Reprezentacja wiedzy
Charakterystyki bohaterów literackich

Paweł Obrok

Skompilowano 3 lipca 2010

1 Problem

Dokument opisuje zaimplementowane przeze mnie narzędzie do automatycznej ekstrakcji charakterystyk bohaterów na podstawie tekstu literackiego. Charakterystyka bierze pod uwagę powiązania danego bohatera z innymi postaciami w tekście i wyznacza dla niego współczynniki pozytywności (liczba od -1 do 1, gdzie -1 oznacza skrajnie negatywnego, a 1 skrajnie pozytywnego bohatera) oraz główności (liczba od 0 do 1 przybliżająca procentowe znaczenie bohatera dla tekstu).

2 Szkic rozwiązania

Proponowane przeze mnie rozwiązanie składa się z dwóch części: modułu rozpoznającego bohaterów oraz modułu dokonującego ich oceny.

2.1 Moduł rozpoznawania bohaterów

Wyszukiwanie bohaterów w tekście odbywa się w oparciu o ukryty model Markova (Hidden Markov Model - HMM). Wybrałem ten sposób, gdyż chciałem uniknąć konieczności czasochłonnego uczenia systemu z nadzorem, a metoda HMM pozwala osiągnąć dość dobre rezultaty wykorzystujące uczenie bez lub z minimalnym nadzorem.

Słowa tekstu są po pewnych przekształceniach traktowane jako widoczne tokeny HMM. HMM ma tylko dwa ukryte stany 'hero' i 'default'. Podając przykładowych bohaterów można z tekstu o nich wygenerować wiele próbnych danych i wyliczyć dla nich model HMM.

2.2 Moduł oceny bohaterów

Ocena bohaterów polega na znalezieniu słów, które są najczęściej stowarzyszone z danym bohaterem i następnie zsumowaniu ocen tych słów z wagami zależnymi od częstości i bliskości danego słowa do bohatera. Podobny proces pozwala ocenić pozytywność/negatywność powiązania między bohaterami, jednak tym razem odległość jest liczona jako suma odległości od dwóch bohaterów.

Przy wyborze tego rozwiązania kierowałem się również chęcią ograniczenia czasu potrzebnego na otagowanie tekstu. Do testów otagowałem około 1000 najczęściej występujących słów, co pozwoliło uzyskać sensowne wyniki. Nie wydaje

się to dużo, zwłaszcza, że tagowanie pojedynczych słów jest o wiele prostsze niż np. dokonywanie rozbioru zdania.

3 Implementacja

Jako platformę implementacyjną wybrałem język ruby i interpreter jruby. Ten pierwszy ze względu na możliwość bardzo szybkiego prototypowania, natomiast ten drugi ze względu na lepszą wydajność w porównaniu z MRI i dostępność wysokiej jakości bibliotek pisanych dla javy.

3.1 Hidden Markov Model

W projekcie korzystam z javowej implementacji HMM do dekodowania i obliczania prawdopodobieństwa sekwencji stanów. Więcej informacji pod adresem `http://www.run.montefiore.ulg.ac.be/~francois/software/jahmm/`.

3.1.1 Obliczanie parametrów modelu

System rozpoczyna obliczenie z dwoma przykładowymi stringami. Jeden z nich jest przykładem słowa, które jest bohaterem, a drugi przykładem słowa, które nie jest bohaterem, ale mogłoby być z nim łatwo pomylone (np. "Geralt"i "Brokilon"). Poniższy pseudokod pozwala obliczyć prawdopodobieństwa stanów początkowych, prawdopodobieństwa przejść między stanami oraz prawdopodobieństwa emisji symboli. Symbolami HMM są nie same słowa, ale pewne tokeny uzyskiwane ze słów przez zastąpienie każdego ciągu wielkich liter znakiem „X", każdego ciągu małych liter znakiem „x" oraz każdego ciągu innych znaków znakiem „.". W pseudokodzie przyjmujemy, że stan „0" oznacza zwykłe słowo, a „1" — bohatera.

```
initial = [0, 0]
transitions = [[0,0],[0,0]]
emissions = [[0]*possible_symbols.size]*2

for paragraph in text.paragraphs
  obs_seq = paragraph.split.map(x => x.to_symbol)
  state_seq = obs_seq.map(x => x == example_hero ? 1 : 0)

  initial[state_sequence.first] += 1
  for i in 0..(state_seq.size - 1)
     if i + 1 < state_seq.size
      transitions[state_seq[i]][state_seq[i+1]] += 1
    end
    emissions[state_seq[i]][obs_seq[i]] += 1
  end
end
```

Po wykonaniu tego kodu należy oczywiście znormalizować jeszcze wszystkie uzyskane tablice tak, aby tablica initial sumowała się do 1 oraz każdy wiersz w tablicah transitions i emissions sumował się do 1.

3.1.2 Dekodowanie

Dekodowanie to obliczenie najbardziej prawdopodobnej sekwencji stanów dla danej sekwencji symboli i danego modelu. W Jahmm jest ono zaimplementowane za pomocą algorytmu Viterbiego. Poniższy pseudokod prezentuje ten algorytm. Emissions, transitions, initial to tablice obliczone jak w poprzednim podrozdziale. Obs_seq to sekwencja słów przekształconych w tokeny dla danego akapitu. Oblicza on również prawdopodobieństwo najbardziej prawdopodobnej sekwencji stanów, z którego również korzystam.

```
V = [[0,0]]*obs_seq.size
path = {}

for y in 0..1
  V[0][y] = initial[y] * emissions[y][obs_seq[0]]
  path[y] = [y]
end

for t in 1..(obs_seq.size - 1)
  newpath = {}

  for y in states:
    temp = (0..(no_states-1)).
      map(x => V[t-1][x]*transitions[y][x]*emissions[x][obs_seq[t]])
    prob = temp.max
    state = temp.index_of_max
    V[t][y] = prob
    newpath[y] = path[state] + [y]
  end

  path = newpath
end

temp = (0..(no_states-1)).map(x => V[obs_seq.size - 1][x])
probability = temp.max
state_seq = path[temp.index_of_max]
```

Korzystając z algorytmu Viterbiego i nauczonego modelu dla każdego akapitu obliczam najbardziej prawdopodobną sekwencję stanów oraz jej prawdopodobieństwo. Następnie dla każdego słowa wyemitowanego w stanie „hero" obliczam sumę prawdopodobieństw sekwencji stanów, w których tak się dzieje po wszystkich akapitach tekstu. Za postaci uznaję te słowa, które przekroczą pewien próg tak zdefiniowanego współczynnika. Na podstawie tego współczynnika obliczana jest także „główność" bohatera - powstaje ona przez znormalizowanie współczynników tak, aby sumowały się do 1.

3.2 Ocena bohaterów

Aby ocenić bohaterów obliczam macierz relacji między nimi. Macierz relacji R to tablica $n \times n$, gdzie n to liczba wykrytych postaci. R_{ij} zawiera liczbę z przedziału $[-1, 1]$ będącą sumą odwrotności sum odległości słów od i-tego

i j-tego bohatera przemnożonych przez ich „pozytywność", gdzie pozytywność to pewna liczba z przedziału $[-1, 1]$. Poniższy pseudokod prezentuje sposób obliczania macierzy relacji przy założeniu, że każde słowo odnosi się jedynie do co najwyżej dwóch sąsiednich postaci. Założenie to pozwala zmniejszyć czas działania algorytmu z $O(n^2 * w)$ do $O(n * w)$, gdzie w to liczba słów w tekście.

W pseudokodzie pominąłem optymalizację utrzymującą dla każdego bohatera, które z wystąpień jego imienia jest w danym momencie najbliższe, gdyż wprowadza ona zbytnią komplikację.

Names to tablica zawierająca rozpoznane imiona bohaterów.

```
relations = [[0]*names.size]*names.size
word_nos = relations.clone
words = text.split
name_locs = []

for i in 0..(names.size - 1)
  name_locs[index] = text.select_indices(x => x == names[i]
end

for i in 0..(text.size -1)
  word = text[i]
  dists = name_locs.map(x => x.map(y => (i - y).abs).min)

  indicec = [dists.index_of_min, dists.index_of_second_min]

  for j in indices
    dist1 = dists[j]
    for k in indices
      dist2 = dists[k]
      if (dist1 + dist2) != 0
        relations[i][j] += word.goodness*1.0/(dist1+dist2)
        word_nos[i][j] += 1
      end
    end
  end
end

for i in 0..(names.size - 1)
  for j in 0..(names.size - 1)
    relations[i][j] /= word_nos[i][j]
  end
end
```

Po wykonaniu tego pseudokodu diagonala macierzy R zawiera pozytywność każdego bohatera z osobna, natomiast elementy poza diagonalą odpowiadają pozytywności powiązań bohatera. Wyniki te są dodatkowo normalizowane: dodatnie elementy diagonali są liniowo przekształcane tak, aby maksymalny element przekształcony został w 1, podobnie dla ujemnych. Pozadiagonalne elementy każdego wiersza są podobnie „rozciągane".

Niejako „przy okazji" w trakcie wykonania tego algorytmu obliczany jest dla każdego z bohaterów ranking słów występujących blisko niego. Słowa nacecho-

wane emocjonalnie, które występują najczęściej, są następnie prezentowane w charakterystyce jako słowa kluczowe dla danego bohatera.

Ostatecznie pozytywność bohatera to jego pozytywność plus średnia ważona pozytywności pozostałych bohaterów, gdzie wagami są odpowiednie wpisy z macierzy relacji.

4 Testy

We wszystkich testach wykorzystałem model HMM nauczony na podstawie książki wszystkich opowiadań Sapkowskiego dostępnych na wierzbie. Przykładowe dane były wygenerowane automatycznie z tych opowiadań na podstawie następujących wiadomości: „Geralt" jest bohaterem, natomiast „Brokilon" nie jest bohaterem. Do oceny bohaterów dostępny był plik zawierający ocenione około 1000 najpopularniejszych rzeczowników, czasowników, przymiotników i przysłówków pojawiających się w tych opowiadaniach. Oceny dokonałem ręcznie, brałem częściowo pod uwagę możliwe konteksty występowania tych słów w tej konkretnej książce.

Wykonałem testowe uruchomienia programu na wymienionych wyżej opowiadaniach oraz na książkach „W pustyni i w puszczy" i „Ogniem i mieczem" Sienkiewicza. Surowe wyniki znajdują się w plikach tekstowych odpowiednio *doc/wiedzmin.txt*, *doc/w_pustyni.txt*, *doc/ogniem.txt*.

4.1 Format wyników

Format wyników jest raczej samoobjaśniający. Na uwagę zasługuje pole pozytywność: liczba w nawiasie jest pozytywnością przed modyfikacją ze względu na powiązania z innymi bohaterami.

4.2 Uwagi do wyników

Program zdecydowanie najlepiej poradził sobie z tekstem, z którego wzięte były dane treningowe. Dostarczył zadowalającą próbkę bohaterów, udało mu się znaleźć kilka sensownych relacji. Co więcej sposób brania pod uwagę relacji z innymi postaciami wydaje się być tutaj dobrze dobrany — przykładowo Nivellen jest traktowany jako bohater negatywny, jednak po głębszej analizie okazuje się, że większość jego negatywności „pochodzi" z kontaktu z Bruxą (bohaterem czysto negatywnym) i po dodaniu „punktów" za powiązania zostaje oceniony pozytywnie.

Dla tekstu „W pustyni i w puszczy" programowi udało się wyróżnić głównych bohaterów oraz odkryć, że Gebhr jest postacią złą do szpiku kości. Słowa kluczowe znalezione dla poszczególnych bohaterów były w miarę sensowne.

Tekst „Ogniem i Mieczem" wypadł zdecydowanie najgorzej. Ze względu na brak możliwości wykrywania odmian imion bohaterów program „podzielił" każdego z nich na wielu, z których każdy został scharakteryzowany osobno.

5 Wnioski i możliwości rozwoju

Przedstawione podejście jest bardzo proste i niewystarczające do automatycznego streszczania tekstów, jednak można je traktować jako punkt wyjścia

dla bardziej rozbudowanego mechanizmu. Należy zauważyć, że mimo stosunkowo niewielkiej ilości ręcznie wygenerowanych danych treningowych system zachowuje się sensownie. Następujące poprawki mogą pomóc w ominięciu problemów napotkanych w trakcie testów:

1. Przybliżone odmienianie imion w oparciu o podobne słowa lub wzorce odmiany imion
2. Trenowanie na danym tekście lub na tekście o zbliżonym stylu
3. Rozszerzenie i poprawienie listy słów i ich nacechowania emocjonalnego w kierunku jej większej ogólności

Najbardziej interesujące możliwości rozwoju programu, to:

1. Wykrywanie odnoszenia się do bohatera przez kilka różnych zwrotów
2. Wspieranie nazw bohaterów będących całymi frazami, na przykład „Borch Trzy Kawki"
3. Rozbudowanie mechanizmu wykrywającego odnoszenie się słów do bohatera. Np. czasowniki zwrotne odnoszą się tylko do jednego bohatera i nie powinny wpływać na powiązania między postaciami.