# Прикладные задачи анализа данных

## Градиентный бустинг: теория

**Дьяконов А.Г.**

**Московский государственный университет**
**имени М.В. Ломоносова (Москва, Россия)**

# Идея градиентного бустинга

**Задача регрессии**

$$(x_i, y_i)_{i=1}^m$$

**дифференцируемая функция ошибки**

$$L(y, a)$$

**уже есть алгоритм** $a(x)$ **строим** $b(x)$**:**

$$a(x_i) + b(x_i) = y_i, i \in \{1,2,\ldots,m\}.$$

**Надо:**

$$\sum_{i=1}^m L(y_i, a(x_i) + b(x_i)) \to \min,$$

**а не**

$$\sum_{i=1}^m L(y_i - a(x_i), b(x_i)) \to \min$$

# Проблема

**Задача**

$$\sum_{i=1}^{m} L(y_i, a(x_i) + b(x_i)) \to \min$$

**может не решаться аналитически**

$$F = \sum_{i=1}^{m} L(y_i, a(x_i) + b_i) \to \min_{(b_1,\ldots,b_m)}$$

**Функция** $F(b_1,\ldots,b_m)$ **убывает в направлении антиградиента, поэтому выгодно считать**

$$b_i = -L'(y_i, a(x_i)), \quad i \in \{1,2,\ldots,m\},$$

**новая задача для настройки второго алгорпитма:**

$$(x_i, -L'(y_i, a(x_i)))_{i=1}^{m}.$$

# Алгоритм градиентного бустинга (примитивный вариант)

- **Строим алгоритм в виде**

$$a_n(x) = \sum_{t=1}^{n} b_t(x),$$

**для удобства можно даже считать, что** $a_0(x) \equiv 0$.

- **Пусть построен** $a_t(x)$**, тогда обучаем алгоритм** $b_t(x)$ **на выборке**

$$(x_i, -L'(y_i, a_t(x_i)))_{i=1}^{m}$$

- $a_{t+1}(x) = a_t(x) + b_t(x)$.

**Вот почему называется градиентный бустинг…**

# Частный случай

## Регрессия с СКО

$$L(y,a) = \frac{1}{2}(y-a)^2$$

$$L'(y,a) = -(y-a)$$

**Задача для настройки следующего алгоритма**

$$(x_i,\ y_i - a_t(x_i))_{i=1}^m$$

**т.е. очень логично:**

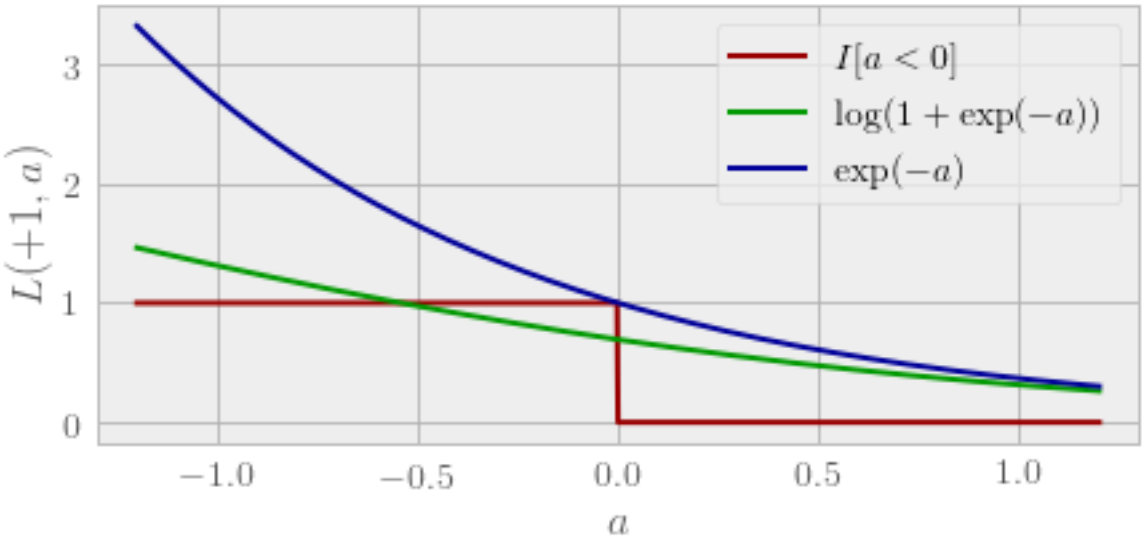
**настраиваемся на невязку!**

# Частный случай

## Классификация на два класса

**надо найти дифференцируемую функцию ошибки...**

- **предполагаем, что алгоритм выдаёт вещественные значения**
- **делаем функцию похожей на «совпадение»**



# Частный случай

## Классификация на два класса

**Логистическая функция ошибки:**

$$L(y,a) = \log(1+e^{-y\cdot a}),\ a \in (-\infty,+\infty),\ y \in \{-1,+1\},$$

$$L'(y,a) = -\frac{y}{1+e^{-y\cdot a}}.$$

**Функция ошибки типа Adaboost:**

$$L(y,a) = e^{-y\cdot a},\ a \in (-\infty,+\infty),\ y \in \{-1,+1\},$$

$$L(y,a) = -ye^{-y\cdot a}.$$

# Итерация градиентного бустинга

**Как решать задачу**

$$(x_i, -L'(y_i, a_t(x_i)))_{i=1}^{m}?$$

**Любым простым методом!**

**Мы уже настраиваемся на нужную функцию ошибки.**

## Проблема

**Шаг в сторону антиградиента**

- **не приводит в локальный минимум (сразу) ⇒ итерации**
- **мы всё равно не можем сделать такой шаг, а лишь шаг по ответам какого-то алгоритма модели ⇒ не нужно стремиться шагать именно туда**

**Дальше решение проблем…**

# Наискорейший спуск

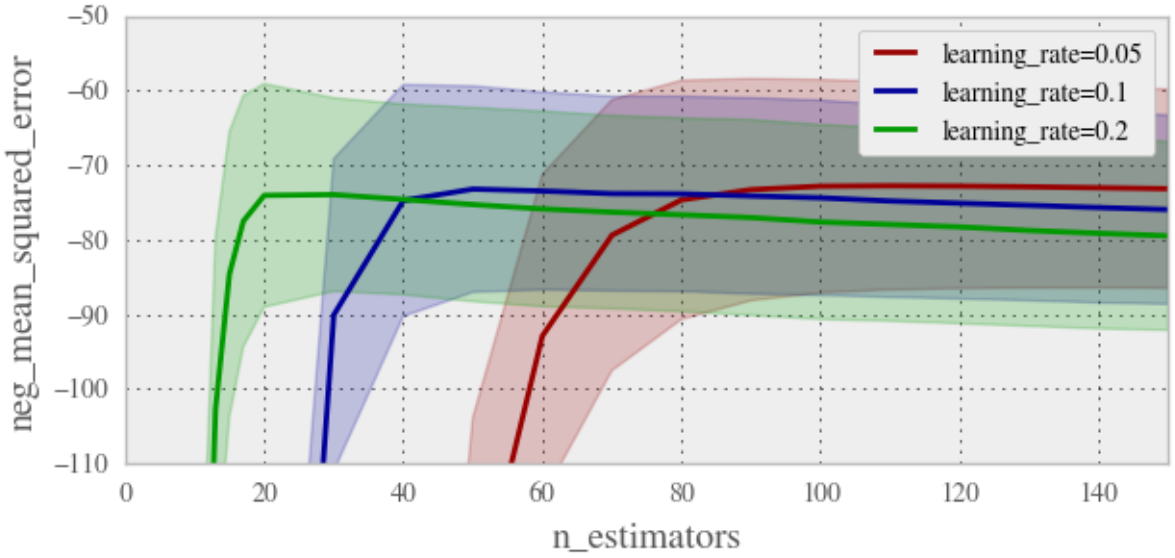
$$\sum_{i=1}^{m} L(y_i, a_t(x_i) + \eta \cdot b_t(x_i)) \to \min_{\eta},$$

$$a_{t+1}(x) = a_t(x) + \eta_t \cdot b_t(x) = \eta_1 \cdot b_1(x) + \ldots + \eta_t \cdot b_t(x)$$

# Эвристика сокращения – Shrinkage

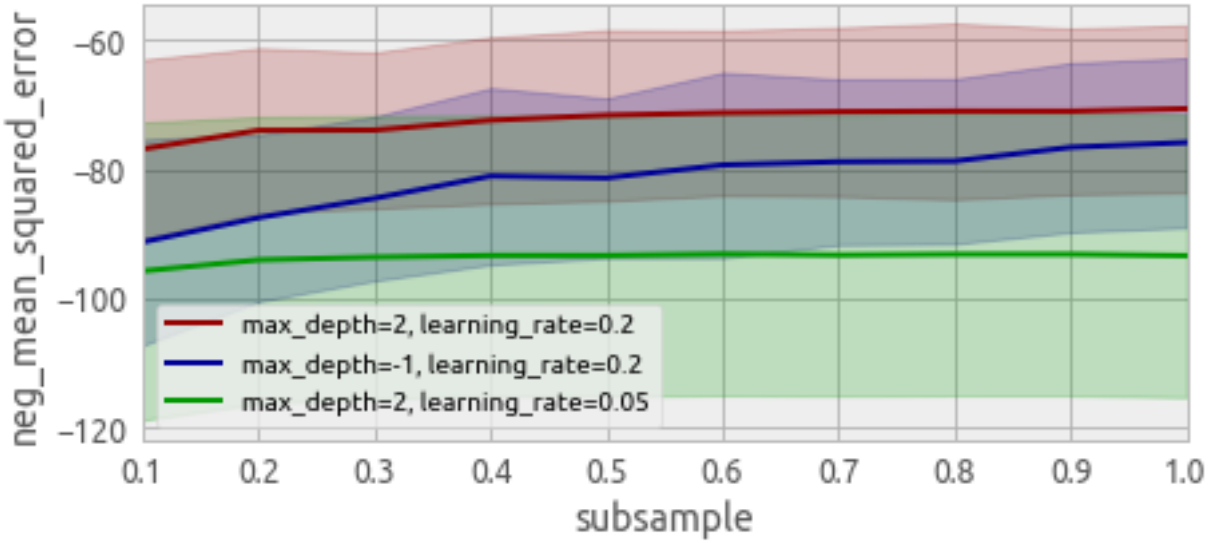
$$a_{t+1}(x) = a_t(x) + \eta \cdot b_t(x),$$

$\eta \in (0, 1]$ – **скорость (темп) обучения (learning rate)**



# Стохастический градиентный бустинг (Stochastic gradient boosting)

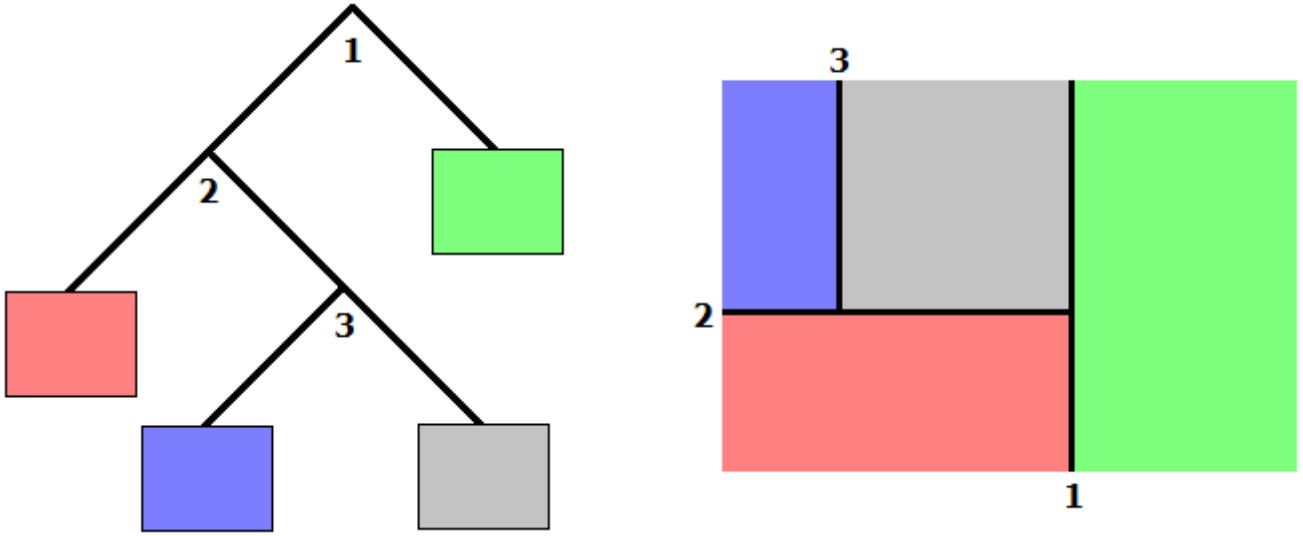
## Идея бэгинга Бреймана



# TreeBoost – градиентный бустинг над деревьями

**Решающее дерево:**

$$b(x) = \sum_j \beta_j I[x \in X_j].$$



# TreeBoost – градиентный бустинг над деревьями

## Наша основная задача

$$\sum_{i=1}^{m} L(y_i, a(x_i) + \sum_j \beta_j I[x \in X_j]) \to \min$$

## Разбиваем по областям:

$$\sum_{x_i \in X_j} L(y_i, a(x_i) + \beta_j) \to \min_{\beta_j}.$$

# Продвинутые методы оптимизации

**Наша основная задача**

$$F = \sum_{i=1}^{m} L(y_i, a(x_i) + b_i) \to \min_{(b_1,\ldots,b_m)},$$

**заметим, что**

$$F = \sum_{i=1}^{m} L(y_i, a(x_i) + b_i) \approx$$

$$\sum_{i=1}^{m} \left[ L(y_i, a(x_i)) + L'(y_i, a(x_i)) \cdot b_i + \frac{1}{2} L''(y_i, a(x_i)) \cdot b_i^2 \right]$$

**(частные производные по второму аргументу функции ошибки)**

# Продвинутые методы оптимизации

$$\sum_{i=1}^{m}\left[ g_i b_i + \frac{1}{2} h_i b_i^2 \right] \to \min,$$

$$g_i = L(y_i, a(x_i))',$$

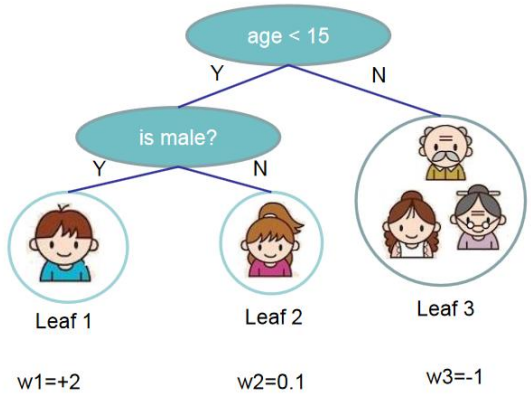
$$h_i = L''(y_i, a(x_i)).$$

**Сделаем оптимизацию с регуляризацией.**

# Продвинутые методы оптимизации

**Пусть дерево $b(x)$ делит пространство объектов на $T$ областей $X_1,\ldots,X_T$, в каждой области $X_j$ принимает значение $\beta_j$.**

$$\Phi = \sum_{i=1}^{m}\left[ g_i b_i + \frac{1}{2} h_i b_i^2 \right] + \gamma T + \lambda \frac{1}{2}\sum_{j=1}^{T} \beta_j^2 \to \min$$




$$\Omega = \gamma 3 + \tfrac{1}{2}\lambda(4 + 0.01 + 1)$$

# Продвинутые методы оптимизации

$$\Phi = \sum_{j=1}^{T}\left[\sum_{x_i \in X_j}\left[g_i\beta_j + \frac{1}{2}h_i\beta_j^2\right] + \lambda\frac{1}{2}\beta_j^2\right] + \gamma T =$$

$$= \sum_{j=1}^{T}\left[\beta_j \sum_{x_i \in X_j} g_i + \frac{1}{2}\beta_j^2\left(\sum_{x_i \in X_j} h_i + \lambda\right)\right] + \gamma T$$

**Приравнивая производную к нулю:**

$$\beta_j = -\frac{\sum_{x_i \in X_j} g_i}{\sum_{x_i \in X_j} h_i + \lambda}.$$

# Продвинутые методы оптимизации

**Минимальное значение (при фиксированной структуре дерева)**

$$\Phi_{\min} = -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum\limits_{x_i \in X_j} g_i\right)^2}{\sum\limits_{x_i \in X_j} h_i + \lambda} + \gamma T.$$

**Можно использовать при построении дерева для его оценки.**

# Современные реализации градиентного бустинга

1. sklearn.ensemble.

GradientBoostingRegressor

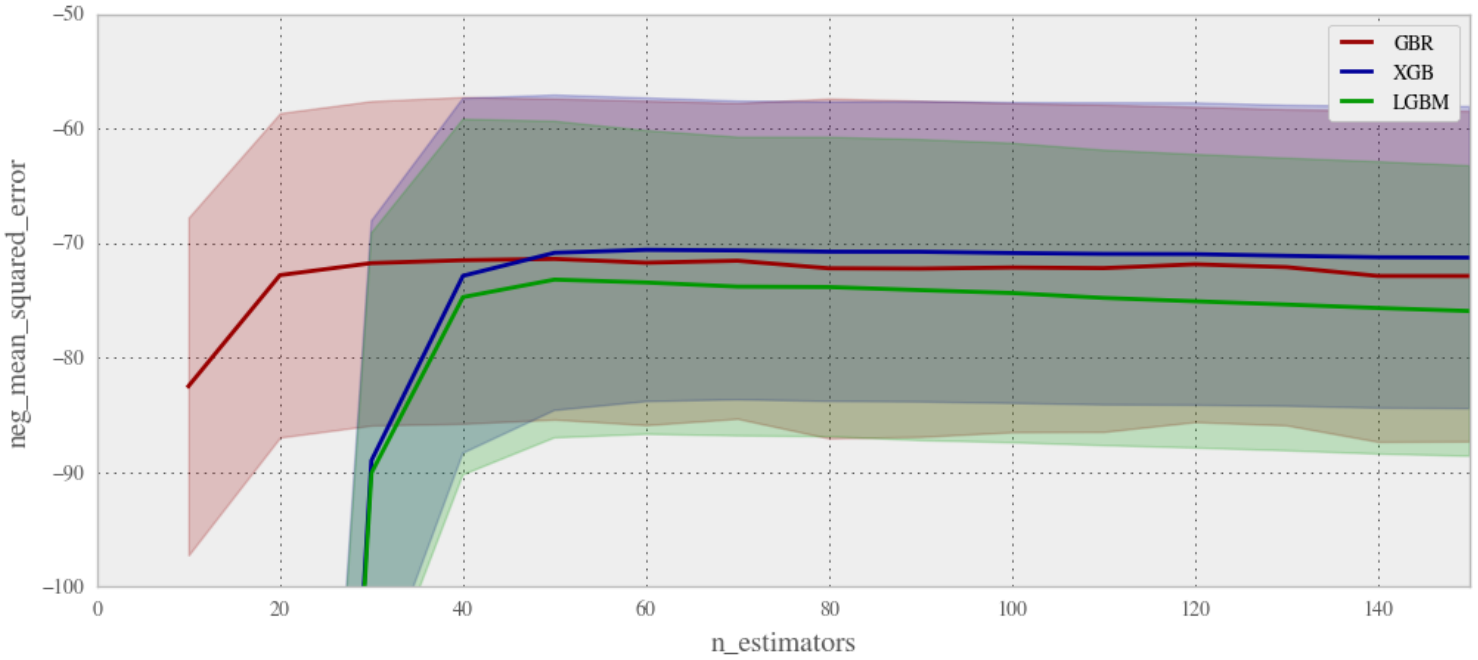
GradientBoostingClassifier

2. XGBoost (eXtreme Gradient Boosting)

https://github.com/dmlc/xgboost

3. LightGBM, Light Gradient Boosting Machine

https://github.com/Microsoft/LightGBM



# Параметры градиентного бустинга

- `objective` – **параметры определяющие, какая задача решается и в каком формате будет ответ**
- `eval_metric` – **значения какой функции ошибки смотреть на контроле** (как правило, задание этого параметра не означает, что эту функцию будем минимизировать при настройке бустинга)

## Параметры, определяющие тип бустинга

- `booster` – **какой бустинг проводить: над решающими деревьями или линейный**
- **способы построения деревьев** (`grow_policy` – порядок построения дерева: на следующем шаге расщеплять вершину, ближайшую к корню, или на которой ошибка максимальна)

**параметр «распределение» см. дальше**

## Основные параметры:

- `eta / learning_rate` – темп (скорость) обучения
- `num_iterations / n_estimators` – число итераций бустинга
- `early_stopping_round` – если на отложенном контроле заданная функция ошибки не уменьшается такое число итераций, обучение останавливается

## Параметры ограничивающие сложность дерева:

- `max_depth` – максимальная глубина
- `max_leaves / num_leaves` – максимальное число вершин в дереве
- `gamma / min_gain_to_split` – порог на уменьшение функции ошибки при расщеплении в дереве
- `min_data_in_leaf` – минимальное число объектов в листе
- `min_sum_hessian_in_leaf` – минимальная сумма весов объектов в листе, минимальное число объектов, при котором делается расщепление

## Параметры формирования подвыборок

- `subsample / bagging_fraction` – **какую часть объектов обучения использовать для построения одного дерева**
- `colsample_bytree / feature_fraction` – **какую часть признаков использовать для построения одного дерева**
- `colsample_bylevel` – **какую часть признаков использовать для построения расщепления в дереве**

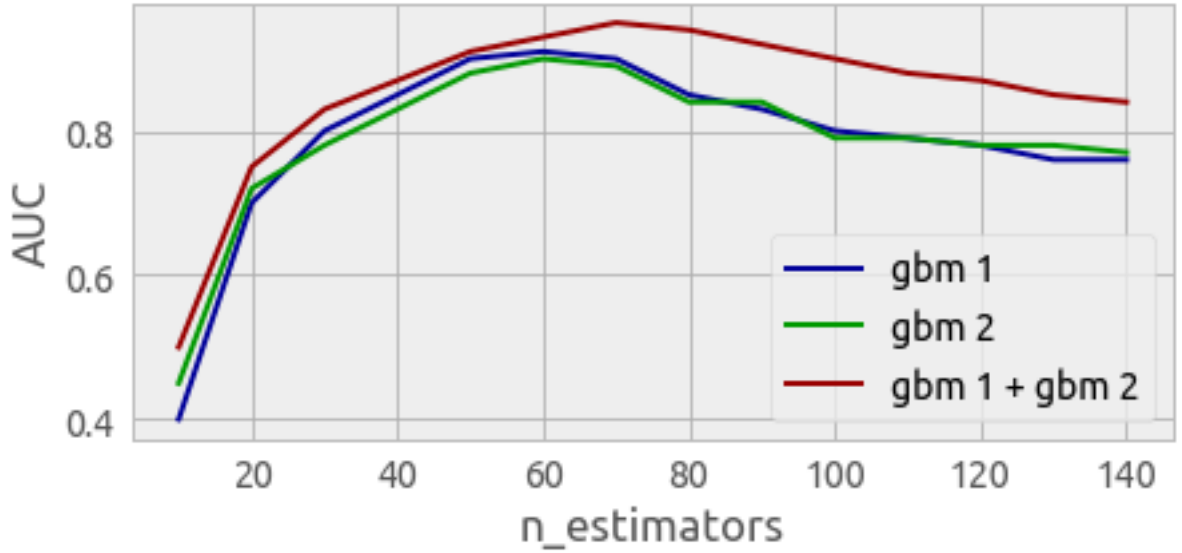
## Параметры регуляризации:

- `lambda / lambda_l2` **(L2)**
- `alpha / lambda_l1` **(L1)**

## Параметры которые помогают обучать бустинг быстрее:

- **число используемых потоков**
- **CPU / GPU**
- **хранить модель в ОЗУ**
- **метод поиска расщепления**

# Сумма бустингов




**Качество может улучшиться,**
**но оптимальные параметры меняются!**