Содержание

1

1.1 Онлайн тестирование. A/B testing, interleaving

Простые кликовые метрики:

- Click
  - Click-through rate (CTR) — сколько раз кликнули
  - Click rank (reciprocal rank) — как далеко был сделан клик
  - Abandonment
- Time
  - Dwell time (сколько смотрят на страничку)
  - Time to first click
  - Time to last click
- Queries
  - Number of reformulations
  - Number of abandoned queries

A/B testing
Control/Treatment
Treatment даём $0.5 - 1.0\%$ пользователей
Минусы:

- пользователи сильно разные
- не очень чувствительны (т.к. пользователи и там, и там пару верхних строчек смотрят тоже)
- нужно много статистики собрать

Interleaving
Смешиваем Control и Treatment Team Draft interleaving: на каждую позицию подкидываем монетку и берём следующий итем из соответствующей выборки (надо учитывать одинаковые итемы)
Можем сравнивать только одинаковые системы (нельзя дизайн сравнить, например)

1.2 Поиск дубликатов. Hash. Shingles

Хотим бороться с Mirroring сайтами, например
Если хотим совсем одинаковый контент детектить, можно просто страничку хешировать.

Near duplicate detecting

- Выписываем все n-граммы ($n \in [4, 9]$)
- Считаем хэши от n-грамм
- Считаем Jaccard coefficient (расстояние между множествами)
  $J(d_1, d_2) = \frac{|H(d_1) \cap H(d_2)|}{|H(d_1) \cup H(d_2)|}$
- Говорим, что дубликаты, если $J(d_1, d_2) > treshold$
- n-грамм много, получится долго, так что на практике просто берут первые k (по алфавиту) хэшей

2

2.1 Оффлайн тестирование. Precision@k, recall@k, F-measure, MAP

Это оценки, которые не учитывают ранк (т.е. просто нашли/не нашли)

$$Precision = \frac{\#(relevant\ items\ retrieved)}{\#(items\ revrieved)} = P(relevant|retrieved)$$

$$Recall = \frac{\#(relevant\ items\ retrieved)}{\#(relevant\ items)} = P(retrieved|relevant)$$

F-measure

$$F_\beta = \frac{1}{\alpha\frac{1}{recall} + (1-\alpha)\frac{1}{precision}}$$

где $\beta = \frac{1-\alpha}{\alpha}$ (насколько precision важнее recall)
F_1-мера — среднее гармоническое просто

@k

$$P@k = \frac{\#(relevant\ items\ at\ k)}{k}$$

Хотим, чтобы была поближе к 1

$$R@k = \frac{\#(relevant\ items\ at\ k)}{\#(relevantitems)}$$

Странная штука, в абсолютной величине мало что значит

Reciprocal rank

$$RR = \frac{1}{rank\ of\ the\ first\ relevant\ item}$$

В идеале, конечно, 1

Average precision (AP)

$$AP = \frac{\sum_k^{\#(relevant\ items)} P@k}{\#(relevant\ items)}$$

Average over multiple queries
mean P@k, mean R@k, MRR, MAP (mean average precision = *средняя средняя точность, хаха*)

2.2 Обратный индекс. Типы обратных списков

Inverted index

Делается из forward index'a. По терму даёт документы, в которых он встречается.
Храним словарь:

- Число документов
- Указатель на начало списка (inverted list)
- Какие-то ещё *мета*-данные

Какое-нибудь B-дерево или хэш-таблица там..

Inverted lists

- Document identifiers
  просто список документов, где терм встречается
- Frequencies
  для каждого документа ещё храним, сколько раз там есть этот терм
  можно раздавать им weight/score, например, слово в заголовке/abstract/external link важнее
  учитывать частотность тоже можно
- Positions
  храним пары <документ, позиция> чтобы искать только *tropical fish*, а hetropical *fish* или *tropical* hefish не искать

В каком порядке это всё писать?
Как бы, проходить все документы каждый раз не очень полезно, мы большую часть из них вообще никогда выдавать не будем.
Можно посортить документы по весу
Но тогда пары (tropical fish) искать не удобно :(
Сделаем несколько бинов по весу, а внутри их посортим уже лексикографически

3

3.1 Оффлайн тестирование. DCG, NDCG, Rank-biased precision, expected reciprocal rank

Учитываем поведение пользователя на поисковой страничке.

Discounted cumulative gain (DCG)
R_i — graded relevance ($[0 \dots 4]$)
Тут Gain — польза для пользователя

$$CG@k = \sum_{i=1}^{k} (2^{R_i} - 1)$$

$$DCG@k = \sum_{i=1}^{k} \frac{2^{R_i} - 1}{\log(i+1)}$$

Normalized

$$NDCG@k = \frac{DCG@k}{DCG_{ideal}@k}$$

Ideal — считаем DCG от оптимального набора relevance (4, 4, 3, 2, 2, 1, 0), например

Rank-biased precision
Моделируем поведение пользователя:

- Смотрим следующий итем с вер-ю θ
- Соответственно, выходим с вер-ю $(1 - \theta)$
- Вер-ть посмотреть на k-ый итем, $P(look\ at\ k) = \theta^{k-1}$
- Матожидание числа итемов: $Avg\#exam = \sum k \cdot \theta^{k-1}(1-\theta) = \frac{1}{1-\theta}$
- Utility at rank k, $U@k = P(look\ at\ k) \cdot R_k = \theta^{k-1} \cdot R_k$
- RBP — усреднённая $U@k$, $RBP = \frac{\sum U@k}{Avg\#exam}$
- θ близка к 1

Expected reciprocal rank (ERR)

- Считаем, что пользователь останавливается когда находит релевантный результат
- Вероятность остановиться, $\zeta_i = \frac{2^{R_i}-1}{2^{R_{max}}}$
- $ERR = \sum \frac{1}{k} \cdot \theta^{k-1} \cdot \zeta_k \cdot \prod^{k-1}(1 - \zeta_i)$
  θ на случай, если пользователю наскучит искать

3.2 Кроулеры. Порядок обхода

Crawling — потому что они как паучки, которые ползают по мировой паутине.
Основные два объекта, которые он собирает — содержание страницы и ссылки.
Общая схема, что делает кроулер: берёт страничку, данные кладёт в storage, смотрит на исходящие ссылки, если там что-то интересное, кладёт их в очередь

Какие бывают кроулеры:

- Vertical Search Engine — ищет в одной конкретной области
- Personal Crawler
- Shopbots — Выискивают цены)
- Feed Crawlers
- Archive Crawlers — работают на системы, занимающиеся архивацией интернета
- Mirroring Systems — смотрят, обновилась ли основная страница, чтобы обновлять зеркала

Какие бывают странички:

- Private — что вообще не подключено к мировой сети
- Surface Web — куда можно попасть по ссылке (индексируется)
- Deep Web — куда нельзя попасть не заполняя форму (не индексируется)
- **D**a**R**k **W**e**B** — где происходят *тёмные делишки*

Есть понятие *crawler politeness*: нужно представляться, читать robots.txt, не нагружать сайт сильно-сильно

В каком порядке обрабатывать сайты?
Есть такое понятие *frontier* — странички, про которые мы знаем, что они есть (у нас есть ссылки на них), но ещё не посетили их. Вот по нему в каком-нибудь порядке и ходим (random/bfs/in-degree/potential impact on search quality)

4

4.1 Сбор тестовой коллекции. Оценка релевантности

Test collection:

- Test documents
  репрезентативная коллекция для вашей поисковой системы, похожего содержания, размера и типа
  Да, это сложно
- Test queries
  Откуда брать запросы? Если у вас есть уже какая-то простая более-менее работающая система, можно собирать из неё логи
  Ну или можно какой-нибудь соц опрос устроить
- Ground truth
  Это делают только люди (assessors). Что за люди:
  - юзеры (через какую-нибудь форму обратной связи)
  - независимые (не пользователи) эксперты
  - crowdsourcing (не очень хорошие асессоры)

  Нужно достаточно много оценок, желательно, чтобы оценки хорошо покрывались (было и много 0, и много 4). И нужно, чтобы каждую пару запрос-документ несколько человек оценило. По логике всё, в общем.

Есть много уже собранных датасетов (их к конференциям/соревнованиям разным собирают). Самый главный — TREC (Test REtrieval Conference).

Оценки релевантности (*relevance*): $[0, 4]$, $[0, 3]$
4 = супер хорошо, 2-3 = нормально.
Бывает ещё когда 4 = vital (супер обязательно, например, страничка из вики)
Ещё бывает -1 = spam

Assessment pooling.
Depth-k pooling. Берём несколько простых систем (или не простых), задаём каждой запрос, берём от каждой top-k, даём асессорам на оценку.
Смотрим ещё на **inter-assessor agreement.** Считаем Cohen's kappa coefficient (для двух асессоров).
$K = \frac{P(A)-P(E)}{1-P(E)}$,
$P(E)$ — ожидаемая вероятность совпадения (сумма произведений для каждой оценки),
$P(A)$ — совпадения на самом деле.
Если пацанов несколько, считаем среднее попарное K.
Если $K > 0.8$ — хорошо, $K < 0.67$ — плохо, скорее всего плохо написали инструкцию для асессоров.

4.2 Ранжирование. RankNet, LambdaRank

Тут будет ML ... (нет)
Ну вот у нас есть вектор $(q, x_1 \dots x_n)$ (q — запрос, x_i — фичи документа), хотим по ним предсказывать релевантность y.
В качестве фичей можно использовать как раз всё, что использовали раньше (до мля): TF-IDF всякие, BM25, PageRank, HITS и ещё кучу всего.
Хорошо ещё кликовые фичи работают.
В чём их проблема: есть эффект доверия поисковику, поэтому кликовые фичи получается как бы включают в себя ещё и результат всей системы ранжирования. Ещё так трудно найти что-то новое, важность клика не даст нам поднять хороший сайт со второй страницы выдачи на первую

Есть три типа алгоритмов, как обучать машины ранжированию:

- Pointwise — просто предсказывать релевантность (классификация/регрессия) (ну как я в начале написала)
  Беда в том, что мы можем хорошо угадывать порядок (документов), но при этом плохо попадать в само число (релевантность)
- Pairwise — для пары документов определяем, кто их них лучше (компаратор, короче)
- Listwise — ну а тут список целиком

RankNet
Он делает Pairwise
Так, у нас есть Pointwise scoring function $f(x_i)$
Ground truth: $\overline{P}_{ij} = \mathcal{I}(x_i > x_j)$
Хотим приближать вероятность $(x_i > x_j)$ с помощью логистической регрессии
$P_{ij} = P(x_i > x_j) = \frac{1}{1+e^{-\sigma(f_i - f_j)}}$ Считаем pairwise loss function (это просто кросс-энтропия), ну, собственно, её и оптимизируем.
$C = -\overline{P}_{ij} \log P_{ij} - (1 - \overline{P}_{ij}) \log(1 - P_{ij}) = (1 - \overline{P}_{ij})\sigma(f_i - f_j) + \log(1 + e^{-\sigma(f_i - f_j)})$

тут я устала от этого вашего машинного обучения, тем более были только беспруфные кукареки какие-то ...

5

5.1 Кроулеры. Обновление базы. Freshness age, expected age. Очереди обходов

Хотим хорошо обновлять свою базу, т.е:

- Детектить изменения
  есть свойство `last modified` у странички
- Обрабатывать изменения
- Предсказывать изменения
- Выбирать, когда и в каком порядке что апдейтить

Freshness

$F_p(t) = \begin{cases} 1 & \text{страничка } \textbf{НЕ} \text{ поменялась с последнего crawl'a} \\ 0 & \text{иначе} \end{cases}$

Плохая метрика, т.к. не различает для часто меняющихся страничек, обновляем мы их раз в час, или раз в год

Age

$A_p(t)$ = время, прошедшее с **первого** изменения странички, произошедшего после последнего crawl'a

Expected age

λ — среднее число изменений в день

$A_p(\lambda, t) = \int\limits_0^t P_{changed}(x)(t-x)dx = \int\limits_0^t \lambda e^{-\lambda x}(t-x)dx$ Тактика: походили на сайт достаточно часто, выяснили λ, потом используем её.

Какие страницы вообще надо апдейтить (что учитываем):

- Average daily click count
- Estimated update frequency

Делаем несколько очередей:

- Новостные сайты (популярные и обновляются часто)
- Популярные, но обновляются не так часто
- Остальные сайты

5.2 Ранжирование. TF-IDF, BM25

Это простые методы, которыми пользовались, пока не научились обучать машины (сейчас они используются как фичи для машинок)

Вообще, ранкинг бывает трёх типов:

- Term-based
- Link-based
- ML-based

В этом вопросе всё term-based

Vector space model
Документ — это вектор термов (1/weight на позиции соответствующего терма из словаря). Понятно, что запрос — это тоже вектор, но более sparse.
Самый просто вариант — искать самый близкий к вектору-запросу вектор-документ.
Используем для этого косинусное расстояние (косинус угла между векторами)

$$Cosine(D_i, Q) = \frac{D_i \cdot Q}{||D_i||\ ||Q||} = \frac{\sum d_{i,j} \cdot q_j}{\sqrt{(\sum d_{i,j}^2) \cdot (\sum q_j^2)}}$$

Скалярное произведение быстро считается (используя inverted index), т.к. не 0 там будет только для термов из запроса. Ну а нижние корни — это просто константы.
Веса (weights):

- binary (0/1)
- Term Frequency (обычно просто raw count)
- TF-IDF

TF-IDF
На самой деле это целый класс подходов к оценке частот термов.
$d_{ik} = \text{TF-IDF}(i,k) = tf_{ik} \cdot idf_{ik}$

Term Frequency (TF)
$tf_{ik} = \frac{f_{ik}}{\sum_j f_{ij}}$,
где f_{ik} — сколько раз терм k встречается в документе D_i

Inversed Document Frequency (IDF)
$idf_k = \log \frac{N}{n_k}$,
где N — число документов, n_k — число документов, содержащих терм k
IDF как бы нормирует всякие частые слова типа местоимений
log потому N может быть большим ...
Вместо TF и/или вместо IDF могут использоваться что-то другое, но похожее (в основном, там разные нормализации и сглаживания)
Для запросов тоже нужно считать term weights, обычно говорят, что это делается “in a similar manner”.
Но для этого нам нужна коллекция запросов. Пока её нет, можно только бинарные/унарные характеристики для TF/IDF делать.
Можно ещё модифицировать всё это дело, чтобы учитывать релевантность. Это называется Roccio modification

$$q'_j = \alpha q_j + \beta \frac{1}{|Rel|} \sum_{D_i \in Rel} d_{ij} - \gamma \frac{1}{|Nonrel|} \sum_{D_i \in Nonrel} d_{ij}$$

Общий смысл: для каждого терма в запросе ещё учитываем, сколько раз он встречался в релевантных и нерелевантных документах (релевантность относительно запроса).
Reasonable parameters (α, β, γ): (8, 16, 4) или (1, 0.75, 0.15)

TF saturation
$\frac{TF}{TF+k}$

BestMatching25 (BM25)
Это формула для расстояния (вместо косинусного)

$$BM25(D_i, Q) = \sum_{t \in Q} \frac{f_{it} \cdot (k+1)}{(f_{it} + k(1 - b + b\frac{l(D_i)}{avgl(D)})} \log \frac{N - n_t + 0.5}{n_t + 0.5}$$

($l(D)$ — длина документа D, $avgl(D)$ — средняя длина)
(Легчайшая формула, я считаю)

Приходите к нам в BM25, у нас есть:

- $\frac{f_{it}}{f_{it}+k}$ — TF saturation
- $\frac{l(D_i)}{avgl(D)}$ — учитывание длины документа
- $\log \frac{N-n_t+0.5}{n_t+0.5}$ — fancy IDF
- k, b — mAgIc CoNsTaNtS, $k \in [1.2, 2], b = 0.75$

Мне кажется, в тот момент, когда люди добавляют три коэффициента в формулу, надо переставать говорить, что она чем-то мотивирована
Жесть, какой большой вопрос ...

6

6.1 Препроцессинг текста. Стемминг

Text processing pipeline:

- Удаляем пробелы и знаки препинания
- Удаляем большие буквы (в начале предложений), имена собственные не трогаем (entity recognition)
- Удаляем stop-слова
- Конвертируем term $\rightarrow$ stem
- Работаем с фразами
- Применяем какие-то language-specific processing rules

Стоп-слова это кто (who?)

- Frequency-based
  Задаём порог частотности, всё, что чаще — в мусорку
- Dictionary-based
  Создаём мусорный словарь

Последнее время не очень их любят, потому что они часто есть во фразах (устойчивых выражений)

Stemming

- Dictionary-based
  храним словарь <“инфинитив”, все возможные словоформы>
  нужно аккуратно работать с омонимами/омоформами (словами, которые пишутся одинаково, но значат разное)
  Проблема: если слова нет в словаре, то грустим
- Algorithmic
  Самый известный (для английского) — Porter stemmer (там куча последовательных ифов)
- Hybrid
  Krovetz stemmer: Пока слова нет словаре, пытаемся отрезать от него кусочек (суффикс (чаще) или префикс (реже))
  В итоге получаем инфинитивы (а не стемы, как в Porter stemmer)

Все эти стеммеры делаются отдельно для каждого языка, поэтому на английском лучше искать в гугле, а на русском — в яндексе

Что делаем с фразами: ищем частые n-граммы / ищем *noun phrases* / в inverted индексе ищем tropical fish

6.2 Ранжирование. PageRank, HITS

Вот эти ребята — это Link-based ranking methods
Web graph: вершины — документы, рёбра — ссылки, на рёбрах написаны тексты ссылок

Random walk
Давайте по нему случайно ходить (как пользователи): начинаем на рандомной странице, переходим по ссылке

$$p(d_i) = \sum_{j:d_j \to d_i} \frac{p(d_j)}{|k : d_j \to d_k|}$$

Ну понятно, просто формула как в Марковских цепях

Teleportation
Если попали на страницу, из которой нет исходящих ссылок, не будем же мы там вечно сидеть. В таком случае мы просто равновероятно прыгнем на любую страницу в интернете

$$p(d_i) = \alpha \frac{1}{N}$$

Ну и понятно, что телепортироватся можно не только от безысходности, но и просто так

PageRank
Собственно, итоговая формула

$$p(d_i) = (1 - \alpha) \sum_{j:d_j \to d_i} \frac{p(d_j)}{|k : d_j \to d_k|} + \alpha \frac{1}{N}$$

Ну и по Эргодической теореме Маркова у нас существует стационарное распределение π.
Тогда $\pi(d)$ — это и есть PageRank.
PageRank придумал Larry Page, один из основателей гугла.
Есть два способа стационарное состояние найти: алгоритмом Гаусса (потому что $\pi = \pi P$) или найти распределение на каком-нибудь большом шаге: $\pi \sim x_0 P^n, n \to \infty$, говорят, что делают вторым.
Что не так с PageRank? Он никак не учитывает слова на ссылке.

Hypertext-included topic search (HITS)
Это всё создавалось в начала 2000, тогда интернет был маленьким и считалось, что он выглядит как-то так:

- есть Хабы (Hubs) и Авторитеты (Authorities)
- Авторитет — страничка с ответом на наш запрос
- Хаб — страничка с хорошим списком ссылок, в том числе со ссылкой на наш Авторитет
- Т.е. у нас двудольный такой граф получается
- Hub score: $h(d) = \sum_{y:d \to y} a(y)$
- Authority score: $a(d) = \sum_{y:y \to d} h(y)$

- Тогда алгоритм примерно такой:
  - С помощью тупой системы (например, на основе BM25) собрать список документов, примерно относящихся к запросу
  - Построить по этим документам web-подграф
  - Выдать всему изначально h и a равные 1
  - Итеративно их пересчитываем (пока меняются) (с нормализацией — на корень из суммы квадратов делим)
  - Выдаём пользователю top-scoring хабы и авторитеты

  Можно запускать HITS для каждого терма в отдельности, потом использовать полученные h и a как веса

7

7.1 Обратный индекс. Two-pass index, one-pass index with merging

Это про то, как так хранить индекс, чтобы при добавлении одного документа, всё перезаписывать не приходилось

Two-pass index
Сначала один раз проходимся по forward индексу, считаем, число записей $\Rightarrow$ сколько нужно памяти выделить. Потом эту память выделяем. Потом ещё раз проходимся и уже на нужные места всё кладём

One-pass index with merging
Делаем два маленьких индекса $\rightarrow$ сливаем в один побольше
Ну, и конечно, для всего этого юзаем Map-Reduce

7.2 Расширение запроса. Term association, relevance feedback

Тезаурусы
Нужны ещё, чтобы находить связи между словами
“tank aquarium” $\rightarrow$ + “fish”
Бывают мануальные, ещё есть WordNet
Controlled vocabulary — каноничный термин (для каждой области), по которому нужно всё искать.
Есть ещё автоматические (by context similarity) (тут было про 4 метрики разные)

Relevance feedback

- Юзер задаёт вопрос
- Система возвращает initial set результатов
- Какие-то результаты помечаются как (не)релевантные
- Система учитывает это и переделывает выдачу

Как оценивать релевантность

- (Explicit) Relevance feedback —- нанять асессоров (или пользователей попросить)
- Pseudo-relevance — top-k из выдачи
  типа мы спросили “tank aquarium”, в первых k документах “fish tank” $\Rightarrow$ добавляем “fish”
- Implicit relevance — смотрим, куда пользователи кликают

Using query log
Можем смотреть, чем расширять термы в старых запросах
Удобно, потому что запросы короче документов

8

8.1 Типы подходов к объявлению обратного индекса

No merge
Отдельно храним *Old Main index* и *Delta index* (несколько)
Минус: для каждого запроса нужно идти и в основной индекс и в дельты, потом смотреть, где у нас что-то поменялось (даты сравнивать)

Incremental update
Примерно как вектор (типа есть свободное место (capacity - size))
Минус: иногда нужно расширять буфер, при этом нельзя будет писать/читать

Immediate merge
Переписываем весь индекс каждый раз, когда приходит дельта

Lazy merge
Потихоньку сливаем индекс, причём ещё не слитые дельты тоже как-то организованно храним
Выглядит примерно как пополняемые структуры, типа мы сливаем дельты, пока они не станут супер-большими, тогда их вливаем в большой индекс
Тут была шутка про Валрусы

8.2 Дополнение запроса. RW on query flow graph. RW on co-clicked graph

Тут оцениваются в основном логи. Типа пользователь либо сам дописал, что мы ему предложили, либо кликнул
Можно расширить эту концепцию и нарисовать граф переходов (с вероятностями)

Random walk (RW) on query flow graph
Начинаем с исходного запроса, смотрим, куда можно доблуждать, выбираем самое вероятное и предлагаем

Random walk (RW) on co-clicked bipartite graph
Двудольный граф: запросы — сайты.
$w(i,k)$ — сколько раз по запросу i кликают на сайт k (и наоборот)
$p_{ij} = \sum\limits_{k \in V_2} \frac{w(i,k)}{z_i} \frac{w(k,j)}{z_j}$
Для каждого сайта считаем $h_i(t+1) = \sum\limits_j p_{ij} h_j(t)$ — с какой вероятностью дойдём до запроса i через время t
Выдаём сайты с минимальным средним h_i

9

9.1 Распределение обратного индекса, подокументное, позапросное

Это как мы индекс по нодам распределяем

Document based

- все упоминания одного документа держатся на одной ноде
- более сбалансированное (для терма трудно предсказать, во скольких документах он встретится, а для документа легко)
- запрос уходит на несколько нод сразу $\Rightarrow$ можно параллелить
- но приходится тогда на каждой ноде хранить весь словарь
- ещё такой строится легче

Term-based

- менее сбалансированное
- вся информация для терма лежит на одной ноде, не нужно потом ничего объединять
- файлы с атрибутами страниц приходится дублировать
- конструировать сложно

9.2 Обработка запросов. Исправление опечаток, поиск омофонов

Смысл обработки запросов: дойти от query (запроса) до intent (то, что юзер на самом деле хочет).

Основные этапы:

- Normalisation — убираем капитализацию, перевод в одну кодировку (utf-8)
- Spelling correction
  тут на следующий этап передаёт не только спелл-чекнутый запрос, но и оригинальный
- Segmentation — terms, phrases, urls ...
- Stemming
- Annotation — entity extraction, geotagging
- Tern expansion — synonyms, plurals ...
- Бывает ещё query relaxation — это когда у нас ничего не нашлось на оригинальный запрос, но на менее точный можно найти

Если запрос мультиязычный, один из зыков превращается в entity (потому что это, скорее всего, какое-нибудь название)

Дальше будет про *Spell checking*

Simple typos
Берём какой-то подмножество слов (например, с той же первой буквой + примерно такой же длиной)
Считаем **Damerau-Levenshtein distance** (как просто Левенштейна + можно делать свап соседних букв)
Есть другой способ: k**-gram index optimization**
Разбиваем наше слово на k-граммы, потом ищем слова, у которых Jaccard coefficient ($|A \cap B|/|A \cup B|$) (по числу общих k-грам) самый большой

Homophones & Soundex Code
Омофоны — слова, которые одинаково слышатся.

Soundex code:

- Оставляем первую букву (in uppercase)
- Заменяем {a, e, o, i, u, y, h, w} (все гласные + h + w) на дефис
- Остальные буквы бьются на 6 классов (по похожести) и заменяются цифрами
- Удаляем подряд-идущие одинаковые символы
- Удаляем все дефисы
- Берём первые три цифры (если их меньше, то нули)

Ну, понятно, что всё мы так не засечём (например, удаление буквы)

Multiple corrections

Noisy channel model

- Человек хочет написать слово w с вероятностью $P(w)$
- Человек пытается его написать
- Но шумный канал (мозг) заставляет его написать слово e с вероятностью $P(e|w)$

Ранжируем по $P(w|e) = \frac{P(e|w)P(w)}{P(e)} \propto P(e|w)P(w)$ (пропорционально)

$P(w) = \frac{tf(w)}{\sum\limits_{w_i \in C} tf(w_i)} \leftarrow$ учитываем популярность слов

$P(e|w)$ например Дамерау-Левенштейном считаем

Considering context

Теперь ранжируем по $P(w|e)\hat{P}(w)$, где $\hat{P}(w) = \lambda P(w) + (1 - \lambda)P(w|w_p)$

$P(w|w_p)$ — вероятность в контексте