Lecture 2

Корбут Даниил
Deep Learning Research Engineer, Insilico Medicine

telegram: @rtriangle
vk: rtriangle
email: korbut.daniel@gmail.com

План занятия

- Описание итоговых проектов
- Решающие деревья
- Ансамбли деревьев
- Xgboost
- Lightgbm
- Catboost
- Сравнение Xgboost, Lightgbm, Catboost

Итоговые проекты

1. “Рабочий” проект
2. Соревнование на kaggle
3. “Улучшенное” домашнее задание

1. “Рабочий проект”

1) Описание задачи: данные и их объём, метрики
2) Ограничения: по памяти, по времени, …
3) Проведённые эксперименты
4) Итоговый алгоритм
5) Идеи для улучшения

https://www.picbon.org/tag/ulkovarasto

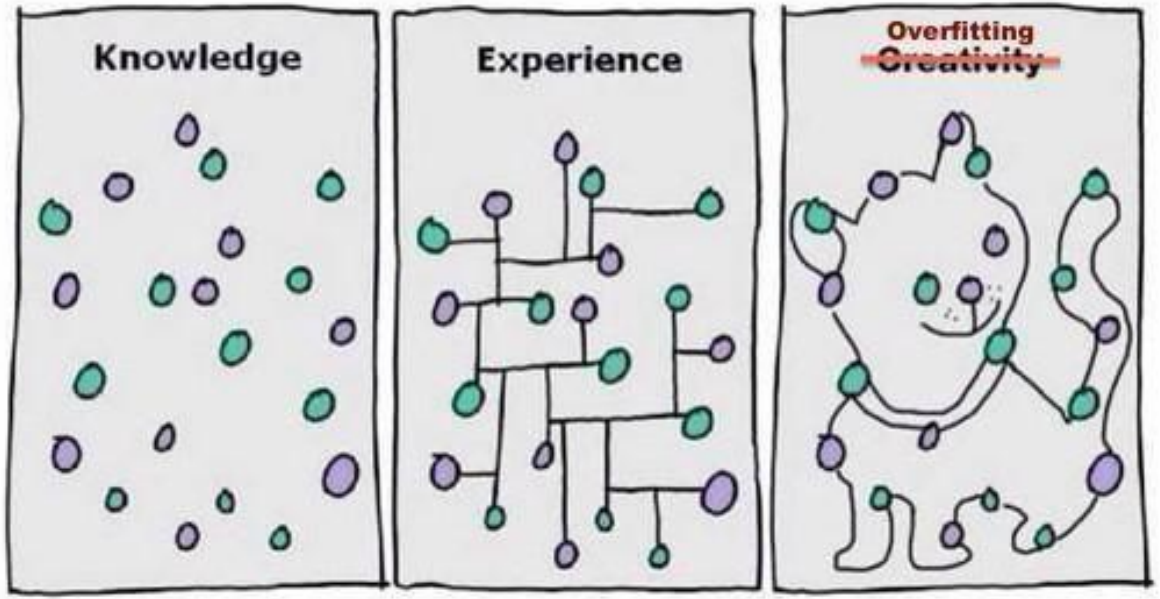
2. Соревнование на kaggle

1. Описание данных
2. .ipynb с экспериментами
   a. exploratory data analysis
   b. генерация признаков
   c. разбиение train-dev
   d. эксперименты с моделями
   e. финальный сабмит
   f. идеи для улучшения

http://www.shivambansal.com/blog/kaggle-bot/

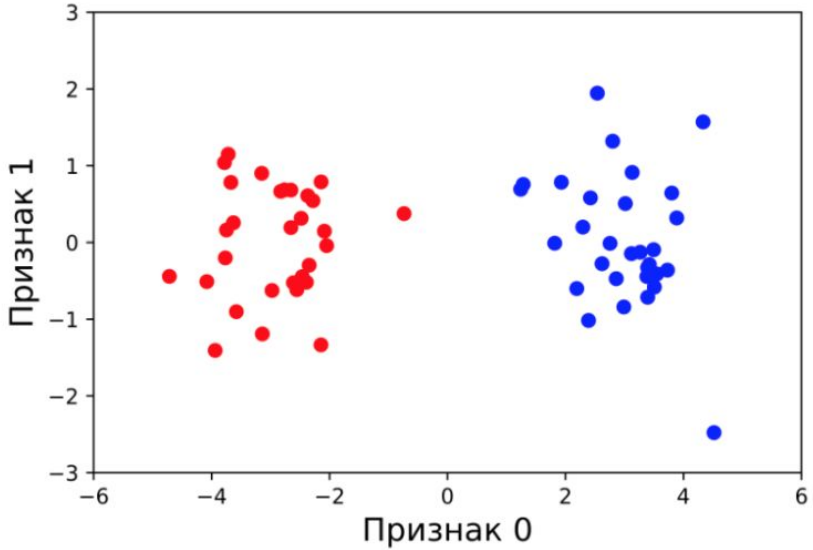
3. “Улучшенное” домашнее задание

1. Идея улучшения
2. Эксперименты
3. ...



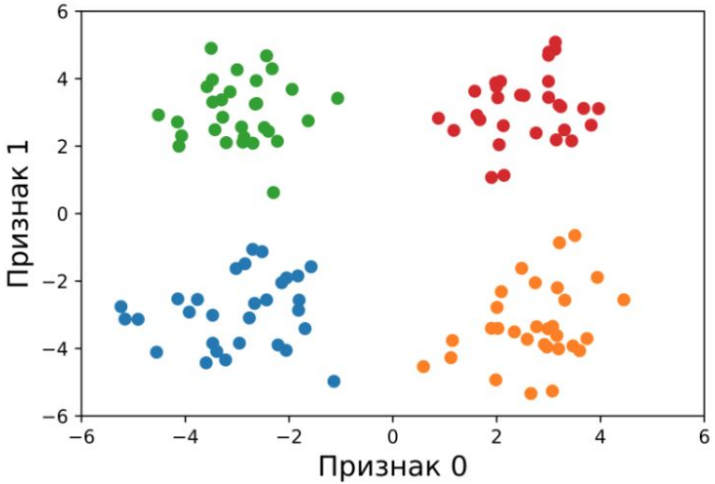
https://twitter.com/gagan_s

Решающие деревья



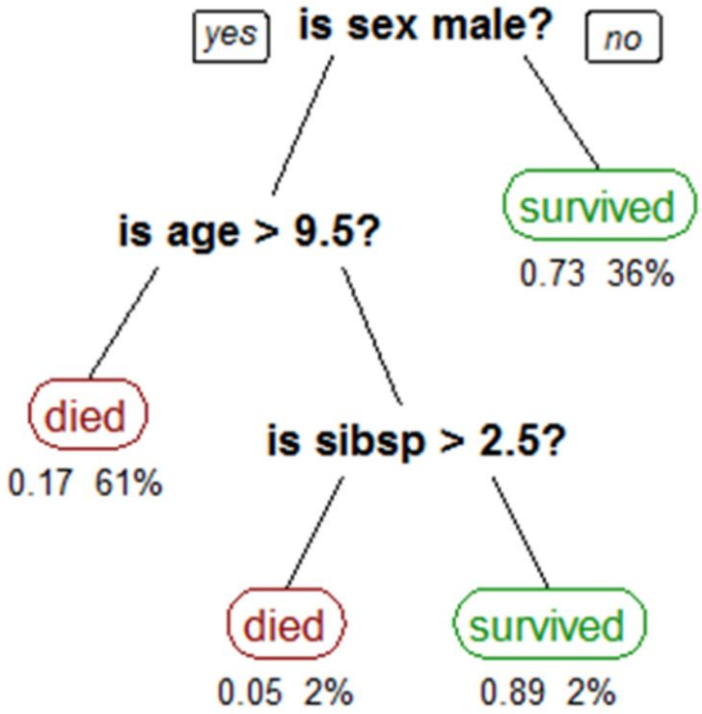

```
def classify(X):
    if X[0] < 0:
        return "red"
    else:
        return "blue"
```

Решающие деревья

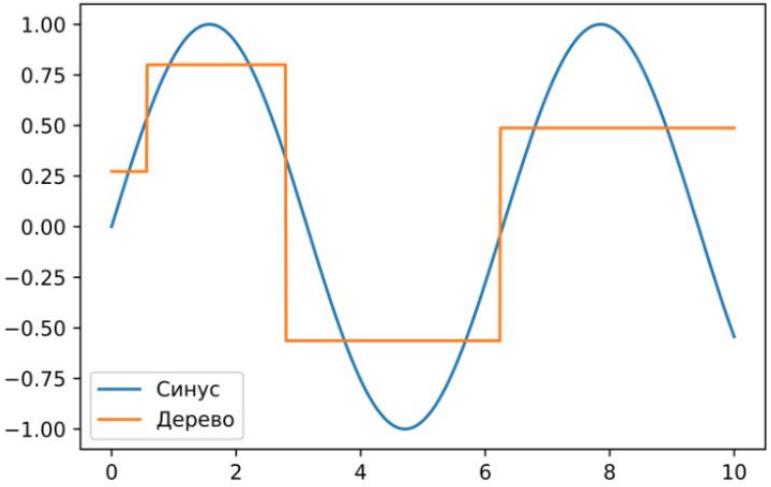


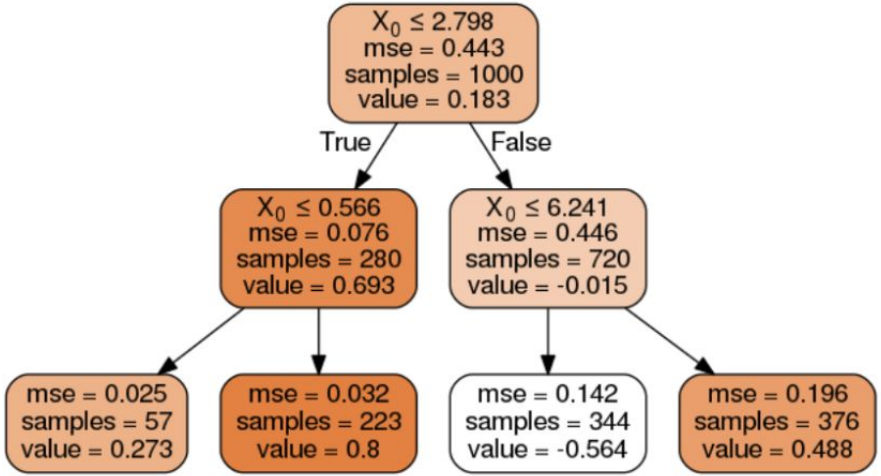

```
def classify(X):
    if X[0] < 0:
        if X[1] < 0:
            return "blue"
        else:
            return "green"
    else:
        if X[1] > 0:
            return "red"
        else:
            return "orange"
```

Решающие деревья

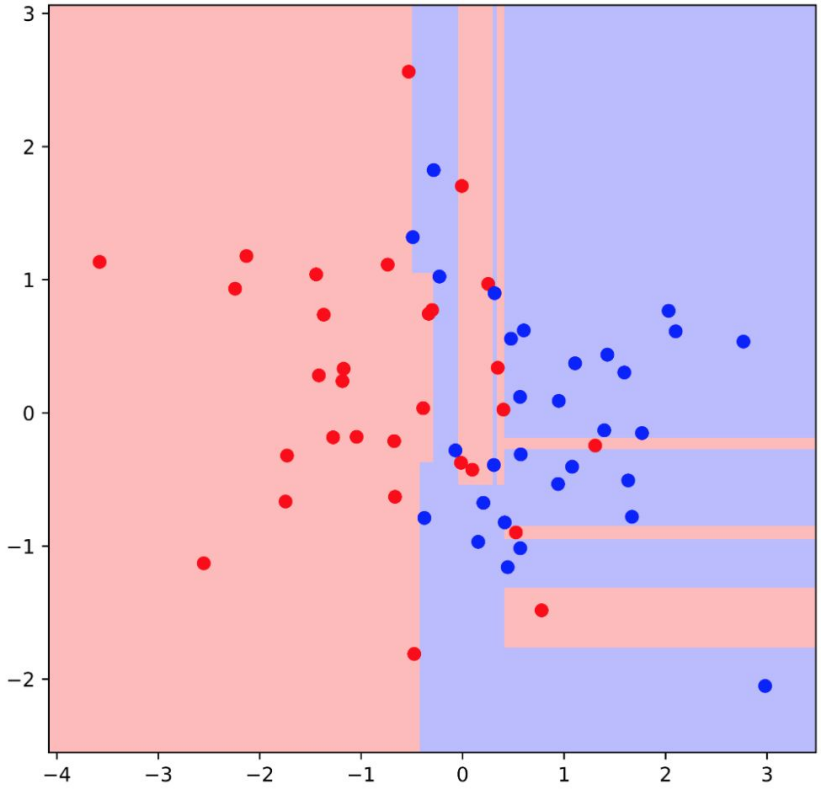


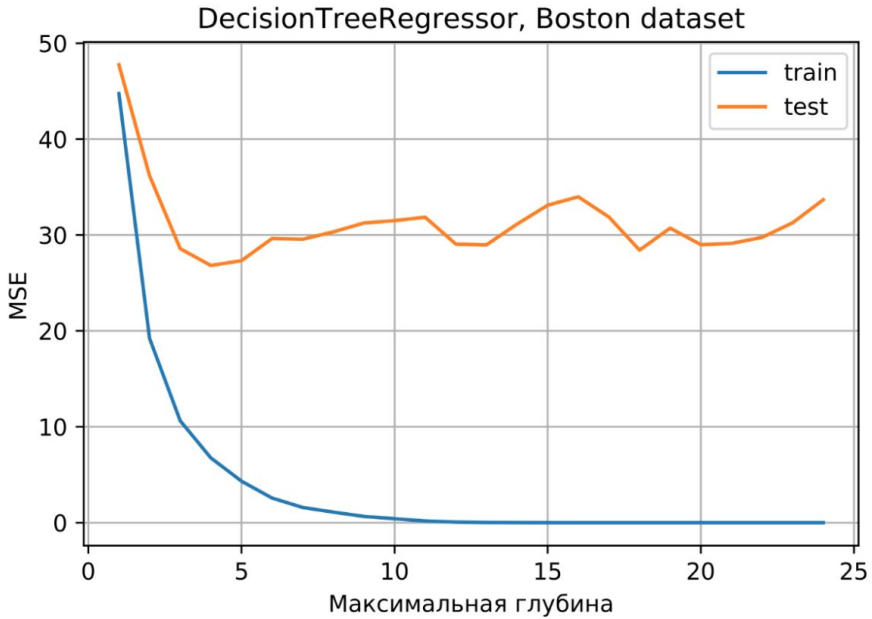
Решающие деревья





Решающие деревья





Решающие деревья

Индекс неоднородности - величина, оценивающая неоднородность выборки

Для задачи регрессии

$$\text{MSE: } H(Y) = \frac{1}{|Y|} \sum_{i=1}^{|Y|} (y_i - \overline{y})^2$$

$$\overline{y} = \frac{1}{|Y|} \sum_{i=1}^{|Y|} y_i$$

Решающие деревья

Для классификации (P_i - доля класса i в X, L - число классов):

- Энтропия: $H(X) = -\sum_{i=1}^{L} P_i \log P_i$
- Джини: $H(X) = \sum_{i=1}^{L} P_i(1 - P_i)$
- Misclassification: $H(X) = 1 - \max_{1..L} P_i$

Замечание: нужно считать, что $P_i log(P_i) = 0$ при $P_i = 0$

Решающие деревья

Для классификации (P_i - доля класса i в X, L - число классов):

- Энтропия: $H(X) = -\sum_{i=1}^{L} P_i \log P_i$
- Джини: $H(X) = \sum_{i=1}^{L} P_i(1 - P_i)$
- Misclassification: $H(X) = 1 - \max_{1..L} P_i$

Замечание: нужно считать, что $P_i log(P_i) = 0$ при $P_i = 0$

Решающие деревья

Уменьшение среднего индекса неоднородности при разбиении: $I(Q, f, v) = H(Q) - \frac{|L|}{|Q|} H(L) - \frac{|R|}{|Q|} H(R)$

Q - выборка, f - признак, v - порог, L и R - соответсвующие им разбиения выборки Q на две части.

- Будем строить дерево от корня (стартовая вершина) к листьям (вершины, из которых некуда идти)
- В начале в стартовой вершине лежит вся выборка

Решающие деревья

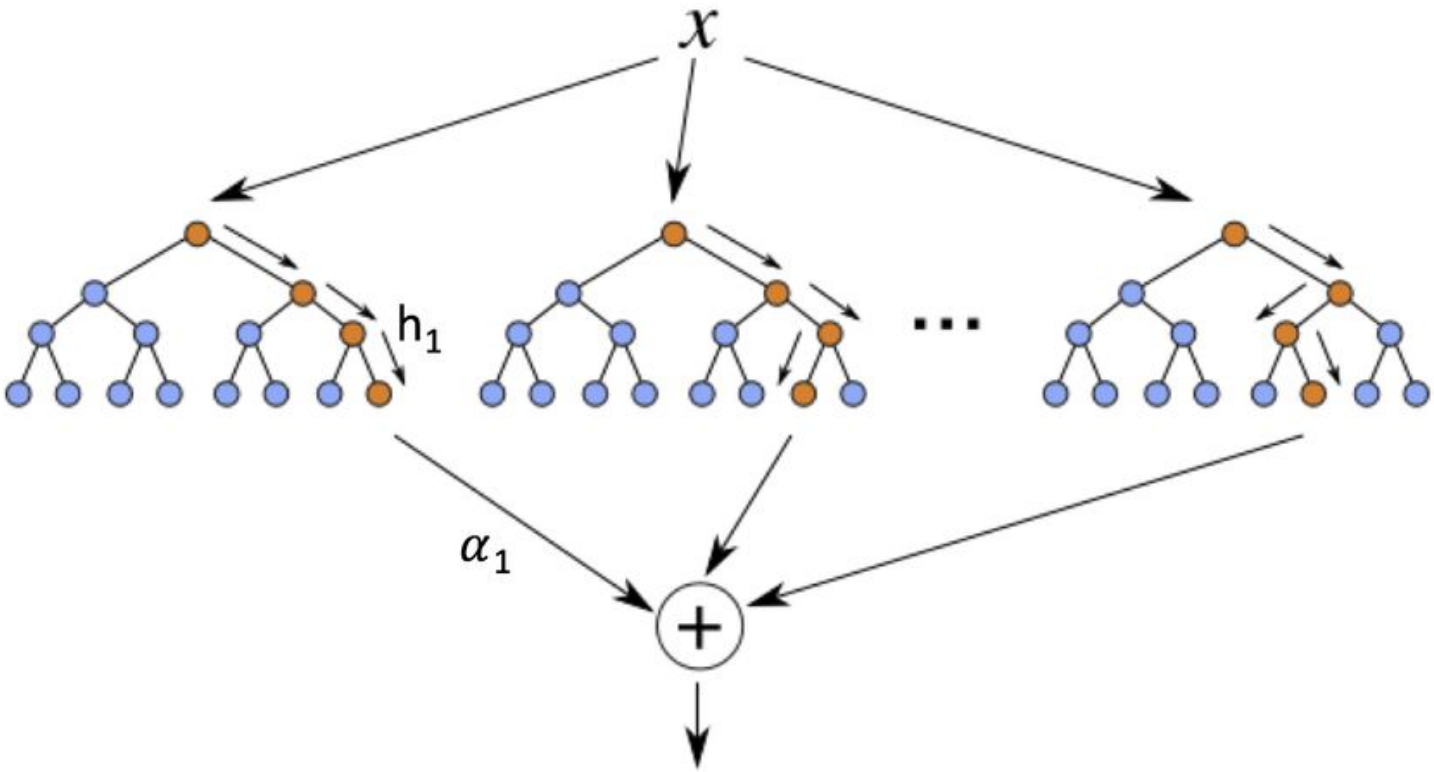
- Если в текущей вершине выполнен критерий останова - ничего не делаем в этой вершине.
- Выбрать f и v так, чтобы $I(Q, f, v)$ было максимально, например, перебрав все признаки и пороги.
- Разделим данную выборку на L и R согласно выбранным f и v, создадим двух потомков текущей вершиы и положим в них L и R соответственно.
- Повторим для каждой дочерней вершины.

Решающие деревья

Таким образом, конкретный метод построения решающего дерева определяется:

1. Видом предикатов в вершинах
2. Критерием информативности
3. Критерием останова
4. Методом обработки пропущенных значений
5. Методом стрижки
6. Работой с категориальными признаками: можно создавать по потомку для каждого значения категориального признака, а можно кодировать средним значением переменной среди элементов данного класса

Ансамбли деревьев



https://www.vaishalilambe.com/blog/data-science-algorithms-random-forest

Ансамбли деревьев

Пусть есть слабые классификаторы, дающие правильный ответ с вероятностью p, не намного большей, чем случайный предсказатель.
Как в таком случае усреднить предсказания?
Проблема: алгоритм построения дерева детерминирован. Т.е., обучаясь на одном датасете с использованием одних и тех же признаков, будем получать одинаковые деревья.
Давайте построим случайный лес из случайных деревьев!

Ансамбли деревьев

Bootstrap:
Пусть дана выборка X из n объектов. Выберем несколько раз, например n, равновероятно случайный объект из выборки X (выбор с повторениями). Выборку составленную из этих объектов назовём bootstrap-выборкой.

Пример:
Из [1,2,3] могут получиться выборки [1,2,2], [3,1,2], [3,3,2] и тд

Ансамбли деревьев

Каждое дерево строится с использованием разных образцов бутстрэпа из исходных данных. Примерно 37% примеров остаются вне выборки бутстрэпа и не используются при построении k-го дерева.

Докажем: Пусть в выборке n объектов. На каждом шаге все объекты попадают в подвыборку с возвращением равновероятно, т.е отдельный объект — с вероятностью 1/n. Вероятность того, что объект не попадёт в выборку все n раз
(1-1/n)^n. При n->inf получаем один из замечательных пределов, а именно 1/e. Таким образом, вероятность попадания конкретного объекта в подвыборку 1-1/e=63%.

Ансамбли деревьев

Получили итоговый алгоритм для построения **случайного леса (Random Forest)** с k деревьями:

- Сгенерируем k bootstrap-подвыборок исходного датасета
- Обучим на каждой выборке своё дерево, но при построении дерева в каждом узле при поиске лучшего разбиения признака используем не все, а m случайных признаков, и ищем разбиение только по ним (m=n^0.5 для классификации, m=n/3 для регрессии)
- Ответ всего алгоритма - класс, за который проголосовало наибольшее количество кандидатов либо среднее значение для классификации и регрессии соответственно

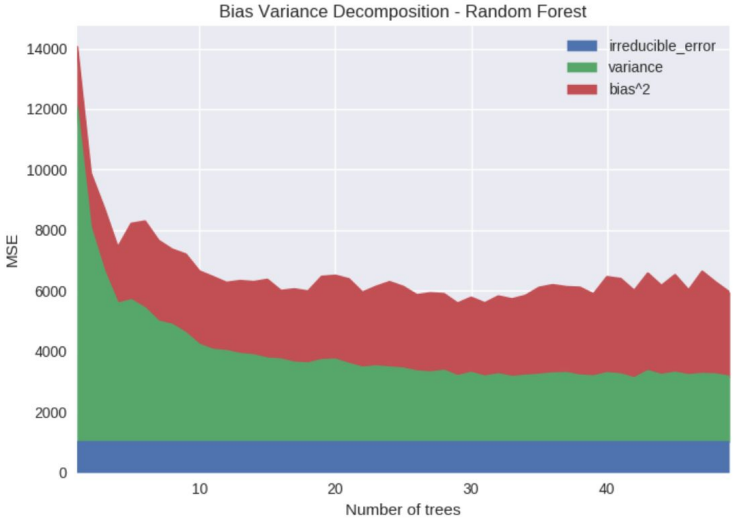
Ансамбли деревьев

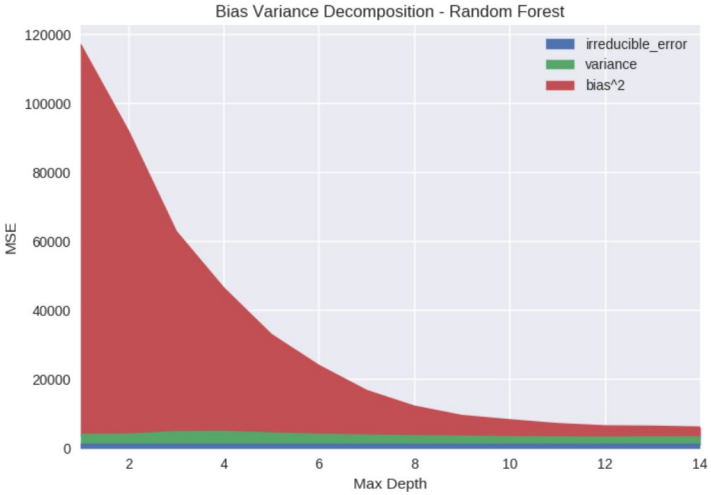
Замечания:

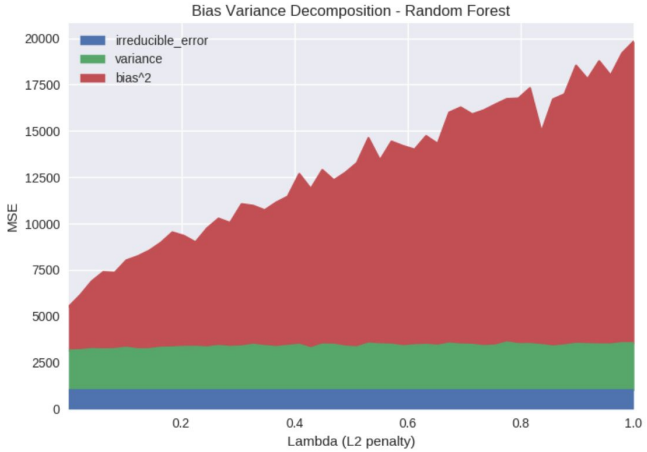
- Строим деревья максимально глубокими для выделения сложных зависимостей, из-за усреднения переобучение не будет мешать (уменьшение variance, const bias)
- Нужно быть аккуратнее с выборками, в которых пропорции классов сильно отличаются
- Случайный лес не переобучается при росте числа деревьев
- “Из коробки” можно получить feature importance для отбора признаков/интерпретации

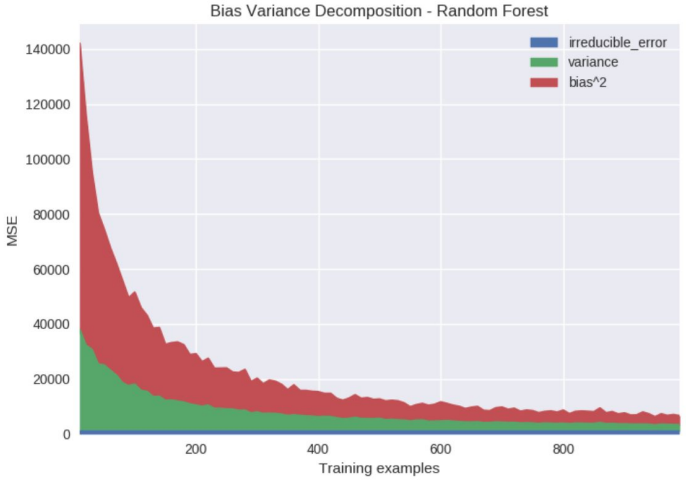
Random forest

https://devblogs.nvidia.com/bias-variance-decompositions-using-xgboost/









Ансамбли деревьев - бустинг

Будем строить алгоритм как $A(x) = \sum_{i=0}^{N} b_i(x)$, где b_i - базовые алгоритмы.

Начальное приближение выбирается произвольно, например, среднее значение целевой переменной.

$b_0(x) = \overline{y}$

Уже построили $A_{N-1}(x) = \sum_{i=0}^{N-1} b_i(x)$

Задача: $\min_{b_N} \sum_{i=1}^{|X|} L(y_i, \sum_{i=0}^{N-1} b_i(x_i) + b_N(x_i))$

Какой сдвиг b_N в пространстве алгоритмов будет давать наискорейшее убывание функции потерь?

Ансамбли деревьев - бустинг

Ответ: такой что $b_N(x_k) = -\frac{\partial L}{\partial a}(y_1, \sum_{i=0}^{N-1} b_i(x_k))$
Итого: новый алгоритм будем обучать на исходных признаках и целевых значения, указанных выше. Ответ обученного алгоритма на каждом шаге - сумма ответов алгоритмов, полученных на предыдущих шагах.
Заметим, что мы осуществляем по сути градиентный спуск в пространстве алгоритмов, поэтому, как и алгоритме градиентного спуска полезно добавить множитель λ, чтобы осуществлять шаг не на всю длину градиента, а только в его направлении:

$$A_N(x) = \sum_{i=0}^{N-1} b_i(x) + \lambda b_N(x_i)$$

Ансамбли деревьев - бустинг

Если $L(y, a) = (a - y)^2$, то новые целевые значения, на которые обучается очередной алгоритм, вычислются очень просто: $-\frac{\partial L}{\partial a} = 2(y - a)$ Так как мы ввели шаг алгоритма, то двойку можно убрать и новый алгоритм нужно обучать на вектор ошибок предыдущих алгоритмов: $(y_1 - A_{N-1}(x_1), ..., y_{|X|} - A_{N-1}(x_{|X|}))$ с исходными признаками.

Ансамбли деревьев - бустинг

- В качестве базовых алгоритмов предлагается использовать неглубокие решающие деревья
- В отличие от случайного леса, алгоритм тяжело распараллелить
- С увеличением количества деревьев уменьшение bias, const variance

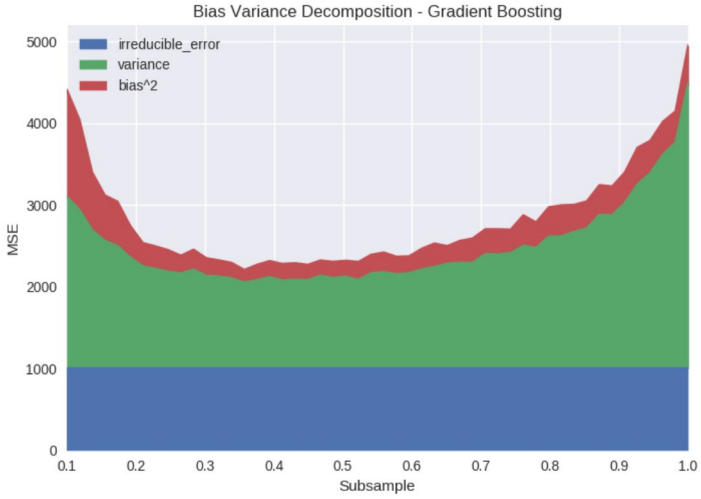
XGboost

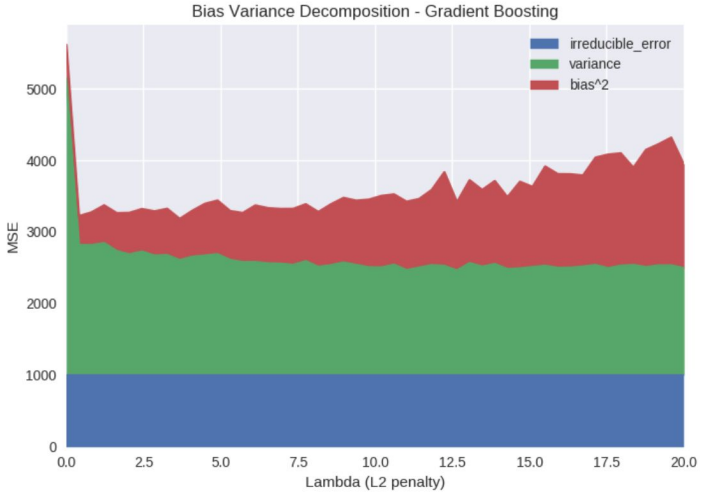
Weighted Quantile Sketch: разбиение интервалов значений признаков по квантилям + по разному учитываем интервалы для уже классифицированных объектов

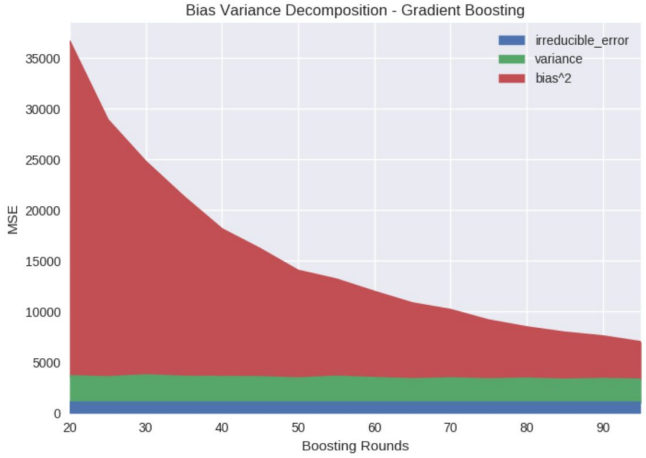
Sparsity-aware: обработка пропущенных значений признаков (дефолтная классификация)

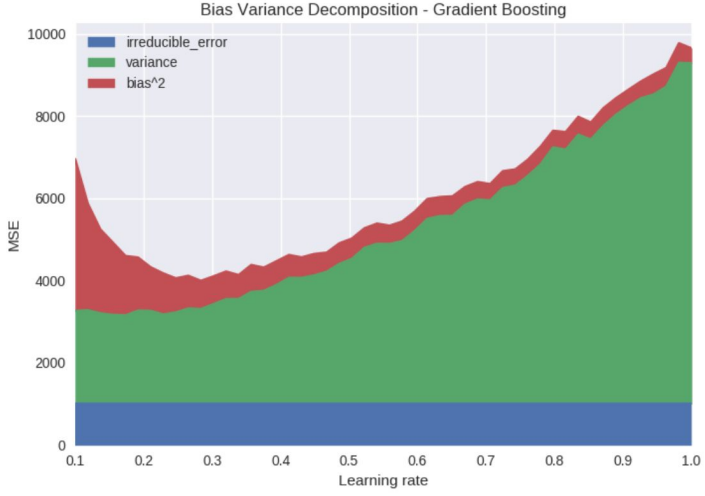
XGboost

https://devblogs.nvidia.com/bias-variance-decompositions-using-xgboost/









Практические рекомендации

Random Forest	• N_estimators • Max_depth • Min_samples_split • Min_samples_leaf • Max features	• 120, 300, 500, 800, 1200 • 5, 8, 15, 25, 30, None • 1, 2, 5, 10, 15, 100 • 1, 2, 5, 10 • Log2, sqrt, None
Xgboost	• Eta • Gamma • Max_depth • Min_child_weight • Subsample • Colsample_bytree • Lambda • alpha	• 0.01,0.015, 0.025, 0.05, 0.1 • 0.05-0.1,0.3,0.5,0.7,0.9,1.0 • 3, 5, 7, 9, 12, 15, 17, 25 • 1, 3, 5, 7 • 0.6, 0.7, 0.8, 0.9, 1.0 • 0.6, 0.7, 0.8, 0.9, 1.0 • 0.01-0.1, 1.0 , RS* • 0, 0.1, 0.5, 1.0 RS*

LightGBM

Когда данных много, XGBoost учится оочень долго. Тут приходит на помощь LightGBM.

Основные преимущества перед XGBoost:

- Скорость и эффективность (гистограммы для непрерывных признаков)
- Меньше потребляемость памяти (объединение фичей)
- Возможность работать с огромными датасетами
- Параллельность

Tuning Parameters of Light GBM

Благодаря разбиению по листьям алгоритм обучается быстрее, но больше подвержен overfitting-у. Давайте посмотрим, какими гиперпараметрами контролировать процесс обучения.

1. **For best fit**:
   a. num_leaves
   b. min_data_in_leaf
   c. max_depth
2. **For faster speed**
   a. bagging_fraction
   b. feature_fraction
   c. max_bin
3. **For better accuracy**:
   a. num_leaves
   b. max_bin

Catboost

Библиотека нацелена на датасеты с кат. фичами.

С помощью параметра ***one_hot_max_size*** можно заставить алгоритм рассматривать колонки с количеством уникальных значений <= ***one_hot_max_size*** в качестве категориальных. По умолчанию все числовые. Можно явно задать номера колонок с категориальными фичами с помощью параметра ***cat_features.***

```
from catboost import CatBoostRegressor
# Initialize data
cat_features = [0,1,2]
train_data = [["a","b",1,4,5,6],["a","b",4,5,6,7],["c","d",30,40,50,60]]
test_data = [["a","b",2,4,6,8],["a","d",1,4,50,60]]
train_labels = [10,20,30]
# Initialize CatBoostRegressor
model = CatBoostRegressor(iterations=2, learning_rate=1, depth=2)
# Fit model
model.fit(train_data, train_labels, cat_features)
```

https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

Catboost

Yandex CatBoost

Для остальных категориальных колонок, в которых уникальное число значений больше, чем ***one_hot_max_size***, CatBoost использует технику кодирования, сходную с mean-encoding, но позволяющую бороться с переобучением.

Процесс кодирования:

1) Перемешаем датасет (генерируется несколько перестановок)
2) Преобразуем непрерывные признаки к целочисленным
3) Все категориальные фичи преобразуются к числовым по формулам:

$$avg\_target = \frac{countInClass + prior}{totalCount + 1}$$

Let $\sigma = (\sigma_1, \ldots, \sigma_n)$ be the permutation, then $x_{\sigma_p,k}$ is substituted with

$$\frac{\sum_{j=1}^{p-1} [x_{\sigma_j,k} = x_{\sigma_p,k}] Y_{\sigma_j} + a \cdot P}{\sum_{j=1}^{p-1} [x_{\sigma_j,k} = x_{\sigma_p,k}] + a}, \qquad (1)$$

Сравнение библиотек

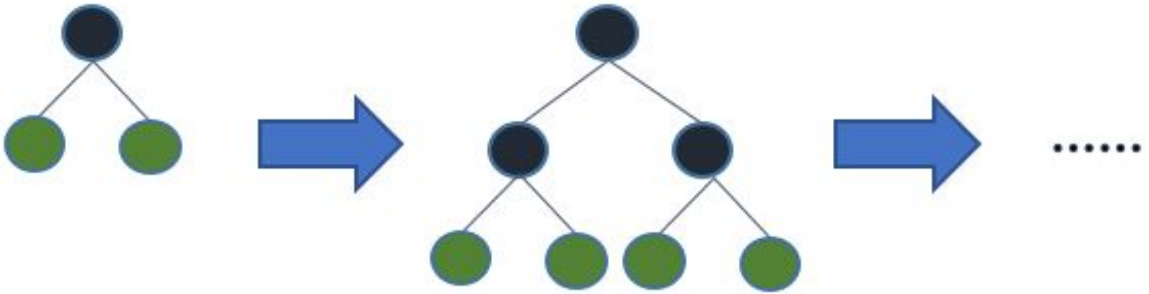
Model	Rounds	Train RMSE	Validation RMSE	Train time	Public Score
LightGBM	5000	1.505	1.60372	7min 48s	1.6717
XGBoost	2000	1.568	1.64924	54min 54s	1.6946
Catboost	1000	1.52184	1.61231	2min 24s	1.6722
Ensemble	--	--	--	--	1.6677

https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

Function	XGBoost	CatBoost	Light GBM
Important parameters which control overfitting	1. **learning_rate or eta** – optimal values lie between 0.01-0.2 2. **max_depth** 3. **min_child_weight:** similar to min_child leaf; default is 1	1. **Learning_rate** 2. **Depth** - value can be any integer up to 16. Recommended - [1 to 10] 3. No such feature like min_child_weight 4. **l2-leaf-reg**: L2 regularization coefficient. Used for leaf value calculation (any positive integer allowed)	1. **learning_rate** 2. **max_depth**: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune **num_leaves** (number of leaves in a tree) which should be smaller than 2^(max_depth). It is a very important parameter for LGBM 3. **min_data_in_leaf**: default=20, alias= min_data, min_child_samples
Parameters for categorical values	Not Available	1. **cat_features**: It denotes the index of categorical features 2. **one_hot_max_size**: Use one-hot encoding for all features with number of different values less than or equal to the given parameter value (max – 255)	1. **categorical_feature:** specify the categorical features we want to use for training our model
Parameters for controlling speed	1. **colsample_bytree**: subsample ratio of columns 2. **subsample**: subsample ratio of the training instance 3. **n_estimators**: maximum number of decision trees; high value can lead to overfitting	1. **rsm:** Random subspace method. The percentage of features to use at each split selection 2. No such parameter to subset data 3. **iterations**: maximum number of trees that can be built; high value can lead to overfitting	1. **feature_fraction:** fraction of features to be taken for each iteration 2. **bagging_fraction:** data to be used for each iteration and is generally used to speed up the training and avoid overfitting 3. **num_iterations:** number of boosting iterations to be performed; default=100

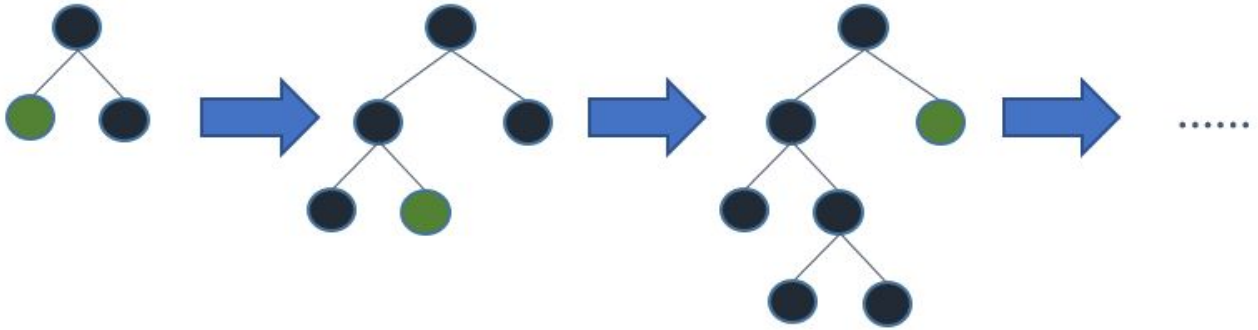
Сравнение библиотек

XGBoost



Level-wise tree growth

LightGBM



Leaf-wise tree growth

Links

- https://github.com/esokolov/ml-course-msu/blob/master/ML16/lecture-notes/Sem04_trees.pdf
- https://www.kaggle.com/dmilla/introduction-to-decision-trees-titanic-dataset
- Catboost: https://arxiv.org/pdf/1706.09516.pdf https://arxiv.org/pdf/1810.11363.pdf
- XGboost: https://arxiv.org/pdf/1603.02754.pdf
- LightGBM: https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf
- https://www.kaggle.com/kmezhoud/a-simple-xgboost-application
- Визуализация XGBoost: http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html

Links

- https://habr.com/ru/company/ods/blog/324402/
- https://habr.com/ru/post/192000/
- http://statistica.ru/theory/metod-butstrepa-i-ego-primenenie-v-sovremennom-analize-dannykh/
- https://devblogs.nvidia.com/bias-variance-decompositions-using-xgboost/
- https://arxiv.org/pdf/1809.04559.pdf
- https://habr.com/ru/company/ods/blog/327250/
- https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

Links for Homework

- https://habr.com/en/company/ods/blog/325422/
- https://medium.com/vickdata/a-simple-guide-to-scikit-learn-pipelines-4ac0d974bdcf