

Komparativní analýza Bayesiánského a klasického přístupu k třídění a vyhledávání informací

Matěj Dvořák

1. dubna 2021

1 Teoretická část

Před existencí internetového vyhledávače Google byly vyhledávače často neefektivní: nacházely spam a nenacházely relevantní stránky s jinými formami klíčových slov. Proto se na ně uživatelé nespolehali jako na jediný způsob, jak nacházet informace na internetu. Jednou z alternativ byly webové adresáře (web directories), veřejně dostupné indexy odkazů tříděné podle tématu. Mezi takové adresáře patřilo například Yahoo! Directory nebo DMOZ.

Adresáře byly udržovány ručně na základě podnětů od uživatelů. Tento systém se s růstem počtu dokumentů na internetu ukázal neúnosným, zatímco vyhledávače šlo škálovat. Ve výsledku tak dnes DMOZ ani Yahoo! Directory oficiálně neexistují a dobrovolníky vedené Curlie (vycházející z DMOZ) je těžce zastaralé, značná část odkazů už ani nefunguje. Výsledkem je dnešní stav, kde má na vyhledávání téměř monopol Google.

Vyhledávače na základě klíčových slov ale také nejsou ideální. Dobře si poradí s konkrétními požadavky (např. chybová hláška nebo název restaurace). Necháme-li stranou problematiku jejich monetizace, stále tu je a) existence SEO, celého průmyslu založeného na snaze vyhledávací algoritmy obehřát, a podstatněji b) omezená schopnost vyhledávače pracovat do hloubky se širokým dotazem. Požadavku o několika slovech odpovídá řada dokumentů, výběr z nich ale probíhá arbitrárně nebo na základě řady faktorů, jako jsou stáří odkazu, další klíčová slova, nebo informace o vyhledávajícím uživateli (např. lokace). Někdy uživatel nemusí správné klíčové slovo znát.

Adresáře místo toho fungují na základě tématických okruhů. Je-li adresář dobře veden, každý okruh obsahuje řadu podokruhů korespondujících k různým podtématům. Adresář tak slouží jako encyklopedie a počáteční bod pro ruční vyhledávání informací.

Problém s udržováním adresářů je ale stále platný. Tento projekt si klade za cíl jej částečně automatizovat - vyhledávat texty podobné již přidaným stránkám. Uživatel, manažer adresáře, by pak musel jen adresář prohlubovat a složky dále dělit.

2 Implementační část

2.1 Klasifikace

Program funguje na základě statistické metody Naive Bayes, kterou se řídí při rozhodování, kam daný dokument zařadit. Nejjednodušším případem je rozhodování mezi dvěma kategoriemi.

2.1.1 Dvě kategorie

Dokument modelujeme jako seznam slov v něm obsažených (viz část Tokenizace). U kategorií vybudujeme součet pro všechny dokumenty v nich obsažené korpus, v němž je uveden počet výskytů každého slova. Při porovnávání dvou kategorií pracujeme s normalizovanými korpusy. U většího z korpusů podělíme výskyty všech slov stejnou konstantou tak, aby měly oba výsledné korpusy dohromady stejný počet slov. U každého slova určíme počet jeho výskytů v obou korpusech. Označme výskyt slova v kategorii A a , v

kategorii B *b*. Pravděpodobnost pro dané slovo, že dokument jej obsahující patří do kategorie A, určíme jako:

$$P = \frac{a + k}{a + b + 2k}$$

Ve výpočtu zahrnujeme konstantu *k*, takzvaný "smoothing parameter", podstatnou zejména v případě, že *a* nebo *b* je velmi malé. Program využívá konzervativní Laplace smoothing, kde *k* = 1 (Lidstone smoothing, nižší hodnoty *k*, by v takových případech vedlo k vyšší váze takových slov). Takto určíme pravděpodobnosti pro všechna slova v dokumentu. Dále vybereme relevantní klíčová slova.

Nevyužíváme všechna slova v dokumentu, protože bychom mohli zesílit šum. Pro každou z kategorií uvažujeme jen *X* nejsilnějších slov proti kategorii, kde *X* je polovina počtu slov nad stanovenou "zajímavostí" (konkrétně slova s pravděpodobností pro jednu z kategorií pod 0.2). *X* má nastavenou minimální hodnotu, takže vždy uvažujeme alespoň 12 slov, ale většinou je zajímavých slov více. Smysl tohoto výběru je snaha o nalezení stejného množství důkazů proti oběma stranám a porovnání jejich celkové síly.

Skóre pro tato klíčová slova vynásobíme. Tento proces provedeme stejně pro kategorie v opačném pořadí. Výsledné pravděpodobnosti pro obě kategorie vynásobíme společnou konstantou tak, aby se sečetly na 1 (tomuto dále v textu budeme říkat "normalizace pravděpodobnosti", NP).

2.1.2 Zobecnění pro N kategorií

Problém zařazení do jedné z několika kategorií má několik řešení. Při volbě modelu jsem se zejména řídil požadavkem, aby obsah jedné kategorie neovlivnil rozhodování mezi ostatními kategoriemi. Proto například nevolím metodu, kde se každá kategorie srovná s celým vzorkem, a vyhraje ta s nejlepším výsledkem.¹ Každá kategorie by byla v takovém případě součástí všech rozhodování. Pokud by například jedna z kategorií byla "úplně mimo", všechna ostatní skóre by mohla vyjít poblíž jedné a rozlišovací schopnost by se ztratila. Toto je případ např. nejvyšší úrovně přiloženého rozdělení, kde by takto působily legální dokumenty.

Namísto toho začínám nalezením pravděpodobností pro každou dvojici kategorií. Dále vycházím z předpokladu, že správná kategorie by měla porazit všechny ostatní. Jakákoli kategorie tedy může být sama o sobě kontrolní pro jakoukoli jinou. U každé kategorie získáme výslednou pravděpodobnost příslušnosti vynásobením pravděpodobností proti všem ostatním kategoriím a následnou NP všech výsledků.

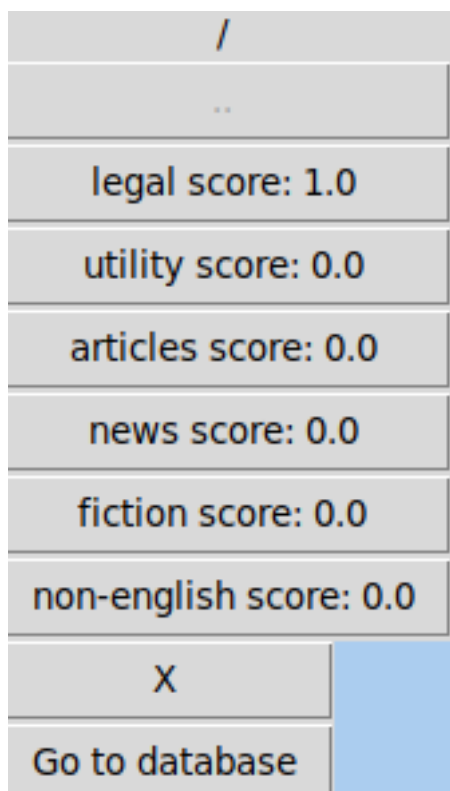
	articles	fiction	legal	news	non-english	utility	
articles		0.0	-237.4	0.0	0.0	-0.065	-237.465
fiction	-321.719		-719.083	-156.049	0.0	-257.189	-1454.04
legal	0.0	0.0		0.0	0.0	0.0	0.0
news	-44.469	0.0	-404.777		0.0	-65.507	-514.753
non-english	-735.722	-480.616	-1171.384	-607.362		-705.945	-3701.029
utility	-2.769	0.0	-222.496	0.0	0.0		-225.265

Na této ukázce je vidět klasifikace Wikipedia Terms of Use. Čísla v tabulce jsou logaritmy jednotlivých pravděpodobností. Jak je vidět, kategorie legal porazila všechny ostatní kategorie a jejich vzájemná skóre už nejsou podstatná.

Další výhodou tohoto systému je fakt, že výstupem je stále pravděpodobnost, nikoli pouze pořadí.

¹Tuto metodu používá, pokud jsem to pochopil (poznámka: ověřit) například Klassify: <https://github.com/fatihelikli/klassify>

2.1.3 Hierarchický systém



Kategorie může obsahovat podkategorie. Při rozhodování se berou v úvahu všechny dokumenty v kategorii a všech úrovních jejích podkategorií. Klasifikace pak může být víceřadová. Na obrázku je vidět okno při vysvětlení klasifikace, kde je možné kliknutím na příslušné tlačítko pokračovat v klasifikaci v libovolné podkategorii. Další části programu využívají funkci place, která pro daný text najde nejpravděpodobnější umístění (v každém kroku se dá cestou s nejvyšší pravděpodobností).

2.1.4 Benchmark klasifikace

Tento systém jsem testoval na často užívaném datasetu 20 Newsgroups². Hierarchie jsem sestavil dle doporučení autora datasetu³. Výsledná přesnost byla 6093 z 7528 dokumentů klasifikovaných správně, tj. zhruba 81%, což je srovnatelné s alternativními technologiemi. Naive Bayes s dodatečnými "vylepšeními" jako stemming (zjednodušení na základní gramatické tvary slov) a odstranění stop words (velmi častých slov bez významového obsahu) získal (dle prvního zdroje) výsledek 6100 z 7528, z čehož usuzuji, že efekt těchto technologií je zanedbatelný v rámci chyby měření. K podobnému výsledku dochází i většina literatury, např. zde⁴ dochází autoři k výsledku, že v angličtině různé lematizační algoritmy vedou k menšímu korpusu, ale vždy za cenu poklesu efektivity. Na škále, na které pracuje tento projekt, nemá zmenšování korpusu prioritu.

2.2 Crawler

Systém crawleru vychází z předpokladu, že stránky budou spíše odkazovat na stránky s podobným tématem. Crawler vychází z počáteční URL adresy a cílové kategorie. Algoritmus crawleru v pseudokódu: 1. Ze zatím prošlých URL vybereme to s nejlepším skóre. 2. Najdeme všechny odkazy vedoucí z tohoto URL. 3. Vyloučíme všechny nefunkční odkazy⁵ a všechny odkazy vedoucí na již nám známé domény. 4.

²<https://ana.cachopo.org/datasets-for-single-label-text-categorization>

³<http://qwone.com/~jason/20Newsgroups/>

⁴Toman, Michal & Tesar, Roman & Jezek, Karel. (2006). Influence of Word Normalization on Text Classification.

⁵Odkazy na nefunkční nebo nedostupné stránky, ale také stránky zakazující funkci crawleru souborem robots.txt.

Stáhneme všechny zbylé cílové stránky, oskórujeme. Vybereme z nich tu s nejlepším skóre. Skóre měříme v náhodném pořadí, je-li odkazů moc, můžeme po daném počtu přestat. 5. Na nové doméně několikrát (počet je nastavitelný) sledujeme odkazy zůstávající uvnitř domény, vždy opět na ten s nejlepším skóre. Všechna URL po cestě ukládáme do fronty. 6. Opakujeme kroky 1-5 tolikrát, kolikrát je specifikováno. [discovered složka, ale tu ještě budu trochu upravovat]

2.3 Zdrojová data

2.3.1 Tokenizace

Většina dokumentů je získána stažením HTML z internetu. V takovém případě nejprve odstraníme všechny text v tazích `jscript` nebo `style` a následně všechny HTML tagy. Dále text zbavíme diakritiky a všech nealfanumerických znaků. Všechny odstraněný text nahrazujeme mezerami. Výsledek rozdělíme na tokeny podle mezer.

2.3.2 Opakované tokeny

Během vývoje jsem zkoušel dva extrémní způsoby řešení opakování tokenů. Buď opakování nepovažujeme za problém a tokeny počítáme vícekrát, nebo počítáme každý token pouze jednou. V prvním případě může hromadné opakování tokenů na jedné stránce pokrýt vnímání tokenů (což se například stalo s malými čísly, která disproporčně naznačovala matematiku, když přitom může jít např. o nadpisy kapitol), ve druhém případě nefungují častá slova ve dlouhých textech (jako je například kategorie fiction). Proto text rozdělujeme na úseky o dané délce (několika stovek slov), ve kterých tokeny neopakujeme, mezi nimi ale ano. [až bude vyřešené #89, tak detaily o vysvětlovači]

2.3.3 Boilerplate

[Doplnit podle toho, jestli tam někde něco nepotřebuje opravit]

2.4 Uživatelské prostředí

Grafické prostředí je limitováno schopnostmi knihovny LTK, v rámci nichž funguje analogicky k procházení souborového systému. Ježto nemusí uživateli zobrazovat dokumenty ani velké množství kategorií najednou, je na obrazovce hodně volného místa, takže jsou všechny možnosti zobrazeny najednou namísto ukrytí do submenu. [až bude opravené #94, tak screenshot upravovátoru databáze].

Hlavní netradiční částí UI je systém přesouvání souborů a kategorií. Ježto v knihovně není jednoduché vytvořit click-and-drag, systém běžně používaný pro přesouvání, inspiroval jsem se na C2 Wiki a využil systém interně nazývaný Bucket: soubory nebo složky může uživatel zkopírovat do části Bucket, ve které zůstávají při pohybu skrz souborový systém, a ze které mohou být přesunuty do jiného místa. Umožňuje to jednodušší přesuny mezi vzdálenými kategoriemi. Podobně přesouváme i celé kategorie. Tlačítka s módy umožňují výběr mezi přesunem, smazáním, nebo jen odstraněním z části Bucket.

2.5 Design

2.5.1 Použité technologie a knihovny

Program jsem se rozhodl vytvořit v jazyce Common Lisp. Roli při výběru hrály: kombinace funkcionálního a imperativního paradigmatu, rychlost výpočtů, jednoduchá prefixová syntaxe vhodná pro zápis matematických vzorců na více řádků (což usnadňuje práci s výpočty), a předchozí zkušenost se souvisejícími problémy v tomto jazyce. Klasifikace je naprogramována převážně funkcionálně, crawler převážně imperativně.

LTK⁶: Grafické rozhraní. Jde o port Tcl/Tk pro Common Lisp a koncepčně nejjednodušší řešení, ale některé funkce v knihovně chybí (zejména volba barvy tlačítka a okno pro výběr souboru - které jsem ale vytvořil vlastní a snad funguje).

⁶<http://www.peter-herth.de/ltk/index.html>

Drakma⁷: Stahování HTML. Nejprve jsem používal knihovnu Dexador⁸, rozšíření Drakma, ale ukázalo se, že trpí chybou, která způsobuje únik paměti. Tuto chybu jsem nahlásil⁹, ale knihovna není udržovaná a nepřišla žádná reakce.

QURI¹⁰ umožňuje jednak úpravy znaků v názvech (např. `example.com/B%C3%A1g%C5%99i` -> `example.com/Bágři`), jednak práci s odkazy - najde cíl odkazu i pro pasti, jako je dvojí lomítko pro relativní protokol, a dokáže obecně najít klíčovou část domény pro odkazy (mimo jiné pozná, že `.co.uk` je jen jedna koncovka).

Plump¹¹ dokáže rozklíčovat escaped znaky v HTML (`’`; `-'`).

Trivial-timeout¹² zastaví načítání nereagující stránky.

Cl-strings¹³ obsahuje řadu utilit pro práci s textem.

Alexandria¹⁴ umožňuje kopírování struktury hash table.

2.5.2 Ukládání dat

Program má tři druhy dat: Korpusy, konfigurační data ke složkám, a zdrojové dokumenty. Zdrojové dokumenty jsou z nich zdaleka největší, proto je každý dokument uložen v samostatném souboru. Při vložení souboru se vytvoří jeho tři verze - hrubý text tak, jak byl vložen, text očištěný dle tokenizace, a text s odstraněnými duplikátními prvky (viz Boilerplate). Tyto zpracované verze textu by bylo lze vytvořit znovu, např. v případě změny povolených znaků.

Ostatní data nepřesahují pro typické používání jednotky MB, jsou tedy uloženy pro všechny kategorie dohromady v jednom souboru. To má výhodu jednoduchého exportu, viz manuál.

2.5.3 Databáze?

Data program při spuštění načte ze souborů a opět je do nich uloží, pokud uživatel uloží změny. Jinak má celý korpus v pracovní paměti, což zrychluje klasifikaci.

V současném stavu má korpus 8.1 MB. Obsahuje kolem 100000 různých slov a 200 kategorií. Počet různých slov už řádově nevzroste, např. Oxford English Dictionary obsahuje kolem 600000 slov¹⁵, ale řada z nich je reálně nevyužívaná. Počet kategorií může růst s vývojem. Spíše ale porostou počty slov, což na velikost dat prakticky nebude mít vliv. Na problémy by mohl tento systém narazit při řádově větším počtu kategorií.

3 Technická dokumentace

3.1 Požadavky spuštění

Pro GUI je nutný grafický systém Wish z Tk. Instalovat lze např. pomocí APT: `"sudo apt install tk"`. Pro kompilaci ze zdroje je také potřeba stejně instalovat kompilátor SBCL.

3.2 Navigace

Program může být spuštěn jako jediný spustitelný soubor nebo kompilací zdrojového kódu (a voláním `(main)`). Po spuštění se uživatel dostane do klasifikačního okna, kde může zjistit klasifikaci vloženého dokumentu, případně si ji nechat vysvětlit (zobrazit pravděpodobnostní tabulky a klíčová slova).

Druhé prostředí umožňuje úpravy databáze trénovacích dat. Uživatel se může pohybovat mezi jednotlivými kategoriemi, zobrazit v nich obsažené soubory, komentáře k hierarchickému systému, procházet korpus. Může soubory přidávat (buď pomocí URL odkazu, nebo vepsáním, nebo vložním souboru). Může také

⁷<https://github.com/edicl/drakma>

⁸<https://github.com/fukamachi/dexador>

⁹<https://github.com/fukamachi/dexador/issues/97>

¹⁰<https://github.com/fukamachi/quri>

¹¹<https://github.com/Shinmera/plump>

¹²<https://common-lisp.net/project/trivial-timeout/>

¹³<https://github.com/diogoalexandre/franco/cl-strings>

¹⁴<https://gitlab.common-lisp.net/alexandria/alexandria>

¹⁵<https://www.oed.com/>

přesouvat, přejmenovávat, a mazat kategorie. Vše je uloženo najednou tlačítkem "uložit změny" (Save Changes), při zavření okna vyskočí možnost uložit případné neuložené změny.

[Třetí prostředí umožňuje spuštění crawleru a prezentaci jeho výsledků (až bude #23).]

Mezi prostředími lze vždy přejít tlačítkem ve spodní části okna [až bude uklizené tlačítko, kam patří].