VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KLASIFIKACE DOKUMENTŮ POMOCÍ UMĚLÉ INTELIGENCE

ARTIFICIAL INTELLIGENCE DOCUMENT CLASSIFICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ONDŘEJ MOLNÁR**
AUTHOR

VEDOUCÍ PRÁCE **Ing. LENKA TŘEŠTÍKOVÁ**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Molnár Ondřej**
Program: Informační technologie
Název: **Klasifikace dokumentů pomocí umělé inteligence**
Artificial Intelligence Document Classification
Kategorie: Bezpečnost

Zadání:

1. Seznamte se se standardy ISO 27000, prostudujte principy pro zpracování textu pomocí umělé inteligence a analyzujte způsob, jak automaticky, na základě uživatelských dokumentů, klasifikovat dokumenty.
2. Na základě analýzy navrhněte architekturu pro automatickou klasifikaci dokumentů v prostředí MS Office.
3. Implementujte navržený systém složený z doplňku v MS Office, klasifikační služby a databáze.
4. Nástroj otestujte na vhodném testovacím vzorku dat. Provedte výkonnostní testy.
5. Diskutujte možnosti dalšího rozšíření.

Literatura:

- Řada norem ISO/IEC 27000
- Dle doporučení vedoucího.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Třeštíková Lenka, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Tato práce se zabývá klasifikací dokumentů za použití umělé inteligence. Popisuje principy klasifikace a strojového učení. Seznamuje s metodami UI a dále detailně představuje metodu klasifikace naivního Bayese. Poté líčí praktickou implementaci klasifikátoru do prostředí MS Office a diskutuje další možná rozšíření.

Abstract

This paper deals with document classification using artificial intelligence. It describes the principles of classification and machine learning. It also introduces AI methods and presents Naive Bayes classification method in detail. Provides practical implementation of the classifier in MS Office and discusses other possible extensions.

Klíčová slova

umělá inteligence, UI, klasifikace textu, klasifikace dokumentů, klasifikátor, naivní Bayes, strojové učení, zpracování přirozeného jazyka

Keywords

Artificial intelligence, AI, Text Classification, Document classification, Classifier, Naive Bayes, Machine learning, Natural language processing

Citace

MOLNÁR, Ondřej. *Klasifikace dokumentů pomocí umělé inteligence*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Lenka Třeštíková

Klasifikace dokumentů pomocí umělé inteligence

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením paní Ing. Lenky Třeštíkové a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ondřej Molnár
16. května 2019

Poděkování

Chtěl bych poděkovat Ing. Lence Třeštíkové za vedení mé bakalářské práce, cenné rady a odborný dohled. Rovněž děkuji za spolupráci s kolegy ze společnosti AEC.

Obsah

Kapitola 1

Úvod

Každá firma se s narůstajícím počtem dokumentů dostane do stavu, kdy je nutné své písemnosti zabezpečit – hlavně před ztrátou a zneužitím informací obsažených v nich. Někteří to řeší tím, že si vytvoří interní směrnice, kterými se zaměstnanci musí řídit při práci s informacemi. Mnohdy tyto směrnice doplní i školeními. Jiní zase používají speciální nástroje, které uživatelům pomáhají. Ze zkušeností firmy AEC a.s. vychází fakt, že účinnost školení bývá pouze krátkodobá. Tak jaké je tedy lepší řešení?

Společnost AEC se zabývá bezpečností informací a shodou okolností pro ně taktéž pracuji. Takže při vytváření této práce jsem s kolegy často spolupracoval a čerpal z jejich bohatých poznatků. Firma poskytuje řešení problematiky klasifikace dokumentů formou doplňku do kancelářského balíku MS Office. Díky doplňku DocTag uživatel už nemusí manuálně vytvářet klasifikační značku v dokumentu, celý proces je automatizován. Scénář je následující – uživatel vytváří nějaký dokument a je povinen ho označit mírou důvěrnosti. Může tak učinit více způsoby (kliknutím na tlačítko klasifikace, makrem, převzetím z přílohy), ale bez klasifikace ten dokument zkrátka není možné uložit. Nástroj samozřejmě popisuje jednotlivé stupně, aby uživatel mohl patřičně vybrat klasifikaci dokumentu. Ovšem i tak se často uživatelé potýkají s problémem, že neví, který stupeň vybrat nebo často spěchají a potřebují se rychle rozhodnout. To může vést ke špatnému označení dokumentu. Právě to má za cíl vyřešit tato bakalářská práce – doporučit uživateli klasifikační stupeň pomocí umělé inteligence na základě předem nasbíraných dat.

První kapitola má za cíl seznámit čtenáře s problematikou klasifikování textu v přirozeném jazyce. V první podkapitole je možno se dočíst, co to klasifikování je, jaké druhy existují a závěrem podkapitoly je důležitá zmínka o krocích, které je nutné učinit ještě před samotným vyhodnocením klasifikace. Další podkapitola se zaměřuje na uvedení do tématiky umělé inteligence. Popisuje jak je vnímána z veřejného konsensu a naopak z vědeckého hlediska, vnáší do povědomí trochu historie a prezentuje základní terminologii. Poslední podkapitola je věnována konkrétním běžně používaným klasifikátorům.

V druhé kapitole se práce zabývá vlastním řešením. Nejprve je řešen návrh, ve kterém je popsán samotný doplněk DocTag, do kterého se řešení přidává. Poté tato kapitola líčí, jaká rozhodnutí musela být učiněna ještě před samotnou implementací, aby se předešlo problémům a zvažuje různé metody implementace. Kapitola je zakončena konkrétním postupem realizace modulu umělé inteligence. Další navazující kapitolou je přehled o testování aplikace. Text zmiňuje, jak testování probíhalo, jaké byly dosažené výsledky a jaké jsou možnosti budoucího rozšíření aplikace. V závěru této práce je celkové vyhodnocení.

Kapitola 2

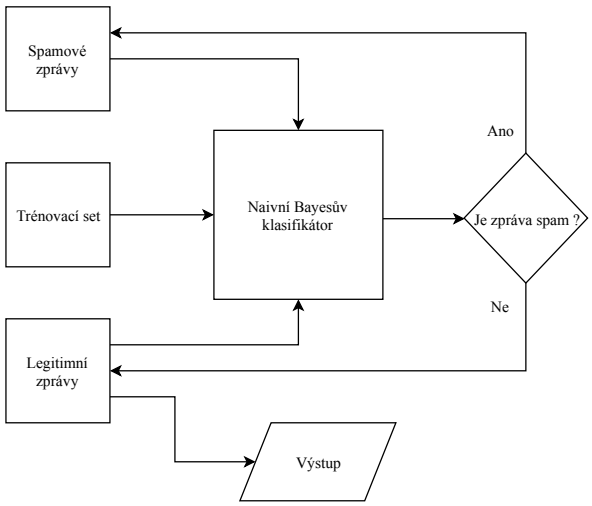
Principy klasifikace a strojového učení

Tato kapitola je analýzou celé problematiky klasifikování. Čtenář se dozví, jak je na klasifikaci nahlíženo. Jaké kroky je vhodné učinit před samotnou klasifikací (obrázek 2.4). Dále se obeznámí s umělou inteligencí, v jakých oborech se využívá, jak je možno ji dělit a v jaké formě se typicky implementuje. Následně může získat povědomí o konkrétních klasifikátorech – neuronových sítích a naivního Bayese.

2.1 Klasifikace dokumentů

Klasifikace neboli česky třídění či členění je proces, kdy přiřazujeme nějakou entitu do určitých tříd nebo skupin. Typickým příkladem mohou být známky ve škole, typ řidičského průkazu nebo třeba rozlišení krevní skupiny. Dále taky můžeme dělit typy klasifikací podle zdroje [28] na:

- Binární klasifikace je takové třídění, kde je možno rozlišovat pouze ano/ne. Kupříkladu může být rozlišování prádla podle barvy - bílé/barevné nebo třeba spamový filtr na obrázku 2.1, který vyhodnocuje, zda se jedná o nevyžádanou poštu či nikoliv.



Obrázek 2.1: Ukázka spamového filtru

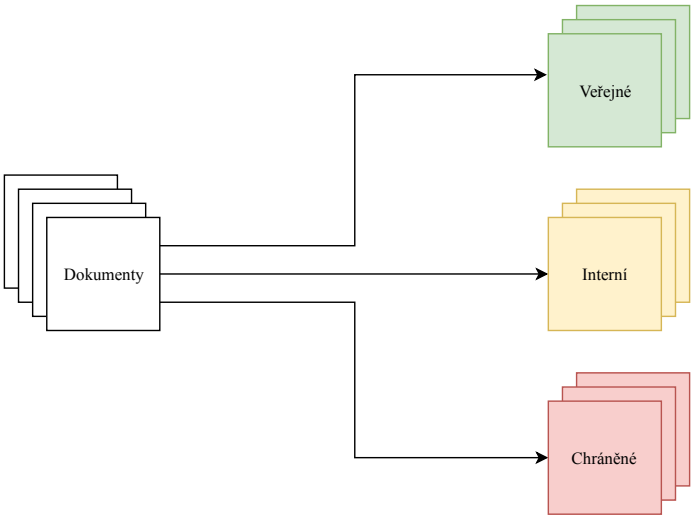
- *Multi-class* klasifikace člení entitu do skupin exkluzivně, tedy smí spadat pouze do jedné kategorie. Pro představu můžeme rozlišovat třeba loď - může být rybářská, vyhlídková nebo nákladní.
- *Multi-label* klasifikace dovoluje přidělit určitému objektu dokonce více skupin. Příkladem je jednoduchá tabulka 2.1, která popisuje 4 různé dokumenty, přičemž každý dokument se může zabývat jedním nebo až čtyřmi tématy najednou.

Č. dok.	**Věda**	**Sport**	**Zábava**	**Politika**
1	X			X
2		X		
3	X		X	
4	X	X		

Tabulka 2.1: Tématické dokumenty

Taktéž lze definovat klasifikaci textu podle publikace [29] jako úlohu přiřazení booleovské hodnoty každému páru $< d_j, c_i > \in D \times C$, kde D je soubor dokumentů a $C = \{c_1, ..., c_N\}$ je množina předem definovaných kategorií. Hodnota T (*True*) přiřazená dvojici $< d_j, c_i >$ značí přiřazení dokumentu d_j do kategorie c_i, zatímco hodnota F (*False*) vyjadřuje, že dokument d_j nespadá pod třídu c_j. Pokud bychom chtěli být formálnější, lze klasifikaci dokumentu charakterizovat jako úkol nahradit neznámou cílovou funkci $\breve{\Phi} : D \times C \rightarrow \{T, F\}$ (popisující jak by dokumenty měly být klasifikovány), funkcí $\Phi : D \times C \rightarrow \{T, F\}$ nazývanou jako klasifikátor (nebo model, hypotéza, pravidlo).

V této práci bude především klasifikací myšleno rozdělování elektronických dokumentů do tříd, které má společnost AEC definována jako veřejné, interní nebo chráněné (vizte obrázek 2.2). Tyto třídy spadají do skupiny *Multi-class*.



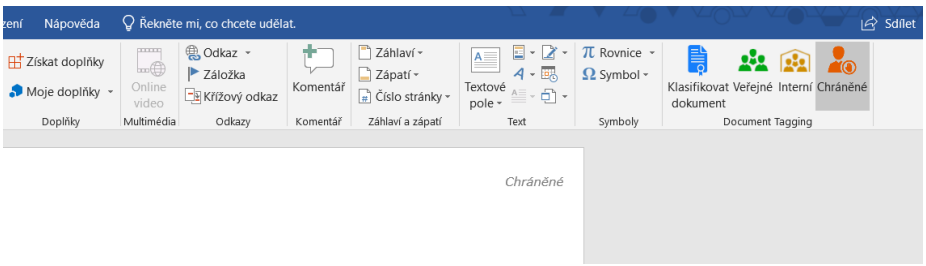
Obrázek 2.2: Členění dokumentů

2.1.1 Klasifikační stupně

Stupně značí míru důvěrnosti jednotlivých dokumentů ve společnosti. Firmy si je mohou pojmenovat různě, samozřejmě tak, aby vystihly samotnou důležitost. Tyto stupně jsou definovány taky slovně, většinou i směrnicí. V doplňku DocTag je pak možno přidat k těmto označením taky vizuální prvky, aby to bylo na první pohled v dokumentu zřejmé. Můžeme se pak setkat s vyobrazeným textem dané třídy, různou barvou, velikostí fontu, pozicí textu a s dalšími úpravami.

Další proměnnou klasifikačních možností je jejich počet. Pravidlem bývá, že čím vyšší počet tříd je na výběr, tím těžší je uživatelovo rozhodování. To ovšem není žádná novinka. Podobný jev můžeme pozorovat dennodenně v obchodech, když si vybíráme nějaký produkt, který má mnoho příchutí. Buď u toho strávíme mnoho času, anebo nás to natolik odradí, že si nevybereme nic. Takovou situaci by se jistě každý specialista AEC pokusil eliminovat. Ze zkušeností vyplývá, že firmy volí 3 až 5 stupňů. Počet tříd a jejich definice ve společnosti AEC jsou následující:

- Veřejné – veřejně přístupné informace, jejichž zveřejnění nemá negativní vliv na chod společnosti. Příkladem jsou volně přístupné publikovatelné informace. V dokumentu tento stupeň není reprezentován vizuální značkou.
- Interní – informace přístupné všem zaměstnancům nebo vybrané skupině, ale nikomu vně společnosti bez souhlasu vlastníka informací. Jejich zveřejnění nebo neoprávněný přístup může mít negativní vliv na provoz společnosti. Jsou to např. informace důležité pro provoz, vnitřní uspořádání a činnost společnosti, vnitřní předpisy apod. V souborech je tento stupeň vyznačen slovem „Interní“ v pravém horním rohu dokumentu.
- Chráněné – důvěrné informace jsou určené pouze vybraným jednotlivcům nebo skupině uživatelů. Nesmí být volně přístupné ostatním zaměstnancům ani subjektům vně společnosti. Do této kategorie patří osobní údaje, strategické informace, informace obchodního charakteru a jiné citlivé informace vyplývající z činnosti společnosti. Př.: smlouvy, osobní údaje, strategické informace, hesla do systému, informace uložené v CRM (Customer relationship management). Dokumenty s tímto označením jsou opatřeny textem „Důvěrné“ v pravém horním rohu dokumentu (vizte obrázek 2.3).



Obrázek 2.3: Dokument klasifikován stupněm chráněné

2.1.2 Předzpracování dokumentu

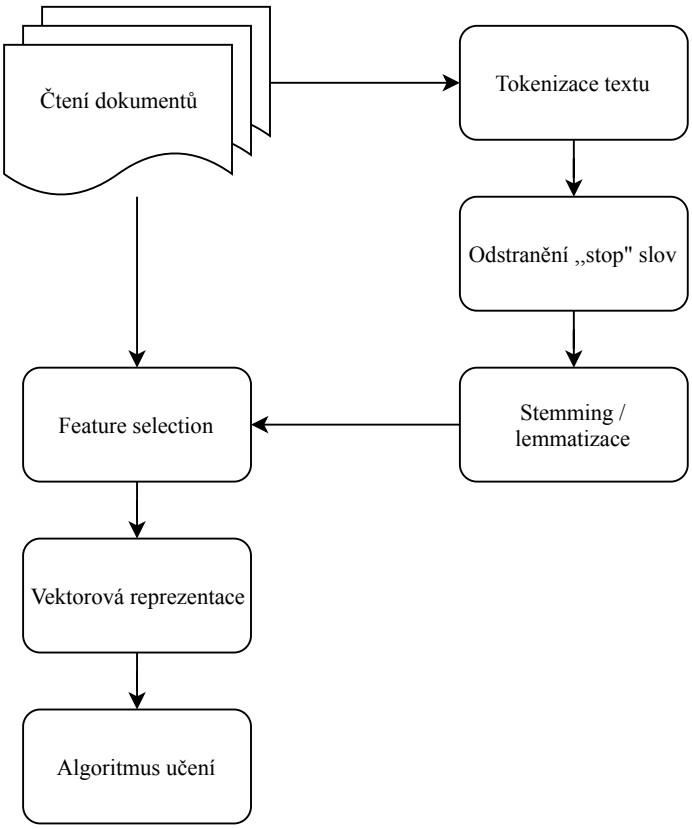
Ještě než se vykoná samotné vyhodnocení klasifikace dokumentu, je před tím potřeba udělat několik kroků. Tyto kroky nejsou naprosto nezbytné. Klasifikace může fungovat i bez nich,

ale pokud má algoritmus pracovat rychle a spolehlivě jsou skvělou pomocí. Samotná série kroků se nazývá předzpracování textu a jedná se o proces, který slouží k eliminaci faktorů, na kterých dané jazykové struktury v textu závisí. Podle literatury [15] se tento proces skládá ze dvou částí – *feature extraction*, což je první krok k „očesání“ dokumentu, aby se z něho stal čistý slovní formát. Do tohoto kroku patří tokenizace, odstranění „stop“ slov a „stemming“ nebo lemmatizace. Dokument sám o sobě obsahuje velké množství slov a mnoho z nich je irelevantních nebo vytváří šum. Často se taky používá tzv. redukce dimenzí – vyloučení velkého počtu klíčových slov, založená na pravděpodobnostním procesu a slouží k vytvoření menší dimenze vektoru.

Druhou částí je *feature selection* (FS), která zlepšuje škálovatelnost, efektivitu a přesnost textového klasifikátoru. Hlavním účelem FS je vybrat podmnožinu slov z původního dokumentu. Dělá se to tak, že se ponechávají slova s nejvyšší hodnotou podle předem definované míry důležitosti. Vybraná slova ponechávají původní význam a přináší lepší porozumění datům a algoritmu učení. Lepší náhled může poskytnout diagram 2.4.

Zdroj [18] potvrzuje, že mezi základní způsoby předzpracování dokumentu patří:

- tokenizace,
- odstranění „stop“ slov,
- lemmatizace nebo *stemming*,
- vektorizace.



Obrázek 2.4: Diagram předzpracování textu, převzato z [15]

Tokenizace

Je proces, který vezme vstupní text dokumentu a rozdělí ho na „tokeny“, což jsou významové celky. Zpravidla to bývají slova, ale mohou to být i fráze. Během toho je možné odstranit i interpunkční znaménka. Dále je možno nahrazovat např. bankovní účty, telefonní čísla nebo jiná data za „tokeny“, které vyjadřují, co tam bylo – konkrétní data nejsou potřeba. Taktéž se běžně konvertují velká písmena na malá. Důležité je také dát si pozor a uvažovat nad jednotlivými přirozenými jazyky, které jsou zpracovávány, protože např. moderní japonština kombinuje čínské znaky a západní znaky. Dalším úskalí, jak práce [8] zmiňuje, je vůbec určit daná slova jako „token“. Řešitel musí rozhodnout zda např. New York, vědecko-technický, nebo třeba 12 000 má být jedno slovo. Podle autora je vhodné, z hlediska dalšího zpracování, brát za slovo jednotku, co nejkratší.

Odstranění „stop“ slov

Takovým slovům je možno z publikace [18] porozumět tak, že nijak nenapomáhají určit kategorii dokumentu. V takovém případě je možno je bez váhání odstranit. Seznam těchto slov se většinou vytváří manuálně a to tak, že se seřadí všechna slova podle četnosti a poté podle uvážení, zda určují třídu, se smažou nebo nechají být.

Stemming

Při zpracování přirozeného jazyka není podstatný tvar zpracovávaných slov. Jedno slovo může být v textu použito v různých tvarech, ale význam nemění. Právě proto je důležité „odseknout“ části slova, po kterém dostaneme pouze základ. Takovému procesu se říká „stemming“ a v podstatě nám získává kořen slova. Nutno podotknout, že takto získaný kořen, nemusí odpovídat klasickému lingvistickému kořeni. Cílem je zajistit, aby se slovo v různých tvarech přiřadilo pouze na jediné slovo tzv. *stem*. Za jeden z nejznámějších „stemmovacích“ algoritmů je považován Porterův algoritmus [31]. Běžné použití „stemmeru“ můžeme očekávat u internetových vyhledávačů.

Lemmatizace

Je velmi podobný proces „stemmingu“, ale jedná se o něco složitější přístup. Využívá syntaktických pravidel daného jazyka pro určení příslušného slovního druhu konkrétních slov. Načež se využije pravidel pro daný slovní druh, aby bylo možné získat základní tvar slova. Konkrétní příklad v anglickém jazyce znázorňující rozdíl na výstupu mezí „stemmingem“ a lemmatizací je možno vidět na příkladu níže. Byl převzat z publikace [13].

Stemming:
introduction, introducing, introduces ⇒ introduc
gone, going, goes ⇒ go
Lemmatizace:
introduction, introducing, introduces ⇒ introduce
gone, going, goes, went ⇒ go

Vektorizace

Při zpracování dokumentů se často používá jejich vektorová reprezentace, kde každá část vektoru představuje určitý *token*. Dokument je zpracovaný jako vektor v mnohorozměrném prostoru, kde platí, že vektory dokumentů, které mají hodně společných „tokenů“, tak mají

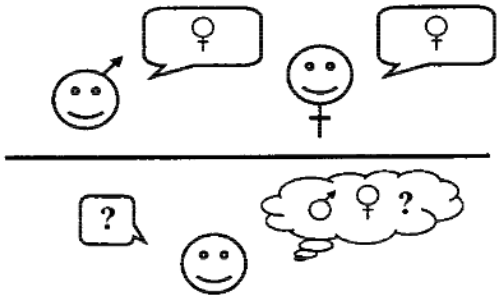
v prostoru od sebe menší vzdálenost. Platí to ovšem i naopak, vektory dvou souborů, které používají úplně jiné výrazy, budou mít vzdálenost od sebe větší. Vytváření shluků podobných dokumentů se využívá při klasifikaci bez učitele. Zdroj [18] uvádí několik možností převodu dokumentu do vektorové podoby. Nejjednodušším případem je převod, kdy jednotlivé části vektoru mohou nabývat pouze hodnot 1 a 0 a jsou určeny podle toho, zda daný *token* vektor obsahuje či nikoliv (v literatuře o strojovém učení je možno se setkat s výrazem „one hot encoding“). Další možností by bylo přihlédnout taky k počtu výskytů jednotlivých „tokenů“ v dokumentu. Nicméně, poté je nutno uvážit i délku dokumentu pomocí normalizace vektoru. Často se využívají i různé statistiky o trénovací sadě v tzv. inverzní dokumentové frekvenci (*TF-IDF*), což je transformace, kdy se „tokenu“, který se často vyskytuje ve velkém počtu dokumentů přiřazuje menší váha jako „tokenu“, který se nachází v menším počtu dokumentů. Při klasifikaci textu s vektorovou reprezentací dokumentů se používají normalizované vektory.

2.2 Umělá inteligence

V dnešní době se můžeme setkat s pojmem umělá inteligence poměrně často. Co si tedy může člověk pod takovým výrazem vlastně představit? Jako programátor mohu tvrdit, že jakýkoliv software je umělá inteligence, protože algoritmus dokáže odpovídat na možné vstupy, které může uživatel zadat nebo mu bude odpovídat na základě jeho chování. Ačkoliv to nemusí nutně znamenat umělou inteligenci. Pravá umělá inteligence (dále jen UI) je počítačový systém, který se dokáže učit sám od sebe. Dokáže se zlepšovat na základě minulých iterací, získávat dovednosti a znalosti. Takový druh UI, který můžete vidět ve skvělých scifi filmech v televizi nebo knihách, jako je např. *Já, robot* je stále daleko v budoucnosti. Dnes se spíše setkáte s tzv. pseudo UI, která zpracovává řeč, obraz nebo předpovídá možné scénáře. Příkladem může být Siri od společnosti Apple, což je virtuální asistentka, která odpovídá, co nejrelevantněji může, na uživatelovy dotazy. Nebo například Tesla automobil, který sice ještě není zcela autonomní, ale pomalu a jistě se k tomu blíží. Prozatím se dokáže skvěle řídit dopravními značkami a předpisy.

2.2.1 Turingův test

Britský matematik Alan Turing v 50. letech 20. století vytvořil test, který se používá jako důkaz inteligence stroje nebo algoritmu. Tento test [11] je vlastně imitační hra, kdy ve dvou oddělených místnostech sedí testující člověk a ve druhém je počítač s programem a nějaký další člověk. Testující následně pokládá otázky v přirozené řeči a předává je do druhé místnosti. V této druhé místnosti otázky zodpovídá buď počítač nebo druhý člověk. Odpovědi jsou poté odeslány zpět tazateli a pokud testující člověk nedokáže rozpoznat, jestli komunikuje se strojem nebo s člověkem, pak tento algoritmus splňuje Turingův test. Původní verze testu podle zdroje [26] zahrnovala muže a ženu ve stejné místnosti a tazatele v místnosti druhé. Úkolem tazatele bylo rozpoznat, který z dvojice je žena a každý z dvojice měl za úkol přesvědčit tazatele, že je pouze on/ona žena. Ke konverzaci byly povolené veškeré témata od matematiky po poezii nebo např. od počasí po šachy. Postupem času se test upravil a žena byla nahrazena počítačovým programem. Proč tomu tak bylo, je možné se dočíst blíže v práci [26].



Obrázek 2.5: Schéma původního Turingova testu, převzato z [26]

2.2.2 Techniky UI

Podle Bullinaria [2] má UI identifikované kořeny ve větším počtu starších vědních oborů jako jsou: filozofie, matematika, psychologie či biologie.

Právě proto nyní existuje mnoho odvětví zabývající se touto disciplínou a používají mnoho různých technik. Mnoho z nich využívá jak inženýrské aspekty, tak i vědecké. Příkladem mohou být:

- Neuronové sítě – Schmidhuber [27] popisuje standardní neuronové sítě jako propojené procesory zvané neurony, kde každý z nich vytváří sekvenci aktivací nějaké reálné funkce. Vstupní neurony se aktivují přes senzory vnímající dané prostředí. Ostatní neurony jsou aktivovány přes váhová propojení z předešlých aktivovaných neuronů. Některé neurony mohou ovlivňovat prostředí tím, že spouští různé akce. Používají se na klasifikaci, modelování mozku a další. Jak vypadá model takové sítě je možno vidět na obrázku 2.6.

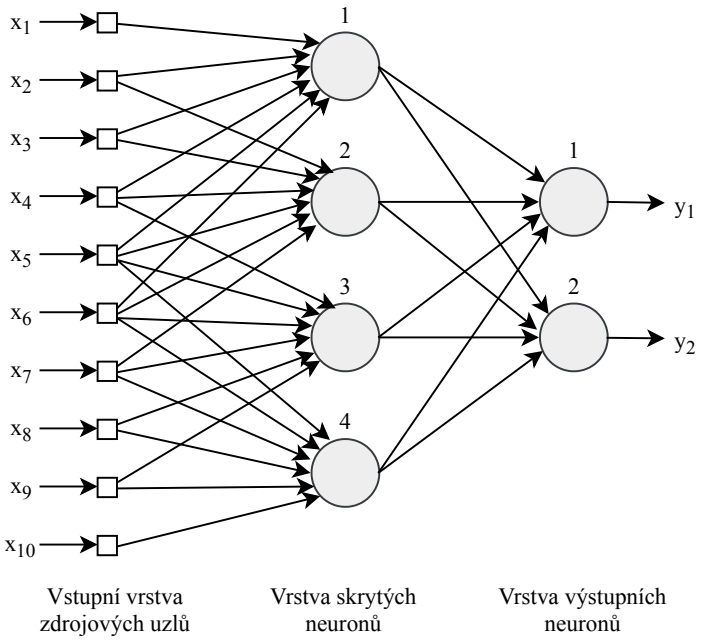
- Evoluční algoritmy – zdroj [12] popisuje, že standardní úlohou těchto algoritmů je optimalizace řešení, i když téměř nikdy negarantují nalezení optimálního řešení v rozumné době. Tyto algoritmy jsou běžně reprezentovány v genetických algoritmech, genetickém programování.

- Strojové vidění – kniha [9] uvádí strojové vidění jako disciplínu se silnými vazbami v matematice, počítačové vědě a se slabší vazbou na fyziku, psychologii vnímání a neurovědu. Prakticky se používá pak na rozpoznání objektů a porozumění obrázku.

- Robotika – je obor, který se zabývá designem, konstrukcí a použitím strojů (robotů) za účelem provedení nějaké úlohy, kterou běžně vykonávají lidé. V praxi se pak může jednat o autonomní prozkoumávání, inteligentní řízení (např. pohyb robotickou rukou).

- Expertní systémy – pramen [17] zmiňuje, že expertní systém slouží k zahrnutí lidské expertizy takovým způsobem, aby lidé, kteří nemají takovou znalost mohli získat vědomosti na požádání přes dotazovací systém. Mohou to být kupříkladu podpůrné rozhodovací systémy, vyučovací systémy aj.

- Zpracování řeči – slouží k automatickému převodu mluvené řeči do textu. Samotné rozpoznávání můžeme dělit na závislé na mluvčím nebo nezávislé. Jedná se o velmi výpočetně náročný proces. Praktických příkladů je mnoho, např. ovládání počítače nebo jiného elektronického zařízení pomocí hlasu.



Obrázek 2.6: Částečně propojená dopředná neuronová síť, převzato z [10]

- Zpracování přirozeného jazyka – článek [3] vysvětluje tento vědní obor jako teoreticky motivovaný rozsah výpočetních technik pro automatickou analýzu a reprezentaci lidského jazyka. Klasickou úlohou tohoto oboru může být strojový překlad, automatická oprava textů, automatické vytváření referencí, získávání informací z databáze.
- Plánování – je cílově orientovaný rozhodovací proces. Umělá inteligence pomocí různých algoritmů vyhodnocuje možné kroky v daných situacích. Pokud je tak naprogramována, může dané kroky i dělat. Často se plánování užívá právě ve hrách.
- Strojové učení – podoblast zabývající se vývojem umělé inteligence, jejíž cílem je učit se z dat a přizpůsobovat se změnám. Běžným příkladem je indukce rozhodovacích stromů, prohledávání prostoru verzí.

2.2.3 Společné aspekty

Podle výše zmíněného autora [2] i mnohem odlišné systémy (kupříkladu expertní systémy založené na pravidlech a neuronové sítě) mají mnoho společných aspektů. Čtyři důležité jsou:

- Reprezentace – vědomosti musejí být nějak reprezentovány - možná jako řada „if-then“ pravidel, formou sémantické sítě nebo v podobě propojených vahách umělé neuronové sítě.
- Učení – automatické získávání vědomostí z prostředí - jako je získání pravidel pro expertní systém nebo určení vah pro neuronovou síť.
- Pravidla – explicitně mohou být vestavěné do expertního systému „knowledge“ inženýrem nebo implicitně ve váhách naučených neuronovou sítí.

- Vyhledávání – může mít více podob, například hledání sekvence stavů, které rychle vedou k vyřešení problému nebo vyhledání vhodné sady vah pro neuronovou síť díky minimalizaci ohodnocující (*fitness*) funkcí.

2.2.4 Základní dělení

Zdroj [24] uvádí, že existují tři typy zpětné vazby, které určují tři hlavní způsoby učení:

- Učení bez učitele – „agent“ se učí vzory na vstupu, i když žádná zpětná vazba není poskytnuta. Nejběžnější úloha bez učitele je „clustering“, což je detekování potenciálně použitelných „clusterů“ na vstupu. Např. taxi agent by si mohl vyvinout koncept „dobrých dopravních dnů“ a „špatných dopravních dnů“ bez jakýchkoliv označených příkladů od učitele.
- Zpětnovazební učení – agent se učí ze sérií zpětné vazby – odměny nebo tresty. Například, když taxi agent nedostane spropitné na konci cesty, indikuje to, že něco udělal špatně. Případně agent může získat dva body za výhru na konci partie v šachách, což značí, že udělal něco správně. Agent potom musí rozhodnout, které akce nejvíce vedly ke zpětné vazbě.
- Učení s učitelem – agent pozoruje nějaké příklady, které přichází v páru. Prvním z dvojice je vstup a druhým je výstup. Pro lepší porozumění si můžeme představit, že náš agent má rozpoznávat objekty na obrázku. Pošleme mu tedy obrázek a řekneme mu, že je to autobus.

Ve skutečnosti tato rozdělení nejsou tak jednoznačná. V kombinovaném učení (učení s učitelem a bez něho) dostáváme pár označených příkladů a musíme si nějak poradit s větší kolekcí neoznačených vstupů. Dokonce ani označené příklady nemusejí být někdy stoprocentně pravdivě, jak doufáme. Představte si, že chcete vytvořit systém na rozpoznávání věku osob na fotkách. Nasbíráte nějaké označené vzorky tím, že vyfotíte lidi a zeptáte se na jejich věk – to je učení s učitelem. Nicméně ve skutečnosti, lidé mohou lhát o svém věku. Není to jen o tom, že v datech je náhodný šum. Nepřesnosti jsou spíše systematické a abychom je odkryli, tak je nutné využít učení bez učitele, které zahrnuje: obrázky, věk nahlášený lidmi a skutečný (neznámý) věk. Tudíž jak šum, tak nedostatek označených vstupů vytvářejí spojení mezi učením s učitelem a učením bez něho.

2.2.5 Slovníček pojmů při učení s učitelem

Chtěl bych v této podsekci stručně představit základní terminologii tohoto způsobu učení. Jedná se totiž o běžné pojmy v literatuře.

Trénovací sada

Úloha učení s učitelem je následující: mějme trénovací sadu o N příkladech vstupních-výstupních párů $(x_1, y_1), (x_2, y_2), ...(x_N, y_N)$, kde každé y_j bylo vygenerováno neznámou funkcí $y = f(x)$. Najděme pak funkci h, která se přibližuje pravé funkci f.

Hypotéza

V příkladu výše může x a y nabývat jakékoliv hodnoty, nemusí to být ani nutně čísla. Funkce h je hypotéza. Učení je hledání jedné hypotézy, z prostoru všech možných hypotéz, která bude správně vyhodnocovat, dokonce i na nových příkladech mimo trénovací sadu.

Testovací sada

Pro měření přesnosti hypotézy pak předkládáme testovací sadu příkladů, která je odlišná od trénovací sady.

Generalizace

Říkáme, že hypotéza h zobecňuje (generalizuje) dobře, pokud správně předpovídá hodnotu y nových příkladů. Někdy je funkce f stochastická – není to přímo funkce nějakého x, ale co musíme zjistit je podmíněné pravděpodobnostní rozdělení, $P(Y|x)$.

Klasifikace

Když je výstup y jedna hodnota z množiny konečných stavů (např. slunečno, oblačno, zataženo, přeháňky), tak je problém učení nazýván klasifikací.

Regrese

Pokud je hodnota y číslo (třeba počet stupňů), pak se problému učení říká regrese. (Technicky vzato podle [24], řešením regresního problému je nalezení podmíněné pravděpodobnosti nebo průměrné hodnoty z y, protože pravděpodobnost, že jsme našli přesnou hodnotu z množiny reálných čísel pro dané y je nulová.)

2.2.6 Používané metody

- Algoritmus k-nejbližších sousedů – podle publikace [24] jde o nalezení k prvků, které jsou nejblíže k prvku x_q. Pro klasifikaci se nejprve najde množina $NN(k, x_q)$, poté se provede pluralitní hlasování sousedů a podle většiny hlasů se přiřadí hledaný prvek x_q do příslušného klasifikačního stupně.

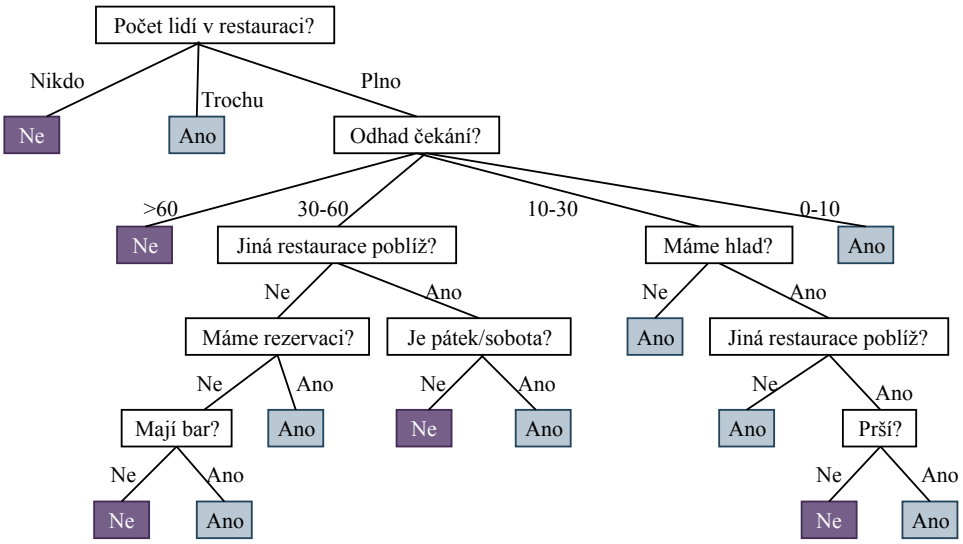
- Podpůrné vektory (Support vector machines) – kniha [32] popisuje SVM jako vybírání menšího počtu kritických hraničních instancí z každé třídy, zvaných podpůrné vektory a vytvoření diskriminační funkce. Tento instančně orientovaný návrh překračuje omezení lineárních hranic, tím že zahrnuje extra nelineární výrazy do funkce, která umožňuje vytvořit kvadratické, kubické a vyšší řády hranic. Tato stejná technika může být uplatněna i na perceptron nebo „least square“ regresi, aby obsahovala komplexnější rozhodovací hranice.

- Bayesovské sítě [24] – sítě mohou představovat v podstatě jakékoliv plně sdružené pravděpodobnostní rozdělení. Bayesovská síť je orientovaný graf, který je v každém vrcholu popsán kvantitativní informací o pravděpodobnosti. Plná specifikace je následující:

  1. Každý vrchol odpovídá náhodné proměnné, která může být diskrétní nebo spojitá.
  2. Sada orientovaných hran nebo šipek propojuje páry vrcholů. Pokud směřuje šipka z vrcholu X do vrcholu Y, X označujeme jako rodiče Y. Graf nemá orientovanou kružnici (označován jako orientovaný acyklický graf).
  3. Každý vrchol X_i má podmíněné pravděpodobnostní rozdělení $P(X_i|Rodiče(X_i))$, které popisuje vliv rodičů na vrchol.

- Neuronové sítě – je kolekce jednotek spojených dohromady. Vlastnosti sítě jsou určeny její topologií a vlastnostmi neuronů. Jednoduše řečeno – neurony v síti „sepínají“, když nějaká lineární kombinace vstupů přesáhne nějaký práh (implementace lineárního klasifikátoru).

- Rozhodovací stromy – literatura [24] uvádí, že tento model patří k nejjednodušším a přesto nejúspěšnějším formám strojového učení. Rozhodovací strom představuje funkci, která přijímá vektor atribučních hodnot a vrací „rozhodnutí“ – jednu výstupní hodnotu. Vstupní i výstupní hodnoty mohou být diskrétní nebo spojité. Strom dojde k rozhodnutí na základě provedení sekvence testů. Každý vnitřní uzel ve stromu odpovídá jednomu testu hodnoty, která je na vstupu a větve vedoucí z daného uzlu jsou označeny možnými hodnotami atributu. Reprezentace rozhodovacích stromů je přirozená pro nás lidi. Vskutku mnoho manuálů je popsáno jako jeden velký rozhodovací strom. Lépe to dokáže vystihnout obrázek 2.7, který popisuje situaci, kdy zákazníci přijdou do restaurace a rozhodují se, zda-li mají čekat na stůl.



Obrázek 2.7: Rozhodovací strom čekání na stůl v restauraci, převzato z [24]

Pro dosažení lepších výsledků je doporučeno modely kombinovat, známé techniky kombinování jsou:

- Bootstrap aggregating („Bagging“) – je první efektivní metoda podle [24], která kombinuje hypotézy naučené z více „bootsrapových“ datových sad, kde je každá hypotéza vygenerovaná pod-vzorkováním původní datové sady.

- Boosting – podle zdroje [7] se jedná o obecnou a prokazatelně účinnou metodu tvorby velmi přesného predikčního pravidla, které je vytvořeno kombinací hrubých a mírně nepřesných pravidel.

- Stacking – článek [25] představuje tuto techniku jako vytváření souboru klasifikátorů. Souborem je myšlena sada klasifikátorů, jejíž jednotlivé výstupy (rozhodnutí) jsou nějakým způsobem kombinovány, aby klasifikovaly nové vstupní data.

Pro testování přesnosti modelu je možno využít:

- Křížové validace – pramen [33] uvádí, že je to převzorkovací technika, která využívá mnoho náhodných trénovacích a testovacích dat. Výhodou křížové validace je to, že všechna pozorování nebo vzory v dostupném vzorku jsou použity pro testování a mnoho z nich jsou taky použity pro trénování modelu. Analýza křížové validace přináší cenné poznatky o spolehlivosti neuronových sítí s ohledem na variaci vzorkování.

- Bootstrap statistiky – podle autora Foxe [6] se jedná o obecný přístup k statistické inferenci sestavením výběrového rozdělení („sampling distribution“) z dat, co máme k dispozici. Taktéž se jedná o převzorkovací techniku, ale na rozdíl od křížové validace, tato technika využívá nahrazování stávajících dat, čímž vytváří data nová.

2.3 Klasifikátory

V posledních letech autoři Mahender a Kord [16] zmiňují, že úloha automatické klasifikace textů je značně studována a je zaznamenán obrovský pokrok v této oblasti. Dále zmiňují, že žádné reprezentační schéma a klasifikátor nemůže být prohlášen za model obecný pro jakoukoliv aplikaci. Různé algoritmy mají různé výsledky v závislosti na sběru dat.

2.3.1 Naivní Bayesův klasifikátor

Vychází z Bayesovy věty o podmíněných pravděpodobnostech, která je hojně zkoumána v souvislosti se strojovým učením a taky v systémech pro dobývání znalostí. Tento model je jeden z nejjednodušších modelů bayesovských klasifikátorů a předpokládá, že všechny atributy (nebo příznaky, v angličtině „features“) daného vzorku jsou na sobě nezávislé, vzhledem ke kontextu klasifikační třídy. Jinými slovy v souvislosti s klasifikací textu - každé slovo v daném dokumentu (nebo i jednodušeji každé slovo ve větě) není závislé na ostatních slovech. Navzdory tomu, že je ve valné většině případů tento předpoklad špatný, dosahuje klasifikace tímto způsobem velmi dobrých výsledků. Kvůli předpokladu [19] nezávislosti je možno naučit se parametry každého atributu odděleně a to značně zjednodušuje učení, i když je počet atributů obrovský. Podle zdroje [19] tento způsob klasifikace zvládá přesně určit klasifikaci jednoduchých dokumentů s velikostí slovníku menší než 100. Nicméně komplexnější úlohy na reálných datech z webu, UseNet a „newswire“ článků, zvládá vyhodnotit se slovníky o velikosti v řádů tisíců.

Přes velkou popularitu „naivní Bayes“ způsobil jisté zmatení v komunitě zabývající se klasifikováním dokumentů. Důvodem je to, že zde existují dva rozdílné modely, které se běžně používají. Oba používají předpoklad nezávislosti a oba se nazývají stejně, pokud tedy není blíže specifikovaná varianta.

První model reprezentuje dokument jako vektor binárních atributů značící, která slova se vyskytují a která ne. Na počet výskytů daného slova není brán zřetel. Při výpočtu pravděpodobnosti dokumentu se násobí pravděpodobnost všech hodnot atributů, včetně pravděpodobnosti nevyskytujících se slov, která se v dokumentu neobjevují. S takovým přístupem je možno se setkat běžně u bayesovských sítí, který je vhodný pro úlohy, kde máme pevně daný počet atributů. Takovýto druh je označován jako „multi-variate“ Bernoulliho model.

Druhý model vnímá dokument jako počet výskytů jednotlivých slov (ve strojovém učení se označuje jako „bag of words“). Stejně jako v prvním modelu, je pořadí slov ztraceno a při výpočtu pravděpodobnosti se násobí slova, která se vyskytují. Tento typ se nazývá multinomialní model a jedná se o běžnější a základnější variantu. Takový postup používají

výzkumníci v statistickém modelování jazyka pro zpracování řeči pod názvem „unigram language model“. Pochopitelně se stejný přístup využívá i u klasifikace textu.

Základní pojmy

Bayesova věta vychází z následujícího:

- $P(A)$ je pravděpodobnost, že nastal jev A,
- $P(B|A)$ je pravděpodobnost jevu B, za podmínky, že nastal jev A,
- $P(A \cap B)$ je pravděpodobnost, že nastal jev A i B.

$$P(A \cap B) = P(B|A) \cdot P(A) \tag{2.1}$$

$$P(B \cap A) = P(A|B) \cdot P(B) \tag{2.2}$$

$$\text{Protože platí } P(A \cap B) = P(B \cap A), \text{ pak:} \tag{2.3}$$

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2.4}$$

Rovnici 2.4 je dále možno v literatuře [23] rozdělit na výrazy:

1. $P(A|B)$ **Aposteriorní pravděpodobnost** (posterior) – pravděpodobnost hypotézy A poté, co jsme viděli data B,
2. $P(B|A)$ **Věrohodnost** (likelihood) – pravděpodobnost dat B, když hypotéza A je pravdivá,
3. $P(A)$ **Apriorní pravděpodobnost** (prior) – pravděpodobnost hypotézy A před tím, než vidíme nějaká data,
4. $P(B)$ **Normalizační konstanta** – pravděpodobnost dat B.

Naivní Bayesův předpoklad zmiňuje, že všechny příznaky jsou pravděpodobnostně nezávislé a pro třídu c ho můžeme vyjádřit podle zdroje [20] následovně:

$$p(x|y = c) = \prod_{i=1}^{D} p(x_i|y = c) \tag{2.5}$$

Klasifikace pomocí Bayese

Předpokládejme, že chceme přiřadit dokument do jedné z tříd C (např. $C = 2$ a třídami jsou spam a legitimní zpráva). Jednoduchá reprezentace, pomocí modelu zvaného „bag of words“, ignoruje pořadí slov a pouze vyhodnocuje počet výskytů pro každé slovo zvlášť. Řekněme, že je v jazyce D slov (můžeme vždy vynutit mapování všech slov, která nespadají do slovníku daného jazyka, na speciální symbol „unk“, vyjadřující neznámé slovo). Potom může být dokument reprezentován jako p-vektor výskytů (histogram frekvence slov). Nechť $X = k$ znamená, že slovo se objevuje přesně k-krát. Pro $k = 0 : K - 1$. (Pokud se slovo vyskytuje

vícekrát než $K-1$ v dokumentu, budeme se k němu chovat jako kdyby se objevovalo $K-1$ krát. K je zde uživatelem zvolená vrchní hranice.) V takovém případě, můžeme podmíněnou hustotu pravděpodobnosti vyjádřit jako součin mnohočlenů [20]:

$$p(x|Y=c,\theta) = \prod_{i=1}^{D}\prod_{k=1}^{K}\theta_{ick}^{I(x_i=k)} \tag{2.6}$$

kde θ_{ick} je pravděpodobnost pozorování i-tého slova, které má počet výskytů k při zkoumané třídě c. Myšlenka za tím je taková, že počet výskytů pro dané slovo v dokumentu, by měl dávat jakési povědomí o typu dokumentu.

$$\theta_{ick} = p(X_i = k|Y = c) \tag{2.7}$$

Dokonce mnohem jednodušší reprezentací může být zjišťování, zda se slovo vyskytuje, nebo ne, přičemž nás nezajímá kolikrát. Mějme binární příznakový vektor x. V takovém případě, je možno vyjádřit hustotu podmíněné pravděpodobnosti jako součin binomického rozdělení (Bernoulliova schéma) pomocí:

$$p(x|Y=c,\theta) = \prod_{i=1}^{D}\theta_{ic}^{xi}(1-\theta_i c)^{1-x_i} \tag{2.8}$$

kde θ_{ic} je pravděpodobnost slova i, vyskytující se ve třídě c. Dále $x_i = 1$ znamená, že slovo i se vyskytuje a $x_i = 0$ je opakem. Navzdory jednoduchosti modelu, řešení funguje překvapivě dobře a je základem většiny spamových filtrů, jak uvádí článek [20].

Příklad vyhodnocení

Rád bych v této podsekci představil jednoduchý příklad, na kterém je možno vidět, jak tento algoritmus funguje. Tento příklad byl převzat ze zdroje [14] a znázorňuje, zda člověk půjde ven dělat nějaký sport či nikoliv. Mějme trénovací sadu, kterou je možno reprezentovat tabulkou 2.2 níže:

Den	Stav	Teplota	Vlhkost	Vítr	Sportovat?
1	slunečno	horko	vysoká	slabý	ne
2	slunečno	horko	vysoká	silný	ne
3	zataženo	horko	vysoká	slabý	ano
4	přeháňky	mírná	vysoká	slabý	ano
5	přeháňky	chladno	normální	slabý	ano
6	přeháňky	chladno	normální	silný	ne
7	zataženo	chladno	normální	silný	ano
8	slunečno	mírná	vysoká	slabý	ne
9	slunečno	chladno	normální	slabý	ano
10	přeháňky	mírná	normální	slabý	ano
11	slunečno	mírná	normální	silný	ano
12	zataženo	mírná	vysoká	silný	ano
13	zataženo	horko	normální	slabý	ano
14	přeháňky	mírná	vysoká	silný	ne

Tabulka 2.2: Trénovací data, převzato z [14]

Nyní si z nasbíraných dat můžeme vytvořit statistiku, která bude znázorňovat frekvenci a pravděpodobnost každého atributu v tabulce. Prvně si spočítáme v kolika případech jsme sportovali: $p(\text{Sportovat} = Ano) = 9/14$. Z toho dostaneme i počet, kdy jsme nesportovali: $p(\text{Sportovat} = Ne) = 5/14$. Následně vyjádříme každý atribut v závislosti na aktivitě v daných dnech. Za zmínku stojí, že tento příklad obsahuje dvě třídy (sportovat/nesportovat) a tedy se jedná o binární klasifikaci.

Stav	**Sportovat=Ano**	**Sportovat=Ne**
slunečno	2/9	3/5
zataženo	4/9	0/5
přeháňky	3/9	2/5

Tabulka 2.3: Tabulka stavu počasí

Teplota	**Sportovat=Ano**	**Sportovat=Ne**
horko	2/9	2/5
mírná	4/9	2/5
chladno	3/9	1/5

Tabulka 2.4: Tabulka teplot

Vlhkost	**Sportovat=Ano**	**Sportovat=Ne**
vysoká	3/9	4/5
normální	6/9	1/5

Tabulka 2.5: Tabulka vlhkosti

Vítr	**Sportovat=Ano**	**Sportovat=Ne**
silný	3/9	3/5
slabý	6/9	2/5

Tabulka 2.6: Tabulka síly větru

Když už máme data vyjádřena, je čas na testovací fázi. Dejme tomu, že chceme zjistit jestli máme jít dnes sportovat na základě počasí. Dnes je venku slunečno, chladno, vlhkost je vysoká a vítr silný. Nebo můžeme zapsat jako $x' = (\text{Stav} = \textit{slunečno}, \text{Teplota} = \textit{chladno}, \text{Vlhkost} = \textit{vysoká}, \text{Vítr} = \textit{silný})$. Najdeme si tedy pro každý atribut pravděpodobnost pro třídu sportovat:

- $p_{11} = p(\text{Stav} = \textit{slunečno}|\text{Sportovat} = Ano) = 2/9$
- $p_{12} = p(\text{Teplota} = \textit{chladno}|\text{Sportovat} = Ano) = 3/9$
- $p_{13} = p(\text{Vlhkost} = \textit{vysoká}|\text{Sportovat} = Ano) = 3/9$

- $p_{14} = p(\text{Vítr} = \textit{silný}|\text{Sportovat} = \textit{Ano}) = 3/9$
- $p_A = p(\text{Sportovat} = \textit{Ano}) = 9/14$

Poté uděláme to stejné pro třídu nesportovat:

- $p_{21} = p(\text{Stav} = \textit{slunečno}|\text{Sportovat} = Ne) = 3/5$
- $p_{22} = p(\text{Teplota} = \textit{chladno}|\text{Sportovat} = Ne) = 1/5$
- $p_{23} = p(\text{Vlhkost} = \textit{vysoká}|\text{Sportovat} = Ne) = 4/5$
- $p_{24} = p(\text{Vítr} = \textit{silný}|\text{Sportovat} = Ne) = 3/5$
- $p_N = p(\text{Sportovat} = Ne) = 5/14$

Nakonec pomocí klasifikačního pravidla maximální aposteriorní pravděpodobnosti vynásobíme dané atributy spolu s pravděpodobností, zda sportovat či nikoliv a to pro každou třídu:

$$p(\text{Ano}|x') = (p_{11} \cdot p_{12} \cdot p_{13} \cdot p_{14}) \cdot p_A \tag{2.9}$$
$$p(\text{Ano}|x') \approx 0.0053$$
$$p(\text{Ne}|x') = (p_{21} \cdot p_{22} \cdot p_{23} \cdot p_{24}) \cdot p_N \tag{2.10}$$
$$p(\text{Ne}|x') \approx 0.0206$$

Nyní na základě porovnání těchto dvou výsledků klasifikátor zvolí vyšší hodnotu a vrátí pro daný testovací příklad třídu. V tomto případě bude závěrem vyhodnocení „nesportovat“.

Metoda Log-Sum-Exp

Jeden praktický problém vyvstává, když využíváme generativní klasifikátory. Konkrétně to, že $p(\vec{x}|Y)$ je často velmi malé číslo. Pravděpodobnost pozorování nějakého určitého vysoce dimenzionálního vektoru je velice malá, když $\sum_{\vec{x}} p(\vec{x}|Y) = 1$. Právě to, může vést k podtečení, což je stav, kdy počítačový program provádí nějaký výpočet a výsledek je natolik malý, že ho počítač nemůže uložit do paměti. Díky metodě (Log-Sum-Exp) převodu čísel do logaritmické soustavy je možno pracovat s malými čísly. Tato metoda se obecně používá ve strojovém učení nejen u bayesovského klasifikátoru. Jak celý proces funguje je možno vidět níže na příkladu převzatého z [5]. Mějme n-dimenzionální vektor a chceme spočítat:

$$y = \log \sum_{i=1}^{n} e_i^x \tag{2.11}$$

Například počítáme aposteriorní pravděpodobnost rekurzivně:

$$p(h_t|v_{1:t}) \equiv \alpha(h_t) = p(v_t|h_t) \sum_{h_t} \alpha(h_{t-1}) p(h_t|h_{t-1}) \tag{2.12}$$

Protože α může být docela nízké, je běžné řešit tento problém v logaritmické soustavě:

$$\log \alpha(h_t) = \log p(v_t|h_t) + \log \sum_{h_t} \exp\left(\log \alpha(h_{t-1}) + \log p(h_t|h_{t-1})\right) \tag{2.13}$$

Autor Eisele [5] dokazuje, že následující rovnice platí:

$$\log \sum_{i=1}^{n} e^{x_i} = a + \log \sum_{i=1}^{n} e^{x_i - a} \tag{2.14}$$

„Pro libovolné a. To znamená, že je možno posunout střed exponenciální sumy. Typická hodnota pro a je maximum, které způsobuje, že nejvyšší hodnota je nula, a dokonce i kdyby ostatní hodnoty podtekly, dostanete rozumný výsledek:“

$$a = max x_i \tag{2.15}$$

Důkaz [5] log-sum-exp metody:

$$y = \log \sum_{i=1}^{n} e^{x_i} \tag{2.16}$$

$$\Leftrightarrow \quad e^{y} = \sum_{i=1}^{n} e^{x_i} \tag{2.17}$$

$$\Leftrightarrow \quad e^{-a} e^{y} = e^{-a} \sum_{i=1}^{n} e^{x_i} \tag{2.18}$$

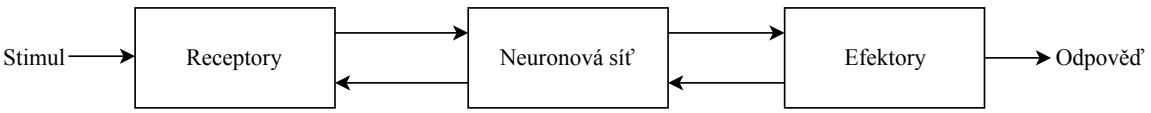
$$\Leftrightarrow \quad e^{y-a} = \sum_{i=1}^{n} e^{-a} e^{x_i} \tag{2.19}$$

$$\Leftrightarrow \quad y - a = \log \sum_{i=1}^{n} e^{x_i - a} \tag{2.20}$$

$$\Leftrightarrow \quad y = a + \log \sum_{i=1}^{n} e^{x_i - a} \tag{2.21}$$

2.3.2 Umělá neuronová síť

Na lidský mozek může být nahlíženo jako třífázový systém (obrázek 2.8). Centrálním prvkem tohoto systému je mozek, který je tvořen formou neuronové sítě, které přijímají informace, vnímají je a dělají vhodná rozhodnutí. Umělé neuronové sítě vycházejí z reálného fungování našeho mozku, proto taky nesou stejný název. Synapse nebo nervová zakončení jsou elementární strukturální a funkční jednotky, které zprostředkovávají interakce mezi jednotlivými neurony. Jak jsou tyto sítě modelovány pomocí neuronů v počítači se můžete dočíst níže.



Obrázek 2.8: Diagram znázorňující nervový systém, převzato z [10]

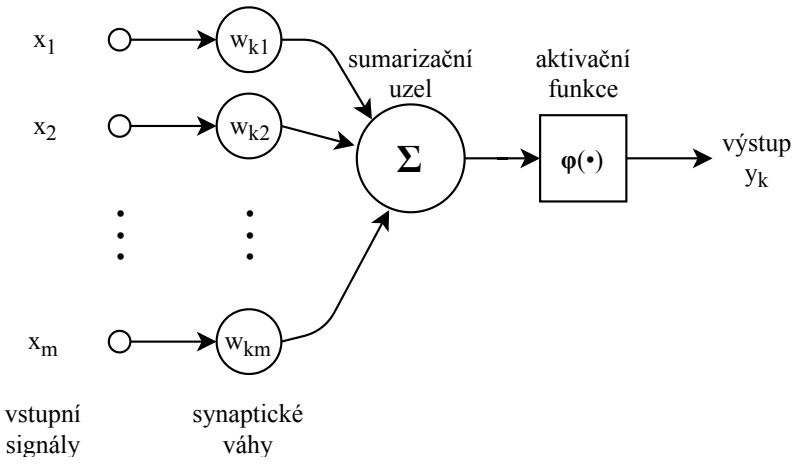
Základní pojmy

Literatura [1] popisuje, že základním „kamenem“ je neuron nebo taky *jednotka*. Každá jednotka přijímá množinu vstupů (obrázek 2.9), které jsou vyjádřeny vektorem $\vec{X}_i$, který

v tomto případě koresponduje s frekvencemi slov v $i$tém dokumentu. Každý neuron je taky spojen s množinou vah A, které se používají pro výpočet vstupů funkcí $f(\cdot)$. Typickou funkcí, která se běžně používá v neuronových sítích je následující lineární funkce:

$$p_i = A \cdot \vec{X_i} \tag{2.22}$$

Takže vektor $\vec{X_i}$ je sestaven z lexikonu slov d. Váhový vektor A by měl rovněž obsahovat prvky z d. Nyní předpokládejme, že řešíme binární klasifikační problém, kdy označení bereme z množiny $\{+1, -1\}$. Autoři pak v publikaci [1] předpokládají, že označení třídy vektoru $\vec{X_i}$ je zjištěno z y_i. V tom případě, znaménko funkce p_i vrací klasifikaci.



Obrázek 2.9: Perceptron – matematický model biologického neuronu, převzato z [10]

Podle zdroje [16] používají výzkumníci pro textovou klasifikaci jednovrstvý perceptron, kvůli jednoduchosti implementace. Případně vícevrstvý perceptron, který je sofistikovanější a taky se hojně používá pro klasifikační úlohy.

Aktivační funkce

Funkce, značená $\varphi(v)$, definuje výstup neuronu z hlediska indukovaného lokálního pole v. Existují dva základní typy aktivačních funkcí:

1. Prahová funkce – tento typ funkce můžeme definovat následovně:

$$\varphi(v) = \begin{cases} 1 & \text{pokud} \quad v \geq 0 \\ 0 & \text{pokud} \quad v < 0 \end{cases} \tag{2.23}$$

   V literatuře tato forma prahové funkce je označována jako Heavisideova funkce. Po označení výstupu neuronu k tato funkce bývá vyjádřena trochu odlišně:

$$y_k = \begin{cases} 1 & \text{pokud} \quad v_k \geq 0 \\ 0 & \text{pokud} \quad v_k < 0 \end{cases} \tag{2.24}$$

   v_k je zde lokální indukované pole:

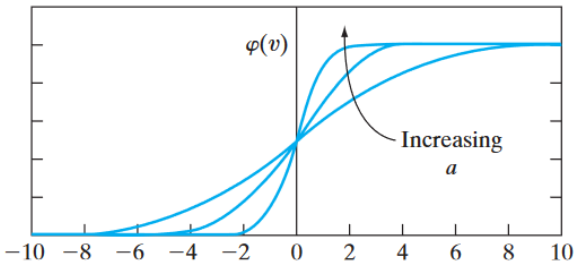
$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{2.25}$$

Výstup neuronu nabývá hodnoty 1 ,pokud není indukované lokální pole tohoto neuronu negativní a naopak 0 pokud je.

2. Sigmoidální funkce – graf této funkce je tvarován do písmene „S“ a je to nejčastěji užívaná forma aktivační funkce, která se používá k vytvoření neuronových sítí. Je definována podle [10] jako silně rostoucí funkce, která vykazuje rovnováhu mezi lineárním a nelineárním chováním. Příkladem „sigmoidální“ funkce je logistická funkce, která je definována jako:

$$\varphi(v) = \frac{1}{1 + \exp(-av)} \tag{2.26}$$

kde a je parametrem sklonu funkce. Změnou parametru a dosahujeme toho, aby měla „sigmoidální“ funkce různé sklony, což je možno vidět na obrázku 2.10. V limitě, jak se parametr sklonu přibližuje nekonečnu, se ze sigmoidální funkce stává prahová funkce, která nabývá hodnot 0 a 1. Na rozdíl od prahové funkce, sigmoidální funkce nabývá spojitého rozsahu od 0 po 1. Navíc je možné si všimnout, že sigmoidální funkce je diferencovatelná, zatímco prahová funkce není.



Obrázek 2.10: Sigmoidální funkce, převzato z [10]

Obě funkce výše byly definovány na rozsahu hodnot od 0 po +1. Někdy, ale potřebujeme, abychom měli aktivační funkci v rozsahu −1 po +1, tím pádem aktivační funkce je lichá funkce indukovaného lokálního pole. Konkrétně prahová funkce je potom definována jako:

$$\varphi(v) = \begin{cases} 1 & \text{pokud} \quad v > 0 \\ 0 & \text{pokud} \quad v = 0 \\ -1 & \text{pokud} \quad v < 0 \end{cases} \tag{2.27}$$

Běžně je tato funkce označovaná jako funkce signum. Pro odpovídající formu sigmoidální funkce můžeme využít hyperbolický tangens definovaný jako:

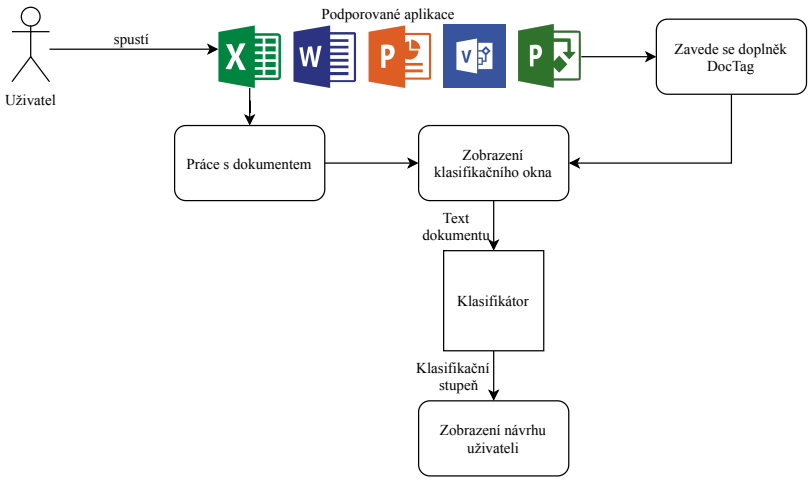
$$\varphi(v) = tanh(v) \tag{2.28}$$

Upravením aktivační funkce typu sigmoid, aby nabývala i záporných hodnot (vyjádřeno rovnicí 2.28), může dojít k získání praktických výhod na rozdíl od logistické funkce v rovnici 2.26.

Kapitola 3

Klasifikační nástroj využívající strojového učení

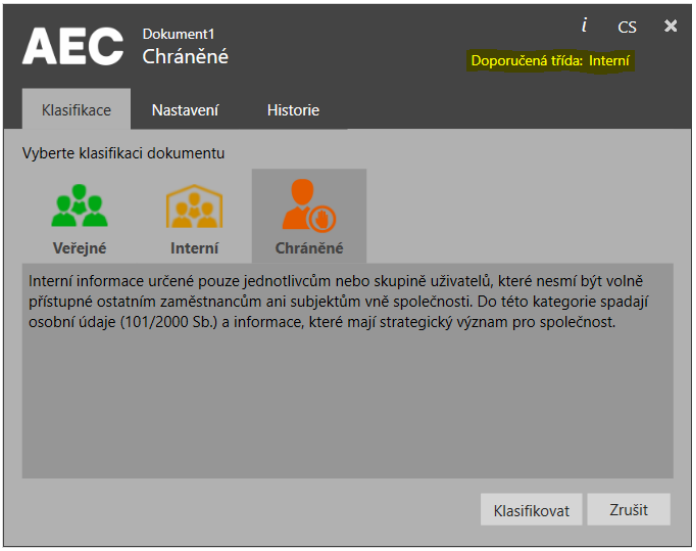
Cílem práce bylo doplnit do klasifikační aplikace DocTag společnosti AEC modul, který bude zodpovědný za návrh klasifikace pro uživatele. Navrhování stupňů má být podporováno ve všech aplikacích Office, které se ve firmách běžně používají. Jde tedy o Word, Excel, PowerPoint, Project a Visio. Dalším požadavkem bylo, aby klasifikátor doporučil třídu za krátký čas, jinak by modul nebyl použitelný. Po konzultaci s kolegy jsem usoudil, že by to mělo být maximálně do 3 vteřin. Dále bylo nutné se zamyslet, kde bude celý proces probíhat, jestli v klientském počítači nebo na serveru a když na serveru tak, jak bude výsledek distribuován, ale k tomu se vrátím později, protože je nutno zvážit více faktorů. Pak bylo potřeba rozhodnout jakou formou se bude uživateli doporučení stupně zobrazovat. Možnosti byly následující: „pop-up“ okno (vyskakovací okno), vyjíždění lišty shora, zobrazení bočního panelu s návrhem, přidání ikony do „ribbonu“ a zobrazení návrhu přímo v klasifikačním okně. Pro lepší porozumění je k dispozici diagram 3.1 celé „workflow“. Taktéž se počítá s tím, že pokud bude řešení úspěšné, nasadí se do většího počtu firem. Mohou to být firmy menší (o 40 zaměstnancích), anebo taky větší třeba v řádech tisíců zaměstnanců. Z toho vyplývá, že na správném návrhu opravdu záleží, jinak bychom následně mohli strávit spoustu času nad opravováním chyb.



Obrázek 3.1: Postup práce k navržení klasifikačního stupně

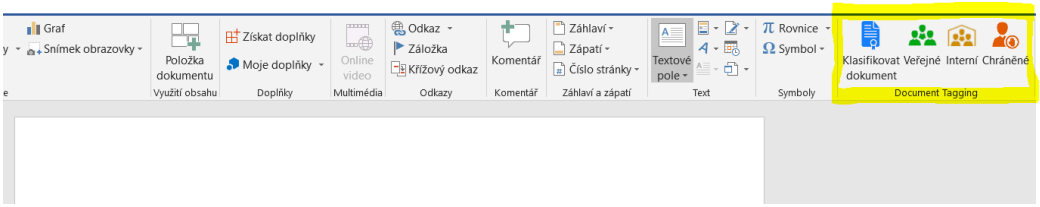
3.1 Návrh

Běžný uživatel, který má doplněk nainstalovaný bude klasicky pracovat s nějakou již zmíněnou aplikací (např. Word) a ve chvíli, kdy si zobrazí klasifikační okno (obrázek 3.2), tak se na vstup klasifikátoru odešle text dokumentu a vrátí se výsledný klasifikační stupeň. Uživatel se poté může rozhodnout, zda souhlasí s návrhem a vybere uvedený stupeň, anebo zvolí jiný stupeň. V případě, že by zvolil odlišný stupeň od navrhovaného, tak je nutno poslat text do učícího algoritmu, aby se klasifikátor mohl zdokonalit.



Obrázek 3.2: Klasifikační okno aplikace DocTag s navrženým stupněm od klasifikátoru

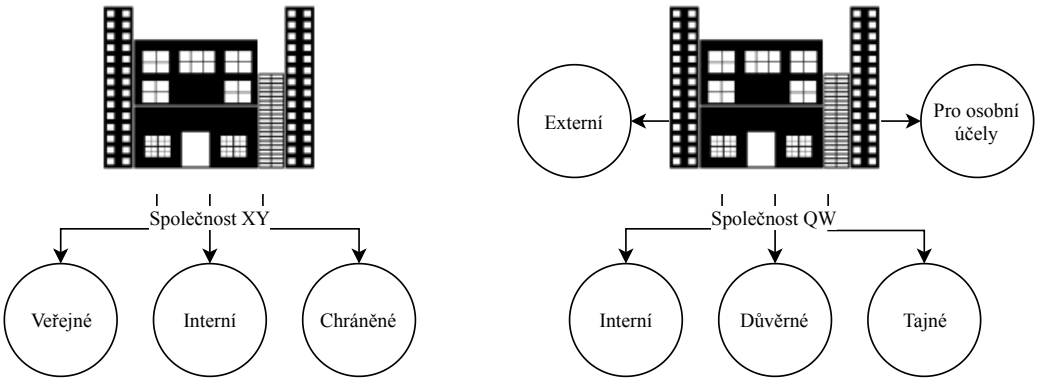
Nicméně je nutno zmínit, že aplikace DocTag podporuje i rychlou klasifikaci, při které ani nezobrazuje klasifikační okno. Klasifikace v takovém případě probíhá tím, že uživatel pouze klikne v „ribbonu“ (obrázek 3.3) na tlačítko a dokument se klasifikuje. Právě proto, bylo potřeba se zamyslet nad dalším způsobem, jak uživateli neinvazivně (nechceme uživatele otravovat) zobrazit navržený stupeň, čímž se opět vracíme k možnostem zobrazení. Z rozhovoru s kolegy jsem došel k závěru, že nejlepším řešením bude nechat vyjíždět lištu shora. To ovšem není všechno, je důležité vyřešit, kdy se má tato lišta objevit. Ovšem existuje pro to jednoduché řešení – v okamžiku, kdy uživatel klikne na nějaké tlačítko rychlé klasifikace, tak se spustí proces stejný jako v obrázku 3.1. Tudíž, pošle se aktuální text dokumentu a klasifikátor vyhodnotí stupeň. Každopádně je třeba zde udělat jednu změnu, abychom opět uživatele tolik nerušili, lišta se zjeví jen v případě, že se vyhodnocený stupeň liší od toho, co vybral uživatel. Tím, že se vyhodnocovací proces spustí až po kliknutí na některé z tlačítek „ribbonu“ nemusíme vůbec řešit nějaké změny v dokumentu (aplikace často přepočítávají výsledky ihned po změně v textu – např. počet znaků v dokumentu). Ovšem, kdyby byl takový požadavek, bylo by nutno vymyslet nějaké úsporné řešení (např. časovač, který by po 5 minutách spouštěl daný proces).



Obrázek 3.3: DocTag ribbon v aplikaci Word

Na obrázku 3.3 je možno vidět „ribbon“ se 4 tlačítky. První z nich s názvem „Klasifikovat dokument“ vyvolává klasifikační okno (obrázek 3.2) a další tři za ním patří do tzv. rychlé klasifikace, která byla zmíněná v odstavci výše.

Nyní je na čase vyřešit tu nejtěžší otázku a to, jaký klasifikátor je nejvhodnější. V podsekci 2.2.6 jsem uvedl různé metody umělé inteligence. Nejprve jsem problém chtěl řešit pomocí neuronové sítě, ale musel bych čelit mnoha rozhodnutím, se kterými nemám moc zkušeností. Prvně bych musel vybrat vhodný typ neuronové sítě, u některých sítí je třeba promyslet i jaké vrstvy implementovat. Následně by bylo potřeba vybrat správnou aktivační funkci v daných vrstvách a na závěr rozhodnout kolik bude mít každá vrstva neuronů. Další překážkou by bylo, že abych dokázal neuronovou síť korektně naučit, musel bych sestavit velký trénovací set (korpus), což by mohl být problém z toho důvodu, že není mnoho důvěrných či interních dokumentů volně k dispozici. Dalším úskalím je to, že každá společnost, která doplněk využívá, může mít menší nebo větší počet klasifikačních stupňů a v takovém případě by nebylo možné předávat naučenou neuronovou síť různým firmám (obrázek 3.4). Kvůli všem zmíněným potížím jsem se nakonec rozhodl, že nejlepším řešením bude využít „naivního“ Bayese. Implementace je poměrně jednoduchá, vyhodnocení probíhá rychle a funguje i s menším vzorkem trénovacích dat. Tím pádem by mělo být možné ve spolupráci se zákaznickou firmou poměrně rychle naučit klasifikátor na jiném počtu klasifikačních stupňů než má právě AEC. V podstatě je to ideální řešení, pokud chceme rychle zjistit, zda dané řešení je přívětivé či nikoliv.

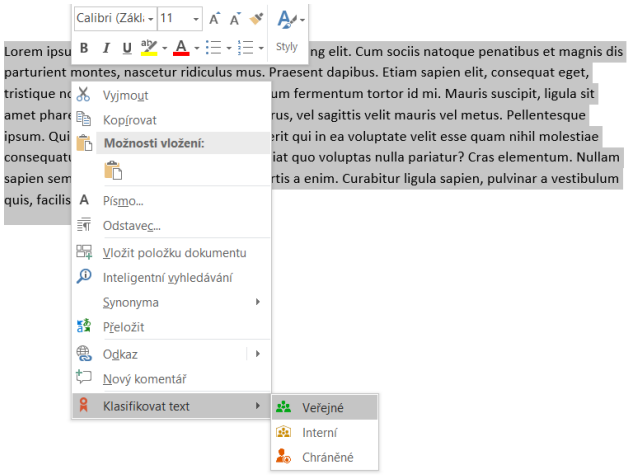


Obrázek 3.4: Rozdílný počet tříd ve dvou společnostech

Když je rozhodnuto o druhu klasifikátoru, zbývá nyní určit, kde se bude text zpracovávat nebo jestli se případně zpracování rozdělí mezi klienta a server. Mohlo by to fungovat tak, že by se text předzpracoval na klientské straně a vytvořené tokeny by se posílaly

na server. Nicméně já osobně zastávám názor, co může bez větších problémů běžet na serveru, nechť tam běží. Aplikace DocTag již teď bere nějaký ten výkon. Když k tomu přičteme možnost, že uživatel má více doplňků a při práci používá různý počet aplikací, ještě v kombinaci se starším „železem“, zaděláváme si na problém. Podle mě by klientská aplikace měla být, co nejméně náročná, jak to jen jde. Navíc je daleko jednodušší vylepšovat řešení na serveru než na všech nainstalovaných klientech. Takže jsem se rozhodl, že doplněk DocTag bude získávat text z dokumentu a tento text pouze odešle na server, kde proběhne jeho celkové zpracování a výsledek se pošle zpět onu klientovi. Dalším poznatkem je fakt, že „add-in“ už nyní komunikuje se serverem za účelem stahování publikované konfigurace (nastavení), takže přidání další režie na komunikaci s klasifikátorem by nemělo způsobovat žádný problém. Ovšem následně jsem si uvědomil, že stávající komunikace probíhá přes WCF (Windows Communication Foundation). Tento fakt je poměrně omezující, pokud budeme myslet dopředu. Když se zamyslíme nad samotným klasifikováním, možná bychom chtěli někdy v budoucnu klasifikovat dokumenty i bez samotné aplikace DocTag. Dovedu si představit poměrně reálný scénář, kdy budu na jiné počítačové stanici, kde není doplněk, ale budu chtít vytvořit dokument a klasifikovat jej. V takovém případě by bylo ideálním řešením REST nebo SOAP api na nějakých firemních stránkách. Koneckonců konfigurace „DocTagu“ se spravuje na webových stránkách, a tak by přidání API mělo být poměrně jednoduché. Tato volba se zdá být nejvhodnější, a protože chceme mít službu, co možná nejjednodušší (není potřeba žádné extra zabezpečení), bylo vybráno REST api.

Předposlední potíží bylo vymyslet jak daný klasifikátor efektivně naučit. Určitě je dobré mít trénovací množinu dat, ze kterého bude klasifikátor vycházet, ale když připravíme uživatelům rozhraní přes které mohou modul zdokonalit, může to být ještě lepší. Po konzultaci s vedoucím týmu aplikace DocTag jsem obdržel návrh na přidání položky do kontextové nabídky, která by měla za cíl odesílat označený text na server. Označený text by uživatel mohl klasifikovat jedním ze stupňů a klasifikátor by se díky tomu mohl učit. Pro lepší představu je zde obrázek 3.5 již implementovaného rozšíření kontextové nabídky.



Obrázek 3.5: Rozšířená kontextová nabídka o klasifikování textu v aplikaci Word

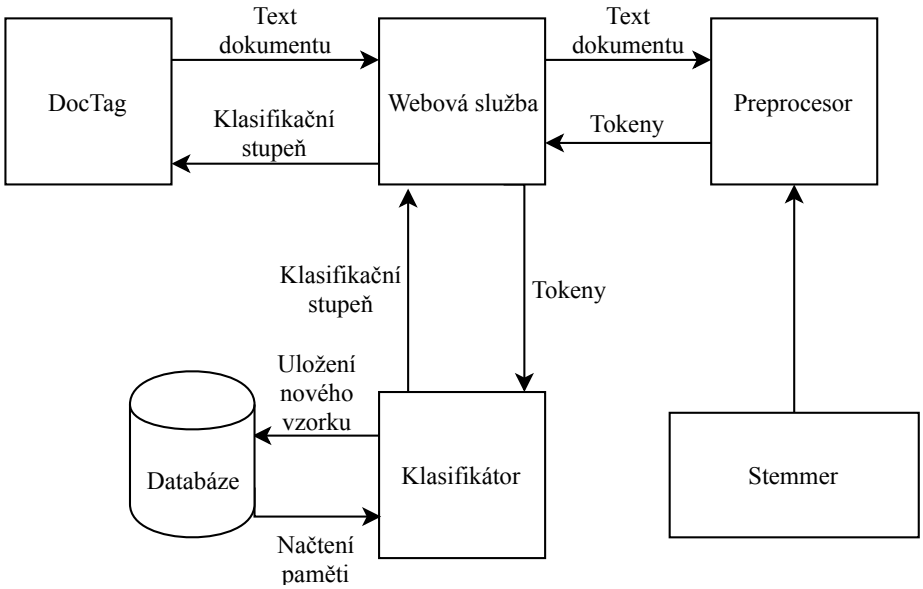
Posledním rozhodnutím, které bylo nutné učinit před samotnou implementací, bylo vybrat vhodný lingvistický jazyk. Zvolil jsem právě angličtinu, protože je dostupných daleko více zdrojů v tomto jazyce než v češtině.

3.2 Implementace

V souladu se zadáním je tato praktická část vytvořena v prostředí .NET (konkrétně starší verze 4 – kvůli možné nekompatibilitě s novými verzemi v jiných firmách) za použití programovacího jazyka C#. Postupoval jsem prototypově, protože se tak dá velmi rychle zjistit, zda řešení funguje a není nutná žádná složitá implementace. Navíc díky tomuto přístupu je možné řešení rychle přenášet na jiná prostředí, anebo prototyp kdykoliv „zahodit“ pokud nesplňuje požadavky. Tím, že se jedná o co možná nejjednodušší výtvory, smazáním daného prototypu neztrácíme mnoho času a můžeme ihned začít od začátku.

Během práce jsem využíval rozhraní („interfaces“) pro předpis některých tříd. Díky nim je pak možné jednoduše nahradit nějakou konkrétní implementaci komponenty za jinou (např. implementaci „stemmeru“ Porterovým algoritmem nahradit za Lovinsovo [22] řešení). Dále je pak celý modul, zodpovědný za vyhodnocování klasifikace, vytvořen nezávisle na doplňku DocTag. Navzdory tomu, že řešení bylo děláno kvůli rozšíření „DocTagu“, bylo implementováno odděleně. Díky oddělení jednotlivých komponent je pak možno lépe spravovat jejich kód a taky je dosaženo znovupoužitelnosti pro další budoucí aplikace.

Postup praktické části je popsán po částech níže v chronologickém pořadí, jak byly implementovány. Celou aplikaci je možno rozdělit na komponenty: DocTag, preprocesor, stemmer, klasifikátor, databázi, webová služba (vizte obrázek 3.6).



Obrázek 3.6: Implementované řešení

3.2.1 DocTag

Je doplněk MS Office firmy AEC, který je zaváděn ve chvíli, kdy se spouští jedna z aplikací Office. Pro vytvoření „ribbonu“ (tlačítek na kartě např. Vložení) je nutno definovat celkovou strukturu v xml souboru. To znamená: popisy, ikony, tlačítka, chování jednotlivých tlačítek (při kliknutí, při najetí myši), velikost, dále je možno v tomto souboru nastavit i jiné ovládací prvky než jen „ribbon“. Ovšem je nutné si dát pozor, protože Microsoft rozlišuje dvě verze syntaxe tohoto xml souboru a liší se datem. Tedy pokud chceme využívat novější prvky, tak musíme v hlavičce uvést novější verzi. Problém může nastat pokud „upgradujeme“ na

novější verzi, ale některé prvky jsou definovány starou verzí. Pak nezbývá nic jiného než tuto syntaxi převést do nové verze, jinak tyto prvky nebudou vůbec zobrazeny.

Konkrétně jsem do xml souboru přidal rozšíření kontextové nabídky o klasifikaci označeného textu. V souboru jsem definoval, že zde má být rozbalovací položka „Klasifikovat text“. Dále jsem určil, že po najetí na tuto položku se zde objeví další možnosti a to už konkrétní třídy. Po kliknutí na jednu ze zobrazených tříd se odešle označený text do webové služby. Dále se dokument klasifikuje na zvolený stupeň a to za předpokladu, že dokument nebyl již klasifikován, anebo byl klasifikován nižším stupněm. Tímto řešením je tedy možno klasifikátor učit cíleně na konkrétní text a uživatel může v tomto dokumentu označit dokonce více textů.

Nicméně, hlavní stěžejní části „DocTagu“, kterou bylo třeba přidat je získávání textu z dokumentu, která tam dosud nebyla. Obecně by se nemělo jednat o těžký úkol, ale bohužel každá aplikace Office byla nejspíš programována jiným týmem. Tudíž api, (aplikační programové rozhraní – slouží k využití funkcionality jednoho programu jiným programem) každé aplikace zvlášť, bylo potřeba nejprve nastudovat a vyzkoušet. Bez „rozběhnutí“ této části by programování dalších částí nemělo vůbec smysl. To se nakonec povedlo téměř ve všech aplikacích až na Project, kde nastal ještě navíc problém s editováním kontextové nabídky. Takže u aplikace Project mi nezbylo nic jiného než ji vynechat. Naštěstí se nejedná o základní uživatelskou aplikaci balíku Office a tolik to tedy nevadí.

3.2.2 Preprocesor

Tato komponenta je zodpovědná za předzpracování textu dokumentu. Jedná se tedy konkrétně o odstranění „stop“ slov, poté „tokenizaci“ textu a nakonec „stemování“ slov. Slovník „stop“ slov, který preprocesor využívá, byl získán díky zdroji [4]. Nemusel jsem si tedy vytvářet svůj vlastní, což by mi zabralo docela dost času. Ostatně mnoho stránek na internetu poskytuje tyto slovníky, protože strojové učení je velice oblíbené téma.

Pro rozdělení slov na „tokeny“ jsem využil již existující funkcionality v .NET a to v knihovně Accord [30] pod licencí LGPL. „Tokenizer“ z tohoto zdroje převádí v textu všechna velká písmena na malá, odstraňuje interpunkci a dělí text na jednotlivá slova. Poté, abych docílil toho, aby různé tvary stejného slova byly stále převáděny na stejné slovo jsem využil „stemmer“, který je popsán níže v další podsekci.

3.2.3 Stemmer

Význam „stemmeru“ už byl představen v podsekci výše. Spíše je důležité jak byl implementován v kódu. Na světě existuje mnoho těchto algoritmů, samozřejmě nejpopulárnějším je Porterův[31] algoritmus. Díky této popularitě je k dispozici i zdroj [21], který ho poskytuje a dokonce i v C# pod licencí MIT. Kód nebylo nijak třeba upravovat, protože byl perfektně funkční. Samozřejmě při přidávání do mého programu jsem počítal i s variantou dalšího „stemmeru“, takže byla logika navržena tak, aby se dal jednoduše nahradit.

3.2.4 Klasifikátor

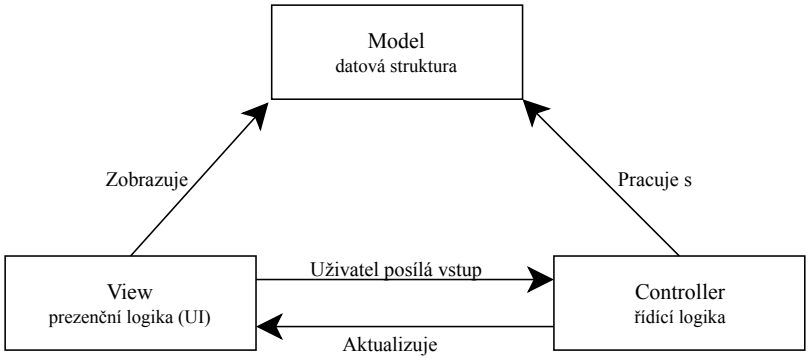
Než jsem se pustil do samotného programování klasifikátoru, podíval jsem se na stávající řešení, která mi poskytla určitý rozhled a pak jsem věděl, kterým směrem se vydat. Při vytváření prototypu jsem postupoval tak, že jsem si nejprve vytvořil konzolovou aplikaci. Pak jsem vytvořil příslušné třídy a na minimálním vzorku dat testoval funkcionalitu. Nejprve to bylo formou binárního klasifikátoru. Zjišťoval jsem, jestli je daná zpráva spam nebo

není. Vytvořil jsem si dvě třídy a těm přiřadil několik vět, které danou třídu reprezentovaly. Ve chvíli, kdy byl prototyp funkční jsem se posunul blíže k problematice. Vytvořil jsem si tři klasifikační třídy (pro veřejný, interní a chráněný stupeň) a opět je naplnil testovacími daty. Po pár testech jsem byl překvapen, že i s tak málo daty, algoritmus funguje poměrně dobře. Nyní byl klasifikátor připraven na implementaci do „DocTagu“.

Podle návrhu bylo třeba zapracovat celý modul klasifikace na server, aby se umělá inteligence vyvíjela na jednom místě a z ní mohli čerpat všichni klienti. Ještě než jsem se přesunul k vytváření webové služby, musel jsem přidat metody, které budou volány pro doporučení klasifikačního stupně. S tím souviselo i zapracování textového pole na „front-endu“, které mělo signalizovat doporučený stupeň od klasifikátoru. Taktéž bylo třeba ještě doplnit metodu pro klasifikaci textu přes kontextovou nabídku (vyvolanou označením textu a stisknutím pravého tlačítka). Následně už jen zbývalo vytvořit webovou službu, přes kterou budou klienti se serverem komunikovat.

3.2.5 Webová služba

Díky návrhu jsem už věděl, že budu implementovat službu pomocí rozhraní „REST“ api. Toto rozhraní je použitelné pro jednotný a snadný přístup ke zdrojům. Všechny zdroje mají identifikátor URI (jednotný identifikátor zdroje) a REST určuje, jak se k datům přistupuje. Pro přístup ke zdrojům se využívají 4 metody a to: vytvoření dat, získání požadovaných dat, změnu a smazání. Mě ovšem konkrétně zajímalo pouze vytvoření dat za použití metody POST, protože text dokumentu je nutný poslat do služby, která se o klasifikaci postará a poté si naučená data v databázi ponechá.



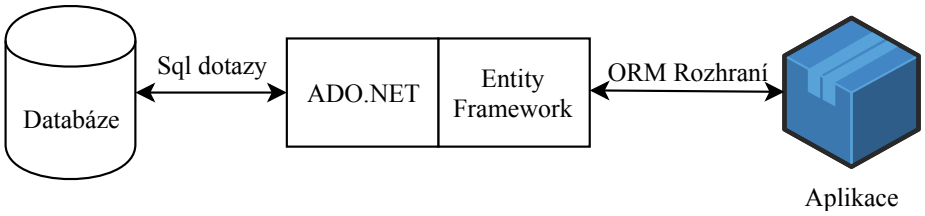
Obrázek 3.7: Princip fungování MVC architektury

Opět jsem přistupoval k této komponentě podobně. Prvně jsem si vytvořil jednoduchý projekt, který spustil webové stránky za použití MVC (model-view-controller) architektury (obrázek 3.7). Vytvořil jsem kontrolér, který bude zodpovědný za vyřizování požadavků a testoval jsem, jestli při přístupu na danou stránku dostanu korektní odpověď. Po mnoha snažení jsem dospěl k tomu, že mi daná stránka po zadání správného URI vrátila očekávaný výsledek. Pak jsem přešel k implementaci klasifikátoru, který již byl hotov, přímo do tohoto api. Netrvalo to dlouho a řešení bylo doplněno. Pro ověření správné funkcionality jsem si vytvořil další konzolovou aplikaci, která měla za cíl simulovat klienta aplikace DocTag (opět kvůli jednoduchosti). Jednalo se pouze o vytvoření spojení, odeslání požadavku a očekávání správné odpovědi. Ovšem objevily se nějaké problémy. Výchozím chováním kontroléru pro metodu POST bylo očekávat parametry dotazu přímo v adrese (URL) stránky. To ovšem

nebylo elegantní řešení, vzhledem k tomu, že text dokumentu bývá rozsáhlý a určitě nechceme přenášená data zobrazovat jako součást URL. Právě proto jsem našel způsob jak definovat, že chci parametry dotazu posílat přímo v těle zprávy HTTP požadavku. Potom už komunikace mezi konzolovou aplikací a webovou službou probíhalo správně. Tudíž jsem vzal odesílání požadavků z konzolové aplikace a vložil ho přímo do doplňku DocTag.

3.2.6 Databáze

Posledním krokem této pomyslné skládačky bylo ukládat zpracované výsledky, aby se mohla umělá inteligence vyvíjet. Využil jsem Entity „Frameworku“, což je ORM (mapovač objektů na relační data) a postupoval podle pracovního postupu zvaného Code First (postup, kdy nejprve naprogramujeme třídy programu, ze kterých se následně vytváří potřebná struktura databáze). Poté se po spuštění webové aplikace sám Entity Framework (obrázek 3.8) postaral o vytvoření tabulek v databázi. Nezbývalo už nic jiného než přidat do webové služby samotnou práci s databází. Tedy načítání dat z databáze v případě nového požadavku na doporučení klasifikace a uložení dat do databáze po klasifikování dokumentu uživatelem.



Obrázek 3.8: Komunikace mezi aplikací a databází přes Entity Framework

Kapitola 4

Testování

Tato kapitola se zabývá testováním samotného klasifikačního nástroje. Nachází se zde popis testovaných dokumentů, na jak velkém množství testování probíhalo a co konkrétně bylo předmětem testování. Vyhodnocení samotných testů je diskutováno v sekci 4.2. Dále je v závěru kapitoly zmíněn další rozvoj aplikace 4.3, který popisuje, jaké jsou další možnosti rozšíření stávajícího řešení.

4.1 Metodika testování

Vzhledem k tomu, že klasifikátor byl učen pouze na větách bylo třeba zjistit, jak si povede s celými dokumenty, protože to je hlavní téma práce. Nejprve bylo nutné si obstarat testovací vzorky dat, na kterých se aplikace bude testovat. Tyto vzorky jsem si sám vytvářel, protože dokumenty nejsou obyčejně k dostání. Tedy kromě veřejných dokumentů, ty bylo možné stáhnout z internetu v podstatě odkudkoliv. Co se týká interních a chráněných souborů, buď jsem si je kompletně vytvořil celé sám, anebo jsem se inspiroval daty ve firmě AEC, které byly k dispozici. Pochopitelně jsem citlivé údaje nahradil smyšlenými. Dokumenty jsem vytvářel tak, aby splňovaly informační hodnotu definovanou jednotlivými stupni (vizte 2.1.1). Konkrétně se jednalo o 32 dokumentů, které byly v zastoupení: 11 veřejných, 11 interních a 10 chráněných. Délka jednotlivých dokumentů se pohybovala okolo 673 znaků, resp. bajtů. Když už byla testovací sada připravena, následovalo samotné testování. Zajímal mě hlavně poměr pravdivě pozitivních klasifikací a falešně pozitivních klasifikací. Pravdivě pozitivní jsou ty, u kterých je jisté, že spadají do dané klasifikační třídy a po otestování klasifikátorem jsou touto třídou opravdu označeny. Opakem je tomu u falešně pozitivních – dokumenty jsou klasifikovány stupněm jiným než očekáváme. Dalším zkoumáním bylo výkonnostní testování, tedy jak rychlé je vyhodnocení s rostoucí velikostí zkoumaného dokumentu.

4.2 Výstupy testování

Poměr falešně pozitivních a úspěšně pozitivních klasifikací

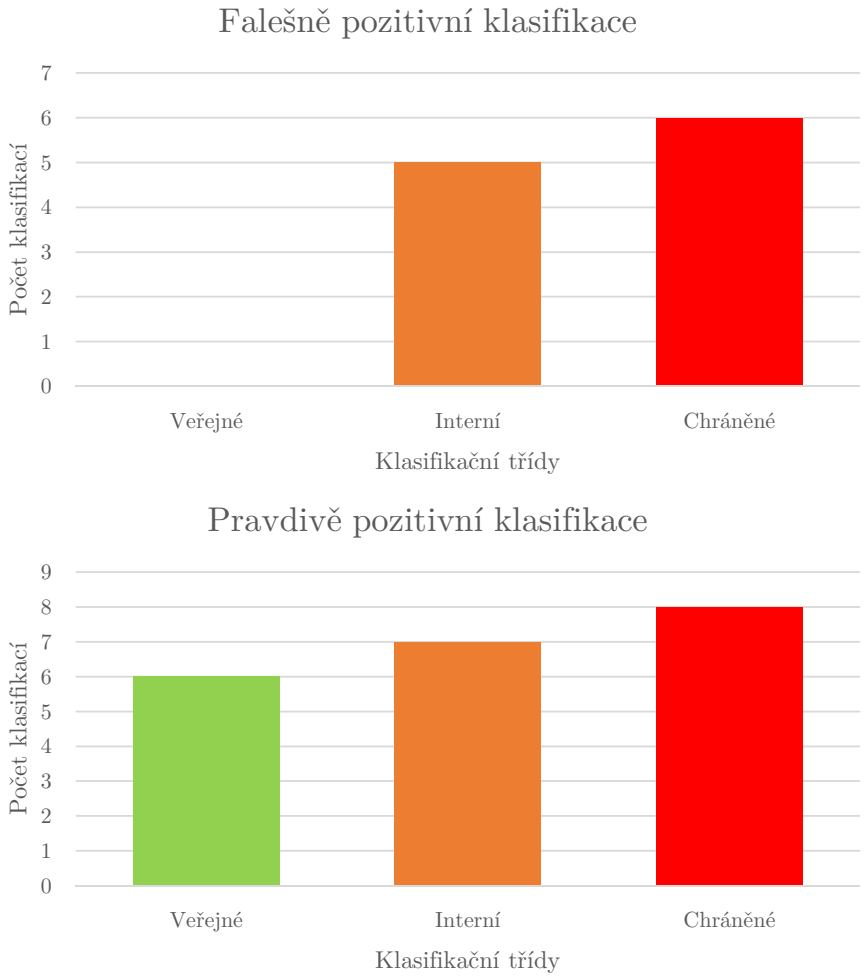
Při spouštění tohoto testu jsem nejprve vytvořil dvakrát delší dokumenty než jsem uvedl v sekci 4.1. Následně jsem došel k zjištění, že klasifikátor naučený na pouhých větách nestačí. Bylo potřeba rozšířit trénovací korpus, a tak jsem posílil každou třídu o pět dokumentů. Bohužel ani to nedosahovalo dobrých výsledků. Zmenšil jsem tedy dokumenty zpět na původní velikost a už se dalo mluvit o úspěchu. Pochopitelně nejlepším zjištěním by bylo, že všechny dokumenty jsou klasifikovány správně, ale neměl jsem vytvořenou dostatečně velkou tréno-

Třída	Počet korektních klasifikací
Veřejné	6 z 11
Interní	7 z 11
Chráněné	8 z 10

Tabulka 4.1: Přehled korektních vyhodnocení klasifikátoru

vací sadu. Navíc jsem v testovacích dokumentech připravil i pár specialit. Příkladem může být třeba soubor s hesly (interní), který zmiňoval, že se máme vyvarovat triviálním heslům, což bylo vyhodnoceno jako chráněný dokument (kvůli zmínkám o heslech). Výsledky testování dokumentů je možno vidět v tabulce 4.1.

Počet falešně pozitivních a pravdivě pozitivních klasifikací pro každou třídu je pak možno vidět na obrázku 4.1. Jak je jistě viditelné z prvního grafu – žádná klasifikace třídy interní nebo chráněné není špatně mapována na třídu veřejné. Těžko by se dalo mluvit o příčině, vzhledem k slabě naučenému klasifikátoru.



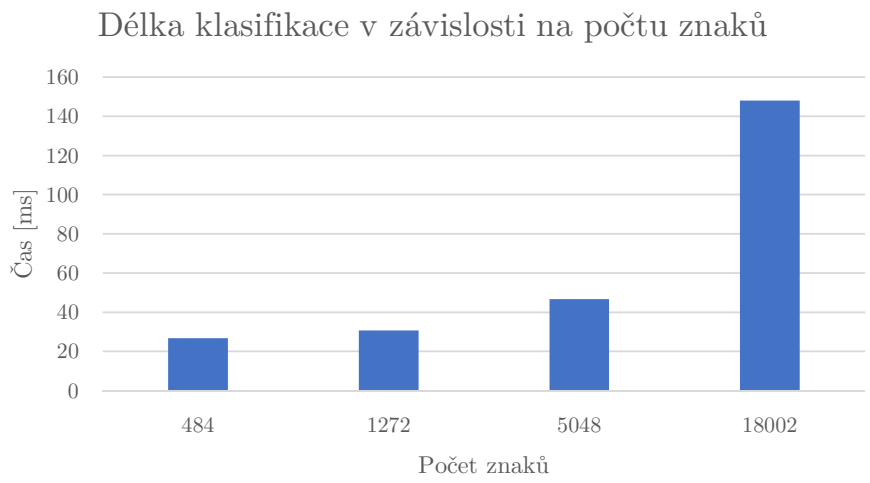
Obrázek 4.1: Porovnání falešně pozitivních a pravdivě pozitivních klasifikací

Závěrem tohoto testování je ponaučení, že je příště potřeba nasadit řešení přímo do firmy, kde se bude klasifikátor od uživatelů učit. Učení klasifikátoru za pomocí pouze jeho autora má totiž dvě velké nevýhody:

- učení zabírá dlouhou dobu,
- nebude kvalitní, pokud nejsou poskytnuta reálná data (obzvlášť ty chráněná se těžko shání).

Test rychlosti vyhodnocení klasifikace

Cílem bylo zkoumat dobu rozhodovacího procesu klasifikace v závislosti na velikosti dokumentu. Jak by možná mohl někdo předpokládat, čím je větší dokument, tím bude delší analýza a zpracování klasifikátoru. To je možná zřejmé, ale taky bylo potřeba zjistit zda aplikace splňuje požadavek v návrhu – tedy jestli je zobrazení doporučeného stupně dostatečně rychlé (pod tři vteřiny podle kapitoly 3). Splněný požadavek dokazuje obrázek 4.2 s přehledem a zároveň popisuje lineární chování.



Obrázek 4.2: Graf rychlosti klasifikace

Tento test byl vyhodnocen tím způsobem, že se klasifikovali různě velké soubory bez ohledu na výsledek klasifikace. Vyhodnocení klasifikace sestává z předzpracování textu, výpočtu třech pravděpodobností (pro každou třídu) a výběru té nejpravděpodobnější.

4.3 Další rozvoj aplikace

Možnosti rozvoje modulu klasifikátoru jsou obrovské, protože je textová klasifikace resp. zpracování přirozeného jazyka velmi populární téma. Jedna z příležitostí je využít jinou technologii např. již zmíněnou neuronovou síť. Pokud bychom chtěli vyměnit Bayesův klasifikátor za neuronovou síť, museli bychom přijít se speciálním typem, který by podporoval proměnný počet klasifikačních stupňů, aby bylo možné jedno naše řešení implementovat ve

firmách s odlišným počtem klasifikačních tříd. Druhou možností by bylo poskytovat nenaučený modul s tím, že bude aktivní až po nějaké době než se naučí. To znamená, že bychom klasifikátor v zákaznické společnosti nainstalovali a určitý čas by lidé klasifikovali pouze manuálně s tím, že se data odesílají neuronové síti. Doba, kdy by se mělo řešení aktivovat, by závisela na počtu učiněných klasifikací. Logicky by firma s větším počtem zaměstnanců, kteří klasifikují měla naučit „neuronovku“ dříve než firma s menším počtem. Ovšem jaký počet klasifikací by byl dostatečný pro korektní použití si netroufám ani odhadovat a to hlavně kvůli tomu, že uživatelé nemusí vždy vybírat stupně správně.

Dalšími rozšířeními by mohlo být:

- nahrazení „tokenů“, které jednoznačně udávají míru důvěrnosti za klíčové slovo (např. místo data narození bude token $date),
- podpora více přirozených jazyků (ruština, španělština, atd.),
- nahrazení „stemmeru“ za lemmatizátor,
- využiti „n-gramů“ – namísto počítání jednotlivých slov, bychom mohli počítat sekvence (např. dvojice slov),
- využití TF-IDF – metody, která udává jak moc je dané slovo relevantní v dokumentu v souvislosti s celým korpusem,
- podpora změny nějaké komponenty klasifikátoru (administrátor na serveru by mohl třeba použít jiný slovník nebo stemmer).

Kapitola 5

Závěr

Tato bakalářská práce měla za cíl prozkoumat možnosti využití umělé inteligence pro klasifikaci dokumentů a nalézt vhodná řešení pro implementaci automatické klasifikace na základě uživatelských dat. V této oblasti bylo popsáno několik metod, kterými se dá úkol splnit. Dále byla jedna metoda představena detailněji právě proto, že byla i využita pro implementaci. Celkově hodnotím celou tuto problematiku jako velmi bohatou na možnosti, kterými je možné cíle dosáhnout.

Implementaci jsem nejprve chtěl řešit formou neuronových sítí. Ovšem ty vyžadují velký počet dat, aby fungovaly správně. Při studování problematiky jsem narazil na dost problémů a právě proto jsem se nakonec uchýlil k jednoduššímu řešení – naivního Bayese. Když jsem měl funkční prototyp, který byl schopen poznat klasifikaci z krátkých vět, pustil jsem se do vytváření a editace dalších komponent. Celkově jsem rozšířil stávající systém aplikace DocTag ve vrstvách: prezentační (zobrazování doporučení stupně, rozšíření kontextové nabídky), aplikační (zpracování textu, komunikace jednotlivých komponent) a datové (ukládání výsledků).

Pro testování aplikace jsem používal dva scénáře. Prvním bylo sledování poměru *false positive* a *true positive*. Jinými slovy úspěšnost daného řešení. Druhým pak bylo měření doby zpracování v závislosti na rostoucí velikosti testovacího dokumentu. Z výsledků vyplynulo, že algoritmus funguje velmi rychle, možná i nad očekávání. Bohužel na druhou stranu klasifikátor není moc spolehlivý, co se týče určování klasifikační třídy, ve chvíli, kdy je mu poskytnut nějaký větší dokument. Může za to hlavně fakt, že byl primárně učen na věty. Nicméně řešení považuji za využitelné s tím, že pro maximalizaci úspěšnosti se musí nasadit přímo v dané společnosti, kde klasifikátor bude od uživatelů sbírat data po nějakou vhodnou dobu. Taktéž je možno za tímto účelem rozšířit aplikaci jednou z metod, které byly navrženy v sekci 4.3.

Literatura

[1] Aggarwal, C. C.; Zhai, C.: *Mining text data*. Springer Science & Business Media, 2012.

[2] Bullinaria, J. A.: *IAI : The Roots, Goals and Sub-fields of AI*. [Online; navštíveno 19.01.2019].
URL https://www.cs.bham.ac.uk/~jxb/IAI/w2.pdf

[3] Cambria, E.; White, B.: Jumping NLP curves: A review of natural language processing research. *IEEE Computational intelligence magazine*, ročník 9, č. 2, 2014: s. 48–57.

[4] Doyle, D.: *English Stopwords*. [Online; navštíveno 19.01.2019].
URL https://www.ranks.nl/stopwords

[5] Eisele, R.: The log-sum-exp trick in machine learning. *Systems Architect and DBA*, 2016.

[6] Fox, J.; Weisberg, S.: Bootstrapping regression models. *An R and S-PLUS Companion to Applied Regression: A Web Appendix to the Book. Sage, Thousand Oaks, CA. URL http://cran. r-project. org/doc/contrib/Fox-Companion/appendix-bootstrapping. pdf*, 2002.

[7] Freund, Y.; Schapire, R.; Abe, N.: A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, ročník 14, č. 771-780, 1999: str. 1612.

[8] Hajič, J.: Statistické modelování a automatická analýza přirozeného jazyka (morfologie, syntax, překlad). *Slovenčina a čeština v počítačovom spracovaní. Bratislava: Veda*, 2001: s. 11–33.

[9] Hartley, R.; Zisserman, A.: *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[10] Haykin, S. S.; Haykin, S. S.; Haykin, S. S.; aj.: *Neural networks and learning machines*, ročník 3. Pearson education Upper Saddle River, 2009.

[11] Institut biostatistiky a analýz Masarykovy univerzity: *Turingův test*. [Online; navštíveno 20.01.2019].
URL http://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--uvod--posouzeni-inteligence-strojoveho-algoritmu--turinguv-test

[12] Jansen, T.: *Analyzing evolutionary algorithms: The computer science perspective*. Springer Science & Business Media, 2013.

[13] Jivani, A. G.; aj.: A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, ročník 2, č. 6, 2011: s. 1930–1938.

[14] Keller, F.: Naive Bayes Classifiers. *Connect. Stat. Lang. Process. Course Univ. Saarlandes*, 2002.

[15] Khan, A.; Baharudin, B.; Lee, L. H.; aj.: A review of machine learning algorithms for text-documents classification. *Journal of advances in information technology*, ročník 1, č. 1, 2010: s. 4–20.

[16] Korde, V.; Mahender, C. N.: Text classification and classifiers: A survey. *International Journal of Artificial Intelligence & Applications*, ročník 3, č. 2, 2012: str. 85.

[17] Lucas, P.; Van Der Gaag, L.: *Principles of expert systems*. Addison-Wesley Wokingham, 1991.

[18] Manning, C.; Raghavan, P.; Schütze, H.: Introduction to information retrieval. *Natural Language Engineering*, ročník 16, č. 1, 2010: s. 100–103.

[19] McCallum, A.; Nigam, K.; aj.: A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, ročník 752, Citeseer, 1998, s. 41–48.

[20] Murphy, K. P.; aj.: Naive bayes classifiers. *University of British Columbia*, ročník 18, 2006: str. 60.

[21] Nemec, D.: *porter2-stemmer*. [Online; navštíveno 15.01.2019].
URL https://github.com/nemec/porter2-stemmer

[22] Paice, C. D.: An evaluation method for stemming algorithms. In *SIGIR'94*, Springer, 1994, s. 42–50.

[23] Pavel Kordík, J. M.: *Vytěžování znalostí z dat*. [Online; navštíveno 02.04.2019].
URL https://edux.fit.cvut.cz/oppa/BI-VZD/prednasky/p7-Bayes.pdf

[24] Russell, S. J.; Norvig, P.: *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[25] Sakkis, G.; Androutsopoulos, I.; Paliouras, G.; aj.: Stacking classifiers for anti-spam filtering of e-mail. *arXiv preprint cs/0106040*, 2001.

[26] Saygin, A. P.; Cicekli, I.; Akman, V.: Turing test: 50 years later. *Minds and machines*, ročník 10, č. 4, 2000: s. 463–518.

[27] Schmidhuber, J.: Deep learning in neural networks: An overview. *Neural networks*, ročník 61, 2015: s. 85–117.

[28] Scikit-learn developers: *Multiclass and multilabel algorithms*. [Online; navštíveno 21.01.2019].
URL https://scikit-learn.org/stable/modules/multiclass.html

[29] Sebastiani, F.: Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, ročník 34, č. 1, 2002: s. 1–47.

[30] Souza, C.; Kirillov, A.; Catalano, M. D.; aj.: The Accord.NET Framework. 2014, doi:10.5281/zenodo.1029480.
URL http://accord-framework.net

[31] Willett, P.: The Porter stemming algorithm: then and now. *Program*, ročník 40, č. 3, 2006: s. 219–223.

[32] Witten, I. H.; Frank, E.; Hall, M. A.; aj.: *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[33] Zhang, G.; Hu, M. Y.; Patuwo, B. E.; aj.: Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis. *European journal of operational research*, ročník 116, č. 1, 1999: s. 16–32.

Příloha A

Obsah CD

src Zdrojové kódy aplikace.

report Tento text ve formátu PDF.

src Zdrojové texty v LaTeXu a použité obrázky.