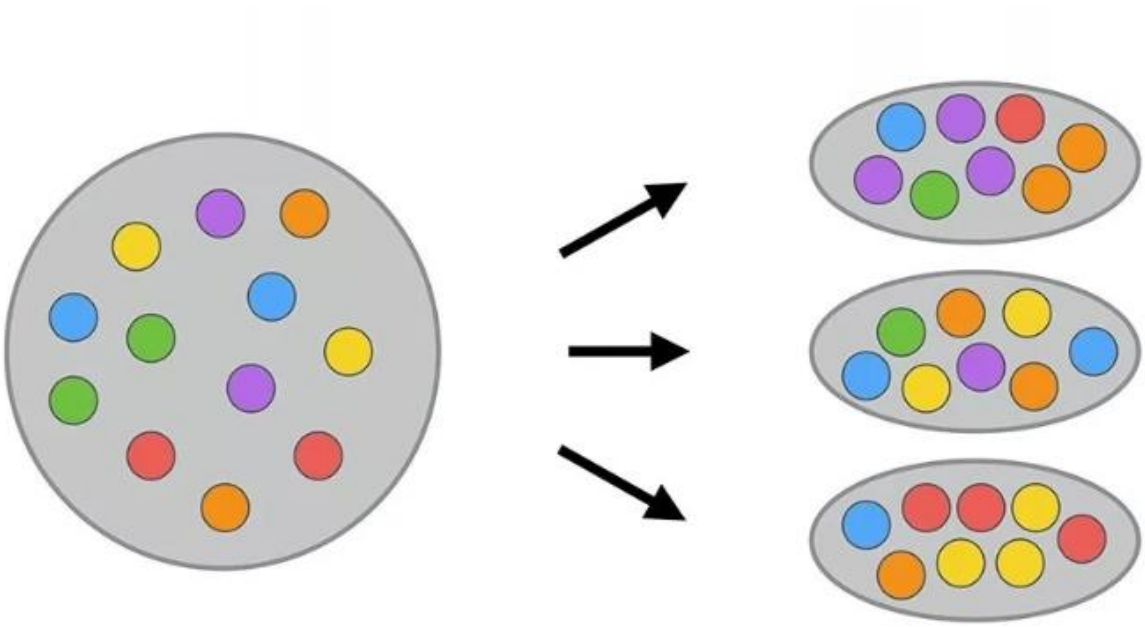
Ансамблевые методы

Бутстрап

Бутстрап – это статистический метод, который используется для оценки надежности и распределения статистик на основе многократного случайного отбора подвыборок с возвращением из исходной выборки. Каждая такая подвыборка имеет тот же размер, что и исходная, и может содержать повторяющиеся элементы. Таким образом, в полученной в конечном итоге бутстрап-выборке некоторые элементы исходной выборки будут встречаться несколько раз, а некоторые (примерно 37% выборки) будут вовсе отсутствовать, и при повторении N раз мы получим N разных выборок длиной l. Например, если у нас есть исходная выборка вида [a, b, c, d, e], возможными бутстрап-выборками могут быть [a, b, a, c, b] или [b, e, e, d, b] и т.д.Это позволяет моделировать вариативность оценок без предположений о распределении данных.



Главное преимущество бутстрапа – возможность построения доверительных интервалов, оценки ошибки и других характеристик статистик даже при небольшом объёме данных или неизвестных распределениях. Метод широко применяется в экономике, биостатистике и машинном обучении для анализа устойчивости моделей.

В задачах машинного обучения бутстрап лежит в основе алгоритмов ансамблевых методов, например, бэггинга. Здесь на разных бутстрап-выборках параллельно обучаются несколько моделей, а их результаты объединяются для повышения точности и устойчивости итогового прогноза.

Бэггинг

Бэггинг (от англ. bootstrap aggregating, бутстрэп-агрегирование) – это метод ансамблевого обучения в машинном обучении, направленный на повышение стабильности и точности моделей. Суть метода в следующем:

Из исходного набора данных многократно создаются случайные подвыборки с возвращением (бутстрэп-сэмплинг), каждая из которых используется для обучения отдельной базовой модели.

Все модели обучаются параллельно и независимо друг от друга на этих подвыборках.

Результаты моделей объединяются (агрегируются) путём голосования при классификации или усреднения при регрессии, чтобы получить итоговое предсказание.

Бэггинг помогает уменьшить разброс (дисперсию) моделей, снижая переобучение и повышая общую устойчивость и точность предсказаний. Он особенно эффективен для алгоритмов с высокой вариативностью, таких как деревья решений. Примером бэггинга является алгоритм случайного леса, где много деревьев обучаются на разных случайных подвыборках и случайных подмножествах признаков.

Основные преимущества бэггинга:

Уменьшение дисперсии ошибки.

Повышение точности по сравнению с одиночной моделью (на 10-40%).

Возможность параллельного обучения базовых моделей.

Универсальность – можно применять к разным алгоритмам.

Недостатки:

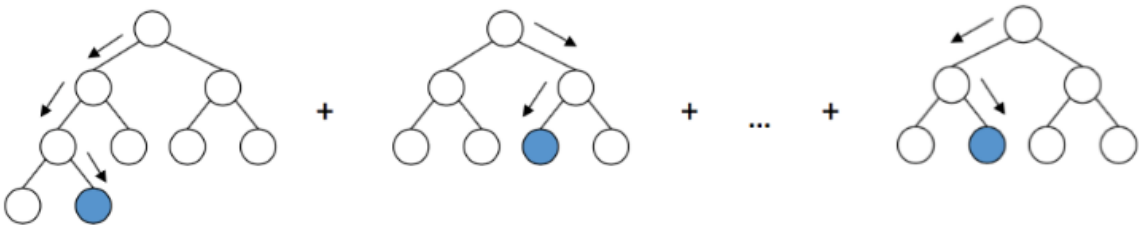
Результат зависит от случайного формирования выборок (недетерминированность).

Иногда сложнее интерпретировать итоговую модель.

Нет жёсткой математической гарантии улучшения точности, но на практике метод хорошо зарекомендовал себя.

Случайный лес

Случайный лес (Random Forest) – это ансамблевый алгоритм машинного обучения, который строит множество деревьев решений на разных случайных подвыборках данных и объединяет их результаты для повышения точности и устойчивости предсказаний.



Алгоритм работы случайного леса:

1. Сначала из исходного обучающего набора формируются случайные выборки с возвращением (бутстрап-выборки).
2. Для каждой такой выборки строится свое дерево решений. При построении дерева в каждом узле случайным образом выбирается ограниченное число признаков. Есть некоторые практически рекомендации по построению случайных лесов: в задачах классификации рекомендуется брать $m = \sqrt{d}$, где d - общее число признаков, и строить дерево до тех пор, пока в каждом листе не останется по одному объекту, а в задаче регрессии принимать m=d/3 и строить дерево, пока в листьях не останется по пять объектов. Далее построенные деревья объединяются в композицию, и при предсказаниях с его помощью используется усредненный ответ на каждом дереве.
3. Каждое дерево обучается независимо и выдает свой прогноз.
4. Для классификации итоговое решение принимается на основе голосования большинства деревьев; для регрессии – усреднением предсказаний.

Этот подход снижает переобучение и уменьшает разброс ошибки, так как деревья разнообразны и агрегированы вместе дают более стабильный результат.

Метод случайных подпространств – это техника, непосредственно связанная со случайным лесом и другими ансамблевыми методами, которая заключается в том, что для построения каждой базовой модели (например, дерева решения) используется случайное подмножество признаков (признаковое подпространство).

Сравнение с обычным построением дерева

В классическом дереве решений на каждом шаге выбирается лучший признак из всех.

В методе случайных подпространств на каждом шаге рассматривается только случайный ограниченный набор признаков, что способствует большей

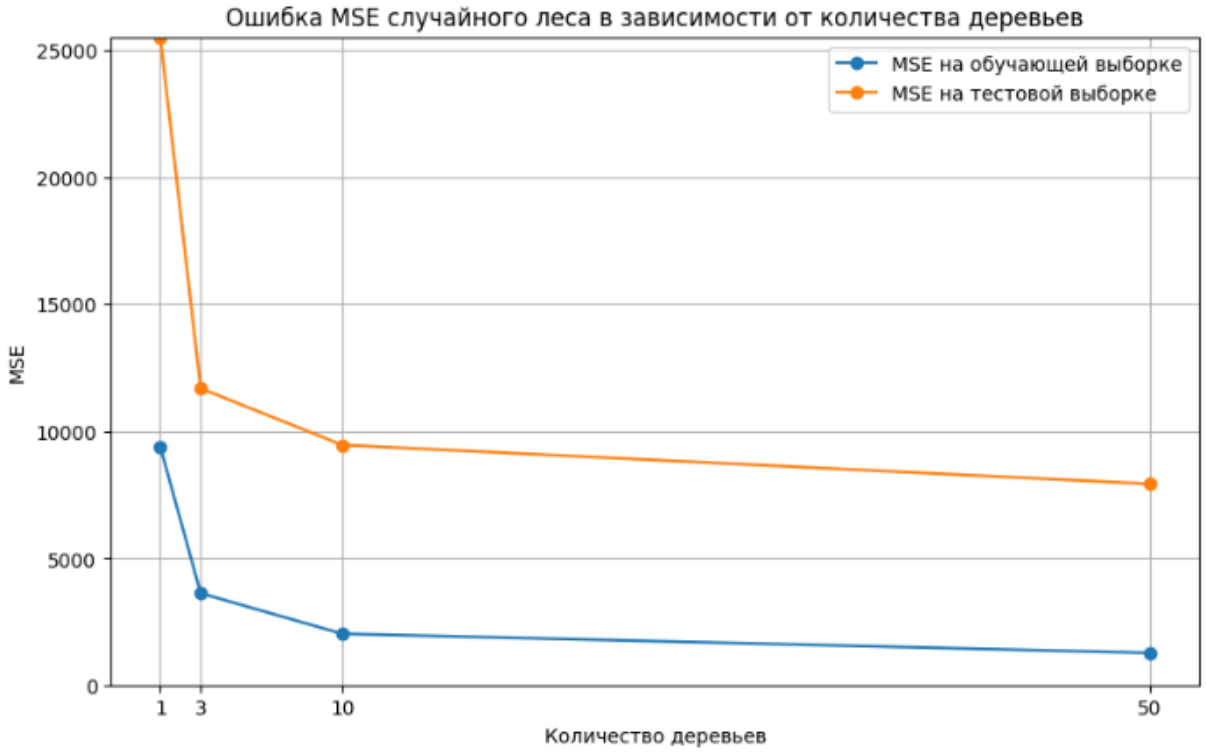
вариативности деревьев и меньшей коррелированности моделей, улучшая тем самым качество ансамбля.

Таким образом, случайный лес использует бутстрап-выборки для данных и метод случайных подпространств для выбора признаков в каждом дереве, создавая множество разнообразных деревьев, чьи прогнозы объединяются в единую итоговую модель. Это приводит к высокой точности, устойчивости к переобучению и хорошей способности моделировать сложные зависимости в данных.

Пример. Построим несколько моделей для регрессии с разным количеством деревьев и сравним.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Создание синтетического набора данных для регрессии
X, y = make_regression(n_samples=1000, n_features=20, noise=0.1,
random_state=42)

# Разделение на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Количество деревьев для обучения
n_trees = [1, 3, 10, 50]
train_mse = []
test_mse = []

# Обучение модели и вычисление MSE
for n in n_trees:
    model = RandomForestRegressor(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)

    # Прогнозирование
    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)

    # Вычисление MSE
    train_error = mean_squared_error(y_train, train_pred)
    test_error = mean_squared_error(y_test, test_pred)

    train_mse.append(train_error)
    test_mse.append(test_error)

# Построение графиков
plt.figure(figsize=(10, 6))
plt.plot(n_trees, train_mse, marker='o', label='MSE на обучающей выборке')
plt.plot(n_trees, test_mse, marker='o', label='MSE на тестовой выборке')
plt.title('Ошибка MSE случайного леса в зависимости от количества деревьев')
plt.xlabel('Количество деревьев')
```

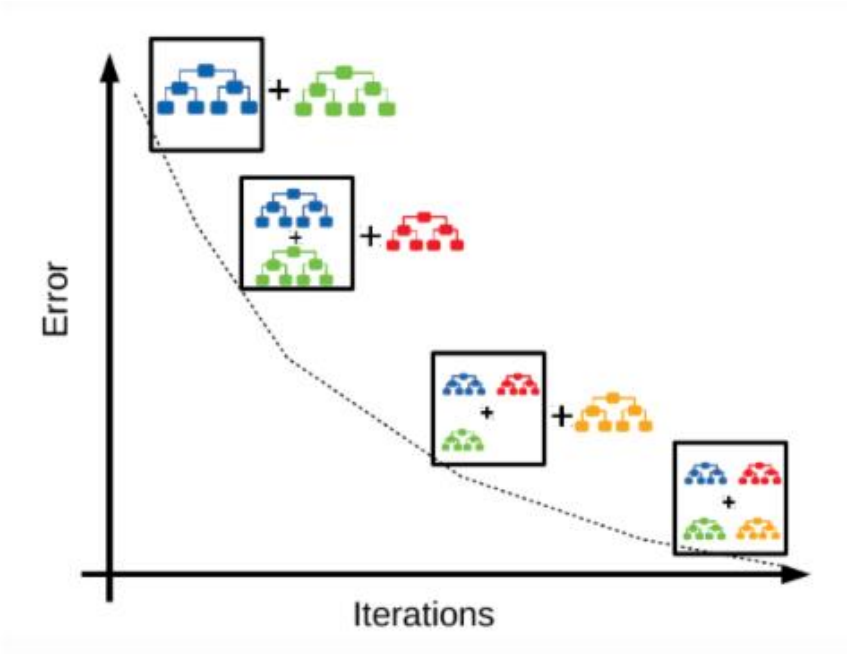
```
plt.ylabel('MSE')
plt.xticks(n_trees)
plt.ylim(0, max(max(train_mse), max(test_mse)) + 1)
plt.legend()
plt.grid()
plt.show()
```



Бустинг

Бустинг – это ансамблевый метод машинного обучения, направленный на создание сильной модели путем последовательного объединения множества слабых моделей. Его основная идея заключается в том, что каждая следующая модель обучается на данных с учетом ошибок, допущенных предыдущими моделями, причем объектам с неправильной классификацией в предыдущих итерациях присваиваются больший вес. В результате достигается постепенное уменьшение ошибок и повышение точности итогового предсказания.

В процессе бустинга модели обучаются поочерёдно: сначала обучается первая слабая модель (например, дерево решений), затем ошибки этой модели выделяются и используются для обучения следующей модели, которая старается лучше исправить ошибки предыдущей. Таким образом, каждое новое слабое классификатор или регрессор последовательно улучшает качество предсказаний за счет фокусирования на трудных для классификации объектах.



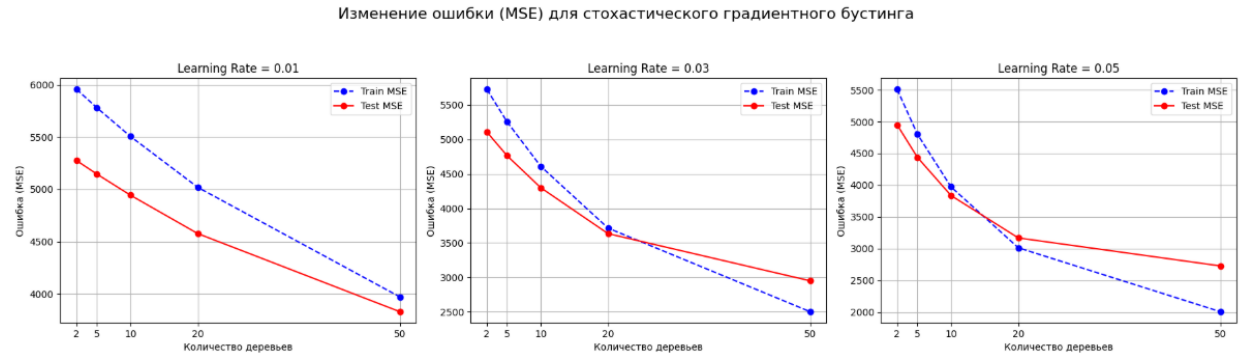
Отличие бустинга от других ансамблевых методов, например бэггинга, заключается в последовательном обучении моделей, а не параллельном. В бэггинге базовые модели обучаются независимо на разных случайных подвыборках, а в бустинге каждая модель зависит от результатов предыдущих, что позволяет более эффективно уменьшать смещение (bias) и улучшать способность к обобщению.

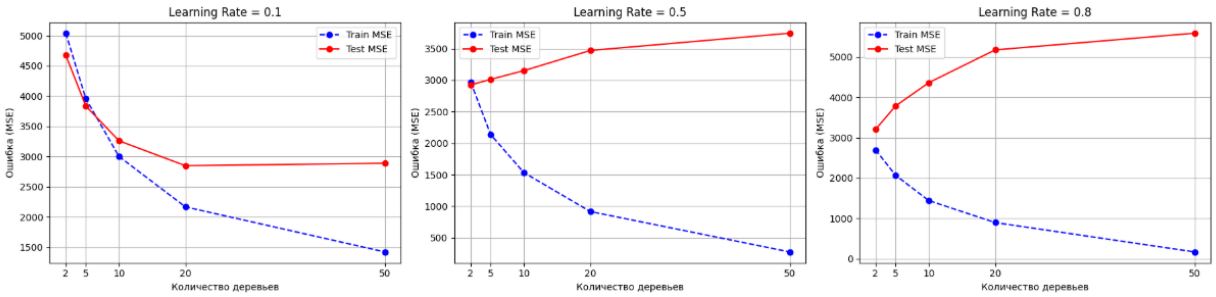
Основные преимущества бустинга – это высокая точность и способность преобразовывать слабые, порой почти случайные модели в сильные предсказатели. Он прост в реализации и находит широкое применение в задачах классификации и регрессии, особенно с использованием таких алгоритмов как AdaBoost и градиентный бустинг. Однако при неосторожной настройке бустинг может привести к переобучению, поэтому требует грамотной кросс-валидации и выбора параметров.

Пример. Реализовать градиентный бустинг с разными скоростями обучения.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
# Загрузка датасета
data = load_diabetes()
X = data.data
y = data.target
# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Количество деревьев и скорости обучения
```

```
n_trees = [2, 5, 10, 20, 50]
learning_rates = [0.01, 0.03, 0.05, 0.1, 0.5, 0.8]  # Скорости обучения
max_depth = 3  # Уровень глубины дерева
subsample = 0.8  # Доля обучающей выборки для стохастического градиентного бустинга

# Создание фигуры и подграфиков
fig, axs = plt.subplots(2, 3, figsize=(18, 10))
axs = axs.flatten()  # Упрощаем индексацию подграфиков

# Обучение моделей и вычисление MSE для каждой комбинации скорости обучения
for idx, lr in enumerate(learning_rates):
    train_errors = []
    test_errors = []

    for n in n_trees:
        model = GradientBoostingRegressor(n_estimators=n, max_depth=max_depth, learning_rate=lr, subsample=subsample, random_state=42)
        model.fit(X_train, y_train)

        # Прогнозирование
        y_train_pred = model.predict(X_train)
        y_test_pred = model.predict(X_test)

        # Вычисление MSE
        train_error = mean_squared_error(y_train, y_train_pred)
        test_error = mean_squared_error(y_test, y_test_pred)

        train_errors.append(train_error)
        test_errors.append(test_error)

    # Построение графиков для каждой скорости обучения
    axs[idx].plot(n_trees, train_errors, marker='o', label='Train MSE', linestyle='--', color='blue')
    axs[idx].plot(n_trees, test_errors, marker='o', label='Test MSE', linestyle='-', color='red')
    axs[idx].set_title(f'Learning Rate = {lr}')
    axs[idx].set_xlabel('Количество деревьев')
    axs[idx].set_ylabel('Ошибка (MSE)')
    axs[idx].set_xticks(n_trees)
    axs[idx].grid()
    axs[idx].legend()

plt.suptitle('Изменение ошибки (MSE) для стохастического градиентного бустинга', fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Подгонка под заголовок
plt.show()
```





Видно, что такой гиперпараметр, как скорость обучения подбирается эмпирически, так же как и количество алгоритмов, вложенность деревьев и др.