Отлично! Давайте перейдем к четвертому уроку.

Урок 4: Классификация и Основные Алгоритмы (k-Nearest Neighbors, Decision Trees)

Цель урока:

Изучить основные алгоритмы классификации, такие как k-ближайших соседей (k-Nearest Neighbors, k-NN) и деревья решений (Decision Trees). Понять, как они работают, их преимущества и недостатки, а также как их применять на практике.

Теоретическая часть (15 минут)

1. **Алгоритм k-ближайших соседей (k-Nearest Neighbors, k-NN):**
   - **Описание:** k-NN — это простой и интуитивно понятный алгоритм классификации. Он относит новый объект к тому классу, который наиболее часто встречается среди k ближайших соседей этого объекта в обучающей выборке.
   - **Как работает:**
     a. Для нового образца найти k ближайших соседей в обучающей выборке.
     b. Определить классы этих соседей.
     c. Отнести новый образец к тому классу, который встречается наиболее часто среди найденных соседей.
   - **Метрика расстояния:** Для определения “близости” обычно используется евклидово расстояние, но можно использовать и другие метрики, такие как манхэттенское расстояние.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

   - **Преимущества:**
     - Простота и интуитивная понятность.
     - Не требует обучения модели (т.е. является “ленивым” алгоритмом).
   - **Недостатки:**
     - Затратен по времени и памяти для больших наборов данных, так как нужно хранить все обучающие данные.
     - Чувствителен к масштабу признаков, поэтому необходимо нормализовать данные.
     - Производительность зависит от выбора k и метрики расстояния.
2. **Деревья решений (Decision Trees):**

- **Описание:** Дерево решений — это модель, которая разбивает пространство признаков на области с помощью условий (разделений) на каждом узле дерева. Каждое условие представляет собой проверку значения одного признака.
- **Как работает:**
  a. Начинается с корня дерева, который представляет весь набор данных.
  b. Данные рекурсивно разбиваются на основе условий, создавая ветви и узлы.
  c. Разбиение происходит до тех пор, пока каждый лист дерева не будет содержать образцы одного класса или не будет достигнута максимальная глубина дерева.
- **Критерии разбиения:** Для выбора лучшего разделения используется критерий информативности, такой как:
  - **Gini impurity (индекс Джини):** Мера “чистоты” узла. Чем меньше значение, тем лучше разбиение.

$$Gini(D) = 1 - \sum_{i=1}^{C} p_i^2$$

    где p_i — доля объектов класса i в наборе D.
  - **Information Gain (прирост информации):** Основан на энтропии, которая измеряет неопределенность данных.
- **Преимущества:**
  - Простота и интерпретируемость: легко визуализировать и понять, как модель принимает решения.
  - Не требует нормализации данных.
- **Недостатки:**
  - Склонность к переобучению, особенно на глубоких деревьях.
  - Неустойчивость к небольшим изменениям в данных (различные выборки могут приводить к совершенно разным деревьям).

Практическая часть (15 минут)

Теперь давайте применим оба алгоритма на практике с использованием `scikit-learn` и посмотрим, как они работают на датасете Iris.

1. **k-Nearest Neighbors (k-NN):**

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# 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 k-NN (k=3)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Прогнозирование на тестовой выборке
y_pred_knn = knn.predict(X_test)

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

2. **Decision Tree:**

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

# Создание и обучение модели дерева решений
dt = DecisionTreeClassifier(max_depth=3)
dt.fit(X_train, y_train)

# Прогнозирование на тестовой выборке
y_pred_dt = dt.predict(X_test)

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

# Визуализация дерева решений
plt.figure(figsize=(12,8))
tree.plot_tree(dt, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()
```

Пояснение коду:

- **k-NN:**
  - Мы создаем и обучаем модель k-NN с $k = 3$.
  - Модель классифицирует новый образец, используя 3 ближайших соседа.
  - Оцениваем точность модели и смотрим отчет о классификации.
- **Decision Tree:**
  - Мы создаем и обучаем модель дерева решений с максимальной глубиной 3, чтобы избежать переобучения.
  - Визуализируем дерево, чтобы увидеть, как модель принимает решения на основе условий.

Что дальше?

Попробуйте выполнить приведенные примеры на вашей машине и поэкспериментировать с параметрами (например, с числом соседей k для k-NN или максимальной глубиной дерева для Decision Tree). Если возникнут вопросы по алгоритмам, метрикам или чему-то еще, дайте знать! Мы также можем перейти к следующему уроку, если вы готовы.