Композиции

Машинное обучение, 20!7

Спасибо К. В. Воронцову, МФТИ, Data Factory Яндекса и кофеину.

Малютин Е. А.

Содержание

Планчик

Деревья

Зачем?

- бывают категориальные данные
- бывают сложности с метриками
- обратимся, например, к регрессии
  - легко обучается
  - восстанавливает только простые зависимости
  - усложнение - через спрямляющие пространства (и не только)

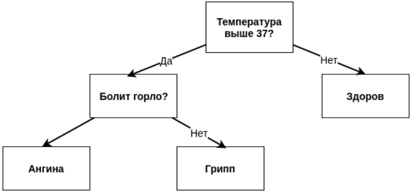


Рис.: Схема работы врача в Николаевской больнице

Решающие деревья

Условия:

- Самый популярный вариант: $[x_j < t]$

Прогноз в листе

- Регрессия
  - вещественное число
- Классификация
  - класс
  - распределение вероятностей над классами

Обучение деревьев

Поиск разбиения

- пусть в вершине m оказалась выборка X_m
- $Q(X_m, j, t)$ – критерий ошибки условия $[x^j < t]$
- ищем лучшие параметры перебором j и t:
  $$Q(X_m, j, t) \to \min_{j,t}$$
- разбиваем X_m на две части
  $$X_l = \{x \in X_m \mid [x^j \leq t]\}$$
  $$X_r = \{x \in X_m \mid [x^j \leq t]\}$$
- смыть – повторить

Критерий останова:

- В какой момент прекращать разбиение?
- В вершине один объект?

Критерии информативности

Обобщённый критерий ошибки:

$$Q(X_m,\ j,\ t)\ =\ \frac{|X_l|}{X_m}H(X_l)+\frac{|X_r|}{X_m}H(X_r)$$

Критерий информативности:

- $H(x)$
- Зависит от ответов на выборке X_m
- Чем меньше разброс ответов, тем меньше значение $H(x)$

Критерии информативности:

Регрессия

- $\bar{y}(X) = \frac{1}{|X_m|} \sum y_i$ – среднее
- $H(X) = \frac{1}{|X_m|} \sum (y_i - \bar{y}(X))^2$ – банальная дисперсия

Критерии информативности

Классификация

Тут все немного сложнее:

- Введём вспомогательную величину:
  $p_k = \frac{1}{|X|} \sum_{i \in X} [y_i = k]$
- Критерий Джини:

$$H(X) = \sum_{k=1}^{K} p_k(1 - p_k);$$

если $p_1 = 1; p_2 = p_3 = ... = p_K = 0$, то $H(X) = 0$

Критерии информативности

Классификация

Тут все немного сложнее:

- Введём вспомогательную величину:
$$p_k = \frac{1}{|X|} \sum_{i \in X} [y_i = k]$$
- Критерий Джини:
$$H(X) = \sum_{k=1}^{K} p_k(1 - p_k);$$
если $p_1 = 1; p_2 = p_3 = ... = p_K = 0$, то $H(X) = 0$
- Энтропийный критерий:
$$H(X) = \sum_{k=1}^{K} p_k \; ln(p_k);$$

Деревья

Резюме

- Легкость интерпретации результатов
- Не требует выбора входных атрибутов (сам выберет значимые)
- Точность модели сопоставима с другими методами (напр., НС (#антихайп))
- Быстрый процесс обучения
- Возможность обработки пропущенных значений
- Хорошо работают с категориальными типами данных
- Легко переобучаются
- Неустойчивы
- Прунниг – долго и дорого

Деревья

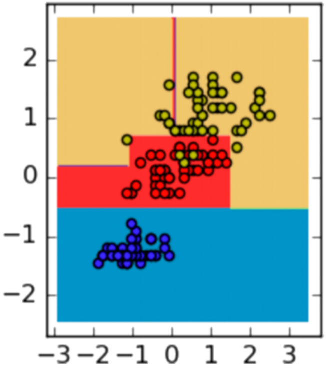


Рис.: Классификация здорового человека

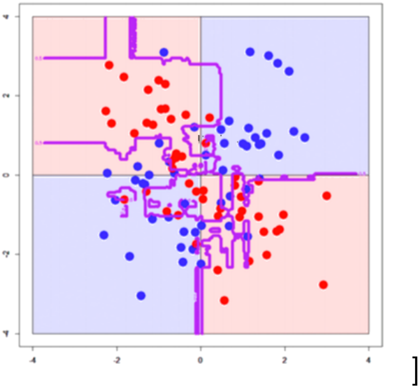


Рис.: Классификация курильщика

Композиции
Мотивация

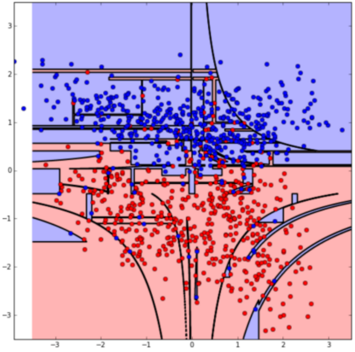


Рис.: Переобучили

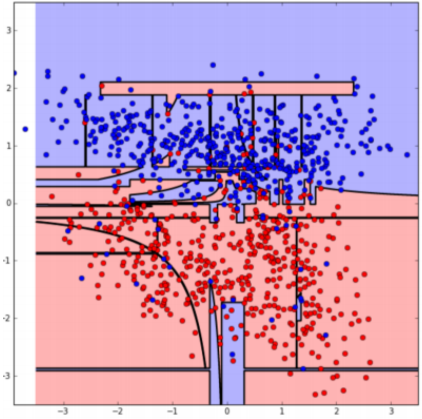


Рис.: Выкинули пару элементов

Композиции

Идеи

- Берём, и обучаем N деревьев

Композиции

Идеи

- Берём, и обучаем N деревьев
- 1. Усредняем

Композиции

Идеи

- Берём, и обучаем N деревьев
- 1. Усредняем
  - $a(x) = \frac{1}{N}\sum\limits_{n=1}^{N} b_n(x)$ – для регрессии

Композиции

Идеи

- Берём, и обучаем N деревьев
- 1. Усредняем
  - $a(x) = \frac{1}{N}\sum\limits_{n=1}^{N} b_n(x)$ – для регрессии
  - $a(x) = \frac{1}{N} sign(\sum\limits_{n=1}^{N} b_n(x))$

Композиции

Идеи

- Берём, и обучаем N деревьев
- 1. Усредняем
  - $a(x) = \frac{1}{N}\sum\limits_{n=1}^{N} b_n(x)$ – для регрессии
  - $a(x) = \frac{1}{N} sign(\sum\limits_{n=1}^{N} b_n(x))$
- 2. Рандомизация

Композиции

Идеи

- Берём, и обучаем N деревьев
- 1. Усредняем
  - $a(x) = \frac{1}{N}\sum\limits_{n=1}^{N} b_n(x)$ – для регрессии
  - $a(x) = \frac{1}{N} sign(\sum\limits_{n=1}^{N} b_n(x))$
- 2. Рандомизация
  - Бутстрап: выборка из l элементов с возвращениями. Вероятность конкретно элемента попасть в выборку – $\frac{2}{3}$

Композиции

Идеи

- Берём, и обучаем N деревьев
- 1. Усредняем
  - $a(x) = \frac{1}{N}\sum\limits_{n=1}^{N} b_n(x)$ – для регрессии
  - $a(x) = \frac{1}{N} sign(\sum\limits_{n=1}^{N} b_n(x))$
- 2. Рандомизация
  - Бутстрап: выборка из l элементов с возвращениями. Вероятность конкретно элемента попасть в выборку – $\frac{2}{3}$
  - Случайные подмножества: размер как гиперпараметр

Композиции

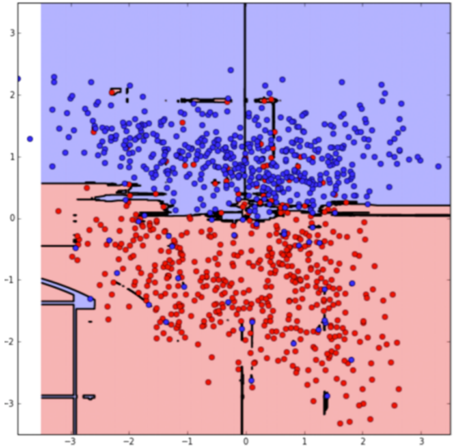
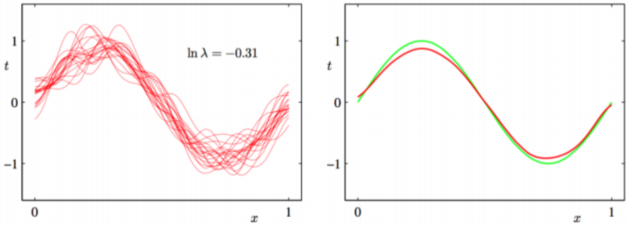


Рис.: 100 деревьев на бутстрапе

Разложение ошибки:

- **Шум** – компонента ошибки алгоритма, которая будет проявляться даже на идеальной модели в этой задаче.
- **Смещение** – отклонение, усредненного по различным обучающим выборкам, прогноза заданной модели от прогноза идеальной модели.
- **Разброс** – дисперсия ответов моделей, обученных по различным обучающим выборкам.



Композиции

Для ошибок

- **Деревья:**

Композиции

Для ошибок

- **Деревья:**
  - низкое смещение

Композиции

Для ошибок

- **Деревья:**
  - низкое смещение
  - большой разброс

Композиции

Для ошибок

- **Деревья:**
  - низкое смещение
  - большой разброс
- **Линейные алгоритмы:**

Композиции

Для ошибок

- **Деревья:**
  - низкое смещение
  - большой разброс
- **Линейные алгоритмы:**
  - смещение может быть большим

Композиции

Для ошибок

- **Деревья:**
  - низкое смещение
  - большой разброс
- **Линейные алгоритмы:**
  - смещение может быть большим
  - низкий разброс

Композиции

Для ошибок

- **Деревья:**
  - низкое смещение
  - большой разброс
- **Линейные алгоритмы:**
  - смещение может быть большим
  - низкий разброс
- **Композиция, в общем случае:**

Композиции

Для ошибок

- **Деревья:**
  - низкое смещение
  - большой разброс
- **Линейные алгоритмы:**
  - смещение может быть большим
  - низкий разброс
- **Композиция, в общем случае:**
  - не меняет смещение

Композиции

Для ошибок

- **Деревья:**
  - низкое смещение
  - большой разброс
- **Линейные алгоритмы:**
  - смещение может быть большим
  - низкий разброс
- **Композиция, в общем случае:**
  - не меняет смещение
  - (разброс) = 1/N (разброс алгоритма) + (корреляция)

Композиции

Для ошибок

- **Деревья:**
  - низкое смещение
  - большой разброс
- **Линейные алгоритмы:**
  - смещение может быть большим
  - низкий разброс
- **Композиция, в общем случае:**
  - не меняет смещение
  - (разброс) = 1/N (разброс алгоритма) + (корреляция)
  - При незалежності алгоритмов – уменьшаем разброс в N раз!

Основы независимости

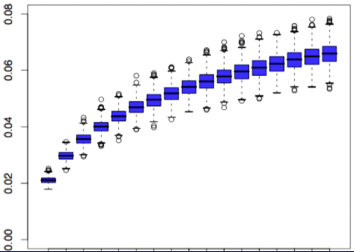
- **Беггинг:** Обучение базовых алгоритмов происходит на случайных подвыборках обучающей выборки. Причем чем меньше размер случайной подвыборки, тем более независимыми получаются базовые алгоритмы.

Основы незалежності

- **Беггинг:** Обучение базовых алгоритмов происходит на случайных подвыборках обучающей выборки. Причем чем меньше размер случайной подвыборки, тем более независимыми получаются базовые алгоритмы.
- **Метод случайных подпространств:** выбирается случайное подмножество признаков (столбцов матрицы "объекты–признаки") и очередной базовый алгоритм обучается только на этих признаках. Доля выбираемых признаков является гиперпараметром этого метода.

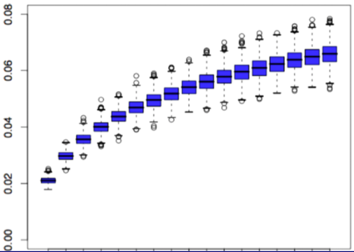
Случайные леса

- А можно ли рандомизировать сам процесс обучения дерева?



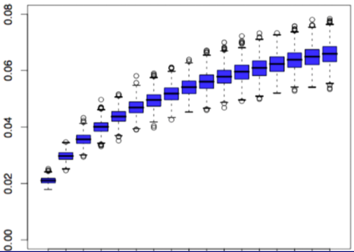
Случайные леса

- А можно ли рандомизировать сам процесс обучения дерева?
- Можно:



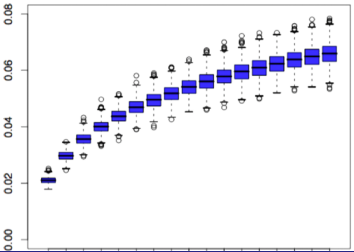
Случайные леса

- А можно ли рандомизировать сам процесс обучения дерева?
- Можно:
    - **как это происходит в дереве:** пусть в вершине m оказалась выборка X_m



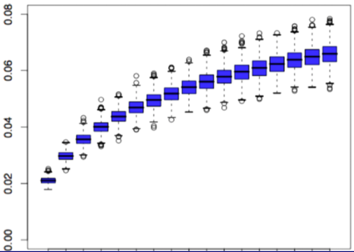
Случайные леса

- А можно ли рандомизировать сам процесс обучения дерева?
- Можно:
  - **как это происходит в дереве:** пусть в вершине m оказалась выборка X_m
  - $Q(X_m, j, t)$ – критерий ошибки условия $\left[x^j \leq t\right]$

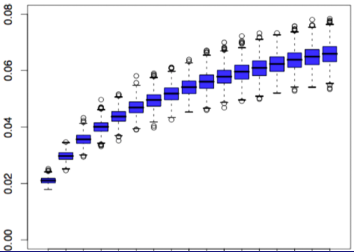


Случайные леса

- А можно ли рандомизировать сам процесс обучения дерева?
- Можно:
  - **как это происходит в дереве:** пусть в вершине m оказалась выборка X_m
  - $Q(X_m, j, t)$ – критерий ошибки условия $\left[x^j \leq t\right]$
  - $Q(X_m, j, t) \to \min\limits_{j,t}$



- А можно ли рандомизировать сам процесс обучения дерева?
- Можно:
  - **как это происходит в дереве:** пусть в вершине m оказалась выборка X_m
  - $Q(X_m, j, t)$ – критерий ошибки условия $\left[x^j \leq t\right]$
  - $Q(X_m, j, t) \to \min\limits_{j,t}$
  - **случайный лес:** ищем j среди **подмножества** признаков размера q



Случайные леса

Эвристики для q

- **Регрессия:** $q = \frac{d}{3}$
- **Классификация:** $q = \sqrt{d}$

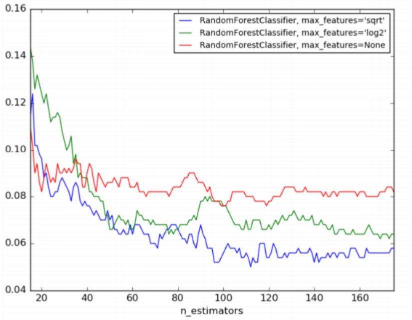


Рис.: Ошибка на тесте при росте числа деревьев

Случайные леса

Трюки

- Распараллеливание. Леса – идеальный алгоритм для этих целей.

Минусы

- Много деревьев $\rightarrow$ много вычислительных ресурсов
- Ненаправленный

Случайные леса

Трюки

- Распараллеливание. Леса – идеальный алгоритм для этих целей.
- Оценка качества работы:

Минусы

- Много деревьев → много вычислительных ресурсов
- Ненаправленный

Случайные леса

Трюки

- Распараллеливание. Леса – идеальный алгоритм для этих целей.
- Оценка качества работы:
  - $\frac{1}{3}$ объектов в обучении не побывало

Минусы

- Много деревьев $\rightarrow$ много вычислительных ресурсов
- Ненаправленный

Случайные леса

Трюки

- Распараллеливание. Леса – идеальный алгоритм для этих целей.
- Оценка качества работы:
  - $\frac{1}{3}$ объектов в обучении не побывало
  - а значит их можно использовать для теста! OOB – out-of-bag score:

$$OOB = \sum_{i=1}^{l} L(y_i, \frac{1}{\sum [x_i \notin X_n]} \sum [x_i \notin X_n] \, b_n(x_i))$$

Минусы

- Много деревьев $\rightarrow$ много вычислительных ресурсов
- Ненаправленный

Бустинг

Идея

- последовательное обучение алгоритмов

Пример

Бустинг

Идея

- последовательное обучение алгоритмов
- каждый следующий исправляет ошибки предыдущих

Пример

Бустинг

Идея

- последовательное обучение алгоритмов
- каждый следующий исправляет ошибки предыдущих
- достаточно простых базовых алгоритмов

Пример

Бустинг

Идея

- последовательное обучение алгоритмов
- каждый следующий исправляет ошибки предыдущих
- достаточно простых базовых алгоритмов

Пример

- регрессия на MSE: $MSE(a, X) = \sum$

Бустинг

Идея

- последовательное обучение алгоритмов
- каждый следующий исправляет ошибки предыдущих
- достаточно простых базовых алгоритмов

Пример

- регрессия на MSE: $MSE(a, X) = \sum$
- простой алгоритм: $b_1(x) = argmin_b \sum (b(x_i) - y_i)^2$

Бустинг

Идея

- последовательное обучение алгоритмов
- каждый следующий исправляет ошибки предыдущих
- достаточно простых базовых алгоритмов

Пример

- регрессия на MSE: $MSE(a, X) = \sum$
- простой алгоритм: $b_1(x) = argmin_b \sum(b(x_i) - y_i)^2$
- Второй строим так, что бы $b1(x) + b2(x)$ имели наименьшую ошибку $b_2(x) = argmin \sum(b_1(x) + b_2(x) - y_i)^2$

Бустинг

Идея

- последовательное обучение алгоритмов
- каждый следующий исправляет ошибки предыдущих
- достаточно простых базовых алгоритмов

Пример

- регрессия на MSE: $MSE(a, X) = \sum$
- простой алгоритм: $b_1(x) = argmin_b \sum (b(x_i) - y_i)^2$
- Второй строим так, что бы $b1(x) + b2(x)$ имели наименьшую ошибку $b_2(x) = argmin \sum (b_1(x) + b_2(x) - y_i)^2$
- в общем виде: $b_N(x) = \sum (b_N(x) - y_i(x) + \sum_{i=1}^{N-1} (b_i(x)))^2$

Градиентный бустинг

Мотивация

- инициализирующий алгоритм:

Градиентный бустинг

Мотивация

- инициализирующий алгоритм:
  - $b_0(x)$ – первый алгоритм

Градиентный бустинг

Мотивация

- инициализирующий алгоритм:
  - $b_0(x)$ – первый алгоритм
  - $b_0(x) = 0$

Градиентный бустинг

Мотивация

- инициализирующий алгоритм:
  - $b_0(x)$ – первый алгоритм
  - $b_0(x) = 0$
  - $b_0(x) = \frac{1}{l} \sum y_i$ – средний

Градиентный бустинг

Мотивация

- инициализирующий алгоритм:
  - $b_0(x)$ – первый алгоритм
  - $b_0(x) = 0$
  - $b_0(x) = \frac{1}{l}\sum y_i$ – средний
  - $b_0(x) = argmax \sum [y = y_i]$ – самый частый класс, для классификации

Градиентный бустинг

Мотивация

- инициализирующий алгоритм:
  - $b_0(x)$ – первый алгоритм
  - $b_0(x) = 0$
  - $b_0(x) = \frac{1}{l}\sum y_i$ – средний
  - $b_0(x) = argmax \sum [y = y_i]$ – самый частый класс, для классификации
- $a_{N-1}(x) = \sum b_n(x)$ – композиция N алгоритмов

Градиентный бустинг

Мотивация

- инициализирующий алгоритм:
  - $b_0(x)$ – первый алгоритм
  - $b_0(x) = 0$
  - $b_0(x) = \frac{1}{l}\sum y_i$ – средний
  - $b_0(x) = argmax \sum [y = y_i]$ – самый частый класс, для классификации
- $a_{N-1}(x) = \sum b_n(x)$ – композиция N алгоритмов
- $\sum L(y_i, a_{N-1}(x_i) + b_i(x)) \to \min\limits_{b}$ – задача оптимизации

Градиентный бустинг

Мотивация

- инициализирующий алгоритм:
  - $b_0(x)$ – первый алгоритм
  - $b_0(x) = 0$
  - $b_0(x) = \frac{1}{l}\sum y_i$ – средний
  - $b_0(x) = argmax \sum [y = y_i]$ – самый частый класс, для классификации
- $a_{N-1}(x) = \sum b_n(x)$ – композиция N алгоритмов
- $\sum L(y_i, a_{N-1}(x_i) + b_i(x)) \to \min\limits_{b}$ – задача оптимизации
- $s = (s_1, s_2 ... s_l)$ – вектор сдвигов, переформулируем задачу в форме: $\sum L(y_i, a_{N-1}(x_i) + s_i) \to \min\limits_{s}$

Градиентный бустинг

Мотивация

- инициализирующий алгоритм:
  - $b_0(x)$ – первый алгоритм
  - $b_0(x) = 0$
  - $b_0(x) = \frac{1}{l}\sum y_i$ – средний
  - $b_0(x) = argmax \sum [y = y_i]$ – самый частый класс, для классификации
- $a_{N-1}(x) = \sum b_n(x)$ – композиция N алгоритмов
- $\sum L(y_i, a_{N-1}(x_i) + b_i(x)) \to \min\limits_b$ – задача оптимизации
- $s = (s_1, s_2 \ldots s_l)$ – вектор сдвигов, переформулируем задачу в форме: $\sum L(y_i, a_{N-1}(x_i) + s_i) \to \min\limits_s$
- оптимальный сдвиг: $-\nabla F$, алгоритм учится предсказывать его

Градиентный бустинг

Алгоритм

1. Инициализация: инициализация композиции $a_0(x) = b_0(x)$, то есть построение простого алгоритма b_0.

Резюме

- Последовательно строим композицию
- Базовый алгоритм обучается на антиградиенте ошибки
- Результат – градиентный спуск в пространстве алгоритмов

Градиентный бустинг

Алгоритм

1. Инициализация: инициализация композиции $a_0(x) = b_0(x)$, то есть построение простого алгоритма b_0.
2. 2. Шаг итерации:

Резюме

- Последовательно строим композицию
- Базовый алгоритм обучается на антиградиенте ошибки
- Результат – градиентный спуск в пространстве алгоритмов

Градиентный бустинг

Алгоритм

1. Инициализация: инициализация композиции $a_0(x) = b_0(x)$, то есть построение простого алгоритма b_0.
2. 2. Шаг итерации:
   - Вычисляется вектор сдвига: $s = -\nabla F = (-L_z^{'}(y_l, a_{n-1}(x_l)), \ldots, -L_z^{'}(y_l, a_{n-1}(x_l)))$

Резюме

- Последовательно строим композицию
- Базовый алгоритм обучается на антиградиенте ошибки
- Результат – градиентный спуск в пространстве алгоритмов

Градиентный бустинг

Алгоритм

1. Инициализация: инициализация композиции $a_0(x) = b_0(x)$, то есть построение простого алгоритма b_0.
2. 2. Шаг итерации:
   - Вычисляется вектор сдвига: $s = -\nabla F = (-L_z^{'}(y_l, a_{n-1}(x_l)), ..., -L_z^{'}(y_l, a_{n-1}(x_l)))$
   - Строится алгоритм: $b_n(x) = argmin\frac{1}{l}\sum(b(x_i) - s_i)^2$

Резюме

- Последовательно строим композицию
- Базовый алгоритм обучается на антиградиенте ошибки
- Результат – градиентный спуск в пространстве алгоритмов

Градиентный бустинг

Алгоритм

1. Инициализация: инициализация композиции $a_0(x) = b_0(x)$, то есть построение простого алгоритма b_0.
2. 2. Шаг итерации:
   - Вычисляется вектор сдвига: $s = -\nabla F = (-L_z^{'}(y_l, a_{n-1}(x_l)), ..., -L_z^{'}(y_l, a_{n-1}(x_l)))$
   - Строится алгоритм: $b_n(x) = argmin\frac{1}{l}\sum(b(x_i) - s_i)^2$
   - Алгоритм $b_n(x)$ добавляется в композицию: $a_n(x) = \sum b_i(x)$

Резюме

- Последовательно строим композицию
- Базовый алгоритм обучается на антиградиенте ошибки
- Результат – градиентный спуск в пространстве алгоритмов

Градиентный бустинг

Алгоритм

1. Инициализация: инициализация композиции $a_0(x) = b_0(x)$, то есть построение простого алгоритма b_0.
2. 2. Шаг итерации:
   - Вычисляется вектор сдвига: $s = -\nabla F = (-L_z^{'}(y_l, a_{n-1}(x_l)), ..., -L_z^{'}(y_l, a_{n-1}(x_l)))$
   - Строится алгоритм: $b_n(x) = argmin\frac{1}{l}\sum(b(x_i) - s_i)^2$
   - Алгоритм $b_n(x)$ добавляется в композицию: $a_n(x) = \sum b_i(x)$
3. Если надо – останавливаемся

Резюме

- Последовательно строим композицию
- Базовый алгоритм обучается на антиградиенте ошибки
- Результат – градиентный спуск в пространстве алгоритмов

Градиентный бустинг

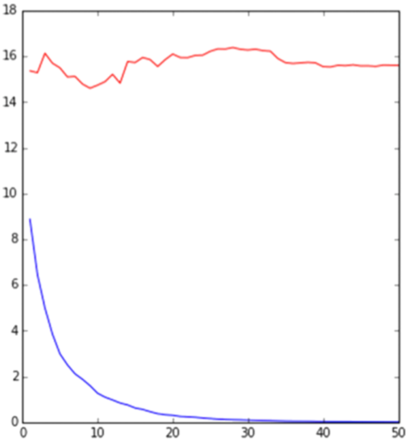


Рис.: Красным – test; Синим – train

Почему

- Алгоритм приближает вектор антиградиента

Что делать?

- $a_N(x) = a_{N-1}(x) + \kappa a_N(x)$ – сокращение размера шага
- Стохастический градиентный бустинг – каждый алгоритм обучаем на подвыборках

Градиетный бустинг

Переобучение

Почему

- Алгоритм приближает вектор антиградиента
- Алгоритм – слабый, приближение – плохое

Что делать?

- $a_N(x) = a_{N-1}(x) + \kappa a_N(x)$ – сокращение размера шага
- Стохастический градиентный бустинг – каждый алгоритм обучаем на подвыборках

Градиетный бустинг

Переобучение

Почему

- Алгоритм приближает вектор антиградиента
- Алгоритм – слабый, приближение – плохое
- Вместо градиентного спуска – случайное блуждание

Что делать?

- $a_N(x) = a_{N-1}(x) + \kappa a_N(x)$ – сокращение размера шага
- Стохастический градиентный бустинг – каждый алгоритм обучаем на подвыборках

Переобучение