Классификация

Задача классификации – это задача обучения с учителем, исходные данные размечены, то есть известны классы объектов. Основная цель классификации

* это определение класса, к которому относится некоторый объект.

Существует множество алгоритмов машинного обучения, которые решают задачу классификации, рассмотрим некоторые из них:

* + логистическая регрессия;
  + Support Vector Machine (SVM);
  + дерево решений, случайный лес, градиентный бустинг;
  + k Nearest Neighbor (KNN);

# Метрики оценки качества модели для задачи классификации

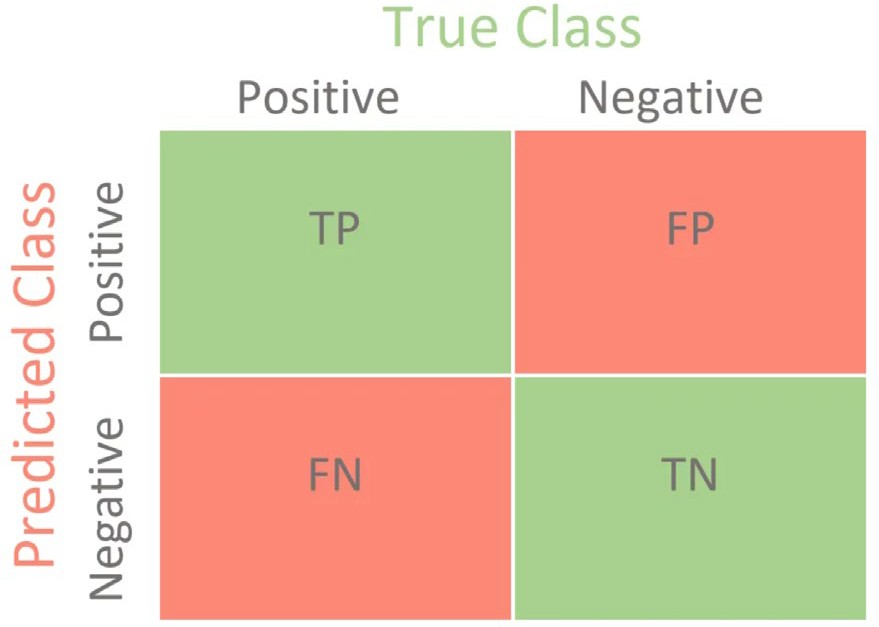
**Accuracy** (sklearn.metrics.accuracy\_score). Доля правильных ответов.

Сумма верно предсказанных

𝐴𝐴𝐴𝐴𝐴𝐴𝐴𝐴𝐴𝐴𝐴𝐴𝐴𝐴𝐴𝐴 =

Количество всех объектов

Самая понятная оценка модели, но при этом показывает некорректный результат, если классы не сбалансированы. Например, у нас может быть всего 100 объектов, из них 10 принадлежат 1 классу и 90 объектов – 0 классу. Если взять константную модель, которая всегда предсказывает 0, то значение accuracy будет 0.9, что кажется неплохим результатом. Но это не так, модель будет работать очень плохо для объектов 1 класса. Как оценить баланс классов? Например, можно построить гистограмму.

Для расчета следующих метрик необходимо рассмотреть confusion matrix (матрицу ошибок).

True Positive – модель предсказала 1 там, где класс был равен 1. True Negative – модель предсказала 0 там, где класс был равен 0. False Positive – модель предсказала 1 там, где класс был равен 0. False Negative – модель предсказала 0 там, где класс был равен 1.

Главная диагональ матрицы, т.е. True Positive и True Negative наиболее важная характеристика матрицы. Чем лучше модель, тем большие значения стоят на главной диагонали.

**Precision** (sklearn.metrics.precision\_score). Точность.

𝑇𝑇𝑃𝑃

𝑃𝑃𝐴𝐴𝑃𝑃𝐴𝐴𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃 = 𝑇𝑇𝑃𝑃 + 𝐹𝐹𝑃𝑃

Precision показывает, какая доля предсказанных 1 является действительно классом 1.

|  |  |
| --- | --- |
| **Модель** | **Таргет** |
| 1 | 1 |
| 0 | 1 |
| 1 | 1 |
| 1 | 0 |
| 0 | 0 |

Здесь 𝑇𝑇𝑃𝑃 = 2, то есть верные предсказания класса 1.

𝐹𝐹𝑃𝑃 = 1, то есть модель предсказала 1, но при этом ошиблась.

Получаем precision = 2.

3

Precision не учитывает какую долю объектов класса 1 модель не предсказала. Но это делает следующая метрика – recall.

**Recall** (sklearn.metrics.recall\_score). Полнота.

𝑇𝑇𝑃𝑃

𝑅𝑅𝑃𝑃𝐴𝐴𝐴𝐴𝑅𝑅𝑅𝑅 = 𝑇𝑇𝑃𝑃 + 𝐹𝐹𝐹𝐹

Recall показывает долю верно предсказанных объектов класса 1.

|  |  |
| --- | --- |
| **Модель** | **Таргет** |
| 1 | 1 |
| 0 | 1 |
| 1 | 1 |
| 1 | 0 |
| 0 | 0 |

Здесь 𝑇𝑇𝑃𝑃 = 2, то есть верные предсказания класса 1.

𝐹𝐹𝐹𝐹 = 1, то есть модель предсказала 0 там, где должна быть 1.

Получаем recall = 2.

3

**F-мера**. Самая полезная метрика, которая совмещает в себе Precision и Recall.

𝐹𝐹𝛽𝛽

= (1 + 𝛽𝛽2) 𝑃𝑃𝐴𝐴𝑃𝑃𝐴𝐴𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃 ⋅ 𝑅𝑅𝑃𝑃𝐴𝐴𝐴𝐴𝑅𝑅𝑅𝑅

𝛽𝛽2 ⋅ 𝑃𝑃𝐴𝐴𝑃𝑃𝐴𝐴𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃 + 𝑅𝑅𝑃𝑃𝐴𝐴𝐴𝐴𝑅𝑅𝑅𝑅

𝛽𝛽 – это параметр, который определяет приоритет точности или полноты. Это зависит от задачи. Если 𝛽𝛽 > 1, то Recall в приоритете, если 𝛽𝛽 < 1, то Precision. Например, если нам необходимо определить, существует ли у человека болезнь сердца, основываясь на анализах. В данной задаче необходимо максимально минимизировать False Negative, то есть модель предсказывает отсутствие болезни там, где она есть. Это крайне серьезная ошибка модели, которая в реальной жизни может привести к плачевным последствиям. Поэтому нам необходимо максимизировать метрику Recall.

Чаще всего используется 𝐹𝐹1-мера (sklearn.metrics.f1\_score), для нее 𝛽𝛽 = 1.

То есть предпочтение не отдается ни точности, ни полноте.

2 ⋅ 𝑃𝑃𝐴𝐴𝑃𝑃𝐴𝐴𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃 ⋅ 𝑅𝑅𝑃𝑃𝐴𝐴𝐴𝐴𝑅𝑅𝑅𝑅

𝐹𝐹1 =

𝑃𝑃𝐴𝐴𝑃𝑃𝐴𝐴𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃𝑃 + 𝑅𝑅𝑃𝑃𝐴𝐴𝐴𝐴𝑅𝑅𝑅𝑅

Рассмотрим некоторые алгоритмы классификации подробнее.

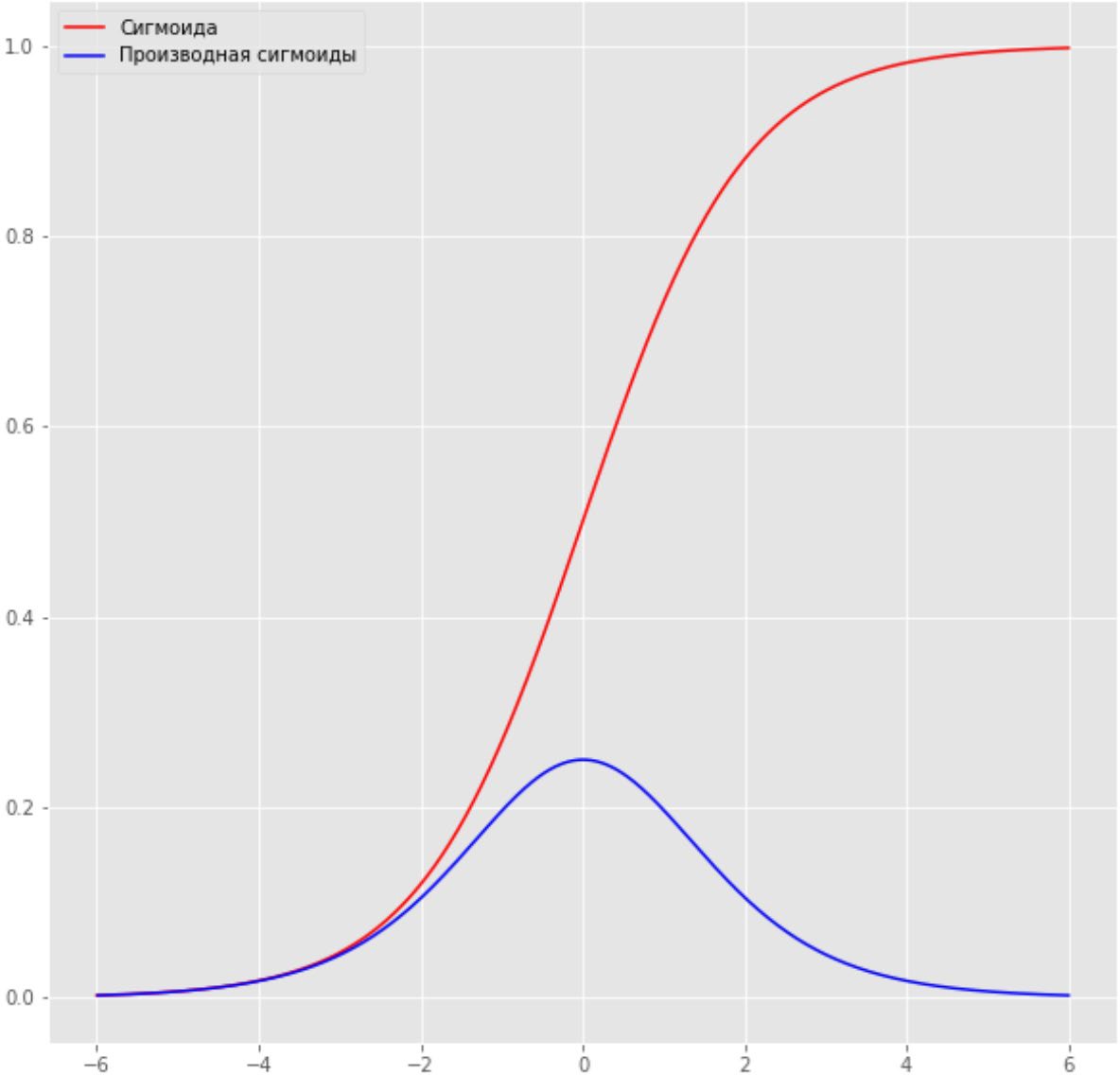
# Логистическая регрессия

От линейной регрессии отличается тем, что в качестве отклика (результата модели) мы получаем бинарное значение, то есть 0 или 1 (если класса всего два).

На самом деле мы предсказываем не метки классов для каждого объекта, а вероятность, с которой объект принадлежит к классу. Если результат больше 0.5, то это 1 класс, если меньше, то 0 класс.

Одно из ограничений применения метода логистической регрессии является то, что она показывает хорошие результаты, когда данные можно разделить гиперплоскостью. То есть, когда исходный набор данных линейно разделим. На основе распределения данных следует подбирать модель, которая наилучшим образом опишет это распределение.

Функция оценки вероятности – сигмоида.



1

𝜎𝜎(𝐴𝐴) =

1 + 𝑃𝑃−𝑦𝑦

Если через эту функцию мы пропустим линейную комбинацию наших признаков и весов, то получим оценку вероятности принадлежности объекта к классу.

1

𝜎𝜎(𝑥𝑥, 𝑤𝑤) = 1 + 𝑃𝑃−(𝑤𝑤0+𝑤𝑤1𝑥𝑥1+⋯+𝑤𝑤𝑚𝑚𝑥𝑥𝑚𝑚)

Веса 𝜔𝜔 отвечают за положение сигмоиды. Задача состоит в том, чтобы подобрать такую сигмоиду, которая будет наилучшим образом проходить через точки в пространстве. А какое положение сигмоиды является лучшим –

решает функция потерь или функция стоимости. Для бинарной классификации используется так называемая *бинарная кросс-энтропия*

𝑛𝑛

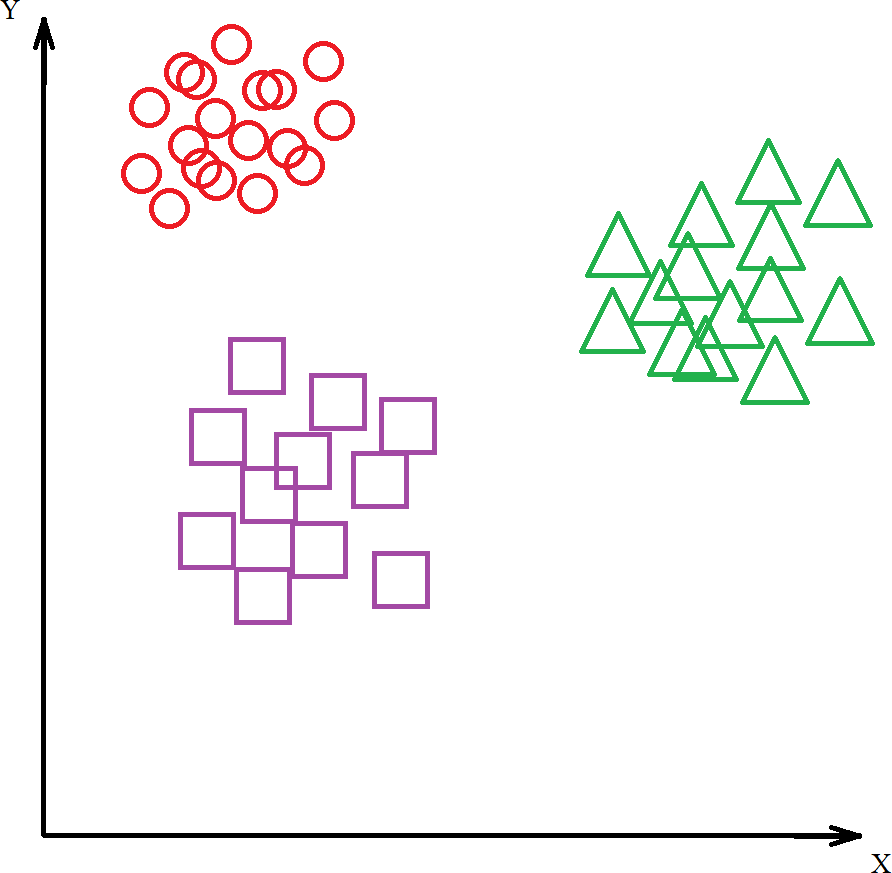
𝐵𝐵𝐵𝐵𝐵𝐵(𝑥𝑥, 𝜔𝜔, 𝐴𝐴) = �(−𝐴𝐴𝑖𝑖 ⋅ ln�𝜎𝜎(𝑥𝑥, 𝑤𝑤)� − (1 − 𝐴𝐴𝑖𝑖) ⋅ ln(1 − 𝜎𝜎(𝑥𝑥, 𝑤𝑤)))

1

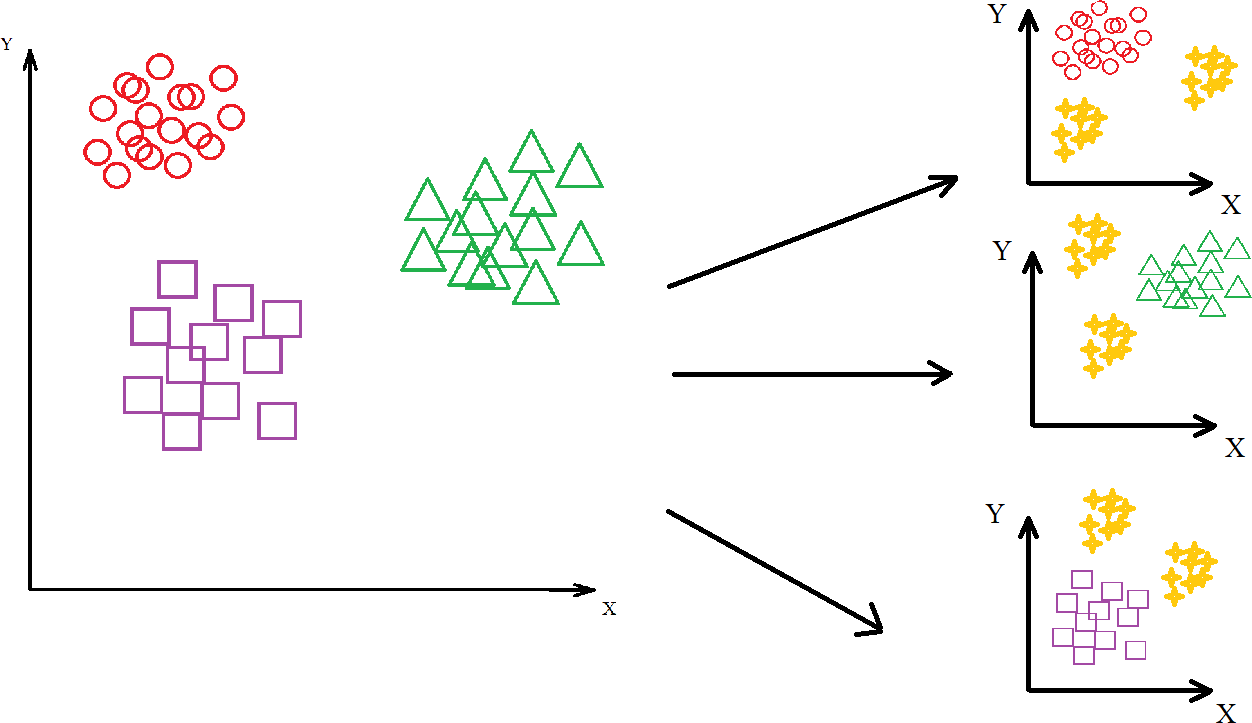
В случае многоклассовой классификации (когда больше одного класса в данных) применяется метод *один против всех*.

Допустим у нас 3 класса:

* + Круг
  + Квадрат
  + Треугольник

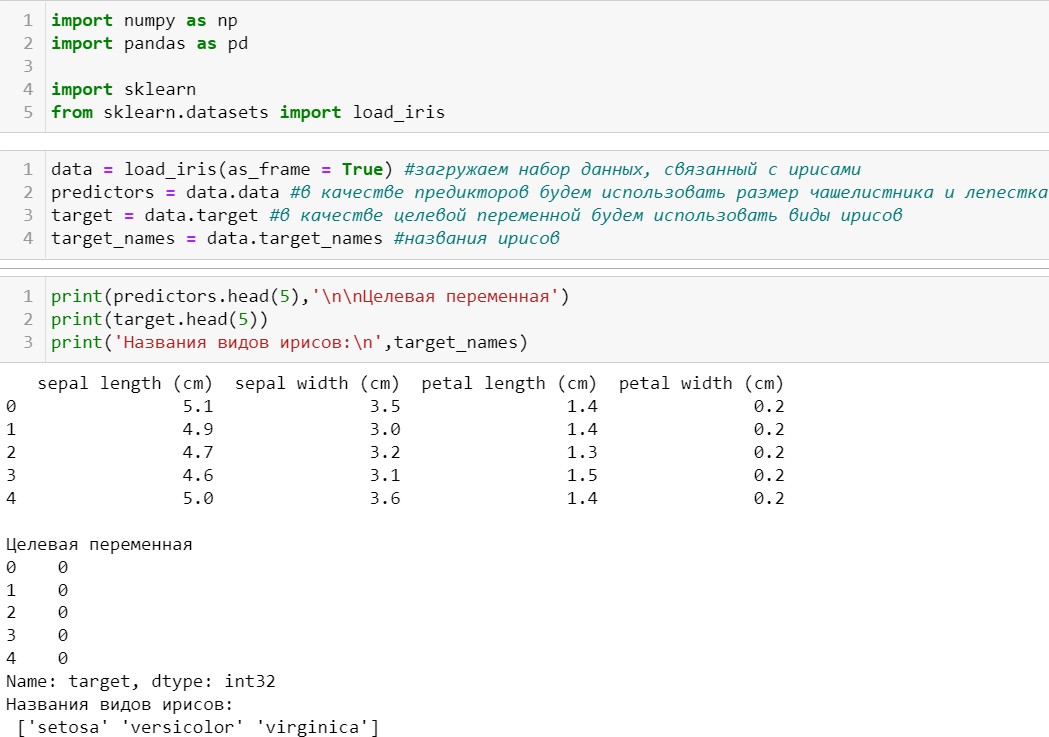


Тогда мы выбираем один из классов, а все остальные объекты объединяем и принимаем за какой-то другой класс:



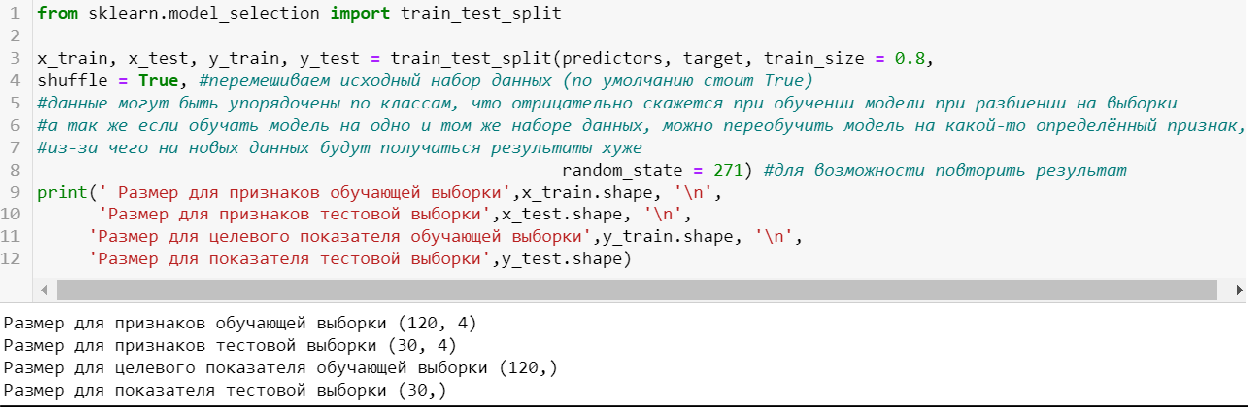
Рассматривается каждый из случаев и рассчитывается вероятность принадлежности объекта к каждому из классов. Где вероятность больше, к тому классу и приписывается объект.

Пример с использованием Python

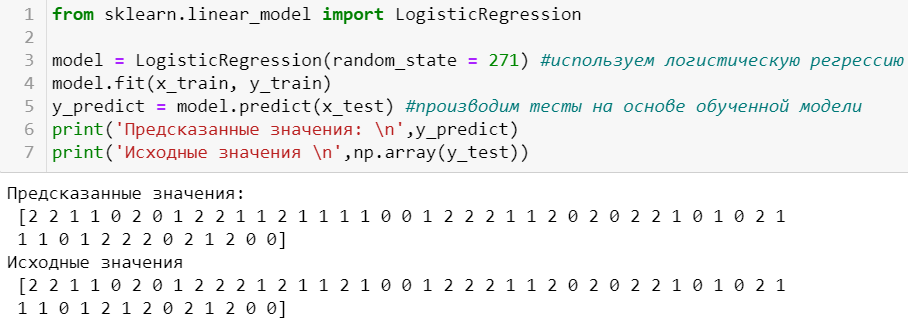
Загрузим из библиотеки sklearn набор данных ирисов. В данном случае за нас уже произведена предобработка данных и классы сбалансированы.

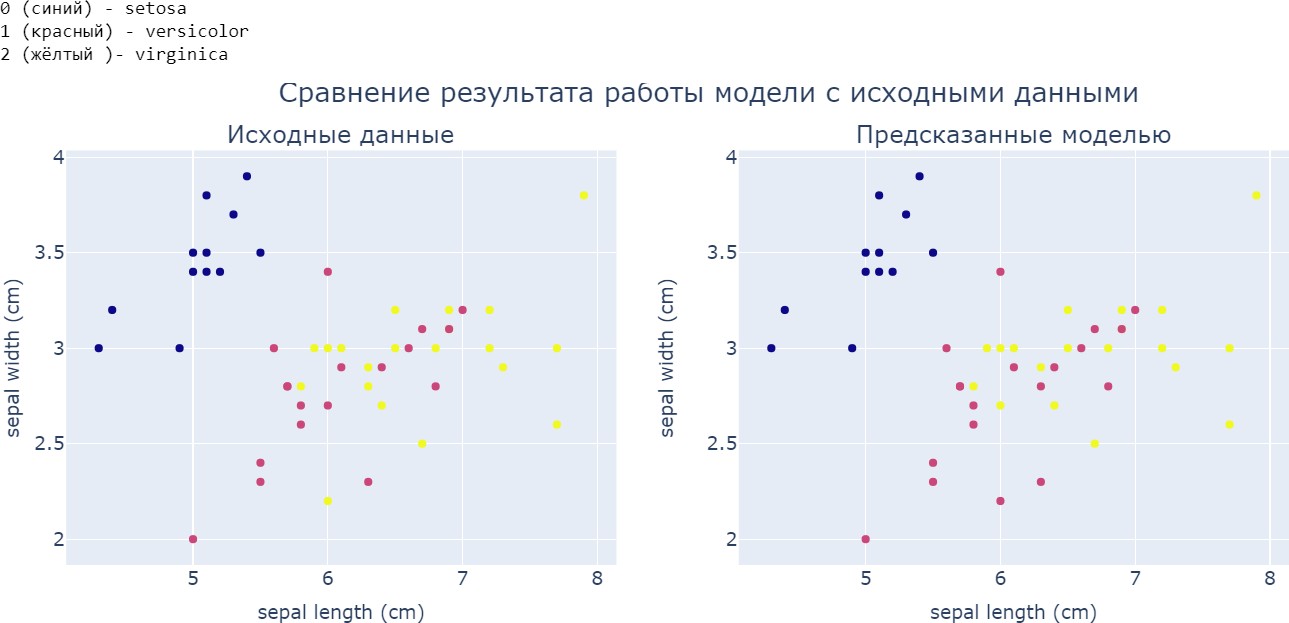
На основе размеров лепестка и чашелистика мы хотим определить вид ириса.

Для начала разобьём наши данные на обучающую и тестовые выборки. Как правило, данные разбивают в процентном соотношении 80% на 20% соответственно, но могут быть и другие сочетания. Для этого воспользуемся методом **train\_test\_split()**.

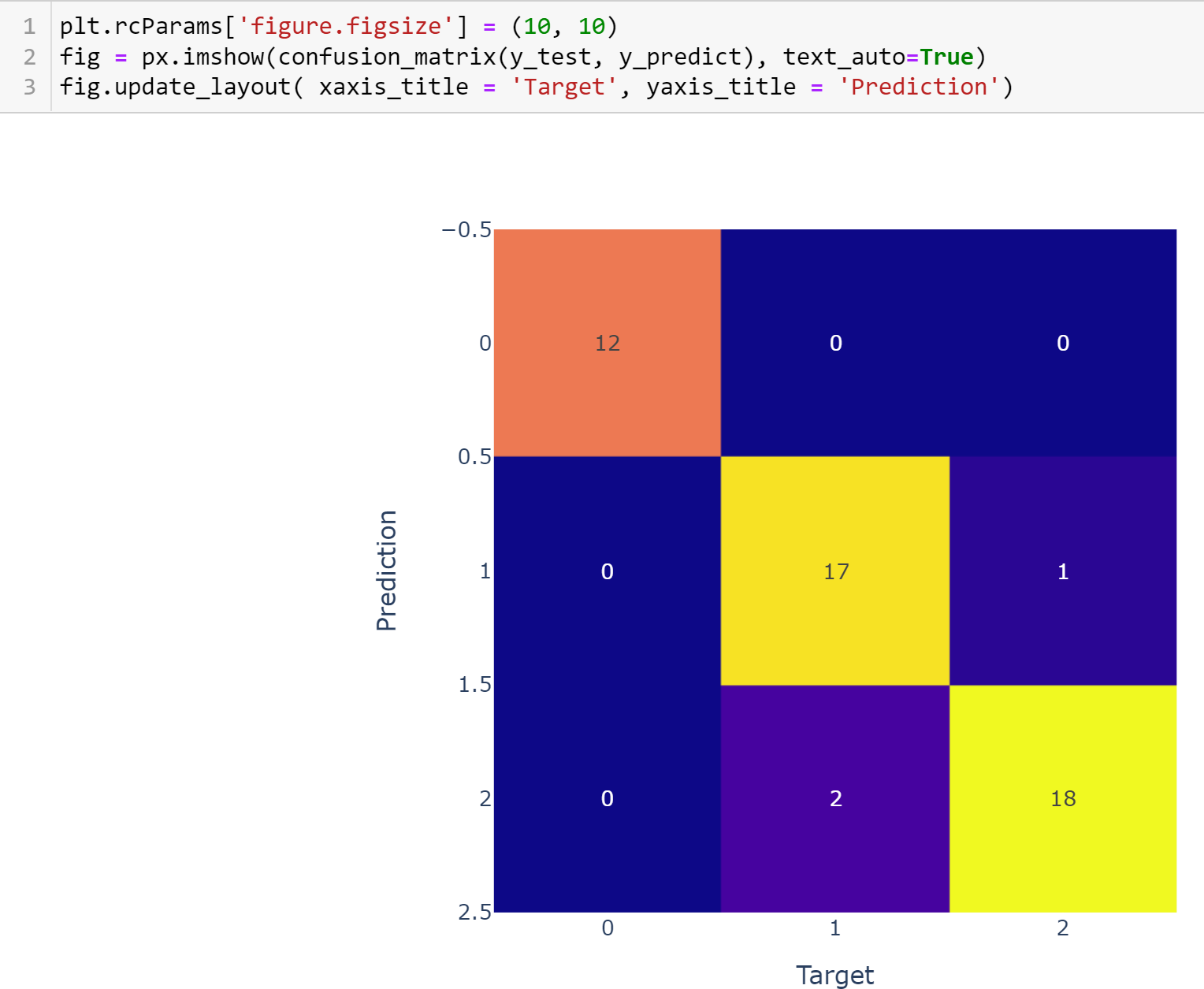


Для минимизации неудачного разбиения данных на тестовую и обучающую выборки применяют кросс-валидацию. Это метод, когда набор данных разбиваются несколько раз на обучающую и тестовую выборки и на этих данных происходит обучение модели.





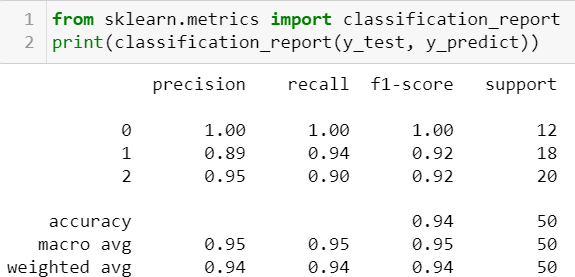
Воспользуемся матрицей ошибок для проверки эффективности нашей модели. В нашем случае 3 класса, поэтому матрица будет размером 3 на 3.



Интерпретируем результат:

На главной диагонали отображается количество правильно предсказанных значений. Рассмотрим второй класс (третья строчка и столбец матрицы). Из 20 объектов 18 ирисов было предсказано правильно, а 2 вида ириса было предсказано неправильно и модель отнесла их ко второму классу.

Оценка качества модели на тестовых данных:



Support: количество наблюдений для каждого класса

Macro avg: среднее арифметическое показателя между классами

weighted avg: средневзвешенное значение рассчитывается путем произведения оценки показателя каждого класса на его количество

наблюдений, последующее суммирование результата и деление результата на сумму наблюдений.

Пример с recall:

Macro avg = (1.00 + 0.94 + 0.90) / 3 = 2.84 / 3 = 0.947

Weighted avg = (1.00 \* 12 + 0.94 \* 18 + 0.90 \* 20) / (12+18+20) =

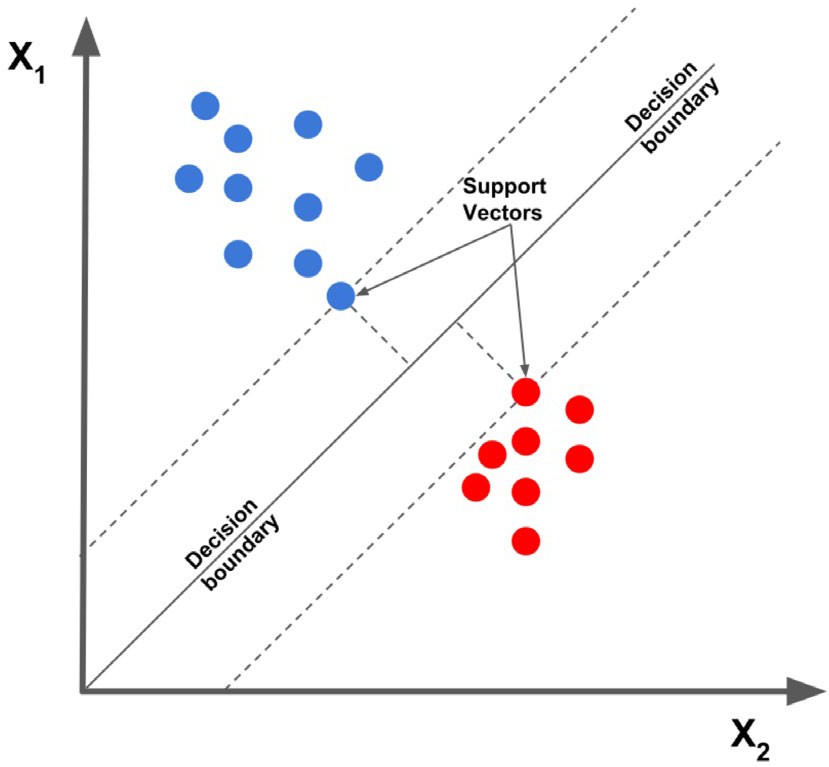
= (12 + 16,92 + 18) / 50 = 0,94

Macro avg используем, когда есть дисбаланс классов и нам важно учесть качество модели на маленьком классе.

Weighted avg используется, когда маленьким классом можно пренебречь, то есть, если модель ошибается на маленьком классе, то Weighted avg не покажет более низкое качество модели, в отличие от Macro avg, который каждому классу отдает одинаковую значимость.

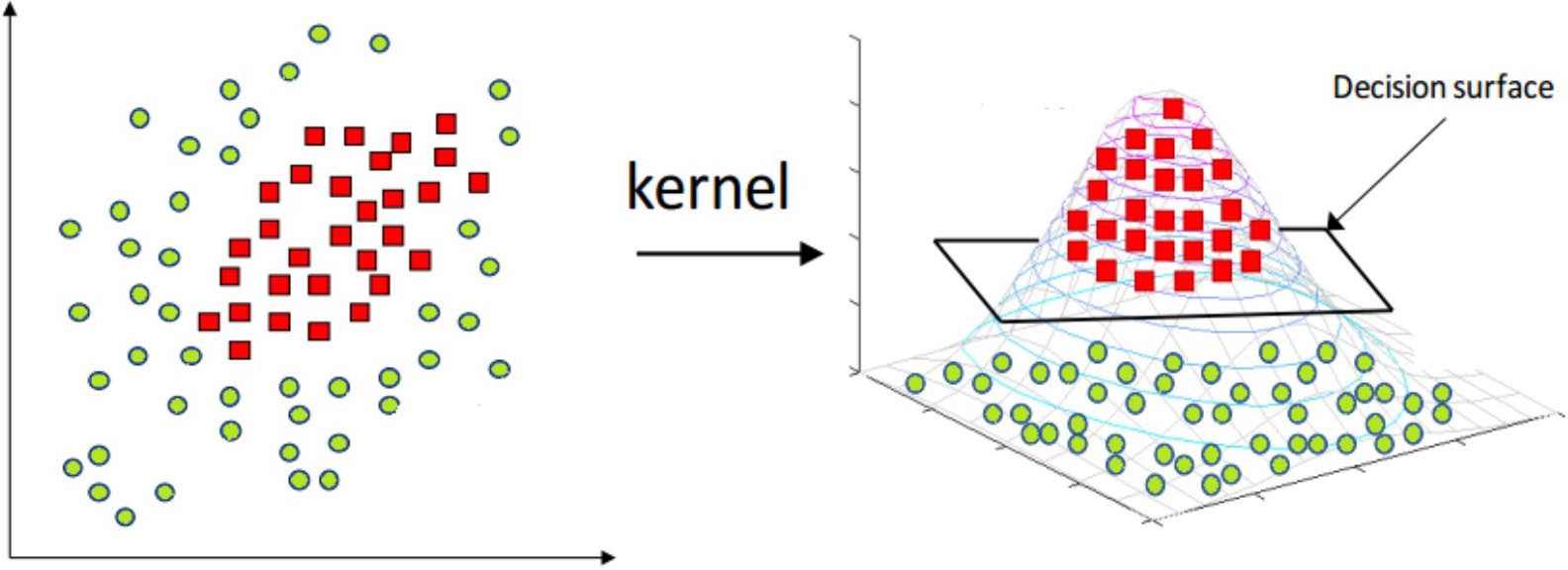
# SVM (метод опорных векторов)

В данном алгоритме мы не пытаемся как-то разделить данные некоторой линией, не подбираем идеальное положение этой линии. В методе опорных векторов происходит *преобразование исходных данных* таким образом, чтобы данные можно было эффективно разделить на классы в другом пространстве. SVM имеет так называемый *ядерный трюк*, который и отвечает за преобразование данных таким образом, чтобы «растащить» объекты разных классов подальше друг от друга и провести прямую или гиперплоскость так, чтобы расстояние между объектами разных классов до линии/гиперплоскости было максимально.



Те объекты, которые находятся ближе всего к разделительной линии, и на основе которых максимизирует расстояние, называются опорными векторами. Преобразование данных происходит с помощью наращивания пространства параметров. То есть появляется новый признак (новое пространство), которые является комбинацией имеющихся признаков. Это позволяет нам увеличить зазор между опорными векторами и разделительной

линией.



Существует несколько ядерных функций (функций преобразования):

* + 𝑘𝑘(𝑥𝑥1, 𝑥𝑥2) = (𝑥𝑥1 ⋅ 𝑥𝑥2 + 𝐴𝐴)𝑚𝑚 – полиномиальная функция;
  + 𝑘𝑘(𝑥𝑥1

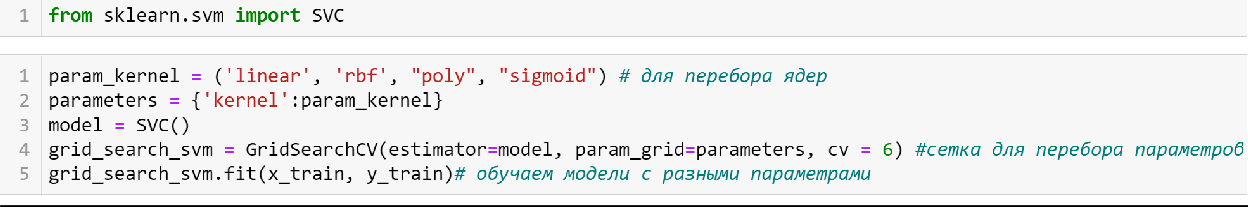
, 𝑥𝑥2

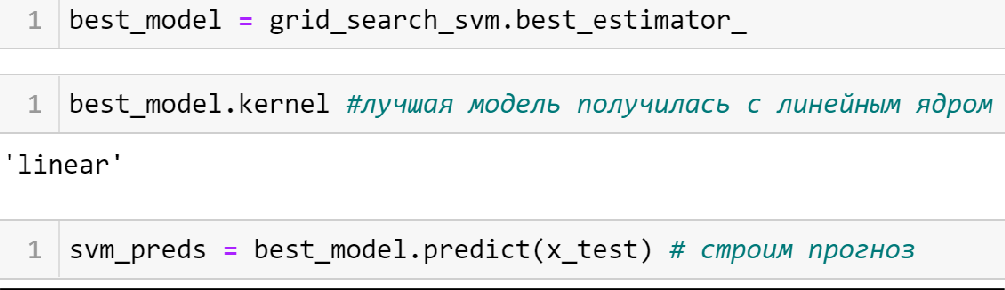
) = 𝑃𝑃−𝛾𝛾(𝑥𝑥1−𝑥𝑥2)2 – RBF (радиально-базисная функция);

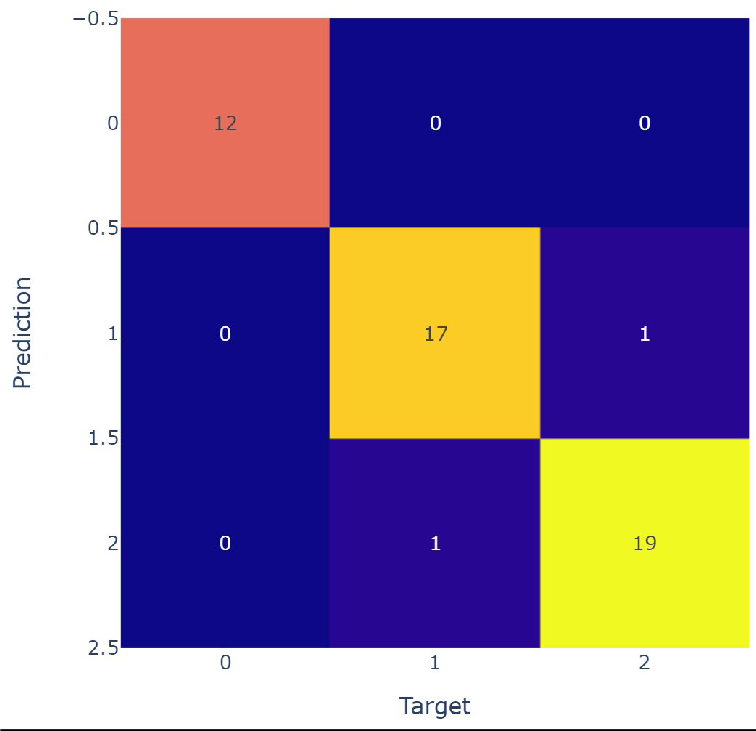
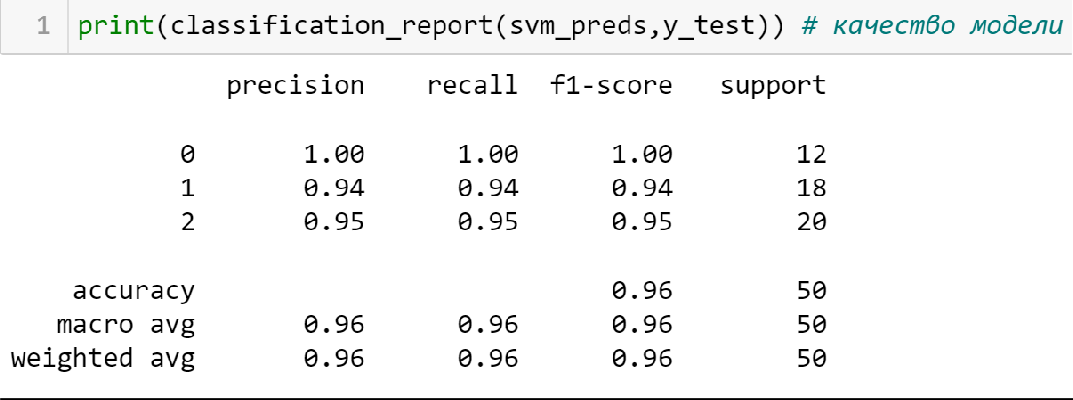
* + 𝑘𝑘(𝑥𝑥1, 𝑥𝑥2) = 𝑥𝑥1 ⋅ 𝑥𝑥2 – линейное ядро;
  + 𝑘𝑘(𝑥𝑥1, 𝑥𝑥2) = tanh (𝑥𝑥1, 𝑥𝑥2) *–* сигмоидальная функция*.*

𝑘𝑘(𝑥𝑥1, 𝑥𝑥2), k – ядро, 𝑥𝑥1 и 𝑥𝑥2 – известная пара объектов в исходном пространстве.

Пример с использованием Python







# KNN

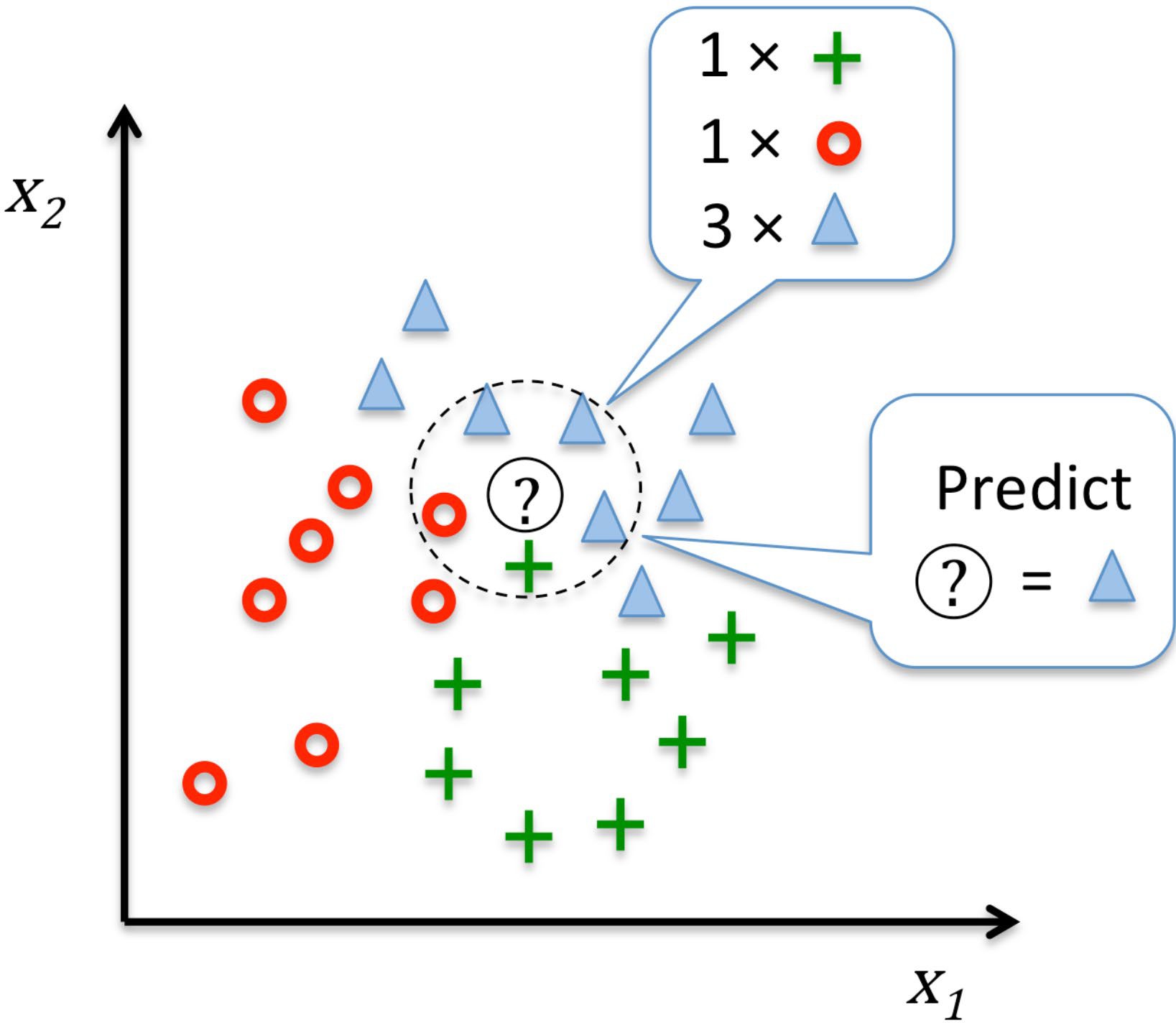
Самый простой алгоритм классификации. Есть объект, необходимо определить, к какому классу он принадлежит. Берется k ближайших соседей объекта. Ближайшие соседи определяются с помощью некоторой метрики, как правило – это евклидово расстояние (сумма разниц координат объектов в квадрате под корнем). Если объект характеризуется 3 признаками, то расстояние до другого объекта вычисляется так

3

‖𝑥𝑥1 − 𝑥𝑥2‖2 = ��(𝑥𝑥1𝑖𝑖 − 𝑥𝑥2𝑖𝑖)2

𝑖𝑖=1

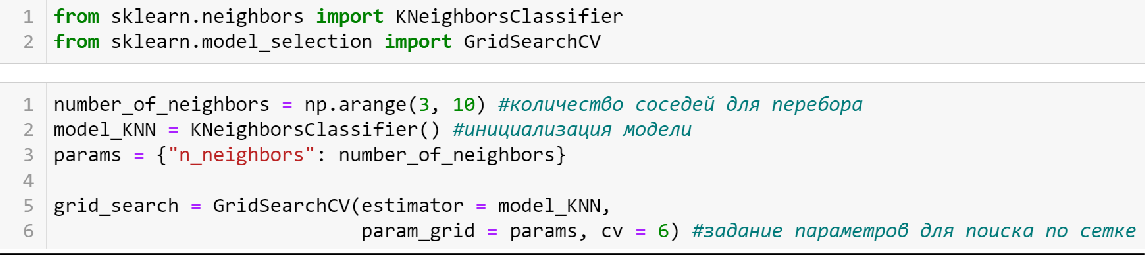
При использовании такой метрики некоторые признаки могут сильнее влиять на расстояние, чем другие признаки, например заработок вычисляется в тысячах, а возраст в десятках. Тогда квадрат разности заработков людей намного сильнее влияет на близость объектов, нежели возраст. Необходимо нормализовать данные.

Далее новому объекту присваивается тот класс, который имеют большинство соседей.

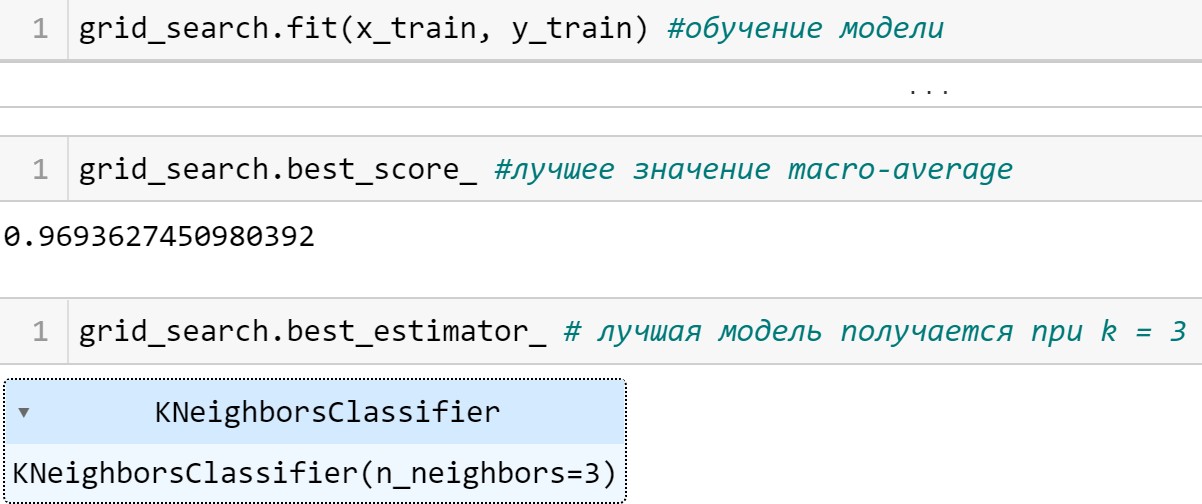
*Новый объект будет относится к классу треугольников*

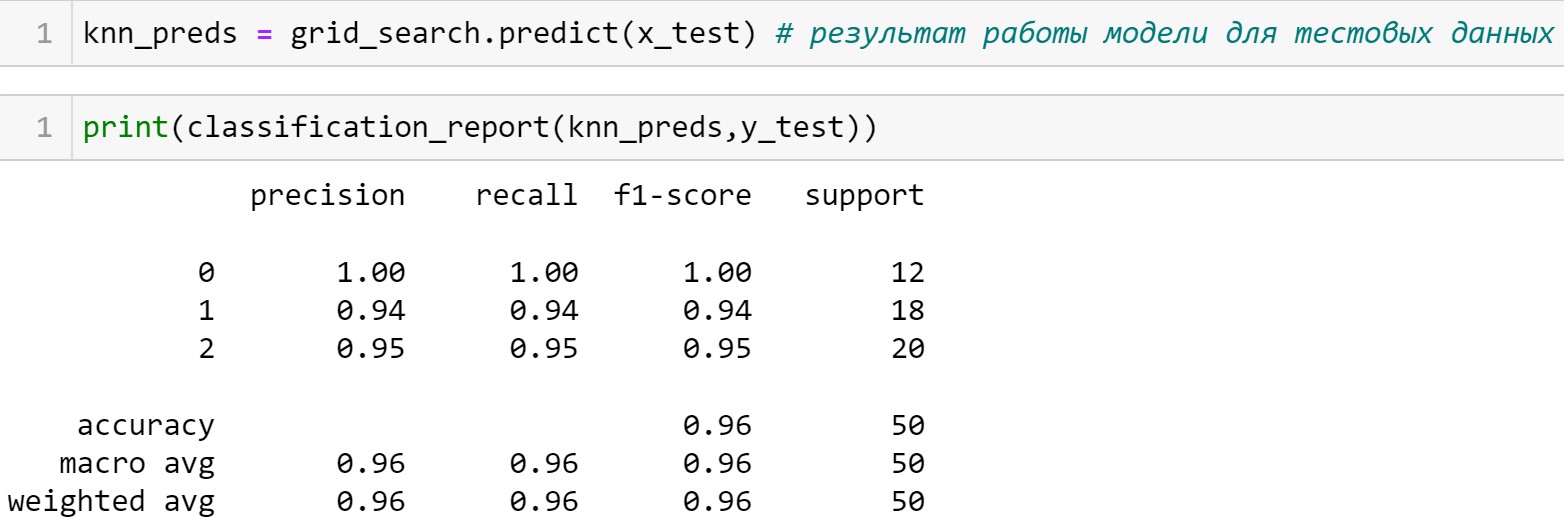
Гиперпараметром алгоритма является количество соседей k. Если k = 1, то класс объекта будет такой же, как и у соседа. Если k = количеству всех элементов в выборке, то новый объект будет иметь класс большинства. Это приведет к тому, что все следующие объекты будут иметь один и тот же класс. Для выбора k можно использовать метод перебора. То есть обучаем модели для разных k и берем то k, которое дает лучшее качество модели.

Пример с использованием Python

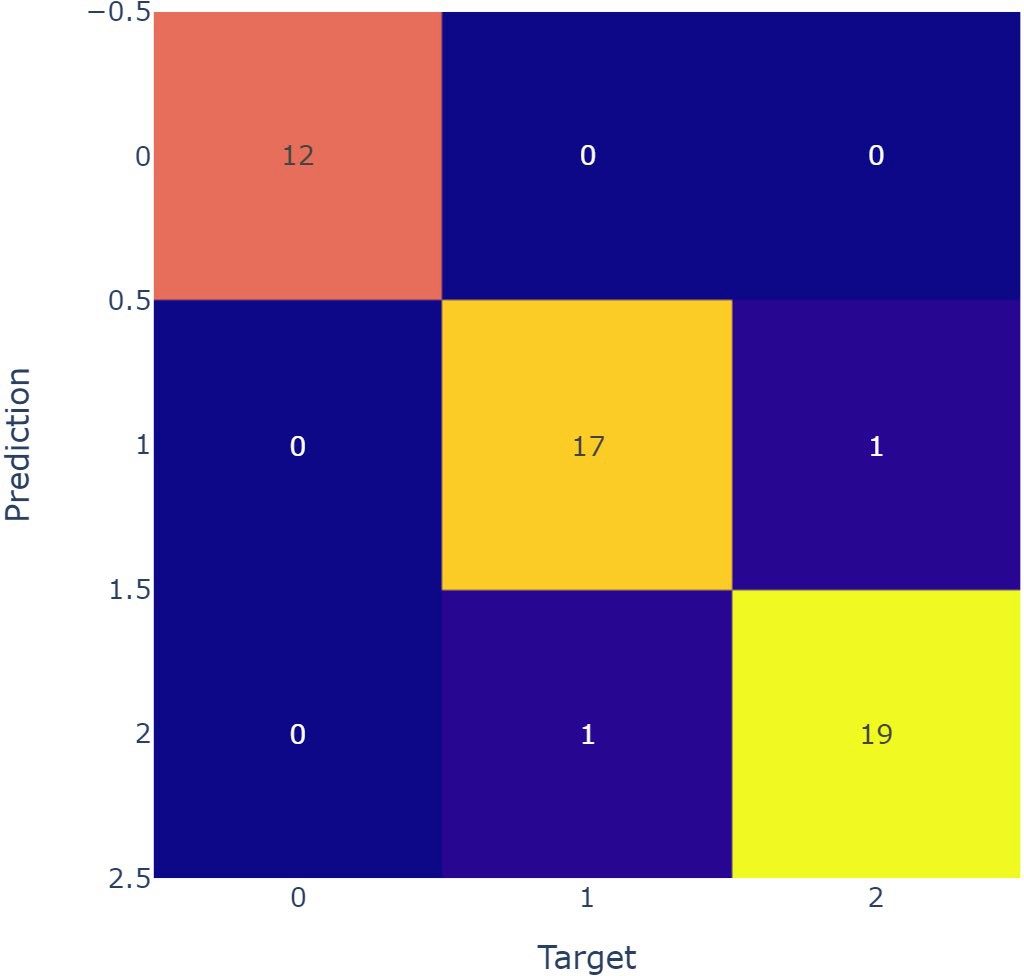


Поиск по сетке GridSearchCV используется для подбора гиперпараметров модели, в данном случае – перебор количества соседей, обучение модели для каждого значения.





Матрица ошибок:



# Практическое задание

1. Найти данные для классификации. Данные в группе повторяться не должны. Предобработать данные, если это необходимо.
2. Изобразить гистограмму, которая показывает баланс классов. Сделать выводы.
3. Разбить выборку на тренировочную и тестовую. Тренировочная для обучения модели, тестовая для проверки ее качества.
4. Применить алгоритмы классификации: логистическая регрессия, SVM, KNN. Построить матрицу ошибок по результатам работы моделей (использовать confusion\_matrix из sklearn.metrics).
5. Сравнить результаты классификации, используя accuracy, precision, recall и f1-меру (можно использовать classification\_report из sklearn.metrics). Сделать выводы.
6. Оформить отчет о проделанной работе.