KARPOV.COURSES >>>
КОНСПЕКТ

> Конспект > 3 урок > Функции потерь и базовые подходы к обучению моделей ранжирования

> Оглавление

> Три подхода к обучению моделей ранжирования

1. Pointwise (поточечный)

Функция ошибки по конкретному объекту (в пару к запросу).

$$\sum_{q,j} l(f(\boldsymbol{x}_j^q), r_j^q) \to \min$$

Присутствует зависимость $\boldsymbol{x}$ от j-го документа и запроса q

Модель — функция от $\boldsymbol{x}$

Функция потерь рассчитывается между результатом вычисления $f(\boldsymbol{x}_j^q)$ и мерой релевантности r_j^q

2. Pairwise (попарный)

Функция ошибки по паре объектов (в пару к запросу).

$$\sum_{q} \sum_{i,j:\, r_i^q > r_j^q} l(f(\boldsymbol{x}_i^q) - f(\boldsymbol{x}_j^q)) \to \min$$

3. Listwise (списочный)

Функция ошибки на всём списке документов (для конкретного запроса).

$$l(\{f(\boldsymbol{x}_j^q)\}_{j=1}^{m_q},\ \{r_j^q\}_{j=1}^{m_q}) \to \min$$

m_q штук документов в группе для запроса q

> Pointwise-подход

Примеры

- Косинусное расстояние между эмбеддингами пары документа и запроса;
- Логистическая регрессия над признаками запроса и документа (с добавлением интеракций признаков);
- MLP;
- Другие индексы (BM25 и аналоги).

Логистическая регрессия

Представим, что мы ищем релевантные текстовые документы в ответ на такой же текстовый запрос.

Можно, к примеру, нагенерить разных фичей, например:

- Количество общих слов;
- Схожесть темы, полученная, например, с помощью тематического моделирования или LDA (латентного размещения Дирихле).

Поверх этого строим логистическую регрессию и обучаем её на задачу классификации, где положительный ответ — документ релевантен запросу, и наш таргет равен единице.

В случае простого классификатора никак не учитываем специфику задачи ранжирования, и в режиме инференса просто сравниваем логиты, или предсказанные моделью скоры, затем упорядочиваем по ним.

Индекс BM25 (Best matching)

BM25 — функция ранжирования, используемая в поисковых системах для оценки релевантности документов относительно некоторого запроса.

Разработана в 80-х годах прошлого века, однако используется в большинстве систем и по сей день как один из факторов ранжирования.

Существуют и более продвинутые варианты — например, BM25F, которая заточена специально под работу с интернет-страницами, с разными полями данных с этих страниц.

BM25 — это по сути улучшенный вариант TF-IDF.

BM25 — функция поверх "мешков слов", которая помогает отранжировать набор документов на основе анализа частотности появлений общих слов между запросом и документом. Эта функция не учитывает расположение в тексте, для неё все слова одинаковы с точки зрения применимости.

Формула оценки релевантности

Запишем формулу для оценки релевантности некоторого документа D запросу Q:

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

$f(q_i, D)$ — частота слова Q_i в документе D (запрос Q_i содержит в себе слова $Q_1, ..., Q_n$) $\rightarrow TF$ из $TF - IDF$.

N — количество уникальных слов в запросе (бежим по каждому из них и считаем значение некоторого произведения).

TF

Так как BM25 — аналог TF-IDF, то TF представлено дробью, в которой k_1 и b — коэффициенты. На них мы закроем глаза: они обычно зафиксированы для конкретной системы и формулы и могут подбираться исходя из заданной метрики.

Основная часть — это функция f от Q_i и D, тот самый счётчик, т.е. сколько раз слово q_i встречается в документе D.

В знаменателе делаем нормировку с учётом длины документа — это норма D, т.е. количество слов в документе, делённое на среднюю длину документа в коллекции документов, в данном случае $avgdl$.

Эта дробь тем больше, чем чаще слово встречается в документе. То есть, простыми словами, если вы ищете что-то про туризм, и в каком-то коротком тексте это слово встречается 50 раз, то это означает, что документ с высокой вероятностью релевантен вашему запросу.

IDF

Теперь перейдём к IDF — отвечает за то, как часто слово само по себе встречается в корпусе.

Классическая формула для IDF из вычисления TF-IDF:

$IDF = \log \frac{N}{n(q_i)}$, где

N — общее количество документов (размер нашей базы)

$n(q_i)$ — количество документов, содержащих слово Q_i

Сглаженный вариант IDF:

$$IDF(q_i) = \ln(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1)$$

Чем реже встречается слово, тем выше ценность каждого попадания этого слова в документ. Иными словами, мы "штрафуем" очень частые слова, снижая их вес.

Вернёмся к примеру с туризмом: если у нас все документы про туризм, то само попадание слова "туризм" в текст ничего не значит, и IDF будет крайне мал. Если же у нас случайная выборка документов, то это слово будет "маячком", который выделяет именно этот документ из всего множества.

Если проанализировать формулу IDF, то можно обнаружить, что выражение может принимать отрицательные значения, если слово встречается очень часто, более чем в половине документов. Поэтому для любых двух почти идентичных документов один из них, который содержит такое слово, может быть оценён ниже, чем тот, в котором его нет. А само это слово для нас ничего не значит — это просто некоторое частотное слово, например, *предлог* или *союз*.

Это частая ситуация, в которую мы не хотим попадать, поэтому на практике формулу IDF корректируют различными способами.

Например, просто не учитывают отрицательные элементы суммы. Это полный аналог исключению частотных стоп-слов из предложений.

Недостатки BM25

- Значение отрицательно, если производится расчёт для слова, входящего более чем в половину документов (частотные слова, stop-слова);
- Функция сконструирована вручную и ничего не обучается (два параметра k_1 и b можно перебрать).

Достоинства BM25

- Всё еще очень хороший и невероятно быстрый способ фильтрации кандидатов из огромного корпуса документов;
- Формула максимально проста.

> Pairwise-подход

На практике попарные подходы работают лучше, чем точечные, потому что прогнозирование относительного порядка ближе к самой природе задачи ранжирования, чем прогнозирование метки класса в случае классификации или оценки релевантности в случае регрессии.

RankNet

RankNet — один из первых подходов к ранжированию с использованием ML.

В поточечном подходе некоторая функция $f(x)$, которая порождает скаляр, т.е. одно вещественное число, по которому упорядочиваем документы, или производим ранжирование. Однако при обучении мы не учитываем специфику задачи ранжирования с точки зрения корректного упорядочивания объектов.

Идея: обучить $f(x)$, которая всё так же выдаёт скаляр, но ранжирование по нему будет связано с задачей ранжирования.

В качестве модели может использоваться любой алгоритм машинного обучения, однако для простоты, как и авторы оригинального подхода, мы будем рассматривать подход с маленькой нейросетью.

Для тренировки модели нам необходимо разложить все данные по запросам, т.е. партиционировать.

Для конкретного запроса можем выбрать два документа с разными лейблами (разной релевантностью).

Допустим, документ A отмечен как "идеальное соответствие запросу", а документ B — "очень плохое соответствие запросу". Тогда мы можем обозначить это следующим образом:

$A \triangleright B$ — документ A должен быть отранжирован выше документа B

$P(A \triangleright B)$ — вероятность того, что документ A должен быть отранжирован выше, чем B, т.е. чем ближе значение к 1, тем мы более уверены в том, что документ A релевантнее нашему запросу, чем документ B.

Нейросеть — это такая функция f, которая некоторый вектор признаков R размера d отображает в скаляр (некоторую условную релевантность), т.е. эта часть такая же, как и в поточечном подходе:

$f : R^d \mapsto R$

Однако для двух документов A и B, описываемых векторами x_1 и x_2, мы можем оценить, что релевантность первого документа нашему запросу выше, чем второго:

$f(x_1) > f(x_2)$

В качестве функции для оптимизации мы берём классическую функцию потерь: кросс-энтропию C:

$C_{ij} \equiv C(o_{ij}) = -\overline{P}_{ij} \log P_{ij} - (1 - \overline{P}_{ij}) \log(1 - P_{ij})$, где

P_{ij} — предсказание модели (вероятность того, что документ i находится в выдаче выше, чем документ с индексом j)

$\overline{P}_{ij}$ — целевая метка (*таргет*)

Таргет равен 1, если i-й документ действительно имеет релевантность выше, чем j-й, и 0, если ситуация зеркальна, т.е. на самом деле j-й объект должен быть отранжирован выше.

При оптимизации C распределение, порождаемое нашей моделью f, становится максимально близким к эмпирическому распределению вероятностей, которое получается на основе оценок релевантности в датасете.

При попарном подходе и такой формализации обучения ранжированию эта задача может быть сформулирована как классификация пар объектов на две категории: правильно ранжированные и неправильно ранжированные.

Таким образом, имея пару документов, мы пытаемся выяснить оптимальный порядок для этой пары и сравнить его с реальными данными, т.е. с разметкой.

Функция потерь для RankNet и сама постановка задачи при обучении направлены на минимизацию количества инверсий, или перестановок (неправильный порядок пары результатов, т.е. когда *ставим результат с более низкой истинной оценкой выше результата с более высокой оценкой* в ранжированном списке), в итоговой выдаче.

o_i — предсказание нашего алгоритма для одного объекта (*логит* или *скор*):

$o_i \equiv f(x_i)$,

$o_i \in (-\infty, +\infty)$ — вещественное число, не вероятность

Сделав предсказание для двух объектов, i-го и j-го документов, можем вычесть значение одного из другого и обозначить это как o_{ij}:

$o_{ij} \equiv f(x_i) - f(x_j)$

Это можно интерпретировать как разницу в предсказанных оценках релевантности. Если она положительна, то, по мнению модели, первый документ более релевантен, если отрицательна — менее релевантен.

Для превращения этого в вероятность, т.е. нормирования в интервал $[0, 1]$, мы можем воспользоваться обычной логистической функцией. Разность логитов будем использовать как степень для числа e:

$P_{ij} \equiv \dfrac{e^{o_{ij}}}{1 + e^{o_{ij}}}$ — функция отображения предсказания (логита) в вероятность.

Если разность логитов очень мала, т.е. сильно меньше нуля, то в числителе мы получаем число около 0, а в знаменателе сумму, которая стремится к 1. Соответственно, значение отношения будет около 0.

То есть, если разность отрицательна, то, по мнению модели, второй документ **НЕ** менее релевантен, чем первый, а значит наш таргет $\overline{P}_{ij} = 0$.

Если сумма положительна, то дробь стремится к 1, т.е. к высокой вероятности того, что первый документ выше второго в ранжировании.

Тогда функцию потерь, или функцию стоимости (cost function) можно переписать следующим образом:

$C_{ij} = -\overline{P}_{ij} o_{ij} + \log(1 + e^{o_{ij}})$

Плюсы:

- Линейная асимптотика $\rightarrow$ более робастна при шумных метках.
- При таргете 0.5 симметрична, позволяет тренироваться на объектах одного ранга.

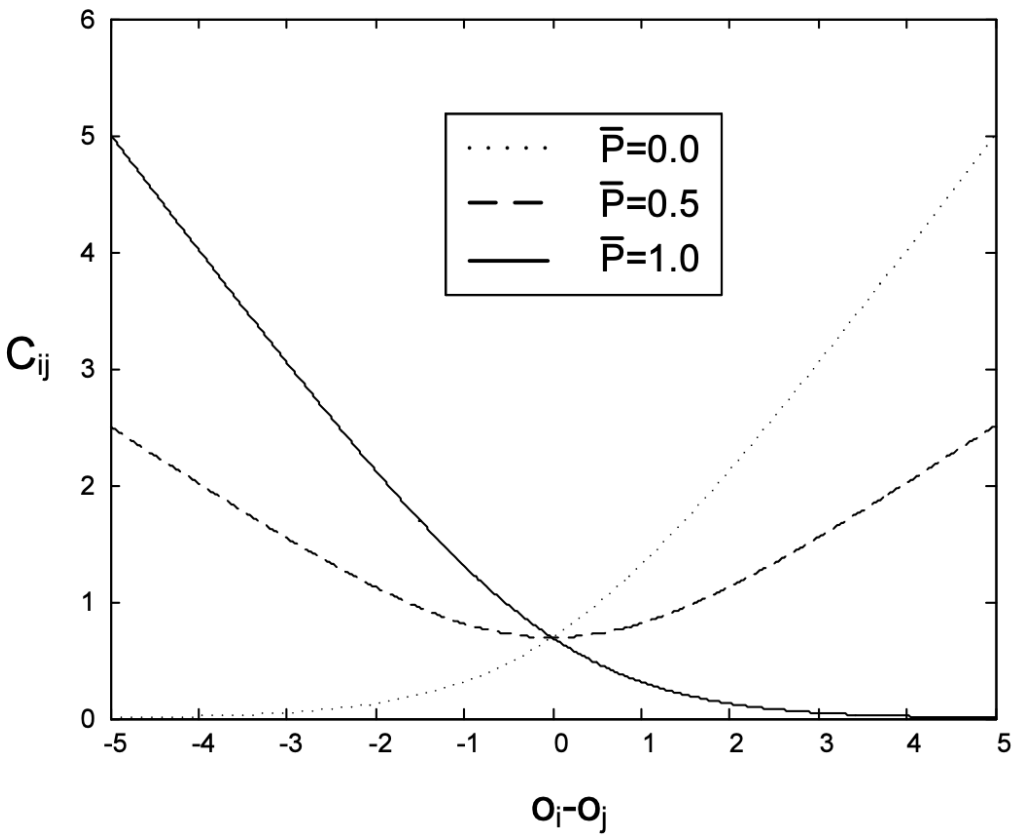
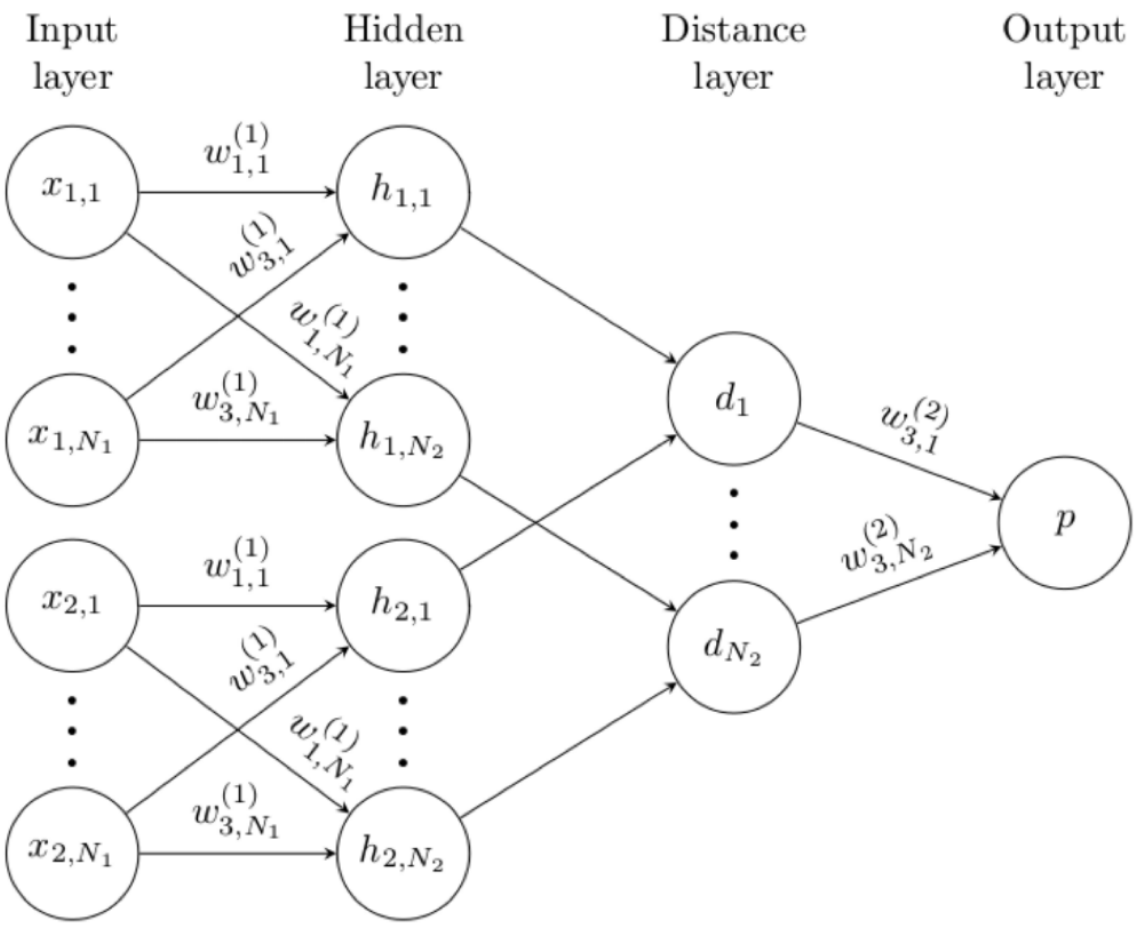


График значений функции ошибки для всех рассмотренных случаев целевого значения (таргета)

Видно, что для бинарных оценок 0 или 1 функция симметрична с точностью до знака.

В случае одинаковой релевантности, т.е. когда таргет равен 0.5, функция симметрична относительно 0, и её оптимум находится в 0. То есть, когда логиты для двух документов совпадают и их разность равна 0, каждый из них равновероятно может быть выше другого.

Аналогия с сиамскими нейронными сетями



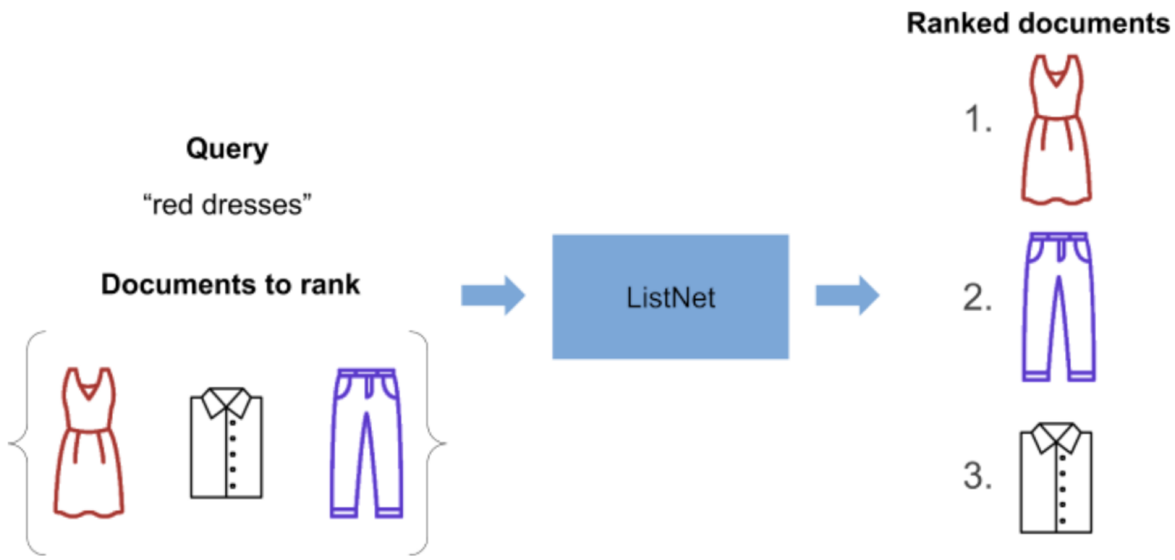
- Также две "ветки" с одинаковыми (*shared*) весами.
- Функцию дистанции выполняет вычитание скалярных величин (не векторных).

> Listwise-подход

В попарном подходе мы говорили о том, что расчёт функции потерь зависит от двух объектов.

Однако в listwise, как следует из названия, мы должны использовать функцию потерь, которая рассчитывается на всём множестве релевантных запросу документов.

Пример



Запрос: "red dresses" (красные платья), дано три документа для ранжирования

Предположим, что мы использовали какую-то базовую предварительную модель, или кандидатную модель, для фильтрации мусора и получили всего три документа.

Эти объекты подаются в алгоритм, который назван авторами подхода как ListNet, после чего мы получаем отранжированный список объектов, где на первом месте стоит самый релевантный документ — в нашем случае это красное платье, т.е. идеальный матч с запросом, а далее идут остальные объекты в каком-то порядке.

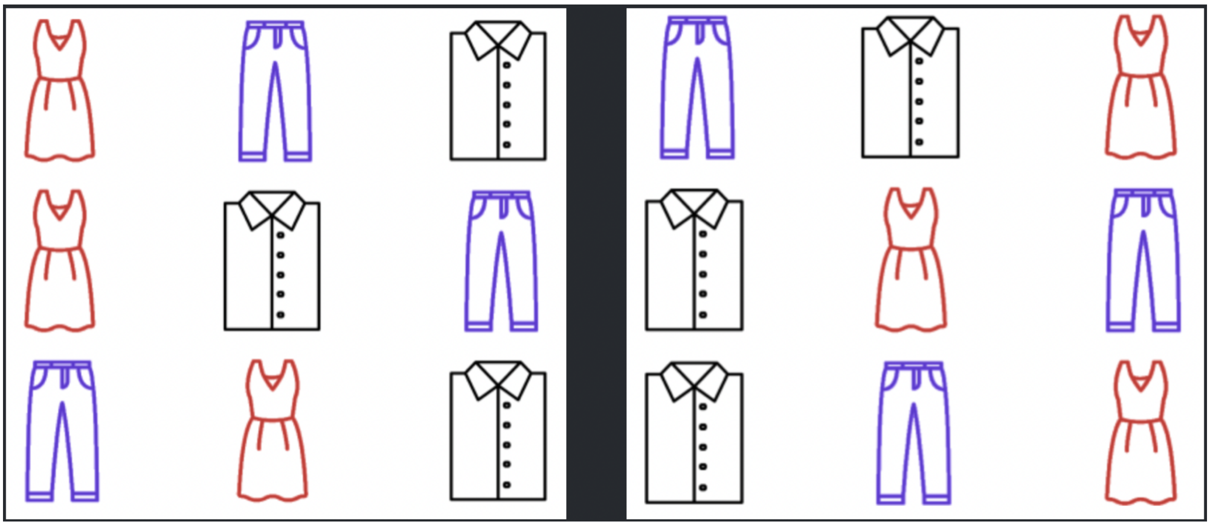
Алгоритм производит некоторые скоры, по которым упорядочивается выдача. Эти скоры необходимо приводить к вероятностям релевантности документа запросу.

То есть, в отличие от попарного подхода, когда мы определяли вероятность того, что некоторый документ выше другого в выдаче или, другими словами, релевантнее, в данном случае мы возвращаемся к простой схеме оценки релевантности.

Набор вероятностей представляет собой распределение вероятностей, которое мы и хотим сделать похожим на наше истинное распределение. Истинное распределение мы будем пытаться получить из разметки.

Здесь полагается, что разметка релевантности может быть любой — и бинарной, и многоуровневой.

Существует некоторая неопределённость в предсказании отранжированных списков. То есть, как минимум, может быть шумная разметка, когда один разметчик решил, что релевантность документа нашему запросу равна тройке, а другой — что она равна четвёрке. На самом деле мы не знаем, как именно отранжировать список документов наилучшим образом. Поэтому делается предположение, что любая перестановка документов возможна, но при этом разные перестановки могут иметь разную вероятность, вычисленную на основе некоторой функции ранжирования.



Пример всех возможных перестановок из трёх объектов: в каких-то сначала идёт наше красное платье, в каких-то рубашка и т.д.

Можем говорить, что представлено полное множество перестановок Ω_n, указывая размер множества объектов, на которых рассчитываются перестановки n.

Каждая перестановка π характеризуется полным указанием, какой объект стоит на первой, на второй и так далее до позиции n.

$\pi = \langle \pi(1), \pi(2), ..., \pi(n) \rangle$

Каждое π_i указывает на конкретный объект в перестановке.

$$P_s(\pi) = \prod_{j=1}^{n} \frac{\phi(s_{\pi(j)})}{\sum_{k=j}^{n} \phi(s_{\pi(k)})}$$ — вероятность возникновения такой перестановки

И в числителе, и в знаменателе к скору, или к логиту, j-го объекта конкретной перестановки π_i применяется функция преобразования скоров.

К этой функции указываются следующие требования:

- Возрастающая;
- Строго положительная.

То есть, чем больше логит, тем выше значение этой функции, при этом ни при каких обстоятельствах она не может стать отрицательной (*иначе бы мы могли получать отрицательные вероятности, чего быть не может*).

Под эти требования подходит много функций, но самая распространенная — экспонента, то есть возведение е в степень логита с индексом π_j.

Рассмотрим знаменатель: здесь сумма от j-го до n-го (последнего) объекта, суммируем мы в точности те же значения, что и в числителе — некоего рода нормализация.

Смотрим, какую долю от суммы всех скоров составляет наш текущий j-й объект.

Вернёмся к примеру с тремя видами одежды.

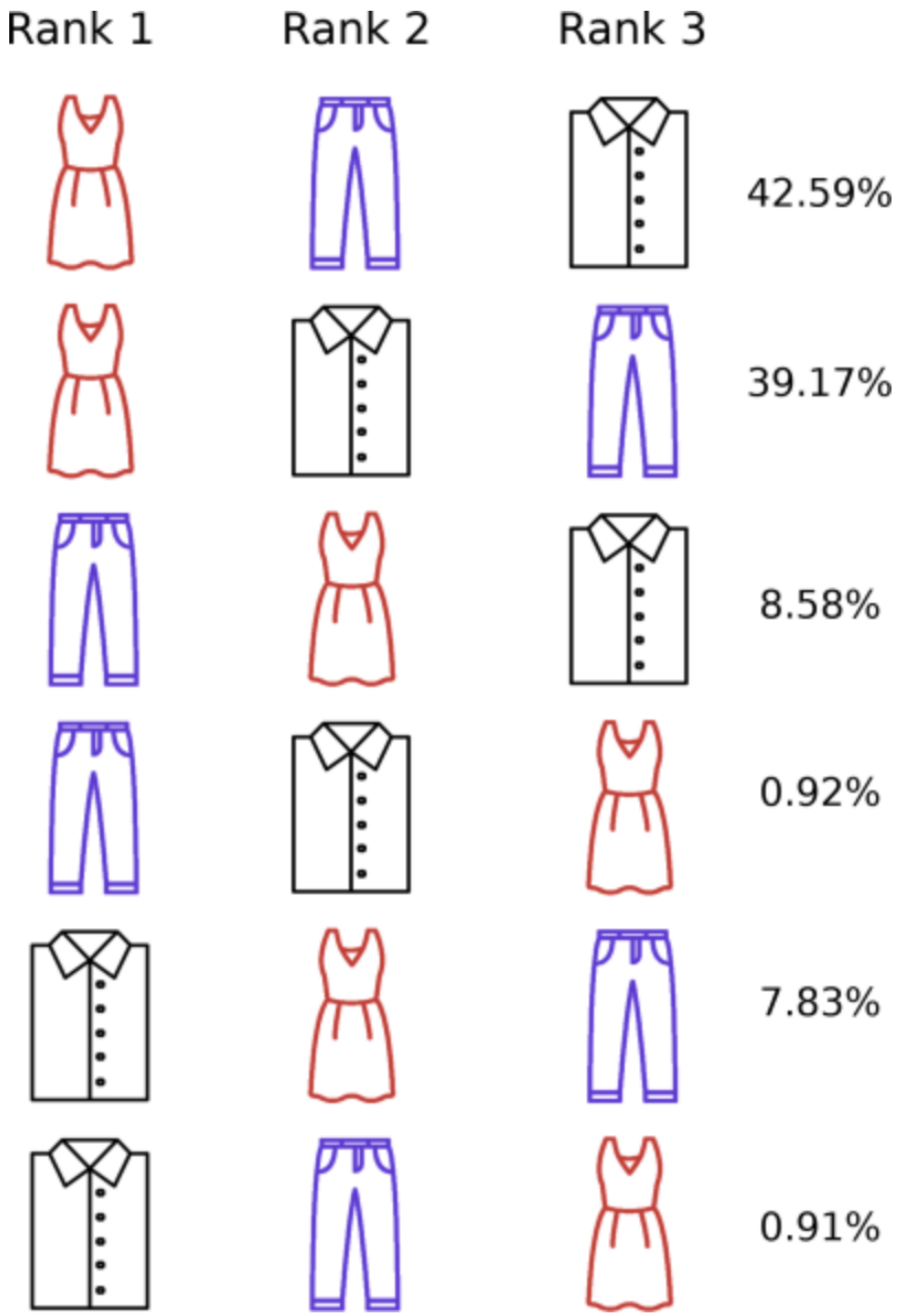
Перестановка вида "платье, штаны, рубашка" со скорами

Допустим, есть перестановка вида "`платье`, `штаны`, `рубашка`", при этом скоры, которые производит наша модель, указаны на картинке под каждым объектом.

Оценим вероятность такой перестановки:

$$P_s(\pi) = \prod_{j=1}^{n} \frac{\phi(s_{\pi(j)})}{\sum_{k=j}^{n} \phi(s_{\pi(k)})} = \frac{\phi(s_1)}{\phi(s_1) + \phi(s_2) + \phi(s_3)} \cdot \frac{\phi(s_2)}{\phi(s_2) + \phi(s_3)} \cdot \frac{\phi(s_3)}{\phi(s_3)} = \frac{e^{s_{dress}}}{e^{s_{dress}} + e^{s_{pants}} + e^{s_{shirt}}} \cdot \frac{e^{s_{pants}}}{e^{s_{pants}} + e^{s_{shirt}}} \cdot \frac{e^{s_{shirt}}}{e^{s_{shirt}}} = 0.3917\ (39.17\%)$$

Эти же расчёты можно применить ко всем возможным перестановкам документов и таким образом вычислить вероятности возникновения таких комбинаций.



Сумма всех вероятностей, т.е. возникновения всех перестановок, равна 1

Выводы для метода:

- Наибольшая вероятность у перестановки, в которой объекты отсортированы в порядке убывания.
- Наименьшая вероятность у перестановки, в которой объекты отсортированы в порядке возрастания.
- Количество перестановок равно $n!$ (много).

TopOneProbability — вероятность того, что объект будет находиться на первом месте в отранжированном списке:

$P_s(j) = \sum_{\pi(1)=j,\ \pi\in\Omega_n} P_s(\pi)$ — сумма вероятностей всех таких перестановок из множества Ω_n, в которых первый и есть j-й объект.

Можно заметить, что для получения для всех n объектов этой вероятности (*TopOneProbability)* всё ещё необходимо считать все перестановки, т.е. никакого выигрыша мы не получаем.

С точки зрения математики эту вероятность можно найти куда более простым путем:

$P_s(j) = \frac{\phi(s_j)}{\sum_{k=1}^{n}\phi(s_k)}$ — вероятность того, что j-й объект будет первым в списке, равна отношению нашей экспоненциальной функции от предсказанного логита к сумме всех преобразованных логитов.

Такая операция называется SoftMax:

- Преобразует множество вещественных чисел в величины от 0 до 1.
- Сумма всех элементов множества будет равняться 1 (суммарной вероятности всех событий).

Благодаря SoftMax не нужно считать все перестановки — можно получить скоры и преобразовать их в TopOneProbability.

Для обучения нашего алгоритма, для расчёта градиентов и градиентного спуска по весам, можно использовать любую функцию потерь, которая оптимизирует расстояние между двумя распределениями вероятностей.

Это может быть классическая кросс-энтропия:

$$L(y^{(i)}, z^{(i)}) = -\sum_{j=1}^{n} P_{y^{(i)}}(j)\log(P_{z^{(i)}}(j)),$$

где на вход передаются листы реальных меток y для i-го запроса, а также предсказанные скоры z, и где мы суммируем по всем n документам в выдаче.

Можно использовать что-то более специфичное, например дивергенцию Кульбака-Лейблера, которая является несимметричной мерой удалённости друг от друга двух вероятностных распределений:

$$D_{KL}(P||Q) = -\sum_{x \in X} P(x) \log(\frac{Q(x)}{P(x)})$$

Эта мера равна 0, когда два дискретных распределения полностью совпадают.