ADSC
Applied Data Science Center

Индустриальный анализ данных. Статья 2

О цикле

Данный цикл статей основан на семинарских занятиях открытого курса Data Mining in Action по направлению “индустриальный анализ данных”. На семинарах (и в статьях по ним) мы будем говорить о том, как правильно формализовать задачи, в расплывчатой форме поставленные заказчиком, понимать, какие данные нужны, и строить решения, применимые в бизнесе.
Общую лекцию для всего потока можно посмотреть здесь, а презентацию к ней скачать здесь.

Сегодня в статье:

Кейс. Рекомендации официантам

Вступление

К вам приходит владелец крупной сети ресторанов и излагает свою проблему. Он хочет, чтобы в конце обслуживания официанты говорили: "может быть, вы хотите заказать еще вот это блюдо?". Люди бы соглашались, покупали и приносили бы дополнительный доход. От вас заказчик хочет, чтобы вы

разработали для него систему рекомендаций блюд для каждого отдельного заказа. Что должен в конце предложить официант, чтобы все сказали “да, отличная идея, я возьму”?

Общение с заказчиком

Задавайте вопросы. Получайте ответы. Поймите, что от вас хотят, и узнайте, в чем проблема.

Вопрос: Сколько блюд должен рекомендовать официант?

Ответ: Мы планируем сделать следующую систему: у официанта есть девайс, куда он записывает весь заказ и нажимает кнопку, после чего ему высвечиваются три рекомендации. Три - потому что на экран столько влезает.

Вопрос: У нас есть какие-то персональные данные о человеке?

Ответ: У нас нет. У официанта - да. Он ведь видит клиента.

Вопрос: Может ли официант предоставить нам эту информацию? Пол, возраст?

Ответ: Нет, официанты не могут вводить это в систему.

Вопрос: Есть ли какие-то скидочные карты, карты лояльности?

Ответ: Нет. У нас солидный ресторан, мы без скидок.

Вопрос: Есть история покупок каждого клиента?

Ответ: Нет. Мы знаем, что он заказал сейчас. Но специальных персональных карт, чтобы идентифицировать человека и хранить его глобальную историю заказов, у нас нет. Вы не знаете, кто это. Текущий чек - это все, что доступно о клиенте. Возможно, доступно кое-что еще, но вы пока об этом не спрашивали.

Вопрос: Может быть, у нас есть история чеков? Если да, за какой период?

Ответ: Есть. Скажем, года за два.

Вопрос: Нет ли данных о том, что официанты предлагали раньше?

Ответ: Мы не запускали такую систему раньше. Официанты никогда не предлагали ничего дополнительно.

Вопрос: У нас есть время заказа? Время, когда клиент сел и во сколько заказал?

Ответ: Есть.

Вопрос: Если людей за столиком несколько, то чек все равно один?

Ответ: Да.

Вопрос: А меню постоянное?

Ответ: Более-менее. Есть позиции от шеф-повара, которые иногда меняются...

Вопрос: Можно ли по позициям в чеке понять, какой товар заказали раньше другого?

Ответ: Да, мы знаем, в каком порядке они добавлялись в чек. Но чтобы не переусложнять систему, давайте не будем сейчас разбирать такие случаи, как “заказал”, а потом “еще добавил десерт”. Представим, что бывает только так: клиент выбрал, заказал, съел и оплатил.

Вопрос: Мы хотим оптимизировать суммы дополнительно заказанных блюд? Наша цель - это повысить продажи или лояльность клиентов?

Ответ: Мы хотим заработать, но при этом не заспамить человека предложениями.

Вопрос: А мы должны предлагать что-то из одной категории блюд? Выбрать один из трех предложенных десертов, например. Или из разных категорий?

Ответ: Как угодно.

Формулировка задачи

Теперь, исходя из полученной информации, нужно ответить на следующие вопросы:

- Как будет использоваться система?

Ответ: Мы сформировали заказ, система выдала три рекомендации. Официант посмотрел, решил и предложил клиенту одно из блюд. Как именно он решил - мы обсудим позже.

- Какие есть данные и какие данные доступны на момент предсказания?

Ответ: Чеки, которые дают нам время и дату заказа, его состав и сумму, расположение ресторана. Нам важно знать, где находится ресторан: у метро или на отшибе. Кроме этого, у разных ресторанов может быть не совсем одинаковое меню. Это тоже нужно учитывать.

- Как нам обучить на этих данных модель?

Ответ: Прежде чем обучать модель, необходимо подумать, как мы оценивали бы на исторических данных качество произвольной рекомендательной модели.

- В таком случае что объявить целевой переменной, чтобы можно было оценивать качество?

Ответ: У вас есть состав заказа в чеке. Мы можем взять все чеки и убрать из каждого одну случайную позицию (блюдо). Это блюдо - таргет, который наша система в идеале должна рекомендовать. Так мы получаем выборку, на которой можем обучаться и оценивать качество произвольной рекомендательный системы.

- Как измерять качество?

Ответ: Задача очень похожа на задачу классификации, поэтому можно было бы взять стандартные метрики. Но эти метрики будут плохо отражать выгоду от использования системы. Поэтому лучше использовать метрики из области рекомендательных систем -- долю “правильных” рекомендаций: предложено ли среди трех вариантов (наша система дает нам три позиции, так устроен девайс) то блюдо, которое мы вытащили из чека.

- Как предсказывать список из трех позиций?

Ответ: Можно было бы решать как задачу многоклассовой классификации, но блюд может 200, 2000 или 20000. Это очень много и обучить такую модель весьма непросто. Поэтому часто поступают следующим образом. Давайте вместо классификации на множество классов будем брать пару чек-товар и предсказывать, был этот товар скрыт или нет? Таким образом, мы получаем бинарную классификацию. А как выбирать, какое блюдо рекомендовать к чеку? Можно сортировать по вероятности: чем более вероятна эта пара, тем лучше рекомендовать блюдо из неё.

- Как мы будем создавать негативные примеры?

Ответ: У нас есть только один правильный ответ. А все остальные - неправильные.

- Среди неправильных примеров теоретически могут попастся хорошие. Что делать в таком случае?

Ответ: Во-первых, если вариант действительно хороший, то он будет попадаться и как положительный пример, и как негативный. Вопрос лишь в том, как часто. Ведь мы будем сортировать объекты по предсказанию модели, поэтому по-настоящему важны не абсолюты, а отношения между ними. Во-вторых, представьте, что у нас двести позиций в меню. Тогда сколько негативных примеров приходится на один позитивный? 199. Многовато.
В таких ситуациях делают **негативное семплирование**: берут в выборку один позитивный объект и не 199 негативных, а случайные n из них. Например, пять.

Таким образом мы создаём некоторый баланс между классами, и модели проще обучиться.

Важные детали

Как всегда, давайте уточним некоторые “подводные камни”.

1. Официант ввел позиции заказа, и девайс выдал три рекомендации. Из них он должен предложить только одно блюдо и только один раз. Что он предложит?
Можно предлагать самое дорогую позицию из рекомендованных. Но это не значит, что наша прибыль вырастет. Алгоритм не может учесть этого, он не знает, согласится ли человек взять это блюдо или нет.
Важно, чтобы мы **задумались** над этим вопросом и продумали стратегию. Например, мы можем сказать официанту: “твори”. И он будет выбирать из трех вариантов тот, который, как ему кажется, лучше подходит в данной ситуации. Официант, в отличие от нас, знает, что за человек сидит за столиком. Это единственная наша надежда учесть, какой клиент перед нами.
Наиболее подходящее блюдо из рекомендованных может быть не на первом месте. Однако если мы просто покажем рекомендации с пунктами один-три, то официанты поймут, что первое релевантнее остальных, и начнут, независимо от того, кто сидит за столиком, предлагать им первое блюдо (даже если уместнее было бы третье). Поэтому мы не должны дать им возможность узнать, что рекомендации ранжированы. Для этого можно, например, менять рекомендации местами.

2. Мы не оптимизируем деньги. Придумали себе какую-то систему, зачем рекомендуем и какая от этого выгода - непонятно… Когда мы формировали нашу выборку, мы не брали в расчет, что не угадать подходящее блюдо за *X* рублей - это не то же самое, что и не угадать блюдо за *Y* рублей.
Можно сделать так, чтобы система выдавала нам не просто ответ, угадали мы блюдо или нет, а то, на какую сумму мы угадали. После чего ввести новую метрику и оптимизировать уже ее.

3. Может оказаться, что какой-то ресторан находится на отшибе, где люди бедные и не будут ничего брать дополнительно, что бы им не предложили. Или это ресторан возле бизнес-центра, где люди берут только ланч и все. Логика поведения в разных ресторанах отличается. Поэтому можно делать предсказание по тройке чек-товар-ресторан. А кроме - добавить время заказа или другие признаки, которые вы хотите учесть.

Дизайн эксперимента

И вот вам позволяют провести эксперимент. У нас есть две недели и много ресторанов. Это наш шанс собрать больше данных. Как бы мы проводили онлайн эксперимент в этой задаче? Как сделали бы A\B тест?
Предлагаем вам подумать над этим вопросом самостоятельно в чате направления в Telegram. Вот несколько подсказок и вопросов:

- Можно разбить официантов на две группы: часть из них работает по старой схеме, а часть - по новой, используя систему рекомендаций.
- Важно помнить: наша главная задача в данный момент - это не сделать финальную модель с хорошим качеством, а собрать как можно данных.
- Но какие данные мы хотим собирать? И как нам обучить модель после, чтобы получить максимальную прибыль?

Разбор домашнего задания

Задача

Входные данные

У вас имеется поток данных (генератор data_stream). Поля - это случайные величины (так сделано для упрощения генерации данных). Есть три поля (названы по уровню сложности задания).

Задание

У вас есть куча временных рядов, вы хотите научиться предсказывать следующее значение по 1000 предыдущим. 1000 признаков окна - это слишком много, и вы решили заменить их 5-ю: средним, дисперсией, минимумом, медианой и максимумом. Однако все эти признаки надо подсчитать, причем хочется уметь это делать быстро (в течение часа).

Для каждого поля нужно сделать следующее:

1. Пробежаться по данным окном размера 1000 (окно сдвигается на 1, то есть следующее окно пересекается с предыдущим по 999 элементам).
2. Для каждого окна посчитать среднее значение поля и его дисперсию. Делайте yield этих значений, получая генератор tuple.

3. Для каждого окна найти минимум, медиану и максимум в нём. Делайте yield этих значений, получая генератор tuple.

Ответом, который нужно будет записать в гугл форму, является среднее значение tuple по получившемуся потоку, округлённое до 2-го знака.

Замечания

1. Обратите внимание, как генерируются поля. Постарайтесь понять особенность каждого поля и то, как это можно использовать. Желательно, чтобы для каждого поля у вас было своё решение, максимально эффективно использующее знание об этом поле.
2. Полезные библиотеки: itertools, numpy, collections + все, что можно найти в интернете и поставить через pip install.
3. **Медианой отсортированного массива arr считайте значение arr[len(arr) // 2].**

Если измерять время работы функций временем работы функции example, то примерное время работы такое: одновременно среднее, дисперсия - 1.17. Одновременно минимум, максимум и медиана: easy - 0.87 medium - 2.11 nightmare - 2.85.

Разбор

Код решения можно посмотреть здесь. Ниже представлены некоторые комментарии.

Итак, какие три потока данных есть?
Первый - это **easy**. Поток монотонный. Соответственно, когда вы будете пробегать окном по нему, там очень легко находить медиану, максимум, среднее и прочее.
Суть **medium** была в том, что различных значений очень мало, и это можно было как-то использовать.
Ну а **nightmare** был просто произвольным потоком достаточно больших произвольных чисел.

Среднее и дисперсия (общее для всех полей)
Проблема в точности: когда мы начинаем складывать большие float, они переполняются, и мы теряем в точности каких-то знаков.

Для решения этой проблемы нужно было использовать библиотеку decimal (стандартная библиотека Python), которая помогает вычислять более точно. Так можно задать, сколько знаков после запятой вы хотите хранить, и выполнять вычисления при помощи этого.
Дисперсию и среднее при переходе из одного окна к другому можно рассчитывать прибавлением и вычитанием (за счет того, что вам важна только сумма и сумма квадратов чисел в окне).
Чтобы все не считалось слишком долго, вы можете оставить деление на самый конец. Практически все формулы для дисперсии и среднего вы можете выразить как сумму чего-то, разделенную на сумму чего-то другого (или какое-то n). Сумму сверху вы можете посчитать в int, потому что все числа целые. В данном случае вас будет интересовать сумма квадратов чисел, просто сумма чисел и по каждому окну квадрат суммы чисел в этом окне. Все эти числа можно насчитать.

Еще один момент: вам нужно итерироваться окном по какому-то потоку. Как?
Первый способ: заводите переменную очередь и просто берете очередное значение, добавляете в конец. Если у вас больше тысячи значений, убираете одно значение из начала.
Второй способ (тоже можно делать стандартными средствами Python): использовать модуль itertools. В нем есть метод tee: он принимает поток и возвращает вам два указателя (итератора) на его начало. Теперь вы можете честно брать два указателя и итерироваться ими, поочередно их переключая. Внутри tee все организовано, как та же самая очередь: когда вам нужно сдвинуть какой-то один итератор, вы просто вызываете next от этого итератора, и он пододвигается.

Итак, вы можете насчитать много больших больших сумм и сделать деление только в самом конце, используя decimal. Поскольку в Python есть длинная арифметика для целых чисел, то суммы будут рассчитаны максимально точно. А после, когда вы вызовете decimal от суммы, вы ничего не потеряете. Поделить одно decimal на другое - это тоже хорошая операция.

Если бы мы использовали decimal везде и поставили у него точность два знака, то на самом деле это бы не работало по следующей причине: когда вы складываете два числа с точностью до второго знака, у вас точность суммы - это первый знак. И из-за этого, когда нужно было 10 тысяч чисел сложить с точностью два знака, вы бы просто потеряли на несколько порядков. Поэтому деление нужно было оставить на самый конец.

Мораль: в Python не все идеально с точностью. Когда нужно предсказать до пятого знака после запятой, лучше до последнего работать с int.

Минимум, медиана и максимум

Для поля **easy** (здесь все отсортировано уже по построению): вы содержите очередь, добавляете в начало, добавляете в конец и просто смотрите первое значение, последнее и посередине.

Для поля **medium** можно было использовать тот факт, что у вас всего 256 значений. Вам не нужно итерироваться очередью: можно просто хранить массив на 256 чисел и знать, сколько раз каждое число встретилось. После этого просто используем питоновский бинарный поиск для того, чтобы понять, сколько позиций вы только что уже посмотрели. За счет того, что это все написано на numpy, работать будет достаточно быстро. Идея в том, чтобы использовать то, что у вас очень мало значений.

Можно было не делать все эти предыдущие выкладки, а написать один общий код для всех полей. Многие из слушателей использовали модуль bisect. Кроме этого, можно было бы найти библиотеку, которая поддерживает отсортированные списки с хорошими структурами данных, которые умеют хранить значения в отсортированном порядке и вставлять в произвольное место.

Для поля **nightmare** можно повторить решение для easy (так как он тоже монотонен в общем виде), но надо найти дополнительную библиотеку.

Правильные ответы

	Easy	Medium	Nightmare
Среднее	4999675.28	127.48	499880345.88
Дисперсия	83439.34	5455.17	83228908564031114.59
Минимум	4999175.79	0.02	1017512.29
Медиана	4999675.78	127.60	500438415.64
Максимум	5000174.76	254.98	999017359.97

Теперь мы разобрались с различными библиотеками Python и поняли, что это такое. Дальше будет Machine Learning: сложно, но по-другому.

Оставить отзыв по прочитанной статье вы можете здесь. По всем вопросам пишите на почту курса: dmia@applieddatascience.ru