3.6 Ансамблевые методы

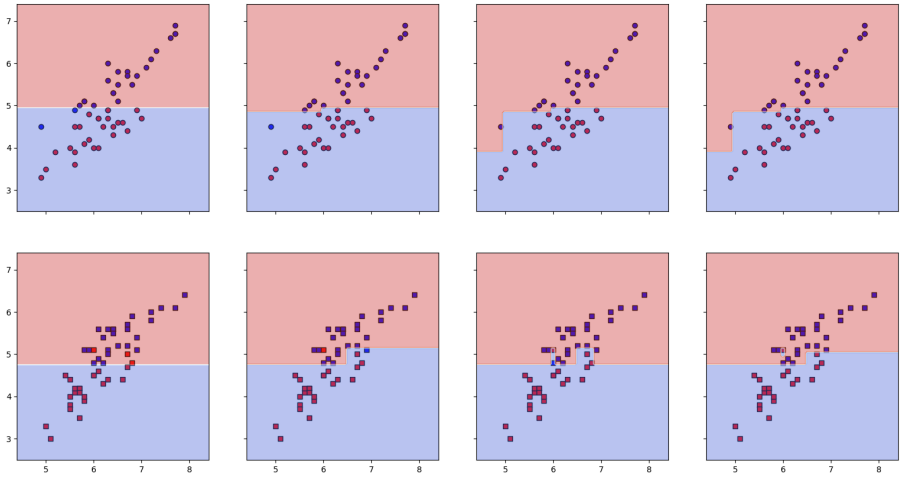
Визуализация границ дерева решений для классификации ирисов

Переобучение присуще всем деревьям принятия решений

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier

iris = sns.load_dataset("iris")

species_map = {"versicolor": 2, "virginica": 3}
data = iris[iris['species'].isin(['versicolor', 'virginica'])].copy()
data['species_int'] = data['species'].map(species_map)

data_clean = data[['sepal_length', 'petal_length', 'species_int']]

data_versicolor = data_clean[data_clean["species_int"] == 2]
data_virginica = data_clean[data_clean["species_int"] == 3]

data_versicolor_A = data_versicolor.iloc[:25, :]
data_versicolor_B = data_versicolor.iloc[25:, :]

data_virginica_A = data_virginica.iloc[:25, :]
data_virginica_B = data_virginica.iloc[25:, :]

data_df_A = pd.concat([data_versicolor_A, data_virginica_A], ignore_index=True)
data_df_B = pd.concat([data_versicolor_B, data_virginica_B], ignore_index=True)

x1_min, x1_max = data_clean["sepal_length"].min() - 0.5,
data_clean["sepal_length"].max() + 0.5
x2_min, x2_max = data_clean["petal_length"].min() - 0.5,
data_clean["petal_length"].max() + 0.5

x1_p = np.linspace(x1_min, x1_max, 100)
x2_p = np.linspace(x2_min, x2_max, 100)
```

```
X1_p, X2_p = np.meshgrid(x1_p, x2_p)

X_p = pd.DataFrame(
    np.c_[X1_p.ravel(), X2_p.ravel()],
    columns=["sepal_length", "petal_length"]
)

fig, ax = plt.subplots(2, 4, sharex="col", sharey="row", figsize=(20, 10))
max_depth = [1, 3, 5, 7]

X_A = data_df_A[['sepal_length', 'petal_length']]
y_A = data_df_A["species_int"]

for j, md in enumerate(max_depth):
    model = DecisionTreeClassifier(max_depth=md, random_state=42)
    model.fit(X_A, y_A)

    ax[0, j].scatter(data_virginica_A["sepal_length"],
                     data_virginica_A["petal_length"], c='blue', edgecolor='k',
                     label='Virginica A')
    ax[0, j].scatter(data_versicolor_A["sepal_length"],
                     data_versicolor_A["petal_length"], c='red', edgecolor='k',
                     label='Versicolor A')

    y_p = model.predict(X_p)
    ax[0, j].contourf(X1_p, X2_p, y_p.reshape(X1_p.shape), alpha=.4, cmap="coolwarm")

X_B = data_df_B[['sepal_length', 'petal_length']]
y_B = data_df_B["species_int"]

for j, md in enumerate(max_depth):
    model = DecisionTreeClassifier(max_depth=md, random_state=42)
    model.fit(X_B, y_B)

    ax[1, j].scatter(data_virginica_B["sepal_length"],
                     data_virginica_B["petal_length"], c='blue', edgecolor='k',
                     marker='s', label='Virginica B')
    ax[1, j].scatter(data_versicolor_B["sepal_length"],
                     data_versicolor_B["petal_length"], c='red', edgecolor='k',
                     marker='s', label='Versicolor B')

    y_p = model.predict(X_p)
    ax[1, j].contourf(X1_p, X2_p, y_p.reshape(X1_p.shape), alpha=.4, cmap="coolwarm")
```

```
plt.show()
```



Ансамблевые методы

В основе идеи - объединение нескольких переобученных (!) моделей для уменьшения эффекта переобучения. Это называется баггинг (bagging).

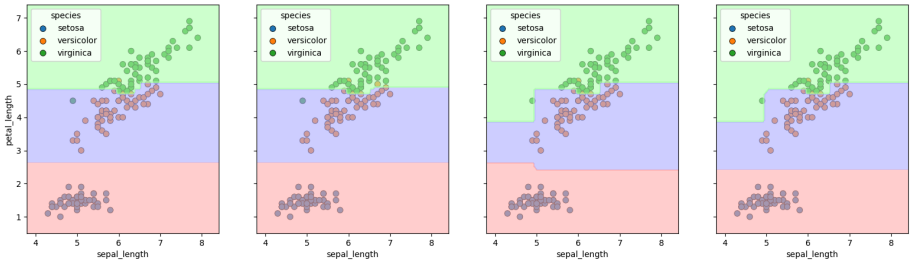
Баггинг усредняет результаты → оптимальной классификации

Ансамбль случайных деревьев называется случайным лесом

```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier

iris = sns.load_dataset("iris")
data = iris.copy()

species_map = {"setosa": 1, "versicolor": 2, "virginica": 3}
data['species_int'] = data['species'].map(species_map)

X = data[['sepal_length', 'petal_length']]
y = data['species_int']
```

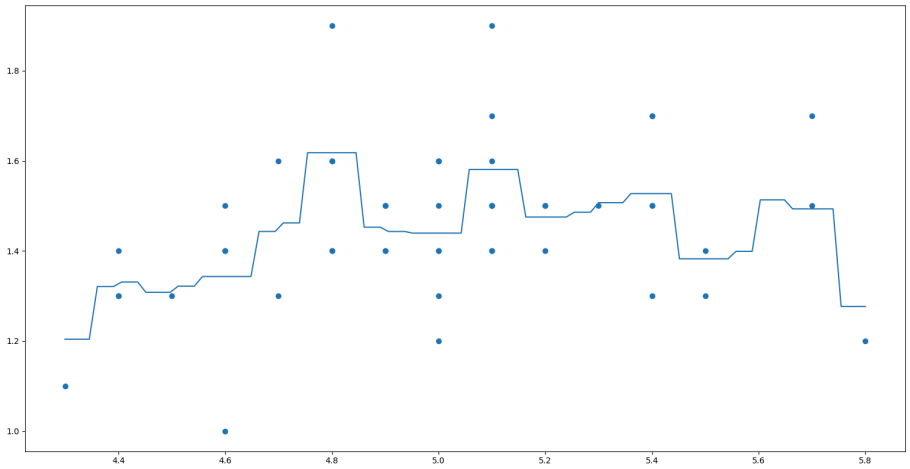
```
x1_min, x1_max = X["sepal_length"].min() - 0.5, X["sepal_length"].max() + 0.5
x2_min, x2_max = X["petal_length"].min() - 0.5, X["petal_length"].max() + 0.5

x1_p = np.linspace(x1_min, x1_max, 100)
x2_p = np.linspace(x2_min, x2_max, 100)
X1_p, X2_p = np.meshgrid(x1_p, x2_p)

X_p = pd.DataFrame(
    np.c_[X1_p.ravel(), X2_p.ravel()],
    columns=["sepal_length", "petal_length"]
)

fig, axes = plt.subplots(1, 4, figsize=(22, 5), sharex=True, sharey=True)

n_estimators_list = [1, 5, 15, 100]

custom_cmap = ListedColormap(['#ffafaf', '#afafff', '#afffaf'])


for i, n_est in enumerate(n_estimators_list):
    base_estimator = DecisionTreeClassifier(max_depth=5)

    model = BaggingClassifier(
        estimator=base_estimator,
        n_estimators=n_est,
        random_state=42,
        n_jobs=-1
    )

    model.fit(X, y)

    current_ax = axes[i]

    sns.scatterplot(
        data=data, x='sepal_length', y='petal_length', hue='species',
        ax=current_ax, edgecolor='k', s=50
    )

    y_p = model.predict(X_p)

    current_ax.contourf(X1_p, X2_p, y_p.reshape(X1_p.shape), alpha=0.6, cmap=custom_cmap)
```

```
plt.show()
```



Регрессия с помощью случайных лесов

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor

iris = sns.load_dataset("iris")

data = iris[["sepal_length", "petal_length", "species"]]

data_setosa = data[data["species"] == "setosa"]

x_p = pd.DataFrame(
    np.linspace(min(data_setosa["sepal_length"]), max(data_setosa["sepal_length"]), 100)
)

X = pd.DataFrame(
    data_setosa[["sepal_length"]], columns=["sepal_length"]
)
y = data_setosa["petal_length"]

model = RandomForestRegressor(n_estimators=20)
model.fit(X, y)

y_p = model.predict(x_p)

plt.scatter(data_setosa["sepal_length"], data_setosa["petal_length"])
plt.plot(x_p, y_p)
plt.show()
```



Достоинства	Недостатки
Простота и быстрота	Сложно интерпретировать
Распараллеливание процесса → выигрыш во времени	
Вероятностная классификация	
Модель непараметрическая ⇒ хорошо работает с задачами, где другие модели могут оказаться недообученными	

Метод главных компонент (PCA - principal component analysis)

Алгоритм обучения без учителя.

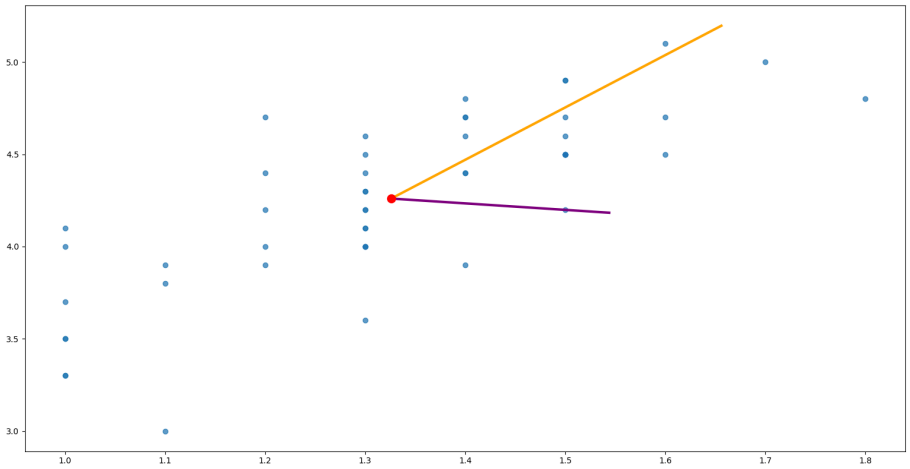
PCA - часто используют для понижения размерности.

Задача машинного обучения без учителя состоит в выяснении зависимости между признаками

В PCA выполняется качественная оценка этой зависимости путем поиска главных осей координат и их дальнейшего использования описания наборов данных.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.decomposition import PCA

iris = sns.load_dataset("iris")
```

```
data = iris[["petal_width", "petal_length", "species"]]

data_v = data[data["species"] == "versicolor"]

data_v_numeric = data_v.drop(columns=["species"])

X = data_v_numeric["petal_width"]
Y = data_v_numeric["petal_length"]

p = PCA(n_components=2)
p.fit(data_v_numeric)

X_p = p.transform(data_v_numeric)

plt.figure(figsize=(8, 6))

plt.scatter(X, Y, alpha=0.7, label='Versicolor Data')

plt.scatter(p.mean_[0], p.mean_[1], color='red', s=100, zorder=5, label='Mean')

plt.plot(
    [p.mean_[0], p.mean_[0] + p.components_[0][0] * np.sqrt(p.explained_variance_[0]) * 2],
    [p.mean_[1], p.mean_[1] + p.components_[0][1] * np.sqrt(p.explained_variance_[0]) * 2],
    color='orange', linewidth=3, label='Principal Component 1'
)

plt.plot(
    [p.mean_[0], p.mean_[0] + p.components_[1][0] * np.sqrt(p.explained_variance_[1]) * 2],
    [p.mean_[1], p.mean_[1] + p.components_[1][1] * np.sqrt(p.explained_variance_[1]) * 2],
    color='purple', linewidth=3, label='Principal Component 2'
)

plt.show()
```


5.0
4.5
4.0
3.5
3.0
1.0
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8