

# Тема: Алгоритмы классификации

Сергей Витальевич Рыбин  
[svrybin@etu.ru](mailto:svrybin@etu.ru)

СПбГЭТУ «ЛЭТИ», кафедра «Алгоритмической математики»

11 ноября 2023 г.



СПбГЭТУ «ЛЭТИ»  
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ



- 1 Задача классификации это *обучение с учителем (supervised learning)*.

- 1 Задача классификации это *обучение с учителем (supervised learning)*.
- 2 Имеется множество объектов  $\mathbf{X} = \{\mathbf{X}_1, \dots, \mathbf{X}_n\}$  и множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .

# Формальная постановка задачи классификации

- 1 Задача классификации это *обучение с учителем* (*supervised learning*).
- 2 Имеется множество объектов  $\mathbf{X} = \{\mathbf{X}_1, \dots, \mathbf{X}_n\}$  и множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 3 Каждый объект представлен набором признаков  $\mathbf{X}_i = \{x_{i_1}, \dots, x_{i_m}\}$ ,  $i \in 1 : n$ .

# Формальная постановка задачи классификации

- 1 Задача классификации это *обучение с учителем (supervised learning)*.
- 2 Имеется множество объектов  $\mathbf{X} = \{\mathbf{X}_1, \dots, \mathbf{X}_n\}$  и множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 3 Каждый объект представлен набором признаков  $\mathbf{X}_i = \{x_{i_1}, \dots, x_{i_m}\}$ ,  $i \in 1 : n$ .
- 4 На множестве  $\mathbf{X}$  на основе признаков задана функция расстояния между объектами  $\rho(\mathbf{X}_i, \mathbf{X}_j) > 0$ ,  $i \neq j$ .

# Формальная постановка задачи классификации

- 1 Задача классификации это *обучение с учителем (supervised learning)*.
- 2 Имеется множество объектов  $\mathbf{X} = \{\mathbf{X}_1, \dots, \mathbf{X}_n\}$  и множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 3 Каждый объект представлен набором признаков  $\mathbf{X}_i = \{x_{i_1}, \dots, x_{i_m}\}$ ,  $i \in 1 : n$ .
- 4 На множестве  $\mathbf{X}$  на основе признаков задана функция расстояния между объектами  $\rho(\mathbf{X}_i, \mathbf{X}_j) > 0$ ,  $i \neq j$ .
- 5 Задана обучающая выборка объектов  $\tilde{\mathbf{X}} = \{\mathbf{X}_{j_1}, \dots, \mathbf{X}_{j_k}\} \subset \mathbf{X}$ . Для каждого элемента обучающей выборки задана метка класса. То есть определено  $k$  пар

$$(\mathbf{X}_{j_s}, y_m), j_s \in j_1 : j_k, m \in 1 : p. \quad (1)$$

Такой набор пар (1) называют *совокупностью прецедентов*, а сам процесс обучения — *обучение по прецедентам*.

# Формальная постановка задачи классификации

- 1 Задача классификации это *обучение с учителем* (*supervised learning*).
- 2 Имеется множество объектов  $\mathbf{X} = \{\mathbf{X}_1, \dots, \mathbf{X}_n\}$  и множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 3 Каждый объект представлен набором признаков  $\mathbf{X}_i = \{x_{i_1}, \dots, x_{i_m}\}$ ,  $i \in 1 : n$ .
- 4 На множестве  $\mathbf{X}$  на основе признаков задана функция расстояния между объектами  $\rho(\mathbf{X}_i, \mathbf{X}_j) > 0$ ,  $i \neq j$ .
- 5 Задана обучающая выборка объектов  $\tilde{\mathbf{X}} = \{\mathbf{X}_{j_1}, \dots, \mathbf{X}_{j_k}\} \subset \mathbf{X}$ . Для каждого элемента обучающей выборки задана метка класса. То есть определено  $k$  пар

$$(\mathbf{X}_{j_s}, y_m), j_s \in j_1 : j_k, m \in 1 : p. \quad (1)$$

Такой набор пар (1) называют *совокупностью прецедентов*, а сам процесс обучения — *обучение по прецедентам*.

- 6 Построить функцию  $f : \mathbf{X} \rightarrow \mathbf{Y}$  (модель, алгоритм), которая любому объекту из  $\mathbf{X}$  ставит в соответствие метку класса из  $\mathbf{Y}$ .



# Формальная постановка задачи классификации

- 1 Задача классификации это *обучение с учителем (supervised learning)*.
- 2 Имеется множество объектов  $\mathbf{X} = \{\mathbf{X}_1, \dots, \mathbf{X}_n\}$  и множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 3 Каждый объект представлен набором признаков  $\mathbf{X}_i = \{x_{i_1}, \dots, x_{i_m}\}$ ,  $i \in 1 : n$ .
- 4 На множестве  $\mathbf{X}$  на основе признаков задана функция расстояния между объектами  $\rho(\mathbf{X}_i, \mathbf{X}_j) > 0$ ,  $i \neq j$ .
- 5 Задана обучающая выборка объектов  $\tilde{\mathbf{X}} = \{\mathbf{X}_{j_1}, \dots, \mathbf{X}_{j_k}\} \subset \mathbf{X}$ . Для каждого элемента обучающей выборки задана метка класса. То есть определено  $k$  пар

$$(\mathbf{X}_{j_s}, y_m), j_s \in j_1 : j_k, m \in 1 : p. \quad (1)$$

Такой набор пар (1) называют *совокупностью прецедентов*, а сам процесс обучения — *обучение по прецедентам*.

- 6 Построить функцию  $f : \mathbf{X} \rightarrow \mathbf{Y}$  (модель, алгоритм), которая любому объекту из  $\mathbf{X}$  ставит в соответствие метку класса из  $\mathbf{Y}$ .

## Типы классификации

- ✓ *Бинарная классификация* (Binary Classification). Два класса меток,  $\mathbf{Y} = \{y_1, y_2\}$ . Например обнаружение спама (спам или не спам).

# Формальная постановка задачи классификации

- 1 Задача классификации это *обучение с учителем (supervised learning)*.
- 2 Имеется множество объектов  $\mathbf{X} = \{\mathbf{X}_1, \dots, \mathbf{X}_n\}$  и множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 3 Каждый объект представлен набором признаков  $\mathbf{X}_i = \{x_{i_1}, \dots, x_{i_m}\}$ ,  $i \in 1 : n$ .
- 4 На множестве  $\mathbf{X}$  на основе признаков задана функция расстояния между объектами  $\rho(\mathbf{X}_i, \mathbf{X}_j) > 0$ ,  $i \neq j$ .
- 5 Задана обучающая выборка объектов  $\tilde{\mathbf{X}} = \{\mathbf{X}_{j_1}, \dots, \mathbf{X}_{j_k}\} \subset \mathbf{X}$ . Для каждого элемента обучающей выборки задана метка класса. То есть определено  $k$  пар

$$(\mathbf{X}_{j_s}, y_m), j_s \in j_1 : j_k, m \in 1 : p. \quad (1)$$

Такой набор пар (1) называют *совокупностью прецедентов*, а сам процесс обучения — *обучение по прецедентам*.

- 6 Построить функцию  $f : \mathbf{X} \rightarrow \mathbf{Y}$  (модель, алгоритм), которая любому объекту из  $\mathbf{X}$  ставит в соответствие метку класса из  $\mathbf{Y}$ .

## Типы классификации

- ✓ *Бинарная классификация* (Binary Classification). Два класса меток,  $\mathbf{Y} = \{y_1, y_2\}$ . Например обнаружение спама (спам или не спам).
- ✓ *Мультиклассовая классификация* (Multi-Class Classification). Здесь  $|\mathbf{Y}| > 2$ . Например, классификация изображений, музыкальных жанров.

# Формальная постановка задачи классификации

- 1 Задача классификации это *обучение с учителем (supervised learning)*.
- 2 Имеется множество объектов  $\mathbf{X} = \{\mathbf{X}_1, \dots, \mathbf{X}_n\}$  и множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 3 Каждый объект представлен набором признаков  $\mathbf{X}_i = \{x_{i_1}, \dots, x_{i_m}\}$ ,  $i \in 1:n$ .
- 4 На множестве  $\mathbf{X}$  на основе признаков задана функция расстояния между объектами  $\rho(\mathbf{X}_i, \mathbf{X}_j) > 0$ ,  $i \neq j$ .
- 5 Задана обучающая выборка объектов  $\tilde{\mathbf{X}} = \{\mathbf{X}_{j_1}, \dots, \mathbf{X}_{j_k}\} \subset \mathbf{X}$ . Для каждого элемента обучающей выборки задана метка класса. То есть определено  $k$  пар

$$(\mathbf{X}_{j_s}, y_m), j_s \in j_1:j_k, m \in 1:p. \quad (1)$$

Такой набор пар (1) называют *совокупностью прецедентов*, а сам процесс обучения — *обучение по прецедентам*.

- 6 Построить функцию  $f: \mathbf{X} \rightarrow \mathbf{Y}$  (модель, алгоритм), которая любому объекту из  $\mathbf{X}$  ставит в соответствие метку класса из  $\mathbf{Y}$ .

## Типы классификации

- ✓ *Бинарная классификация* (Binary Classification). Два класса меток,  $\mathbf{Y} = \{y_1, y_2\}$ . Например обнаружение спама (спам или не спам).
- ✓ *Мультиклассовая классификация* (Multi-Class Classification). Здесь  $|\mathbf{Y}| > 2$ . Например, классификация изображений, музыкальных жанров.
- ✓ *Классификация по нескольким меткам* (Multi-Label Classification). Объект может одновременно иметь несколько меток различных классов:

$$f(\mathbf{X}_i) = \{y_{i_1}, \dots, y_{i_m}\} \subseteq \mathbf{Y}, m \geq 2.$$

Можно считать, что  $f$  это отображение в бинарные вектора размерности  $|\mathbf{Y}|$ : каждая  $i$ -й компонента вектора равна 1 или 0, в зависимости от наличия метки  $y_i$  у объекта.



А *Переобучение* (overtraining, overfitting) — вероятность ошибки обученного алгоритма на объектах тестовой выборки значительно выше, чем средняя ошибка на обучающей выборке. Возникает при использовании **избыточно сложных** моделей.

**А** *Переобучение* (overtraining, overfitting) — вероятность ошибки обученного алгоритма на объектах тестовой выборки значительно выше, чем средняя ошибка на обучающей выборке. Возникает при использовании **избыточно сложных** моделей.

**?** Как бороться с переобучением:

**А** *Переобучение* (overtraining, overfitting) — вероятность ошибки обученного алгоритма на объектах тестовой выборки значительно выше, чем средняя ошибка на обучающей выборке. Возникает при использовании **избыточно сложных** моделей.

**?** Как бороться с переобучением:

- ✓ Увеличить данные в наборе.

**А** *Переобучение* (overtraining, overfitting) — вероятность ошибки обученного алгоритма на объектах тестовой выборки значительно выше, чем средняя ошибка на обучающей выборке. Возникает при использовании **избыточно сложных** моделей.

**?** Как бороться с переобучением:

- ✓ Увеличить данные в наборе.
- ✓ Уменьшить число параметров модели (упростить модель).



**А** *Переобучение* (overtraining, overfitting) — вероятность ошибки обученного алгоритма на объектах тестовой выборки значительно выше, чем средняя ошибка на обучающей выборке. Возникает при использовании **избыточно сложных** моделей.

**?** Как бороться с переобучением:

- ✓ Увеличить данные в наборе.
- ✓ Уменьшить число параметров модели (упростить модель).
- ✓ Регуляризация (будет обсуждаться далее).

**А** *Переобучение* (overtraining, overfitting) — вероятность ошибки обученного алгоритма на объектах тестовой выборки значительно выше, чем средняя ошибка на обучающей выборке. Возникает при использовании **избыточно сложных** моделей.

**?** Как бороться с переобучением:

- ✓ Увеличить данные в наборе.
- ✓ Уменьшить число параметров модели (упростить модель).
- ✓ Регуляризация (будет обсуждаться далее).

**В** *Недообучение* (underfitting) — алгоритм обучения не дает малой величины средней ошибки на обучающей выборке. Возникает при использовании **недостаточно сложных** моделей.

**А** *Переобучение* (overtraining, overfitting) — вероятность ошибки обученного алгоритма на объектах тестовой выборки значительно выше, чем средняя ошибка на обучающей выборке. Возникает при использовании **избыточно сложных** моделей.

**?** Как бороться с переобучением:

- ✓ Увеличить данные в наборе.
- ✓ Уменьшить число параметров модели (упростить модель).
- ✓ Регуляризация (будет обсуждаться далее).

**В** *Недообучение* (underfitting) — алгоритм обучения не дает малой величины средней ошибки на обучающей выборке. Возникает при использовании **недостаточно сложных** моделей.

**?** Как бороться с недообучением:

**А** *Переобучение* (overtraining, overfitting) — вероятность ошибки обученного алгоритма на объектах тестовой выборки значительно выше, чем средняя ошибка на обучающей выборке. Возникает при использовании **избыточно сложных** моделей.

**?** Как бороться с переобучением:

- ✓ Увеличить данные в наборе.
- ✓ Уменьшить число параметров модели (упростить модель).
- ✓ Регуляризация (будет обсуждаться далее).

**В** *Недообучение* (underfitting) — алгоритм обучения не дает малой величины средней ошибки на обучающей выборке. Возникает при использовании **недостаточно сложных** моделей.

**?** Как бороться с недообучением:

- ✓ Добавить параметры в модель (усложнить модель).

**А** *Переобучение* (overtraining, overfitting) — вероятность ошибки обученного алгоритма на объектах тестовой выборки значительно выше, чем средняя ошибка на обучающей выборке. Возникает при использовании **избыточно сложных** моделей.

**?** Как бороться с переобучением:

- ✓ Увеличить данные в наборе.
- ✓ Уменьшить число параметров модели (упростить модель).
- ✓ Регуляризация (будет обсуждаться далее).

**В** *Недообучение* (underfitting) — алгоритм обучения не дает малой величины средней ошибки на обучающей выборке. Возникает при использовании **недостаточно сложных** моделей.

**?** Как бороться с недообучением:

- ✓ Добавить параметры в модель (усложнить модель).
- ✓ Уменьшить регуляризацию.

# Переобучение и недообучение

**А** *Переобучение* (overtraining, overfitting) — вероятность ошибки обученного алгоритма на объектах тестовой выборки значительно выше, чем средняя ошибка на обучающей выборке. Возникает при использовании **избыточно сложных** моделей.

**?** Как бороться с переобучением:

- ✓ Увеличить данные в наборе.
- ✓ Уменьшить число параметров модели (упростить модель).
- ✓ Регуляризация (будет обсуждаться далее).

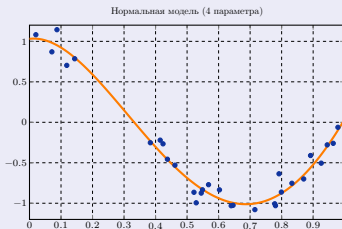
**В** *Недообучение* (underfitting) — алгоритм обучения не дает малой величины средней ошибки на обучающей выборке. Возникает при использовании **недостаточно сложных** моделей.

**?** Как бороться с недообучением:

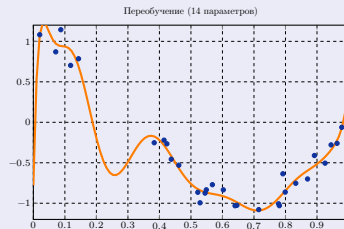
- ✓ Добавить параметры в модель (усложнить модель).
- ✓ Уменьшить регуляризацию.



(a)



(b)



(c)



## Идея

- Метод опорных векторов (*Support Vector Machine, SVM*) — бинарный классификатор ( $\mathbf{Y} = \{y_1, y_2\}$ ). Считаем, что  $\mathbf{Y} = \{-1, 1\}$ .



## Идея

- *Метод опорных векторов (Support Vector Machine, SVM)* — бинарный классификатор ( $\mathbf{Y} = \{y_1, y_2\}$ ). Считаем, что  $\mathbf{Y} = \{-1, 1\}$ .
- Строим гиперплоскость, разделяющую объекты выборки *оптимальным* способом. Считаем, что чем больше расстояние между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет *ошибка* классификатора.

## Идея

- *Метод опорных векторов (Support Vector Machine, SVM)* — бинарный классификатор ( $\mathbf{Y} = \{y_1, y_2\}$ ). Считаем, что  $\mathbf{Y} = \{-1, 1\}$ .
- Строим гиперплоскость, разделяющую объекты выборки *оптимальным* способом. Считаем, что чем больше расстояние между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет *ошибка* классификатора.

## Разделяющая гиперплоскость, линейно разделимая выборка

- В пространстве признаков  $\mathbf{R}^n$  гиперплоскость  $L$  определяется уравнением:  $\langle \mathbf{X}, \mathbf{W} \rangle = b$ . Здесь  $\mathbf{W}$  — вектор нормали,  $b = \langle \mathbf{P}, \mathbf{W} \rangle$ , где  $\mathbf{P}$  — точка гиперплоскости.

## Идея

- Метод опорных векторов (*Support Vector Machine, SVM*) — бинарный классификатор ( $\mathbf{Y} = \{y_1, y_2\}$ ). Считаем, что  $\mathbf{Y} = \{-1, 1\}$ .
- Строим гиперплоскость, разделяющую объекты выборки *оптимальным* способом. Считаем, что чем больше расстояние между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет *ошибка* классификатора.

## Разделяющая гиперплоскость, линейно разделимая выборка

- В пространстве признаков  $\mathbf{R}^n$  гиперплоскость  $\mathbf{L}$  определяется уравнением:  $\langle \mathbf{X}, \mathbf{W} \rangle = b$ . Здесь  $\mathbf{W}$  — вектор нормали,  $b = \langle \mathbf{P}, \mathbf{W} \rangle$ , где  $\mathbf{P}$  — точка гиперплоскости.
- Гиперплоскость  $\mathbf{L}$  делит пространство  $\mathbf{R}^n$  на два полупространства:  $\langle \mathbf{X}, \mathbf{W} \rangle - b > 0$  и  $\langle \mathbf{X}, \mathbf{W} \rangle - b < 0$

# Метод опорных векторов. Идея

## Идея

- Метод опорных векторов (*Support Vector Machine, SVM*) — бинарный классификатор ( $\mathbf{Y} = \{y_1, y_2\}$ ). Считаем, что  $\mathbf{Y} = \{-1, 1\}$ .
- Строим гиперплоскость, разделяющую объекты выборки *оптимальным* способом. Считаем, что чем больше расстояние между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет *ошибка* классификатора.

## Разделяющая гиперплоскость, линейно разделимая выборка

- В пространстве признаков  $\mathbf{R}^n$  гиперплоскость  $\mathbf{L}$  определяется уравнением:  $\langle \mathbf{X}, \mathbf{W} \rangle = b$ . Здесь  $\mathbf{W}$  — вектор нормали,  $b = \langle \mathbf{P}, \mathbf{W} \rangle$ , где  $\mathbf{P}$  — точка гиперплоскости.
- Гиперплоскость  $\mathbf{L}$  делит пространство  $\mathbf{R}^n$  на два полупространства:  $\langle \mathbf{X}, \mathbf{W} \rangle - b > 0$  и  $\langle \mathbf{X}, \mathbf{W} \rangle - b < 0$
- Гиперплоскость *разделяет* множества  $A$  и  $B$ , если элементы этих множеств лежат по разные стороны от гиперплоскости:

$$\begin{cases} \langle \mathbf{X}, \mathbf{W} \rangle - b > 0, & \text{для всех } \mathbf{X} \in A, \\ \langle \mathbf{X}, \mathbf{W} \rangle - b < 0 & \text{для всех } \mathbf{X} \in B. \end{cases} \quad (2)$$

# Метод опорных векторов. Идея

## Идея

- Метод опорных векторов (*Support Vector Machine, SVM*) — бинарный классификатор ( $\mathbf{Y} = \{y_1, y_2\}$ ). Считаем, что  $\mathbf{Y} = \{-1, 1\}$ .
- Строим гиперплоскость, разделяющую объекты выборки *оптимальным* способом. Считаем, что чем больше расстояние между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет *ошибка* классификатора.

## Разделяющая гиперплоскость, линейно разделимая выборка

- В пространстве признаков  $\mathbf{R}^n$  гиперплоскость  $\mathbf{L}$  определяется уравнением:  $\langle \mathbf{X}, \mathbf{W} \rangle = b$ . Здесь  $\mathbf{W}$  — вектор нормали,  $b = \langle \mathbf{P}, \mathbf{W} \rangle$ , где  $\mathbf{P}$  — точка гиперплоскости.
- Гиперплоскость  $\mathbf{L}$  делит пространство  $\mathbf{R}^n$  на два полупространства:  $\langle \mathbf{X}, \mathbf{W} \rangle - b > 0$  и  $\langle \mathbf{X}, \mathbf{W} \rangle - b < 0$
- Гиперплоскость *разделяет* множества  $A$  и  $B$ , если элементы этих множеств лежат по разные стороны от гиперплоскости:

$$\begin{cases} \langle \mathbf{X}, \mathbf{W} \rangle - b > 0, & \text{для всех } \mathbf{X} \in A, \\ \langle \mathbf{X}, \mathbf{W} \rangle - b < 0 & \text{для всех } \mathbf{X} \in B. \end{cases} \quad (2)$$

- Обучающая выборка  $\tilde{\mathbf{X}}$  *линейно разделима*, если существует гиперплоскость, разделяющая классы  $\mathbf{C}_1$  и  $\mathbf{C}_2$ :

$$\mathbf{C}_1 = \{\mathbf{X} \in \tilde{\mathbf{X}} \mid f(\mathbf{X}) = -1\}, \quad \mathbf{C}_2 = \{\mathbf{X} \in \tilde{\mathbf{X}} \mid f(\mathbf{X}) = 1\} \quad (3)$$

# Метод опорных векторов. Идея

## Идея

- Метод опорных векторов (*Support Vector Machine, SVM*) — бинарный классификатор ( $\mathbf{Y} = \{y_1, y_2\}$ ). Считаем, что  $\mathbf{Y} = \{-1, 1\}$ .
- Строим гиперплоскость, разделяющую объекты выборки *оптимальным* способом. Считаем, что чем больше расстояние между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет *ошибка* классификатора.

## Разделяющая гиперплоскость, линейно разделимая выборка

- В пространстве признаков  $\mathbf{R}^n$  гиперплоскость  $\mathbf{L}$  определяется уравнением:  $\langle \mathbf{X}, \mathbf{W} \rangle = b$ . Здесь  $\mathbf{W}$  — вектор нормали,  $b = \langle \mathbf{P}, \mathbf{W} \rangle$ , где  $\mathbf{P}$  — точка гиперплоскости.
- Гиперплоскость  $\mathbf{L}$  делит пространство  $\mathbf{R}^n$  на два полупространства:  $\langle \mathbf{X}, \mathbf{W} \rangle - b > 0$  и  $\langle \mathbf{X}, \mathbf{W} \rangle - b < 0$
- Гиперплоскость *разделяет* множества  $A$  и  $B$ , если элементы этих множеств лежат по разные стороны от гиперплоскости:

$$\begin{cases} \langle \mathbf{X}, \mathbf{W} \rangle - b > 0, & \text{для всех } \mathbf{X} \in A, \\ \langle \mathbf{X}, \mathbf{W} \rangle - b < 0 & \text{для всех } \mathbf{X} \in B. \end{cases} \quad (2)$$

- Обучающая выборка  $\tilde{\mathbf{X}}$  *линейно разделима*, если существует гиперплоскость, разделяющая классы  $\mathbf{C}_1$  и  $\mathbf{C}_2$ :

$$\mathbf{C}_1 = \{\mathbf{X} \in \tilde{\mathbf{X}} \mid f(\mathbf{X}) = -1\}, \quad \mathbf{C}_2 = \{\mathbf{X} \in \tilde{\mathbf{X}} \mid f(\mathbf{X}) = 1\} \quad (3)$$

- Если обучающая выборка  $\tilde{\mathbf{X}}$  линейно разделима, тогда функцию классификации  $f: \mathbf{X} \rightarrow \{-1, 1\}$  (называют **дискриминантной функцией**) можно определить следующим образом:

$$f(\mathbf{X}) = \text{sign}(\langle \mathbf{X}, \mathbf{W} \rangle - b) = \text{sign}\left(\sum_{i=1}^n w_i x_i - b\right), \quad \mathbf{W} = (w_1, \dots, w_n), \mathbf{X} = (x_1, \dots, x_n) \in \tilde{\mathbf{X}}. \quad (4)$$

---

<sup>1</sup>Источник: <https://www.cleanpng.com/png-support-vector-machine-machine-learning-statistica-1095606/>

- ✓ Разделяющих гиперплоскостей может быть несколько. На рисунке 2 таких гиперплоскостей две:  $A$  и  $B$ .

---

<sup>1</sup>Источник: <https://www.cleanpng.com/png-support-vector-machine-machine-learning-statistica-1095606/>



- ✓ Разделяющих гиперплоскостей может быть несколько. На рисунке 2 таких гиперплоскостей две: *A* и *B*.
- ✓ Как выбрать ту, для которой минимальное расстояние до объектов обоих классов будет максимальным?

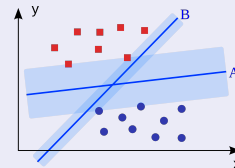


Рис. 2. Пример гиперплоскости<sup>1</sup>

<sup>1</sup>Источник: <https://www.cleanpng.com/png-support-vector-machine-machine-learning-statistica-1095606/>

# SVM. Выбор гиперплоскости

- ✓ Разделяющих гиперплоскостей может быть несколько. На рисунке 2 таких гиперплоскостей две: *A* и *B*.
- ✓ Как выбрать ту, для которой минимальное расстояние до объектов обоих классов будет максимальным?
- ✓ Метод опорных векторов выбирает гиперплоскость, максимизирующую *отступ* («зазор») между классами. Она называется *оптимальной разделяющей* гиперплоскостью. На рисунке 2 видно, такой зазор у гиперплоскости *A*, больше.

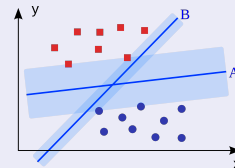


Рис. 2. Пример гиперплоскости<sup>1</sup>

<sup>1</sup>Источник: <https://www.cleanpng.com/png-support-vector-machine-machine-learning-statistica-1095606/>

# SVM. Выбор гиперплоскости

- ✓ Разделяющих гиперплоскостей может быть несколько. На рисунке 2 таких гиперплоскостей две:  $A$  и  $B$ .
- ✓ Как выбрать ту, для которой минимальное расстояние до объектов обоих классов будет максимальным?
- ✓ Метод опорных векторов выбирает гиперплоскость, максимизирующую *отступ («зазор»)* между классами. Она называется *оптимальной разделяющей* гиперплоскостью. На рисунке 2 видно, такой зазор у гиперплоскости  $A$ , больше.
- ✓ Элементы, наиболее близкие к разделяющей гиперплоскости, называют *опорными векторами* — отсюда название метода. На рисунке 2 они лежат на параллельных гиперплоскостях  $L_1$  и  $L_2$ , образующих зазор.

$$L_1 : \langle \mathbf{X}, \mathbf{W} \rangle - b = 1, \quad L_2 : \langle \mathbf{X}, \mathbf{W} \rangle - b = -1. \quad (5)$$

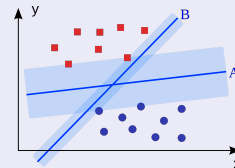


Рис. 2. Пример гиперплоскости<sup>1</sup>

<sup>1</sup>Источник: <https://www.cleanpng.com/png-support-vector-machine-machine-learning-statistica-1095606/>

# SVM. Выбор гиперплоскости

- ✓ Разделяющих гиперплоскостей может быть несколько. На рисунке 2 таких гиперплоскостей две:  $A$  и  $B$ .
- ✓ Как выбрать ту, для которой минимальное расстояние до объектов обоих классов будет максимальным?
- ✓ Метод опорных векторов выбирает гиперплоскость, максимизирующую *отступ* («зазор») между классами. Она называется *оптимальной разделяющей* гиперплоскостью. На рисунке 2 видно, такой зазор у гиперплоскости  $A$ , больше.
- ✓ Элементы, наиболее близкие к разделяющей гиперплоскости, называют *опорными векторами* — отсюда название метода. На рисунке 2 они лежат на параллельных гиперплоскостях  $L_1$  и  $L_2$ , образующих зазор.

$$L_1 : \langle \mathbf{X}, \mathbf{W} \rangle - b = 1, \quad L_2 : \langle \mathbf{X}, \mathbf{W} \rangle - b = -1. \quad (5)$$

- ✓ Как определить вектор нормали  $\mathbf{W}$  и параметр  $b$  оптимальной разделяющей гиперплоскости?

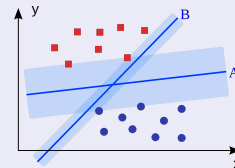


Рис. 2. Пример гиперплоскостей<sup>1</sup>

<sup>1</sup>Источник: <https://www.cleanpng.com/png-support-vector-machine-machine-learning-statistica-1095606/>

# SVM. Выбор гиперплоскости

- ✓ Разделяющих гиперплоскостей может быть несколько. На рисунке 2 таких гиперплоскостей две:  $A$  и  $B$ .
- ✓ Как выбрать ту, для которой минимальное расстояние до объектов обоих классов будет максимальным?
- ✓ Метод опорных векторов выбирает гиперплоскость, максимизирующую *отступ* («зазор») между классами. Она называется *оптимальной разделяющей* гиперплоскостью. На рисунке 2 видно, такой зазор у гиперплоскости  $A$ , больше.
- ✓ Элементы, наиболее близкие к разделяющей гиперплоскости, называют *опорными векторами* — отсюда название метода. На рисунке 2 они лежат на параллельных гиперплоскостях  $L_1$  и  $L_2$ , образующих зазор.

$$L_1 : \langle \mathbf{X}, \mathbf{W} \rangle - b = 1, \quad L_2 : \langle \mathbf{X}, \mathbf{W} \rangle - b = -1. \quad (5)$$

- ✓ Как определить вектор нормали  $\mathbf{W}$  и параметр  $b$  оптимальной разделяющей гиперплоскости?
- ✓ Из геометрических соображений и (5), легко получить что величина зазора равна  $2/\|\mathbf{W}\|$  (по  $1/\|\mathbf{W}\|$  для каждого класса) — достигается на опорных векторах.

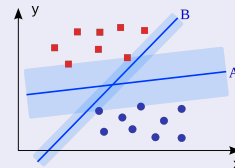


Рис. 2. Пример гиперплоскостей<sup>1</sup>

<sup>1</sup>Источник: <https://www.cleanpng.com/png-support-vector-machine-machine-learning-statistica-1095606/>

# SVM. Выбор гиперплоскости

- ✓ Разделяющих гиперплоскостей может быть несколько. На рисунке 2 таких гиперплоскостей две:  $A$  и  $B$ .
- ✓ Как выбрать ту, для которой минимальное расстояние до объектов обоих классов будет максимальным?
- ✓ Метод опорных векторов выбирает гиперплоскость, максимизирующую *отступ* («зазор») между классами. Она называется *оптимальной разделяющей* гиперплоскостью. На рисунке 2 видно, такой зазор у гиперплоскости  $A$ , больше.
- ✓ Элементы, наиболее близкие к разделяющей гиперплоскости, называют *опорными векторами* — отсюда название метода. На рисунке 2 они лежат на параллельных гиперплоскостях  $L_1$  и  $L_2$ , образующих зазор.

$$L_1 : \langle \mathbf{X}, \mathbf{W} \rangle - b = 1, \quad L_2 : \langle \mathbf{X}, \mathbf{W} \rangle - b = -1. \quad (5)$$

- ✓ Как определить вектор нормали  $\mathbf{W}$  и параметр  $b$  оптимальной разделяющей гиперплоскости?
- ✓ Из геометрических соображений и (5), легко получить что величина зазора равна  $2/\|\mathbf{W}\|$  (по  $1/\|\mathbf{W}\|$  для каждого класса) — достигается на опорных векторах.
- ✓ Для максимального зазора нужно максимизировать  $1/\|\mathbf{W}\|$  или минимизировать  $\|\mathbf{W}\|^2$ .

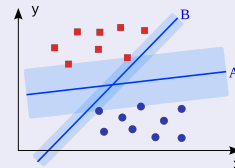


Рис. 2. Пример гиперплоскостей<sup>1</sup>

<sup>1</sup>Источник: <https://www.cleanpng.com/png-support-vector-machine-machine-learning-statistica-1095606/>

# SVM. Выбор гиперплоскости

- ✓ Разделяющих гиперплоскостей может быть несколько. На рисунке 2 таких гиперплоскостей две:  $A$  и  $B$ .
- ✓ Как выбрать ту, для которой минимальное расстояние до объектов обоих классов будет максимальным?
- ✓ Метод опорных векторов выбирает гиперплоскость, максимизирующую *отступ* («зазор») между классами. Она называется *оптимальной разделяющей* гиперплоскостью. На рисунке 2 видно, такой зазор у гиперплоскости  $A$ , больше.
- ✓ Элементы, наиболее близкие к разделяющей гиперплоскости, называют *опорными векторами* — отсюда название метода. На рисунке 2 они лежат на параллельных гиперплоскостях  $L_1$  и  $L_2$ , образующих зазор.

$$L_1 : \langle \mathbf{X}, \mathbf{W} \rangle - b = 1, \quad L_2 : \langle \mathbf{X}, \mathbf{W} \rangle - b = -1. \quad (5)$$

- ✓ Как определить вектор нормали  $\mathbf{W}$  и параметр  $b$  оптимальной разделяющей гиперплоскости?
- ✓ Из геометрических соображений и (5), легко получить что величина зазора равна  $2/\|\mathbf{W}\|$  (по  $1/\|\mathbf{W}\|$  для каждого класса) — достигается на опорных векторах.
- ✓ Для максимального зазора нужно максимизировать  $1/\|\mathbf{W}\|$  или минимизировать  $\|\mathbf{W}\|^2$ .
- ✓ Получаем задачу квадратичного программирования с линейными ограничениями:

$$\begin{cases} \|\mathbf{W}\|^2 \rightarrow \min_{\mathbf{W}, b} \\ y_i (\langle \mathbf{X}_i, \mathbf{W} \rangle - b) \geq 1, \mathbf{X}_i \in \tilde{\mathbf{X}}, i \in 1 : k. \end{cases} \quad (6)$$

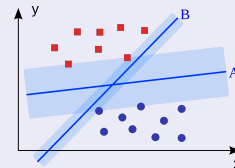


Рис. 2. Пример гиперплоскостей<sup>1</sup>

<sup>1</sup>Источник: <https://www.cleanpng.com/png-support-vector-machine-machine-learning-statistica-1095606/>

# SVM. Выбор гиперплоскости

- ✓ Разделяющих гиперплоскостей может быть несколько. На рисунке 2 таких гиперплоскостей две:  $A$  и  $B$ .
- ✓ Как выбрать ту, для которой минимальное расстояние до объектов обоих классов будет максимальным?
- ✓ Метод опорных векторов выбирает гиперплоскость, максимизирующую *отступ* («зазор») между классами. Она называется *оптимальной разделяющей* гиперплоскостью. На рисунке 2 видно, такой зазор у гиперплоскости  $A$ , больше.
- ✓ Элементы, наиболее близкие к разделяющей гиперплоскости, называют *опорными векторами* — отсюда название метода. На рисунке 2 они лежат на параллельных гиперплоскостях  $L_1$  и  $L_2$ , образующих зазор.

$$L_1 : \langle \mathbf{X}, \mathbf{W} \rangle - b = 1, \quad L_2 : \langle \mathbf{X}, \mathbf{W} \rangle - b = -1. \quad (5)$$

- ✓ Как определить вектор нормали  $\mathbf{W}$  и параметр  $b$  оптимальной разделяющей гиперплоскости?
- ✓ Из геометрических соображений и (5), легко получить что величина зазора равна  $2/\|\mathbf{W}\|$  (по  $1/\|\mathbf{W}\|$  для каждого класса) — достигается на опорных векторах.
- ✓ Для максимального зазора нужно максимизировать  $1/\|\mathbf{W}\|$  или минимизировать  $\|\mathbf{W}\|^2$ .
- ✓ Получаем задачу квадратичного программирования с линейными ограничениями:

$$\begin{cases} \|\mathbf{W}\|^2 \rightarrow \min_{\mathbf{W}, b} \\ y_i (\langle \mathbf{X}_i, \mathbf{W} \rangle - b) \geq 1, \mathbf{X}_i \in \tilde{\mathbf{X}}, i \in 1 : k. \end{cases} \quad (6)$$

- ✓ Задача оптимизации (6) является выпуклой (*взяв квадрат нормы  $\mathbf{W}$* ) — решение единственно (*множители Лагранжа*). Равенство в ограничениях в (6) достигается на опорных векторах.

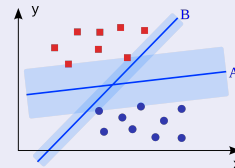


Рис. 2. Пример гиперплоскостей<sup>1</sup>

<sup>1</sup>Источник: <https://www.cleanpng.com/png-support-vector-machine-machine-learning-statistica-1095606/>





---

<sup>2</sup>Источник: <https://logic.pdmi.ras.ru/~yura/internet/07ianote.pdf>

? Линейно разделимые выборки встречаются очень редко. Что делать в этом случае?

<sup>2</sup>Источник: <https://logic.pdmi.ras.ru/~yura/internet/07ianote.pdf>

? Линейно разделимые выборки встречаются очень редко. Что делать в этом случае?

1 **Решение:** пробуем «вложить» обучающую выборку  $\tilde{\mathbf{X}}$  в пространство  $\mathbf{H}$  более высокой размерности с помощью специального отображения

$$\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}. \quad (7)$$

<sup>2</sup>Источник: <https://logic.pdmi.ras.ru/~yura/internet/07ianote.pdf>

? Линейно разделимые выборки встречаются очень редко. Что делать в этом случае?

1 **Решение:** пробуем «вложить» обучающую выборку  $\tilde{\mathbf{X}}$  в пространство  $\mathbf{H}$  более высокой размерности с помощью специального отображения

$$\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}. \quad (7)$$

2 Отображение  $\psi$  в (7) выбираем так, чтобы в пространстве  $\mathbf{H}$  выборка была линейно разделима.

<sup>2</sup>Источник: <https://logic.pdmi.ras.ru/~yura/internet/07ianote.pdf>

? Линейно разделимые выборки встречаются очень редко. Что делать в этом случае?

1 **Решение:** пробуем «вложить» обучающую выборку  $\tilde{\mathbf{X}}$  в пространство  $\mathbf{H}$  более высокой размерности с помощью специального отображения

$$\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}. \quad (7)$$

2 Отображение  $\psi$  в (7) выбираем так, чтобы в пространстве  $\mathbf{H}$  выборка была линейно разделима.

3 Пространство  $\mathbf{H}$  называется *спрямяющим*, а функция перехода  $\psi$  — спрямяющим отображением.

<sup>2</sup>Источник: <https://logic.pdmi.ras.ru/~yura/internet/07ianote.pdf>

- ❓ Линейно разделимые выборки встречаются очень редко. Что делать в этом случае?
- 1 **Решение:** пробуем «вложить» обучающую выборку  $\tilde{\mathbf{X}}$  в пространство  $\mathbf{H}$  более высокой размерности с помощью специального отображения
- $$\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}. \quad (7)$$
- 2 Отображение  $\psi$  в (7) выбираем так, чтобы в пространстве  $\mathbf{H}$  выборка была линейно разделима.
- 3 Пространство  $\mathbf{H}$  называется *спрямяющим*, а функция перехода  $\psi$  — спрямяющим отображением.
- 4 Пример такого спрямяющего отображения на рисунке 3.

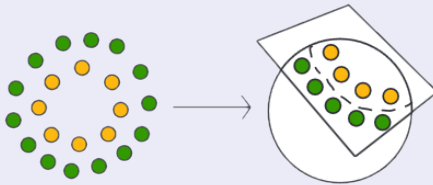


Рис. 3. Пример спрямяющего отображения<sup>2</sup>

<sup>2</sup>Источник: <https://logic.pdmi.ras.ru/~yura/internet/07ianote.pdf>

- ❓ Линейно разделимые выборки встречаются очень редко. Что делать в этом случае?
- 1 **Решение:** пробуем «вложить» обучающую выборку  $\tilde{\mathbf{X}}$  в пространство  $\mathbf{H}$  более высокой размерности с помощью специального отображения
- $$\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}. \quad (7)$$
- 2 Отображение  $\psi$  в (7) выбираем так, чтобы в пространстве  $\mathbf{H}$  выборка была линейно разделима.
- 3 Пространство  $\mathbf{H}$  называется *спрямяющим*, а функция перехода  $\psi$  — спрямяющим отображением.
- 4 Пример такого спрямяющего отображения на рисунке 3.

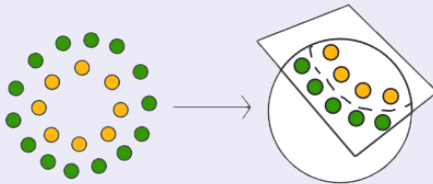


Рис. 3. Пример спрямяющего отображения<sup>2</sup>

- 5 На плоскости  $\mathbf{R}^2$  обучающая выборка желтых и зеленых элементов не является линейно разделимой.

<sup>2</sup>Источник: <https://logic.pdmi.ras.ru/~yura/internet/07ianote.pdf>

- ❓ Линейно разделимые выборки встречаются очень редко. Что делать в этом случае?
- 1 **Решение:** пробуем «вложить» обучающую выборку  $\tilde{\mathbf{X}}$  в пространство  $\mathbf{H}$  более высокой размерности с помощью специального отображения
- $$\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}. \quad (7)$$
- 2 Отображение  $\psi$  в (7) выбираем так, чтобы в пространстве  $\mathbf{H}$  выборка была линейно разделима.
- 3 Пространство  $\mathbf{H}$  называется *спрямяющим*, а функция перехода  $\psi$  — спрямяющим отображением.
- 4 Пример такого спрямяющего отображения на рисунке 3.

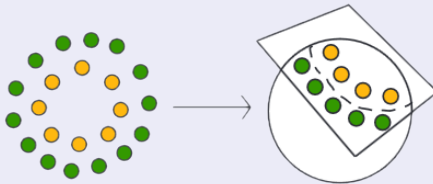


Рис. 3. Пример спрямяющего отображения<sup>2</sup>

- 5 На плоскости  $\mathbf{R}^2$  обучающая выборка желтых и зеленых элементов не является линейно разделимой.
- 6 Однако, если отображением  $\psi$  «вложить» обучающую выборку в  $\mathbf{R}^3$  (на сферу), то они разделяются плоскостью.

<sup>2</sup>Источник: <https://logic.pdmi.ras.ru/~yura/internet/07ianote.pdf>





- 1 После спрямляющего отображения  $\psi : \tilde{X} \rightarrow H$  заменяем:

1 После спрямляющего отображения  $\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}$  заменяем:

✓ элементы выборки  $\mathbf{X} \in \tilde{\mathbf{X}}$  на  $\psi(\mathbf{X}) \in \mathbf{H}$ ;

1 После спрямляющего отображения  $\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}$  заменяем:

- ✓ элементы выборки  $\mathbf{X} \in \tilde{\mathbf{X}}$  на  $\psi(\mathbf{X}) \in \mathbf{H}$ ;
- ✓ скалярное произведение  $\langle \mathbf{X}, \mathbf{W} \rangle_{\mathbf{R}^n}$  в пространстве  $\mathbf{R}^n$  на скалярное произведение  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$  в пространстве  $\mathbf{H}$ .

- 1 После спрямляющего отображения  $\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}$  заменяем:
  - ✓ элементы выборки  $\mathbf{X} \in \tilde{\mathbf{X}}$  на  $\psi(\mathbf{X}) \in \mathbf{H}$ ;
  - ✓ скалярное произведение  $\langle \mathbf{X}, \mathbf{W} \rangle_{\mathbf{R}^n}$  в пространстве  $\mathbf{R}^n$  на скалярное произведение  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$  в пространстве  $\mathbf{H}$ .
- 2 Для классификатора SVM (4) требуются только скалярные произведения  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$ .

- 1 После спрямляющего отображения  $\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}$  заменяем:
  - ✓ элементы выборки  $\mathbf{X} \in \tilde{\mathbf{X}}$  на  $\psi(\mathbf{X}) \in \mathbf{H}$ ;
  - ✓ скалярное произведение  $\langle \mathbf{X}, \mathbf{W} \rangle_{\mathbf{R}^n}$  в пространстве  $\mathbf{R}^n$  на скалярное произведение  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$  в пространстве  $\mathbf{H}$ .
- 2 Для классификатора SVM (4) требуются только скалярные произведения  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$ .
- 3 Вычислять в явном виде  $\psi(\mathbf{X})$  дорого — может иметь большую размерность.

- 1 После спрямляющего отображения  $\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}$  заменяем:
  - ✓ элементы выборки  $\mathbf{X} \in \tilde{\mathbf{X}}$  на  $\psi(\mathbf{X}) \in \mathbf{H}$ ;
  - ✓ скалярное произведение  $\langle \mathbf{X}, \mathbf{W} \rangle_{\mathbf{R}^n}$  в пространстве  $\mathbf{R}^n$  на скалярное произведение  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$  в пространстве  $\mathbf{H}$ .
- 2 Для классификатора SVM (4) требуются только скалярные произведения  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$ .
- 3 Вычислять в явном виде  $\psi(\mathbf{X})$  дорого — может иметь большую размерность.
- 4 Объединяем вычисление  $\psi(\mathbf{X})$  и скалярного произведения в единую операцию.

- 1 После спрямляющего отображения  $\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}$  заменяем:
  - ✓ элементы выборки  $\mathbf{X} \in \tilde{\mathbf{X}}$  на  $\psi(\mathbf{X}) \in \mathbf{H}$ ;
  - ✓ скалярное произведение  $\langle \mathbf{X}, \mathbf{W} \rangle_{\mathbf{R}^n}$  в пространстве  $\mathbf{R}^n$  на скалярное произведение  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$  в пространстве  $\mathbf{H}$ .
- 2 Для классификатора SVM (4) требуются только скалярные произведения  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$ .
- 3 Вычислять в явном виде  $\psi(\mathbf{X})$  дорого — может иметь большую размерность.
- 4 Объединяем вычисление  $\psi(\mathbf{X})$  и скалярного произведения в единую операцию.
- 5 Ищем *ядро* такую функцию  $K(\mathbf{X}, \mathbf{Y})$ , что:

$$K(\mathbf{X}, \mathbf{Y}) = \langle \psi(\mathbf{X}), \psi(\mathbf{Y}) \rangle_{\mathbf{H}}, \mathbf{X}, \mathbf{Y} \in \tilde{\mathbf{X}}. \quad (8)$$



1 После спрямляющего отображения  $\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}$  заменяем:

- ✓ элементы выборки  $\mathbf{X} \in \tilde{\mathbf{X}}$  на  $\psi(\mathbf{X}) \in \mathbf{H}$ ;
- ✓ скалярное произведение  $\langle \mathbf{X}, \mathbf{W} \rangle_{\mathbf{R}^n}$  в пространстве  $\mathbf{R}^n$  на скалярное произведение  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$  в пространстве  $\mathbf{H}$ .

2 Для классификатора SVM (4) требуются только скалярные произведения  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$ .

3 Вычислять в явном виде  $\psi(\mathbf{X})$  дорого — может иметь большую размерность.

4 Объединяем вычисление  $\psi(\mathbf{X})$  и скалярного произведения в единую операцию.

5 Ищем *ядро* такую функцию  $K(\mathbf{X}, \mathbf{Y})$ , что:

$$K(\mathbf{X}, \mathbf{Y}) = \langle \psi(\mathbf{X}), \psi(\mathbf{Y}) \rangle_{\mathbf{H}}, \mathbf{X}, \mathbf{Y} \in \tilde{\mathbf{X}}. \quad (8)$$

6 Описанный прием называется *трюком с ядром* (kernel trick).

# Линейная неразделимость. Трюк с ядром

1 После спрямляющего отображения  $\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}$  заменяем:

- ✓ элементы выборки  $\mathbf{X} \in \tilde{\mathbf{X}}$  на  $\psi(\mathbf{X}) \in \mathbf{H}$ ;
- ✓ скалярное произведение  $\langle \mathbf{X}, \mathbf{W} \rangle_{\mathbf{R}^n}$  в пространстве  $\mathbf{R}^n$  на скалярное произведение  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$  в пространстве  $\mathbf{H}$ .

2 Для классификатора SVM (4) требуются только скалярные произведения  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$ .

3 Вычислять в явном виде  $\psi(\mathbf{X})$  дорого — может иметь большую размерность.

4 Объединяем вычисление  $\psi(\mathbf{X})$  и скалярного произведения в единую операцию.

5 Ищем *ядро* такую функцию  $K(\mathbf{X}, \mathbf{Y})$ , что:

$$K(\mathbf{X}, \mathbf{Y}) = \langle \psi(\mathbf{X}), \psi(\mathbf{Y}) \rangle_{\mathbf{H}}, \mathbf{X}, \mathbf{Y} \in \tilde{\mathbf{X}}. \quad (8)$$

6 Описанный прием называется *трюком с ядром* (kernel trick).

i Ядром может служить любая *положительно определенная* и *симметричная* функция двух переменных.

# Линейная неразделимость. Трюк с ядром

- 1 После спрямляющего отображения  $\psi : \tilde{\mathbf{X}} \rightarrow \mathbf{H}$  заменяем:
  - ✓ элементы выборки  $\mathbf{X} \in \tilde{\mathbf{X}}$  на  $\psi(\mathbf{X}) \in \mathbf{H}$ ;
  - ✓ скалярное произведение  $\langle \mathbf{X}, \mathbf{W} \rangle_{\mathbf{R}^n}$  в пространстве  $\mathbf{R}^n$  на скалярное произведение  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$  в пространстве  $\mathbf{H}$ .
- 2 Для классификатора SVM (4) требуются только скалярные произведения  $\langle \psi(\mathbf{X}), \psi(\mathbf{W}) \rangle_{\mathbf{H}}$ .
- 3 Вычислять в явном виде  $\psi(\mathbf{X})$  дорого — может иметь большую размерность.
- 4 Объединяем вычисление  $\psi(\mathbf{X})$  и скалярного произведения в единую операцию.
- 5 Ищем *ядро* такую функцию  $K(\mathbf{X}, \mathbf{Y})$ , что:

$$K(\mathbf{X}, \mathbf{Y}) = \langle \psi(\mathbf{X}), \psi(\mathbf{Y}) \rangle_{\mathbf{H}}, \mathbf{X}, \mathbf{Y} \in \tilde{\mathbf{X}}. \quad (8)$$
- 6 Описанный прием называется *трюком с ядром* (kernel trick).

- i Ядром может служить любая *положительно определенная* и *симметричная* функция двух переменных.
- i Положительная определенность нужна для корректного решения задачи оптимизации (определения разделяющей гиперплоскости).



1 *Полиномиальное* ядро (Polynomial kernel):

$$K(\mathbf{X}, \mathbf{Y}) = (\langle \mathbf{X}, \mathbf{Y} \rangle + \alpha)^d, \quad d > 0. \quad (9)$$

- 1 *Полиномиальное* ядро (Polynomial kernel):

$$K(\mathbf{X}, \mathbf{Y}) = (\langle \mathbf{X}, \mathbf{Y} \rangle + \alpha)^d, \quad d > 0. \quad (9)$$

- 2 *Гауссова радиальная базисная функция* (Gaussian radial basis function, RBF):

$$K(\mathbf{X}, \mathbf{Y}) = e^{-\gamma/\|\mathbf{X}-\mathbf{Y}\|^2}, \quad \gamma > 0. \quad (10)$$

# Часто используемые ядра

- 1 *Полиномиальное* ядро (Polynomial kernel):

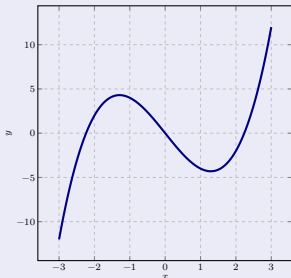
$$K(\mathbf{X}, \mathbf{Y}) = (\langle \mathbf{X}, \mathbf{Y} \rangle + \alpha)^d, \quad d > 0. \quad (9)$$

- 2 *Гауссова радиальная базисная функция* (Gaussian radial basis function, RBF):

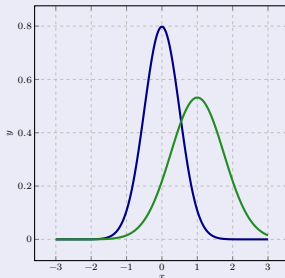
$$K(\mathbf{X}, \mathbf{Y}) = e^{-\gamma/\|\mathbf{X}-\mathbf{Y}\|^2}, \quad \gamma > 0. \quad (10)$$

- 3 *Сигмоидальное* ядро (Sigmoid kernel):

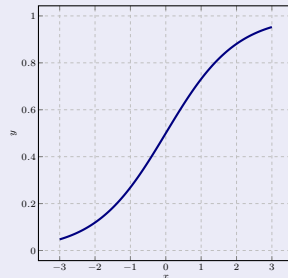
$$K(\mathbf{X}, \mathbf{Y}) = \tanh(\gamma \langle \mathbf{X}, \mathbf{Y} \rangle + r). \quad (11)$$



(а) Полиномиальное ядро



(б) Гауссово ядро



(в) Сигмоидальное ядро

Рис. 4. Часто используемые ядра





```
# Connecting the necessary libraries
import numpy as np
from sklearn import svm

# Creating a data set
np.random.seed(0)
X = np.r_[np.random.randn(20, 2) - [2, 2],
          np.random.randn(20, 2) + [2, 2]]
Y = [0] * 20 + [1] * 20

# Fit the model
clf = svm.SVC(kernel='linear')
clf.fit(X, Y)

# Draw the separating hypoplane
w = clf.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(-5, 5)
yy = a * xx - (clf.intercept_[0]) / w[1]

# Construct lines through the reference vectors
b = clf.support_vectors_[0]
yy_down = a * xx + (b[1] - a * b[0])
b = clf.support_vectors_[-1]
yy_up = a * xx + (b[1] - a * b[0])
```

```
# Connecting the necessary libraries
import numpy as np
from sklearn import svm

# Creating a data set
np.random.seed(0)
X = np.r_[np.random.randn(20, 2) - [2, 2],
          np.random.randn(20, 2) + [2, 2]]
Y = [0] * 20 + [1] * 20

# Fit the model
clf = svm.SVC(kernel='linear')
clf.fit(X, Y)

# Draw the separating hypoplane
w = clf.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(-5, 5)
yy = a * xx - (clf.intercept_[0]) / w[1]

# Construct lines through the reference vectors
b = clf.support_vectors_[0]
yy_down = a * xx + (b[1] - a * b[0])
b = clf.support_vectors_[-1]
yy_up = a * xx + (b[1] - a * b[0])
```

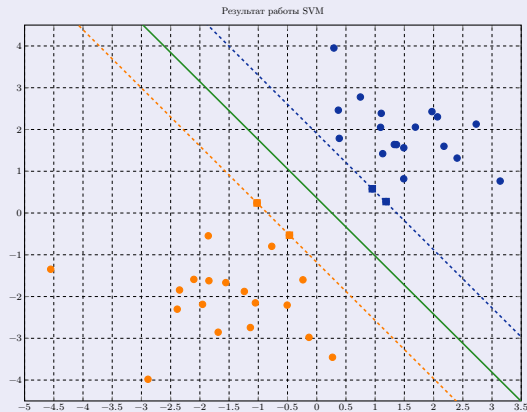
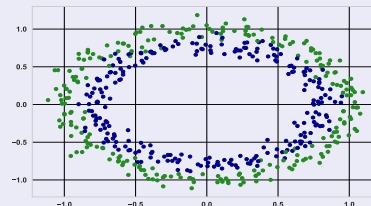


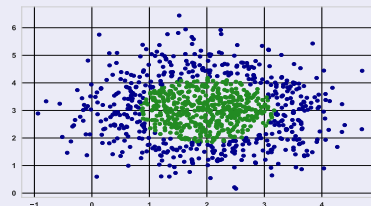
Рис. 5



1 Рассмотрим два примера линейно неразделимых множеств. В каждом примере два класса: рисунок 6:a и рисунок 6:b.



(a)



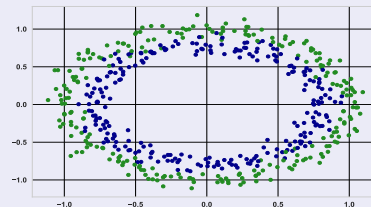
(b)

Рис. 6

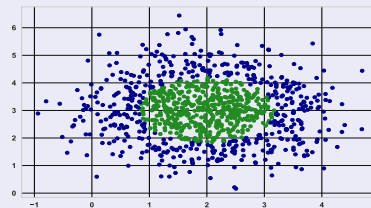
1 Рассмотрим два примера линейно неразделимых множеств. В каждом примере два класса: рисунок 6:a и рисунок 6:b.

2 Множества X1 и X2 сгенерированы следующим образом:

```
from sklearn.datasets import make_circles
X1, y1 = make_circles(n_samples=500, noise=0.06, random_state=42)
from sklearn.datasets import make_gaussian_quantiles
X2, y2 = make_gaussian_quantiles(n_features=2, n_classes=2,
                                  n_samples=1000, mean=(2,3))
```



(a)



(b)

Рис. 6

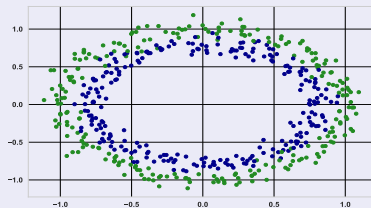
1 Рассмотрим два примера линейно неразделимых множеств. В каждом примере два класса: рисунок 6:a и рисунок 6:b.

2 Множества X1 и X2 сгенерированы следующим образом:

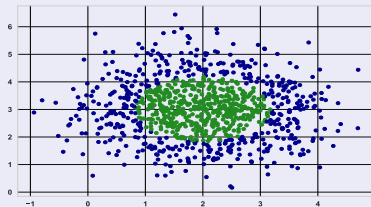
```
from sklearn.datasets import make_circles
X1, y1 = make_circles(n_samples=500, noise=0.06, random_state=42)
from sklearn.datasets import make_gaussian_quantiles
X2, y2 = make_gaussian_quantiles(n_features=2, n_classes=2,
                                  n_samples=1000, mean=(2,3))
```

3 Используем *линейное ядро*. Качество работы алгоритма (*Accuracy*) ожидаемо очень низкое, соответственно 0.496 и 0.623:

```
from sklearn import svm
linear_svc = svm.SVC(kernel='linear').fit(X, y)
```



(a)



(b)

Рис. 6

1 Рассмотрим два примера линейно неразделимых множеств. В каждом примере два класса: рисунок 6:a и рисунок 6:b.

2 Множества X1 и X2 сгенерированы следующим образом:

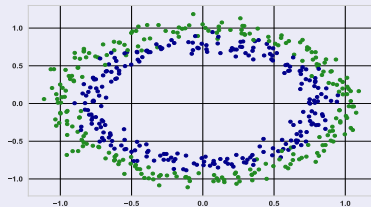
```
from sklearn.datasets import make_circles
X1, y1 = make_circles(n_samples=500, noise=0.06, random_state=42)
from sklearn.datasets import make_gaussian_quantiles
X2, y2 = make_gaussian_quantiles(n_features=2, n_classes=2,
                                  n_samples=1000, mean=(2,3))
```

3 Используем *линейное ядро*. Качество работы алгоритма (*Accuracy*) ожидаемо очень низкое, соответственно 0.496 и 0.623:

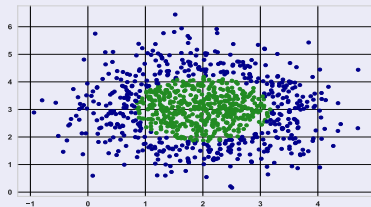
```
from sklearn import svm
linear_svc = svm.SVC(kernel='linear').fit(X, y)
```

4 Используем *RBF-ядро*. Качество работы алгоритма: 0.930 и 0.993:

```
rbf_svc = svm.SVC(kernel='rbf', gamma=0.7, C=1).fit(X, y)
```



(a)



(b)

Рис. 6

1 Рассмотрим два примера линейно неразделимых множеств. В каждом примере два класса: рисунок 6:a и рисунок 6:b.

2 Множества X1 и X2 сгенерированы следующим образом:

```
from sklearn.datasets import make_circles
X1, y1 = make_circles(n_samples=500, noise=0.06, random_state=42)
from sklearn.datasets import make_gaussian_quantiles
X2, y2 = make_gaussian_quantiles(n_features=2, n_classes=2,
                                  n_samples=1000, mean=(2,3))
```

3 Используем *линейное ядро*. Качество работы алгоритма (*Accuracy*) ожидаемо очень низкое, соответственно 0.496 и 0.623:

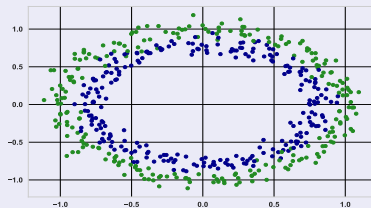
```
from sklearn import svm
linear_svc = svm.SVC(kernel='linear').fit(X, y)
```

4 Используем *RBF-ядро*. Качество работы алгоритма: 0.930 и 0.993:

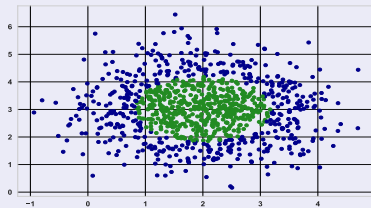
```
rbf_svc = svm.SVC(kernel='rbf', gamma=0.7, C=1).fit(X, y)
```

5 Используем *полиномиальное ядро*, степени 2. Качество работы алгоритма: 0.942 и 0.782:

```
poly_svc = svm.SVC(kernel='poly', degree=2, C=1).fit(X, y)
```



(a)



(b)

Рис. 6





- Пробуем применить бинарную классификацию SVM, когда число возможных классов больше двух:  $\mathbf{Y} = \{y_1, \dots, y_p\}, p > 2$ .

- Пробуем применить бинарную классификацию SVM, когда число возможных классов больше двух:  $\mathbf{Y} = \{y_1, \dots, y_p\}, p > 2$ .
- Разобьем задачу на бинарные классификаторы по схеме *«один против остальных»* (One-vs-Rest).

- Пробуем применить бинарную классификацию SVM, когда число возможных классов больше двух:  $\mathbf{Y} = \{y_1, \dots, y_p\}$ ,  $p > 2$ .
- Разобьем задачу на бинарные классификаторы по схеме *«один против остальных»* (One-vs-Rest).
- Построим  $p$  бинарных классификаторов SVM.

- Пробуем применить бинарную классификацию SVM, когда число возможных классов больше двух:  $\mathbf{Y} = \{y_1, \dots, y_p\}$ ,  $p > 2$ .
- Разобьем задачу на бинарные классификаторы по схеме «*один против остальных*» (One-vs-Rest).
- Построим  $p$  бинарных классификаторов SVM.

## Алгоритм

- 1 Для каждого признака  $y_i \in \mathbf{Y}$  строим бинарный классификатор  $f_i$ .

- Попробуем применить бинарную классификацию SVM, когда число возможных классов больше двух:  $\mathbf{Y} = \{y_1, \dots, y_p\}$ ,  $p > 2$ .
- Разобьем задачу на бинарные классификаторы по схеме «один против остальных» (One-vs-Rest).
- Построим  $p$  бинарных классификаторов SVM.

## Алгоритм

- 1 Для каждого признака  $y_i \in \mathbf{Y}$  строим бинарный классификатор  $f_i$ .
- 2 Переводим метки классов из  $\mathbf{Y}$  в  $\{-1, 1\}$ :

$$\begin{cases} \tilde{y}_j = 1, & \text{если } j = i, \\ \tilde{y}_j = -1, & \text{если } j \neq i. \end{cases} \quad (12)$$

# Мультиклассовый SVM. Один против остальных

- Пробуем применить бинарную классификацию SVM, когда число возможных классов больше двух:  $\mathbf{Y} = \{y_1, \dots, y_p\}$ ,  $p > 2$ .
- Разобьем задачу на бинарные классификаторы по схеме «один против остальных» (One-vs-Rest).
- Построим  $p$  бинарных классификаторов SVM.

## Алгоритм

- 1 Для каждого признака  $y_i \in \mathbf{Y}$  строим бинарный классификатор  $f_i$ .
- 2 Переводим метки классов из  $\mathbf{Y}$  в  $\{-1, 1\}$ :

$$\begin{cases} \tilde{y}_j = 1, & \text{если } j = i, \\ \tilde{y}_j = -1, & \text{если } j \neq i. \end{cases} \quad (12)$$

- 3 Строим классификатор  $f_i$  на множестве прецедентов  $\{(\mathbf{X}_s, \tilde{y}_j) \mid \mathbf{X}_s \in \tilde{\mathbf{X}}, j \in 1:p\}$ .

$$f_i(\mathbf{X}) = \langle \mathbf{X}, \mathbf{W} \rangle - b, \quad \mathbf{W} \in \mathbf{R}^n, \mathbf{X} \in \tilde{\mathbf{X}}. \quad (13)$$

# Мультиклассовый SVM. Один против остальных

- Пробуем применить бинарную классификацию SVM, когда число возможных классов больше двух:  $\mathbf{Y} = \{y_1, \dots, y_p\}, p > 2$ .
- Разобьем задачу на бинарные классификаторы по схеме «*один против остальных*» (One-vs-Rest).
- Построим  $p$  бинарных классификаторов SVM.

## Алгоритм

- 1 Для каждого признака  $y_i \in \mathbf{Y}$  строим бинарный классификатор  $f_i$ .
- 2 Переводим метки классов из  $\mathbf{Y}$  в  $\{-1, 1\}$ :

$$\begin{cases} \tilde{y}_j = 1, & \text{если } j = i, \\ \tilde{y}_j = -1, & \text{если } j \neq i. \end{cases} \quad (12)$$

- 3 Строим классификатор  $f_i$  на множестве прецедентов  $\{(\mathbf{X}_s, \tilde{y}_j) \mid \mathbf{X}_s \in \tilde{\mathbf{X}}, j \in 1:p\}$ .

$$f_i(\mathbf{X}) = \langle \mathbf{X}, \mathbf{W} \rangle - b, \quad \mathbf{W} \in \mathbf{R}^n, \mathbf{X} \in \tilde{\mathbf{X}}. \quad (13)$$

- ✓ Оптимальная разделяющая гиперплоскость классификатора  $f_i$  отделяет точки  $i$ -го класса от точек всех остальных классов.



# Мультиклассовый SVM. Один против остальных

- Пробуем применить бинарную классификацию SVM, когда число возможных классов больше двух:  $\mathbf{Y} = \{y_1, \dots, y_p\}, p > 2$ .
- Разобьем задачу на бинарные классификаторы по схеме «*один против остальных*» (One-vs-Rest).
- Построим  $p$  бинарных классификаторов SVM.

## Алгоритм

- 1 Для каждого признака  $y_i \in \mathbf{Y}$  строим бинарный классификатор  $f_i$ .
- 2 Переводим метки классов из  $\mathbf{Y}$  в  $\{-1, 1\}$ :

$$\begin{cases} \tilde{y}_j = 1, & \text{если } j = i, \\ \tilde{y}_j = -1, & \text{если } j \neq i. \end{cases} \quad (12)$$

- 3 Строим классификатор  $f_i$  на множестве прецедентов  $\{(\mathbf{X}_s, \tilde{y}_j) \mid \mathbf{X}_s \in \tilde{\mathbf{X}}, j \in 1:p\}$ .

$$f_i(\mathbf{X}) = \langle \mathbf{X}, \mathbf{W} \rangle - b, \quad \mathbf{W} \in \mathbf{R}^n, \mathbf{X} \in \tilde{\mathbf{X}}. \quad (13)$$

- ✓ Оптимальная разделяющая гиперплоскость классификатора  $f_i$  отделяет точки  $i$ -го класса от точек всех остальных классов.
- ✓ Элементы  $i$ -го класса находятся в положительном, остальные элементы — в отрицательном полупространстве.

# Мультиклассовый SVM. Один против остальных

- Пробуем применить бинарную классификацию SVM, когда число возможных классов больше двух:  $\mathbf{Y} = \{y_1, \dots, y_p\}, p > 2$ .
- Разобьем задачу на бинарные классификаторы по схеме «один против остальных» (One-vs-Rest).
- Построим  $p$  бинарных классификаторов SVM.

## Алгоритм

- 1 Для каждого признака  $y_i \in \mathbf{Y}$  строим бинарный классификатор  $f_i$ .
- 2 Переводим метки классов из  $\mathbf{Y}$  в  $\{-1, 1\}$ :

$$\begin{cases} \tilde{y}_j = 1, & \text{если } j = i, \\ \tilde{y}_j = -1, & \text{если } j \neq i. \end{cases} \quad (12)$$

- 3 Строим классификатор  $f_i$  на множестве прецедентов  $\{(\mathbf{X}_s, \tilde{y}_j) \mid \mathbf{X}_s \in \tilde{\mathbf{X}}, j \in 1:p\}$ .

$$f_i(\mathbf{X}) = \langle \mathbf{X}, \mathbf{W} \rangle - b, \quad \mathbf{W} \in \mathbf{R}^n, \mathbf{X} \in \tilde{\mathbf{X}}. \quad (13)$$

- ✓ Оптимальная разделяющая гиперплоскость классификатора  $f_i$  отделяет точки  $i$ -го класса от точек всех остальных классов.
- ✓ Элементы  $i$ -го класса находятся в положительном, остальные элементы — в отрицательном полупространстве.
- ✓ Значение  $f_i(\mathbf{X})$  в (13) пропорционально расстоянию (со знаком) от элемента  $\mathbf{X}$  до гиперплоскости.

# Мультиклассовый SVM. Один против остальных

- Пробуем применить бинарную классификацию SVM, когда число возможных классов больше двух:  $\mathbf{Y} = \{y_1, \dots, y_p\}, p > 2$ .
- Разобьем задачу на бинарные классификаторы по схеме «*один против остальных*» (One-vs-Rest).
- Построим  $p$  бинарных классификаторов SVM.

## Алгоритм

- 1 Для каждого признака  $y_i \in \mathbf{Y}$  строим бинарный классификатор  $f_i$ .
- 2 Переводим метки классов из  $\mathbf{Y}$  в  $\{-1, 1\}$ :

$$\begin{cases} \tilde{y}_j = 1, & \text{если } j = i, \\ \tilde{y}_j = -1, & \text{если } j \neq i. \end{cases} \quad (12)$$

- 3 Строим классификатор  $f_i$  на множестве прецедентов  $\{(\mathbf{X}_s, \tilde{y}_j) \mid \mathbf{X}_s \in \tilde{\mathbf{X}}, j \in 1:p\}$ .

$$f_i(\mathbf{X}) = \langle \mathbf{X}, \mathbf{W} \rangle - b, \quad \mathbf{W} \in \mathbf{R}^n, \mathbf{X} \in \tilde{\mathbf{X}}. \quad (13)$$

- ✓ Оптимальная разделяющая гиперплоскость классификатора  $f_i$  отделяет точки  $i$ -го класса от точек всех остальных классов.
  - ✓ Элементы  $i$ -го класса находятся в положительном, остальные элементы — в отрицательном полупространстве.
  - ✓ Значение  $f_i(\mathbf{X})$  в (13) пропорционально расстоянию (со знаком) от элемента  $\mathbf{X}$  до гиперплоскости.
- 4 Принадлежность элемента  $\mathbf{X}$  классу  $i$  определяем по максимальному расстоянию  $f_i$ :

$$i = \operatorname{argmax}_{j \in 1:p} f_j(\mathbf{X}). \quad (14)$$



# Один против остальных. Пример на Питоне

```
# Connecting the necessary libraries
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Creating a data set
iris = datasets.load_iris(); X = iris.data; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

# Feature Scaling using StandardScaler
sc = StandardScaler().fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model and define strategy and fit model
ovr = OneVsRestClassifier(SVC()).fit(X_tr_std, y_tr)

# make predictions
y_ovr = ovr.predict(X_t_std);
```

# Один против остальных. Пример на Питоне

```
# Connecting the necessary libraries
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

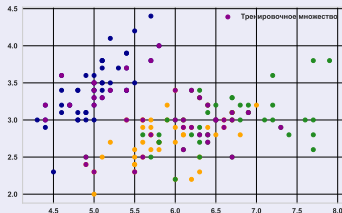
# Creating a data set
iris = datasets.load_iris(); X = iris.data; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

# Feature Scaling using StandardScaler
sc = StandardScaler().fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model and define strategy and fit model
ovr = OneVsRestClassifier(SVC()).fit(X_tr_std, y_tr)

# make predictions
y_ovr = ovr.predict(X_t_std);
```



(а) Исходный набор (Ирисы)

# Один против остальных. Пример на Питоне

```
# Connecting the necessary libraries
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

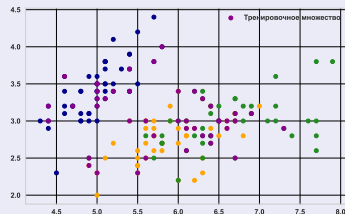
# Creating a data set
iris = datasets.load_iris(); X = iris.data; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

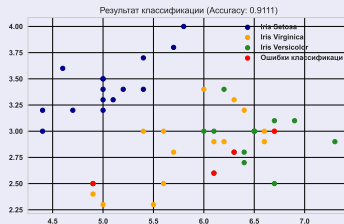
# Feature Scaling using StandardScaler
sc = StandardScaler().fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model and define strategy and fit model
ovr = OneVsRestClassifier(SVC()).fit(X_tr_std, y_tr)

# make predictions
y_ovr = ovr.predict(X_t_std);
```



(a) Исходный набор (Ирисы)



(b) Результаты классификации








Рассматриваемый подход *«каждый против каждого»* (One-vs-One) является противоположным предыдущему. Иногда его называют *парным подходом*.




Рассматриваемый подход *«каждый против каждого»* (One-vs-One) является противоположным предыдущему. Иногда его называют *парным подходом*.

## Алгоритм

 Рассматриваемый подход *«каждый против каждого»* (One-vs-One) является противоположным предыдущему. Иногда его называют *парным подходом*.

## Алгоритм

- 1 Для каждой пары признаков  $y_i, y_j \in \mathbf{Y} = \{y_1, \dots, y_p\}$  строим бинарный классификатор  $f_{ij}$ .

 Рассматриваемый подход «*каждый против каждого*» (One-vs-One) является противоположным предыдущему. Иногда его называют *парным подходом*.


## Алгоритм

- 1 Для каждой пары признаков  $y_i, y_j \in \mathbf{Y} = \{y_1, \dots, y_p\}$  строим бинарный классификатор  $f_{ij}$ .
- 2 Классификатор  $f_{ij}$  строим по подвыборке  $\mathbf{X}_{ij}$ , содержащей только объекты классов  $y_i$  и  $y_j$ .

 Рассматриваемый подход «*каждый против каждого*» (One-vs-One) является противоположным предыдущему. Иногда его называют *парным подходом*.

## Алгоритм

- 1 Для каждой пары признаков  $y_i, y_j \in Y = \{y_1, \dots, y_p\}$  строим бинарный классификатор  $f_{ij}$ .
- 2 Классификатор  $f_{ij}$  строим по подвыборке  $X_{ij}$ , содержащей только объекты классов  $y_i$  и  $y_j$ .
- 3 Нужно построить  $p(p-1)/2$  бинарных классификаторов для всех упорядоченных пар признаков  $1 \leq i < j \leq p$ .

 Рассматриваемый подход «*каждый против каждого*» (One-vs-One) является противоположным предыдущему. Иногда его называют *парным подходом*.

## Алгоритм

- 1 Для каждой пары признаков  $y_i, y_j \in Y = \{y_1, \dots, y_p\}$  строим бинарный классификатор  $f_{ij}$ .
- 2 Классификатор  $f_{ij}$  строим по подвыборке  $X_{ij}$ , содержащей только объекты классов  $y_i$  и  $y_j$ .
- 3 Нужно построить  $p(p-1)/2$  бинарных классификаторов для всех упорядоченных пар признаков  $1 \leq i < j \leq p$ .
- 4 Для классификации нового объекта  $X$ , его подаем на вход каждого из построенных бинарных классификаторов.

 Рассматриваемый подход «*каждый против каждого*» (One-vs-One) является противоположным предыдущему. Иногда его называют *парным подходом*.

## Алгоритм

- 1 Для каждой пары признаков  $y_i, y_j \in Y = \{y_1, \dots, y_p\}$  строим бинарный классификатор  $f_{ij}$ .
- 2 Классификатор  $f_{ij}$  строим по подвыборке  $X_{ij}$ , содержащей только объекты классов  $y_i$  и  $y_j$ .
- 3 Нужно построить  $p(p-1)/2$  бинарных классификаторов для всех упорядоченных пар признаков  $1 \leq i < j \leq p$ .
- 4 Для классификации нового объекта  $X$ , его подаем на вход каждого из построенных бинарных классификаторов.
- 5 Каждый из них проголосует за свой класс.

$$\begin{cases} f_{ij}(X) = 1, \text{ если } X \text{ определен, как } y_i, \\ f_{ij}(X) = -1, \text{ если } X \text{ определен, как } y_j. \end{cases} \quad (15)$$

**i** Рассматриваемый подход «*каждый против каждого*» (One-vs-One) является противоположным предыдущему. Иногда его называют *парным подходом*.

## Алгоритм

- 1 Для каждой пары признаков  $y_i, y_j \in Y = \{y_1, \dots, y_p\}$  строим бинарный классификатор  $f_{ij}$ .
- 2 Классификатор  $f_{ij}$  строим по подвыборке  $X_{ij}$ , содержащей только объекты классов  $y_i$  и  $y_j$ .
- 3 Нужно построить  $p(p-1)/2$  бинарных классификаторов для всех упорядоченных пар признаков  $1 \leq i < j \leq p$ .
- 4 Для классификации нового объекта  $X$ , его подаем на вход каждого из построенных бинарных классификаторов.
- 5 Каждый из них проголосует за свой класс.

$$\begin{cases} f_{ij}(X) = 1, \text{ если } X \text{ определен, как } y_i, \\ f_{ij}(X) = -1, \text{ если } X \text{ определен, как } y_j. \end{cases} \quad (15)$$

**i** Аналогия с турниром  $p$  команд по круговой системе (без ничьих, победа — 1 очко, поражение — -1).



# Мультиклассовый SVM. Каждый против каждого

**i** Рассматриваемый подход «*каждый против каждого*» (One-vs-One) является противоположным предыдущему. Иногда его называют *парным подходом*.

## Алгоритм

- 1 Для каждой пары признаков  $y_i, y_j \in Y = \{y_1, \dots, y_p\}$  строим бинарный классификатор  $f_{ij}$ .
- 2 Классификатор  $f_{ij}$  строим по подвыборке  $X_{ij}$ , содержащей только объекты классов  $y_i$  и  $y_j$ .
- 3 Нужно построить  $p(p-1)/2$  бинарных классификаторов для всех упорядоченных пар признаков  $1 \leq i < j \leq p$ .
- 4 Для классификации нового объекта  $X$ , его подаем на вход каждого из построенных бинарных классификаторов.
- 5 Каждый из них проголосует за свой класс.

$$\begin{cases} f_{ij}(X) = 1, \text{ если } X \text{ определен, как } y_i, \\ f_{ij}(X) = -1, \text{ если } X \text{ определен, как } y_j. \end{cases} \quad (15)$$

- i** Аналогия с турниром  $p$  команд по круговой системе (без ничьих, победа — 1 очко, поражение — -1).
- i** Побеждает класс  $y^*$ , набравший наибольшее число очков — «дома» и «в гостях» (выигравшей круговой турнир):

$$y^* = \operatorname{argmax}_{i \in 1:p} \left( \sum_{1 \leq i < j \leq p} (f_{ij}(X) == 1) + \sum_{1 \leq j < i \leq p} (f_{ji}(X) == -1) \right). \quad (16)$$



# Каждый против каждого. Пример на Питоне

```
# Connecting the necessary libraries
from sklearn.svm import SVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Creating a data set
iris = datasets.load_iris()
X = iris.data; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

# Feature Scaling using StandardScaler
sc = StandardScaler().fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# define strategy and fit model
ovo = OneVsOneClassifier(SVC()).fit(X_tr_std, y_tr)

# make predictions
y_ovo = ovo.predict(X_t_std)
```

```
# Connecting the necessary libraries
from sklearn.svm import SVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

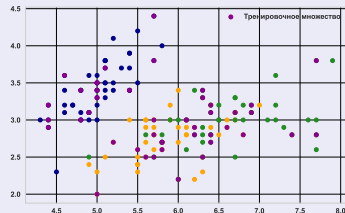
# Creating a data set
iris = datasets.load_iris()
X = iris.data; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                         random_state=42, stratify=y)

# Feature Scaling using StandardScaler
sc = StandardScaler().fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# define strategy and fit model
ovo = OneVsOneClassifier(SVC()).fit(X_tr_std, y_tr)

# make predictions
y_ovo = ovo.predict(X_t_std)
```



(а) Исходный набор (Ирисы)

```
# Connecting the necessary libraries
from sklearn.svm import SVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

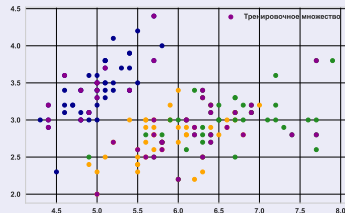
# Creating a data set
iris = datasets.load_iris()
X = iris.data; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                         random_state=42, stratify=y)

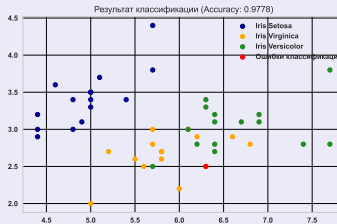
# Feature Scaling using StandardScaler
sc = StandardScaler().fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# define strategy and fit model
ovo = OneVsOneClassifier(SVC()).fit(X_tr_std, y_tr)

# make predictions
y_ovo = ovo.predict(X_t_std)
```



(a) Исходный набор (Ирисы)



(b) Результаты классификации

---

<sup>3</sup>Источник: [https://www.researchgate.net/publication/267953942\\_Video\\_-based\\_Fall\\_Detection\\_in\\_Elderly's\\_Houses](https://www.researchgate.net/publication/267953942_Video_-based_Fall_Detection_in_Elderly's_Houses)

## Идея

- *Гипотеза компактности*: если функция расстояния  $\rho$  между элементами введена «удачно», то похожие элементы гораздо чаще лежат в одном классе, чем в разных.

<sup>3</sup>Источник: [https://www.researchgate.net/publication/267953942\\_Video\\_-based\\_Fall\\_Detection\\_in\\_Elderly's\\_Houses](https://www.researchgate.net/publication/267953942_Video_-based_Fall_Detection_in_Elderly's_Houses)

## Идея

- ☛ *Гипотеза компактности*: если функция расстояния  $\rho$  между элементами введена «удачно», то похожие элементы гораздо чаще лежат в одном классе, чем в разных.
- ☛ Смотрим на соседей элемента, какие из них преобладают, таким является и сам элемент — **мы похожи на свое окружение**.

<sup>3</sup>Источник: [https://www.researchgate.net/publication/267953942\\_Video\\_-based\\_Fall\\_Detection\\_in\\_Elderly's\\_Houses](https://www.researchgate.net/publication/267953942_Video_-based_Fall_Detection_in_Elderly's_Houses)



## Идея

- ☛ *Гипотеза компактности*: если функция расстояния  $\rho$  между элементами введена «удачно», то похожие элементы гораздо чаще лежат в одном классе, чем в разных.
- ☛ Смотрим на соседей элемента, какие из них преобладают, таким является и сам элемент — **мы похожи на свое окружение**.

## Алгоритм

**Данные**: обучающая выборка  $\{(\mathbf{X}_1, y_1), \dots, (\mathbf{X}_m, y_m) \mid \mathbf{X}_i \in \tilde{\mathbf{X}} \subset \mathbf{X}\}$ , функция расстояния  $\rho(\mathbf{X}, \mathbf{Y})$ ,  $\mathbf{X}, \mathbf{Y} \in \mathbf{X}$  и число соседей —  $k$ .

<sup>3</sup>Источник: [https://www.researchgate.net/publication/267953942\\_Video\\_-based\\_Fall\\_Detection\\_in\\_Elderly's\\_Houses](https://www.researchgate.net/publication/267953942_Video_-based_Fall_Detection_in_Elderly's_Houses)

## Идея

- ☛ *Гипотеза компактности*: если функция расстояния  $\rho$  между элементами введена «удачно», то похожие элементы гораздо чаще лежат в одном классе, чем в разных.
- ☛ Смотрим на соседей элемента, какие из них преобладают, таким является и сам элемент — **мы похожи на свое окружение**.

## Алгоритм

**Данные:** обучающая выборка  $\{(\mathbf{X}_1, y_1), \dots, (\mathbf{X}_m, y_m) \mid \mathbf{X}_i \in \tilde{\mathbf{X}} \subset \mathbf{X}\}$ , функция расстояния  $\rho(\mathbf{X}, \mathbf{Y})$ ,  $\mathbf{X}, \mathbf{Y} \in \mathbf{X}$  и число соседей —  $k$ .

- 1 Вычисляем расстояния от  $\mathbf{U} \in \mathbf{X}$  до каждого из объектов обучающей выборки.

<sup>3</sup>Источник: [https://www.researchgate.net/publication/267953942\\_Video\\_-based\\_Fall\\_Detection\\_in\\_Elderly's\\_Houses](https://www.researchgate.net/publication/267953942_Video_-based_Fall_Detection_in_Elderly's_Houses)

# Классификатор к-ближайших соседей (KNN)

## Идея

- ☛ *Гипотеза компактности*: если функция расстояния  $\rho$  между элементами введена «удачно», то похожие элементы гораздо чаще лежат в одном классе, чем в разных.
- ☛ Смотрим на соседей элемента, какие из них преобладают, таким является и сам элемент — **мы похожи на свое окружение**.

## Алгоритм

**Данные:** обучающая выборка  $\{(\mathbf{X}_1, y_1), \dots, (\mathbf{X}_m, y_m) \mid \mathbf{X}_i \in \tilde{\mathbf{X}} \subset \mathbf{X}\}$ , функция расстояния  $\rho(\mathbf{X}, \mathbf{Y})$ ,  $\mathbf{X}, \mathbf{Y} \in \mathbf{X}$  и число соседей —  $k$ .

- 1 Вычисляем расстояния от  $\mathbf{U} \in \mathbf{X}$  до каждого из объектов обучающей выборки.
- 2 Сортируем их в порядке возрастания:

$$\rho(\mathbf{X}_{i_1}, \mathbf{U}) \leq \rho(\mathbf{X}_{i_2}, \mathbf{U}) \leq \dots \leq \rho(\mathbf{X}_{i_m}, \mathbf{U}). \quad (17)$$

<sup>3</sup>Источник: [https://www.researchgate.net/publication/267953942\\_Video\\_-based\\_Fall\\_Detection\\_in\\_Elderly's\\_Houses](https://www.researchgate.net/publication/267953942_Video_-based_Fall_Detection_in_Elderly's_Houses)

# Классификатор к-ближайших соседей (KNN)

## Идея

- ☛ *Гипотеза компактности*: если функция расстояния  $\rho$  между элементами введена «удачно», то похожие элементы гораздо чаще лежат в одном классе, чем в разных.
- ☛ Смотрим на соседей элемента, какие из них преобладают, таким является и сам элемент — **мы похожи на свое окружение**.

## Алгоритм

**Данные**: обучающая выборка  $\{(\mathbf{X}_1, y_1), \dots, (\mathbf{X}_m, y_m) \mid \mathbf{X}_i \in \tilde{\mathbf{X}} \subset \mathbf{X}\}$ , функция расстояния  $\rho(\mathbf{X}, \mathbf{Y})$ ,  $\mathbf{X}, \mathbf{Y} \in \mathbf{X}$  и число соседей —  $k$ .

- 1 Вычисляем расстояния от  $\mathbf{U} \in \mathbf{X}$  до каждого из объектов обучающей выборки.
- 2 Сортируем их в порядке возрастания:

$$\rho(\mathbf{X}_{i_1}, \mathbf{U}) \leq \rho(\mathbf{X}_{i_2}, \mathbf{U}) \leq \dots \leq \rho(\mathbf{X}_{i_m}, \mathbf{U}). \quad (17)$$

- 3 Класс объекта  $\mathbf{U}$  — это класс, наиболее часто встречающийся среди ближайших  $k$  соседей в (17):  $\mathbf{X}_{i_1}, \dots, \mathbf{X}_{i_k}$ .

<sup>3</sup>Источник: [https://www.researchgate.net/publication/267953942\\_Video\\_-based\\_Fall\\_Detection\\_in\\_Elderly's\\_Houses](https://www.researchgate.net/publication/267953942_Video_-based_Fall_Detection_in_Elderly's_Houses)

# Классификатор к-ближайших соседей (KNN)

## Идея

- Гипотеза компактности: если функция расстояния  $\rho$  между элементами введена «удачно», то похожие элементы гораздо чаще лежат в одном классе, чем в разных.
- Смотрим на соседей элемента, какие из них преобладают, таким является и сам элемент — **мы похожи на свое окружение**.

## Алгоритм

**Данные:** обучающая выборка  $\{(X_1, y_1), \dots, (X_m, y_m) \mid X_i \in \tilde{X} \subset X\}$ , функция расстояния  $\rho(X, Y)$ ,  $X, Y \in X$  и число соседей —  $k$ .

- 1 Вычисляем расстояния от  $U \in X$  до каждого из объектов обучающей выборки.
- 2 Сортируем их в порядке возрастания:

$$\rho(X_{i_1}, U) \leq \rho(X_{i_2}, U) \leq \dots \leq \rho(X_{i_m}, U). \quad (17)$$

- 3 Класс объекта  $U$  — это класс, наиболее часто встречающийся среди ближайших  $k$  соседей в (17):  $X_{i_1}, \dots, X_{i_k}$ .

**Пример:** на рисунке 9 видно, что результат классификации зависит от числа ближайших соседей.

- В задачах с двумя классами число соседей обычно берут нечетным, чтобы не возникало ситуаций неоднозначности, когда одинаковое число соседей принадлежат разным классам.

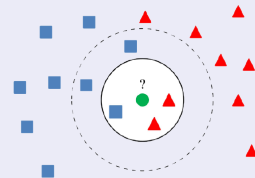


Рис. 9. Иллюстрация работы метода<sup>3</sup>

<sup>3</sup>Источник: [https://www.researchgate.net/publication/267953942\\_Video\\_-based\\_Fall\\_Detection\\_in\\_Elderly's\\_Houses](https://www.researchgate.net/publication/267953942_Video_-based_Fall_Detection_in_Elderly's_Houses)



**i Обучающая выборка.** Опрос футбольных болельщиков ( $X_1, \dots, X_{10}$ ), два признака — возраст и пол (М — 0, Ж — 1), три класса — клубы «Зенит» (0), «Спартак» (1) и «Динамо» 2. Таблица — 1.

Таблица 1

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$
Возраст	32	40	16	34	55	40	20	15	55	15
Пол	0	0	1	1	0	0	1	0	1	0
Клуб	0	2	0	0	2	0	2	0	1	1

**i Обучающая выборка.** Опрос футбольных болельщиков ( $X_1, \dots, X_{10}$ ), два признака — возраст и пол (М — 0, Ж — 1), три класса — клубы «Зенит» (0), «Спартак» (1) и «Динамо» 2. Таблица — 1.

Таблица 1

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$
Возраст	32	40	16	34	55	40	20	15	55	15
Пол	0	0	1	1	0	0	1	0	1	0
Клуб	0	2	0	0	2	0	2	0	1	1

**i Параметры алгоритма.** Метрика расстояния —  $L_2 : \rho_2(X_i(x_i, y_i), X_j(x_j, y_j)) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ , число соседей —  $k = 3$ .



**Обучающая выборка.** Опрос футбольных болельщиков ( $X_1, \dots, X_{10}$ ), два признака — возраст и пол (М — 0, Ж — 1), три класса — клубы «Зенит» (0), «Спартак» (1) и «Динамо» 2. Таблица — 1.

Таблица 1

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$
Возраст	32	40	16	34	55	40	20	15	55	15
Пол	0	0	1	1	0	0	1	0	1	0
Клуб	0	2	0	0	2	0	2	0	1	1

- Параметры алгоритма.** Метрика расстояния —  $L_2 : \rho_2(X_i(x_i, y_i), X_j(x_j, y_j)) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ , число соседей —  $k = 3$ .
- Определить класс объекта:**  $U = \{5, 1\}$  — *за кого болеет пятилетняя девочка?*

**i Обучающая выборка.** Опрос футбольных болельщиков ( $X_1, \dots, X_{10}$ ), два признака — возраст и пол (М — 0, Ж — 1), три класса — клубы «Зенит» (0), «Спартак» (1) и «Динамо» 2. Таблица — 1.

Таблица 1

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$
Возраст	32	40	16	34	55	40	20	15	55	15
Пол	0	0	1	1	0	0	1	0	1	0
Клуб	0	2	0	0	2	0	2	0	1	1

**i Параметры алгоритма.** Метрика расстояния —  $L_2 : \rho_2(X_i(x_i, y_i), X_j(x_j, y_j)) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ , число соседей —  $k = 3$ .

**i Определить класс объекта:**  $U = \{5, 1\}$  — *за кого болеет пятилетняя девочка?*

**1** Вычисляем расстояния от  $U$  до каждого из объектов обучающей выборки — таблица 2.

Таблица 2

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$
$\rho_2(U, X_i)$	27.02	35.01	11.00	9.00	50.01	35.01	15.00	10.00	50.00	10.05

**Обучающая выборка.** Опрос футбольных болельщиков ( $X_1, \dots, X_{10}$ ), два признака — возраст и пол (М — 0, Ж — 1), три класса — клубы «Зенит» (0), «Спартак» (1) и «Динамо» 2. Таблица — 1.

Таблица 1

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$
Возраст	32	40	16	34	55	40	20	15	55	15
Пол	0	0	1	1	0	0	1	0	1	0
Клуб	0	2	0	0	2	0	2	0	1	1

**Параметры алгоритма.** Метрика расстояния —  $L_2 : \rho_2(X_i(x_i, y_i), X_j(x_j, y_j)) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ , число соседей —  $k = 3$ .

**Определить класс объекта:**  $U = \{5, 1\}$  — *за кого болеет пятилетняя девочка?*

**1** Вычисляем расстояния от  $U$  до каждого из объектов обучающей выборки — таблица 2.

Таблица 2

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$
$\rho_2(U, X_i)$	27.02	35.01	11.00	9.00	50.01	35.01	15.00	10.00	50.00	10.05

**2** Сортируем их в порядке возрастания и рассматриваем первые  $k$ :

$$\rho_2(U, X_4) < \rho_2(U, X_8) < \rho_2(U, X_{10}) < \dots \tag{18}$$

**i Обучающая выборка.** Опрос футбольных болельщиков ( $X_1, \dots, X_{10}$ ), два признака — возраст и пол (М — 0, Ж — 1), три класса — клубы «Зенит» (0), «Спартак» (1) и «Динамо» 2. Таблица — 1.

Таблица 1

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$
Возраст	32	40	16	34	55	40	20	15	55	15
Пол	0	0	1	1	0	0	1	0	1	0
Клуб	0	2	0	0	2	0	2	0	1	1

**i Параметры алгоритма.** Метрика расстояния —  $L_2 : \rho_2(X_i(x_i, y_i), X_j(x_j, y_j)) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ , число соседей —  $k = 3$ .

**i Определить класс объекта:**  $U = \{5, 1\}$  — *за кого болеет пятилетняя девочка?*

**1** Вычисляем расстояния от  $U$  до каждого из объектов обучающей выборки — таблица 2.

Таблица 2

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$
$\rho_2(U, X_i)$	27.02	35.01	11.00	9.00	50.01	35.01	15.00	10.00	50.00	10.05

**2** Сортируем их в порядке возрастания и рассматриваем первые  $k$ :

$$\rho_2(U, X_4) < \rho_2(U, X_8) < \rho_2(U, X_{10}) < \dots \tag{18}$$

**i Ответ:** класс 0, болеет за «Зенит».



```
# Connecting the necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import datasets
from sklearn.model_selection import train_test_split

# Load IRIS dataset
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

# Feature Scaling using StandardScaler
sc = StandardScaler()
sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model
knn = KNeighborsClassifier(n_neighbors=5, p=2, weights='uniform',
                          algorithm='auto').fit(X_tr_std, y_tr)
predicted = knn.predict(X_t_std)
```

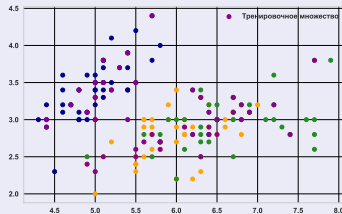
```
# Connecting the necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import datasets
from sklearn.model_selection import train_test_split

# Load IRIS dataset
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

# Feature Scaling using StandardScaler
sc = StandardScaler()
sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model
knn = KNeighborsClassifier(n_neighbors=5, p=2, weights='uniform',
                          algorithm='auto').fit(X_tr_std, y_tr)
predicted = knn.predict(X_t_std)
```



(а) Исходный набор (Ирисы)

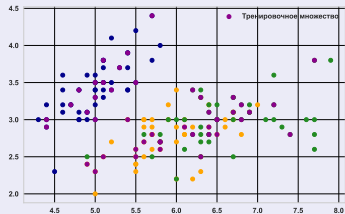
```
# Connecting the necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import datasets
from sklearn.model_selection import train_test_split

# Load IRIS dataset
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

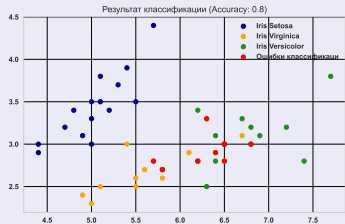
# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                         random_state=42, stratify=y)

# Feature Scaling using StandardScaler
sc = StandardScaler()
sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model
knn = KNeighborsClassifier(n_neighbors=5, p=2, weights='uniform',
                          algorithm='auto').fit(X_tr_std, y_tr)
predicted = knn.predict(X_t_std)
```




(a) Исходный набор (Ирисы)



(b) Результаты классификации





 Возможна ситуация, когда частота появления для двух или более классов окажется одинаковой. Как выбирать решение в этой ситуации?

- i Возможна ситуация, когда частота появления для двух или более классов окажется одинаковой. Как выбирать решение в этой ситуации?
- 1 Разумно предположить, что не все обучающие элементы имеют одинаковую значимость для определения класса объекта  $U$ .

- i Возможна ситуация, когда частота появления для двух или более классов окажется одинаковой. Как выбирать решение в этой ситуации?
- 1 Разумно предположить, что не все обучающие элементы имеют одинаковую значимость для определения класса объекта  $U$ .
- 2 Будем считать, что чем дальше пример расположен от классифицируемого объекта в пространстве признаков, тем ниже его значимость для определения класса.

- i Возможна ситуация, когда частота появления для двух или более классов окажется одинаковой. Как выбирать решение в этой ситуации?
- 1 Разумно предположить, что не все обучающие элементы имеют одинаковую значимость для определения класса объекта  $U$ .
- 2 Будем считать, что чем дальше пример расположен от классифицируемого объекта в пространстве признаков, тем ниже его значимость для определения класса.
- 3 Введем **взвешивание (штраф)** элементов в зависимости от их близости к объекту  $U$ .

- i Возможна ситуация, когда частота появления для двух или более классов окажется одинаковой. Как выбирать решение в этой ситуации?
- 1 Разумно предположить, что не все обучающие элементы имеют одинаковую значимость для определения класса объекта  $U$ .
- 2 Будем считать, что чем дальше пример расположен от классифицируемого объекта в пространстве признаков, тем ниже его значимость для определения класса.
- 3 Введем **взвешивание (штраф)** элементов в зависимости от их близости к объекту  $U$ .
- 4 В качестве штрафа возьмем сумму величин расстояний от примеров  $j$ -го класса до классифицируемого объекта  $U$  (иногда это значение называют *показателем близости*):

$$Q_j = \sum_{i=1}^{n_j} \rho(\mathbf{X}_{ij} U), \quad (19)$$

где  $n_j$  — число элементов класса  $j$ , попавших в  $k$  ближайших соседей,  $\mathbf{X}_{i,j}$  —  $i$ -й элемент класса  $j$ .

- ❶ Возможна ситуация, когда частота появления для двух или более классов окажется одинаковой. Как выбирать решение в этой ситуации?
- ❶ Разумно предположить, что не все обучающие элементы имеют одинаковую значимость для определения класса объекта  $U$ .
- ❷ Будем считать, что чем дальше пример расположен от классифицируемого объекта в пространстве признаков, тем ниже его значимость для определения класса.
- ❸ Введем **взвешивание (штраф)** элементов в зависимости от их близости к объекту  $U$ .
- ❹ В качестве штрафа возьмем сумму величин расстояний от примеров  $j$ -го класса до классифицируемого объекта  $U$  (иногда это значение называют *показателем близости*):

$$Q_j = \sum_{i=1}^{n_j} \rho(\mathbf{X}_{ij}, U), \quad (19)$$

где  $n_j$  — число элементов класса  $j$ , попавших в  $k$  ближайших соседей,  $\mathbf{X}_{i,j}$  —  $i$ -й элемент класса  $j$ .

- ❺ **Побеждает** тот класс, для которого величина штрафа  $Q_j$  окажется наименьшей

- i Возможна ситуация, когда частота появления для двух или более классов окажется одинаковой. Как выбирать решение в этой ситуации?
- 1 Разумно предположить, что не все обучающие элементы имеют одинаковую значимость для определения класса объекта  $\mathbf{U}$ .
- 2 Будем считать, что чем дальше пример расположен от классифицируемого объекта в пространстве признаков, тем ниже его значимость для определения класса.
- 3 Введем **взвешивание (штраф)** элементов в зависимости от их близости к объекту  $\mathbf{U}$ .
- 4 В качестве штрафа возьмем сумму величин расстояний от примеров  $j$ -го класса до классифицируемого объекта  $\mathbf{U}$  (иногда это значение называют *показателем близости*):

$$Q_j = \sum_{i=1}^{n_j} \rho(\mathbf{X}_{i,j} \mathbf{U}), \quad (19)$$

где  $n_j$  — число элементов класса  $j$ , попавших в  $k$  ближайших соседей,  $\mathbf{X}_{i,j}$  —  $i$ -й элемент класса  $j$ .

- 5 **Побеждает** тот класс, для которого величина штрафа  $Q_j$  окажется наименьшей
- 6 Иногда, в качестве штрафа рассматривают величину обратную  $Q_j$  в (19) (или ее квадрату). В этом случае побеждает класс с максимальным штрафом.



1. Возможна ситуация, когда частота появления для двух или более классов окажется одинаковой. Как выбирать решение в этой ситуации?
1. Разумно предположить, что не все обучающие элементы имеют одинаковую значимость для определения класса объекта  $U$ .
2. Будем считать, что чем дальше пример расположен от классифицируемого объекта в пространстве признаков, тем ниже его значимость для определения класса.
3. Введем **взвешивание (штраф)** элементов в зависимости от их близости к объекту  $U$ .
4. В качестве штрафа возьмем сумму величин расстояний от примеров  $j$ -го класса до классифицируемого объекта  $U$  (иногда это значение называют *показателем близости*):

$$Q_j = \sum_{i=1}^{n_j} \rho(\mathbf{X}_{i,j} U), \quad (19)$$

где  $n_j$  — число элементов класса  $j$ , попавших в  $k$  ближайших соседей,  $\mathbf{X}_{i,j}$  —  $i$ -й элемент класса  $j$ .

5. **Побеждает** тот класс, для которого величина штрафа  $Q_j$  окажется наименьшей
6. Иногда, в качестве штрафа рассматривают величину обратную  $Q_j$  в (19) (или ее квадрату). В этом случае побеждает класс с максимальным штрафом.
7. Вероятность того, что классы получают одинаковое число голосов снижается.

- ❶ Возможна ситуация, когда частота появления для двух или более классов окажется одинаковой. Как выбирать решение в этой ситуации?
- ❶ Разумно предположить, что не все обучающие элементы имеют одинаковую значимость для определения класса объекта  $U$ .
- ❷ Будем считать, что чем дальше пример расположен от классифицируемого объекта в пространстве признаков, тем ниже его значимость для определения класса.
- ❸ Введем **взвешивание (штраф)** элементов в зависимости от их близости к объекту  $U$ .
- ❹ В качестве штрафа возьмем сумму величин расстояний от примеров  $j$ -го класса до классифицируемого объекта  $U$  (иногда это значение называют *показателем близости*):

$$Q_j = \sum_{i=1}^{n_j} \rho(\mathbf{X}_{ij}U), \quad (19)$$

где  $n_j$  — число элементов класса  $j$ , попавших в  $k$  ближайших соседей,  $\mathbf{X}_{i,j}$  —  $i$ -й элемент класса  $j$ .

- ❺ **Побеждает** тот класс, для которого величина штрафа  $Q_j$  окажется наименьшей
- ❻ Иногда, в качестве штрафа рассматривают величину обратную  $Q_j$  в (19) (или ее квадрату). В этом случае побеждает класс с максимальным штрафом.
- ❼ Вероятность того, что классы получают одинаковое число голосов снижается.
- ❶ Такая модификация алгоритма называется *методом  $k$  взвешенных ближайших соседей*.

- ❗ Возможна ситуация, когда частота появления для двух или более классов окажется одинаковой. Как выбирать решение в этой ситуации?
- 1 Разумно предположить, что не все обучающие элементы имеют одинаковую значимость для определения класса объекта  $U$ .
- 2 Будем считать, что чем дальше пример расположен от классифицируемого объекта в пространстве признаков, тем ниже его значимость для определения класса.
- 3 Введем **взвешивание (штраф)** элементов в зависимости от их близости к объекту  $U$ .
- 4 В качестве штрафа возьмем сумму величин расстояний от примеров  $j$ -го класса до классифицируемого объекта  $U$  (иногда это значение называют *показателем близости*):

$$Q_j = \sum_{i=1}^{n_j} \rho(\mathbf{X}_{ij}U), \quad (19)$$

где  $n_j$  — число элементов класса  $j$ , попавших в  $k$  ближайших соседей,  $\mathbf{X}_{i,j}$  —  $i$ -й элемент класса  $j$ .

- 5 **Побеждает** тот класс, для которого величина штрафа  $Q_j$  окажется наименьшей
- 6 Иногда, в качестве штрафа рассматривают величину обратную  $Q_j$  в (19) (или ее квадрату). В этом случае побеждает класс с максимальным штрафом.
- 7 Вероятность того, что классы получают одинаковое число голосов снижается.
- ❗ Такая модификация алгоритма называется *методом  $k$  взвешенных ближайших соседей*.

- ❗ Для работы со взвешенным KNN нужно запустить классификатор `KNeighborsClassifier` с параметром `weights = 'distance'`.

- ❶ Возможна ситуация, когда частота появления для двух или более классов окажется одинаковой. Как выбирать решение в этой ситуации?
- ❶ Разумно предположить, что не все обучающие элементы имеют одинаковую значимость для определения класса объекта  $U$ .
- ❷ Будем считать, что чем дальше пример расположен от классифицируемого объекта в пространстве признаков, тем ниже его значимость для определения класса.
- ❸ Введем **взвешивание (штраф)** элементов в зависимости от их близости к объекту  $U$ .
- ❹ В качестве штрафа возьмем сумму величин расстояний от примеров  $j$ -го класса до классифицируемого объекта  $U$  (иногда это значение называют *показателем близости*):

$$Q_j = \sum_{i=1}^{n_j} \rho(\mathbf{X}_{ij}U), \quad (19)$$

где  $n_j$  — число элементов класса  $j$ , попавших в  $k$  ближайших соседей,  $\mathbf{X}_{i,j}$  —  $i$ -й элемент класса  $j$ .

- ❺ **Побеждает** тот класс, для которого величина штрафа  $Q_j$  окажется наименьшей
- ❻ Иногда, в качестве штрафа рассматривают величину обратную  $Q_j$  в (19) (или ее квадрату). В этом случае побеждает класс с максимальным штрафом.
- ❼ Вероятность того, что классы получают одинаковое число голосов снижается.
- ❶ Такая модификация алгоритма называется *методом  $k$  взвешенных ближайших соседей*.

- ❶ Для работы со взвешенным KNN нужно запустить классификатор *KNeighborsClassifier* с параметром *weights = 'distance'*.
- ❶ По умолчанию *weights = 'uniform'* — все точки в  $k$ -окрестности имеют одинаковый вес. С этим значением был пример предыдущего слайда.



## Достоинства

- 1 Простота алгоритма и реализации — алгоритм требует только значения  $k$  и метрики расстояния, поэтому его разработка очень быстрая.

## Достоинства

- 1 Простота алгоритма и реализации — алгоритм требует только значения  $k$  и метрики расстояния, поэтому его разработка очень быстрая.
- 2 Не чувствителен к выбросам.

## Достоинства

- 1 Простота алгоритма и реализации — алгоритм требует только значения  $k$  и метрики расстояния, поэтому его разработка очень быстрая.
- 2 Не чувствителен к выбросам.
- 3 Легко адаптируется: по мере добавления новых обучающих выборок алгоритм корректируется с учетом любых новых данных, поскольку все обучающие данные сохраняются в памяти.



## Достоинства

- 1 Простота алгоритма и реализации — алгоритм требует только значения  $k$  и метрики расстояния, поэтому его разработка очень быстрая.
- 2 Не чувствителен к выбросам.
- 3 Легко адаптируется: по мере добавления новых обучающих выборок алгоритм корректируется с учетом любых новых данных, поскольку все обучающие данные сохраняются в памяти.

## Недостатки

- 1 Алгоритм называют *ленивым алгоритмом*, поскольку в нем нет фазы обучения, модель не строится и вычисления на данных обучения не выполняются. Алгоритм сохраняет только все данные, которые он получает на этапе обучения. Поэтому во время прогнозирования модель ищет ближайших соседей из всего обучающего набора, что делает его дорогостоящим.

## Достоинства

- 1 Простота алгоритма и реализации — алгоритм требует только значения  $k$  и метрики расстояния, поэтому его разработка очень быстрая.
- 2 Не чувствителен к выбросам.
- 3 Легко адаптируется: по мере добавления новых обучающих выборок алгоритм корректируется с учетом любых новых данных, поскольку все обучающие данные сохраняются в памяти.

## Недостатки

- 1 Алгоритм называют *ленивым алгоритмом*, поскольку в нем нет фазы обучения, модель не строится и вычисления на данных обучения не выполняются. Алгоритм сохраняет только все данные, которые он получает на этапе обучения. Поэтому во время прогнозирования модель ищет ближайших соседей из всего обучающего набора, что делает его дорогостоящим.
- 2 Можно использовать приближенные алгоритмы — «Приближенный ближайший сосед» (Approximate Nearest Neighbor, ANN), которые ускоряют поиск ближайшего соседа за счет предварительной обработки/индексирования данных.

## Достоинства

- 1 Простота алгоритма и реализации — алгоритм требует только значения  $k$  и метрики расстояния, поэтому его разработка очень быстрая.
- 2 Не чувствителен к выбросам.
- 3 Легко адаптируется: по мере добавления новых обучающих выборок алгоритм корректируется с учетом любых новых данных, поскольку все обучающие данные сохраняются в памяти.

## Недостатки

- 1 Алгоритм называют *ленивым алгоритмом*, поскольку в нем нет фазы обучения, модель не строится и вычисления на данных обучения не выполняются. Алгоритм сохраняет только все данные, которые он получает на этапе обучения. Поэтому во время прогнозирования модель ищет ближайших соседей из всего обучающего набора, что делает его дорогостоящим.
- 2 Можно использовать приближенные алгоритмы — «Приближенный ближайший сосед» (Approximate Nearest Neighbor, ANN), которые ускоряют поиск ближайшего соседа за счет предварительной обработки/индексирования данных.
- 3 Нужно определять оптимальное значение числа ближайших соседей —  $k$ .

## Достоинства

- 1 Простота алгоритма и реализации — алгоритм требует только значения  $k$  и метрики расстояния, поэтому его разработка очень быстрая.
- 2 Не чувствителен к выбросам.
- 3 Легко адаптируется: по мере добавления новых обучающих выборок алгоритм корректируется с учетом любых новых данных, поскольку все обучающие данные сохраняются в памяти.

## Недостатки

- 1 Алгоритм называют *ленивым алгоритмом*, поскольку в нем нет фазы обучения, модель не строится и вычисления на данных обучения не выполняются. Алгоритм сохраняет только все данные, которые он получает на этапе обучения. Поэтому во время прогнозирования модель ищет ближайших соседей из всего обучающего набора, что делает его дорогостоящим.
- 2 Можно использовать приближенные алгоритмы — «Приближенный ближайший сосед» (Approximate Nearest Neighbor, ANN), которые ускоряют поиск ближайшего соседа за счет предварительной обработки/индексирования данных.
- 3 Нужно определять оптимальное значение числа ближайших соседей —  $k$ .
- 4 Большой объем памяти — необходимо хранить обучающую выборку целиком.

## Достоинства

- 1 Простота алгоритма и реализации — алгоритм требует только значения  $k$  и метрики расстояния, поэтому его разработка очень быстрая.
- 2 Не чувствителен к выбросам.
- 3 Легко адаптируется: по мере добавления новых обучающих выборок алгоритм корректируется с учетом любых новых данных, поскольку все обучающие данные сохраняются в памяти.

## Недостатки

- 1 Алгоритм называют *ленивым алгоритмом*, поскольку в нем нет фазы обучения, модель не строится и вычисления на данных обучения не выполняются. Алгоритм сохраняет только все данные, которые он получает на этапе обучения. Поэтому во время прогнозирования модель ищет ближайших соседей из всего обучающего набора, что делает его дорогостоящим.
- 2 Можно использовать приближенные алгоритмы — «Приближенный ближайший сосед» (Approximate Nearest Neighbor, ANN), которые ускоряют поиск ближайшего соседа за счет предварительной обработки/индексирования данных.
- 3 Нужно определять оптимальное значение числа ближайших соседей —  $k$ .
- 4 Большой объем памяти — необходимо хранить обучающую выборку целиком.
- 5 *Проклятие размерности*. Алгоритм имеет тенденцию становиться жертвой проклятия размерности (*феномен пика* или *феномен Хьюза*): он не очень хорошо работает с входными данными большой размерности. Этот термин означает, что при фиксированном количестве обучающих элементов средняя прогностическая способность классификатора сначала увеличивается по мере увеличения числа используемых признаков, но после достижения определенной размерности она начинает ухудшаться, а не постоянно улучшаться.

# KNN. Достоинства и недостатки

## Достоинства

- 1 Простота алгоритма и реализации — алгоритм требует только значения  $k$  и метрики расстояния, поэтому его разработка очень быстрая.
- 2 Не чувствителен к выбросам.
- 3 Легко адаптируется: по мере добавления новых обучающих выборок алгоритм корректируется с учетом любых новых данных, поскольку все обучающие данные сохраняются в памяти.

## Недостатки

- 1 Алгоритм называют *ленивым алгоритмом*, поскольку в нем нет фазы обучения, модель не строится и вычисления на данных обучения не выполняются. Алгоритм сохраняет только все данные, которые он получает на этапе обучения. Поэтому во время прогнозирования модель ищет ближайших соседей из всего обучающего набора, что делает его дорогостоящим.
- 2 Можно использовать приближенные алгоритмы — «Приближенный ближайший сосед» (Approximate Nearest Neighbor, ANN), которые ускоряют поиск ближайшего соседа за счет предварительной обработки/индексирования данных.
- 3 Нужно определять оптимальное значение числа ближайших соседей —  $k$ .
- 4 Большой объем памяти — необходимо хранить обучающую выборку целиком.
- 5 *Проклятие размерности*. Алгоритм имеет тенденцию становиться жертвой проклятия размерности (*феномен пика* или *феномен Хьюза*): он не очень хорошо работает с входными данными большой размерности. Этот термин означает, что при фиксированном количестве обучающих элементов средняя прогностическая способность классификатора сначала увеличивается по мере увеличения числа используемых признаков, но после достижения определенной размерности она начинает ухудшаться, а не постоянно улучшаться.
- 6 Высокая зависимость результатов классификации от выбранной метрики  $\rho(\mathbf{X}, \mathbf{Y})$ . Если признаки имеют разные масштабы измерения или имеется смесь числовых и категориальных, необходима стандартизация.



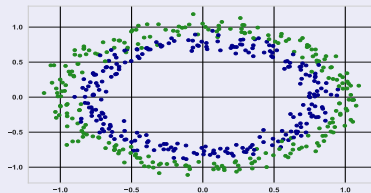
- 1 Параметр  $k$  в классификаторе *KNeighborsClassifier* по умолчанию равен 5.



- 1 Параметр  $k$  в классификаторе *KNeighborsClassifier* по умолчанию равен 5.
- 2 Иногда значение  $k$  устанавливают с помощью равенства:  $k = \sqrt{n}$ , где  $n$  — число элементов в обучающем наборе.

- 1 Параметр  $k$  в классификаторе `KNeighborsClassifier` по умолчанию равен 5.
- 2 Иногда значение  $k$  устанавливают с помощью равенства:  $k = \sqrt{n}$ , где  $n$  — число элементов в обучающем наборе.
- 3 Хороший вариант — использовать *метод локтя*, вычисляемый по ошибкам классификатора для различных  $k$ .

- 1 Параметр  $k$  в классификаторе `KNeighborsClassifier` по умолчанию равен 5.
- 2 Иногда значение  $k$  устанавливают с помощью равенства:  $k = \sqrt{n}$ , где  $n$  — число элементов в обучающем наборе.
- 3 Хороший вариант — использовать *метод локтя*, вычисляемый по ошибкам классификатора для различных  $k$ .
- 4 Продемонстрируем процесс на линейно неразделимом множестве (пример для алгоритма SVM) — рисунок 11:а.



(а)

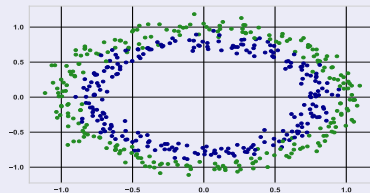
Рис. 11

# KNN. Выбор гиперпараметра $k$

- 1 Параметр  $k$  в классификаторе `KNeighborsClassifier` по умолчанию равен 5.
- 2 Иногда значение  $k$  устанавливают с помощью равенства:  $k = \sqrt{n}$ , где  $n$  — число элементов в обучающем наборе.
- 3 Хороший вариант — использовать *метод локтя*, вычисляемый по ошибкам классификатора для различных  $k$ .
- 4 Продемонстрируем процесс на линейно неразделимом множестве (пример для алгоритма SVM) —рисунок 11:а.
- 5 Генерируем множество, разделяем его на тестовую и тренировочную части, стандартизируем.

```
X, y = make_circles(n_samples=500, noise=0.06, random_state=42)
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

sc = StandardScaler()
sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)
```



(a)



(b)

Рис. 11

# KNN. Выбор гиперпараметра $k$

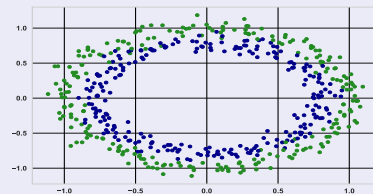
- 1 Параметр  $k$  в классификаторе `KNeighborsClassifier` по умолчанию равен 5.
- 2 Иногда значение  $k$  устанавливают с помощью равенства:  $k = \sqrt{n}$ , где  $n$  — число элементов в обучающем наборе.
- 3 Хороший вариант — использовать *метод локтя*, вычисляемый по ошибкам классификатора для различных  $k$ .
- 4 Продемонстрируем процесс на линейно неразделимом множестве (пример для алгоритма SVM) —рисунок 11:a.
- 5 Генерируем множество, разделяем его на тестовую и тренировочную части, стандартизируем.

```
X, y = make_circles(n_samples=500, noise=0.06, random_state=42)
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)
```

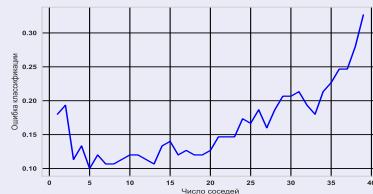
```
sc = StandardScaler()
sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)
```

- 6 Для  $k \in 1 : 40$  проверяем результаты работы модели: график ошибок классификации в зависимости от  $k$  на рисунке 11:b. Можно взять  $k \in 5 : 13$ .

```
error_rate = []
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_tr_std, y_tr)
    pred_i = knn.predict(X_t_std)
    error_rate.append(np.mean(pred_i != y_t))
```



(a)



(b)

Рис. 11



**i Гиперпараметр** — это характеристики модели, значение которой не определяется на основе данных. Например, метрика расстояния, число соседей для KNN или тип ядра для SVM. Оно должно быть установлено до начала обучения модели.

- Гиперпараметр** — это характеристики модели, значение которой не определяется на основе данных. Например, метрика расстояния, число соседей для KNN или тип ядра для SVM. Оно должно быть установлено до начала обучения модели.
- Как организовать перебор наборов гиперпараметров? Самый естественный — сделать перебор по сетке (**Grid SearchCV**).










- Гиперпараметр** — это характеристики модели, значение которой не определяется на основе данных. Например, метрика расстояния, число соседей для KNN или тип ядра для SVM. Оно должно быть установлено до начала обучения модели.
- Как организовать перебор наборов гиперпараметров? Самый естественный — сделать перебор по сетке (**Grid SearchCV**).
- GridSearchCV находит наилучшие параметры, путем обычного перебора: он создает модель для каждой возможной комбинации параметров.

- i** **Гиперпараметр** — это характеристики модели, значение которой не определяется на основе данных. Например, метрика расстояния, число соседей для KNN или тип ядра для SVM. Оно должно быть установлено до начала обучения модели.
- i** Как организовать перебор наборов гиперпараметров? Самый естественный — сделать перебор по сетке (**Grid SearchCV**).
- i** GridSearchCV находит наилучшие параметры, путем обычного перебора: он создает модель для каждой возможной комбинации параметров.
  - 1** Для каждого гиперпараметра фиксируется несколько значений.

- i** **Гиперпараметр** — это характеристики модели, значение которой не определяется на основе данных. Например, метрика расстояния, число соседей для KNN или тип ядра для SVM. Оно должно быть установлено до начала обучения модели.
- i** Как организовать перебор наборов гиперпараметров? Самый естественный — сделать перебор по сетке (**Grid SearchCV**).
- i** GridSearchCV находит наилучшие параметры, путем обычного перебора: он создает модель для каждой возможной комбинации параметров.
  - 1** Для каждого гиперпараметра фиксируется несколько значений.
  - 2** Перебираются все комбинации значений различных гиперпараметров, на каждой из этих комбинаций модель обучается и тестируется.

- i** **Гиперпараметр** — это характеристики модели, значение которой не определяется на основе данных. Например, метрика расстояния, число соседей для KNN или тип ядра для SVM. Оно должно быть установлено до начала обучения модели.
- i** Как организовать перебор наборов гиперпараметров? Самый естественный — сделать перебор по сетке (**Grid SearchCV**).
- i** GridSearchCV находит наилучшие параметры, путем обычного перебора: он создает модель для каждой возможной комбинации параметров.
  - 1** Для каждого гиперпараметра фиксируется несколько значений.
  - 2** Перебираются все комбинации значений различных гиперпараметров, на каждой из этих комбинаций модель обучается и тестируется.
  - 3** Выбирается комбинация, на которой модель показывает лучшее качество.

-  **Гиперпараметр** — это характеристики модели, значение которой не определяется на основе данных. Например, метрика расстояния, число соседей для KNN или тип ядра для SVM. Оно должно быть установлено до начала обучения модели.
-  Как организовать перебор наборов гиперпараметров? Самый естественный — сделать перебор по сетке (**Grid SearchCV**).
-  GridSearchCV находит наилучшие параметры, путем обычного перебора: он создает модель для каждой возможной комбинации параметров.
  -  Для каждого гиперпараметра фиксируется несколько значений.
  -  Перебираются все комбинации значений различных гиперпараметров, на каждой из этих комбинаций модель обучается и тестируется.
  -  Выбирается комбинация, на которой модель показывает лучшее качество.
-  Такой подход может быть весьма по ресурсно затратным.

- i** **Гиперпараметр** — это характеристики модели, значение которой не определяется на основе данных. Например, метрика расстояния, число соседей для KNN или тип ядра для SVM. Оно должно быть установлено до начала обучения модели.
- i** Как организовать перебор наборов гиперпараметров? Самый естественный — сделать перебор по сетке (**Grid SearchCV**).
- i** GridSearchCV находит наилучшие параметры, путем обычного перебора: он создает модель для каждой возможной комбинации параметров.
  - 1** Для каждого гиперпараметра фиксируется несколько значений.
  - 2** Перебираются все комбинации значений различных гиперпараметров, на каждой из этих комбинаций модель обучается и тестируется.
  - 3** Выбирается комбинация, на которой модель показывает лучшее качество.
- i** Такой подход может быть весьма по ресурсно затратным.
- A** Проведем перебор гиперпараметров для алгоритма KNN на линейно неразделимом множестве предыдущего слайда — рисунок 11:а.

**i** **Гиперпараметр** — это характеристики модели, значение которой не определяется на основе данных. Например, метрика расстояния, число соседей для KNN или тип ядра для SVM. Оно должно быть установлено до начала обучения модели.

**i** Как организовать перебор наборов гиперпараметров? Самый естественный — сделать перебор по сетке (**Grid SearchCV**).

**i** GridSearchCV находит наилучшие параметры, путем обычного перебора: он создает модель для каждой возможной комбинации параметров.

- 1 Для каждого гиперпараметра фиксируется несколько значений.
- 2 Перебираются все комбинации значений различных гиперпараметров, на каждой из этих комбинаций модель обучается и тестируется.
- 3 Выбирается комбинация, на которой модель показывает лучшее качество.

**i** Такой подход может быть весьма по ресурсно затратным.

**A** Проведем перебор гиперпараметров для алгоритма KNN на линейно неразделимом множестве предыдущего слайда — рисунок 11:a.

**B** Множество разделено на тестовую и тренировочную части и стандартизировано. Настраиваем Grid SearchCV:

```
from sklearn.model_selection import GridSearchCV
grid_params = { 'n_neighbors' : [3,4,5,6,7,8,9,9,10,11,12,13,14,15,16,17,18,19,20],
                'weights' : ['uniform','distance'],
                'metric' : ['minkowski','euclidean','manhattan']}
knn = KNeighborsClassifier()
gs = GridSearchCV(knn, grid_params, scoring='accuracy', refit=True)
```

**i** **Гиперпараметр** — это характеристики модели, значение которой не определяется на основе данных. Например, метрика расстояния, число соседей для KNN или тип ядра для SVM. Оно должно быть установлено до начала обучения модели.

**i** Как организовать перебор наборов гиперпараметров? Самый естественный — сделать перебор по сетке (**Grid SearchCV**).

**i** GridSearchCV находит наилучшие параметры, путем обычного перебора: он создает модель для каждой возможной комбинации параметров.

- 1 Для каждого гиперпараметра фиксируется несколько значений.
- 2 Перебираются все комбинации значений различных гиперпараметров, на каждой из этих комбинаций модель обучается и тестируется.
- 3 Выбирается комбинация, на которой модель показывает лучшее качество.

**i** Такой подход может быть весьма по ресурсно затратным.

**A** Проведем перебор гиперпараметров для алгоритма KNN на линейно неразделимом множестве предыдущего слайда — рисунок 11:a.

**B** Множество разделено на тестовую и тренировочную части и стандартизировано. Настраиваем Grid SearchCV:

```
from sklearn.model_selection import GridSearchCV
grid_params = { 'n_neighbors' : [3,4,5,6,7,8,9,9,10,11,12,13,14,15,16,17,18,19,20],
                'weights' : ['uniform','distance'],
                'metric' : ['minkowski','euclidean','manhattan']}
knn = KNeighborsClassifier()
gs = GridSearchCV(knn, grid_params, scoring='accuracy', refit=True)
```

**C** Перебираем параметры и визуализируем лучший результат:

```
g_res = gs.fit(X_tr_std, y_tr)
print(g_res.best_params_)
{'metric': 'minkowski', 'n_neighbors': 4, 'weights': 'distance'}
```





□ *Дерево решений* — метод представления решающих правил в ориентированном дереве.

- ❑ *Дерево решений* — метод представления решающих правил в ориентированном дереве.
- ❑ *Ориентированное дерево* —ориентированный граф, не содержащий циклов.

- ❑ *Дерево решений* — метод представления решающих правил в ориентированном дереве.
- ❑ *Ориентированное дерево* —ориентированный граф, не содержащий циклов.
- ❑ Вершина, не имеющая входящих дуг — *корень дерева*.

- ☐ *Дерево решений* — метод представления решающих правил в ориентированном дереве.
- ☐ *Ориентированное дерево* —ориентированный граф, не содержащий циклов.
- ☐ Вершина, не имеющая входящих дуг — *корень дерева*.
- ☐ Во все остальные вершины входит ровно одна дуга.

- ❑ *Дерево решений* — метод представления решающих правил в ориентированном дереве.
- ❑ *Ориентированное дерево* —ориентированный граф, не содержащий циклов.
- ❑ Вершина, не имеющая входящих дуг — *корень дерева*.
- ❑ Во все остальные вершины входит ровно одна дуга.
- ❑ Вершины делим на два типа — *листья* и *узлы*.

- ☐ *Дерево решений* — метод представления решающих правил в ориентированном дереве.
- ☐ *Ориентированное дерево* —ориентированный граф, не содержащий циклов.
- ☐ Вершина, не имеющая входящих дуг — *корень дерева*.
- ☐ Во все остальные вершины входит ровно одна дуга.
- ☐ Вершины делим на два типа — *листья* и *узлы*.
- ☐ Вершины не имеющие исходящих дуг — *листья*, остальные вершины — *узлы* (включая корень дерева).

- ☐ *Дерево решений* — метод представления решающих правил в ориентированном дереве.
- ☐ *Ориентированное дерево* — ориентированный граф, не содержащий циклов.
- ☐ Вершина, не имеющая входящих дуг — *корень дерева*.
- ☐ Во все остальные вершины входит ровно одна дуга.
- ☐ Вершины делим на два типа — *листья* и *узлы*.
- ☐ Вершины не имеющие исходящих дуг — *листья*, остальные вершины — *узлы* (включая корень дерева).
- ☒ В классическом определении ориентированного графа все его вершины называются узлами. Здесь узел — вершина, где принимается решение.



- ❑ *Дерево решений* — метод представления решающих правил в ориентированном дереве.
- ❑ *Ориентированное дерево* — ориентированный граф, не содержащий циклов.
- ❑ Вершина, не имеющая входящих дуг — *корень дерева*.
- ❑ Во все остальные вершины входит ровно одна дуга.
- ❑ Вершины делим на два типа — *листья* и *узлы*.
- ❑ Вершины не имеющие исходящих дуг — *листья*, остальные вершины — *узлы* (включая корень дерева).
- ❗ В классическом определении ориентированного графа все его вершины называются узлами. Здесь узел — вершина, где принимается решение.
- ❑ Для каждого листа определена метка соответствующего класса  $y_i \in \mathbf{Y} = \{y_1, \dots, y_p\}$ .

- ❑ *Дерево решений* — метод представления решающих правил в ориентированном дереве.
- ❑ *Ориентированное дерево* — ориентированный граф, не содержащий циклов.
- ❑ Вершина, не имеющая входящих дуг — *корень дерева*.
- ❑ Во все остальные вершины входит ровно одна дуга.
- ❑ Вершины делим на два типа — *листья* и *узлы*.
- ❑ Вершины не имеющие исходящих дуг — *листья*, остальные вершины — *узлы* (включая корень дерева).
- ❗ В классическом определении ориентированного графа все его вершины называются узлами. Здесь узел — вершина, где принимается решение.
- ❑ Для каждого листа определена метка соответствующего класса  $y_i \in \mathbf{Y} = \{y_1, \dots, y_p\}$ .
- ❑ Каждый узел сопоставляется с одной из входных переменных.

- ☐ *Дерево решений* — метод представления решающих правил в ориентированном дереве.
- ☐ *Ориентированное дерево* — ориентированный граф, не содержащий циклов.
- ☐ Вершина, не имеющая входящих дуг — *корень дерева*.
- ☐ Во все остальные вершины входит ровно одна дуга.
- ☐ Вершины делим на два типа — *листья* и *узлы*.
- ☐ Вершины не имеющие исходящих дуг — *листья*, остальные вершины — *узлы* (включая корень дерева).
- ☒ **i** В классическом определении ориентированного графа все его вершины называются узлами. Здесь узел — вершина, где принимается решение.
- ☐ Для каждого листа определена метка соответствующего класса  $y_i \in Y = \{y_1, \dots, y_p\}$ .
- ☐ Каждый узел сопоставляется с одной из входных переменных.
- ☐ Для каждой пары вершин  $u$  и  $v$  соединенных дугой  $[u, v]$ , задана функция перехода  $f(u, v) : u \rightarrow v$  по значению предиката (обычно их записывают на дугах).

- ☐ *Дерево решений* — метод представления решающих правил в ориентированном дереве.
- ☐ *Ориентированное дерево* — ориентированный граф, не содержащий циклов.
- ☐ Вершина, не имеющая входящих дуг — *корень дерева*.
- ☐ Во все остальные вершины входит ровно одна дуга.
- ☐ Вершины делим на два типа — *листья* и *узлы*.
- ☐ Вершины не имеющие исходящих дуг — *листья*, остальные вершины — *узлы* (включая корень дерева).
- ☒ **i** В классическом определении ориентированного графа все его вершины называются узлами. Здесь узел — вершина, где принимается решение.
- ☐ Для каждого листа определена метка соответствующего класса  $y_i \in Y = \{y_1, \dots, y_p\}$ .
- ☐ Каждый узел сопоставляется с одной из входных переменных.
- ☐ Для каждой пары вершин  $u$  и  $v$  соединенных дугой  $[u, v]$ , задана функция перехода  $f(u, v) : u \rightarrow v$  по значению предиката (обычно их записывают на дугах).
- ☐ К каждому листу есть единственный путь из корня. Чтобы классифицировать новый объект, нужно пройти по этому пути.

---

<sup>4</sup>На основании задачи из <https://ru.wikipedia.org/wiki/>

На рисунке 12 пример дерева решений<sup>4</sup>.

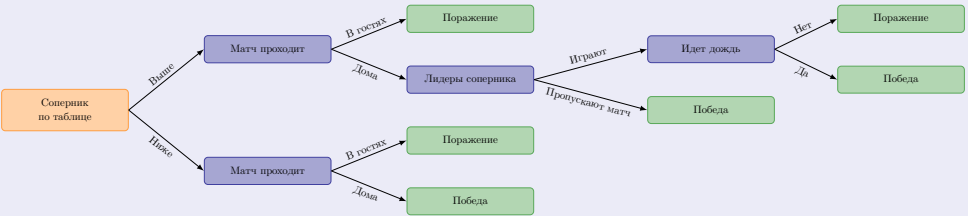


Рис. 12

<sup>4</sup>На основании задачи из <https://ru.wikipedia.org/wiki/>

На рисунке 12 пример дерева решений<sup>4</sup>.

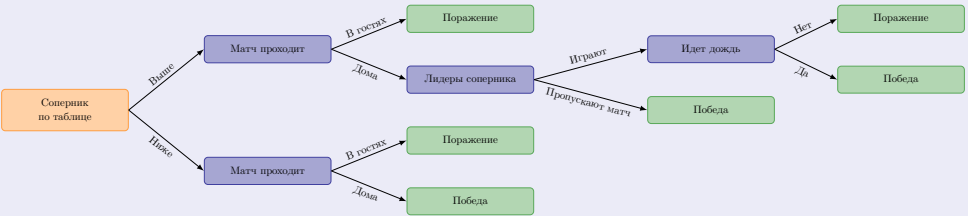


Рис. 12

□ Корень дерева — желтого цвета, листья — зеленого, узлы — фиолетового.

<sup>4</sup>На основании задачи из <https://ru.wikipedia.org/wiki/>

На рисунке 12 пример дерева решений<sup>4</sup>.

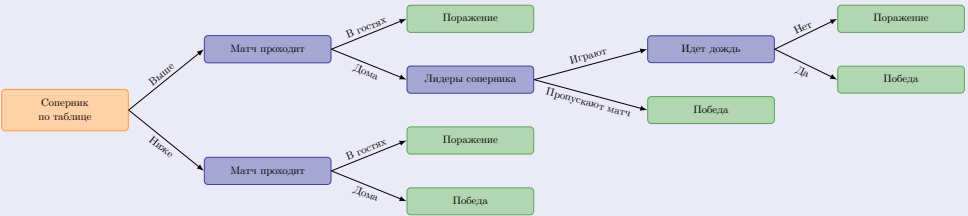


Рис. 12

- ❑ Корень дерева — желтого цвета, листья — зеленого, узлы — фиолетового.
- ❑ Два класса классификации: «Победа», «Поражение».

<sup>4</sup>На основании задачи из <https://ru.wikipedia.org/wiki/>



На рисунке 12 пример дерева решений<sup>4</sup>.

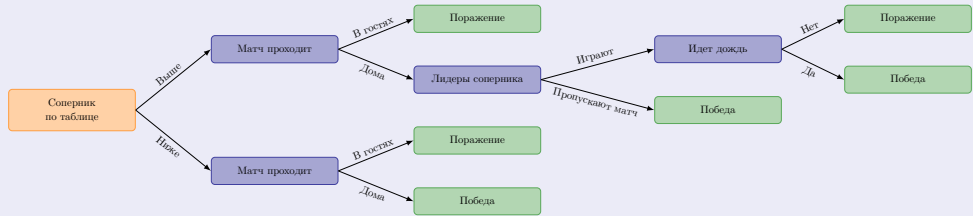


Рис. 12

- ❑ Корень дерева — желтого цвета, листья — зеленого, узлы — фиолетового.
- ❑ Два класса классификации: «Победа», «Поражение».
- ❑ Четыре предиката:  $P$ (Соперник по таблице),  $Q$ (Матч проходит),  $S$ (Лидеры соперника) и  $T$ (Дождь).

<sup>4</sup>На основании задачи из <https://ru.wikipedia.org/wiki/>

На рисунке 12 пример дерева решений<sup>4</sup>.

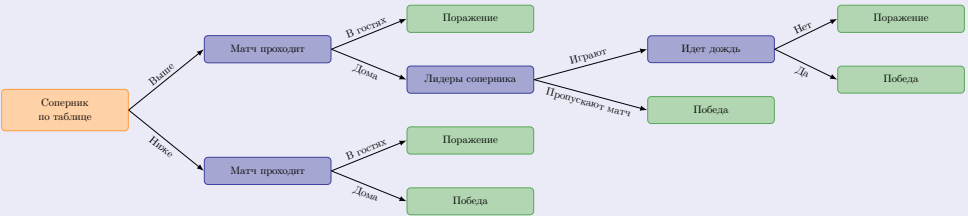


Рис. 12

- ❑ Корень дерева — желтого цвета, листья — зеленого, узлы — фиолетового.
- ❑ Два класса классификации: «Победа», «Поражение».
- ❑ Четыре предиката:  $P$ (Соперник по таблице),  $Q$ (Матч проходит),  $S$ (Лидеры соперника) и  $T$ (Дождь).
- ❓ «Соперник по таблице» — выше, «Матч проходит» — дома, «Лидеры соперника» — играют, «Дождь» — не идет.

<sup>4</sup>На основании задачи из <https://ru.wikipedia.org/wiki/>

На рисунке 12 пример дерева решений<sup>4</sup>.

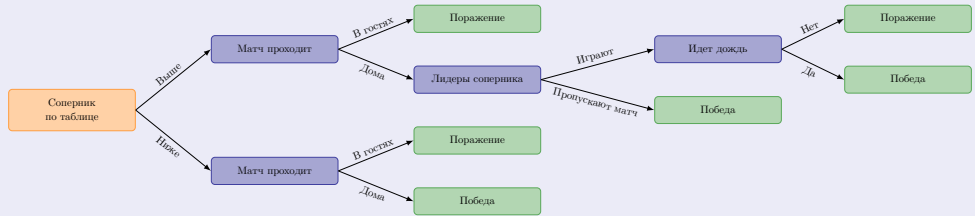


Рис. 12

- ❑ Корень дерева — желтого цвета, листья — зеленого, узлы — фиолетового.
- ❑ Два класса классификации: «Победа», «Поражение».
- ❑ Четыре предиката:  $P$ (Соперник по таблице),  $Q$ (Матч проходит),  $S$ (Лидеры соперника) и  $T$ (Дождь).
- ❓ «Соперник по таблице» — выше, «Матч проходит» — дома, «Лидеры соперника» — играют, «Дождь» — не идет.
- ❖ Проходим по дереву решений — наша команда проиграет.

<sup>4</sup>На основании задачи из <https://ru.wikipedia.org/wiki/>

На рисунке 12 пример дерева решений<sup>4</sup>.

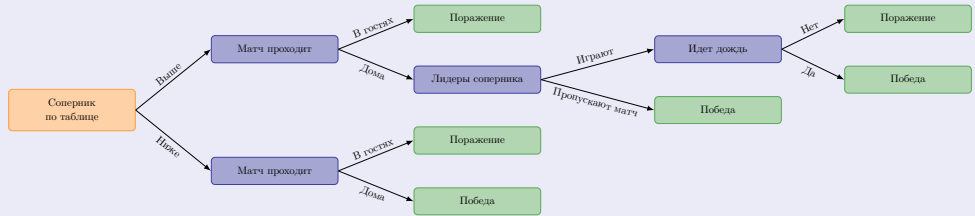


Рис. 12

- ❑ Корень дерева — желтого цвета, листья — зеленого, узлы — фиолетового.
- ❑ Два класса классификации: «Победа», «Поражение».
- ❑ Четыре предиката:  $P$ (Соперник по таблице),  $Q$ (Матч проходит),  $S$ (Лидеры соперника) и  $T$ (Дождь).
- ❓ «Соперник по таблице» — выше, «Матч проходит» — дома, «Лидеры соперника» — играют, «Дождь» — не идет.
- ❖ Проходим по дереву решений — наша команда проиграет.

**i** В примере показано *бинарное* дерево решений: у корня и всех узлов по две исходящих дуги. Это наиболее часто встречающийся на практике вариант.

<sup>4</sup>На основании задачи из <https://ru.wikipedia.org/wiki/>



## Идея построения

- 1 Последовательно проводим рекурсивное разбиение обучающего множества на подмножества на основании некоторой *функции потерь*.

## Идея построения

- 1 Последовательно проводим рекурсивное разбиение обучающего множества на подмножества на основании некоторой *функции потерь*.
- 2 Разбиения продолжаем до тех пор, пока все узлы в конце всех ветвей не будут «объявлены» листьями.

## Идея построения

- 1 Последовательно проводим рекурсивное разбиение обучающего множества на подмножества на основании некоторой *функции потерь*.
- 2 Разбиения продолжаем до тех пор, пока все узлы в конце всех ветвей не будут «объявлены» листьями.
- 3 Узел становится листом в двух случаях:



## Идея построения

- 1 Последовательно проводим рекурсивное разбиение обучающего множества на подмножества на основании некоторой *функции потерь*.
- 2 Разбиения продолжаем до тех пор, пока все узлы в конце всех ветвей не будут «объявлены» листьями.
- 3 Узел становится листом в двух случаях:
  - ✓ содержит единственный объект, или объекты только одного класса;

## Идея построения

- 1 Последовательно проводим рекурсивное разбиение обучающего множества на подмножества на основании некоторой *функции потерь*.
- 2 Разбиения продолжаем до тех пор, пока все узлы в конце всех ветвей не будут «объявлены» листьями.
- 3 Узел становится листом в двух случаях:
  - ✓ содержит единственный объект, или объекты только одного класса;
  - ✓ выполнено пользовательское условие остановки разбиения, например: минимально допустимое число примеров в узле или максимальная глубина дерева.

## Идея построения

- 1 Последовательно проводим рекурсивное разбиение обучающего множества на подмножества на основании некоторой *функции потерь*.
- 2 Разбиения продолжаем до тех пор, пока все узлы в конце всех ветвей не будут «объявлены» листьями.
- 3 Узел становится листом в двух случаях:
  - ✓ содержит единственный объект, или объекты только одного класса;
  - ✓ выполнено пользовательское условие остановки разбиения, например: минимально допустимое число примеров в узле или максимальная глубина дерева.

## Вопросы

- ? В каком порядке выбирать признаки. Нужен критерий «оптимальности» выбора.

## Идея построения

- 1 Последовательно проводим рекурсивное разбиение обучающего множества на подмножества на основании некоторой *функции потерь*.
- 2 Разбиения продолжаем до тех пор, пока все узлы в конце всех ветвей не будут «объявлены» листьями.
- 3 Узел становится листом в двух случаях:
  - ✓ содержит единственный объект, или объекты только одного класса;
  - ✓ выполнено пользовательское условие остановки разбиения, например: минимально допустимое число примеров в узле или максимальная глубина дерева.

## Вопросы

- ? В каком порядке выбирать признаки. Нужен критерий «оптимальности» выбора.
- ? Как оценить качество разбиения объектов в данном узле?

# Построение дерева решений

## Идея построения

- 1 Последовательно проводим рекурсивное разбиение обучающего множества на подмножества на основании некоторой *функции потерь*.
- 2 Разбиения продолжаем до тех пор, пока все узлы в конце всех ветвей не будут «объявлены» листьями.
- 3 Узел становится листом в двух случаях:
  - ✓ содержит единственный объект, или объекты только одного класса;
  - ✓ выполнено пользовательское условие остановки разбиения, например: минимально допустимое число примеров в узле или максимальная глубина дерева.

## Вопросы

- ? В каком порядке выбирать признаки. Нужен критерий «оптимальности» выбора.
- ? Как оценить качество разбиения объектов в данном узле?

## Решение

- А** Построение оптимального дерева решений является NP-полной задачей (полиномиально проверяемой). Нужен полный перебор (*backtracking*) — согласно *теореме Кэли* число различных деревьев с  $n$  вершинами равно  $n^{n-2}$ .

# Построение дерева решений

## Идея построения

- 1 Последовательно проводим рекурсивное разбиение обучающего множества на подмножества на основании некоторой *функции потерь*.
- 2 Разбиения продолжаем до тех пор, пока все узлы в конце всех ветвей не будут «объявлены» листьями.
- 3 Узел становится листом в двух случаях:
  - ✓ содержит единственный объект, или объекты только одного класса;
  - ✓ выполнено пользовательское условие остановки разбиения, например: минимально допустимое число примеров в узле или максимальная глубина дерева.

## Вопросы

- ? В каком порядке выбирать признаки. Нужен критерий «оптимальности» выбора.
- ? Как оценить качество разбиения объектов в данном узле?

## Решение

- A Построение оптимального дерева решений является NP-полной задачей (полиномиально проверяемой). Нужен полный перебор (*backtracking*) — согласно *теореме Кэли* число различных деревьев с  $n$  вершинами равно  $n^{n-2}$ .
- B Будем опираться на эвристику, строить «жадный» алгоритм — оптимальное решение выбираем локально в каждом узле.

# Построение дерева решений

## Идея построения

- 1 Последовательно проводим рекурсивное разбиение обучающего множества на подмножества на основании некоторой *функции потерь*.
- 2 Разбиения продолжаем до тех пор, пока все узлы в конце всех ветвей не будут «объявлены» листьями.
- 3 Узел становится листом в двух случаях:
  - ✓ содержит единственный объект, или объекты только одного класса;
  - ✓ выполнено пользовательское условие остановки разбиения, например: минимально допустимое число примеров в узле или максимальная глубина дерева.

## Вопросы

- ? В каком порядке выбирать признаки. Нужен критерий «оптимальности» выбора.
- ? Как оценить качество разбиения объектов в данном узле?

## Решение

- A Построение оптимального дерева решений является NP-полной задачей (полиномиально проверяемой). Нужен полный перебор (*backtracking*) — согласно *теореме Кэли* число различных деревьев с  $n$  вершинами равно  $n^{n-2}$ .
- B Будем опираться на эвристику, строить «жадный» алгоритм — оптимальное решение выбираем локально в каждом узле.
- C Далее рассмотрим две функции потерь — *критерий Джини* и *критерий прироста информации (ID3)*.

# Построение дерева решений

## Идея построения

- 1 Последовательно проводим рекурсивное разбиение обучающего множества на подмножества на основании некоторой *функции потерь*.
- 2 Разбиения продолжаем до тех пор, пока все узлы в конце всех ветвей не будут «объявлены» листьями.
- 3 Узел становится листом в двух случаях:
  - ✓ содержит единственный объект, или объекты только одного класса;
  - ✓ выполнено пользовательское условие остановки разбиения, например: минимально допустимое число примеров в узле или максимальная глубина дерева.

## Вопросы

- ? В каком порядке выбирать признаки. Нужен критерий «оптимальности» выбора.
- ? Как оценить качество разбиения объектов в данном узле?

## Решение

- A Построение оптимального дерева решений является NP-полной задачей (полиномиально проверяемой). Нужен полный перебор (*backtracking*) — согласно *теореме Кэли* число различных деревьев с  $n$  вершинами равно  $n^{n-2}$ .
- B Будем опираться на эвристику, строить «жадный» алгоритм — оптимальное решение выбираем локально в каждом узле.
- C Далее рассмотрим две функции потерь — *критерий Джини* и *критерий прироста информации (ID3)*.
- i Дерево решений можно построить так, чтоб в каждом листе был ровно один объект. Но как правило это не делается — такое дерево будет *переобученным* (оно слишком настроится на обучающую выборку и будет плохо работать на новых данных).





**А** *Критерий Джини* (Gini) — один из эвристических способов формализовать представление о качестве разбиения в дереве. Критерий Джини используется для создания только двоичных разбиений в дереве.

- А *Критерий Джини* (Gini) — один из эвристических способов формализовать представление о качестве разбиения в дереве. Критерий Джини используется для создания только двоичных разбиений в дереве.
- В Идея в уменьшении «неопределенности» в узле. То есть в узле должно быть как можно больше примеров одного класса и как можно меньше других классов. То есть — это мера дисперсии между классами.

- А** *Критерий Джини* (Gini) — один из эвристических способов формализовать представление о качестве разбиения в дереве. Критерий Джини используется для создания только двоичных разбиений в дереве.
- В** Идея в уменьшении «неопределенности» в узле. То есть в узле должно быть как можно больше примеров одного класса и как можно меньше других классов. То есть — это мера дисперсии между классами.
- С** **Пример.** В узле 100 примеров двух классов: 50 — первого и 50 — второго.

- А** *Критерий Джини* (Gini) — один из эвристических способов формализовать представление о качестве разбиения в дереве. Критерий Джини используется для создания только двоичных разбиений в дереве.
- В** Идея в уменьшении «неопределенности» в узле. То есть в узле должно быть как можно больше примеров одного класса и как можно меньше других классов. То есть — это мера дисперсии между классами.
- С** **Пример.** В узле 100 примеров двух классов: 50 — первого и 50 — второго.  
При разбиении стало  $35 + 10$  примеров в одном узле и  $10 + 45$  в другом — интуитивно «неопределенность» уменьшилась.

- А** *Критерий Джини* (Gini) — один из эвристических способов формализовать представление о качестве разбиения в дереве. Критерий Джини используется для создания только двоичных разбиений в дереве.
- В** Идея в уменьшении «неопределенности» в узле. То есть в узле должно быть как можно больше примеров одного класса и как можно меньше других классов. То есть — это мера дисперсии между классами.
- С** **Пример.** В узле 100 примеров двух классов: 50 — первого и 50 — второго.  
При разбиении стало  $35 + 10$  примеров в одном узле и  $10 + 45$  в другом — интуитивно «неопределенность» уменьшилась.  
Идеальное разбиение:  $50 + 0$  и  $0 + 50$ .

**А** *Критерий Джини* (Gini) — один из эвристических способов формализовать представление о качестве разбиения в дереве. Критерий Джини используется для создания только двоичных разбиений в дереве.

**В** Идея в уменьшении «неопределенности» в узле. То есть в узле должно быть как можно больше примеров одного класса и как можно меньше других классов. То есть — это мера дисперсии между классами.

**С** **Пример.** В узле 100 примеров двух классов: 50 — первого и 50 — второго.

При разбиении стало 35 + 10 примеров в одном узле и 10 + 45 в другом — интуитивно «неопределенность» уменьшилась.

Идеальное разбиение: 50 + 0 и 0 + 50.

## Формальное определение

✓ Пусть узел  $U$  содержит данные  $n$  классов,  $p_i$ ,  $i \in 1; n$  — вероятность класса  $y_i$  в  $U$ . *Индекс* Джини для  $U$ :

$$\text{Gini}(U) = 1 - \sum_{i=1}^n p_i^2, \quad 0 \leq \text{Gini}(U) \leq 1. \quad (20)$$

Если  $\text{Gini}(U) = 0$  — все примеры результирующего множества относятся к одному классу. При  $\text{Gini}(U) = 1$ , классы представлены в равных пропорциях и равновероятны.

# Деревья решений. Критерий Джини

**А** *Критерий Джини* (Gini) — один из эвристических способов формализовать представление о качестве разбиения в дереве. Критерий Джини используется для создания только двоичных разбиений в дереве.

**В** Идея в уменьшении «неопределенности» в узле. То есть в узле должно быть как можно больше примеров одного класса и как можно меньше других классов. То есть — это мера дисперсии между классами.

**С** **Пример.** В узле 100 примеров двух классов: 50 — первого и 50 — второго.

При разбиении стало 35 + 10 примеров в одном узле и 10 + 45 в другом — интуитивно «неопределенность» уменьшилась.

Идеальное разбиение: 50 + 0 и 0 + 50.

## Формальное определение

✓ Пусть узел  $U$  содержит данные  $n$  классов,  $p_i, i = 1; n$  — вероятность класса  $y_i$  в  $U$ . *Индекс* Джини для  $U$ :

$$\text{Gini}(U) = 1 - \sum_{i=1}^n p_i^2, \quad 0 \leq \text{Gini}(U) \leq 1. \quad (20)$$

Если  $\text{Gini}(U) = 0$  — все примеры результирующего множества относятся к одному классу. При  $\text{Gini}(U) = 1$ , классы представлены в равных пропорциях и равновероятны.

✓ Разбиваем узел  $U$  на два: узел  $U_1$  — с числом примеров  $N_1$  и узел  $U_2$  — с числом примеров  $N_2$ . Показатель качества разбиения:

$$\text{Gini}_{split}(U) = \frac{N_1}{N_1 + N_2} \text{Gini}(U_1) + \frac{N_2}{N_1 + N_2} \text{Gini}(U_2). \quad (21)$$



# Деревья решений. Критерий Джини

**А** *Критерий Джини* (Gini) — один из эвристических способов формализовать представление о качестве разбиения в дереве. Критерий Джини используется для создания только двоичных разбиений в дереве.

**В** Идея в уменьшении «неопределенности» в узле. То есть в узле должно быть как можно больше примеров одного класса и как можно меньше других классов. То есть — это мера дисперсии между классами.

**С** **Пример.** В узле 100 примеров двух классов: 50 — первого и 50 — второго.

При разбиении стало 35 + 10 примеров в одном узле и 10 + 45 в другом — интуитивно «неопределенность» уменьшилась.

Идеальное разбиение: 50 + 0 и 0 + 50.

## Формальное определение

✓ Пусть узел  $U$  содержит данные  $n$  классов,  $p_i, i \in 1; n$  — вероятность класса  $y_i$  в  $U$ . *Индекс* Джини для  $U$ :

$$\text{Gini}(U) = 1 - \sum_{i=1}^n p_i^2, \quad 0 \leq \text{Gini}(U) \leq 1. \quad (20)$$

Если  $\text{Gini}(U) = 0$  — все примеры результирующего множества относятся к одному классу. При  $\text{Gini}(U) = 1$ , классы представлены в равных пропорциях и равновероятны.

✓ Разбиваем узел  $U$  на два: узел  $U_1$  — с числом примеров  $N_1$  и узел  $U_2$  — с числом примеров  $N_2$ . Показатель качества разбиения:

$$\text{Gini}_{split}(U) = \frac{N_1}{N_1 + N_2} \text{Gini}(U_1) + \frac{N_2}{N_1 + N_2} \text{Gini}(U_2). \quad (21)$$

✓ Определяем локально наилучшее разбиение  $U^*$ , для которого значение  $\text{Gini}_{split}(U)$  в (21) минимально:

$$U^* = \arg \min_{U=U_1 \cup U_2} (\text{Gini}_{split}(U)) \quad (22)$$



**A** *Прирост информации* (Iterative Dichotomiser, ID3) — показывает, насколько информативен признак, измеряя изменение энтропии после разделения данных по этому признаку. Прирост информации иногда называют *дивергенцией Кульбака–Лейблера*.

- А *Прирост информации* (Iterative Dichotomiser, ID3) — показывает, насколько информативен признак, измеряя изменение энтропии после разделения данных по этому признаку. Прирост информации иногда называют *дивергенцией Кульбака–Лейблера*.
- В Энтропия — это мера беспорядка или неопределенности в наборе данных. Энтропия измеряет, насколько «смешанным» или неоднородным являются данные.

- Ⓐ *Прирост информации* (Iterative Dichotomiser, ID3) — показывает, насколько информативен признак, измеряя изменение энтропии после разделения данных по этому признаку. Прирост информации иногда называют *дивергенцией Кульбака–Лейблера*.
- Ⓑ Энтропия — это мера беспорядка или неопределенности в наборе данных. Энтропия измеряет, насколько «смешанным» или неоднородным являются данные.
- Ⓒ Энтропия контролирует, как дерево решений решает разделить данные. Разделение происходит по признаку, для которого уменьшение энтропии (прирост информации) является максимальным.

- Ⓐ *Прирост информации* (Iterative Dichotomiser, ID3) — показывает, насколько информативен признак, измеряя изменение энтропии после разделения данных по этому признаку. Прирост информации иногда называют *дивергенцией Кульбака–Лейблера*.
- Ⓑ Энтропия — это мера беспорядка или неопределенности в наборе данных. Энтропия измеряет, насколько «смешанным» или неоднородным являются данные.
- Ⓒ Энтропия контролирует, как дерево решений решает разделить данные. Разделение происходит по признаку, для которого уменьшение энтропии (прирост информации) является максимальным.
- Ⓓ С каждым разделением данные становятся менее «смешанными» и более однородными, что приводит к снижению энтропии, то есть к приросту информации.

- А** *Прирост информации* (Iterative Dichotomiser, ID3) — показывает, насколько информативен признак, измеряя изменение энтропии после разделения данных по этому признаку. Прирост информации иногда называют *дивергенцией Кульбака–Лейблера*.
- В** Энтропия — это мера беспорядка или неопределенности в наборе данных. Энтропия измеряет, насколько «смешанным» или неоднородным являются данные.
- С** Энтропия контролирует, как дерево решений решает разделить данные. Разделение происходит по признаку, для которого уменьшение энтропии (прирост информации) является максимальным.
- Д** С каждым разделением данные становятся менее «смешанными» и более однородными, что приводит к снижению энтропии, то есть к приросту информации.
- Е** Прирост информации показывает, насколько неопределенность в наборе данных/узле была уменьшена после разделения по признаку — насколько признак является хорошим выбором для разделения.

- A** *Прирост информации* (Iterative Dichotomiser, ID3) — показывает, насколько информативен признак, измеряя изменение энтропии после разделения данных по этому признаку. Прирост информации иногда называют *дивергенцией Кульбака–Лейблера*.
- B** Энтропия — это мера беспорядка или неопределенности в наборе данных. Энтропия измеряет, насколько «смешанным» или неоднородным являются данные.
- C** Энтропия контролирует, как дерево решений решает разделить данные. Разделение происходит по признаку, для которого уменьшение энтропии (прирост информации) является максимальным.
- D** С каждым разделением данные становятся менее «смешанными» и более однородными, что приводит к снижению энтропии, то есть к приросту информации.
- E** Прирост информации показывает, насколько неопределенность в наборе данных/узле была уменьшена после разделения по признаку — насколько признак является хорошим выбором для разделения.
- F** **Формальное определение.** Пусть узел  $U$  содержит данные  $n$  классов,  $p_i, i \in 1; n$  — вероятность класса  $y_i$  в  $U$ . Определим  $E(U)$  *энтропию* для  $U$ :

$$E(U) = \sum_{i=1}^n -p_i \log_2 p_i, \quad 0 \leq E(U) \leq 1. \quad (23)$$



- А** *Прирост информации* (Iterative Dichotomiser, ID3) — показывает, насколько информативен признак, измеряя изменение энтропии после разделения данных по этому признаку. Прирост информации иногда называют *дивергенцией Кульбака–Лейблера*.
- В** Энтропия — это мера беспорядка или неопределенности в наборе данных. Энтропия измеряет, насколько «смешанным» или неоднородным являются данные.
- С** Энтропия контролирует, как дерево решений решает разделить данные. Разделение происходит по признаку, для которого уменьшение энтропии (прирост информации) является максимальным.
- Д** С каждым разделением данные становятся менее «смешанными» и более однородными, что приводит к снижению энтропии, то есть к приросту информации.
- Е** Прирост информации показывает, насколько неопределенность в наборе данных/узле была уменьшена после разделения по признаку — насколько признак является хорошим выбором для разделения.
- Ф** **Формальное определение.** Пусть узел  $U$  содержит данные  $n$  классов,  $p_i, i \in 1; n$  — вероятность класса  $y_i$  в  $U$ . Определим  $E(U)$  *энтропию* для  $U$ :

$$E(U) = \sum_{i=1}^n -p_i \log_2 p_i, \quad 0 \leq E(U) \leq 1. \quad (23)$$

- Г** Если все примеры в узле принадлежат к одному классу, то энтропия будет равна 0. Если узел содержит равное количество положительных и отрицательных примеров, то энтропия равна 1, т. е. высока.

# Деревья решений. Прирост информации (ID3)

- А** *Прирост информации* (Iterative Dichotomiser, ID3) — показывает, насколько информативен признак, измеряя изменение энтропии после разделения данных по этому признаку. Прирост информации иногда называют *дивергенцией Кульбака–Лейблера*.
- В** Энтропия — это мера беспорядка или неопределенности в наборе данных. Энтропия измеряет, насколько «смешанным» или неоднородным являются данные.
- С** Энтропия контролирует, как дерево решений решает разделить данные. Разделение происходит по признаку, для которого уменьшение энтропии (прирост информации) является максимальным.
- Д** С каждым разделением данные становятся менее «смешанными» и более однородными, что приводит к снижению энтропии, то есть к приросту информации.
- Е** Прирост информации показывает, насколько неопределенность в наборе данных/узле была уменьшена после разделения по признаку — насколько признак является хорошим выбором для разделения.
- Ф** **Формальное определение.** Пусть узел  $U$  содержит данные  $n$  классов,  $p_i, i \in 1; n$  — вероятность класса  $y_i$  в  $U$ . Определим  $E(U)$  *энтропию* для  $U$ :

$$E(U) = \sum_{i=1}^n -p_i \log_2 p_i, \quad 0 \leq E(U) \leq 1. \quad (23)$$

- Г** Если все примеры в узле принадлежат к одному классу, то энтропия будет равна 0. Если узел содержит равное количество положительных и отрицательных примеров, то энтропия равна 1, т. е. высока.
- и** **Пример.** В узле  $U$  имеется 14 примеров двух классов: 9 — первого и 5 — второго. Тогда, согласно (23) энтропия в узле:

$$E(U) = -\frac{9}{14} \log_2 \left( \frac{9}{14} \right) - \frac{5}{14} \log_2 \left( \frac{5}{14} \right) = 0.940$$



**A** Алгоритм *CART* (Classification And Regression Tree, «Дерево Классификации и Регрессии») — один из самых популярных алгоритмов бинарных деревьев решений.

- А Алгоритм *CART* (Classification And Regression Tree, «Дерево Классификации и Регрессии») — один из самых популярных алгоритмов бинарных деревьев решений.
- В Для разбиения в узлах используется критерий Джини.

- А Алгоритм *CART* (Classification And Regression Tree, «Дерево Классификации и Регрессии») — один из самых популярных алгоритмов бинарных деревьев решений.
- В Для разбиения в узлах используется критерий Джини.
- С Основное CART от других алгоритмов — *механизм отсечения дерева* (minimal cost-complexity tree pruning).

- А Алгоритм *CART* (Classification And Regression Tree, «Дерево Классификации и Регрессии») — один из самых популярных алгоритмов бинарных деревьев решений.
- В Для разбиения в узлах используется критерий Джини.
- С Основное CART от других алгоритмов — *механизм отсечения дерева* (minimal cost-complexity tree pruning).

## Отсечение ветвей (pruning)

- ✓ Основная проблема — большое количество возможных отсеченных поддеревьев для одного дерева.

- А Алгоритм *CART* (Classification And Regression Tree, «Дерево Классификации и Регрессии») — один из самых популярных алгоритмов бинарных деревьев решений.
- В Для разбиения в узлах используется критерий Джини.
- С Основное CART от других алгоритмов — *механизм отсечения дерева* (minimal cost-complexity tree pruning).

## Отсечение ветвей (pruning)

- ✓ Основная проблема — большое количество возможных отсеченных поддеревьев для одного дерева.
- ✓ *Идея метода* — рассматриваем не все поддеревья, а только «лучших» представителей.



- А Алгоритм *CART* (Classification And Regression Tree, «Дерево Классификации и Регрессии») — один из самых популярных алгоритмов бинарных деревьев решений.
- В Для разбиения в узлах используется критерий Джини.
- С Основное CART от других алгоритмов — *механизм отсечения дерева* (minimal cost-complexity tree pruning).

## Отсечение ветвей (pruning)

- ✓ Основная проблема — большое количество возможных отсеченных поддеревьев для одного дерева.
- ✓ *Идея метода* — рассматриваем не все поддеревья, а только «лучших» представителей.
- ✓ Пусть  $R(T_n)$  — *ошибка классификации* дерева  $T$  с  $n$  листьями: равна отношению числа неправильно классифицированных примеров к числу примеров в обучающей выборке.

- А Алгоритм *CART* (Classification And Regression Tree, «Дерево Классификации и Регрессии») — один из самых популярных алгоритмов бинарных деревьев решений.
- В Для разбиения в узлах используется критерий Джини.
- С Основное отличие CART от других алгоритмов — *механизм отсечения дерева* (minimal cost-complexity tree pruning).

## Отсечение ветвей (pruning)

- ✓ Основная проблема — большое количество возможных отсеченных поддеревьев для одного дерева.
- ✓ *Идея метода* — рассматриваем не все поддеревья, а только «лучших» представителей.
- ✓ Пусть  $R(T_n)$  — *ошибка классификации* дерева  $T$  с  $n$  листьями: равна отношению числа неправильно классифицированных примеров к числу примеров в обучающей выборке.
- ✓ Введем в  $R(T_n)$  штраф за размер дерева:

$$R_{\alpha}(T_n) = R(T_n) + \alpha|n|, \quad \alpha \geq 0. \quad (24)$$

- А Алгоритм *CART* (Classification And Regression Tree, «Дерево Классификации и Регрессии») — один из самых популярных алгоритмов бинарных деревьев решений.
- В Для разбиения в узлах используется критерий Джини.
- С Основное CART от других алгоритмов — *механизм отсечения дерева* (minimal cost-complexity tree pruning).

## Отсечение ветвей (pruning)

- ✓ Основная проблема — большое количество возможных отсеченных поддеревьев для одного дерева.
- ✓ *Идея метода* — рассматриваем не все поддеревья, а только «лучших» представителей.
- ✓ Пусть  $R(T_n)$  — *ошибка классификации* дерева  $T$  с  $n$  листьями: равна отношению числа неправильно классифицированных примеров к числу примеров в обучающей выборке.
- ✓ Введем в  $R(T_n)$  штраф за размер дерева:
$$R_{\alpha}(T_n) = R(T_n) + \alpha|n|, \quad \alpha \geq 0. \quad (24)$$
- ✓ Введение  $\alpha$  в (24) — пример *регуляризации* для баланса между качеством классификации и размером дерева.

- А Алгоритм *CART* (Classification And Regression Tree, «Дерево Классификации и Регрессии») — один из самых популярных алгоритмов бинарных деревьев решений.
- В Для разбиения в узлах используется критерий Джини.
- С Основное отличие CART от других алгоритмов — *механизм отсечения дерева* (minimal cost-complexity tree pruning).

## Отсечение ветвей (pruning)

- ✓ Основная проблема — большое количество возможных отсеченных поддеревьев для одного дерева.
  - ✓ *Идея метода* — рассматриваем не все поддеревья, а только «лучших» представителей.
  - ✓ Пусть  $R(T_n)$  — *ошибка классификации* дерева  $T$  с  $n$  листьями: равна отношению числа неправильно классифицированных примеров к числу примеров в обучающей выборке.
  - ✓ Введем в  $R(T_n)$  штраф за размер дерева:
- $$R_\alpha(T_n) = R(T_n) + \alpha|n|, \quad \alpha \geq 0. \quad (24)$$
- ✓ Введение  $\alpha$  в (24) — пример *регуляризации* для баланса между качеством классификации и размером дерева.
  - ✓ Ищем последовательность вложенных в  $T_n$  поддеревьев (с корнем из  $T_n$ ):

$$T_1 \subset T_2 \subset \dots \subset T_n, \quad \alpha_1 > \alpha_2 > \dots > \alpha_n = 0. \quad (25)$$

- А Алгоритм *CART* (Classification And Regression Tree, «Дерево Классификации и Регрессии») — один из самых популярных алгоритмов бинарных деревьев решений.
- В Для разбиения в узлах используется критерий Джини.
- С Основное CART от других алгоритмов — *механизм отсечения дерева* (minimal cost-complexity tree pruning).

## Отсечение ветвей (pruning)

- ✓ Основная проблема — большое количество возможных отсеченных поддеревьев для одного дерева.
- ✓ *Идея метода* — рассматриваем не все поддеревья, а только «лучших» представителей.
- ✓ Пусть  $R(T_n)$  — *ошибка классификации* дерева  $T$  с  $n$  листьями: равна отношению числа неправильно классифицированных примеров к числу примеров в обучающей выборке.
- ✓ Введем в  $R(T_n)$  штраф за размер дерева:

$$R_{\alpha}(T_n) = R(T_n) + \alpha|n|, \quad \alpha \geq 0. \quad (24)$$

- ✓ Введение  $\alpha$  в (24) — пример *регуляризации* для баланса между качеством классификации и размером дерева.
- ✓ Ищем последовательность вложенных в  $T_n$  поддеревьев (с корнем из  $T_n$ ):

$$T_1 \subset T_2 \subset \dots \subset T_n, \quad \alpha_1 > \alpha_2 > \dots > \alpha_n = 0. \quad (25)$$

□ Дерево  $T_1$  — состоит из одной вершины — корня дерева  $T_n$ .

- А Алгоритм *CART* (Classification And Regression Tree, «Дерево Классификации и Регрессии») — один из самых популярных алгоритмов бинарных деревьев решений.
- В Для разбиения в узлах используется критерий Джини.
- С Основное CART от других алгоритмов — *механизм отсечения дерева* (minimal cost-complexity tree pruning).

## Отсечение ветвей (pruning)

- ✓ Основная проблема — большое количество возможных отсеченных поддеревьев для одного дерева.
- ✓ *Идея метода* — рассматриваем не все поддеревья, а только «лучших» представителей.
- ✓ Пусть  $R(T_n)$  — *ошибка классификации* дерева  $T$  с  $n$  листьями: равна отношению числа неправильно классифицированных примеров к числу примеров в обучающей выборке.
- ✓ Введем в  $R(T_n)$  штраф за размер дерева:

$$R_\alpha(T_n) = R(T_n) + \alpha|n|, \quad \alpha \geq 0. \quad (24)$$

- ✓ Введение  $\alpha$  в (24) — пример *регуляризации* для баланса между качеством классификации и размером дерева.
- ✓ Ищем последовательность вложенных в  $T_n$  поддеревьев (с корнем из  $T_n$ ):

$$T_1 \subset T_2 \subset \dots \subset T_n, \quad \alpha_1 > \alpha_2 > \dots > \alpha_n = 0. \quad (25)$$

- Дерево  $T_1$  — состоит из одной вершины — корня дерева  $T_n$ .
- Каждое дерево  $T_k$ ,  $k \in 1 : n - 1$  минимизирует ошибку в (24) для  $\alpha \in (\alpha_{k+1}, \alpha_k]$ .

- А Алгоритм *CART* (Classification And Regression Tree, «Дерево Классификации и Регрессии») — один из самых популярных алгоритмов бинарных деревьев решений.
- В Для разбиения в узлах используется критерий Джини.
- С Основное CART от других алгоритмов — *механизм отсечения дерева* (minimal cost-complexity tree pruning).

## Отсечение ветвей (pruning)

- ✓ Основная проблема — большое количество возможных отсеченных поддеревьев для одного дерева.
- ✓ *Идея метода* — рассматриваем не все поддеревья, а только «лучших» представителей.
- ✓ Пусть  $R(T_n)$  — *ошибка классификации* дерева  $T$  с  $n$  листьями: равна отношению числа неправильно классифицированных примеров к числу примеров в обучающей выборке.
- ✓ Введем в  $R(T_n)$  штраф за размер дерева:

$$R_\alpha(T_n) = R(T_n) + \alpha|n|, \quad \alpha \geq 0. \quad (24)$$

- ✓ Введение  $\alpha$  в (24) — пример *регуляризации* для баланса между качеством классификации и размером дерева.
- ✓ Ищем последовательность вложенных в  $T_n$  поддеревьев (с корнем из  $T_n$ ):

$$T_1 \subset T_2 \subset \dots \subset T_n, \quad \alpha_1 > \alpha_2 > \dots > \alpha_n = 0. \quad (25)$$

- Дерево  $T_1$  — состоит из одной вершины — корня дерева  $T_n$ .
- Каждое дерево  $T_k$ ,  $k \in 1 : n - 1$  минимизирует ошибку в (24) для  $\alpha \in (\alpha_{k+1}, \alpha_k]$ .

- И Последовательность поддеревьев (25) можно эффективно найти путем обхода дерева  $T_n$ .





# Алгоритм CART. Пример на Питоне

```
# Connecting the necessary libraries
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load IRIS dataset. Take the first two features
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

# Feature Scaling using StandardScaler
sc = StandardScaler()
sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model
model = DecisionTreeClassifier().fit(X_tr, y_tr)
pred_y = model.predict(X_t)
```

```
# Connecting the necessary libraries
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

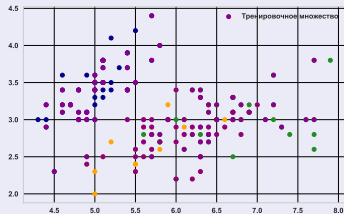
# Load IRIS dataset. Take the first two features
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                         random_state=42, stratify=y)

# Feature Scaling using StandardScaler
sc = StandardScaler()
sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model
model = DecisionTreeClassifier().fit(X_tr, y_tr)
pred_y = model.predict(X_t)
```

- 1 Исходный набор и тренировочная выборка (Ирисы Фишера) изображены на рисунке 13:а.



(а) Исходный набор (Ирисы)

# Алгоритм CART. Пример на Питоне

```
# Connecting the necessary libraries
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

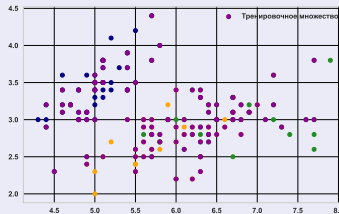
# Load IRIS dataset. Take the first two features
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

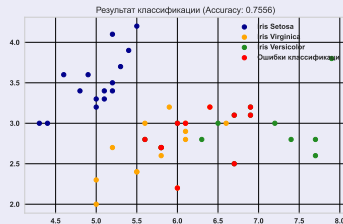
# Feature Scaling using StandardScaler
sc = StandardScaler()
sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model
model = DecisionTreeClassifier().fit(X_tr, y_tr)
pred_y = model.predict(X_t)
```

- 1 Исходный набор и тренировочная выборка (Ирисы Фишера) изображены на рисунке 13:а.
- 2 Результат классификации на тестовой выборке показан на рисунке 13:б. Ошибки классификации отмечены красным цветом.



(а) Исходный набор (Ирисы)



(б) Результаты классификации

# Алгоритм CART. Пример на Питоне

```
# Connecting the necessary libraries
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

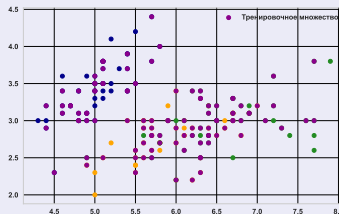
# Load IRIS dataset. Take the first two features
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                         random_state=42, stratify=y)

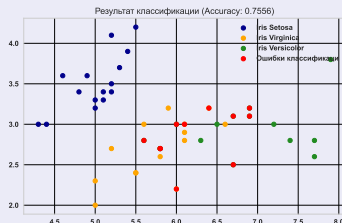
# Feature Scaling using StandardScaler
sc = StandardScaler()
sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model
model = DecisionTreeClassifier().fit(X_tr, y_tr)
pred_y = model.predict(X_t)
```

- 1 Исходный набор и тренировочная выборка (Ирисы Фишера) изображены на рисунке 13:а.
- 2 Результат классификации на тестовой выборке показан на рисунке 13:б. Ошибки классификации отмечены красным цветом.
- 3 Доля правильных ответов:  $\text{Assigasy} = 0.7556$ .



(а) Исходный набор (Ирисы)



(б) Результаты классификации



✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.

- ✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.
- ✓ *Ансамблирование* — обучение нескольких моделей и объединение результатов по некоторому правилу.

- ✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.
- ✓ *Ансамблирование* — обучение нескольких моделей и объединение результатов по некоторому правилу.
- ✓ Деревья ансамбля строим методом *бэггинга* (bagging — bootstrap + aggregation).



- ✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.
- ✓ *Ансамблирование* — обучение нескольких моделей и объединение результатов по некоторому правилу.
- ✓ Деревья ансамбля строим методом *бэггинга* (bagging — bootstrap + aggregation).

## Алгоритм

**Данные:** выборка  $\mathbf{X}$  размера  $n$  и число  $m \approx \sqrt{n}$  — неполное количество признаков для обучения.

- ✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.
- ✓ *Ансамблирование* — обучение нескольких моделей и объединение результатов по некоторому правилу.
- ✓ Деревья ансамбля строим методом *бэггинга* (bagging — bootstrap + aggregation).

## Алгоритм

**Данные:** выборка  $\mathbf{X}$  размера  $n$  и число  $m \approx \sqrt{n}$  — неполное количество признаков для обучения.

- Бутстрэп (bootstrap):

# Ансамбли. Random Forest

- ✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.
- ✓ *Ансамблирование* — обучение нескольких моделей и объединение результатов по некоторому правилу.
- ✓ Деревья ансамбля строим методом *бэггинга* (bagging — bootstrap + aggregation).

## Алгоритм

**Данные:** выборка  $\mathbf{X}$  размера  $n$  и число  $m \approx \sqrt{n}$  — неполное количество признаков для обучения.

□ Бутстрэп (bootstrap):

- 1 Равномерно берем из выборки  $n$  объектов *с возвращением*. Получаем новую выборку  $\mathbf{X}_1 \subset \mathbf{X}$ .

# Ансамбли. Random Forest

- ✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.
- ✓ *Ансамблирование* — обучение нескольких моделей и объединение результатов по некоторому правилу.
- ✓ Деревья ансамбля строим методом *бэггинга* (bagging — bootstrap + aggregation).

## Алгоритм

**Данные:** выборка  $\mathbf{X}$  размера  $n$  и число  $m \approx \sqrt{n}$  — неполное количество признаков для обучения.

□ Бутстрэп (bootstrap):

- 1 Равномерно берем из выборки  $n$  объектов *с возвращением*. Получаем новую выборку  $\mathbf{X}_1 \subset \mathbf{X}$ .
- 2 Повторяя предыдущий шаг  $M$  раз, генерируем подвыборки  $\mathbf{X}_1, \dots, \mathbf{X}_M$ .

# Ансамбли. Random Forest

- ✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.
- ✓ *Ансамблирование* — обучение нескольких моделей и объединение результатов по некоторому правилу.
- ✓ Деревья ансамбля строим методом *бэггинга* (bagging — bootstrap + aggregation).

## Алгоритм

**Данные:** выборка  $\mathbf{X}$  размера  $n$  и число  $m \approx \sqrt{n}$  — неполное количество признаков для обучения.

□ Бутстрэп (bootstrap):

- 1 Равномерно берем из выборки  $n$  объектов *с возвращением*. Получаем новую выборку  $\mathbf{X}_1 \subset \mathbf{X}$ .
- 2 Повторяя предыдущий шаг  $M$  раз, генерируем подвыборки  $\mathbf{X}_1, \dots, \mathbf{X}_M$ .

□ Бэггинг (bagging):

# Ансамбли. Random Forest

- ✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.
- ✓ *Ансамблирование* — обучение нескольких моделей и объединение результатов по некоторому правилу.
- ✓ Деревья ансамбля строим методом *бэггинга* (bagging — bootstrap + aggregation).

## Алгоритм

**Данные:** выборка  $\mathbf{X}$  размера  $n$  и число  $m \approx \sqrt{n}$  — неполное количество признаков для обучения.

□ Бутстрэп (bootstrap):

- 1 Равномерно берем из выборки  $n$  объектов *с возвращением*. Получаем новую выборку  $\mathbf{X}_1 \subset \mathbf{X}$ .
- 2 Повторяя предыдущий шаг  $M$  раз, генерируем подвыборки  $\mathbf{X}_1, \dots, \mathbf{X}_M$ .

□ Бэггинг (bagging):

- 3 На каждой выборке  $\mathbf{X}_i$  строим свой классификатор  $a_i(\mathbf{X})$ .

# Ансамбли. Random Forest

- ✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.
- ✓ *Ансамблирование* — обучение нескольких моделей и объединение результатов по некоторому правилу.
- ✓ Деревья ансамбля строим методом *бэггинга* (bagging — bootstrap + aggregation).

## Алгоритм

**Данные:** выборка  $\mathbf{X}$  размера  $n$  и число  $m \approx \sqrt{n}$  — неполное количество признаков для обучения.

□ Бутстрэп (bootstrap):

- 1 Равномерно берем из выборки  $n$  объектов *с возвращением*. Получаем новую выборку  $\mathbf{X}_1 \subset \mathbf{X}$ .
- 2 Повторяя предыдущий шаг  $M$  раз, генерируем подвыборки  $\mathbf{X}_1, \dots, \mathbf{X}_M$ .

□ Бэггинг (bagging):

- 3 На каждой выборке  $\mathbf{X}_i$  строим свой классификатор  $a_i(\mathbf{X})$ .
- 4 Разбиение проводим не по всем  $n$  признакам, а по  $m$  случайно выбранным. Как правило используется критерий Джини.

# Ансамбли. Random Forest

- ✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.
- ✓ *Ансамблирование* — обучение нескольких моделей и объединение результатов по некоторому правилу.
- ✓ Деревья ансамбля строим методом *бэггинга* (bagging — bootstrap + aggregation).

## Алгоритм

**Данные:** выборка  $\mathbf{X}$  размера  $n$  и число  $m \approx \sqrt{n}$  — неполное количество признаков для обучения.

□ Бутстрэп (bootstrap):

- 1 Равномерно берем из выборки  $n$  объектов *с возвращением*. Получаем новую выборку  $\mathbf{X}_1 \subset \mathbf{X}$ .
- 2 Повторяя предыдущий шаг  $M$  раз, генерируем подвыборки  $\mathbf{X}_1, \dots, \mathbf{X}_M$ .

□ Бэггинг (bagging):

- 3 На каждой выборке  $\mathbf{X}_i$  строим свой классификатор  $a_i(\mathbf{X})$ .
- 4 Разбиение проводим не по всем  $n$  признакам, а по  $m$  случайно выбранным. Как правило используется критерий Джини.
- 5 В большинстве случаев каждое дерево строится до полного исчерпания подвыборки и не подвергается процедуре прунинга, в отличие CART.



# Ансамбли. Random Forest

- ✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.
- ✓ *Ансамблирование* — обучение нескольких моделей и объединение результатов по некоторому правилу.
- ✓ Деревья ансамбля строим методом *бэггинга* (bagging — bootstrap + aggregation).

## Алгоритм

**Данные:** выборка  $\mathbf{X}$  размера  $n$  и число  $m \approx \sqrt{n}$  — неполное количество признаков для обучения.

□ Бутстрэп (bootstrap):

- 1 Равномерно берем из выборки  $n$  объектов *с возвращением*. Получаем новую выборку  $\mathbf{X}_1 \subset \mathbf{X}$ .
- 2 Повторяя предыдущий шаг  $M$  раз, генерируем подвыборки  $\mathbf{X}_1, \dots, \mathbf{X}_M$ .

□ Бэггинг (bagging):

- 3 На каждой выборке  $\mathbf{X}_i$  строим свой классификатор  $a_i(\mathbf{X})$ .
- 4 Разбиение проводим не по всем  $n$  признакам, а по  $m$  случайно выбранным. Как правило используется критерий Джини.
- 5 В большинстве случаев каждое дерево строится до полного исчерпания подвыборки и не подвергается процедуре прунинга, в отличие CART.
- 6 *Голосование*: каждое дерево относит классифицируемый объект к одному из классов — побеждает класс, за который проголосовало наибольшее число деревьев.

# Ансамбли. Random Forest

- ✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.
- ✓ *Ансамблирование* — обучение нескольких моделей и объединение результатов по некоторому правилу.
- ✓ Деревья ансамбля строим методом *бэггинга* (bagging — bootstrap + aggregation).

## Алгоритм

**Данные:** выборка  $\mathbf{X}$  размера  $n$  и число  $m \approx \sqrt{n}$  — неполное количество признаков для обучения.

□ *Бутстрэп* (bootstrap):

- 1 Равномерно берем из выборки  $n$  объектов *с возвращением*. Получаем новую выборку  $\mathbf{X}_1 \subset \mathbf{X}$ .
- 2 Повторяя предыдущий шаг  $M$  раз, генерируем подвыборки  $\mathbf{X}_1, \dots, \mathbf{X}_M$ .

□ *Бэггинг* (bagging):

- 3 На каждой выборке  $\mathbf{X}_i$  строим свой классификатор  $a_i(\mathbf{X})$ .
- 4 Разбиение проводим не по всем  $n$  признакам, а по  $m$  случайно выбранным. Как правило используется критерий Джини.
- 5 В большинстве случаев каждое дерево строится до полного исчерпания подвыборки и не подвергается процедуре прунинга, в отличие CART.
- 6 *Голосование*: каждое дерево относит классифицируемый объект к одному из классов — побеждает класс, за который проголосовало наибольшее число деревьев.

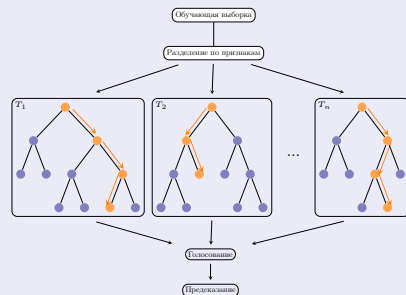


Рис. 14. Пример алгоритма

# Ансамбли. Random Forest

- ✓ *Random Forest* («случайный лес») — алгоритм основан на использовании *ансамбля* (комитета) решающих деревьев.
- ✓ *Ансамблирование* — обучение нескольких моделей и объединение результатов по некоторому правилу.
- ✓ Деревья ансамбля строим методом *бэггинга* (bagging — bootstrap + aggregation).

## Алгоритм

**Данные:** выборка  $\mathbf{X}$  размера  $n$  и число  $m \approx \sqrt{n}$  — неполное количество признаков для обучения.

□ *Бутстрэп* (bootstrap):

- 1 Равномерно берем из выборки  $n$  объектов *с возвращением*. Получаем новую выборку  $\mathbf{X}_1 \subset \mathbf{X}$ .
- 2 Повторяя предыдущий шаг  $M$  раз, генерируем подвыборки  $\mathbf{X}_1, \dots, \mathbf{X}_M$ .

□ *Бэггинг* (bagging):

- 3 На каждой выборке  $\mathbf{X}_i$  строим свой классификатор  $a_i(\mathbf{X})$ .
- 4 Разбиение проводим не по всем  $n$  признакам, а по  $m$  случайно выбранным. Как правило используется критерий Джини.
- 5 В большинстве случаев каждое дерево строится до полного исчерпания подвыборки и не подвергается процедуре прунинга, в отличие CART.
- 6 *Голосование*: каждое дерево относит классифицируемый объект к одному из классов — побеждает класс, за который проголосовало наибольшее число деревьев.



Оптимальное число деревьев —  $M$  подбирается так, чтобы минимизировать ошибку классификатора на тестовой выборке.

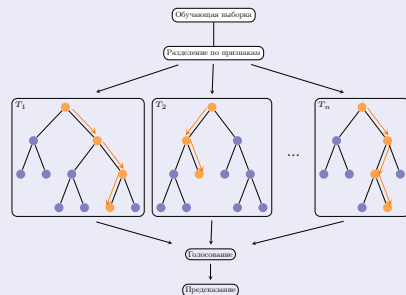


Рис. 14. Пример алгоритма



```
# Connecting the necessary libraries
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load IRIS dataset. Take the first two features
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

# Feature Scaling using StandardScaler
sc = StandardScaler(); sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model (n_estimators - Number of trees)
clf = RandomForestClassifier(n_estimators=100)

#Train the model using the training sets
clf.fit(X_tr_std,y_tr); y_pred = clf.predict(X_t_std)
```

```
# Connecting the necessary libraries
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load IRIS dataset. Take the first two features
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

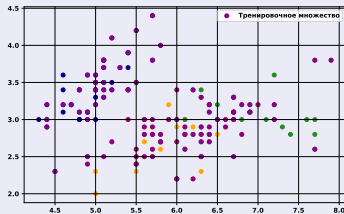
# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

# Feature Scaling using StandardScaler
sc = StandardScaler(); sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model (n_estimators - Number of trees)
clf = RandomForestClassifier(n_estimators=100)

# Train the model using the training sets
clf.fit(X_tr_std, y_tr); y_pred = clf.predict(X_t_std)
```

- 1 Исходный набор и тренировочная выборка (Ирисы Фишера) изображены на рисунке 15.a.



(a) Исходный набор (Ирисы)

```
# Connecting the necessary libraries
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load IRIS dataset. Take the first two features
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

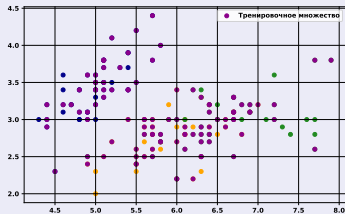
# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                         random_state=42, stratify=y)

# Feature Scaling using StandardScaler
sc = StandardScaler(); sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model (n_estimators - Number of trees)
clf = RandomForestClassifier(n_estimators=100)

# Train the model using the training sets
clf.fit(X_tr_std, y_tr); y_pred = clf.predict(X_t_std)
```

- 1 Исходный набор и тренировочная выборка (Ирисы Фишера) изображены на рисунке 15:а.
- 2 Результат классификации на тестовой выборке показан на рисунке 15:б. Ошибки классификации отмечены красным цветом.



(а) Исходный набор (Ирисы)



(б) Результаты классификации

# Random Forest. Пример на Питоне

```
# Connecting the necessary libraries
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load IRIS dataset. Take the first two features
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

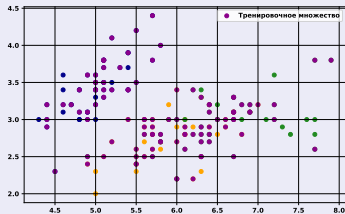
# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

# Feature Scaling using StandardScaler
sc = StandardScaler(); sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)

# Fit the model (n_estimators - Number of trees)
clf = RandomForestClassifier(n_estimators=100)

# Train the model using the training sets
clf.fit(X_tr_std, y_tr); y_pred = clf.predict(X_t_std)
```

- 1 Исходный набор и тренировочная выборка (Ирисы Фишера) изображены на рисунке 15:a.
- 2 Результат классификации на тестовой выборке показан на рисунке 15:b. Ошибки классификации отмечены красным цветом.
- 3 Доля правильных ответов:  $\text{Accuracy} = 0.778$ .



(a) Исходный набор (Ирисы)

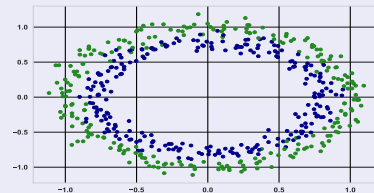


(b) Результаты классификации





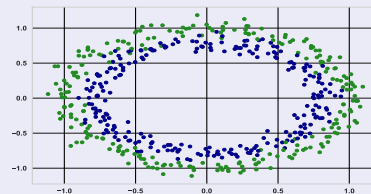
1 Продemonстрируем процесс на линейно неразделимом множестве (пример для алгоритма SVM) —рисунок 16:а.



(a)

Рис. 16

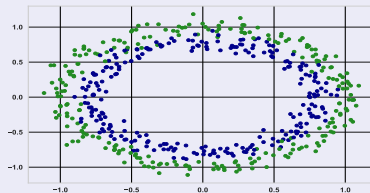
- 1 Продemonстрируем процесс на линейно неразделимом множестве (пример для алгоритма SVM) — рисунок 16:a.
- 2 Основной (но не единственный) гиперпараметр в классификаторе Random Forest это число деревьев.



(a)

Рис. 16

- 1 Продemonстрируем процесс на линейно неразделимом множестве (пример для алгоритма SVM) —рисунок 16:а.
- 2 Основной (но не единственный) гиперпараметр в классификаторе Random Forest это число деревьев.
- 3 Можно оптимизировать только его, используя цикл по параметру, аналогично настройке числа соседей в классификаторе KNN.



(a)

Рис. 16

- 1 Продemonстрируем процесс на линейно неразделимом множестве (пример для алгоритма SVM) —рисунок 16:а.
- 2 Основной (но не единственный) гиперпараметр в классификаторе Random Forest это число деревьев.
- 3 Можно оптимизировать только его, используя цикл по параметру, аналогично настройке числа соседей в классификаторе KNN.
- 4 Генерируем множество, разделяем его на тестовую и тренировочную части, стандартизируем.

```
X, y = make_circles(n_samples=500, noise=0.06, random_state=42)
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                         random_state=42, stratify=y)

sc = StandardScaler()
sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)
```

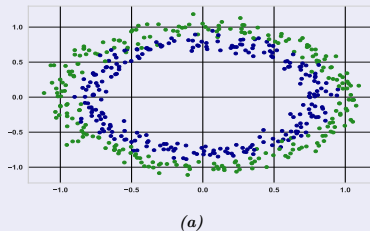


Рис. 16

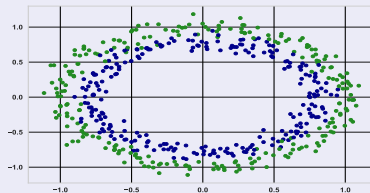
- 1 Продemonстрируем процесс на линейно неразделимом множестве (пример для алгоритма SVM) —рисунок 16:a.
- 2 Основной (но не единственный) гиперпараметр в классификаторе Random Forest это число деревьев.
- 3 Можно оптимизировать только его, используя цикл по параметру, аналогично настройке числа соседей в классификаторе KNN.
- 4 Генерируем множество, разделяем его на тестовую и тренировочную части, стандартизируем.

```
X, y = make_circles(n_samples=500, noise=0.06, random_state=42)
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                         random_state=42, stratify=y)
```

```
sc = StandardScaler()
sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)
```

- 5 Для числа деревьев от 100 до 200 проверяем результаты работы модели.

```
error_rate = []
for i in range(100,200):
    cl = RandomForestClassifier(n_estimators=i)
    cl.fit(X_tr_std, y_tr)
    pred_i = cl.predict(X_t_std)
    error_rate.append(np.mean(pred_i != y_t))
```



(a)

Рис. 16

# Random Forest. Оптимальное число деревьев

- 1 Продemonстрируем процесс на линейно неразделимом множестве (пример для алгоритма SVM) —рисунок 16:a.
- 2 Основной (но не единственный) гиперпараметр в классификаторе Random Forest это число деревьев.
- 3 Можно оптимизировать только его, используя цикл по параметру, аналогично настройке числа соседей в классификаторе KNN.
- 4 Генерируем множество, разделяем его на тестовую и тренировочную части, стандартизируем.

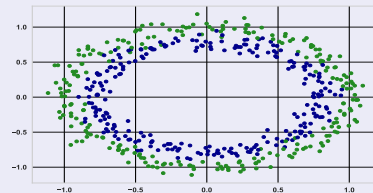
```
X, y = make_circles(n_samples=500, noise=0.06, random_state=42)
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                         random_state=42, stratify=y)
```

```
sc = StandardScaler()
sc.fit(X_tr)
X_tr_std = sc.transform(X_tr); X_t_std = sc.transform(X_t)
```

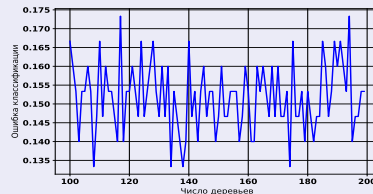
- 5 Для числа деревьев от 100 до 200 проверяем результаты работы модели.

```
error_rate = []
for i in range(100,200):
    cl = RandomForestClassifier(n_estimators=i)
    cl.fit(X_tr_std, y_tr)
    pred_i = cl.predict(X_t_std)
    error_rate.append(np.mean(pred_i != y_t))
```

- 6 График ошибок классификации на рисунке 16:b.



(a)



(b)

Рис. 16





- 1 Рассмотрим выбор оптимальных параметров для Random Forest с помощью Grid SearchCV.

- 1 Рассмотрим выбор оптимальных параметров для Random Forest с помощью Grid SearchCV.
- 2 Проведем перебор гиперпараметров для алгоритма Random Forest на линейно неразделимом множестве предыдущего слайда — рисунок 16:а.

- 1 Рассмотрим выбор оптимальных параметров для Random Forest с помощью Grid SearchCV.
- 2 Проведем перебор гиперпараметров для алгоритма Random Forest на линейно неразделимом множестве предыдущего слайда — рисунок 16:а.
- 3 Множество разделено на тестовую и тренировочную части и стандартизировано. Настраиваем Grid SearchCV:

```
from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier(random_state=42)
param_grid = { 'n_estimators': [100, 200],
               'max_features': ['auto', 'sqrt', 'log2'],
               'max_depth' : [4,5,6,7,8],
               'criterion' :['gini', 'entropy']}
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, scoring='accuracy', refit=True)
```

# Random Forest. Выбор гиперпараметров с помощью GridSearch

- 1 Рассмотрим выбор оптимальных параметров для Random Forest с помощью Grid SearchCV.
- 2 Проведем перебор гиперпараметров для алгоритма Random Forest на линейно неразделимом множестве предыдущего слайда — рисунок 16:а.
- 3 Множество разделено на тестовую и тренировочную части и стандартизировано. Настраиваем Grid SearchCV:

```
from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier(random_state=42)
param_grid = { 'n_estimators': [100, 200],
               'max_features': ['auto', 'sqrt', 'log2'],
               'max_depth' : [4,5,6,7,8],
               'criterion' :['gini', 'entropy']}
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, scoring='accuracy', refit=True)
```

- 4 Перебираем параметры и визуализируем лучший результат:

```
g_res = CV_rfc.fit(X_tr_std, y_tr)
print(g_res.best_params_)
{'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 114}
```



**A** Функция `train_test_split()` из библиотеки [Scikit-learn](#) используется для разделения набора данных на обучающий и тестовый наборы.

- А Функция `train_test_split()` из библиотеки [Scikit-learn](#) используется для разделения набора данных на обучающий и тестовый наборы.
- В Гиперпараметр случайного состояния обозначается `random_state`. Принимает одно из следующих значений.

- А Функция `train_test_split()` из библиотеки [Scikit-learn](#) используется для разделения набора данных на обучающий и тестовый наборы.
- В Гиперпараметр случайного состояния обозначается `random_state`. Принимает одно из следующих значений.  
**None** — значение по умолчанию. Используется случайное состояние из [np.random](#).



- А Функция `train_test_split()` из библиотеки *Scikit-learn* используется для разделения набора данных на обучающий и тестовый наборы.
- В Гиперпараметр случайного состояния обозначается `random_state`. Принимает одно из следующих значений.
  - None** — значение по умолчанию. Используется случайное состояние из *np.random*.Для нескольких вызовов функция будет давать *разные* результаты.

# Разделения датасета, параметр `random_state`

- А Функция `train_test_split()` из библиотеки *Scikit-learn* используется для разделения набора данных на обучающий и тестовый наборы.
- В Гиперпараметр случайного состояния обозначается `random_state`. Принимает одно из следующих значений.

**None** — значение по умолчанию. Используется случайное состояние из *np.random*.

Для нескольких вызовов функция будет давать *разные* результаты.

**int** — любое целое неотрицательное число. Наиболее популярные значения — 0 и 42.

# Разделения датасета, параметр `random_state`

- Ⓐ Функция `train_test_split()` из библиотеки *Scikit-learn* используется для разделения набора данных на обучающий и тестовый наборы.
- Ⓑ Гиперпараметр случайного состояния обозначается `random_state`. Принимает одно из следующих значений.

**None** — значение по умолчанию. Используется случайное состояние из *np.random*.

Для нескольких вызовов функция будет давать *разные* результаты.

**int** — любое целое неотрицательное число. Наиболее популярные значения — 0 и 42.

Для нескольких вызовов функция будет давать *одинаковые* результаты.

# Разделения датасета, параметр `random_state`

- Ⓐ Функция `train_test_split()` из библиотеки *Scikit-learn* используется для разделения набора данных на обучающий и тестовый наборы.
- Ⓑ Гиперпараметр случайного состояния обозначается `random_state`. Принимает одно из следующих значений.
  - None** — значение по умолчанию. Используется случайное состояние из *np.random*.  
Для нескольких вызовов функция будет давать *разные* результаты.
  - int** — любое целое неотрицательное число. Наиболее популярные значения — 0 и 42.  
Для нескольких вызовов функция будет давать *одинаковые* результаты.
- Ⓒ Использование числа 42 в качестве параметра `random_state` в машинном обучении является отсылкой к научно-фантастическому сериалу Дугласа Адамса «*Автостопом по галактике*».

# Разделения датасета, параметр `random_state`

- A** Функция `train_test_split()` из библиотеки *Scikit-learn* используется для разделения набора данных на обучающий и тестовый наборы.
- B** Гиперпараметр случайного состояния обозначается `random_state`. Принимает одно из следующих значений.
- None** — значение по умолчанию. Используется случайное состояние из *np.random*.  
Для нескольких вызовов функция будет давать *разные* результаты.
  - int** — любое целое неотрицательное число. Наиболее популярные значения — 0 и 42.  
Для нескольких вызовов функция будет давать *одинаковые* результаты.
- C** Использование числа 42 в качестве параметра `random_state` в машинном обучении является отсылкой к научно-фантастическому сериалу Дугласа Адамса *«Автостопом по галактике»*.  
В сериале число 42 известно как *«Ответ на главный вопрос жизни, Вселенной и всего остального»*, хотя сам вопрос неизвестен.

# Разделения датасета, параметр `random_state`

- A** Функция `train_test_split()` из библиотеки *Scikit-learn* используется для разделения набора данных на обучающий и тестовый наборы.
- B** Гиперпараметр случайного состояния обозначается `random_state`. Принимает одно из следующих значений.
- None** — значение по умолчанию. Используется случайное состояние из *np.random*.  
Для нескольких вызовов функция будет давать *разные* результаты.
  - int** — любое целое неотрицательное число. Наиболее популярные значения — 0 и 42.  
Для нескольких вызовов функция будет давать *одинаковые* результаты.
- C** Использование числа 42 в качестве параметра `random_state` в машинном обучении является отсылкой к научно-фантастическому сериалу Дугласа Адамса *«Автостопом по галактике»*.
- В сериале число 42 известно как *«Ответ на главный вопрос жизни, Вселенной и всего остального»*, хотя сам вопрос неизвестен.
- Это юмористическая и причудливая ссылка, которая была принята сообществом программистов и специалистов по обработке данных.

# Разделения датасета, параметр `random_state`

**A** Функция `train_test_split()` из библиотеки *Scikit-learn* используется для разделения набора данных на обучающий и тестовый наборы.

**B** Гиперпараметр случайного состояния обозначается `random_state`. Принимает одно из следующих значений.

**None** — значение по умолчанию. Используется случайное состояние из *np.random*.

Для нескольких вызовов функция будет давать *разные* результаты.

**int** — любое целое неотрицательное число. Наиболее популярные значения — 0 и 42.

Для нескольких вызовов функция будет давать *одинаковые* результаты.

**C** Использование числа 42 в качестве параметра `random_state` в машинном обучении является отсылкой к научно-фантастическому сериалу Дугласа Адамса *«Автостопом по галактике»*.

В сериале число 42 известно как *«Ответ на главный вопрос жизни, Вселенной и всего остального»*, хотя сам вопрос неизвестен.

Это юмористическая и причудливая ссылка, которая была принята сообществом программистов и специалистов по обработке данных.

На самом деле нет конкретной причины использовать число 42 вместо любого другого значения в качестве параметра `random_state`.

# Разделения датасета, параметр `random_state`

**A** Функция `train_test_split()` из библиотеки *Scikit-learn* используется для разделения набора данных на обучающий и тестовый наборы.

**B** Гиперпараметр случайного состояния обозначается `random_state`. Принимает одно из следующих значений.

**None** — значение по умолчанию. Используется случайное состояние из *np.random*.

Для нескольких вызовов функция будет давать *разные* результаты.

**int** — любое целое неотрицательное число. Наиболее популярные значения — 0 и 42.

Для нескольких вызовов функция будет давать *одинаковые* результаты.

**C** Использование числа 42 в качестве параметра `random_state` в машинном обучении является отсылкой к научно-фантастическому сериалу Дугласа Адамса *«Автостопом по галактике»*.

В сериале число 42 известно как *«Ответ на главный вопрос жизни, Вселенной и всего остального»*, хотя сам вопрос неизвестен.

Это юмористическая и причудливая ссылка, которая была принята сообществом программистов и специалистов по обработке данных.

На самом деле нет конкретной причины использовать число 42 вместо любого другого значения в качестве параметра `random_state`.

**D** Другие варианты использования параметра `random_state`.



# Разделения датасета, параметр `random_state`

- A** Функция `train_test_split()` из библиотеки *Scikit-learn* используется для разделения набора данных на обучающий и тестовый наборы.
- B** Гиперпараметр случайного состояния обозначается `random_state`. Принимает одно из следующих значений.
- None** — значение по умолчанию. Используется случайное состояние из *np.random*.  
Для нескольких вызовов функция будет давать *разные* результаты.
  - int** — любое целое неотрицательное число. Наиболее популярные значения — 0 и 42.  
Для нескольких вызовов функция будет давать *одинаковые* результаты.
- C** Использование числа 42 в качестве параметра `random_state` в машинном обучении является отсылкой к научно-фантастическому сериалу Дугласа Адамса *«Автостопом по галактике»*.
- В сериале число 42 известно как *«Ответ на главный вопрос жизни, Вселенной и всего остального»*, хотя сам вопрос неизвестен.
- Это юмористическая и причудливая ссылка, которая была принята сообществом программистов и специалистов по обработке данных.
- На самом деле нет конкретной причины использовать число 42 вместо любого другого значения в качестве параметра `random_state`.
- D** Другие варианты использования параметра `random_state`.
- В кластеризации *KMeans* параметр определяет генерацию случайных чисел для инициализации центроидов.

# Разделения датасета, параметр `random_state`

- A** Функция `train_test_split()` из библиотеки *Scikit-learn* используется для разделения набора данных на обучающий и тестовый наборы.
- B** Гиперпараметр случайного состояния обозначается `random_state`. Принимает одно из следующих значений.
- None** — значение по умолчанию. Используется случайное состояние из *np.random*.  
Для нескольких вызовов функция будет давать *разные* результаты.
  - int** — любое целое неотрицательное число. Наиболее популярные значения — 0 и 42.  
Для нескольких вызовов функция будет давать *одинаковые* результаты.
- C** Использование числа 42 в качестве параметра `random_state` в машинном обучении является отсылкой к научно-фантастическому сериалу Дугласа Адамса *«Автостопом по галактике»*.
- В сериале число 42 известно как *«Ответ на главный вопрос жизни, Вселенной и всего остального»*, хотя сам вопрос неизвестен.
- Это юмористическая и причудливая ссылка, которая была принята сообществом программистов и специалистов по обработке данных.
- На самом деле нет конкретной причины использовать число 42 вместо любого другого значения в качестве параметра `random_state`.
- D** Другие варианты использования параметра `random_state`.
- В кластеризации *KMeans* параметр определяет генерацию случайных чисел для инициализации центроидов.
  - В классификаторе *Random Forest* параметр контролирует случайность начальной выборки, и выборку объектов при поиске наилучшего разделения в каждом узле.



## Постановка задачи

## Постановка задачи

- 1 Имеется множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .

## Постановка задачи

- 1 Имеется множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 2 Имеется объект с набором признаков  $\mathbf{X} = \{x_1, \dots, x_k\}$ , его метка класса  $y \in \mathbf{Y}$  неизвестна.

## Постановка задачи

- 1 Имеется множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 2 Имеется объект с набором признаков  $\mathbf{X} = \{x_1, \dots, x_k\}$ , его метка класса  $y \in \mathbf{Y}$  неизвестна.
- 3 Нужно определить вероятности принадлежности  $\mathbf{X}$  классам  $y_i$ .

## Постановка задачи

- 1 Имеется множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 2 Имеется объект с набором признаков  $\mathbf{X} = \{x_1, \dots, x_k\}$ , его метка класса  $y \in \mathbf{Y}$  неизвестна.
- 3 Нужно определить вероятности принадлежности  $\mathbf{X}$  классам  $y_i$ .
- 4 Класс, которому соответствует наибольшая вероятность, будет решением по методу *Байесовской классификации*.



## Постановка задачи

- 1 Имеется множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 2 Имеется объект с набором признаков  $\mathbf{X} = \{x_1, \dots, x_k\}$ , его метка класса  $y \in \mathbf{Y}$  неизвестна.
- 3 Нужно определить вероятности принадлежности  $\mathbf{X}$  классам  $y_i$ .
- 4 Класс, которому соответствует наибольшая вероятность, будет решением по методу *Байесовской классификации*.

## Теорема Байеса

## Постановка задачи

- 1 Имеется множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 2 Имеется объект с набором признаков  $\mathbf{X} = \{x_1, \dots, x_k\}$ , его метка класса  $y \in \mathbf{Y}$  неизвестна.
- 3 Нужно определить вероятности принадлежности  $\mathbf{X}$  классам  $y_i$ .
- 4 Класс, которому соответствует наибольшая вероятность, будет решением по методу *Байесовской классификации*.

## Теорема Байеса

- 1 Формируем множество гипотез  $H_1, \dots, H_p$ , где гипотеза  $H_i$  означает, что  $y = y_i$  (объект  $\mathbf{X}$  принадлежит классу  $y_i$ ).

## Постановка задачи

- 1 Имеется множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 2 Имеется объект с набором признаков  $\mathbf{X} = \{x_1, \dots, x_k\}$ , его метка класса  $y \in \mathbf{Y}$  неизвестна.
- 3 Нужно определить вероятности принадлежности  $\mathbf{X}$  классам  $y_i$ .
- 4 Класс, которому соответствует наибольшая вероятность, будет решением по методу *Байесовской классификации*.

## Теорема Байеса

- 1 Формируем множество гипотез  $H_1, \dots, H_p$ , где гипотеза  $H_i$  означает, что  $y = y_i$  (объект  $\mathbf{X}$  принадлежит классу  $y_i$ ).
- 2  $P(H_i|\mathbf{X}) = P(y = y_i|\mathbf{X})$  — условная вероятность принадлежности  $\mathbf{X}$  классу  $y_i$ .

# Байесовский классификатор. Теорема Байеса

## Постановка задачи

- 1 Имеется множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 2 Имеется объект с набором признаков  $\mathbf{X} = \{x_1, \dots, x_k\}$ , его метка класса  $y \in \mathbf{Y}$  неизвестна.
- 3 Нужно определить вероятности принадлежности  $\mathbf{X}$  классам  $y_i$ .
- 4 Класс, которому соответствует наибольшая вероятность, будет решением по методу *Байесовской классификации*.

## Теорема Байеса

- 1 Формируем множество гипотез  $H_1, \dots, H_p$ , где гипотеза  $H_i$  означает, что  $y = y_i$  (объект  $\mathbf{X}$  принадлежит классу  $y_i$ ).
- 2  $P(H_i | \mathbf{X}) = P(y = y_i | \mathbf{X})$  — условная вероятность принадлежности  $\mathbf{X}$  классу  $y_i$ .
- 3 *Теорема Байеса:*

$$P(y = y_i | \mathbf{X}) = \frac{P(\mathbf{X} | y = y_i) P(y = y_i)}{P(\mathbf{X})}, \quad P(\mathbf{X}) = \sum_{j=1}^p P(\mathbf{X} | y = y_j) P(y = y_j). \quad (26)$$

# Байесовский классификатор. Теорема Байеса

## Постановка задачи

- 1 Имеется множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 2 Имеется объект с набором признаков  $\mathbf{X} = \{x_1, \dots, x_k\}$ , его метка класса  $y \in \mathbf{Y}$  неизвестна.
- 3 Нужно определить вероятности принадлежности  $\mathbf{X}$  классам  $y_i$ .
- 4 Класс, которому соответствует наибольшая вероятность, будет решением по методу *Байесовской классификации*.

## Теорема Байеса

- 1 Формируем множество гипотез  $H_1, \dots, H_p$ , где гипотеза  $H_i$  означает, что  $y = y_i$  (объект  $\mathbf{X}$  принадлежит классу  $y_i$ ).
- 2  $P(H_i | \mathbf{X}) = P(y = y_i | \mathbf{X})$  — условная вероятность принадлежности  $\mathbf{X}$  классу  $y_i$ .
- 3 *Теорема Байеса:*

$$P(y = y_i | \mathbf{X}) = \frac{P(\mathbf{X} | y = y_i) P(y = y_i)}{P(\mathbf{X})}, \quad P(\mathbf{X}) = \sum_{j=1}^p P(\mathbf{X} | y = y_j) P(y = y_j). \quad (26)$$

- i** Теорема Байеса связывает (*априорные* – заданные до опыта) вероятности событий  $y = y_i$  с *апостериорными*.

# Байесовский классификатор. Теорема Байеса

## Постановка задачи

- 1 Имеется множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 2 Имеется объект с набором признаков  $\mathbf{X} = \{x_1, \dots, x_k\}$ , его метка класса  $y \in \mathbf{Y}$  неизвестна.
- 3 Нужно определить вероятности принадлежности  $\mathbf{X}$  классам  $y_i$ .
- 4 Класс, которому соответствует наибольшая вероятность, будет решением по методу *Байесовской классификации*.

## Теорема Байеса

- 1 Формируем множество гипотез  $H_1, \dots, H_p$ , где гипотеза  $H_i$  означает, что  $y = y_i$  (объект  $\mathbf{X}$  принадлежит классу  $y_i$ ).
- 2  $P(H_i | \mathbf{X}) = P(y = y_i | \mathbf{X})$  — условная вероятность принадлежности  $\mathbf{X}$  классу  $y_i$ .
- 3 *Теорема Байеса:*

$$P(y = y_i | \mathbf{X}) = \frac{P(\mathbf{X} | y = y_i) P(y = y_i)}{P(\mathbf{X})}, \quad P(\mathbf{X}) = \sum_{j=1}^p P(\mathbf{X} | y = y_j) P(y = y_j). \quad (26)$$

- i** Теорема Байеса связывает (*априорные* – заданные до опыта) вероятности событий  $y = y_i$  с *апостериорными*.

□  $P(\mathbf{X})$  — безусловная вероятность объекта  $\mathbf{X}$ , не зависит от метки класса  $y$  и является константой.

# Байесовский классификатор. Теорема Байеса

## Постановка задачи

- 1 Имеется множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 2 Имеется объект с набором признаков  $\mathbf{X} = \{x_1, \dots, x_k\}$ , его метка класса  $y \in \mathbf{Y}$  неизвестна.
- 3 Нужно определить вероятности принадлежности  $\mathbf{X}$  классам  $y_i$ .
- 4 Класс, которому соответствует наибольшая вероятность, будет решением по методу *Байесовской классификации*.

## Теорема Байеса

- 1 Формируем множество гипотез  $H_1, \dots, H_p$ , где гипотеза  $H_i$  означает, что  $y = y_i$  (объект  $\mathbf{X}$  принадлежит классу  $y_i$ ).
- 2  $P(H_i | \mathbf{X}) = P(y = y_i | \mathbf{X})$  — условная вероятность принадлежности  $\mathbf{X}$  классу  $y_i$ .
- 3 *Теорема Байеса:*

$$P(y = y_i | \mathbf{X}) = \frac{P(\mathbf{X} | y = y_i) P(y = y_i)}{P(\mathbf{X})}, \quad P(\mathbf{X}) = \sum_{j=1}^p P(\mathbf{X} | y = y_j) P(y = y_j). \quad (26)$$

- i** Теорема Байеса связывает (*априорные* – заданные до опыта) вероятности событий  $y = y_i$  с *апостериорными*.

□  $P(\mathbf{X})$  — безусловная вероятность объекта  $\mathbf{X}$ , не зависит от метки класса  $y$  и является константой.

□  $P(y = y_i)$  — априорная вероятность класса  $y_i$ .

# Байесовский классификатор. Теорема Байеса

## Постановка задачи

- 1 Имеется множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 2 Имеется объект с набором признаков  $\mathbf{X} = \{x_1, \dots, x_k\}$ , его метка класса  $y \in \mathbf{Y}$  неизвестна.
- 3 Нужно определить вероятности принадлежности  $\mathbf{X}$  классам  $y_i$ .
- 4 Класс, которому соответствует наибольшая вероятность, будет решением по методу *Байесовской классификации*.

## Теорема Байеса

- 1 Формируем множество гипотез  $H_1, \dots, H_p$ , где гипотеза  $H_i$  означает, что  $y = y_i$  (объект  $\mathbf{X}$  принадлежит классу  $y_i$ ).
- 2  $P(H_i | \mathbf{X}) = P(y = y_i | \mathbf{X})$  — условная вероятность принадлежности  $\mathbf{X}$  классу  $y_i$ .
- 3 *Теорема Байеса:*

$$P(y = y_i | \mathbf{X}) = \frac{P(\mathbf{X} | y = y_i) P(y = y_i)}{P(\mathbf{X})}, \quad P(\mathbf{X}) = \sum_{j=1}^p P(\mathbf{X} | y = y_j) P(y = y_j). \quad (26)$$

- i** Теорема Байеса связывает (*априорные* – заданные до опыта) вероятности событий  $y = y_i$  с *апостериорными*.

- $P(\mathbf{X})$  — безусловная вероятность объекта  $\mathbf{X}$ , не зависит от метки класса  $y$  и является константой.
- $P(y = y_i)$  — априорная вероятность класса  $y_i$ .
- $P(\mathbf{X} | y = y_i)$  — вероятность встретить объект  $\mathbf{X}$  среди объектов класса  $y_i$ .



# Байесовский классификатор. Теорема Байеса

## Постановка задачи

- 1 Имеется множество меток классов  $\mathbf{Y} = \{y_1, \dots, y_p\}$ .
- 2 Имеется объект с набором признаков  $\mathbf{X} = \{x_1, \dots, x_k\}$ , его метка класса  $y \in \mathbf{Y}$  неизвестна.
- 3 Нужно определить вероятности принадлежности  $\mathbf{X}$  классам  $y_i$ .
- 4 Класс, которому соответствует наибольшая вероятность, будет решением по методу *Байесовской классификации*.

## Теорема Байеса

- 1 Формируем множество гипотез  $H_1, \dots, H_p$ , где гипотеза  $H_i$  означает, что  $y = y_i$  (объект  $\mathbf{X}$  принадлежит классу  $y_i$ ).
- 2  $P(H_i | \mathbf{X}) = P(y = y_i | \mathbf{X})$  — условная вероятность принадлежности  $\mathbf{X}$  классу  $y_i$ .
- 3 *Теорема Байеса:*

$$P(y = y_i | \mathbf{X}) = \frac{P(\mathbf{X} | y = y_i) P(y = y_i)}{P(\mathbf{X})}, \quad P(\mathbf{X}) = \sum_{j=1}^p P(\mathbf{X} | y = y_j) P(y = y_j). \quad (26)$$

- i** Теорема Байеса связывает (*априорные* – заданные до опыта) вероятности событий  $y = y_i$  с *апостериорными*.

- $P(\mathbf{X})$  — безусловная вероятность объекта  $\mathbf{X}$ , не зависит от метки класса  $y$  и является константой.
- $P(y = y_i)$  — априорная вероятность класса  $y_i$ .
- $P(\mathbf{X} | y = y_i)$  — вероятность встретить объект  $\mathbf{X}$  среди объектов класса  $y_i$ .
- $P(y = y_i | \mathbf{X})$  — апостериорная вероятность принадлежности  $\mathbf{X}$  классу  $y_i$ .



## Вычисление класса

1 По теореме Байеса (26) определяем искомый класс  $y^*$ :

$$y^* = \operatorname{argmax}_{y_i \in Y} \left\{ \frac{P(\mathbf{X} | y = y_i) P(y = y_i)}{P(\mathbf{X})} \right\} = \operatorname{argmax}_{y_i \in Y} \{ P(\mathbf{X} | y = y_i) P(y = y_i) \} \quad (27)$$

## Вычисление класса

- 1 По теореме Байеса (26) определяем искомый класс  $y^*$ :

$$y^* = \operatorname{argmax}_{y_i \in Y} \left\{ \frac{P(\mathbf{X} | y = y_i) P(y = y_i)}{P(\mathbf{X})} \right\} = \operatorname{argmax}_{y_i \in Y} \{ P(\mathbf{X} | y = y_i) P(y = y_i) \} \quad (27)$$

- 2 Соответствующая гипотеза  $H_i : y = y_i$  называется *максимальной апостериорной гипотезой* (maximum a posteriori hypothesis, MAP).

# Наивный байесовский классификатор

## Вычисление класса

- 1 По теореме Байеса (26) определяем искомый класс  $y^*$ :

$$y^* = \operatorname{argmax}_{y_i \in Y} \left\{ \frac{P(\mathbf{X} | y = y_i) P(y = y_i)}{P(\mathbf{X})} \right\} = \operatorname{argmax}_{y_i \in Y} \{ P(\mathbf{X} | y = y_i) P(y = y_i) \} \quad (27)$$

- 2 Соответствующая гипотеза  $H_i : y = y_i$  называется *максимальной апостериорной гипотезой* (maximum a posteriori hypothesis, MAP).

## «Наивность» классификатора

- 3 Считаем, что признаки объекта  $\{x_1, \dots, x_k\}$  *не зависят друг от друга*. Тогда:

$$P(\mathbf{X} | y = y_i) = \prod_{j=1}^k P(x_j | y = y_i) \quad (28)$$

# Наивный байесовский классификатор

## Вычисление класса

- 1 По теореме Байеса (26) определяем искомый класс  $y^*$ :

$$y^* = \operatorname{argmax}_{y_i \in Y} \left\{ \frac{P(\mathbf{X} | y = y_i) P(y = y_i)}{P(\mathbf{X})} \right\} = \operatorname{argmax}_{y_i \in Y} \{ P(\mathbf{X} | y = y_i) P(y = y_i) \} \quad (27)$$

- 2 Соответствующая гипотеза  $H_i : y = y_i$  называется *максимальной апостериорной гипотезой* (maximum a posteriori hypothesis, MAP).

## «Наивность» классификатора

- 3 Считаем, что признаки объекта  $\{x_1, \dots, x_k\}$  *не зависят друг от друга*. Тогда:

$$P(\mathbf{X} | y = y_i) = \prod_{j=1}^k P(x_j | y = y_i) \quad (28)$$

- 4 С учетом (28) вычисление  $y^*$  в (27) можно упростить:

$$y^* = \operatorname{argmax}_{y_i \in Y} \left\{ P(y = y_i) \prod_{j=1}^k P(x_j | y = y_i) \right\} = \operatorname{argmax}_{y_i \in Y} \left\{ \log P(y = y_i) + \sum_{j=1}^k \log P(x_j | y = y_i) \right\} \quad (29)$$

# Наивный байесовский классификатор

## Вычисление класса

- 1 По теореме Байеса (26) определяем искомый класс  $y^*$ :

$$y^* = \operatorname{argmax}_{y_i \in Y} \left\{ \frac{P(\mathbf{X} | y = y_i) P(y = y_i)}{P(\mathbf{X})} \right\} = \operatorname{argmax}_{y_i \in Y} \{ P(\mathbf{X} | y = y_i) P(y = y_i) \} \quad (27)$$

- 2 Соответствующая гипотеза  $H_i : y = y_i$  называется *максимальной апостериорной гипотезой* (maximum a posteriori hypothesis, MAP).

## «Наивность» классификатора

- 3 Считаем, что признаки объекта  $\{x_1, \dots, x_k\}$  *не зависят друг от друга*. Тогда:

$$P(\mathbf{X} | y = y_i) = \prod_{j=1}^k P(x_j | y = y_i) \quad (28)$$

- 4 С учетом (28) вычисление  $y^*$  в (27) можно упростить:

$$y^* = \operatorname{argmax}_{y_i \in Y} \left\{ P(y = y_i) \prod_{j=1}^k P(x_j | y = y_i) \right\} = \operatorname{argmax}_{y_i \in Y} \left\{ \log P(y = y_i) + \sum_{j=1}^k \log P(x_j | y = y_i) \right\} \quad (29)$$

## Зачем логарифмировать?

- 1 Если признаков очень много, то в (29) будет вычислять произведение большого числа очень маленьких чисел — возможно получение *арифметического переполнения снизу*.

# Наивный байесовский классификатор

## Вычисление класса

- 1 По теореме Байеса (26) определяем искомый класс  $y^*$ :

$$y^* = \operatorname{argmax}_{y_i \in Y} \left\{ \frac{P(\mathbf{X} | y = y_i) P(y = y_i)}{P(\mathbf{X})} \right\} = \operatorname{argmax}_{y_i \in Y} \{ P(\mathbf{X} | y = y_i) P(y = y_i) \} \quad (27)$$

- 2 Соответствующая гипотеза  $H_i : y = y_i$  называется *максимальной апостериорной гипотезой* (maximum a posteriori hypothesis, MAP).

## «Наивность» классификатора

- 3 Считаем, что признаки объекта  $\{x_1, \dots, x_k\}$  *не зависят друг от друга*. Тогда:

$$P(\mathbf{X} | y = y_i) = \prod_{j=1}^k P(x_j | y = y_i) \quad (28)$$

- 4 С учетом (28) вычисление  $y^*$  в (27) можно упростить:

$$y^* = \operatorname{argmax}_{y_i \in Y} \left\{ P(y = y_i) \prod_{j=1}^k P(x_j | y = y_i) \right\} = \operatorname{argmax}_{y_i \in Y} \left\{ \log P(y = y_i) + \sum_{j=1}^k \log P(x_j | y = y_i) \right\} \quad (29)$$

## Зачем логарифмировать?

- 1 Если признаков очень много, то в (29) будет вычислять произведение большого числа очень маленьких чисел — возможно получение *арифметического переполнения снизу*.
- 2 Логарифм — монотонная функция, ее применение к обоим частям (29) не изменит значения  $\operatorname{argmax}$ .





5 Для номинальных (категориальных) признаков (обозначающих принадлежность объекта к какой-то категории) легко вычислить вероятности в (29) по обучающей выборке:

$$\left\{ \begin{array}{l} P(y = y_i) = \frac{N_i}{N}, \\ P(x_j | y = y_i) = \frac{N_{ij}}{\sum_{l=1}^k N_{il}}. \end{array} \right. \quad (30)$$

5 Для номинальных (категориальных) признаков (обозначающих принадлежность объекта к какой-то категории) легко вычислить вероятности в (29) по обучающей выборке:

$$\begin{cases} P(y = y_i) = \frac{N_i}{N}, \\ P(x_j | y = y_i) = \frac{N_{ij}}{\sum_{l=1}^k N_{il}}. \end{cases} \quad (30)$$

✓  $N$  — число элементов в выборке,

5 Для номинальных (категориальных) признаков (обозначающих принадлежность объекта к какой-то категории) легко вычислить вероятности в (29) по обучающей выборке:

$$\begin{cases} P(y = y_i) = \frac{N_i}{N}, \\ P(x_j | y = y_i) = \frac{N_{ij}}{\sum_{l=1}^k N_{il}}. \end{cases} \quad (30)$$

✓  $N$  — число элементов в выборке,

✓  $N_i$  — число элементов класса  $y_i$ ,

5 Для номинальных (категориальных) признаков (обозначающих принадлежность объекта к какой-то категории) легко вычислить вероятности в (29) по обучающей выборке:

$$\begin{cases} P(y = y_i) = \frac{N_i}{N}, \\ P(x_j | y = y_i) = \frac{N_{ij}}{\sum_{l=1}^k N_{il}}. \end{cases} \quad (30)$$

- ✓  $N$  — число элементов в выборке,
- ✓  $N_i$  — число элементов класса  $y_i$ ,
- ✓  $N_{ij}$  — число элементов класса  $y_i$  с признаком  $x_j$ .

# Наивный байесовский классификатор. Номинальные признаки

5 Для номинальных (категориальных) признаков (обозначающих принадлежность объекта к какой-то категории) легко вычислить вероятности в (29) по обучающей выборке:

$$\begin{cases} P(y = y_i) = \frac{N_i}{N}, \\ P(x_j | y = y_i) = \frac{N_{ij}}{\sum_{l=1}^k N_{il}}. \end{cases} \quad (30)$$

- ✓  $N$  — число элементов в выборке,
- ✓  $N_i$  — число элементов класса  $y_i$ ,
- ✓  $N_{ij}$  — число элементов класса  $y_i$  с признаком  $x_j$ .

## Проблема нулевой частоты

1 Если в обучающей выборке класс  $y_i$  и значение признака  $x_j$  не встречаются вместе, то  $P(x_j | y = y_i) = 0$ .

# Наивный байесовский классификатор. Номинальные признаки

5 Для номинальных (категориальных) признаков (обозначающих принадлежность объекта к какой-то категории) легко вычислить вероятности в (29) по обучающей выборке:

$$\begin{cases} P(y = y_i) = \frac{N_i}{N}, \\ P(x_j | y = y_i) = \frac{N_{ij}}{\sum_{l=1}^k N_{il}}. \end{cases} \quad (30)$$

- ✓  $N$  — число элементов в выборке,
- ✓  $N_i$  — число элементов класса  $y_i$ ,
- ✓  $N_{ij}$  — число элементов класса  $y_i$  с признаком  $x_j$ .

## Проблема нулевой частоты

- 1 Если в обучающей выборке класс  $y_i$  и значение признака  $x_j$  не встречаются вместе, то  $P(x_j | y = y_i) = 0$ .
- 2 В (29) получим ноль или логарифм нуля.

5 Для номинальных (категориальных) признаков (обозначающих принадлежность объекта к какой-то категории) легко вычислить вероятности в (29) по обучающей выборке:

$$\begin{cases} P(y = y_i) = \frac{N_i}{N}, \\ P(x_j | y = y_i) = \frac{N_{ij}}{\sum_{l=1}^k N_{il}}. \end{cases} \quad (30)$$

- ✓  $N$  — число элементов в выборке,
- ✓  $N_i$  — число элементов класса  $y_i$ ,
- ✓  $N_{ij}$  — число элементов класса  $y_i$  с признаком  $x_j$ .

## Проблема нулевой частоты

- 1 Если в обучающей выборке класс  $y_i$  и значение признака  $x_j$  не встречаются вместе, то  $P(x_j | y = y_i) = 0$ .
- 2 В (29) получим ноль или логарифм нуля.
- 3 **Решение:** размытие (сглаживание) по Лапласу в (30):

$$P(x_j | y = y_i) = \frac{N_{ij} + \alpha}{\sum_{l=1}^k (N_{il} + \alpha)}, \quad \alpha > 0. \quad (31)$$



# Наивный байесовский классификатор. Номинальные признаки

5 Для номинальных (категориальных) признаков (обозначающих принадлежность объекта к какой-то категории) легко вычислить вероятности в (29) по обучающей выборке:

$$\begin{cases} P(y = y_i) = \frac{N_i}{N}, \\ P(x_j | y = y_i) = \frac{N_{ij}}{\sum_{l=1}^k N_{il}}. \end{cases} \quad (30)$$

- ✓  $N$  — число элементов в выборке,
- ✓  $N_i$  — число элементов класса  $y_i$ ,
- ✓  $N_{ij}$  — число элементов класса  $y_i$  с признаком  $x_j$ .

## Проблема нулевой частоты

- 1 Если в обучающей выборке класс  $y_i$  и значение признака  $x_j$  не встречаются вместе, то  $P(x_j | y = y_i) = 0$ .
- 2 В (29) получим ноль или логарифм нуля.
- 3 **Решение:** размытие (сглаживание) по Лапласу в (30):

$$P(x_j | y = y_i) = \frac{N_{ij} + \alpha}{\sum_{l=1}^k (N_{il} + \alpha)}, \quad \alpha > 0. \quad (31)$$

- 4 Обычно  $\alpha = 1$ .



## Постановка задачи

- 1 Обучающая выборка из текстов  $T_1, \dots, T_k$ .

## Постановка задачи

- 1 Обучающая выборка из текстов  $T_1, \dots, T_k$ .
- 2 Тексты размечены на два класса:  $y_1 = S$  — «спам»,  $y_2 = N$  — «не спам». Признаки  $x_j, j \in 1 : k$  — слова текстов.

## Постановка задачи

- 1 Обучающая выборка из текстов  $T_1, \dots, T_k$ .
- 2 Тексты размечены на два класса:  $y_1 = S$  — «спам»,  $y_2 = N$  — «не спам». Признаки  $x_j, j \in 1 : k$  — слова текстов.
- 3 Нужно определить метку класса для нового текста  $T$ .

## Постановка задачи

- 1 Обучающая выборка из текстов  $T_1, \dots, T_k$ .
  - 2 Тексты размечены на два класса:  $y_1 = S$  — «спам»,  $y_2 = N$  — «не спам». Признаки  $x_j, j \in 1 : k$  — слова текстов.
  - 3 Нужно определить метку класса для нового текста  $T$ .
- i** Предполагаем что вероятность появления слов в тексте не зависит друг от друга.

## Постановка задачи

- 1 Обучающая выборка из текстов  $T_1, \dots, T_k$ .
  - 2 Тексты размечены на два класса:  $y_1 = S$  — «спам»,  $y_2 = N$  — «не спам». Признаки  $x_j, j \in 1 : k$  — слова текстов.
  - 3 Нужно определить метку класса для нового текста  $T$ .
- i Предполагаем что вероятность появления слов в тексте не зависит друг от друга.

## Пример

- ✓ Обучающая выборка:  $T_1(S)$  — «предлагаю услуги агента»,  $T_2(S)$  — «срочно купить квартиру»,  $T_3(N)$  — «нужно купить лекарства».

## Постановка задачи

- 1 Обучающая выборка из текстов  $T_1, \dots, T_k$ .
  - 2 Тексты размечены на два класса:  $y_1 = S$  — «спам»,  $y_2 = N$  — «не спам». Признаки  $x_j, j \in 1 : k$  — слова текстов.
  - 3 Нужно определить метку класса для нового текста  $T$ .
- i** Предполагаем что вероятность появления слов в тексте не зависит друг от друга.

## Пример

- ✓ Обучающая выборка:  $T_1(S)$  — «предлагаю услуги агента»,  $T_2(S)$  — «срочно купить квартиру»,  $T_3(N)$  — «нужно купить лекарства».
- ✓ Новый текст:  $T$  — «нужно купить продукты».



# Наивный байесовский классификатор. Классификация спама

## Постановка задачи

- 1 Обучающая выборка из текстов  $T_1, \dots, T_k$ .
  - 2 Тексты размечены на два класса:  $y_1 = S$  — «спам»,  $y_2 = N$  — «не спам». Признаки  $x_j, j \in 1 : k$  — слова текстов.
  - 3 Нужно определить метку класса для нового текста  $T$ .
- i Предполагаем что вероятность появления слов в тексте не зависит друг от друга.

## Пример

- ✓ Обучающая выборка:  $T_1(S)$  — «предлагаю услуги агента»,  $T_2(S)$  — «срочно купить квартиру»,  $T_3(N)$  — «нужно купить лекарства».
- ✓ Новый текст:  $T$  — «нужно купить продукты».
- ✓ Вероятности вычисляем согласно (30) и (31) с коэффициентом размытия  $\alpha = 1$ .

$$N = 3, N_1 = 2, N_2 = 1, \sum_{l=1}^k N_{S,j} = 6 - \text{число слов в } S, \sum_{l=1}^k N_{T,l} = 3 - \text{число слов в } T,$$

$$N_{S,\text{нужно}} = 0, N_{S,\text{купить}} = 1, N_{S,\text{продукты}} = 0, \quad N_{T,\text{нужно}} = 1, N_{T,\text{купить}} = 1, N_{T,\text{продукты}} = 0$$

Применим формулу (29). Получаем:

$$S : \log P(S) + \log P(\text{нужно} | S) + \log P(\text{купить} | S) + \log P(\text{продукты} | S) = \log \frac{2}{3} + \log \frac{1}{6+6} + \log \frac{2}{6+6} + \log \frac{1}{6+6} \approx -3.1126$$

$$N : \log P(N) + \log P(\text{нужно} | N) + \log P(\text{купить} | N) + \log P(\text{продукты} | S) = \log \frac{1}{3} + \log \frac{2}{3+3} + \log \frac{2}{3+3} + \log \frac{1}{3+3} \approx -2.2094$$

# Наивный байесовский классификатор. Классификация спама

## Постановка задачи

- 1 Обучающая выборка из текстов  $T_1, \dots, T_k$ .
  - 2 Тексты размечены на два класса:  $y_1 = S$  — «спам»,  $y_2 = N$  — «не спам». Признаки  $x_j, j \in 1 : k$  — слова текстов.
  - 3 Нужно определить метку класса для нового текста  $T$ .
- i Предполагаем что вероятность появления слов в тексте не зависит друг от друга.

## Пример

- ✓ Обучающая выборка:  $T_1(S)$  — «предлагаю услуги агента»,  $T_2(S)$  — «срочно купить квартиру»,  $T_3(N)$  — «нужно купить лекарства».
- ✓ Новый текст:  $T$  — «нужно купить продукты».
- ✓ Вероятности вычисляем согласно (30) и (31) с коэффициентом размытия  $\alpha = 1$ .

$$N = 3, N_1 = 2, N_2 = 1, \sum_{l=1}^k N_{S,j} = 6 - \text{число слов в } S, \sum_{l=1}^k N_{T,l} = 3 - \text{число слов в } T,$$

$$N_{S,\text{нужно}} = 0, N_{S,\text{купить}} = 1, N_{S,\text{продукты}} = 0, \quad N_{T,\text{нужно}} = 1, N_{T,\text{купить}} = 1, N_{T,\text{продукты}} = 0$$

Применим формулу (29). Получаем:

$$S : \log P(S) + \log P(\text{нужно} | S) + \log P(\text{купить} | S) + \log P(\text{продукты} | S) = \log \frac{2}{3} + \log \frac{1}{6+6} + \log \frac{2}{6+6} + \log \frac{1}{6+6} \approx -3.1126$$

$$N : \log P(N) + \log P(\text{нужно} | N) + \log P(\text{купить} | N) + \log P(\text{продукты} | S) = \log \frac{1}{3} + \log \frac{2}{3+3} + \log \frac{2}{3+3} + \log \frac{1}{3+3} \approx -2.2094$$

- ✓ Выиграл класс  $N$  — текст  $T$  не будет помечен как спам.



**i** **Полиномиальный.** Используется, когда данные имеют полиномиальное распределение.

**i Полиномиальный.** Используется, когда данные имеют полиномиальное распределение.

Как правило используется для задач классификации документов, т. е. для определения того, к какой категории принадлежит конкретный документ, например «Спорт», «Политика», «Образование» и т. д. Классификатор использует частоту слов из этого документа в качестве признаков.

**i Полиномиальный.** Используется, когда данные имеют полиномиальное распределение.

Как правило используется для задач классификации документов, т. е. для определения того, к какой категории принадлежит конкретный документ, например «Спорт», «Политика», «Образование» и т. д. Классификатор использует частоту слов из этого документа в качестве признаков.

В библиотеке [\*sklearn\*](#):

```
from sklearn.naive_bayes import MultinomialNB
```

**i Полиномиальный.** Используется, когда данные имеют полиномиальное распределение.

Как правило используется для задач классификации документов, т. е. для определения того, к какой категории принадлежит конкретный документ, например «Спорт», «Политика», «Образование» и т. д. Классификатор использует частоту слов из этого документа в качестве признаков.

В библиотеке [sklearn](#):

```
from sklearn.naive_bayes import MultinomialNB
```

**i Классификатор Бернулли.** Работает аналогично полиномиальному классификатору, но признаки являются независимыми логическими переменными.

# Типы наивных байесовских классификаторов

**i Полиномиальный.** Используется, когда данные имеют полиномиальное распределение.

Как правило используется для задач классификации документов, т. е. для определения того, к какой категории принадлежит конкретный документ, например «Спорт», «Политика», «Образование» и т. д. Классификатор использует частоту слов из этого документа в качестве признаков.

В библиотеке [sklearn](#):

```
from sklearn.naive_bayes import MultinomialNB
```

**i Классификатор Бернулли.** Работает аналогично полиномиальному классификатору, но признаки являются независимыми логическими переменными.

Как и полиномиальная модель, этот подход популярен для задач классификации документов, где используются признаки встречаемости (т. е. встречается слово в документе или нет), а не частоты терминов (т. е. частота появления слова в документе).



# Типы наивных байесовских классификаторов

**i Полиномиальный.** Используется, когда данные имеют полиномиальное распределение.

Как правило используется для задач классификации документов, т. е. для определения того, к какой категории принадлежит конкретный документ, например «Спорт», «Политика», «Образование» и т. д. Классификатор использует частоту слов из этого документа в качестве признаков.

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import MultinomialNB
```

**i Классификатор Бернулли.** Работает аналогично полиномиальному классификатору, но признаки являются независимыми логическими переменными.

Как и полиномиальная модель, этот подход популярен для задач классификации документов, где используются признаки встречаемости (т. е. встречается слово в документе или нет), а не частоты терминов (т. е. частота появления слова в документе).

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import BernoulliNB
```

# Типы наивных байесовских классификаторов

**i Полиномиальный.** Используется, когда данные имеют полиномиальное распределение.

Как правило используется для задач классификации документов, т. е. для определения того, к какой категории принадлежит конкретный документ, например «Спорт», «Политика», «Образование» и т. д. Классификатор использует частоту слов из этого документа в качестве признаков.

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import MultinomialNB
```

**i Классификатор Бернулли.** Работает аналогично полиномиальному классификатору, но признаки являются независимыми логическими переменными.

Как и полиномиальная модель, этот подход популярен для задач классификации документов, где используются признаки встречаемости (т. е. встречается слово в документе или нет), а не частоты терминов (т. е. частота появления слова в документе).

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import BernoulliNB
```

**i Гауссовский.** Предполагается, что вероятность признаков является гауссовой, следовательно, условная вероятность определяется выражением:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

# Типы наивных байесовских классификаторов

**i Полиномиальный.** Используется, когда данные имеют полиномиальное распределение.

Как правило используется для задач классификации документов, т. е. для определения того, к какой категории принадлежит конкретный документ, например «Спорт», «Политика», «Образование» и т. д. Классификатор использует частоту слов из этого документа в качестве признаков.

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import MultinomialNB
```

**i Классификатор Бернулли.** Работает аналогично полиномиальному классификатору, но признаки являются независимыми логическими переменными.

Как и полиномиальная модель, этот подход популярен для задач классификации документов, где используются признаки встречаемости (т. е. встречается слово в документе или нет), а не частоты терминов (т. е. частота появления слова в документе).

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import BernoulliNB
```

**i Гауссовский.** Предполагается, что вероятность признаков является гауссовой, следовательно, условная вероятность определяется выражением:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import GaussianNB
```

# Типы наивных байесовских классификаторов

**i Полиномиальный.** Используется, когда данные имеют полиномиальное распределение.

Как правило используется для задач классификации документов, т. е. для определения того, к какой категории принадлежит конкретный документ, например «Спорт», «Политика», «Образование» и т. д. Классификатор использует частоту слов из этого документа в качестве признаков.

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import MultinomialNB
```

**i Классификатор Бернулли.** Работает аналогично полиномиальному классификатору, но признаки являются независимыми логическими переменными.

Как и полиномиальная модель, этот подход популярен для задач классификации документов, где используются признаки встречаемости (т. е. встречается слово в документе или нет), а не частоты терминов (т. е. частота появления слова в документе).

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import BernoulliNB
```

**i Гауссовский.** Предполагается, что вероятность признаков является гауссовой, следовательно, условная вероятность определяется выражением:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import GaussianNB
```

**i Классификатор категориальных признаков.** Используется для классификации с дискретными признаками, которые категориально распределены.

# Типы наивных байесовских классификаторов

**i Полиномиальный.** Используется, когда данные имеют полиномиальное распределение.

Как правило используется для задач классификации документов, т. е. для определения того, к какой категории принадлежит конкретный документ, например «Спорт», «Политика», «Образование» и т. д. Классификатор использует частоту слов из этого документа в качестве признаков.

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import MultinomialNB
```

**i Классификатор Бернулли.** Работает аналогично полиномиальному классификатору, но признаки являются независимыми логическими переменными.

Как и полиномиальная модель, этот подход популярен для задач классификации документов, где используются признаки встречаемости (т. е. встречается слово в документе или нет), а не частоты терминов (т. е. частота появления слова в документе).

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import BernoulliNB
```

**i Гауссовский.** Предполагается, что вероятность признаков является гауссовой, следовательно, условная вероятность определяется выражением:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import GaussianNB
```

**i Классификатор категориальных признаков.** Используется для классификации с дискретными признаками, которые категориально распределены.

В библиотеке *sklearn*:

```
from sklearn.naive_bayes import CategoricalNB
```



# Гауссовский байесовский классификатор. Пример на Питоне

```
# Connecting the necessary libraries
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

# Load IRIS dataset. Take the first two features
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

# Fit the model
gnb = GaussianNB()
y_pred = gnb.fit(X_tr, y_tr).predict(X_t)
```

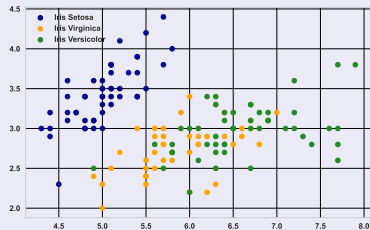
## Гауссовский байесовский классификатор. Пример на Питоне

```
# Connecting the necessary libraries
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

# Load IRIS dataset. Take the first two features
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

# Fit the model
gnb = GaussianNB()
y_pred = gnb.fit(X_tr, y_tr).predict(X_t)
```



(a)

- 1 Dataset (Ирисы Фишера) изображен на рисунке 17:а.



## Гауссовский байесовский классификатор. Пример на Питоне

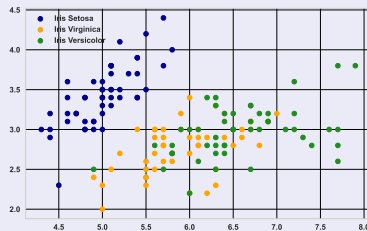
```
# Connecting the necessary libraries
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

# Load IRIS dataset. Take the first two features
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

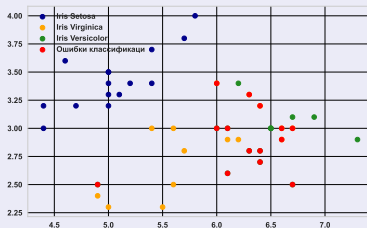
# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

# Fit the model
gnb = GaussianNB()
y_pred = gnb.fit(X_tr, y_tr).predict(X_t)
```

- 1 Dataset (Ирисы Фишера) изображен на рисунке 17:a.
- 2 Результат классификации на тестовой выборке показан на рисунке 17:b. Ошибки классификации отмечены красным цветом.



(a)



(b)

## Гауссовский байесовский классификатор. Пример на Питоне

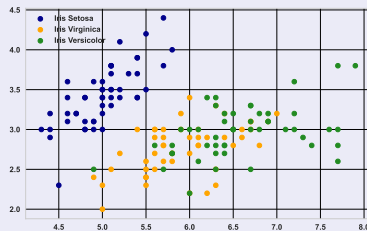
```
# Connecting the necessary libraries
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

# Load IRIS dataset. Take the first two features
iris = datasets.load_iris()
X = iris.data[:, [0,1]]; y = iris.target

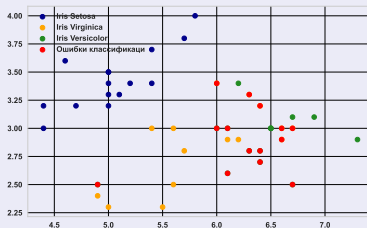
# Create train and test split
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.3,
                                       random_state=42, stratify=y)

# Fit the model
gnb = GaussianNB()
y_pred = gnb.fit(X_tr, y_tr).predict(X_t)
```

- 1 Dataset (Ирисы Фишера) изображен на рисунке 17:a.
- 2 Результат классификации на тестовой выборке показан на рисунке 17:b. Ошибки классификации отмечены красным цветом.
- 3 Доля правильных ответов: Accuracy = 0.689.



(a)



(b)



# Полиномиальный байесовский классификатор. Пример на Питоне

- 1 Используем полиномиальный байесовский классификатор для классификации текстовых документов на два класса  $\{0, 1\}$  на основе частоты слов в документах.

# Полиномиальный байесовский классификатор. Пример на Питоне

- 1 Используем полиномиальный байесовский классификатор для классификации текстовых документов на два класса  $\{0, 1\}$  на основе частоты слов в документах.
- 2 Загружаем библиотеку для модели и конвертер текстовых признаков в числовые.

```
from sklearn.naive_bayes import MultinomialNB  
from sklearn.feature_extraction.text import CountVectorizer
```

# Полиномиальный байесовский классификатор. Пример на Питоне

1 Используем полиномиальный байесовский классификатор для классификации текстовых документов на два класса  $\{0, 1\}$  на основе частоты слов в документах.

2 Загружаем библиотеку для модели и конвертер текстовых признаков в числовые.

```
from sklearn.naive_bayes import MultinomialNB  
from sklearn.feature_extraction.text import CountVectorizer
```

3 Загружаем текстовые данные и преобразуем их в числовые

```
docs = ['This is a sample document.', 'Another document to test.', 'A third sample for testing.']  
vectorizer = CountVectorizer()  
X = vectorizer.fit_transform(docs)
```

# Полиномиальный байесовский классификатор. Пример на Питоне

1 Используем полиномиальный байесовский классификатор для классификации текстовых документов на два класса  $\{0, 1\}$  на основе частоты слов в документах.

2 Загружаем библиотеку для модели и конвертер текстовых признаков в числовые.

```
from sklearn.naive_bayes import MultinomialNB  
from sklearn.feature_extraction.text import CountVectorizer
```

3 Загружаем текстовые данные и преобразуем их в числовые

```
docs = ['This is a sample document.', 'Another document to test.', 'A third sample for testing.']  
vectorizer = CountVectorizer()  
X = vectorizer.fit_transform(docs)
```

4 Открываем модуль классификации и проводим обучение.

```
clf = MultinomialNB()  
clf.fit(X, [0, 1, 0])
```

# Полиномиальный байесовский классификатор. Пример на Питоне

1 Используем полиномиальный байесовский классификатор для классификации текстовых документов на два класса  $\{0, 1\}$  на основе частоты слов в документах.

2 Загружаем библиотеку для модели и конвертер текстовых признаков в числовые.

```
from sklearn.naive_bayes import MultinomialNB  
from sklearn.feature_extraction.text import CountVectorizer
```

3 Загружаем текстовые данные и преобразуем их в числовые

```
docs = ['This is a sample document.', 'Another document to test.', 'A third sample for testing.']  
vectorizer = CountVectorizer()  
X = vectorizer.fit_transform(docs)
```

4 Открываем модуль классификации и проводим обучение.

```
clf = MultinomialNB()  
clf.fit(X, [0, 1, 0])
```

5 Предсказываем класс нового текста.

```
new_doc = ['This is another test document.']  
new_X = vectorizer.transform(new_doc)  
predicted_class = clf.predict(new_X)
```



# Полиномиальный байесовский классификатор. Пример на Питоне

1 Используем полиномиальный байесовский классификатор для классификации текстовых документов на два класса  $\{0, 1\}$  на основе частоты слов в документах.

2 Загружаем библиотеку для модели и конвертер текстовых признаков в числовые.

```
from sklearn.naive_bayes import MultinomialNB  
from sklearn.feature_extraction.text import CountVectorizer
```

3 Загружаем текстовые данные и преобразуем их в числовые

```
docs = ['This is a sample document.', 'Another document to test.', 'A third sample for testing.']  
vectorizer = CountVectorizer()  
X = vectorizer.fit_transform(docs)
```

4 Открываем модуль классификации и проводим обучение.

```
clf = MultinomialNB()  
clf.fit(X, [0, 1, 0])
```

5 Предсказываем класс нового текста.

```
new_doc = ['This is another test document.']  
new_X = vectorizer.transform(new_doc)  
predicted_class = clf.predict(new_X)
```

6 Предсказан класс 1.



# Наивный байесовский классификатор. Преимущества и недостатки

## Преимущества

- 1 Прост в реализации и имеет высокую скорость обработки.

# Наивный байесовский классификатор. Преимущества и недостатки

## Преимущества

- 1 Прост в реализации и имеет высокую скорость обработки.
- 2 Хорошо работает с категориальными данными.

# Наивный байесовский классификатор. Преимущества и недостатки

## Преимущества

- 1 Прост в реализации и имеет высокую скорость обработки.
- 2 Хорошо работает с категориальными данными.
- 3 Небольшой объем входных данных для обучения.

# Наивный байесовский классификатор. Преимущества и недостатки

## Преимущества

- 1 Прост в реализации и имеет высокую скорость обработки.
- 2 Хорошо работает с категориальными данными.
- 3 Небольшой объем входных данных для обучения.
- 4 Если признаки действительно независимы — классификатор оптимален.

# Наивный байесовский классификатор. Преимущества и недостатки

## Преимущества

- 1 Прост в реализации и имеет высокую скорость обработки.
- 2 Хорошо работает с категориальными данными.
- 3 Небольшой объем входных данных для обучения.
- 4 Если признаки действительно независимы — классификатор оптимален.

## Недостатки

- 1 Условие независимости признаков — очень сильное условие.

# Наивный байесовский классификатор. Преимущества и недостатки

## Преимущества

- 1 Прост в реализации и имеет высокую скорость обработки.
- 2 Хорошо работает с категориальными данными.
- 3 Небольшой объем входных данных для обучения.
- 4 Если признаки действительно независимы — классификатор оптимален.

## Недостатки

- 1 Условие независимости признаков — очень сильное условие.
- 2 В задачах, где есть зависимость между признаками объекта, показывает низкое качество классификации.



# Наивный байесовский классификатор. Преимущества и недостатки

## Преимущества

- 1 Прост в реализации и имеет высокую скорость обработки.
- 2 Хорошо работает с категориальными данными.
- 3 Небольшой объем входных данных для обучения.
- 4 Если признаки действительно независимы — классификатор оптимален.

## Недостатки

- 1 Условие независимости признаков — очень сильное условие.
- 2 В задачах, где есть зависимость между признаками объекта, показывает низкое качество классификации.
- 3 При отсутствии в выборке пары (класс, признак) требуется сглаживание по Лапласу.



- 1 Рассмотрим результаты прогнозирования классов для рассмотренных ранее мультиклассовых классификаторов:

- 1 Рассмотрим результаты прогнозирования классов для рассмотренных ранее мультиклассовых классификаторов:  
**Мультиклассовый SVM. Один против остальных**

- 1 Рассмотрим результаты прогнозирования классов для рассмотренных ранее мультиклассовых классификаторов:

**Мультиклассовый SVM. Один против остальных**

**Мультиклассовый SVM. Каждый против каждого**

1 Рассмотрим результаты прогнозирования классов для рассмотренных ранее мультиклассовых классификаторов:

**Мультиклассовый SVM. Один против остальных**

**Мультиклассовый SVM. Каждый против каждого**

**Классификатор к-ближайших соседей**

1 Рассмотрим результаты прогнозирования классов для рассмотренных ранее мультиклассовых классификаторов:

**Мультиклассовый SVM. Один против остальных**

**Мультиклассовый SVM. Каждый против каждого**

**Классификатор к-ближайших соседей**

**Алгоритм CART**

1 Рассмотрим результаты прогнозирования классов для рассмотренных ранее мультиклассовых классификаторов:

**Мультиклассовый SVM. Один против остальных**

**Мультиклассовый SVM. Каждый против каждого**

**Классификатор к-ближайших соседей**

**Алгоритм CART**

**Random Forest**



1 Рассмотрим результаты прогнозирования классов для рассмотренных ранее мультиклассовых классификаторов:

**Мультиклассовый SVM. Один против остальных**

**Мультиклассовый SVM. Каждый против каждого**

**Классификатор k-ближайших соседей**

**Алгоритм CART**

**Random Forest**

**Гауссовский байесовский классификатор**

- 1 Рассмотрим результаты прогнозирования классов для рассмотренных ранее мультиклассовых классификаторов:

Мультиклассовый SVM. Один против остальных

Мультиклассовый SVM. Каждый против каждого

Классификатор  $k$ -ближайших соседей

Алгоритм CART

Random Forest

Гауссовский байесовский классификатор

- 2 Тестовая выборка (Ирисы Фишера) — рисунок 13:а. Взяты два первых признака: *sepal length* и *sepal width*.

- 1 Рассмотрим результаты прогнозирования классов для рассмотренных ранее мультиклассовых классификаторов:
  - Мультиклассовый SVM. Один против остальных
  - Мультиклассовый SVM. Каждый против каждого
  - Классификатор к-ближайших соседей
  - Алгоритм CART
  - Random Forest
  - Гауссовский байесовский классификатор
- 2 Тестовая выборка (**Ирисы Фишера**) — рисунок 13:а. Взяты два первых признака: *sepal length* и *sepal width*.
- 3 Используя библиотеку **Yellowbrick** построим диаграммы ошибок прогнозирования классов для каждого классификатора.

- 1 Рассмотрим результаты прогнозирования классов для рассмотренных ранее мультиклассовых классификаторов:

Мультиклассовый SVM. Один против остальных

Мультиклассовый SVM. Каждый против каждого

Классификатор k-ближайших соседей

Алгоритм CART

Random Forest

Гауссовский байесовский классификатор

- 2 Тестовая выборка (Ирисы Фишера) — рисунок 13:а. Взяты два первых признака: *sepal length* и *sepal width*.

- 3 Используя библиотеку **Yellowbrick** построим диаграммы ошибок прогнозирования классов для каждого классификатора.

Диаграмма ошибок прогнозирования классов позволит быстро понять, насколько хорошо классификатор прогнозирует правильные классы.

- 1 Рассмотрим результаты прогнозирования классов для рассмотренных ранее мультиклассовых классификаторов:

Мультиклассовый SVM. Один против остальных

Мультиклассовый SVM. Каждый против каждого

Классификатор к-ближайших соседей

Алгоритм CART

Random Forest

Гауссовский байесовский классификатор

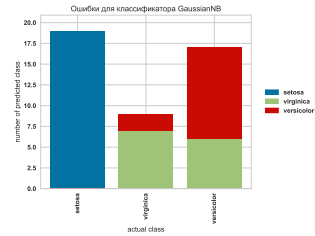
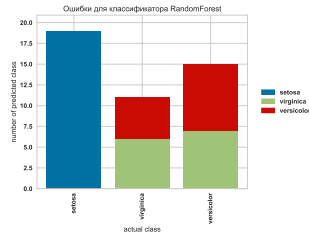
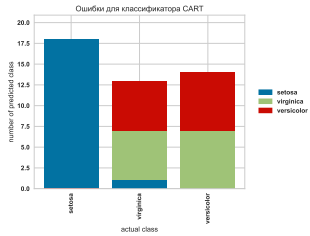
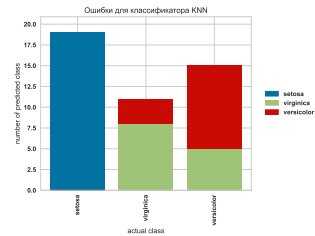
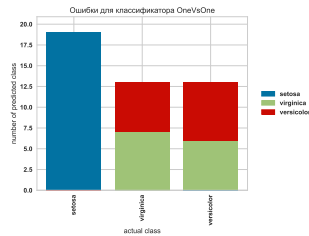
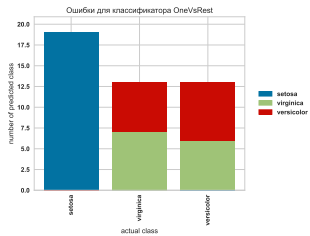
- 2 Тестовая выборка (Ирисы Фишера) — рисунок 13:а. Взяты два первых признака: *sepal length* и *sepal width*.
- 3 Используя библиотеку **Yellowbrick** построим диаграммы ошибок прогнозирования классов для каждого классификатора.

Диаграмма ошибок прогнозирования классов позволит быстро понять, насколько хорошо классификатор прогнозирует правильные классы.

- i Пример для классификатора **CART**: (Xtrain, ytrain) и (Xtest, ytest) — тренировочное и тестовое множества.

```
from sklearn.tree import DecisionTreeClassifier
from yellowbrick.classifier import ClassPredictionError
#####
classes = ["setosa", "virginica", "versicolor"]
visualizer = ClassPredictionError(DecisionTreeClassifier(), classes=classes)
visualizer.fit(Xtrain, ytrain)
visualizer.score(Xtest, ytest)
visualizer.show()
```









## Определения

Имеется объекты двух классов  $y \in \{0, 1\}$ ,  $y = 1$  – «положительный класс»,  $y = 0$  – «отрицательный класс» и классификатор, определяющий принадлежность объекта  $X$  одному из классов  $a(X) \in \{0, 1\}$ .

*Таблица 3. Основные величины классификации*

	$y = 1$	$y = 0$
$a(X) = 1$	True Positive — TP (истинно-положительный)	False Positive — FP (ложно-положительный)
$a(X) = 0$	False Negative — FN (ложно-отрицательный)	True Negative — TN (истинно-отрицательный)

## Определения

Имеется объекты двух классов  $y \in \{0, 1\}$ ,  $y = 1$  – «положительный класс»,  $y = 0$  – «отрицательный класс» и классификатор, определяющий принадлежность объекта  $X$  одному из классов  $a(X) \in \{0, 1\}$ .

*Таблица 3. Основные величины классификации*

	$y = 1$	$y = 0$
$a(X) = 1$	True Positive — TP (истинно-положительный)	False Positive — FP (ложно-положительный)
$a(X) = 0$	False Negative — FN (ложно-отрицательный)	True Negative — TN (истинно-отрицательный)

TP — число положительных объектов, классифицированных как положительные.

## Определения

Имеется объекты двух классов  $y \in \{0, 1\}$ ,  $y = 1$  — «положительный класс»,  $y = 0$  — «отрицательный класс» и классификатор, определяющий принадлежность объекта  $X$  одному из классов  $a(X) \in \{0, 1\}$ .

*Таблица 3. Основные величины классификации*

	$y = 1$	$y = 0$
$a(X) = 1$	True Positive — TP (истинно-положительный)	False Positive — FP (ложно-положительный)
$a(X) = 0$	False Negative — FN (ложно-отрицательный)	True Negative — TN (истинно-отрицательный)

TP — число положительных объектов, классифицированных как положительные.

TN — число отрицательных объектов, классифицированных как отрицательные.

## Определения

Имеется объекты двух классов  $y \in \{0, 1\}$ ,  $y = 1$  – «положительный класс»,  $y = 0$  – «отрицательный класс» и классификатор, определяющий принадлежность объекта  $X$  одному из классов  $a(X) \in \{0, 1\}$ .

Таблица 3. Основные величины классификации

	$y = 1$	$y = 0$
$a(X) = 1$	True Positive — TP (истинно-положительный)	False Positive — FP (ложно-положительный)
$a(X) = 0$	False Negative — FN (ложно-отрицательный)	True Negative — TN (истинно-отрицательный)

- TP — число положительных объектов, классифицированных как положительные.
- TN — число отрицательных объектов, классифицированных как отрицательные.
- FP — число отрицательных объектов, классифицированных как положительные (*ошибка классификации I рода*).

## Определения

Имеется объекты двух классов  $y \in \{0, 1\}$ ,  $y = 1$  – «положительный класс»,  $y = 0$  – «отрицательный класс» и классификатор, определяющий принадлежность объекта  $X$  одному из классов  $a(X) \in \{0, 1\}$ .

Таблица 3. Основные величины классификации

	$y = 1$	$y = 0$
$a(X) = 1$	True Positive — TP (истинно-положительный)	False Positive — FP (ложно-положительный)
$a(X) = 0$	False Negative — FN (ложно-отрицательный)	True Negative — TN (истинно-отрицательный)

- TP — число положительных объектов, классифицированных как положительные.
- TN — число отрицательных объектов, классифицированных как отрицательные.
- FP — число отрицательных объектов, классифицированных как положительные (ошибка классификации I рода).
- FN — число положительных объектов, классифицированных как отрицательные (ошибка классификации II рода).

## Определения

Имеется объекты двух классов  $y \in \{0, 1\}$ ,  $y = 1$  – «положительный класс»,  $y = 0$  – «отрицательный класс» и классификатор, определяющий принадлежность объекта  $X$  одному из классов  $a(X) \in \{0, 1\}$ .

Таблица 3. Основные величины классификации

	$y = 1$	$y = 0$
$a(X) = 1$	True Positive — TP (истинно-положительный)	False Positive — FP (ложно-положительный)
$a(X) = 0$	False Negative — FN (ложно-отрицательный)	True Negative — TN (истинно-отрицательный)

- TP — число положительных объектов, классифицированных как положительные.
- TN — число отрицательных объектов, классифицированных как отрицательные.
- FP — число отрицательных объектов, классифицированных как положительные (ошибка классификации I рода).
- FN — число положительных объектов, классифицированных как отрицательные (ошибка классификации II рода).

## Пример<sup>5</sup>

<sup>5</sup>Источник: блог А.Г. Дьяконова (<https://dyakonov.org/>)

## Определения

Имеется объекты двух классов  $y \in \{0, 1\}$ ,  $y = 1$  – «положительный класс»,  $y = 0$  – «отрицательный класс» и классификатор, определяющий принадлежность объекта  $X$  одному из классов  $a(X) \in \{0, 1\}$ .

Таблица 3. Основные величины классификации

	$y = 1$	$y = 0$
$a(X) = 1$	True Positive — TP (истинно-положительный)	False Positive — FP (ложно-положительный)
$a(X) = 0$	False Negative — FN (ложно-отрицательный)	True Negative — TN (истинно-отрицательный)

- TP — число положительных объектов, классифицированных как положительные.
- TN — число отрицательных объектов, классифицированных как отрицательные.
- FP — число отрицательных объектов, классифицированных как положительные (ошибка классификации I рода).
- FN — число положительных объектов, классифицированных как отрицательные (ошибка классификации II рода).

## Пример<sup>5</sup>

Студент сдает экзамен.

<sup>5</sup>Источник: блог А.Г. Дьяконова (<https://dyakonov.org/>)

## Определения

Имеется объекты двух классов  $y \in \{0, 1\}$ ,  $y = 1$  – «положительный класс»,  $y = 0$  – «отрицательный класс» и классификатор, определяющий принадлежность объекта  $X$  одному из классов  $a(X) \in \{0, 1\}$ .

Таблица 3. Основные величины классификации

	$y = 1$	$y = 0$
$a(X) = 1$	True Positive — TP (истинно-положительный)	False Positive — FP (ложно-положительный)
$a(X) = 0$	False Negative — FN (ложно-отрицательный)	True Negative — TN (истинно-отрицательный)

- TP — число положительных объектов, классифицированных как положительные.
- TN — число отрицательных объектов, классифицированных как отрицательные.
- FP — число отрицательных объектов, классифицированных как положительные (ошибка классификации I рода).
- FN — число положительных объектов, классифицированных как отрицательные (ошибка классификации II рода).

## Пример<sup>5</sup>

Студент сдает экзамен.

Класс 1 — студент учил и знает («положительный» студент), иначе — класс 0 («отрицательный» студент).



## Определения

Имеется объекты двух классов  $y \in \{0, 1\}$ ,  $y = 1$  – «положительный класс»,  $y = 0$  – «отрицательный класс» и классификатор, определяющий принадлежность объекта  $X$  одному из классов  $a(X) \in \{0, 1\}$ .

Таблица 3. Основные величины классификации

	$y = 1$	$y = 0$
$a(X) = 1$	True Positive — TP (истинно-положительный)	False Positive — FP (ложно-положительный)
$a(X) = 0$	False Negative — FN (ложно-отрицательный)	True Negative — TN (истинно-отрицательный)

- TP — число положительных объектов, классифицированных как положительные.
- TN — число отрицательных объектов, классифицированных как отрицательные.
- FP — число отрицательных объектов, классифицированных как положительные (ошибка классификации I рода).
- FN — число положительных объектов, классифицированных как отрицательные (ошибка классификации II рода).

## Пример<sup>5</sup>

Студент сдает экзамен.

Класс 1 — студент учил и знает («положительный» студент), иначе — класс 0 («отрицательный» студент).

Классификатор – экзаменатор. Зачет — в класс 1, пересдача — в класс 0.

<sup>5</sup>Источник: блог А.Г. Дьяконова (<https://dyakonov.org/>)

## Определения

Имеется объекты двух классов  $y \in \{0, 1\}$ ,  $y = 1$  – «положительный класс»,  $y = 0$  – «отрицательный класс» и классификатор, определяющий принадлежность объекта  $X$  одному из классов  $a(X) \in \{0, 1\}$ .

Таблица 3. Основные величины классификации

	$y = 1$	$y = 0$
$a(X) = 1$	True Positive — TP (истинно-положительный)	False Positive — FP (ложно-положительный)
$a(X) = 0$	False Negative — FN (ложно-отрицательный)	True Negative — TN (истинно-отрицательный)

- TP — число положительных объектов, классифицированных как положительные.
- TN — число отрицательных объектов, классифицированных как отрицательные.
- FP — число отрицательных объектов, классифицированных как положительные (ошибка классификации I рода).
- FN — число положительных объектов, классифицированных как отрицательные (ошибка классификации II рода).

## Пример<sup>5</sup>

- Студент сдает экзамен.
- Класс 1 — студент учил и знает («положительный» студент), иначе — класс 0 («отрицательный» студент).
- Классификатор – экзаменатор. Зачет — в класс 1, пересдача — в класс 0.
- Ошибка I рода — «не учил, но сдал», ошибка II рода — «учил, но не сдал».

<sup>5</sup>Источник: блог А.Г. Дьяконова (<https://dyakonov.org/>)



## Матрица ошибок

Таблицу 3 с результатами классификации обычно записывают в виде матрицы — *матрица ошибок* (Confusion matrix).

Иногда говорят *матрица несоответствий*.

$$CM = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

## Accuracy

## Матрица ошибок

Таблицу 3 с результатами классификации обычно записывают в виде матрицы — *матрица ошибок* (Confusion matrix).

Иногда говорят *матрица несоответствий*.

$$CM = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

## Accuracy

*Accuracy* (Точность) — доля правильных ответов алгоритма:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (32)$$

## Матрица ошибок

Таблицу 3 с результатами классификации обычно записывают в виде матрицы — *матрица ошибок* (Confusion matrix). Иногда говорят *матрица несоответствий*.

$$CM = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

## Accuracy

*Accuracy* (Точность) — доля правильных ответов алгоритма:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (32)$$

При оценке кластеризации использовался аналогичный подход — *индекс Рэнда*.

## Матрица ошибок

Таблицу 3 с результатами классификации обычно записывают в виде матрицы — *матрица ошибок* (Confusion matrix). Иногда говорят *матрица несоответствий*.

$$CM = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

## Accuracy

*Accuracy* (Точность) — доля правильных ответов алгоритма:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (32)$$

При оценке кластеризации использовался аналогичный подход — *индекс Рэнда*.

Метрику можно использовать для многоклассовой классификации.

## Матрица ошибок

Таблицу 3 с результатами классификации обычно записывают в виде матрицы — *матрица ошибок* (Confusion matrix). Иногда говорят *матрица несоответствий*.

$$CM = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

## Accuracy

*Accuracy* (Точность) — доля правильных ответов алгоритма:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (32)$$

При оценке кластеризации использовался аналогичный подход — *индекс Рэнда*.

Метрику можно использовать для многоклассовой классификации.

Плохо работает для сильно несбалансированных классов.



## Матрица ошибок

Таблицу 3 с результатами классификации обычно записывают в виде матрицы — *матрица ошибок* (Confusion matrix). Иногда говорят *матрица несоответствий*.

$$CM = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

## Accuracy

*Accuracy* (Точность) — доля правильных ответов алгоритма:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (32)$$

При оценке кластеризации использовался аналогичный подход — *индекс Рэнда*.

Метрику можно использовать для многоклассовой классификации.

Плохо работает для сильно несбалансированных классов.

Пример<sup>6</sup>:

<sup>6</sup>Источник: <https://habr.com/ru/company/ods/blog/328372/>

## Матрица ошибок

Таблицу 3 с результатами классификации обычно записывают в виде матрицы — *матрица ошибок* (Confusion matrix).

Иногда говорят *матрица несоответствий*.

$$CM = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

## Accuracy

*Accuracy* (Точность) — доля правильных ответов алгоритма:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (32)$$

При оценке кластеризации использовался аналогичный подход — *индекс Рэнда*.

Метрику можно использовать для многоклассовой классификации.

Плохо работает для сильно несбалансированных классов.

Пример<sup>6</sup>:

Имеется: 100 не-спам писем («положительный» класс), 90 — классификатор определил верно ( $TP = 90$ ,  $FN = 10$ ),

<sup>6</sup>Источник: <https://habr.com/ru/company/ods/blog/328372/>

## Матрица ошибок

Таблицу 3 с результатами классификации обычно записывают в виде матрицы — *матрица ошибок* (Confusion matrix). Иногда говорят *матрица несоответствий*.

$$CM = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

## Accuracy

*Accuracy* (Точность) — доля правильных ответов алгоритма:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (32)$$

При оценке кластеризации использовался аналогичный подход — *индекс Рэнда*.

Метрику можно использовать для многоклассовой классификации.

Плохо работает для сильно несбалансированных классов.

Пример<sup>6</sup>:

Имеется: 100 не-спам писем («положительный» класс), 90 — классификатор определил верно ( $TP = 90$ ,  $FN = 10$ ), 10 спам-писем («отрицательный» класс), 5 — классификатор также определил верно ( $TN = 5$ ,  $FP = 5$ ).

<sup>6</sup>Источник: <https://habr.com/ru/company/ods/blog/328372/>

## Матрица ошибок

Таблицу 3 с результатами классификации обычно записывают в виде матрицы — *матрица ошибок* (Confusion matrix). Иногда говорят *матрица несоответствий*.

$$CM = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

## Accuracy

*Accuracy* (Точность) — доля правильных ответов алгоритма:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (32)$$

При оценке кластеризации использовался аналогичный подход — *индекс Рэнда*.

Метрику можно использовать для многоклассовой классификации.

Плохо работает для сильно несбалансированных классов.

Пример<sup>6</sup>:

Имеется: 100 не-спам писем («положительный» класс), 90 — классификатор определил верно ( $TP = 90$ ,  $FN = 10$ ), 10 спам-писем («отрицательный» класс), 5 — классификатор также определил верно ( $TN = 5$ ,  $FP = 5$ ).

Тогда

$$Accuracy = \frac{90 + 5}{90 + 5 + 5 + 10} \approx 0.8636. \quad (33)$$

<sup>6</sup>Источник: <https://habr.com/ru/company/ods/blog/328372/>

## Матрица ошибок

Таблицу 3 с результатами классификации обычно записывают в виде матрицы — *матрица ошибок* (Confusion matrix). Иногда говорят *матрица несоответствий*.

$$CM = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

## Accuracy

*Accuracy* (Точность) — доля правильных ответов алгоритма:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (32)$$

При оценке кластеризации использовался аналогичный подход — *индекс Рэнда*.

Метрику можно использовать для многоклассовой классификации.

Плохо работает для сильно несбалансированных классов.

Пример<sup>6</sup>:

Имеется: 100 не-спам писем («положительный» класс), 90 — классификатор определил верно ( $TP = 90$ ,  $FN = 10$ ),  
10 спам-писем («отрицательный» класс), 5 — классификатор также определил верно ( $TN = 5$ ,  $FP = 5$ ).

Тогда

$$Accuracy = \frac{90 + 5}{90 + 5 + 5 + 10} \approx 0.8636. \quad (33)$$

Будем предсказывать все письма как не-спам ( $TP = 100$ ,  $FN = 0$ ,  $TN = 0$ ,  $FP = 10$ ), тогда:

$$Accuracy^* = \frac{100 + 0}{100 + 0 + 10 + 0} \approx 0.9091 > Accuracy. \quad (34)$$

<sup>6</sup>Источник: <https://habr.com/ru/company/ods/blog/328372/>



? Как оценить точность в случае дисбаланса классов?

? Как оценить точность в случае дисбаланса классов?

Есть специальный аналог — *сбалансированная точность* (*Balanced Accuracy, BA*):

$$BA = \frac{1}{2} \left( \frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right). \quad (35)$$



? Как оценить точность в случае дисбаланса классов?

Есть специальный аналог — *сбалансированная точность* (*Balanced Accuracy, BA*):

$$BA = \frac{1}{2} \left( \frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right). \quad (35)$$

Рассмотри предыдущий пример:

? Как оценить точность в случае дисбаланса классов?

Есть специальный аналог — *сбалансированная точность* (*Balanced Accuracy, BA*):

$$BA = \frac{1}{2} \left( \frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right). \quad (35)$$

Рассмотри предыдущий пример:

1 В первом случае:  $TP = 90$ ,  $FN = 10$ ,  $TN = 5$ ,  $FP = 5$ . Из (35) получаем:

$$BA = \frac{1}{2} \left( \frac{90}{90 + 10} + \frac{5}{5 + 5} \right) = 0.7. \quad (36)$$

? Как оценить точность в случае дисбаланса классов?

Есть специальный аналог — *сбалансированная точность* (*Balanced Accuracy, BA*):

$$BA = \frac{1}{2} \left( \frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right). \quad (35)$$

Рассмотри предыдущий пример:

1 В первом случае:  $TP = 90$ ,  $FN = 10$ ,  $TN = 5$ ,  $FP = 5$ . Из (35) получаем:

$$BA = \frac{1}{2} \left( \frac{90}{90 + 10} + \frac{5}{5 + 5} \right) = 0.7. \quad (36)$$

1 Во втором случае:  $TP = 100$ ,  $FN = 0$ ,  $TN = 0$ ,  $FP = 10$ . Тогда из (35):

$$BA^* = \frac{1}{2} \left( \frac{100}{100 + 0} + \frac{0}{0 + 10} \right) = 0.5 < BA. \quad (37)$$

? Как оценить точность в случае дисбаланса классов?

Есть специальный аналог — *сбалансированная точность* (*Balanced Accuracy, BA*):

$$BA = \frac{1}{2} \left( \frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right). \quad (35)$$

Рассмотри предыдущий пример:

1 В первом случае:  $TP = 90$ ,  $FN = 10$ ,  $TN = 5$ ,  $FP = 5$ . Из (35) получаем:

$$BA = \frac{1}{2} \left( \frac{90}{90 + 10} + \frac{5}{5 + 5} \right) = 0.7. \quad (36)$$

1 Во втором случае:  $TP = 100$ ,  $FN = 0$ ,  $TN = 0$ ,  $FP = 10$ . Тогда из (35):

$$BA^* = \frac{1}{2} \left( \frac{100}{100 + 0} + \frac{0}{0 + 10} \right) = 0.5 < BA. \quad (37)$$

i Если в бинарной задаче классификации представителей двух классов примерно поровну —  $TP + FN \approx TN + FP$ , то  $BA \approx \text{Accuracy}$ .



*Точность* (Precision<sup>a</sup>) — доля объектов, определенных классификатором положительными и при этом действительно являющимися положительными.

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (38)$$

---

<sup>a</sup>перевод совпадает с Accuracy

*Точность* (Precision<sup>a</sup>) — доля объектов, определенных классификатором положительными и при этом действительно являющимися положительными.

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (38)$$

- 1 Мера не дает записать все объекты в один класс, так как в этом случае увеличивается FP.

---

<sup>a</sup>перевод совпадает с Ассигасу

*Точность* (Precision<sup>a</sup>) — доля объектов, определенных классификатором положительными и при этом действительно являющимися положительными.

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (38)$$

- 1 Мера не дает записать все объекты в один класс, так как в этом случае увеличивается FP.
- 2 Мера позволяет отличать этот класс от других классов.

---

<sup>a</sup>перевод совпадает с Accuracy



*Точность* (Precision<sup>a</sup>) — доля объектов, определенных классификатором положительными и при этом действительно являющимися положительными.

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (38)$$

- 1 Мера не дает записать все объекты в один класс, так как в этом случае увеличивается FP.
- 2 Мера позволяет отличать этот класс от других классов.
- 3 Применима в условиях несбалансированных выборок.

---

<sup>a</sup>перевод совпадает с Ассигасу

*Точность* (Precision<sup>a</sup>) — доля объектов, определенных классификатором положительными и при этом действительно являющимися положительными.

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (38)$$

- 1 Мера не дает записать все объекты в один класс, так как в этом случае увеличивается FP.
- 2 Мера позволяет отличать этот класс от других классов.
- 3 Применима в условиях несбалансированных выборок.

Для предыдущего примера (Assigasy):

$$CM = \begin{pmatrix} 90 & 5 \\ 10 & 5 \end{pmatrix}, \text{ Precision} = \frac{90}{90 + 5} \approx 0.9473, \quad CM^* = \begin{pmatrix} 100 & 10 \\ 0 & 0 \end{pmatrix}, \text{ Precision}^* = \frac{100}{100 + 10} \approx 0.9091 < \text{Precision}.$$

---

<sup>a</sup>перевод совпадает с Assigasy

*Точность* (Precision<sup>a</sup>) — доля объектов, определенных классификатором положительными и при этом действительно являющимися положительными.

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (38)$$

- 1 Мера не дает записать все объекты в один класс, так как в этом случае увеличивается FP.
- 2 Мера позволяет отличать этот класс от других классов.
- 3 Применима в условиях несбалансированных выборок.

Для предыдущего примера (Assigasy):

$$CM = \begin{pmatrix} 90 & 5 \\ 10 & 5 \end{pmatrix}, \text{Precision} = \frac{90}{90 + 5} \approx 0.9473, \quad CM^* = \begin{pmatrix} 100 & 10 \\ 0 & 0 \end{pmatrix}, \text{Precision}^* = \frac{100}{100 + 10} \approx 0.9091 < \text{Precision}.$$

При записи в один класс (не-спам) точность уменьшилась.

---

<sup>a</sup>перевод совпадает с Assigasy

*Точность* (Precision<sup>a</sup>) — доля объектов, определенных классификатором положительными и при этом действительно являющимися положительными.

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (38)$$

- 1 Мера не дает записать все объекты в один класс, так как в этом случае увеличивается FP.
- 2 Мера позволяет отличать этот класс от других классов.
- 3 Применима в условиях несбалансированных выборок.

Для предыдущего примера (Assigasy):

$$CM = \begin{pmatrix} 90 & 5 \\ 10 & 5 \end{pmatrix}, \text{Precision} = \frac{90}{90 + 5} \approx 0.9473, \quad CM^* = \begin{pmatrix} 100 & 10 \\ 0 & 0 \end{pmatrix}, \text{Precision}^* = \frac{100}{100 + 10} \approx 0.9091 < \text{Precision}.$$

При записи в один класс (не-спам) точность уменьшилась.

- 4 Когда классификатор принимает много неверных Positive решений, это увеличивает знаменатель в (38) и снижает значение Precision.

---

<sup>a</sup>перевод совпадает с Assigasy

*Точность* (Precision<sup>a</sup>) — доля объектов, определенных классификатором положительными и при этом действительно являющимися положительными.

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (38)$$

- 1 Мера не дает записать все объекты в один класс, так как в этом случае увеличивается FP.
- 2 Мера позволяет отличать этот класс от других классов.
- 3 Применима в условиях несбалансированных выборок.

Для предыдущего примера (Assigasy):

$$CM = \begin{pmatrix} 90 & 5 \\ 10 & 5 \end{pmatrix}, \text{Precision} = \frac{90}{90 + 5} \approx 0.9473, \quad CM^* = \begin{pmatrix} 100 & 10 \\ 0 & 0 \end{pmatrix}, \text{Precision}^* = \frac{100}{100 + 10} \approx 0.9091 < \text{Precision}.$$

При записи в один класс (не-спам) точность уменьшилась.

- 4 Когда классификатор принимает много неверных Positive решений, это увеличивает знаменатель в (38) и снижает значение Precision.
- 5 Из (38) очевидно, что значение Precision высоко, если:

---

<sup>a</sup>перевод совпадает с Assigasy

*Точность* (Precision<sup>a</sup>) — доля объектов, определенных классификатором положительными и при этом действительно являющимися положительными.

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (38)$$

- 1 Мера не дает записать все объекты в один класс, так как в этом случае увеличивается FP.
- 2 Мера позволяет отличать этот класс от других классов.
- 3 Применима в условиях несбалансированных выборок.

Для предыдущего примера (Assigasy):

$$CM = \begin{pmatrix} 90 & 5 \\ 10 & 5 \end{pmatrix}, \text{Precision} = \frac{90}{90 + 5} \approx 0.9473, \quad CM^* = \begin{pmatrix} 100 & 10 \\ 0 & 0 \end{pmatrix}, \text{Precision}^* = \frac{100}{100 + 10} \approx 0.9091 < \text{Precision}.$$

При записи в один класс (не-спам) точность уменьшилась.

- 4 Когда классификатор принимает много неверных Positive решений, это увеличивает знаменатель в (38) и снижает значение Precision.
- 5 Из (38) очевидно, что значение Precision высоко, если:
  - ✓ классификатор делает много корректных предсказаний класса Positive (максимизирует True Positive метрику);

---

<sup>a</sup>перевод совпадает с Assigasy

*Точность* (Precision<sup>a</sup>) — доля объектов, определенных классификатором положительными и при этом действительно являющимися положительными.

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (38)$$

- 1 Мера не дает записать все объекты в один класс, так как в этом случае увеличивается FP.
- 2 Мера позволяет отличать этот класс от других классов.
- 3 Применима в условиях несбалансированных выборок.

Для предыдущего примера (Assigasy):

$$CM = \begin{pmatrix} 90 & 5 \\ 10 & 5 \end{pmatrix}, \text{Precision} = \frac{90}{90 + 5} \approx 0.9473, \quad CM^* = \begin{pmatrix} 100 & 10 \\ 0 & 0 \end{pmatrix}, \text{Precision}^* = \frac{100}{100 + 10} \approx 0.9091 < \text{Precision}.$$

При записи в один класс (не-спам) точность уменьшилась.

- 4 Когда классификатор принимает много неверных Positive решений, это увеличивает знаменатель в (38) и снижает значение Precision.
- 5 Из (38) очевидно, что значение Precision высоко, если:
  - ✓ классификатор делает много корректных предсказаний класса Positive (максимизирует True Positive метрику);
  - ✓ классификатор делает меньше неверных Positive предсказаний (минимизирует False Positive).

<sup>a</sup>перевод совпадает с Assigasy

*Точность* (Precision<sup>a</sup>) — доля объектов, определенных классификатором положительными и при этом действительно являющимися положительными.

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (38)$$

- 1 Мера не дает записать все объекты в один класс, так как в этом случае увеличивается FP.
- 2 Мера позволяет отличать этот класс от других классов.
- 3 Применима в условиях несбалансированных выборок.

Для предыдущего примера (Assigasy):

$$CM = \begin{pmatrix} 90 & 5 \\ 10 & 5 \end{pmatrix}, \text{Precision} = \frac{90}{90 + 5} \approx 0.9473, \quad CM^* = \begin{pmatrix} 100 & 10 \\ 0 & 0 \end{pmatrix}, \text{Precision}^* = \frac{100}{100 + 10} \approx 0.9091 < \text{Precision}.$$

При записи в один класс (не-спам) точность уменьшилась.

- 4 Когда классификатор принимает много неверных Positive решений, это увеличивает знаменатель в (38) и снижает значение Precision.
- 5 Из (38) очевидно, что значение Precision высоко, если:
  - ✓ классификатор делает много корректных предсказаний класса Positive (максимизирует True Positive метрику);
  - ✓ классификатор делает меньше неверных Positive предсказаний (минимизирует False Positive).
- i Если Precision высока, **можно доверять** решению классификатора по определению очередного элемента как Positive.

<sup>a</sup>перевод совпадает с Assigasy



**Точность** (Precision<sup>a</sup>) — доля объектов, определенных классификатором положительными и при этом действительно являющимися положительными.

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (38)$$

- 1 Мера не дает записать все объекты в один класс, так как в этом случае увеличивается FP.
- 2 Мера позволяет отличать этот класс от других классов.
- 3 Применима в условиях несбалансированных выборок.

Для предыдущего примера (Assigasy):

$$CM = \begin{pmatrix} 90 & 5 \\ 10 & 5 \end{pmatrix}, \text{Precision} = \frac{90}{90 + 5} \approx 0.9473, \quad CM^* = \begin{pmatrix} 100 & 10 \\ 0 & 0 \end{pmatrix}, \text{Precision}^* = \frac{100}{100 + 10} \approx 0.9091 < \text{Precision}.$$

При записи в один класс (не-спам) точность уменьшилась.

- 4 Когда классификатор принимает много неверных Positive решений, это увеличивает знаменатель в (38) и снижает значение Precision.
- 5 Из (38) очевидно, что значение Precision высоко, если:
  - ✓ классификатор делает много корректных предсказаний класса Positive (максимизирует True Positive метрику);
  - ✓ классификатор делает меньше неверных Positive предсказаний (минимизирует False Positive).
- i Если Precision высока, **можно доверять** решению классификатора по определению очередного элемента как Positive.
- i Таким образом, метрика Precision помогает узнать, **насколько точен** классификатор, когда он говорит, что элемент принадлежит классу Positive.

<sup>a</sup>перевод совпадает с Assigasy



*Полнота, чувствительность, sensitivity, TPR (true positive rate)* (Recall) — доля истинно положительных классификаций. Показывает, какая доля объектов, реально относящихся к положительному классу, правильно классифицирована.

$$\text{Recall} = \frac{TP}{TP + FN}. \quad (39)$$

Recall — отношение числа Positive элементов, корректно классифицированных как Positive, к общему количеству Positive объектов.

*Полнота, чувствительность, sensitivity, TPR (true positive rate)* (Recall) — доля истинно положительных классификаций. Показывает, какая доля объектов, реально относящихся к положительному классу, правильно классифицирована.

$$\text{Recall} = \frac{TP}{TP + FN}. \quad (39)$$

Recall — отношение числа Positive элементов, корректно классифицированных как Positive, к общему количеству Positive объектов.

- 1 Применима в условиях несбалансированных выборок.

*Полнота, чувствительность, sensitivity, TPR (true positive rate)* (Recall) — доля истинно положительных классификаций. Показывает, какая доля объектов, реально относящихся к положительному классу, правильно классифицирована.

$$\text{Recall} = \frac{TP}{TP + FN}. \quad (39)$$

Recall — отношение числа Positive элементов, корректно классифицированных как Positive, к общему количеству Positive объектов.

1 Применима в условиях несбалансированных выборок.

Для примера с Ассигасу (матрица ошибок СМ):

$$\text{Recall} = \frac{90}{90 + 10} = 0.9$$

*Полнота, чувствительность, sensitivity, TPR (true positive rate)* (Recall) — доля истинно положительных классификаций. Показывает, какая доля объектов, реально относящихся к положительному классу, правильно классифицирована.

$$\text{Recall} = \frac{TP}{TP + FN}. \quad (39)$$

Recall — отношение числа Positive элементов, корректно классифицированных как Positive, к общему количеству Positive объектов.

1 Применима в условиях несбалансированных выборок.

Для примера с Ассигасу (матрица ошибок СМ):

$$\text{Recall} = \frac{90}{90 + 10} = 0.9$$

2 Метрика Recall измеряет способность классификатора определять объекты, относящиеся к классу Positive. Чем выше Recall, тем больше Positive объектов было найдено.

*Полнота, чувствительность, sensitivity, TPR (true positive rate)* (Recall) — доля истинно положительных классификаций. Показывает, какая доля объектов, реально относящихся к положительному классу, правильно классифицирована.

$$\text{Recall} = \frac{TP}{TP + FN}. \quad (39)$$

Recall — отношение числа Positive элементов, корректно классифицированных как Positive, к общему количеству Positive объектов.

1 Применима в условиях несбалансированных выборок.

Для примера с Ассигасу (матрица ошибок СМ):

$$\text{Recall} = \frac{90}{90 + 10} = 0.9$$

2 Метрика Recall измеряет способность классификатора определять объекты, относящиеся к классу Positive. Чем выше Recall, тем больше Positive объектов было найдено.

3 Значение  $\text{Recall} \in [0, 1]$  показывает процент Positive объектов, которые верно классифицированы. Например, если имеется 100 элементов Positive и  $\text{Recall} = 0.72$ , получается, что классификатор корректно определил 72% объектов класса Positive.

*Полнота, чувствительность, sensitivity, TPR (true positive rate)* (Recall) — доля истинно положительных классификаций. Показывает, какая доля объектов, реально относящихся к положительному классу, правильно классифицирована.

$$\text{Recall} = \frac{TP}{TP + FN}. \quad (39)$$

Recall — отношение числа Positive элементов, корректно классифицированных как Positive, к общему количеству Positive объектов.

1 Применима в условиях несбалансированных выборок.

Для примера с Ассигасу (матрица ошибок СМ):

$$\text{Recall} = \frac{90}{90 + 10} = 0.9$$

2 Метрика Recall измеряет способность классификатора определять объекты, относящиеся к классу Positive. Чем выше Recall, тем больше Positive объектов было найдено.

3 Значение  $\text{Recall} \in [0, 1]$  показывает процент Positive объектов, которые верно классифицированы. Например, если имеется 100 элементов Positive и  $\text{Recall} = 0.72$ , получается, что классификатор корректно определил 72% объектов класса Positive.

4 Метрика Recall не зависит от того, как предсказываются Negative элементы, в отличие от Precision.



*Полнота, чувствительность, sensitivity, TPR (true positive rate)* (Recall) — доля истинно положительных классификаций. Показывает, какая доля объектов, реально относящихся к положительному классу, правильно классифицирована.

$$\text{Recall} = \frac{TP}{TP + FN}. \quad (39)$$

Recall — отношение числа Positive элементов, корректно классифицированных как Positive, к общему количеству Positive объектов.

1 Применима в условиях несбалансированных выборок.

Для примера с Ассигасу (матрица ошибок СМ):

$$\text{Recall} = \frac{90}{90 + 10} = 0.9$$

2 Метрика Recall измеряет способность классификатора определять объекты, относящиеся к классу Positive. Чем выше Recall, тем больше Positive объектов было найдено.

3 Значение  $\text{Recall} \in [0, 1]$  показывает процент Positive объектов, которые верно классифицированы. Например, если имеется 100 элементов Positive и  $\text{Recall} = 0.72$ , получается, что классификатор корректно определил 72% объектов класса Positive.

4 Метрика Recall не зависит от того, как предсказываются Negative элементы, в отличие от Precision.

i Большое значение Recall означает, что классификатору **можно доверять** в его способности обнаруживать представителей класса Positive.

*Полнота, чувствительность, sensitivity, TPR (true positive rate)* (Recall) — доля истинно положительных классификаций. Показывает, какая доля объектов, реально относящихся к положительному классу, правильно классифицирована.

$$\text{Recall} = \frac{TP}{TP + FN}. \quad (39)$$

Recall — отношение числа Positive элементов, корректно классифицированных как Positive, к общему количеству Positive объектов.

1 Применима в условиях несбалансированных выборок.

Для примера с Ассигасу (матрица ошибок СМ):

$$\text{Recall} = \frac{90}{90 + 10} = 0.9$$

2 Метрика Recall измеряет способность классификатора определять объекты, относящиеся к классу Positive. Чем выше Recall, тем больше Positive объектов было найдено.

3 Значение  $\text{Recall} \in [0, 1]$  показывает процент Positive объектов, которые верно классифицированы. Например, если имеется 100 элементов Positive и  $\text{Recall} = 0.72$ , получается, что классификатор корректно определил 72% объектов класса Positive.

4 Метрика Recall не зависит от того, как предсказываются Negative элементы, в отличие от Precision.

i Большое значение Recall означает, что классификатору **можно доверять** в его способности обнаруживать представителей класса Positive.

i Если классификатор верно определит **все** Positive объекты, значение Recall будет равно 1.0, даже если все представители класса Negative были **ошибочно** определены как Positive.



**i** Метрики Precision и Recall не зависят (в отличие от Accuracy) от соотношения классов, потому применимы в условиях несбалансированных выборок.

- i** Метрики Precision и Recall не зависят (в отличие от Accuracy) от соотношения классов, потому применимы в условиях несбалансированных выборок.
- 1** Метрика Precision учитывает классификацию как Positive, так и Negative объектов. Recall же использует при расчете только представителей класса Positive. То есть, Precision зависит как от Negative, так и от Positive элементов, а Recall — только от Positive.

- i** Метрики Precision и Recall не зависят (в отличие от Accuracy) от соотношения классов, потому применимы в условиях несбалансированных выборок.
- 1** Метрика Precision учитывает классификацию как Positive, так и Negative объектов. Recall же использует при расчете только представителей класса Positive. То есть, Precision зависит как от Negative, так и от Positive элементов, а Recall — только от Positive.
- 2** Precision принимает во внимание, когда объект определяется как Positive, но не заботится о верной классификации всех объектов класса Positive. Recall в свою очередь учитывает корректность предсказания всех Positive объектов, но не заботится об ошибочной классификации представителей Negative как Positive.

- i** Метрики Precision и Recall не зависят (в отличие от Accuracy) от соотношения классов, потому применимы в условиях несбалансированных выборок.
- 1** Метрика Precision учитывает классификацию как Positive, так и Negative объектов. Recall же использует при расчете только представителей класса Positive. То есть, Precision зависит как от Negative, так и от Positive элементов, а Recall — только от Positive.
- 2** Precision принимает во внимание, когда объект определяется как Positive, но не заботится о верной классификации всех объектов класса Positive. Recall в свою очередь учитывает корректность предсказания всех Positive объектов, но не заботится об ошибочной классификации представителей Negative как Positive.
- 3** Если метрика Recall велика, а Precision — мала, такой классификатор правильно определяет большинство Positive объектов, но имеет много ложных срабатываний (классификаций Negative элементов как Positive).

- i** Метрики Precision и Recall не зависят (в отличие от Accuracy) от соотношения классов, потому применимы в условиях несбалансированных выборок.
- 1** Метрика Precision учитывает классификацию как Positive, так и Negative объектов. Recall же использует при расчете только представителей класса Positive. То есть, Precision зависит как от Negative, так и от Positive элементов, а Recall — только от Positive.
- 2** Precision принимает во внимание, когда объект определяется как Positive, но не заботится о верной классификации всех объектов класса Positive. Recall в свою очередь учитывает корректность предсказания всех Positive объектов, но не заботится об ошибочной классификации представителей Negative как Positive.
- 3** Если метрика Recall велика, а Precision — мала, такой классификатор правильно определяет большинство Positive объектов, но имеет много ложных срабатываний (классификаций Negative элементов как Positive).
- 4** Если метрика Precision большая, но Recall — низкая, то классификатор делает точные предсказания, определяя класс Positive, но число таких прогнозов очень мало.



- i** Метрики Precision и Recall не зависят (в отличие от Accuracy) от соотношения классов, потому применимы в условиях несбалансированных выборок.
- 1** Метрика Precision учитывает классификацию как Positive, так и Negative объектов. Recall же использует при расчете только представителей класса Positive. То есть, Precision зависит как от Negative, так и от Positive элементов, а Recall — только от Positive.
- 2** Precision принимает во внимание, когда объект определяется как Positive, но не заботится о верной классификации всех объектов класса Positive. Recall в свою очередь учитывает корректность предсказания всех Positive объектов, но не заботится об ошибочной классификации представителей Negative как Positive.
- 3** Если метрика Recall велика, а Precision — мала, такой классификатор правильно определяет большинство Positive объектов, но имеет много ложных срабатываний (классификаций Negative элементов как Positive).
- 4** Если метрика Precision большая, но Recall — низкая, то классификатор делает точные предсказания, определяя класс Positive, но число таких прогнозов очень мало.

## Что использовать?

- i** Метрики Precision и Recall не зависят (в отличие от Accuracy) от соотношения классов, потому применимы в условиях несбалансированных выборок.
- 1** Метрика Precision учитывает классификацию как Positive, так и Negative объектов. Recall же использует при расчете только представителей класса Positive. То есть, Precision зависит как от Negative, так и от Positive элементов, а Recall — только от Positive.
- 2** Precision принимает во внимание, когда объект определяется как Positive, но не заботится о верной классификации всех объектов класса Positive. Recall в свою очередь учитывает корректность предсказания всех Positive объектов, но не заботится об ошибочной классификации представителей Negative как Positive.
- 3** Если метрика Recall велика, а Precision — мала, такой классификатор правильно определяет большинство Positive объектов, но имеет много ложных срабатываний (классификаций Negative элементов как Positive).
- 4** Если метрика Precision большая, но Recall — низкая, то классификатор делает точные предсказания, определяя класс Positive, но число таких прогнозов очень мало.

## Что использовать?

Решение о том, использовать метрику Precision или Recall, зависит от конкретной задачи.

- i** Метрики Precision и Recall не зависят (в отличие от Accuracy) от соотношения классов, потому применимы в условиях несбалансированных выборок.
- 1** Метрика Precision учитывает классификацию как Positive, так и Negative объектов. Recall же использует при расчете только представителей класса Positive. То есть, Precision зависит как от Negative, так и от Positive элементов, а Recall — только от Positive.
- 2** Precision принимает во внимание, когда объект определяется как Positive, но не заботится о верной классификации всех объектов класса Positive. Recall в свою очередь учитывает корректность предсказания всех Positive объектов, но не заботится об ошибочной классификации представителей Negative как Positive.
- 3** Если метрика Recall велика, а Precision — мала, такой классификатор правильно определяет большинство Positive объектов, но имеет много ложных срабатываний (классификаций Negative элементов как Positive).
- 4** Если метрика Precision большая, но Recall — низкая, то классификатор делает точные предсказания, определяя класс Positive, но число таких прогнозов очень мало.

## Что использовать?

Решение о том, использовать метрику Precision или Recall, зависит от конкретной задачи.

- A** Если стоит задача определения всех только Positive элементов (не заботясь о том, будут ли Negative элементы классифицированы как Positive), можно использовать Recall.

- i** Метрики Precision и Recall не зависят (в отличие от Accuracy) от соотношения классов, потому применимы в условиях несбалансированных выборок.
- 1** Метрика Precision учитывает классификацию как Positive, так и Negative объектов. Recall же использует при расчете только представителей класса Positive. То есть, Precision зависит как от Negative, так и от Positive элементов, а Recall — только от Positive.
- 2** Precision принимает во внимание, когда объект определяется как Positive, но не заботится о верной классификации всех объектов класса Positive. Recall в свою очередь учитывает корректность предсказания всех Positive объектов, но не заботится об ошибочной классификации представителей Negative как Positive.
- 3** Если метрика Recall велика, а Precision — мала, такой классификатор правильно определяет большинство Positive объектов, но имеет много ложных срабатываний (классификаций Negative элементов как Positive).
- 4** Если метрика Precision большая, но Recall — низкая, то классификатор делает точные предсказания, определяя класс Positive, но число таких прогнозов очень мало.

## Что использовать?

Решение о том, использовать метрику Precision или Recall, зависит от конкретной задачи.

- A** Если стоит задача определения всех только Positive элементов (не заботясь о том, будут ли Negative элементы классифицированы как Positive), можно использовать Recall.
- B** Если нужно общее предсказание класса Positive (с учетом Negative объектов, которые были ошибочно классифицированы как Positive) желательно использовать Precision.

- i** Метрики Precision и Recall не зависят (в отличие от Accuracy) от соотношения классов, потому применимы в условиях несбалансированных выборок.
- 1** Метрика Precision учитывает классификацию как Positive, так и Negative объектов. Recall же использует при расчете только представителей класса Positive. То есть, Precision зависит как от Negative, так и от Positive элементов, а Recall — только от Positive.
- 2** Precision принимает во внимание, когда объект определяется как Positive, но не заботится о верной классификации всех объектов класса Positive. Recall в свою очередь учитывает корректность предсказания всех Positive объектов, но не заботится об ошибочной классификации представителей Negative как Positive.
- 3** Если метрика Recall велика, а Precision — мала, такой классификатор правильно определяет большинство Positive объектов, но имеет много ложных срабатываний (классификаций Negative элементов как Positive).
- 4** Если метрика Precision большая, но Recall — низкая, то классификатор делает точные предсказания, определяя класс Positive, но число таких прогнозов очень мало.

## Что использовать?

Решение о том, использовать метрику Precision или Recall, зависит от конкретной задачи.

- A** Если стоит задача определения всех только Positive элементов (не заботясь о том, будут ли Negative элементы классифицированы как Positive), можно использовать Recall.
- B** Если нужно общее предсказание класса Positive (с учетом Negative объектов, которые были ошибочно классифицированы как Positive) желательно использовать Precision.
- i** В Python библиотеке **Scikit-learn** есть модуль **metrics**, который можно использовать для вычисления метрик в матрице ошибок.



## Мотивация

## Мотивация

- 1 Найти оптимальный баланс между метриками Precision и Recall.



## Мотивация

- 1 Найти оптимальный баланс между метриками Precision и Recall.
- 2 Нужна метрика, объединяющая информацию о точности и полноте классификатора.

## Мотивация

- 1 Найти оптимальный баланс между метриками Precision и Recall.
- 2 Нужна метрика, объединяющая информацию о точности и полноте классификатора.

## F-мера

## Мотивация

- 1 Найти оптимальный баланс между метриками Precision и Recall.
- 2 Нужна метрика, объединяющая информацию о точности и полноте классификатора.

## F-мера

*F-мера* (F-score) — представляет собой среднее гармоническое между точностью и полнотой.

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (40)$$

## Мотивация

- 1 Найти оптимальный баланс между метриками Precision и Recall.
- 2 Нужна метрика, объединяющая информацию о точности и полноте классификатора.

## F-мера

*F-мера* (F-score) — представляет собой среднее гармоническое между точностью и полнотой.

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (40)$$

В формуле (40) точность и полнота имеют одинаковый вес — F-мера будет одинаково убывать при уменьшении и точности и полноты.

## Мотивация

- 1 Найти оптимальный баланс между метриками Precision и Recall.
- 2 Нужна метрика, объединяющая информацию о точности и полноте классификатора.

## F-мера

*F-мера* (F-score) — представляет собой среднее гармоническое между точностью и полнотой.

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (40)$$

В формуле (40) точность и полнота имеют одинаковый вес — F-мера будет одинаково убывать при уменьшении и точности и полноты.

F-мера достигает максимума при максимальной полноте и точности, и близка к нулю, если один из аргументов близок к нулю.

## Мотивация

- 1 Найти оптимальный баланс между метриками Precision и Recall.
- 2 Нужна метрика, объединяющая информацию о точности и полноте классификатора.

## F-мера

*F-мера* (F-score) — представляет собой среднее гармоническое между точностью и полнотой.

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (40)$$

В формуле (40) точность и полнота имеют одинаковый вес — F-мера будет одинаково убывать при уменьшении и точности и полноты.

F-мера достигает максимума при максимальной полноте и точности, и близка к нулю, если один из аргументов близок к нулю.

Для примера с Ассигасу (матрица ошибок СМ):

$$F_1 = \frac{2 \times 0.9473 \times 0.9}{0.9473 + 0.9} \approx 0.923.$$

## Мотивация

- 1 Найти оптимальный баланс между метриками Precision и Recall.
- 2 Нужна метрика, объединяющая информацию о точности и полноте классификатора.

## F-мера

*F-мера* (F-score) — представляет собой среднее гармоническое между точностью и полнотой.

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (40)$$

В формуле (40) точность и полнота имеют одинаковый вес — F-мера будет одинаково убывать при уменьшении и точности и полноты.

F-мера достигает максимума при максимальной полноте и точности, и близка к нулю, если один из аргументов близок к нулю.

Для примера с Ассигасу (матрица ошибок СМ):

$$F_1 = \frac{2 \times 0.9473 \times 0.9}{0.9473 + 0.9} \approx 0.923.$$

Иногда рассматривают модифицированную F-меру. В (40) можно задать разные веса для точности и полноты:

$$F_\beta = \frac{(1 + \beta^2) \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}}, \quad \beta > 0. \quad (41)$$

## Мотивация

- 1 Найти оптимальный баланс между метриками Precision и Recall.
- 2 Нужна метрика, объединяющая информацию о точности и полноте классификатора.

## F-мера

*F-мера* (F-score) — представляет собой среднее гармоническое между точностью и полнотой.

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (40)$$

В формуле (40) точность и полнота имеют одинаковый вес — F-мера будет одинаково убывать при уменьшении и точности и полноты.

F-мера достигает максимума при максимальной полноте и точности, и близка к нулю, если один из аргументов близок к нулю.

Для примера с Ассигасу (матрица ошибок СМ):

$$F_1 = \frac{2 \times 0.9473 \times 0.9}{0.9473 + 0.9} \approx 0.923.$$

Иногда рассматривают модифицированную F-меру. В (40) можно задать разные веса для точности и полноты:

$$F_\beta = \frac{(1 + \beta^2) \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}}, \quad \beta > 0. \quad (41)$$

- i** Для  $0 < \beta < 1$  приоритет в (41) получает точность (Precision), а при  $\beta > 1$  приоритет отдается полноте (Recall).



## Мотивация

- 1 Найти оптимальный баланс между метриками Precision и Recall.
- 2 Нужна метрика, объединяющая информацию о точности и полноте классификатора.

## F-мера

*F-мера* (F-score) — представляет собой среднее гармоническое между точностью и полнотой.

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (40)$$

В формуле (40) точность и полнота имеют одинаковый вес — F-мера будет одинаково убывать при уменьшении и точности и полноты.

F-мера достигает максимума при максимальной полноте и точности, и близка к нулю, если один из аргументов близок к нулю.

Для примера с Ассигасу (матрица ошибок СМ):

$$F_1 = \frac{2 \times 0.9473 \times 0.9}{0.9473 + 0.9} \approx 0.923.$$

Иногда рассматривают модифицированную F-меру. В (40) можно задать разные веса для точности и полноты:

$$F_\beta = \frac{(1 + \beta^2) \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}}, \quad \beta > 0. \quad (41)$$

- i Для  $0 < \beta < 1$  приоритет в (41) получает точность (Precision), а при  $\beta > 1$  приоритет отдается полноте (Recall).
- i При  $\beta = 1$  получаем классическую F-меру (40). Иногда ее называют *сбалансированной* F-мерой или мерой F1.



- 1 Визуализируем результаты классификации для рассмотренных ранее мультиклассовых классификаторов:

- 1 Визуализируем результаты классификации для рассмотренных ранее мультиклассовых классификаторов:

**Мультиклассовый SVM. Один против остальных**

- 1 Визуализируем результаты классификации для рассмотренных ранее мультиклассовых классификаторов:

**Мультиклассовый SVM. Один против остальных**

**Мультиклассовый SVM. Каждый против каждого**

- 1 Визуализируем результаты классификации для рассмотренных ранее мультиклассовых классификаторов:

**Мультиклассовый SVM. Один против остальных**

**Мультиклассовый SVM. Каждый против каждого**

**Классификатор k-ближайших соседей**

- 1 Визуализируем результаты классификации для рассмотренных ранее мультиклассовых классификаторов:

**Мультиклассовый SVM. Один против остальных**

**Мультиклассовый SVM. Каждый против каждого**

**Классификатор k-ближайших соседей**

**Алгоритм CART**

- 1 Визуализируем результаты классификации для рассмотренных ранее мультиклассовых классификаторов:

**Мультиклассовый SVM. Один против остальных**

**Мультиклассовый SVM. Каждый против каждого**

**Классификатор k-ближайших соседей**

**Алгоритм CART**

**Random Forest**



1 Визуализируем результаты классификации для рассмотренных ранее мультиклассовых классификаторов:

**Мультиклассовый SVM. Один против остальных**

**Мультиклассовый SVM. Каждый против каждого**

**Классификатор k-ближайших соседей**

**Алгоритм CART**

**Random Forest**

**Гауссовский байесовский классификатор**

- 1 Визуализируем результаты классификации для рассмотренных ранее мультиклассовых классификаторов:
  - Мультиклассовый SVM. Один против остальных
  - Мультиклассовый SVM. Каждый против каждого
  - Классификатор к-ближайших соседей
  - Алгоритм CART
  - Random Forest
  - Гауссовский байесовский классификатор
- 2 Тестовая выборка (Ирисы Фишера) — рисунок 13:а. Взяты два первых признака: *sepal length* и *sepal width*.

- 1 Визуализируем результаты классификации для рассмотренных ранее мультиклассовых классификаторов:
  - Мультиклассовый SVM. Один против остальных
  - Мультиклассовый SVM. Каждый против каждого
  - Классификатор k-ближайших соседей
  - Алгоритм CART
  - Random Forest
  - Гауссовский байесовский классификатор
- 2 Тестовая выборка (**Ирисы Фишера**) — рисунок 13:а. Взяты два первых признака: *sepal length* и *sepal width*.
- 3 Используя библиотеку **Yellowbrick** визуализируем результаты для каждого классификатора.

- 1 Визуализируем результаты классификации для рассмотренных ранее мультиклассовых классификаторов:

Мультиклассовый SVM. Один против остальных

Мультиклассовый SVM. Каждый против каждого

Классификатор k-ближайших соседей

Алгоритм CART

Random Forest

Гауссовский байесовский классификатор

- 2 Тестовая выборка (**Ирисы Фишера**) — рисунок 13:а. Взяты два первых признака: *sepal length* и *sepal width*.

- 3 Используя библиотеку **Yellowbrick** визуализируем результаты для каждого классификатора.

Визуализатор результатов классификации отображает оценки Precision, Recall и F-score. Чтобы упростить интерпретацию и обнаружение проблем, числовые оценки объединены с цветовой тепловой картой.

- 1 Визуализируем результаты классификации для рассмотренных ранее мультиклассовых классификаторов:

Мультиклассовый SVM. Один против остальных

Мультиклассовый SVM. Каждый против каждого

Классификатор k-ближайших соседей

Алгоритм CART

Random Forest

Гауссовский байесовский классификатор

- 2 Тестовая выборка (**Ирисы Фишера**) — рисунок 13:a. Взяты два первых признака: *sepal length* и *sepal width*.

- 3 Используя библиотеку **Yellowbrick** визуализируем результаты для каждого классификатора.

Визуализатор результатов классификации отображает оценки Precision, Recall и F-score. Чтобы упростить интерпретацию и обнаружение проблем, числовые оценки объединены с цветовой тепловой картой.

- i Пример для классификатора **CART**: (Xtrain, ytrain) и (Xtest, ytest) — тренировочное и тестовое множества.

```
from sklearn.tree import DecisionTreeClassifier
from yellowbrick.classifier import ClassificationReport
#####
classes = ["setosa", "virginica", "versicolor"]
visualizer = ClassificationReport(DecisionTreeClassifier(), classes=classes, support=False)
visualizer.fit(Xtrain, ytrain)
visualizer.score(Xtest, ytest)
visualizer.show()
```



Показатели для классификатора OneVsRest



Показатели для классификатора OneVsOne



Показатели для классификатора KNN



Показатели для классификатора CART



Показатели для классификатора RandomForest



Показатели для классификатора GaussianNB







**i** Впервые она была создана для использования радиолокационного обнаружения сигналов во время Второй мировой войны. США использовали ROC для повышения точности обнаружения японских самолетов с помощью радара. Поэтому ее и называют **рабочей характеристикой приемника**.

**И** Впервые она была создана для использования радиолокационного обнаружения сигналов во время Второй мировой войны. США использовали ROC для повышения точности обнаружения японских самолетов с помощью радара. Поэтому ее и называют **рабочей характеристикой приемника**.

**А** *ROC-кривая* (рабочая характеристика приемника, receiver operating characteristic, ROC) — графическое представление зависимости двух величин: *чувствительности*<sup>а</sup> и *False Positive Rate* (FPR).

---

<sup>а</sup>Recall, TPR (true positive rate) — была введена ранее: (39) — доля истинно положительных классификаций

**И** Впервые она была создана для использования радиолокационного обнаружения сигналов во время Второй мировой войны. США использовали ROC для повышения точности обнаружения японских самолетов с помощью радара. Поэтому ее и называют **рабочей характеристикой приемника**.

**А** *ROC-кривая* (рабочая характеристика приемника, receiver operating characteristic, ROC) — графическое представление зависимости двух величин: *чувствительности*<sup>а</sup> и *False Positive Rate* (FPR).

**В** FPR — доля ложно положительных классификаций в общем числе отрицательных классификаций, (можно интерпретировать как вероятность «ложной тревоги»).

---

<sup>а</sup>Recall, TPR (true positive rate) — была введена ранее: (39) — доля истинно положительных классификаций

**И** Впервые она была создана для использования радиолокационного обнаружения сигналов во время Второй мировой войны. США использовали ROC для повышения точности обнаружения японских самолетов с помощью радара. Поэтому ее и называют **рабочей характеристикой приемника**.

**А** *ROC-кривая* (рабочая характеристика приемника, receiver operating characteristic, ROC) — графическое представление зависимости двух величин: *чувствительности*<sup>а</sup> и *False Positive Rate* (FPR).

**В** FPR — доля ложно положительных классификаций в общем числе отрицательных классификаций, (можно интерпретировать как вероятность «ложной тревоги»).

**С** Величина FPR показывает долю неверных срабатываний классификатора к общему числу объектов за пределами класса — **как часто классификатор ошибается** при отнесении того или иного объекта к классу.

$$\text{TPR} = \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}}. \quad (42)$$

---

<sup>а</sup>Recall, TPR (true positive rate) — была введена ранее: (39) — доля истинно положительных классификаций



- 1 Бинарный классификатор  $a(X)$  оценивает вероятность принадлежности объекта  $X$  к положительному классу.

- 1 Бинарный классификатор  $a(\mathbf{X})$  оценивает вероятность принадлежности объекта  $\mathbf{X}$  к положительному классу.
- 2 Введем параметр  $\mu \in [0, 1]$ , по которому будем разбивать выборку на два класса:

$$\begin{cases} a(\mathbf{X}) \geq \mu - \text{«положительный» класс,} \\ a(\mathbf{X}) < \mu - \text{«отрицательный» класс.} \end{cases} \quad (43)$$

Параметр  $\mu$  в (43) часто называют *порогом*, или *точкой отсечения* (cut-off value).

- 1 Бинарный классификатор  $a(\mathbf{X})$  оценивает вероятность принадлежности объекта  $\mathbf{X}$  к положительному классу.
- 2 Введем параметр  $\mu \in [0, 1]$ , по которому будем разбивать выборку на два класса:

$$\begin{cases} a(\mathbf{X}) \geq \mu - \text{«положительный» класс,} \\ a(\mathbf{X}) < \mu - \text{«отрицательный» класс.} \end{cases} \quad (43)$$

Параметр  $\mu$  в (43) часто называют *порогом*, или *точкой отсечения* (cut-off value).

- 3 По оси OX графика ROC-кривой — FPR, по оси OY — TPR.



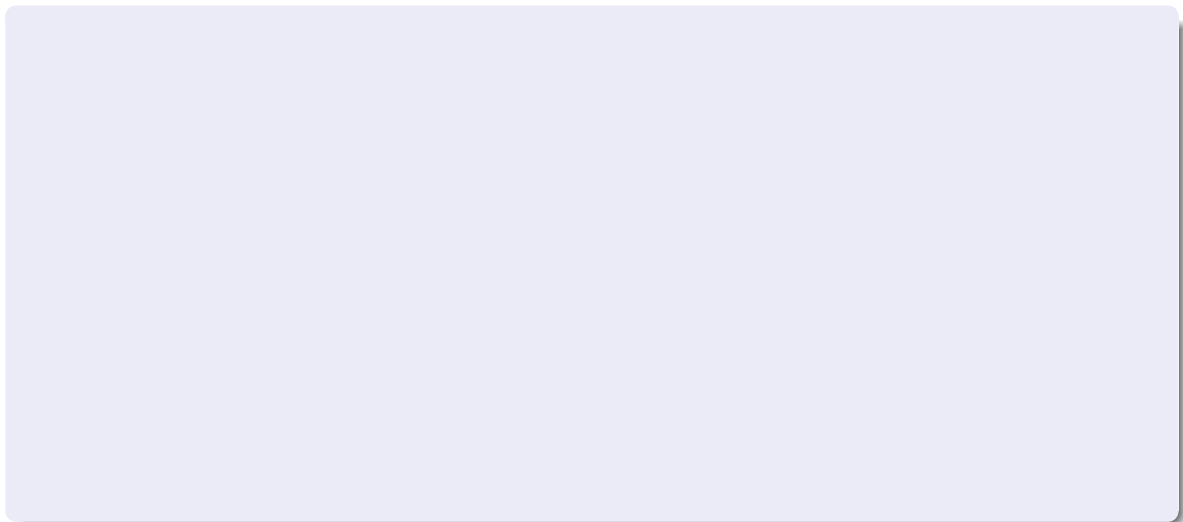
- 1 Бинарный классификатор  $a(\mathbf{X})$  оценивает вероятность принадлежности объекта  $\mathbf{X}$  к положительному классу.
- 2 Введем параметр  $\mu \in [0, 1]$ , по которому будем разбивать выборку на два класса:

$$\begin{cases} a(\mathbf{X}) \geq \mu - \text{«положительный» класс,} \\ a(\mathbf{X}) < \mu - \text{«отрицательный» класс.} \end{cases} \quad (43)$$

Параметр  $\mu$  в (43) часто называют *порогом*, или *точкой отсечения* (cut-off value).

- 3 По оси OX графика ROC-кривой — FPR, по оси OY — TPR.
- 4 Вычисляем FPR, TPR по (43) для всех  $\mu \in [0, 1]$  (с некоторым шагом). Получаем точки ROC-кривой на плоскости:  $(\text{FPR}(\mu), \text{TPR}(\mu))$ .





✓ Имеется 6 объектов «положительного» класса ( $y = 1$ ) и 4 объекта «отрицательного» класса ( $y = 0$ ). Вероятности, предсказанные классификатором — в таблице 4. Объекты расположены в порядке возрастания вероятностей:

*Таблица 4. Результаты классификации*

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$
$y$	1	1	0	0	1	1	0	1	1	0
$p(X)$	0.07	0.15	0.24	0.35	0.47	0.52	0.69	0.77	0.85	0.94

✓ Имеется 6 объектов «положительного» класса ( $y = 1$ ) и 4 объекта «отрицательного» класса ( $y = 0$ ). Вероятности, предсказанные классификатором — в таблице 4. Объекты расположены в порядке возрастания вероятностей:

Таблица 4. Результаты классификации

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$
$y$	1	1	0	0	1	1	0	1	1	0
$p(X)$	0.07	0.15	0.24	0.35	0.47	0.52	0.69	0.77	0.85	0.94

✓ Вычисляем  $FPR(\mu)$  и  $TPR(\mu)$  для  $\mu$  с шагом 0.1 согласно (42) — таблица 5:

Таблица 5

$\mu$	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
TP	6	5	4	4	4	3	2	2	1	0	0
TN	0	0	0	1	2	2	2	3	3	3	4
FP	4	4	4	3	2	2	2	1	1	1	0
FN	0	1	2	2	2	3	4	4	5	6	6
FPR	1.0	1.0	1.0	0.75	0.5	0.5	0.5	0.25	0.25	0.25	0
TPR	1.0	0.833	0.667	0.667	0.667	0.5	0.33	0.33	0.167	0	0



⇒ ROC-кривая по таблице 5 на рисунке 18 синим цветом.

⇒ ROC-кривая по таблице 5 на рисунке 18 синим цветом.

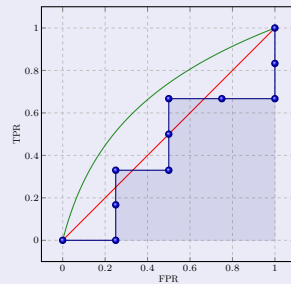


Рис. 18



- ⇒ ROC-кривая по таблице 5 на рисунке 18 синим цветом.
- ⇒ Положительная диагональ ( $FPR = TPR$ ) показана на рисунке 18 красным цветом.

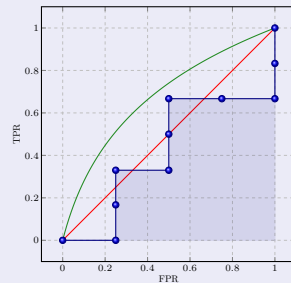


Рис. 18

- ⇒ ROC-кривая по таблице 5 на рисунке 18 синим цветом.
- ⇒ Положительная диагональ ( $FPR = TPR$ ) показана на рисунке 18 красным цветом.
- ⇒ Методом сравнения ROC-кривых (для различных классификаторов) является оценка площади под кривыми — *Area Under Curve, AUC*, изменяется от 0 до 1.

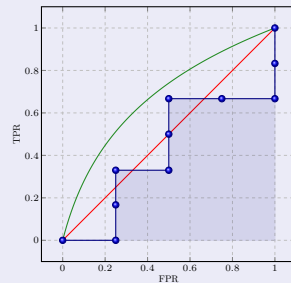


Рис. 18

- ⇒ ROC-кривая по таблице 5 на рисунке 18 синим цветом.
- ⇒ Положительная диагональ ( $FPR = TPR$ ) показана на рисунке 18 красным цветом.
- ⇒ Методом сравнения ROC-кривых (для различных классификаторов) является оценка площади под кривыми — *Area Under Curve, AUC*, изменяется от 0 до 1.
- ⇒ Чем выше показатель AUC, тем качественнее классификатор.

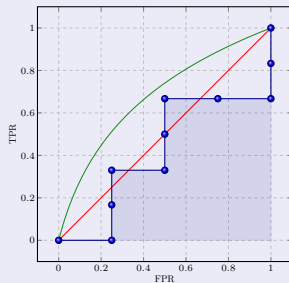


Рис. 18

- ⇒ ROC-кривая по таблице 5 на рисунке 18 синим цветом.
- ⇒ Положительная диагональ ( $FPR = TPR$ ) показана на рисунке 18 красным цветом.
- ⇒ Методом сравнения ROC-кривых (для различных классификаторов) является оценка площади под кривыми — *Area Under Curve, AUC*, изменяется от 0 до 1.
- ⇒ Чем выше показатель AUC, тем качественнее классификатор.
- ⇒ Значение 0.5 и ниже говорит о плохом качестве метода классификации — *это случай нашего примера!*

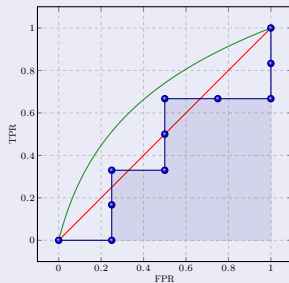


Рис. 18

# ROC-кривая. AUC

- ⇒ ROC-кривая по таблице 5 на рисунке 18 синим цветом.
- ⇒ Положительная диагональ ( $FPR = TPR$ ) показана на рисунке 18 красным цветом.
- ⇒ Методом сравнения ROC-кривых (для различных классификаторов) является оценка площади под кривыми — *Area Under Curve, AUC*, изменяется от 0 до 1.
- ⇒ Чем выше показатель AUC, тем качественнее классификатор.
- ⇒ Значение 0.5 и ниже говорит о плохом качестве метода классификации — *это случай нашего примера!*
- ⇒ В нашем примере  $AUC \approx 0.42$ .

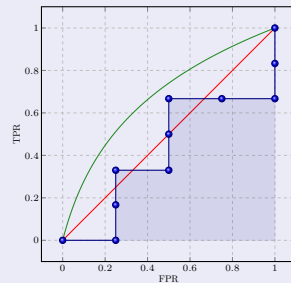


Рис. 18

# ROC-кривая. AUC

- ⇒ ROC-кривая по таблице 5 на рисунке 18 синим цветом.
- ⇒ Положительная диагональ ( $FPR = TPR$ ) показана на рисунке 18 красным цветом.
- ⇒ Методом сравнения ROC-кривых (для различных классификаторов) является оценка площади под кривыми — *Area Under Curve, AUC*, изменяется от 0 до 1.
- ⇒ Чем выше показатель AUC, тем качественнее классификатор.
- ⇒ Значение 0.5 и ниже говорит о плохом качестве метода классификации — *это случай нашего примера!*
- ⇒ В нашем примере  $AUC \approx 0.42$ .
- ⇒ В таблице 6 показана примерная экспертная шкала для значений AUC, по которой можно судить о качестве классификатора:

Таблица 6. Экспертная шкала значений AUC

Диапазон AUC	Качество классификатора
$0.9 \leq AUC$	Отличное
$0.8 \leq AUC < 0.9$	Очень хорошее
$0.7 \leq AUC < 0.8$	Хорошее
$0.6 \leq AUC < 0.7$	Среднее
$AUC < 0.6$	Плохое

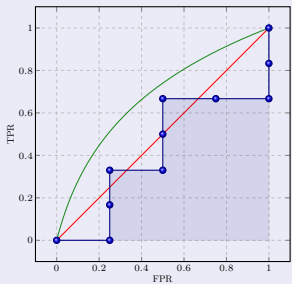


Рис. 18

# ROC-кривая. AUC

- ⇒ ROC-кривая по таблице 5 на рисунке 18 синим цветом.
- ⇒ Положительная диагональ ( $FPR = TPR$ ) показана на рисунке 18 красным цветом.
- ⇒ Методом сравнения ROC-кривых (для различных классификаторов) является оценка площади под кривыми — *Area Under Curve, AUC*, изменяется от 0 до 1.
- ⇒ Чем выше показатель AUC, тем качественнее классификатор.
- ⇒ Значение 0.5 и ниже говорит о плохом качестве метода классификации — *это случай нашего примера!*
- ⇒ В нашем примере  $AUC \approx 0.42$ .
- ⇒ В таблице 6 показана примерная экспертная шкала для значений AUC, по которой можно судить о качестве классификатора:

Таблица 6. Экспертная шкала значений AUC

Диапазон AUC	Качество классификатора
$0.9 \leq AUC$	Отличное
$0.8 \leq AUC < 0.9$	Очень хорошее
$0.7 \leq AUC < 0.8$	Хорошее
$0.6 \leq AUC < 0.7$	Среднее
$AUC < 0.6$	Плохое

- ⇒ ROC-кривая для хорошего классификатора — на рисунке 18 зеленым цветом.

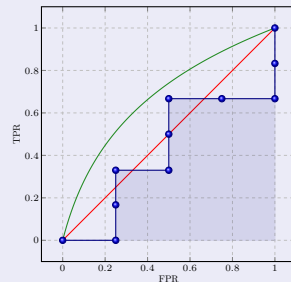


Рис. 18





```
# Connecting the necessary libraries
from sklearn.metrics import roc_curve, auc
from sklearn.datasets import make_classification
from sklearn.naive_bayes import GaussianNB
X, y = make_classification(n_samples=500, random_state=0)
# Fit the model
model = GaussianNB()
model.fit(X,y)
y_score = model.predict_proba(X)[:, 1]
# Calculation fpr = False Positive Rate, tpr = True Positive Rate and AUC
fpr, tpr, thresholds = roc_curve(y, y_score)
AUC = auc(fpr, tpr)
```

# ROC-кривая, AUC. Пример на Питоне

```
# Connecting the necessary libraries
from sklearn.metrics import roc_curve, auc
from sklearn.datasets import make_classification
from sklearn.naive_bayes import GaussianNB
X, y = make_classification(n_samples=500, random_state=0)
# Fit the model
model = GaussianNB()
model.fit(X,y)
y_score = model.predict_proba(X)[: , 1]
# Calculation fpr = False Positive Rate, tpr = True Positive Rate and AUC
fpr, tpr, thresholds = roc_curve(y, y_score)
AUC = auc(fpr, tpr)
```

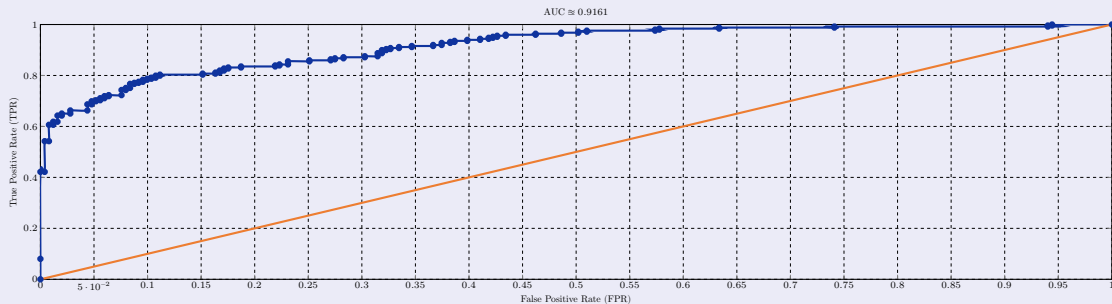


Рис. 19



- 1 Построим ROC-кривую для для рассмотренных ранее бинарных классификаторов:

- 1 Построим ROC-кривую для для рассмотренных ранее бинарных классификаторов:

**Метод опорных векторов с полиномиальным ядром**

- 1 Построим ROC-кривую для для рассмотренных ранее бинарных классификаторов:

**Метод опорных векторов с полиномиальным ядром**

**Метод опорных векторов с гауссовым ядром**

1 Построим ROC-кривую для для рассмотренных ранее бинарных классификаторов:

**Метод опорных векторов с полиномиальным ядром**

**Метод опорных векторов с гауссовым ядром**

**Классификатор  $k$ -ближайших соседей**

1 Построим ROC-кривую для для рассмотренных ранее бинарных классификаторов:

**Метод опорных векторов с полиномиальным ядром**

**Метод опорных векторов с гауссовым ядром**

**Классификатор  $k$ -ближайших соседей**

**Алгоритм CART**



1 Построим ROC-кривую для для рассмотренных ранее бинарных классификаторов:

**Метод опорных векторов с полиномиальным ядром**

**Метод опорных векторов с гауссовым ядром**

**Классификатор  $k$ -ближайших соседей**

**Алгоритм CART**

**Random Forest**

1 Построим ROC-кривую для для рассмотренных ранее бинарных классификаторов:

**Метод опорных векторов с полиномиальным ядром**

**Метод опорных векторов с гауссовым ядром**

**Классификатор  $k$ -ближайших соседей**

**Алгоритм CART**

**Random Forest**

**Гауссовский байесовский классификатор**

- 1 Построим ROC-кривую для для рассмотренных ранее бинарных классификаторов:

**Метод опорных векторов с полиномиальным ядром**

**Метод опорных векторов с гауссовым ядром**

**Классификатор  $k$ -ближайших соседей**

**Алгоритм CART**

**Random Forest**

**Гауссовский байесовский классификатор**

- 2 Используем визуализатор библиотеки **Yellowbrick**.

- 1 Построим ROC-кривую для для рассмотренных ранее бинарных классификаторов:
  - Метод опорных векторов с полиномиальным ядром
  - Метод опорных векторов с гауссовым ядром
  - Классификатор  $k$ -ближайших соседей
  - Алгоритм CART
  - Random Forest
  - Гауссовский байесовский классификатор
- 2 Используем визуализатор библиотеки **Yellowbrick**.
- 3 В качестве тестовой выборки взят набор электронных писем (со спамом и без него), предоставляемый используемой библиотекой.

- 1 Построим ROC-кривую для для рассмотренных ранее бинарных классификаторов:
    - Метод опорных векторов с полиномиальным ядром
    - Метод опорных векторов с гауссовым ядром
    - Классификатор  $k$ -ближайших соседей
    - Алгоритм CART
    - Random Forest
    - Гауссовский байесовский классификатор
  - 2 Используем визуализатор библиотеки **Yellowbrick**.
  - 3 В качестве тестовой выборки взят набор электронных писем (со спамом и без него), предоставляемый используемой библиотекой.
- Всего 4601 элемент, признаков — 27.

1 Построим ROC-кривую для для рассмотренных ранее бинарных классификаторов:

Метод опорных векторов с полиномиальным ядром

Метод опорных векторов с гауссовым ядром

Классификатор  $k$ -ближайших соседей

Алгоритм CART

Random Forest

Гауссовский байесовский классификатор

2 Используем визуализатор библиотеки **Yellowbrick**.

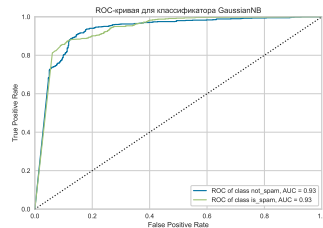
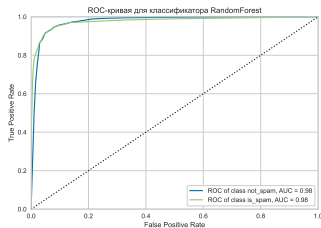
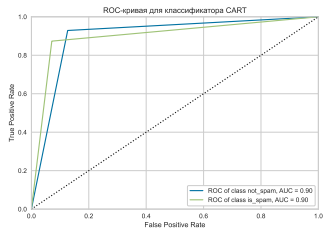
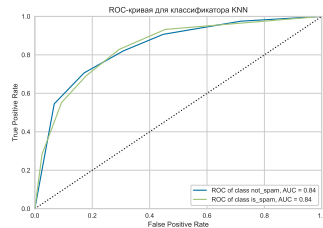
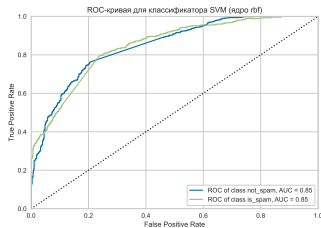
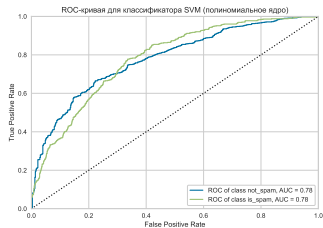
3 В качестве тестовой выборки взят набор электронных писем (со спамом и без него), предоставляемый используемой библиотекой.

Всего 4601 элемент, признаков — 27.

i Пример для классификатора *Random Forest*: (Xtrain, ytrain) и (Xtest, ytest) — тренировочное и тестовое множества.

```
from sklearn.ensemble import RandomForestClassifier
from yellowbrick.classifier import ROCAUC
from yellowbrick.datasets import load_spam
# Load dataset
X, y = load_spam()
classes = ["not_spam", "is_spam"]
model = RandomForestClassifier(random_state=42, n_estimators=20)
visualizer = ROCAUC(model, classes=classes, micro=False, macro=False)
visualizer.fit(Xtrain, ytrain)
visualizer.score(Xtest, ytest)
visualizer.show()
```











- 1 Пусть задача классификации на  $m$  классов ставится как  $m$  задач об отделении класса  $i \in 1 : m$  от остальных (например SVM по схеме «один против остальных»).

- 1 Пусть задача классификации на  $m$  классов ставится как  $m$  задач об отделении класса  $i \in 1 : m$  от остальных (например SVM по схеме «один против остальных»).
- 2 Для каждой задачи считаем свою матрицу ошибок.

- 1 Пусть задача классификации на  $m$  классов ставится как  $m$  задач об отделении класса  $i \in 1 : m$  от остальных (например SVM по схеме «один против остальных»).
- 2 Для каждой задачи считаем свою матрицу ошибок.
- 3 Существует два метода получения итогового значения метрик из  $m$  матриц ошибок, различающихся порядком усреднения.

- 1 Пусть задача классификации на  $m$  классов ставится как  $m$  задач об отделении класса  $i \in 1 : m$  от остальных (например SVM по схеме «один против остальных»).
- 2 Для каждой задачи считаем свою матрицу ошибок.
- 3 Существует два метода получения итогового значения метрик из  $m$  матриц ошибок, различающихся порядком усреднения.

## Микроусреднение

- 1 Пусть задача классификации на  $m$  классов ставится как  $m$  задач об отделении класса  $i \in 1 : m$  от остальных (например SVM по схеме «один против остальных»).
- 2 Для каждой задачи считаем свою матрицу ошибок.
- 3 Существует два метода получения итогового значения метрик из  $m$  матриц ошибок, различающихся порядком усреднения.

## Микроусреднение

- ✓ Усредняем элементы матрицы ошибок (TP, FP, TN, FN) для  $m$  бинарных классификаторов:

$$TP = \frac{1}{m} \sum_{i=1}^m TP_i, \quad FP = \frac{1}{m} \sum_{i=1}^m FP_i, \quad TN = \frac{1}{m} \sum_{i=1}^m TN_i, \quad FN = \frac{1}{m} \sum_{i=1}^m FN_i. \quad (44)$$

- 1 Пусть задача классификации на  $m$  классов ставится как  $m$  задач об отделении класса  $i \in 1 : m$  от остальных (например SVM по схеме «один против остальных»).
- 2 Для каждой задачи считаем свою матрицу ошибок.
- 3 Существует два метода получения итогового значения метрик из  $m$  матриц ошибок, различающихся порядком усреднения.

## Микроусреднение

- ✓ Усредняем элементы матрицы ошибок (TP, FP, TN, FN) для  $m$  бинарных классификаторов:

$$TP = \frac{1}{m} \sum_{i=1}^m TP_i, \quad FP = \frac{1}{m} \sum_{i=1}^m FP_i, \quad TN = \frac{1}{m} \sum_{i=1}^m TN_i, \quad FN = \frac{1}{m} \sum_{i=1}^m FN_i. \quad (44)$$

- ✓ По усредненной матрице (44) ошибок вычисляем Precision, Recall,  $F$ -меру.



- 1 Пусть задача классификации на  $m$  классов ставится как  $m$  задач об отделении класса  $i \in 1 : m$  от остальных (например SVM по схеме «один против остальных»).
- 2 Для каждой задачи считаем свою матрицу ошибок.
- 3 Существует два метода получения итогового значения метрик из  $m$  матриц ошибок, различающихся порядком усреднения.

## Микроусреднение

- ✓ Усредняем элементы матрицы ошибок (TP, FP, TN, FN) для  $m$  бинарных классификаторов:

$$TP = \frac{1}{m} \sum_{i=1}^m TP_i, \quad FP = \frac{1}{m} \sum_{i=1}^m FP_i, \quad TN = \frac{1}{m} \sum_{i=1}^m TN_i, \quad FN = \frac{1}{m} \sum_{i=1}^m FN_i. \quad (44)$$

- ✓ По усредненной матрице (44) ошибок вычисляем Precision, Recall,  $F$ -меру.

## Макроусреднение

# Оценка качества многоклассовой классификации

- 1 Пусть задача классификации на  $m$  классов ставится как  $m$  задач об отделении класса  $i \in 1 : m$  от остальных (например SVM по схеме «один против остальных»).
- 2 Для каждой задачи считаем свою матрицу ошибок.
- 3 Существует два метода получения итогового значения метрик из  $m$  матриц ошибок, различающихся порядком усреднения.

## Микроусреднение

- ✓ Усредняем элементы матрицы ошибок (TP, FP, TN, FN) для  $m$  бинарных классификаторов:

$$TP = \frac{1}{m} \sum_{i=1}^m TP_i, \quad FP = \frac{1}{m} \sum_{i=1}^m FP_i, \quad TN = \frac{1}{m} \sum_{i=1}^m TN_i, \quad FN = \frac{1}{m} \sum_{i=1}^m FN_i. \quad (44)$$

- ✓ По усредненной матрице (44) ошибок вычисляем Precision, Recall,  $F$ -меру.

## Макроусреднение

- ✓ Считаем Precision, Recall для каждого классификатора отдельно. Затем усредняем.

$$\text{Precision}_k = \frac{TP_k}{TP_k + FP_k}, \quad \text{Recall}_k = \frac{TP_k}{TP_k + FN_k}, \quad k \in 1 : m, \quad \text{Precision} = \frac{1}{m} \sum_{i=1}^m \text{Precision}_i, \quad \text{Recall} = \frac{1}{m} \sum_{i=1}^m \text{Recall}_i. \quad (45)$$

# Оценка качества многоклассовой классификации

- 1 Пусть задача классификации на  $m$  классов ставится как  $m$  задач об отделении класса  $i \in 1 : m$  от остальных (например SVM по схеме «один против остальных»).
- 2 Для каждой задачи считаем свою матрицу ошибок.
- 3 Существует два метода получения итогового значения метрик из  $m$  матриц ошибок, различающихся порядком усреднения.

## Микроусреднение

- ✓ Усредняем элементы матрицы ошибок (TP, FP, TN, FN) для  $m$  бинарных классификаторов:

$$TP = \frac{1}{m} \sum_{i=1}^m TP_i, \quad FP = \frac{1}{m} \sum_{i=1}^m FP_i, \quad TN = \frac{1}{m} \sum_{i=1}^m TN_i, \quad FN = \frac{1}{m} \sum_{i=1}^m FN_i. \quad (44)$$

- ✓ По усредненной матрице (44) ошибок вычисляем Precision, Recall,  $F$ -меру.

## Макроусреднение

- ✓ Считаем Precision, Recall для каждого классификатора отдельно. Затем усредняем.

$$\text{Precision}_k = \frac{TP_k}{TP_k + FP_k}, \quad \text{Recall}_k = \frac{TP_k}{TP_k + FN_k}, \quad k \in 1 : m, \quad \text{Precision} = \frac{1}{m} \sum_{i=1}^m \text{Precision}_i, \quad \text{Recall} = \frac{1}{m} \sum_{i=1}^m \text{Recall}_i. \quad (45)$$

- A Порядок усреднения влияет на результат в случае дисбаланса классов.

# Оценка качества многоклассовой классификации

- 1 Пусть задача классификации на  $m$  классов ставится как  $m$  задач об отделении класса  $i \in 1 : m$  от остальных (например SVM по схеме «один против остальных»).
- 2 Для каждой задачи считаем свою матрицу ошибок.
- 3 Существует два метода получения итогового значения метрик из  $m$  матриц ошибок, различающихся порядком усреднения.

## Микроусреднение

- ✓ Усредняем элементы матрицы ошибок (TP, FP, TN, FN) для  $m$  бинарных классификаторов:

$$TP = \frac{1}{m} \sum_{i=1}^m TP_i, \quad FP = \frac{1}{m} \sum_{i=1}^m FP_i, \quad TN = \frac{1}{m} \sum_{i=1}^m TN_i, \quad FN = \frac{1}{m} \sum_{i=1}^m FN_i. \quad (44)$$

- ✓ По усредненной матрице (44) ошибок вычисляем Precision, Recall,  $F$ -меру.

## Макроусреднение

- ✓ Считаем Precision, Recall для каждого классификатора отдельно. Затем усредняем.

$$\text{Precision}_k = \frac{TP_k}{TP_k + FP_k}, \quad \text{Recall}_k = \frac{TP_k}{TP_k + FN_k}, \quad k \in 1 : m, \quad \text{Precision} = \frac{1}{m} \sum_{i=1}^m \text{Precision}_i, \quad \text{Recall} = \frac{1}{m} \sum_{i=1}^m \text{Recall}_i. \quad (45)$$

- А** Порядок усреднения влияет на результат в случае дисбаланса классов.
- В** При дисбалансе классификатор редко предсказывает маленький класс. Значения TP и FN при классификации этого класса против остальных тоже будут малы. При микроусреднении вклад маленького класса в общую метрику — небольшой.

# Оценка качества многоклассовой классификации

- 1 Пусть задача классификации на  $m$  классов ставится как  $m$  задач об отделении класса  $i \in 1 : m$  от остальных (например SVM по схеме «один против остальных»).
- 2 Для каждой задачи считаем свою матрицу ошибок.
- 3 Существует два метода получения итогового значения метрик из  $m$  матриц ошибок, различающихся порядком усреднения.

## Микроусреднение

- ✓ Усредняем элементы матрицы ошибок (TP, FP, TN, FN) для  $m$  бинарных классификаторов:

$$TP = \frac{1}{m} \sum_{i=1}^m TP_i, \quad FP = \frac{1}{m} \sum_{i=1}^m FP_i, \quad TN = \frac{1}{m} \sum_{i=1}^m TN_i, \quad FN = \frac{1}{m} \sum_{i=1}^m FN_i. \quad (44)$$

- ✓ По усредненной матрице (44) ошибок вычисляем Precision, Recall,  $F$ -меру.

## Макроусреднение

- ✓ Считаем Precision, Recall для каждого классификатора отдельно. Затем усредняем.

$$\text{Precision}_k = \frac{TP_k}{TP_k + FP_k}, \quad \text{Recall}_k = \frac{TP_k}{TP_k + FN_k}, \quad k \in 1 : m, \quad \text{Precision} = \frac{1}{m} \sum_{i=1}^m \text{Precision}_i, \quad \text{Recall} = \frac{1}{m} \sum_{i=1}^m \text{Recall}_i. \quad (45)$$

- А** Порядок усреднения влияет на результат в случае дисбаланса классов.
- В** При дисбалансе классификатор редко предсказывает маленький класс. Значения TP и FN при классификации этого класса против остальных тоже будут малы. При микроусреднении вклад маленького класса в общую метрику — небольшой.
- С** Для макроусреднения среднее считается уже для нормированных величин — вклад каждого класса будет одинаковым.



*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

- 1 Исходный набор данных случайным образом делится на две части  $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{test}}$  — набор для обучения и набор для проверки.



*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

- 1 Исходный набор данных случайным образом делится на две части  $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{test}}$  — набор для обучения и набор для проверки.
- 2 Классификатор обучается, используя набор для обучения  $\mathbf{X}_{\text{train}}$  — *удержание*.

*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

- 1 Исходный набор данных случайным образом делится на две части  $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{test}}$  — набор для обучения и набор для проверки.
- 2 Классификатор обучается, используя набор для обучения  $\mathbf{X}_{\text{train}}$  — *удержание*.
- 3 Тестируется на наборе  $\mathbf{X}_{\text{test}}$  для проверки качества классификатора.

*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

- 1 Исходный набор данных случайным образом делится на две части  $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{test}}$  — набор для обучения и набор для проверки.
- 2 Классификатор обучается, используя набор для обучения  $\mathbf{X}_{\text{train}}$  — *удержание*.
- 3 Тестируется на наборе  $\mathbf{X}_{\text{test}}$  для проверки качества классификатора.

**Достоинства:** не требует больших вычислительных затрат (существенно для больших наборов данных).

*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

- 1 Исходный набор данных случайным образом делится на две части  $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{test}}$  — набор для обучения и набор для проверки.
- 2 Классификатор обучается, используя набор для обучения  $\mathbf{X}_{\text{train}}$  — *удержание*.
- 3 Тестируется на наборе  $\mathbf{X}_{\text{test}}$  для проверки качества классификатора.

**Достоинства:** не требует больших вычислительных затрат (существенно для больших наборов данных).

**Недостатки:** высокая дисперсия, результат существенно зависит от разбиения.

*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

- 1 Исходный набор данных случайным образом делится на две части  $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{test}}$  — набор для обучения и набор для проверки.
- 2 Классификатор обучается, используя набор для обучения  $\mathbf{X}_{\text{train}}$  — *удержание*.
- 3 Тестируется на наборе  $\mathbf{X}_{\text{test}}$  для проверки качества классификатора.

**Достоинства:** не требует больших вычислительных затрат (существенно для больших наборов данных).

**Недостатки:** высокая дисперсия, результат существенно зависит от разбиения.

## k-блочная кросс-валидация (k-Fold Cross Validation)

*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

- 1 Исходный набор данных случайным образом делится на две части  $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{test}}$  — набор для обучения и набор для проверки.
- 2 Классификатор обучается, используя набор для обучения  $\mathbf{X}_{\text{train}}$  — *удержание*.
- 3 Тестируется на наборе  $\mathbf{X}_{\text{test}}$  для проверки качества классификатора.

**Достоинства:** не требует больших вычислительных затрат (существенно для больших наборов данных).

**Недостатки:** высокая дисперсия, результат существенно зависит от разбиения.

## k-блочная кросс-валидация (k-Fold Cross Validation)

- 1 Обучающая выборка разбивается на  $k$  *непересекающихся* и *одинаковых* по объему частей. Эти части называются *фолдами* (fold).

*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

- 1 Исходный набор данных случайным образом делится на две части  $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{test}}$  — набор для обучения и набор для проверки.
- 2 Классификатор обучается, используя набор для обучения  $\mathbf{X}_{\text{train}}$  — *удержание*.
- 3 Тестируется на наборе  $\mathbf{X}_{\text{test}}$  для проверки качества классификатора.

**Достоинства:** не требует больших вычислительных затрат (существенно для больших наборов данных).

**Недостатки:** высокая дисперсия, результат существенно зависит от разбиения.

## k-блочная кросс-валидация (k-Fold Cross Validation)

- 1 Обучающая выборка разбивается на  $k$  *непересекающихся* и *одинаковых* по объему частей. Эти части называются *фолдами* (fold).
- 2 Проводим  $k$  итераций. На каждой итерации происходит следующее:

*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

- 1 Исходный набор данных случайным образом делится на две части  $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{test}}$  — набор для обучения и набор для проверки.
- 2 Классификатор обучается, используя набор для обучения  $\mathbf{X}_{\text{train}}$  — *удержание*.
- 3 Тестируется на наборе  $\mathbf{X}_{\text{test}}$  для проверки качества классификатора.

**Достоинства:** не требует больших вычислительных затрат (существенно для больших наборов данных).

**Недостатки:** высокая дисперсия, результат существенно зависит от разбиения.

## k-блочная кросс-валидация (k-Fold Cross Validation)

- 1 Обучающая выборка разбивается на  $k$  *непересекающихся* и *одинаковых* по объему частей. Эти части называются *фолдами* (fold).
- 2 Проводим  $k$  итераций. На каждой итерации происходит следующее:
  - ✓ Классификатор обучается на  $k - 1$  части обучающей выборки.



*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

- 1 Исходный набор данных случайным образом делится на две части  $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{test}}$  — набор для обучения и набор для проверки.
- 2 Классификатор обучается, используя набор для обучения  $\mathbf{X}_{\text{train}}$  — *удержание*.
- 3 Тестируется на наборе  $\mathbf{X}_{\text{test}}$  для проверки качества классификатора.

**Достоинства:** не требует больших вычислительных затрат (существенно для больших наборов данных).

**Недостатки:** высокая дисперсия, результат существенно зависит от разбиения.

## k-блочная кросс-валидация (k-Fold Cross Validation)

- 1 Обучающая выборка разбивается на  $k$  *непересекающихся* и *одинаковых* по объему частей. Эти части называются *фолдами* (fold).
- 2 Проводим  $k$  итераций. На каждой итерации происходит следующее:
  - ✓ Классификатор обучается на  $k - 1$  части обучающей выборки.
  - ✓ Классификатор тестируется на части обучающей выборки, которая *не участвовала* в обучении.

*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

- 1 Исходный набор данных случайным образом делится на две части  $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{test}}$  — набор для обучения и набор для проверки.
- 2 Классификатор обучается, используя набор для обучения  $\mathbf{X}_{\text{train}}$  — *удержание*.
- 3 Тестируется на наборе  $\mathbf{X}_{\text{test}}$  для проверки качества классификатора.

**Достоинства:** не требует больших вычислительных затрат (существенно для больших наборов данных).

**Недостатки:** высокая дисперсия, результат существенно зависит от разбиения.

## k-блочная кросс-валидация (k-Fold Cross Validation)

- 1 Обучающая выборка разбивается на  $k$  *непересекающихся* и *одинаковых* по объему частей. Эти части называются *фолдами* (fold).
- 2 Проводим  $k$  итераций. На каждой итерации происходит следующее:
  - ✓ Классификатор обучается на  $k - 1$  части обучающей выборки.
  - ✓ Классификатор тестируется на части обучающей выборки, которая *не участвовала* в обучении.
- 3 Финальный результат получается усреднением  $k$  получившихся тестовых результатов.

*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

- 1 Исходный набор данных случайным образом делится на две части  $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{test}}$  — набор для обучения и набор для проверки.
- 2 Классификатор обучается, используя набор для обучения  $\mathbf{X}_{\text{train}}$  — *удержание*.
- 3 Тестируется на наборе  $\mathbf{X}_{\text{test}}$  для проверки качества классификатора.

**Достоинства:** не требует больших вычислительных затрат (существенно для больших наборов данных).

**Недостатки:** высокая дисперсия, результат существенно зависит от разбиения.

## k-блочная кросс-валидация (k-Fold Cross Validation)

- 1 Обучающая выборка разбивается на  $k$  *непересекающихся* и *одинаковых* по объему частей. Эти части называются *фолдами* (fold).
- 2 Проводим  $k$  итераций. На каждой итерации происходит следующее:
  - ✓ Классификатор обучается на  $k - 1$  части обучающей выборки.
  - ✓ Классификатор тестируется на части обучающей выборки, которая *не участвовала* в обучении.
- 3 Финальный результат получается усреднением  $k$  получившихся тестовых результатов.
- i Каждая из  $k$  частей используется для тестирования один раз.

*Кросс-валидация (Cross Validation)* — метод для оценки качества работы классификатора на независимых данных.

## Метод удержания (Hold-Out Validation)

- 1 Исходный набор данных случайным образом делится на две части  $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{test}}$  — набор для обучения и набор для проверки.
- 2 Классификатор обучается, используя набор для обучения  $\mathbf{X}_{\text{train}}$  — *удержание*.
- 3 Тестируется на наборе  $\mathbf{X}_{\text{test}}$  для проверки качества классификатора.

**Достоинства:** не требует больших вычислительных затрат (существенно для больших наборов данных).

**Недостатки:** высокая дисперсия, результат существенно зависит от разбиения.

## k-блочная кросс-валидация (k-Fold Cross Validation)

- 1 Обучающая выборка разбивается на  $k$  *непересекающихся* и *одинаковых* по объему частей. Эти части называются *фолдами* (fold).
  - 2 Проводим  $k$  итераций. На каждой итерации происходит следующее:
    - ✓ Классификатор обучается на  $k - 1$  части обучающей выборки.
    - ✓ Классификатор тестируется на части обучающей выборки, которая *не участвовала* в обучении.
  - 3 Финальный результат получается усреднением  $k$  получившихся тестовых результатов.
- i Каждая из  $k$  частей используется для тестирования один раз.
  - i Обычно  $k = 10$  (в случае малого размера выборки  $k = 5$ ).



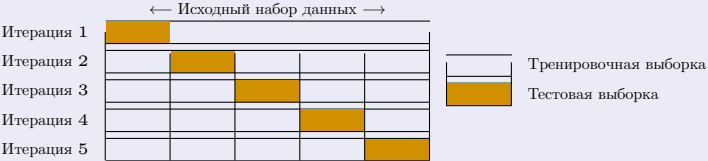


Рис. 20. Схема k-блочной кросс-валидации ( $k = 5$ )



Рис. 20. Схема k-блочной кросс-валидации ( $k = 5$ )

## Стратификация (stratification)

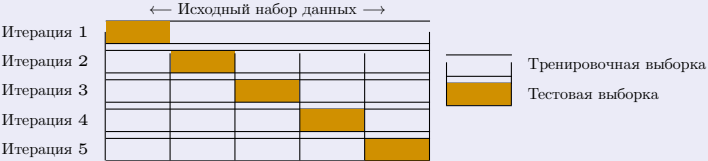


Рис. 20. Схема k-блочной кросс-валидации ( $k = 5$ )

## Стратификация (stratification)

? Что делать при большом дисбалансе классов?



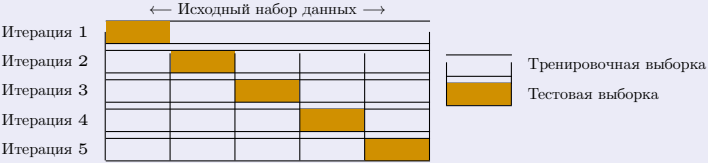


Рис. 20. Схема k-блочной кросс-валидации ( $k = 5$ )

## Стратификация (stratification)

- ? Что делать при большом дисбалансе классов?
- \* Изменение в методике: каждый блок содержит примерно такую же пропорцию классов, как в исходной выборке — *стратифицированная кросс-валидация по k блокам*.



Рис. 20. Схема k-блочной кросс-валидации ( $k = 5$ )

## Стратификация (stratification)

- ? Что делать при большом дисбалансе классов?
- \* Изменение в методике: каждый блок содержит примерно такую же пропорцию классов, как в исходной выборке — *стратифицированная кросс-валидация по k блокам*.

## Поэлементная кросс-валидация (Leave-One-Out)

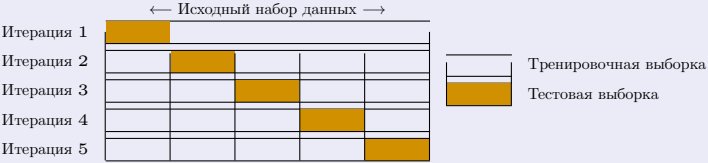


Рис. 20. Схема k-блочной кросс-валидации ( $k = 5$ )

## Стратификация (stratification)

- ? Что делать при большом дисбалансе классов?
- \* Изменение в методике: каждый блок содержит примерно такую же пропорцию классов, как в исходной выборке — *стратифицированная кросс-валидация по k блокам*.

## Поэлементная кросс-валидация (Leave-One-Out)

- 1 Частный случай метода k-блочной кросс-валидации, где размер фолда — 1. Остальные элементы исходного множества составляют тестовую выборку. Число итераций равно количеству точек данных в исходной выборке.

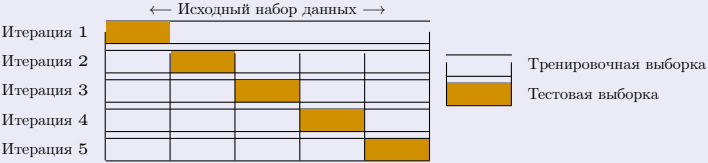


Рис. 20. Схема k-блочной кросс-валидации ( $k = 5$ )

## Стратификация (stratification)

- ? Что делать при большом дисбалансе классов?
- \* Изменение в методике: каждый блок содержит примерно такую же пропорцию классов, как в исходной выборке — *стратифицированная кросс-валидация по k блокам*.

## Поэлементная кросс-валидация (Leave-One-Out)

- 1 Частный случай метода k-блочной кросс-валидации, где размер фолда — 1. Остальные элементы исходного множества составляют тестовую выборку. Число итераций равно количеству точек данных в исходной выборке.
- i Недостаток — большой объем вычислений.



```
# Load dataset
from sklearn.datasets import load_iris
iris = load_iris(); X = iris.data; y = iris.target

# Load model
from sklearn.ensemble import RandomForestClassifier
rfm = RandomForestClassifier(n_estimators=100)

# 1. Hold-out cross-validation
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
rfm.fit(X_train, y_train)

# 2. K-fold cross-validation
from sklearn.model_selection import cross_val_score, KFold
kf = KFold(n_splits=5)
score = cross_val_score(rfm, X, y, cv=kf)

# 3. Stratified k-fold cross-validation
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits = 3)
score = cross_val_score(rfm, X, y, cv=skf)

# 4. Leave-One-Out Cross-Validation
from sklearn.model_selection import LeaveOneOut
LOOCV=LeaveOneOut()
score = cross_val_score(rfm, X, y, cv=LOOCV)
```



## Постановка задачи

- 1 В обучающем наборе данных доля объектов некоторого класса слишком низкая – **миноритарный** класс (англ. minority), а другого — слишком большая — **мажоритарный** класс (англ. majority).



## Постановка задачи

- 1 В обучающем наборе данных доля объектов некоторого класса слишком низкая – **миноритарный** класс (англ. minority), а другого — слишком большая — **мажоритарный** класс (англ. majority).
- 2 Эта ситуация — несбалансированность классов (**class imbalance**), а классификация в условиях несбалансированности классов — несбалансированная классификация (**unbalanced classification**).

## Постановка задачи

- 1 В обучающем наборе данных доля объектов некоторого класса слишком низкая – **миноритарный** класс (англ. minority), а другого — слишком большая — **мажоритарный** класс (англ. majority).
- 2 Эта ситуация — несбалансированность классов (**class imbalance**), а классификация в условиях несбалансированности классов — несбалансированная классификация (**unbalanced classification**).
- 3 Несбалансированность классов создает проблемы при решении задач классификации.

## Постановка задачи

- ❶ В обучающем наборе данных доля объектов некоторого класса слишком низкая – **миноритарный** класс (англ. minority), а другого — слишком большая — **мажоритарный** класс (англ. majority).
- ❷ Эта ситуация — несбалансированность классов (**class imbalance**), а классификация в условиях несбалансированности классов — несбалансированная классификация (**unbalanced classification**).
- ❸ Несбалансированность классов создает проблемы при решении задач классификации.
  - ✓ Построенные на таких данных модели имеют «перекос» в сторону мажоритарного класса, т.е. с большей вероятностью присваивают его метку класса новым наблюдениям при практическом использовании модели. Это — переоценка (**overestimation**).

## Постановка задачи

- ❶ В обучающем наборе данных доля объектов некоторого класса слишком низкая – **миноритарный** класс (англ. minority), а другого — слишком большая — **мажоритарный** класс (англ. majority).
- ❷ Эта ситуация — несбалансированность классов (**class imbalance**), а классификация в условиях несбалансированности классов — несбалансированная классификация (**unbalanced classification**).
- ❸ Несбалансированность классов создает проблемы при решении задач классификации.
  - ✓ Построенные на таких данных модели имеют «перекос» в сторону мажоритарного класса, т.е. с большей вероятностью присваивают его метку класса новым наблюдениям при практическом использовании модели. Это — переоценка (**overestimation**).
  - ✓ Если модель построена так, что отдает предпочтение миноритарному классу, это — недооценка (**underestimation**).

## Постановка задачи

- 1 В обучающем наборе данных доля объектов некоторого класса слишком низкая – **миноритарный** класс (англ. minority), а другого — слишком большая — **мажоритарный** класс (англ. majority).
- 2 Эта ситуация — несбалансированность классов (**class imbalance**), а классификация в условиях несбалансированности классов — несбалансированная классификация (**unbalanced classification**).
- 3 Несбалансированность классов создает проблемы при решении задач классификации.
  - ✓ Построенные на таких данных модели имеют «перекос» в сторону мажоритарного класса, т.е. с большей вероятностью присваивают его метку класса новым наблюдениям при практическом использовании модели. Это — переоценка (**overestimation**).
  - ✓ Если модель построена так, что отдает предпочтение миноритарному классу, это — недооценка (**underestimation**).
- i Цена ошибочной классификации может быть разной. Неверная классификация объектов миноритарного класса часто обходится значительно дороже, чем ошибочная классификация объектов мажоритарного класса. Например, при медицинских или финансовых исследованиях.

## Постановка задачи

- 1 В обучающем наборе данных доля объектов некоторого класса слишком низкая – **миноритарный** класс (англ. minority), а другого — слишком большая — **мажоритарный** класс (англ. majority).
- 2 Эта ситуация — несбалансированность классов (**class imbalance**), а классификация в условиях несбалансированности классов — несбалансированная классификация (**unbalanced classification**).
- 3 Несбалансированность классов создает проблемы при решении задач классификации.
  - ✓ Построенные на таких данных модели имеют «перекос» в сторону мажоритарного класса, т.е. с большей вероятностью присваивают его метку класса новым наблюдениям при практическом использовании модели. Это — переоценка (**overestimation**).
  - ✓ Если модель построена так, что отдает предпочтение миноритарному классу, это — недооценка (**underestimation**).
- i Цена ошибочной классификации может быть разной. Неверная классификация объектов миноритарного класса часто обходится значительно дороже, чем ошибочная классификация объектов мажоритарного класса. Например, при медицинских или финансовых исследованиях.
- 4 Методы балансировки можно разделить на случайные и специальные.

## Постановка задачи

- 1 В обучающем наборе данных доля объектов некоторого класса слишком низкая – **миноритарный** класс (англ. minority), а другого — слишком большая — **мажоритарный** класс (англ. majority).
- 2 Эта ситуация — несбалансированность классов (**class imbalance**), а классификация в условиях несбалансированности классов — несбалансированная классификация (**unbalanced classification**).
- 3 Несбалансированность классов создает проблемы при решении задач классификации.
  - ✓ Построенные на таких данных модели имеют «перекос» в сторону мажоритарного класса, т.е. с большей вероятностью присваивают его метку класса новым наблюдениям при практическом использовании модели. Это — переоценка (**overestimation**).
  - ✓ Если модель построена так, что отдает предпочтение миноритарному классу, это — недооценка (**underestimation**).
- i Цена ошибочной классификации может быть разной. Неверная классификация объектов миноритарного класса часто обходится значительно дороже, чем ошибочная классификация объектов мажоритарного класса. Например, при медицинских или финансовых исследованиях.
- 4 Методы балансировки можно разделить на случайные и специальные.
- 5 Балансировка классов производится:

## Постановка задачи

- 1 В обучающем наборе данных доля объектов некоторого класса слишком низкая – **миноритарный** класс (англ. minority), а другого — слишком большая — **мажоритарный** класс (англ. majority).
- 2 Эта ситуация — несбалансированность классов (**class imbalance**), а классификация в условиях несбалансированности классов — несбалансированная классификация (**unbalanced classification**).
- 3 Несбалансированность классов создает проблемы при решении задач классификации.
  - ✓ Построенные на таких данных модели имеют «перекос» в сторону мажоритарного класса, т.е. с большей вероятностью присваивают его метку класса новым наблюдениям при практическом использовании модели. Это — переоценка (**overestimation**).
  - ✓ Если модель построена так, что отдает предпочтение миноритарному классу, это — недооценка (**underestimation**).
- i Цена ошибочной классификации может быть разной. Неверная классификация объектов миноритарного класса часто обходится значительно дороже, чем ошибочная классификация объектов мажоритарного класса. Например, при медицинских или финансовых исследованиях.
- 4 Методы балансировки можно разделить на случайные и специальные.
- 5 Балансировка классов производится:
  - ✓ увеличением числа объектов миноритарного класса — передискретизация (**undersampling**);



## Постановка задачи

- 1 В обучающем наборе данных доля объектов некоторого класса слишком низкая – **миноритарный** класс (англ. minority), а другого — слишком большая — **мажоритарный** класс (англ. majority).
- 2 Эта ситуация — несбалансированность классов (**class imbalance**), а классификация в условиях несбалансированности классов — несбалансированная классификация (**unbalanced classification**).
- 3 Несбалансированность классов создает проблемы при решении задач классификации.
  - ✓ Построенные на таких данных модели имеют «перекос» в сторону мажоритарного класса, т.е. с большей вероятностью присваивают его метку класса новым наблюдениям при практическом использовании модели. Это — переоценка (**overestimation**).
  - ✓ Если модель построена так, что отдает предпочтение миноритарному классу, это — недооценка (**underestimation**).
- i Цена ошибочной классификации может быть разной. Неверная классификация объектов миноритарного класса часто обходится значительно дороже, чем ошибочная классификация объектов мажоритарного класса. Например, при медицинских или финансовых исследованиях.
- 4 Методы балансировки можно разделить на случайные и специальные.
- 5 Балансировка классов производится:
  - ✓ увеличением числа объектов миноритарного класса — передискретизация (**undersampling**);
  - ✓ сокращением числа объектов мажоритарного класса — субдискретизация (**oversampling**);

## Постановка задачи

- 1 В обучающем наборе данных доля объектов некоторого класса слишком низкая – **миноритарный** класс (англ. minority), а другого — слишком большая — **мажоритарный** класс (англ. majority).
- 2 Эта ситуация — несбалансированность классов (**class imbalance**), а классификация в условиях несбалансированности классов — несбалансированная классификация (**unbalanced classification**).
- 3 Несбалансированность классов создает проблемы при решении задач классификации.
  - ✓ Построенные на таких данных модели имеют «перекос» в сторону мажоритарного класса, т.е. с большей вероятностью присваивают его метку класса новым наблюдениям при практическом использовании модели. Это — переоценка (**overestimation**).
  - ✓ Если модель построена так, что отдает предпочтение миноритарному классу, это — недооценка (**underestimation**).
- i Цена ошибочной классификации может быть разной. Неверная классификация объектов миноритарного класса часто обходится значительно дороже, чем ошибочная классификация объектов мажоритарного класса. Например, при медицинских или финансовых исследованиях.
- 4 Методы балансировки можно разделить на случайные и специальные.
- 5 Балансировка классов производится:
  - ✓ увеличением числа объектов миноритарного класса — передискретизация (**undersampling**);
  - ✓ сокращением числа объектов мажоритарного класса — субдискретизация (**oversampling**);
  - ✓ иногда применяют сочетание обоих подходов.



**i** Загружаем необходимые модули и создаем несбалансированный набор данных (90% : 10%). Баланс классов визуализируем с помощью **Yellowbrick**.

```
from yellowbrick.target import ClassBalance
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import CondensedNearestNeighbour
from imblearn.under_sampling import TomekLinks
from imblearn.under_sampling import OneSidedSelection
from imblearn.under_sampling import NeighbourhoodCleaningRule
X, y = make_classification(n_samples=200, n_features=2, n_redundant=0, n_clusters_per_class=1, weights=[0.90],
                          flip_y=0, random_state=1)
```

- i** Загружаем необходимые модули и создаем несбалансированный набор данных (90% : 10%). Баланс классов визуализируем с помощью **Yellowbrick**.

```
from yellowbrick.target import ClassBalance
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import CondensedNearestNeighbour
from imblearn.under_sampling import TomekLinks
from imblearn.under_sampling import OneSidedSelection
from imblearn.under_sampling import NeighbourhoodCleaningRule
X, y = make_classification(n_samples=200, n_features=2, n_redundant=0, n_clusters_per_class=1, weights=[0.90],
                          flip_y=0, random_state=1)
```

- 1** Случайное удаление объектов мажоритарного класса (**Random Undersampling**)

- i** Загружаем необходимые модули и создаем несбалансированный набор данных (90% : 10%). Баланс классов визуализируем с помощью **Yellowbrick**.

```
from yellowbrick.target import ClassBalance
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import CondensedNearestNeighbour
from imblearn.under_sampling import TomekLinks
from imblearn.under_sampling import OneSidedSelection
from imblearn.under_sampling import NeighbourhoodCleaningRule
X, y = make_classification(n_samples=200, n_features=2, n_redundant=0, n_clusters_per_class=1, weights=[0.90],
                          flip_y=0, random_state=1)
```

- 1** Случайное удаление объектов мажоритарного класса (**Random Undersampling**)

Рассчитывается число  $N$  мажоритарных объектов, которое необходимо удалить для достижения требуемого уровня соотношения различных классов. Случайным образом выбираются и удаляются  $N$  мажоритарных объектов.

```
undersample = RandomUnderSampler(random_state=0)
X, y = undersample.fit_resample(X, y)
```

- i** Загружаем необходимые модули и создаем несбалансированный набор данных (90% : 10%). Баланс классов визуализируем с помощью **Yellowbrick**.

```
from yellowbrick.target import ClassBalance
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import CondensedNearestNeighbour
from imblearn.under_sampling import TomekLinks
from imblearn.under_sampling import OneSidedSelection
from imblearn.under_sampling import NeighbourhoodCleaningRule
X, y = make_classification(n_samples=200, n_features=2, n_redundant=0, n_clusters_per_class=1, weights=[0.90],
                          flip_y=0, random_state=1)
```

- 1** Случайное удаление объектов мажоритарного класса (**Random Undersampling**)

Рассчитывается число  $N$  мажоритарных объектов, которое необходимо удалить для достижения требуемого уровня соотношения различных классов. Случайным образом выбираются и удаляются  $N$  мажоритарных объектов.

```
undersample = RandomUnderSampler(random_state=0)
X, y = undersample.fit_resample(X, y)
```

- 2** Поиск связей Томека (**Tomek Links**)

- i** Загружаем необходимые модули и создаем несбалансированный набор данных (90% : 10%). Баланс классов визуализируем с помощью **Yellowbrick**.

```
from yellowbrick.target import ClassBalance
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import CondensedNearestNeighbour
from imblearn.under_sampling import TomekLinks
from imblearn.under_sampling import OneSidedSelection
from imblearn.under_sampling import NeighbourhoodCleaningRule
X, y = make_classification(n_samples=200, n_features=2, n_redundant=0, n_clusters_per_class=1, weights=[0.90],
                          flip_y=0, random_state=1)
```

- 1** Случайное удаление объектов мажоритарного класса (**Random Undersampling**)

Рассчитывается число  $N$  мажоритарных объектов, которое необходимо удалить для достижения требуемого уровня соотношения различных классов. Случайным образом выбираются и удаляются  $N$  мажоритарных объектов.

```
undersample = RandomUnderSampler(random_state=0)
X, y = undersample.fit_resample(X, y)
```

- 2** Поиск связей Томека (**Tomek Links**)

Пусть объекты  $X_i$  и  $X_j$  принадлежат различным классам и расстояние между ними  $\rho(X_i, X_j)$ .



# Сокращение мажоритарного класса

- i** Загружаем необходимые модули и создаем несбалансированный набор данных (90% : 10%). Баланс классов визуализируем с помощью **Yellowbrick**.

```
from yellowbrick.target import ClassBalance
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import CondensedNearestNeighbour
from imblearn.under_sampling import TomekLinks
from imblearn.under_sampling import OneSidedSelection
from imblearn.under_sampling import NeighbourhoodCleaningRule
X, y = make_classification(n_samples=200, n_features=2, n_redundant=0, n_clusters_per_class=1, weights=[0.90],
                          flip_y=0, random_state=1)
```

- 1** Случайное удаление объектов мажоритарного класса (**Random Undersampling**)

Рассчитывается число  $N$  мажоритарных объектов, которое необходимо удалить для достижения требуемого уровня соотношения различных классов. Случайным образом выбираются и удаляются  $N$  мажоритарных объектов.

```
undersample = RandomUnderSampler(random_state=0)
X, y = undersample.fit_resample(X, y)
```

- 2** Поиск связей Томека (**Tomek Links**)

Пусть объекты  $X_i$  и  $X_j$  принадлежат различным классам и расстояние между ними  $\rho(X_i, X_j)$ .

Пара  $(X_i, X_j)$  называется **связью Томека**, если они относятся к разным классам и не существует объекта  $X_k$  такого, что

$$\rho(X_i, X_k) < \rho(X_i, X_j) \quad \text{или} \quad \rho(X_j, X_k) < \rho(X_j, X_i).$$

# Сокращение мажоритарного класса

- i** Загружаем необходимые модули и создаем несбалансированный набор данных (90% : 10%). Баланс классов визуализируем с помощью **Yellowbrick**.

```
from yellowbrick.target import ClassBalance
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import CondensedNearestNeighbour
from imblearn.under_sampling import TomekLinks
from imblearn.under_sampling import OneSidedSelection
from imblearn.under_sampling import NeighbourhoodCleaningRule
X, y = make_classification(n_samples=200, n_features=2, n_redundant=0, n_clusters_per_class=1, weights=[0.90],
                          flip_y=0, random_state=1)
```

- 1** Случайное удаление объектов мажоритарного класса (**Random Undersampling**)

Рассчитывается число  $N$  мажоритарных объектов, которое необходимо удалить для достижения требуемого уровня соотношения различных классов. Случайным образом выбираются и удаляются  $N$  мажоритарных объектов.

```
undersample = RandomUnderSampler(random_state=0)
X, y = undersample.fit_resample(X, y)
```

- 2** Поиск связей Томека (**Tomek Links**)

Пусть объекты  $X_i$  и  $X_j$  принадлежат различным классам и расстояние между ними  $\rho(X_i, X_j)$ .

Пара  $(X_i, X_j)$  называется **связью Томека**, если они относятся к разным классам и не существует объекта  $X_k$  такого, что

$$\rho(X_i, X_k) < \rho(X_i, X_j) \quad \text{или} \quad \rho(X_j, X_k) < \rho(X_j, X_i).$$

Связи Томека объединяют близко расположенные объекты различных классов.

- i** Загружаем необходимые модули и создаем несбалансированный набор данных (90% : 10%). Баланс классов визуализируем с помощью **Yellowbrick**.

```
from yellowbrick.target import ClassBalance
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import CondensedNearestNeighbour
from imblearn.under_sampling import TomekLinks
from imblearn.under_sampling import OneSidedSelection
from imblearn.under_sampling import NeighbourhoodCleaningRule
X, y = make_classification(n_samples=200, n_features=2, n_redundant=0, n_clusters_per_class=1, weights=[0.90],
                          flip_y=0, random_state=1)
```

- 1** Случайное удаление объектов мажоритарного класса (**Random Undersampling**)

Рассчитывается число  $N$  мажоритарных объектов, которое необходимо удалить для достижения требуемого уровня соотношения различных классов. Случайным образом выбираются и удаляются  $N$  мажоритарных объектов.

```
undersample = RandomUnderSampler(random_state=0)
X, y = undersample.fit_resample(X, y)
```

- 2** Поиск связей Томека (**Tomek Links**)

Пусть объекты  $X_i$  и  $X_j$  принадлежат различным классам и расстояние между ними  $\rho(X_i, X_j)$ .

Пара  $(X_i, X_j)$  называется **связью Томека**, если они относятся к разным классам и не существует объекта  $X_k$  такого, что

$$\rho(X_i, X_k) < \rho(X_i, X_j) \quad \text{или} \quad \rho(X_j, X_k) < \rho(X_j, X_i).$$

Связи Томека объединяют близко расположенные объекты различных классов.

Все объекты мажоритарного класса, со связями Томека удаляются из набора данных.

```
undersample = TomekLinks()
X, y = undersample.fit_resample(X, y)
```



## 3 Правило сосредоточенного ближайшего соседа (**Condensed Nearest Neighbor Rule**)

## 3 Правило сосредоточенного ближайшего соседа (**Condensed Nearest Neighbor Rule**)

- ✓ Из исходного множество объектов  $X$  извлекаются все объекты миноритарного класса и (случайным образом) один мажоритарный. Полученное множество, обозначим как  $S$ .

## 3 Правило сосредоточенного ближайшего соседа (**Condensed Nearest Neighbor Rule**)

- ✓ Из исходного множество объектов  $X$  извлекаются все объекты миноритарного класса и (случайным образом) один мажоритарный. Полученное множество, обозначим как  $S$ .
- ✓ Все объекты из  $X$  классифицируются по правилу одного ближайшего соседа (1-NN).

## 3 Правило сосредоточенного ближайшего соседа (**Condensed Nearest Neighbor Rule**)

- ✓ Из исходного множество объектов  $X$  извлекаются все объекты миноритарного класса и (случайным образом) один мажоритарный. Полученное множество, обозначим как  $S$ .
- ✓ Все объекты из  $X$  классифицируются по правилу одного ближайшего соседа (1-NN).
- ✓ Объекты, получившие ошибочную метку класса, добавляются во множество  $S$ . В множестве  $S$  достигается баланс классов.

```
undersample = CondensedNearestNeighbour(n_neighbors=1)  
X, y = undersample.fit_resample(X, y)
```



## 3 Правило сосредоточенного ближайшего соседа (**Condensed Nearest Neighbor Rule**)

- ✓ Из исходного множество объектов  $X$  извлекаются все объекты миноритарного класса и (случайным образом) один мажоритарный. Полученное множество, обозначим как  $S$ .
- ✓ Все объекты из  $X$  классифицируются по правилу одного ближайшего соседа (1-NN).
- ✓ Объекты, получившие ошибочную метку класса, добавляются во множество  $S$ . В множестве  $S$  достигается баланс классов.

```
undersample = CondensedNearestNeighbour(n_neighbors=1)  
X, y = undersample.fit_resample(X, y)
```

## 4 Односторонний сэмплинг (**One-side sampling, one-sided selection — OSS**). Сочетание двух предыдущих подходов.

## 3 Правило сосредоточенного ближайшего соседа (**Condensed Nearest Neighbor Rule**)

- ✓ Из исходного множество объектов  $X$  извлекаются все объекты миноритарного класса и (случайным образом) один мажоритарный. Полученное множество, обозначим как  $S$ .
- ✓ Все объекты из  $X$  классифицируются по правилу одного ближайшего соседа (1-NN).
- ✓ Объекты, получившие ошибочную метку класса, добавляются во множество  $S$ . В множестве  $S$  достигается баланс классов.

```
undersample = CondensedNearestNeighbour(n_neighbors=1)  
X, y = undersample.fit_resample(X, y)
```

## 4 Односторонний сэмплинг (**One-side sampling, one-sided selection — OSS**). Сочетание двух предыдущих подходов.

- ✓ Сначала применяется правило сосредоточенного ближайшего соседа.

# Сокращение мажоритарного класса (2)

## 3 Правило сосредоточенного ближайшего соседа (**Condensed Nearest Neighbor Rule**)

- ✓ Из исходного множество объектов  $X$  извлекаются все объекты миноритарного класса и (случайным образом) один мажоритарный. Полученное множество, обозначим как  $S$ .
- ✓ Все объекты из  $X$  классифицируются по правилу одного ближайшего соседа (1-NN).
- ✓ Объекты, получившие ошибочную метку класса, добавляются во множество  $S$ . В множестве  $S$  достигается баланс классов.

```
undersample = CondensedNearestNeighbour(n_neighbors=1)
X, y = undersample.fit_resample(X, y)
```

## 4 Односторонний сэмплинг (**One-side sampling, one-sided selection — OSS**). Сочетание двух предыдущих подходов.

- ✓ Сначала применяется правило сосредоточенного ближайшего соседа.
- ✓ Затем удаляются все мажоритарные объекты со связями Томека.

```
undersample = OneSidedSelection()
X3, y3 = undersample.fit_resample(X, y)
```

# Сокращение мажоритарного класса (2)

## 3 Правило сосредоточенного ближайшего соседа (**Condensed Nearest Neighbor Rule**)

- ✓ Из исходного множество объектов  $X$  извлекаются все объекты миноритарного класса и (случайным образом) один мажоритарный. Полученное множество, обозначим как  $S$ .
- ✓ Все объекты из  $X$  классифицируются по правилу одного ближайшего соседа (1-NN).
- ✓ Объекты, получившие ошибочную метку класса, добавляются во множество  $S$ . В множестве  $S$  достигается баланс классов.

```
undersample = CondensedNearestNeighbour(n_neighbors=1)  
X, y = undersample.fit_resample(X, y)
```

## 4 Односторонний сэмплинг (**One-side sampling, one-sided selection — OSS**). Сочетание двух предыдущих подходов.

- ✓ Сначала применяется правило сосредоточенного ближайшего соседа.
- ✓ Затем удаляются все мажоритарные объекты со связями Томека.

```
undersample = OneSidedSelection()  
X3, y3 = undersample.fit_resample(X, y)
```

## 5 Правило «очищающего» соседа (**Neighborhood cleaning rule — NCR**)

# Сокращение мажоритарного класса (2)

## 3 Правило сосредоточенного ближайшего соседа (**Condensed Nearest Neighbor Rule**)

- ✓ Из исходного множество объектов  $X$  извлекаются все объекты миноритарного класса и (случайным образом) один мажоритарный. Полученное множество, обозначим как  $S$ .
- ✓ Все объекты из  $X$  классифицируются по правилу одного ближайшего соседа (1-NN).
- ✓ Объекты, получившие ошибочную метку класса, добавляются во множество  $S$ . В множестве  $S$  достигается баланс классов.

```
undersample = CondensedNearestNeighbour(n_neighbors=1)  
X, y = undersample.fit_resample(X, y)
```

## 4 Односторонний сэмплинг (**One-side sampling, one-sided selection — OSS**). Сочетание двух предыдущих подходов.

- ✓ Сначала применяется правило сосредоточенного ближайшего соседа.
- ✓ Затем удаляются все мажоритарные объекты со связями Томека.

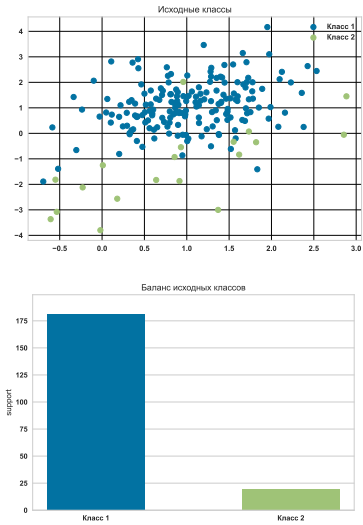
```
undersample = OneSidedSelection()  
X3, y3 = undersample.fit_resample(X, y)
```

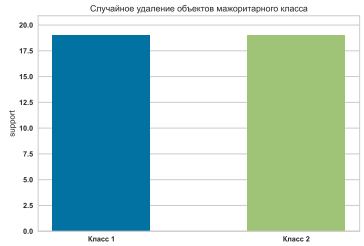
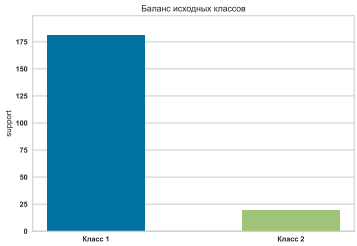
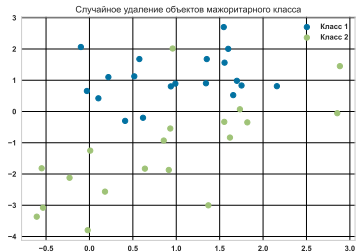
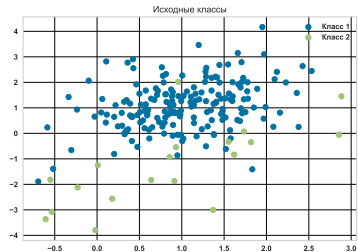
## 5 Правило «очищающего» соседа (**Neighborhood cleaning rule — NCR**)

Все объекты классифицируются по правилу трех ближайших соседей (3-NN). Удаляются мажоритарные объекты: получившие правильную метку класса и соседи миноритарных объектов, неверно классифицированных.

```
undersample = NeighbourhoodCleaningRule()  
X, y = undersample.fit_resample(X, y)
```

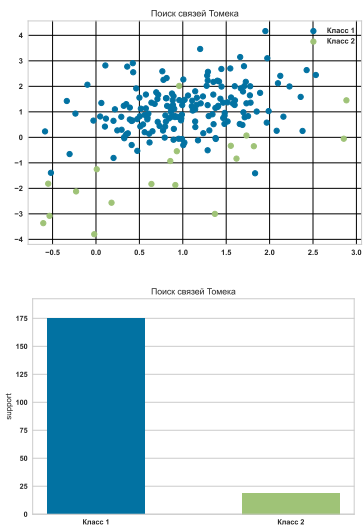


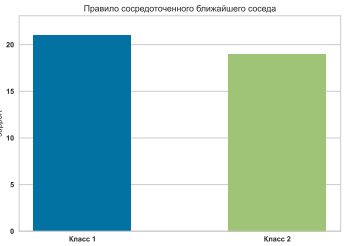
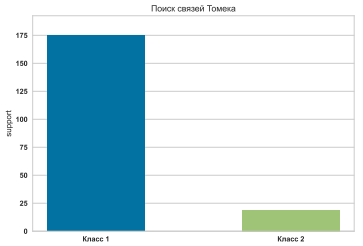
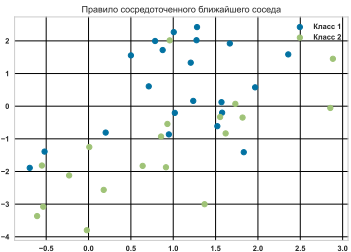
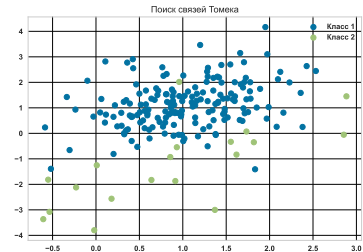




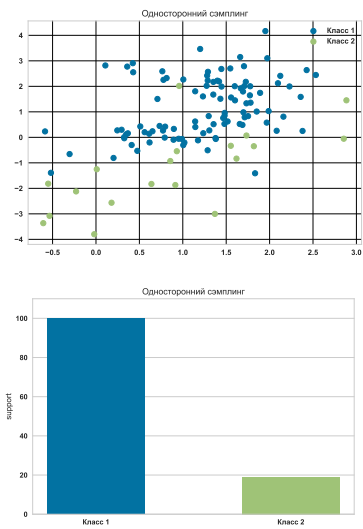


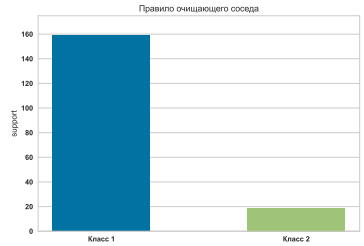
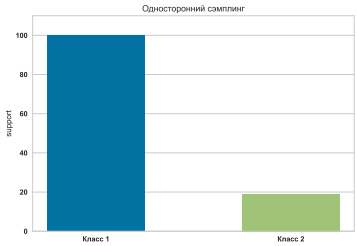
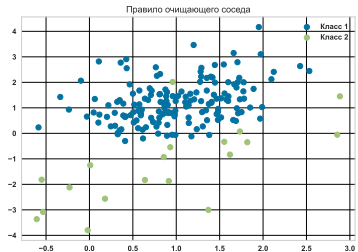
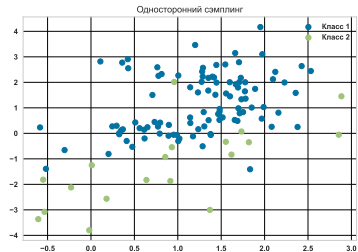














- 1 Дублирование примеров миноритарного класса (Oversampling). В зависимости от необходимого соотношения классов, выбирается случайным образом соответствующее число объектов для дублирования.



- 1 Дублирование примеров миноритарного класса (Oversampling). В зависимости от необходимого соотношения классов, выбирается случайным образом соответствующее число объектов для дублирования.
- 2 Алгоритм SMOTE (**S**ynthetic **M**inority **O**versampling **T**echnique). Генерация некоторого количества искусственных объектов, **похожих** на миноритарные объекты, но не дублирующих их.

- 1 Дублирование примеров миноритарного класса (Oversampling). В зависимости от необходимого соотношения классов, выбирается случайным образом соответствующее число объектов для дублирования.
- 2 Алгоритм SMOTE (**S**ynthetic **M**inority **O**versampling **T**echnique). Генерация некоторого количества искусственных объектов, **похожих** на миноритарные объекты, но не дублирующих их.
  - ✓ Находим разность  $\mathbf{D} = \mathbf{X}_b - \mathbf{X}_a$ , где  $\mathbf{X}_a, \mathbf{X}_b$  — вектора признаков соседних объектов из миноритарного класса, полученные алгоритмом ближайшего соседа KNN.

- 1 Дублирование примеров миноритарного класса (Oversampling). В зависимости от необходимого соотношения классов, выбирается случайным образом соответствующее число объектов для дублирования.
- 2 Алгоритм SMOTE (**S**ynthetic **M**inority **O**versampling **T**echnique). Генерация некоторого количества искусственных объектов, **похожих** на миноритарные объекты, но не дублирующих их.
  - ✓ Находим разность  $\mathbf{D} = \mathbf{X}_b - \mathbf{X}_a$ , где  $\mathbf{X}_a, \mathbf{X}_b$  — вектора признаков соседних объектов из миноритарного класса, полученные алгоритмом ближайшего соседа KNN.
  - ✓ Получаем  $\hat{\mathbf{D}} = \alpha \mathbf{D}$ , где  $\alpha \in (0, 1)$  — случайное число.

- 1 Дублирование примеров миноритарного класса (Oversampling). В зависимости от необходимого соотношения классов, выбирается случайным образом соответствующее число объектов для дублирования.
- 2 Алгоритм SMOTE (**S**ynthetic **M**inority **O**versampling **T**echnique). Генерация некоторого количества искусственных объектов, **похожих** на миноритарные объекты, но не дублирующих их.
  - ✓ Находим разность  $\mathbf{D} = \mathbf{X}_b - \mathbf{X}_a$ , где  $\mathbf{X}_a, \mathbf{X}_b$  — вектора признаков соседних объектов из миноритарного класса, полученные алгоритмом ближайшего соседа KNN.
  - ✓ Получаем  $\hat{\mathbf{D}} = \alpha \mathbf{D}$ , где  $\alpha \in (0, 1)$  — случайное число.
  - ✓ Вычисляем вектор признаков нового объекта:  $\mathbf{X} = \mathbf{X}_a + \hat{\mathbf{D}}$ .

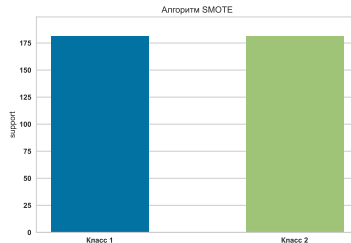
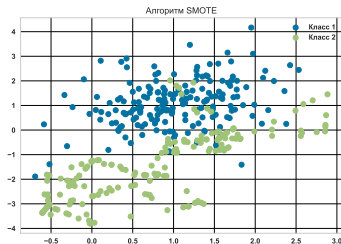
- 1 Дублирование примеров миноритарного класса (Oversampling). В зависимости от необходимого соотношения классов, выбирается случайным образом соответствующее число объектов для дублирования.
- 2 Алгоритм SMOTE (**S**ynthetic **M**inority **O**versampling **T**echnique). Генерация некоторого количества искусственных объектов, **похожих** на миноритарные объекты, но не дублирующих их.
  - ✓ Находим разность  $\mathbf{D} = \mathbf{X}_b - \mathbf{X}_a$ , где  $\mathbf{X}_a, \mathbf{X}_b$  — вектора признаков соседних объектов из миноритарного класса, полученные алгоритмом ближайшего соседа KNN.
  - ✓ Получаем  $\hat{\mathbf{D}} = \alpha \mathbf{D}$ , где  $\alpha \in (0, 1)$  — случайное число.
  - ✓ Вычисляем вектор признаков нового объекта:  $\mathbf{X} = \mathbf{X}_a + \hat{\mathbf{D}}$ .
  - ✓ Степень похожести объектов  $\mathbf{X}_a, \mathbf{X}_b$  можно регулировать путем изменения числа ближайших соседей алгоритма KNN.

# Увеличение миноритарного класса

1 Дублирование примеров миноритарного класса (Oversampling). В зависимости от необходимого соотношения классов, выбирается случайным образом соответствующее число объектов для дублирования.

2 Алгоритм SMOTE (Synthetic Minority Oversampling Technique). Генерация некоторого количества искусственных объектов, **похожих** на миноритарные объекты, но не дублирующих их.

- ✓ Находим разность  $\mathbf{D} = \mathbf{X}_b - \mathbf{X}_a$ , где  $\mathbf{X}_a, \mathbf{X}_b$  — вектора признаков соседних объектов из миноритарного класса, полученные алгоритмом ближайшего соседа KNN.
- ✓ Получаем  $\hat{\mathbf{D}} = \alpha \mathbf{D}$ , где  $\alpha \in (0, 1)$  — случайное число.
- ✓ Вычисляем вектор признаков нового объекта:  $\mathbf{X} = \mathbf{X}_a + \hat{\mathbf{D}}$ .
- ✓ Степень похожести объектов  $\mathbf{X}_a, \mathbf{X}_b$  можно регулировать путем изменения числа ближайших соседей алгоритма KNN.
- ✓ Пример работы алгоритма SMOTE для множества на рисунке 21 приведен на рисунке ниже.





**Анализ тональности.** Методы классификации применяются для анализа и классификации текстовых данных (например, отзывов клиентов, сообщений в социальных сетях) для определения настроений (по различным шкалам) и понимания общественного мнения.



**Анализ тональности.** Методы классификации применяются для анализа и классификации текстовых данных (например, отзывов клиентов, сообщений в социальных сетях) для определения настроений (по различным шкалам) и понимания общественного мнения.

**Классификация спама по электронной почте.** Является одним из наиболее распространенных способов применения классификации, помогает фильтровать нежелательные или вредоносные электронные письма.

**Анализ тональности.** Методы классификации применяются для анализа и классификации текстовых данных (например, отзывов клиентов, сообщений в социальных сетях) для определения настроений (по различным шкалам) и понимания общественного мнения.

**Классификация спама по электронной почте.** Является одним из наиболее распространенных способов применения классификации, помогает фильтровать нежелательные или вредоносные электронные письма.

**Классификация документов.** Упорядочение документов по категориям в соответствии с их содержанием. Автоматическая классификация документов, таких как новостные статьи, юридические документы или заявки в службу поддержки, на соответствующие категории, способствует эффективному поиску и организации документов.

**Анализ тональности.** Методы классификации применяются для анализа и классификации текстовых данных (например, отзывов клиентов, сообщений в социальных сетях) для определения настроений (по различным шкалам) и понимания общественного мнения.

**Классификация спама по электронной почте.** Является одним из наиболее распространенных способов применения классификации, помогает фильтровать нежелательные или вредоносные электронные письма.

**Классификация документов.** Упорядочение документов по категориям в соответствии с их содержанием. Автоматическая классификация документов, таких как новостные статьи, юридические документы или заявки в службу поддержки, на соответствующие категории, способствует эффективному поиску и организации документов.

**Классификация токсичных комментариев.** Классификации текстовых комментариев на токсичные и нетоксичные, помогает выявлять и модерировать вредный или оскорбительный контент на онлайн-платформах.

**Анализ тональности.** Методы классификации применяются для анализа и классификации текстовых данных (например, отзывов клиентов, сообщений в социальных сетях) для определения настроений (по различным шкалам) и понимания общественного мнения.

**Классификация спама по электронной почте.** Является одним из наиболее распространенных способов применения классификации, помогает фильтровать нежелательные или вредоносные электронные письма.

**Классификация документов.** Упорядочение документов по категориям в соответствии с их содержанием. Автоматическая классификация документов, таких как новостные статьи, юридические документы или заявки в службу поддержки, на соответствующие категории, способствует эффективному поиску и организации документов.

**Классификация токсичных комментариев.** Классификации текстовых комментариев на токсичные и нетоксичные, помогает выявлять и модерировать вредный или оскорбительный контент на онлайн-платформах.

**Классификация изображений.** Файлы изображения классифицируются по заранее определенным классам, часто используют с методами обнаружения объектов.

**Анализ тональности.** Методы классификации применяются для анализа и классификации текстовых данных (например, отзывов клиентов, сообщений в социальных сетях) для определения настроений (по различным шкалам) и понимания общественного мнения.

**Классификация спама по электронной почте.** Является одним из наиболее распространенных способов применения классификации, помогает фильтровать нежелательные или вредоносные электронные письма.

**Классификация документов.** Упорядочение документов по категориям в соответствии с их содержанием. Автоматическая классификация документов, таких как новостные статьи, юридические документы или заявки в службу поддержки, на соответствующие категории, способствует эффективному поиску и организации документов.

**Классификация токсичных комментариев.** Классификации текстовых комментариев на токсичные и нетоксичные, помогает выявлять и модерировать вредный или оскорбительный контент на онлайн-платформах.

**Классификация изображений.** Файлы изображения классифицируются по заранее определенным классам, часто используют с методами обнаружения объектов.

**Распознавание лиц.** Идентификация и аутентификация людей на основе черт лица, поиска приложений в системах безопасности, контроля доступа и наблюдения.

**Анализ тональности.** Методы классификации применяются для анализа и классификации текстовых данных (например, отзывов клиентов, сообщений в социальных сетях) для определения настроений (по различным шкалам) и понимания общественного мнения.

**Классификация спама по электронной почте.** Является одним из наиболее распространенных способов применения классификации, помогает фильтровать нежелательные или вредоносные электронные письма.

**Классификация документов.** Упорядочение документов по категориям в соответствии с их содержанием. Автоматическая классификация документов, таких как новостные статьи, юридические документы или заявки в службу поддержки, на соответствующие категории, способствует эффективному поиску и организации документов.

**Классификация токсичных комментариев.** Классификации текстовых комментариев на токсичные и нетоксичные, помогает выявлять и модерировать вредный или оскорбительный контент на онлайн-платформах.

**Классификация изображений.** Файлы изображения классифицируются по заранее определенным классам, часто используют с методами обнаружения объектов.

**Распознавание лиц.** Идентификация и аутентификация людей на основе черт лица, поиска приложений в системах безопасности, контроля доступа и наблюдения.

**Распознавание голоса.** Классификация используется в системах распознавания речи для классификации произносимых слов или фраз, что позволяет использовать такие приложения, как голосовые помощники, службы транскрипции и идентификацию говорящего.

**Анализ тональности.** Методы классификации применяются для анализа и классификации текстовых данных (например, отзывов клиентов, сообщений в социальных сетях) для определения настроений (по различным шкалам) и понимания общественного мнения.

**Классификация спама по электронной почте.** Является одним из наиболее распространенных способов применения классификации, помогает фильтровать нежелательные или вредоносные электронные письма.

**Классификация документов.** Упорядочение документов по категориям в соответствии с их содержанием. Автоматическая классификация документов, таких как новостные статьи, юридические документы или заявки в службу поддержки, на соответствующие категории, способствует эффективному поиску и организации документов.

**Классификация токсичных комментариев.** Классификации текстовых комментариев на токсичные и нетоксичные, помогает выявлять и модерировать вредный или оскорбительный контент на онлайн-платформах.

**Классификация изображений.** Файлы изображения классифицируются по заранее определенным классам, часто используют с методами обнаружения объектов.

**Распознавание лиц.** Идентификация и аутентификация людей на основе черт лица, поиска приложений в системах безопасности, контроля доступа и наблюдения.

**Распознавание голоса.** Классификация используется в системах распознавания речи для классификации произносимых слов или фраз, что позволяет использовать такие приложения, как голосовые помощники, службы транскрипции и идентификацию говорящего.

**Диагностика заболеваний.** Алгоритмы классификации анализируют данные пациентов, симптомы и результаты медицинских анализов, чтобы классифицировать заболевания.

**Анализ тональности.** Методы классификации применяются для анализа и классификации текстовых данных (например, отзывов клиентов, сообщений в социальных сетях) для определения настроений (по различным шкалам) и понимания общественного мнения.

**Классификация спама по электронной почте.** Является одним из наиболее распространенных способов применения классификации, помогает фильтровать нежелательные или вредоносные электронные письма.

**Классификация документов.** Упорядочение документов по категориям в соответствии с их содержанием. Автоматическая классификация документов, таких как новостные статьи, юридические документы или заявки в службу поддержки, на соответствующие категории, способствует эффективному поиску и организации документов.

**Классификация токсичных комментариев.** Классификации текстовых комментариев на токсичные и нетоксичные, помогает выявлять и модерировать вредный или оскорбительный контент на онлайн-платформах.

**Классификация изображений.** Файлы изображения классифицируются по заранее определенным классам, часто используют с методами обнаружения объектов.

**Распознавание лиц.** Идентификация и аутентификация людей на основе черт лица, поиска приложений в системах безопасности, контроля доступа и наблюдения.

**Распознавание голоса.** Классификация используется в системах распознавания речи для классификации произносимых слов или фраз, что позволяет использовать такие приложения, как голосовые помощники, службы транскрипции и идентификацию говорящего.

**Диагностика заболеваний.** Алгоритмы классификации анализируют данные пациентов, симптомы и результаты медицинских анализов, чтобы классифицировать заболевания.

**Рекомендательные системы.** Классификация контента для предоставления персонализированных рекомендаций, улучшает взаимодействие с пользователем в электронной коммерции, потоковых платформах.



**Анализ тональности.** Методы классификации применяются для анализа и классификации текстовых данных (например, отзывов клиентов, сообщений в социальных сетях) для определения настроений (по различным шкалам) и понимания общественного мнения.

**Классификация спама по электронной почте.** Является одним из наиболее распространенных способов применения классификации, помогает фильтровать нежелательные или вредоносные электронные письма.

**Классификация документов.** Упорядочение документов по категориям в соответствии с их содержанием. Автоматическая классификация документов, таких как новостные статьи, юридические документы или заявки в службу поддержки, на соответствующие категории, способствует эффективному поиску и организации документов.

**Классификация токсичных комментариев.** Классификации текстовых комментариев на токсичные и нетоксичные, помогает выявлять и модерировать вредный или оскорбительный контент на онлайн-платформах.

**Классификация изображений.** Файлы изображения классифицируются по заранее определенным классам, часто используют с методами обнаружения объектов.

**Распознавание лиц.** Идентификация и аутентификация людей на основе черт лица, поиска приложений в системах безопасности, контроля доступа и наблюдения.

**Распознавание голоса.** Классификация используется в системах распознавания речи для классификации произносимых слов или фраз, что позволяет использовать такие приложения, как голосовые помощники, службы транскрипции и идентификацию говорящего.

**Диагностика заболеваний.** Алгоритмы классификации анализируют данные пациентов, симптомы и результаты медицинских анализов, чтобы классифицировать заболевания.

**Рекомендательные системы.** Классификация контента для предоставления персонализированных рекомендаций, улучшает взаимодействие с пользователем в электронной коммерции, потоковых платформах.

**Кибербезопасность.** Методы классификации используются для обнаружения и классификации сетевых вторжений и киберугроз, различая обычный сетевой трафик и вредоносные действия.