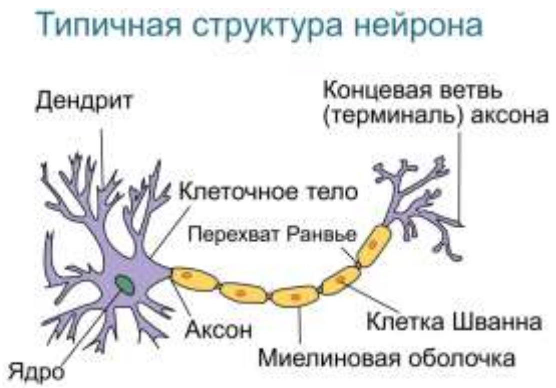
Искусственный нейрон
Персептрон
Градиентный спуск
Логистическая регрессия

Искусственный нейрон – модель естественного нейрона

- Нейрон – клетка, структурно-функциональная единица нервной системы



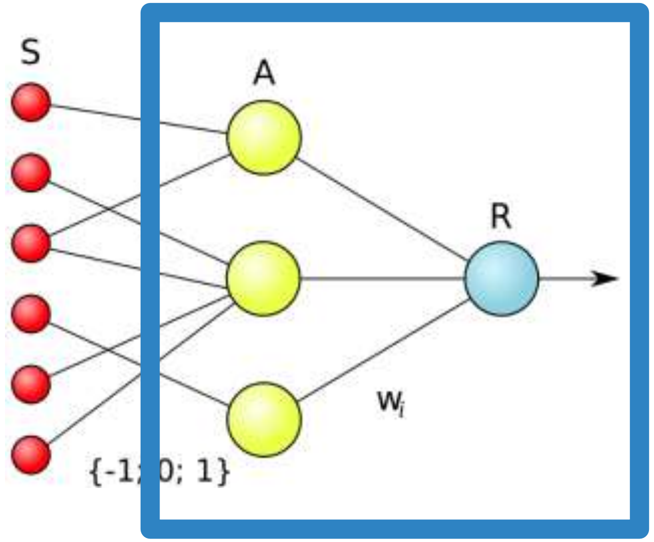
- Модель мозга описана в 1943 году учёными Уорреном Мак-Каллаком и Уолтером Питтсом в статье «Логическое исчисление идей, относящихся к нервной активности». Идея: использовать в качестве вычислительных машин.

$$N_2(t) . \equiv . N_1(t-1).$$

Треугольник – тело нейрона, линии – аксоны, цифра – номер нейрона п.п., N_i – действие нейрона

Персептрон

- Персептрон (от англ. Perception – восприятие) - математическая модель восприятия информации мозгом
- Фрэнк Розенблатт (1957) – впервые ввел термин персептрон и предложил алгоритм обучения



- S – рецепторы (сенсорные). Задают тормозные, возбуждающие связи или их отсутствие.
- A – сумматор с порогом (ассоциативные). Возбуждается если сумма входных сигналов больше порога.
- R – выходной сигнал (реагирующий).

Двоичная классификация и функция решения

- Есть данные, которые можно разбить на два класса 1 (положительный) и -1 (отрицательный)
- Определим ***z*** – т.н. чистый вход (net input) - линейную комбинацию входных значений ***x*** и весов ***w***

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \qquad w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \qquad z = w_1 x_1 + \cdots + w_m x_m$$

- Определим функцию решений ***ϕ(z)***, которая принимает на вход ***z***. Если чистый вход для отдельного образца $x^{(i)}$ не меньше чем порог θ, прогнозируем класс 1, иначе класс -1.

$$\phi(z) = \begin{cases} 1, & \text{если } z \geq \theta \\ -1, & \text{если } z < \theta \end{cases}$$

Работа функции решения

Для простоты перенесем порог θ в левую часть и определим его как нулевой вес

$$w_0 = -\theta, x_0 = 1$$

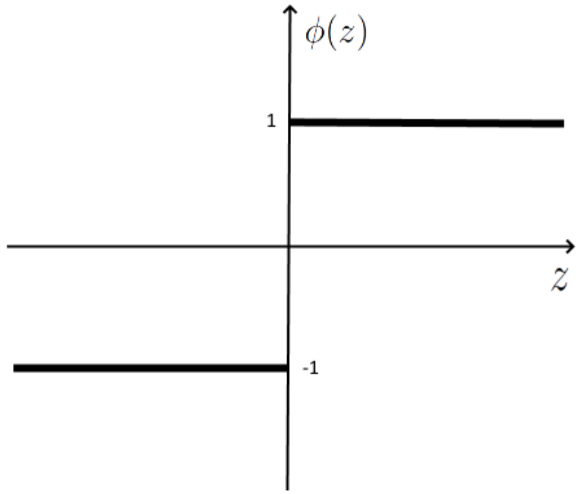
Тогда ***z*** и ***ϕ(z)*** запишутся как:

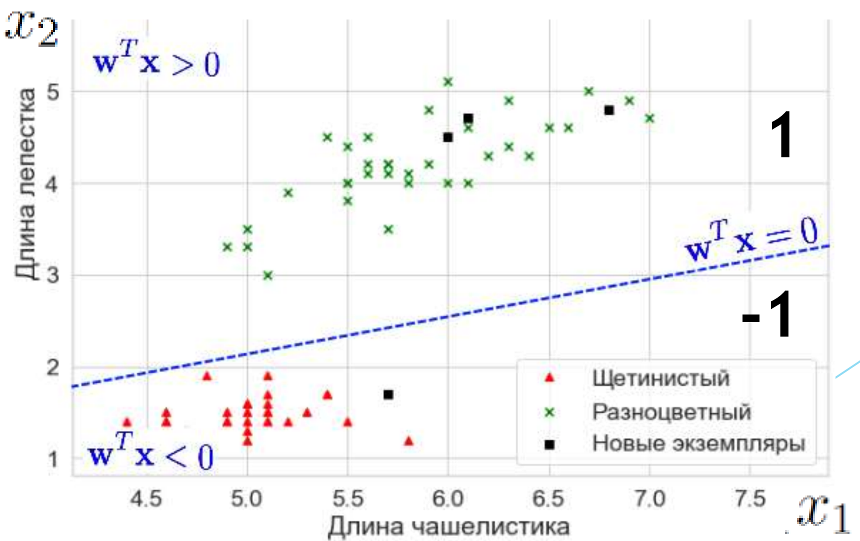
$$z = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \mathbf{w}^T\mathbf{x}$$

$$\phi(z) = \begin{cases} 1, & \text{если } z \geq 0 \\ -1, & \text{если } z < 0 \end{cases}$$

или

$$\phi(z) = \mathbf{sign}(z)$$





Процесс обучения (методом Розенблатта)

- Инициализировать веса нулями или небольшими случайными числами
- Для каждого обучающего образца $x^{(i)}$
  - Вычислить выходное значение (метку класса) $\hat{y} = \mathbf{sign}(\mathbf{w}^T\mathbf{x})$
  - Обновить веса

$$w_j := w_j + \Delta w_j$$

Скорость обучения (константа между 0.0 и 1.0)

$$\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)}$$

Реальная метка

Спрогнози-рованная метка

Процесс обучения (методом Розенблатта)

$$z = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \mathbf{w}^T\mathbf{x}$$

$$\hat{y} = \mathbf{sign}(\mathbf{w}^T\mathbf{x})$$

$$w_j := w_j + \Delta w_j$$

$$\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)}$$

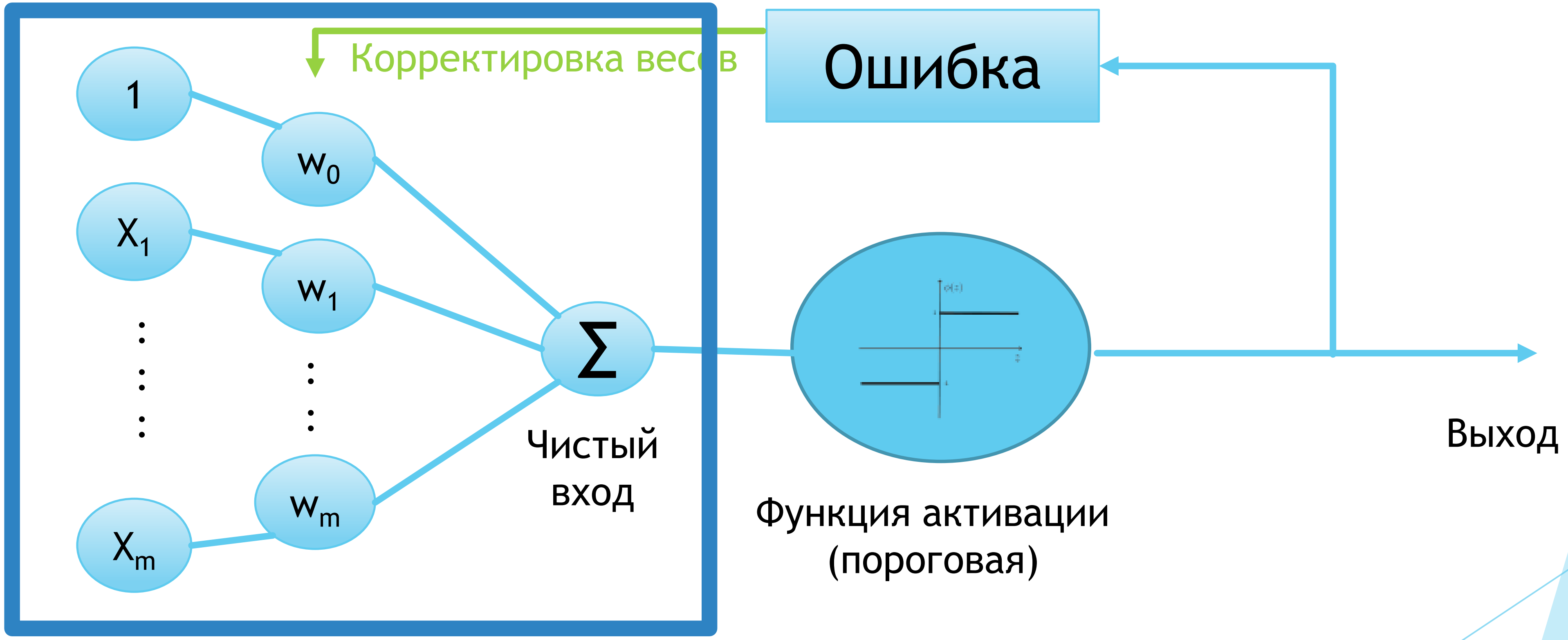
Возможные варианты:

$$y^{(i)} = \hat{y}^{(i)} : \Delta w_j = 0$$

$$y^{(i)} = 1; \hat{y}^{(i)} = -1 : \Delta w_j = 2\eta x_j^{(i)}$$

$$y^{(i)} = -1; \hat{y}^{(i)} = 1 : \Delta w_j = -2\eta x_j^{(i)}$$

Процесс обучения (методом Розенблатта)



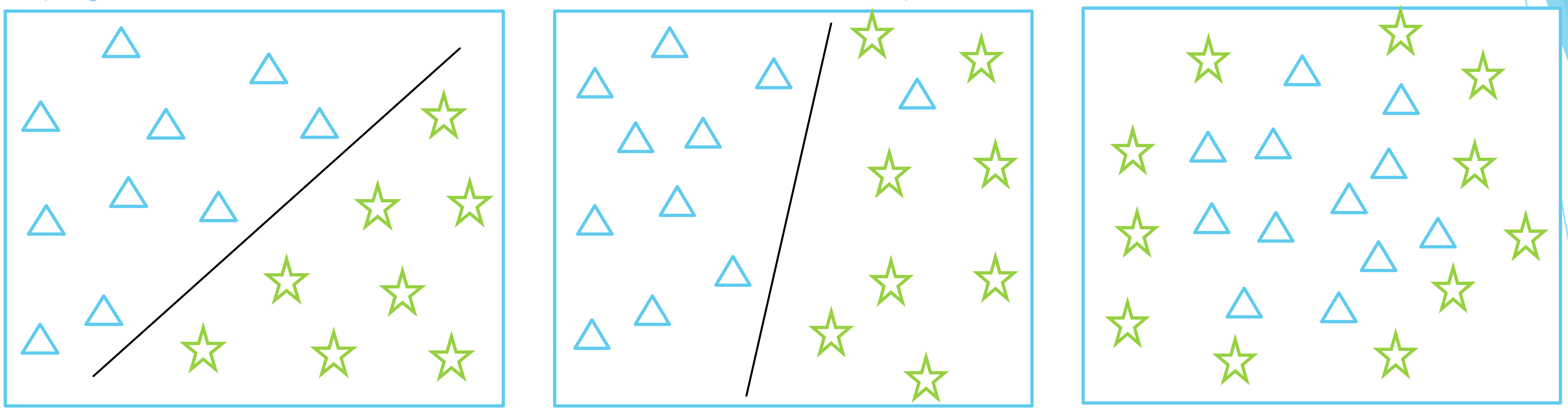
Вопросы (ответы дальше)

- Прогнав весь набор данных не обязательно получим хорошо натренированный персептрон…?
- Можем ли мы его натренировать так, чтобы он мог применяться в любой задаче?
- Что делать если классов больше чем два?

Эпохи обучения (прогнав весь набор..)

- Пройдя весь набор данных, мы можем оценить сколько экземпляров мы классифицировали неверно
- Ещё раз запустить весь алгоритм обучения с самого начала, прогнав ещё раз весь набор данных (новая эпоха обучения).
- Цель – ещё раз подстроить уже настроенные веса, чтобы классификация была ещё точнее.
- Повторять обучение на новых эпохах пока не выполнится одно из условий:
    - Достигнем точности (образцы будут классифицироваться с заранее заданной точностью)
    - Пройдёт заданное количество эпох

Линейно разделимые множества (применять в любой задаче..)



Линейно разделимое множество

Линейно неразделимые множества

- Доказано, что персептрон сходится, если два класса линейно разделимы
- Если же нет, то веса никогда не перестанут обновляться. Остановить обучение можно только достигнув максимальное количество эпох

Если классов больше чем два

- В рассмотренных примерах только две метки классов -1 и 1.
- Если классов больше, можно воспользоваться методикой OvA (One-versus-All) (то же самое OvR One-versus-Rest) или OvO (One-versus-One)

OvA

- Допустим у нас есть *n* классов
- Создадим для классов такие классификаторы, которые будут решать задачу бинарной классификации – отделение *i*-го класса от остальных. Всего *n* классификаторов
- Чтобы определить к какому классу относится новый экземпляр, необходимо применить все классификаторы и назначить метку того, который демонстрирует большую меру уверенности

OvO

- Допустим у нас есть *n* классов
- Создадим для каждой пары классов классификаторы, которые будут решать задачу бинарной классификации – отделение *одного* класса от другого. Всего $n(n-1)/2$ классификаторов.
- Чтобы определить к какому классу относится новый экземпляр, необходимо применить все классификаторы и назначить ту метку, которую выдало большинство из использованных классификаторов

https://scikit-learn.org/stable/modules/multiclass.html

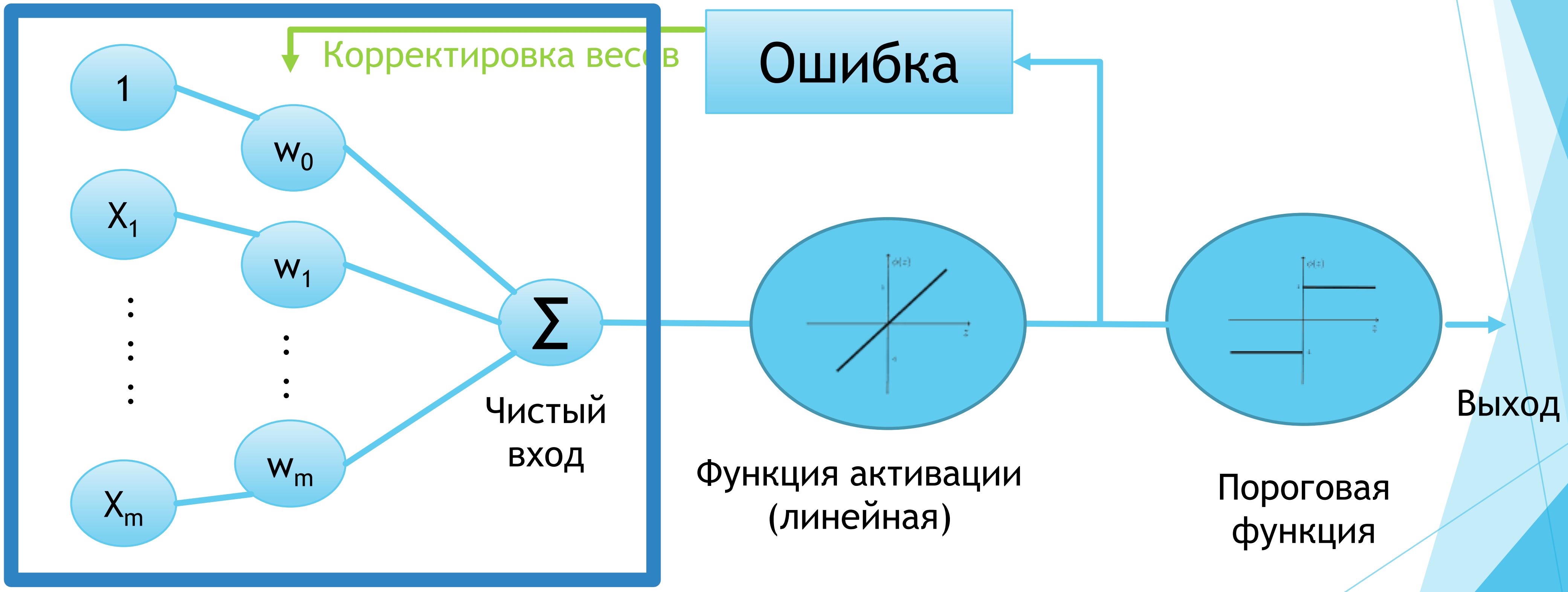
Адаптивный нейрон. Правило Видроу-Хоффа

- Веса обновляются на основе линейной функции активации (а не единичной ступенчатой)

$$\phi(\mathbf{w}^T\mathbf{x}) = \mathbf{w}^T\mathbf{x}$$

- Для финального прогноза используем пороговую функцию
- В отличие от предыдущего метода обучения, где сравнивались реальные метки с прогнозами, здесь сравниваются настоящие метки с непрерывным выходом – значениями линейной функции активации
- Сравниваем настоящие метки с непрерывным выходом, вычисляем ошибку, обновляем веса

Процесс обучения (правило Видроу-Хоффа)



Процесс обучения (правило Видроу-Хоффа) Целевая функция

- Введем следующую функцию издержек – среднеквадратическая ошибка предсказания (Sum of Squared Error)

$$S(\mathbf{w}) = \frac{1}{2}\sum_i \left(y^{(i)} - \phi(z^{(i)})\right)^2$$

- Необходимо минимизировать данную функцию изменяя веса

$$w_j := w_j + \Delta w_j$$

- Функция S является дифференцируемой и выпуклой, можем применить алгоритм градиентного спуска (Gradient Descent) определив изменение весов как антиградиент функции, умноженный на темп обучения η

$$\Delta \mathbf{w} = -\eta \nabla S(\mathbf{w})$$

Процесс обучения (правило Видроу-Хоффа) Изменение веса

- Вычислим частную производную функции S относительно каждого веса:

$$\frac{\partial S}{\partial w_j} = -\sum_i \left(y^{(i)} - \phi(z^{(i)})\right) x_j^{(i)}$$

- Таким образом изменение j-го веса можно записать в следующем виде

$$\Delta w_j = -\eta \frac{\partial S}{\partial w_j} = \eta \sum_i \left(y^{(i)} - \phi(z^{(i)})\right) x_j^{(i)}$$

Процесс обучения (правило Видроу-Хоффа) Изменение весов. Пакетный градиентный спуск

['greɪdɪənt dɪ'sent]

- Учитывая, что все веса обновляются одновременно, правило обучения принимает вид:

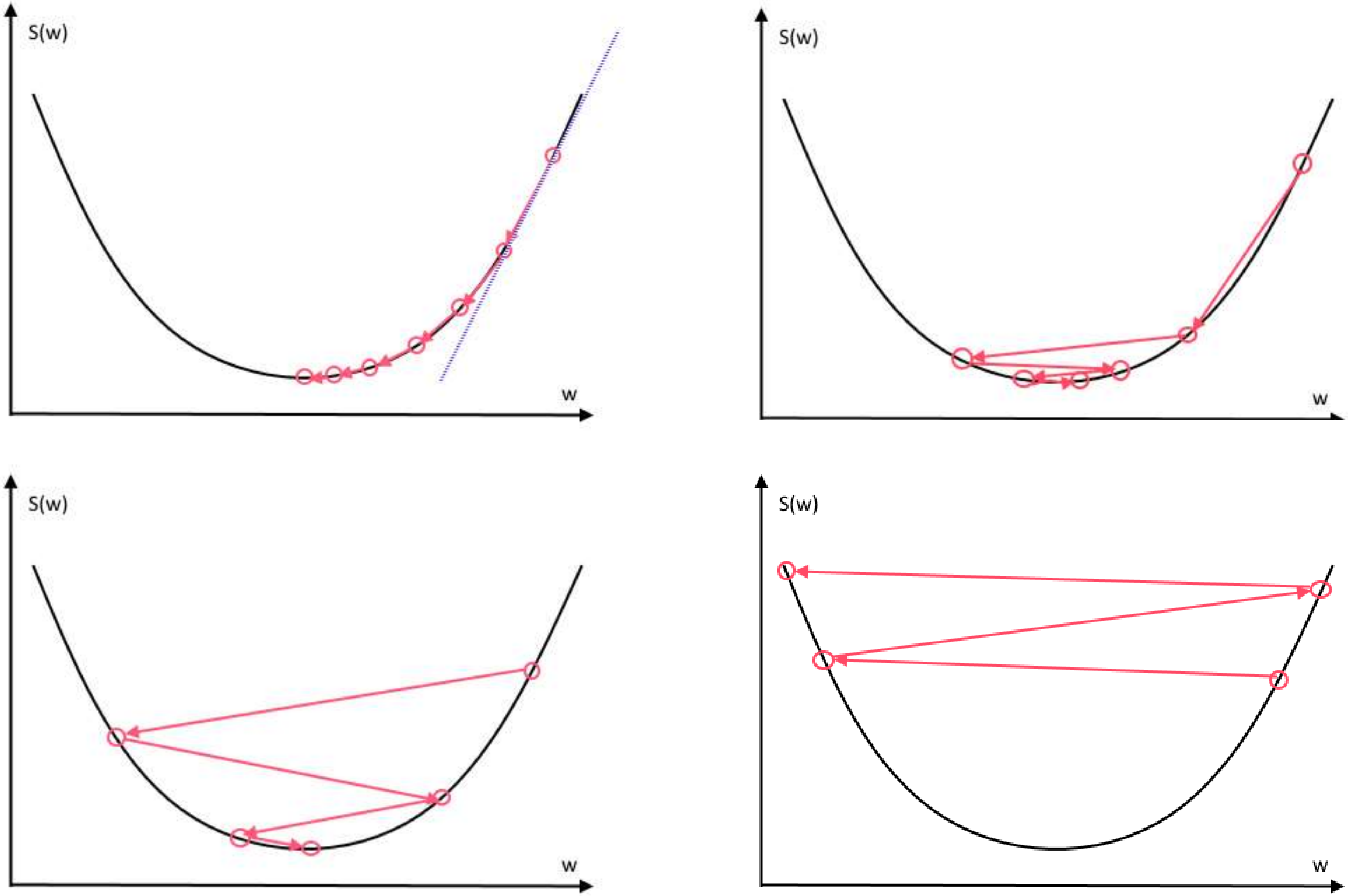
$$w := w + \Delta w$$

- Заметим, что обновление весов осуществляется на основе всех образцов в тренировочном наборе. Такой подход называется «пакетным» (batch) градиентным спуском

Сумма по *i* – по всем образцам

$$\Delta w_j = \eta \sum_i \left(y^{(i)} - \phi(z^{(i)}) \right) x_j^{(i)}$$

Градиентный спуск для разных темпов обучения



Гиперпараметры

- Гиперпараметр – параметр (внешний по отношению к модели), который устанавливается перед обучением, во время обучения постоянный (априорный параметр)
- Не извлекается в процессе обучения
- В данном примере два гиперпараметра – ***темп обучения*** η и ***количество эпох*** обучения
- Параметры модели – параметры (внутренние по отношению к модели) изменяются в процессе обучения модели (например, веса w)

https://datascience.stackexchange.com/questions/14187/what-is-the-difference-between-model-hyperparameters-and-model-parameters

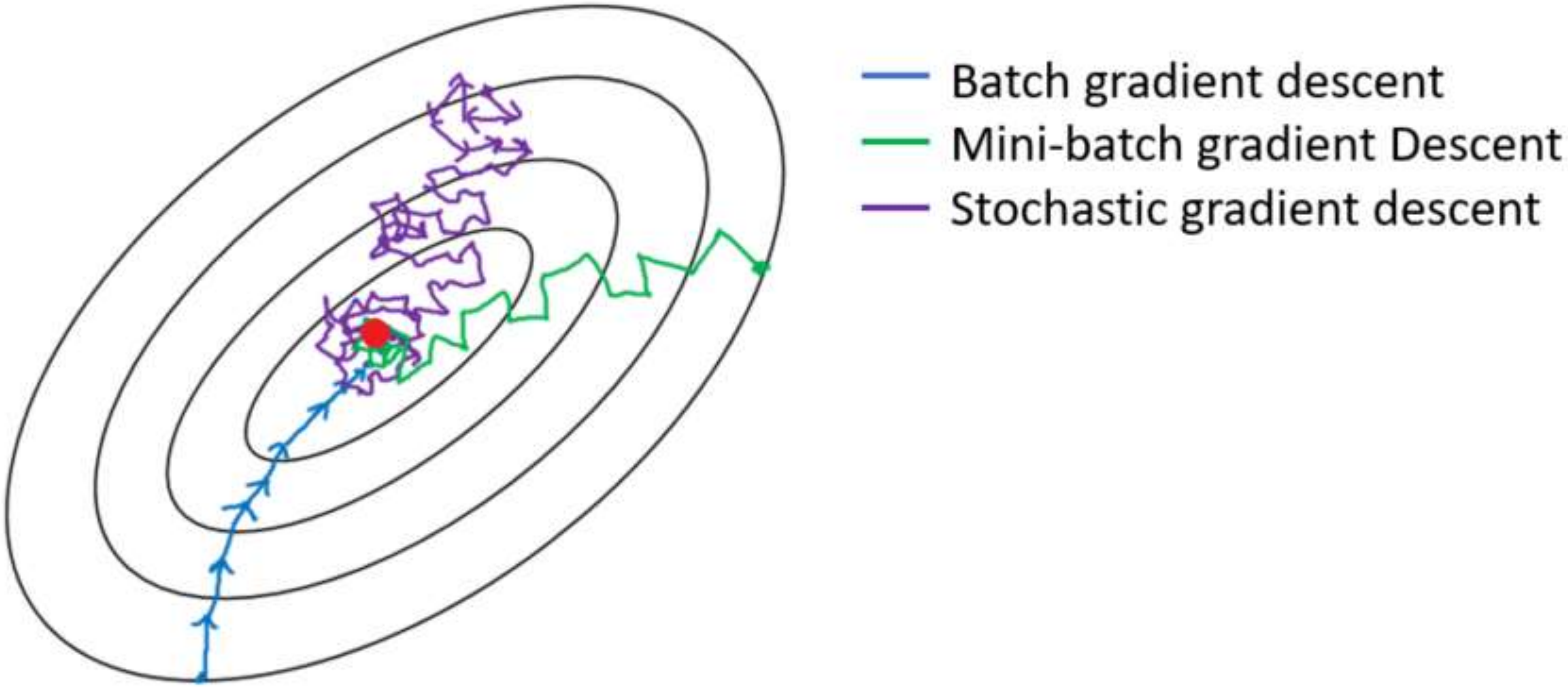
Варианты градиентного спуска

- Пакетный – рассчитывается из всего тренировочного набора
  - (-) Для изменения веса требуется проход по всем образцам. Каждый шаг – проход по всему набору -> крайне медленный на больших наборах
  - (+) Обеспечивает стабильный градиент ошибок и стабильную сходимость
- Стохастический – рассчитывается на одном-единственном тренировочном примере
  - (-) Зашумленный градиент, шаг может привести к увеличению ошибки вместо её уменьшения, а минимум может быть не достигнут (останемся в его окрестности)
  - (+) Более быстрая сходимость за счет частого обновления весов. Менее затратен в вычислительном плане
- Мини-пакетный – применение пакетного градиентного спуска к подмножеству данных (например, 50 образцов за раз)
  - (-) Минимум может быть не достигнут (хотя подойдёт ближе чем стохастический градиентный спуск)
  - (+) Более стабильные обновления и остановка ближе к минимуму

https://suniljangirblog.wordpress.com/2018/12/13/variants-of-gradient-descent/

Варианты градиентного спуска

- После обучения нет никаких отличий: очень похожие модели и прогнозы



https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

https://suniljangirblog.wordpress.com/2018/12/13/variants-of-gradient-descent/

Персептрон и градиентный спуск в Python

- sklearn.linear_model
  - Perceptron – реализация точно такая же как в SGDClassifier
  - SGDClassifier – реализация линейной модели с обучением, использующим стохастический градиентный спуск (есть возможность переключения на мини-пакетный)

В стохастическом градиентном спуске фиксированная скорость обучения η часто заменяется адаптивной – изменяемой со временем. Например, так (c_1,c_2 - константы):

$$\frac{c_1}{[\text{номер эпохи}] + c_2}$$

lection_4_Perceptron

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html

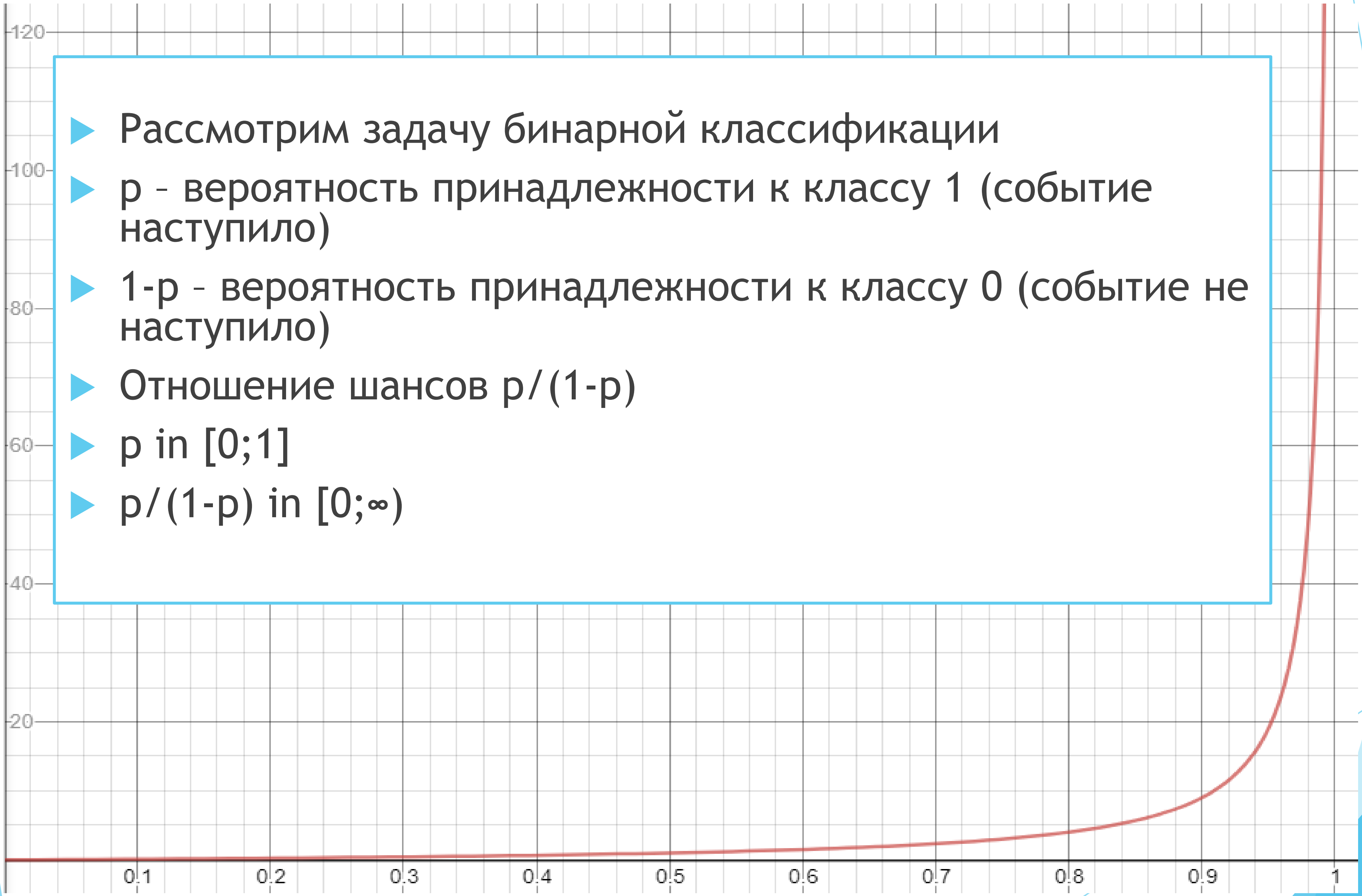
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

Логистическая регрессия

- Модель для задач классификации
- Простой и более мощный алгоритм построения линейного классификатора
- Применяется для прогнозирования вероятности некоторого события (апостериорные вероятности принадлежности к классам)

Логистическая регрессия

- Рассмотрим задачу бинарной классификации
- p – вероятность принадлежности к классу 1 (событие наступило)
- 1-p – вероятность принадлежности к классу 0 (событие не наступило)
- Отношение шансов p/(1-p)
- p in [0;1]
- p/(1-p) in [0;∞)

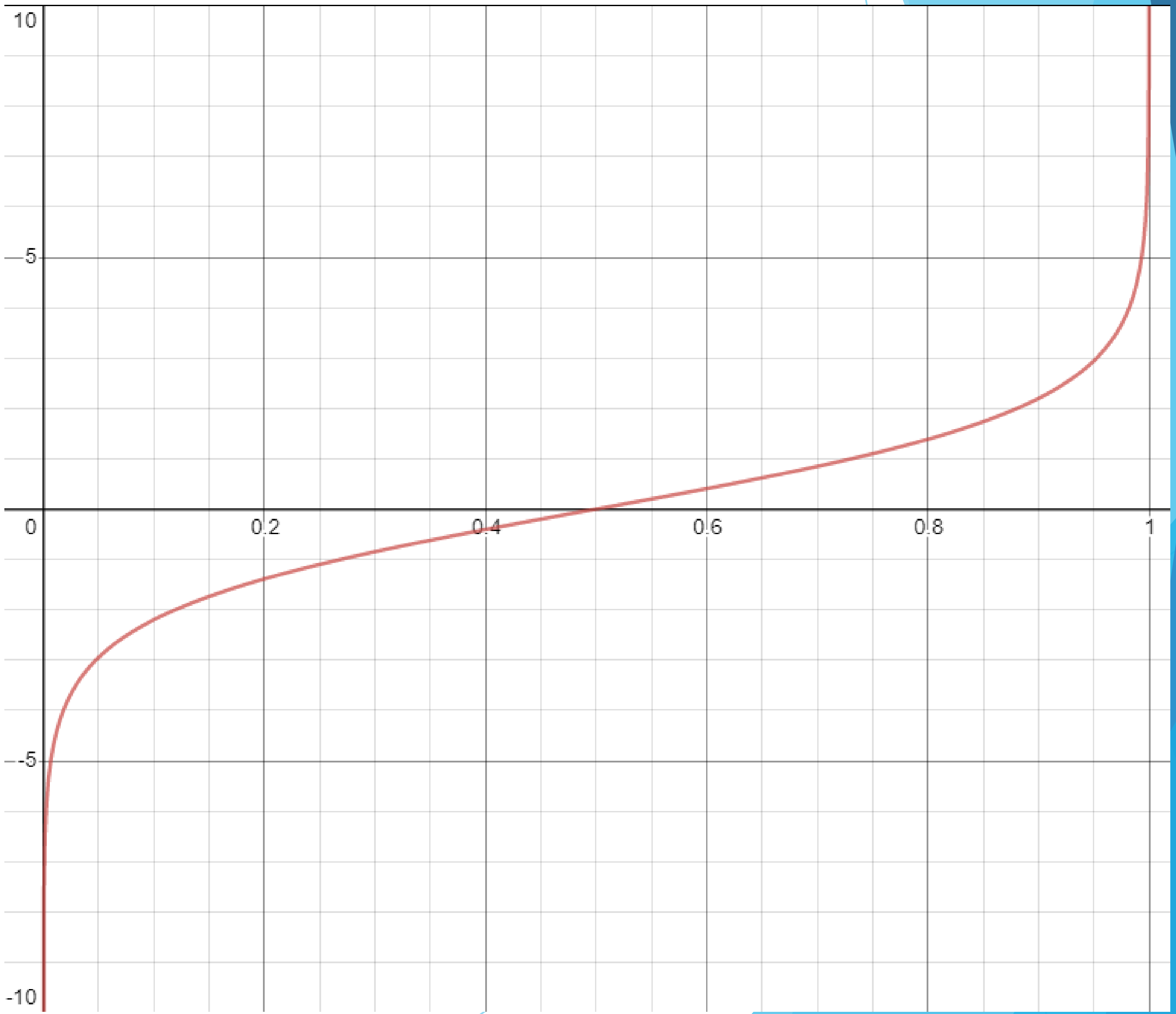


Logit

- Определим функцию logit как ln отношения шансов
- p in [0;1]

$$\mathbf{logit}(p) = \ln \frac{p}{1-p}$$

- logit(p) in (- ∞;∞)



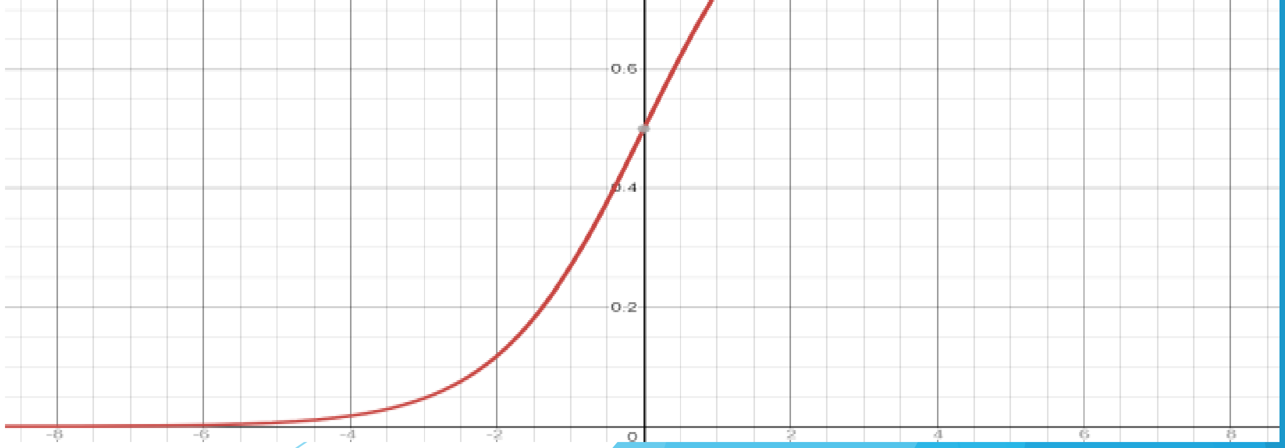
Выражение линейной связи между логарифмом отношения шансов и значениями признаков

$$\mathbf{logit}\,(p(y=1|x)) = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \mathbf{w}^T\mathbf{x}$$

- p(y=1|x) – условная вероятность, что y принадлежит классу 1 при наличии признаков x
- Поскольку нас интересует вероятность, это обратная функция для функции logit

$$p(y=1|x) = \mathbf{logit}^{-1}\,(\mathbf{w}^T\mathbf{x})$$

$$\mathbf{logit}^{-1}\,(z) = \phi(z) = \frac{1}{1+e^{-z}}$$

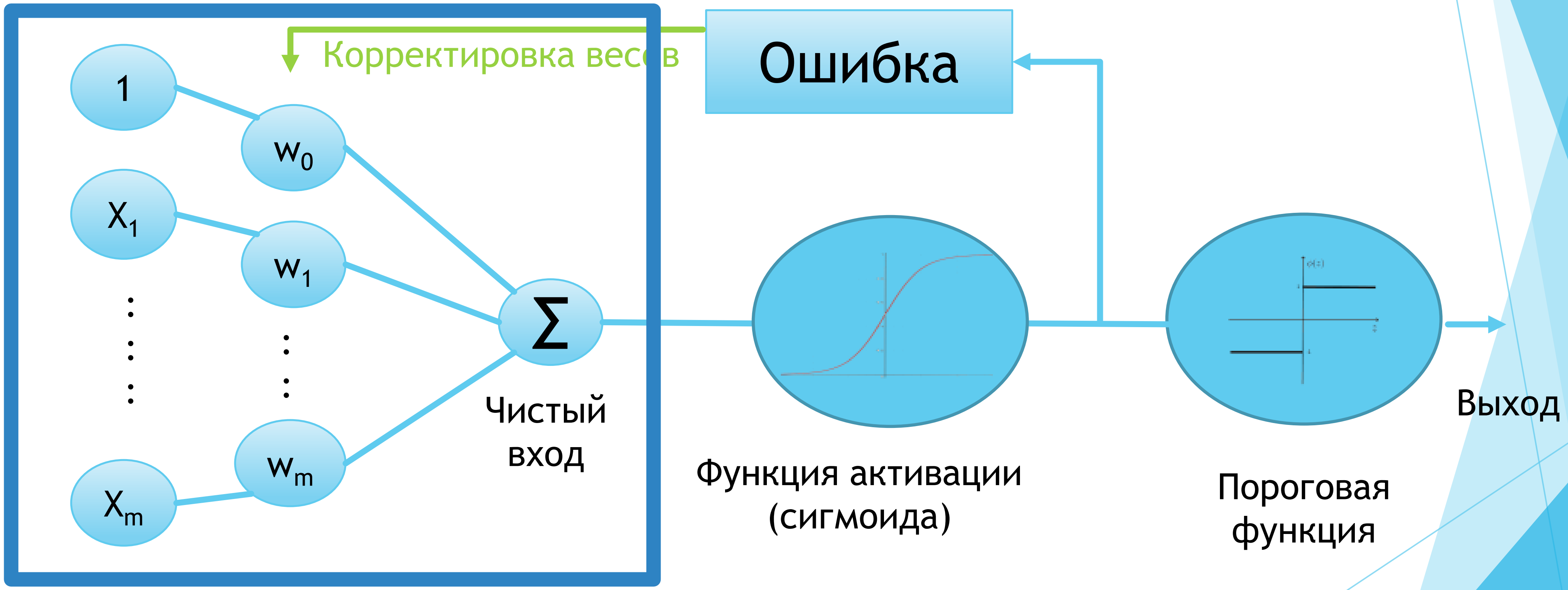


- z – чистый вход
- Обратная функция называется логистической

функцией или сигмоидой

Зная значение функции z можно найти вероятность

Процесс обучения с сигмоидальной функцией активации



Принадлежность к классам

$$\hat{y} = \begin{cases} 1, & \text{если } \phi(z) \geq 0.5 \\ 0, & \text{если } \phi(z) < 0.5 \end{cases}$$

Вероятность принадлежности экземпляра к классу
(если убрать пороговую функцию)

$$\hat{y} = 1 \text{ с вероятностью } \phi(z)$$
$$\hat{y} = 0 \text{ с вероятностью } 1 - \phi(z)$$

- Более гибкий подход – даёт оценку вероятности
- Может использоваться в:
  - Прогноз погоды – дождь с вероятностью …
  - Медицина – наличие болезни с вероятностью … при наличии симптомов
  - Банки - вероятность что клиент вернет кредит … при условиях

  (выдавать-не выдавать)

Функция распределения, правдоподобие

$$P(y=1|x)=\phi(z)$$

$$P(y=0|x)=1-\phi(z)$$

- Функцию распределения y при заданном x можно записать в таком виде

$$P(y|x)=\phi(z)^y(1-\phi(z))^{1-y}, y\in\{0,1\}$$

- Фактически это распределение Бернулли, где p = $\phi(z)$
- Рассмотрим функцию правдоподобия L(w) – вероятность наблюдать вектор **y** у выборки X. Объекты рассматриваются независимо друг от друга

$$L(w)=P(y|x;w)=\prod_i P(y^{(i)}|x^{(i)};w)$$

- Необходимо максимизировать правдоподобие (чтобы вероятность на каждом объекте была максимальна)

Целевая функция

$$L(w) = P(y|x;w) = \prod_i P(y^{(i)}|x^{(i)};w)$$

Перепишем, подставив функцию распределения

$$\prod_i P(y^{(i)}|x^{(i)};w) = \prod_i \left(\phi(z^{(i)})\right)^{y^{(i)}} \left(1-\phi(z^{(i)})\right)^{1-y^{(i)}}$$

На практике проще максимизировать логарифм этой функции, т.н. логарифмическую функцию правдоподобия

$$l(w) = \ln(L(w)) = \sum_i \left[y^{(i)} \ln\left(\phi(z^{(i)})\right) + \left(1-y^{(i)}\right) \ln\left(1-\phi(z^{(i)})\right)\right]$$

Введем целевую функцию как –l(w) и поставим задачу её минимизации

$$S(w) = \sum_i \left[-y^{(i)} \ln\left(\phi(z^{(i)})\right) - \left(1-y^{(i)}\right) \ln\left(1-\phi(z^{(i)})\right)\right]$$

Log-Loss, кросс-энтропия (насколько близко прогнозируемое распределение к истинному)

Процесс обучения. Изменение веса

- Вычислим частную производную функции S относительно каждого веса:

$$\frac{\partial S}{\partial w_j} = -\sum_i \left(y^{(i)} - \phi(z^{(i)})\right) x_j^{(i)}$$

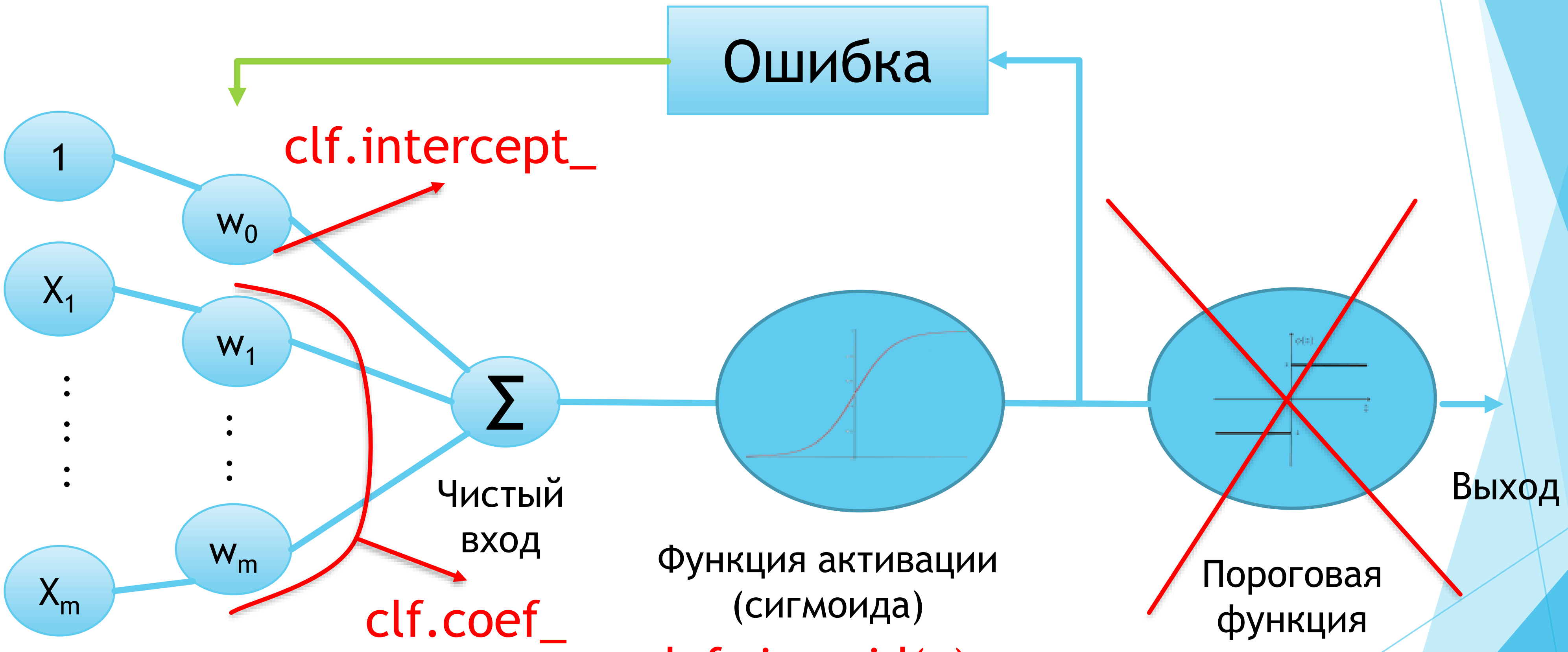
- Таким образом изменение j-го веса можно записать в следующем виде

$$\Delta w_j = -\eta \frac{\partial S}{\partial w_j} = \eta \sum_i \left(y^{(i)} - \phi(z^{(i)})\right) x_j^{(i)}$$

Аналогично правилу Видроу-Хоффа

lection_4_LogisticRegression

Процесс обучения с сигмоидальной функцией активации. Результат из Python. Ручной расчет




```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

lection_4_LogisticRegression-2class-coef_

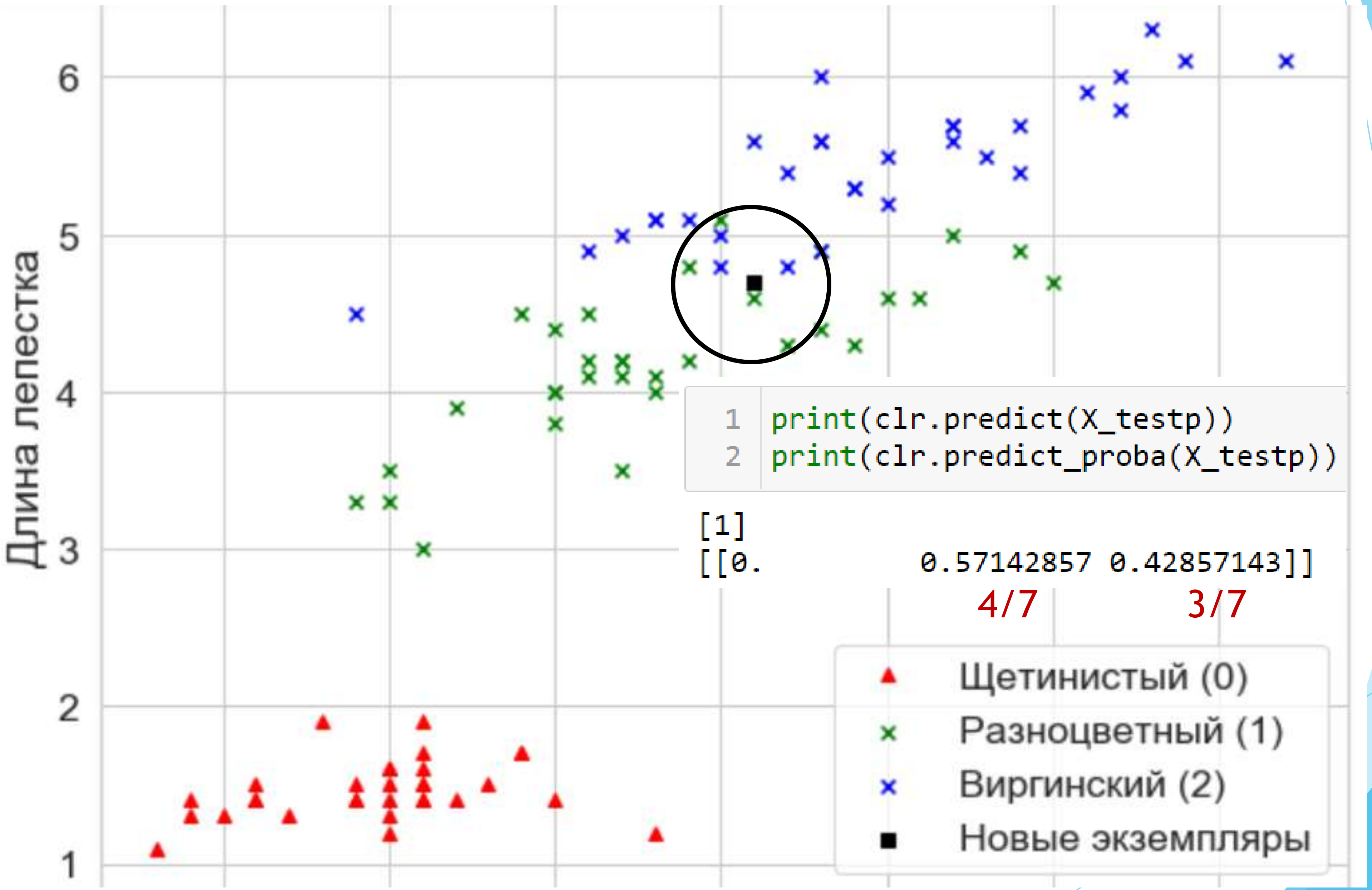
Вопросы

- Какой вариант градиентного спуска отработает быстрее если набор данных очень большой?
- Какой вариант градиентного спуска быстрее достигнет окрестности минимума?
- Какой вариант градиентного спуска даст более точное решение?

Вопросы

- OvA. Чтобы определить к какому классу относится новый экземпляр, необходимо применить все классификаторы и назначить метку того, который демонстрирует большую меру уверенности. Как вы думаете, что такое демонстрирует большую меру уверенности?
- Хороша ли идея немедленно остановить стохастический градиентный спуск, когда ошибка проверки возрастает?
- Мы хотим предсказывать метку класса только в случае если вероятность принадлежности этому классу больше чем 0.7. Как это можно сделать?

Predict_proba для kNN(7)

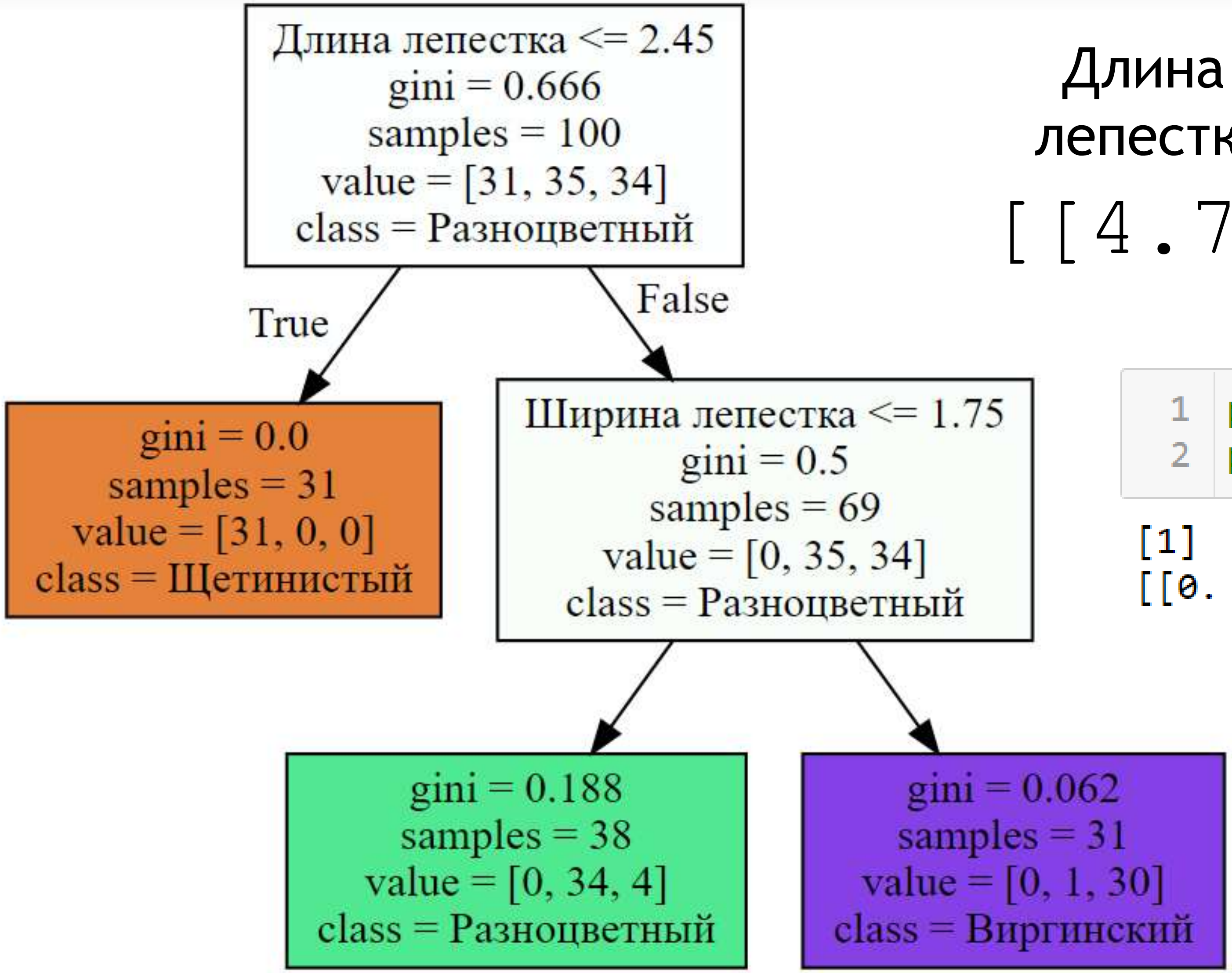



```
print(clr.predict(X_testp))
print(clr.predict_proba(X_testp))
```

```
[1]
[[0.         0.57142857 0.42857143]]
```

4/7 3/7

Predict_proba для DecisionTreeClassifier(max_depth=2)



Длина лепестка	Ширина лепестка
[[4.7,	1.2]]

```
print(clr.predict(X_testp))
print(clr.predict_proba(X_testp))
```

```
[1]
[[0.         0.89473684 0.10526316]]
```

34/38 4/38