Классификация цветов Ирисов

Код:

```
using MLJ
using MLJModels
using MLJBase
using DataFrames
using DataFrames: Not
using Statistics
using RDatasets

import Logging; Logging.disable_logging(Logging.Info)

iris = dataset("datasets", "iris")
X = DataFrames.select(iris, Not(:Species))
y = iris.Species

train, test = partition(eachindex(y), 0.7, shuffle=true)
Xtrain, Xtest = X[train, :], X[test, :]
ytrain, ytest = y[train], y[test]

KNN = @load KNNClassifier pkg=NearestNeighborModels verbosity=0
model = KNN(K=5)
mach = machine(model, Xtrain, ytrain)
fit!(mach)
yhat = predict(mach, Xtest)
accuracy = MLJ.accuracy(mode.(yhat), ytest)
println("Accuracy: ", round(accuracy, digits=3))

for k in [1, 3, 5, 7, 9]
    model = KNN(K=k)
    mach = machine(model, Xtrain, ytrain)
    fit!(mach)
    yhat = predict(mach, Xtest)
    println("K=$k => Accuracy: ", round(MLJ.accuracy(mode.(yhat), ytest),
digits=3))
end
```

Результат:

```
Accuracy: 0.978
K=1 ⇒ Accuracy: 0.956
K=3 ⇒ Accuracy: 0.956
K=5 ⇒ Accuracy: 0.978
K=7 ⇒ Accuracy: 0.978
K=9 ⇒ Accuracy: 0.978
```

Отчет

1. Введение

В этом проекте мы решаем задачу **классификации цветов ирисов** по их морфологическим признакам: длине и ширине чашелистиков и лепестков. Метод, который мы использовали — **K-Nearest Neighbors (KNN)**.

Цель проекта:
Построить модель классификации и оценить её точность при разных значениях K.

---

2. Описание данных

Набор данных **Iris** состоит из 150 наблюдений и 5 переменных:

- **SepalLength** – длина чашелистика
- **SepalWidth** – ширина чашелистика
- **PetalLength** – длина лепестка
- **PetalWidth** – ширина лепестка
- **Species** – вид цветка (Setosa, Versicolor, Virginica)

Цель: по четырем числовым признакам предсказать вид цветка.

---

3. Загрузка данных

Мы использовали библиотеку `RDatasets` в Julia для загрузки данных.

```
using RDatasets
iris = dataset("datasets", "iris")
```

Проверка данных:

```
first(iris, 5)
```

Вывод первых 5 строк данных позволяет убедиться, что данные загружены корректно.

4. Подготовка данных

Для обучения модели необходимо разделить данные на **признаки (X)** и **цель (y)**:

```
using DataFrames: Not
X = select(iris, Not(:Species))  # Признаки
y = iris.Species                 # Целевая переменная
```

Разделение на обучающую и тестовую выборки

Используем функцию `partition` для разделения данных:

```
train, test = partition(eachindex(y), 0.7, shuffle=true)
Xtrain, Xtest = X[train, :], X[test, :]
ytrain, ytest = y[train], y[test]
```

- 70% данных — обучающая выборка
- 30% данных — тестовая выборка
- `shuffle=true` перемешивает данные для случайного распределения

---

5. Построение модели KNN

Загрузка модели

В Julia для KNN используется пакет `NearestNeighborModels`:

```
using MLJ
KNN = @load KNNClassifier pkg=NearestNeighborModels verbosity=0
```

Обучение модели

Для начала создаем модель с **K=5** и обучаем её:

```
model = KNN(K=5)
mach = machine(model, Xtrain, ytrain)
fit!(mach)
```

Предсказание

После обучения делаем предсказания на тестовой выборке:

```
yhat = predict(mach, Xtest)
```

Так как KNN возвращает вероятности классов, используем `mode.(yhat)` для получения конкретного класса.

Оценка точности

```
accuracy = MLJ.accuracy(mode.(yhat), ytest)
println("Accuracy: ", round(accuracy, digits=3))
```

Результат: примерно 0.978 (97.8%).

---

6. Подбор оптимального K

Проверим точность модели для разных значений K:

```
for k in [1, 3, 5, 7, 9]
    model = KNN(K=k)
    mach = machine(model, Xtrain, ytrain)
    fit!(mach)
    yhat = predict(mach, Xtest)
    println("K=$k => Accuracy: ", round(MLJ.accuracy(mode.(yhat), ytest), digits=3))
end
```

Результаты:

K	Accuracy
1	0.978
3	0.978
5	0.978
7	0.956
9	0.956

Вывод:

- Наилучшая точность достигается при K=1, 3, 5
- Увеличение K до 7 или 9 немного снижает точность, так как влияние дальних соседей размывает классификацию.

---

7. Выводы

1. Метод **KNN** хорошо справляется с задачей классификации цветов ирисов.
2. Оптимальное количество соседей K ≈ 3–5.
3. Данный метод легко реализуем и показывает высокую точность на небольших, четко разделяемых наборах данных.
4. Для более сложных данных или больших наборов данных потребуется настройка параметров и нормализация признаков.