s lineárním jádrem.

V poslední řadě se práce zabývá klasifikací pomocí neuronových sítí. Byly provedeny četné experimenty, kde se testoval vliv počtu neuronů ve skrytých vrstvách, vliv aktivačních funkcí, počet a způsob propojení vrstev a další. Výsledky těchto experimentů lze vidět v přiložených tabulkách. Bylo zjištěno, že lze dosáhnout výsledků obdobných klasifikaci SVC, avšak v tomto případě ne lepších. Nejlépe se tak osvědčil poměrně základní, avšak efektivní klasifikátor založený na podpůrných vektorech.

9. Seznam literatury

[1] Řehůřek Radim, Sojka Petr. "Software Framework for Topic Modelling with Large Corpora", THE LREC WORKSHOP ON NEW CHALLENGES FOR NLP FRAMEWORKS, 2010.

[2] Dehak Najim, A. Torres-Carrasquillo Pedro, Reynolds Douglas, Dehak R. "Language Recognition via I-Vectors and Dimensionality Reduction", Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH, 2011.

[3] Zbyněk Zajíc, Lucie Zajícová, Josef V. Psutka, Petr Salajka, Jaromír Novotný, Aleš Pražák, Luděk Müller. "First Insight into the Processing of the Language Consulting Center Data", SPECOM 2018, p. 778-787, Springer, 2018.

[4] Andrew R. Webb. "Statistical Pattern Recognition", John Wiley & Sons Ltd, 2002.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffery Deany. "Efficient estimation of word representations in vector space", Proceedings of the International Conference on Learning Representations, 2013.

[6] Quoc Le, Tomas Mikolov. "Distributed Representations of Sentences and Documents", Proceedings of the 31st International Conference on International Conference on Machine Learning, Vol. 32, 2014.

[7] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, Jeff Dean. "Distributed representations of words and phrases and their compositionality", Advances in Neural Information Processing Systems, 2013.

[8] Ingo Steinwart, Andreas Christmann. "Support Vector Machines", Springer, 2008.

[9] Geoffrey J. McLachlan. "Discriminant Analysis and Statistical Pattern Recognition", Wiley-Interscience, 2005.

[10] Dan Jurafsky, James H. Martin. "Speech and Language Processing" (3rd ed. draft), 2018.

[11] Oppy, Graham and Dowe, David, "The Turing Test", The Stanford Encyclopedia of Philosophy (Spring 2019 Edition).

[12] Hořejš Jiří, Kubát Miroslav, Lažanský Jiří, Mařík Vladimír, Štěpánek Petr, Štěpánková Olga, Zdráhal Zdeněk. "Umělá inteligence", Akademie věd České republiky, 1993.

[13] Adrian A. Hopgood. "Intelligent Systems for Engineers and Scientists", CRC PR INC., 2011

[14] Straková Jana, Straka Milan. "Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition", Assiciation for Computational Linguistics, 2014.

[15] Sepp Hochreiter, Jürgen Schmidhuber. "Long Short-Term Memory", Neural computation, 1997.

[16] Geoffrey E. Hinton. "Boltzmann machine", Scholarpedia, 2007.

[17] David Powers. "EVALUATION: FROM PRECISION, RECALL AND F-MEASURE TO ROC, INFORMEDNESS, MARKEDNESS & CORRELATION", Mach. Learn. Technol., 2008.

10. Přílohy

A. Tabulky s výsledky

Všechny experimenty v každé z přiložených tabulek jsou výsledkem průměru 10 cyklů křížové validace. V těchto cyklech bylo zaznamenáno nejvyšší a nejnižší dosažená přesnost klasifikátorů a bylo spočteno jejich průměrné skóre.

Tabulky znázorňující vliv přítomnosti třídy balast a použití lemmatizace

Všechny následující tabulky vždy používají vyvážení tříd a jedno shodné nastavení TFDIF:

- max_df = 0,5; min_df = 0,01

Tabulka 10.1: Výsledky klasifikátorů s následujícím nastavením:

- Bez třídy balast
- Použití lemmatizace bez přídavných informací

Typ metody	Průměrné skóre	Minimální skóre	Maximální skóre
SVC	67,04 %	65,46 %	70,07 %
LDA	57,43 %	58,22 %	63,82 %
LDA-SVC	60,26 %	57,24 %	64,80 %
ANN	64,41 %	59,54 %	70,72 %

Tabulka 10.2: Výsledky klasifikátorů s následujícím nastavením:

- Použití třídy balast
- Použití lemmatizace bez přídavných informací

Typ metody	Průměrné skóre	Minimální skóre	Maximální skóre
SVC	66,88 %	63,37 %	71,04 %
LDA	63,22 %	59,16 %	66,09 %
LDA-SVC	62,85 %	58,66 %	65,59 %
ANN	64,18 %	59,65 %	66,83 %

Tabulka 10.3: Výsledky klasifikátorů s následujícím nastavením:

- Bez třídy balast
- Použití lemmatizace s přídavnými informacemi

Typ metody	Průměrné skóre	Minimální skóre	Maximální skóre
SVC	66,25 %	62,83 %	69,08 %
LDA	57,47 %	52,96 %	61,84 %
LDA-SVC	55,92 %	51,97 %	60,20 %
ANN	62,83 %	57,57 %	66,12 %

Tabulka 10.4: Výsledky klasifikátorů s následujícím nastavením:

- Použití třídy balast
- Použití lemmatizace s přídavnými informacemi

Typ metody	Průměrné skóre	Minimální skóre	Maximální skóre
SVC	66,41 %	63,12 %	69,06 %
LDA	62,43 %	60,40 %	65,10 %
LDA-SVC	61,63 %	58,66 %	64,60 %
ANN	63,89 %	61,39 %	65,84 %

Tabulka 10.5: Výsledky klasifikátorů s následujícím nastavením:

- Bez třídy balast
- Bez lemmatizace

Typ metody	Průměrné skóre	Minimální skóre	Maximální skóre
SVC	68,29 %	64,14 %	71,05 %
LDA	56,15 %	51,32 %	61,84 %
LDA-SVC	54,51 %	50,00 %	60,20 %
ANN	62,83 %	58,88 %	66,45 %

Tabulka 10.6: Výsledky klasifikátorů s následujícím nastavením:

- Použití třídy balast
- Bez lemmatizace

Typ metody	Průměrné skóre	Minimální skóre	Maximální skóre
SVC	66,51 %	60,89 %	69,55 %
LDA	60,02 %	54,21 %	62,13 %
LDA-SVC	59,41 %	52,72 %	61,88 %
ANN	62,70 %	59,65 %	65,10 %

Tabulky znázorňující vliv různého nastavení TFIDF

Tabulka 10.7: Výsledky klasifikátorů s nastavením TFIDF:

- max_df = 0,5; min_df = 0,01

Typ metody	**Průměrné skóre**	**Minimální skóre**	**Maximální skóre**
SVC	66,88 %	63,37 %	71,04 %
LDA	63,22 %	59,16 %	66,09 %
LDA-SVC	62,85 %	58,66 %	65,59 %
ANN	64,18 %	59,65 %	66,83 %

Tabulka 10.8: Výsledky klasifikátorů s nastavením TFIDF:

- max_df = 0,5; min_df = 0,001

Typ metody	**Průměrné skóre**	**Minimální skóre**	**Maximální skóre**
SVC	66,16 %	63,61 %	68,56 %
LDA	18,34 %	14,11 %	20,54 %
LDA-SVC	29,90 %	11,39 %	45,54 %
ANN	59,65 %	54,21 %	63,86 %

Tabulka 10.9: Výsledky klasifikátorů s nastavením TFIDF:

- max_df = 0,8; min_df = 0,01

Typ metody	**Průměrné skóre**	**Minimální skóre**	**Maximální skóre**
SVC	66,21 %	64,60 %	68,81 %
LDA	63,14 %	60,15 %	66,34 %
LDA-SVC	62,50 %	59,65 %	64,85 %
ANN	66,16 %	62,13 %	69,31 %

Tabulky znázorňující vliv různého poměru trénovací a testovací množiny dat

Tabulka 10.10: Výsledky klasifikátorů pro rozdělení trénovací ku testovací množině v poměru 9:1

Typ metody	Průměrné skóre	Minimální skóre	Maximální skóre
SVC	66,88 %	63,37 %	71,04 %
LDA	63,22 %	59,16 %	66,09 %
LDA-SVC	62,85 %	58,66 %	65,59 %
ANN	64,18 %	59,65 %	66,83 %

Tabulka 10.11: Výsledky klasifikátorů pro rozdělení trénovací ku testovací množině v poměru 4:1

Typ metody	Průměrné skóre	Minimální skóre	Maximální skóre
SVC	66,01 %	63,00 %	68,07 %
LDA	61,86 %	58,79 %	63,24 %
LDA-SVC	61,65 %	59,16 %	63,37 %
ANN	64,84 %	62,00 %	67,82 %

Tabulka 10.12: Výsledky klasifikátorů pro rozdělení trénovací ku testovací množině v poměru 7:3

Typ metody	Průměrné skóre	Minimální skóre	Maximální skóre
SVC	66,39 %	64,60 %	69,06 %
LDA	61,75 %	59,65 %	63,70 %
LDA-SVC	61,09 %	58,75 %	62,46 %
ANN	64,92 %	63,45 %	66,09 %

Tabulka 10.13: Výsledky klasifikátorů pro rozdělení trénovací ku testovací množině v poměru 3:2

Typ metody	Průměrné skóre	Minimální skóre	Maximální skóre
SVC	65,94 %	65,16 %	67,14 %
LDA	59,10 %	57,49 %	61,88 %
LDA-SVC	58,46 %	56,37 %	62,25 %
ANN	62,70 %	61,26 %	64,05 %

Tabulka 10.14: Výsledky klasifikátorů pro rozdělení trénovací ku testovací množině v poměru 1:1

Typ metody	Průměrné skóre	Minimální skóre	Maximální skóre
SVC	65,89 %	64,88 %	67,11 %
LDA	58,15 %	57,06 %	59,48 %
LDA-SVC	57,18 %	55,52 %	59,14 %
ANN	63,16 %	60,87 %	64,93 %

Ostatní tabulky zohledňující primárně na SVC, LDA a LDA-SVC

Tabulka 10.15: Výsledky klasifikátorů využívajícím parametrizaci doc2vec

Typ metody	Průměrné skóre	Minimální skóre	Maximální skóre
SVC	42,95 %	39,85 %	50,25 %
LDA	25,27 %	22,28 %	28,22 %
LDA-SVC	36,51 %	28,96 %	44,55 %
ANN	42,95 %	39,85 %	50,25 %

Následující tabulky jsou výsledky experimentů na vyvážených třídách s použitím třídy balast, využívá se lemmatizace bez přídavných informací a je použito TFIDF s jednotným nastavením parametrů:

- max_df = 0,5; min_df = 0,01

Tabulka 10.16: Výsledky SVC klasifikátoru pro různé použití transformačních jader

Jádro	Průměrné skóre	Minimální skóre	Maximální skóre
Lineární	66,88 %	63,37 %	71,04 %
RBF	43,39 %	40,10 %	47,52 %
Poly (n=3)	41,31 %	39,11 %	46,29 %
Sigmoid	41,29 %	38,12 %	48,27 %

Tabulka 10.17: Výsledky LDA-SVC klasifikátoru pro různou redukci dimenze vektoru příznaků

- Redukce značí, na jaké procento z původní dimenze se snižovalo (tj. když původní počet příznaků byl 1000, redukce je 90 %, pak je nový vektor o rozměru 900)
- SVC klasifikátor využívá jádro RBF

Redukce	Průměrné skóre	Minimální skóre	Maximální skóre
90 %	63,86 %	61,39 %	65,35 %
75 %	62,85 %	58,66 %	65,59 %
60 %	63,59 %	61,14 %	67,33 %
50 %	63,29 %	59,16 %	67,33 %
40 %	63,22 %	57,43 %	68,81 %
25 %	62,43 %	59,16 %	66,34 %

Tabulky znázorňující různá nastavení ANN

Tabulka 10.18: Výsledky neuronové sítě pro různé nastavení dropout vrstvy

Dropout	**Průměrné skóre**	**Minimální skóre**	**Maximální skóre**
0 %	50,89 %	48,92 %	53,23 %
10 %	52,18 %	50,66 %	55,25 %
20 %	54,55 %	52,40 %	56,30 %
30 %	53,45 %	52,65 %	55,16 %
50 %	57,37 %	54,72 %	63,20 %
75 %	63,46 %	61,10 %	67,42 %
90 %	67,12 %	64,32 %	72,45 %

Tabulka 10.19: Výsledky neuronové sítě pro různé nastavení objemu várky (batch_size):

batch_size	**Průměrné skóre**	**Minimální skóre**	**Maximální skóre**
128	67,12 %	64,32 %	72,45 %
256	67,67 %	65,40 %	70,32 %
512	65,44 %	62,53 %	68,31 %
1024	60,54 %	58,46 %	62,30 %
2048	55,54 %	54,70 %	57,40 %

Tabulka 10.20: Výsledky neuronové sítě pro různé nastavení aktivačních funkcí skrytých vrstev:

Aktivační funkce	**Průměrné skóre**	**Minimální skóre**	**Maximální skóre**
Sigmoid	67,12 %	64,32 %	72,45 %
Tanh	55,24 %	54,26 %	57,82 %
Linear	54,20 %	53,94 %	56,06 %
Relu	59,85 %	58,72 %	62,00 %

Tabulka 10.21: Výsledky neuronové sítě při nastavení optimizátoru *adam* vs *SGD*:

Optimizátor	**Průměrné skóre**	**Minimální skóre**	**Maximální skóre**
adam	67,12 %	64,32 %	72,45 %
sgd	41,13 %	40,05 %	43,45 %

Tabulka 10.22: Výsledky neuronové sítě pro různé nastavení počtu neuronů v 1 skryté vrstvě:

Počet neuronů	**Průměrné skóre**	**Minimální skóre**	**Maximální skóre**
7	65,44 %	63,21 %	66,40 %
50	64,70 %	63,01 %	65,46 %
100	66,78 %	65,20 %	68,20 %
200	67,12 %	64,32 %	72,45 %
400	66,58 %	63,72 %	69,33 %
700	66,98 %	64,70 %	70,79 %
1000	66,23 %	63,34 %	68.60 %
1500	64,60 %	63,26 %	66,15 %
2000	64,40 %	63,65 %	69,33 %

Tabulka 10.23: Výsledky neuronové sítě pro různé nastavení počtu neuronů se 2 skrytými vrstvy:

Počet neuronů (1. vrstva)	**Počet neuronů** (2. vrstva)	**Průměrné skóre**	**Minimální skóre**	**Maximální skóre**
100	100	68,21 %	64,42 %	72,88 %
100	50	65,59 %	62,86 %	68,15 %
100	25	64,90 %	61,92 %	66,16 %
200	200	67,52 %	65,82 %	68,60 %
200	100	66,03 %	64,36 %	69,75 %
200	50	66,73 %	63,94 %	68,71 %
400	400	65,44 %	62,85 %	66,01 %
400	200	65,34 %	64,84 %	67,38 %
400	100	67,32 %	64,81 %	69,60 %

B. Obrázky

Obrázek 10.1. – Průběh shlukování tříd