Практическая работа 5

Тема: Построение и оптимизация моделей классификации и регрессии с использованием деревьев решений и ансамблей моделей

Цель: освоить методы построения и оценки моделей классификации/регрессии с использованием деревьев решений и ансамблей моделей (стекинг, бэггинг, бустинг) и оценить их производительность

Содержание

ЗАДАНИЕ

1. **Найти и подготовить данные для задачи классификации или регрессии:**
    - Найдите набор данных для решения задачи классификации или регрессии. Данные не должны повторяться между участниками группы.
    - Подготовьте данные для анализа: проверьте их на наличие пропусков, выбросов и при необходимости произведите очистку данных.
2. **Построить дерево решений:**
    - Реализуйте дерево решений для задачи классификации или регрессии с использованием библиотеки scikit-learn.
    - Проведите обучение модели на ваших данных.
3. **Подобрать гиперпараметры дерева решений:**
    - С помощью GridSearchCV выполните подбор оптимальных гиперпараметров для дерева решений (например, глубина дерева, минимальное количество образцов для разделения узла и т.д.).
    - Оцените качество полученной модели с подобранными параметрами с использованием метрик, таких как точность, F1-score, RMSE или R^2 (в зависимости от задачи).
4. **Реализовать ансамбли моделей:**
    - Реализуйте следующие ансамбли моделей:
    - **Бэггинг** (bagging)
    - **Бустинг** (например, с помощью XGBoost, CatBoost или LightGBM)
    - **Стекинг** (stacking)
    - Обучите ансамбли на тех же данных и сравните их с результатами дерева решений.
5. **Оценить качество моделей:**
    - Оцените качество ансамблей моделей с использованием

метрик, аналогичных тем, что использовались для дерева решений.

- Сравните производительность различных ансамблей с базовой моделью дерева решений и сделайте выводы, какая модель лучше решает задачу.

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Шаг 1. Найти и подготовить данные для задачи классификации или регрессии

Теория:

Классификация — это задача машинного обучения, в которой модель должна предсказать категорию (класс) для каждого примера из набора данных. Пример: предсказание того, будет ли клиент банка кредитоспособным (да или нет).

Регрессия — задача предсказания числового значения на основе входных данных. Пример: предсказание цены дома на основе его характеристик.

Подготовка данных:

Очистка данных: Данные часто содержат пропуски, выбросы и ошибки, которые могут негативно повлиять на обучение модели. Методы обработки пропусков включают удаление строк с пропущенными значениями, их замещение средними или наиболее частыми значениями.

Масштабирование данных: Некоторые алгоритмы чувствительны к масштабу данных (например, линейные модели и деревья решений), поэтому важно привести признаки к единому масштабу (например, с помощью стандартизации или нормализации).

Шаг 2. Построить дерево решений для задачи классификации или регрессии

Дерево решений — это один из наиболее интуитивно понятных и популярных алгоритмов машинного обучения. Оно используется как для задач

классификации, так и для регрессии, представляя собой древовидную структуру, где каждый внутренний узел соответствует условию на одном из признаков, ветвление — это результат проверки этого условия, а листовые узлы содержат предсказания модели.

В основе дерева решений лежит принцип “жадного” алгоритма, который на каждом шаге выбирает лучший разбиение данных на основе критериев, таких как Gini, Information Gain для классификации или Mean Squared Error (MSE) для регрессии.

Преимущества и недостатки дерева решений представлены в таблицы 1.

Таблица 1 — Преимущества и недостатки дерева решений

Преимущества	**Недостатки**
Простота интерпретации	Переобучение
Поддержка работы с категориальными и числовыми данными	Чувствительность к изменениям данных
Устойчивость к выбросам	Низкая точность по сравнению с ансамблями

Основные понятия и принципы работы дерева решений:

Древовидная структура:

- Корень дерева — это начальная точка, с которой начинается разбиение данных по определенным признакам;
- Внутренние узлы — каждый узел представляет проверку условия по какому-то признаку (например, “если возраст > 30 лет”);
- Ветви — это возможные исходы проверки условия в узле (например, “Да” или “Нет”);
- Листовые узлы (листья) — конечные узлы, которые содержат предсказание (класс в задаче классификации или значение в задаче регрессии);

Процесс построения дерева решений:

- Алгоритм последовательно выбирает признаки, которые лучше всего делят данные на подгруппы, минимизируя неопределенность или ошибку;

- Это разбиение продолжается до тех пор, пока данные не будут разделены на достаточно однородные подмножества, либо не будет достигнуты определенные ограничения (например, максимальная глубина дерева);

Алгоритмы построения дерева решений:

- ID3 (Iterative Dichotomiser 3) — один из первых алгоритмов построения деревьев, использует критерий энтропии для выбора наилучшего разбиения;
- CART (Classification and Regression Trees) — алгоритм, использующий критерии Gini (для классификации) и среднеквадратическую ошибку (для регрессии);

Важные параметры дерева решений

1. Глубина дерева (max_depth): максимальная глубина дерева, которая контролирует его сложность. Чем больше глубина дерева, тем больше риск переобучения, так как дерево может “запомнить” тренды в данных, которые не обобщаются на новых данных.
2. Минимальное количество образцов для разделения (min_samples_split): определяет минимальное количество данных, которые должны быть в узле, чтобы выполнить разбиение. Это помогает избегать слишком большого количества разбиений на небольших наборах данных.
3. Минимальное количество образцов в листе (min_samples_leaf): минимальное число данных, которые должны остаться в конечном узле (листе). Это также влияет на размер дерева и может предотвратить переобучение.
4. Критерий разбиения (criterion): метрика, которая используется для выбора наилучшего разбиения данных. В задачах классификации чаще всего используется:

- Gini impurity (нечистота Джини): измеряет вероятность того, что выбранный элемент будет неправильно классифицирован, если он будет случайно помечен по распределению классов.
- Entropy (энтропия): измеряет неопределенность или хаотичность в данных. Чем ниже энтропия, тем более однородны данные в узле.

В задачах регрессии используется Mean Squared Error (MSE) — метрика, которая измеряет разницу между реальными значениями и предсказанными значениями дерева.

Пример алгоритма построения дерева решений

Алгоритм можно свести к следующим шагам:

1. Начало с корневого узла: Все данные находятся в одном узле. Алгоритм выбирает признак, по которому разделяются данные, основываясь на критерии, который минимизирует неопределенность (например, Gini impurity).
2. Последующее разбиение: Для каждой ветви алгоритм снова выбирает признак для разделения данных. Этот процесс повторяется рекурсивно.
3. Остановка: Дерево продолжает расти до тех пор, пока не достигнет одного из условий:
   - Все данные в узле принадлежат к одному классу (в задаче классификации);
   - Прогнозируемое значение достаточно точно предсказывается (в задаче регрессии);
   - Лимит по глубине дерева или минимальное количество данных в узле достигнуто;
4. Прогнозирование: после того как дерево обучено, для предсказания модель просто проходит по дереву от корневого узла до листа, проверяя условия на каждом шаге, пока не достигнет листового узла, где

находится ответ (класс или числовое значение).

Оптимизация дерева решений

Для предотвращения переобучения дерева решений применяются различные техники:

- Обрезка (Pruning): это процесс удаления частей дерева, которые вносят мало полезной информации и могут быть результатом “шумовых” данных;
- Подбор гиперпараметров: Параметры, такие как глубина дерева, минимальное количество образцов в узле, могут быть настроены для улучшения производительности модели. Один из популярных методов настройки гиперпараметров — это GridSearchCV, который автоматически находит оптимальные значения гиперпараметров с использованием кросс-валидации;

Пример кода для создания дерева решений на Python представлен в листинге 1

Листинг 1 — Пример кода для создания дерева решений на Python

```
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
# Загрузка данных
iris = load_iris()
X, y = iris.data, iris.target
# Разделение на обучающие и тестовые данные
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
# Создание дерева решений для классификации
clf = tree.DecisionTreeClassifier(max_depth=3)
# Обучение модели
clf.fit(X_train, y_train)
# Прогнозирование
y_pred = clf.predict(X_test)
# Оценка точности
from sklearn.metrics import accuracy_score
print("Accuracy:", accuracy_score(y_test, y_pred))
# Визуализация дерева
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,8))
tree.plot_tree(clf, filled=True)
plt.show()
```

Это код для решения задачи классификации с использованием дерева решений. Вы можете изменить параметры, такие как max_depth или criterion, чтобы лучше понять, как они влияют на производительность модели.

Таким образом, дерево решений — это мощный, но простой инструмент, который может быть эффективно использован как для задач классификации, так и для регрессии

Шаг 3. Подобрать гиперпараметры дерева решений с использованием GridSearchCV

Для того чтобы построить оптимальную модель, необходимо настроить ее гиперпараметры.

Гиперпараметры — это параметры, которые не обучаются на данных, а задаются до начала обучения модели и напрямую влияют на её структуру и производительность. В случае дерева решений это могут быть глубина дерева, минимальное количество образцов в узлах и другие параметры.

Основные гиперпараметры представлены в таблице 2

Таблица 2 — Основные гиперпараметры дерева решений

Гиперпараметры	**Описание**
`max_depth` (максимальная глубина дерева)	Определяет максимальное количество уровней в дереве.
	Чем больше глубина, тем более сложные зависимости может моделировать дерево, но также возрастает риск переобучения
	Если значение не указано, дерево будет расти до тех пор, пока все листья не станут чистыми (однородными) или пока в узле не останется минимальное количество примеров для разбиения
`min_samples_split` (минимальное	Определяет минимальное количество данных, которые должны быть в узле,

количество образцов для разделения узла)	чтобы он мог быть разделен
	Чем выше значение, тем меньше разбиений произойдет, что может помочь избежать переобучения
	Значение может быть как абсолютным числом (например, 10), так и долей от общего количества данных (например, 0.1, что означает 10% от всех данных)
`min_samples_leaf` (минимальное количество образцов в листе)	Устанавливает минимальное количество данных в конечных узлах (листьях)
	Более высокие значения предотвращают создание очень маленьких листьев, что также может снизить риск переобучения
`criterion` (критерий разбиения)	Определяет, какая метрика будет использоваться для оценки качества разделений в узлах дерева
	Для классификации можно использовать: • **Gini** (нечистота Джини) — измеряет степень “неоднородности” данных в узле. • **Entropy** (энтропия) — измеряет уровень неопределенности или “хаотичности” в данных.
	Для регрессии: • **MSE (Mean Squared Error)** — среднеквадратичная ошибка между фактическими и предсказанными значениями.
`max_features` (максимальное количество признаков для поиска разбиений)	Определяет, сколько признаков будет рассмотрено при каждом разделении узла
	Возможные значения: • "auto" — все признаки, • "sqrt" — квадратный корень из общего числа признаков, • "log2" — логарифм по основанию 2 от общего числа признаков, • Число (например, 5) — количество признаков, которые будут случайно выбраны для разбиения.
	Этот параметр особенно полезен для борьбы с переобучением
`max_leaf_nodes` (максимальное количество листьев)	Ограничивает количество конечных узлов (листьев) в дереве
	Это полезно для контроля сложности модели и предотвращения переобучения
`min_impurity_decrease` (минимальное уменьшение нечистоты)	Определяет минимальное значение уменьшения нечистоты, которое должно быть достигнуто при разделении узла
	Если уменьшение нечистоты меньше указанного значения, разбиение не выполняется

Подбор гиперпараметров с использованием GridSearchCV

GridSearchCV — это метод для автоматического подбора оптимальных гиперпараметров модели. Он перебирает все возможные комбинации гиперпараметров из заданной сетки и оценивает качество модели для каждой комбинации с использованием кросс-валидации, и выбирает те, которые дают наилучшие результаты на основе кросс-валидации.

Кросс-валидация — это метод оценки модели, при котором данные разбиваются на несколько подмножеств (фолдов). Модель обучается на одном подмножестве и тестируется на другом. Процесс повторяется для всех возможных разбиений, и результаты усредняются.

Шаги работы GridSearchCV

1. Задайте сетку гиперпараметров для поиска

Определите диапазоны или возможные значения для гиперпараметров модели. Например, для глубины дерева (`max_depth`) это может быть диапазон от 3 до 10, а для критерия разбиения — выбор между Gini и Entropy.

2. Настройте кросс-валидацию

Укажите количество фолдов для кросс-валидации (обычно используют 5 или 10 фолдов). Кросс-валидация разбивает данные на тренировочные и тестовые наборы, что позволяет объективно оценить производительность модели.

3. Запустите процесс поиска

GridSearchCV перебирает все возможные комбинации гиперпараметров, обучая и оценивая модель для каждой комбинации. По окончании работы выбирается лучшая комбинация гиперпараметров.

Пример кода представлен в листинге 2

Листинг 2 — Пример кода использования GridSearchCV с деревом решений

```
from sklearn.datasets import load_iris
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.tree import DecisionTreeClassifier
# Загрузка данных
iris = load_iris()
X, y = iris.data, iris.target
# Разделение данных на тренировочные и тестовые
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Определение модели
dt = DecisionTreeClassifier()
# Определение сетки гиперпараметров
param_grid = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5],
    'criterion': ['gini', 'entropy']
}
# Настройка GridSearchCV
grid_search = GridSearchCV(estimator=dt, param_grid=param_grid, cv=5, scoring='accuracy')
# Обучение и подбор гиперпараметров
grid_search.fit(X_train, y_train)
# Лучшие гиперпараметры
print("Лучшие параметры:", grid_search.best_params_)
# Оценка модели с лучшими параметрами
best_model = grid_search.best_estimator_
print("Точность на тестовых данных:", best_model.score(X_test, y_test))
```

Как работает GridSearchCV

1. `param_grid` задает сетку гиперпараметров, которые будут перебираться.
2. `GridSearchCV` принимает на вход модель (в данном случае дерево решений), сетку гиперпараметров и количество фолдов для кросс-валидации (здесь — 5).
3. `scoring` — метрика, по которой оценивается модель. В данном случае это `accuracy` (точность), но могут быть использованы и другие метрики, например, F1-score или AUC-ROC.

4. По завершении работы `grid_search.best_params_` возвращает комбинацию гиперпараметров, которая дала наилучшие результаты.

Важные замечания

Сложность и время выполнения: чем больше сетка гиперпараметров, тем больше времени займет подбор. Для ускорения можно использовать `RandomizedSearchCV`, который случайным образом выбирает комбинации гиперпараметров для тестирования, вместо перебора всех возможных.

Выбор метрики: важно правильно выбрать метрику, по которой будет оцениваться модель. В зависимости от задачи можно выбрать такие метрики, как точность (`accuracy`), среднеквадратическая ошибка (MSE), или F1-score для несбалансированных данных.

Преимущества использования GridSearchCV

Автоматизация: `GridSearchCV` автоматизирует процесс подбора гиперпараметров, что позволяет значительно сэкономить время.

Кросс-валидация: Использование кросс-валидации делает оценку модели более устойчивой и надежной, так как она тестируется на разных подмножествах данных.

Оптимизация модели: `GridSearchCV` помогает выбрать наилучшие гиперпараметры для конкретных данных, что может существенно улучшить производительность модели.

Таким образом, `GridSearchCV` является важным инструментом для настройки гиперпараметров и повышения точности моделей, таких как дерево решений.

Шаг 4. Реализовать ансамбли моделей (стекинг, бэггинг, бустинг)

Ансамбли моделей — это метод машинного обучения, который заключается в объединении нескольких моделей для создания более точных и надежных предсказаний. Суть ансамблей в том, что комбинация нескольких моделей может исправить ошибки отдельных моделей и улучшить общую производительность. Существует несколько подходов к созданию ансамблей, каждый из которых использует разные стратегии для улучшения предсказаний

1. Бэггинг (Bagging):

Bagging (Bootstrap Aggregating) — это метод, который строит несколько моделей (обычно однотипных, например, деревья решений) на различных подвыборках данных и усредняет их предсказания для улучшения точности и стабильности.

Основная идея — уменьшить дисперсию модели. В отличие от простого дерева решений, которое может быть подвержено переобучению, бэггинг снижает влияние шумов в данных за счет усреднения предсказаний нескольких моделей.

*Пример***:** Random Forest — это типичный пример бэггинга, где множество деревьев решений обучаются на разных подвыборках данных, а их предсказания усредняются (для регрессии) или выбирается наиболее частый класс (для классификации).

Шаги работы бэггинга

1. Создание нескольких подвыборок данных с заменой (bootstrap).
2. Обучение отдельной модели на каждой подвыборке.
3. Усреднение предсказаний всех моделей.

Пример кода представлен в листинге 3.

Листинг 3 — Пример кода бэггинга с использованием Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
# Загрузка данных
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size=0.2, random_state=42)
# Инициализация и обучение Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
# Оценка точности
accuracy = rf.score(X_test, y_test)
print(f"Точность Random Forest: {accuracy}")
```

Преимущества и недостатки представлены в таблице 3.

Таблица 3 — Преимущества и недостатки Бэггинга

Преимущества	**Недостатки**
Уменьшает дисперсию модели, снижая вероятность переобучения	Потребляет больше вычислительных ресурсов, так как требует обучения нескольких моделей
Устойчив к шуму в данных	Может не помочь, если отдельные модели склонны к систематической ошибке (смещение)

2. Бустинг (Boosting)

Boosting — это метод, который обучает модели последовательно, каждая следующая модель исправляет ошибки предыдущей. В отличие от бэггинга, где модели независимы, бустинг делает модели зависимыми друг от друга.

Бустинг уменьшает смещение (bias) модели, но может увеличить дисперсию, поэтому важна правильная настройка гиперпараметров для предотвращения переобучения.

Популярные реализации бустинга:

- XGBoost — библиотека для градиентного бустинга с высокой производительностью.
- CatBoost — оптимизированная библиотека бустинга, особенно хорошо работающая с категориальными данными.
- LightGBM — библиотека для градиентного бустинга, оптимизированная для работы с большими объемами данных.

Шаги работы бустинга:

1. Первая модель обучается на всех данных и делает предсказания.
2. Ошибки этой модели выделяются, и следующая модель фокусируется на них, стараясь их исправить.
3. Процесс повторяется, пока не будет достигнуто определенное количество итераций или не улучшатся метрики.

Пример кода представлен в листинге 4.

Листинг 4 — Пример кода для бустинга с использованием XGBoost

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
# Загрузка данных
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size=0.2, random_state=42)
# Инициализация и обучение XGBoost
xg_clf      =      xgb.XGBClassifier(objective='multi:softmax',
n_estimators=100, seed=42)
xg_clf.fit(X_train, y_train)
# Оценка точности
accuracy = xg_clf.score(X_test, y_test)
print(f"Точность XGBoost: {accuracy}")
```

Преимущества и недостатки бустинга приведены в таблице 4.

Таблица 4 — Преимущества и недостатки бустинга

Преимущества	**Недостатки**
Высокая точность, особенно на сложных задачах с большими объемами данных	Склонность к переобучению, если не контролировать количество итераций
Исправляет ошибки предыдущих моделей, постепенно улучшая качество	Более сложен в настройке, чем бэггинг, особенно в отношении гиперпараметров

4. Стекинг (Stacking):

Stacking — это метод, при котором несколько моделей объединяются таким образом, что их предсказания используются как входные данные для “метамодели” (модель второго уровня), которая и делает окончательное предсказание.

Стекинг может использовать разные типы моделей на первом уровне (например, деревья решений, логистическая регрессия, случайные леса), что делает этот метод очень гибким.

Шаги работы стекинга:

1. Несколько базовых моделей обучаются на исходных данных.
2. Результаты предсказаний этих моделей используются для обучения “метамодели”.
3. Метамодель делает итоговое предсказание на основе предсказаний базовых моделей.

Пример кода приведен в листинге 5.

Листинг 5 — Пример кода для стекинга с использованием библиотеки sklearn:

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
# Загрузка данных
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size=0.2, random_state=42)
# Определение базовых моделей
base_estimators = [
    ('rf',              RandomForestClassifier(n_estimators=100,
random_state=42)),
    ('dt',                   DecisionTreeClassifier(max_depth=3,
random_state=42))
]
# Метамодель
final_estimator = LogisticRegression()
# Создание стекинг-классификатора
stacking_clf  =  StackingClassifier(estimators=base_estimators,
final_estimator=final_estimator)
# Обучение и оценка модели
stacking_clf.fit(X_train, y_train)
accuracy = stacking_clf.score(X_test, y_test)
print(f"Точность стекинг-классификатора: {accuracy}")
```

Преимущества и недостатки представлены в таблице 5.

Таблица 5 — Преимущества и недостатки

Преимущества	**Недостатки**
Повышает точность за счет использования предсказаний нескольких моделей.	Сложность в реализации и настройке
Гибкость, так как можно комбинировать разные типы моделей	Требует больших вычислительных ресурсов, так как необходимо обучать несколько моделей

Сравнение ансамблевых методов

Сравнение ансамблевых методов представлена в таблице 6

Таблица 6 — Сравнение ансамблевых методов

Метод	Описание	Преимущества	Недостатки
Бэггинг	Параллельное обучение нескольких моделей на разных подвыборках данных	Уменьшает дисперсию модели, устойчива к шуму	Не всегда улучшает производительность при систематических ошибках
Бустинг	Последовательное обучение моделей, где каждая исправляет ошибки предыдущей	Высокая точность на сложных данных	Склонность к переобучению, требует тонкой настройки
Стекинг	Комбинация нескольких моделей, где метамодель делает финальное предсказание	Повышает точность за счет использования разных моделей	Сложная реализация, большие вычислительные затраты

Таким образом, каждый из методов ансамблирования может быть полезен в зависимости от данных и задачи. Бэггинг полезен для уменьшения дисперсии и устойчивости к шуму, бустинг помогает исправлять ошибки и добиваться высокой точности, а стекинг позволяет объединить различные модели для достижения лучших результатов.

Шаг 5. Оценить качество моделей

После того как вы построили и обучили несколько моделей (дерево решений и ансамбли моделей), важно оценить их качество. Оценка модели позволяет определить, насколько хорошо она справляется с предсказанием новых, ранее невидимых данных. В машинном обучении для этого используются специальные метрики, которые измеряют, как точно модель решает задачу

Для оценки качества моделей используют метрики, которые зависят от типа задачи.

Accuracy (точность)

Точность — это доля правильно классифицированных примеров от общего числа примеров.

$$Accuracy = \frac{\text{Количество правильных предсказаний}}{\text{Общее количество примеров}}$$

Пример: если из 100 тестовых примеров модель правильно предсказала 90, то точность составит 90%.

Матрица ошибок (Confusion Matrix)

Матрица ошибок показывает количество верных и неверных предсказаний по каждому классу. Это важный инструмент для анализа результатов классификации, особенно если данные несбалансированы.

Пример для бинарной классификации представлен в виде таблицы 7.

Таблица 7 — Пример для бинарной классификации

	Предсказано 1	**Предсказано 0**
Истинное 1	TP (True Positive)	FN (False Negative)
Истинное 0	FP (False Positive)	TN (True Negative)

Precision (Точность)

Precision показывает долю правильных положительных предсказаний среди всех предсказанных положительных классов

$$Precision = \frac{TP}{TP + FP}$$

Используется, когда важно минимизировать ложноположительные предсказания.

Recall (Полнота)

Recall показывает долю правильно предсказанных положительных примеров среди всех истинных положительных примеров.

$$Recall = \frac{TP}{TP + FN}$$

Используется, когда важно минимизировать ложные отрицательные результаты.

F1-score

F1-score — это гармоническое среднее между точностью и полнотой, что делает его полезным, когда данные несбалансированы.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Если нужно балансировать между точностью и полнотой, то эта метрика особенно полезна.

ROC-AUC (Area Under the Curve)

ROC-кривая показывает соотношение между долей ложноположительных и истинноположительных предсказаний при изменении порога классификации.

AUC (Area Under the Curve) — это площадь под ROC-кривой. Чем больше площадь, тем лучше модель.

Пример кода для оценки классификационной модели представлен в листинге 6

Листинг 6 — Пример кода для оценки классификационной модели

```
from sklearn.metrics import accuracy_score, confusion_matrix,
precision_score, recall_score, f1_score, roc_auc_score

# Предсказания модели
y_pred = model.predict(X_test)

# Метрики
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

print(f"Точность: {accuracy}")
print(f"Точность (Precision): {precision}")
print(f"Полнота (Recall): {recall}")
print(f"F1-score: {f1}")

# Матрица ошибок
cm = confusion_matrix(y_test, y_pred)
print(f"Матрица ошибок:\n {cm}")
```

В задаче регрессии результаты — это непрерывные числовые значения, поэтому метрики отличаются от тех, что используются в классификации.

Среднеквадратическая ошибка (Mean Squared Error, MSE)

MSE измеряет среднюю квадратичную разницу между истинными и предсказанными значениями. Представлено на рисунке 1.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Рисунок 1 — MSE

Пример: если модель предсказывает цену дома, MSE измеряет, насколько в среднем предсказанные цены отличаются от истинных.

Среднеквадратичная ошибка (Root Mean Squared Error, RMSE)

Это квадратный корень из MSE. RMSE выражает среднюю ошибку модели в тех же единицах измерения, что и сами предсказания. Представлено на рисунке 2

$$RMSE = \sqrt{MSE}$$

Рисунок 2 — RMSE

Чем ниже значение RMSE, тем лучше модель.

Средняя абсолютная ошибка (Mean Absolute Error, MAE)

MAE измеряет среднюю абсолютную разницу между истинными и предсказанными значениями. Представлено на рисунке 3.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Рисунок 3 — MAE

MAE проще интерпретировать, так как она не квадратична, и её значения выражены в тех же единицах, что и предсказания.

Коэффициент детерминации (R^2)

R^2 измеряет, какую долю изменчивости в данных модель способна объяснить. Оно принимает значения от 0 до 1. Чем ближе значение к 1, тем лучше модель объясняет данные. Представлено на рисунке 4.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}i)^2}{\sum i = 1^n(y_i - \bar{y})^2}$$

Рисунок 4 — R^2

Пример кода для оценки регрессионной модели представлен в листинге 7.

Листинг 7 — Пример кода для оценки регрессионной модели

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Предсказания модели
y_pred = model.predict(X_test)

# Метрики
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Среднеквадратическая ошибка (MSE): {mse}")
print(f"Корень из среднеквадратической ошибки (RMSE): {rmse}")
print(f"Средняя абсолютная ошибка (MAE): {mae}")
print(f"Коэффициент детерминации (R²): {r2}")
```

Выбор правильной метрики

Выбор метрики зависит от задачи и типа данных:

- Для **классификации**: Если классы сбалансированы, можно использовать **accuracy**. Для несбалансированных данных лучше подходят **precision**, **recall** и **F1-score**.
- Для **регрессии**: **RMSE** и **MAE** измеряют ошибки в тех же единицах, что и предсказанные значения, что делает их легко интерпретируемыми. **R^2** помогает понять, насколько хорошо модель объясняет данные.

Сравнение разных моделей

Чтобы оценить и сравнить разные модели, можно построить таблицу с метриками для каждой модели (дерево решений, случайный лес, XGBoost, стекинг и т.д.) и проанализировать, какая модель показала лучшие результаты на тестовых данных. Представлено на рисунке 5.

Модель	Accuracy	Precision	Recall	F1-score	RMSE	MAE	R^2
Decision Tree	0.85	0.82	0.83	0.82	N/A	N/A	N/A
Random Forest	0.90	0.88	0.89	0.89	N/A	N/A	N/A
XGBoost	0.92	0.91	0.92	0.91	N/A	N/A	N/A
Стекинг	0.93	0.92	0.92	0.92	N/A	N/A	N/A

Рисунок 5 — Метрики для каждой модели

Или для регрессии, представлено на рисунке 6

Модель	MSE	RMSE	MAE	R^2
Decision Tree	2.34	1.53	1.21	0.75
Random Forest	1.80	1.34	1.01	0.85
XGBoost	1.60	1.26	0.95	0.88
Стекинг	1.50	1.22	0.90	0.90

Рисунок 6 — Регрессия

Заключение по оценке моделей

После расчета всех метрик можно сделать выводы о том, какая модель показывает наилучшие результаты в зависимости от задачи (классификация или регрессия). Использование нескольких метрик поможет более полно оценить производительность модели и избежать ошибок, связанных с выбором только одной метрики.