情報検索

文書集合から目的の文書を探す

アドホック検索

- クエリを投げて検索
- google とか
- 2 つのモデル

  正確に一致したもの
  確率的に絞る

- 関心対象というかなんと言うか

  結果を対話的に改善する手法 (relevance feedback; 適合性フィードバック)
  複数のデータベースの結果を統合する手法 (databse marging)
  破損データに対しても有効か (スキャンした文書とか)
  英語以外の文書ではどうか

情報検索

文書集合から目的の文書を探す

コーパスを用いて検索する分野

- text categorization

  文書に予めカテゴリを付与しておく

- routing, filtering

  text categorization の特殊ケース
  カテゴリが「関係あり」と「無関係」の２つ
  アドホック検索との違いは教師データの有無
  routing は文書をランキング
  filtering は普通に判定

データ構造

inverted index(転置インデックス)

- 単語別
- 文章に単語が出現する頻度
- 拡張として、出現する位置情報ももたせる

  句の探索が容易になる
  「car insurance rates」で「rates for car insurance」が検索できない
  句ごとに inverted index を作れば解決

- ストップリスト

  前置詞などのそれ自体に意味のない単語

stemming（語幹処理）

4.2.3 でやったらしい

laughing, laugh, laughs $\Rightarrow$ laugh-c

問題点

- 意味的に違う単語を混同してしまう

  gall, gallery $\Rightarrow$ gall-

- 人間が見てもさっぱり

評価方法

Evaluation	Ranking 1	Ranking 2	Ranking 3
	d1: ✓	d10: ×	d6: ×
	d2: ✓	d9: ×	d1: ✓
	d3: ✓	d8: ×	d2: ✓
	d4: ✓	d7: ×	d10: ×
	d5: ✓	d6: ×	d9: ×
	d6: ×	d1: ✓	d3: ✓
	d7: ×	d2: ✓	d5: ✓
	d8: ×	d3: ✓	d4: ✓
	d9: ×	d4: ✓	d7: ×
	d10: ×	d5: ✓	d8: ×
precision at 5	1.0	0.0	0.4
precision at 10	0.5	0.5	0.5
uninterpolated av. prec.	1.0	0.3544	0.5726
interpolated av. prec. (11-point)	1.0	0.5	0.6440

google だったら一番目 ranking1 が一番いい

精度は全部同じ

評価方法をなにか考える

評価方法

Evaluation	Ranking 1	Ranking 2	Ranking 3
	d1: ✓	d10: ×	d6: ×
	d2: ✓	d9: ×	d1: ✓
	d3: ✓	d8: ×	d2: ✓
	d4: ✓	d7: ×	d10: ×
	d5: ✓	d6: ×	d9: ×
	d6: ×	d1: ✓	d3: ✓
	d7: ×	d2: ✓	d5: ✓
	d8: ×	d3: ✓	d4: ✓
	d9: ×	d4: ✓	d7: ×
	d10: ×	d5: ✓	d8: ×
precision at 5	1.0	0.0	0.4
precision at 10	0.5	0.5	0.5
uninterpolated av. prec.	1.0	0.3544	0.5726
interpolated av. prec. (11-point)	1.0	0.5	0.6440

cutoff

- 先頭 N 個の精度

uninterpolated average precision

評価方法

Evaluation	Ranking 1	Ranking 2	Ranking 3
	d1: ✓	d10: ×	d6: ×
	d2: ✓	d9: ×	d1: ✓
	d3: ✓	d8: ×	d2: ✓
	d4: ✓	d7: ×	d10: ×
	d5: ✓	d6: ×	d9: ×
	d6: ×	d1: ✓	d3: ✓
	d7: ×	d2: ✓	d5: ✓
	d8: ×	d3: ✓	d4: ✓
	d9: ×	d4: ✓	d7: ×
	d10: ×	d5: ✓	d8: ×
precision at 5	1.0	0.0	0.4
precision at 10	0.5	0.5	0.5
uninterpolated av. prec.	1.0	0.3544	0.5726
interpolated av. prec. (11-point)	1.0	0.5	0.6440

uninterpolated averaeg precision

- 関連文書を見つけた時点での精度をそれぞれ計算
- その平均
- Ranking3 $\Rightarrow$ $average(\frac{1}{2}, \frac{2}{3}, \frac{3}{6}, \frac{4}{7}, \frac{5}{8}) = 0.5726$

評価方法

Evaluation	Ranking 1	Ranking 2	Ranking 3
	d1: ✓	d10: ×	d6: ×
	d2: ✓	d9: ×	d1: ✓
	d3: ✓	d8: ×	d2: ✓
	d4: ✓	d7: ×	d10: ×
	d5: ✓	d6: ×	d9: ×
	d6: ×	d1: ✓	d3: ✓
	d7: ×	d2: ✓	d5: ✓
	d8: ×	d3: ✓	d4: ✓
	d9: ×	d4: ✓	d7: ×
	d10: ×	d5: ✓	d8: ×
precision at 5	1.0	0.0	0.4
precision at 10	0.5	0.5	0.5
uninterpolated av. prec.	1.0	0.3544	0.5726
interpolated av. prec. (11-point)	1.0	0.5	0.6440

interpolated average precision

- recall が 10*N%の時点での精度を計算

評価方法

評価値

- precision と recall はトレードオフ
- average precision は両方を測れて便利
- F 値 (8.1) も便利

評価方法

- いくつかクエリを投げてみる
- T テストをしてみる（Section6.2.3）

Probability Ranking Principle

確率の降順で文書をランキングすること

多くの検索システムが採用している

文書が独立であることを仮定している

- 判例：重複した文書がある場合
- 片方だけ出すのが良い設計だが、これは PRP に反する

情報の variance は考慮しない

- 「余分な情報が含まれているのを許可するか」を考慮しない

Vector Space Model

おなじみ

文書とクエリが、単語のベクトルで表現される

ベクトルの各要素は対応する単語の重み

- シンプルなものは、単語の出現頻度など

クエリベクトルと類似した文書ベクトルを探す

ベクトル類似度

- コサイン類似度（Section8.5.1）がよく使用される

単語の重み

notation

- $tf_{i,j}$ ⇒ 文書 d_j 中の単語 w_i の出現回数
- df_i ⇒ 文書集合 D 中の w_i が出現する文書の数
- cf_i ⇒ 文書集合 D 中の w_i が出現する回数
- N ⇒ 文書の総数

tf による重み

- $\sqrt{tf}, \log(tf)$ の形で用いる
- n 回出現しても、n 倍重要ではない

tf.idf

- 多くの文書に出現する単語は重要でないことが多い

  例:to, try

- $$weight(i,j) \begin{cases} (1+\log(tf_{i,j}))\log\frac{N}{df_i} & if\ \ tf_{i,j} \geq 1 \\ 0 & if\ \ tf_{i,j} = 0 \end{cases}$$
- tf.idf は tf と df の逆数を利用した重みの総称

  上のは ltn と呼ばれる

Term Distribution Model

単語の分布に関するモデルを作ってみる

- tf.idf に変わる重み付けが発見できる
- IR の他の分野にも利用できる

$P_i(k)$ を見積もりたい

- ある文書に w_i が k 回出現する割合

以下のものを紹介

- ポアソン分布
- 2 ポアソンモデル
- K mixture

ポアソン分布

有名な離散確率分布

$$p(k;\lambda_i) = e^{-\lambda_i}\frac{\lambda_i^k}{k!}$$

パラメータ $\lambda_i = \frac{cf_i}{N}$ とすることが多い

$P_i(k) = p(k;\lambda_i)$ としてみる

- 単語の出現確率がポアソン分布に従う

以下の条件が成り立てば、この仮定は正しい

- 短い文章中にある単語が出現する確率は文章の長さに比例する
- 短い文章中に同じ単語が 2 回以上出る確率は一回出る確率と比べると無視して良い
- 重複しない区間の出現はそれぞれ独立

非内容語は OK。内容語はダメ

- 内容語は複数回出現するのが普通

2 ポアソンモデル

ポアソン分布を 2 つ使用

$$tp(k;\pi;\lambda_1,\lambda_2) = \pi e^{-\lambda_1}\frac{\lambda_1^k}{k!} + (1-\pi)e^{-\lambda_2}\frac{\lambda_2^k}{k!}$$

内容語の出現が 2 つに分類できることを勘案

- 偶然出現
- 文章の主題に関わる出現

欠点

- $P_i(2) < P_i(3)$ になってしまう
- 実際は $P_i(0) > P_i(1) > P_i(2) > P_i(3) > \ldots$
- 使うポアソン分布を増やせば解決

  計算量がやばい

K mixture

$$P_i(k) = (1-\alpha)\delta_{k,0} + \frac{\alpha}{\beta+1}\left(\frac{\beta}{\beta+1}\right)^k$$

where $\delta_{k,0} = 1$ iff $k = 0$ and $\delta_{k,0} = 0$ otherwise and α and β are parameters that can be fit using the observed mean λ and the observed inverse document frequency IDF as follows.

$$\begin{aligned}
\lambda &= \frac{\mathrm{cf}}{N} \\
\mathrm{IDF} &= \log_2 \frac{N}{\mathrm{df}} \\
\beta &= \lambda \times 2^{\mathrm{IDF}} - 1 = \frac{\mathrm{cf}-\mathrm{df}}{\mathrm{df}} \\
\alpha &= \frac{\lambda}{\beta}
\end{aligned}$$

k=0 では予測値と実測値が一致する

$$\frac{P_i(k)}{P_i(k-1)} = \frac{\beta}{\beta+1} = \frac{cf-df}{cf} = (\text{一定})$$

- これは内容語では成り立たない
- 非内容語よりも精度が悪い

Inverse document frequency

IDF 的なものをを単語分布的な考え方から導出する

Residual inverse document frequency

$$RIDF = IDF - \log_2(1 - p(0; \lambda_i)) = \log_2 \frac{N}{df} - log_2(1 - p(0; \lambda_i))$$

ポアソン分布による df と実際の df の差

ポアソン分布は非内容語しか従わない

- RIDF が大きければ内容語

Latent Semantic Indexing(LSI)

	user	interface	HCI	interaction
Query	1	1		
Document1	1	1	1	1
Document2			1	1

Query と Document2 は類似度 0

でも関連がありそう

LSI

- 似た単語を同じ次元にマップ
- 次元削減
- SVD を使用 (Singular Value Decomposition)

LSI

超ざっくりしたイメージ

	d1	d2	d3	d4	d5	d6
宇宙飛行士	1	0	1	0	0	0
惑星	0	1	0	0	0	0
月	1	1	0	0	0	0
自家用車	1	0	0	1	1	0
トラック	0	0	0	1	0	1

	d1	d2	d3	d4	d5	d6
宇宙	2	2	1	0	0	0
車	1	0	0	2	1	1

SVD

$A_{t\times d} = T_{t\times n} S_{n\times n} (D_{d\times n})^T,\ n = min(t, d)$

$\widetilde{A} = T_{t\times k} S_{k\times k} (D_{d\times k})^T,\ k < n$

$\left\| A - \widetilde{A} \right\|_2$ が最小になる

$T^T T = I, D^T D = I$ が成り立つ

S は対角行列

$B = S_{2\times 2} D_{2\times n}$ として $B^T B$ を計算すると各文書間の類似度が出る

- $A^T A = (TSD^T)^T TSD^T = DS^T T^T TSD^T = (DS)(DS)^T$
- $A^T A = (TS)(TS)^T$ であるから単語間の類似度も同様

クエリや新しい文書は T^T を掛ければ ok

- $A = TSD^T \Leftrightarrow T^T A = SD^T$

SVD

欠点

- 最小二乗法を使っている

  正規分布を仮定している
  $\widetilde{A}$ を作るときに負の数が出うる
  行列に出現回数でなく重みを用いれば解決できることもある。

- 計算コストが大きい

  pseudo-feedback 同時出現を利用した方法

Discourse Segmentation

文章をいくつかのパートに分割したい

TextTiling

- 語彙を見て主題が変わるところを探すイメージ

まず、テキストを固定サイズで区切る

- テキストでは 20words がよいとしている

区切り目を gap と呼ぶ

3 つのコンポーネントからなる

- cohesion scorer

  gap で前後の話題がつながっているか判定

- depth scorer

  gap の cohesion を周囲の gap の chesion と比較
  周りより小さければ、depth は大きくなる

- boundary selector

  分割箇所を決める

Discourse Segmentation

cohesion scorer

いくつか提案されている

- vector space scoring

  vector space model のような感じで相関係数を計算
  語彙が似ていれば、主題が同じというイメージ

- block comparison

  一番性能がいい
  vector space scoring で idf を使用しない

- vocaburary introduction

Discourse Segmentation

depth scorer

谷の高さの和

smoothing を行う場合も

- テキスト参照

文章が粉々にならないようにヒューリスティックが必要な場合もある

boundary selecter

平均 μ と分散 σ を計算

depth が $\mu - c\sigma$ 以上の gap で切る

c は適当