Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

PLZEŇ, 2018 JAN BENEŠ

Před svázáním místo této stránky vložit zadání práce s podpisem děkana.

PROHLÁŠENÍ

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

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

V Plzni dne .....

.....

PODĚKOVÁNÍ

Tímto bych chtěl poděkovat panu Ing. Janu Lehečkovi za vedení této bakalářské práce. Zároveň děkuji panu Ing. Janu Švecovi, Ph.D. za konzultace.

Abstrakt

Tato práce se zabývá problematikou segmentace textu. K tomuto úkonu jsou v ní použity jak tradiční metody založené na shlukové analýze, tak moderní přístupy využívající algoritmů hlubokého strojového učení. V teoretické části jsou tyto metody podrobně popsány a v praktické vyhodnoceny.

Klíčová slova

Strojové učení, segmentace textu, klasifikace, python

Abstract

This work presents the field of text segmentation. This task is realized by implementing traditional methods of clustering as well as by using methods based on modern state of the art deep learning techniques. These methods are thoroughly described in the theoretical section and evaluated in the practical.

Key words

Machine learning, text segmentation, classification, python

Obsah

Úvod

Segmentace textu dle tématu je problematika, zabývající se rozdělením textu do tématicky koherentních celků. Tento abstraktní popis lze přeformulovat na problém vyhodnocení, zda se na dané pozici vyskytuje tématický předěl, či nikoliv. Z této formulace je možné usoudit, že je segmentace úzce spjata s vědní disciplínou klasifikace.

Cílem klasifikace je roztřídění objektů do kategorií. V kontextu této práce je výše zmíněným objektem libovolný český text a kategorií je pak nějaké téma. Jelikož text může tématicky spadat do více kategorií (například Politika a Evropská unie), je v této práci použita takzvaná *multi-label* klasifikace (viz sekce 1.3). Techniky klasifikace, a strojového učení[1] obecně, nabývají čím dál většího významu s tím, jak roste množství dat, generovaných lidstvem. Abychom z těchto dat měli užitek, je nutné je strojově zpracovat do uchopitelné formy. Typickým příkladem takového zpracování jsou moderní internetové vyhledávače, kde strojové učení hraje integrální roli v jejich fungování.

V práci jsou využity převážně tradiční metody typu SVMs 1.5. SVMs podávají excelentní výsledky při klasifikaci textu, což je důvod, proč jsou zde tyto algoritmy aplikovány. Výjimku v tradičnosti pak tvoří použití moderních rekurentních neuronových sítí 1.6.2 typu LSTM, které sice byli objeveny už v roce 1997 [1] ale široké uplatnění našly až s příchodem velkého výpočetního výkonu grafických karet.

[1] Oblast umělé inteligence dávající strojům schopnost učit se bez explicitního naprogramování

Cíle práce

Tato práce se skládá ze tří kapitol. Cílem první kapitoly 1 je představení teoretického základu a způsobu vyhodnocení kvality v této práci použitých klasifikačních algoritmů. Druhá kapitola 2 se nese ve stejném duchu s tím rozdílem, že je zaměřena na problematiku segmentace. Poznatky z prvních dvou teoretických částí pak použiji při vypracování třetí kapitoly 3, ve které je popsána realizace hlavního cíle této práce. Tímto cílem je návrh, implementace a vyhodnocení různých segmentačních algoritmů.

Kapitola 1

Klasifikace

1.1 Problematika

Klasifikace je slovo pocházející z latiny a v doslovném překladu znamená třídění. Snažíme se tedy roztřídit vstupní data do kategorií.

Tato problematika nachází široké uplatnění v oblasti automatizace lidské práce, jelikož je jednou z nejdůležitějších činností, kterou náš mozek vykonává při orientaci v tomto světě.

1.2 Extrakce vlastností

Extrakce vlastností (anglicky *feature extraction*) je proces převedení vstupních dat na vektory příznaků. Příznakem může být libovolný parametr objektu, jemuž se snažíme přiřadit kategorii. Typickým příkladem může být v případě klasifikace typu vozidla například počet kol či délka auta. Obecně platí, že čím více vypovídající příznaky vyberu, tím lépe bude klasifikační algoritmus fungovat. Ve výše zmíněném příkladě bude určitě vhodné vybrat jako příznak počet kol. Na druhou stranu barva vozidla nebude pro klasifikaci relevantní. Jednotlivé příznaky by dále měly být co nejméně redundantní. Pokud pracuji s mnoha vysoce korelovanými příznaky, roste výpočetní složitost a úspěšnost klasifikace nikoliv. Pro odstranění vysoce korelovaných příznaků lze například použít „analýzu hlavních komponent“ (anglicky *principal component analysis*), jejíž popis je nad rámec této práce.

1.2.1 Vektorizace textu

U textu nelze příznaky odvodit tak, jako v případě výše zmíněné klasifikace typu vozidla, proto je potřeba pro jejich stanovení použít sofistikovanější metody. V této práci používám takzvaný „Bag-of-words“ model [2], jenž lze vytvořit následujícími třemi úkony:

1. **Tokenizace** je proces převedení korpusu[1] na tokeny. Tokenem může být slovo či posloupnost slov. Této posloupnosti se v problematice strojového učení říká *n-gram*, kde n je počet prvků v posloupnosti. V této práci je n = 1, jelikož korpus je velmi rozsáhlý a zvýšení výpočetních nároků by bylo neúměrné k zisku v kvalitě klasifikace. Dále je jednotlivým tokenům přiděleno ID[2]. Jako separátor jednotlivých řetězců se většinou používá mezera a interpunkce.

2. **Počítání** výskytů tokenů v jednotlivých dokumentech.

3. **Vážení** je proces přiřazení váhy různým slovům podle jejich relevance pro klasifikaci. Například slovo „aby“ je mnohem méně relevantní než slovo „parlament“. V této práci je použita takzvaná *tf-idf metodika* vážení.

1.2.1.1 Tf–idf metodika vážení

Tf–idf metodika je často používaný způsob nastavení vah tokenů, který spočívá ve vynásobení hodnot *term frequency* (*tf*) a *inverse document frequency* (*idf*).

- **Term frequency** je míra výskytu nějakého tokenu v rámci jednoho dokumentu. Existuje několik variant výpočtu této míry. Nejjednodušší variantou je sečtení všech výskytů daného tokenu v dokumentu. V této práci je použito sublineární škálování [3, Sekce 6.4]:

$$tf(t,d) = \begin{cases} 1 + \log f_{t,d}, & \text{pro } f_{t,d} > 0. \\ 0, & \text{jinak.} \end{cases} \tag{1.1}$$

  Kde t je daný token, d je aktuální dokument a $f_{t,d}$ je počet výskytů tokenu t v dokumentu d.

[1] Seznam všech článků
[2] Unikátní identifikátor

- **Inverse document frequency** je míra, jež udává, jak relevantní informaci nám dané slovo poskytuje. Většinou používáme následující vzorec pro výpočet:

$$idf(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \tag{1.2}$$

  Kde D je korpus dokumentů a d je jeden dokument. Čitatel tedy představuje počet všech dokumentů v korpusu a jmenovatel počet všech dokumentů, kde se daný token t vyskytuje. Vidíme, že čím je vyšší počet výskytů v rámci celého korpusu, tím je hodnota *idf* nižší. Opět existuje několik variant výpočtu této hodnoty.

Z definic hodnot je vidět, že *idf* je tedy jakousi protiváhou hodnoty *tf*. Výslednou hodnotu relevance daného slova pro daný dokument získáme součinem [4]:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D) \tag{1.3}$$

Kde parametry *t, d, D* znamenají totéž, co v rovnici 1.2.

1.3 Typy klasifikace

Obecně existují tři typy klasifikace, které se liší přístupem k přidělování kategorií:

- **Binární klasifikace** znamená, že klasifikátor třídí vstupy do jedné ze dvou kategorií. Typickým příkladem je stroj, jenž třídí ovoce na zralé a nezralé kusy.

- **Multi-class klasifikace** se liší od binární klasifikace tím, že vstupní vektor může být zařazen do většího množství kategorií. Příkladem této úlohy může být klasifikace vozidla podle typu na nákladní vůz, osobní automobil a dodávku. Je důležité podotknout, že klasifikovaný objekt musí být zařazen právě do jedné z kategorií.

- **Multi-label klasifikace** se liší od multi-class klasifikace tím, že objekt nemusí být zařazen právě do jedné kategorie, nýbrž do libovolného počtu kategorií. Tento typ klasifikace je využit v této práci při kategorizaci článků. Většina klasifikátorů nepodporuje multi-label klasifikaci nativně. Tento problém lze obejít

takzvanou *one vs the rest* strategií (někdy též *one vs all*) [5]. Tato strategie spočívá v natrénování tolika klasifikátorů, kolik je kategorií, přičemž každý z klasifikátorů určuje zda klasifikovaný objekt náleží do příslušné kategorie či nikoliv. Z principu této strategie vyplývá, že objekt nemusí být zařazen do žádné kategorie.

1.4 Způsoby trénování klasifikátorů

Trénování klasifikátoru je proces, kdy modelu nastavujeme parametry s cílem minimalizovat klasifikační ztrátu. Tento proces lze rozdělit na tři základní typy [6]:

1.4.1 Učení s učitelem

Učení s učitelem je první typ, který využívá externí informaci (takzvaná trénovací data) pro nalezení funkční závislosti mezi vstupem a výstupem. V oblasti klasifikace si pod touto závislostí můžeme představit vztah mezi objektem a kategorií, do které je daný objekt zařazen. Proces učení s učitelem probíhá tak, že pomocí modelu vytvoříme predikci a vypočteme ztrátu z predikcí a referenčních hodnot. Na základě této ztráty pak upravíme nastavení modelu. Tento proces opakujeme dokud nedosáhneme určité velikosti ztráty, případně nějakého počtu iterací. Jak je z názvu slyšet, figuruje v této metodě nějaký učitel. Tímto učitelem je algoritmus, jenž podle referenčních dat upraví nastavení.

Mezi klasifikátory, které jsou trénovány pomocí učení s učitelem patří například *Bayesovský klasifikátor* nebo *SVM algoritmus*, jenž je popsán v sekci 1.5.

Jelikož mám k dispozici korpus dokumentů s přidělenými tématy, budu se v této práci zabývat převážně touto metodou učení.

1.4.2 Učení bez učitele

Učení bez učitele je druhý způsob trénování, který se vyznačuje tím, že externí informaci v podobě kategorizovaných trénovacích dat nemá. V tomto případě je tedy nutno pro roztřídění dat do kategorií použít nějakou míru podobnosti.

Tento způsob trénování většinou dosahuje výrazně horších výsledků, proto se

při praktické aplikaci často data před trénováním manuálně rozdělí do kategorií a použije se „konkurenční“ metoda 1.4.1.

1.4.3 Kombinovaná metoda

Jak už z názvu vyplývá, tato metoda je kombinací předešlých dvou metod. Tato metoda se používá tehdy, pokud máme trénovací data, jež jsou jen z části označeny referenčními kategoriemi. Proces učení v tomto případě probíhá tak, že model natrénujeme na označených datech. Tento model pak použijeme pro vytvoření predikcí na datech zbývajících, a predikce, jimiž si je model „jist“ (vysoká pravděpodobnost), zařadíme k referenčním datům. Takto označená data pak použijeme pro opětovné trénování a celý proces opakujeme.

1.5 Support Vector Machines

Support vector machines (zkráceně SVMs) je skupina metod strojového učení, jež mají široké uplatnění v klasifikaci (SVM) a regresi (SVR). SVMs mají diskriminativní[3] charakter a vyznačují se robustností a flexibilitou. V sekci níže popíšu použití těchto metod v klasifikačních úlohách.

1.5.1 Matematický základ SVM

SVMs jsou trénovány pomocí učení s učitelem. Pro natrénování proto potřebujeme dataset[4] $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$, kde m je počet trénovacích vzorků, x_i je vstupní vektor, y_i je třída vstupního vektoru x_i a $\forall i : i \leq m \wedge i \in \mathbb{N}$. Jelikož SVM je binární klasifikátor $y_i \in \{-1, +1\}$.

Dělící nadrovina SVM má následující tvar [7]:

$$\vec{\omega} \cdot \vec{x} + b = 0 \tag{1.4}$$

Kde $(\vec{\omega}, b)$ jsou parametry klasifikátoru. Geometrický význam vektoru $\vec{\omega}$ je normála

[3] Model, jenž explicitně popisuje hranice tříd v příznakovém prostoru, nikoliv však jak jsou samotné třídy generovány

[4] Kolekce dat

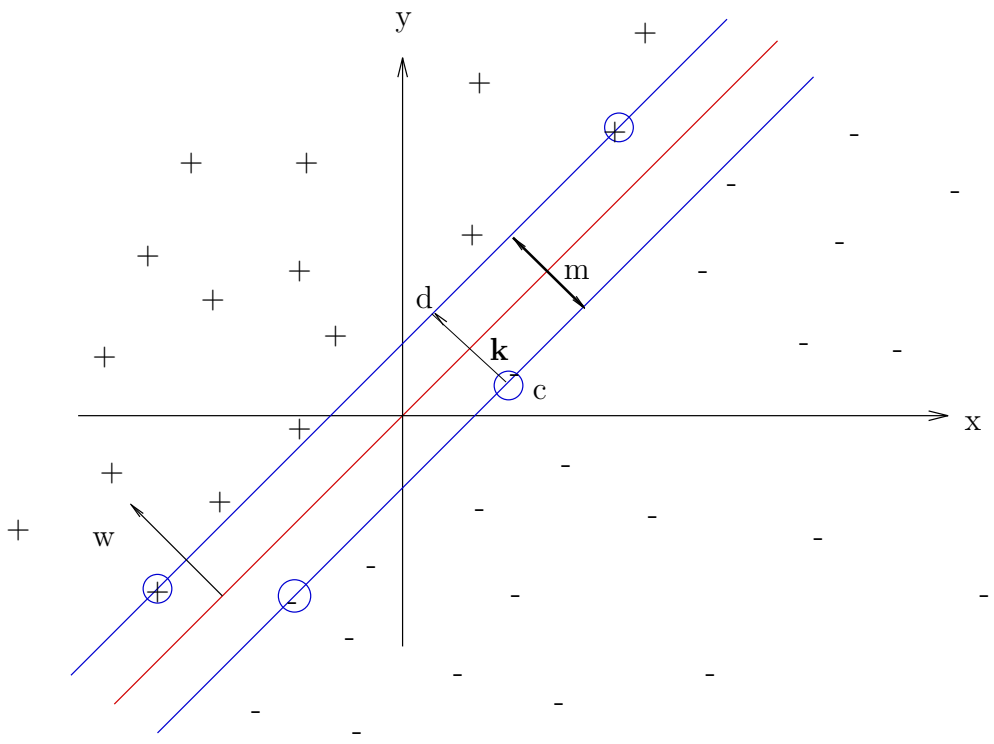
dané nadroviny. Tento vektor má stejnou dimenzi jako vstupní data.

Vektory, jenž leží nejblíže nadrovině definované rovnicí 1.4 se nazývají **podpůrné vektory**. Tyto vektory jsou označeny na obrázku 1.1 modrými kroužky a definujeme pomocí nich nadroviny daných tříd:

$$\vec{\omega} \cdot \vec{x_p} + b = +1 \tag{1.5}$$

$$\vec{\omega} \cdot \vec{x_n} + b = -1 \tag{1.6}$$

Kde $\vec{x_p}$ v rovnici 1.5 je podpůrný vektor pozitivní třídy (dále označeno jako T_{+1}) a $\vec{x_n}$ v rovnici 1.6 je podpůrný vektor negativní třídy (dále označeno T_{-1}).



Obrázek 1.1: Vizualizace podpůrných vektorů [8]

Při samotném trénování se snažíme maximalizovat vzdálenost podpůrných vektorů [9] od dělící nadroviny 1.4 přičemž nesmíme porušit následující podmínky:

$$\vec{\omega} \cdot \vec{x_i} + b \geq +1, \forall x_i \in T_{+1} \tag{1.7}$$

$$\vec{\omega} \cdot \vec{x_i} + b \leq -1, \forall x_i \in T_{-1} \tag{1.8}$$

Nerovnice 1.7 a 1.8 lze jednoduchou úpravou sloučit do jedné. Nejprve rovnici 1.8 vynásobíme třídou vektoru $\vec{x_i}$:

$$y_i \cdot (\vec{\omega} \cdot \vec{x_i} + b) \geq y_i \cdot (-1) \tag{1.9}$$

Jelikož se v daném případě y_i vždy rovná *-1*, bylo nutné změnit znaménko nerovnosti. Po úpravě nerovnice 1.9 dostaneme finální tvar:

$$y_i \cdot (\vec{\omega} \cdot \vec{x_i} + b) \geq 1 \tag{1.10}$$

Nerovnice 1.10 elegantně nahrazuje nerovnice 1.7 a 1.8 při zachování stejného matematického významu.

Vzdálenost podpůrných vektorů (na obrázku 1.1 označena m) od dělící nadroviny 1.4 je možné vyjádřit z rovnic 1.5 a 1.6 v závislosti na normále $\vec{\omega}$. Vektor $\vec{d}$ leží na nadrovině třídy T_{+1}, proto pro něj platí rovnice:

$$\vec{\omega} \cdot \vec{d} + b = 1 \tag{1.11}$$

Vektor $\vec{d}$ můžeme vyjádřit jako součet vektoru $\vec{k}$ a vektoru $\vec{c}$, který leží na podpůrné rovině třídy T_{-1}:

$$\vec{\omega} \cdot (\vec{c} + \vec{k}) + b = 1 \tag{1.12}$$

Vektor $\vec{k}$ má stejnou velikost jako je vzdálenost m a stejný směr jako normála nadroviny $\vec{\omega}$, proto jej lze za pomocí tohoto skaláru a vektoru vyjádřit:

$$\vec{\omega} \cdot (\vec{c} + m\frac{\vec{\omega}}{\|\vec{\omega}\|}) + b = 1 \tag{1.13}$$

Roznásobíme závorku:

$$\vec{\omega} \cdot \vec{c} + m\frac{\vec{\omega} \cdot \vec{\omega}}{\|\vec{\omega}\|} + b = 1 \tag{1.14}$$

Z definice skalárního součinu vyplývá, že skalární součin vektoru se sebou samým

je euklidovská norma umocněná na druhou, proto platí:

$$\vec{\omega} \cdot \vec{c} + m\frac{\|\vec{\omega}\|^2}{\|\vec{\omega}\|} + b = 1 \tag{1.15}$$

$$\vec{\omega} \cdot \vec{c} + m\,\|\vec{\omega}\| + b = 1 \tag{1.16}$$

Převedeme člen s normou na pravou stranu:

$$\vec{\omega} \cdot \vec{c} + b = 1 - m\,\|\vec{\omega}\| \tag{1.17}$$

Vektor $\vec{c}$ leží na nadrovině třídy T_{-1}, proto se levá strana rovnice 1.17 rovná *-1*:

$$-1 = 1 - m\,\|\vec{\omega}\| \tag{1.18}$$

$$m\,\|\vec{\omega}\| = 2 \tag{1.19}$$

$$m = \frac{2}{\|\vec{\omega}\|} \tag{1.20}$$

Z rovnice 1.20 vidíme, že maximalizaci vzdálenosti m lze dosáhnout minimalizací normy normály $\vec{\omega}$. Tento poznatek využijeme k formulaci finálního optimalizačního problému:

$$Minimalizujte\ \ \|\vec{\omega}\|\ \ tak,\ aby\ \forall i = 1, \ldots, n\ platilo : y_i \cdot (\vec{\omega} \cdot \vec{x_i} + b) \geq +1 \tag{1.21}$$

Problém 1.21 lze převést na konvexní kvadratickou formu, jež je mnohem snáze řešitelná. Dostaneme tak tvar, jenž je nejčastěji používán v literatuře zabývající se danou problematikou [7]:

$$Minimalizujte\ \frac{1}{2}\,\|\vec{\omega}\|^2\ \ tak,\ aby\ \forall i = 1, \ldots, n\ platilo : y_i \cdot (\vec{\omega} \cdot \vec{x_i} + b) - 1 \geq 0 \tag{1.22}$$

Tento optimalizační problém představuje samotné **trénování klasifikátoru**. Na trénování SVMs je zajímavé, že nastavení klasifikátoru ovlivní pouze podpůrné vektory.

Při samotném hledání řešení většinou převedeme primární minimalizační problém 1.22 na duální maximalizační, jehož řešení je v kontextu SVMs totožné (jedná se o

takzvanou silnou dualitu viz [7]):

$$\vec{\alpha} = \arg\max_{\vec{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j (\vec{x_i} \cdot \vec{x_j}) \tag{1.23}$$

při splnění podmínek:

$$\vec{\alpha_i} \geq 0; \forall i = 1, \ldots, m; \ \sum_{i=1}^{m} \alpha_i y_i = 0$$

Vektor $\vec{\alpha}$ je vektor takzvaných *Lagrangeových multiplikátorů* a lze pomocí nich přejít k řešení $\vec{\omega}$:

$$\vec{\omega} = \sum_{i=1}^{m} \alpha_i y_i \vec{x_i} \tag{1.24}$$

Nyní máme natrénovaný klasifikátor a můžeme přistoupit k samotné klasifikaci.

1.5.1.1 Klasifikace pomocí SVM

Klasifikace pomocí SVM probíhá tak, že dosadíme vektor ke klasifikaci $\vec{x_k}$ do nerovnic, jež vychází z rovnice dělící nadroviny 1.4:

$$\vec{\omega} \cdot \vec{x_k} + b \geq 0 \tag{1.25}$$

$$\vec{\omega} \cdot \vec{x_k} + b < 0 \tag{1.26}$$

Pokud je pro vektor $\vec{x_k}$ splněna nerovnice 1.25 patří vektor do třídy T_{+1}. Analogicky platí, že v případě splnění nerovnice 1.26 přísluší vektor třídě T_{-1}.

Jestliže byl pro natrénování použit duální problém 1.23, je vhodné použít hypotézu[5], jež je nativní danému problému [9]:

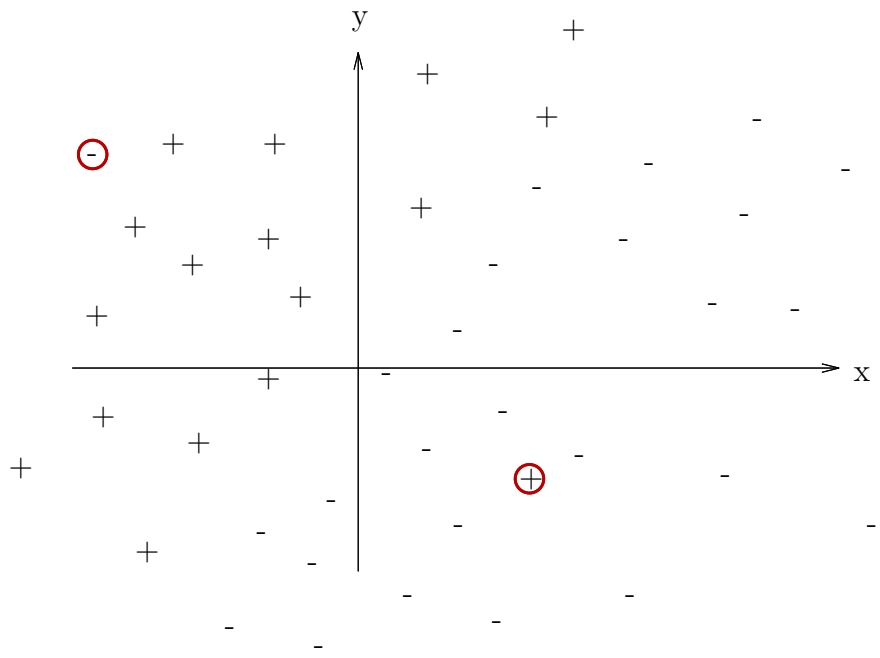
$$h(\vec{x_i}) = sign(\sum_{j=1}^{m} \alpha_j y_j (\vec{x_i} \cdot \vec{x_j}) + b) \tag{1.27}$$

1.5.1.2 Neseparabilní případ

Optimalizační problém 1.22 včetně jeho duální verze mají tu vlastnost, že fungují korektně pouze na lineárně separabilních datech. Na reálných trénovacích datech se

[5]Funkce definující závislost mezi prostorem dat a prostorem kategorií/tříd

často stává, že není možné dvě třídy dokonale rozdělit lineární hypotézou. Tento případ je vidět na obrázku 1.2, kde body označeny červenými kroužky znemožňují separaci s nulovou ztrátou. Z toho důvodu je potřeba algoritmu umožnit „udělat chybu“.



Obrázek 1.2: Rozložení dat v rovině [8]

Toho lze docílit zavedením přídavné proměnné ξ do omezujících podmínek:

$$y_i \cdot (\vec{\omega} \cdot \vec{x_i} + b) \geq 1 - \xi \tag{1.28}$$

Samotná podmínka 1.28 by pak byla splněna pro libovolně velké ξ. Abychom předešli tomuto problému, zavedeme takzvaný regularizační[6] parametr do optimalizačního problému:

$$(\vec{\omega}, b, \xi) = \arg\min_{\vec{\omega},b,\xi} \frac{1}{2} \|\vec{\omega}\|^2 + C \sum_{i=1}^{m} \xi_i \tag{1.29}$$

Tato verze klasifikátoru je anglicky nazývána *soft-margin SVM*.

Pro zavedení regularizace do duálního problému 1.23 změníme omezující podmínku:

[6]Zavedení dodatečné informace jako prevenci proti přetrénování

$$0 \leq \vec{\alpha}_i \leq C; \forall i = 1, \ldots, m; \sum_{i=1}^{m} \alpha_i y_i = 0$$

Hyperparametr[7] C nám umožňuje nastavit váhu regularizačního prvku.

1.5.2 Nelineární SVM

Výše je popsáno, jak aplikovat SVM na data, jež jsou do velké míry lineárně separovatelná. Obecně však data často lineárně separovat nelze. V takovém případě je nutné klasifikovat data ve vyšší dimenzi než je nativní dimenze dat. Transformace dat do vyšší dimenze je výpočetně náročná, proto byla vyvinuta metoda, jež provede klasifikaci ve vyšší dimenzi bez samotné transformace dat. Tato metoda se nazývá *jádrový trik* (anglicky *kernel trick*) a jejím výstupem je skalární součin vektorů ve vyšší dimenzi. Definice jádrové funkce má následující tvar [7]:

Definice 1.5.1. Mějme funkci $\Phi : \mathbb{R}^d \to H$, kde $\mathbb{R}^d$ je nativní dimenze dat. Funkci $K : \mathbb{R}^d \to \mathbb{R}$ definovanou jako $K(\vec{x}, \vec{x}') = < \Phi(\vec{x}), \Phi(\vec{x}') >_H$, kde $< \cdot, \cdot >_H$ označuje skalární součin v prostoru H, nazýváme **jádrovou funkcí**.

Jádrový trik lze aplikovat velmi jednoduše. Stačí v duálním maximalizačním problému 1.23 a příslušné hypotéze 1.27 použít jádrovou funkci místo skalárního součinu.

1.5.2.1 Přehled jader

Aplikace jádrového triku je velmi flexibilní a umožňuje vytvoření jádrové transformace na míru danému problému. Vytvoření vlastní jádrové transformace však vyžaduje expertní znalost problematiky a ve většině případů je vhodnější použít obecně známe definice jader.

V této práci je použit takzvaný **lineární kernel**, jenž je definován jako [7]:

$$K(\vec{x}, \vec{x}') = \vec{x} \cdot \vec{x}' \tag{1.30}$$

Tento typ jádra je vhodné použít v situacích, kde klasifikovaná data mají velké množství příznaků, jelikož v takových situacích mapování dat do vyšších dimenzí

[7] Parametr, jenž je nastaven před samotným trénováním

nepřináší žádné zlepšení klasifikace. Textová data jsou vektorizována do vektorů vysokých dimenzí a bývají lineárně separovatelná [10]. Z toho důvodu je lineární kernel téměř vždy optimální volbou pro klasifikaci textu. Další výhodou lineárního jádra je výpočetní nenáročnost oproti komplexnějším formám transformací.

Lineární kernel je nejjednodušším typem **polynomického jádra** [7], které je definováno rovnicí:

$$K(\vec{x}, \vec{x}') \;=\; (\vec{x} \cdot \vec{x}' + c)^d \tag{1.31}$$

Kde c je konstanta a d je stupeň polynomu. Polynomický kernel vyššího řádu umožňuje klasifikátoru naučení komplexní rozhodovací hranice. Při příliš vysokém řádu polynomu je však klasifikátor s tímto jádrem náchylný k přetrénování[8].

Posledním v této práci zmíněným jádrem je takzvaný **RBF kernel** [7], jenž má následující matematický zápis:

$$K(\vec{x}, \vec{x}') \;=\; \exp\left(-\frac{\|\vec{x} - \vec{x}'\|^2}{2\sigma^2}\right) \tag{1.32}$$

Kde σ je volný parametr. Výhodou RBF jádra je schopnost naučení se velmi komplexních hypotéz 5.

1.5.3 Vlastnosti SVM

Kategorizace textu se vyznačuje velkým množstvím relevantních příznaků [10]. Je proto nevhodné použití metod, jež redukují dimenzi příznakového prostoru. SVMs se vyznačují svou efektivitou při práci s velmi vysokými dimenzemi, z toho důvodu v kontextu kategorizace textu podávají velmi dobré výsledky. Tato vlastnost je prakticky ověřena v praktické části 3.1.2 této práce, kde jsou textové úseky transformovány do vektorů vysokých dimenzí, při zachování přijatelné výpočetní náročnosti. Další výhodou SVM je možnost výběru jádra, jež nám umožňuje velmi efektivně aplikovat SVM na velké množství různých problémů. Na druhou stranu tato flexibilita vyžaduje určitou zkušenost při výběru modelu, jelikož samotný výběr jádra není pokryt matematickou teorií.

[8] Vlastnost, kdy má klasifikátor velkou úspěšnost na trénovacích datech a nízkou na testovacích

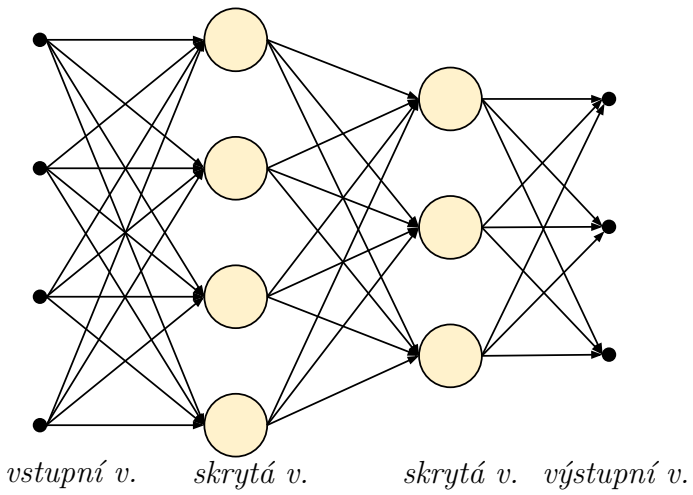
1.6 Umělé neuronové sítě

Umělá neuronová síť je výpočetní model, jenž je inspirován biologickou strukturou mozku. Tento model je velmi flexibilní a lze jej tak aplikovat na velké množství problémů. Díky příchodu vysoce výkonných grafických čipů na trh je nyní možné úspěšně natrénovat hluboké neuronové sítě[9], jež se vyznačují schopností zachytit vysoce komplexní závislosti.

Neuronové sítě lze rozdělit podle směru šíření informace na dva základní typy:

1.6.1 Dopředné neuronové sítě

Prvním typem jsou dopředné neuronové sítě (znázorněno na obrázku 1.3). Jak je z názvu slyšet, informace se v tomto případě šíří pouze od vstupů k výstupům. Nenastane tak situace, kdy výstup vrstvy ovlivní výstup té samé vrstvy při dalším průchodu informace. Nejjednodušší realizací tohoto typu je jednovrstvý **perceptron**. Perceptron je algoritmus lineární klasifikace, jenž byl zformulován v 50. letech 20. století [11]. Jak lze z doby objevení usoudit, jedná se o jednu z prvních neuronových sítí.



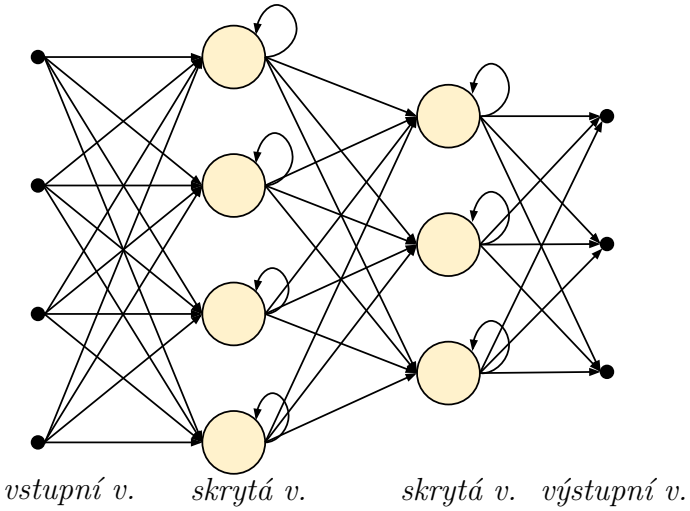
Obrázek 1.3: Dopředná neuronová síť [12]

Dopředné neuronové sítě jsou typicky nasazovány na problémy, kde hledáme funkční závislost mezi vstupem a výstupem.

[9]Sítě s více skrytými vrstvami

1.6.2 Zpětnovazební neuronové sítě

Druhým typem jsou zpětnovazební neuronové sítě (v některé literatuře označovány jako rekurentní). Jak je znázorněno na obrázku 1.4, informace se v tomto případě šíří zpět na vstup. Výstup tak v tomto případě není závislý pouze na vstupu, ale také na stavu v němž se síť nacházela v předešlé iteraci. Mezi zástupce zpětnovazebních neuronových sítí patří například **Hopfieldova síť**, jež je používána jako autoasociativní paměť[10].



Obrázek 1.4: Zpětnovazební neuronová síť [12]

Rekurentní neuronové sítě jsou používány všude tam, kde potřebujeme zpracovat sekvenci vstupů, které jsou na sobě časově závislé. Typickou aplikací je tedy rozpoznávání řeči nebo psaného textu, kde rozdělení pravděpodobnosti aktuální predikce je silně ovlivněno předešlými vstupy (hlásky, písmena). Tento typ neuronové sítě (konkrétně verze LSTM 1.6.2.1) je používán i v této práci, kdy je na vstup přivedena sekvence predikcí z klasifikátoru témat a na výstupu očekávám informaci o tématických hranicích viz 3.3.

U rekurentních neuronových sítí se výrazně projevily 2 problémy [13], které nastávají při použití *algoritmu zpětně propagace* na hluboké sítě. Samotná definice zpětnovazebních sítí neimplikuje, že by se jednalo o sítě hluboké. Síť se však v tomto případě trénuje speciální verzí *algoritmu zpětné propagace* (takzvaný *algo-*

[10]Paměť, jež je adresována kusem sebe samé

ritmus zpětné propagace v čase, anglicky *backpropagation through time*), což má za následek umocnění těchto problémů.

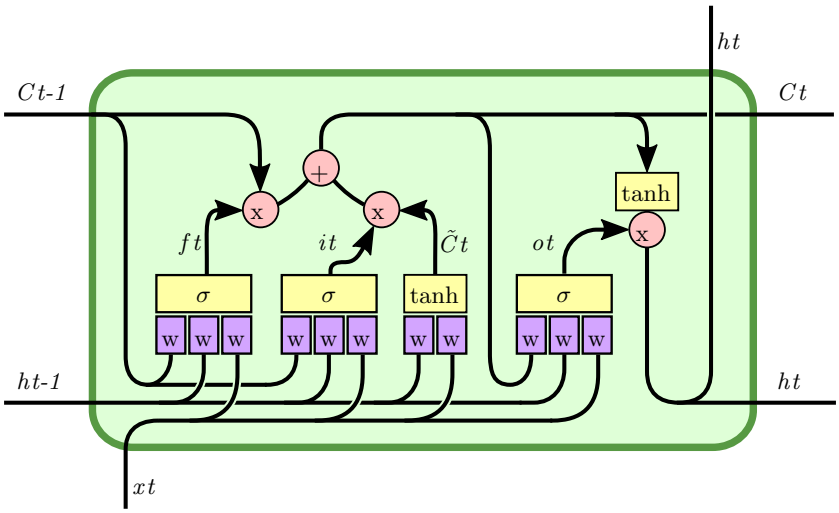
Prvním problémem je takzvaný *mizící gradient* (anglicky *vanishing gradient*). Tento problém se dá přirovnat k zapomínání, jelikož brání rekurentním neuronovým sítím ve vytvoření dlouhodobých závislostí. V momentě, kdy se změny propagují zpět do minulých časových kroků či hlubších vrstev, se tyto změny exponenciálně blíží k 0. Tento jev nastává u rekurentních sítí s lineární aktivační funkcí v případě, kdy největší vlastní číslo váhové matice je menší než 1. U sítí s nelineární aktivační funkcí je tento problém též přítomen, ale nelze říci, že největší vlastní číslo musí být menší než 1, aby problém nastal [13].

Druhý problém má zcela opačný charakter a nazývá se problém *explodujícího gradientu* (anglicky *exploding gradient*). Při zpětné propagaci změny exponenciálně rostou, což zapříčiní, že je model nestabilní a nic se nenaučí.

Efektem výše zmíněných problémů bylo, že hluboké a rekurentní neuronové sítě byly velmi obtížně natrénovatelné.

1.6.2.1 Long short-term memory

Long short-term memory (zkráceně LSTM) je typ rekurentní neuronové sítě, který problém *mizícího* a *explodujícího gradientu* řeší [14].



Obrázek 1.5: LSTM [15]

Jeden typ LSTM jednotky/neuronu je znázorněn na obrázku 1.5. LSTM zavádí

dodatečné vrstvy, jež precizně kontrolují co se má zapomenout či zapamatovat.

První taková vrstva se anglicky nazývá *forget gate layer* (na obrázku označena f_t), jejíž výstup je vložen do sigmoidální aktivační funkce. Tato vrstva určuje, jaká informace se má zapomenout.

Další vrstva je anglicky nazývána *input gate layer* (na obrázku označena i_t). Výstup této vrstvy je též vložen do sigmoidální funkce, proto nabývá hodnot na intervalu $(0, 1)$. Jak je vidět z obrázku 1.5, tento výstup je vynásoben výstupem vrstvy, jež produkuje takzvané *kandidátní hodnoty* (označeno $\tilde{C}_t$). Kombinace těchto dvou vrstev tedy určí, jaké hodnoty budou přidány do vnitřního stavu LSTM jednotky.

Poslední vrstvou je vrstva výstupní (označena o_t). Tato vrstva určuje, jaká informace se dostane na samotný výstup celé jednotky. Její výstup je vynásoben aktuálním stavem C_t, který byl normován na interval $(-1, 1)$ pomocí funkce *hyperbolický tangens*.

1.7 Vyhodnocení

Existuje mnoho metrik vyhodnocení kvality algoritmu. Tyto metriky vznikly jako reakce na specifické požadavky různých aplikací [16].

1.7.1 Úspěšnost

Nejjednodušší takovou metrikou je *úspěšnost* (anglicky *accuracy*), jejíž matematický zápis je následující:

$$accuracy(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} 1(\hat{y}_i = y_i) \tag{1.33}$$

Kde n je počet predikcí, $\hat{y}_i$ je predikovaná hodnota a y_i je příslušná pravdivá hodnota. Úspěšnost má jeden zásadní nedostatek. V případě, kdy nejsou pravdivé hodnoty rovnoměrně distribuovány mezi možnostmi (anglicky nazýváno *skewed data*), má tato metrika velmi špatnou vypovídací hodnotu [16]. Typickým příkladem této vlastnosti může být test rakoviny, který by pokaždé vrátil negativní výsledek. Tento klasifikátor (též nazýván *triviální neakceptor*[11]) by měl velkou úspěšnost v

[11] Klasifikátor, jenž vrátí pokaždé negativní odpověď

absolutních hodnotách (málo lidí má ve skutečnosti rakovinu), ale test by byl ve skutečnosti k ničemu.

1.7.2 F-míra

F-míra (též nazýváno F_1 skóre) je metrika, jež řeší problémy spojené s *úspěšností* 1.7.1. Pro její definici nejprve zavedu tabulku, která je anglicky nazývána *table of confusion* nebo *confusion matrix*.

		True class	
		Positive	Negative
Predicted class	Positive	**True Positive**	**False Positive**
	Negative	**False Negative**	**True Negative**

Tabulka 1.1: Table of confusion

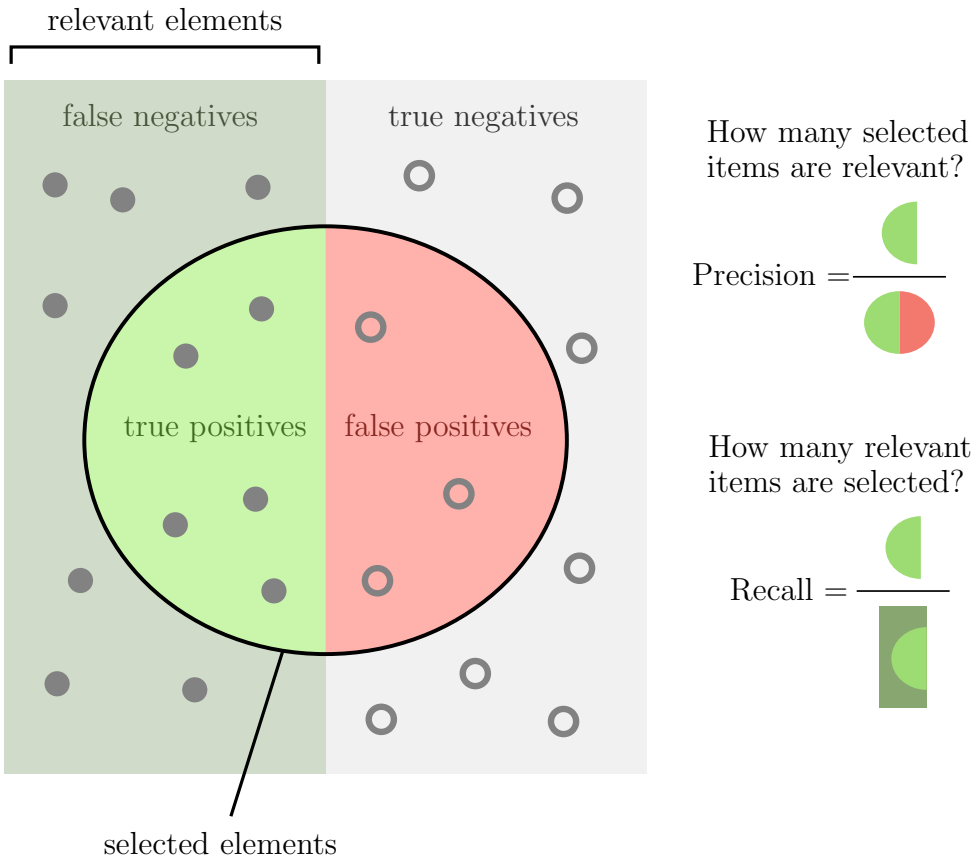
Hodnoty v tabulce jsou definovány takto:
True positives - počet pozitivních výstupů správně identifikovaných jako pozitivní
False positives - počet pozitivních výstupů, jenž byly špatně klasifikovány jako pozitivní
True negatives - počet negativních výstupů správně klasifikovaných jako negativní
False negatives - počet negativních výstupů, které byly špatně klasifikovány jako negativní

F-míra je pak definovaná pomocí dvou veličin. První veličinou je přesnost (anglicky nazývána *precision*), která je definována jako:

$$Precision = \frac{True\ positives}{True\ positives + False\ positives} \tag{1.34}$$

Druhou veličinou je úplnost (anglicky *recall*):

$$Recall = \frac{True\ positives}{True\ positives + False\ negatives} \tag{1.35}$$



Obrázek 1.6: Infografika přesnosti a úplnosti [17]

Nyní můžeme definovat samotnou F-míru:

$$F_1 = 2 \cdot \frac{1}{\frac{1}{Recall} + \frac{1}{Precision}} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{1.36}$$

F-míra je velmi často používanou metrikou, jelikož podává relevantní hodnotu i v případě, kdy pracujeme se zkosenými daty[12]. Tato metrika tedy řeší zásadní nedostatek úspěšnosti 1.7.1.

[12]Data, jež z velké části spadají do jedné kategorie

1.7.2.1 Vyhodnocení *multi-label* klasifikace

Definice 1.36 má ten nedostatek, že lze použít jen v případě dichotomie[13]. Existuje však několik způsobů, jak se s tímto nedostatkem vypořádat:

1. *Micro averaging* spočívá ve výpočtu *True positives*, *False positives* a *False negatives* globálně. Neboli jsou tyto hodnoty vypočteny pro každou pozici *[vzorek, třída]* nezávisle. Na těchto hodnotách je pak vypočtena F míra.

2. *Macro averaging* je přístup, kdy jsou hodnoty nejprve vypočítány pro jednotlivé třídy. Z těchto hodnot je pak určen průměr.

3. *Weighted averaging* je způsob podobný *macro averaging* s tím rozdílem, že hodnoty pro dané třídy jsou váženy počtem přiřazení k této třídě (tento počet je anglicky nazýván *support*).

4. *Samples averaging* je přístup, který na rozdíl od *macro* a *weighted averaging* není vypočítán pro jednotlivé třídy, ale pro jednotlivé vzorky. Hodnoty na jednotlivých vzorcích jsou při výpočtu výsledné míry zprůměrovány. Tento přístup je použit v této práci a má smysle jej aplikovat pouze na *multi-label* problémy.

Popis a názvy těchto způsobů jsou převzaty z dokumentace knihovny *scikit-learn* [18].

[13] Třídění do dvou vylučujících se skupin

Kapitola 2

Segmentace

2.1 Problematika

Segmentace textu je soubor problémů, jehož řešení má široké uplatnění. Lidstvo má k dispozici obrovské množství textových dokumentů, jenž často nemají známou strukturu. Algoritmy segmentace nám v takovém případě mohou pomoci s nalezením struktury, což výrazně ulehčuje orientaci lidem i strojům.

Hlavní doménou segmentace je rozdělení dlouhého textu do tématicky koherentních celků. Zde narážíme na problém jak definovat co je to téma tak, aby mu stroj „porozuměl“. Jak už bylo zjištěno v mnoha jiných tezích, tento úkon je poměrně komplexní [19]. Jelikož mám k dispozici dostatečně velké množství dat 3.1, mohu si dovolit použít moderní přístup strojového učení. Tento přístup spočívá v natrénování klasifikátoru, který automaticky extrahuje implicitní znalost z dat o tom, co definuje jednotlivá témata.

2.2 Vyhodnocení

Tato kapitola je určitou návazností na kapitolu 1.7, jelikož metriky používané pro vyhodnocení kvality klasifikace je možné použít i na segmentaci textu. Existují však metriky vytvořené přímo pro vyhodnocení segmentace, které berou v potaz specifické požadavky dané úlohy.

2.2.1 P_k metrika

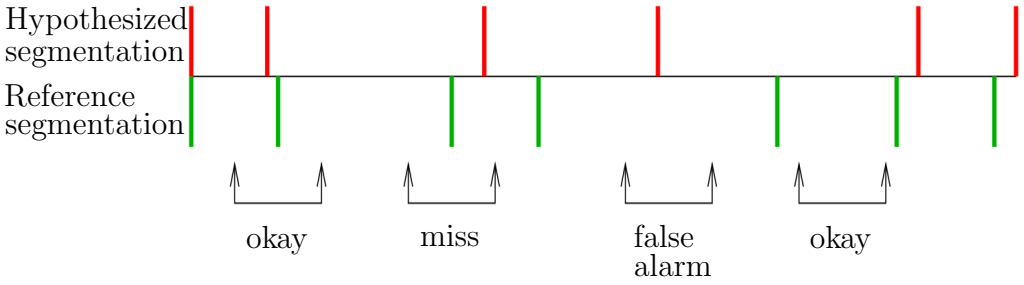
P_k metrika byla vyvinuta jako reakce na nedostatky F míry 1.7.2 při použití pro vyhodnocení kvality segmentace textu [20]. Chybou F míry je, že nebere v potaz vzdálenost predikované tématické hranice od skutečné. Je žádoucí, aby algoritmus který nalezne hranici ve vzdálenosti jedné věty od skutečné hranice byl ohodnocen lépe než algoritmus, který hranici nastaví uprostřed tématického celku. Tato metrika udává pravděpodobnost, že náhodně zvolená dvojice pozic ve vzdálenosti k je nekonzistentně klasifikována [20]. Nekonzistencí je v tomto případě myšleno, že pro jednu konkrétní segmentaci dvojice pozic leží ve stejném tématickém úseku a pro jinou segmentaci v různých. Touto pozicí je v kontextu této práce odstavec, ale může jí být třeba slovo či věta.

Definice metriky je následující:

$$P_k = \frac{1}{N-k} \sum_{i=1}^{N-k} (|q(ref_i, ref_{i+k}) - q(hyp_i, hyp_{i+k})| > 0) \tag{2.1}$$

Kde $q(i, j)$ je funkce, udávající zda se pozice i a j nachází v různých tématických celcích či nikoliv. N je počet vět (případně odstavců) v segmentovaném textu a k je velikost „okénka“, neboli vzdálenost pozic i a j.

Na obrázku 2.1 jsou vyobrazeny situace, které při posuvu „okénka“ mohou nastat:



Obrázek 2.1: P_k metrika [21]

Jelikož P_k udává pravděpodobnost chyby, jedná se o číslo na intervalu $< 0, 1 >$ a platí, že čím nižší číslo P_k je, tím lépe daný segmentační algoritmus pracuje.

2.2.1.1 Volba parametru *k*

Empiricky bylo zjištěno, že nastavení k na polovinu průměrné délky tématických celků podává nejkonzistentnější výsledky při vyhodnocení triviálních segmentačních algoritmů[1]. Hodnota P_k je v takovém případě přibližně $\frac{1}{2}$.

2.2.2 WindowDiff metrika

P_k metrika 2.2.1 má několik závažných problémů [22]:

- Penalizace *false negatives* (*miss*) více než *false positives* (*false alarm*)
- Nebere v potaz počet tématických hranic
- Senzitivita na variabilní délku segmentů
- Příliš vysoká penalizace umístění hranice, jež je blízko referenční pozici (takzvaná *near-miss error*)
- Málo intuitivní - nejlepší algoritmus ohodnocen nulou a triviální algoritmus přibližně polovinou

Ukázalo se, že jednoduchá modifikace [22] P_k metriky odstraní většinu problémů výše. Touto modifikací je penalizace algoritmu v případě, že na daném úseku je jiný počet tématických hranic než je referenční počet. Takto nově vniklá metrika byla pojmenována *WindowDiff* a má následující matematický zápis:

$$WindowDiff = \frac{1}{N-k}\sum_{i=1}^{N-k}(|b(ref_i, ref_{i+k}) - b(hyp_i, hyp_{i+k})| > 0) \tag{2.2}$$

Kde $b(i,j)$ představuje počet hranic mezi pozicemi i a j. Parametry N a k znamenají totéž, co u rovnice 2.1.

[1] Algoritmy, jenž predikují změnu tématu všude, náhodně nebo nikde

Kapitola 3

Praktická část

Jelikož mi bylo dáno k dispozici poměrně velké množství dat s přidělenými tématy, vytvořil jsem algoritmus, který využívá informaci o tématu článků pro segmentaci textu. Alternativní přístupy [23], které též používají v procesu segmentace klasifikátor SVM 1.5, většinou pracují pouze s informací o tématických hranicích článků, nikoliv však s informací o samotných tématech. Tímto se přístup v této práci liší od probádaných alternativ a může tak poskytnou jiný pohled na segmentaci textu.

V první části této kapitoly je popsáno zpracování dat a trénování klasifikátorů. V druhé části je otestováno několik přístupů využívajících predikce natrénovaných klasifikátorů pro segmentaci textu.

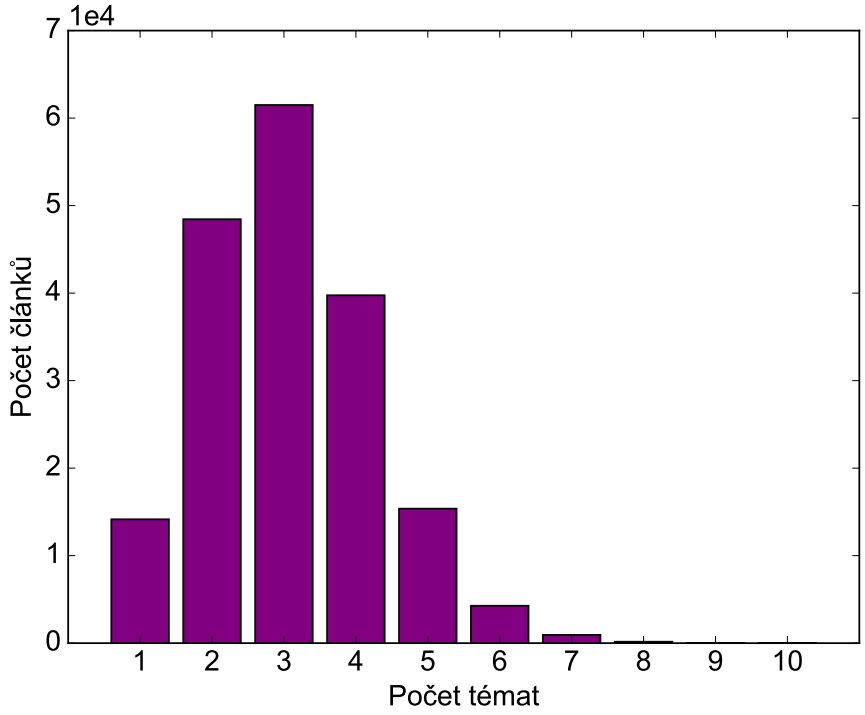
Celá práce je naprogramována pomocí jazyka *Python* s využitím knihoven *NumPy, SciPy*, *scikit-learn*, *Keras* a *TensorFlow*.

3.1 Zpracování dat

Vstupními daty jsou dva soubory (trénovací a testovací data) s českými zpravodajskými články. Na prvním řádku každého článku v souboru jsou přidělená témata od člověka. Data jsou naformátována tak, že na každém dalším řádku je jeden odstavec daného článku. Unikátních témat je v trénovacím korpusu[1] 577. Každý článek má přidělené alespoň jedno téma a maximální počet přidělených témat článku je 10. Četnosti témat jsou vizualizovány na obrázku 3.1. Nejčastěji přiřazené téma je *USA*, které přísluší 16367 článkům. Na opačném konci je téma přiděleno 7 článkům

[1] Viz poznámka na straně 5

a je jím *Super_Bowl*. V trénovacím korpusu je 205128 článků, v testovacím 43847 článků.



Obrázek 3.1: Rozložení četností témat v korpusu

3.1.1 Held out data

Nejprve je nutné z trénovacích dat vyjmout tzv *held out* data (také někdy označováno jako *cross validation* data). Tato data se používají pro optimalizaci hyperparametrů[2]. V kontextu této práce je hyperparametrem práh, jenž optimalizuji s využitím F-míry 1.7.2. V tomto případě jsem vyjmul 10 % dat. V absolutních číslech se jedná o 20513 článků. Pro natrénování klasifikátorů bude použito 184615 článků. Je důležité data nenechat zároveň i v trénovacích datech. Kdybych tak neučinil, mohl bych dostat práh, který funguje dobře jen na *held out* datech.

3.1.2 Vektorizace

Dále je nutné data převést do vektorizované formy. K tomuto úkonu jsem použil třídu *TfidfVectorizer* z knihovny *scikit-learn*. *TfidfVectorizer* používá takzvanou Tf-

[2]Viz poznámka na straně 14

idf metodiku, jež je popsána v sekci 1.2.1. Tato třída spojuje funkcionalitu tříd *CountVectorizer* a *TfidfTransformer*. Pro transformaci témat do maticové formy jsem využil třídu *MultiLabelBinarizer*, která převede seznam témat do binární matice Y. Použití tříd je vysoce intuitivní:

Kód 3.1: Použití tříd TfidfVectorizer a MultiLabelBinarizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
...
vectorizer = TfidfVectorizer(ngram_range=(1, 1), sublinear_tf=True)
binarizer = MultiLabelBinarizer()
X = vectorizer.fit_transform(corpus)
Y = binarizer.fit_transform(topics)
```

Metoda *fit_transform* nastaví instancím parametry potřebné pro transformaci textu a zároveň vrátí vstupní data v maticové formě.

X je řídká matice[3] s rozměry 184615x601479. Index řádku je jakýsi identifikátor jednotlivých článků. Tento identifikátor je validní i v matici Y, jež je popsána níže. Index sloupce pak identifikuje tokeny. Na pozicích v matici jsou vypočteny váhy tokenů.

Rozměry matice Y jsou 184615x577. Hodnoty jsou binární a reprezentují, zda dané téma bylo přiřazeno článku či nikoliv.

3.2 Trénování klasifikátoru

V této části práce je zdokumentována praktická realizace trénování SVM klasifikátoru. Teorii k trénování SVM jsem podrobně popsal v sekci 1.5.1.

3.2.1 Trénování na článcích

Z nadpisu této kapitoly lze usoudit, že je v této části SVM natrénováno na celých článcích. To znamená, že je klasifikátoru předložen vektor celého článku, nikoliv vektor jeho částí.

[3]Matice, jež má většinu prvků nulových

Kód 3.2: Trénování SVM

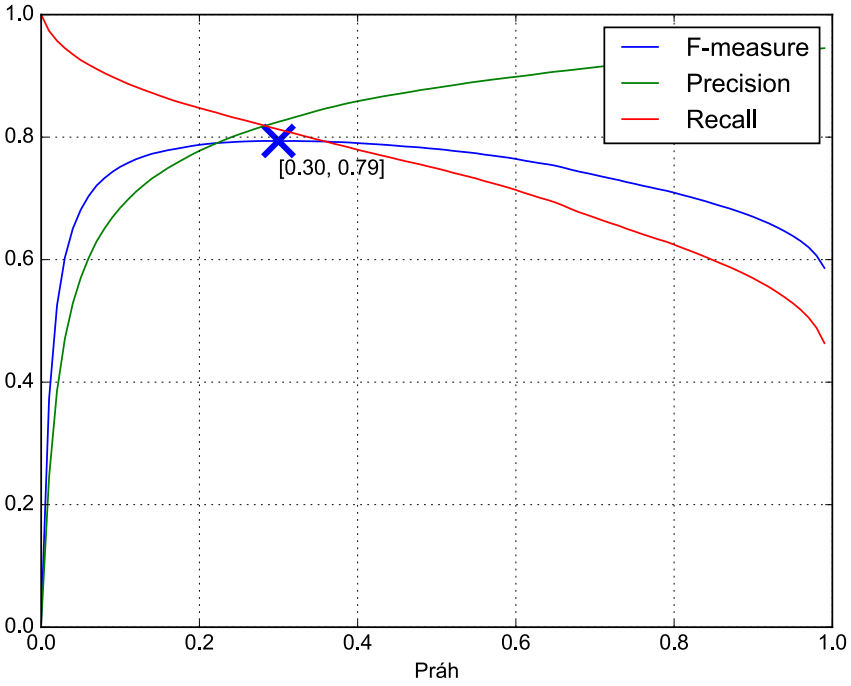
```
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.calibration import CalibratedClassifierCV
...
classifier_one_class = CalibratedClassifierCV(LinearSVC(), cv=3)
classifier = OneVsRestClassifier(classifier_one_class, n_jobs=1)
classifier.fit(X, Y)
```

Jak je vidno z kódu 3.2, je použit lineární kernel (definován v sekci 1.5.2). Je žádoucí, aby výstupem klasifikátoru byla pravděpodobnost. Z toho důvodu je instance klasifikátoru zabalena do třídy *CalibratedClassifierCV*. Tato třída je implementací kalibračního mechanismu, jenž využívá *Plattovu sigmoidální funkci* a *isotonickou regresi*.

3.2.1.1 Prahování výstupu

Pro vyhodnocení kvality klasifikace jsem použil F-míru. Jak je popsáno v sekci 1.7.2, tato metrika je vypočtena ze dvou čísel: *Precision* a *Recall*. Obecně platí, že čím je rozdíl těchto hodnot v absolutní hodnotě menší, tím víc se nastavení klasifikátoru blíží optimu. V praktické realizaci je tento proces implementováno použitím cyklu, jenž iteruje přes množinu hodnot (v mém případě interval mezi 0 a 1 vzorkovaný po setině), ve kterém je vypočtena F-míra pro daný práh. Jelikož jsem použil kalibrovaný klasifikátor, práh je pravděpodobnost, neboli číslo v intervalu $< 0, 1 >$.

Hodnota, pro kterou je na *held out* datech nejvyšší míra, je nejlepším prahem. Tato hodnota je na obrázku 3.2 označena modrým křížkem. Pro daný případ je práh **0,3** (30 %) téměř optimální. Vidíme, že není přímo v globálním maximu, což je způsobeno vzorkováním po setině. Jakmile jsem našel optimální práh, mohu přistoupit k samotnému vyhodnocení.



Obrázek 3.2: Vývoj přesnosti, úplnosti a F-míry v závislosti na prahu

3.2.1.2 Vyhodnocení

Vyhodnocení se provádí na testovacích datech. Opět je použita F-míra a pro daný klasifikátor nabývá hodnoty $F_1 = \mathbf{0{,}766}$ (P = 0,803; R = 0,785). Hodnoty *Precision (P)* a *Recall (R)* si jsou poměrně blízké, proto můžeme říci, že nastavení prahu se blíží globálnímu optimu. Hodnota 76,6 % je poměrně dobrá, jelikož musíme brát v potaz šum v datech a nekonzistenci při určování témat lidmi.

3.2.2 Trénování na odstavcích

V sekci 3.2.1 byl obsah článků převeden do jednoho vektoru. V této sekci jsou vstupními daty vektory jednotlivých odstavců, kterým byla přiřazena témata podle následujících pravidel:

1. Každé téma článku musí být přiřazeno alespoň jednomu odstavci
2. Každý odstavec musí mít alespoň jedno téma
3. Odstavcům přidělit jen témata zdrojového článku

Tato pravidla byla aplikována na predikce témat klasifikátoru, jehož trénování je popsáno v sekci 3.2.1.

Trénováním klasifikátoru na odstavcích by mohlo být dosaženo lepších výsledků, jelikož různé odstavce často mívají zcela odlišná témata. Je velmi časté, že se například v článku pojednávajícím o sportovních výsledcích vyskytne odstavec zabývající se jen a pouze počasím. V takových situacích by měl klasifikátor pracující s odstavci podat lepší výsledky.

Nalezení optimálního prahu vstupních dat za pomocí F-míry by vyžadovalo natrénování tolika klasifikátorů, kolik by bylo možných prahů. Tento přístup by byl velmi výpočetně náročný, proto jsem otestoval dva alternativní přístupy, které tento problém obcházejí.

Prvním přístupem je naprahování podle největší mezery mezi predikcemi. Tento přístup se v literatuře nazývá *MCut* a spočívá v následujících krocích [24]:

1. Seřazení predikcí podle velikosti
2. Nalezení největšího rozdílu mezi dvěma po sobě jdoucími predikcemi
3. Průměr těchto dvou predikcí je hledaným prahem

Druhý přístup je jednodušší. Spočívá ve vyhledání nejvyšší hodnoty predikce, přičemž polovina této hodnoty je hledaným jako práh.

Trénování takto zpracovaných dat, je totožné s kódem 3.2.

3.2.2.1 Nalezení prahu a vyhodnocení

Pro nalezení prahu a vyhodnocení je potřeba *held out* a testovací data zpracovat stejně jako jsou zpracována data testovací. Proces nalezení prahu a vyhodnocení je pak totožný s procesem, jenž je popsán v sekci 3.2.

Klasifikátor, který byl natrénován na datech s tématy roztříděnými pomocí prahování podle poloviny největší predikce, má optimální práh **0,39** (39 %). Pro tento práh byla naměřena F-míra $F_1 = \mathbf{0{,}708}$ (P = 0,717; R = 0,739).

U druhé verze klasifikátoru, který je natrénovaný na datech s roztříděnými tématy pomocí největší mezery mezi predikcemi, byl nalezen práh **0,91** (91 %). Výsledek je v tomto případě horší: $F_1 = \mathbf{0{,}672}$ (P = 0,672; R = 0,672).

Vidíme, že klasifikátor natrénovaný na odstavcích dosáhl horších výsledků než klasifikátor natrénovaný na článcích. Tento fakt je pravděpodobně způsoben nedostatečnými daty. Proces třídění témat k odstavcům zanáší do dat příliš mnoho šumu, což degraduje kvalitu klasifikátoru.

3.3 Segmentace

Nyní se dostáváme k jádru této práce, kterým je segmentace textu.

3.3.1 Binární SVM

Abychom otestovali, jak důležitý je klasifikátor tématu pro segmentaci, natrénoval jsem binární SVM na kosinových vzdálenostech dvou po sobě jdoucích *tf-idf* vektorů odstavců (viz 1.2.1).

Kód 3.3: Trénování binárního SVM

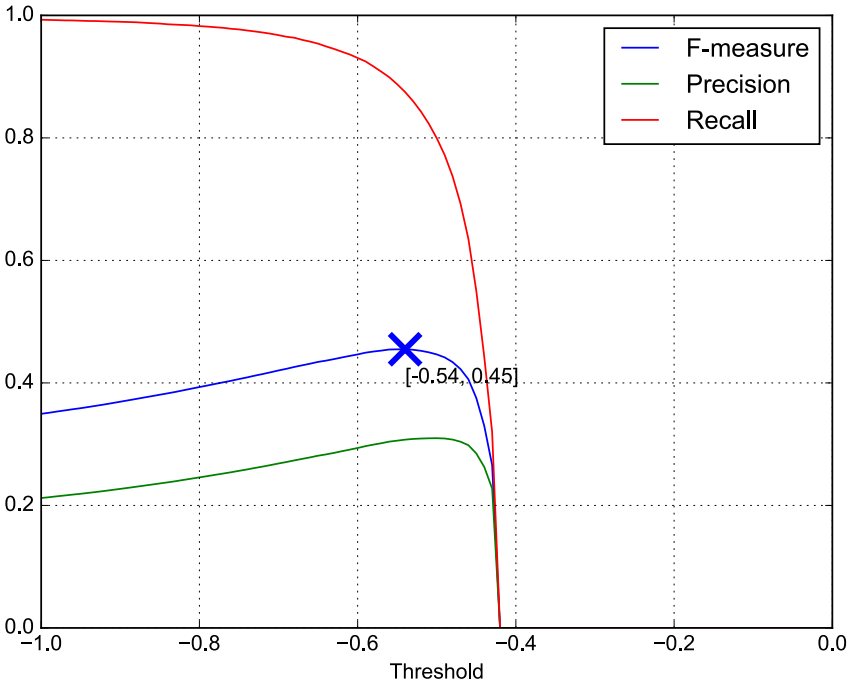
```
from sklearn.svm import LinearSV
...
classifier = LinearSVC(random_state=0)
classifier.fit(x_dists, y_true)
```

Kód trénování je velmi podobný kódu 3.2 s tím rozdílem, že *LinearSVC* není zabaleno do rozšiřujících tříd. Vstupem jsou nyní vzdálenosti a binární hodnoty informující o tématických předělech.

3.3.1.1 Vyhodnocení

Po nalezení optimálního prahu **-0,55** (na výstupu není pravděpodobnost, proto práh nemusí být v intervalu $< 0, 1 >$) F-míra nabyla hodnoty $F_1 =$ **0,444** (P = 0,296; R = 0,888). Jak už bylo zmíněno v kapitole 3.2.1.1, čím blíže jsou si hodnoty přesnosti a úplnosti tím více se nastavení klasifikátoru blíží optimu. V tomto případě jsou si i přes optimální nastavení prahu tyto hodnoty velmi vzdáleny. Když se koukneme na vývoj prahování 3.3 vidíme, že úplnost se nejprve blíží hodnotě 1, zatímco přesnost se drží na intervalu $< 0, 2; 0, 3 >$. Při dosažení určitého prahu pak všechny hodnoty

začnou padat k nule. Z takového vývoje lze usoudit, že tento klasifikátor není o mnoho lepší než triviální algoritmus[4] přidělující na každou pozici tématický předěl.



Obrázek 3.3: Vývoj přesnosti, úplnosti a F-míry v závislosti na prahu

3.3.2 Shlukovací algoritmus

Jednou možností jak k segmentaci přistupovat je použití poznatků shlukové analýzy[5]. Tato i všechny další metody popsané v této práci hodnoty z prediktoru témat používají.

3.3.2.1 K-means

Jedním z přístupů otestovaných v této práci je nasazení algoritmu k-means na predikce témat textových úseků. U mnoha shlukovacích algoritmů je potřeba předem zadat počet shluků, což je často problematické, jelikož tuto informaci většinou nemáme k dispozici.

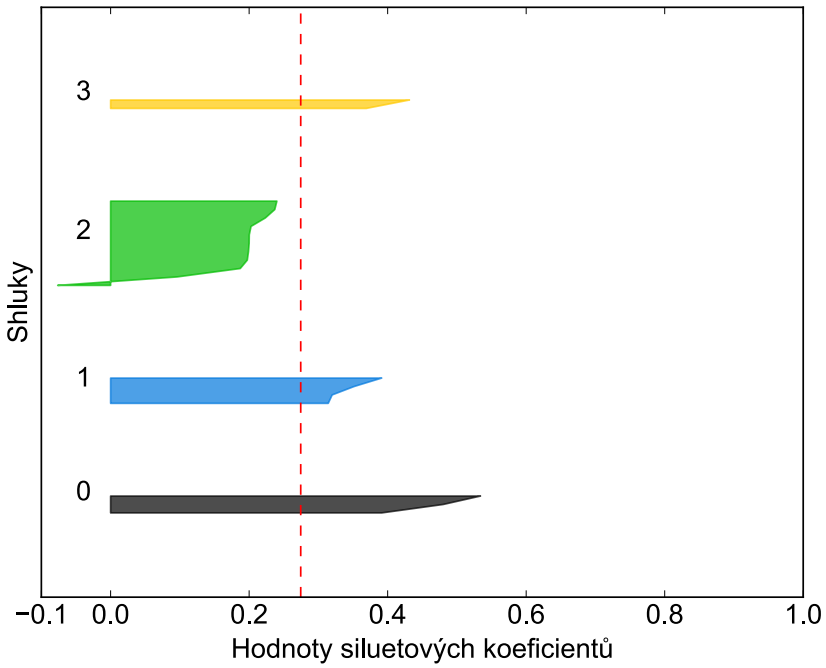
Jednou možností, jak tento problém obejít, je spuštění algoritmu pro několik

[4] Viz poznámka na straně 26
[5] Statistická metoda, sloužící k třídění prvků do skupin na základě podobnosti

různých počtů shluků a vyhodnocení kvality výstupu pomocí takzvaného *siluetového koeficientu*. Ten je definován jako:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{3.1}$$

Kde *a(i)* je průměrná vzdálenost mezi bodem i a ostatními body v rámci shluku. Hodnota *b(i)* je pak průměrná vzdálenost mezi bodem i a body, které náleží do jiných shluků.



Obrázek 3.4: Vizualizace siluetových koeficientů při použití metody K-means

Siluetový koeficient nabývá hodnot z intervalu $< -1, 1 >$ a platí, že čím vyšší hodnota je, tím lépe. Při vyhodnocování proto vybereme počet shluků, pro který je hodnota tohoto koeficientu nejvyšší.

Na obrázku 3.4 jsou znázorněné hodnoty koeficientů jednotlivých prvků. Červeně čárkovanou přímkou je označena průměrná hodnota.

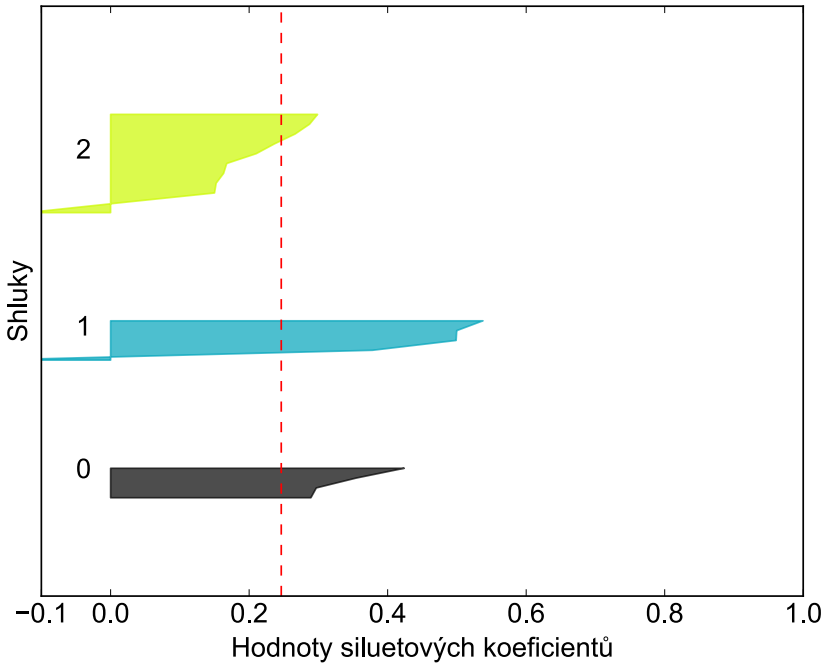
Jelikož by spuštění algoritmu na všech predikcích najednou bylo při segmentaci dlouhého textu extrémně paměťově a výpočetně náročné, pouštím algoritmus na sekvencích 20 predikcí dlouhých.

Na celém testovacím datasetu byl tento přístup ohodnocen F-mírou $F_1 =$ **0,400**

(P = 0,282; R = 0,691). Hodnota 0,4 je hodně nízká, což se ukázalo být způsobené použitím algoritmu, jenž nebere v potaz sekvenčnost dat.

3.3.2.2 Sekvenční shlukovací algoritmus

Jelikož neexistuje veřejně známý algoritmus, který by sekvenčnost dat bral v úvahu, naprogramoval jsem algoritmus vlastní.



Obrázek 3.5: Vizualizace siluetových koeficientů při použití vlastního shlukovacího algoritmu

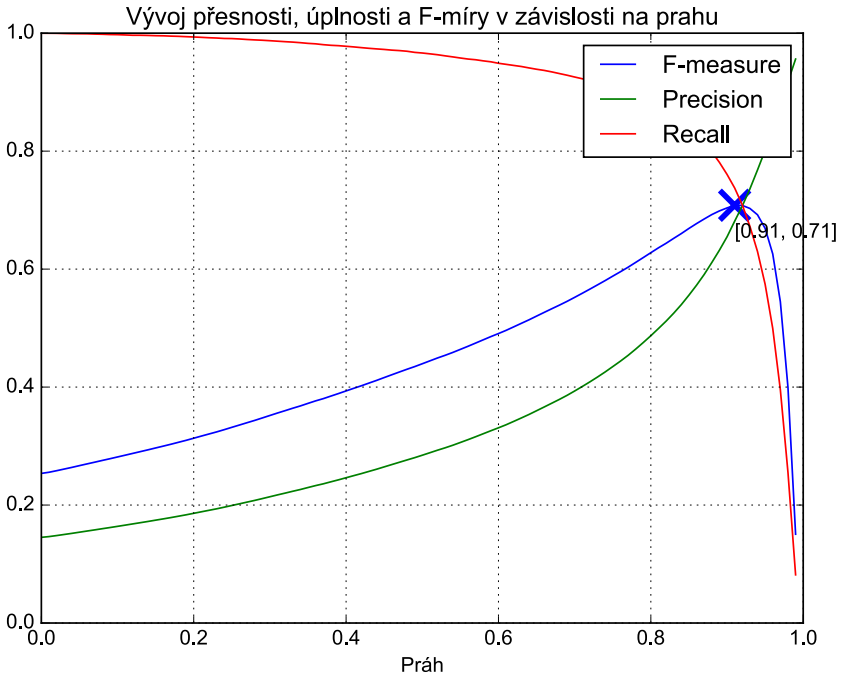
Ten funguje tak, že hledá nejmenší vzdálenost mezi po sobě jdoucími predikcemi. Predikce, jež k sobě mají nejblíže, jsou pak spojeny do jednoho shluku. Jako centra shluků jsou použity průměry predikcí. Používám zde kosinovou vzdálenost, jelikož se ukázala být pro danou aplikaci nejvhodnější (viz 3.3.3).

Pro tento shlukovací algoritmus byla změřena hodnota F-míry $F_1 = \mathbf{0{,}662}$ (P = 0,666; R = 0,658).

3.3.3 Prahování vzdáleností

Prahování vzdáleností je velmi jednoduchá metoda, která se ukázala být poměrně efektivní. Tato metoda spočívá v naprahování vzdáleností (popsáno v sekci 3.2.1) mezi predikcemi.

Pro euklidovskou vzdálenost je F-míra $F_1 = \mathbf{0{,}507}$ (P = 0,525; R = 0,490) a pro kosinovou $F_1 = \mathbf{0{,}678}$ (P = 0,663; R = 0,694).



Obrázek 3.6: Prahování kosinových vzdáleností

Vidíme, že kosinová vzdálenost podává v tomto případě mnohem lepší výsledky, což je způsobeno tím, že tato vzdálenost klade větší důraz na směr vektorů než vzdálenost euklidovská. Jelikož směr vektorů v prostoru predikcí udává téma a velikost vektoru jak moc je příslušnost k tématu silná, dává toto opodstatnění smysl. Další výhodou kosinové vzdálenosti je, že pro vektory v kladné části prostoru dává hodnotu v intervalu $< 0, 1 >$. Jelikož komponenty vektoru predikcí jsou pravděpodobnosti témat na daných pozicích, není tak nutné používat normalizaci.

3.3.4 Metody odvozené od prahování vzdáleností

Abychom lépe zachytili kontext, v jakém se daná predikce nachází, vytvořil jsem dva algoritmy, které vzdálenosti nějakým způsobem před prahováním zpracují.

První algoritmus nejprve vypočte, jak moc se aktuální vzdálenost liší od předchozích (viz kód 3.4). Toho je docíleno odečtením aktuální vzdálenosti od průměru n předešlých vzdáleností. Pro $n = 4$ je F-míra $F_1 =$ **0,392** (P = 0,316; R = 0,515).

Kód 3.4: Funkce zpracování vzdáleností

```
def slide_window(y_dists, n=4):
    y_pred = np.zeros((y_dists.shape[0], 1))
    for i in range(n, y_dists.shape[0]):
        y_pred[i] = np.abs(np.mean(y_dists[i − n:i]) − y_dists[i])
    return y_pred
```

Druhý algoritmus je velice podobný s tím rozdílem, že nepočítá rozdíl mezi aktuální vzdáleností a průměrem n předchozích, ale mezi aktuální vzdáleností a okolím velikosti ε. Pro $\varepsilon = 2$ byla změřena F-míra $F_1 =$ **0,491** (P = 0,449; R = 0,541).

Z naměřených hodnot je zřejmé, že tyto metody nejsou přínosem.

3.3.5 LSTM

Segmentace textu má vysoce sekvenční charakter (pravděpodobnost předělu v aktuálním úseku je vysoce závislá na předchozích hodnotách), což mnoho v této práci otestovaných přístupů opomíjí. Tuto vlastnosti je schopna plně využít neuronová síť typu LSTM (viz 1.6.2). Jak už bylo zmíněno na začátku této kapitoly pro vytvoření modelu použiju knihovnu *Keras*. *Keras* je API[6], díky kterému je jednodušší použít komplexní knihovny strojového učení. V této práci je tou komplexní knihovnou *TensorFlow*.

Kód 3.5: Vytvoření modelu pomocí knihovny Keras

```
from keras import Sequential
from keras.layers import LSTM, TimeDistributed, Dense
```

[6]Programovací rozhraní aplikace

```
...
model = Sequential()
model.add(LSTM(32, input_shape=(200, 577), stateful=False, return_sequences=True))
model.add(TimeDistributed(Dense(1, activation='sigmoid')))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Jak je vidět z kódu 3.5, na výstupu je aktivační funkce *sigmoid*. Tato funkce zajistí, že je výstup normalizován na interval $< 0, 1 >$. Pokud se výstupní hodnota blíží *1*, model tím predikuje, že se na dané pozici vyskytuje tématický předěl.

3.3.5.1 Zpracování dat

LSTM sítě na vstupu očekávají data ve tvaru *[počet sekvencí, délka sekvence, počet příznaků]*. Vstupní data mají rozměry *[délka jedné dlouhé sekvence, počet příznaků]*, proto je potřeba je rozdělit na více sekvencí.

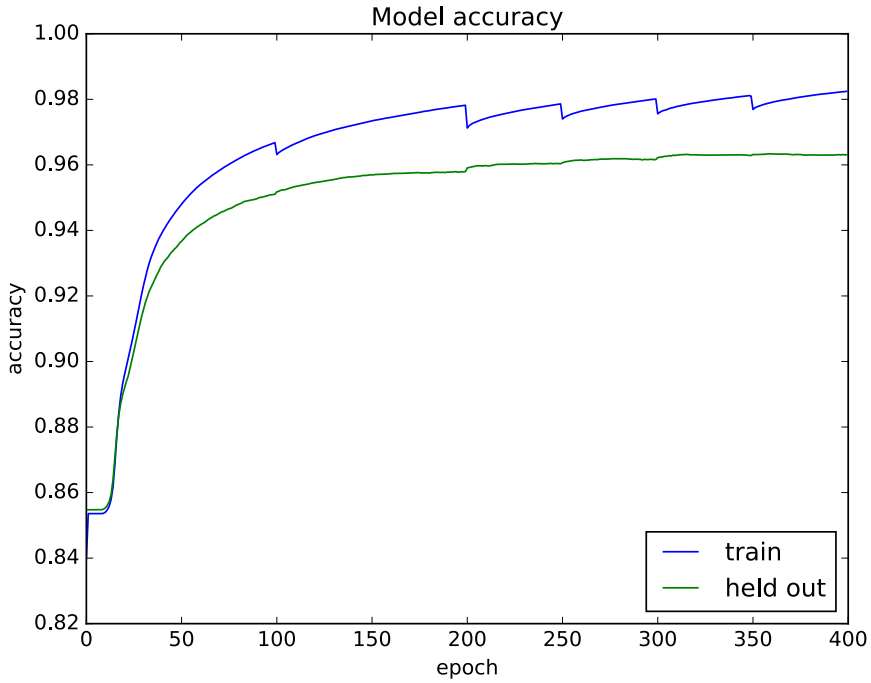
Délka sekvence určuje počet vektorů v jedné sekvenci. Po dosažení konce sekvence se stav sítě vymaže a síti je předložena sekvence nová. Počet příznaků je buď 577 v případě, kdy je síť trénována na nezpracovaných predikcích SVM nebo 1 pokud je trénována na kosinových vzdálenostech predikcí.

3.3.5.2 Trénování sítí

Jak jsem se zmínil v minulé podsekci, pro natrénování sítě jsem použil 2 typy vstupních dat. Prvním typem jsou nezpracované predikce dimenze 577 a druhým typem vzdálenosti těchto predikcí, jež mají dimenzi 1.

Síť jsem trénoval tak dlouho, dokud docházelo ke zlepšení úspěšnosti 1.7.1 na *held out* datech. Může se stát, že se úspěšnost stále zlepšuje na trénovacích datech a zároveň se zhoršuje na *held out* datech. Takový vývoj úspěšnosti je typickým indikátorem přetrénování[7].

[7] Viz poznámka na straně 15



Obrázek 3.7: Vývoj úspěšnosti v průběhu trénování

Na obrázku 3.7 je vyobrazen vývoj úspěšnosti při trénování sítě na nezpracovaných predikcích z SVM. Vidíme, že po 400 epochách se přestává úspěšnost na *held out* datech zlepšovat, proto jsem trénování ukončil. Epochou je v kontextu trénování modelů myšleno jedna dopředná a zpětná propagace na všech trénovacích vzorcích. Abych omezil pravděpodobnost přetrénování, zamíchal jsem po 100, 200, 250, 300 a 350 epochách články, čímž jsem vytvořil sekvence nové. Událost zamíchání je na obrázku zřetelně vidět, jelikož dochází k propadu úspěšnosti na trénovacích datech.

Síť je natrénovaná na predikcích SVM. Tyto predikce byly vytvořeny na trénovacích datech, proto jsou značně zkresleny. Jak je vidět v následující podsekci, síť i přesto dosahuje dobrých výsledků. To může být způsobeno tím, že LSTM dává důraz na trend v sekvencích, nikoliv však na konkrétní hodnoty predikcí.

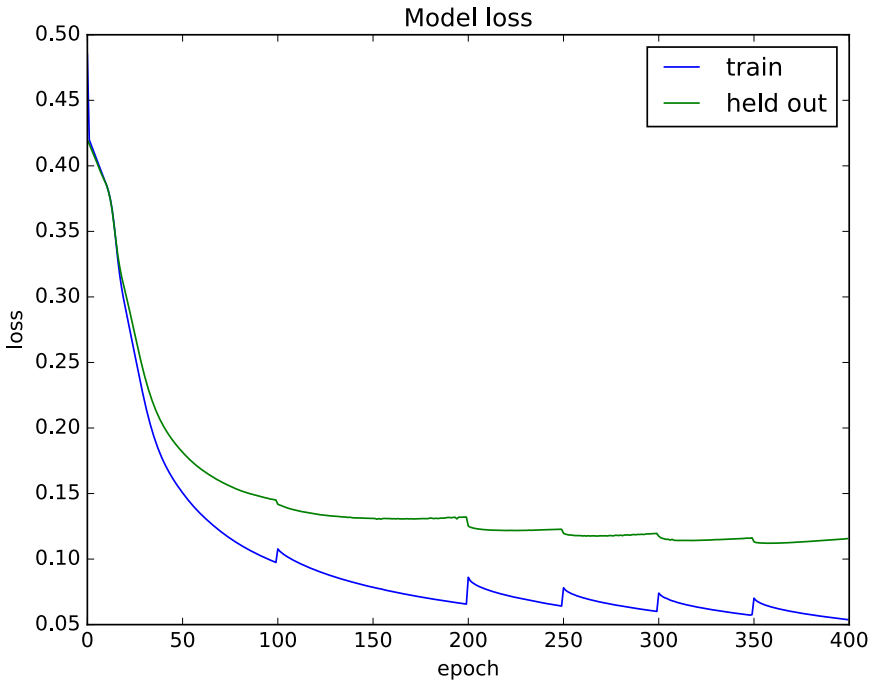
Na obrázku 3.8 je znázorněn vývoj ztráty. Jak by se dalo očekávat, tento vývoj má oproti vývoji úspěšnosti inverzní charakter. Použitý typ ztrátové funkce[8] se anglicky nazývá *cross entropy* a její definice pro N trénovacích vzorků je následující [25]:

[8]Funkce mapující odhad ztráty na množinu reálných čísel

$$J(w) = -\frac{1}{N}\sum_{n=1}^{N}[y_n \log(\hat{y}_n) + (1 - y_n)\log(1 - \hat{y}_n)] \tag{3.2}$$

Kde y_n je referenční a $\hat{y}_n$ predikovaná hodnota pro n-tý trénovací vzorek.

Tato ztrátová funkce zjednodušeně řečeno udává, jak moc se rozložení pravděpodobnosti výstupu modelu liší od referenčního rozložení.



Obrázek 3.8: Vývoj ztráty v průběhu trénování

3.3.5.3 Vyhodnocení

Jak již bylo popsáno v sekci 1.7.1, úspěšnost podává na zkosených datech[9] zkreslené výsledky. Přítomnost tématický předělů těmito zkosenými daty nepochybně je (na většině pozicích předěl není), proto pro finální vyhodnocení použiji F-míru 1.7.2.

Síť natrénovaná na kosinových vzdálenostech byla ohodnocena F-mírou $F_1 =$ **0,727** (P = 0,738; R = 0,716). Druhá verze natrénovaná na nezpracovaných predikcích dosáhla ještě lepší hodnoty $F_1 =$ **0,854** (P = 0,886; R = 0,824).

[9]Viz poznámka na straně 22

3.4 Srovnání výsledků

V tabulce 3.4 jsou seřazeny výsledky různých segmentačních přístupů. *WindowDiff* a P_k metriky udávají míru chybovosti algoritmu, proto mají tyto metriky oproti F míře inverzní charakter. Délka okénka k byla zvolena na hodnotu poloviny průměrné délky tématických celků 2.2.1.1, proto se hodnota P_k triviálního algoritmu[10] blíží k 0,5. Průměrná délka článku je 7,22 odstavců. Parametr k je proto nastaven na hodnotu **4**. Tento algoritmus přiděluje hodnoty náhodně podle rovnoměrného rozdělení.

	F_1	P_k	*WindowDiff*
Triviální algoritmus	0,216	0,479	0,836
Prahování rozdílu od předešlých vzdáleností 3.3.4	0,392	0,339	0,495
K-means 3.3.2.1	0,400	0,387	0,626
Binární SVM 3.3.1	0,444	0,376	0,660
Prahování rozdílu od okolí 3.3.4	0,491	0,299	0,418
Prahování euklidovské vzdálenosti 3.3.3	0,507	0,337	0,404
Sekvenční shlukovací algoritmus 3.3.2.2	0,662	0,240	0,325
Prahování kosinové vzdálenosti 3.3.3	0,678	0,234	0,284
LSTM - kosinové vzdálenosti 3.3.5	0,727	0,202	0,244
LSTM - nezpracované predikce 3.3.5	**0,854**	**0,105**	**0,133**

Tabulka 3.1: Tabulka výsledků

Vidíme, že nejvyšší dosažená hodnota F míry **0,854** je o 17,6 % lepší než druhý nejlépe fungující přístup nepoužívající LSTM. Toto zjištění potvrzuje domněnku, že model využívající sekvenčnosti dat podá výrazně lepší výsledky.

[10] Viz poznámka na straně 26

Závěr

V úvodu této práce jsem uvedl několik cílů. Cílem prvních dvou kapitol byl popis teoretického základu použitých klasifikačních algoritmů. Znalosti, které jsem při vypracování těchto kapitol získal, jsem intenzivně využil při praktické realizaci, proto věřím, že účel teoretických částí byl naplněn..

Hlavním cílem práce byla implementace různých segmentačních algoritmů. V tabulce 3.4 je uvedeno celkem 9 způsobů segmentace textu (triviální algoritmus nepočítám), které dosahují značně variabilních výsledků. V případě přístupu užívající LSTM je výsledek poměrně dobrý, z toho důvodu si troufám říci, že hlavní cíl této práce byl splněn.

Při vyhodnocení přístupů jsem ověřil známé pravidlo, které říká, že jednoduchá řešení většinou fungují lépe a robustněji než řešení komplexní. Implementací vlastního shlukovacího algoritmu jsem strávil mnoho času a ve výsledku funguje hůř než v této práci nejjednodušší řešení, kterým je naprahování kosinových vzdáleností. Výjimku z tohoto pravidla však tvoří rekurentní neuronové sítě typu LSTM, které jsou vysoce komplexní, ale podávají zdaleka nejlepší výsledky. Kolem moderních modelů typu LSTM, které jsou označovány jako modely hlubokého strojového učení, je v posledních letech mnoho nadšení. Toto nadšení se ukázalo být opodstatněné.

Rekurentní neuronové sítě by zároveň mohly být i cestou k dosažení ještě lepších výsledků. Nevýhodou modelů hlubokého strojového učení je, že existuje velké množství hyperparametrů[11], kterými se dá ovlivnit chování modelu. Počítačová optimalizace těchto parametrů vyžaduje obrovský výpočetní výkon, proto jsou pro efektivní nastavení vyžadovány expertní znalosti. Z toho důvodu se dá usoudit, že nastavení modelu pravděpodobně není optimální. Výhodou těchto moderních přístupů oproti

[11] Viz poznámka na straně 14

tradičním však je, že při volbě dostatečně komplexní struktury zvětšení trénovacího datasetu téměř vždy přinese zlepšení prediktivních schopností modelu [26]. Proto je pravděpodobné, že zvětšení datasetu je cesta k dosažení ještě lepších výsledků.

Seznam použitých zdrojů

[1] Sepp Hochreiter a Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (lis. 1997), s. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[2] *The Bag of Words representation.* URL: http://scikit-learn.org/stable/modules/feature_extraction.html#the-bag-of-words-representation (cit. 02. 02. 2018).

[3] Christopher D. Manning, Prabhakar Raghavan a Hinrich Schütze. *Introduction to Information Retrieval.* New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521865719, 9780521865715.

[4] Adam Berger a John Lafferty. "Information Retrieval As Statistical Translation". In: *SIGIR Forum* 51.2 (srp. 2017), s. 219–226. ISSN: 0163-5840. DOI: 10.1145/3130348.3130371. URL: http://doi.acm.org/10.1145/3130348.3130371.

[5] *One Vs The Rest.* URL: http://scikit-learn.org/stable/modules/multiclass.html#one-vs-the-rest (cit. 03. 02. 2018).

[6] Jason Brownlee. *Supervised and Unsupervised Machine Learning Algorithms.* Zář. 2016. URL: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

[7] Alexandre Kowalczyk. *Support Vector Machines Succinctly.* 2017.

[8] *Optimal hyperplane with margins.* URL: https://commons.wikimedia.org/wiki/File:Separatrice_lineaire_avec_marges.svg (cit. 03. 02. 2018).

[9] Kristin P. Bennett a Erin J. Bredensteiner. "Duality and Geometry in SVM Classifiers". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, s. 57–64. ISBN: 1-55860-707-2. URL: http://dl.acm.org/citation.cfm?id=645529.657972.

[10] Thorsten Joachims. "Text Categorization with Support Vector Machines: Learning with Many Relevant Features". In: *Proceedings of the 10th European Conference on Machine Learning*. ECML'98. Chemnitz, Germany: Springer-Verlag, 1998, s. 137–142. ISBN: 3-540-64417-2, 978-3-540-64417-0. DOI: 10.1007/BFb0026683. URL: https://doi.org/10.1007/BFb0026683.

[11] Rosenblatt F. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological Review* (1958).

[12] *Deep Recurrent Neural Network (RNN).* URL: https://dbo7ngidws8s9.cloudfront.net/wp-content/uploads/2017/11/rnn_network.svg (cit. 27. 02. 2018).

[13] Razvan Pascanu, Tomas Mikolov a Yoshua Bengio. "On the Difficulty of Training Recurrent Neural Networks". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, s. III-1310–III-1318. URL: http://dl.acm.org/citation.cfm?id=3042817.3043083.

[14] Christopher Olaph. *Understanding LSTM Networks*. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[15] *LSTM with peephole connections.* 2016. URL: https://kvitajakub.github.io/img/lstm-peepholes.svg.

[16] C. J. Van Rijsbergen. *Information Retrieval*. 2nd. Newton, MA, USA: Butterworth-Heinemann, 1979. ISBN: 0408709294.

[17] *Precision, recall.* URL: https://commons.wikimedia.org/wiki/File:Precisionrecall.svg (cit. 13. 02. 2018).

[18] *From binary to multiclass and multilabel*. URL: http://scikit-learn.org/stable/modules/model_evaluation.html#from-binary-to-multiclass-and-multilabel (cit. 10. 04. 2018).

[19] Matthew Purver. "Topic Segmentation". In: *Spoken Language Understanding: Systems for Extracting Semantic Information from Speech*. Ed. G. ur a R. de Mori. Wiley, 2011, s. 91–317. ISBN: 978-0-470-68824-3. URL: http://www.eecs.qmul.ac.uk/~mpurver/papers/purver11slu.pdf.

[20] Doug Beeferman, Adam Berger a John Lafferty. "Statistical Models for Text Segmentation". In: *Mach. Learn.* 34.1-3 (ún. 1999), s. 177–210. ISSN: 0885-6125. DOI: 10.1023/A:1007506220214. URL: https://doi.org/10.1023/A:1007506220214.

[21] Regina Barzilay. "Text Segmentation". In: (2005).

[22] Lev Pevzner a Marti A. Hearst. "A Critique and Improvement of an Evaluation Metric for Text Segmentation". In: *Comput. Linguist.* 28.1 (březen 2002), s. 19–36. ISSN: 0891-2017. DOI: 10.1162/089120102317341756. URL: http://dx.doi.org/10.1162/089120102317341756.

[23] Maria Georgescul, Alexander Clark a Susan Armstrong. "Word Distributions for Thematic Segmentation in a Support Vector Machine Approach". In: *Proceedings of the Tenth Conference on Computational Natural Language Learning*. CoNLL-X '06. New York City, New York: Association for Computational Linguistics, 2006, s. 101–108. URL: http://dl.acm.org/citation.cfm?id=1596276.1596296.

[24] Christine Largeron, Christophe Moulin a Mathias Géry. "MCut: A Thresholding Strategy for Multi-label Classification". In: *Proceedings of the 11th International Conference on Advances in Intelligent Data Analysis*. IDA'12. Helsinki, Finland: Springer-Verlag, 2012, s. 172–183. ISBN: 978-3-642-34155-7. DOI: 10.1007/978-3-642-34156-4_17. URL: http://dx.doi.org/10.1007/978-3-642-34156-4_17.

[25] Michael A. Nielsen. *Neural Networks and Deep Learning*. URL: http://neuralnetworksanddeeplearning.com/chap3.html.

[26] *How To Improve Deep Learning Performance*. Zář. 2016. URL: https://machinelearningmastery.com/improve-deep-learning-performance/.