1. Лекции

1.1 28.09.2023

При фильтрации на основе контента мы опираемся на описание. Из контента извлекается и обрабатывается информация. Два этапа: подготовка данных (автономное обучение, весь контент прогоняется через анализатор и получаем формальную структуру); пользователь взаимодействует с контентом и мы получаем информацию, на основании которой мы получаем профиль пользователя.

Три основных блока:

- Анализатор создает профиль системы;
- Блок создания профиля пользователя;
- Блок извлечения нужных элементов (сравнивает профиль пользователя и профиль элемента и подбирает подходящий).

1.1.1 Блок анализатор контента

Анализатор контента – блок, который формализует описательные данные в формальные, обработку которых уже можно автоматизировать.

На этом этапе мы определяем, что является контентом и какие признаки мы хотим выделять.

Мета-данные – это данные о данных. Спасибо. Если элементами являются фильмы – то мета-данные – это жанр, актеры и прочее. Можно выделить два основных типа: факты и теги. Факты – объективная, неоспоримая информация об объекте (год выхода, например). Теги – ключевые слова – это оценка субъективная, кто-то скажет, что экшн, кто-то, что классическая английская комедия и прочее. Теги могут быть простыми и широкими, а могут быть узконаправленная. Одна из проблем тегов – это разное мнение людей и разное их выражение.

Центральная проблема в обучении рек-системы – это получение мета-данных.

Входные данные – описание документов (объектов). В общем случае – свободный текст на естественном языке. Самый простой способ – считаем частотность слов. Можно слова объединять в темы.

Самый простой способ обработать текст – собрать по нему словарь. Ок. Но как только текст токенизируется, мы теряем часть информации. Второй

момент связан с тем, что в русском языке есть зависимость смысла от расположения слов в предложении.

Также не забываем, что нужно использовать стоп-слова, набор их зависит от предметной области и задачи. Затем советуют убрать минимумы и максимумы, но с минимумами неоднозначно: если слово встречается 1 раз, то аналогию мы навряд ли найдем, однако оно может очень хорошо помочь в определении специфики предмета.

Вторая задача словаря – разобраться с лексическими формами. $pussy == kitty$. Простой способ – стэмминг, отрезаем конец слов, он чаще всего отвечает за изменение формы, однако под раздачу часто отрезается суффикс, коты на равны котлетам. Второй подход – лемматизация. И кот, и котенок, и кошка сведутся к общему знаменателю. Но способ более трудоемкий.

tf-idf – крута. В модели два показателя: частота слов в документе и обратная частота документа. Обычно частота – это количествона на общее количество, а в обратном – обратно, етить-колотить.

$$tf(x,y) = \frac{n_x}{N_y} \tag{1}$$

n – сколько раз слово встретилось; N – сколько всего слов в документе.

В самых простых случаях без x и y – будем получать меньшую точность модели.

Есть еще логорифмическая мера:

$$tf(x,y) = log_{10}(1 + n_{x,y}) \tag{2}$$

$$IDF = \frac{numberofdocuments}{numberofdocumentswithwordx} \tag{3}$$

Эти величины перемножаются. Чтобы уравнять их от IDF берут десятичный логорифм.

$$IDF(x,y) = log_{10}(\frac{numberofdocuments}{numberofdocuments_with_word_x})//left - right \tag{4}$$

$$\textbf{tf-idf}(x,y) = tf(x,y) \cdot idf(x,y) \tag{5}$$

Если какое-то слово имеет высокий вес – значит у него высокая частота в одном документе и низкая частота во всем документе. Чем меньше вес – тем более распространенный.

100 слов. кот = 12 раз. $tf(cat) = 0.12$.

10 000 слов. 300 котов. $idf(cat) = log_{10}\frac{10000}{300} \approx 1.52$. $w(cat) = tf(cat) \cdot idf(cat) \approx 0.18$.

Тыры-пыры мы тут уже обсуждаем LDA. word2vec как-то по итогу обернулся в LDA – Latent Dirichlet Allocation (скрытое распределение Дирихле). То есть анализируем не слова, а косвенные признаки и обобщения.

Дирихле отсылает нас к математическому аппарату метода.

Берем слова документа и строим по ним корпус тем. Корпус – совокупность документов. Задача LDA – обнаружить темы в документе и выполнить автоматическую классификацию документов по этим темам. Классификация – подходит документ или нет. Тема – набор терминов, отдельных слов или фраз, которые вместе описывают тему. Перед запуском алгосика лучше причесать текст, конечно же.

В подходе LDA используется 4 предположения:

- Смысл текста определяется набором ключевых слов тех или иных тем;
- Некоторые термины могут быть неоднозначными, поэтому слово будет относить к теме на процент;
- Большинство документов разные темы встречаются с разной частотой;
- В рамках темы определенные термины используются чаще, чем другие.

пупа и лупа пошли получать зарплату, однако в бухгалтерии все перепутали. По итогу пупа получил за пупу, а лупа за лупу. Ой, то есть ничего не перепутали.

LDA – генеративная модель. А я дегенеративная модель. Хе. Хе. Хе.

LDA – статистическая модель совместного распределения вероятностей.

$$P(X, Y)$$

P(X | Y=y) (6)

X – наблюдаемая переменная, Y – целевая переменная. Нужно перейти

от наблюдаемой X к метке Y.

Общая идея алгоритма: на вход подаются документы и K – количество тем, которые надо выделить. Результатом работы алгоритма будет список из K тем, каждая тема представляет собой вектор, компоненты которого показывают с какой вероятностью термин встречается в этой теме. Второй результат – список векторов, компоненты которых – вероятность отношение i-го документа к теме.

Алгоритмов выделения тем несколько. Мы рассматриваем алгоритм генерации выборки Гиббса.

Каждый документ предобрабатываем, анализируем только слова, а не расположение. Задача создания тем – установить связи между словами и темами, между словами и документами. Слова, относящиеся к одной теме чаще всего будут в одном документе.

Схема выборки Гиббса начинается с случайной установки связей. Затем для каждого слова определяется вероятность принадлежности к той или иной теме. Сопоставляя все слова всех документов мы получим вероятности встреч слова в теме.

Первый настроечный коэф – количество тем. Если маленькое, то будет бо-бо. Если очень большое, то тоже. Ну надо же.

Второй – порог, по которому отсекаем слова, принадлежащие и непринадлежащие темам. При отсечении шума неизбежна потеря информации.

Я хочу домой.

Откуда взят хорошее описание? Хороший вопрос, никак, пошел в жопу!

В общем случае берут некоторое количество тем с потолка. Запускают, смотрят, перекрываются ли полученные темы.

Еще два параметра – α, β. – k-компонентный вектор, отвечающий за выраженность тем в документах. При высоком – доки ближе друг к другу. При низких – разделяем узко-специализированные тексты. Чаще берут $\frac{50}{k}$.

β большое – в теме больше терминов. Маленькое – хз. Лучше всего ≈ 0.01.

Мы разобрались с контентом.

Создание профиля пользователя

Если есть LDA, то смотрим, что понравилось юзеру и сравниваем вектора. Темы здесь нам помогут.

В общем случае алгосик: взять все потребленные элементы, для каждого элемента LDA ищет похожие, рассчитываем оценку на основе сходства, отсортировать по убыванию оценки и релевантности. Но можно и транспонировать, то есть создать вектор LDA для пользователя.

В модели tf-idf все проще, у нас есть векторы с тегами и фактами. Есть список, что нравится пользователю, по нему составляем профиль пользователя, можно ввести веса, связанные с оценками, по итогу получаем вектор вкусов пользователя. Лучше всего еще применить нормализацию. Если наш пользователь любит квашеную капусту и жрет торты, навряд ли он их ест вместе.

Плюсы и минусы фильтрации контента. Плюсы: можно рекомендовать что-то после первой оценки или выбора; она менее чувствительно к глобальной популярности. Минусы: те оценки, которые мы вычисляем, приобретают общую силу; выдача результатов строго определена, никаких неожиданностей; ограниченное содержание.

Там формула, но нам важно только, что K – количество тем, V – количество слов в словаре, M – количество документов, $N_d(d = 1..M)$ – количество слов в документе d, N – общее количество слов, $_k, k = 1..K$ – вес темы k в документе, $\beta_w, w = 1..V$ – вес слова w в теме. $\phi_{kw}, k = 1..K, w = 1..V$ – вероятность нахождения слова в документе. Еще “тэта”, вероятность какая-то. Если че ϕ_kподчиняется$Dirichlet(\beta)$, а тета по альфе.

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

$$p(d, w) = \sum_{z\text{принадлежит}Z} \text{тета} \cdot \phi \cdot p(d) \tag{7}$$

1.1.2 05.10.23

Пора сдавать лабки. Всего их 7.

Матричная векторизация

Попытаемся скрестить ежа с ужом.

Для работы потребуется матрица оценок. Есть явные и неявные оценки, помним. Процесс называется матричной факторизацией. Блять, что тут происходит, нихуя не понятно.

При разложение матрицы на произведение трех из трех разных матриц можем получить три группы информации. Эта тема и дает скрытые факторы,

которые мы хотим проанализировать. Самим раскладывать нахрен не надо, взять готовое.

Самая главная проблема – пустота в клетках, потому что для нас пустая клетка не равна нулю. Обработка этих клеток и является из основных проблем. Тут есть модификации SVD++, funcSVD. Матричная факторизация – эо метод обнаружения скрытых факторов в рамках коллабаративной фильтрации.

Чем меньше в матрице данных, тем быстрее считаем. Общая идея в факторизации заключается в том, что мы сжимаем пространство так, чтобы расстояние менялось пропорционально, дальнее остается дальним, ближнее остается ближним. К тому же мы предполагаем, что в данных есть скрытый смысл.

Факторизация позволяет снизить размерность вычислений. Снизив ее, мы можем выделить области.

Есть векторы вкусов пользователя и есть векторы объектов. Все вместе – матрица оценок. Для удобства пользователи – строки, объекты – столбцы. n пользователей и m столбцов. На первом этапе пустоты матрицы заполняем нулями. $R = U \cdot V$, U – матрица пользователей, V – матрица объектов. $R(n \cdot m), U(n \cdot d), V(d \cdot m)$.

$$r_{ij} = \sum_{k=1}^{d} U_{ik} V_{kj} \tag{8}$$

Классический метод СВД чего боже где я

Матрицу будем раскладывать на 3 произведения. $R_{n \cdot m} = U_{n \cdot n} \cdot \Sigma_{n \cdot m} \cdot V^T_{m \cdot m}$. $U^T U = I_n, V^T V = I_m$.

Веса в матрице выстраиваются по невозрастанию, среди множества чисел можно выделить наиболее важных d, а остальные принудтельно зануляем. Все получается круто, но, к сожалению, теряем на этом точность.

$R'_{n \cdot m} = U'_{n \cdot d} \cdot \Sigma'_{d \cdot d} + V'^T_{d \cdot m}$.

Существует два способа справиться с пустыми клетками. Можно вычислять средние значения. Можно имеющиеся строки нормализовывать, и тогда чувак относится к фильму ни хорошо, ни плохо.

Еще есть алгоритм улучшения результата с использованием базисного предиктора. Так SVD превращается в FunkSVD.

Плюсы и минусы SVD. Одно из основных плюсов – легкость добавления в систему новых пользователей. Как генерить оценки с использованием SVD – два подхода. Простой – прямым перебором. Второй – выделение окрестностей, а дальше уже фильтрация в окрестности.

Недостатки: нули в матрице, заполнение ими. Необходимость регулярного обновления модели. Второй – вычислительные требования. Метод не является интуитивно понятным.

Модификация

SVD++, Funk SVD. Любая оценка пользователя ставит субъективную. Базовым предиктором называют сумму трех параметров – $b_{Ui} = \mu + b_U + b_i$ – базовый предиктор оценки пользователя U объекта i. μ – среднее арифметическое всех доступных оценок. b_U – занижает пользак или завышает оценки. b_i – отклонение объекта от некоторого среднего уровня.

$$b_U = \frac{1}{|I_U|} \sum_{ibelongsI_U} (r_i - \mu)$$

$\mathrm{b}_i = \frac{1}{|U_i|} \sum_{UbelongsU_i} (r_{Ui} - b_U - \mu)$
$\mathrm{b}_{Ui} = r_{Ui} = \mu + b_U + b_i + V_U^T U_i$
$\mathrm{L}(\mu, b_i, b_U, B_u, U_i) = \sum (r_{iU} - r_{iU})^2$(9)

Считаются частные производные, смещаемся в сторону, обратную вычисленному градиенту.

Переобучение системы тоже приветствуется, оно имеется, все там оптимизируется и получается итоговая формула

Существует итерационный процесс, который помогает высчитывает эту какаху. В них гамма – скорость обучения, тысячные. Лямбда – параметр регуляризации, защита от переобучения, настроечный.

$$b_i = b_i + \omega(e_{iU} - \lambda b_i)$$

(10)

Модель нуждается в постоянном пересчете, причем каждый раз, когда пользователь ставит оценку. Если ипользуем неявные оценки – ее наличие не

означает, что пользователь воспользовался товаром.

NMF

И еще один метод – NMF (non-negative matrix factorization).

$V = W \cdot H$ – элементы матрицы неотрицательны.

Размеры матриц меньше, чем у исходной. Этот подход позволяет повысить интерпретируемость. Также этот метод помогает, когда оценки бинарные, то есть рейтинги отсутствуют. V – исходная матрица, W – матрица весов, H – матрица скрытых факторов, показывает, насколько данный фактор важен и актуален для пользователя.

1.2 Лекция 12.10.23

оаоаоаоаоаоооооммммммммм гречка

1.2.1 Гибридные рекомендательные системы

Главное преимущество – увеличение точности и более гибкая настройка. Недостатки тоже есть. Например. большая сложность. Кроме того, усложняется интерпретируемость результата. Выделяют основные виды скрещиваний – коллаборативный и контентный алгоритм. Три варианта: каждый по отдельности и объединить результаты, включить правила одного подхода в подходы другого, построение общей методики, которая объединит оба подхода. Гибриды дают более точные результаты – доказано эмпирически. Плюс невилируются недостатки алгоритмов по отдельности. Основные методы гибридизации:

- Переключение – в зависимости от исходных данных применяется тот или иной алгоритм;
- Смешивание – когда одновременно или последовательно отрабатывают алгоритмы, а их результаты объединяются в общую выборку;
- Взвешенное объединение результатов – тут у алгоритмов появляется вес и строится линейная комбинация результатов;
- Комбинирование признаков – когда признаки пользователей и объектов объединяются и передаются на вход рекомендательному алгоритму какому-то;
- Увеличение функции – результаты одного метода становятся входными данными для другого алгоритма;
- Каскадная система – выстраивается строгая иерархия рек-алгов и алгоритмы ниже по иерархии обладают правом разрывать связи, которые

нашли алгоритмы более высокого уровня;

- Мета-объединение – берем исходные данные, применяем к ним рек-алг, получаем модель, а полученную модель передаем на вход следующей рекомендательной системе.

Впереди планеты всей Netflix. Weird flex :peka:

В общем случае данные, которыми мы располагаем можно разделить на типы:

- Понимание задачи;
- Метаданные контента;
- Поведенческие данные;
- Демографические данные;
- Контекст задачи.

РС на основе знаний – это все, что строится на знании эксперта, для нее используются понимание задачи и контекст задачи.

РС на основе контента – строится на основе метаданных контента и поведенческих данных.

Коллаборативная фильтрация item-based – для нее нужны поведенческие данные и контекст задачи.

Коллаборативная фильтрация user-based – демографические данные, поведенческие данные.

Все это объединяется в гибридный выход.

Выделяют 3 основных видов гибридов:

- Монолитная система – в них элементы того или иного подхода объединяются в единое целое с образованием некоторого нового алгоритма; Монолит требует много сил, но резы могут быть очень интересными;
- Ансамбль рекомендаторов – это несколько алгоритмов, результаты которых объединяются в итоговый результат; Такая система состоит из множества компонентов, причем эти компоненты могут браться из разных алгоритмов; В общем случае там есть картинка, сфоткаю или нет, посмотришь и узнаешь, потомок; В основном тут смешивают контентный подход и коллаборативную фильтрацию, берут лучшую часть из выбранных алгоритмов; Это может упростить фильтрацию;
- Смешанная гибридная рекомендательная система – такая система возвращает комбинацию их результатов всех алгоритмов, входящих в ее

состав. Обычно строится иерархия использованных рекомендаторов, например, иерархия по персонализации – первым отрабатывает тот алгоритм, который учитывает все и сразу, затем вторым отработает более общий алгоритм, на каждом шаге уменьшается степень персонализации. Это нам дает. ГЫГЫГЫ. При таком подходе получаем достаточно большой список рекомендаций, в списке будут как те, что его точно заинтересуют, так и просто популярные объекты для расширения горизонтов. Неоспоримым достоинством является то, что рекомендации всегда и есть и их достаточно, кроме того они приемлемого качества. Хорошо было бы эти реки упорядочить.

Общая идея ансамбля – результаты разных рекомендаторов объединяются в один по тому или иному правилу. В зависимости от правил такие системы тоже будут делить на что-то. Переключаемый ансамбль – из имеющихся алгоритмов выбирается тот, что подходит лучше всего под текущую ситуацию.

Можно еще строить взвешенные оценки. Бубубу. Быбыбы. Оаоаоаоаоао. Мммммммм. Пусть есть два алгоритма – один коллаборат-фильтр, а второй – фильтр на основе контента. Оба хороши, епты, пацаны ваще ребята, но хорошили они одинаково, бля. Фильтрация на основе контента позволит подобрать похожие по типу данные, в том же жанре, но тогда оценки пользователя не учитываются. Но коллаб не анализирует тему. Ну вот так, епты, на деле хуйня. Так пажжи ебана, хорошо было бы их объединить с использованием весовых коэфов, бля. r_{cb} – контент-бейзд хуйня, r_{cf} – коллаб, $r_h = r_{cf} \cdot 0.6 + r_{cb} \cdot 0.4$, можно вычислять коэффициенты линейной регрессией. В общем случае для всей обучающей выборки подсчитывается сумма квадратических отклонений и минимизируется. В общем случае $r_h = \sum_{j=1}^{L} w_i g_i(user, item)$, чтобы найти веса w используется линейка. Но вес может быть не только скалярным, но если юзать там функцию, то будет современно невъебенно охуенно. Такой подход будет называться признако-смешанным линейным сочетанием.

FWLS – Feature Weighted Linear Stacking

Общая идея – гибридная оценка $r_h = \sum_{i=1}^{L} w_i(user, item) \cdot g_i(user, item)$. То есть вес является признако-взвешенной. 40 тонн, хули вы хотели.

В качестве весов для работы используются мета-функции, всевозможные оценки пользователей, журнал оценок пользователя, журнал оценок объ-

екта. Весовая функция рассматривается как линейная комбинация мета-функций: $w_i(user, item) = \sum_{k=1}^{M} V_{ki} f_k(user, item)$. Получается страшная хуета: $r_h = \sum_{i=1}^{L} (\sum_{k=1}^{M} V_{ki} \cdot f_k(user, item)) g_i(user, item)$

Построение гибрида – 7 шагов. Сначала исходные данные делятся на обучающий, тестовый и хуй знает какой. На втором шаге мы обучаем каждый отдельный рекоммендатор, тут, если один из рекоммендаторов отъебнет, то либо удаляем данные, на которых он не работает, либо придумываем блок, который будет предугадывать ту оценку, которой не хватает. Дальше для каждой точки из обучающего набора формируем прогнозы. Потом вычисляется взвешенная функция для учебной и тестовой выборок. Далее собираем оценки в общую форму, все, что неизвестно – вычисляем математикой линейной регрессии. В конце тестируем гибрид, насколько он пиздач или хуев поевший. Если хуев поевший – хуи изо рта вынуть и пускаем дальше обрабатывать.

1.2.2 26.10.23

Методы оценки качества

Все системы, работающие с данными, нуждаются в том, чтобы их поддерживали и настраивали.

В оценивании рекомендательной системы есть три основных этапа:

- Оценка алгоритма
- Автономное или оффлайн тестирование (тестирование алгоритма и регрессионное тестирование)
- Онлайн тестирование (тестирование контролируемой группой, A/B тестирование, поддержка и исследование системы в процессе ее работы)

Первый этап – оценка алгоритма. Мы нашли алгоритм и решили, что будем использовать это или то. Прежде чем его начать реализовывать хорошо в ручную на маленьком наборе проверить его на бумаге, чтобы понять, работает ли оно в нашем случае.

Второй этап. Постоянно проверять, чтобы исправление одной ошибки не приводило к новым.

Готовый алгоритм проверяем уже на доброжелательных пользователях. Если контрольная группа что-то смогла – выпускаем для 20% и так далее.

Чтобы оценить результаты нужно сравнивать результаты рекомендаций. Как понять, правильная ли рекомендация? Нужно подтвердить или проверить гипотезу, что система А дает рекомендации чаще, используемая поль-

зователями, чем система Б. Тогда можно сказать, что система А дает результаты лучше, чем система Б.

4 варианта поведения пользователя:

- Зашел-вышел, ничего не увидел
- Пользователь перебирает несколько страниц, рекомендации видел, но ими не пользовался
- Пользователь увидел рекомендацию и нажал на нее
- Пользователь выбрал несколько рекомендаций, тут уже в зависимости от контекста

Какие могут быть цели с точки зрения рекомендательной системы? Получение прибыли, то есть пользователь должен платить и приходить снова. Можно зарабатывать деньги на довольстве, но можно и не обращать на это внимание.

Что сделает довольным пользователя? Пользак хочет, чтобы система понимала его вкусы, никто не любит кривые рекомендации. В рекомендациях хорошо присутствие приятной неожиданности. Система также должна работать со всем каталогом.

А как оценить вкусы пользователя? Можно взять то, что пользователь уже оценил, построить его якобы несуществующую оценку и сравнить с тем, что имеется. Важно также понимать, как интерпретировать оценку. Чаще всего от 7 до 10 – понравится, от 3 до 7 – может понравиться при опр условиях, от 0 до 2 – не хочу. Система работает хорошо, если она предугадывает уже проставленную оценку.

Разнообразие – параметр. Богатые становятся еще богаче, а бедные еще беднее. Если есть какой-то объект с высокой популярностью, то он будет часто рекомендоваться и чаще рекомендоваться пользователям. Поскольку начальная цель – помочь пользаку разобраться с большим объемом каталога – надо следить за этим. Можно разбить каталог на две части – очень популярная и остальные.

Охват – параметр. Оно близко к понятию разнообразия. Самый простой способ оценить охват – для всех пользователей запустить рекомендатор, и посмотреть, скольким пользователям рекомендатор смог что-то порекомендовать. Формулу писать не буду, помилуйте.

$$o_{user} = \frac{\sum_{u \in U} P_u}{|U|}, P_u = 1if|recset| > 0 \quad (11)$$

Также есть охват по каталогу. Внезапно.

$$o_{item} = \frac{|recset|}{|I|} \quad (12)$$

Снова говорим о проверке на бумаге. Также проводится оценка сложности алгоритма и оценка сложности его реализации.

Также проверяются данные, которые нужны алгоритма. Если до начала реализации окажется, что нам чего-то не хватает – хорошо. Также надо решать там, что будет с пропусками, если данных недостаточно.

Ответили на все вопросики – приступаем к реализации алгоритма. Если блок нужно тестировать – его надо тестировать. Цитаты великих.

Как оцениваем качество? Хорошо запустить его для каждого пользователя и сравнить их с теми оценками, которые проставил пользователя. Исчерпывающее тестирование в сложных реках невозможно. Нужно делать по какой-то выборке. Три сценария проверки: автономная, контролируемый эксперимент и онлайн. Первая базируется на одном правдивом наборе, одна часть является обучающей, на других проверяем точность и прочие метрики MAE, MSE, RMSE, сколько можно уже повторять.

У нас есть 4 комбинации. Рекомендовано – посмотрено, рекомендовано – не посмотрено и тд. Короче обычная матрица ошибок.

Точность $\frac{TP}{TP+FP}$, можно вычислять для k элементов. Ее еще обзывают $p = k(u)$, количество релевантных к этому числу k.

Отклик = $\frac{TP}{TP+FN}$

Средняя точность по m элементам: $CT = \frac{\sum_{k=1}^{m} p}{m}$

От популярных к менее популярным и накладываем штраф по популярности, чем более популярен – тем больше штраф. $hzcho = \sum_{i=1}^{k} \frac{2^{rel[i]}-1}{log_2([i]+2)}$

Надо создать выборку. У. Выбрать можно случайно. Альтернатива – стратефицированная выборка (статистические выбранные параметры выборки совпадали с параметрами основной).

Тестовой набор надо использовать один раз. Короче используем K-fold. Можно разделять на всякие вещи. Например, разделять оценки пользователей на обучающие и тестовые. Можно рандомить, но это плохо. Поэтому кру-

че всех k-fold.

Рано или поздно заканчивается оффлайн. Начинается контроль говна и палок на своих. После контроля идем в а-б тесты.

В конце-концов тестирование должно быть непрерывным.

на рубежке будут вопросы по 8 лекциям + задача, можно пользоваться своими лабами (дана матрица, заполнить пропуски, берем лабу, считаем и списываем). В пятницу.

1.3 Лекция 02.11.23

Консультация будет 08.12 13:50, 513Л

1.3.1 Алгоритмы обучения ранжированию, байесовской персонализированное и прочая хуета

Иногда имеет смысл не угадывать, понравится ли пользователю что-то, а ранжировать, правильно расположить объекты: наверху наиболее подходящие, ниже – менее релевантные.

оаоаоаоаоаоаоаоаоаоаоаоаоммммммммммм

Задача ранжирования созвучна гибридным рекомендательным системам – учитывается множество факторов. Грубо говоря у нас есть несколько параметров, по которым мы можем ранжировать, и мы их приписываем какому-то весу, а потом сидим по 228 блять нахуй так делать.

В принципе, можно ранг брать как оценку рекомендательной системы.

Переранжирование – повторное ранжирование списка. Сначала отсортировали по популярности, а затем используем вывод рекомендательной системы. Алгоритм обучения ранжирования – LTR (Learning to Rank) – модель обучения с учителем. Предоставляем входные и выходные данные. Для рекомендательной системы на вход подается идентификатор пользователя, на выход получаем ранжированный список элементов.

Обучение ранжированию – класс задач машобуча, суть которых состоит в автоматизированном построении ограничений для ранжирующей модели по обучающей выборки для последующего применения к неизвестным объектам со сходной структурой. Делят на три части: точечные, парные, списочные.

Точечный алгоритм LTR рассматривает один объект за раз и определяет для каждого из них ранг. В качестве примеров используются пары признак – значение релевантности. Ранг отвечает только за положение в списке. При-

меняют при этом подходы классификации или регрессии. При таком подходе оценка каждого элемента не зависит от оценки одного элемента. Недостаток подхода – объекты оцениваются независимо.

Парный алгоритм – элементы сравниваются парами. В каждой паре делаем вывод о том, какой из элементов более релевантен. Ключевой момент – абсолютное упорядочивание, один объект точно лучше, чем другой, строгое неравенство. Проблема – каждая пара смотрится независимо.

Списочные алгоритмы – на вход все документы, элементы, объекты запроса. На выходе – перестановка элементов в соответствии с рангом. Как правило используются вероятностные модели и являются намного более сложными.

Как оценить качество ранжирования? Хуй знает. Ладно, не хуй знает. Сейчас будут метрики качества ранжирования.

Бубубу, задача ранжирования в математическом виде. Я не хочу это писать, ты знаешь. Список образуют рекомендуемые элементы, а релевантность определяется тем, чем пользователь воспользуется. Формально? Формально!

$$U = \{u_i\}, i = 1..N \tag{13}$$

$$E = \{e_j\}, j = 1..M \tag{14}$$

$$u \in U; r : E \to \mathbb{R}; e \in E; r(e) \tag{15}$$

$$\{r|e\}_{e \in E} \tag{16}$$

Получаем π – перестановка. Не. Я нахуй не буду это писать.

Методы оценки точности. Если кто-то говорит про точность, то используется бинарный эталон.

Точность на k элементов – precision at k (P@k). k – сколько лучших мы показываем.

$$p@k = \frac{1}{k} \sum_{k=1}^{k} r_{true}(\pi^{-1}(k)) \tag{17}$$

Дальше то же самое, только усредненное. Average Precision at k (ap@k).

Какой долбоеб придумывал эти названия? Да похуй, у нас же есть вообще сервис аналитики Groot, блять. Дерево нахуй. Говорящее.

Дальше еще mean average precision at k. map@k.

$$mapp@k = \frac{1}{N}\sum_{j=1}^{N} apa@k_j \tag{18}$$