Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу «Информационный поиск»

Студент: В. М. Филиппов
Преподаватель: А. А. Кухтичев
Группа: М8О-410Б

Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №5 «TF-IDF»

Необходимо сделать ранжированный поиск на основании схемы ранжирования TF-IDF. Теперь, если запрос содержит в себе только термины через пробелы, то его надо трактовать как нечёткий запрос, т.е. допускать неполное соответствие документа терминам запроса и т.п. Примеры запросов:

- роза цветок
- московский авиационный институт

Если запрос содержит в себе операторы булева поиска, то запрос надо трактовать как булев, т.е. соответствие должно быть строгим, но порядок выдачи должен быть определён ранжированием TF-IDF. Например:

- роза & цветок
- московский & авиационный & институт

В отчёте нужно привести несколько примеров выполнения запросов, как удачных, так и не

1 Описание

В рамках этой лабораторной работы необходимо доработать поисковый движок, так чтобы он мог ранжировать результаты поиска по метрике TF-IDF

1 TF-IDF

TF-IDF — это статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции или корпуса.

В отличие от булева поиска, который просто говорит «да» или «нет», TF-IDF позволяет ранжировать документы, выделяя те, где искомое слово является наиболее значимым.

Метрика состоит из двух множителей:

- TF (Term Frequency) — частота слова, которая определяет, насколько часто слово встречается в конкретном документе.

$$TF(t,d) = \frac{n_{td}}{\sum_k n_{kd}} \tag{1}$$

  Или можно вот так. Это необходимо, чтобы длинные документы, в которых много раз может встречаться одно и тоже слово, не портили выдачу.

$$wf_{td} = \begin{cases} 1 + \log(tf_{td}), & \text{если } tf_{td} > 0 \\ 0, & \text{в противном случае} \end{cases} \tag{2}$$

- IDF (Inverse Document Frequency) — Обратная частота документа, которая снижает вес слов, которые встречаются слишком часто во всех документах (например, «и», «в», «что», «который»).

$$IDF(t,D) = \log\left(\frac{N}{|\{d \in D : t \in d\}|}\right) \tag{3}$$

Итоговая формула TF-IDF.

$$\text{TF-IDF}(t,d,D) = TF(t,d) \cdot IDF(t,D) \tag{4}$$

2 Исходный код

Для реализации TF-IDF метрики применяется следующее: для каждого токена в документе при добавлении его в индекс мы считаем сколько раз он в нём встретился. Это будет метрика TF. При дампинге на диск в постинг списке теперь будет храниться пара (doc_id, tf).

При поиске алгоритм такой же (получаем токены из запроса, формируем ОПН, получаем документы, которые удовлетворяют нашему запросу) до момента возврата результата поиска. В методе processResult вызывается метод rankResults, внутри которого происходит следующее:

1. В цикле для каждой термы запроса мы считаем IDF

$$IDF(t) = \ln\left(\frac{N}{1 + df_t}\right) \tag{5}$$

   IDF здесь я высчитывал таким образом, чтобы в случае, если токен встречается в каждом документе, это не убило метрику (IDF будет 0 в таком случае).

2. Для каждого документа в постинг списке данного токена проверяем, находится ли этот документ в том, что выдала булева часть алгоритм, если да, то добавляем этому документу score равный $tf * idf$

После того, как каждому документу в выдаче назначен рейтинг, результаты запроса сортируются согласно этому рейтингу.

1 Примеры запросов

При запросе manchester | united поисковик выдал в топе титульную статью с Википедии. Я очень сильно удивился, когда увидел, что в топе выдачи статья на футболиста Джона Айткена, но перейдя по ссылке я понял, что это редирект на статью со списком игроков МЮ за всю историю. 3 ссылка это опять же редирект на статью про Уэйна Рунни, легенду манчестерского футбола.

```
 : manchester | united
Tokens searching: [manchest, unit, ]
 5
https://en.wikipedia.org/wiki/John_Aitken_(footballer,_born_1870) 30.3702
https://en.wikipedia.org/wiki/Manchester_United_F.C. 29.7001
https://en.wikipedia.org/wiki/Wazza_(footballer) 28.2861
https://en.wikipedia.org/wiki/Timeline_of_English_football 27.2021
https://en.wikipedia.org/wiki/Ryan_Giggs 26.5246
 : 0.0281726
 : 99459
```

В этом запросе я попытался найти что-то про неолимпийские медали за теннис или баскетбол. В итоге я получил статьи со всякими разными наградами, в числе которых есть и награды за теннис, и за баскетбол.

```
 : !olympic & medals & (tennis | basketball)
Tokens searching: [olymp, med, tenni, basketbal, ]
 5
https://en.wikipedia.org/wiki/List_of_awards_named_after_people 38.679
https://en.wikipedia.org/wiki/Southeastern_Conference_Athlete_of_the_Year 32.5334
https://en.wikipedia.org/wiki/Southeastern_Conference 32.5334
https://en.wikipedia.org/wiki/List_of_multiple_SEA_Games_medalists 30.988
https://en.wikipedia.org/wiki/Sporting_goods 29.3921
 : 0.0133622
 : 1489
```

Здесь я попытался сделать так, чтобы мне рассказали именно про спортсменов, убрав из выдачи футбол, чтобы не найти списки общих наград. У меня получилось, почти все статьи рассказывают нам о спортсменах.

```
 : !olympic & medals & (tennis | basketball) & !football
Tokens searching: [olymp, med, tenni, basketbal, footbal, ]
 5
https://en.wikipedia.org/wiki/List_of_multiple_SEA_Games_medalists 30.988
https://en.wikipedia.org/wiki/Althea_Gibson 28.7204
https://en.wikipedia.org/wiki/John_Lucas_II 27.788
https://en.wikipedia.org/wiki/E._Lilyan_Spencer 27.0169
https://en.wikipedia.org/wiki/Wii_Sports 26.8262
 : 0.014639
 : 872
```

Поисковик почти не выдаёт статьи с theguardian. Например по запросу "manchester united win arsenal"ссылка на новостной портал встречается лишь на 36 месте. Во-первых, вероятно это связано с непропорциальностью корпуса документов: статей Википедии гораздо больше, чем TheGuardian. Во-вторых, статьи на Википедии длиннее и из-за этого портят метрику TF, даже несмотря на то, что я попытался это скомпенсировать логарифмом.

Время поиска слегка увеличилось из-за того, что приходится сортировать документы по релевантности.

```
#pragma once

#include "index.h"
#include "tokenizer.h"

class IIndexator {
protected:
    std::shared_ptr<Tokenizer> tokenizer;
    std::shared_ptr<RamIndexSource> source;

```

```
public:
    IIndexator(std::shared_ptr<RamIndexSource> src, std::shared_ptr<Tokenizer> tok);
    virtual void addDocument(const std::string_view& url_view, const std::string_view&
        doc_view) = 0;
};

class TFIDFIndexator : public IIndexator {
public:
    TFIDFIndexator(std::shared_ptr<RamIndexSource> src, std::shared_ptr<Tokenizer> tok)
        ;
    void addDocument(const std::string_view& url_view, const std::string_view& doc_view
        ) override;
};
```

```
#include "indexator.h"

TFIDFIndexator::TFIDFIndexator(std::shared_ptr<RamIndexSource> src, std::shared_ptr<
    Tokenizer> tok)
    : IIndexator(src, std::move(tok)) {}

void TFIDFIndexator::addDocument(const std::string_view& url_view, const std::
    string_view& doc_view) {
    uint32_t doc_id = source->getTotalDocs();
    source->urls.emplace_back(url_view);

    tokenizer->tokenize(doc_view);
    const std::vector<std::string>& tokens = tokenizer->getTokens();
    if (tokens.empty()) return;

    HashMap<uint32_t> local_counts;

    for (const std::string& token : tokens) {
        local_counts.get(token)++;
    }

    local_counts.traverse([&](const std::string& term, const uint32_t& tf_val) {
        if (tf_val > 0) {
            std::vector<TermInfo>& global_postings = source->index.get(term);
            global_postings.push_back({doc_id, tf_val});
        }
    });
}
```

```
#pragma once
#include <cstdlib>
#include <cstring>
#include <vector>

#include "index.h"
#include "tokenizer.h"
```

```
8
9  class ISearcher {
10 protected:
11     std::shared_ptr<Tokenizer> tokenizer;
12     std::shared_ptr<IIndexSource> source;
13
14 public:
15     ISearcher(std::shared_ptr<IIndexSource> src, std::shared_ptr<Tokenizer> tok);
16     virtual ~ISearcher() = default;
17     virtual std::vector<std::pair<std::string, double>> findDocument(const std::string&
           query);
18
19 protected:
20     int getPriority(const std::string& op);
21     bool isOperator(const std::string& token);
22     std::vector<int> evaluate(const std::vector<std::string>& tokens, int total_docs);
23
24     std::vector<int> fetchDocIds(const std::string& term);
25     virtual std::vector<std::pair<std::string, double>> processResults(const std::
           vector<int>& docIds,
26                                                                        const std::vector<std
                                                                              ::string>& terms) =
                                                                               0;
27
28     std::vector<std::string> parseQuery(const std::string& query);
29     std::vector<std::string> sortingStation(const std::vector<std::string>& tokens);
30 };
31 class TFIDFSearcher : public ISearcher {
32 public:
33     TFIDFSearcher(std::shared_ptr<IIndexSource> src, std::shared_ptr<Tokenizer> tok);
34
35 private:
36     std::vector<std::pair<int, double>> rankResults(const std::vector<int>& doc_ids,
           const std::vector<std::string>& terms);
37     std::vector<std::pair<std::string, double>> processResults(const std::vector<int>&
           docIds,
38                                                                const std::vector<std::string
                                                                      >& terms) override;
39 };
```

```
1  TFIDFSearcher::TFIDFSearcher(std::shared_ptr<IIndexSource> src, std::shared_ptr<
       Tokenizer> tok) : ISearcher(src, tok) {}
2
3  std::vector<std::pair<int, double>> TFIDFSearcher::rankResults(const std::vector<int>&
        doc_ids,
4                                                                 const std::vector<std::string
                                                                      >& terms) {
5      int N = source->getTotalDocs();
6      std::vector<double> scores(N, 0.0);
7
```

```
    std::vector<bool> isRelevant(N, false);
    for (int id : doc_ids) isRelevant[id] = true;

    for (const auto& term : terms) {
        auto postings = source->getPostings(term);

        double idf = std::log((double)N / (1 + postings.size()));

        for (const auto& entry : postings) {
            if (isRelevant[entry.doc_id]) {
                scores[entry.doc_id] += (1.0 + std::log((double)entry.tf)) * idf;
            }
        }
    }

    std::vector<std::pair<int, double>> ranked;
    for (int id : doc_ids) {
        ranked.push_back({id, scores[id]});
    }

    std::sort(ranked.begin(), ranked.end(), [](const auto& a, const auto& b) { return a
        .second > b.second; });
    return ranked;
}

std::vector<std::pair<std::string, double>> TFIDFSearcher::processResults(const std::
    vector<int>& docIds,
                                                                const std::vector<
                                                                    std::string>&
                                                                    terms) {
    auto ranked = rankResults(docIds, terms);

    std::vector<std::pair<std::string, double>> result_urls;
    for (const auto& pair : ranked) {
        result_urls.push_back({source->getUrl(pair.first), pair.second});
    }
    return result_urls;
}
```

3 Выводы

При выполнении данной лабораторной работы, я узнал про то, как поисковики могут ранжировать результаты. Также я узнал про плюсы и минусы конкретной метрики TF-IDF. Мною был реализован собственный поисковик, использующий инвертированный индекс. В ходе разработки я столкнулся с особенностями подсчета TF и пришел к выводу о необходимости выбора между "сырым"подсчетом вхождений и их логарифмированием для сглаживания весов длинных документов.

Список литературы

[1] Маннинг, Рагхаван, Шютце *Введение в информационный поиск* — Издательский дом «Вильямс», 2011. Перевод с английского: доктор физ.-мат. наук Д. А. Клюшина — 528 с. (ISBN 978-5-8459-1623-4 (рус.))

[2] Understanding TF-IDF (Term Frequency-Inverse Document Frequency) [Электронный ресурс] // GeeksForGeeks. — URL: `https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/` (дата обращения: 18.12.2025).