

# Композиции деревьев

Артём Филатов

28 января 2018 г.

# Где деревья проигрывают?

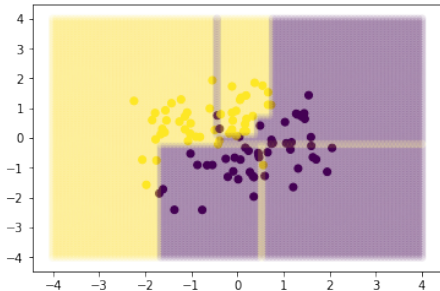


Рис.: Решающее дерево на плоскости

Простая идея: объединим деревья в композиции и построим один сложный классификатор.

Но смешивать нужно различные алгоритмы! Как этого добиться?

- Обучим различные алгоритмы (различные гиперпараметры)
- Обучим на разных подвыборках
- Обучим на различных признаках
- ...

# Алгоритм построения случайного леса

Случайный лес объединяет две идеи из вышеперечисленных: слабые алгоритмы (деревья) обучаются на различных подвыборках и на различных признаках.

- Подвыборка на которой будет строиться данное дерево - это подвыборка с повторением из исходной выборки того же размера (*бутстрэп*).
- На каждом ветвлении дерева подмножество оптимизированных признаков выбирается случайно.
- В итоге, предсказания всех деревьев усредняются.

---

**Algorithm 1:** Построение случайного леса

---

**Data:**  $X, y, num\_trees, \dots$

**Result:** Random Forest

forest = {};

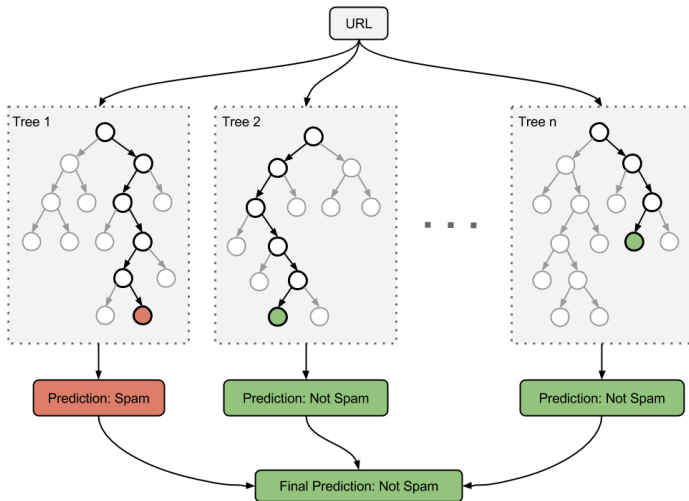
**for**  $i$  **in**  $num\_trees$  **do**

    sample  $X_{bs}, y_{bs}$  from  $X, y$ ;

    build random tree with  $X_{bs}, y_{bs}$ ;

    add tree to forest ;

---



1

<sup>1</sup><https://adpearance.com/blog/automated-inbound-link-analysis>

Мат. ожидание ошибки любого по алгоритма можно разложить следующим образом

$$\mathbb{E}[(y - \hat{f}(x))^2] = \text{Bias}[\hat{f}(x)]^2 + \text{Var}(\hat{f}(x)) + \sigma^2$$

Идея Leo Brieman'а была в том, что усреднение алгоритмов может снизить  $\text{Var}(\hat{f}(x))$ , не изменяя смещение (но только тогда, когда деревья максимально "независимы").

# Особенности случайного леса

- ❶ Переобучение леса не растёт с ростом количества деревьев (но это не значит, что его нет!)
- ❷ Быстро обучается на графических картах



# Количество деревьев в лесе

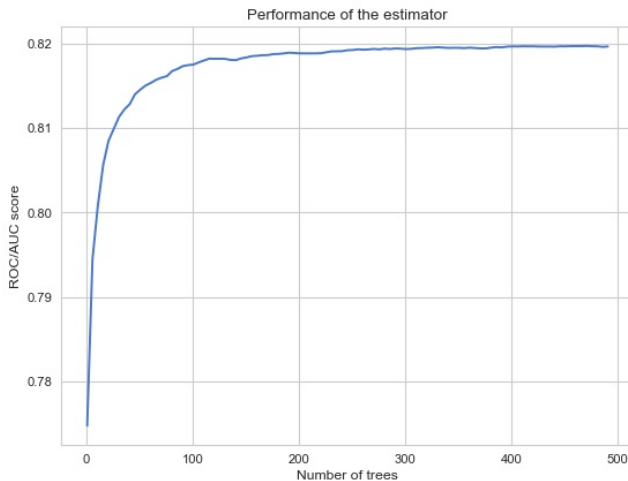


Рис.: Зависимость качества от количества деревьев

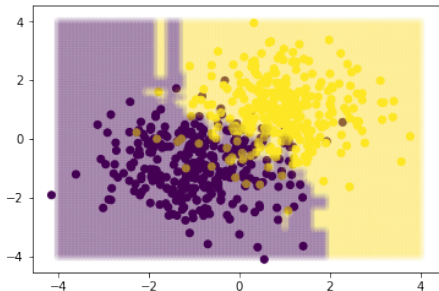


Рис.: Случайный лес на плоскости

# Как мы ещё можем объединить слабые алгоритмы?

Идея: пусть каждое следующее дерево исправляет ошибки предыдущего ансамбля деревьев.

Пусть мы имеем ансамбль деревьев  $H_n = \sum_{i=1}^n h_i$ . Тогда

$$h_{i+1} = \operatorname{argmin}_h \sum_i (H_n(x_i) - y_i)^2$$

# Алгоритм градиентного бустинга

---

## Algorithm 2: Построение случайного леса

---

**Data:**  $X, y, num\_trees, \dots$

**Result:** Gradient Boosting

```
ensemble = 0; for  $i$  in  $num\_trees$  do  
    build new tree with  $(X, y - ensemble(X))$ ;  
    add tree to ensemble ;
```

---

# Особенности бустинга

- 1 Высокая степень переобучения
- 2 Медленное обучение
- 3 Высокая эффективность на многих задачах

# Обобщение на любую функцию потерь

Заметим, что обучаясь на величину ошибки, мы на самом деле обучаемся на градиент функции потерь на предыдущем шаге (отсюда и название!).

$$\nabla \sum_i (y_i - H_n(x_i))^2 = \begin{pmatrix} \vdots \\ 2(H_n(x_i) - y_i) \\ \vdots \end{pmatrix}$$

Таким образом, мы можем взять произвольную функцию потерь и с помощью ее градиента обучать следующий алгоритм.

Таким образом мы получаем бустинг для классификации!

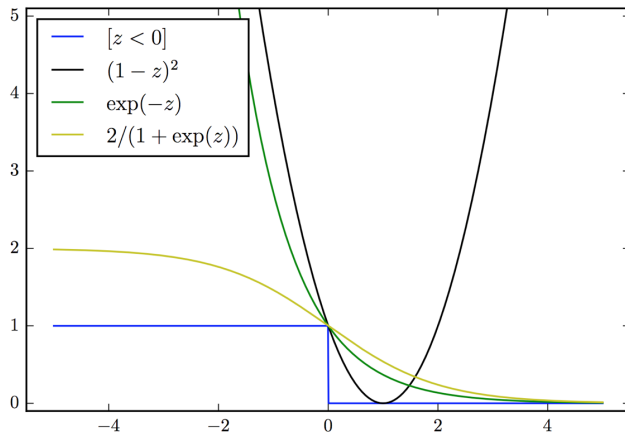
# Бустинг для классификации

Решаем следующую задачу

$$h_{i+1} = \operatorname{argmin}_h \sum_i [\operatorname{sign}(H_n(x_i) + h(x_i)) \neq y_i] =$$
$$\operatorname{argmin}_h \sum_i [(H_n(x_i) + h(x_i)) y_i < 0]$$

Но оптимизировать её не выйдет. Возьмем оценку сверху.

# Оценки сверху для индикаторной функции





Рассмотрим для примера следующую функцию потерь:

$$L(H(x_i), y_i) = e^{-y_i H(x_i)}$$

Модифицируем композицию сделав вес каждого элемента адаптивным

$$H_n(x) = \sum \alpha_i h_i(x)$$

Тогда задача обучения следующего алгоритма будет иметь следующий вид

$$h = \arg \min_h \sum_i e^{-y_i (H_n(x_i) + \alpha h(x_i))} = \arg \min_h \sum_i w_i e^{-y_i \alpha h(x_i)}$$

- xgboost: регуляризация деревьев, аппроксимация функции потерь с помощью разложения Тейлора.
- adaboost: экспоненциальная функция потерь, веса на каждом объекте.
- catboost: oblivious decision trees.

# Пример

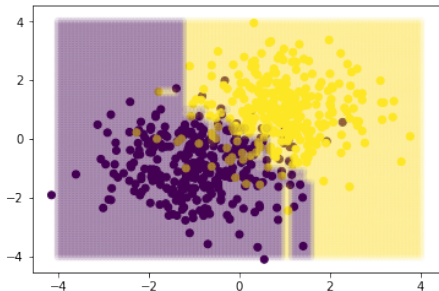


Рис.: Градиентный бустинг на плоскости

<http://arogozhnikov.github.io>

- 1 Градиентный бустинг последовательно строит композицию
- 2 Базовый алгоритм приближает антиградиент функции ошибки
- 3 Градиентный спуск в пространстве алгоритмов