Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Rozpoznávání názvů značek v sociálních mediích

Plzeň, 2016 Bc. Lukáš Witz

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 23. června 2016

Bc. Lukáš Witz

Abstrakt

Cílem práce je prozkoumat metody používané pro rozpoznávání pojmenovaných entit, z těchto technik jednu vybrat, implementovat, ověřit funkčnost porovnáním s již existujícími systémy a následně aplikovat na data pocházející ze sociálních médií, v nichž identifikuje názvy organizací a produktů.

Implementované řešení staví na knihovně pro strojové učení Brainy a pro realizaci používá jí poskytovaný algoritmus Conditional Random Fields. Vytvořený systém na standardním korpusu (Czech Named Entity Corpus) dosahuje podobných výsledků jako ten, který se snaží napodobit.

Systém pro korpus dosahuje úspěšnosti 70,69 % (Micro F-measure strict) a pro data pocházející ze sociálních sítí 83,04 %.

Hlavním přínosem této práce je vytvoření systému umožňujícího rozpoznávání pojmenovaných entit v textu a otestování jeho výkonnosti na komentářích pocházející z internetového fóra zaměřujícího se především na jednu doménu, kterou jsou telekomunikace.

Klíčová slova: rozpoznávání pojmenovaných entit, strojové učení, sociální média

Abstract

The goal of this thesis is to survey methods used for the Named-entity recognition, to choose one, implement it, verify the functionality by comparing it to an already existing system and apply it on data extracted from social media and recognize names of products and organisations.

The implemented solution builds on a machine learning library named Brainy and uses its Conditional Random Fields implementation. The created system achieves similar results on the Czech Named Entity Corpus as the system we try to reproduce.

The system performance for the corpus (measured in Micro F-measure strict) is 70.69 and it scores 83.04 for the social media data.

The main benefit this thesis brings is a system able to recognise named entities and test its performance on comments from a forum focusing on telecommunication.

Keywords: Named-entity recognition, machine learning, social media

Poděkování

Mnohokrát děkuji vedoucímu práce, Ing. Michalu Konkolovi, Ph.D., za jeho trpělivost a odborné vedení. Dále děkuji Ing. Vojtěchu Fričovi za nesčetné rady. Obrovská vděčnost patří rodině, která mne po celou dobu studia podporovala. Speciálně bych pak chtěl vyjádřit dík mému bratru, PhDr. Petru Witzovi, za pomoc s překlady. Za podporu a za korekturu bych též rád vyjádřil díky Ing. Robertu Adamcovi.

Obsah

1 Úvod

Příchod internetu, sociálních médií a snadno dostupné výpočetní techniky dal vzniknout tzv. *Informační společnosti*, kterou [1] popisuje takto: „Společnost založená na integraci informačních a komunikačních technologií do všech oblastí společenského života v takové míře, že zásadně mění společenské vztahy a procesy. Nárůst informačních zdrojů a komunikačních toků vzrůstá do té míry, že ho nelze zvládat dosavadními informačními a komunikačními technologiemi.“

Jak definice říká, lidé a stroje produkují dnešními technologiemi v reálném čase nezpracovatelné množství surových *dat*, která většinou bez užitku zabírají místo na úložištích a jejich osudem je tak čekat na případné budoucí využití či smazání. Přitom je při inteligentním pohledu na takovéto zdroje dat možné extrahovat užitečné *informace* například nalezením skrytých vzorů a podobností atd.

Z těchto důvodů, obzvláště v posledních letech, nastal obrovský rozmach automatického zpracování videa, obrázků a textu, na který se tato práce především soustředí. Automatizace analýzy tak dnes již není pouhým rozmarem či výhradou vědeckých pracovišť, ale přímo nutností pro fungování některých služeb a pro nabytí konkurenční výhody. Je proto zapotřebí provádět třídění dokumentů (textů), jejich rozbory, překlady a mnoho dalších netriviálních a pro člověka (nejen časově) náročných operací.

Jednou z takových činností je vyhledávání a označování *pojmenovaných entit*, čili vlastních jmen, názvů lokací a organizací, produktů atd. na které se soustředí tato práce. Jejím cílem je prozkoumat algoritmy používané pro tyto účely, aplikovat je na texty, získané ze sociálních medií, a změřit jejich účinnost a funkčnost.

2 Strojové učení

Vedle obrazových a zvukových dat je internet zdrojem dat textových, publikovaných v podobě novinových článků, blogů a uživatelských komentářů, na které se tato práce především zaměřuje. Tyto texty jsou psány v lidem přirozeném jazyce, a proto se i obor zabývající se jejich analýzou nazývá *Zpracování přirozeného jazyka* (*Natural language processing* – NLP). v úvodu již bylo řečeno, že oblastí, která nás z NLP bude zajímat nejvíce, je *Rozpoznávání pojmenovaných entit* (*Named entity recognition* – NER).

Hlavním přístupem k řešení problematiky NER jsou postupy založené na strojovém učení (SU). Z důvodů uvedených dále byly tyto metody využity i při realizaci této práce. Pro pochopení a správné porozumění dalšímu textu je na místě strojové učení uvést a zadefinovat jeho základní pojmy. Pojem *Strojové učení* definuje Arthur Samuel jako „Obor, který dává počítačům schopnost *učit se* bez toho, aby byly přímo naprogramovány".

Aby byla definice úplná a pochopitelná, je zapotřebí též objasnit pojem *učení* ve spojení s počítačovým programem. Tom M. Mitchell tuto schopnost popisuje následovně: „O programu prohlásíme, že má schopnost se učit ze zkušenosti E s ohledem na množinu úkolů T a výkonnostní míru P, pokud s rostoucí zkušeností E roste jeho výkon ve vykonávání T měřený mírou P".

Proces učení zahrnuje získávání nových deklarativních znalostí, vývoj motorických a kognitivních schopností skrze instruktáže či cvičení, organizace nových znalostí do obecných efektivních reprezentací a objevování nových faktů a teorií skrze pozorování a experimenty [3]. Od samých počátků počítačového věku se vědci snažili opatřit takovými schopnostmi i stroj. Vyřešení takového problému vždy bylo, a stále je, největší výzvou v oboru *umělé inteligence*.

Je žádoucí vlastností systému, aby byl schopen samostatně dynamicky reagovat na nová data a nové podněty, měnit svou vnitřní strukturu nebo hodnoty parametrů tak, aby získané poznatky správně aplikoval a zlepšil svůj výkon za běhu bez zásahu programátora. Právě tato schopnost se označuje jako *učení*.

Strojové učení je disciplína, která není žádnou novinkou. Její nedávný rozmach je zapříčiněn převážně příchodem nových a relativně levných výpočetních technologií, které jsou svým výkonem schopné vdechnout život dlouho

existujícím algoritmům aplikovatelným na velká data. Dalším důvodem k tomuto vývoji je dostupnost vhodných dat dostatečného rozsahu, která lze snadno, rychle a často i zadarmo získat z různých zdrojů na internetu.

Motivací k použití strojového učení obvykle lze nalézt celou řadu. Pro příklad:

- Provádíme analýzu dat, jejichž vnitřní podobnost či souvislost je dopředu neznámá a cílem je ji nalézt.
- o datech a požadovaném výstupu máme dobrou znalost, ale množina pravidel popisujících jejich vzájemný vztah je příliš velká či těžko definovatelná, a proto chceme její sestavení ponechat na algoritmu.
- Program je vyvíjen pro dynamické prostředí či prostředí, které není dopředu zcela známé, kde je potřeba reagovat na změnu podmínek.

2.1 Metody učení

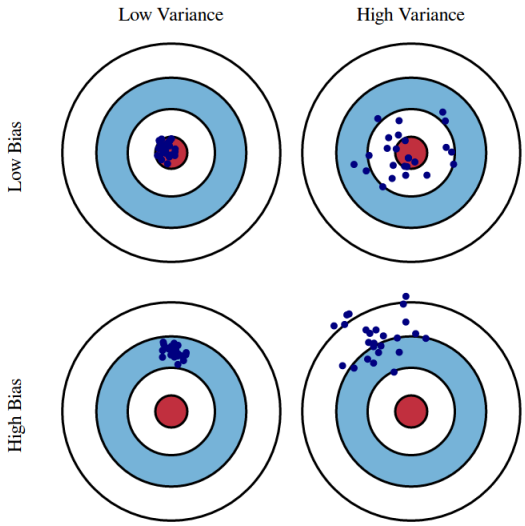
Učení systému znamená vytvoření odvozené funkce, která po dokončení trénování dokáže provádět správné přiřazení výstupních hodnot k dosud neviděným vstupům. Počáteční seznámení systému s podobou zpracovávaných dat může probíhat za přispění tzv. učitele nebo bez jeho pomoci.

2.1.1 Učení s učitelem

První způsob, učení s učitelem (*supervised learning*) probíhá tak, že do systému vstupují data, tzv. vektor příznaků, společně s přidanou informací, dodanou učitelem, kterou představuje očekávaný výstup. Algoritmus pak podle těchto dvojic upravuje své parametry tak, aby byl schopen správně tvořit požadované výstupy. Nevýhodou tohoto postupu je nutnost tvorby označkovaných dat, což je často netriviální a časově náročné. Navíc při tomto typu trénování systému narážíme na následující 4 typy chyb.

První z nich je nalezení kompromisu mezi *vychýlením* a *rozptylem* (*Bias-variance dilemma*). Chyba způsobená vychýlením je dána rozdílem mezi očekávanou, či průměrnou, odpovědí systému a správnou hodnotou. Rozptyl je chápán jako variabilita odpovědí systému pro jeden vstup. Vysoký bias může

být způsoben aplikací jednoduchého algoritmu na složitý problém. Užití komplikovaného algoritmu na jednoduchý problém na druhou stranu může mít za následek vyšší variance. Vysoká míra variance můžu způsobit tzv. *overfitting*, neboli modelování náhodného šumu v trénovacích datech namísto požadovaného výstupu. Oba problémy srozumitelně ilustruje obrázek 2.1 převzatý z [2].



Obrázek 2.1: Vizualizace Bias-variance dilemma

Zašuměný výstup

Poslední problém se týká očekávaných výstupů, které algoritmu předkládáme společně s daty. v některých případech mohou totiž obsahovat chyby, jako je selhání lidského faktoru, chyba měření atd., a je tedy nežádoucí, aby se algoritmus snažil dosahovat totožných výsledků pro dané vstupy. Pokud jsou takovéto chyby v datech výjimečné, lze je ignorovat či odstranit jako tzv. *outliers*. Některé metody s výhodou používají metodu *včasného zastavení* (*early stopping*).

2.1.2 Učení bez učitele

Druhým způsobem, jak lze učení systému uskutečnit, je *učení bez učitele*, (*unsupervised learning*), kdy do systému vstupují samotná data bez jakékoliv další informace o očekávaném výstupu. Systém si musí na jejich základě

vytvořit formální reprezentaci použitelnou pro tvorbu rozhodnutí a předvídání neviděných dat. z tohoto pohledu lze tento typ učení považovat za hledání vzorů a vnitřních podobností. Klasickými zástupci učení bez učitele jsou shlukování či již zmíněná redukce dimenzionality. Základním stavebním kamenem je *pravděpodobnostní model* vstupních dat. Takový pravděpodobnostní model pak může sloužit pro klasifikaci, či datovou kompresi.

3 Zpracování přirozeného jazyka

Zpracování přirozeného jazyka, někdy též *počítačová lingvistika* (*computational liguistics*), se obecně zabývá analýzou a tvorbou textů a mluveného slova. Snahou je dosažení porozumění vstupu dodaného člověkem, používajícím pro něj přirozený jazyk, a schopnost poskytnutí výstupu ve stejné formě, tedy inteligentní komunikace mezi člověkem a strojem.

NLP kombinuje mnoho vědních disciplín, především pak ty matematické a jazykovědní. Konkrétními úlohami NLP jsou například rozpoznávání řeči a její syntéza, která souvisí s generováním přirozeného jazyka, strojovým překladem apod. Významnou oblastí je pak extrakce informací, dolování dat z textu či automatická sumarizace. Úloha řešená v této práci se nazývá *rozpoznávání pojmenovaných entit*.

3.1 Rozpoznávání pojmenovaných entit

Výrazem *pojmenovaná entita* rozumíme například názvy osob, organizací a míst či čísla a datumy. Význam jejich nalezení pro NLP spočívá především ve vyznačení důležitých úseků textu, které usnadňuje další, sémantickou, analýzu textu. Jako příklad použijme větu „Hledám spojení z **Plzně** do **Olomouce** v **18:30**.“. Systém pro vyhledávání dopravního spojení, který má poskytnout odpověď, musí být schopen rozpoznat tři vyznačené entity (parametry vyhledávání). Jejich identifikace je podmínkou nutnou a pro vykonání kýžené funkce by se s výhradou dala považovat za dostačující, ačkoliv je samozřejmě ještě zapotřebí správně určit výchozí bod a destinaci trasy.

Jiné uplatnění NER lze nalézt v oblasti bližší této práci: označení takovýchto slov či úseků textu umožňuje jej indexovat a tyto důležité úseky např. uložit do databáze pro pozdější použití. Při hledání jejich výskytu v dokumentu pak již není nutné prohledávat celý text, ale postačí jednoduchý a rychlý náhled do vytvořené struktury, který vydá informace o výskytech hledaných klíčových slov.

NER je netriviální úloha a je zapotřebí konkrétní systém přizpůsobit podle daného případu použití. Roli hraje především jazyk a typ textu, tedy např. smlouva, novinový článek, blog, chat atd., dále pak oblast, kterou se

text zabývá či účel systému: pokud se soustředíme na označování pouze některé konkrétní skupiny, např. osob, budeme tomu moci více přizpůsobit celý systém a zvýšit tak i jeho výkon.

3.2 Metody rozpoznávání pojmenovaných entit

Za dobu existence problému vyhledávání pojmenovaných entit vzniklo několik přístupů k realizaci řešení majících rozdílný přístup k problému i různou úspěšnost. Následuje přehled nejvýznamnějších metod řešení, které budou, alespoň ve zjednodušené podobě, vysvětleny.

3.2.1 Pravidlový přístup

Pokud rozlišujeme entity, jejichž podoba je dopředu známá a je v rámci celého textu pro ně charakteristická a jedinečná, je nejsnazší cestou ruční sestavení pravidel, které tuto podobu zachycují. Pravidlem může být například regulární výraz – popis délky, kombinace malých a velkých písmen, pozice interpunkce atd.

Udržování sady pravidel je obtížný úkol. Jejich nevýhoda tkví v nutnosti ruční redefinice pravidel pro kvalitní výstup při změně charakteru dat. Často je zapotřebí přidat informace o kontextu či příliš obecná pravidla více konkretizovat. Přesto lze nalézt mnoho realizací, které tuto metodu s úspěchem využívají. z výše uvedeného důvodu se zaměříme spíše na metody strojového učení, které tento nedostatek odstraňují.

3.2.2 Statistický přístup

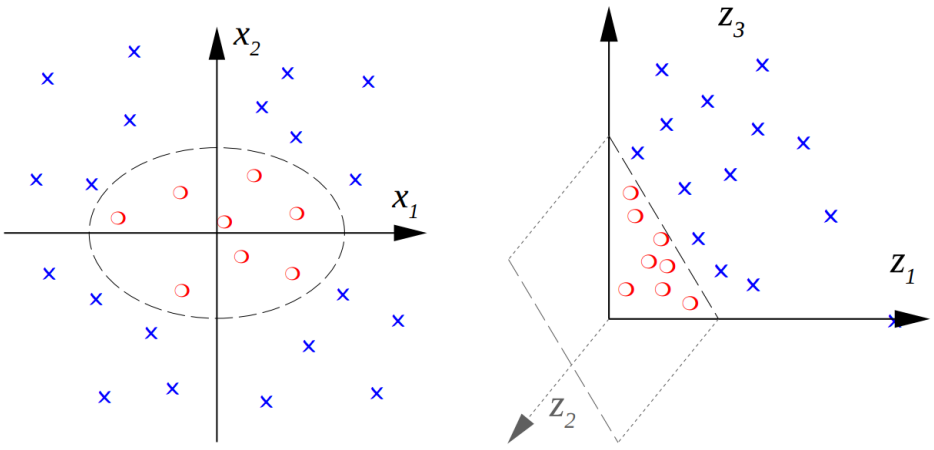
Složitější, ale obecně silnější, metodou je statistický přístup, respektive využití strojového učení, který umožňuje do procesu rozpoznávání zahrnout složitější jazykové příznaky a kontext vstupu. Cenou za vysoký výkon algoritmu je potřeba velkých trénovacích dat, která byla diskutována výše (2.1.1).

Support vector machine

Mnohé z metod strojového učení stavějí na relativně jednoduchých, ale efektivních, metodách, například jednovrstvé neuronové sítě. Jejich výraznou nevýhodou je ale nevhodnost pro řešení obecných klasifikačních úloh daná schopností nalézt oddělovací hranice tříd ve vstupním prostoru pouze v lineární podobě.

Tento jejich nedostatek řeší složitější metody, například vícevrstvé neuronové sítě, které umožňují identifikovat hranice složitějších tvarů. Problémem těchto složitějších metod ovšem obecně bývá vyšší složitost, komplikovanější metody učení a možnost uvíznutí při výpočtu pokutové funkce.

Populární mezi těmito složitějšími metodami jsou *podpůrné vektory* (*Support vector machine* – SVM), které jsou schopé nalézt nejenom triviální lineární hranice, ale i ty velmi složité, nelineární, za využití projekce původního prostoru do prostoru o vyšší dimenzi, ve kterém jsou třídy lineárně separovatelné. SVM provádí klasifikaci vstupu do dvou tříd, v případě, kdy je tříd více, použije se více binárních klasifikátorů. Často uváděným příkladem je separace dvou tříd oddělených kružnicí či elipsou, které lze separovat lineárně přidáním dimenze 3.1. Tohoto lze využít obecně pro libovolná data. Při projekci do dostatečným počtem dimenzí lze vždy najít separující nadrovinu, tzn. vždy lze oddělit N různých bodů v prostoru o dimenzi alespoň $N - 1$ [11].



Obrázek 3.1: Separace tříd za využití SVM [12]

Problémem SVM, a obecně problémem klasifikace, je optimalizace umís-

tění hranice. Vzorky vyskytující se v trénovacích datech nemusejí dostatečně popisovat celková data a je tedy zapotřebí předjímat neviděné stavy a jejich kategorizaci. Vzhledem k tomu, že v D-dimenzionálním prostoru je hranice definována rovnicí o D parametrech, hrozí navíc nebezpečí přetrénování klasifikátoru a ztráta jeho obecnosti, pokud D je blízké N. z tohoto důvodu se hledá řešení s maximální „bezpečnostní zónou“.

Pro formální popis SVM mějme n k-dimenzionálních vektorů reálných čísel $\mathbf{x}_i$ a celá čísla y_i, kde $y_i = \{-1, 1\}$. Hodnota y_i indikuje třídu vektoru i. Cílem trénování klasifikátoru je nalezení nadroviny oddělující body z těchto tříd. Tento problém lze převést [11] na hledání parametrů α_i maximalizujících výraz:

$$\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \qquad \alpha_i \geq 0, \sum_i \alpha_i y_i = 0.$$

Tento výraz má efektivně nalezitelné jedno jediné globální maximum. Po nalezení optimálních hodnot α_i lze optimální oddělovač vyjádřit jako

$$h(x) = sign(\sum_i \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i)).$$

Platí, že separátor má nulové váhy α_i pro všechny body kromě těch, které jsou nejblíže vlastnímu oddělovači. Právě ty jsou pak nazývány *support vectors*. Ostatní datové body a vektory, kterých je mnohem více, nejsou pro SVM podstatné a počet parametrů popisujících optimální separátor zdaleka nedosahuje N.

Obecně separátor hledáme ve vícerozměrném prostoru $F(\mathbf{x})$, i když se může stát, že bude lineární separátor možno vytyčit v originálním vstupním prostoru. Pro hledání ve vícerozměrném prostoru nahradíme člen $x_i \cdot x_j$ členem $F(\mathbf{x}_i) \cdot F(\mathbf{x}_j)$. Tento skalární součin není ale vždy nutné explicitně počítat.

Pravá část výrazu se nazývá *kernel function* (*jádrová funkce*) $K(\cdot, \cdot)$, která může být aplikována na dvojice vstupních dat pro určení výsledku skalárního součinu v odpovídajícím prostoru. Lze tedy najít lineární oddělovače ve vícerozměrném prostoru $F(x)$ prostou náhradou $x_i \cdot x_j$ jádrovou funkcí $K(x_i, x_j)$. Jinak řečeno, lze provádět učení ve vícerozměrném prostoru pouze za počítání jádrové funkce místo úplného seznamu atributů pro všechny body.

Je-li zapotřebí hledat oddělovač ve vícerozměrných prostorech, lze najít jinou podobu jádrové funkce. Podle *Mercerova teorému* jakákoliv „rozumná“ jádrová funkce odpovídá nějakému prostoru atributů, a to i prostorům s vysokou dimenzionalitou. Polynomická jádrová funkce $K(x_i, x_j) = (1 + \mathbf{x}_i \cdot \mathbf{x}_j)^d$ například odpovídá prostoru atributů, jehož dimenze roste exponenciálně s d.

Dalším krokem po nalezení lineárního oddělovače je jeho projekce zpět do původního prostoru. Výsledná projekce bude mít obvykle nelineární charakter. Může se jednat o komplikovanou křivku, kružnici atd.

Maximum entropy classifier

Model maximální entropie staví na principu vytvoření co nejméně jiných předpokladů o datech, než těch danými z trénovacích dat vytvořenými omezeními, na základě kterých tvoří pravděpodobnostní model. Vytváříme množinu příznaků, nebo lépe funkcí, popisujících vztah mezi příznaky a výstupem. Pravděpodobnostní model (rozdělení), který splňuje takové podmínky, má nejvyšší možnou *entropii*. Je zaručené, že existuje, je jedinečný a má exponenciální tvar [9]

$$p(o|h) = \frac{1}{Z(h)} \cdot \prod_{j=1}^{k} \alpha_j^{f_j(h,o)},$$

kde o značí výstup, h kontext a $Z(h)$ je normalizační faktor zaručující, že $p(o|h)$ bude pravděpodobnostní rozdělení. Navíc odpovídá rozdělení *maximum-likelihood* (maximální věrohodnosti).

Příznaky používané v ME jsou binární. Příkladem budiž příznaková funkce:

$$f_j(h, o) = \begin{cases} 1 & slovo = \text{Praha}, o = \text{LOC-B} \\ 0 & \text{jinak} \end{cases}.$$

Parametry modelu α jsou pak na tomto základě získávány podle zvoleného algoritmu.

ME klasifikátor poté přiřazuje každému slovu nezávisle jednu z těchto tříd:

- začátek pojmenované entity (B),

- slovo uvnitř pojmenované entity (C),
- poslední slovo pojmenované entity (L),
- jediné slovo v rámci pojmenované entity (U).

Systém ale může při klasifikaci vytvořit nevalidní sekvenci, např. „LOC-B PER-L“, která postrádá smysl. Takové sekvence lze odstranit, pokud budeme uvažovat pravděpodobnost přechodu $P(c_i|c_j)$ mezi třídami c_i a c_j přiřazenými dvěma po sobě jdoucím slovům. Pokud je takový přechod možný, bude pravděpodobnost rovna 1, v opačném případě bude nulová. Uvažujme nyní dokument D obsahující větu s. Pravděpodobnost tříd $c_1, \ldots, c_n$ přiřazených slovům v s určíme jako:

$$P(c_1, \ldots, c_n|s, D) = \prod_{i=1}^{n} P(c_i|D) \cdot P(c_i|c_{i-1}),$$

kde $P(c_i|s, D)$ je určeno ME klasifikátorem. Na závěr je použit Viterbiho algoritmus k nalezení sekvence tříd s největší pravděpodobností.

Neorientované pravděpodobnostní grafické modely

Základním stavebním kamenem aplikací zabývajících se zpracováním signálu, obrazu, biologických dat a přirozeného textu nebo aplikací počítajících například výhodnost pozic a tahů v deskových hrách je schopnost předpovídat neviděné či neznáme veličiny a proměnné, které jsou na sobě určitým způsobem závislé. Formálně hledáme výstupní vektor $\mathbf{y} = \{y_0, \ldots, y_t\}$ (při analýze textu by jednotlivé prvky vektoru příslušely například jednotlivým tokenům) na základě vektoru *příznaků*. Příznakem by mohla být pozice slova, přítomnost velkého písmena atd.

Pokud je $\mathbf{x}$ množina n náhodných proměnných, pak $P(\mathbf{x})$ je sdružená pravděpodobnost všech těchto proměnných. Mějme dvě podmnožiny $\mathbf{x}$: $\mathbf{x}_A$ a $\mathbf{x}_B$, které jsou na sobě při daném $\mathbf{x}_C$ podmíněně nezávislé. Pravděpodobnost $P(.)$ dodržuje tuto podmíněnou nezávislost, pokud platí výrok [10]

$$P(\mathbf{x}_A|\mathbf{x}_B, \mathbf{x}_C) = P(\mathbf{x}_A|\mathbf{x}_C),$$

resp.

$$\begin{aligned} P(\mathbf{x}_A, \mathbf{x}_B|\mathbf{x}_C) &= \frac{P(\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_C)}{P(\mathbf{x}_C)} \\ &= \frac{P(\mathbf{x}_A|\mathbf{x}_B, \mathbf{x}_C)P(\mathbf{x}_B, \mathbf{x}_C)}{P(\mathbf{x}_C)} \\ &= P(\mathbf{x}_A|\mathbf{x}_C)P(\mathbf{x}_B|\mathbf{x}_C). \end{aligned}$$

Často se používá zkrácený zápis [10] $\mathbf{x}_A \perp \mathbf{x}_B|\mathbf{x}_C$.

Máme-li tedy $\mathbf{x}$ a seznam výroků o podmíněné nezávislosti, je cílem našeho snažení nalézt skupinu pravděpodobnostních rozdělení nad $\mathbf{x}$, které odpovídají takovým výrokům. Mějme neorientovaný graf $G = (X, E)$, jehož uzly odpovídají naší množině náhodných proměnných. Pro hrany musí platit, že pokud odstraněním všech uzlů náležících množině $\mathbf{x}_C$ odstraníme všechny všechny cesty z $\mathbf{x}_A$ do $\mathbf{x}_B$, platí $\mathbf{x}_A \perp \mathbf{x}_B|\mathbf{x}_C$.

Jedním ze způsobů, jak reprezentovat vzájemné vztahy mezi výstupními proměnnými, je tvorba grafického modelu, jehož představiteli jsou Markovské sítě či Isingův model atd.

Hidden Markov model

Knihovna použitá při realizaci této práce používá pro klasifikaci algoritmu *Conditional Random Fields* (CRF), který je rozšířením Skrytého Markovova modelu *Hidden Markov Model* (HMM). Pro lepší porozumění CRF nejdříve představíme HMM.

Skrytý Markovův model (HMM) je statistický model pojmenovaný po ruském matematikovi Andreyi Markovovi reprezentující *Markovův* (Markovský) *proces*, což je náhodný proces, jehož budoucí pravděpodobnosti jsou určeny jeho nejnovějšími hodnotami. Čili vzdálená minulost je nedůležitá ve srovnání se znalostí minulosti blízké. Ve zpracování jazyka se obvykle omezuje použitá historie pouze na několik předchozích značek.

V jednoduchých modelech (jakým je například Markovův řetězec) je stav automatu pozorovateli viditelný, v HMM jej však není možné pohledem zvenčí určit, neboť je skrytý, a zřejmý je pouze na něm závislý výstup, který je při každém kroku produkován. Jedná se tedy o konečný stavový automat,

kde každý přechod mezi stavy má určitou pravděpodobnost. Platí, že přechod ze stavu S_{t-1} do stavu S_t je nezávislý na historii, tedy všech stavech předcházejících stavu S_{t-1}. Sledujeme-li tedy posloupnost výstupů systému generujících určitý textový řetězec, získáváme i představu o posloupnosti stavů. Pozorování není kontinuální, nýbrž probíhá v diskrétním čase.

Formálně je HMM stochastický konečný automat s jediným možným vstupem, definovaný konečnou množinou vnitřních stavů S, množinou možných pozorování O a tří pravděpodobnostních rozdělení, definovaný jako

$$\Lambda = (A, B, \Pi),$$

kde A označuje množinu hodnot pravděpodobností, označovaných jako a, kde

$$a = P(s|s'),$$

kde $P(s|s')$ označuje pravděpodobnost přechodu ze stavu s' do s, pro $s, s' \in S$. B označuje množinu hodnot pravděpodobností generování – výstupu, označených jako b. $P_0(s), \in S$ označuje pravděpodobnost výchozího stavu. Definice [13] říká:

$$\pi = \{p_i = P(s_i[t = 1])\}$$
$$S = \{s_i\} \quad i = 1, ..., N,$$

kde N označuje počet skrytých stavů a t označuje čas.

$$A = \{a_{ij} = P(s_j[t + 1]|s_i[t])\}$$

A je tedy pravděpodobnost, že následuje stav s_j za předpokladu, že aktuálním stavem je s_i.

$$O = \{o_k\} \quad k = 1, ..., M,$$

kde M označuje počet pozorovaných (vstupních) hodnot. O je pravděpodobnost, že výstup je o_k za předpokladu, že aktuálním stavem je q_i.

$$B = \{b_{ik} = b_i(o_k) = P(o_k|s_i)\}$$

Vzhledem k výše popsaným vlastnostem můžeme určit sdruženou pravděpodobnost posloupnosti stavů a pozorování na základě rovnice

$$P(S_{1...t}, O_{1...t}) = P(S_1)P(O_1|S_1) \prod_{t=2}^{T} P(S_t|S_{t-1})P(O_t|S_t).$$

Problémy HMM Použití HMM se váže se třemi problémy:

- *problém vyhodnocení* – jak efektivně spočítat pro konkrétní pozorovanou posloupnost pravděpodobnost, že bude vygenerována modelem,
- *problém dekódovací* – jak vybrat odpovídající posloupnost stavů k zadanému modelu, která by co nejlépe popisovala pozorovanou posloupnost,
- *problém učení* – jak odhadnout parametry modelu λ k pozorované posloupnosti O tak, aby pravděpodobnost $P(O|\lambda)$ byla maximální – pozorované posloupnosti používané k odhadování parametrů modelu se říká trénovací posloupnost, neboť slouží právě k trénování HMM.

Pro určení parametru modelu je používán MLE (*maximum likelihood*) odhad, který ovšem nelze použít vždy. Jsme ale schopni za využití numerických iterativních či gradientních metod, např. *Baum-Welch* či ekvivalentní *expectation-maximization* algoritmus, nalézt model λ takový, že pro pravděpodobnost $P(O|\lambda)$ nalezneme alespoň lokální maximum.

Conditional Random Fields

Termínem CRF se označuje neorientovaný grafický model, který se používá k určení podmíněných pravděpodobností výstupních uzlů podle uzlů vstupních. Každý uzel, v našem případě slovo, popisuje náhodnou proměnnou. CRF lze také jinak popsat jako model s konečným počtem stavů a nenormalizovanými pravděpodobnostmi přechodů. Na rozdíl od HMM nemá přesně určené hodnoty přechodů mezi stavy a disponuje možností mnohonásobných funkcí generujících příznaky. Obecně dosahuje lepších výsledků ve zpracování dat se závislostmi vyššího řádu, které většinou lépe odpovídají reálnému modelu. v poslední době se pro určení parametrů tohoto modelu často využívá algoritmus *L-BFGS* (*Limited-Memory BFGS*), který rychle konverguje [16].

Formálně se CRF definuje takto: nechť je X náhodnou proměnnou dat, v tomto případě vět, a Y náhodnou proměnnou k X příslušných značek. Všechny prvky Y_i z Y náleží abecedě χ, která tedy obsahuje množinu všech použitých značek. Vytvoříme model $P(X|Y)$ z párů pozorovaných hodnot a sekvencí značek a CRF pak můžeme vyjádřit následovně:

Nechť $G = (V, E)$ je grafem, u kterého platí $Y = (Y_v), v \in V$ tak, že Y je indexováno body vektoru G. Poté (X, Y) vyjadřuje CRF v případě, kdy

náhodné proměnné Y_v jsou podmíněné X a vyhovují Markovské vlastnosti s ohledem na graf vyjádřený vzorcem:

$$p(Y_v|X, Y_w, w \neq v) = p(Y_v|X, Y_w, w \sim v),$$

kde $w \sim v$ značí, že prvky w a v spolu ve vektoru G sousedí.

Ačkoliv by G, X i Y mohly nabývat různých forem grafů, pro potřeby této práce je budeme považovat za jednoduché řetězce:

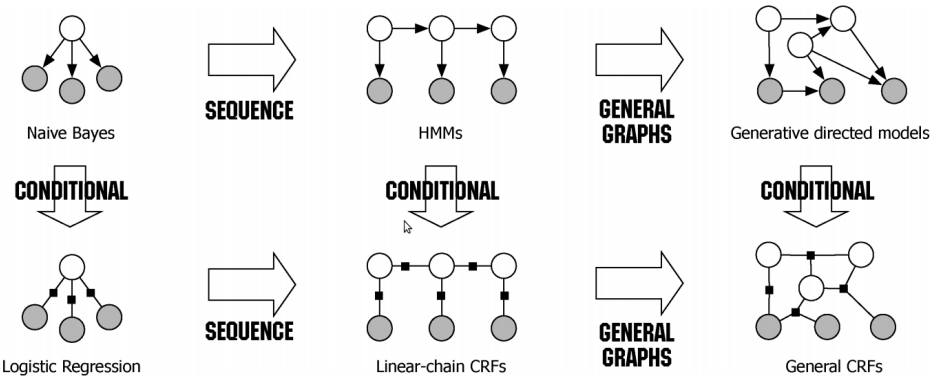
$$G = (V = \{1, 2, \ldots, m\}$$

$$E = \{(i, i\ +1)\}), X = (X_1, X_2, ..., X_n)$$

$$Y = (Y_1, Y_2, \ldots, Y_n).$$

Linear-chain CRF

Již bylo řečeno, že pro realizaci této práce byla použita metoda CRF. Konkrétněji se jedná o její variantu *Linear-chain*, o které lze říci, že je neorientovaným grafickým modelem HMM. Vztah mezi HMM a Linear-chain, a dalšími modely, názorně zachycuje obrázek 3.2



Obrázek 3.2: Diagram vztahů mezi naivním Bayesem, logistickou regresí, HMM, Linear-chain CRF, generativními modely a obecným CRF [15]

Definujme Linear-chain CRF formálně [15]: mějme podmíněné rozdělení $p(y|x)$ odvozené ze sdruženého rozdělení HMM $p(y, x)$, což je ve skutečnosti CRF s vybranou jednou konkrétní skupinou příznakových funkcí (*feature functions*), což jsou funkce, které na vstupu očekávají větu s, pozici i slova

ve větě, třídu l_i současného slova, třídu l_{i-1} předchozího slova a na výstupu poskytuje reálné číslo, často však pouze 0 či 1).

Použijeme-li koncept příznakových funkcí, předchozí rovnici můžeme zapsat ve stručnější podobě. Každá taková funkce má podobu $f_k(y_t, y_{t-1}, x_t)$. Mějme příznak $f_{ij}(y, y', x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{y'=j\}}$ pro každý přechod (i, j) a jeden příznak $f_{io}(y, y', x) = \mathbf{1}_{\{y=i\}\mathbf{1}_{\{x=o\}}}$ pro každou dvojici pozorování (i, o). Nyní můžeme zapsat HMM jako:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t).$$

$$Z(\mathbf{x}) = \sum_{y} \prod_{t=1}^{T} \exp \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t).$$

3.2.3 Evaluace rozpoznávání pojmenovaných entit

Vzhledem k tomu, že je zapotřebí nějakým způsobem poměřovat výkonnost systémů či různých verzí téhož a provádět jeho ladění, je zapotřebí zavést způsob, jak úspěšnost měřit.

Na rozdíl od některých jiných úkolů v oblasti NLP se pro úlohu rozpoznávání pojmenovaných entit používají běžné metriky. Uvažujme klasifikaci objektů do dvou tříd *pozitivní* (P) a *negativní* (N):

- *precision* (*přesnost*) – značí, jak velká část objektů zařazených do P je skutečně z P,
- *recall* (*úplnost*) – značí, jak velká část z celkového skutečného P byla zařazena do P,
- *F-measure* (*F-míra*) – je dána *harmonickým průměrem* předchozích dvou.

Při klasifikaci objektů do dvou tříd P a N rozlišujeme tyto třídy výsledků, které jsou názorně zobrazeny na obrázku 3.3

- *skutečně pozitivní* (TP) – značí správné zařazení objektu do třídy P,

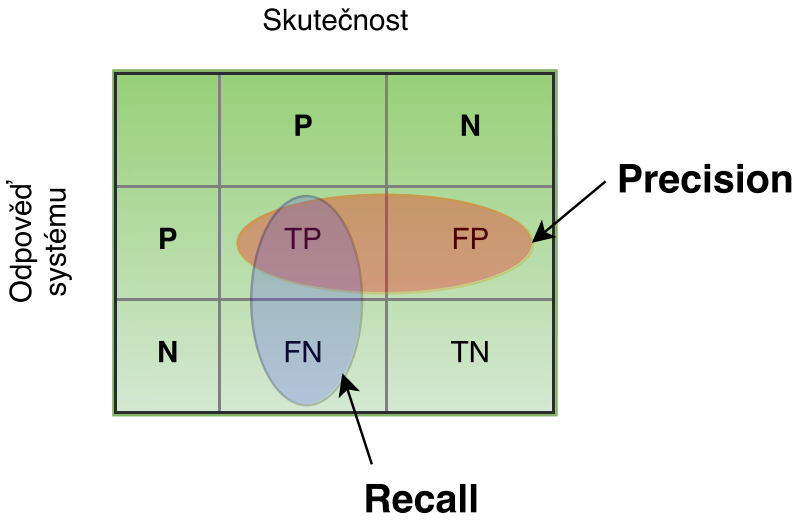
- *skutečně negativní* (TN) – značí správné zařazení objektu do třídy N,
- *falešně pozitivní* (FP) – značí zařazení objektu, který patří do skupiny N, do třídy P,
- *falešně negativní* (FN) - značí zařazení objektu, který patří do skupiny P, do třídy N.

Za využití těchto skupin můžeme snadno definovat:

$$\text{Precision} = \frac{P}{P + FP}$$

$$\text{Recall} = \frac{P}{P + FN}$$

$$\text{F} - \text{measure} = \frac{2P}{2P + FP + FN}$$



Obrázek 3.3: Přesnost a úplnost

Přesnost a úplnost jsou dva různé pohledy na výsledek klasifikace, které se navíc navzájem ovlivňují. Zvýšením jednoho se obvykle sníží druhé a jejich harmonický průměr dává celkový pohled na výsledky nehledě na to, která z metrik je lepší. z tohoto důvodu se systémy soustředí podle konkrétní úlohy na jednu z metrik.

Chceme-li aplikovat právě definovaný postup na NER, musíme určit mapování výsledků poskytnutých systémem na třídy TP, TN, FP a FN.

Obecně platí, že systém se při označování entit v textu může dopustit těchto chyb:

- neoznačení entity,
- označení entity, která entitou není,
- správné označení hranic entity (tzv. *spanu*), ale špatně určená třída,
- špatně určené hranice entity, např. místo *Jan Kraus* označeno jako entita pouze *Jan*,
- kombinace předchozího: špatně určené hranice entity a navíc špatně určená její třída.

Existuje pak několik přístupů, jak mapování provádět.

MUC-6 evaluace

První způsob vyhodnocování výsledků NER systémů přišel s příchodem úlohy jako takové [5]. Výběr metrik byl uskutečněn v rámci konferencí MUC a je založen na způsobech měření ostatních úloh z oblasti NLP, které se do té doby používaly.

MUC-6 hodnotí zvlášť kvalitu hledání hranic entit a jejich typu. Typ je považován za správný, pokud souhlasí s typem určeným učitelem a pokud je jím označena část textu zahrnující celou původní entitu. Porovnávání textu zahrnuje i jeho modifikace, což znamená, že i entita mající nestejnou, upravenou, podobu je považována za správnou.

Pro span i typ jsou počítány správné odpovědi systému (COR), počet učitelem nalezených entit (POS) a počet nalezených entit (ACT). Výkon systému pak určujeme jako součet těchto čísel pro typ i span, z něhož určujeme klasickým způsobem precision, recall a harmonický průměr.

CoNLL evaluace

Tento způsob evaluace je popsán v [4, 6]. Jedná se o striktní metodu, která považuje za správnou odpověď pouze tu, která je zcela bezchybná – musí být správně určeny hranice i třída. Pokud tedy systém místo **Ing. Pavel Novák** označí pouze **Pavel Novák**, je systém penalizován za dvě chyby (FP za **Pavel Novák** a FN za **Ing. Pavel Novák**).

ACE evaluace

Zkratka reprezentuje následovníka MUC, The Automatic Content Extraction. Dvě úlohy na programu se zabývají rozpoznáváním pojmenovaných entit: detekce entit a jejich vztahů a rozpoznávání a normalizace časových údajů. Obě úlohy rozšiřují standardní definici NER.

Evaluace je velmi komplexní a zahrnuje mechanismy mající za úkol překonat problémy související vyhodnocováním: s částečnou shodou, špatným typem atd. ACE též pracuje s pojmy jako podtyp, třída, odkaz na entitu apod. Evaluace je založena na jiných, než běžně používaných, metrikách. Využívá speciální systém ohodnocování, kde každý druh chyby a každý typ entity má jinou důležitost (váhu).

Velkou výhodou je možnost zhodnotit výkon systému podle přesně daných podrobných kritérií. Naproti tomu je ale nutné dodržet stejný způsob hodnocení pro porovnání dvou systémů a navíc je tento způsob neintuitivní a obtížný.

Lenient evaluace

Anglické slovo „lenient“ znamená mírný. k vyjádření výkonu systému používá běžné míry a je založen na GATE evaluaci. Na rozdíl od varianty *strict* se nepoužívá pro potřeby NER příliš často, přestože se pro vyhodnocení výkonu systému může v některých případech hodit více.

Snaží se řešit přílišnou striktnost CoNLL při posuzování výsledků. Odpověď, která je téměř správná, je dokonce dvakrát penalizována. Mírná evaluace považuje za správný výsledek i ten, kde se entita nalézá v identifikovaných hranicích a byl určen správný typ.

Pro tento způsob vyhodnocování musíme zavést jiné třídy používané pro ohodnocení správnosti odpovědi systému:

- *not marked* (nm) – entita nebyla systémem nalezena,
- *not correct* (nc) – označen úsek textu, který není entitou,
- *partially marked* (pm) – posuzuje se pro každou entitu a značí správně určený typ, ale špatně určené hranice,
- *partially correct* – posuzuje se pro každou entitu rozpoznanou systémem. Entita je označena správným typem, ale jsou nepřesně určené hranice,
- *correct* (c) – entita má správně určené hranice a správně určený typ.

Na základě této definice lze pak upravit výpočet metrik následujícím způsobem:

$$\text{Precision} = \frac{c + pc}{c + nc + pc}$$
$$\text{Recall} = \frac{c + pm}{c + pm + nm}$$
$$\text{F} - \text{measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Strict evaluation

Drobnou úpravou je striktní evaluace, která se liší ve výpočtu metrik [17].

$$\text{Precission} = \frac{c}{c + nc + pc}$$
$$\text{Recall} = \frac{c}{c + nm + pm}$$
$$\text{F} - \text{measure} = \frac{2 \cdot c}{2 \cdot c + nc + nm + pc + pm}$$

Tyto dva způsoby evaluace společně vytvářejí horní, danou typem lenient, a dolní, danou typem strict, mez výkonu systému.

Micro a Macro-average

Pomocí těchto ukazatelů se lze dozvědět úspěšnost systému ve vztahu k celým datům, resp. všem třídám (typům) pojmenovaných entit, nebo při použití pro konkrétní třídy i ve vztahu k jednotlivým třídám, kdy pro každý jednotlivý typ získáme zvlášť příslušné ukazatele. Nevýhodou jedné metriky pro celý systém je možná ignorace chyby pro partikulární třídy. Pokud jsou špatně identifikovány téměř všechny entity minoritní třídy, která má malý podíl na celkovém počtu entit, dopad na celkovou f-míru bude minimální, přestože celá jedna třída bude mít tentýž ukazatel neuspokojivý, čehož bychom si všimli pouze z podrobnějších statistik. z tohoto důvodu je vhodné zavést takovou metriku, která bude brát v potaz partikulární výsledky pro jednotlivé třídy.

Pro tyto potřeby se v oblasti zpracování textu používají modifikované výše uvedené metriky. v angličtině obvykle používáme označení *Micro-averaged* (B_{micro}) a *Macro-averaged* (B_{macro}) precision/recall/F-Score [18]. Nejčastěji narazíme na využití v binární klasifikaci $B(tp, tn, fp, fn)$ (tp – true positive, ...). Výpočet provedeme následovně: nechť $L = \{\lambda_j : j = 1 \dots q\}$ je množina všech tříd pojmenovaných entit a nechť tp_λ, fp_λ, tn_λ a fn_λ jsou četnosti jednotlivých kategorií výsledků klasifikace pro třídu λ. Pak platí:

$$B_{macro} = \frac{1}{q} \sum_{\lambda=1}^{q} B(tp_\lambda, fp_\lambda, tn_\lambda, fn_\lambda)$$

$$B_{micro} = B\left(\sum_{\lambda=1}^{q} tp_\lambda, \sum_{\lambda=1}^{f} p_\lambda, \sum_{\lambda=1}^{q} fn_\lambda, \sum_{\lambda=1}^{q} tn\lambda \right)$$

Rozdíl mezi oběma mírami je takový, že zatímco B_{macro} dává stejnou váhu všem třídám, B_{micro} dává stejnou váhu všem rozhodnutím systému napříč textem. B_{micro} je tak vhodnější pro měření efektivity na větších třídách, pro měření téhož na menších třídách je vhodnější B_{macro}.

Pro naše potřeby budeme využívat Micro- a Macro-averaged F-Score, které navíc rozšíříme o dvě varianty: *strict* a *lenient*. Efektivitu systému tak při experimentech budeme hodnotit 4 čísly:

- Micro-averaged F-Measure Strict,
- Micro-averaged F-Measure Lenient,

- Macro-averaged F-Measure Strict,
- Macro-averaged F-Measure Lenient.

3.2.4 Typy pojmenovaných entit

NER je obvykle používáno pro předzpracování vstupu pro další úkoly z oblasti NLP, proto rozlišované typy pojmenovaných entit záleží především na konkrétním použití a zájmu systému.

Obvyklým předmětem zájmu ale bývají původně cílené skupiny (proper names), mezi něž patří osoby, lokace a organizace, souhrnně označovány od MUC-6 jako *enamex*.

Lokace může být dále rozlišována na několik podtříd, např. město, stát či země [7], osoby například na politiky, komiky,... Není nutné hledat pouze podskupiny, můžeme naopak vytvářet zastřešující termíny dobře popisující určitou oblast zájmu. ACE, pro ukázku, zavádí typ „facility“ (zařízení), které reprezentuje organizaci či určitou lokaci, či typ „GPE“ (geo-political entity), který zahrnuje místa mající nějaký typ řídícího, nebo spíše z anglického *government* vládnoucího, orgánu.

CoNLL zavádí typ „miscellaneous“ (různé, ostatní), který zahrnuje jména a názvy, které zcela nezapadají do kategorie *enamex*. v některých případech zahrnuje typ „ product“. Podobně pak bývají řazeny mimo typy *datum* a *čas*, dle MUC souhrnně „timex“, a *peníze* a *procenta* („numex“).

Bez zajímavosti není ani oblast chemie, genetiky a bioinformatiky. Se vznikem a rozšířením korpusu GENIA vznikly nové typy entit jako „protein“, „DNA“, „lék“ či „chemikálie“.

Z výše uvedeného lze vidět, že pro každou doménu lze typy entit upravit a dospecifikovat podle konkrétní potřeby.

3.2.5 Příznaky

Základem systému určeného pro NER a založeném na strojovém učení je klasifikátor, který na základě extrahovaných *příznaků* entity rozpoznává, resp.

přiřazuje slovu každou z možností s určitou pravděpodobností. Slovem *příznak* je myšlen matematický popis určité konkrétní vlastnosti daného slova. Klasifikátor pak má možnost využívat více příznaků, které mohou mít různou míru důležitosti danou vahami.

Příznaky jsou deskriptory či charakteristické atributy slov určené pro algoritmické zpracování. Příkladem budiž jednoduchý binární pravdivostní příznak, pravda či nepravda, indikující přítomnost velkého písmena na začátku slova. Text lze pak reprezentovat jako vektor příznaků, kde každé slovo je zastoupeno jedním nebo více atributy, které bývají rozlišovány na tyto 3 typy:

- booleovské – binární, s hodnotami pravda nebo nepravda (1/0),
- numerické – číselná hodnota, například délka ve znacích,
- nominální – obsahuje upravenou podobu slova, ku příkladu slovo pouze s malými písmeny.

Pravidlové systémy pak aplikují množinu předepsaných pravidel nad těmito vektory. Velmi jednoduchý systém by například obsahoval pravidla *délka* $>$ *2* a *jeVelkePismeno(prvniPismeno)*. Ve skutečných systémech je množina pravidel ale mnohem komplexnější a je vytvářena automaticky učícími se algoritmy. Dále se soustřeďme na příznaky nejčastěji používané pro rozpoznávání a klasifikaci pojmenovaných entit.

Příznaky lze rozlišit na tři typy:

- slovo samotné – velká písmena na začátku slov, přítomnost interpunkce a číslic, speciálních znaků, morfologie slova, slovní druh a jeho pozice v textu, okolí slova, pro některé typy entit bývá typické okolí, čili slovo(a) před ní a po ní. s vysokou pravděpodobností bude např. po titulu **Ing.** následovat jméno osoby, stejně tak zkratka **s.r.o.** bude pravděpodobně následovat po názvu společnosti,
- přítomnost ve slovníku, např. měst, křestních jmen, příjmení, organizací...,
- dokument a korpus – četnost slova, výskyty slova napsaného s malými či velkými písmeny, pozice ve větě, odstavci, dokumentu. Meta informace – URL, pozice v XML...

Na daném slově můžeme pozorovat podobu znaků, ze kterých je složené – malá a velká písmena, speciální znaky, tečky, délku, přítomnost čísel a další:

1. **velká písmena**
   (a) velké písmeno na začátku slova
   (b) celé slovo je kapitálkami
   (c) mix (eBay, YouTube, iPhone)
2. **přítomnost interpunkce**
   (a) obsahuje tečku nebo jí končí (Sv., D.A.S.)
   (b) apostrof, pomlčka (O'Connor, T-Mobile)
3. **přítomnost čísla** – číslice mohou indikovat široké spektrum informací, jako datum, procenta, intervaly, identifikátory apod. Dvouciferná a čtyřciferná čísla mohou reprezentovat například roky či období, pokud jsou jedno- až dvouciferná čísla následována tečkami, může se jednat o datum.
   (a) římské číslice
   (b) slova obsahující číslice (W3Schools)
4. **zvláštní znaky**
   (a) znaky řecké abecedy
   (b) ampersand – &
   (c) paragraf – §
5. **morfologie slova**
   (a) předpona, přípona, stem, jednotné číslo
6. **funkce nad slovy** – umožňují další modifikace či například extrakce nealfanumerických znaků
   (a) n-gramy
   (b) všechna písmena velká či malá
   (c) délka

(d) extrakce vzorů – úkolem je převést slovo podle určitého klíče na jednoduchou množinu znaků udávající podobu slova. Například převedení sekvence velkých písmen písmenem „A“, všech malých písmen písmenem „a“ a všech číslic číslicí „1“. Slovo „iPhone“ by tak například bylo reprezentováno (kondenzovaným, sumarizovaným) vzorem „aAa“.

Slovníky mají v úloze NER zvláštní postavení. Používáme slovníky např. křestních jmen a příjmení, firem, měst, řek, zemí atd. Na první pohled se může zdát, že nalezení daného slova ve slovníku automaticky indikuje, že se jedná o pojmenovanou entitu daného typu. z důvodu polysémie to však nemusí být vždy pravda, například řeka Vydra a stejnojmenné zvíře – pokud bude dokonce slovo na začátku věty, nemůžeme se dokonce ani spolehnout na velké písmeno na začátku slova. v některých případech může navíc slovo mít naprosto totožnou podobu a být například názvem města i státu, jako je tomu v případě názvu „New York“).

Používané slovníky se dají rozdělit na 3 kategorie:

1. obecný slovník – do této kategorie spadá slovník obsahující běžné zkratky,
2. slovník entit,
3. pomocný slovník.

3.2.6 Problémy spojené s NER

S rozlišováním pojmenovaných entit jsou spojeny další problémy, které nemusí přímo souviset s nalezením pozice entity v textu.

Identifikujeme-li v předloženém textu osoby za účelem dalšího zpracování, pro příklad budeme chtít sestavit seznam všech entit a (počet) míst jejich výskytu. Ve vytvořené databázi nás bude zajímat současný český prezident. Nyní ale nastává situace, kdy bychom v rejstříku nalezli samostatný záznam pro entitu „Zeman“, „Miloš Zeman“, „(prezident) České republiky“ atd. Pokud ale potřebujeme získat všechny relevantní výskyty, musíme tyto výskyty rozumně spojit.

Narážíme na dva problémy. Zaprvé musíme identifikovat, že tyto entity ukazují na stejnou osobu. Naivní přístup by mohl být například slovník syno-

nym. Tím se ale dostáváme k druhému problému. Provést množinový součet by bylo unáhlené. Slovo „prezident“ zcela zřejmě nemusí být pouze referencí na Miloše Zemana. Ve starších textech bude poukazovat na Václava Klause či v jiném kontextu na libovolného prezidenta jakékoliv země či dokonce společnosti.

Jiným často řešeným problémem bývá *disambiguace* slov či frází, se kterým se potýkají ve velké míře webové vyhledávače [8]. *Odstranění dvojsmyslů*, jak název napovídá, přináší řešení *mnohoznačnosti* (*polysémie*) slov, neboli nalezení skutečného významu n-gramů či samostatných slov. Příklad mnohoznačného slova budiž výraz „Kabát“, který stejně tak dobře může symbolizovat součást oděvu, což není pojmenovaná entita, jako české hudební těleso či příjmení, což jsou, v závislosti na používaných třídách, dvě odlišné třídy (**organizace/kapela** a **osoba**). Pro rozpoznání původního významu je tak zapotřebí brát v potaz kontext a další sémantické indikátory.

V případě, že kontext postrádáme, je úloha jen obtížně řešitelná. v případě webového vyhledávače tak můžeme použít naivní statistický přístup, kdy množinu odpovědí vytvoříme na základě počtu výsledků pro jednotlivé významy či na základě četnosti prohlížení jednotlivých hesel. Vyhledáváme-li tak například heslo „Java“, budeme spíše preferovat stránky zabývající se populárním a často vyhledávaným programovacím jazykem než stránky zabývající se indonéským ostrovem. Přesnějších výsledků můžeme dosáhnout tak, že budeme (dlouhodobě) sledovat dotazy pokládané uživatelem a na základě statistiky jeho chování a historie upřesníme naše odpovědi.

3.3 Předzpracování

3.3.1 Tokenizace

Text je pro počítač pouhá sekvence znaků, proto je zapotřebí text nejprve předzpracovat. Prvním krokem bývá obvykle segmentace textu na lingvistické jednotky, čili aplikace lexikální analýzy za účelem nalezení nejmenších jednotek textu (*tokenů*). Jednotlivé tokeny jsou nazývány *pozice*. Ve většině případů se jedná o *grafické slovo*, kde oddělovačem slov založených na latince bývají především bílé znaky. Úkol je však netriviální pro jazyky typu *scriptio continua*, které nepoužívají žádné zjevné dělení slov, mezi které patří například čínština, ale i stará řečtina atd. Ve většině případů se od slov odděluje

i interpunkce a považuje se za samostatný token. z textu se mohou odstraňovat i tzv. *stopslova*, často se aplikuje i *case folding*, čili převedení na malá písmena.

Lepší algoritmy dokáží rozpoznat i víceslovné či složené *termy* (zkratky jako a.s., emotikony., URI, víceslovné názvy, stažené tvary...). Pokud je cílem mít tokeny jednoslovné, označuje se algoritmus jako *Low-Level tokenizer*, naopak pokud chceme hledat i víceslovné tokeny (např. `Samsung Galaxy S7`), hovoříme o *High-Level* tokenizaci, která je lingvisticky mnohem náročnější.

Výsledek tokenizace se dále používá jako vstup pro další metody NLP. Proto je jeho kvalita velmi důležitá, neboť případná chyba by se přenášela systémem dál a ovlivnila by všechny navazující algoritmy.

Univerzálnost Oproti ostatním úlohám z oblasti zpracování přirozeného jazyka se tokenizace považuje často za úlohu jednoduchou a nezajímavou. Vzhledem k tomu, že je však, jak bylo řečeno, velmi důležitá pro navazující algoritmy, vznikla celá řada metod, které mají za úkol zlepšit řešení obecně i pro konkrétní potřeby. Pro různé úlohy je často zapotřebí různý přístup - pro knižní jazyk bude zajisté zapotřebí jiný než například pro příspěvky chatovací místnosti obsahující překlepy, špatně vkládanou interpunkci, neočekávané znaky apod. Tokenizer je tak potřeba tvořit s přihlédnutím na zdrojová data.

3.3.2 Stemming

Definice pojmů Lexém se označuje jedna položka ve slovníku – všechny tvary slova sdílí jeden lexém. Lemma označuje základní reprezentativní podobu (alolex) lexému, která se uvádí ve slovnících, odtud též slovníkový tvar. Konkrétní instanci (realizaci) slova nazýváme token.

Stemming je termín často používaný v oblasti jazykové morfologie. Přestavuje proces úpravy tvaru slova vzniklého ohýbáním či odvozením, konkrétněji hledání kmene (stemu) slova, který často bývá chybně zaměňován za kořen, se kterým se obecně nemusí, ale může, shodovat.

V českém jazyce se tato úprava hojně využívá k předzpracování textu v oblasti získávání informací, typicky například pro webové vyhledávače. Při této aplikaci však může mít i negativní dopad na výsledky z důvodu rozšíření

množiny hledaných slov.

Cílem tohoto snažení je zlepšit proces získávání informací tím, že příbuzná slova převedeme na stejný tvar, čímž zredukujeme variabilitu jazyka za zachování informací, které text obsahoval.

Všechny jazyky, s výjimkou Esperanta, jsou neregulární a zatímco více formální slova se řídí stanovenými pravidly, většina běžně užívaných slov vzniká aplikováním nepravidelného skloňování a časování. Existují tak algoritmy, za nimiž se skrývá dlouhý jazykovědný výzkum, které poskytují velmi kvalitní výsledky, a proti nim stojí často velmi jednoduché systémy, které si poradí pouze s nejčastějšími a nejběžnějšími případy.

Smysl má stemming především pro jazyky z Indo-Evropské skupiny, jelikož z principu není aplikovatelná pro čínštinu apod. Své využití nachází především pro ty jazyky, kde slova vznikají flexí či obecně gramatickými pravidly, typicky užitím afixu – tzn. předpony (prefixu) a přípony (sufixu).

Afixy Přípony jsou typicky rozlišovány tři základních typů: A-přípony (z anglického attached): slovo připojené k jinému slovu. Typickým představitelem je portugalština, která tato slova připojuje přes pomlčku, čímž je poměrně snadné je odstranit. Složitější situace je ve španělštině či italštině, kde se slova spojují bez oddělovače, například v italštině se takto pojí sloveso se zájmenem.

I-přípony (inflectional): upravuje slova na základě základních pravidel jazyka, za existence některých výjimek. Typickým příkladem je tvorba minulého času v angličtině užitím přípony -ed (fixed, added). Typická je i tvorba množného čísla, přivlastňovací xyz's, či sloveso ve třetí osobě čísla jednotného (feels).

D-přípona (derivational): z jednoho slova vytváří jiné, které může být i jiného slovního druhu (readable, washable).

Přístupy a algoritmy

Pro představu se podívejme na základní algoritmy, na kterých staví systémy pro stemming a lemmatizaci:

Algoritmus hrubé síly Jedná se o slovníkovou metodu využívající tabulku dvojic vyskloňovaný tvar slova – kořen slova. Pro každé slovo je tak prohledávána tato mapa (tabulka), z čehož plyne i největší slabina: jedno slovo (např. „července“) může být odvozeno od více základů, tedy „červenec“, „červenka“, které by měly mít v takové mapě dva různé záznamy. Hledání stemů za užití slovníků je vhodné u jazyků, u kterých je častá nepravidelná flexe, například v angličtině: mouse, mice, foot, feet.

Suffix stripping algoritmy Tato metoda je řízena pravidly a nepotřebuje tak žádný seznam slov/slovník. Vývoj a optimalizace takových pravidel je mnohem jednodušší než sestavování tabulky pro algoritmus hrubé síly, ovšem je zapotřebí dostatečná odborná znalost daného jazyka. Problémem jsou nepravidelnosti a výjimky v jazyce, které užitím této metody nelze vyřešit. Tento typ algoritmů se tedy nasazuje pouze na takovém jazyce, či na takové jeho podmnožině, pro který jsou pravidla jasně daná s velmi malou množinou výjimek. Hranice mezi slovníkovými a algoritmickými systémy je nicméně někdy tenká: algoritmický stemmer může využívat seznamy výjimek, které lze považovat za účinné slovníky.

Stochastické algoritmy Posledním zástupcem je rodina stochastických (statistických) metod, které jsou založeny na metodách strojového učení, obvykle bez učitele, jehož užitím jsou odhadnuty parametry stematizačního modelu. Pro nalezení stemu je tak využíván pravděpodobnostní model, odtud název skupiny, který obvykle představuje množinu pravidel podobných těm, se kterými pracuje předchozí zmíněný algoritmus. Nejpravděpodobnější stem předkládaného slova pak nalezne model na základě naučených pravidel. Jejich výhodou je absence nutnosti spolupráce s jazykovým expertem, tvorby slovníku či to, že umožňují snadnou migraci pro jiné jazyky.

Výběr vhodnějšího přístupu je netriviální a měl by se provádět vždy s přihlédnutím k dané úloze. Obecně lze však říci, že slovníkový systém bude vždy jen tak dobrý, jak dobrý je jeho základní kámen, tedy slovník. Pokud narazí na cizí (nové) slovo, je bezradný a selže. Je tak vždy zapotřebí velmi rozsáhlý, dobře sestavený a aktuální slovník. Pokud tak pro daný jazyk existuje algoritmický stemmer, je doporučeníhodné mu dát přednost kvůli rychlosti, paměťové nenáročnosti a větší přizpůsobivosti. Pokud však je nejdůležitější přesnost výsledků a výše zmíněné nevýhody nejsou oproti tomuto požadavku významné, častěji se volí slovníkový přístup. Statistický přístup má již výše zmíněnou výhodu: není zapotřebí vytvoření obrovského slovníku a není za-

potřebí spolupráce s jazykovým expertem.

Chyby systému U stemmeru rozlišujeme tyto dvě chybové metriky: nadměrným stemmingem (over-stemming) a nedostatečným stemmingem (understemming). První zmíněný typ chyby znamená přehnaně agresivní ořezávání (zkracování) slov, které má za následek přiřazení příliš mnoha stemů jednomu lexému, tedy chyba typu *false positive*. Opačný problém nastává u nedostatečné stemmatizace, kdy existují slova, která by měla být přiřazena ke stejnému lexému, ale nejsou, tzv.chyba typu *false negative*. Snahou stemmerů by mělo být minimalizovat obě tyto chyby, což je netriviální úkol, neboť zmenšením jedné chyby se obvykle zvětší druhá z nich.

Na základě těchto chyb se rozlišují stemmery na *agresivní*, které slova zkracují více, a tzv. *light*, které zachovávají více srozumitelné podoby slov, v některých případech je lepší ponechat slovo v nezměněné podobě než vytvořit příliš krátký stem.

Příklady systémů

Standardem v oblasti algoritmických systémů pro zpracování Evropských jazyků je jazyk Snowball, který umožňuje snadnou implementaci pravidly řízených systémů, přesto, že se jeho použitím často přichází o výkon, který by daný algoritmus poskytoval při samostatné ruční implementaci, například porter_stem token filter je výrazně rychlejší, než implementace užitím Snowball [Elasticsearch: TDG]). Přesto je často používaným prostředkem, ne-li k produkčním stemmerům, tak alespoň k ověření jejich návrhu (proof-of-concept) a ladění.

Na druhé straně stojí systém Hunspell, jehož jádrem je slovník, ale který umožňuje s výhodou využít i doplnění o pravidlový přístup. Dnes je prakticky nemožné se s ním nesetkat, protože kromě stemmeru poskytuje veškerou funkcionalitu potřebnou pro kontrolu pravopisu a lze jej tak nalézt v kancelářském balíku LibreOffice, nejpoužívanějších webových prohlížečích či v Eclipse IDE.

3.3.3 Lemmatizace

Úkol velmi podobný tomu, jaký jsme uvedli u stemmingu, má i lemmatizace. Obě metody mají za úkol zmenšit prostor slov pro danou úloh, tedy minimalizovat slovník, a z toho důvodu jsou v některých případech zaměnitelné. Hlavním rozdílem je, že výstupem lemmatizace jsou jazykově korektní slova, tj. slova, která v daném jazyce skutečně existují, z čehož jasně plyne, že v případě potřeby zachování smysluplných slov na výstupu je úloha lemmatizace nezastupitelná.

Je též zřejmé, že na rozdíl od stemmatizace je nemožné provést trénink metodou učení bez učitele. Pro tvorbu systému je tak vždy zapotřebí ručně anotovaný korpus či exportem vytvořená sada pravidel.

4 Vstupní data

Aby mělo vůbec smysl se algoritmy zpracování přirozeného jazyka, potažmo vyhledáváním pojmenovaných entit, zabývat, potřebujeme data, na která budeme tyto metody aplikovat. Již bylo řečeno, že od konkrétní podoby dat se odvíjí úspěšnost NLP algoritmů a tím i potřeba jejich modifikací. Mimo to je často zapotřebí provést, v závislosti na podobě vstupu jeho předzpracování – např. vykonat extrakci ze složitějších konstrukcí (JSON, XML, ...), či použít metod k odstranění nežádoucích znaků apod.

Charakter zpracovávaného textu může být různorodý. Může se jednat o dlouhé, souvislé knižní texty psané spisovným jazykem nebo například o krátké komentáře ze sociálních sítí, které jsou psané nespisovným jazykem, obsahují překlepy, nestandardní sekvence znaků, emotikony atd.

4.1 Původ a charakteristika dat

Data, která jsou vstupem systému vytvářeného v rámci této práce, jsou dvojího typu:

1. komentáře ze sociálních sítí – příspěvky z fóra zabývajícího se mobilními technologiemi, psány volnou formou, s překlepy, nespisovně, obsahují emotikony a URL odkazy, navíc mohou, i nemusejí, obsahovat diakritiku,
2. korpus CNEC – texty jsou z větší části psány spisovnou češtinou, dokumenty, rozuměno články, jsou delší a ucelenější než je tomu v případě komentářů.

Tak, jako u všech podobných systémů, je i pro ten, který je výstupem této práce, důležité mít možnost jej otestovat na standardně používaných datech, používaných pro evaluaci systémů s podobným zaměřením a zároveň je pak možné systém porovnat s již existujícími řešeními.

Proto byl použit korpus pojmenovaných entit Czech Named Entity Corpus (CNEC, 2), který je standardem pro testování úspěšnosti NER systémů. Použitý trénovací korpus obsahuje 7144 vět a 15185 pojmenovaných entit,

které byly ručně označeny. Korpus určený pro testování pak obsahuje 890 vět a 2035 pojmenovaných entit.

Data jsou již v předzpracované podobě, kdy v něm obsažené dokumenty byly tokenizovány a lemmatizovány, čímž tak byl prostý text obohacen o další informace. Pro tuto práci jsou však zjevně nejpodstatnější označené pojmenované entity. Při anotaci byla použita hierarchie typů pojmenovaných entit, viz příloha B. Používaná data jsou označkována pouze za využití 1. úrovně hierarchie. Konkrétněji se v nich vyskytují entity typu:

- *číslo v adrese* – obvykle poštovní směrovací číslo či číslo domu,
- *geografický název* – názvy měst, kontinentů apod.,
- *instituce* – názvy organizací, institucí apod.,
- *název media* – například URL, televizní a radiové stanice,
- *číselný výraz* – pořadová čísla, částky, skóre zápasu,
- *názvy a pojmy* – produkty, měny, jednotky, knihy, filmy,
- *osobní jména* – názvy osob (jméno, příjmení) a zvířat, tituly atd.,
- *čas a datum* – čas, den, měsíc, rok, datum, svátek.

Podobu dat ilustrujme na následujícím příkladu. Lze si povšimnout, že každý řádek obsahuje jeden token:

```
Vpravo vpravo Db-------------O
nahoře nahoře Db-------------O
: : Z:-------------O
José José_;G_;Y NNMS1-----A----B-P
Raoúl Raoúl X@-------------I-P
Capablanca Capablanca X@-------------I-P
,,Z:-------------O
tento tento PDYS1----------O
kosmopolita kosmopolita NNMS1-----A----O
z z-1 RR--2----------O
Kuby Kuba-2_;G NNFS2-----A----B-G
```

Dalším zdrojem dat jsou příspěvky pocházejí z internetového fóra zabývajícího se mobilními zařízeními, telefonními operátory a jejich tarify atd. Lze tedy očekávat, že pojmenované entity v těchto datech budou především označení produktů, jako tarifů, mobilních telefonů apod., a organizací. Očekávat lze též data, čísla, například ceny, či jména uživatelů fóra.

Tato data mají podobu pole JSON objektů pro každý komentář. Tento JSON, kromě samotného textu příspěvku, obsahuje metadata jako jméno autora, čas publikování příspěvku, jeho URL a zařazení do hierarchie zdrojového fóra. Pro ilustraci podoby dat vkládáme jeden takovýto JSON objekt.

```
{
  "entities" : ["O2", "T-Mobile", "Vodafone",
                "Telecommunications"],
  "authorName" : "radim",
  "id" : "MobilMania-141201000001CET-0",
  "time" : "2015-12-04 12:27:00 CET",
  "text" : "Vodafone nabízí v˜retenci vylepšený Odepiš:
            50 minut do sítě, neomezené SMS do sítě, 20 MB
            dat, za 15 Kč. Data se nezastaví, \"jen\"
            zpomalí. 3,50 Kč/min., 1,50 Kč/SMS. Na účtování
            jsem se neptal, ale tipuju odepišovské 60+1.",
  "type" : "comment",
  "url" : "http://forum.mobilmania.cz/viewtopic.php
           ?f=1&t=1119189&start=3390",
  "forumPath" : "O2, T-Mobile, Vodafone, U:fon a˜další v˜ČR
                 [->] Retenční nabídky Vodafone (vše kromě
                 Tarifomatu)"
}
```

Zdrojem těchto dat je zadavatel práce.

4.1.1 Porovnání zdrojů vstupních dat

Již bylo nastíněno, že texty z těchto dvou zdrojů jsou rozdílné a každý z nich má svá specifika, která budou ovlivňovat výstup systému. Je proto zapotřebí se s vlastnostmi dat seznámit, aby bylo možno na ně nastavením systému reagovat a zvýšit tak jeho kvalitu.

Je běžné kvalitu systému hodnotit a porovnávat na standardních korpusech, aby byly zaručeny stejné podmínky. Implementace tak bude nejprve testována na CNEC korpusu, bude pro tento systém optimalizována a teprve poté proběhnout pokusy s daty, která nás nejvíce zajímají, pocházejících ze sociálních medií.

Rozdílným nastavením systému je v tomto případě myšleno především využití jiných, případně jinak nastavených extrahovaných příznaků. Pomiňme skutečnost, že vzhledem k formátu vstupu bude zapotřebí dvou rozdílných parserů či dodatečné úpravy dat, nehledě na skutečnost, že bude zapotřebí opatřit anotacemi nepředzpracované texty.

Pro systém jsou data 1 oproti 2 obtížněji zpracovatelná, vzhledem k vyšší četnosti překlepů a chybějícím bílým znakům, které způsobí chyby při tokenizaci. Zároveň jsou mezi uživateli, autory komentářů, populární novotvary, vlastní autorská inovace typu vlastních aliasů pro organizace a výrobky, například „Micro$oft“, „Mrkvosoft“ atd., či například nerozlišování malých a velkých písmen, což úlohu NER značně znesnadňuje.

4.1.2 Anotace dat

Přestože již současné systémy dosahují vysoké míry úspěšnosti, má před sebou NER ještě dlouhou cestu k dokonalosti. Velkým problémem je například nutnost anotace velkého trénovacího vzorku dat, využitelnost napříč jazyky a doménami atd. Ruční příprava dat je drahá a zdlouhavá a proto se ji snaží mnoho společností přenechat na jiných tak, aby náklady na ni byly co možná nejmenší. Trendem je delegovat práci především na uživatele, kteří se o vytvoření trénovacích dat mohou postarat sami, dobrovolně a zadarmo. Pravděpodobnost úspěchu je vyšší, pokud navíc mohou danou činnost pokládat za službu či výhodu pro ně samotné. Pro tento způsob se vžil termín *crowdsourcing*. Lze na něj narazit nejen v oblasti NLP, ale například i v internetových galeriích, kde uživatelé označují obličeje a připravují tak trénovací data pro automatické označování na fotografiích nahrávaných na sociální sítě. Dále jej lze využít při automatických překladech, kontrolách pravopisu či ve slovnících používaných k predikci slov u klávesnic na mobilních zařízeních, kde uživatel svými návrhy lepších alternativ zlepšuje kvalitu systému rozšiřováním množiny dat použitelných k trénování predikce. Pro aplikace NER mohou být zajímavé například příspěvky sdílené na sociálních sítích, v nichž samotný autor příspěvku označí osoby či firmy a kromě jednoduchého vyzna-

čení pojmenované entity poskytne propojení či referenci na konkrétní entitu. Texty postrádající vyznačení pojmenovaných entit je nutno ručně zpracovat a tyto značky do nich doplnit.

Před zahájením anotování je nutno stanovit pravidla, jakými se bude řídit, a zároveň vytvořit či vybrat nástroj k tomuto účelu sloužící. Vedoucím práce byl vybrán nástroj realizovaný jako jím zadaná bakalářská práce, který je schopen tuto roli zastat, usnadnit a urychlit ruční značkování dat.

Vzhledem k tomu, že tento nástroj na vstupu vyžaduje množinu textových souborů, jsou tyto připraveny užitím jednoduchých Groovy skriptů. Pro zjednodušení případné další práce s daty či touto prací jsou všechny předzpracující skripty přiloženy. První skript (viz C) připravuje z originálních JSON objektů prosté textové soubory, kde každý řádek odpovídá textu jednoho komentáře. Vzhledem k možnosti dalšího použití, snazší orientaci v datech, a tím i zrychlenému anotování, byly tyto soubory dále upraveny tak, aby se každý dokument nacházel v samostatném souboru, příslušný skript viz D. Takto připravené texty jsou již použity jako vstup anotačního nástroje.

Pro jednotný postup při anotaci, je nutné definovat typy pojmenovaných entit, jejich hierarchii a způsob, jak identifikovat jejich příslušníky. Protože je práce zaměřena na vyhledávání značek a produktů v textu, je logické zavedení typů *produkt* a *organizace*. v úvahu též připadají standardně používané entity typu *osoba*, *geografický pojem*, *číslo* atd. Před další rozvahou je nutné poznamenat, že anotační nástroj umožňuje v textu vyznačovat až dvě různé třídy pojmenované entity pro jeden a ten samý úsek dokumentu, čehož lze s výhodou využít tam, kde je těžké rozhodnout o skutečném zařazení entity („Kalašnikov“ – osoba i produkt) či v případech typu „Samsung Galaxy S6“, kde lze všechny tři tokeny označit za součást názvu produktu, ale zároveň lze vyznačit název organizace. Pokud bychom v tomto případě označili celý text jako název produktu, ztratíme informaci o výskytu entity výrobce. v opačném případě označíme pouze část oficiálního názvu.

Situace se více komplikuje, pokud bychom měli rozlišovat další třídy, jako čísla či geografické názvy, například „Google Android 6.0“ či hůře „Aston Martin One 77“, kde se setkávají v rámci jednoho názvu entity tří typů. z tohoto důvodu bylo rozhodnuto o omezení počtu rozlišovaných tříd na 2, a to „produkt“ a „organizace“. Přestože prvotní implementace ověřující kvalitu systému je optimalizována pro rozpoznávání standardních tříd, bylo vzhledem k zaměření práce, kterézto cílem je co nejlépe vyhledat v textu produkty a značky, takovéto rozhodnutí ospravedlnitelné, ba dokonce vhodné, neboť

lze extrahované příznaky více specializovat.

Po definici klasifikačních tříd je dále zapotřebí popsat podobu jejich zástupců, pokud možno tak, aby nebylo sporu o jejich příslušnosti a aby bylo zaručeno, že rozdíl mezi anotacemi dvou různých anotátorů bude minimální, pokud možno žádný.

Používané třídy pojmenovaných entit

Za entity typu produkt považujme všechna označení produktů, kterými jsou mobilní telefony, tarify mobilních operátorů, finanční produkty atd. Za součást názvu vždy označíme celý název společně s názvem organizace, například „Samsung Galaxy Ace II“. v tomto případě navíc opatříme anotací *organizace* i slovo „Samsung“.

Za organizace budeme považovat názvy firem a společností. Sporné mohou být názvy webových portálů či webových služeb, např. „Koupil jsem si ABC na XYZ.cz“. Název portálu v tomto případě označíme oběma anotacemi, obdobně označíme i entitu v případě, kdy text obsahuje referenci na sociální síť.

V případě, kdy je pojmenovaná entita součástí URL, neoznačujeme ji. Výjimkou budiž případy, kdy se jedná o název domény („XYZ.cz“), kdy vyznačíme pro daný úsek texty obě třídy.

Výstup anotačního nástroje

Výstupem nástroje je pro každý dokument jeden soubor ve formátu XML, který obsahuje seznam anotací provedených v daném souboru:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<AnnotationLayer name="NE">
  <Annotation>
    <Start>76</Start>
    <End>83</End>
    <Type>product</Type>
  </Annotation>
  <Annotation>
    <Start>134</Start>
    <End>141</End>
    <Type>product</Type>
  </Annotation>
</AnnotationLayer>
```

Lze si povšimnout, že soubor obsahuje jednoduchý list anotací, kde každá je definovaná spanem, tedy začátkem a koncem nalezené pojmenované entity, a typem.

5 Realizace

Cílem této práce je vytvoření NER aplikace pro použití na sociálních sítích. Vzhledem k tomu, že algoritmy strojového učení, popsané v 3, které jsou jejím základem, jsou implementačně netriviální a navíc již existují jiné, mnoha uživateli ověřené, knihovny, které je v optimalizované podobě poskytují, je na místě použít některou z těchto hotových implementací.

Výběr knihovny je nutné provést ještě před zahájením implementace, neboť zásadně ovlivňuje podobu aplikace. Vedoucím práce byla doporučena knihovna *Brainy*. Vzhledem k tomu, že je tato implementována v jazyce Java, byl z důvodů snadného použití, a zachování rychlosti, pro vlastní tvorbu vybrán právě tento programovací jazyk, konkrétněji ve verzi 1.8.

Vzhledem k zásadní roli používané knihovny následuje její krátký popis, který ozřejmuje podobu dalších částí aplikace.

5.1 Brainy

Knihovna Brainy [19] je určená pro strojové učení, distribuovaná pod licencí *Apache 2* a je vyvíjena na Západočeské univerzitě v Plzni skupinou *Natural Language Processing Group*. Následující popis je založen na dokumentaci k nástroji dostupné na webových stránkách (`http://home.zcu.cz/~konkol/Brainy/documentation.pdf`, 14.6.2016).

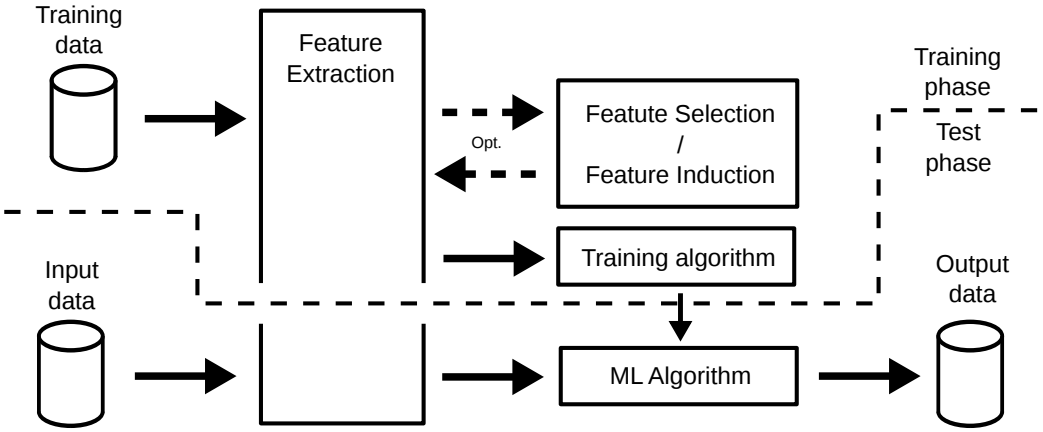
Sestává ze tří hlavních komponent:

1. primární komponenta určená pro strojové učení poskytuje rozhraní pro základní úlohy jako je klasifikace či shlukování,
2. komponenta poskytující matematický aparát a aparát lineární algebry. Sestává ze dvou hlavních částí: implementace struktur a algoritmů lineární algebry a optimalizačních algoritmů,
3. komponenta pro zpracování příznaků zodpovídající za správné vytvoření matic a vektorů z objektů dodaných uživatelem.

Činnost knihovny výmluvně popisuje obrázek 5.1, na kterém je znázorněna fáze trénování, pracující s trénovacími daty, i fáze testovací, využívající data testovací a validační. Úkol této práce se nachází na levé půlce schématu. Je jím příprava vstupů a extrakce příznaků ze vstupních dat, čili transformace vstupních objektů na matice a vektory, se kterými pracuje matematický aparát knihovny. Tomuto účelu slouží definovaná rozhraní umožnující implementaci vlastních příznaků, v podobě programové či XML definice.

Dalším krokem je zpracování vytvořených struktur a učení systému na jejich základě. v našem případě se jedná zjevně o učení s učitelem. Výstupem trénovacího procesu je klasifikátor, který je dále použitelný pro testovací data.

Knihovna nabízí několik algoritmů, z nichž některé byly popsány výše (3), pro natrénování klasifikátoru. Pro potřeby práce použijeme *CRFTrainer*. Naší implementací se totiž pokusíme zreplikovat výsledky experimentů popsaných ve článku [21], které stavějí právě na algoritmu CRF a využívají knihovnu Brainy. Dosažení podobných výsledků na korpusu CNEC tak bude bráno jako důkaz správnosti implementace systému.



Obrázek 5.1: Schéma činnosti knihovny *Brainy*, Zdroj: `http://home.zcu.cz/~konkol/Brainy/documentation.pdf`

5.2 Struktura aplikace

Před dalším detailním popisem jednotlivých částí implementace následuje popis balíků, ze kterých aplikace sestává. Pro přehled implementovaných tříd včetně jejich vzájemných vztahů je přiložen UML diagram (příloha F). Popis

jednotlivých tříd a jejich funkcionality je dostatečně popsán v rámci JavaDoc.

Všechny dále jmenované balíky jsou součástí balíku `cz.zcu.kiv.witz.dp`:

- `features` – tento balík obsahuje implementace všech použitých příznaků
  - `features.settings` – v tomto balíku nalezneme programové nastavení množiny příznaků pro oba zdroje dat,
- `structs` – balík zahrnuje třídy, které slouží jako reprezentace entit a struktur objevujících se napříč aplikací, jako je například `Token`, `Word`, `Ngram` či `NamedEntity`
  - `structs.wordTree` – obsahuje implementaci stromové struktury, která umožňuje rychlé vyhledávání textových řetězců. Tato struktura je schopná vyhledávat nejen za užití celých slov, ale i aplikací regulárních výrazů,
- `util` – zde lze nalézt třídu umožňující jednoduchou serializaci tříd (což je obzvláště výhodné v momentě, kdy využíváme již jednou natrénovaný klasifikátor opakovaně pro vícero běhů aplikace), `Tokenizer`, třídu pro získání ortografické podoby řetězce či třídu `Pipe`, která umožňuje paralelní zpracování velkých kolekcí dat, nad kterými na jeden průchod provede všechny zadané operace, což se obzvláště hodí při předzpracování vstupních dat
  - `util.fileUtils` – do tohoto balíku byly umístěny třídy pro práci se soubory, jako je načítání vstupních dat,
  - `util.statistics` – třídy v tomto balíku umožňují provádět evaluaci výstupu systému. Obsahují metody pro porovnání očekávaného a skutečného výstupu a pro výpočet úspěšnosti systému.

5.3 Příprava dat

Aby bylo možno Brainy použít ve smyslu této práce, je zapotřebí zajistit následující 4 vstupy:

- klasifikované objekty,

- jejich klasifikaci vytvořenou učitelem,
- začátky sekvencí, reprezentující začátky jednotlivých dokumentů obsažených v datech,
- z nich extrahované příznaky.

V našem případě klasifikovanými objekty rozumíme jednotlivé tokeny vstupního textu. Jejich klasifikací pak samozřejmě příslušnost k té či oné třídě pojmenovaných entit. Ve vztahu k dříve představeným vstupním datům nás tak v případě korpusu CNEC zajímá první (token) a poslední (třída) sloupec dat. Je zapotřebí vytvořit parser, který data převede do dvou stejně dlouhých vektorů obsahujících tyto informace a navíc do vektoru obsahujícího indexy začátků dokumentů.

Aby bylo umožněno co největší zjednodušení parseru, který by byl použitelný jak pro CNEC korpus, tak pro anotovaná data, je vhodné spojit dříve demonstrovaný výstup anotačního nástroje, a data samotná, do formátu blízkého podobě CNEC korpusu. Tento převod se provádí opět za užití triviálního Groovy skriptu, viz příloha E.

5.3.1 Tokenizace

Jednou z fází přípravy vstupních dat je rozdělení na tokeny, neboli tokenizace. Jak bylo možno si povšimnout ve výstupu z anotačního nástroje, jsou hranice anotací dány znaky, nikoliv slovy či tokeny. Je proto zapotřebí tokeny vyhledat a doplnit je o příslušné ručně nalezené třídy.

Pro tokenizaci bylo použito vlastní řešení, které je založeno na parsování textu regulárním výrazem. Základem je dělení textu podle bílých či speciálních znaků, jako je pomlčka, tečka apod., doplněné o možnost přidat výjimky. Tou by mohly být např. některé ustálené zkratky, které není žádoucí dělit, např. „s.r.o.“ atd.

Podoba regulárního výrazu, a tím i počet a podoba výsledných tokenů, výrazně ovlivňuje nejen způsob anotace, ale i navazující zpracování. v komentářích obsažených v textu často dochází k vynechávání či přemíře mezer mezi slovy a interpunkcí. v případě, kdy je mezi dvěma slovy pouze tečka bez mezery, může se jednat o část URL, které nebyly anotovány, ale i o chybu textu.

5.3.2 Stemming

Součástí předzpracování dat jsou i operace, které by při extrakci příznaků byly prováděny opakovaně a je tedy vhodné je z důvodu úspory výpočetního času provést jednorázově předem. Mezi takové operace patří hledání kmene (stemu) slova. Vzhledem ke skutečnosti, že je stemming netriviální úloha, která je však již dobře řešena existujícími knihovnami, použijeme jednu z nich.

High precision stemmer

Knihovna *High precision stemmer* (HPS) [20] je nástroj pro multijazyčný stemming založený na učení bez učitele, tedy bez využití anotovaného korpusu. Je vyvíjen touž skupinou jako knihovna Brainy a je šířený pod licencí *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License*.

Zevrubný popis nástroje HPS není předmětem této práce, pro detailní informace viz [20]. Pro potřeby práce postačí základní popis. Nástroj staví na myšlence, že stemming musí zachovat sémantickou informaci a odstranit informaci morfosyntaktickou. Toho je dosaženo jednak prostřednictvím určování významu na základě kontextu slova a vytváření shluků slov nesoucích podobný sémantický význam a jednak využitím klasifikátoru.

5.3.3 Rozšiřující informace o tokenech

Kromě kmenu slova, resp. tokenu, můžeme s ohledem na extrakci příznaků přidat při předzpracování další informace. Jednou z nich je přítomnost slova či jeho lemmatu/kmene apod. ve slovníku či shluku.

Předešleme, že jeden z implementovaných příznaků umožňuje použití právě takovýchto rozšiřujících informací navěšených na token. Toho je využito v implementaci pro vyhledání slova v množině slovníků, kde každý z nich obsahuje slova příslušící právě jedné klasifikační třídě. Jedná se kupříkladu o slovníky měst, společností, zemí, měn, hor, akademických titulů, jmen a příjmení, univerzit atd. Při předzpracování je každé slovo z textu hledáno v těchto slovnících. v případě, že je nalezeno, je přidána informace o slovníku, který jej obsahoval.

5.4 Extrakce příznaků

Podle výše uvedeného schéma popisujícího fungování knihovny Brainy a jeho popisu je zřejmé, že z připravených trénovacích i testovacích dat, která byla prostřednictvím parseru převedena na vektory obsahující jednotlivé tokeny (jednotlivá slova), jejich klasifikaci a začátky jednotlivých sekvencí (dokumentů), je dále potřeba vyextrahovat příznaky, se kterými bude knihovna dále pracovat. Volba příznaků je velmi důležitý krok a je potřeba ji provádět s rozvahou. Netriviální je i jejich vhodné zkombinování.

Dalším faktorem, resp. další proměnnou, je u některých příznaků *kontext*. Bude-li brán v potaz, nabízí se otázka, na jak vzdálená slova se zaměřit a zda s nimi bude nakládáno stejně, jako se slovem, které je v současné chvíli zpracováváno.

5.4.1 Rozhraní `Feature`

Pro lepší představu o implementační podobě příznaků následuje rozhraní, které musejí tyto třídy implementovat.

```
public interface Feature<T> extends Serializable {
  int getNumberOfFeatures();

  void extractFeature(ListIterator<T> var1,
     FeatueVectorGenerator var2);

  void train(InstanceList<T> var1);
}
```

V popisu rozhraní si lze povšimnout, že implementující třída musí poskytovat informace o délce extrahovaného vektoru (počtu příznaků), a že dobře reflektuje výše uvedené schéma 5.1. v trénovací fázi je pro každý vytvořený příznak volána metoda `train(InstanceList<T>)`, které jsou jako parametry předány zpracovávané instance rozpoznávaných objektů. Metoda `extractFeature` je posléze volána v obou fázích, trénovací i testovací. Jak název napovídá, slouží k extrakci příslušného příznaků a nastavení hodnot generovaného vektoru.

5.4.2 Použité příznaky

Již několikrát bylo zdůrazněno, jak je výběr příznaků důležitý. Následuje výčet těch, které byly vybrány, implementovány a použity k experimentům na popsaných datech. Jejich výběr byl ovlivněn experimentem popsaným ve článku [21].

Ačkoliv Brainy umožňuje prvky vektorů generovaných příznaky nastavovat na desetinná čísla mezi 0.0 a 1.0, všechny implementované příznaky využívají pouze dvou hodnot: 1 a 0.

Ortografická podoba slova

Při hledání způsobů, jak identifikovat pojmenovanou entitu, bude pravděpodobně jedna z prvních myšlenek směřovat k ortografické (*ortografie* – též *pravopis*, je ustálený způsob záznamu zvukové podoby spisovného jazyka systémem grafických znaků) podobě slova. Například osobní jména a geografické názvy začínají velkým písmenem. Některé organizace a produkty mají jinde neobvyklé podoby, jako velká písmena jinde než na první pozici, číslovky uvnitř slov atd.

Nejdůležitější pro tento příznak je samozřejmě volba seznamu vzorů. Aby bylo možno slova s těmito vzory či pravidly porovnávat, je možno sáhnout po regulárních výrazech, které umožňují vytvářet pravidla s vysokou míru složitosti. Druhou možností je transformace slov podle předem daného postupu do zjednodušující podoby a následné porovnání s definovanými vzory. Při implementaci byl použit právě tento způsob.

Postup transformace je jednoduchý: všechny nepřerušené sekvence znaků s touže vlastností jsou nahrazeny jedním zástupným znakem. Totožnou vlastností je míněno především velké/malé písmeno či číslo. Pro ukázku řetězec „Slovo“ nabude podoby „Aa“, řetězec „iPhone“ podoby „aAa“ a obdobně „Servis24“ bude transformován na „Aa1“.

Vektor generovaný tímto příznakem má délku danou počtem definovaných vzorů, kterým je možno slova přiřadit. Na hodnotu 1 je nastaven ten prvek vektoru, který odpovídá vzoru popisujícím dané slovo. Ač se tento příznak může zdát být primitivní, při experimentech dokázal úspěšnost systému velmi pozitivně ovlivnit.

Přítomnost slova ve slovníku

Dalším příkladem příznaku, který se jeví jako zřejmý, je využití slovníku obsahujícího pojmenované entity. v 5.3.3 již bylo jeho použití naznačeno. Slova jsou vyhledávána ve slovnících dodaných vedoucím práce, které byly již v odkazované části práce vyjmenovány.

Využitím seznamů známých entit lze velmi snadno identifikovat často se vyskytující entity (jména, geografické názvy, svátky atd.). v českém jazyce ale narážíme na typický problém, kterým je ohýbání slov. v běžném textu se tyto názvy vyskytují různě vyskloňovaná, přičemž slovo v jakémkoliv tvaru jiném než v podobě, v jaké se ve slovníku nachází, je nemožné v seznamu najít. Tato nevýhoda je však relativně snadno odstranitelná využitím stemmingu položek ve slovnících i slov v nich vyhledávaných.

Délka generovaného vektoru je dána počtem používaných slovníků. Na hodnotu 1 je pak nastaven ten prvek, který odpovídá slovníku, v němž bylo slovo nalezeno.

Afixy

Další příznak je založen na extrakci afixů. Afixem rozumíme morfém, který se připojuje k základnímu morfému (kořeni nebo kmeni) slova. v tomto případě jsou důležité především sufixy (ačkoliv toto označení je nepřesné, neboť extrahovány jsou libovolně dlouhé konce slov).

Tyto koncovky/přípony vypovídají o vlastnostech slov na daných pozicích (tento příznak opět bere v potaz předvolený kontext). Lze z nich například do určité míry rozpoznat slovní druh slova a podobná specifika slov vyskytující v okolí daného slova.

Délka generovaného vektoru je dána počtem nalezených afixů. Nastaven je jako u předchozích vektorů ten prvek vektoru, který odpovídá afixu zpracovávaného slova.

Words a Bag of words

Další příznak se zaměřuje na celá slova. Při průchodu testovacími daty si systém uloží všechna slova, která viděl. Zabývá se jak právě zpracovávanými slovem, tak i slovy v jeho okolí. Extrakce příznaků následně probíhá dvěma možnými způsoby, ze kterých vycházejí dvě varianty a dvě různé implementace. Základem je, aby se systém naučil slova, která indikují přítomnost pojmenované entity a její typ.

První implementace (Words) se zabývá jedním slovem na konkrétní pozici vzhledem ke zpracovávanému tokenu. v generovaném vektoru je nastavován jeden prvek, který mu odpovídá.

Druhá implementace (Bag of word) se zaobírá všemi slovy v definovaném okolí nehledě na jejich pozici. Namísto jedné pozice jsou ve vektoru nastaveny všechny ty, které odpovídají slovům ležícím v okolí slova.

Výhodou příznaku je, že není zapotřebí žádného slovníku ani předem vytvořených dat. Systém se učí přímo z trénovacích dat.

N-gramy

Některé pojmenované entity lze dobře identifikovat podle skupin slov, které se vyskytují v bezprostředním okolí slov. Aby bylo možno tyto skupiny lépe zachytit, vytvořme pro každé slovo sledy po sobě jdoucích slov, které jsou dány jeho kontextem. Tyto skupiny jsou nazývány n-gramy. v případě, kdy jsou vytvářeny sledy dvou po sobě jdoucích slov, hovoříme o *bigramech*, v případě tří slov o *trigramech*.

Výsledkem extrakce bigramů z řetězce „..., že byl Ing. Pavel Novák obeznámen se skutečností...“ pro slovo „Pavel“ bude množina {(byl, Ing.), (Ing., Pavel), (Pavel, Novák), (Novák, obeznámen)}. Důležitá je skutečnost, že při trénování byl s nemalou pravděpodobností zachycen pro entitu typu osoba bigram (byl, Ing.), případně (Ing., Pavel), či možná dokonce (Pavel, Novák) a klasifikátor je již naučen, že přítomnost těchto bigramů indikuje přítomnost pojmenované entity typu osoba.

Word clusters

Základem posledního příznaku je sémantický model, který je vystavěn na základě analýzy kontextů v korpusu. Zpravidla se jedná o tvorbu vektorů popisujících kontext slov. Vzniklý vektorový prostor lze použít k výpočtu podobnosti slov. Aplikací shlukování na tento prostor lze získat *sémantické třídy*. Pro odvozování významu slov jsou používány modely jako *HAL* (Hyperspace Analogue to Language) či *COALS* (Correlated Occurance Analogue to Lexical Semantic) [22].

Tato práce využívá shluky slov vytvořené právě výše zmíněnými metodami, které byly dodány vedoucím práce již připravené k použití. Při trénovací fázi se systém naučí, ze kterých shluků pocházejí slova v okolí pojmenovaných entit všech tříd. Do generovaného vektoru jsou tedy zaznamenávány ty shluky, v nichž se zpracovávané slovo nachází.

Slova v používaných shlucích mají dvě různé podoby. Slova v první skupině shluků jsou zachována ve svém původním tvaru (včetně flexe), v té druhé jsou obsaženy shlukované stemované verze.

5.5 Evaluace

Po implementaci příznaků nastává fáze testování, která jejich implementaci ověří a zároveň povede k optimalizaci jejich používané kombinace a k co nejlepší volbě parametrů tak, aby byla úspěšnost systému co nejvyšší.

Postup volby ideální kombinace příznaků je prováděn iterativně. v prvním kroku jsou provedena měření se všemi příznaky jednotlivě, což, mimo jiné, poskytne prvotní informaci o správnosti implementace a o použitelnosti příznaku na zpracovávaná data. v každé další iteraci je vybrána nejlepší kombinace z předchozího kroku a jsou testovány všechny kombinace s ostatními příznaky. Při výběru příznaku se nevracíme, algoritmus je tzv. *žravý* (greedy).

Následuje formální popis algoritmu výběru příznaků:

Data: F = feature set; Max = 0; Last = 1;
Result: R = $\emptyset$
while *F != $\emptyset$* **do**
  [Last, BestF] = $\underset{f \in F}{\text{argmax}}(evaluate(R, f))$;
  if *Max* $\leq$ *Last* **then**
    F.remove(BestF);
    R.add(BestF);
    Max = Last;
  end
  else
    break;
  end
end

Algoritmus 1: Výběr množiny příznaků

5.5.1 Výsledky

V části 3.2.3 bylo zevrubně vysvětleno, jak lze provádět měření úspěšnosti systému. Hlavní sledovanou mírou při ladění, výběru příznaků a finálním hodnocení bude *Micro-averaged strict F-measure*. Pro upřesnění dodejme, že do vzorce dosazujeme za *true positive* pouze výsledky, které jsou *correct* (3.2.3), za *false positive* považujeme součet výsledků *not correct* a *partially correct* a konečně za *false negative* dosazujeme sumu *not marked* a *partially marked*.

Předkládané výsledky zachycují pro oba typy dat nejprve měření provedená za použití jednotlivých příznaků samostatně. Dále následují vybrané experimenty a detailnější statistika úspěšnosti finální podoby systému.

CNEC

Následují hodnoty naměřené pro systém používající vždy pouze jeden konkrétní příznak. Velikost okolí pro příznaky používající kontext byla určována experimentálně tak, aby příznaky poskytovaly co nejlepší výkon. Pro většinu z nich to znamená, že jsou brány v potaz dvě slova vzad i vpřed. Ostatní parametry příznaků, jako například délka afixů, či velikost prahů, byly zvoleny

opět na základě experimentů.

Příznak	**F-measure**
Affixy	0,55
Word	0,47
Word – stem	0,52
Bag of words	0,13
Bag of words – stem	0,15
Ortografický	0,25
N-gramy	0,18
Slovníkový	0,14
HAL, COALS	0,45
COALS – stem	0,55

Tabulka 5.1: Výsledky experimentů s CNEC pro jeden příznak

Výsledky experimentů provedených na samostatných příznacích (zachycené v tabulce 5.1) prozrazují, že nejlépe systém funguje za použití afixů. Velmi dobré úspěšnosti lze též dosáhnout sémantickou analýzou či na základě slov na určité pozici.

Největší skok v úspěšnosti systému způsobilo současné použití příznaků HAL, COALS, COALS – stem, afixů, a ortografického, které společně zvýšily úspěšnost na 0.70.

Následují výsledky finální podoby systému, který využívá všechny implementované příznaky (přidání žádného příznaku úspěšnost nesnížilo), které navíc zachycují i úspěšnost systému pro jednotlivé třídy pojmenovaných entit:

Třída	Strict			Lenient		
	P	R	F	P	R	F
Geografické názvy	0,73	0,70	0,72	0,80	0,77	0,78
Názvy institucí	0,60	0,54	0,57	0,71	0,65	0,68
Jména osob	0,77	0,79	0,78	0,82	0,85	0,83
Názvy věcí	0,63	0,53	0,57	0,79	0,69	0,73
Časové údaje	0,89	0,83	0,86	0,94	0,89	0,91
Názvy médií	0,69	0,49	0,57	0,78	0,56	0,65
Čísla jako součásti adres	0,71	0,80	0,75	0,80	0,90	0,85

Tabulka 5.2: Výsledky experimentů (CNEC) s finální podobou systému pro jednotlivé třídy

Výsledky pro celý systém dohromady:

Macro F-measure len.	78,09 %
Macro F-measure str.	69,16 %
Micro F-measure len.	79,30 %
Micro F-measure str.	70,69 %

Tabulka 5.3: Výsledky experimentů s finální podobou systému

Zhodnocení výsledků Výsledky experimentů nejsou totožné s výsledky experimentu popsaného v [21], který dosáhl hodnoty Micro F-measure strict **74.08** (ačkoliv navíc používá příznaku založeném na LDA – Latent Dirichlet allocation). Toto zjištění lze vysvětlit rozdílnou implementací příznaků, jejich rozdílným nastavení či například použitím jiné sady vzorů pro ortografický příznak. Dosažené výsledky nicméně znamenají, že implementace systému je korektní, systém funguje a je připraven k použití na experimenty s daty pocházejícími ze sociálních sítí.

Sociální sítě

Pro data získaná ze sociálních sítí byl zopakován popsaný postup pro výběr příznaků. Pro první iteraci byly získány tyto výsledky:

Příznak	F-measure
Affixy	0,80
Word	0,81
Word – stem	0,79
Bag of words	0,36
Bag of words – stem	0,37
Ortografický	0,04
N-gramy	0,58
Slovníkový	0,01
HAL, COALS	0,48
COALS – stem	0,69

Tabulka 5.4: Výsledky experimentů s příspěvky na internetovém fóru pro jeden příznak

Třída	Strict			Lenient		
	P	R	F	P	R	F
Produkty	0,85	0,69	0,76	0,92	0,74	0,82
Organizace	0,95	0,88	0,91	0,96	0,88	0,92

Tabulka 5.5: Výsledky experimentů s finální podobou systému pro jednotlivé třídy

Opět následuje celkové zhodnocení systému:

Macro F-measure len.	86,95 %
Macro F-measure str.	83,85 %
Micro F-measure len.	86,41 %
Micro F-measure str.	83,04 %

Tabulka 5.6: Výsledky experimentů s finální podobou systému

Zhodnocení výsledků Systém na datech pocházejících ze sociálních médií poskytuje lepší výsledky, což je očekávaný výsledek. Oproti CNEC je totiž značně omezena množina slov, která jsou pojmenovanou entitou, což je dáno doménou, kterou se texty zabývají (telekomunikace).

Tento poznatek je podložen faktem vypozorovaným z experimentů: systém dosahuje za použití příznaků využívajících afixů a jednotlivých slov na

daných pozicích úspěšnosti velmi blízké finální podobě sady příznaků, jejichž přidáním se úspěšnost oproti experimentům s CNEC příliš nezvýšila.

Vzhledem k velikosti dat je pravděpodobné, že trénovací data obsahují většinu operátorů, výrobců mobilních zařízení a příslušenství, kterých existuje menší množství než jejich produktů. z tabulky 5.5 je vidět, že právě pro organizace funguje systém lépe. Produktů nejenže existuje více druhů, ale navíc lze použít více označení např. pro tentýž model mobilního telefonu, a to ať už oficiálních, kdy někteří výrobci používají obchodní označení pro tentýž výrobek odlišná od toho oficiálního, či vytvořených uživateli (iPhone 6 = iPhone6 = iP6).

6 Závěr

Cílem této práce bylo prozkoumat metody pro rozpoznávání pojmenovaných entit, a to zejména metody založené na strojovém učení. Dále bylo úkolem vybrat jednu z těchto technik a množinu příznaků, které by byly vhodné pro aplikaci na data pocházejících ze sociálních medií, a zvolené řešení implementovat. Nakonec byl vytvořený systém natrénován a ověřen na datech, jejichž příprava byla součástí práce. Posledním úkolem bylo vyhodnocení dosažených výsledků a jejich porovnání s dalšími systémy.

Teoretická část se zabývá popisem metod používaných ve strojovém učení, problematikou zpracování přirozeného jazyka a jejími dílčími úlohami. Detailně rozebírá rozpoznávání přirozených entit, problémy, které s ním souvisejí, a techniky, které jsou pro jeho realizaci používány.

Součástí praktické části je popis použitých knihoven a implementovaných příznaků, rozbor vstupních dat a identifikace rozdílů mezi dvěma použitými sadami textů s různým původem. Především pak obsahuje popis a výsledky experimentů, jejich diskusi a zhodnocení.

Vytvářený systém dosahuje na standardizovaném korpusu podobné úspěšnosti jako ten, který se snaží napodobit. Případná navazující práce by mohla spočívat v další optimalizaci úspěšnosti implementací nových příznaků či úpravou těch stávajících.

Systém na datech pocházejících ze sociálních sítí dosahuje očekávané úspěšnosti. Vybraná data se zaměřovala na oblast telekomunikací a telekomunikačních technologií. Předmětem dalšího výzkumu by mohly být experimenty s jinými odvětvími či s daty obsahujícími jejich kombinaci.

Nejproblematičtější a časově náročnou částí práce byla příprava dat, která musela být ručně označena v nástroji vyvíjeném paralelně s touto prací.

Literatura

[1] JONÁK, Zdeněk., Informační společnost, *KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV)*, Národní knihovna ČR, Praha, 2003, Citováno: 6.4.2016

[2] FORTMANN-ROE, Scott, *Understanding the Bias-Varhttp://home.zcu.cz/ konkol/Brainy/documentation.pdfiance Tradeoff*, 2012, dostupné (25.4.2016) z http://scott.fortmann-roe.com/docs/BiasVariance.html

[3] CARBONELL, Jaime G., MICHALSKI, Ryszard S., MITCHELL, Tom M., *Machine Learning: An Artificial Intelligence Approach*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1983, 978-3-662-12405-5

[4] SANG, Erik T. K., *Introduction to the conll-2002 shared task: Language-independent named entity recognition*, Proceedings of CoNLL-2002, Taipei, Taiwan, 2002

[5] GRISHMAN, Ralph, SUNDHEIM, Beth, *Message understanding conference-6: a brief history*, Proceedings of the 16th conference on Computational linguistics - Volume 1, Stroudsburg, PA, USA, 1996

[6] SANG, Erik T. K., De MEULDER, Fien, *Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition*, Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 - Volume 4, Stroudsburg, PA, USA, 2003

[7] FLEISCHMAN, Michael, *Automated Subcategorization of Named Entities*, Proc. Conference of the European Chapter of Association for Computational Linguistic, 2001

[8] MANNING, Chris, SCHÜTZE, Hinrich, *Foundations of Statistical Natural Language Processing*, MIT Press. Cambridge, MA, USA, 1999

[9] CHIEU Hai Leong, NG Hwee Tou, *Named Entity Recognition with a Maximum Entropy Approach*, CONLL '03 Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 - Volume 4, Association for Computational Linguistics Stroudsburg, PA, USA, 2003

[10] GUPTA, Rahul, *Conditional Random Fields*, Unpublished report, IIT Bombay, 2006

[11] CORTES, Corinna, VAPNIK, Vladimir, *Support-Vector Networks*, Machine Learning, Kluwer Academic Publishers, Boston, 1995

[12] WESTON, Jason, *Support Vector Machine (and Statistical Learning Theory) Tutorial*, 4 Independence Way, Princeton, USA, dostupné (10.5.2016) z `http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf`

[13] SHOKHIREV, Nikolai: *Hidden Markov Models*, 2010, dostupné (11.5.2016) z `http://www.shokhirev.com/nikolai/abc/alg/hmm/hmm.html`

[14] LAFFERTY, John D., MCCALLUM, Andrew, PEREIRA, Fernando C. N., *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*, ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2001

[15] SUTTON, Charles, MCCALLUM, Andres, *An Introduction to Conditional Random Fields*, Foundations and Trends® in Machine Learning, Vol. 4, No. 4, 2011

[16] VANTUCH, Marek, *Metody strojového učení ve zpracování přirozeného jazyka*, Brno, 2011

[17] KONKOL, Michal, KONOPÍK, Miloslav, *Maximum Entropy Named Entity Recognition for Czech language*, Plzeň, 2016

[18] ASCH, Vincent Van, *Macro- and micro-averaged evaluation measures*, 2013, dostupné (14.6.2016) z `http://www.cnts.ua.ac.be/~vincent/pdf/microaverage.pdf`

[19] KONKOL, Michal: *Brainy: a machine learning library*, ICAISC 2014, LNCS, svazek 8468, Part II, str. 490-499. Springer, Heidelberg, 2014

[20] BRYCHCÍN, Tomáš, KONOPÍK, Miloslav, *HPS: High precision stemmer*, Information Processing & Management, svazek 51, str. 68–91, 2015

[21] KONKOL, M., BRYCHCÍN, T., KONOPÍK, M.: *Latent semantics in Named Entity Recognition*. Expert Systems with Applications, svazek 42(7), str. 3470–3479, 2015.

[22] ROHDE, Douglas L. T., GONNERMAN, Laura M., PLAUT, David C., *An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence*, Communications of the acm, svazek 8, str.627–633, 2006

A Uživatelský manuál

A.1 Systémové požadavky

Pro spouštění aplikace je zapotřebí běhové prostředí Java 1.8. Vzhledem k náročnosti na operační paměť je doporučeno spouštět systém na stroji s alespoň 16 GB RAM.

Pro sestavení aplikace je doporučeno použití systému Maven.

A.2 Sestavení aplikace

Aplikace pro svoje sestavení a svůj běh potřebuje knihovny HPS a Brainy, které nejsou součástí žádného veřejného repozitáře. Proto je doporučeno je nainstalovat do lokálního repozitáře ze složky `lib`.

V kořenovém adresáři projektu spustíme sestavení, které vytvoří spustitelný JAR, příkazem

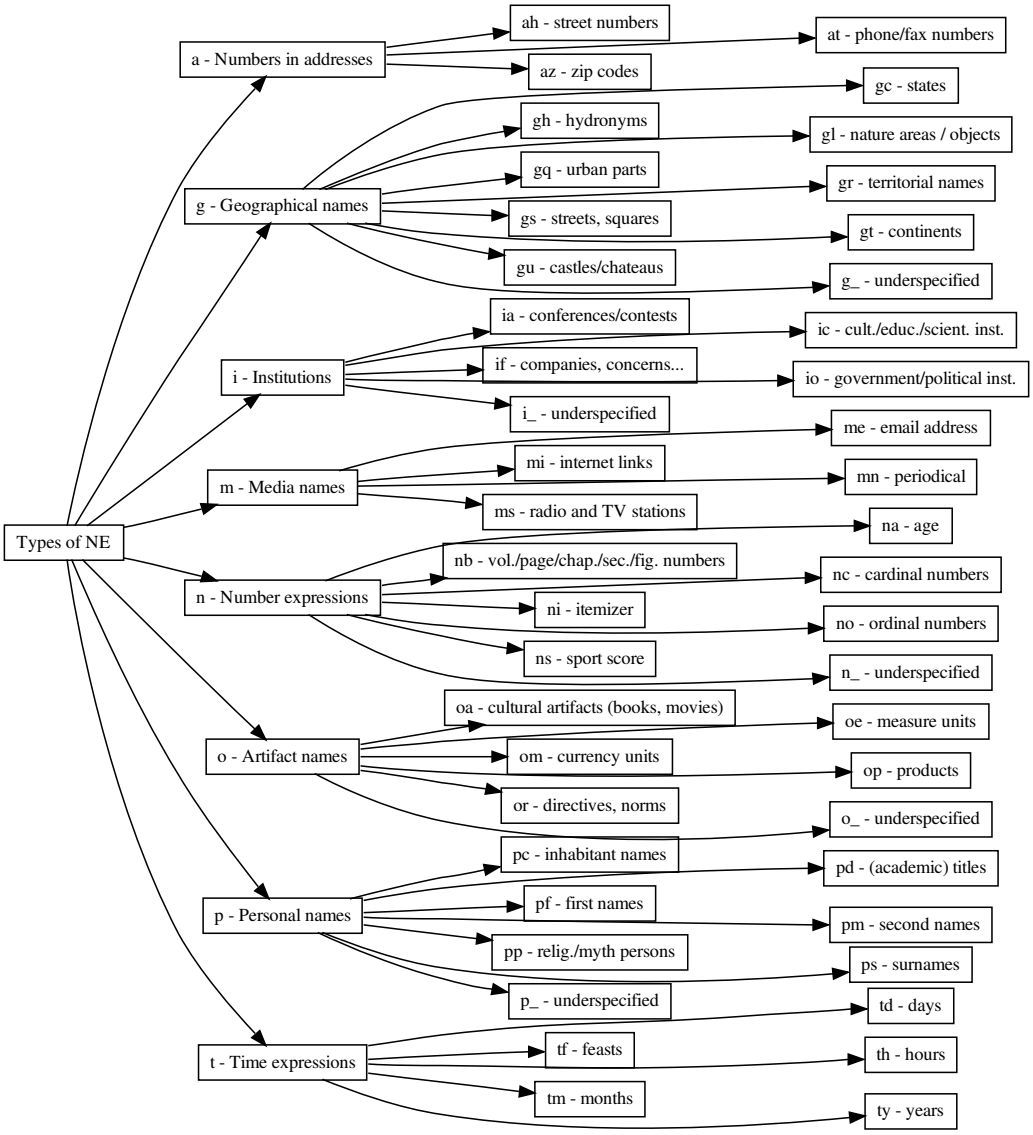
```
mvn package assembly:single
```

A.3 Spuštění aplikace

JAR soubor musí být na stejné cestě jako složka data, která obsahuje data potřebná k běhu. Aplikaci spustíme příkazem

```
java -jar <JAR_path> <train_data_file> <test_data_file>
    [output_file [statistics_file]].
```

B Pojmenované entity v CNEC



Zdroj: `http://ufal.mff.cuni.cz/~strakova/cnec2.0/ne-type-hierarchy.pdf`

C Skript pro extrakci *.txt souboru z JSON

```
import groovy.json.JsonSlurper

if(args.length == 0) {
  System.err.println "usage: groovy ${getClass().
     protectionDomain.codeSource.location.getFile().
     replaceAll(".*/", "")} <filename>"
} else {
  File input = new File(args[0])
  if(!input.exists()) {
    System.err.println "File does not exist!"
  } else {
    def parser = new JsonSlurper()
    def result = parser.parseText(input.text)
    def parsed = result.collect{it.text}.join("\n")
    new File(args.length > 1? args[1] : args[0].
       replaceAll(/\..*$/, ".out")).text = parsed
  }
}
```

D Skript pro rozdělení *.txt souboru na jednotlivé dokumenty

```
if(args.length == 0) {
  System.err.println("usage: groovy ${getClass().
     protectionDomain.codeSource.location.getFile().
     replaceAll(".*/", "")} <filename>")
} else {
  def input = new File(args[0])
  if(!input.exists()) {
    System.out.println("File not found")
  } else {
    def fileName = args[0].replaceAll(/\..*$/, "")
    def outDir = new File(fileName)
    if(outDir.exists()) outDir.delete()
    outDir.mkdir()
    def lines = input.readLines()
    def count = (lines.size() + "").length()
    lines.eachWithIndex{line, index->
      new File("${outDir.getPath()}/${fileName}-${
         String.format('%0'+count+'d', index)}.txt").
         text = line
    }
  }
}
```

E Skript pro přípravu vstupních dat

```
import cz.zcu.kiv.witz.dp.util.Tokenizer
import cz.zcu.kiv.witz.dp.structs.Token

def annotatedFiles = new File(args[0]).listFiles()
def dataFolderLocation = args[1]

def tokenizer = new Tokenizer()

def fileExtLength = '.annotation'.length()
for (File annotatedFile : annotatedFiles) {
  def annotatedFileName = annotatedFile.getName()
  def annotationsText = annotatedFile.text

  def dataFile = new File(dataFolderLocation +
     annotatedFileName.substring(0, annotatedFileName.
     length() - fileExtLength))
  def tokens = tokenizer.tokenize(dataFile.text)

  XmlSlurper slurper = new XmlSlurper()
  def rootNode = slurper.parseText annotationsText
  def annotations = rootNode.Annotation

  tokens.each {token->
    def annotation = annotations.find {it.Start.text().
       toInteger() <= token.getFrom() && token.getFrom
       () < it.End.text().toInteger()}
    if(annotation)
      println "${token.text}\t${annotation.Type.text().
         substring(0,␣3).toUpperCase()}"
    else
      println "${token.text}\tOTH"
  }
  println ""
}

def dataFilesSorted = new File(dataFolderLocation).
   listFiles()
```

```
def annotatedFilesNames = annotatedFiles.collect{it.
    getName()}
String lastFileName = annotatedFiles.sort{it.getName()
    }[-1].getName()
int lastFileNum;
def numFinder = {String name -> int value; name.find
    (/-(\d+).txt/){value = Integer.parseInt(it[1])};
    return value;}
lastFileNum = numFinder(lastFileName)

def notAnnotated = []
for (File dataFile : dataFilesSorted) {
  if (numFinder(dataFile.getName()) <= lastFileNum &&
      !(dataFile.getName() + ".annotation" in
      annotatedFilesNames)) {
    notAnnotated << dataFile.getName()
    def tokens = tokenizer.tokenize(dataFile.text)
    for (Token token : tokens ) {
      println(token.text + "\tOTH")
    }
    println ""
  }
}
```

F UML diagram aplikace