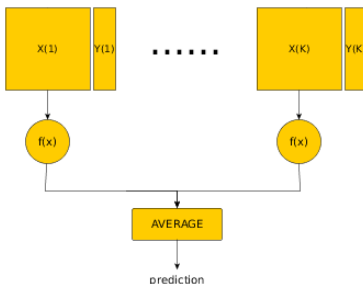


# Методы построения ансамблей

Виктор Китов

[victorkitov.github.io](https://github.com/victorkitov)



Курс поддержан  
фондом  
'Интеллект'



Победитель  
конкурса VK среди  
курсов по IT



# Содержание

- 1 Методы построения разных моделей
- 2 Фиксированная агрегирующая функция (против переобучения)
- 3 Композиции на разных обучающих подвыборках
- 4 Блендинг, стэкинг

## Методы построения разных моделей

- Модели разных семейств.
- Использовать разные гиперпараметры (напр.  $K$  в  $K$ -NN)
- Использовать разные начальные инициализации при настройке градиентными методами.
  - для невыпуклых задач, например, нейросетей
- Использовать разную инициализацию для генератора случайных чисел.
  - рандомизированные алгоритмы, настройка по минибатчам, случ. инициализация

## Методы построения разных моделей

- Предсказывать целевую переменную с разными функциями потерь.
- модуль, квадрат, логистическая, hinge и т.д.
- Настраивать модель предсказывать различные преобразования целевой переменной
  - например  $\ln y$ . Прогнозы настроенной модели обратно преобразовывать  $e^{\hat{y}}$ .
- Настраивать одну модель на разных фрагментах обучающей выборки.
  - подвыборки по объектам, признакам

# Содержание

- 1 Методы построения разных моделей
- 2 Фиксированная агрегирующая функция (против переобучения)
- 3 Композиции на разных обучающих подвыборках
- 4 Блендинг, стэкинг

## Регрессия

$$\hat{y}(x) = \frac{1}{M} \sum_{m=1}^M f_m(x)$$

$$\hat{y}(x) = \frac{1}{\sum_{m=1}^M w_m} \sum_{m=1}^M w_m f_m(x)$$

- Взвешенное усреднение лучше, если модели сильно отличаются по точности.
- Веса  $w_1 \geq 0, \dots, w_M \geq 0$  нужно настраивать на отдельной выборке (не той, на которой обучали  $f_1(x), \dots, f_M(x)$ )
- Альтернатива: медиана/взвешенная медиана

## Классификаторы выдают **вероятности**

- Пусть  $p_y^m(x)$  - вероятность класса  $y$  по мнению классификатора  $m$ .
- Равномерная агрегация:

$$p_c(x) = \frac{1}{M} \sum_{m=1}^M p_c^m(x)$$

- Взвешенная агрегация:

$$p_c(x) = \frac{1}{\sum_{m=1}^M w_m} \sum_{m=1}^M w_m p_c^m(x)$$

- Взвешенное усреднение лучше, если модели сильно отличаются по точности.
- Веса  $w_1 \geq 0, \dots, w_M \geq 0$  нужно настраивать на отдельной выборке (не той, на которой обучали  $f_1(x), \dots, f_M(x)$ )

## Классификаторы выдают **метки классов**

- Голосование по большинству (majority vote)
  - возможен взвешенный учет классификаторов
- Бинарная классификация:  $\hat{y} = +1 \iff$ 
  - $\geq k$  классификаторов выдают +1 (k-out-of-N)
    - возможен взвешенный учет классификаторов
  - все классификаторы выдают +1 (AND, N-out-of-N)
  - хотя бы один выдает +1 (OR, 1-out-of-N)



## Классификаторы выдают **рейтинги**

- Пусть  $g_y^m(x)$  - рейтинг класса  $y$  в модели  $m$ .
- Проблема: рейтинги несравнимы для разных моделей.
- Решение (Brier scores):
  - 1 Стандартизованный рейтинг - #классов ниже по рейтингу:

$$s_y^m(x) = \sum_{i \neq y} \mathbb{I}[g_y^m(x) > g_i^m(x)]$$

$$s_y^m(x) \in \{1, 2, \dots, C - 1\}$$

- 2 Предскажем класс с максимальным усредненным по моделям рейтингом:

$$\hat{y}(x) = \arg \max_y \frac{1}{M} \sum_{m=1}^M s_y^m(x)$$

# Пример

исходные рейтинги

	$g_1(x)$	$g_2(x)$	$g_3(x)$	$g_4(x)$
$f_1(x)$	100	70	34	-25
$f_2(x)$	15	0	-14	-10
$f_3(x)$	0.05	0.6	0.2	0.15

ранжирование

модель	ранжирование классов
$f_1(x)$	$1 \succ 2 \succ 3 \succ 4$
$f_2(x)$	$1 \succ 2 \succ 4 \succ 3$
$f_3(x)$	$2 \succ 3 \succ 4 \succ 1$

рейтинги Бриера

	$g'_1(x)$	$g'_2(x)$	$g'_3(x)$	$g'_4(x)$
$f_1(x)$	3	2	1	0
$f_2(x)$	3	2	0	1
$f_3(x)$	0	3	2	1

усреднённые рейтинги Бриера

$g''_1(x)$	$g''_2(x)$	$g''_3(x)$	$g''_4(x)$
6/3	7/3	3/6	2/6

## Использование голосования (регрессия)

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.metrics import mean_absolute_error

X_train, X_test, Y_train, Y_test =
    get_demo_classification_data()
knn = KNeighborsRegressor(n_neighbors=100)
tree_model = DecisionTreeRegressor()

ensemble = VotingRegressor(    # усредняющий ансамбль
    estimators=[('K-NN', knn), ('DT', tree_model)],
    weights=[0.5, 0.5])
ensemble.fit(X_train, Y_train) # обучение базовых м-лей
Y_hat = ensemble.predict(X_test) # построение прогнозов
print(f'Средний модуль ошибки (MAE): {
    mean_absolute_error(Y_test, Y_hat) : .2 f}')
```

Больше информации. Полный код.

# Использование голосования (классификация) 1

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import brier_score_loss

X_train, X_test, Y_train, Y_test =
    get_demo_classification_data()
# инициализация базовых моделей
log_model = LogisticRegression()
tree_model = DecisionTreeClassifier()
```

## Использование голосования (классификация) 2

```
# Голосование по большинству (для меток классов)
ensemble = VotingClassifier(
    estimators=[('logistic regression', log_model),
                ('decision tree', tree_model)],
    voting='hard', weights=[0.5,0.5])
# обучение базовых моделей ансамбля:
ensemble.fit(X_train, Y_train)
# построение прогнозов:
Y_hat = ensemble.predict(X_test)
print(f'Точность VotingClassifier: \
{100*accuracy_score(Y_test, Y_hat):.1f}%')
```

## Использование голосования (классификация) 3

```
# Ансамбль, усредняющий вероятности классов
ensemble = VotingClassifier(
    estimators=[('logistic regression', log_model), ('
    decision tree', tree_model)],
    voting='soft', weights=[0.5,0.5])
# обучение базовых моделей ансамбля:
ensemble.fit(X_train, Y_train)
# построение прогнозов:
Y_hat = ensemble.predict(X_test)
print(f'Точность VotingClassifier: \
{100*accuracy_score(Y_test, Y_hat):.1 f}%')

P_hat = ensemble.predict_proba(X_test) # вероятности
классов
loss = brier_score_loss(Y_test, P_hat[:,1])
print(f'Мера Бриера отклонения вер-тей: {loss:.2 f}')
```

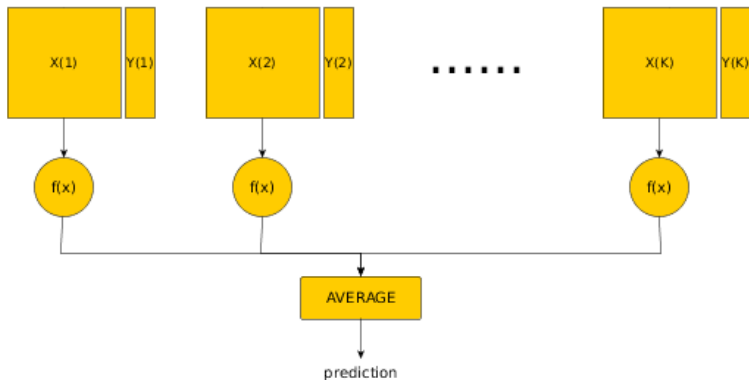
Больше информации. Полный код.

# Содержание

- 1 Методы построения разных моделей
- 2 Фиксированная агрегирующая функция (против переобучения)
- 3 Композиции на разных обучающих подвыборках**
- 4 Блендинг, стэкинг

## Усреднение по выборкам

### Усреднение по выборкам





## Усреднение по выборкам

**Усреднение по выборкам:** если модель переобучается на выборке  $(X, Y)$  можно усреднять множество моделей, обученных на разных реализациях обучающих выборок  $(X_k, Y_k)$ ,  $k = 1, 2, \dots, M$ .

$$\begin{aligned} bias_{X,Y} \left[ \frac{1}{M} \sum_{k=1}^M f(x, X_k, Y_k) \right] &= y(x) - \mathbb{E}_{X,Y} \left[ \frac{1}{M} \sum_{k=1}^M f(x, X_k, Y_k) \right] \\ &= y(x) - \frac{1}{M} \sum_{k=1}^M \mathbb{E}_{X,Y} f(x, X_k, Y_k) = y(x) - \frac{1}{M} \sum_{k=1}^M \mathbb{E}_{X,Y} f(x, X, Y) \\ &= y(x) - \mathbb{E}_{X,Y} f(x, X, Y) \end{aligned}$$

т.е. смещение=смещению 1-го алгоритма.

## Усреднение по выборкам: дисперсия

$$\begin{aligned}\mathrm{Var}_{X,Y} \left[ \frac{1}{M} \sum_{k=1}^M f(x, X_k, Y_k) \right] &= \mathbb{E}_{X,Y} \left[ \frac{1}{M} \sum_{k=1}^M f(x, X_k, Y_k) - \mathbb{E}_{X,Y} \left[ \frac{1}{M} \sum_{k=1}^M f(x, X_k, Y_k) \right] \right]^2 \\&= \mathbb{E}_{X,Y} \left[ \frac{1}{M} \sum_{k=1}^M (f(x, X_k, Y_k) - \mathbb{E}_{X,Y} f(x, X_k, Y_k)) \right]^2 = \\&= \frac{1}{M^2} \mathbb{E}_{X,Y} \left[ \sum_{k=1}^M (f(x, X_k, Y_k) - \mathbb{E}_{X,Y} f(x, X_k, Y_k)) \right]^2 = \\&= \frac{1}{M^2} \sum_{k=1}^M \mathbb{E}_{X,Y} [f(x, X_k, Y_k) - \mathbb{E}_{X,Y} f(x, X_k, Y_k)]^2 + \\&+ \frac{1}{M^2} \sum_{k_1 \neq k_2} \mathbb{E}_{X,Y} [(f(x, X_{k_1}, Y_{k_1}) - \mathbb{E}_{X,Y} f(x, X_{k_1}, Y_{k_1})) (f(x, X_{k_2}, Y_{k_2}) - \mathbb{E}_{X,Y} f(x, X_{k_2}, Y_{k_2}))] \\&= \frac{1}{M^2} \sum_{k=1}^M \mathrm{Var}_{X,Y} [f(x, X_k, Y_k)] + \frac{1}{M^2} \sum_{k_1 \neq k_2} \mathrm{cov} [f(x, X_{k_1}, Y_{k_1}), f(x, X_{k_2}, Y_{k_2})]\end{aligned}$$

## Усреднение по выборкам: дисперсия

При нескоррелированных  $f(x, X_k, Y_k)$ :

$$\begin{aligned} & \frac{1}{M^2} \sum_{k=1}^K \text{Var}_{X,Y} [f(x, X_k, Y_k)] + \\ & + \frac{1}{M^2} \sum_{k_1 \neq k_2} \text{cov} [f(x, X_{k_1}, Y_{k_1}), f(x, X_{k_2}, Y_{k_2})] = \\ & = \frac{1}{M} \text{Var}_{X,Y} [f(x, X_k, Y_k)] \end{aligned}$$

Дисперсия в  $M$  раз меньше.

## Усреднение по выборкам: дисперсия

При частичной скоррелированности, дисперсия тоже уменьшается (используем  $\text{cov}(x, y) \leq \sqrt{\text{Var}(x)}\sqrt{\text{Var}(y)}$ ):

$$\begin{aligned} & \frac{1}{M^2} \sum_{k=1}^K \text{Var}_{X,Y} [f(x, X_k, Y_k)] + \\ & + \frac{1}{M^2} \sum_{k_1 \neq k_2} \text{cov} [f(x, X_{k_1}, Y_{k_1}), f(x, X_{k_2}, Y_{k_2})] = \\ & = \frac{1}{M^2} \sum_{k_1=1}^M \sum_{k_2=1}^M \text{cov} [f(x, X_{k_1}, Y_{k_1}), f(x, X_{k_2}, Y_{k_2})] \leq \\ & \leq \frac{1}{M^2} \sum_{k_1=1}^M \sum_{k_2=1}^M \sqrt{\text{Var}_{X,Y} [f(x, X_{k_1}, Y_{k_1})]} \sqrt{\text{Var}_{X,Y} [f(x, X_{k_2}, Y_{k_2})]} \leq \\ & \leq \text{Var}_{X,Y} [f(x, X, Y)] \end{aligned}$$

## Методы генерации псевдовыборок

Как генерировать  $(X_1, Y_1), \dots (X_M, Y_M)$  по единственной  $(X, Y)$ ?

---

<sup>1</sup>Какова вероятность того, что некоторый объект не попадет в подвыборку?  
Чему равен предел этой вероятности при  $N \rightarrow \infty$ ?

## Методы генерации псевдовыборок

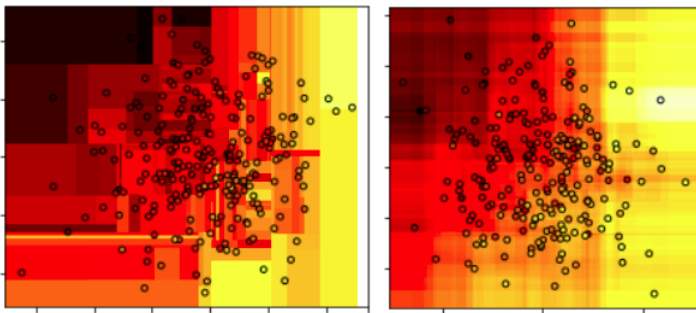
Как генерировать  $(X_1, Y_1), \dots, (X_M, Y_M)$  по единственной  $(X, Y)$ ?

- Кросс-валидационные подвыборки.
- Бэггинг (bagging): случайный выбор  $N$  объектов с возвращением<sup>1</sup>.
- Пэйстинг (pasting): случайный выбор  $K$  объектов без с возвращением.
- Метод случайных подпространств (random subspaces): случайный выбор  $K$  признаков без возвращения.
- Метод случайных фрагментов (random patches): комбинируется сэмплирование объектов и признаков без возвращения.

---

<sup>1</sup>Какова вероятность того, что некоторый объект не попадет в подвыборку?  
Чему равен предел этой вероятности при  $N \rightarrow \infty$ ?

## Бэггинг деревьев



Регрессия: одно дерево и бэггинг над деревьями

## Бэггинг деревьев

Настройка решающего правила в узле CART:

$$\hat{i}, \hat{h} = \arg \max_{f, h \in P(t)} \Delta \phi(t)$$

$P(t)$  для стандартных решающих деревьев:

$$\begin{aligned} P &= \{ \} \\ \text{для каждого } i \text{ in } \{1, \dots, D\} \\ &\quad \text{для каждого } h \text{ из } \textit{unique} \{x_n^i\}_{n: x_n \in t} \\ &\quad P := P \cup (i, h) \end{aligned}$$

Бэггинг над решающими деревьями успешно борется с их переобучением.



## Использование бэггинга (регрессия)

```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error

X_train, X_test, Y_train, Y_test =
    get_demo_classification_data()
model = BaggingRegressor(
    DecisionTreeRegressor(), # базовая модель
    max_samples=0.8, # доля используемых объектов
    max_features=1.0, # доля используемых пр-ков
    n_jobs=-1) # используем все ядра процессора
model.fit(X_train, Y_train) # обучение модели
Y_hat = model.predict(X_test) # построение прогнозов
print(f'Средний модуль ошибки (MAE): \
    {mean_absolute_error(Y_test, Y_hat):.2f}')
```

## Использование бэггинга (классификация)

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, brier_score_loss

X_train, X_test, Y_train, Y_test =
    get_demo_classification_data()
model = BaggingClassifier(
    DecisionTreeClassifier(), # базовая модель
    n_estimators=100, # число базовых моделей
    max_samples=0.8, # доля используемых объектов
    max_features=1.0, # доля используемых признаков
    n_jobs=-1) # используем все ядра процессора
model.fit(X_train, Y_train) # обучение модели
Y_hat = model.predict(X_test) # построение прогнозов
print(f'Точность прогнозов: \
      {100*accuracy_score(Y_test, Y_hat):.1f}%')

P_hat = model.predict_proba(X_test) # вероятности классов
loss = brier_score_loss(Y_test, P_hat[:,1])
print(f'Мера ошибки Бриера: {loss:.2f}')
```

Больше информации. [Полный код](#).

## Случайный лес, особо случайные деревья

$P(t)$  для случайного леса (random forest, RF):

$P = \{\}$ ,  $K = \alpha D$

сэмплируем  $i_1, \dots, i_K$  случайно из  $\{1, \dots, D\}$  # без возвращения  
для каждого  $i$  из  $i_1, \dots, i_K$

для каждого  $h$  из  $\text{unique}\{x_n^i\}_{n: x_n \in t}$

$P := P \cup (i, h)$

$S(t)$  для особо случайных деревьев (extra random trees, ERT):

$S = \{\}$ ,  $K = \alpha D$

сэмплируем  $i_1, \dots, i_K$  случайно из  $\{1, \dots, D\}$  # с возвращением  
для каждого  $f$  in  $d_1, \dots, d_K$

сэмплируем  $h$  случайно из  $\text{unique}\{x_n^i\}_{n: x_n \in t}$  # без  
возвращения

$P := P \cup (f, h)$

## Использование случайного леса (регрессия)

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

X_train, X_test, Y_train, Y_test =
    get_demo_regression_data()
model = RandomForestRegressor(
    n_estimators=100, # число базовых моделей
    bootstrap=True, # обучать на подвыборке/исх. выборке
    max_features=0.5, # доля используемых пр-ков
    n_jobs=-1) # используем все ядра процессора
model.fit(X_train, Y_train) # обучение модели
Y_hat = model.predict(X_test) # построение прогнозов
print(f'Средний модуль ошибки (MAE): \
    {mean_absolute_error(Y_test, Y_hat):.2f}')
```

Особо случайные деревья - ExtraTreesRegressor.

## Использование случайного леса (классификация)

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, brier_score_loss

X_train, X_test, Y_train, Y_test = get_demo_classification_data()
model = RandomForestClassifier(
    n_estimators=100, # число базовых моделей
    bootstrap=True, # обучать на подвыборке/исходной выборке
    max_features=0.5, # доля используемых признаков
    n_jobs=-1) # используем все ядра процессора
model.fit(X_train, Y_train) # обучение модели
Y_hat = model.predict(X_test) # построение прогнозов
print(f'Точность прогнозов: \
      {100*accuracy_score(Y_test, Y_hat):.1f}%')

P_hat = model.predict_proba(X_test) # можно предсказывать
    вероятности классов
loss = brier_score_loss(Y_test, P_hat[:, 1]) # считаем
    качество Бриера на вероятности положительного класса
print(f'Мера ошибки Бриера: {loss:.2f}')
```

Больше информации. Полный код. См. также ExtraTreesClassifier.

## Вневыборочная оценка

Оценка по обучающей выборке - оптимистическая оценка сверху:

$$L = \frac{1}{N} \sum_{n=1}^N \mathcal{L} \left( \frac{1}{M} \sum_{m=1}^M f_m(x_n), y_n \right)$$

## Вневыборочная оценка

Оценка по обучающей выборке - оптимистическая оценка сверху:

$$L = \frac{1}{N} \sum_{n=1}^N \mathcal{L} \left( \frac{1}{M} \sum_{m=1}^M f_m(x_n), y_n \right)$$

Вневыборочная оценка (out-of-bag estimate) - если с бэггингом

$$L_{OOB} = \frac{1}{N} \sum_{n=1}^N \mathcal{L} \left( \frac{1}{|I_n|} \sum_{m \in I_n} f_m(x_n), y_n \right)$$

- $I_n \subset \{1, 2, \dots, M\}$  - набор моделей, не использовавших  $(x_n, y_n)$  для обучения.
- Не требуется дополнительная валидационная выборка.
- Немного пессимистическая оценка потерь снизу.

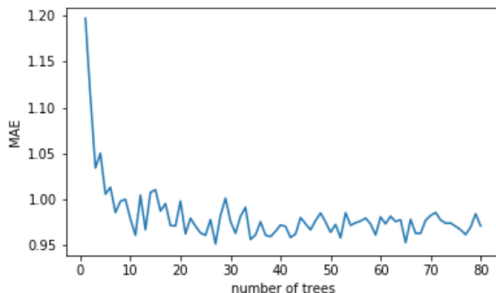
## Комментарии

- Бэггинг, случайный лес, особо случайные деревья:
  - легко распараллеливаются
  - базовые модели не учатся исправлять ошибки друг друга
- Деревья случайный лес, особо случайные деревья могут строиться на одинаковой обучающей выборке
  - `bootstrap=False` в `sklearn`
  - за счет случайности  $P(t)$  модели все равно будут получаться разные



## Число базовых моделей в ансамбле

- Пусть  $M = \#$  базовых моделей.
- Типичная зависимость потерь от  $M$ :



- Нет переобучения с  $\uparrow M$ : просто избыточное усреднение по однотипным моделям.
- Настройка: подбор всех параметров с малым  $M$ , затем  $\uparrow M$  для итоговой модели.

# Содержание

- 1 Методы построения разных моделей
- 2 Фиксированная агрегирующая функция (против переобучения)
- 3 Композиции на разных обучающих подвыборках
- 4 Блендинг, стэкинг

## Обучаемая мета-модель

$$\hat{y}(x) = G(f_1(x), \dots, f_M(x))$$

$G(\cdot) = G_w(\cdot)$  - настраиваемая функция, например

$$G(f_1, f_2, \dots, f_M) = w_0 + w_1 f_1 + \dots + w_M f_M$$

но можно использовать любую.

## Обучаемая мета-модель

$$\hat{y}(x) = G(f_1(x), \dots, f_M(x))$$

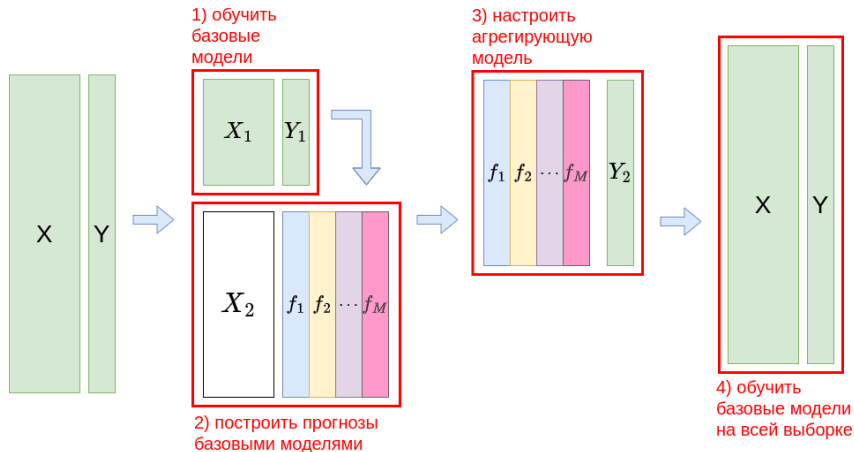
$G(\cdot) = G_w(\cdot)$  - настраиваемая функция, например

$$G(f_1, f_2, \dots, f_M) = w_0 + w_1 f_1 + \dots + w_M f_M$$

но можно использовать любую.

Почему  $G(\cdot)$  нельзя настраивать на той же выборке, что и  $f_1(\cdot), f_2(\cdot), \dots, f_M(\cdot)$ ?

# Блендинг



## Алгоритм стэкинга

Алгоритм стэкинга:

- 1 Инициализируем обучающую выборку  $G(\cdot)$ :  $T' = \{\}$
- 2 Разобьем обучающую выборку  $T = \{(x_n, y_n)\}_{n=1}^N$  на  $K$  блоков:  $T_1, T_2, \dots, T_K$ .
- 3 для  $k = 1, 2, \dots, K$  :  
обучим  $f_1(x), \dots, f_M(x)$  на  $T \setminus T_k$   
для каждого  $(x, y) \in T_k$  :  
дополним  $T'$  объектом  $([f_1(x), \dots, f_M(x)], y)$
- 4 Обучим  $G(\cdot)$  на  $T'$ . Добавим к признакам  $T'$  небольшой шум.
- 5 Перенастроим  $f_1(x), \dots, f_M(x)$  на всей  $T$ .

Небольшой шум на шаге 3 добавляется, чтобы выровнять прогнозы моделей, обученных на разных подвыборках.

## Расширения стэкинга

Кроме прогнозов  $f_1(x), \dots, f_M(x)$  агрегирующая функция может зависеть от:

- исходных признаков  $x$ 
  - в разных частях признакового пространства - разная агрегация
- внутренних представлений  $f_m$  (дискриминантных функций, вероятностей).

## Расширения стэкинга

Кроме прогнозов  $f_1(x), \dots, f_M(x)$  агрегирующая функция может зависеть от:

- исходных признаков  $x$ 
  - в разных частях признакового пространства - разная агрегация
- внутренних представлений  $f_m$  (дискриминантных функций, вероятностей).

Вариант стэкинга: построить прогнозы  $K$  версиями каждой базовой модели, получить  $\{f_m^k(x)\}_{k,m}$ , потом усреднить:

$$\hat{y}(x) = \frac{1}{K} \sum_{k=1}^K G(f_1^k(x), f_2^k(x), \dots, f_M^k(x))$$



## Линейный стэкинг

- Линейный стэкинг:

$$f(x) = \sum_{m=1}^M w_m f_m(x)$$

$$\sum_{n=1}^N \left( \sum_{m=1}^M w_m f_m(x_n) - y_n \right)^2 \rightarrow \min_w$$

- $f_1(x), \dots, f_M(x)$  зависимы (предсказывают один и тот же  $y$ )  
=> нестабильная оценка.
- Более устойчивая оценка:

$$\begin{cases} \sum_{n=1}^N \left( \sum_{m=1}^M w_m f_m(x_n) - y_n \right)^2 + \lambda \sum_{m=1}^M \left( w_m - \frac{1}{M} \right)^2 \rightarrow \min_w \\ w_1 \geq 0, \dots, w_M \geq 0 \end{cases}$$

## Использование стэкинга (регрессия) 1

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Ridge
from sklearn.ensemble import StackingRegressor
from sklearn.metrics import accuracy_score
from sklearn.metrics import brier_score_loss
```

```
X_train, X_test, Y_train, Y_test =
    get_demo_classification_data()
```

```
# Инициализируем базовые модели
```

```
knn = KNeighborsRegressor(n_neighbors=100)
tree_model = DecisionTreeRegressor()
```

## Использование стэкинга (регрессия) 2

```
# Инициализируем стэкинг для регрессии
ensemble = StackingRegressor(
    estimators=
        [ ('K nearest neighbors', knn),
          ('decision tree', tree_model) ],
    final_estimator=Ridge(),
    cv=3,           # количество блоков кросс-валидации
    n_jobs=-1)      # используем все ядра процессора

# обучение базовых и мета-моделей:
ensemble.fit(X_train, Y_train)
Y_hat = ensemble.predict(X_test) # построение прогнозов
print(f'Средний модуль ошибки (MAE): \
      {mean_absolute_error(Y_test, Y_hat):.2f}')
```

## Использование стэкинга (классификация) 1

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import StackingClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import brier_score_loss
```

```
X_train, X_test, Y_train, Y_test =
    get_demo_classification_data()
```

```
# Инициализируем базовые модели
```

```
knn = KNeighborsClassifier(n_neighbors=100)
```

```
tree_model = DecisionTreeClassifier()
```

## Использование стэкинга (классификация) 2

```
# Инициализируем стэкинг для классификации
ensemble = StackingClassifier(
    estimators=[('K nearest neighbors', knn),
                ('decision tree', tree_model)],
    final_estimator=LogisticRegression(),
    cv=3,          # количество блоков кросс-валидации
    n_jobs=-1)     # используем все ядра процессора

# обучение базовых и мета-моделей:
ensemble.fit(X_train, Y_train)
Y_hat = ensemble.predict(X_test) # построение прогнозов
print(f'Точность прогнозов: \
      {100*accuracy_score(Y_test, Y_hat):.1f}%')

P_hat = ensemble.predict_proba(X_test) # вер-ти классов
loss = brier_score_loss(Y_test, P_hat[:,1])
print(f'Ошибка Бриепа: {loss:.2f}')
```

Больше информации. [Полный код.](#)