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

Bakalářská práce

Propojení konverzací ve vlákně fóra

Plzeň 2017 Štěpán Baratta

Místo této strany bude zadání práce.

Prohlášení

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

V Plzni dne 1. května 2017

Štěpán Baratta

Abstract

Internet discussion forums have gained incredible popularity over the recent years. Often, conversations are sorted only by date with the relevant posts hidden in between others. That is the reason conversation threads become quite confusing to understand and sometimes it is hard to navigate in them. This work focuses on the problem of linking user posts in discussion forums, depending on whether a connection between posts exists. The end result is visualized in a tree structure to better see the relationships between posts in the discussion. The first part describes theoretical background of machine learning algorithms and NLP (Natural language processing). The second part contains description of implemented method. We reached the maximum F-score of 0.604. Considering the difficulty of the initial problem, it is a decent result.

Abstrakt

Internetová fóra nebo diskuzní portály na sociálních sítích dnes zažívají ohromnou popularitu u lidí všech věkových skupin. Některá fóra jsou řazená pouze podle data přidání, nikoliv podle konverzací uživatelů. Často se tedy stává, že užitečné příspěvky jsou ztraceny mezi ostatními. To způsobuje, že konverzace jsou nepřehledné a dochází ke ztrátě kontextu. Tato práce se zaměřuje na problém propojování příspěvků v diskuzních portálech podle toho, zda příspěvky spolu souvisejí. Konečný výsledek je vizualizován stromovou strukturou pro lepší orientaci v diskuzi. V první části práce jsou popsány teoretické znalosti z oboru strojového učení a NLP (Zpracováni přirozeného jazyka). Druhá část obsahuje popis implementace a otestování implementované metody. Bylo dosaženo úspěšnosti metriky F-score 0.604, což je vzhledem k obtížnosti úlohy ucházející výsledek.

Obsah

1 Úvod

Online fóra jsou v dnešní době velice rozšířená. Internetová fóra mají široké použití ve všech možných oblastech. Může se jednat o fóra zákaznické, technické podpory nebo různá tématická fóra. Některé mohou sloužit také jako zdroj pro vzdělávání. Uživatelé sdílejí nápady, problémy a řešení těchto problémů. Diskuzních fór dnes existuje veliké množství obsahující nepřeberné množství dat. Výsledkem toho je, že zájem o získávání znalostí a extrakci informací z těchto zdrojů se v poslední době rapidně zvýšil.

Cílem této práce je implementace vhodné metody, která bude schopna z daného vstupu, což bude posloupnost příspěvků z internetového konverzačního vlákna, nalézt související příspěvky (jeden příspěvek reagující na druhý) a následně vizualizovat stromovou strukturu, zobrazující jednotlivá vlákna.

Typická internetová diskuze se skládá z jednotlivých příspěvků psaných různými uživateli. Motivace uživatelů k účasti v těchto diskuzních fórech může být různá, nejčastěji jsou lidmi používána jako platforma pro řešení problémů z různých oblastí. Jeden uživatel vytvoří vlákno (*thread*) s popisem svého problému a spoléhá na ostatní, že poskytnou řešení tohoto problému, popřípadě pouze nějakou odpověď, která se tohoto problému dotýká.

- Nejčastěji jsou příspěvky ve fórech seřazeny v chronologickém pořadí, což není vždy úplně vhodné, jelikož příspěvky často reagují na jiný příspěvek, který je mu vzdálený a čtenáři tohoto fóra může unikat kontext odpovědí.

- Příspěvek, na který odpovídá více lidí, může být ztracen mezi stovkami jiných příspěvků. Pokud se podaří takovýto příspěvek identifikovat, můžeme ho dát do popředí, aby byl více viditelný.

Ve výsledku by tato práce měla z nestrukturovaného konverzačního vlákna vytvořit konverzační strom, kde budou jasně viditelné spolu související příspěvky.

Celá práce je rozdělena na 2 základní části. První část se zabývá teoretickým popisem problematiky. V první kapitole je popsán proces při zpracování přirozeného jazyka, v další kapitole jsou popsány základní algoritmy strojového učení v úloze klasifikace. Poslední kapitola dává přehled o způsobech vyhodnocení klasifikačního modelu.

Druhá část je zaměřena na vlastní zpracování úlohy této práce. Jsou zde popsány zdroje dat, jejich formát a proces jejich označování. Dále architektura aplikace popisující základní moduly v programu. Největší část je věnována návrhu implementace, kde je popsán celý proces stažení dat až po grafický výstup aplikace. Nakonec jsou zobrazeny dosažené výsledky a jejich zhodnocení.

2 NLP

2.1 Zpracování přirozeného jazyka

Zpracování přirozeného jazyka (*Natural Language Processing, NLP*) je odvětví umělé inteligence, které se zabývá analýzou, porozuměním a generováním jazyků, které člověk přirozeně používá.

Některé z hlavních cílů NLP:

- **Strojový překlad** – proces, při kterém je použit počítačový program k přeložení přirozeného textu z jednoho jazyka do druhého. Přesto, že se tento proces jeví jako přímočarý, ve skutečnosti je daleko komplexnější. Překladač musí být schopen interpretovat a analyzovat jednotlivé prvky v textu a zároveň brát ohled na to, jak se jednotlivá slova mohou ovlivňovat. Protože každý jazyk má svá specifika, u každého je nutné dobré pochopení gramatiky, syntaxe a sémantiky jazyka.

- **Sumarizace názorů** – Používá se jako pomocný nástroj pro zpracování a vygenerování stručného a výstižného shrnutí z velkého množství vstupních dat z internetových portálů. Mezi ideální cíle patří například sociální sítě (*Facebook, Twitter...*).

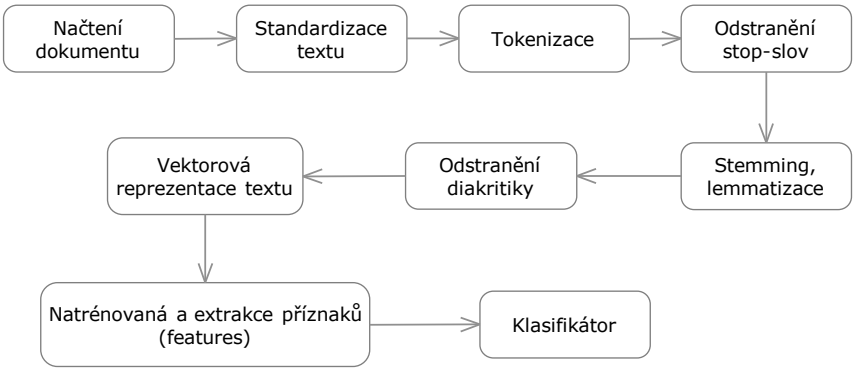
- **Extrakce informací** – Zabývá se nalézáním specifických částí dat v textu dokumentu extrahováním strukturovaných informací z nestrukturovaného textu. Toho může docílit pomocí rozpoznávání referencí na pojmenované entity (*Named entity recognition*) a také rozpoznáváním vztahů mezi entitami. Systémy pro extrakci informací mohou být používány přímo k pochopení abstraktních informací v textu nebo k extrakci konkrétních dat z množiny dokumentů, které mohou být dále zpracovány běžnými metodami dolování z textu (*data-mining*). [14]

- **Klasifikace dokumentů** – Úkolem je přiřazení dokumentu do jedné či více tříd nebo kategorií nejčastěji za pomoci nějakých algoritmů strojového učení. Nemusí se jednat pouze o klasifikaci textu, ale také o obrázky, hudbu atd.

2.2 Předzpracování textu

Pro naši potřebu je nutné vstupní text rozdělit na jednotlivá slova, abychom s nimi mohli dále pracovat. V této kapitole bude popsán postupný proces,

kterým text prochází. Tento postup by měl být univerzální pro jakýkoliv jazyk, lišit se bude pouze v použití jinojazyčných slovníků. Na obrázku 2.1 vidíme posloupnost kroků nutných k předzpracování textu a následné klasifikace.



Obrázek 2.1: Proces zpracování textu

2.2.1 Normalizace textu

Nashromážděná data jsou často uložena v různých formátech, potřebujeme je tedy všechny převést do nějakého jednotného formátu tak, aby práce s nimi byla co nejjednodušší a nejefektivnější. Zde se nabízejí dva hlavní formáty: JSON a XML. V tomto případě mi více vyhovoval XML formát.

2.2.2 Segmentace

V prvním kroku je třeba rozdělit data na jednotlivé segmenty. V našem případě na jednotlivé věty. Zde se objevuje první problém, a to správná detekce jednotlivých vět. Například tečka má v různých kontextech jiný význam, může se vyskytnout jako ukončení zkratky, v řadových číslovkách, případně jako oddělovač tisíců v číslovkách.

2.2.3 Tokenizace

Dalším krokem automatické analýzy je vyčlenění jednotek, z nichž je text z hlediska programu automatické analýzy složen. V případě automatického zpracování textů se v prvním kroku jedná o tokenizaci - tj. rozčlenění textu

na jednotlivá slova, která budou předmětem další analýzy. Takto vytvořené jednotky se nazývají *tokeny*.

2.2.4 Stop-slova

Stop slova jsou taková slova, která se v textu vyskytují velice často, zároveň ale nenesou sémantický význam. V češtině jsou to například:

- **spojky** (*a, i, ani...*)
- **předložky** (*že, před, po...*)
- **zájmena** (*já, ty on, ona...*)
- **často používaná slovesa** (*mít, být...*)

Kvůli jejich nízkému významu se tato slova ze slovníku odstraní, což by mělo zpřesnit výsledky následného zpracování.

Obecnou strategií pro identifikaci stop-slov je seřazení všech slov ve slovníku podle jejich četnosti výskytu. Z nejčastěji se vyskytujících se vytvoří slovník. Jediným problémem tohoto postupu je určení meze, jaké slovo je ještě možné brát jako nevýznamové.

K nalezení jsou i volně dostupné již předvytvořené slovníky. Pro účely této práce jsem použil volně dostupný slovník z adresy

`http://www.ranks.nl/stopwords/czech`

2.2.5 Stemming, Lemmatizace

Hlavním úkolem stemmingu a lemmatizace je převedení slov do jejich kořenové formy za účelem sjednocení slov, což ve výsledku vede k zredukování velikosti slovníku. Jako stemming se běžně označuje čistě heuristický proces, při kterém dojde k odstranění konce, popřípadě předpony slov. Takto vzniklá jednotka se nazývá *stem*. Výsledný *stem* se nutně nemusí shodovat s lingvistickým kořenem slova, často se však shodovat budou.

Lemmatizace na rozdíl od stemmingu se navíc snaží zohledňovat význam slova za použití správné morfologické analýzy slov. Ve výsledku vznikne *lemma* nebo základní tvar slova. Například slovo *barvě* by lemmatizátor změnil na *barva*. Je proto nutné získání kontextu textu, což je proces velice komplikovaný.

Ve skutečnosti jsou termíny *stem* a *lemma* v anglickém jazyce často zaměňovány.

Pro účely této práce byl použit stemmovací model v českém jazyce *HPS (High Precision Stemmer)*. [4]

2.2.6 N-Gramy

N-gramy jsou množiny n slov jdoucí po sobě. Zkoumané položky mohou být jednotlivá písmena, slova, slabiky, fonémy. N-gramu velikosti 1 se říká *unigram*, velikosti 2 *bi-gram* a velikosti 3 *tri-gram*. Například mějme větu "Dneska je venku krásné počasí". Pokud hledáme bigramy (N=2), potom N-gramy budou vypadat následovně:

- Dneska je
- je venku
- venku krásné
- krásné počasí

Vzniknou tedy 4 bi-gramy

Pro N=3 by to vypadalo:

- Dneska je venku
- je venku krásné
- venku krásné počasí

Pokud X je počet slov ve větě K, počet N-gramů pro větu K je

$$Ngrams_k = X - (N - 1) \tag{2.1}$$

2.2.7 Odstranění diakritiky

V českém jazyce se běžně vyskytují diakritická znaménka. Uživatelé diskuzních fór nebo sociálních sítí často diakritiku z různých důvodů nepoužívají, například z důvodu ušetření času rychlejším psaním. Tím se však komplikuje proces zpracování textu. Je třeba sjednotit všechny texty do jednotného stylu, tedy všechny s diakritikou nebo všechny bez diakritiky.

Pokud bychom nechali část textu s diakritikou a druhou bez, dvě stejná slova by byla brána jako odlišná, například slovo *široký* a *siroky* mají stejný význam, kvůli vynechání diakritiky v prvním příkladu budou slova uložena jako dvě odlišná.

Možnosti vyřešení tohoto problému jsou zřejmě dvě. Diakritika je buď do všech slov doplněna nebo je ze všech slov odstraněna a převedena do formy bez háčků a čárek. Nejjednodušším způsobem je odstranění diakritiky ze všech slov.

2.3 Metody vážení slov

Abychom text mohli dále analyzovat a zpracovávat, potřebujeme ho reprezentovat jako vektor v m-rozměrném prostoru příznaků. Příznakem může být například jedno slovo, posloupnost slov (*n-gram*). Hodnoty složek vektoru můžeme určit binárně, podle frekvence příznaku nebo pomocí váhování slov. Mezi nejčastěji používané metody patří TF-IDF.

2.3.1 TF-IDF

TF-IDF (*Term Frequency- Inverse document Frequency*) je metrika používaná v klasifikaci textů. Základním úkolem je přiřazení statistické váhy jednotlivým slovům.

Váhy slov v *TF-IDF* jsou složeny ze dvou složek:

- Term Frequency - normalizovaná četnost slova v dokumentu
- Inverse document frequency - převrácená četnost slova ve všech dokuementech

Četnost slova (*Term frequency*) se zjistí jednoduše pomocí sčítání výskytů výrazu v dokumentu vydělená počtem všech dokumentů 2.2 a převrácenou četnost(*Inverse document frequency*) vypočteme jako logaritmus počtu všech dokumentů vydělený počtem dokumentů, ve kterých se vyskytuje daný výraz. 2.3 Když jsou tyto dvě složky vynásobeny, dostaneme výslednou hodnotu, která je vysoká pro všechny slova, která se v dokumentu vyskytují často a naopak malou hodnotu pro slova, která se vyskytují nejméně často. Touto metodou tedy získáme slova, důležitá pro daný dokument.

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{2.2}$$

kde

$n_{i,j}$ Počet výskytů slova t_i v dokumentu d_j
$\sum_k n_{k,j}$ Součet počtu výskytů všech slov v dokumentu d_j

$$IDF_i = log(\frac{|D|}{|d : t_i \in d|}) \tag{2.3}$$

kde

D Počet dokumentů v korpusu
$|d : t_i \in d|$ Počet dokumentů, které obsahují slovo i

2.4 Metody podobnosti textů

2.4.1 Kosinová podobnost

Kosinová podobnost udává míru podobnosti dvou vektorů, která se získá výpočtem kosinu úhlů těchto vektorů.

Toho se dá využít při zjišťování podobnosti dvou textů. V takovém případě budou vektory reprezentovat četnost jednotlivých slov.

Matematicky se jedná o skalární součin vektorů vydělený součinem jejich velikostí 2.4.

$$similarity = cos(\phi) = \frac{A * B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} * \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.4}$$

kde

A_i jsou složky vektoru A.
B_i jsou složky vektoru B.

2.4.2 Jaccardův index podobnosti

Jaccardův index podobnosti udává míru podobnosti dvou množin v rozmezí 0% - 100%.

Vypočte se jako velikost průniku obou množin dělená velikostí sjednocení těchto množin.

$$JS(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} \tag{2.5}$$

Příklad: Mějme dva sety $A = 0, 1, 2, 5, 6$ a $B = 0, 2, 3, 5, 7, 9$. Jak jsou si sety A a B podobné? Dosazením do vzorce 2.6 dostaneme

$$JS(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} = \frac{|0, 2, 5|}{|0, 1, 2, 3, 5, 6, 7, 9|} = \frac{3}{8} = 0.375 \tag{2.6}$$

[13]

3 Algoritmy strojového učení

Jedním ze základních úkolů strojového učení je úloha klasifikace, což je problém přiřazení tříd objektům. Tyto objekty je nejdříve třeba popsat. V případě, že objekty jsou textové dokumenty, popisujeme je pomocí příznaků (*features*). U klasifikace dokumentů se jedná o učení s učitelem, tzn., že proces klasifikace probíhá ve dvou krocích

- Natrénování modelu na označených trénovacích datech
- Zařazení neoznačených testovacích dat do správných tříd klasifikátorem

3.1 Naive Bayes

Bayesovský klasifikátor přiřadí nejpravděpodobnější třídu danému příkladu pomocí jeho vektoru atributů. Učení těchto klasifikátorů může být značně zjednodušeno za předpokladu, že atributy jsou nezávislé na dané třídě 3.3. [15]

Bayesův teorém

Bayesova věta vychází z toho, že $P(Y|X) * P(X) = P(X|Y) * P(Y)$. Proto platí:

$$P(Y|X) = \frac{P(X|Y) * P(Y)}{P(X)} \tag{3.1}$$

Naivní Bayesovský klasifikátor

Naivní Bayesovský klasifikátor je metoda učení s učitelem, vycházející z Bayesova teorému 3.1.

Z něho plyne:

$$P(h|D) = \frac{P(D|h) * P(h)}{P(D)} \tag{3.2}$$

kde

$P(h)$ apriorní pravděpodobnost jevu h. Toto je náš apriorní předpoklad, nezávisí na datech.

$P(D)$ pravděpodobnost výskytu pozorovaných dat D bez znalosti pravděpodobnosti jevů.

$P(D|h)$ podmíněná pravděpodobnost výskytu dat D, nastal-li jev h.

$P(h|D)$ aposteriorní pravděpodobnost, pravděpodobnost jevu h za předpokladu, že máme pozorovaná data D.

Mějme data, jejichž instance lze popsat n-ticí hodnot atributů $a_i, ...a_n$ a jejich třídu pro klasifikaci, jež nabývá hodnot $y_i \in Y$. Naivní Bayesovský klasifikátor předpokládá nezávislost hodnot jednotlivých atributů, tedy:

$$P(a_1, ..., a_n|y) = \prod_{i=1}^{n} p(a_i|y) \tag{3.3}$$

Můžeme říct, že příklad a je zařazen do třídy, pro kterou je pravděpodobnost $P(a|y)$ maximální. Ve výsledku tedy dostaneme:

$$y_M = argmax_{y \in Y} P(y) * \prod_{i=1}^{n} P(a_i|y) \tag{3.4}$$

Prvním krokem je naučení klasifikátoru, což vyžaduje trénovací množinu dat D. Podle těch by klasifikátor měl pro budoucí testovací data určit správné výstupní třídy. Naším cílem je aproximovat cílovou funkci $f : X \rightarrow Y$, ekvivalentně $P(Y|X)$

Překvapivě, ačkoliv je sestrojení Bayesova klasifikátoru jednoduché, často má větší úspěšnost než jiné, sofistikovanější metody. [9, 17]

3.2 Max entropy

Model maximální entropie odhaduje pravděpodobnosti na základě použití co nejméně předpokladů. Na rozdíl od Bayesovského klasifikátoru nepředpokládá, že atributy jsou na sobě podmíněně nezávislé. Základní ideou je najít podmíněné rozdělení pravděpodobnosti, které má, za daných podmínek, maximální entropii. Rozdělení pravděpodobnosti má exponenciální formu [6, 18]

$$P(y|x) = \frac{1}{Z(h)} * \prod_{j=1}^{k} \lambda_j^{f_j(x,y)} \tag{3.5}$$

kde

$Z(h)$ normalizační funkce
k počet atributů
$f_j(x, y)$ příznaková binární funkce
o výsledná hodnota - *true* nebo *false*

platí tedy

$$f_j(x, y) = \begin{Bmatrix} 1 & pokud\, x > 0,\ y = true \\ 0 & jinak \end{Bmatrix} \tag{3.6}$$

3.3 Support Vector Machine

Model *Suppport Vector Machine (SVM)*, při použití na klasifikaci, oddělí množinu označených trénovacích dat nadrovinou, která je od nich maximálně vzdálená. Na popis nadroviny stačí pouze nejbližší body, kterých je obvykle málo - tyto body se nazývají *podpůrné vektory* (*support vectors*) a odtud název metody. Tato metoda je ze své přirozenosti binární, tedy rozděluje data do dvou tříd. Rozdělující nadrovina je lineární funkcí v prostoru příznaků.

Máme dány trénovací data (x_i, y_i) pro $i = 1...N$, kde $y_i \in -1, 1$, chceme natrénovat klasifikátor $f(x)$ tak, že

$$f(x_i) = \begin{Bmatrix} \geq 0 & y_i = +1 \\ < 0 & y_i = -1 \end{Bmatrix} \tag{3.7}$$

y_i se rovná +1 nebo -1 podle toho, zda dokument spadá do třídy (+) nebo nespadá do třídy (-). Například pokud $y_i * f(x_i) > 0$, dokument spadá do třídy.

Lineární klasifikátor má následující formu:

$$f(x) = w * x + b \tag{3.8}$$

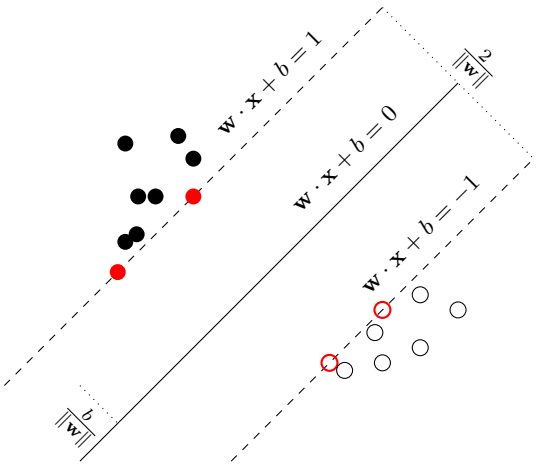
kde

$w -$ váhový vektor

Pro podpůrné vektory zvolíme normalizaci $w * x_+ * b = +1$ pro kladný vektor, respektive $w * x_- * b = -1$ pro záporný vektor.

Okraje(*margin*) potom spočteme jako

$$\frac{w}{\|w\|} * (x_+ - x_-) = \frac{w^T * (x_+ - x_-)}{\|w\|} = \frac{2}{\|w\|} \tag{3.9}$$



Obrázek 3.1: SVM klasifikátor natrénovaný ze vzorků dvou tříd.

Nadrovina rozděluje prostory tříd tak, aby okraje byly co největší. Nalezení nadroviny lze vyjádřit jako následující kvadratický optimalizační problém. [12, 19]

$$\min_{w} \|w\| \text{ tak, aby } y_i * (w^T x_i + b) \geq 1 \text{ pro } i = 1...N \tag{3.10}$$

4 Metody vyhodnocení klasifikace

Existuje mnoho metod k ověření efektivity klasifikátoru, mezi nejčastěji používané však patří přesnost, úplnost a jejich kombinace, F1 score. K získání těchto údajů je třeba nejdříve určit, zda jsme klasifikací získali správně pozitivní (TP), nesprávně pozitivní (FP), nesprávně negativní (FP) správně negativní (TN) (viz. tabulka 4). [8]

TP	Dokument byl klasifikován správně jako patřící ke kategorii
FP	Dokument byl klasifikován nesprávně jako patřící ke kategorii
FN	Dokument nebyl klasifikován jako patřící ke kategorii, ale měl být
TN	Dokument nebyl klasifikován jako patřící ke kategorii a neměl být

Tabulka 4.1: Klasifikace dokumentu

4.1 Přesnost

Přesnost (preciznost) je hodnota určená jako podmíněná pravděpodobnost, že náhodný dokument d je klasifikován do správné kategorie. Reprezentuje schopnost klasifikátoru umístit dokument do správné kategorie oproti všem dokumentům v této kategorii, správných i nesprávných.

$$presnost = \frac{pocet\ vracenych\ relevantnich\ polozek}{celkovy\ pocet\ relevantnich\ dokumentu} \tag{4.1}$$

nebo

$$presnost = \frac{TP}{TP + FP} \tag{4.2}$$

[5, 8]

4.2 Úplnost

Úplnost udává, jaký podíl dokumentů skutečně náležících do pozitivní třídy byl do této třídy klasifikován.

$$uplnost = \frac{pocet\ vracenych\ relevantnich\ polozek}{celkovy\ pocet\ vracenych\ dokumentu} \tag{4.3}$$

nebo

$$uplnost = \frac{TP}{TP + FN} \tag{4.4}$$

4.3 F1 score

Přesnost a úplnost se často kombinují abychom dostali přesnější odhad efektivity klasifikátoru. F1 score spočteme pomocí vztahu

$$F1 = \frac{2 * presnost * uplnost}{presnost + uplnost} \tag{4.5}$$

nebo

$$F1 = \frac{2 * TP}{2 * TP + FP + FN} \tag{4.6}$$

4.4 Cross-validace

Pokud máme limitované množství dat, je vhodné použití cross-validace. Tato metoda rozdělí trénovací množinu dat na část pro testování a zbytek použije pro natrénování. Může se stát, že trénovací množina bude zvolena tak nešťastně, že v ní budou chybět všechny příklady se stejným znakem, klasifikátor nebude schopen se je naučit. Proto se často používá několikanásobné opakování tohoto procesu s tím, že pokaždé se trénovací a testovací množina změní. Na konci se ze všech iterací spočte průměr. Pro nejlepší odhadnutí chyby je ideální volit 10 iterací cross-validace [5]

Další výhodou cross-validace je to, že nám umožňuje zjistit, jak moc se efektivita mění s různými trénovacími množinami. Pokud získáme podobné výsledky pro všechny kombinace trénovacích dat, můžeme si být poměrně jistí, že dosažené výsledky jsou přesné. Pokud se však výsledky výrazně liší, pravděpodobně je chyba v nastavení klasifikátoru.[16]

4.5 Kappa statistika

Kappa statistika určuje shodu mezi dvěma proměnnými X a Y. Kappa může být například použita k porovnání schopností dvou hodnotitelů klasifikovat subjekty do jedné či více skupin. Kappa vždy nabývá hodnoty menší nebo rovno 1, kde hodnota 1 značí absolutní shodu. Ve zvláštních případech může hodnota nabývat záporných hodnot. To značí, že shoda dvou hodnotitelů byla menší, než bylo očekáváno náhodným výběrem.

Zde je jedna z interpretací Kappy (κ): [3]

- Velmi špatná shoda – $\kappa < 0.2$
- Špatná shoda – $0.2 < \kappa < 0.4$
- Středně dobrá shoda – $0.4 < \kappa < 0.6$

- Dobrá shoda – $0.6 < \kappa < 0.8$
- Výborná shoda – $0.8 < \kappa < 1$

Nejdříve je nutné spočítat pozorovanou shodu

$$Pr(a) = \frac{1}{n} \sum_{i=1}^{g} f_{ii} \tag{4.7}$$

kde

$Pr(a)$	pozorovaná shoda mezi hodnotiteli
g	celkový počet kategorií
n	celkový počet subjektů
f_{ii}	četnost počtu i-té kategorie pro hodnotitele X

Tuto hodnotu musíme porovnat s hodnotou očekávané shody, pokud dva hodnotitelé jsou na sobě nezávislí.

$$Pr(e) = \frac{1}{n^2} \sum_{i=1}^{g} f_{i+} f_{+i} \tag{4.8}$$

kde

$Pr(e)$	očekávaná shoda, pokud jsou hodnotitelé na sobě nezávislí
f_{i+}	součet i-té řádky
f_{+i}	součet i-tého sloupce

Kappa statistika je definovaná takto ;

$$\kappa = \frac{Pr(a) - Pr(e)}{1 - Pr(e)} \tag{4.9}$$

Hodnoty $Pr(a)$ a $Pr(e)$ můžeme vypočíst také jako

$$Pr(a) = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.10}$$

Pro očekávanou shodu platí

$$Pr(e) = \frac{(TN + FP) * (TN + FN) + (FN + TP) * (FP + TP)}{Total * Total} \tag{4.11}$$

[1, 2, 5]

5 Propojování diskuzních vláken

5.1 Data

Klasifikace je v terminologii strojového učení považována za metodu učení s učitelem, což je úkol, který na počátku potřebuje výchozí množinu dat, nazývanou také jako *trénovací korpus dat*. Tento korpus umožňuje klasifikátoru natrénování. Úkolem klasifikátoru je naučení se predikovat správné výstupy pro neoznačená *testovací data*.

Obecně platí, že čím větší trénovací množina dat, tím větší pravděpodobnost, že se klasifikátor naučí rozpoznávat důležité znaky mezi jednotlivými objekty. Bude tedy třeba nashromáždit dvojici souborů dat:

- D_{train} - soubor dvojic příspěvků s označenými třídami pro klasifikaci
- D_{test} - soubor dvojic příspěvků s neoznačenými třídami

5.1.1 Formát dat

Abychom data mohli začít zpracovávat, nejdříve je nutná jejich extrakce ze stránek internetových fór a uložení do vhodného formátu. Byl zvolen formát XML z důvodu jeho jednoduchosti a čitelnosti.

Problémem s extrakcí dat z internetových stránek je to, že je příliš složité vytvoření univerzálního nástroje, který by jakékoliv internetové fórum převedl do strukturované formy. Je to z důvodu toho, že každé fórum má odlišnou strukturu, některé neobsahují všechny potřebné informace.

Protože parsování HTML z webových stránek není předmětem této práce, byl vytvořen algoritmus pro extrakci informací pouze ze dvou internetových fór, a to *www.superforum.zive.cz* a *www.emimino.cz/diskuze*.

Pro parsování HTML byla použita knihovna v jazyce JAVA *Jsoup* ve verzi 1.8.3.

V části 5.1 je zobrazena struktura XML dokumentu, v jaké budou jednotlivá konverzační vlákna uložena.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <thread name=...>
3   <post>
4       <id>...</id>
5       <replyTo>...</replyTo>
6       <timestamp>...</timestamp>
```

```
7         <author>...</author>
8         <text>...</text>
9     </post>
10    <post>
11        <id>...</id>
12        <replyTo>...</replyTo>
13        <timestamp>...</timestamp>
14        <author>...</author>
15        <text>...</text>
16    </post>
17          .
18          .
19          .
20  </thread>
```

Listing 5.1: Struktura XML dokumentu

Každý příspěvek je ohraničen elementem *post*, obsahující pět položek. Položka *id* udává identifikační číslo příspěvku, pod kterým na něj bude odkazováno. Element *replyTo* bude obsahovat identifikátor příspěvku, na který tento příspěvek odpovídá, element *timestamp* udává čas vytvoření příspěvku. V elementu *author* bude jméno autora příspěvku a element *text* bude obsahovat samotný text příspěvku.

Trénovací data

K tomu, abychom dosáhli co nejlepších a nejvíce vypovídajících výsledků, bude potřeba nashromáždit co největší množství označených dat. Pro každé konverzační vlákno musíme projít všechny příspěvky a u každého manuálně určit, zda je odpovědí nebo se vztahuje k nějakému předchozímu příspěvku.

Celý proces vytváření trénovacího korpusu dat není možné automatizovat a musí být prováděn manuálně.

Z toho důvodu, že proces vytváření trénovacího korpusu není možné automatizovat, stává se nejvíce časově náročnou z nejdůležitějších částí práce, protože na vytvořeném trénovacím korpusu závisí výsledná efektivita klasifikátoru.

Takovýto způsob přípravy trénovacích dat se však pojí s problémem popsaným níže.

5.1.2 Problém označování dat

Protože označování dat musí být prováděno člověkem, je tento proces náchylný k chybám vzniklým například překlepy či záměnami příspěvků. Množina dvojic příspěvků je veliká, proto není možné jednoduše zkontrolovat správnost těchto dat.

Dalším problémem při označování trénovacích dat je subjektivita každého člověka. Vždy nejde úplně jednoznačně říci, zda spolu dva úseky textu

souvisejí a velice to záleží na subjektivním názoru každého hodnotitele.

Pro vyhodnocení subjektivity byl proveden experiment, který je popsán v sekci 5.5.

5.1.3 Zdroje dat

Je nutné identifikovat vhodné datové zdroje. Každé diskuzní fórum má svá specifika, proto je těžké vytvoření univerzálního nástroje. Lze očekávat, že úspěšnost klasifikace bude velmi závislá na použitém zdroji. Mezi nejčastější zdroje budou patřit:

- **Diskuzní fóra**

  Klady :

  - délka příspěvků
  - informační hodnota příspěvků

  Zápory :

  - málo reakcí mezi uživateli

  Pro diskuzní a názorová fóra je charakteristické, že příspěvky uživatelů jsou seřazeny chronologicky, jedno konverzační vlákno se věnuje jednomu tématu, většinou je to reakce na první příspěvek. Příspěvky jsou delší a je větší pravděpodobnost, že budou obsahovat nějaké faktické informace. Z těchto důvodů se bude jednat o velmi dobrý zdroj pro následné zpracování.

- **Zpravodajství**

  Klady :

  - délka příspěvků
  - přítomnost citací
  - velká informační hodnota
  - časté reakce

  Zápory :

  - složitější získávání dat

  Některé zpravodajské portály umožňují přispívání do diskuze u jednotlivých zpráv. Je zde velká pravděpodobnost, že lidé na sebe budou častěji reagovat, mohou se i citovat, což významně zjednoduší nalezení

souvisejícího příspěvku. Avšak zde hrozí, že se diskutující často odchýlí od původního tématu.

- **Sociální sítě (Facebook, Twitter...)**

  Klady :

  - množství dat
  - jednoduchost nashromáždění

  Zápory :

  - krátké příspěvky
  - malá informační hodnota
  - časté překlepy

  Komentáře u příspěvků na facebooku jsou dalším možným zdrojem. Výhodou zdroje ze sociální sítě je to, že takovýchto dat lze získat obrovské množství. Na druhou stranu, komentáře jsou často velice krátké a neobsahující moc informací. Proto je také těžké tyto krátké texty nějak zpracovávat. Z těchto důvodů pro tuto úlohu facebook není ideálním zdrojem dat.

Obecně se dá říct, že navržená metoda by měla nejlépe fungovat na takové příspěvky, které jsou dostatečně dlouhé, tzn. je možné z nich dostat nějaké informace, je jich možno nashromáždit veliké množství a ze zdrojů, kde budou časté reakce uživatelů jeden na druhého. Dobře se pracuje s vlákny, kde diskutující reagují na výchozí příspěvek, v němž uživatel popisuje nějaký problém a žádá ostatní o radu.

Nakonec byly vybrány dva zdroje, které nejvíce vyhovovaly uvedeným poznatkům

- *superforum.zive.cz*
- *http://www.emimino.cz/diskuse/*

Tabulka 5.1 ukazuje různá statistická data trénovacího datového korpusu.

Statistika	**emimino**	**superforum**	**Celkem**
Celkový počet vláken	25	9	34
Celkový počet označených příspěvků	2767	1166	3933
Průměrná délka vlákna (# příspěvků)	110.7	116.6	113.6
Průměrná délka vlákna (# slov)	4084	4420	4252
Průměrná délka příspěvku (# slov)	36.9	37.9	37.4

Tabulka 5.1: Statistika trénovacího korpusu

5.2 Architektura aplikace

Aplikace je naprogramována v jazyce JAVA ve verzi 1.8. Pro grafické rozhraní byla použita platforma JavaFx. Jako programovací prostředí bylo použito *IntelliJ IDEA* ve verzi 17.1.

Celá aplikace je rozdělena do 5 základních částí (balíků)

- **Main** – Obsahuje třídu s metodou *main*, kde začíná běh celé aplikace. Metoda main volá metodu *start*, což je hlavní vstupní bod pro aplikace JavaFX.
- **Controllers** – Obsahuje třídy, které jsou spojené se svými FXML soubory. Tyto třídy zpracovávají všechen vstup od uživatele.
- **Model** – V tomto balíku jsou například třídy zajišťující trénování a samotnou klasifikaci. Dále se zde zpracovávající výsledky klasifikace.
- **Features** – Pro každý příznak je vytvořena samostatná třída, implementující rozhraní `Feature<>`
- **Data** – Obsahuje definice struktur, do kterých jsou ukládána data. Například *UserPost* reprezentující jeden příspěvek, *ConversationThread* jedno konverzační vlákno a *PostPairWrapper* dvojici příspěvků určenou ke klasifikaci.
- **Tools** – Zde jsou obecné třídy pro načítání XML souborů, výpočet TF-IDF, převod data do jiného formátu...

Díky použití frameworku JavaFX je jednoduché oddělit prezentační část od části logické. Pro definici grafického rozhraní jsou použity soubory *FXML*. Tento formát je skriptovací jazyk založený na formátu XML, umožňující strukturované definování uživatelského rozhraní, které je naprosto oddělené od kódu. Tyto soubory *FXML* jsou namapovány přímo do jazyka JAVA, je proto možné používat API rozhraní k manipulaci těchto dokumentů. Z hlediska architektury MVC tvoří *FXML* soubory část prezentační (View). Kontroler tvoří javovská třída, která je definována jako kontroler souboru *FXML*. Model tedy tvoří objekty, které jsou s kontrolerem spojené.

5.3 Návrh implementace

Cílem této práce na nejnižší úrovni je vytvoření metody, která bude schopna pro dva krátké texty (příspěvky v konverzačním vláknu) říci, zda tyto texty mají nějaké společné znaky, souvisejí spolu, odpovídají jeden na druhého.

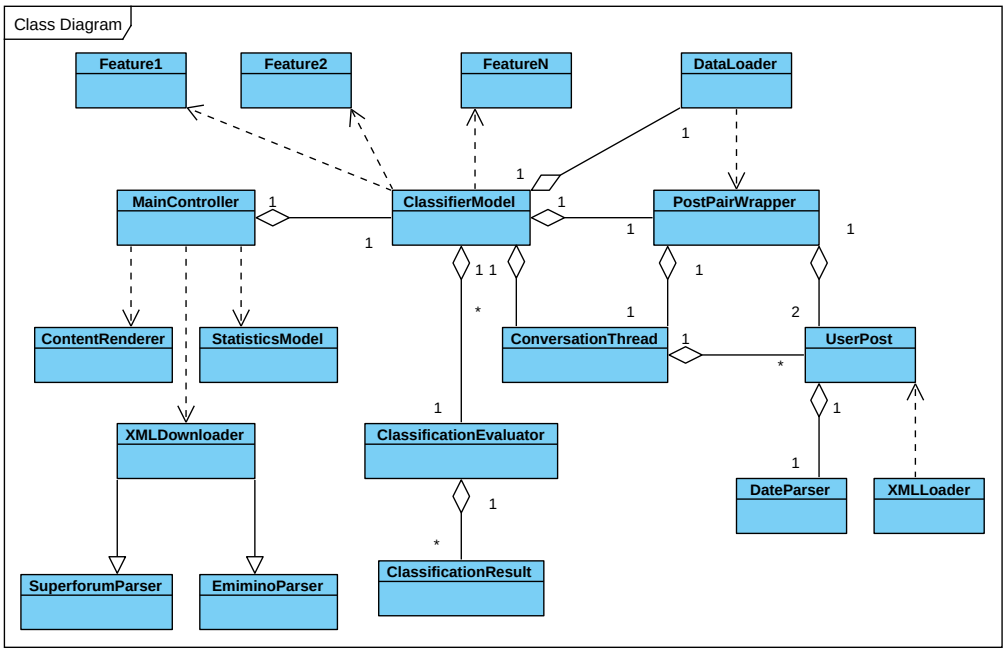
Po tom, co se zpracují všechny texty celého konverzačního fóra, tyto texty budou seřazeny do jednotlivých vláken podle toho, jaký text na který odpovídal. Takto seřazené texty budou ve výsledku vizualizovány do stromové struktury.

Pro implementaci této metody klasifikace dvojic krátkých textů metodou učení s učitelem jsem zvolil klasifikátor maximální entropie, která podle dosažených výsledků dosahovala nejlepšího poměru rychlosti a efektivity. Klasifikátor bude binární, bude mít tedy pouze dvě výstupní třídy. Třída 1 značí souvislost mezi dvěma texty a třída 0 značí, že spolu texty nesouvisí. [10]

5.3.1 Použité knihovny

- **HPS** (*High Precision Stemmer*) – Stemmovací nástroj založený na učení bez učitele z neoznačeného korpusu dat. [4]
- **Brainy** – Knihovna pro úlohy strojového učení implementovaná v programovacím jazyce JAVA. Umožňuje také extrakci a trénování příznaků (features). Podrobný popis je v sekci 5.3.8 [7].
- **Apache Commons** – Obsahuje různé užitečné metody, usnadňující například práci s textem
- **JSoup** – Knihovna pro Javu poskytující programovací rozhraní API pro extrakci a manipulaci dat z jazyka HTML.

5.3.2 UML diagram



Obrázek 5.1: Zjednodušený UML diagram tříd

5.3.3 Extrakce HTML

K získání množiny dat je třeba z internetové stránky s diskuzí extrahovat HTML tagy a uložit je do stanoveného formátu XML. Pro to byla použita knihovna **JSoup**, která pomocí metod programovacího rozhraní dokáže převádět HTML elementy na objekty v javě. V programu toto zajišťuje rodičovská třída `HTMLDownloader`. Úkolem této třídy tedy je extrakce informací z HTML stránky a převedení do paměti aplikace. Zároveň se třída stará o uložení načtených dat do formátu XML.

Třída `HTMLDownloader` obsahuje obecné metody

- *createDocBuilder()* – privátní metoda volaná konstruktorem třídy, která vytvoří instanci třídy `javax.xml.parsers.DocumentBuilder`
- *connectToUrl()* – veřejná metoda zajišťující připojení k URL adrese a vytvoření instance třídy `org.jsoup.odes.Document` pomocí příkazu

  ```
  Document doc = Jsoup.connect(url).get();
  ```

- *saveXML()* – veřejná metoda, která uloží již vytvořený XML dokument na místo na disku.

- *parse()* – abstraktní metoda, kterou musí potomkové dědící od této třídy implementovat.

Protože každé fórum má odlišnou strukturu, je velice obtížné vytvoření obecného nástroje, který by dokázal extrahovat informace ze všech webových stránek. Proto je pro každý zdroj dat vytvořena samostatná třída dědící od třídy `HTMLDownloader`.

5.3.4 Načtení XML

XML soubory, které byly staženy a uloženy na disk, je ve chvíli zahájení klasifikace nutno načíst do paměti programu. Pro načtení těchto dat je použita třída `XMLLoader`. Jejím úkolem je načtení všech XML souborů ve specifikovaném adresáři a následné vytvoření datových objektů, se kterými se bude v průběhu klasifikace pracovat. Používané datové objekty jsou popsány v sekci níže.

Třída `XMLLoader` je implementována pomocí návrhového vzoru Singleton (Jedináček), za běhu programu je tedy zaručeno, že bude vytvořena pouze jedna instance této třídy, přístupná pouze pomocí metody `getInstance()`.

Hlavní metody této třídy jsou

- *loadConversationThreadsFromXml()* – Jejím úkolem je načtení všech příspěvků uložených v XML souborech. Pro každý XML soubor vytvoří instanci třídy ConversationThread a naplní ji odpovídajícími příspěvky. Část 5.2 ukazuje algoritmus načítání XML souboru.

```
1  List<ConversationThread> conversationThreads =
2    new ArrayList<ConversationThread>();
3
4  DocumentBuilder docBuilder = getDocumentBuilder();
5  for (File xmlFile : directoryListing) {
6    if (!getFileExtension(xmlFile).equals("xml"))
7      throw new XMLParsingError("File" + xmlFile.getName() +
           "␣has␣wrong␣extension.");
8
9    Document doc = docBuilder.parse(xmlFile);
10   doc.getDocumentElement().normalize();
11
12   NodeList node = doc.getElementsByTagName("thread");
13   Node s = node.item(0).getAttributes().getNamedItem("name
       ");
14   String site = (s != null) ? s.getNodeValue() : "";
15
16   NodeList postTags = doc.getElementsByTagName("post");
17
18   ConversationThread thread = new ConversationThread(site)
       ;
19   for (int i = 0; i < postTags.getLength(); i++) {
20     Node post = postTags.item(i);
21     if (post.getNodeType() == Node.ELEMENT_NODE) {
22       thread.addUserPost(getUserPostFromNode(post));
23     }
```

```
24      }
25      conversationThreads.add(thread);
26  }
```

Listing 5.2: Načtení konverzačních vláken z XML souboru

5.3.5 Datové objekty

Protože dat je veliké množství, je nutné je nějak v programu reprezentovat. V programu jsou tři typy datových objektů : `UserPost`, `ConversationThread`, `PostPairWrapper`.

Třída `UserPost` obsahuje informace o jednom uživatelském příspěvku. Obsahuje hlavní položky

- *id* – identifikátor příspěvku
- *author* – jméno autora příspěvku
- *postContent* – obsahují původní, nezměněný text příspěvku
- *timestamp* – časová značka, kdy byl příspěvek vložen do diskuze
- *tokens* – seznam tokenů sestavený z textu příspěvku. Neobsahuje speciální znaky, všechna slova jsou převedena na malá písmena a je odstraněna diakritika.
- parentIdsLabeled – seznam identifikátorů příspěvků, na který tento příspěvek odpovídá. Nastavený pouze u označených dat.

V konstruktoru dochází k volání metody `validate()`, která kontroluje, zda načtená data příspěvku jsou validní. V případě, že ne, vypíše se chybová hláška. Ve výsledku se načítání daného příspěvku jednoduše přeskočí a pokračuje se dále. Poté je ihned zpracován text příspěvku, převedením na normalizovaný text a následně je rozdělen na seznam tokenů.

Třída `ConversationThread` reprezentuje jedno konverzační vlákno, obsahující množinu příspěvků s ním spjatých. Jeho položky jsou

- `id` – Identifikační číslo konverzačního vlákna
- `website` – Název stránky, ze které je vlákno staženo
- `userPosts` – Seznam všech uživatelských příspěvků ve vláknu

Posledním datovým objektem je třída `PostPairWrapper`, což je obalová třída dvojice příspěvků, která je určena ke klasifikaci.

- `originThread` – Vlákno, ze kterého dvojice příspěvků vychází
- `post1` a `post2` – Reference na oba příspěvky
- `classificationResult` – Výsledná třída pro klasifikaci určená klasifikátorem

5.3.6 Párování příspěvků

Vstupem klasifikátoru jsou vždy dvojice příspěvků. Je tedy nutné nejdříve jednotlivé příspěvky spárovat. V nejlepším případě potřebujeme získat takové dvojice, u kterých je velká pravděpodobnost, že budou odpovědí jeden na druhého.

Pokud bychom spárovali každý příspěvek se všemi předchozími, vznikla by naprosto nevyvážená množina dat, kde by mezi většinou dvojic neexistovalo spojení. Je tedy třeba určit hloubku, do které budou příspěvky spojovány. Na jednu stranu nechceme nevyvážená data ale zároveň potřebujeme nějaké dvojice, mezi kterými existuje spojení.

Nakonec byla implementována varianta, kdy každý příspěvek je spárován s pěti předchozími příspěvky a zároveň se zakládajícím (prvním) příspěvkem. Podle statistik trénovacího korpusu je velice časté, že na první příspěvek reaguje nejvíce uživatelů, proto je spojen s každým příspěvkem.

V části 5.3 je vidět algoritmus párování příspěvků.

```
1  List<PostPairWrapper> postPairs = new ArrayList<
       PostPairWrapper>();
2  for (ConversationThread thread : conversationThreads) {
3    int currentPostIndex = 0;
4    for (UserPost post : thread.getUserPosts()) {
5      int relativePostDiffDepth = 1;  // Relativni
         vzdalenost od prave zpracovavaneho prispevku
6      while (relativePostDiffDepth <= MAX_POST_PAIRING_DEPTH
         ) {
7        if ((currentPostIndex - relativePostDiffDepth) < 0)
           break;
8
9        UserPost nthPreviousPost =
10       thread.getUserPostAtPosition(currentPostIndex -
           relativePostDiffDepth);
11
12       PostPairWrapper postPair = new PostPairWrapper(post,
           nthPreviousPost, thread);
13       postPairs.add(postPair);
14
15       relativePostDiffDepth++;
16     }
17     currentPostIndex++;
18   }
19 }
```

Listing 5.3: Způsob párování příspěvků mezi sebou

5.3.7 Vláknové zpracování

Grafické rozhraní v JavaFX běží ve vláknu zvaném JavaFX Application thread (JavaFX aplikační vlákno). Program provádí časově náročné úkoly, není tedy dobrý nápad dlouhotrvající úlohy provádět v UI vláknu, protože to nevyhnutelně způsobí zamrznutí uživatelského rozhraní a program nebude reagovat na uživatelův vstup. To je důvod, proč jsou tyto úlohy prováděny na samostatných vláknech. Implementace je provedena pomocí třídy `Task` a přepisem její metody `call`. Pro to, aby se úlohy provedly ve správném pořad jsou přidány do rozhraní `ExecutorService` z balíku `java.util.concurrent`. Díky tomu je také možné sledovat průběh úlohy a dát o tom informaci uživateli v podobě ukazatele `ProgressBar`. V případě fatální chyby v jakékoliv úloze se naplánované úlohy zruší a uživatel je o chybě informován.

Všechny úlohy prováděné ve vlastním vláknu jsou spuštěné ve třídě `ClassifierModel`. Při startu aplikace probíhá načítání jazykového modelu pro stemmer. Pokud je k dispozici adresář s trénovacími soubory, ihned se spustí úloha trénování klasifikátoru `trainingTask()`. Zajistí se přidání všech příznaků, načtou se data ze všech XML souborů a spustí se proces trénování. U úlohy klasifikace jsou obdobně načítána data, avšak z adresáře, kde jsou uložena testovací data.

Poslední úlohou je `crossValidationTask()`. Ta zajišťuje provedení cross validace na trénovacích datech. přijímá čtyři parametry

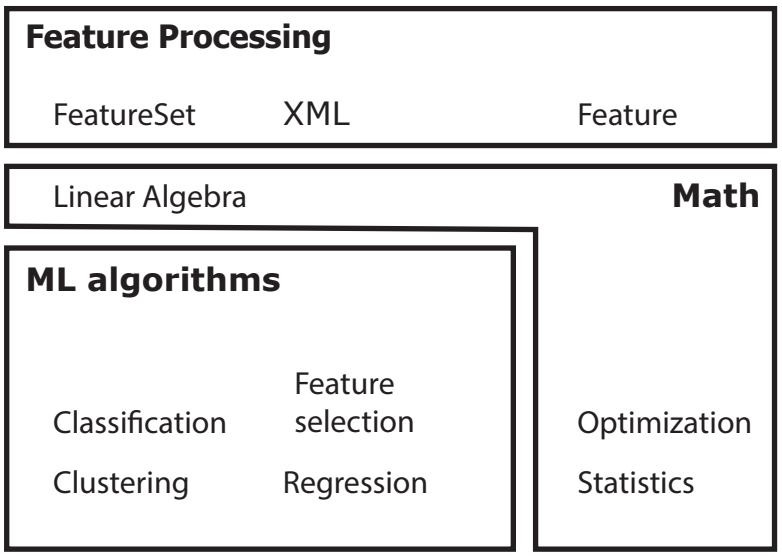
- File[] files – Množina souborů, ze kterých bude cross validace prováděna
- int numOfIterations – Počet iterací cross validace
- double trainingFilesPercentage – Procentuální zastoupení trénovacích dat proti testovacím
- boolean randomize – Pokud je nastavena na nepravda (false), pro trénovací a testovací soubory budou použity stále stejné soubory, aby bylo možné porovnávat výsledky. Pokud je hodnota pravda (true), náhodně se změní množina trénovacích a testovacích souborů.

5.3.8 Knihovna Brainy

Samotné algoritmy klasifikace zajišťuje knihovna Brainy [7]. Obsahuje tři základní části.

- Struktury lineární algebry a algoritmy strojového učení

- Optimalizační algoritmy
- Komponenty zpracování příznaků



Obrázek 5.2: Komponenty knihovny Brainy

Před trénováním klasifikátoru musí být vytvořeny třídy implementující rozhraní `Feature<T>`, kde T je nahrazeno objektem, pro který chceme příznaky extrahovat. Některé příznaky musí být nejdříve natrénovány. K tomu slouží metoda `train`. Ta je volána vždy před procesem extrakce.

Dalším krokem je příprava dat ke trénování. Uživatelská data jsou reprezentována pomocí rozhraní `TrainingInstanceList`.

```
1 BasicTrainingInstanceList<PostPairWrapper> instances =
2   new BasicTrainingInstanceList<PostPairWrapper>(
      trainingDataPairs, null, labels, NUM_OF_LABELS);
```

Listing 5.4: Ukázka přípravy dat

Nyní je možné natrénovat sadu příznaků (feature set) pomocí příkazu `featureSet.train(instances);`

Poté, co jsou připravena data i sada příznaků, můžeme natrénovat klasifikátor.

```
1 DoubleMatrix data = featureSet.getData(instances);
2 IntVector trainingLabels = featureSet.getLabels(instances)
    ;
3 SupervisedClassifierTrainer trainer = new MaxEntTrainer();
4 Classifier = trainer.train(data, trainingLabels,
    NUM_OF_LABELS);
```

Listing 5.5: Ukázka trénování klasifikátoru

Jako poslední proběhne samotná klasifikace

```
1 BasicInstanceList<PostPairWrapper> instances = new
     BasicInstanceList<PostPairWrapper>(testData, null);
2     DoubleMatrix data = featureSet.getData(instances);
3     BasicClassificationResults results =
         BasicClassificationResults.create(NUM_OF_LABELS,
         data.columns());
4     classifier.classify(data, results);
5     IntVector labels = results.getLabels();
```

Listing 5.6: Ukázka klasifikace a získání výsledných tříd

5.3.9 Příznaky – Features

Pro implementaci byl použit příznakový typ klasifikátoru. Příznakové klasifikátory využívají pro svůj vstup data, která jsou vyjádřena vektorem hodnot jednotlivých proměnných (příznaků - features). Tyto příznaky slouží k redukci dimenzionality. [11]

Identifikace takových příznaků bude zásadní pro výslednou efektivitu klasifikátoru (*features*). Předem víme, že texty, se kterými budeme pracovat, jsou většinou krátké úseky textu (průměrně 160 znaků na příspěvek).

Příznaková data jsou většinou mnohorozměrná. Při přípravě je třeba tato data co nejvíce dekomponovat abychom je lépe znormalizovali a aby byly pochopitelné pro algoritmy strojového učení. Toho nejlépe docílíme tím, že složité, komplexní příznaky dekomponujeme do několika jednodušších. Tímto docílíme snížení závislostí do jednodušších nezávislých souvislostí.

Např. mějme atribut s reálnou hodnotou od 0 do 1000. Můžeme vytvořit 10 binárních atributů, kde každý reprezentuje rozmezí hodnot (0-99 pro první, 100-199 pro druhou atd.) a každé hodnotě přiřadíme binární hodnotu 0 nebo 1 podle toho, do jakého intervalu hodnota spadá.

Příznaky implementované v této aplikaci se dají rozdělit do několika kategorií.

Obsahové příznaky

První kategorie příznaků se týká obsahu textu. Příspěvky, které spolu souvisejí budou pravděpodobně používat stejná nebo podobná slova. Autoři příspěvků často v textu uvedou jméno autora, na jehož příspěvek chtějí odpovědět. Někteří uživatelé místo zmínky autora na kterého reagují, použijí citovaný text, což je také dobrým identifikátorem souvislosti.

Strukturní příznaky

Druhá kategorie se týká strukturních znaků každého příspěvku. Patří sem například délka příspěvku. Dalším znamením může být absolutní pozice příspěvku ve vláknu. Pokud jeden uživatel popsal nějaký problém nebo dotaz, následující příspěvky pravděpodobně na tento problém budou reagovat. Podobně je také možné, že příspěvky, které na někoho reagují, budou delší než ty, které se na něco dotazují, protože něco vysvětlují, popisují. Proto je délka příspěvku dalším příznakem. Ve většině případů také platí, že na první, zakládající příspěvek reaguje nejvíce lidí.

Níže jsou popsány příznaky, které byly použity. V kapitole 5.4 budou rozebrány dosažené výsledky při použití různých kombinací příznaků.

Jméno příznaku	**Popis**
Obsahové příznaky	
Quote	Výskyt citace
AuthorReference	Reference autora
SiteName	Název fóra, ze kterého pochází příspěvky
JaccardSim	Jaccardova nevážená podobnost mezi dvojicí příspěvků
N-gram	Seznam n-gramů vyskytující se v trénovacím korpusu
TF-IDF	Podobnost textů
TimeDifference	Časový rozdíl mezi dvěma příspěvky
Strukturní příznaky	
PostLength	Rozdíl počtu slov mezi dvěma příspěvky.
PostPosition	Absolutní pozice příspěvků ve vláknu.
PostDistance	Vzdálenost mezi dvojicí příspěvků.
Příznaky sentimentu	
Thank	Výskyt slov díku v příspěvku

Tabulka 5.2: Popis hlavních příznaků sloužících ke klasifikaci

Výskyt citace

Pokud se v jednom příspěvku vyskytuje citace nějakého jiného příspěvku, můžeme to považovat jako dobrý indikátor souvislosti. Často se stává, že pokud uživatel chce reagovat na příspěvek jiného uživatele, vloží do svého příspěvku citovaný text, na který odpovídá. Pokud se takový případ objeví,

podobnost textů by měla být vysoká a dvojice tak označena jako související. Problém nastává, pokud daný portál obsahující diskuzi podporuje funkci reakce na příspěvek (Facebook). V takovém případě se citace téměř nevyskytuje, protože reagující příspěvky jsou automaticky odsazeny a vloženy za příspěvek, na který reagují.

```
<post>
      <id>59</id>
      <replyTo>0</replyTo>
      <timeStamp> 6. 2. 2008 08:43 </timeStamp>
      <author>mardon</author>
      <text>Jeste se zeptam CASH ADVANCE je
      mozne na vsech pobockach KB?
      Nebude problem pri tak vysoke castce?
      Plati se nejake poplatky?
      </text>
</post>
<post>
      <id>62</id>
      <replyTo>59</replyTo>
      <timeStamp> 6. 2. 2008 09:39 </timeStamp>
      <author>Rop</author>
      <text>Jeste se zeptam CASH ADVANCE
      je mozne na vsech pobockach KB?
      Nebude problem pri tak vysoke castce?
      Plati se nejake poplatky??/quote]
      Cash advance je zalezitost konkretni pobocky.
      U~mBank je to zdarma,
      ale musis najit banku smenarnu s~elektronickym
      terminalem a tech moc neni</text>
</post>
```

Listing 5.7: Ukázka citace v textu

Ve skutečnosti se však ukázalo, že v trénovacích datech se citace vyskytovaly velmi zřídka, proto tento příznak nevykazoval velkou úspěšnost.

Výskyt reference autora

Pokud se v textu nevyskytuje citace, může autor příspěvku dát najevo, že odpovídá na jiný příspěvek tím, že na začátku zmíní jméno autora. To je jasný znak toho, že dotyčný příspěvek reaguje na jiný.

```
<post>
      <id>29</id>
      <replyTo>28</replyTo>
      <timeStamp> 30. 11. 2008 19:47 </timeStamp>
      <author>Misena</author>
      <text>Jak Ti nekdo vylepsil tvou smlouvu pred par
         lety?
      Budes Cc zvysovat aspon na 10 milionu?:-)</text>
</post>
<post>
      <id>30</id>
      <replyTo>29</replyTo>
      <timeStamp> 30. 11. 2008 21:02 </timeStamp>
      <author>S474N</author>
      <text>Misena: nejedna se o~mou smlouvu,
      nevim kde jsi tuto informaci vycetl.
      BTW i kdyby se zvysovala smlouva na 10 milionu,
      tak co vim, je limit u~provize stanoven o~dost nize
```

```
18          </text>
19  </post>
```

Listing 5.8: Ukázka reference autora. Ve spodním příspěvku autor na začátku zmínil jméno autora, na kterého reaguje.

Absolutní pozice příspěvku

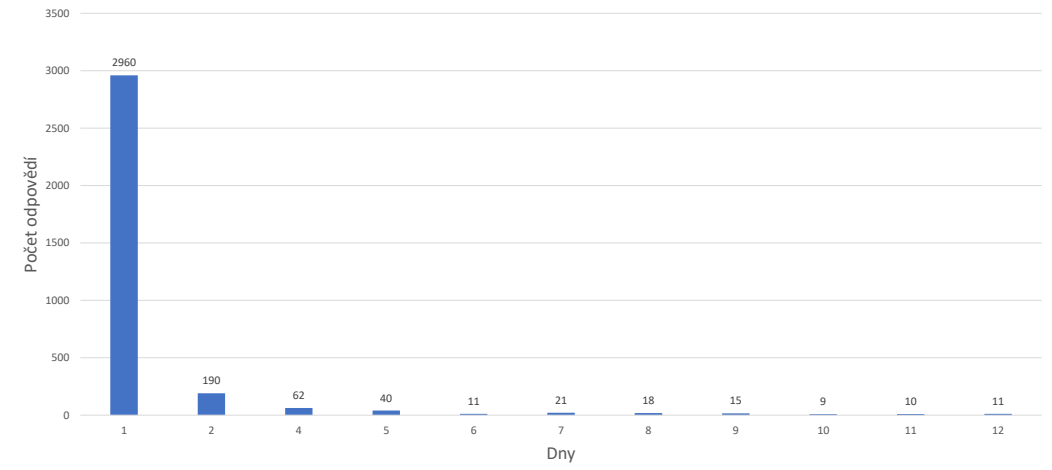
Určuje, na jaké pozici se daný příspěvek ve fóru nachází. Většinou je nejvíce reakcí na první příspěvek.

Časová vzdálenost příspěvků

Dalším identifikátorem toho, že příspěvky spolu souvisejí, může být časový údaj, který je součástí každého příspěvku.

Převod data z textového formátu na objekt třídy `Date` má na starosti třída `DateParser`. Ta ze dvou časových údajů dokáže vypočíst jejich rozdíl a převést ho na počet dnů. Tento údaj tedy udává časový rozdíl mezi dvěma příspěvky.

Ze statistiky v grafu 5.3 je vidět, že naprostá většina uživatelů reaguje během jednoho dne.



Obrázek 5.3: Statistika reakcí na příspěvky v závislosti na čase.

Vzdálenost příspěvků

Dalším indikátorem je příznak určující vzdálenost dvou příspěvků od sebe, tj. počet příspěvků, nacházejících se mezi dvěma zkoumanými příspěvky. Z vy-

tvořených trénovacích dat bylo zjištěno, že 42 procent příspěvků odpovídá na příspěvek předchozí.

N-gramy

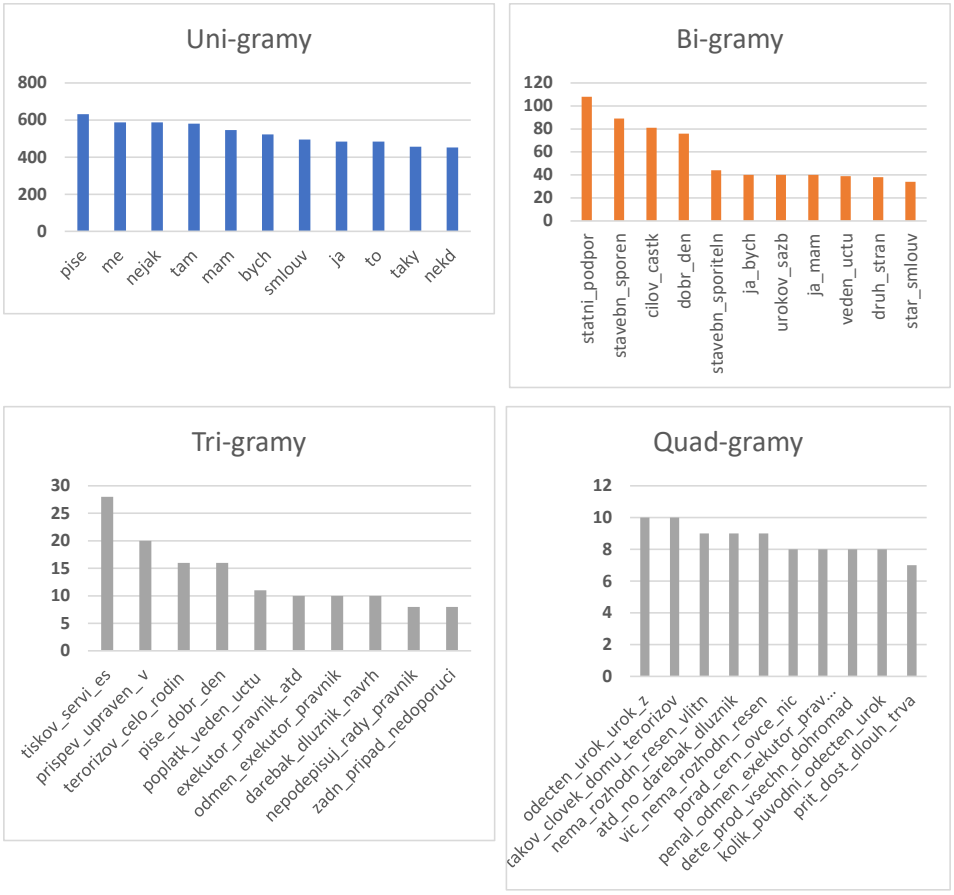
N-gramy jsou příznakem, ke kterému je potřeba natrénování. Část 5.9 ukazuje algoritmus vytvoření množiny n-gramů.

```
HashMap<String, Integer> counts = new HashMap<String,
    Integer>();
ListIterator<PostPairWrapper> iterator = instanceList.
    iterator();
// Prochazime vsechny dvojice prispevku
while(iterator.hasNext()) {
  PostPairWrapper postPair = iterator.next();
  // Ziskame tokeny jednoho prispevku z dvojice
  List<String> tokens = postPair.getNewerPost().getTokens
      ();
  if (tokens == null ) throw new IllegalStateException("No
      tokens");

  for (int i = 0; i < tokens.size(); i++) {
    StringBuilder ngram = new StringBuilder(tokens.get(i))
        ;

    for (int j = 1; j < ngramSize; j++) {
      if (i+j >= tokens.size()) break;
      ngram.append("_").append(tokens.get(i + j));
    }
    // Pridame N-gram do hash mapy, pokud uz v ni existuje
    // inkrementujeme pocet vyskytu, jinak vytvorime novy
        zaznam
    // a nastavime na 1
    counts.merge(ngram.toString(), 1, (a, b) -> a + b);
  }
}

wordMap = new HashMap<String, Integer>();
int index = 0;
for (String record : counts.keySet()) {
  int count = counts.get(record);
  // Pokud je pocet vyskytu n-gramu vyssi nez threshold,
  // pridame ho do vysledne mapy slov
  if (count > threshold) {
    wordMap.put(record, index);
    index++;
  }
}
```

Listing 5.9: Ukázka trénování příznaku N-gram

Prochází se všechny klasifikované dvojice příspěvků. Z jednoho příspěvku ze dvojice se získá normalizovaný text v podobě seznamu tokenů. Postupně se vytváří hash mapa všech n-gramů a jejich počet výskytů. Dále se tato hash mapa prochází a ty položky, které se nachází vícekrát než je hodnota proměnné *threshold*, se přidají do výsledného seznamu.

V grafech na obrázku 5.4 je vidět statistika n-gramů nacházejících se v trénovacím korpusu.



Obrázek 5.4: Statistika N-gramů z trénovacího korpusu

Podobnost příspěvků

Zkoumání podobnosti dvou textů bylo implementováno pomocí výpočtu vektoru vah pomocí TF-IDF (2.3.1) s následným výpočtem kosinové vzdálenosti (2.4.1).

Jméno stránky

Data jsou stažena ze dvou diskuzních fór, na začátku každého XML souboru je definováno, odkud je fórum staženo.

Výskyt slov sentimentu

Pokud se v příspěvku vyskytují slova jako *dekuji, dik, souhlasim, nesouhlasim, opravuji, jeste doplnuji*, je šance, že tento příspěvek je reaguje na jiný

5.3.10 Zpracování výsledků

Po proběhnutí klasifikace jsou vráceny výsledky v podobě výsledných tříd pro klasifikaci. Tyto výsledky jsou předány třídě `ClassifierEvaluator`, která výsledky zpracovává a následně vypisuje na obrazovku.

Jejím úkolem je přiřazení klasifikovaných tříd jednotlivým dvojicím příspěvků, vypočtení hodnot TP, FP, TN, FN (True positive, false positive, true negative, false negative, viz. kapitola 4) a získání výsledků o vyhodnocení úspěšnosti klasifikace. Třída `ClassifierEvaluator` spolupracuje se třídou `ClassificationResult`, která obsahuje všechny informace potřebné k vyhodnocení klasifikace. Obsahuje metody

- `getCalculatedPrecision()` – Vypočte přesnost (viz. 4.1)
- `getCalculatedRecall()` – Vypočte Úplnost (viz. 4.2)
- `getCalculatedFScore()` – Vypočte F-score (viz. 4.3)
- `getObservedAccuracy()` – Vypočte pozorovanou přesnost (viz. 4.5)
- `getExpectedAccuracy()` – Vypočte očekávanou přesnost (viz. 4.5)
- `getKappa()` – Vypočte kappa statistiku pomocí pozorované a očekávané přesnosti.
- `getFPR()` – Vypočte podíl falešných negativů (False positive rate)

5.3.11 Grafický výstup

Jak již bylo popsáno, aplikace je naprogramována v Javě pomocí frameworku JavaFX.

Popis grafického uživatelského rozhraní se nachází v následujících souborech

- `MainWindow.fxml`
- `Error.fxml`
- `ContentRenderer.java`

MainWindow.fxml

Obsahuje hlavní okno aplikace. Hlavní okno používá rozvržení

Error.fxml

V případě jakékoliv závažné chyby v aplikaci se zobrazí překrývající okno s názvem a popisem nastalé chyby.

ContentRenderer.java

Třída, která zajišťuje vykreslení stromu konverzačních vláken na obrazovku uživatele.

Obsahuje metodu `getHTML()`, která má za úkol vygenerování HTML kódu popisujícího vykreslení.

Dále obsahuje dvě metody které se starají o různý způsob vykreslení. Metoda `renderByThread` vykreslí strom konverzací podle vláken a metoda `renderByDate` ho vykreslí podle data.

Způsob vygenerování stromu konverzací

Informace o spojených příspěvcích je potřeba nějak strukturovaně uchovávat v paměti. Příspěvky, které jsou spojené s jinými, na ně ve své třídě obsahují referenci v podobě instanční proměnné. Dohromady se tvoří acyklický orientovaný graf (strom). K procházení všech vrcholů tohoto grafu a následné grafické vykreslení je použit algoritmus prohledávání do hloubky (*DFS - Depth first search*).

Níže v sekci 5.10 je vidět ukázka algoritmu *DFS*.

```
1  private String dfs(UserPost initialPost,
       ConversationThread thread) {
2    StringBuilder html = new StringBuilder();
3
4    boolean []visited = new boolean[thread.getUserPosts().
       size()];
5    Stack<UserPost> st = new Stack<>();
6    st.push(initialPost);
7    int level; // Uroven odsazeni prispevku
8    while (!st.empty()) {
9      UserPost curPost = st.pop();
10     int indexOfPost = thread.getUserPosts().indexOf(
         curPost);
11
12     if (visitedPosts.contains(curPost)) continue;
13     visitedPosts.add(curPost);
14
15     if (!visited[indexOfPost]) { //pokud nebyl vrchol
         navstiven
16       visited[indexOfPost] = true;
17       level = curPost.getDepth();
18       // Vykresleni jednoho prispevku
19       renderSinglePost(html, curPost, level);
```

```
20          Stack<UserPost> auxStack = new Stack<>();
21          for (UserPost neighborPost : curPost.getReplies()) {
22            indexOfPost = thread.getUserPosts().indexOf(
                  neighborPost);
23            if (!visited[indexOfPost]) {
24              auxStack.push(neighborPost);
25            }
26          }
27          while (!auxStack.isEmpty()) {
28            st.push(auxStack.pop());
29          }
30        }
31      }
32      return html.toString();
33  }
```

Listing 5.10: Prohledávání grafu do hloubky

5.4 Výsledky klasifikace

Dalším z cílů této práce je otestování implementované metody klasifikace.

Výsledky byly prováděny s různým nastavením všech parametrů a pomocí různých klasifikátorů. Možností nastavení těchto parametrů je veliké množství, proto zde budou popsány pouze ty, kde se dosáhlo nejlepších výsledků. Za nejlepší výsledek byl považován ten, který dosahoval nejlepšího poměru úplnosti a přesnosti, tedy výsledného F-score.

5.4.1 Testovací množina

Všechna testování byla prováděna pomocí 10-iterační cross-validace s poměrem trénovacích a testovacích dat 90/10. Celkový počet dvojic příspěvků byl 22755. Následně byla vypočtena průměrná hodnota z těchto iterací.

Pozn.: Kvůli způsobu párování příspěvků, kdy každý příspěvek je spárován s n předchozími, vzniká nevyvážená množina dat, kdy mezi velkou částí dvojic příspěvků není spojení. Tím zdánlivě dochází k velké úspěšnosti klasifikátoru, ale to je ovlivněno právě velkým množstvím nespojených příspěvků. Následující výsledky jsou tedy brány pouze vzhledem k třídě 1, tedy kdy klasifikátor označil existující spojení.

V tabulce 5.3 vidíme seznam příznaků a jejich výsledné hodnoty. Každý příznak obsahuje hodnoty při zapnutí všech vyšších příznaků

Příznak	Přesnost	Úplnost	F-score
AuthorReference	0.707	0.481	0.564
PostPosition	0.543	0.624	0.571
PostDistance	0.565	0.65	0.589
SiteName	0.554	0.659	0.590
PostLength	0.554	0.664	0.595
N-Gram	0.556	0.662	0.596
CosSim	0.547	0.670	0.594
Quote	0.569	0.636	0.588
TimeDiff	**0.570**	**0.663**	**0.604**

Tabulka 5.3: Různé výsledky klasifikace při zapnutí různých příznaků. Uvedené hodnoty pro každý příznak uvádějí hodnotu, pokud jsou všechny vyšší příznaky zapnuté

Tabulka 5.4 udává výsledky vytvořeného klasifikačního modelu za použití klasifikátoru maximální entropie. Dosáhli jsme výsledné přesnosti 0.578, úplnost byla naměřena 0.663 a hodnota f-score 0.604.

Přesnost	0.57
Úplnost	0.663
F-score	0.604

Tabulka 5.4: Výsledky klasifikace pozitivní třídy při použití klasifikátoru maximální entropie

Jak je vidět z tabulky 5.5, při použití klasifikátoru SVM došlo ke zvýšení přesnosti klasifikace o 0.14, úplnost však byla nižší o 0.37. Natrénování bylo však ve srovnání s klasifikátorem maximální entropie výrazně pomalejší.

Přesnost	0.584
Úplnost	0.626
F-score	0.589

Tabulka 5.5: Výsledky klasifikace při použití klasifikátoru SVM

5.4.2 Ukázka vytvořeného stromu konverzace

Níže je pro ukázku zobrazen strom konverzací vygenerovaný programem.

> **chrudimi** [Tue Nov 25 13:32:00 CET 2014] Pozor na změnu dodržování VOP ze strany O2, pokud Vám vyprší platnost kreditu, zbývající

kredit propadne a nevrátí se ani dalším dobitím! Prý to takto v podmínkách už bylo a jen to od 1.9.2014 začali uplatňovat, hezky potichu, aby se to snad nerozkřiklo. Měl jsem v plánu převést jednu manželky simku na Kaktus a díky této vyfikundaci jsem musel dobít dalších 200 Kč, jelikož nelze převést sim s nulovým kreditem. Přijde mi to jako úmysl, aby se v těchto případech zabránilo útěku ke konkurenci.

> **GymmiCZ** [Tue Nov 25 13:41:00 CET 2014] Celkem se bojím, že na to přistoupí T-Mobile, Vodafone a virtuální operátoři.
>
> **Reku** [Tue Nov 25 14:02:00 CET 2014] A není nemožnost převést simku s nulovým kreditem k jinému operátorovi v rozporu s nějakým nařízením ČTÚ? Protože převodem simky s nezáporným kreditem žádná škoda nevzniká! Kaktus umožňuje přenést i simku s nulovým kreditem. Tak proč darebáci z O2 ne?
>
> **chrudimi** [Tue Nov 25 14:08:00 CET 2014] Převést nelze, protože O2 mi s nulovým kreditem nedá ČVOP pro přenos čísla. Nebo to u Kaktusu funguje jinak?

redfox1 [Tue Nov 25 14:40:00 CET 2014] Možná už to tu zaznělo, ale pokud ne - O2 nyní nevrací propadlý kredit při opětovném dobití, pokud zmeškáte lhůtu 6 měsíců od posledního dobití. Prý nyní "důsledně dodržují smluvní podmínky". Hodně lidí už takto přišlo o peníze, tak si dejte pozor...

chrudimi [Tue Nov 25 15:24:00 CET 2014] To je dobré vědět, ale neřeší to moji situaci, kdy potřebuji převést O2 -> Kaktus, O2 mi ČVOP s nulovým kreditem nedá a to je právě ta finta, na kterou jsem narážel v prvním příspěvku.

> **Reku** [Tue Nov 25 15:34:00 CET 2014] chrudimi - Mě je to jasné! Chtěl jsem jen ukázat, že jinde to jde jinak a O2 je čím dál větší darebák. Platbost kreditu je jen půl roku, pkud nedobiju na 500 a ještě UKRADNOU a nevrátí kredit při dalším dobití a ještě neumožní převést číslo! Fuj fuj O2 !
>
> > **chrudimi** [Tue Nov 25 15:57:00 CET 2014] moje chyba, trochu nepochopení původního příspěvku, těď už je to jasné, jsme na jedné lodi. to radim - O2 Zero už nelze pár měsíců nově založit, přestali ho nabízet zhruba v době, kdy Vodafone "zničil"můj oblíbený tarif Odepiš. Proto jsem tedy volil mezi nulovým paušálem od ČEZ a Kaktusem, jinak bych asi převedl na O2 Zero, abych nemusel řešit převod čísla.

sestraVR6 [Tue Nov 25 18:02:00 CET 2014] Přesně tak potvrzuji, že jsem na předplacené kartě kterou využívám na záložní internet přišel o 380 kaček kupovaná ještě za errortelu !

ze.us [Tue Nov 25 18:20:00 CET 2014] Zero je stale k mani, minuly mesic me na nej sami zmigrovali z ruseneho T!P Relax.

milanBrno [Wed Nov 26 01:14:00 CET 2014] Pro všechny: Důvodem nepřenesení čísla nebyl nulovy kredit, jak píše (špatně) zakladatel vlakna, ale blokace sluzeb (teprve na jejim zaklade doslo k vynulovani kreditu). A blokovana karta (sluzby) nejde prenest u zadneho operatora.

Reku [Wed Nov 26 06:06:00 CET 2014] A jsi si tím milane zcela jistý, že v takovém případě nejde přenos u žádného operátora?

5.4.3 Porovnání

Pokud pro stejnou množinu dat, pro kterou byly získány výše popsané výsledky a místo klasifikátoru byl použit čistě náhodný výběr pro každou dvojici, získané výsledky jsou zobrazené v tabulce 5.4.3.

Přesnost	0.126
Úplnost	0.505
F-score	0.222

Tabulka 5.6: Výsledky klasifikace pozitivní třídy při použití náhodného výběru pro dvojici

V další tabulce vidíme porovnání, kdy bychom pro všechny klasifikované dvojice testovací množiny zvolili výstupní třídu 1.

Přesnost	0.145
Úplnost	1.0
F-score	0.253

Tabulka 5.7: Výsledky klasifikace při zvolení n třídy 1 pro všechny dvojice

5.4.4 Shrnutí výsledků

Implementovaný model byl otestován pouze na českých textech. Byla snaha o nalezení optimálního nastavení parametrů.

Jednoznačně nejlepším příznakem byla reference autora, která se také v trénovacím i testovacím korpusu vyskytovala často. Tento samotný příznak získal výsledek: přesnost 0.707 úplnost 0.481 F-score 0.564. Zapnutím dalších příznaků jsme dosáhli zvýšení úplnosti klasifikace o dalších 18.2%. Hodnota F-score se zvýšila o 4%.

Dalším dobrým příznakem byla absolutní pozice v konverzačním vláknu. To je způsobeno především tím, že mnoho příspěvků odpovídá na výchozí, zakládající příspěvek.

5.5 Testování subjektivity

Označování trénovacích dat je velice subjektivní a dva lidé často mají odlišné přístupy k rozhodování, zda jsou dva krátké příspěvky související. Úloha je o to těžší, že nejsou definována žádná pravidla, říkající, kdy je jeden příspěvek odpovědí na druhý. V důsledku toho mohou vznikat matoucí trénovací data, u kterých mezi spojenými příspěvky nejsou patrné žádné společné znaky a tedy klasifikátor neví, jak se je naučit.

V rámci otestování subjektivity byli tři nezávislí testeři, kteří měli za úkol označit poskytnutá data podle svého uvážení.

Velikost testovacího korpusu

Pro účely tohoto testu bylo shromážděny tři konverzační vlákna ze dvou internetových fór s celkovým počtem testovacího korpusu 120 příspěvků

- forum.zive.cz
- www.emimino.cz/diskuze/

	Shoda	Procentuálně	Kappa
Tester 1 a Tester 2	81	67.5%	0.612
Tester 1 a Tester 3	78	65%	0.597
Tester 2 a Tester 3	87	72.5%	0.634

Tabulka 5.8: Výsledky klasifikace při zvolení výstupní třídy 1 pro všechny dvojice

Z výsledků můžeme vidět, že subjektivní názor jednotlivých testerů není zanedbatelný. Procentuální shoda všech testerů je 68.3%.

6 Závěr

V této práci byl zkoumán problém klasifikace dvojic krátkých textů do jedné ze dvou tříd. Tuto metodu jsme implementovali a vyzkoušeli na dvou internetových fórech.

K řešenému problému nebyl k dispozici žádný trénovací korpus dat. První část tedy zahrnoval sběr a manuální označování dat, které byly použity ke trénování. Celkem bylo nashromážděno 3933 označených příspěvků.

Program dokáže stáhnout a extrahovat potřebné informace z HTML dokumentu, uložit ho do formátu XML. Z tohoto souboru je načte do paměti programu, provede klasifikaci načtených dat a výsledky graficky zobrazí uživateli na obrazovku. Program dále umožňuje provedení cross-validace na trénovacím korpusu a výpis výsledků.

Pro implementaci byl použit programovací jazyk JAVA. Ke tvorbě GUI byl použit framework *JavaFX* umožňující definici uživatelského rozhraní v samostatných souborech *FXML*. Je zde tedy dodrženo oddělení prezentační části od části logické. Pro extrakci informací z HTML byla použita knihovna *JSoup*. Samotnou klasifikaci zajišťovala knihovna *Brainy*.

Implementovaná metoda byla v konečné fázi otestována. Byla vypočtena úspěšnost klasifikátoru a jako nejlepší byl vybrán klasifikátor maximální entropie s nejvyšší naměřenou hodnotou F-score 0.663. Při použití SVM klasifikátoru byla hodnota F-score o 0.14 nižší. Zjistili jsme, že největší podíl na výsledku měl příznak reference autora a absolutní pozice příspěvku ve vláknu. V porovnání s náhodným výběrem se zvedla hodnota f-score o 0.382, úplnost o 0.121 a přesnost o 0.444.

Nakonec byl proveden experiment se třemi nezávislými testery, jejichž úkolem bylo označení poskytnutých data podle svého uvážení. Bylo zjištěno, že rozdíly mezi označením jednotlivých testerů jsou poměrně velké, s průměrnou shodou mezi testery 68.3%.

Výsledky programu splnily očekávání, je zde však rozhodně prostor pro vylepšení. K dosažení větší úspěšnosti klasifikace by pravděpodobně pomohlo nashromáždit více trénovacích dat. Dále by bylo vhodné rozšíření funkčnosti i na jiná internetová diskuzní fóra. Mohly by být také přidány další příznaky a ty stávající vylepšeny.

Literatura

[1] *Confusion Matrix – Another Single Value Metric – Kappa Statistic* [online]. 2011. [cit. 2017/02/27]. Dostupné z: http://standardwisdom.com/softwarejournal/2011/12/confusion-matrix-another-single-value-metric-kappa-statistic.

[2] *Cohen's Kappa Statistic for Measuring Agreement* [online]. 2016. [cit. 2017/04/23]. Dostupné z: https://onlinecourses.science.psu.edu/stat509/node/162.

[3] Altman, D. G. *Practical Statistics for Medical Research.* Chapman and Hall, 1991.

[4] Brychcín, T. – Konopík, M. HPS: High precision stemmer. *Information Processing & Management.* 2015, 51, 1, s. 68 – 91. ISSN 0306-4573. doi: http://dx.doi.org/10.1016/j.ipm.2014.08.006. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0306457314000843.

[5] Frank, I. H. W. . E. *Data Mining - Practical Machine Learning Tools and Techniques Second edition.* 3. Diane Cerra, 2005. ISBN 0-12-088407-0.

[6] Hai Leong Chieu, H. T. N. Named Entity Recognition: A Maximum Entropy Approach Using Global Information. Technical report, Department of Computer Science School of Computing, 2005.

[7] Konkol, M. Brainy: A Machine Learning Library. In Rutkowski, L. et al. (Ed.) *Artificial Intelligence and Soft Computing*, 8468 / *Lecture Notes in Computer Science*. "f": Springer International Publishing, 2014. s. 490–499. doi: 10.1007/978-3-319-07176-3_43. Dostupné z: http://dx.doi.org/10.1007/978-3-319-07176-3_43. ISBN 978-3-319-07175-6.

[8] M. Ikonomakis, V. T. S. K. *Text Classification Using Machine Learning Techniques* [online]. 2005. [cit. 2017/02/28]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.9153&rep=rep1&type=pdf.

[9] Mitchell, T. M. GENERATIVE AND DISCRIMINATIVE CLASSIFIERS: NAIVE BAYES AND REGRESSION. *Machine Learning.* 2015, s. 3.

[10] ONDERKA, J. NOVÉ METODY ZPRACOVÁNÍ TEXTU PRO KLASIFIKACI EMOCÍ.

https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=101912. 2015.

[11] Patočka, M. *Metody strojového učení pro analýzu sentimentu* [online]. 2013. [cit. 2017/03/01]. Dostupné z: `http://is.muni.cz/www/98951/41610771/43823411/43823458/Analyza_a_hodnoc/44563155/56270095/Vicerozmerky_-_kap_11.1_-_klasifikace_uvod.pdf`.

[12] Patočka, M. *Metody strojového učení pro analýzu sentimentu* [online]. 2013. [cit. 2016/01/04]. Dostupné z: `http://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf`.

[13] Phillips, J. M. *Jaccard Similarity and k-Grams* [online]. 2015. [cit. 2017/04/22]. Dostupné z: `https://www.cs.utah.edu/~jeffp/teaching/cs5140-S15/cs5140/L4-Jaccard+nGram.pdf`.

[14] Raymond J. Mooney, R. B. Mining Knowledge from Text Using Information Extraction. *http://www.cs.utexas.edu/ ml/papers/text-kddexplore-05.pdf*. 2013.

[15] Rish, I. An empirical study of the naive Bayes classifier. Technical report, T.J. Watson Research Center, 2010.

[16] Steven Bird, E. L. E. K. *Natural Language Processing with Python* [online]. 2014. [cit. 2017/02/28]. Dostupné z: `http://www.nltk.org/book/ch06.html`.

[17] Tuetschek. *Stochastické metody a jejich aplikace v počítačové lingvistice* [online]. 2010. [cit. 2010/10/24]. Dostupné z: `http://wiki.matfyz.cz/index.php?title=St%C3%A1tnice_I3:_Stochastick%C3%A9_metody_a_jejich_aplikace_v_po%C4%8D%C3%ADta%C4%8Dov%C3%A9_lingvistice`.

[18] Vryniotis, V. Machine Learning Tutorial: The Max Entropy Text Classifier. *http://blog.datumbox.com/machine-learning-tutorial-the-max-entropy-text-classifier/*. 2013.

[19] Zisserman, A. *The SVM classifier* [online]. 2015. [cit. 2016/01/04]. Dostupné z: `https://otik.uk.zcu.cz/bitstream/handle/11025/7621/diploma-thesis.pdf?sequence=1`.

Uživatelská příručka

Program *Conversation Classifier* umí generovat strom konverzací ze vstupu, kterým může být internetové fórum. Vstup musí být ve strukturovaném formátu v *XML*. Program dokáže tento strukturovaný soubor vytvořit z internetové stránky *forum.zive.cz*. Následující části popíší, jak program přeložit, spustit a následné ovládat.

Adresářová struktura

Pro správný běh programu je nutné dodržet následující adresářovou strukturu.

- *HPS*
  - *cz_model.bin*
- *stopwords.txt* - Soubor se seznam stop slov
- *training-data* - V tomto adresáři jsou uložená trénovací data
- *test-data* - V tomto adresáři jsou uložená testovací data
- *Conv-Classify.jar*

Prerekvizity

Aplikace je určená pro běh na operačním systému Windows. Pro její spuštění je nutné mít na počítači nainstalovaný balík Java JRE.

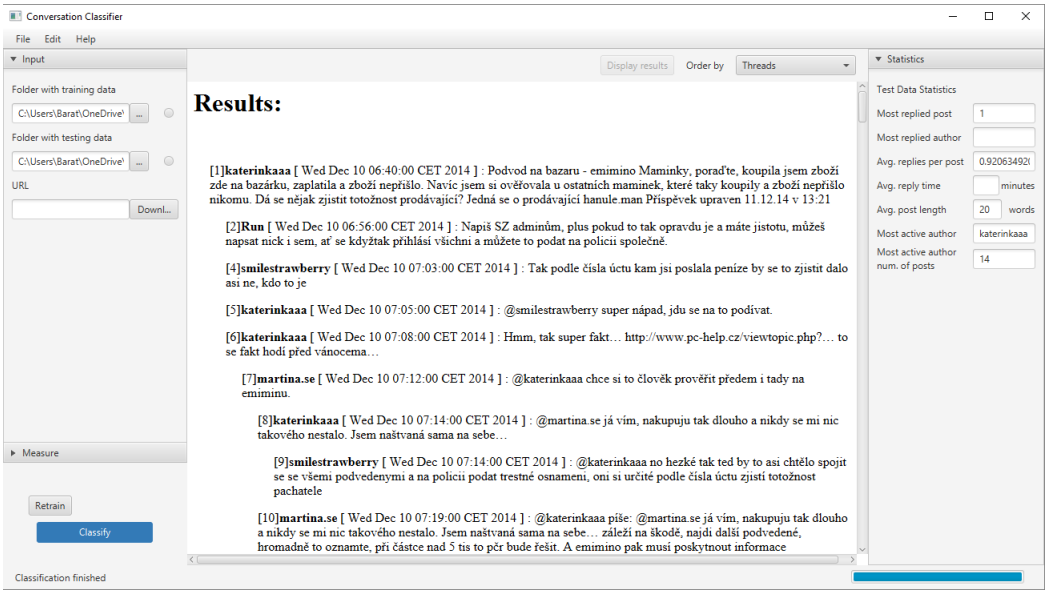
Překlad

Program je již přeložený a zabalený do spustitelného souboru *jar*.

Spuštění a ovládání aplikace

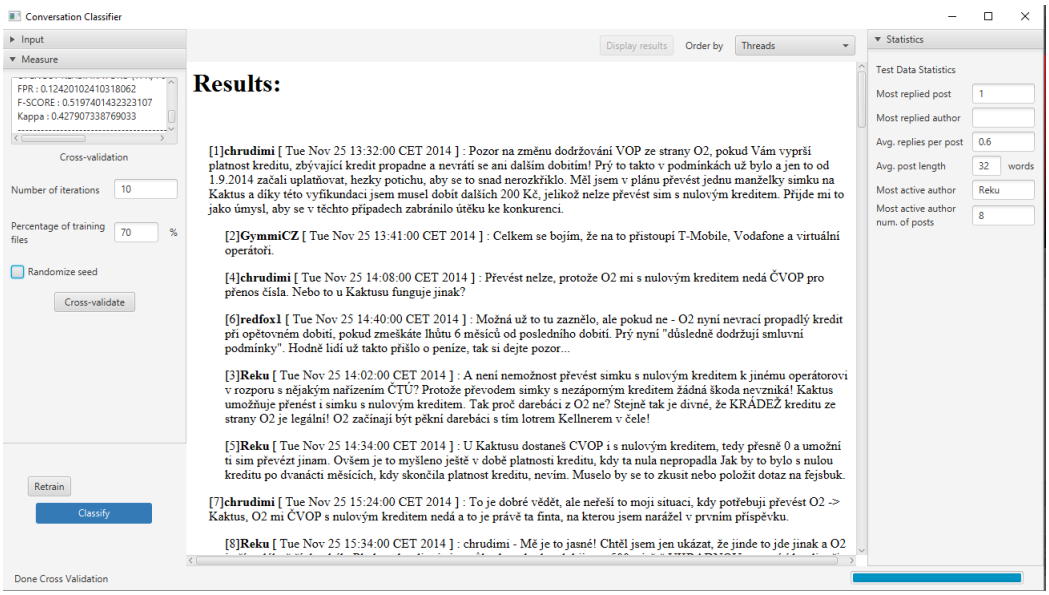
Program spustíme poklepáním na soubor `Conv-Classify.jar`. Ihned po spuštění programu se začnou provádět nezbytné úkoly pro jeho běh. Dochází k načtení jazykového modelu stemmeru a začne se provádět trénování

klasifikátoru pomocí trénovacího korpusu. Trénovací soubory by měly být uloženy ve složce *training-files*, která se musí nacházet na stejné úrovni, jako program. V případě, že chceme mít trénovací soubory uložené na jiném místě, přejdeme do záložky *Input* (viz obrázek 6.1) v levé části aplikace a zvolíme absolutní cestu ke složce s trénovacími daty. Stejně tak zde nastavíme umístění adresáře s testovacími soubory. Výchozí umístění je nastaveno na adresáři *test-files*. Na této záložce můžeme také zvolit URL stránky, ze které chceme extrahovat data do formátu XML, který poté použijeme ke klasifikaci. Tato funkce je však zatím pouze experimentální a funguje pouze pro stránku *forum.zive.cz*.



Obrázek 6.1: Ukázka GUI aplikace - Záložka *Input*.

Pro otestování úspěšnosti klasifikátoru slouží záložka *Mesure* (viz obrázek 6.2). Zde je možné provedení cross validace na trénovacím korpusu dat. Je možné zvolit počet prováděních iterací a poměr dat použitých pro trénování a testování. Vždy se provádí posloupnost iterací složených ze stejných dat, aby bylo možné porovnání výsledků. Pokud chceme kombinaci dat změnit, zaškrtneme pole*Randomize seed*.



Obrázek 6.2: Ukázka GUI aplikace - Záložka *Measure*