МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ (ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ)»

ФАКУЛЬТЕТ ИННОВАЦИЙ И ВЫСОКИХ ТЕХНОЛОГИЙ
КАФЕДРА АНАЛИЗА ДАННЫХ

Выпускная квалификационная работа по направлению 01.03.02 "Прикладные математика и информатика"
На тему:

Гладкая метрика для задачи ранжирования

Студент _____ Левков Мирон Николаевич

Научный руководитель _____ Воронцов Александр Сергеевич

Зам. зав. кафедрой _____ Бунина Елена Игоревна

Москва, 2017

Гладкая метрика для задачи ранжирования

Рассматривается задача построения гладкой метрики ранжирования документов в поисковых запросах. Предложены различные способы сглаживания метрики DCG, также исследованы имеющиеся в данной области результаты. Проведен анализ зависимости качества сглаженных метрик от гиперпараметров и размера выборки поисковых запросов.

Содержание

Введение

Описание задачи ранжирования

Пусть имеется некий поисковый запрос. Задача ранжирования состоит в том, чтобы из списка всех доступных документов выбрать наиболее хорошие (релевантные) и показать их пользователю в поисковой выдаче. Для того, чтобы измерять качество алгоритма, решающего данную задачу, используются оценки релевантности документов запросу, проставленные людьми. Метрика качества дает возможность получить более хороший ранжирующий алгоритм, улучшающий качество поиска.

Обозначения

Как правило метрику ранжирования измеряют не на одном запросе: берется большое количество запросов и вычисляется среднее значение метрики по ним.

Рассмотрим один поисковый запрос.

Обозначение 1.1. Обозначим $\{d_i\}_{i=1}^n$ - набор *документов*, которые нужно ранжировать по данному запросу; $\{r_i\}_{i=1}^n$ - *оценки* этих документов, проставленные асессорами;
$\{s_i\}_{i=1}^n$ - оценки релевантности (*скоры*), выданные ранжирующим алгоритмом

Определение 1.1. Метрика качества ранжирования **DCG** определяется по формуле:

$$DCG = \sum_{i=1}^{k} \frac{r_{p_i}}{\text{discount}(i)}$$

где

$p_1, \ldots, p_n$ - перестановка на множестве $\{1, \ldots, n\}$, т.ч. $s_{p_1} > s_{p_2} > \cdots > s_{p_n}$

k - количество документов, по которым считается метрика ($k \leq n$) - важен в том случае, когда нас интересуют лишь несколько первых документов в выдаче

discount(i) - дисконтирующий фактор, как правило $\frac{1}{\log(i+1)}$ - однако в поисковых метриках Яндекса используется $\frac{1}{i}$, поэтому мы будем использовать его

Постановка задачи

Все DCG-подобные метрики ранжирования имеют такую конструкцию, в которой значение метрики напрямую не зависит от скоров алгоритма (при подсчете значения учитывается лишь перестановка на документах, заданная скорами алгоритма).

Несложно заметить, что для такой конструкции при фиксированном наборе документов и их оценок данная метрика может принимать лишь конечное количество различных значений. При этом изменение значения метрики происходит лишь в случае перестановки местами двух документов в выдаче.

Как было замечено выше, основная проблема подобных метрик в том, что они не имеют гладкой зависимости от скоров. В то же самое время логично ожидать, что, если ранжирующий алгоритм выдал трем документам оценки $\{100, 1, 0.5\}$, а другим трем документам - $\{10, 1, 0.5\}$, то он считает первый документ из первой тройки сильно лучшим, чем первый документ из второй тройки. В добавок, гладкая относительно скоров метрика является более чувствительным инструментом для оценивания качества алгоритма и отслеживания случаев недообучения или переобучения.

В данной работе исследуются разные подходы к построению метрик ранжирования, с целью получения метрики, которая бы гладко зависела от скоров и хорошо коррелировала с DCG. Далее мы уточним эти критерии.

Метрики качества результатов

Хотелось бы, чтобы наша метрика обладала двумя свойствами. Во-первых, метрика должна быть «реалистичной». Т.е. ее локальные минимумы/максимумы должны соответствовать минимумам/максимумам DCG.

Определение 3.1. Пусть $\{y_i\}_{i=1}^k$ - значения метрики DCG на данном пуле запросов для k различных значений набора гиперпараметров. Пусть $\{\hat{y}_i\}_{i=1}^k$ - значения нашей метрики на том же пуле при тех же наборах гиперпараметров. Тогда **ошибкой аппроксимации** метрики DCG нашей метрикой назовем величину $\min\limits_{\alpha,\beta} \frac{1}{k} \sum\limits_{i=1}^{k} (\alpha \cdot \hat{y}_i + \beta - y_i)^2$

Обозначение 3.1. Далее будем обозначать *ошибку аппроксимации*, как $\mathrm{err}_{\mathrm{approx}}$(Metric Name)

Также придуманная метрика должна быть гладкой. При этом необходим способ подсчета гладкости конкретной функции, если она задана не аналитически, а численно. Определим метрику гладкости.

Определение 3.2. Пусть $\{y_i\}_{i=1}^k$ - значения нашей метрики на данном пуле запросов для k различных значений набора гиперпараметров. Тогда **ошибкой гладкости** функции, посчитанной **через усреднение модуля разности** в соседних точках, будем называть величину

$$\mathrm{err}^{\mathrm{abs}}_{\mathrm{smooth}}(y_1, \ldots, y_k) = \sum_{i=2}^{k} |y_i - y_{i-1}| \cdot \frac{1}{|y_k - y_1|}$$

Определение 3.3. Пусть $\{y_i\}_{i=1}^k$ - значения нашей метрики на данном пуле запросов для k различных значений набора гиперпараметров. Тогда **ошибкой гладкости** функции, посчитанной **через нормировку**

дисперсии разности в соседних точках, будем называть величину

$$\mathrm{err}^{\mathrm{std}}_{\mathrm{smooth}}(y_1, \ldots, y_k) = \frac{\mathrm{diff}_{std}}{|\mathrm{diff}_{mean}|}$$

Здесь $\mathrm{diff}_{mean} = \frac{1}{k-1}\sum_{i=2}^{k}(y_i - y_{i-1}), \quad \mathrm{diff}_{std} = \frac{1}{k-1}\sum_{i=2}^{k}(y_i - y_{i-1} - \mathrm{diff}_{mean})^2$

Определение 3.4. Пусть $\{y_i\}_{i=1}^{k}$ - значения нашей метрики на данном пуле запросов для k различных значений набора гиперпараметров. Пусть $window \in \mathbb{Z}, deg \in \mathbb{N}$. Тогда **ошибкой гладкости** функции, посчитанной **с помощью аппроксимации полиномами**, будем называть величину

$$\mathrm{err}^{\mathrm{poly}}_{\mathrm{smooth}}(y_1, \ldots, y_k) = \sum_{i=1}^{k-window+1} \frac{\left(poly_{deg}(y_i, \ldots, y_{i+window-1})_{i+\lfloor\frac{window}{2}\rfloor} - y_{i+\lfloor\frac{window}{2}\rfloor}\right)^2}{k - window}$$

Здесь $poly_{deg}(y_i, \ldots, y_{i+window-1})$ - аппроксимирующий полином степени deg, построенный по window точкам. Т.е. такой полином степени deg, что среднеквадратичное отклонение в точках $\{i, \ldots, i+window-1\}$ минимально.

Возникает ряд проблем с тем, что выбор конкретного способа измерения гладкости для сравнения наших метрик между собой неочевиден. Первая метрика хорошо отображает гладкость в том случае, если функция монотонна (если не учитывать шумовые колебания) - однако же в иных случаях данная метрика может быть плоха из-за нормировки на разность значений в крайних точках.

Вторая метрика ведет себя лучше, однако тоже не является достаточно гибкой и интерпретируемой.

С последней метрикой встает проблема выбора параметров deg и window. Тем не менее эта метрика понятна и действительно способна достаточно хорошо отображать гладкость функции. После перебора разных вариантов выбор был сделан в пользу параметров $deg = 3$, $window = 11$. При этом

стоит учитывать, что гладкость метрики ранжирования измерялась на множествах размера порядка 100 точек.

Видно, что выбор метрики гладкости - отдельная задача, сложность которой заключается в том, что мы пытаемся мерять гладкость дискретно заданной функции. При этом метрика гладкости в работе выбиралась так, чтобы отразить наши представления о «хорошей» функции.

Метрика SoftDCG

Рассмотрим метрику SoftDCG. Данная метрика является попыткой сглаживания метрики DCG. Кратко разберем способ подсчета SoftDCG

Описание метрики

Пусть наш набор документов в поисковом запросе уже упорядочен по скорам. Рассмотрим конкретный документ d_j со скором s_j. Логично предположить, что скор не является точным определением того, на каком месте должен быть документ. Более точно: пусть $s_j - s_{j+1} < \varepsilon, s_{j-1} - s_j > \varepsilon$, тогда, если добавить к оценками ранжирующего алгоритма случайный шум с дисперсией ε, документы (s_j, s_{j+1}) поменяются местами в выдаче более вероятно, чем (s_{j-1}, s_j). Данный факт говорит о близости документов, однако DCG способен учитывать близость документов с точностью до 1 места.

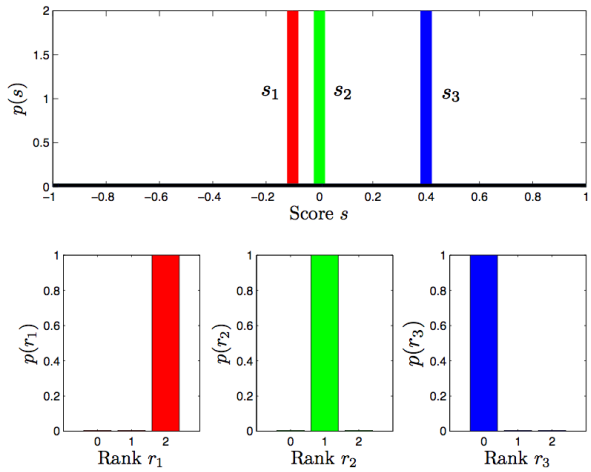


Figure 1: Deterministic scores and ranks: *Three document scores as point (deterministic) values and their corresponding rank distributions. The lowest scoring document s_1 is certain to be ranked in the lowest position 2.*

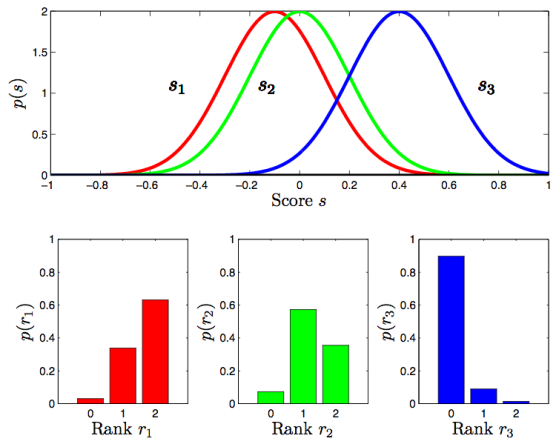


Figure 2: From score to rank distributions: *Smoothed scores for 3 documents and the resulting 3 rank distributions.*

Метрика SoftDCG предполагает следующее: вместо того, чтобы каждому документу сопоставлять конкретное место в поисковой выдаче, можно сопоставить ему нормальное распределение на скорах. Т.е. doc_i имеет скор t_i из распределения $t_i \sim \mathcal{N}(s_i, \sigma^2)$, где σ - гиперпараметр.
В такой модели

$$\pi_{ij} := P(doc_i \text{ выше } doc_j) = P(t_i - t_j > 0) = \int_{-\infty}^{\infty} \mathcal{N}(x|s_i - s_j, 2\sigma^2)dx$$

Из этого выводятся распределения на местах для каждого документа. При этом можно посчитать ожидаемую позицию документа:

$$\bar{pos}(j) = \sum_{i=1, i\neq j}^{n} \pi_{ij}$$

На этом строится подсчет среднего дискаунта каждого документа:

$$\hat{d}(j) = \sum_{i=1}^{n} p_j(i)\text{discount}(i) - \text{средний дискаунт j-го документа}$$

Далее средние дискаунты используются при подсчете SoftDCG:

$$\text{SoftDCG} = \sum_{i=1}^{n} \hat{d}_j \cdot r_j$$

Свойства метрики

В то же самое время есть основания полагать, что данная метрика является не совсем «честной», т.к. средняя позиция документа вычисляется, как мат ожидание числа документов выше. Правильный способ подсчета средней позиции документа делается через введение вероятностного распределения на перестановках документов. Описанные в следующих разделах метрики по разному оценивают среднюю позицию документа.

Метрика NoisedSoftDCG

Рассмотрим другую метрику (назовем ее NoisedSoftDCG). Идея данной метрики берет свое начало в статье про SoftDCG, однако способ подсчета данной метрики несколько иной.

Описание метрики

В SoftDCG каждому документу сопоставлялось некоторое вероятностное распределение на множестве возможных занимаемых позиций. При этом следует отметить, что данное распределение вполне четко задавалось аналитически. Благодаря этому имелась возможность задать такой параметр, как $\hat{d}(j)$ (средний дискаунт), формулой.

В данном же случае при подсчете метрики принимается во внимание лишь тот факт, что при небольшом изменении скоров на случайные величины значение метрики может меняться в силу того, что близкие документы будут переставляться местами.

Алгоритм подсчета NoisedSoftDCG следующий:

(1) Получим скоры ранжирующего алгоритма $s_1, \ldots, s_n$

(2) Сгенерируем случайный шум $\xi_1, \ldots, \xi_n \sim \mathcal{N}(0, \sigma)$

(3) $\hat{s}_1 = s_1 + \xi_1, \ldots, \hat{s}_n = s_n + \xi_n$

(4) Посчитаем значение метрики Value_{DCG}

(5) Повторим шаги (2)-(4) достаточно большое количество раз (T) и вычислим $NoisedDCG = \sum_{i=1}^{T} \text{Value}_{DCG}^{i}$

Свойства метрики

Можно дать некую интуицию по поводу того, почему метрика считается именно так. При достаточно большом T близкие документы будут часто меняться местами, а усреднение DCG при этом даст некий аналог среднего дискаунта. Таким образом ожидается, что при больших T данная метрика будет вести себя хорошо в плане гладкости

Метрика FairSoftDCG

Еще один вариант метрики, идея которой схожа с NoisedDCG. Заметим, что, основываясь на скорах, выданных ранжирующим алгоритмом, можно ввести вероятностное распределение на перестановках всех документов.

Описание метрики

При этом распределение вводится так, что $P(d_i \text{ выше } d_j) \sim \frac{e^{\sigma s_j}}{e^{\sigma s_i}+e^{\sigma s_j}}$ (данное распределение можно ввести единственным образом). Благодаря этому для каждой перестановки на документах $\pi_1, \ldots, \pi_n$ можно посчитать ее вероятность.

Утверждение 6.1. Вероятностное распределение на перестановках однозначно задается вероятностями для всех пар документов и задается формулой:

$$P(\pi_1, \ldots, \pi_n) = \prod_{i=1}^{n-1} \frac{e^{\sigma s_{\pi_i}}}{\sum\limits_{k=j}^{n} e^{\sigma s_{\pi_j}}}$$

Доказательство Рассмотрим следующую модель: для построения поисковой выдачи делается n шагов, на k-м из которых выбирается один из оставшихся $n - k + 1$ документов, причем вероятность выбора каждого документа на данном шаге зависит только от тех документов, которые остались.

При этом можно заметить, что из парных вероятностей следует то, что вероятность выбора конкретного документа i на данном шаге пропорциональна $e^{\sigma s_i}$. Остается только нормировать эти вероятности на 1.

Для этого надо поделить все вероятности на $\sum_{i=k}^{n} e^{\sigma s_{j_i}}$ - т.е. на сумму экспонент скоров оставшихся документов.

□

Теперь перебрав все перестановки документов можно посчитать «честное» мат. ожидание DCG. Собственно это у будет значением FairSoftDCG.

$$FairSoftDCG = \sum_{\pi} P(\pi_1, \ldots, \pi_n) \cdot DCG(\pi_1, \ldots, \pi_n)$$

Свойства метрики

При этом возникает ряд проблем с данной метрикой, т.к. вычислительно данная задача достаточно сложная (всего имеется $n!$ возможных перестановок). В связи с этим был выбран способ подсчета близкого к данной метрике значения - подсчета по топ-k документам:

$$FairSoftDCG = \sum_{\pi} P(\pi_1, \ldots, \pi_k) \cdot DCG(\pi_1, \ldots, \pi_k),$$

где сумма берется по всем возможным размещениям по k из n элементов. k берется таким, чтобы метрику можно было посчитать, т.к. n большое и перебор $n!$ перестановок - долгая операция.

Применение результатов

В данной части будут приведены предполагаемые способы использования искомой метрики. На данный момент идеальной - т.е. гладкой и хорошо аппроксимирующей DCG - метрики не найдено.

Итак. Первый и способ применения более гладкой метрики - это использование ее в качестве инструмента показывающего качество обучения модели. Благодаря большей чувствительности появляется возможность следить за тем, как малые изменения гиперпараметров влияют на качество модели.

Простой пример: благодаря этой метрики можно достаточно хорошо измерять влияние новых элементов ансамбля в моделях градиентного бустинга. Точно так же можно использовать данную метрику для active learning like задач. Благодаря высокой чувствительности можно будет немного дообучать алгоритм на небольших частях датасета и смотреть на изменение качества ранжирования.

Второй и, пожалуй, важный с практической точки зрения способ применения нашей метрики - смешивание нескольких моделей ранжирования. В наши дни в ранжирующих системах крупных компаний используются абсолютно разные по характеру формулы. Эти формулы имеют разный масштаб выдаваемых значений. Каждая из этих формул придумывалась для оптимизации конкретной метрики - причем метрики могут быть кардинально различными по смыслу. При этом хотелось бы уметь смешивать формулы, выданные несколькими различными моделями, в один ансамбль формул, получая таким образом более мощную формулу ранжирования.

Т.к. наша метрика является гладкой и хорошо аппроксимирует DCG, оптимальная комбинация формул согласно этой метрике должна быть в той же окрестности, что и оптимум с точки зрения DCG. В то же самое время естественно ожидать, что при взятии выпуклой комбинации двух формул должна получаться некая выпуклая гладкая кривая для значений метрики.

В качестве своеобразной sanity-check - добавления шума к скорам ранжирующего алгоритма. Здесь имеется ввиду следующее: если добавить к скорам модели шум с неким коэффициентом, это эквивалентно смешиванию двух моделей - нормальной и абсолютно бесполезной. Логично полагать, что оптимум качества в данном случае будет при коэффициенте, с которым берется шум, равном 0 (или близком к 0).

Эксперименты

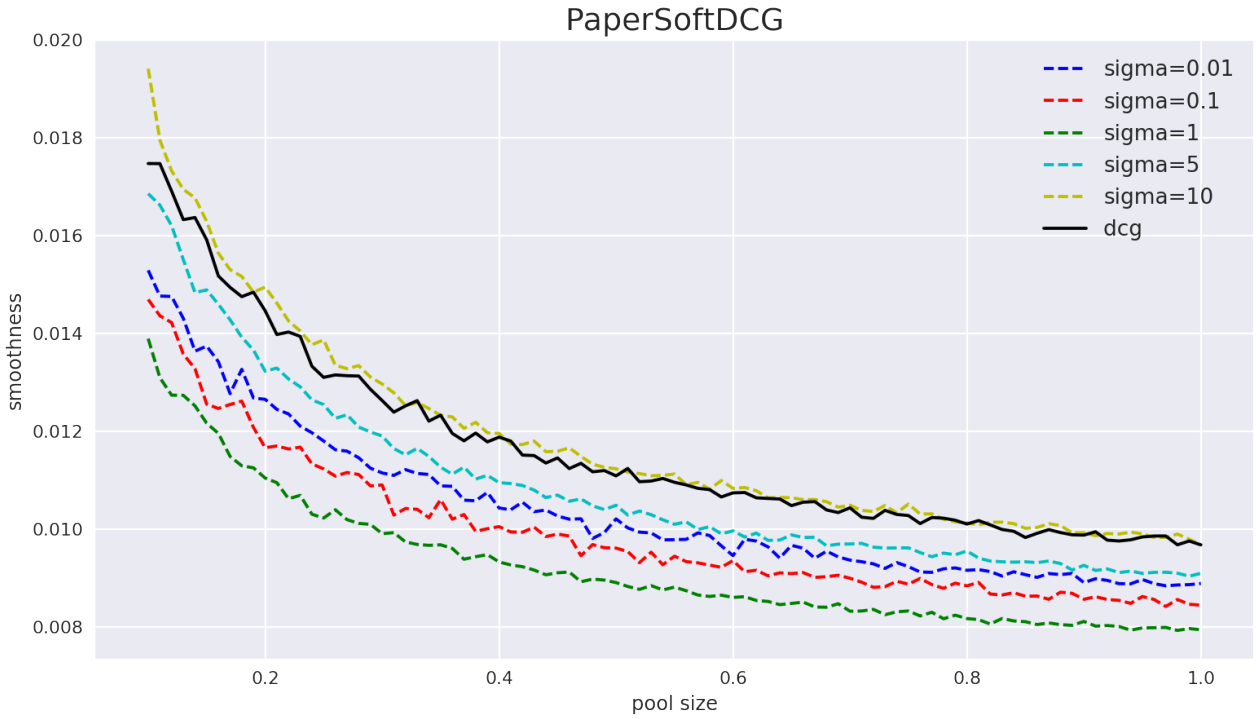
Зависимость гладкости от размера пула запросов

Первый эксперимент является одновременно способом проверить наш метод измерения гладкости на адекватность и способом сравнения между собой рассмотренных ранее метрик ранжирования. Здесь и далее рассуждения будут вестись для метрики DCG, но они точно так же верны для любой метрики из списка SoftDCG, NoisedSoftDCG, FairSoftDCG.

Для начала вспомним, что такое в нашей задаче «размер пула». Размер пула - это количество обрабатываемых поисковых запросов. Для каждого запроса мы считаем значение DCG, а потом производим усреднение данного значения по всем запросам. Все это происходит при каком-то зафиксированном наборе гиперпараметров.

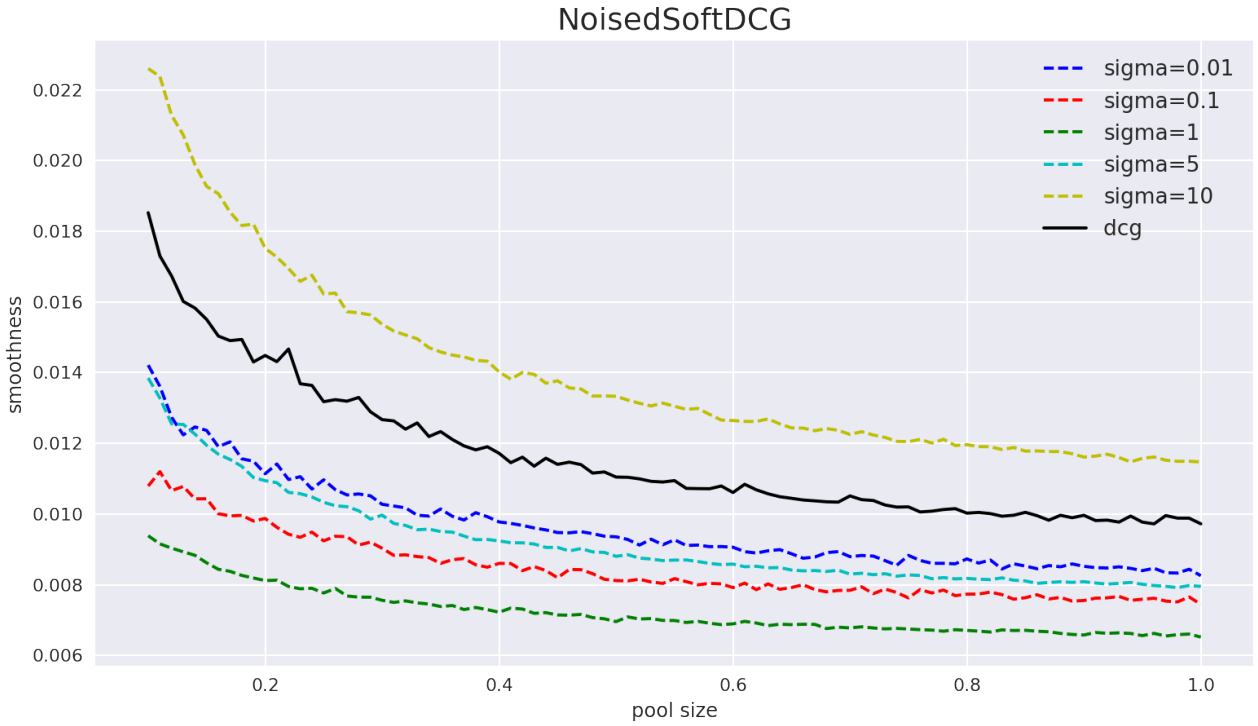
Таким образом для каждого значения набора мы получаем какое-то значение метрики. Например, можно смотреть на значение метрики для ранжирующего алгоритма, который является выпуклой комбинацией двух других алгоритмов, в зависимости от коэффициента, с которым алгоритмы смешиваются. Если рассматривать данный график, на множестве коэффициентов $[0; 1]$, можно посчитать его гладкость. При этом, рост размера пула - т.е. увеличение количества запросов, по которому мы усредняем ответ, должно играть роль сглаживающего фактора.

Данный эксперимент состоял в том, чтобы посмотреть на изменение гладкости с увеличением роста пула. Хочется заметить, что т.к. увеличивать пул мы возможности не имели, делалось эквивалентное действие - его



уменьшение. Ожидалось, что для каждой метрики будет получен график, имеющий обратную пропорциональность - чем больше размер пула, тем меньше ошибка гладкости.

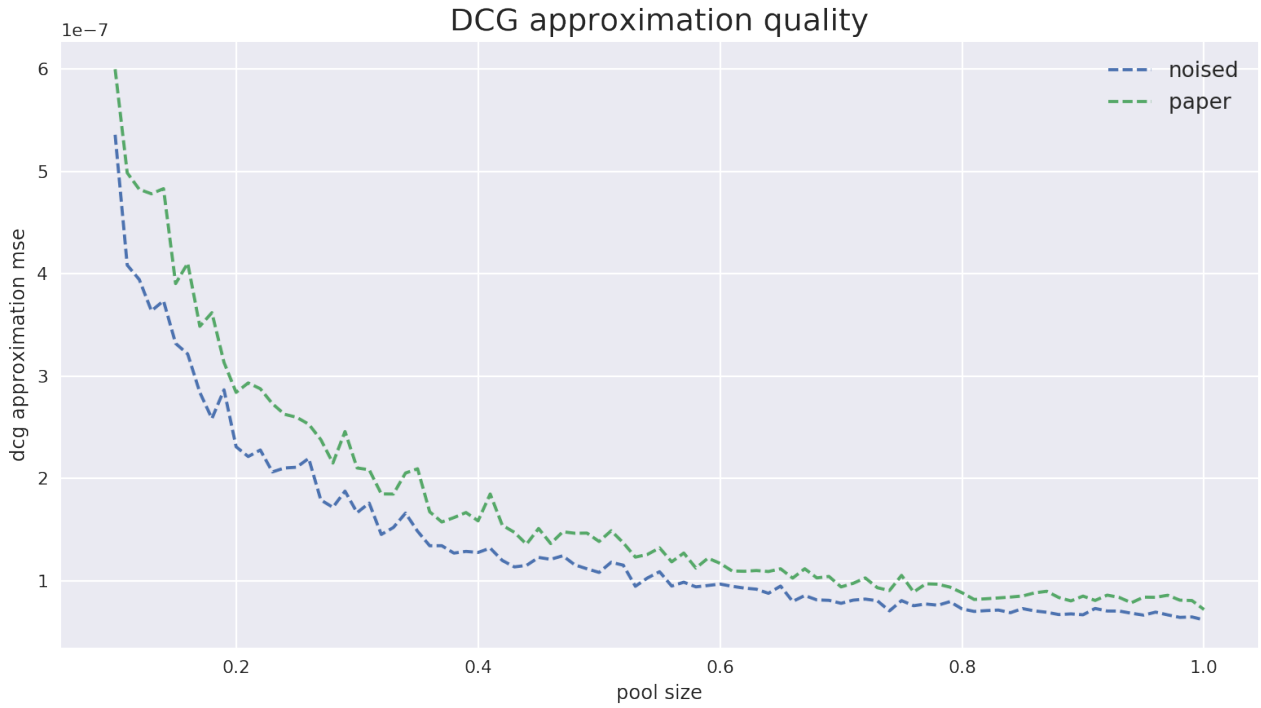
На графиках для сравнения приведена гладкость DCG и гладкость соответствующих метрик. Можно заметить, что при некоторых σ гладкость нашей метрики значительно лучше (по сравнению с DCG). Также видно, что NoisedSoftDCG - более гладкая, чем PaperSoftDCG.



Зависимость качества аппроксимации от размера пула запросов

Аналогично предыдущему эксперименту логично было бы ожидать графики, на которых с увеличением размера пула ошибка аппроксимации DCG уменьшается.

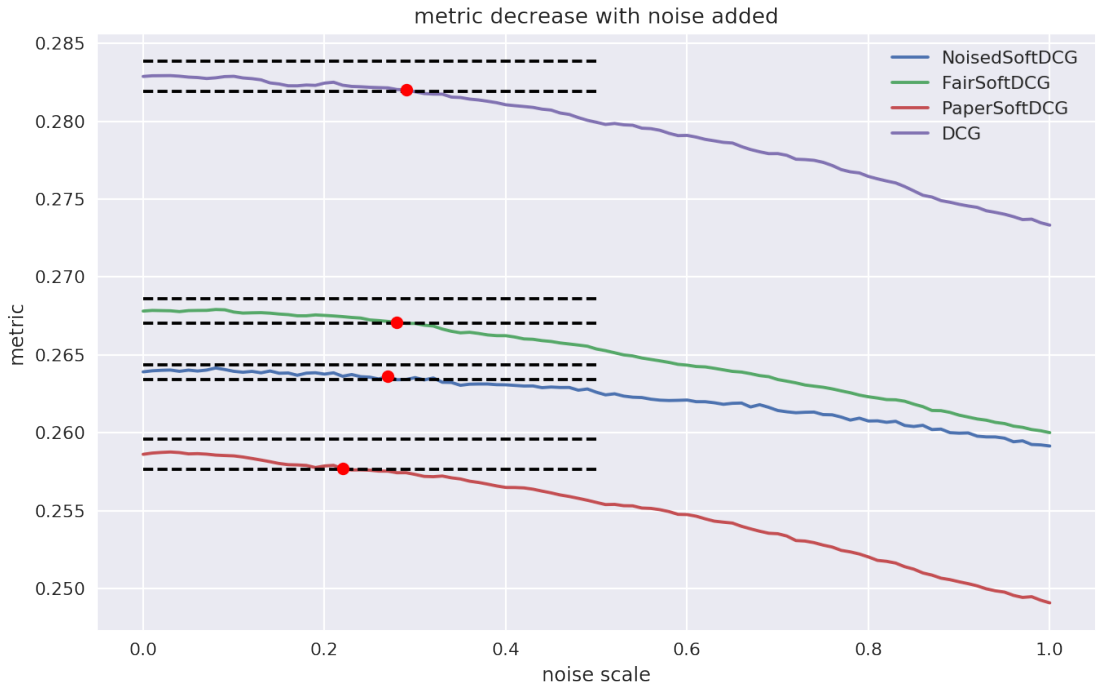
Что же касается аппроксимации DCG, можно заметить, что NoisedSofDCG и здесь ведет себя лучше.



Изменение метрики при добавлении шума

Как упоминалось ранее, первоочередным действием в работе было желание убедиться в том, что метрика гладкости действительно несет ту смысловую нагрузку, которая на нее была возложена. Для этого проводился эксперимент по смешиванию двух ранжирующих формул, одна из которых была настоящей, а вторая - просто сгенерированным случайным шумом.

По оси x отложен коэффициент, с которым прибавляется случайный шум, по оси y - значения метрики. Можно заметить, что наши метрики убывают почти монотонно.

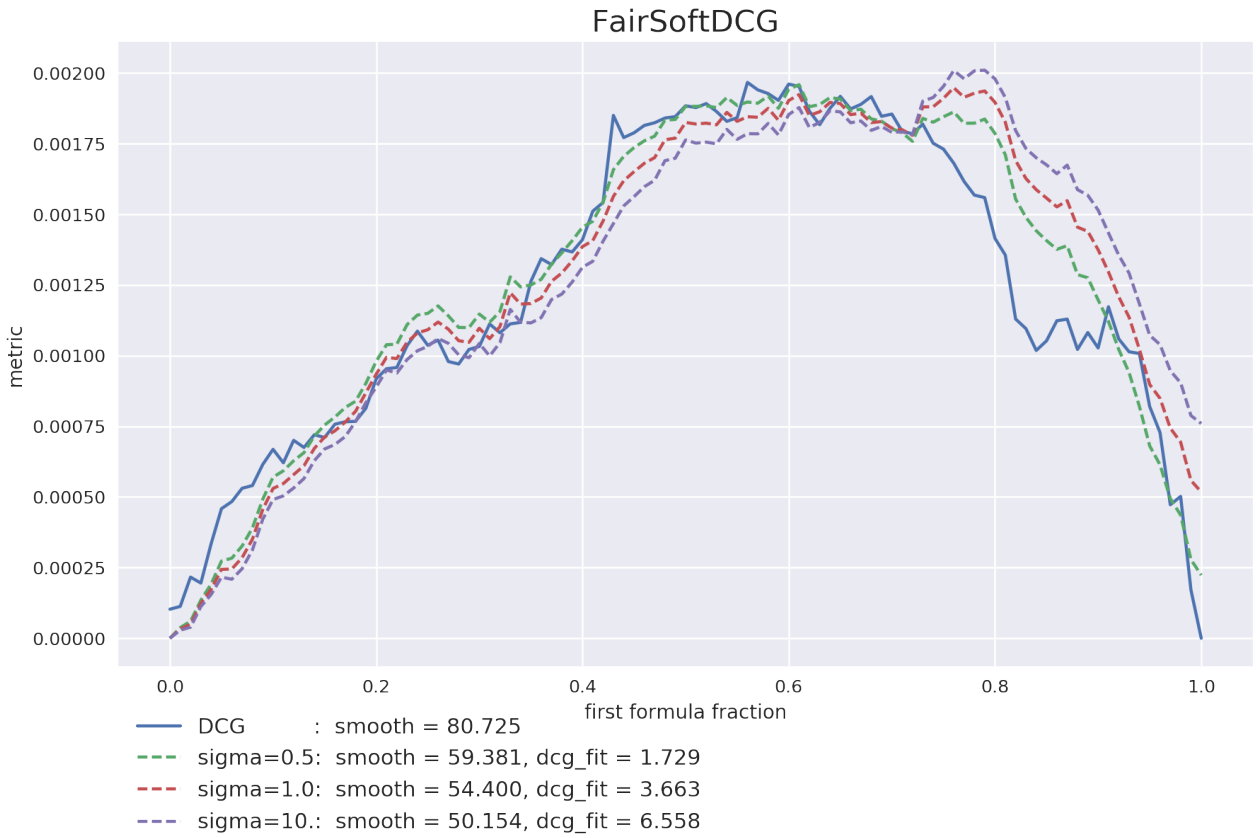


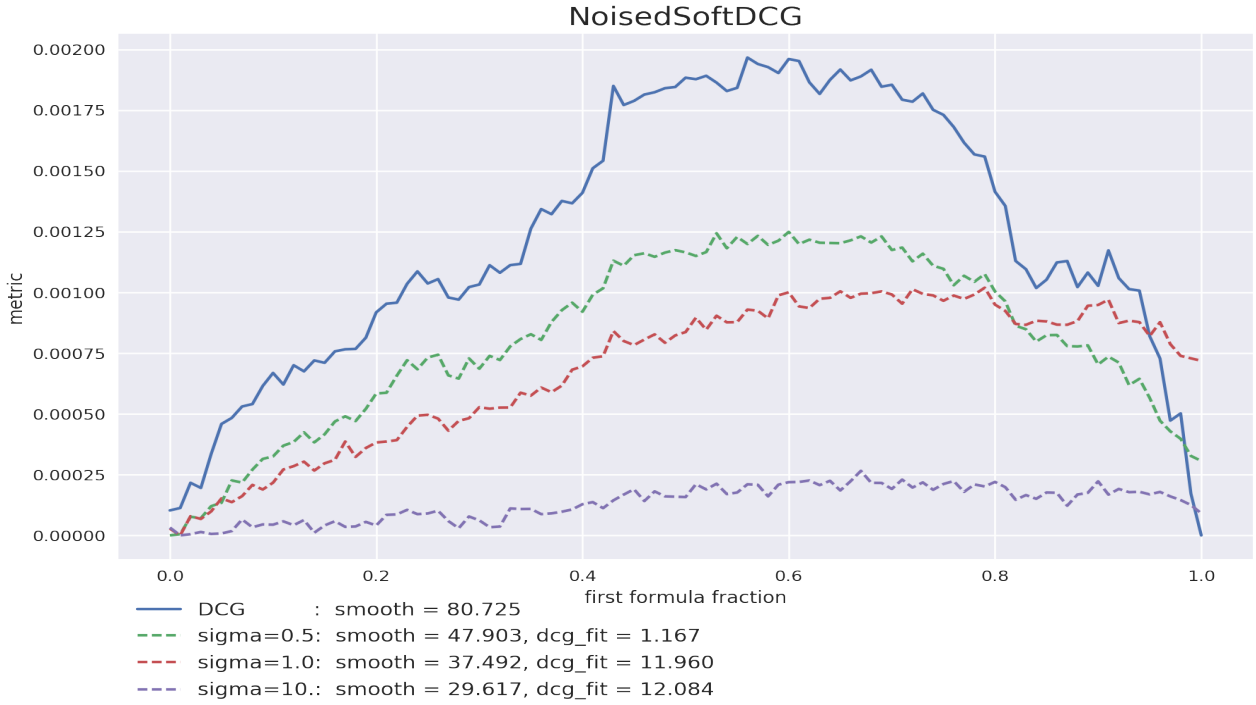
На графике для каждой метрики показана точка, в которой значение метрики выходит из доверительного интервала (ширина доверительного интервала пропорциональна разности значений метрики в крайних точках). Благодаря этому можно судить о чувствительности метрик: лучше всего добавление плохой модели чувствует SoftDCG, хуже всего - FairSoftDCG. При этом все метрики чувствуют плохую модель лучше, чем оригинальная DCG. Данное свойство важно для использования метрики, как способа контролировать переобучение модели.

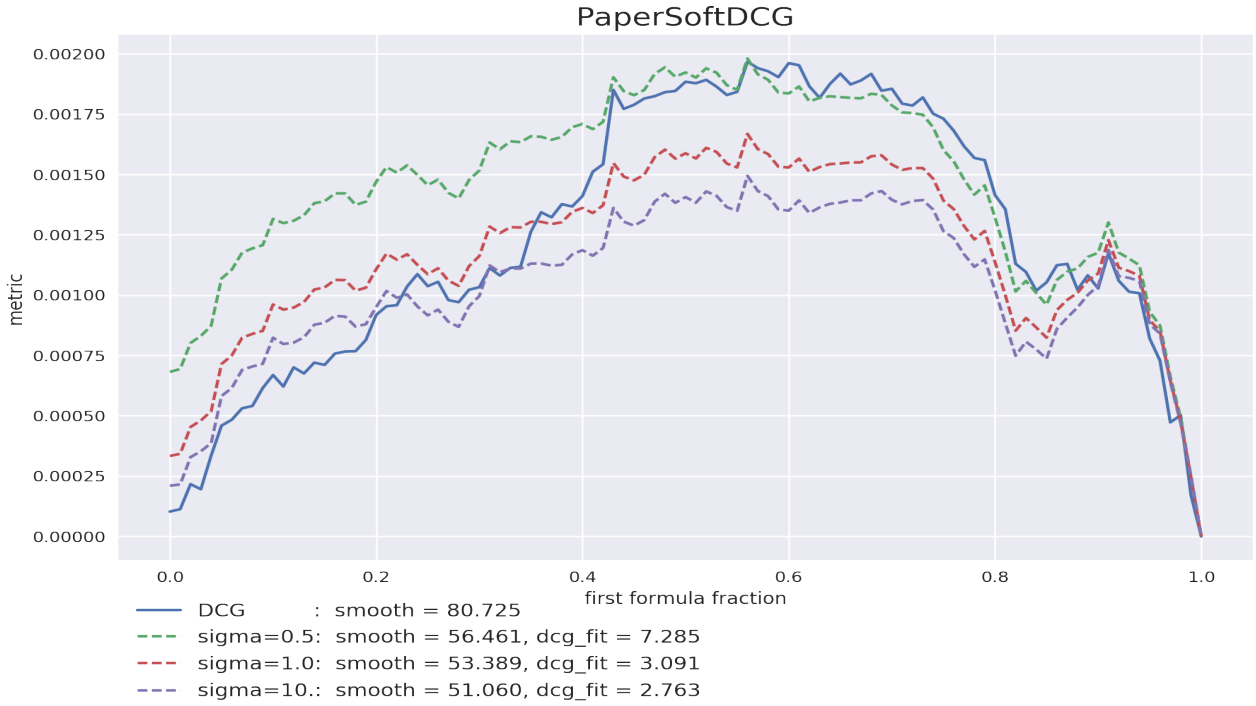
Подобное поведение в данном эксперименте является необходимым условием для класса искомых метрик.

Изменение метрики для комбинации двух формул

Самый важный с точки зрения практического применения эксперимент - эксперимент по смешиванию двух формул ранжирования. Цель эксперимента - проверить, какая из наших метрик лучше отображает реальный оптимум качества при использовании в качестве ранжирующей формулы выпуклую комбинацию двух формул.



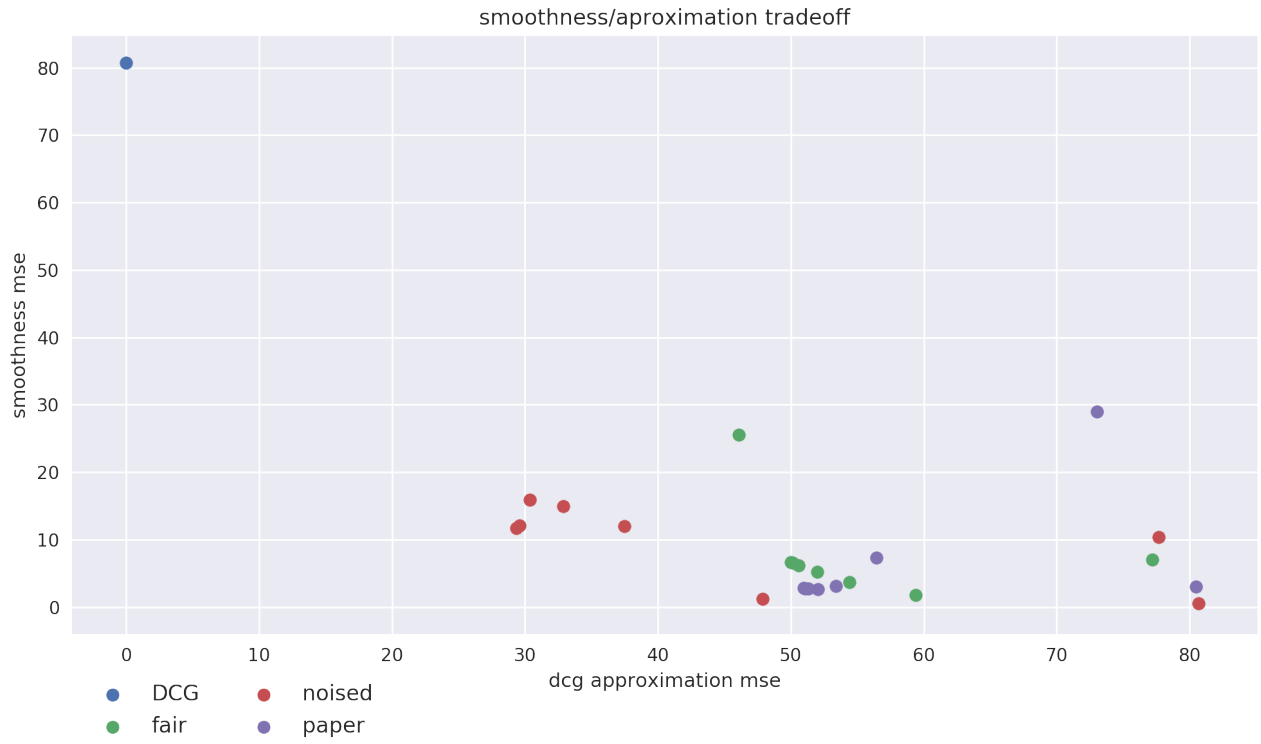

NoisedSoftDCG
metric
first formula fraction
DCG : smooth = 80.725
sigma=0.5: smooth = 47.903, dcg_fit = 1.167
sigma=1.0: smooth = 37.492, dcg_fit = 11.960
sigma=10.: smooth = 29.617, dcg_fit = 12.084


PaperSoftDCG
metric
first formula fraction
DCG : smooth = 80.725
sigma=0.5: smooth = 56.461, dcg_fit = 7.285
sigma=1.0: smooth = 53.389, dcg_fit = 3.091
sigma=10.: smooth = 51.060, dcg_fit = 2.763

Проблема выбора гиперпараметра

Так как главной целью работы было получение гладкой метрики, с помощью которой можно будет удобно и правильно подбирать коэффициенты в смеси ранжирующих формул, остановимся на этом эксперименте.

Как можно было заметить из графиков, приведенных выше, при разных значениях гиперпараметра σ получаются разные качества аппроксимации/гладкости. При этом, логично предположить следующее: чем лучше аппроксимация (т.е. чем ближе σ к 0), тем хуже гладкость кривой. Ниже приведен график в осях, соответствующих данным двум показателям, метрик с разными гиперпараметрами.



Как видно из графика, для каждой метрики в можно варьировать этот гиперпараметр, получая тем самым желаемый компромисс между гладкостью метрики и хорошей аппроксимацией DCG.

Следует отметить, что выбор параметра - неочевидная задача. В итоге, возникает проблема определения σ. Она заключается в том, что необходимо выбирать данный параметр, опираясь, лишь на скоры, выданные каждой из формул. Однако скоры в каждом запросе свои. В добавок функция, которую мы при этом пытаемся минимизировать: $\alpha \cdot approx + \beta \cdot smoothness$, не является гладкой относительно σ.

Данную проблему можно описать следующим образом. Представим себе смесь двух формул ранжирования, где вторая формула совпадает в первой, но выдает значения в 10 раз больше. Очевидно вторая формула более устойчива к добавлению шума. Таким образом, даже если вторая формула хуже, наша метрика может предпочитать отдавать ей больший коэффициент в выпуклой комбинации потому, что выбрана слишком большая σ. При этом, если есть две разные формулы, значения которых нормированы на [0;1], может оказаться, что одна значительно лучше устойчива к шуму, нежели вторая - но при этом нам хотелось бы понимать, что в смеси формул лучше отдавать предпочтение первой.

Также можно рассмотреть графики зависимости ошибок гладкости и аппроксимации от размера пула. Видно, что для каждой метрики данные ошибки сначала убывают с увеличением σ, а потом возрастают. При этом выбор σ для которой они будут оптимальны в применении к конкретным формулам ранжирования неочевиден.

Есть некоторые эвристики по подбору σ, но хорошего решения в ходе исследования данного вопроса найдено не было.

Выводы

В ходе исследования была рассмотрена реализация метрики SoftDCG, идея которой описана в [1]. Также были рассмотрены предложенные автором аналоги этой метрики: NoisedSoftDCG и FairSoftDCG.

При изучении этих метрик были введены такие параметры, как «гладкость» и «аппроксимация DCG». Было проведено исследование касательно изменения этих показателей при изменении различных гиперпараметров. Это дало возможность понять, что представленные метрики в целом являются неким продвижением в нужном направлении - но окончательного решения не дают.

По результатам исследования автор отдает предпочтение метрике NoisedSoftDCG, т.к. в экспериментах она показала себя лучше. Ошибки гладкости и аппроксимации данной метрики меньше, чем у SoftDCG. При этом сложно проводить хорошее сравнение с FairSoftDCG в силу того, что она является вычислительно сложной. Важным свойством NoisedSoftDCG является также то, что при уменьшении размера пула почти в 2 раза мы можем значительно не терять в гладкости и аппроксимации DCG - чего нельзя сказать о SoftDCG.

Важный вывод можно сделать из эксперимента со смешиванием двух формул ранжирования: лучше всего себя в данном случае ведет FairSoftDCG - что совпадает с идеей ее конструкции (это «честный» способ считать SoftDCG). При этом можно заметить, что NoisedDCG показала себя лучше в данном эксперименте при меньших σ - для данной метрики важен параметр T (количество усреднений). При слишком больших σ и малых T шум слишком сильно влияет на метрику, и она приближается к константе.

Метрика	σ	$\text{err}_{\text{smooth}}$	$\text{err}_{\text{approx}}$	$\frac{\text{err}_{\text{smooth}}}{5} + \text{err}_{\text{approx}}$
FairSoftDCG	0.5	59.381	1.729	13.605
FairSoftDCG	1	54.400	3.663	14.543
NoisedDCG	0.5	47.903	1.667	11.248
NoisedDCG	1	37.492	11.960	19.584
SoftDCG	0.5	56.461	7.285	18.577
SoftDCG	1	53.389	3.091	13.769

Была исследована проблема выбора гиперпараметра σ, который соответствует тому, насколько сильно наша метрика стремится раздвинуть между собой документы (фактически - порядок разницы между скорами близких документов).

Список литературы

[1] M. Taylor, J. Guiver, S. Robertson and T. Minka. SoftRank: Optimising Non-Smooth Rank Metrics. Microsoft Research Cambridge, 2016

[2] Christopher J.C. Burges. From RankNet to LambdaRank to LambdaMART. Microsoft Research Technical Report, 2010

[3] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, Hang Li. Learning to Rank: From Pairwise Approach to Listwise Approach. Microsoft Research Technical Report, 2007

Appendix

SoftDCG code

```
def missrank_probs(scores, sigma=1, prob_model='norm'):
    '''Calculates matrix P[ij] of probabilities that
       doc_i will rank above doc_j
    Input:
        scores : array_like, ranking function predictions
        sigma : std
        prob_model : string, one of ['norm', 'exp']
    Output:
        P : 2d array with predicted probabilities
    '''
    if prob_model == 'norm':
        n = scores.shape[0]
        mean = scores.reshape(-1, 1) - scores
        std = np.ones((n,n)) * sigma * 2
```

```
        return norm.cdf(1000000, loc=mean, scale=std) -
            norm.cdf(0, loc=mean, scale=std)
    elif prob_model == 'exp':
        diffs = scores.reshape(-1, 1) - scores
        return np.exp(sigma * diffs)

def rank_distribution(scores, sigma=1, prob_model='norm'):
    '''
    Input:
        scores : array_like, ranking function predictions
    Output:
        P : 2d array with rank distribution for each of N
            documents
    '''
    n = scores.shape[0]
    P = missrank_probs(scores, sigma, prob_model)
    res = np.zeros((n,n))
    res[:,0] = 1
    for i in range(n):
        shifted_distribution = np.roll(res, 1, axis=1)
        shifted_distribution[:, 0] = 0
        prob = P[i].reshape(-1,1)
        prob[i,0] = 0
        res = shifted_distribution * prob + (1 - prob) *
            res
    return res
```

```
def expected_discount(scores, sigma=1, prob_model='norm'):
    P = rank_distribution(scores, sigma, prob_model)
    n = scores.shape[0]
    P /= (1 + np.arange(n))
    return np.sum(P, axis=1)

def soft_dcg(real_marks, scores, sigma=1,
  prob_model='norm'):
    discounts = expected_discount(scores, sigma,
      prob_model)
    return np.sum(real_marks * discounts)
```

NoisedSoftDCG code

```
def noised_soft_dcg(real_marks, scores, n_docs_scored=5,
  sigma=1):
    n = scores.shape[0]
    n_docs_scored = min(n_docs_scored, n)
    discounts = 1. / (np.arange(n_docs_scored) + 1)

    dcg = 0.
    n_generations = 1000
    for it in range(n_generations):
        noised_scores = scores +
          np.random.normal(scale=sigma, size=n)
```

```
        docs_permutation =
            np.argsort(noised_scores)[::-1][:n_docs_scored]
        cur_dcg = np.sum(real_marks[docs_permutation] *
            discounts)
        dcg += cur_dcg

    return dcg / n_generations
```

FairSoftDCG code

```
def fair_soft_dcg(real_marks, scores, n_docs_scored,
    sigma=1, sort_by_real_scores=False):
    n = scores.shape[0]
    if n_docs_scored > 2:
        n = min(n, 7)
    n_docs_scored = min(n_docs_scored, n)
    discounts = 1. / (np.arange(n_docs_scored) + 1)

    dcg = 0.
    real_marks = np.array(real_marks)

    if sort_by_real_scores:
        ind = np.argsort(real_marks)
    else:
        ind = np.argsort(scores)
    ind = ind[::-1][:n]
```

```
real_marks = real_marks[ind]
scores = scores[ind]
scores /= sigma
scores -= scores.max()

# exponentiation problem check
if scores.min() < -20:
    return -1

scores = np.exp(scores)
Z = np.sum(scores)

for prm in permutations(range(n), n_docs_scored):
    prob = 1.
    perm = list(prm)
    sorted_scores = scores[perm]
    z = Z
    for i in range(n_docs_scored):
        prob *= sorted_scores[i] / z
        z -= sorted_scores[i]

    dcg += prob * np.sum(real_marks[perm] * discounts)

return dcg
```

SmoothnessError code

```
def compute_smoothness_via_polynom(metric_values,
    approx_degree=1, window_size=1):
    '''
    computes MSE with approximation polynom
    Returns:
        MSE
    '''
    metric_values = np.array(metric_values)
    n = metric_values.shape[0]
    mse = 0.
    for i in range(n - window_size):
        x = np.arange(window_size, dtype=int) + i
        y = metric_values[x]
        poly = np.poly1d(np.polyfit(x, y, approx_degree))
        mse += (poly[(window_size + 1) // 2] -
            y[(window_size + 1) // 2]) ** 2
    return mse / (n - window_size)
```

ApproximationError code

```
def compute_dcg_deviation(dcg_values, metric_values):
    '''
    computes MSE between dcg and (a * metric + b) dcg
        approximation
```

```
    Returns:
        MSE
    '''
    metric_values = np.array(metric_values)
    dcg_values = np.array(dcg_values)
    metric_values *= np.mean(dcg_values) /
       np.mean(metric_values)
    p = np.poly1d(np.polyfit(metric_values, dcg_values,
       1))
    return ((dcg_values - p(metric_values)) ** 2).mean()
```