ConcordanceCrawler

Závěrečná zpráva ročníkového projektu

Toto je závěrečná zpráva a dokumentace projektu ConcordanceCrawler. Jedná se o ročníkový projekt, který vypracoval Dominik Macháček v letním semestru 2015 na MFF UK v Praze. Vedoucím projektu byl Mgr. Vincent Kríž.

Zadání

Zvolme libovolné cílové slovo v libovolném jazyce. Cílem práce je z Internetu automaticky extrahovat věty, které obsahují zadané cílové slovo. Hlavním cílem je navrhnout a implementovat open-source aplikaci, které na vstupu zadáme cílové slovo a počet vět. Aplikace se následně pokusí požadované množství konkordancí vyhledat a stáhnout.

Úkolem řešitele bude především:
1) analyzovat a evaluovat vhodné Internetové zdroje pro získávání konkordancí;
(2) analyzovat a evaluovat dostupné knihovny v Pythonu pro prohledávaní webu;
(3) analyzovat a evaluovat strategie pro automatickou segmentaci věty;
(3) navrhnout a implementovat:
a) knihovnu pro vyhledávání a stahovaní relevantních dokumentů;
b) knihovnu pro extrakci věty z dokumentu;
c) knihovnu pro určení jazyka a kódování věty;
d) konzolovou aplikaci pro Linux, která umožní uživatelům zadat cílové slovo a uložit extrahované věty;
e) webové rozhraní, které umožní uživatelům zadat extrakční úkol a následně sledovat stav extrakce;
(4) publikovat knihovnu a aplikace jako open-source balíček na pypi.python.org.

Úkolem pro ročníkový projekt jsou body 1, 2, 3a, 3d. Navíc je zde úkol zveřejnit knihovnu a aplikaci na portálu GitHub i s uživatelskou dokumentací. Ostatní úkoly zůstávají do bakalářské práce.

Program má být používán pro lingvistický výzkum, s jeho pomocí by mělo být možné vytvořit korpus alespoň se stovkami tisíc konkordancí. Také by měl být přístupný vědeckým pracovníkům na celém světě, takže by se měl dát snadno nainstalovat a snadno používat.

Vypracování

Hledání vhodných internetových zdrojů pro získávání konkordancí

Vhodným zdrojem pro získávání konkordancí se ukázaly být internetové vyhledávače, které uživatelům dávají odkazy na stránky, na kterých se vyskytuje hledaný termín. Těmito vyhledávači jsou například Google.com, Bing.com nebo Seznam.cz. Bylo by velice jednoduché, kdyby stačilo, aby program zadal dotaz Googlu, přečetl si „snippet“ (úryvek textu z nalezené stránky), a v něm našel konkordanci. Google by udělal velkou část práce za program včetně morfologické analýzy, vydal by stránky se všemi tvary zadaného slova.
Tento přístup ale není možný z několika důvodů:

1) Google.com a další vyhledávače blokují automatický přístup. Pokud získají podezření, že vyhledávání provádí automatický program místo člověka, zobrazí stránku s chybovou hláškou

a CAPTCHA. Vyřešit automaticky CAPTCHA je poměrně složité a je nad rámec tohoto projektu.

Míra blokování a způsob detekce se u různých vyhledávačů mírně liší.

2) Všechny vyhledávače na první stránce výsledků vyhledávání (tzv. SERP, Search Engine Result Page) poskytují informaci, že nalezly přibližně stovky milionů výsledků. V dolní části stránky je odkaz, kterým se dá dostat na další stránku výsledků. Problémem je, že tímto způsobem se dá dostat poměrně malé množství odkazů, u Bing.com pouze prvních 1000, ostatní vyhledávače poskytnou pouze několik stovek stránek. To je příliš málo.

3) Ne ve všech snippetech se dají najít celé věty. Velikost snippetu je omezena, často bývá věta přerušena. Bylo by potřeba navštívit nalezený odkaz a hledat na stránce konkordance.

Bylo by dobré, kdyby nějaký vyhledávač spolupracoval a například poskytl neblokovaný nebo méně omezený přístup pro výzkumné účely. Kontaktoval jsem provozovatele vyhledávačů Seznam.cz a Bing.com s prosbou o spolupráci, ale nedostal jsem žádnou odpověď.

Bing.com poskytuje placené API, s pomocí něhož mohou automatické programy přistupovat k vyhledávači. Tato služba ale poskytuje zdarma pouze 5000 dotazů za měsíc, což je málo. Ani placená verze neposkytuje moc dotazů a je poměrně drahá. Navíc není snadné si zažádat o přístup k bezplatné verzi, a to by velmi zkomplikovalo používání programu.

Dostupné knihovny pro prohledávání webu

Našel jsem, vyzkoušel a otestoval nejméně 6 knihoven v jazyce Python pro získávání odkazů z Google.com. Většina z nich pracovala tak, že vytvořila url stránky výsledků vyhledávače a jednoduše stáhla tu stránku jako html dokument. Poté ho naparsovala a vypsala odkazy.

Některé nástroje se snažily různými jednoduchými způsoby zařídit, aby Google nedetekoval jejich automatický přístup. To se ale většinou příliš nedařilo.

Našel jsem také knihovnu GoogleScraper. Ta slibuje, že dokáže stáhnout 200 výsledků za vteřinu třemi různými módy včetně použití proxy serverů a módu selenium, ve kterém velmi věrně simuluje chování webového prohlížeče, takže ho Google neodhalí. Dokáže stahovat odkazy ze šesti vyhledávačů včetně Bing.com a Google.com. Bohužel po vyzkoušení jsem zjistil, že většina funkcí, které podle popisu má mít, nefunguje. Jediné, na co se dá GoogleScraper spolehlivě použít, je parsování SERP.

Dále jsem získal informaci, že Bing.com téměř vůbec neblokuje přístup pro prvních 200 výsledků každého vyhledávání. Zjistit naprosto přesně, kolik výsledků za časovou jednotku Bing.com povolí, do jaké míry to platí a na jak dlouho blokuje, není snadné, proto jsem od toho upustil. Je ale zřejmé, že blokuje mnohem méně než Google.

Proto jsem dále hledal knihovny, které by uměly stahovat odkazy z Bingu. Našel jsem ale pouze takové, které používaly jeho placené API.

Vybraný postup získávání konkordancí

Můj program stáhne z Bing.com SERP. Poté z ní získá odkazy s pomocí parseru z GoogleScraperu. Nakonec ty odkazy navštíví a najde v nich konkordance.

Jak zvýšit počet různých odkazů, které Bing.com může vydat? Představme si, že chceme stáhnout konkordance se slovem *find*. Pak nám vyhledávač najde jisté množiny odkazů, když necháme hledat postupně *find Alabama*, *find car*, *find cake* a *find kvol*. Dá se očekávat, že ty množiny budou různé a budou mít jen málo společného, protože Bing se bude snažit hledat stránky, na kterých se nachází obě slova, jedno to klíčové slovo, které nás zajímá, a to druhé jako návnada, nějaké nesmyslné nebo náhodné slovo. V programu ho nazývám *bazword*.

Bazword ale nesmí být moc dlouhé, nesmí se skládat z více různých slov, protože potom se vyhledávač bude snažit najít stránky, na kterých je co nejvíce zadaných klíčových slov, i když třeba ne všechna. A přitom může pominout to naše cílové slovo.

Navíc snippet potom bude velmi pravděpodobně složen z více vět, aby se v něm objevila všechna klíčová slova, a žádná věta tam pravděpodobně nebude celá. Takže bude potřeba ještě navštívit stránku a snippet nijak nevyužívat.

Tímto způsobem tedy zvýšíme počet odkazů, které můžeme získat.

Zbývá jen určit, jaký způsob výběru bazwords bude nejlepší. Pro začátek jsem vymyslel a implementoval čtyři způsoby, nechávám uživatele, ať si sám vybere. Do dalších verzí programu je porovnám a navrhnu nejlepší.

Implementované způsoby výběru jsou tyto: náhodná čtyřpísmenná slova, čísla 0, 1, 2, 3, ..., slova z náhodného článku z anglické Wikipedie a slova z titulku náhodného článku z anglické Wikipedie.

Za výstupní formát funkcí na stahování odkazů i konkordancí jsem zvolil obyčejný slovník, který obsahuje položky pro url, klíčové slovo, datum stáhnutí apod. Alternativou by mohla být zvláštní datová třída, ale toto jsem nezvolil, aby byl můj program co nejjednodušší a dobře se s ním pracovalo i dalším programátorům.

Tato struktura se také velice jednoduše konvertuje do jsonu a XML. To jsou výstupní formáty programu, uživatel si může vybrat. XML jsem zvolil proto, že se používá v NLP, json hlavně proto, že je přehlednější, když ho prohlížím v obyčejném textovém editoru nebo když si ho nechávám vypisovat na terminál, dobře se s ním tedy vytváří a ladí program.

Implementace

Celý repozitář se zdrojovým kódem je na GitHubu zde:
`https://github.com/Gldkslfmsd/concordance-crawler`

Zdrojový kód programu sídlí ve složce ConcordanceCrawler. V repozitáři v souboru README.md je popis instalace a používání programu.

Pro jednoduchost jsem použil Python 3.4.0, protože jsem na něj zvyklý, umím v něm programovat a také použitý GoogleScraper funguje jen pro Python 3. Do dalších verzí ale nebude problém přidat podporu i pro Python 2.

Program jsem se snažil psát tak, aby nepotřeboval žádné další vysvětlení, všechny funkce a třídy, které to podle mého názoru potřebují, mají docstringy. Sem zbývá dodat už jen pár komentářů:

- *app.py* je hlavní soubor konzolové aplikace, nachází se zde její hlavní funkce.
- *links.py, visitor.py, bazword.py, parse.py* a *urlrequest.py* jsou soubory, které dělají tu hlavní práci ConcordanceCrawleru. Starají se o stažení odkazů a jejich navštívení. Jiný programátor je může použít ve svém vlastním projektu. Ostatní soubory programu jsou pouze kvůli konzolové aplikaci.
- *urlrequest.py* – zde se nachází funkce, která stáhne dokument nacházející se na url a vrátí ho. Na to mohl stačit příkaz *requests.get(url).text,* ale bylo zde několik problémů:

  - Chci tuto funkci volat na několika místech v různých souborech vždy s některými parametry stejnými. Pak se samozřejmě vyplatí mít na to funkci, která ji zavolá, a ta je umístěna ve zvláštním souboru a importována.

  - Takto samotný příkaz nemá žádný defaultní timeout, tedy nevyhodí žádnou výjimku, když server dlouho neodpovídá, a čeká klidně až donekonečna. Dá se ale nastavit.

  - Během testování jsem objevil zvláštní chybu, kvůli které program zamrzl a čekal donekonečna na nějakém pasivním čekání právě v příkazu *requests.get,* a to i když byl nastaven timeout. Nepřišel jsem na příčinu a důvod chyby, chování se různí na různých počítačích, někdy nenastane vůbec, někdy zhruba po 20 minutách bezproblémového chodu a poté nastává stále dokola. Obvykle na odkazu SERP. Když na stejném počítači v novém promptu zkusím stejnou funkci na stejném odkazu, proběhne ihned a bez problémů, tudíž to nebude tím, že by vyhledávač blokoval můj počítač.

    Problém jsem nevyřešil, ale obešel. Funkci volám ve zvláštním procesu, v hlavním procesu program čeká maximálně 5 minut na výsledek. Pokud se nedočká, přeruší vedlejší proces a vyhodí chybu. Používám zde multiprocessing.Pool, díky němu program vytvoří pouze jeden vedlejší proces pro všechna volání funkce, to šetří paměť i čas. To má za následek, že občas, když je program předčasně zabíjen stiskem Ctrl+C, vypíše stav zásobníku s výjimkou, která vznikla v nějakém vedlejším procesu. A to i

přesto, že jsem se snažil KeyboardInterrupt v tomto procesu odchytit a umlčet. Do dalších verzí se pokusím o chybě zjistit více a opravit ji.

  - Servery mohou měnit obsah dokumentu podle toho, zda k nim přistupuje automatický crawler nebo člověk skrze webový prohlížeč. To pozná podle položky *User-Agent* v hlavičce http požadavku. Například pro lidského uživatele vydá ten opravdový dokument, jak má, ale pro automat jen krátkou zprávu, že automatický provoz blokuje. Proto měním tu hlavičku z defaultní hodnoty na hodnotu, kterou mívá jeden z běžně používaných webových prohlížečů. Používám na to funkci z GoogleScraperu, tam je tato věc vyřešena.
- *visitor.py* – zde se nachází funkce, která navštíví stránku, vezme její viditelný text bez tagů a ten rozdělí na věty. Větná segmentace se prozatím provádí pouze jednoduše, za oddělovače věty se považuje tečka, otazník, vykřičník a konec řádku. Do dalších verzí je třeba ji zlepšit, například špatně rozezná větu s řadovou číslovkou a tečkou, nerozezná ... apod. Dále se zde filtrují věty s klíčovým slovem, prozatím musí stát samostatně ve větě a musí být přesně v tom tvaru, jak ho zadal uživatel.
- *links.py* – tady je mimo jiné funkce, která kontroluje, zda nás vyhledávač neblokuje. Pozná to podle řetězce „Omluvte přerušení“, který se nachází na blokovaných stránkách, to ale bude fungovat jen v České republice. Po blokování nic neudělá, resp. vrací None místo odkazů. Do dalších verzí je třeba zvážit, zda je tato funkce potřeba a jak má vypadat, aby fungovala po celém světě.
- *bazwords.py* – zde v generátorech bazword, které stahují náhodné články z Wikipedie, se nekontroluje, že stahování dopadlo v pořádku. Pokud ne, vyhodí se výjimka, ta se odchytí až dále a připočítá se chyba k počtu chyb za stahování SERP.

Úkoly do dalších verzí

Zde shrnu všechny úkoly, které je nutné nebo vhodné zapracovat do dalších verzí programu.

- Zlepšit větnou segmentaci a filtrování konkordancí se všemi tvary klíčového slova. Na to se vytvoří různé samostatné třídy, podle přání uživatele je program bude volat.
- Porovnat způsoby generování bazwords a navrhnout nejlepší.
- „Vyznačovat“ cílové slovo ve výstupním souboru.
- Zrychlit stahování, současná verze je jednovláknová, dá se ale stahovat paralelně.
- Přidat podporu i pro Python2. Problém je snad jen v jiném pojmenování importovaných knihoven, to se dá snadno vyřešit modulem six. To je potřeba udělat i pro GoogleScraper, ale i to by neměl být problém.
- GoogleScraper používá velmi mnoho velkých knihoven kvůli funkcionalitám, které můj program nevyužívá. Jedná se o knihovny pro práci s databází. Uživatel ConcordanceCrawleru je ale musí nainstalovat i tak. To se dá změnit.
- Dále zbývá vypracovat ostatní úkoly ze specifikace, tedy přidat určování jazyka stahovaného dokumentu a kódování stránky, vyrobit webové rozhraní s demo aplikací a publikovat knihovnu a aplikace na pypi.python.org.