МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Отчет по лабораторной работе №8
«Ускорение. Прыжки по индексу»
по курсу
«Информационный поиск»

Группа: 80-106М
Выполнил: Демин И.А.
Преподаватель: Калинин А.Л.

Москва, 2019

Задание

Необходимо сделать ранжированный поиск на основании схемы ранжирования TF-IDF.

Выполнение

TF-IDF - статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

Данная метрика будет добавляться в поисковую систему, основанную на булевом поиске. Подсчет метрики будет выполняться на этапе построения индекса по следующим причинам:

1. Для IDF используются общие данные, такие как общее количество статей и количество документов, в которых встречается слово. Если с получением статей для конкретного слова проблем нет, то вот общее количество статей можно получить только храня его в данных.

2. TF так же легче получить в момент индексации, так как для этого нужно получить общее количество слов в документе — здесь, опять же, это возможно только в виде лишнего числа в данных.

Бинарная структура и работа алгоритма

Для более быстрой и качественной работы поисковой системы, была значительно изменена структура бинарных файлов. Количество их не изменилось, в bin_file, который записывался как

vbcode_doc_ids, write_freq, for_write_pos_in_file

vbcode_doc_ids — doc_id сжатые VB
write_freq — частоты встречи слова в каждом файле
for_write_pos_in_file — позиции в файлах сжатые VB

теперь добавлен еще и показатель TFIDF следующим образом:

vbcode_doc_ids, write_tfidf, write_freq, for_write_pos_in_file

vbcode_doc_ids — doc_id сжатые VB
write_tfidf — показатель TFIDF для каждого документа
write_freq — частоты встречи слова в каждом файле
for_write_pos_in_file — позиции в файлах сжатые VB

Преимуществами добавления показателей именно так являются:
1. Сохранение размера обратного индекса, хранимого в ОЗУ.
2. Не создается дополнительный файл, что, как было выяснено в предыдущей ЛР сказывается на скорости работы поиска
3. Удобное чтение и использование в момент построения поисковой выдачи

Также в данной ЛР я столкнулся со следующей проблемой — как считать TFIDF сохранив при это возможность использовать скобки и встроенную функцию eval(). Проблема возникла из-за того, что суммарный TFIDF для запроса & считается как сумма, что не вызывало вопросов, а вот для | нужно было делать рассчет в момент пересечения set()-ов, что бы сохранить порядок операций, а у нас были dict-ы (в Python они не поддерживают использование логических операций «из коробки»). Для решения данной проблемы было выбрано такое решение:
1. Изменить структуру при построении поискового результата на такую:
{
    doc_id: [TFIDF, [pos1, pos2, pos3]]
}
для цитатного поиска и на
{
    doc_id: [TFIDF,]
}
для бинарного. Такое преобразование позволило сохранить прежнюю логику без кардинального изменения работы.
2. Расширить возможность стандартного dict() написанием производного класса и определить в нем 2 «магических» метода: __and__() и __or__(), которые как раз и используются при интерпретации логических операций.
3. Схема просчета TDIDF следующая -
    3.1 Для & - сумма
    3.2 Для | - сумма, если слово есть пересечение, иначе только одно значение.

Таким образом удалось сохранить возможность пользоваться прежней логикой поиска и оставить компрессию.

Результаты

Напомню результаты тестовых запросов, а так же ранних версий поисковой системы (поиск со сжатие без ранжирования)

Для поиска по википедии:

"search_sys": *"wiki"*,

"P@1": 0.9655172413793104,
"P@3": 0.9195402298850575,
"P@5": 0.9195402298850575,

"DCG@1": 4.551724137931035,
"DCG@3": 9.223221721921302,
"DCG@5": 12.284175920616793,
"NDCG@1": 0.9103448275862069,
"NDCG@3": 0.6148814481280869,
"NDCG@5": 0.4913670368246717,
"ERR@1": 0.9103448275862069,
"ERR@3": 0.6148814481280869,
"ERR@5": 0.4913670368246717

Для google:

"search_sys": *"qoogle"*,

"P@1": 1.0,
"P@3": 1.0,
"P@5": 1.0,
"DCG@1": 4.896551724137931,
"DCG@3": 9.975266464532151,
"DCG@5": 13.657926517972932,
"NDCG@1": 0.9793103448275862,
"NDCG@3": 0.6650177643021434,
"NDCG@5": 0.5463170607189172,
"ERR@1": 0.9793103448275862,
"ERR@3": 0.6650177643021434,
"ERR@5": 0.5463170607189172

Оценки для предыдущих версий:

"search_sys": *"prev_system_vers"*,

"P@1": 0.8275862068965517,
"P@3": 0.8275862068965518,
"P@5": 0.8275862068965518,
"DCG@1": 3.2413793103448274,
"DCG@3": 7.036870908374482,
"DCG@5": 9.691327170149046,
"NDCG@1": 0.6482758620689654,
"NDCG@3": 0.4691247272249654,
"NDCG@5": 0.38765308680596183,
"ERR@1": 0.6482758620689654,
"ERR@3": 0.4691247272249654,
"ERR@5": 0.38765308680596183

Оценки для системы с ранжированным поиском

"search_sys": *"tfidf"*,

"P@1": 0.9845346456654655 ,
"P@3": 0.9355172413793104,
"P@5": 0.9355172413793104,
"DCG@1": 4.255172413793103,
"DCG@3": 9.37789684334988,
"DCG@5": 12.574954530953492,
"NDCG@1": 0.8910344827586207,
"NDCG@3": 0.6085264562233255,
"NDCG@5": 0.4849981812381396,
"ERR@1": 0.9310344827586207,
"ERR@3": 0.6385264562233255,
"ERR@5": 0.5149981812381396

Качество поиска, ожидаемо, значительно выросло. Стоит, конечно, отметить субъективность данных оценок, так как они производились в разное время и условия несколько отличались, однако при таких показателям созданная система почти соответствует оценкам системы википедии, что можно считать достойным результатом.

Время поиска незначительно увеличилось, это можно объяснить наличием сортировки по TFIDF.

Если убрать ограничение поисковой выдачи, то она совпадает с результатами предыдущих ЛР, что говорит о том, что все нужные статьи были найдены. Это ожидаемое поведение, так как изменения в алгоритме поиска не произошли — добавление TFIDF повлияло только на сортировку при выдаче.

Вывод

В ходе лабораторной работы я познакомился метрикой TF-IDF для ранжированной выдачи и применил ее для своей системы, чем значительно улучшил качество поиска.