Абрамов Александр Сергеевич. ИАД-1.

№1.

① а. Prophet:

$$F(t) = t(t) + S(t) + g(t) + \varepsilon(t)$$

$F(t)$ — значение ряда в момент времени t

$t(t)$ — трендовая составляющая

$S(t)$ — сезонность

$g(t)$ — эффект праздников

$\varepsilon(t)$ — случайная ошибка

б. ARIMA:

Совокупность двух моделей (AR и MA).

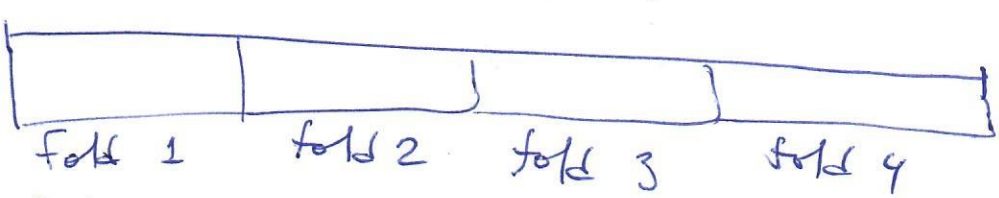
- Считаем d-ю производную ряда
- $AR(t)$ — среднее значение ряда производных до t
- $MA(t)$ — средняя ошибка предсказаний до t
- $ARIMA(t) = AR(t) + MA(t)$

в. RNN.

Фактически, временной ряд — последовательность, поэтому к нему применимы все те же методы, что и к последовательностям, включая RNN: последовательно передаём в модель значения ряда и предсказываем следующие.

② Стандартная K-Fold Кросс-валидация будет предсказывать момент времени t на основе "будущего":

Датасет:



Стандартно: fold 2 предсказывается на основе fold 1, 3, 4.
Это плохо и не имеет смысла в задаче прогнозирования.
Лучше использовать другой подход: берём первые n значений ряда и предсказываем по ним следующие k.

То есть:

1 итерация: по fold 1 предсказываем fold 2 (или его начало)

2 итерация: по fold 1 и fold 2 предсказываем fold 3

3 итерация: по fold 1, fold 2 и fold 3 предсказываем fold 4

Таким образом, при предсказаниях модель не "заглядывает" в будущее.

③ Дополнительные факторы можно представить как временные ряды — зависимость фактора (напр., прогноза погоды) от времени.

Полученные значения можно передавать в модель наравне с предсказываемым рядом: фактически, задачу можно сформулировать так:

Вход:

а. N значений целевого ряда

б. Q значений прогноза погоды в какие-то моменты времени (например, ± неделя)

Выход: K значений целевого ряда

Например, можно взять две RNN (одна "смотрит" на целевой ряд, а другая — на прогноз погоды), объединять их выходные векторы и прогнозировать целевой ряд на основе этого "большого" вектора.

④ Если прогноз идеально соответствует реальным данным погоды (мало ли какие технологии у мутантов), то проблем не будет.

Но если прогноз также составляется какой-то моделью, он, скорее всего, имеет ошибку, из-за чего наша модель будет получать на вход ошибочные данные. Естественно, хорошего прогноза радиации в таком случае она не даст. Особенно учитывая,

Это модель училась на реальных исторических данных и "не знает", насколько прогноз погоды соответствует действительности.

Фактически, модель обучена на одних данных, а предсказания делает по другим. Такая модель не имеет смысла.

Чтобы построить модель правильно, стоит следовать двум принципам:

1) Не заглядывать в будущее: на вход модели должны подаваться только те данные, в качестве которых мы уверены — исторические.

2) Обучать модель на тех же данных, которые будут даваться при прогнозировании. Если уж и хочется использовать прогноз погоды в модели, обучать модель тоже нужно на прогнозе, а не на реальных данных.

№2

① Нормализационный поток заключается в нахождении обратимой дифференцируемой функции f, которая переводит неизвестное распределение реальных данных в известное статистическое распределение (напр. $N(0,1)$) и обратно.

~~То есть, известно $p(x)$ фун~~

То есть, дано объект из распределения, заданного функцией $p_x(x)$. Оно неизвестно, есть только примеры объектов

Выберем распределение $p_z(z)$. Например, $N(0,1)$.

Предполагаем, что объект из $p_x(x)$ переводится в точку из $p_z(z)$ функцией $z = f(x)$.

Тогда $p_x(x) = p_z(f(x)) \cdot \left| \det \frac{\partial f(x_i)}{\partial x_i} \right|$.

Обучим эту $f(x)$ любым удобным способом. Например, градиентным спуском.

Тогда мы хотим брать случайный объект из $p_z(z)$ (сэмплировать) и генерировать по нему объект из $p_x(x)$ с помощью $x = f^{-1}(z)$

② Составим модель из трёх этапов:

1). Сэмплируем случайный шум из некоторого распределения.

2) Кодируем текст в виде некоторого вектора. Например, с помощью RNN или трансформера.

3). Объединяем векторы, полученные на этапах 1/2 и генерируем по ним картинку любым удобным способом.

Фактически, получился Conditional GAN.

Для обучения можно создать ещё одну модель — дискриминатора, который будет предсказывать, настоящая картинка (по картинке и вектору её текстового описания). Таким образом, можно выбрать функцию потерь (например, MSE) и "заставить" генератора "тянуть" её вверх, а дискриминатора — вниз.

Если повезёт, она обучится и будет генерировать картинки.

③ Можно использовать ту же модель, что в п.2, но заменить энкодер текста на энкодер аудио (RNN).

Или можно превращать аудио в текстовое описание (распознавание речи) любой удобной моделью, а затем использовать модель из п.2 в полной мере.

Обучать модель нужно на парах "голосовое описание изображения — картинка".

№3.

① Пусть $R_{n \times m}$ – матрица отзывов пользователей.
- n – количество пользователей
- m – количество фильмов
- Каждый элемент матрицы – число, обозначающее, насколько этому пользователю понравился этот фильм.

Предположим, что $R_{n \times m} = U_{n \times r} \cdot F_{r \times m}$, где
- r – размерность латентного пространства модели.
- $U_{n \times r}$ содержит описания всех пользователей
- $F_{r \times m}$ – описания всех фильмов

Тогда вектор оценок можно получить произведением строки U и столбца F.

Обучаемые параметры – матрицы U, F.

Функционал – MSE: $\sum (R[i][j])^2 - \sum ((UF)[i][j])^2$.
Можно регуляризовать.

Алгоритм обучения:

Вариант 1: обучаем обе матрицы сразу

Вариант 2: замораживаем одну матрицу и обучаем вторую. И так по-очереди.

② $R_{1000 \times 100} = U_{1000 \times 10} \cdot F_{10 \times 100}$

$\Rightarrow$ Обучаемых параметров: $1000 \cdot 10 + 10 \cdot 100 = 11000$ штук.

Слагаемых будет $1000 \times 100 = 100.000$ штук.

Количество пар пользователь – айтем (известных) не влияет на ответ.

На месте неизвестных в R можно просто поставить нули.

③ Эти подходы хуже тем, что они плохо учитывают особенности каждого пользователя. (вообще не учитывают).

Самый популярный:

+ Скорость работы – сделать предсказание нетрудно.
+ Нет проблемы "холодного старта" – что делать, когда появился новый пользователь или фильм.
– Вообще не учитываются особенности пользователей
– Низкое разнообразие
– Популярные становятся ещё популярнее, а непопулярные не рекомендуются.

Случайный:

+ Высокое разнообразие
+ Относительно быстро
– Не учитываются интересы пользователей

④ offline – online:

Улучшение метрик при обучении не всегда соответствует реальному улучшению качества рекомендаций, так как "искусственные" метрики не учитывают всех бизнес-факторов и реальное поведение пользователей.

feedback loop:

Рекомендательная система фактически обучается на своих же предсказаниях. Если фильм рекомендуется, его смотрят больше. А если фильм не рекомендуется, то по нему не собирается достаточное количество данных, из-за чего он никогда не попадёт в рекомендации.

№4.

① а. Сбор данных для обучения.

б. Очистка данных – чем меньше в них ошибок, тем лучше обучится модель.

в. Выбор предобученной модели (GPT, Llama, Mistral и др.). Обучать модель "с нуля" в большинстве случаев нецелесообразно.

г. Передаём часть данных в модель

д. Считаем функцию потерь в зависимости от решаемой задачи (например, кросс-энтропию).

е. Вычисляем градиенты функции потерь по всем / или части параметров.

ж. Обновляем значения всех или части весов.

з. Оцениваем качество.

Зачастую на этапе б. имеет смысл зафиксировать формат входных/выходных данных модели:

- Каждый запрос (prompt) может быть построен в виде диалога.
- Запрос может содержать примеры ("вот пример решения. вот задача. реши по аналогии")

(2)

а. Обучаем всю модель, все параметры.
- Скорее всего, займёт много времени, но не обучится при отсутствии огромного количества вычислительных ресурсов

б. Обучаем некоторые слои. Обычно — последние

в. LoRA:

- для каждой матрицы весов $W_{n \times m}$ делаем две матрицы $A_{n \times r}$ и $B_{r \times m}$, где $r < n$, $r < m$.
- Обучаем A, B, заморозив W.
- Прибавляем к каждому W произведение $A \cdot B$:

$$W \leftarrow W + A \cdot B$$

Фактически, для каждого веса W мы обучим некоторый ΔW, который "подгоняет" модель под нашу задачу.

- В инференсе наша дообученная модель не сложнее исходной.
- Обучаемых параметров в разы меньше, чем в исходной модели, хотя обучаются по сути все слои.

③ prompt – запрос к языковой модели. Некоторый текст, на основе которого модель сгенерирует продолжение.

Zero-shot learning:

Prompt-ы не содержат примеров. Фактически, prompt – произвольный текст, с которым модель должна разобраться самостоятельно.

Few-shot learning.

Каждый prompt содержит не только запрос, но и примеры решения задачи. Таким образом, модель может учиться отвечать на запрос, так как буквально только что ей показали цепочку рассуждений, как решать задачу. При этом можно обучить модель даже описывать свои рассуждения.