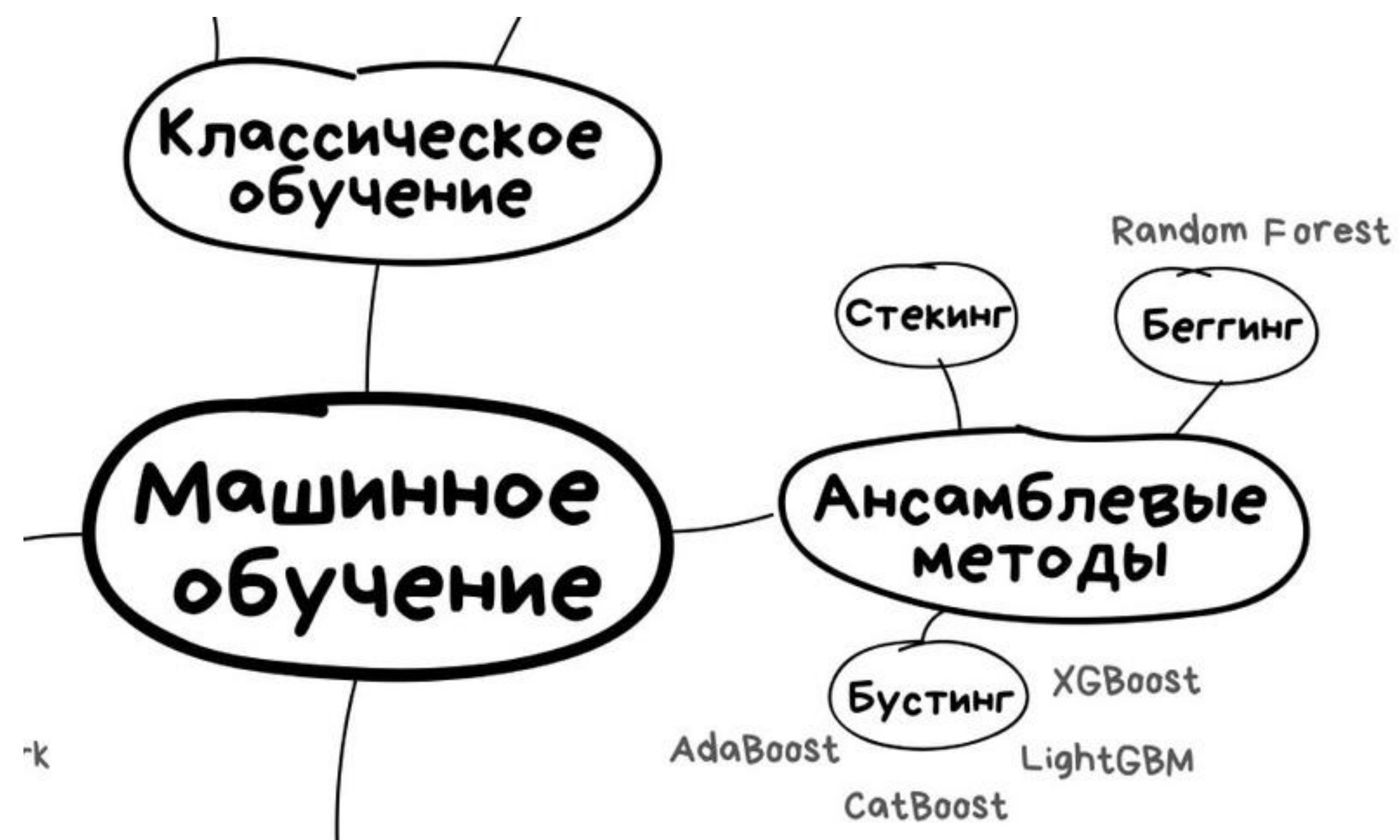


Градиентный бустинг

This is just the beginning of something big.

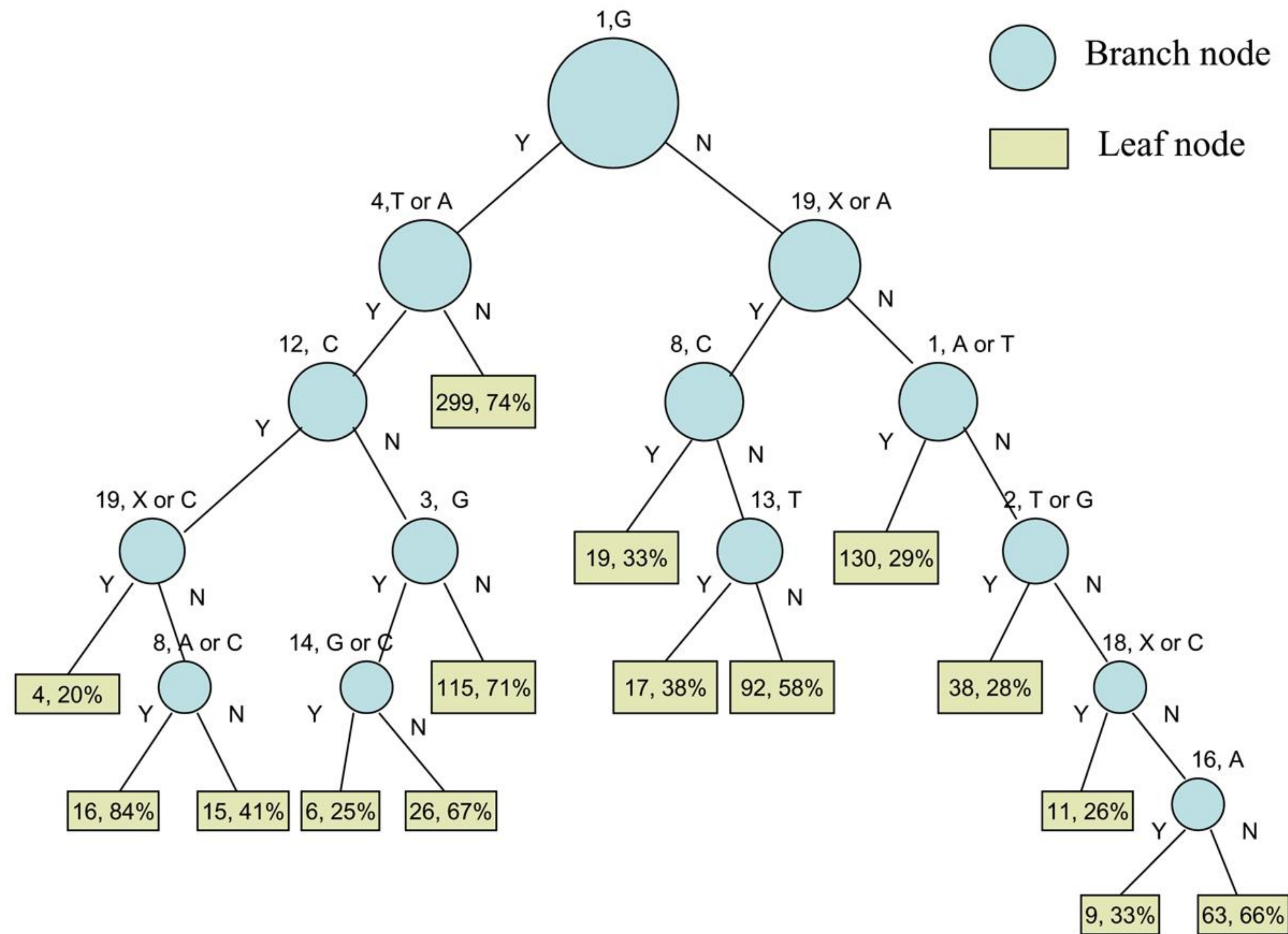




“стакать xgboost-ы”

Бустинг — это метод ансамблевого обучения

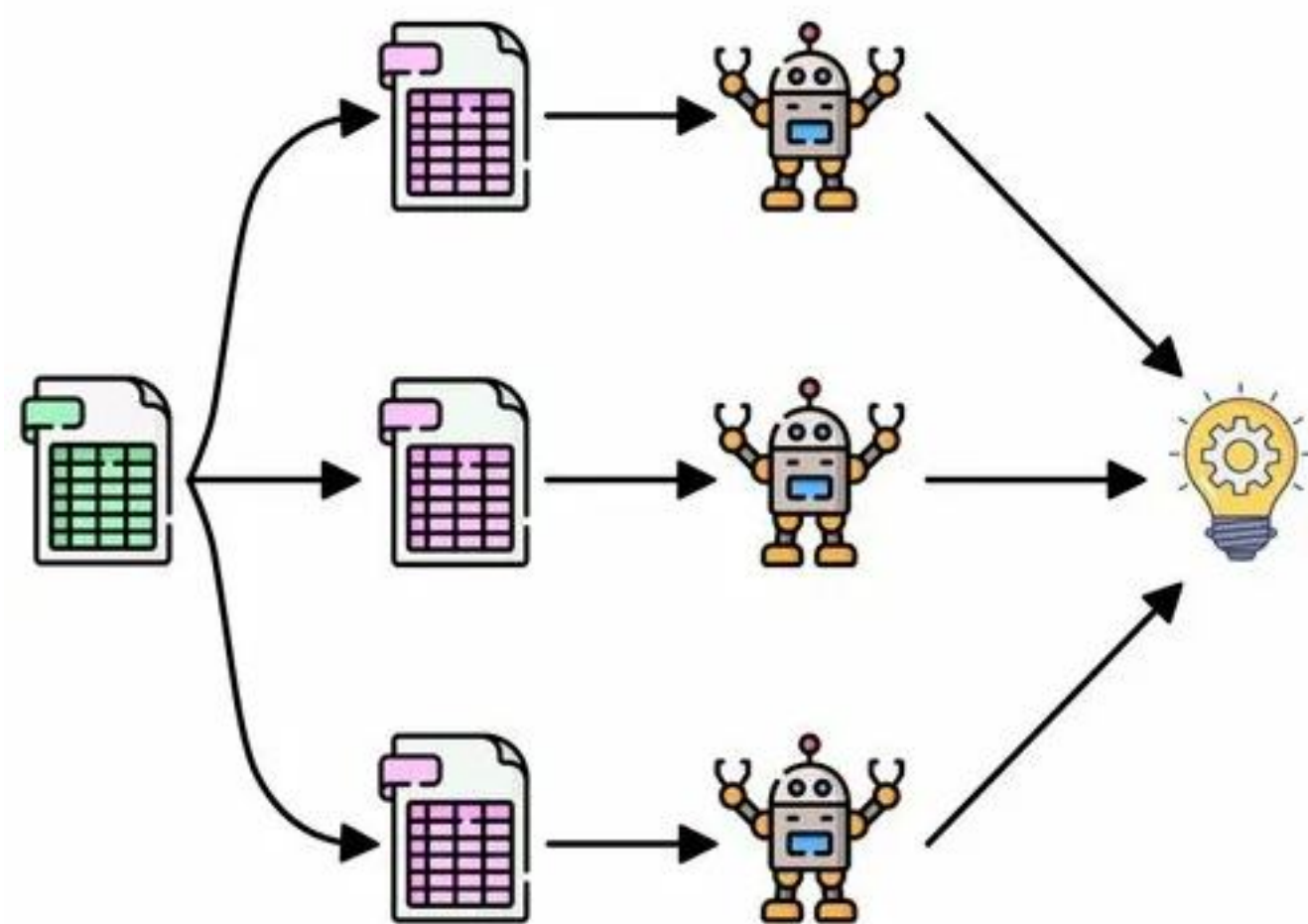




Последовательность улучшений: Создается серия простых моделей (обычно деревьев решений), каждая из которых исправляет ошибки предыдущей модели.

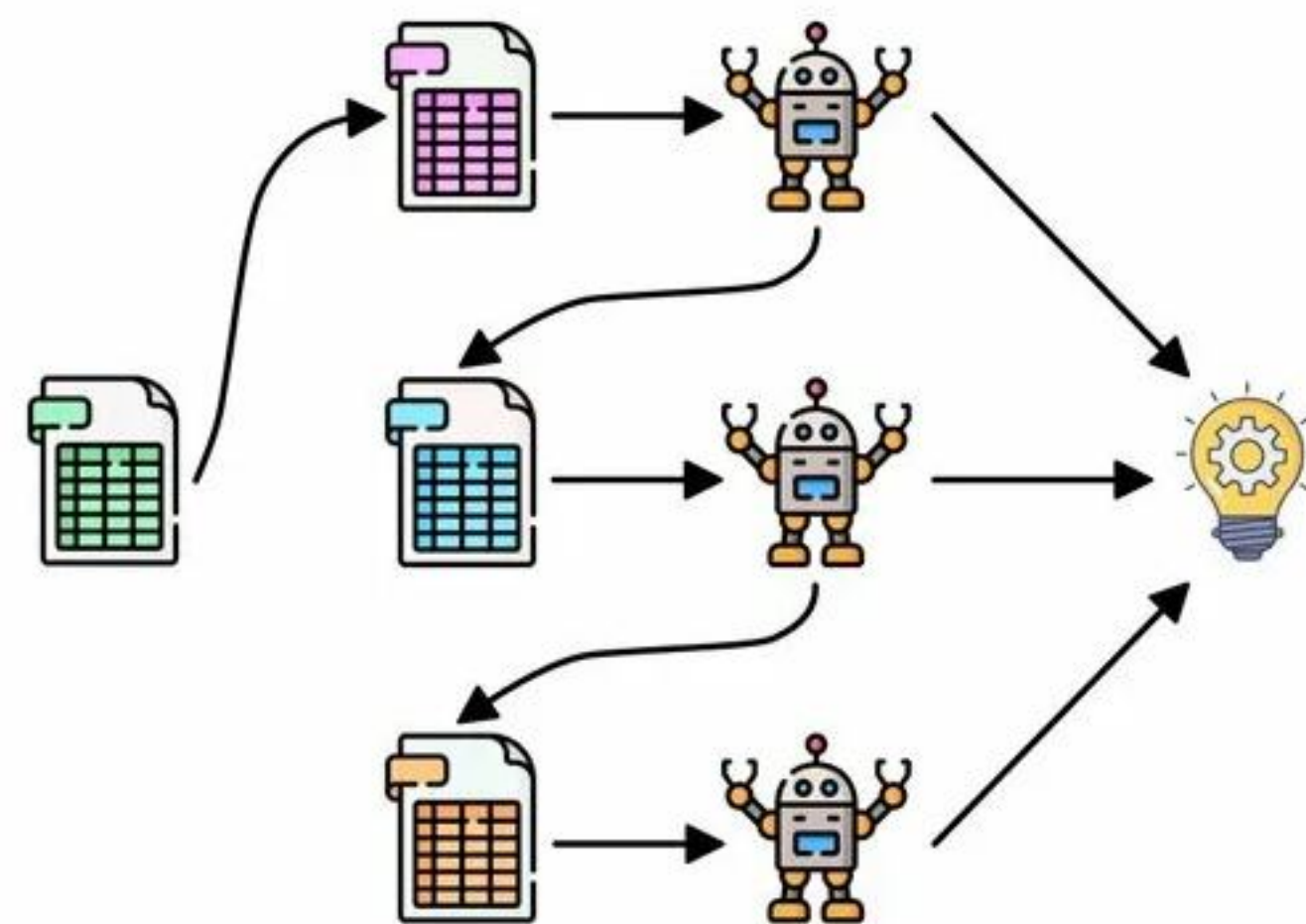
При каждой итерации рассчитывается остаточная ошибка (или **градиент**), которую следует минимизировать.

Bagging



Parallel

Boosting



Sequential

$$L(y, F(x))$$

y — истинное значение

$F(x)$ — предсказание

На каждом шаге алгоритм:

Вычисляет градиент (частные производные) функции потерь по предсказаниям текущей модели.

Строит новое дерево, чтобы корректировать ошибки на основе этого градиента.

Добавляет это дерево к существующей модели с некоторым весом (обычно малым, характеризуемым параметром learning rate).

Градиентный бустинг имеет несколько важных гиперпараметров, которые значительно влияют на его производительность:

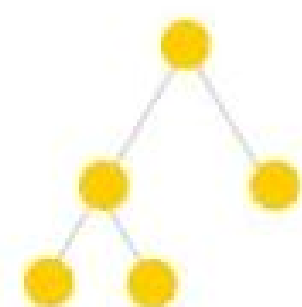
Количество деревьев: Чем больше деревьев, тем лучше модель может подстраиваться под данные, но это может привести к переобучению.


Глубина деревьев: Мелкие деревья обобщают лучше, но могут не уловить сложные закономерности.

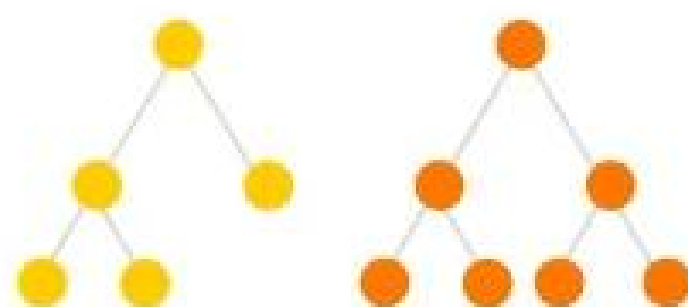
Скорость обучения (learning rate): Контролирует долю, которую каждое дополнительное дерево вносит в итоговую модель. Меньшие значения ведут к лучшей обобщающей способности, но требуют больше деревьев.

Размер подвыборки: Процент данных, которые используются для обучения каждого дерева. Может уменьшить переобучение.

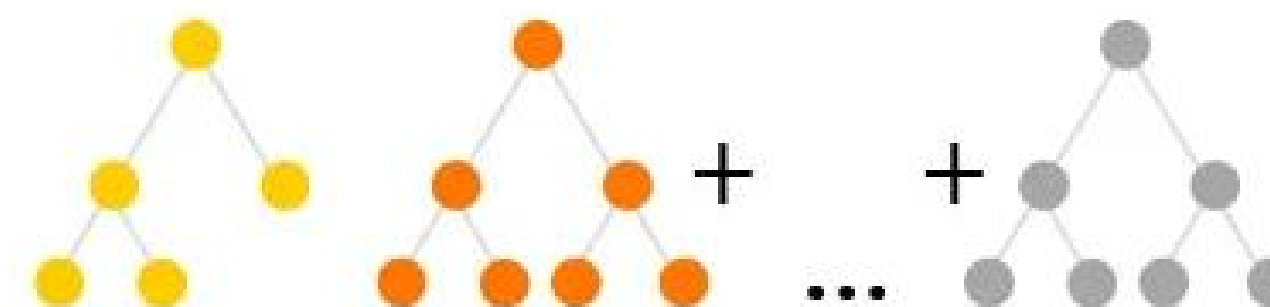
Градиентный бустинг




Ошибка




Ошибка




Ошибка


```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Загрузка данных
iris = load_iris()
X, y = iris.data, iris.target

# Разделение 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)

# Стандартизация данных
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

#1 Что подарить девушке датасаентисту?



Обычные Ирисы



Iris setosa



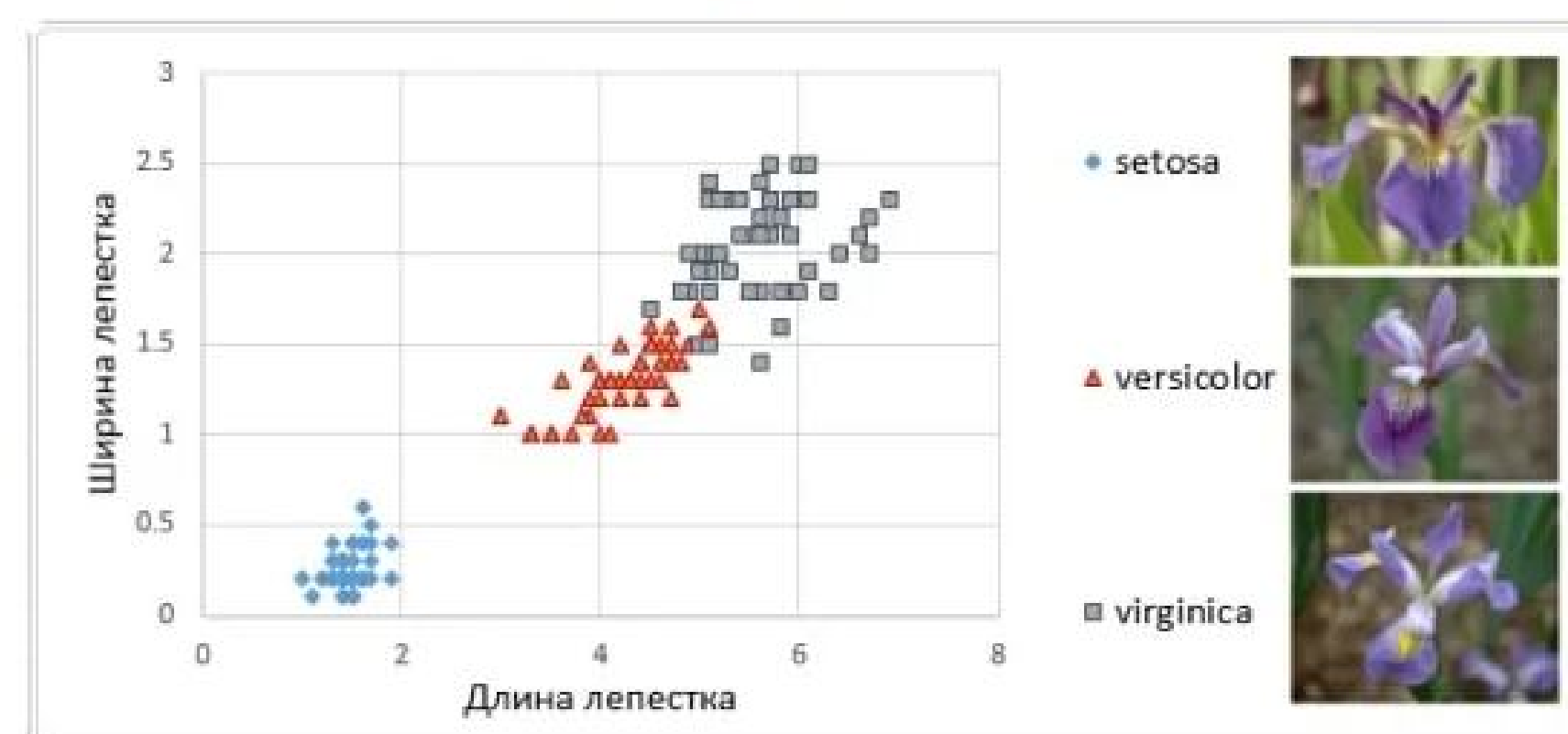
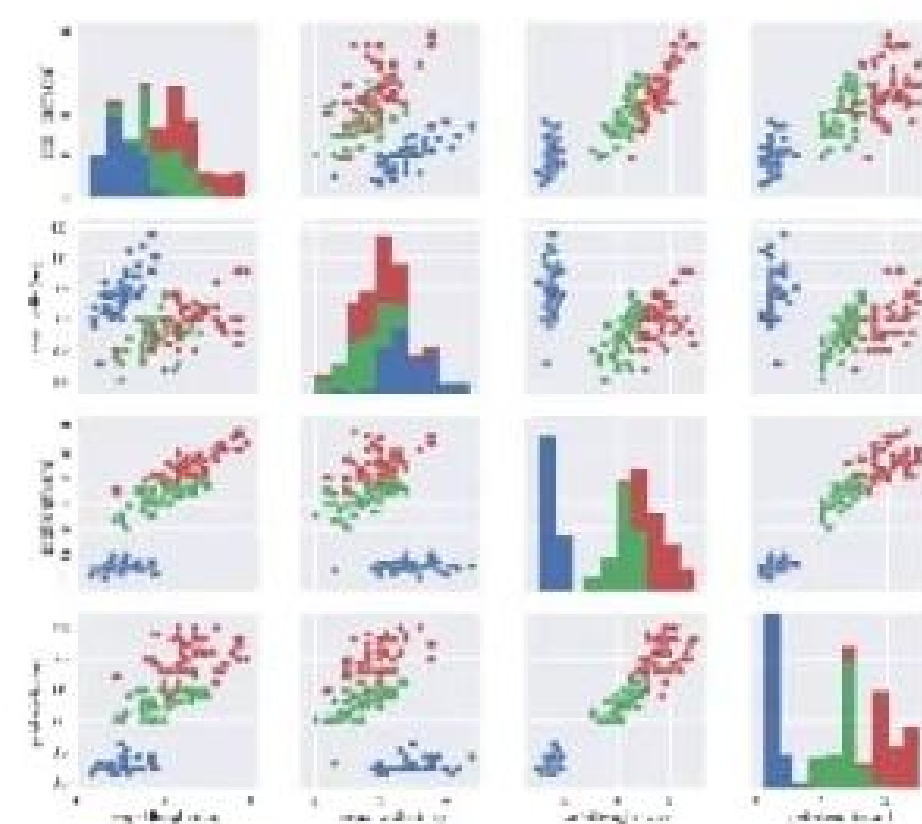
Iris virginica



Iris versicolor



Ирисы Фишера



Ирисы Фишера				
Длина чашелистика	Ширина чашелистика	Длина лепестка	Ширина лепестка	Вид ириса
5.1	3.5	1.4	0.2	<i>setosa</i>
4.9	3.0	1.4	0.2	<i>setosa</i>
4.7	3.2	1.3	0.2	<i>setosa</i>
4.6	3.1	1.5	0.2	<i>setosa</i>
5.0	3.6	1.4	0.2	<i>setosa</i>
5.4	3.9	1.7	0.4	<i>setosa</i>
4.6	3.4	1.4	0.3	<i>setosa</i>
5.0	3.4	1.5	0.2	<i>setosa</i>
4.4	2.9	1.4	0.2	<i>setosa</i>
4.9	3.1	1.5	0.1	<i>setosa</i>
5.4	3.7	1.5	0.2	<i>setosa</i>
4.8	3.4	1.6	0.2	<i>setosa</i>
4.8	3.0	1.4	0.1	<i>setosa</i>



```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

# Создание и обучение модели
gb_model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
gb_model.fit(X_train, y_train)

# Оценка точности модели
y_pred = gb_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
import matplotlib.pyplot as plt

# Извлечение важности признаков
feature_importance = gb_model.feature_importances_

# Визуализация важности признаков
plt.barh(iris.feature_names, feature_importance)
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance in Gradient Boosting Model')
plt.show()
```



```
from sklearn.model_selection import GridSearchCV

# Настройка гиперпараметров
param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5]
}

grid_search = GridSearchCV(GradientBoostingClassifier(random_state=42), param_grid, cv=3, n_jobs=-1, verbose=1)
grid_search.fit(X_train, y_train)

print("Best parameters found by GridSearchCV:")
print(grid_search.best_params_)

# Оценка с оптимальными параметрами
best_gb_model = grid_search.best_estimator_
best_y_pred = best_gb_model.predict(X_test)
best_accuracy = accuracy_score(y_test, best_y_pred)
print(f"Accuracy with best parameters: {best_accuracy:.2f}")
```



```
import xgboost as xgb
```

```
# Обучение модели XGBoost
```

```
xgb_model = xgb.XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
```

```
xgb_model.fit(X_train, y_train)
```

```
# Оценка точности модели XGBoost
```

```
xgb_y_pred = xgb_model.predict(X_test)
```

```
xgb_accuracy = accuracy_score(y_test, xgb_y_pred)
```

```
print(f"XGBoost Accuracy: {xgb_accuracy:.2f}")
```

Gradient Boosting Accuracy (scikit-learn): 0.97
Optimized Gradient Boosting Accuracy: 0.98
XGBoost Accuracy: 0.98



CatBoost

Работа с категориальными признаками: CatBoost может автоматически обрабатывать категориальные признаки, превращая их в числовые представления. Это одна из ключевых причин использования CatBoost для данных с множеством категориальных переменных.

Стабильность и высокая скорость: CatBoost обеспечивает стабильные результаты и часто показывает высокую производительность без необходимости тщательной настройки гиперпараметров.

Уменьшение переобучения: CatBoost используется механизм обучения, который снижает вероятность переобучения по сравнению с другими библиотеками бустинга.

Отказоустойчивость: Дает возможности вставки обработки ошибок и восстановления, что может быть полезно при работе с большими данными.

```
from catboost import CatBoostClassifier
from sklearn.metrics import accuracy_score

# Создание и обучение модели CatBoost
catboost_model = CatBoostClassifier(iterations=100, learning_rate=0.1, depth=3, random_seed=42, verbose=0)
catboost_model.fit(X_train, y_train)

# Оценка точности модели CatBoost
catboost_y_pred = catboost_model.predict(X_test)
catboost_accuracy = accuracy_score(y_test, catboost_y_pred)
print(f"CatBoost Accuracy: {catboost_accuracy:.2f}")
```

CatBoost Accuracy: 0.98

$$\hat{f}(x) = \arg \min_{f(x)} \mathbb{E}_{x,y}[L(y, f(x))]$$

$$\hat{\theta} = \sum_{i=1}^M \hat{\theta}_i,$$

$$L_{\theta}(\hat{\theta}) = \sum_{i=1}^N L(y_i, f(x_i, \hat{\theta}))$$

$$\begin{aligned} \hat{f}(x) &= f(x, \hat{\theta}), \\ \hat{\theta} &= \arg \min_{\theta} \mathbb{E}_{x,y}[L(y, f(x, \theta))]\end{aligned}$$



$$\begin{aligned} \hat{f}(x) &= \sum_{i=0}^{t-1} \hat{f}_i(x), \\ r_{it} &= -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}(x)}, \quad \text{for } i = 1, \dots, n, \\ \theta_t &= \arg \min_{\theta} \sum_{i=1}^n (r_{it} - h(x_i, \theta))^2, \\ \rho_t &= \arg \min_{\rho} \sum_{i=1}^n L(y_i, \hat{f}(x_i) + \rho \cdot h(x_i, \theta_t))\end{aligned}$$

$$\begin{aligned} \hat{f}(x) &= \sum_{i=0}^{t-1} \hat{f}_i(x), \\ (\rho_t, \theta_t) &= \arg \min_{\rho, \theta} \mathbb{E}_{x,y}[L(y, \hat{f}(x) + \rho \cdot h(x, \theta))], \\ \hat{f}_t(x) &= \rho_t \cdot h(x, \theta_t)\end{aligned}$$



1. Инициализировать начальное приближение параметров $\hat{\theta} = \hat{\theta}_0$

2. Для каждой итерации $t = 1, \dots, M$ повторять:

1. Посчитать градиент функции потерь $\nabla L_{\theta}(\hat{\theta})$ при текущем приближении $\hat{\theta}$

$$\nabla L_{\theta}(\hat{\theta}) = \left[\frac{\partial L(y, f(x, \theta))}{\partial \theta} \right]_{\theta=\hat{\theta}}$$

2. Задать текущее итеративное приближение $\hat{\theta}_t$ на основе посчитанного градиента

$$\hat{\theta}_t \leftarrow -\nabla L_{\theta}(\hat{\theta})$$

3. Обновить приближение параметров $\hat{\theta}$:

$$\hat{\theta} \leftarrow \hat{\theta} + \hat{\theta}_t = \sum_{i=0}^t \hat{\theta}_i$$

3. Сохранить итоговое приближение $\hat{\theta}$

$$\hat{\theta} = \sum_{i=0}^M \hat{\theta}_i$$

4. Пользоваться найденной функцией $\hat{f}(x) = f(x, \hat{\theta})$ по назначению