Отлично! Давайте перейдем ко второму уроку.

Урок 2: Основы обучения с учителем (Supervised Learning)

Цель урока:

Изучить основные концепции и метрики, используемые в задачах обучения с учителем, такие как классификация и регрессия, а также проблемы переобучения и недообучения.

Теоретическая часть (15 минут)

1. **Обучение с учителем (Supervised Learning):**
   - **Определение:** В задачах обучения с учителем модель обучается на размеченных данных, где каждый образец (пример) имеет соответствующую метку или целевое значение. Цель модели — научиться предсказывать метки или значения на основе входных данных.
   - **Типы задач:**
     - **Классификация:** Прогнозирование категориальных меток (например, спам/не спам для электронной почты, тип цветка для датасета Iris).
     - **Регрессия:** Прогнозирование непрерывных значений (например, прогнозирование цен на недвижимость).
2. **Метрики оценки моделей:**
   - **Для классификации:**
     - **Accuracy (точность):** Доля правильных предсказаний модели.

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

     - **Precision (точность):** Доля правильно предсказанных положительных классов от всех предсказанных положительных классов.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

     - **Recall (полнота):** Доля правильно предсказанных положительных классов от всех фактических положительных классов.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

     - **F1-score:** Гармоническое среднее между точностью и полнотой, используется, когда важен баланс между этими метриками.

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

- **Для регрессии:**
  - **Mean Absolute Error (MAE):** Среднее абсолютное отклонение предсказанных значений от истинных.

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

  - **Mean Squared Error (MSE):** Среднее квадратическое отклонение предсказанных значений от истинных.

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

  - **Root Mean Squared Error (RMSE):** Квадратный корень из MSE, используемый для того, чтобы вернуть ошибку в тех же единицах, что и исходные данные.

$$\text{RMSE} = \sqrt{\text{MSE}}$$

3. **Проблемы переобучения (Overfitting) и недообучения (Underfitting):**
   - **Переобучение (Overfitting):** Модель слишком сложная и слишком хорошо подстраивается под обучающие данные, включая шум и случайные колебания, но плохо обобщает знания на новые данные.
   - **Недообучение (Underfitting):** Модель слишком простая и не может уловить основные закономерности в данных, что приводит к плохой производительности как на обучающих, так и на тестовых данных.
   - **Как избежать переобучения:** Использование регуляризации, кросс-валидации, уменьшение сложности модели.
   - **Как избежать недообучения:** Увеличение сложности модели, использование более сложных алгоритмов, добавление больше признаков или больше данных.

Практическая часть (15 минут)

Давайте теперь создадим и обучим модель логистической регрессии на датасете Iris и оценим ее, используя различные метрики.

1. **Создание и обучение модели:**

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Загрузка датасета Iris
iris = load_iris()
X, y = iris.data, iris.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)

# Создание и обучение модели логистической регрессии
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Прогнозирование на тестовой выборке
y_pred = model.predict(X_test)

# Оценка модели
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

2. **Пояснение коду:**

- **Метод `fit`:** Обучает модель на обучающих данных `X_train` и `y_train`.
- **Метод `predict`:** Используется для предсказания меток классов на тестовой выборке `X_test`.
- **Метрики:**
  - `accuracy_score` оценивает точность модели.
  - `classification_report` предоставляет отчет с метриками точности (precision), полноты (recall), F1-score для каждого класса.
  - `confusion_matrix` показывает матрицу ошибок, что позволяет понять, какие классы путает модель.

Что дальше?

Попробуйте выполнить этот код на вашей машине и посмотрите на результаты. Если возникнут вопросы по метрикам или по каким-то аспектам модели, дайте знать! Мы можем углубиться в любую тему или двигаться дальше к следующему уроку.

Да, вы абсолютно правы. В обоих уроках мы использовали обучение с учителем (Supervised Learning), а именно задачу классификации. Второй пример очень похож на первый, но с более подробным объяснением метрик оценки модели.

Теперь давайте разберем результаты, которые вы получили, особенно **Confusion Matrix** (матрицу ошибок).

Понимание Confusion Matrix (Матрица ошибок)

Матрица ошибок (Confusion Matrix) — это инструмент, который помогает визуализировать производительность модели классификации, показывая, где модель делает правильные предсказания, а где ошибается. Она представляет собой таблицу, где:

- Строки соответствуют фактическим меткам классов (истинным значениям).
- Столбцы соответствуют предсказанным меткам классов.

Каждое значение в матрице показывает количество наблюдений, предсказанных моделью для конкретного класса по сравнению с истинным классом.

Рассмотрим вашу матрицу:

$$\text{Confusion Matrix} = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 11 \end{bmatrix}$$

- **Строки (True Labels):** Фактические метки классов.
- **Столбцы (Predicted Labels):** Предсказанные метки классов.

Разбор матрицы:

1. **Класс 0 (Iris-setosa):**
   - Истинное количество наблюдений класса 0 — 10.
   - Модель предсказала 10 раз класс 0 как класс 0 (правильные предсказания).
   - Модель **не сделала** ни одного неправильного предсказания для класса 0.
2. **Класс 1 (Iris-versicolor):**
   - Истинное количество наблюдений класса 1 — 9.
   - Модель предсказала 9 раз класс 1 как класс 1 (правильные предсказания).
   - Модель **не сделала** ни одного неправильного предсказания для класса 1.
3. **Класс 2 (Iris-virginica):**
   - Истинное количество наблюдений класса 2 — 11.
   - Модель предсказала 11 раз класс 2 как класс 2 (правильные предсказания).

- Модель **не сделала** ни одного неправильного предсказания для класса 2.

Интерпретация результатов

- **Диагональные элементы** (10, 9, 11) представляют собой количество правильных предсказаний для каждого класса.
- **Элементы вне диагонали** (все нули в данном случае) представляют количество ошибок. Например, если бы модель ошиблась, предсказав класс 0 вместо класса 1, то в матрице появилась бы ненулевая ячейка в позиции (1,0).

Пояснение к результатам:

- **Матрица ошибок показывает идеальное предсказание,** так как все элементы вне диагонали равны нулю. Это значит, что модель правильно классифицировала все образцы из тестового набора.
- **Метрики из классификационного отчета (`Classification Report`) подтверждают это:**
  - Все показатели `precision` , `recall` , и `f1-score` равны 1.0 для всех классов, что указывает на идеальное соответствие предсказаний фактическим меткам.

Итог

Ваши результаты показывают, что модель логистической регрессии смогла идеально предсказать классы для всех тестовых данных. Это может быть связано с тем, что датасет Iris не является сложным для классификации, и все три класса легко различимы по заданным признакам.

Если у вас есть еще вопросы по этому уроку или хотите перейти к следующему уроку, дайте знать!

Да, вы абсолютно правы! Давайте ответим на ваши вопросы по порядку.

1. Классификация с использованием `LogisticRegression`

Ваша задача с датасетом Iris — это действительно задача **классификации**. В датасете Iris нужно предсказать категориальную метку (вид цветка), поэтому мы используем модель классификации.

LogisticRegression — это алгоритм, который, несмотря на своё название, используется для задач классификации, а не регрессии. Он применяется, когда нужно предсказать вероятность принадлежности к одному из двух или более классов.

Почему `LogisticRegression` используется для классификации?

- **Логистическая регрессия** использует логистическую функцию (сигмоиду) для прогнозирования вероятности того, что наблюдение принадлежит к конкретному классу.
- В случае бинарной классификации (два класса) логистическая регрессия предсказывает вероятность принадлежности к одному из двух классов и использует пороговое значение (например, 0.5) для принятия решения о принадлежности к конкретному классу.
- В многоклассовой классификации (как в случае с Iris) логистическая регрессия использует подходы, такие как **“One-vs-Rest”** или **“Softmax”** (в случае многоклассовой логистической регрессии), чтобы предсказать вероятность принадлежности к каждому из нескольких классов.

Логистическая регрессия — один из самых популярных алгоритмов для задач классификации из-за своей простоты и интерпретируемости.

Есть ли другие классы для задач классификации?

Да, `scikit-learn` предоставляет множество алгоритмов для задач классификации, в зависимости от сложности задачи и структуры данных:

- **DecisionTreeClassifier**: Деревья решений, которые строят модель решений на основе значений признаков.
- **RandomForestClassifier**: Ансамблевый метод, использующий множество деревьев решений для улучшения точности и устойчивости модели.
- **Support Vector Classifier (SVC)**: Алгоритм, который ищет гиперплоскость, оптимально разделяющую данные на классы.
- **KNeighborsClassifier**: Метод ближайших соседей, который классифицирует примеры на основе классов их ближайших соседей в обучающем наборе.
- **NaiveBayes**: Простой вероятностный классификатор, основанный на применении теоремы Байеса с “наивным” предположением о независимости признаков.

Все эти алгоритмы предназначены для задач классификации и могут быть выбраны в зависимости от характеристик ваших данных и требований к модели.

2. Регрессия: Прогнозирование непрерывных значений

Регрессия — это тип задачи машинного обучения, в которой мы предсказываем непрерывное значение (в отличие от классификации, где мы предсказываем категорию или метку).

Основные задачи регрессии:

- **Прогнозирование цен**: Например, предсказание цены недвижимости на основе её характеристик (площадь, количество комнат, местоположение и т.д.).
- **Прогнозирование спроса**: Прогнозирование спроса на товар в будущем на основе исторических данных.
- **Прогнозирование числовых значений**: Любая задача, где нужно предсказать числовое значение, такое как температура, вес, рост, доход и т.д.

Основные алгоритмы регрессии:

1. **Линейная регрессия (Linear Regression):**
   - **Описание:** Это простой алгоритм, который ищет линейную зависимость между входными признаками и выходным значением. Он находит прямую, которая минимизирует сумму квадратов отклонений (ошибок) предсказанных значений от истинных.
   - **Формула:** $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$, где y — предсказанное значение, x_i — входные признаки, β_i — коэффициенты модели.
2. **Полиномиальная регрессия (Polynomial Regression):**
   - **Описание:** Расширение линейной регрессии, которое добавляет полиномиальные признаки в модель. Это позволяет моделировать более сложные нелинейные отношения между признаками и целевым значением.
   - **Пример:** Включение квадратичных или кубических признаков, таких как x^2 или x^3, для создания кривой, которая лучше соответствует данным.
3. **Ridge и Lasso регрессия:**
   - **Описание:** Это линейные модели, включающие регуляризацию (L2 для Ridge и L1 для Lasso), чтобы предотвратить переобучение модели, добавляя штраф за слишком большие значения коэффициентов.
   - **Применение:** Ridge регрессия полезна, когда у нас есть много коррелированных признаков, а Lasso также используется для отбора признаков.
4. **Support Vector Regression (SVR):**
   - **Описание:** Это расширение метода опорных векторов (SVM) для задач регрессии. Он пытается найти гиперплоскость в пространстве признаков, которая максимально приближена к данным.
   - **Особенности:** Может моделировать нелинейные зависимости с помощью ядровых функций (например, Radial Basis Function).
5. **Random Forest Regression:**
   - **Описание:** Это ансамблевый метод, который использует множество деревьев решений для построения более устойчивой и точной модели. Каждое дерево предсказывает

значение, а финальное предсказание получается путем усреднения всех предсказаний.
- **Особенности:** Хорошо работает с большими наборами данных и может справляться с нелинейностями и взаимодействиями между признаками.

6. **Gradient Boosting Regression (например, XGBoost):**
   - **Описание:** Этот метод улучшает производительность модели, последовательно обучая слабые модели (например, деревья решений) и исправляя их ошибки.
   - **Особенности:** Очень мощный метод для задач регрессии с высокими требованиями к точности, широко используется в соревнованиях по машинному обучению.

Пример задачи регрессии: Линейная регрессия

Давайте рассмотрим простой пример линейной регрессии для предсказания цены на недвижимость на основе её площади:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Пример данных: площадь (кв. м.) и цена (тыс. $)
X = np.array([[50], [60], [70], [80], [90], [100], [110], [120], [130], [140]])
y = np.array([150, 180, 200, 240, 270, 300, 320, 360, 400, 450])

# 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)

# Создание модели линейной регрессии
model = LinearRegression()
model.fit(X_train, y_train)

# Прогнозирование
y_pred = model.predict(X_test)

# Оценка модели
print("Mean squared error:", mean_squared_error(y_test, y_pred))
print("R2 score:", r2_score(y_test, y_pred))

# Визуализация
plt.scatter(X, y, color='black')
plt.plot(X, model.predict(X), color='blue', linewidth=3)
plt.xlabel('Площадь (кв. м.)')
plt.ylabel('Цена (тыс. $)')
plt.title('Линейная регрессия: площадь и цена недвижимости')
plt.show()
```

В этом примере:

- Мы обучаем модель линейной регрессии предсказывать цену недвижимости на основе её площади.
- Оцениваем модель с помощью **MSE** (mean squared error) и **R2 score** (коэффициент детерминации), который показывает, насколько хорошо модель объясняет вариативность в данных.

Итог

Вы правильно понимаете различие между задачами классификации и регрессии. Ваша задача с Iris — это задача классификации, и мы используем `LogisticRegression` для предсказания категориальных меток. Регрессия используется для предсказания непрерывных значений, таких как цена или вес, и может быть решена с помощью различных алгоритмов, таких как линейная регрессия, Ridge, Lasso, и других.

Если хотите углубиться в любую из этих тем или у вас есть дополнительные вопросы, дайте знать! Мы можем продолжить с более сложными задачами или перейти к следующему уроку.