Бустинг

лекция 9

Алексей Ярошенко

Проверить,
включена ли
запись лекции

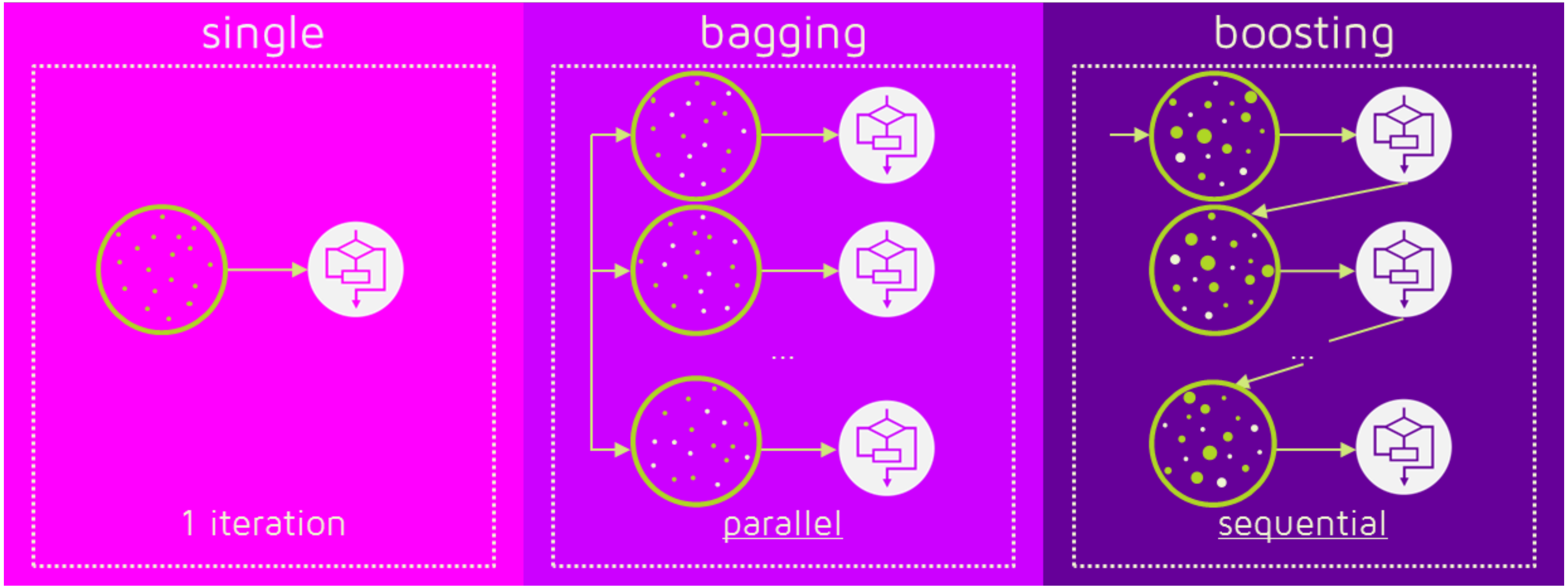
Random Forest. Recap

- Какие по глубине деревья учатся в случайном лесе и почему?
- Что уменьшают деревья в RF, bias или variance?
- RF - параллельный или последовательный алгоритм?

А как последовательно? Boosting

Бустинг

Давайте строить модели не независимо, а чтобы следующая модель исправляла ошибки предыдущих



Бустинг

$$F(x) = f_0(x) + c_1 f_1(x) + \ldots c_n f_n(x)$$

$F(x)$ — ансамбль, $f_i(x)$ — базовый алгоритм

Основные идеи

- строим композицию “слабых” моделей (которые показывают точность немногим лучше рандома)
- итоговая модель - взвешенная сумма базовых
- строим модели **итеративно**, сначала $f_0(x)$ потом $c_1, f_1(x)$ и так далее

Бустинг формально

Данные: $\{(x_i, y_i)\ i = 1\ \ \dots\ \ N\}$

Модель - композиция: $F_n(x) = \sum_{j=1}^{n} c_j f_j(x)$

Функционал: $Q = \sum_{i=1}^{N} L(F_n(x_i), y_i) = \sum_{i=1}^{N} L(\sum_{j=1}^{n} c_j f_j(x_i), y_i)$

1. Взять начальное приближение $f_0(x)$ (например константа)

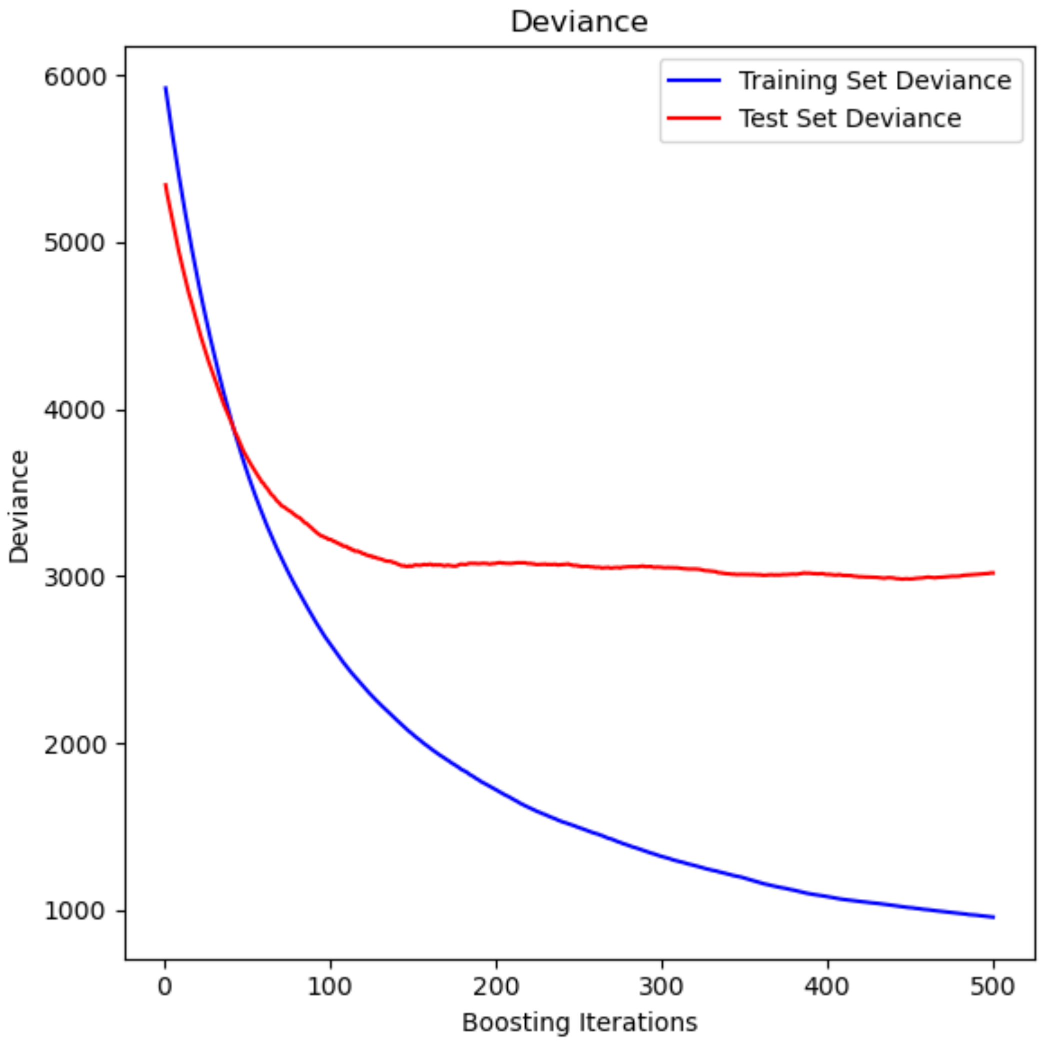
2. $c_k, f_k = argmin_{c,f} \sum_{i=1}^{N} L(F_{k-1}(x_i) + c * f(x_i), y_i)$

3. $F_k = F_{k-1} + c_k f_k$

4. Повторять 2) 3) до сходимости

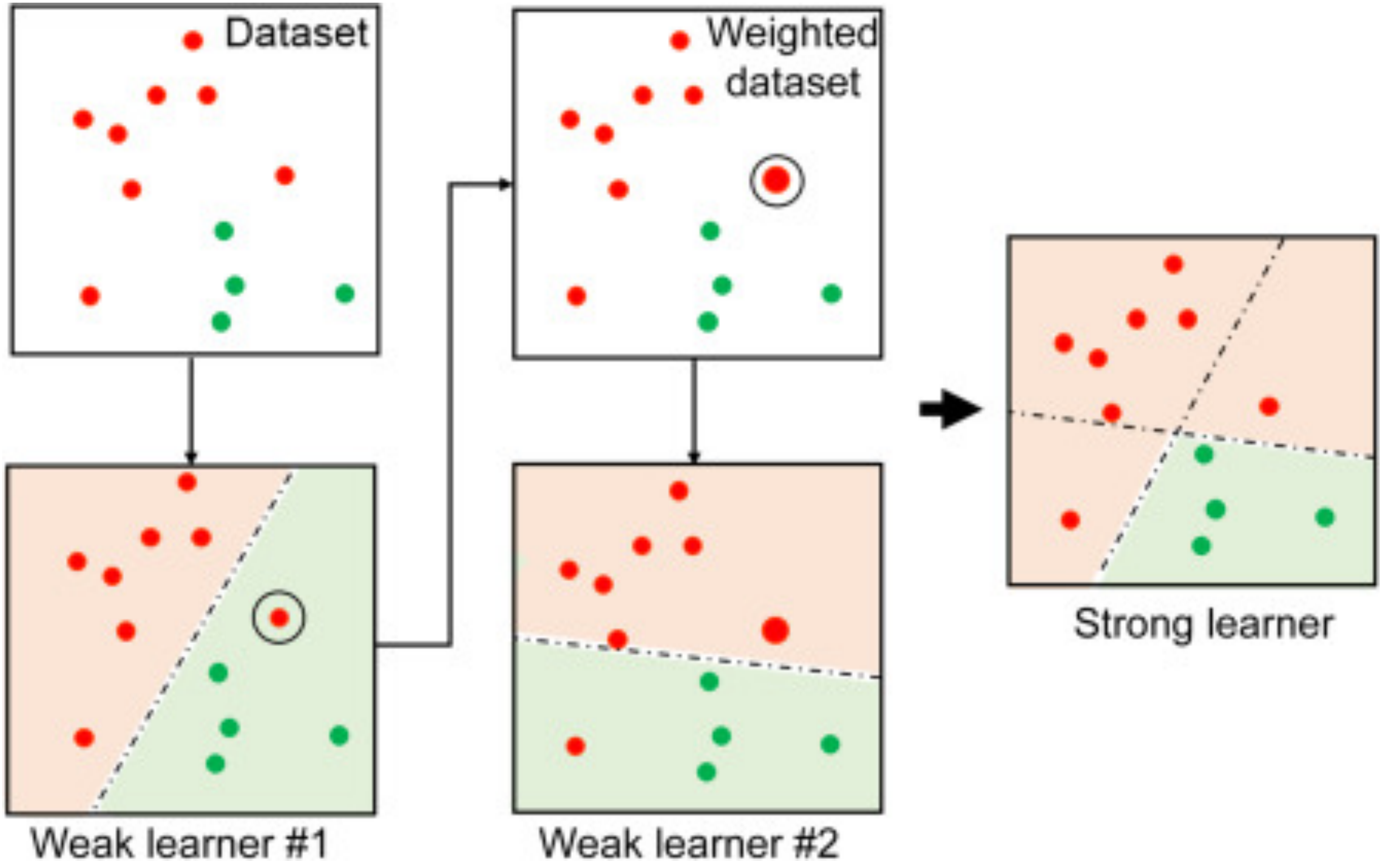
Особенности бустинга

- Нельзя использовать сверх переобученные модели, как в случайном лесе (нулевой лосс на трейне)
- Число алгоритмов надо тоже подбирать



Немного некромантии AdaBoost, LogitBoost, L2Boost…

AdaBoost. Интуиция



AdaBoost. Exp. loss

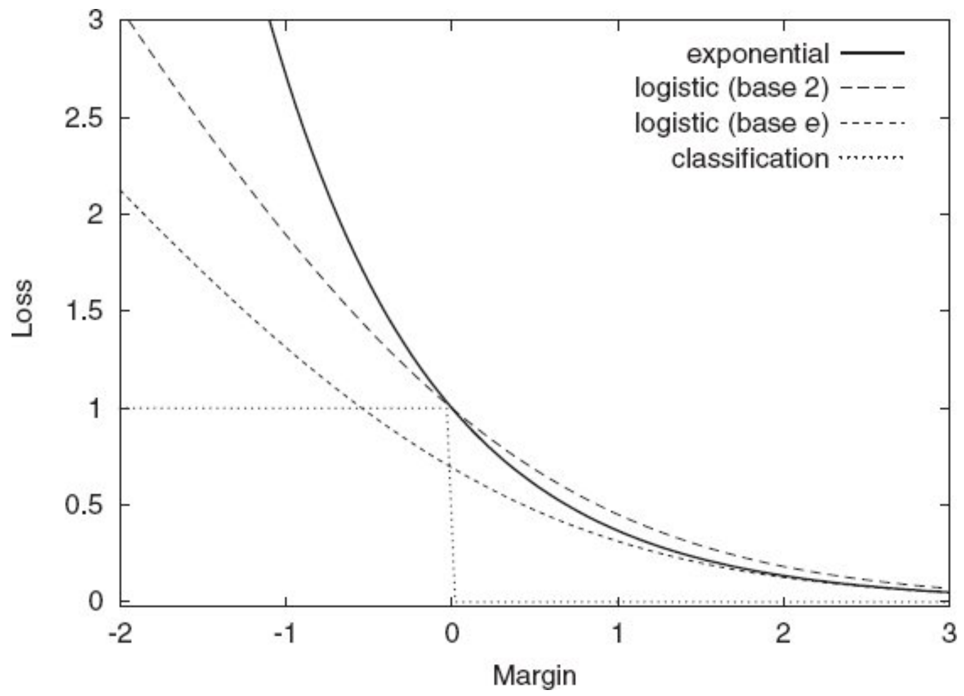
Задача бинарной классификации: $\{(x_i, y_i)i = 1 \ \dots \ N\},\ y_i \in -1, +1$

Модель: $h(x) = sign(\sum_{j=1}^{n} c_j f_j(x))$, обозначим $F(x) = \sum_{j=1}^{n} c_j f_j(x)$

Функционал качества: $err(F) = \frac{1}{N} \sum_{i=1}^{N} I[y_i \neq h(x_i)]$

Ограничение сверху функции потерь:

$$I(y \neq h(x)) = I(y\ F(x) \leq 0) \leq exp(-yF(x))$$



AdaBoost. Веса данных

Пусть есть $c_1, f_1(x) \ \dots \ c_{k-1}, f_{k-1}(x)$, хотим найти c_k, f_k

$$c_k, f_k = argmin_{c,f} \sum_{i=1}^{N} L(F_{k-1}(x_i) + cf(x_i), y_i), \ L(x_i, y_i) = \exp(-y_i F_k(x_i))$$

Перепишем$L(x_i, y_i) = \exp(-y_i(F_{k-1} + c_k f_k(x_i))) = exp(-y_i F_{k-1}) \ exp(-y_i \ c_k \ f_k(x_i))$

Обозначим $w_i = \exp(-y_i F_{k-1})$, тогда

$L(x_i, y_i) = w_i \ exp(-y_i c_k f_k(x_i))$

$$Q = \sum_{i=1}^{N} L(F(x_i), y_i) = \sum_{i=1}^{N} w_i * exp(-y_i c_k f_k(x_i))$$

AdaBoost. Веса классификаторов

Перепишем функционал, используя $e^{c_k y_k f_k} = e^{c_k} I[f_k = y_k] + e^{-c_k} I[f_k \neq y_k]$

$$Q = \sum_{i=1}^{N} L(F(x_i), y_i) = e^{-c_k} \sum_{y_i = f_k(x_i)} w_i + e^{c_K} \sum_{y_i \neq f_k(x_i)} w_i$$

$$= e^{-c_k} \sum w_i - e^{-c_k} \sum_{y_i \neq f_k(x_i)} w_i + e^{c_k} \sum_{y_i \neq f_k(x_i)} w_i$$

$$= (e^{c_k} - e^{-c_k}) \sum_{y_i \neq f_k(x_i)} w_i + e^{-c_k} \sum w_i$$

Оптимально обучить классификатор f_k на выборке с весами w_i

Подставляя f_k и оптимизируя по c_k

$$c_k = \frac{1}{2} \log \frac{1 - eer_k}{err_k}, \text{ где } err_k = \frac{\sum_{i=1}^{N} w_i I[y_i \neq f_k(x_i)]}{\sum_{i=1}^{N} w_i}$$

AdaBoost. Алгоритм

1) Инициализировали веса $w_i = \dfrac{1}{N}$

2) обучили классификатор f_k с весами w_i ,

нашли $err_k = \dfrac{\sum_{i=1}^{N} w_i I[y_i \neq f_k(x_i)]}{\sum_{i=1}^{N} w_i}$

и $c_k = \dfrac{1}{2} \log \dfrac{1 - eer_k}{err_k}$

3) Достроили ансамбль $F_k = F_{k-1} + c_k f_k$,

обновили $w_i = \exp(-y_i F_k(x_i))$

4) Повторять 2) и 3) M шагов

Итоговая модель: $a(x) = sign(\sum_{j=1}^{N} c_j f_j(x_i))$

LogitBoost, L2Boost и прочие представители семейства

$$L(x_i, y_i) = \log(1 + \exp(-y_i F(x_i)))$$

LogitBoost (two classes)

1. Start with weights $w_i = 1/N$ $i = 1, 2, \dots, N$, $F(x) = 0$ and probability estimates $p(x_i) = \frac{1}{2}$.
2. Repeat for $m = 1, 2, \dots, M$:
   (a) Compute the working response and weights
   $$z_i = \frac{y_i^* - p(x_i)}{p(x_i)(1 - p(x_i))},$$
   $$w_i = p(x_i)(1 - p(x_i)).$$
   (b) Fit the function $f_m(x)$ by a weighted least-squares regression of z_i to x_i using weights w_i.
   (c) Update $F(x) \leftarrow F(x) + \frac{1}{2} f_m(x)$ and $p(x) \leftarrow (e^{F(x)})/(e^{F(x)} + e^{-F(x)})$.
3. Output the classifier $\text{sign}[F(x)] = \text{sign}[\sum_{m=1}^{M} f_m(x)]$.

Еще есть L2Boost для MSE.
и т.д.

Новый лосс — новый алгоритм

Минус?

Градиентный бустинг

Вопрос на собеседовании

Почему градиентный бустинг называют именно градиентным?

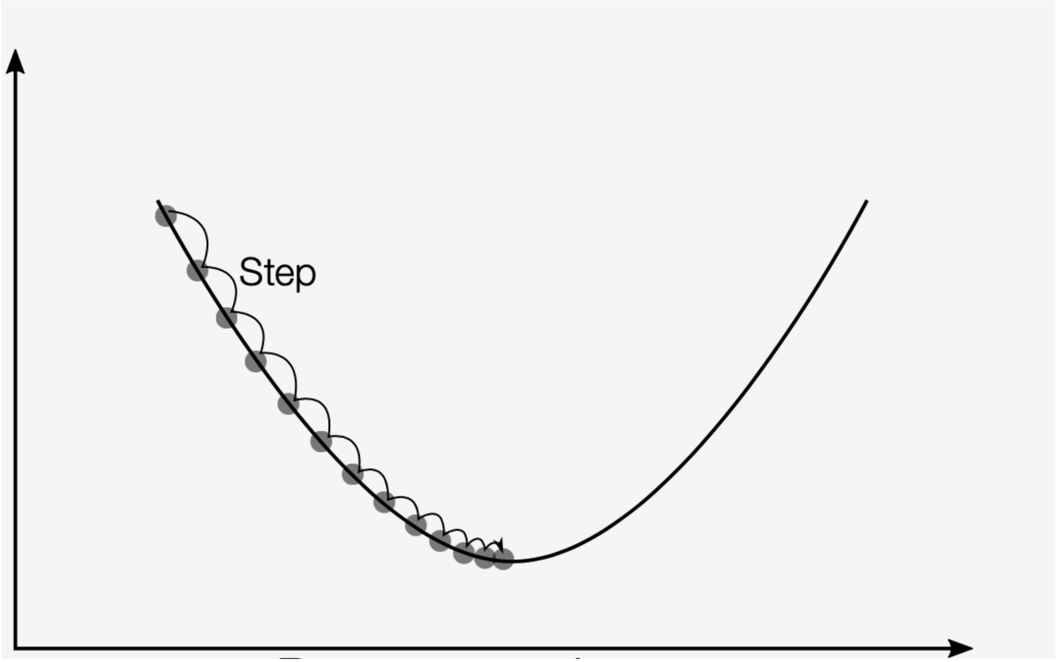
Градиентный спуск. Идем в минимум функции потерь

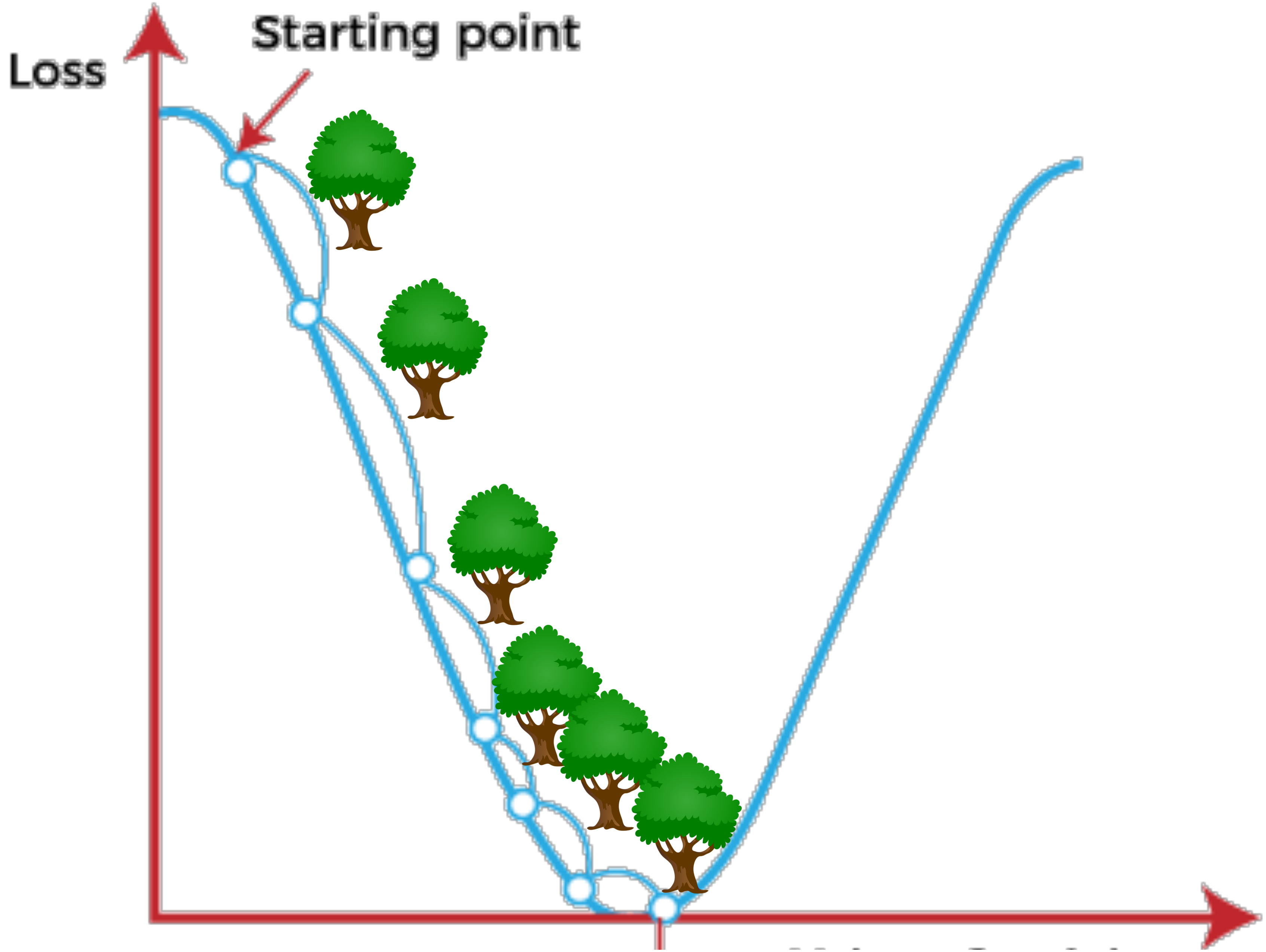
$$MSE(\hat{y}) = (y - \hat{y})^2 \rightarrow min$$

$$\hat{y}_{n+1} = \hat{y}_n - \lambda \frac{\partial MSE(\hat{y})}{\partial \hat{y}}, \lambda - const$$

$$\frac{\partial MSE(\hat{y})}{\partial \hat{y}} = \frac{\partial (y - \hat{y})^2}{\partial \hat{y}} = -2(y - \hat{y})$$

$$\hat{y}_{n+1} = \hat{y}_n + \lambda (y - \hat{y})$$




Starting point
Loss

Давайте приблизим градиент деревом

$\hat{y}_{n+1} = \hat{y}_n + \lambda(y - \hat{y})$

Мы не знаем $y - \hat{y}$, откуда нам взять градиент?

$y - \hat{y} \approx tree_n(X)$ - давайте его предскажем на каждом шаге

Число шагов == числу деревьев.

$\hat{y}_{n+1} = \hat{y}_n + \lambda \cdot tree_n(X)$

Чуть более формально то же самое

Хочется уметь делать бустинг для любого лосса

$$c_k, f_k = argmin_{c,f} \sum_{i=1}^{N} L(F_{k-1}(x_i) + cf(x_i), y_i)$$

Рассмотрим $L(F_{k-1}(x_i) + d_i, y_i)$. Какое взять d_i, чтобы уменьшить значение функции ошибки на x_i? Антиградиент !

То есть если $f_k(x_i) \approx \left.\dfrac{\partial L(x, y)}{\partial x}\right|_{x=F_{k-1}(x_i)}$

то ошибка будет уменьшаться

Алгоритм градиентного бустинга

1. Инициализировать f_0
2. Обучить f_k на выборке $(x_i, -\frac{\partial L}{\partial F}(x_i, F_{k-1}(x_i)))_{i=1}^{N}$, например, на MSE функции потерь. Посчитали антиградиент
3. Найти шаг $c_k = argmin_c \sum_{i=1}^{N} L(F_{k-1}(x_i) + cf_k(x_i), y_i)$
4. Достроить ансамбль $F_k = F_{k-1} + c_k f_k$
5. Повторять 2) 3) 4) M итераций

Найдем размер шага

Можно честно решить задачу одномерной оптимизации:

$$argmin_c \sum_{i=1}^{N} L(F_{k-1}(x_i) + cf_k(x_i), y_i)$$

На практике можно сделать перебор.

Используют shrinkage $F_k = F_{k-1} + \eta c_k f_k, \eta \in (0,1]$

Можно вообще шаг сделать константным:

$F_k = F_{k-1} + \eta f_k$, где η - learning rate

Вопрос на собеседовании

А что будет, если в качестве базовой модели использовать линейную регрессию?

XGBoost

Достоинства

- Быстро учится
- Быстро инферится
- Высокое качество на табличных данных

Сейчас есть разные имплементации XGBoost (библиотеки xgboost, lightgbm, catboost) в проде — норма.

Приблизим функцию потерь в точке

$$f_k = argmin_f \sum_{i=1}^{N} L(F_{k-1}(x_i) + f(x_i), y_i)$$

Разложим в ряд Тейлора до 2-го порядка:

$$\sum_{i=1}^{N} L(F_{k-1}(x_i) + f(x_i), y_i) \approx \sum_{i=1}^{N} L(F_{k-1}(x_i), y_i) + g_i f(x_i) + \frac{1}{2} h_i f(x_i)^2$$

$$g_i = \frac{\partial L(F_{k-1}(x_j), y_i)}{\partial F_{k-1}(x_j)}, h_i = \frac{\partial^2 L(F_{k-1}(x_j), y_i)}{\partial F_{k-1}(x_j)^2}$$

Регуляризация

Наш функционал мерит качество на трейне, но никак не штрафует за сложность

Делаем регуляриацию на число листьев и на значения в них

Дерево решений: $b(x) = \sum_{j=1}^{J} b_j I[x \in R_k]$

$$\sum_{i=1}^{N} g_i f(x_i) + \frac{1}{2} h_i f(x_i)^2 + \gamma T + \lambda \sum_{j=1}^{J} b_j^2$$

Этот функционал потом берется как критерий информативности и деревья строятся **непосредственного** под него

В качестве критерия останова смотрится значение этого функционала.

Строим дерево

Перепишем через сумму по листам дерева

$$L = \sum_{i=1}^{N} g_i \sum_{j=1}^{N} b_j I[x_i \in R_j] + \frac{1}{2} h_i \sum_{j=1}^{J} b_j^2 I[x_i \in R_j] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^{J} b_j^2$$

$$= \sum_{j=1}^{J} [b_j \, (\sum_{i \in R_j} g_j) + \frac{1}{2} b_j^2 \sum_{i \in R_j} (h_i + \lambda)] + \gamma T \to min_{b_j}$$

Можно найти минимум по каждому b_j независимо

$$b_j = \frac{G_j}{H_j + \lambda}, \, G_j = \sum_{i \in R_j} g_i \quad H_j = \sum_{i \in R_j} h_i$$

Подставляя получим ошибку дерева с оптимальными коэф.

$$H(b) = -\frac{1}{2} \sum_{j=1}^{J} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

Можно ее использовать как критерий для построения дерева

Критерий информативности для сплита в дереве

$$Criterion_{split} = \frac{1}{2}[\frac{G_{left}^2}{H_{left} + \lambda} + \frac{G_{right}^2}{H_{right} + \lambda} - \frac{G^2}{H + \lambda}] - \gamma$$

- Учитывает ошибку на трейне
- Учитывает регуляризацию на значение градиента в листах
- Учитывает регуляризацию на число листьев

Если < 0, то останавливаемся

Approximate split finding (Histogram based algorithm)

А давайте будем проходиться не по всем возможным значениям фичи, а только по квантилям.

Т.е. перебирали для сплита до N примеров и зависели от числа уникальных значений.

Теперь перебираем фиксированное число квантилей. К примеру, 256

Получается чуть менее точно, зато быстро

Approximate split finding

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i,\ H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ *to* m **do**
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ *to* m **do**
  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.
  Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ *to* m **do**
  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max score only among proposed splits.

XGBoost

- Дважды дифференцируемый лосс
- Регуляризация на: значения в листьях, число листьев
- Ищем лучший сплит по квантилям фичи
- В критерии информативности есть и лосс, и регуляризация

Сейчас есть разные имплементации XGBoost (библиотеки xgboost, lightgbm, catboost) в проде — норма.

Вопрос на собеседовании

А как в бустинге можно измерять важность признаков?

LightGBM

Оптимизации над XGBoost

Идеи

- новый метод семплирования данных на основе градиента (GOSS)
- новый метод работы с разреженными фичами (EBF)

Улучшения

- оптимизации в коде, бэкенд на плюсах
- может работать на gpu
- больше лоссов / задач / гиперпараметров и т.д.
- удобный интерфейс и т.д.

Gradient Based One Side Sampling (GOSS)

А что если при построении нового дерева больше внимания обращать на объекты, где большой градиент?

Ибо маленький градиент не принесет чего-то значимого в дерево

По умолчанию отключен, если что

Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: $loss$: loss function, L: weak learner
models $\leftarrow$ { }, fact $\leftarrow \frac{1-a}{b}$
topN $\leftarrow$ a $\times$ len(I) , randN $\leftarrow$ b $\times$ len(I)
for $i = 1$ ***to*** d **do**
  preds $\leftarrow$ models.predict(I)
  g $\leftarrow$ $loss$(I, preds), w $\leftarrow$ {1,1,...}
  sorted $\leftarrow$ GetSortedIndices(abs(g))
  topSet $\leftarrow$ sorted[1:topN]
  randSet $\leftarrow$ RandomPick(sorted[topN:len(I)], randN)
  usedSet $\leftarrow$ topSet + randSet
  w[randSet] $\times=$ fact $\triangleright$ Assign weight $fact$ to the small gradient data.
  newModel $\leftarrow$ L(I[usedSet], $-$ g[usedSet], w[usedSet])
  models.append(newModel)

Exclusive Feature Bundling (EFB)

А что если несколько разреженных фичи схлопнуть в одну?

- Не только категориальные фичи
- Если пересечений мало, тоже можно схлопнуть
- near-lossless, по описанию

feature1	feature2	feature_bundle
0	2	6
0	1	5
0	2	6
1	0	1
2	0	2
3	0	3
4	0	4

Exclusive Feature Bundling (EFB) формально

Algorithm 3: Greedy Bundling

Input: F: features, K: max conflict count
Construct graph G
searchOrder $\leftarrow$ G.sortByDegree()
bundles $\leftarrow \{\}$, bundlesConflict $\leftarrow \{\}$
for i ***in*** *searchOrder* **do**
    needNew $\leftarrow$ True
    for $j = 1$ ***to*** *len(bundles)* **do**
        cnt $\leftarrow$ ConflictCnt(bundles[j],F[i])
        if *cnt + bundlesConflict[i]* $\leq K$ **then**
            bundles[j].add(F[i]), needNew $\leftarrow$ False
            break
    if *needNew* **then**
        Add $F[i]$ as a new bundle to *bundles*
Output: *bundles*

Algorithm 4: Merge Exclusive Features

Input: $numData$: number of data
Input: F: One bundle of exclusive features
binRanges $\leftarrow \{0\}$, totalBin $\leftarrow 0$
for f ***in*** F **do**
    totalBin += f.numBin
    binRanges.append(totalBin)
newBin $\leftarrow$ new Bin(numData)
for $i = 1$ ***to*** $numData$ **do**
    newBin[i] $\leftarrow 0$
    for $j = 1$ ***to*** *len(F)* **do**
        if *F[j].bin[i]* $\neq$ *0* **then**
            newBin[i] $\leftarrow$ F[j].bin[i] + binRanges[j]
Output: $newBin$, $binRanges$

CatBoost

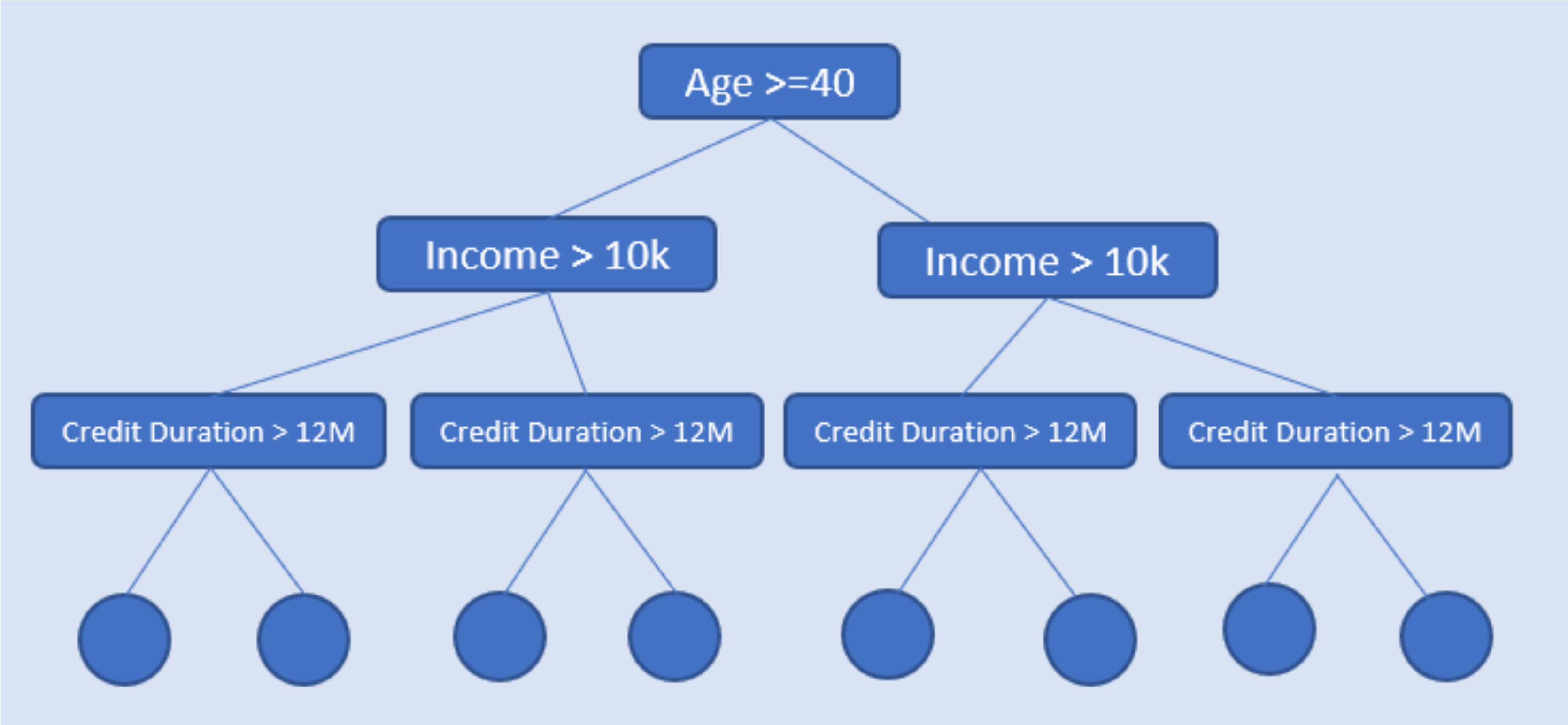
Особенности

Самый свежий (2017)

Отечественный производитель (Яндекс)

- Oblivious trees (симметричные деревья)
- Target Encoding, где постарались избежать ликов данных
- Хорошо работает с категориальными фичами с высокой cardinality
- Может работать с текстами и эмбеддингами
- Много оптимизаций и улучшений

Oblivious Tree



Обычно, деревьев нужно больше, чем в LightGBM: сотни, тысячи

Ordered Target Statistics

А что если превратить категориальную фичу в числовую по такой формуле

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + a\,p}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}.$$

D_k - сабсет, в котором все объекты до i-го

Пример Ordered Target Statistics

Index Order	Feature X		Target Value (y)	p (Average prior Y)	Ordered TS Encode for X
I1	A	...	1	0	0 + 0.1*0 / 0 + 0.1 = **0**
I2	B	...	1	1	0 + 0.1*1 / 0 + 0.1 = **1**
I3	C	...	1	1	0 + 0.1*1 / 0 + 0.1 = **1**
I4	A	...	0	1	1*1+0.1*1 / 1 + 0.1 = **1**
I5	B	...	1	0.75	1*1 + 0.1*0.75 / 1+0.1 = **0.977**
I6	C	...	1	0.8	1*1 + 0.1*0.8 / 1+0.1 = **0.982**
I7	B	...	0	0.83	2*1 + 0.1*0.83 / 2+0.1 = **0.992**
I8	C	...	1	0.714	2*1 + 0.1*0.714 / 2+0.1 = **0.986**
I9	C	...	0	0.75	3*1 + 0.1*0.75 / 3+0.1 = **0.992**
I10	C	...	1	0.667	3*1 + 0.1*0.667 / 4+0.1 = **0.748**

Еще пачка приятоk

- Несколько разновидностей вероятностного bootstrap sampling, чтобы семплировать реже то, на чем часто учимся
- Встроенный таргет энкодинг для эмбеддингов (LDA / KNN)
- Встроенный препроцессинг и BoW для текстов
- Может работать на GPU
- Есть версия для Spark
- Есть много для ранжирования
- Много настроек / лоссов / возможностей. Тюнится почти все
- Overfitting Detector
- Качество обычно немного выше lightgbm или xgboost
- Быстрый инференс
- ...

Спасибо! Задавайте ваши вопросы :)

Алексей Ярошенко
t.me/yaroshenko