Recuperação de Informação

Câmeras Digitais

Bruno Cavalcanti
Guilherme Henrique
Pedro Henrique

Índice Invertido

...

Download das páginas

- Mil páginas positivamente classificadas foram baixadas ao todo
- No total 21250 páginas foram baixadas
- Havest ratio -> 0,0483

Pré-processamento automático

- Atributos mais frequentes escolhidos:
  - Nome
  - Preço
  - Modo de armazenamento
  - Sensibilidade
  - Velocidade do obturador

- Discretização dos atributos numéricos

Detalhes do índice invertido

- Atributos com vários nomes são divididos, gerando múltiplas chaves no índice invertido

Compressão

- Serialização sem compressão -> 198 kilobytes
- Serialização com compressão -> 170 kilobytes

```
name.Sony%1=37,1;
name.5DS%3=499,1;507,1;509,1;
Shutter Speed.Via%4=836,1;891,1;896,1;909,1;
Storage Mode.Eye-Fi%61=2,1;10,1;18,1;24,1;25,1;27,1;30,1;
```

```
name.Sony%1=37,1;
name.5DS%3=499,1;8,1;2,1;
Shutter Speed.Via%4=836,1;55,1;5,1;13,1;
Storage Mode.Eye-Fi%61=2,1;8,1;8,1;6,1;1,1;2,1;3,1;2,1;2,1;
```

Processamento de Consulta

...

Calcular os pesos dos documentos para cada termo

$$\vec{d_j} = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$$

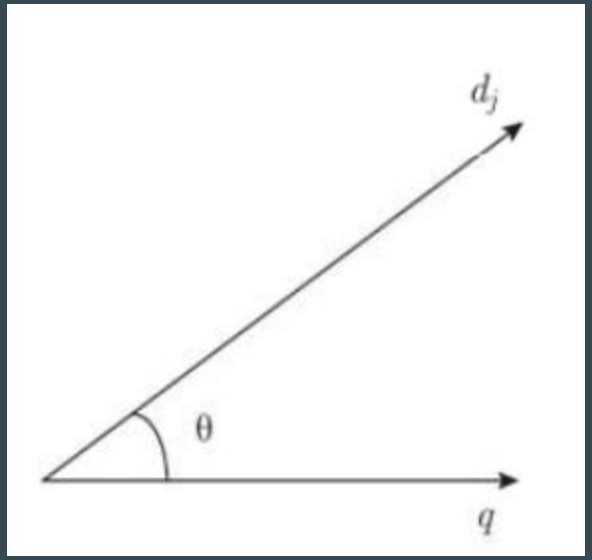
- Com TfIdf
- Sem TfIdf
- Tamanho dos vetores = 827

Calcular os pesos dos termos da consulta

$$\vec{q} = (w_{1,q}, w_{2,q}, \ldots, w_{t,q})$$

- Com TfIdf
- Sem TfIdf
- Tamanho do vetor = 827

Modelo vetorial



Modelo vetorial

$$sim(d_j, q) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

- sim(dj,q) = rank

Spearman Correlation

$$S(\mathcal{R}_1, \mathcal{R}_2) = 1 - \frac{6 \times \sum_{j=1}^{K} (s_{1,j} - s_{2,j})^2}{K \times (K^2 - 1)}$$

name.Sony x name.Compact

Spearman Correlation = 0.76

Spearman Correlation

$$S(\mathcal{R}_1, \mathcal{R}_2) = 1 - \frac{6 \times \sum_{j=1}^{K} (s_{1,j} - s_{2,j})^2}{K \times (K^2 - 1)}$$

shutter speed.Manual x shutter speed.Scene

Spearman Correlation = 0.81

Spearman Correlation

$$S(\mathcal{R}_1, \mathcal{R}_2) = 1 - \frac{6 \times \sum_{j=1}^{K} (s_{1,j} - s_{2,j})^2}{K \times (K^2 - 1)}$$

shutter speed.Manual x shutter speed.Scene

name.Sony x name.Compact

Spearman Correlation = 0.61

Spearman Correlation

$$S(\mathcal{R}_1, \mathcal{R}_2) = 1 - \frac{6 \times \sum_{j=1}^{K} (s_{1,j} - s_{2,j})^2}{K \times (K^2 - 1)}$$

shutter speed.Manual x shutter speed.Scene

name.Sony x name.Compact

sensitivity.Using x sensitivity.1EV

Spearman Correlation = 0.62

Spearman Correlation

$$S(\mathcal{R}_1, \mathcal{R}_2) = 1 - \frac{6 \times \sum_{j=1}^{K} (s_{1,j} - s_{2,j})^2}{K \times (K^2 - 1)}$$

shutter speed.Manual x shutter speed.Scene

name.Sony x name.Compact

sensitivity.Using x sensitivity.1EV

storage mode.Recording x storage mode.SD

Spearman Correlation = 0.47

Spearman Correlation (DA MANEIRA CORRETA)

Consulta	SpearmanCorrelation (TFIDF true vs TFIDF false)

`name.Sony`
`shutter speed.Scene`
`Sensitivity.using`
	0.9999266319266319

`Sensitivity.using` `Storage Mode.Recording`	0.9995928395928396

`Name.Compact`
`Name.Sony`
`Name.FujiFilm`
`Name.Nikon`
	0.9975126135126136

Spearman Correlation (DA MANEIRA CORRETA)

Consulta	SpearmanCorrelation (TFIDF true vs TFIDF false)
price.$1,099 price.$499 Sensitivity.changes	0.999907155907156
Shutter Speed.Manual Sensitivity.limit **price.$119.95**	0.9992393792393792

Conclui que deu valores próximos porque a nossa frequência de termos nos documentos é bastante parecida. Quando um documento possui um atributo, na grande maioria das vezes, sua frequência em relação a aquele documento é 1. Portanto, os vetores dos documentos com pesos de TFIDF e sem TFIDF relacionados a consulta vão retornar ranks parecidos, ainda que a forma de calcular os valores dos vetores sejam diferentes.

Composição da Resposta

...

Project Structure and Framework

Recomendação usando frequência

```
RECOMENDATIONS = INVERTED_INDEX != null ? INVERTED_INDEX.attributes.stream()
        .collect(Collectors.toMap(
                Function.identity(),
                attribute -> INVERTED_INDEX.termDocuments.entrySet().stream()
                        .filter(entry -> entry.getKey().split("\\.")[0].equals(attribute))
                        .sorted((e1, e2) -> e2.getValue().size() - e1.getValue().size())
                        .limit(3)
                        .map(entry -> entry.getKey().split("\\.")[1])
                        .collect(Collectors.toList())
                )
        ) : null;
```

Recomendação usando mutual-information

```
//MUTUAL INFORMATION CALCULATION
Map<String, Double> attributeFrequencies = INVERTED_INDEX.attributes.stream()
        .collect(Collectors.toMap(Function.identity(),
                attribute -> INVERTED_INDEX.termDocuments.entrySet().stream()
                        .filter(entry -> entry.getKey().split( regex: "\\.")[0].equals(attribute))
                        .mapToInt(entry -> entry.getValue().size())
                        .sum() / (double) INVERTED_INDEX.indexedDocuments.size()
                )
        );
```

```
Map<String, Double> termProbabilities = INVERTED_INDEX.termDocuments.entrySet().stream()
        .map(entry -> {
            double termProbability = entry.getValue().size() / (double) INVERTED_INDEX.indexedDocuments.size();
            String attribute = entry.getKey().split( regex: "\\.")[0];
            double attributeProbability = attributeFrequencies.get(attribute);
            return new Pair<>(entry.getKey(), termProbability / (termProbability * attributeProbability));
        })
        .collect(Collectors.toMap(Pair::getKey, Pair::getValue));
```

```
MI_RECOMENDATIONS =  INVERTED_INDEX.attributes.stream()
        .collect(Collectors.toMap(
                Function.identity(),
                attribute -> termProbabilities.entrySet().stream()
                        .filter(entry -> entry.getKey().split( regex: "\\.")[0].equals(attribute))
                        .sorted((e1, e2) -> (int) Math.signum(e2.getValue() - e1.getValue()))
                        .limit(3)
                        .map(entry -> entry.getKey().split( regex: "\\.")[1])
                        .collect(Collectors.toList())
                )
        );
```