WEDT - DOKUMENTACJA KOŃCOWA

Piotr Doniec

Styczeń 2012

1 Cel projektu, zadanie

Dany jest zbiór dokumentów tekstowych (czysty tekst oraz HTML). Należy dokonać podziału zbioru na grupy, tak aby dokumenty należące do pojedynczej grupy były jak najbardziej zbliżone tematycznie do siebie, a jednocześnie odmienne od dokumentów w pozostałych grupach. Grupy mogą (ale nie muszą) - tworzyć strukturę hierarchiczną.

2 Metoda rozwiazania

1. Wczytanie dokumentu do analizy
2. Utworzenie w pamięci 2 kopii dokumentu: bez i z znacznikami HTML. Funkcja usuwająca HTML jest zaimplementowana w NLTK.
3. Usunięcie wszystkich znaków specjalnych, interpunkcji z przetwarzanego dokumentu przy pomocy prostego wyrażenia regularnego.
4. Usunięcie spójników i innych słów nie mających wpływu na treść dokumentu (ang. stopwords). Biblioteka NLTK zawiera listę takich wyrazów dla kilku języków. Dla języka angielskiego twórcy biblioteki przewidzieli 127 wyrazów.
5. Stemming tekstu, zliczenie wystepujących wyrażeń (terms). Wykorzystany jest stemmer Portera. Do prowadzenia statystyk wyrazów wykorzystany jest typ FreqDist umożliwiający łatwe uaktualnianie statystyki a także dostęp do liczby i częstotliwości wystąpienia wyrażenia bez konieczności przeprowadzania dodatkowych obliczeń.
6. Wykorzystanie kopii zawiarającej znaczniki HTML do lepszej oceny treści. Z treści zadania wynika, że dokumenty mogą zawierać znaczniki HTML. Można wykorzystać tę własność w celu poznanania potencjalnie istotnych słów dla dokumentu. Znaczniki HTML można w łatwy sposób poszeregować względem ważności. Im ważniejszy znacznik, tym istotniejsze są słowa które zawiera. W chwili obecnej program uwzględnia tylko znacznik <TITLE></TITLE> i <H1></H1>. Wartościowanie

znaczników odbywa się poprzez zwiększenie liczby wystąpień słowa o pewną stałą zadaną jako parametr.

7. Obliczenie dla każdego dokumentu współczynnika tf-idf zgodnie ze wzorem:
   $(tf - idf)_{i,j} = tf_{i,j} * \log(\frac{|D|}{|d:t_i \in d|})$
   gdzie,
   $tf_{i,j}$ - częstotliwość występowania wyrażenia 'i' w dokumencie 'j'
   $|D|$ - liczba przetwarzanych dokumentów
   $|d : t_i \in d|$ - liczba dokumentów w których występuje wyrażenie 'i'

8. Grupowanie dokumentów na podstawie współczynnika tf-idf. Aktualnie wykorzystywany jest algorytm KMeans oraz odległość Euklidesowa. Próby wykorzystania podobieństwa cosinusowego zakończyły się błędami - wysoce prawdopodobne, że to błąd w bibliotece.

3 Algorytm KMeans

Jest to jeden z najprostszych algorytmów uczących się bez wykorzystania nauczyciela (ang. unsupervised) rozwiązującym problem grupowania. Dane, punkty, a w przypadku tego projektu, dokumnety są przypisywane do grup, których liczba k musi być z góry określona. Spośród dostępnych punktów, dokumnetów należy wybrać k które będą centrami, rdzeniami grup. Można to zrobić w sposób losowy, ale trzeba pamiętać, że wybór punktów centralnych istotnie wpływa na wynik grupowania. Dobrze jest użyć punktów możliwie od siebie oddalonych. W kolejnym kroku, każdy dokument zostaje przypisany do grupy której rdzeń znajduje się najbliżej. Po wykonaniu tej procedury dla wszystkich dokumentów pierwszy etap działania algorytmu jest zakończony. Następuje obliczenie nowych rdzeni dla wszystkich grup poprzez odnalezienie punktu leżącego możliwie blisko środka grupy. W tym momencie ponownie trzeba przypisać dokumenty do grupy której rdzeń jest najbliżej - pętla. Algorytm kończy działanie kiedy nie obserwuje się już zmian punktów rdzeniowych.

4 Narzędzia

Projekt został zaimplementowany w języku Python. Wynika to z łatwości języka i ilością dostępnych bibliotek zapewniających stopień abstrakcji umożliwiający skupienie się na rozwiązaniu problemu bez konieczności implementacji algorytmów składowych.

Głównym ogniwem jest przedstawiona na wykładzie biblioteka nltk zapewniająca większość wymaganej funkcjonalności wykorzystanej w projeckie. Z poziomu nltk możliwa jest między innymi tokenizacja tekstu, dostęp do korpusów oraz stemming wyrazów. Zaimplementowane są także najpopularniejsze funkcje obliczające podobieństwo między dokumentami oraz algorytmy grupowania. Dodatkowo wykoszystano bibliotekę lxml która umożliwia w łatwe i szybkie parsowanie dokumentów HTML.

Kod projekt jest objęty systemem kontroli wersji i dostępny pod adresem: http://github.com/pejotr/doc-clustering

5 Uruchomienie

Kod projektu można pobrać z serwisu Github zarówno jako skompresowane archiwum ZIP, lub przy pomocy GIT CVS. Drugie rozwiązanie jest o tyle lepsze, że umożliwia szybkie uaktualnienie posiadanej wersji kodu do najnowszej dostępnej. Uruchomienie wymaga zainstalowanego interpretera python a także bibliotek *nltk*, *lxml* oraz *numpy*. Wszystkie można zainstalować z repozytorium apt w przypadku Ubuntu, lub pacman w pzypadku ArchLinux. W przypadku Ubuntu, konieczne jest też doinstalowanie pakietu *python-argparse*.

Dostępne parametry modyfikujące działanie algorytmu wyświetlane są po wykonaiu polecenia *python main.py --help* i są to:

`-h -help` Wyświetla wszystkie opcje programu wraz z krótkim opisem

`--usehtml` Używanie tagów HTML do lepszej oceny treści dokumentu, *domyślnie False*

`--cosine` Użycie podobieństwa cosinusowego, zamiast domyślnego euklidesowego

`--title` Waga słów zawartych pomiędzy znacznikami <title></title>, *domyślnie 1*

`--h1` Waga słów zawartych pomiędzy znacznikami <h1></h1>, *domyślnie 1*

`--freq` Liczba najpopularniejszych wyrazów, wykorzystanych w procesie grupowania

`--groups` Maksymalna liczba grup

`--repeats` Liczba powtórzeń w algorytmie KMeans

W folderze data/bag-of-words dostepne są 2 pliki z dokumentami przeznaczonymi do grupowania, pobrane ze strony http://archive.ics.uci.edu/ml/datasets/Bag+of+Words. Cały zbiór zawiera 1500 dokumentów. Na potrzeby projektu, skróciłem go do 108. Format danych zawarty jest na wspomnianej wyżej stronie. Do współpracy z projektem, konieczne jest przetworzenie obu plików w postać dokumentów. W tym celu powstał prosty skrypt *bag2files.py* który w katalogu *files*(trzeba utworzyć ręcznie) umieści wygenerowane dokumenty. Niestety, są to czysto tekstowe dokumenty zatem wartościowanie HMTL nie ma sensu.

W folderze data/dev-data umieściłem dokumenty pobrane ze strony przedmiotu, ale pogrupowane samodzielnie. Numery dokumentów są zachowane, zatem kawa32.txt odpowiada plikowi case32.txt.

6 Testy

Weryfikacja poprawności działania algorytmu została przeprowadzona poprzez porównanie wyniku działania programu z recznym pogrupowaniem dokumentów dokonanym przeze mnie. Ze względu na liczbę do testów posłuzyły dokumenty dostępne na stronie przedmiotu. Zauważono, że domyślne wartości i wykorzystanie najprostszej funkcji podobieństwa (podobieństwa Euklidesowego) nie daje satysfakcjonujących wyników. Najlepszy rezultat uzyskano przy wykorzystananiu podobieństwa cosinusowego, wartościowowaniu znacznika <title></title> z wagą 7 przy 50 powtórzeniach. Odpowiada to uruchomieniu programu z następującymi parametrami: *python2 src/main.py data/dev-data/ --cosine --usehtml --groups=5 --title=7*. Wynik jest przedstawiony w Table 1.

Dokumenty	Grupa
42-48, 26, 30, 32, 35	1
5-13	2
21-25, 27-29, 32-34, 36-41	0
1-4, 14-16	4
17-20	3

Tablica 1: *python main.py dev-data/ --cosine --usehtml --groups=5 --title=7*

Zwiększenie ilości grup do np. 7, sprawia że algorytm rozdzielił informajce dotyczące kawy do wiekszej liczby grup. Pozostałe dokumenty nadal pozostają w tym samym klastrze (Tabela 2).

Dokumenty	Grupa
42-48, 26, 35, 40	3
5-13	4
21-25, 28-29, 31, 33, 36-38	0
27	2
30, 32, 34, 39, 41	1
1-4, 14-16	5
17-20	6

Tablica 2: *python main.py dev-data/ --cosine--usehtml --groups=7 --title=7*

Uruchomienie programu w domyślnej konfigracji tj. bez wartościowania tagów HTML, z podobieństwem Euclidesowym pokazuje jak istotne jest wykorzystanie dodatakowych informacji o strukturze dokumentu (Tablica 3). Widać, że większość dokumentów dotycząca kawy została zakwalifikowana razem z dokumentami na temat finansów.

Ponieważ punkty rdzeniowe grup wybierane są losowo, wyniki przeprowadzone na innym komputerze w innym czasie mogą się nieznacznie różnić od zaprezentowanych.

Dokumenty	Grupa
5-13	4
17-20, 22-47	3
48	1
27	2
1, 2, 14-16	5
4	6
21	0

Tablica 3: *python main.py dev-data/ --groups=7*

7 Wnioski

Mimo dość prostego algorytmu, grupowanie odbywa się lepiej niż początkowo zakładałem. W szczególności, do dobrej grupy trafiają wszystkie dokumenty zawierające internetowe strony domowe. W czasie wielu prób które przeprowadziłem w celu odnalezienie parametrów które zbliżyłyby wynik działania programu do ręcznego grupowania, zauważyłem że często do nie odpowiedniej klasy trafia dokument case32.txt. Mimo że treść tego pliku, dla człowieka, zawiera informacje dotyczące kawy, to dla zaprezentowanego algorytmu nie jest to takie oczywiste i pasuje bardziej do informacji finansowych lub kodu Pascala. Jest to zapewne spowodowane tym, że case32.txt zawiera informacje statystyczne, pogodowe. Algorytm nie jest na tyle inteligentny, żeby zrozumieć choć częściowo sens dokumentu i stąd błędne przypisanie.

Popularna biblioteka NLTK do analizy języka naturalnego dla Pythona wydaje się uszkodzona w przypadku algorymtu grupowania KMeans. Czasem zdarza się, że pojawia sie błąd związany z centrum pustego klastra. Niestety, jest to asercja więc natychmiast przerywa działanie programu. Sugerowane rozwiązanie również nie zdaje egzaminu. Niestety, im więcej dokumentów poddanych analizie tym prawdopodobieństwo wystąpienia błędu jest większe. Poza tym NLTK świetnie się nadaje do tworzenia programów do analizy języka naturalnego. W połączeniu z językiem Python, daje potężne narzędzie w którym szybko i łatwo można testować różne metody i algorytmy w celu rozwiązania problemu, bez niepotrzebnego wnikania w szczegóły implementacyjne.

```
    Error: no centroid defined for empty cluster.
Try setting argument avoid_empty_clusters to True
(...)
AssertionError
```

Testy wykazały, że wartościowanie tagów HTML bardzo pozytywnie wpływają na wynik grupowania. Bez zastosowania wartościowania błędy popełnianie przez algorytm są bardzo duże. Wykorzystanie tagów H1 i TITLE to najprostsza metoda wykorzystania struktury dokumentu jako dodatkowego źródła informacji o dokumencie. Prawdopodobnie w przypadku większej liczby dokumentów,

analiza większej liczby tagów mogła by dodatkowo poprawić wynik.

Gdyby nie problemy z biblioteką, projekt w mojej ocenie można by uznać za w pełni udany. Projekt wykorzystauje zarówno treść dokumentu a także w przypadku HMTL strukturę.

8 Poprawność grupowania

Znanych jest kilka metod sprawdzenia skuteczności, poprawności grupowania. Jednym z nich jest entropia. Entropia grupy C_i o rozmiarze n_i zdefiniowana jest następująco: $E(C_i) = -\frac{1}{\log(c)} \sum_{h=1}^{k} \frac{n_i^h}{n_i} \log(\frac{n_i^h}{n_i})$, gdzie c jest ilością kategorii w zbiorze danych, a n_i^h jest liczbą dokumentów z klasy h przypisanych do grupy i. Inny, prostszy algorytmy *purity*, również wymagaja sklasyfikowanych danych. W związku z tym niemożliwe jest ocenienie jakości grupowania bez uprzedniej klasyfikacji. Można to zrobić tylko ręcznie, o ile zbiór danych nie jest zbyt duży. Mimo że w internecie, zdecydowanie bardziej powszechne są dane do klasyfikowania, można też znaleźć dane dedykowane do grupowania. Niestety trudno z nich skorzystać, gdyż nie udostępnione są jakiekolwiek dane na temat poprawnego grupowania. Sporo jest również danych w formie przygotowanej do obróbki w narzędziac bazodanowych. Ze znalenioznych zbiorów dość ciekawym jest Bag-of-words, nie są to typowe dokumenty, a raczej jak wskazuje nazwa lista słów wraz z iloscią wystąpień i przynależnością do dokumentu. Oczywiście można przeprowadzić grupowanie takiego zbioru, jednak weryfikacja jest dość kłopotliwa.

Literatura

[1] A. Huang, *Similarity Measures for Text Document Clustering*, Department of Computer Science, The University of Waikato, Hamilton, New Zealand

[2] M. Steinbach, G. Karypis, V. Kumar, *A Comparison of Document Clustering Techniques*, Department of Computer Science and Egineering, University of Minnesota

[3] C. Manning, H. Schütze, *Foundations of Statistical Natural Language Processing*,MIT Press, 1999