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 несколько алгоритмов машинного обучения, которые решают задачу классификации:

- логистическая регрессия;
- Support Vector Machine (SVM);
- дерево решений, случайный лес, градиентный бустинг;
- k Nearest Neighbor (KNN);
- нейронные сети;
- Байесовский классификатор.

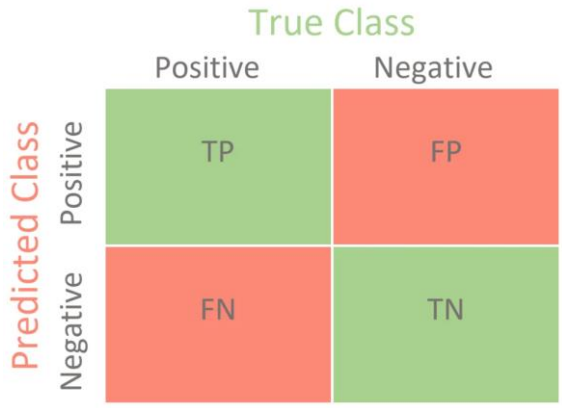
Метрики оценки качества модели для задачи классификации

Accuracy (sklearn.metrics.accuracy_score). Доля правильных ответов.

$$Accuracy = \frac{\text{Сумма верно предсказанных}}{\text{Количество всех объектов}}$$

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? – Просто 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.

$$Precision = \frac{TP}{TP + FP}$$

Recall (sklearn.metrics.recall_score). Полнота.

$$Recall = \frac{TP}{TP + FN}$$

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

$$F_\beta = (1 + \beta^2)\frac{Precision \cdot Recall}{\beta^2 \cdot Precision + Recall}$$

β – это параметр, который определяет приоритет точности или полноты. Это зависит от задачи. Если $\beta > 1$, то Recall в приоритете, если $\beta < 1$, 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 F_1-мера (sklearn.metrics.f1_score), для нее $\beta = 1$. То есть предпочтение не отдается ни точности, ни полноте.

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Рассмотрим некоторые алгоритмы классификации подробнее.

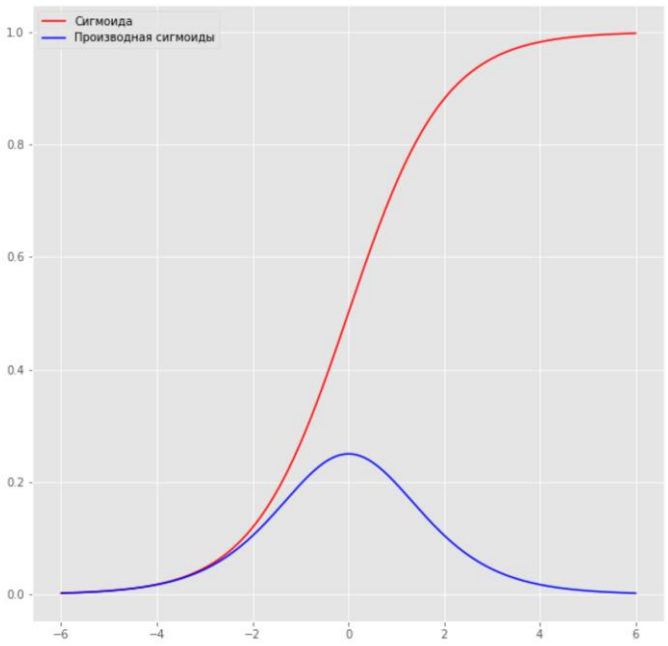
Логистическая регрессия

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.



$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

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

$$\sigma(x, w) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + \cdots + w_m x_m)}}$$

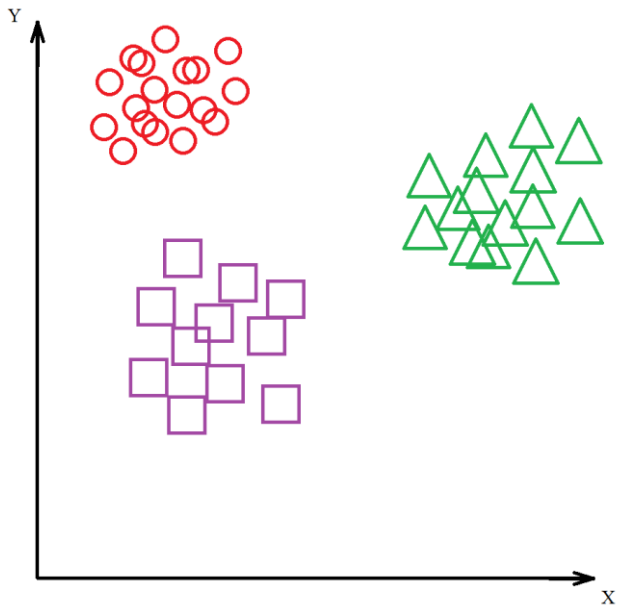
Веса ω отвечают за положение сигмоиды. Задача состоит в том, чтобы подобрать такую сигмоиду, которая будет наилучшим образом проходить через точки в пространстве. А какое положение сигмоиды является лучшим – решает функция потерь или функция стоимости. Для бинарной классификации используется так называемая *бинарная кросс-энтропия*

$$BCE(x, \omega, y) = \sum_{1}^{n} (-y_i \cdot \ln(\sigma(x, w)) - (1 - y_i) \cdot \ln(1 - \sigma(x, w)))$$

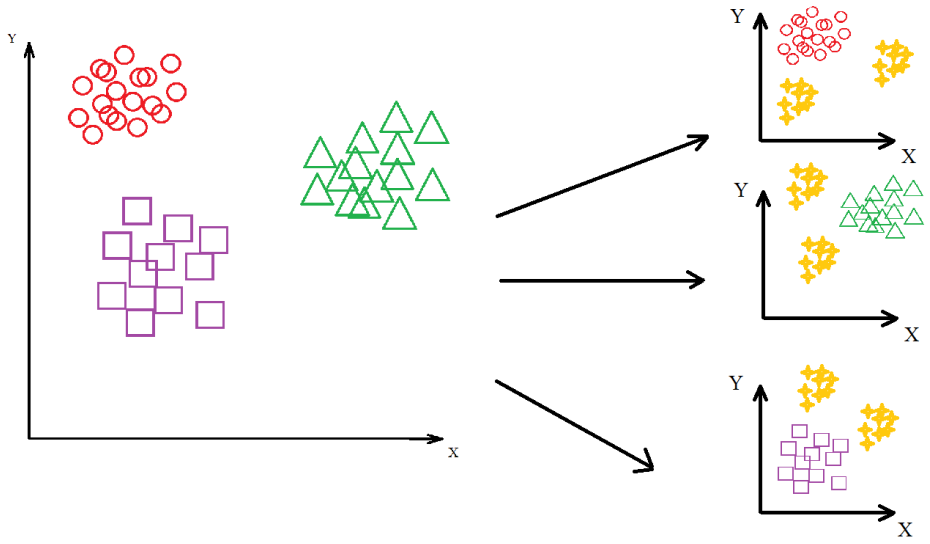
В случае многоклассовой классификации (когда больше одного класса в данных) применяется метод *один против всех*.

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 у к одному 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 и балансировка классов.

```
import numpy as np
import pandas as pd

import sklearn
from sklearn.datasets import load_iris
```

```
data = load_iris(as_frame = True) #загружаем набор данных, связанный с ирисами
predictors = data.data #в качестве предикторов будем использовать размер чашелистника и лепестка
target = data.target #в качестве целевой переменной будем использовать виды ирисов
target_names = data.target_names #названия ирисов
```

```
print(predictors.head(5),'\n\nЦелевая переменная')
print(target.head(5))
print('Названия видов ирисов:\n',target_names)
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

Целевая переменная
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int32
Названия видов ирисов:
 ['setosa' 'versicolor' 'virginica']
```

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()**.

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(predictors, target, train_size = 0.8,
shuffle = True, #перемешиваем исходный набор данных (по умолчанию стоит True)
#данные могут быть упорядочены по классам, что отрицательно скажется при обучении модели при разбиении на выборки
#а так же если обучать модель на одно и том же наборе данных, можно переобучить модель на какой-то определённый признак,
#из-за чего на новых данных будут получаться результаты хуже
                                                random_state = 271) #для возможности повторить результат
print(' Размер для признаков обучающей выборки',x_train.shape, '\n',
      'Размер для признаков тестовой выборки',x_test.shape, '\n',
     'Размер для целевого показателя обучающей выборки',y_train.shape, '\n',
     'Размер для показателя тестовой выборки',y_test.shape)
```

```
Размер для признаков обучающей выборки (120, 4)
Размер для признаков тестовой выборки (30, 4)
Размер для целевого показателя обучающей выборки (120,)
Размер для показателя тестовой выборки (30,)
```

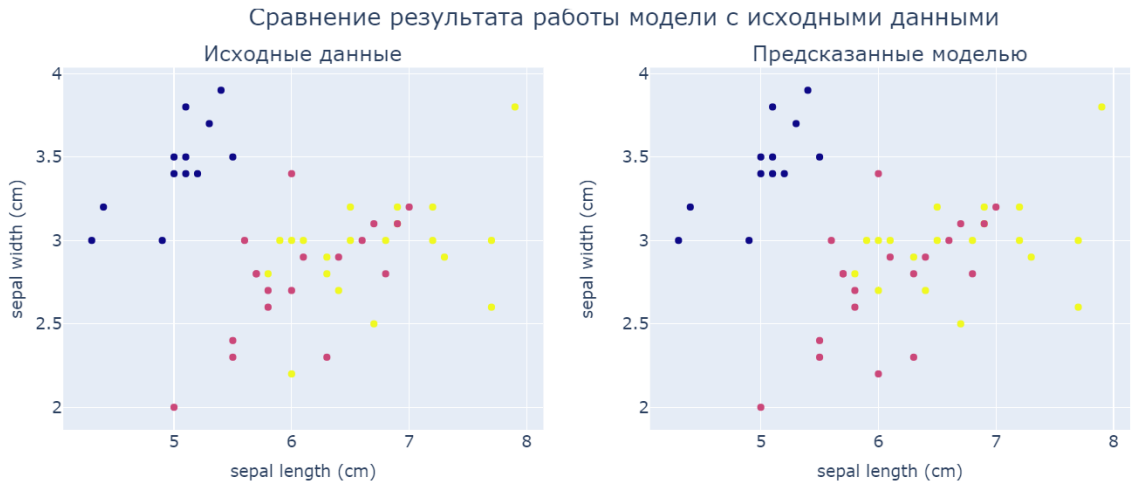
Для уменьшения переобучения и минимизации «удачного» разбиения данных на тестовую и обучающую выборки применяют кросс-валидацию. Это метод, когда набор данных разбиваются несколько раз на обучающую и тестовую выборки и на этих данных происходит обучение модели.

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(random_state = 271) #используем логистическую регрессию
model.fit(x_train, y_train)
y_predict = model.predict(x_test) #производим тесты на основе обученной модели
print('Предсказанные значения: \n',y_predict)
print('Исходные значения \n',np.array(y_test))
```

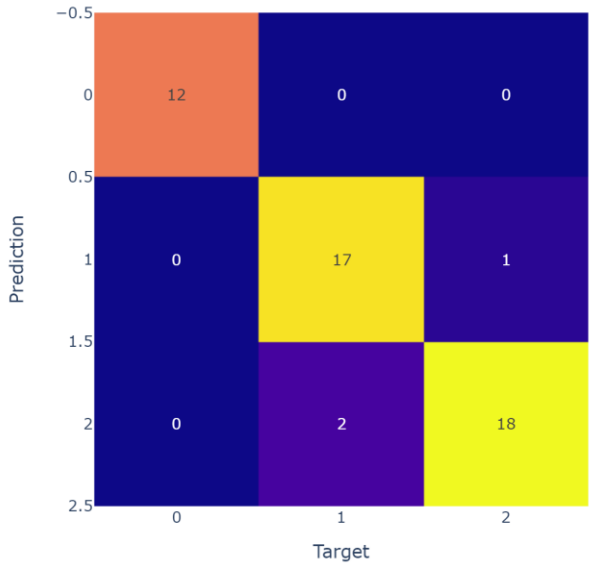
```
Предсказанные значения:
 [2 2 1 1 0 2 0 1 2 2 1 1 2 1 1 1 1 0 0 1 2 2 2 1 1 2 0 2 0 2 2 1 0 1 0 2 1
 1 1 0 1 2 2 2 0 2 1 2 0 0]
Исходные значения
 [2 2 1 1 0 2 0 1 2 2 2 1 2 1 1 2 1 0 0 1 2 2 2 1 1 2 0 2 0 2 2 1 0 1 0 2 1
 1 1 0 1 2 1 2 0 2 1 2 0 0]
```

```
0 (синий) - setosa
1 (красный) - versicolor
2 (жёлтый )- virginica
```



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

```
plt.rcParams['figure.figsize'] = (10, 10)
fig = px.imshow(confusion_matrix(y_test, y_predict), text_auto=True)
fig.update_layout( xaxis_title = 'Target', yaxis_title = 'Prediction')
```



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:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.89      0.94      0.92        18
           2       0.95      0.90      0.92        20

    accuracy                           0.94        50
   macro avg       0.95      0.95      0.95        50
weighted avg       0.94      0.94      0.94        50
```

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.

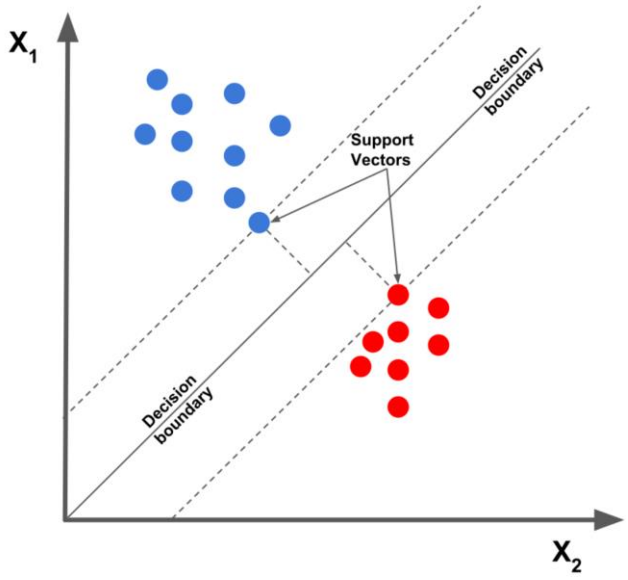
Пример с recall:

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

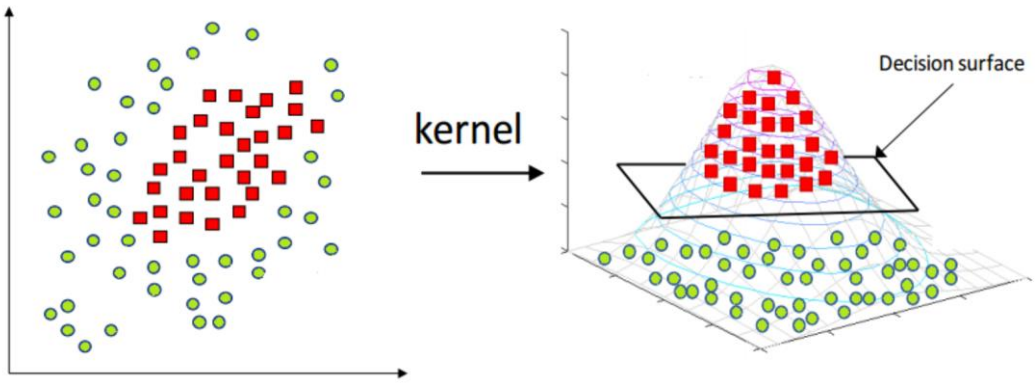
SVM (метод опорных векторов)

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):

- $k(x_1, x_2) = (x_1 \cdot x_2 + a)^m$ – полиномиальная функция;
- $k(x_1, x_2) = e^{-\gamma(x_1 - x_2)^2}$ – RBF (радиально-базисная функция);
- $k(x_1, x_2) = x_1 \cdot x_2$ – линейное ядро;
- $k(x_1, x_2) = \tanh(x_1, x_2)$ – сигмоидальная функция.

$k(x_1, x_2)$ – k – ядро, x_1 и x_2 – известная пара объектов в исходном пространстве.

Пример с использованием Python

```
from sklearn.svm import SVC
```

```
param_kernel = ('linear', 'rbf', "poly", "sigmoid") # для перебора ядер
parameters = {'kernel':param_kernel}
model = SVC()
grid_search_svm = GridSearchCV(estimator=model, param_grid=parameters, cv = 6) #сетка для перебора параметров
grid_search_svm.fit(x_train, y_train)# обучаем модели с разными параметрами
```

```
best_model = grid_search_svm.best_estimator_
```

```
best_model.kernel #лучшая модель получилась с линейным ядром
```

```
'linear'
```

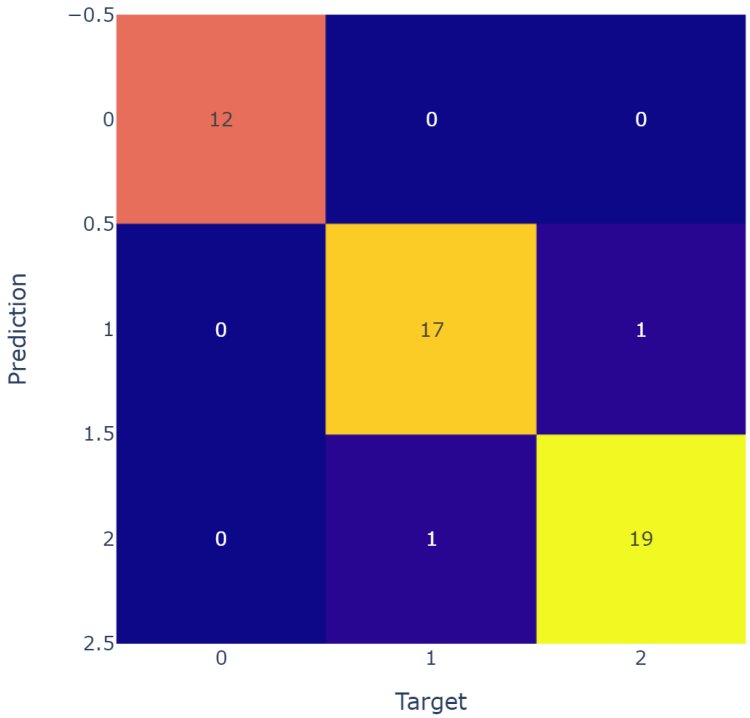
```
svm_preds = best_model.predict(x_test) # строим прогноз
```

```
print(classification_report(svm_preds,y_test)) # качество модели
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.94      0.94      0.94        18
           2       0.95      0.95      0.95        20

    accuracy                           0.96        50
   macro avg       0.96      0.96      0.96        50
weighted avg       0.96      0.96      0.96        50
```



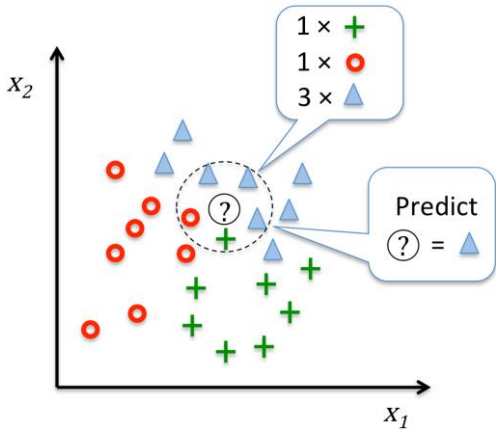
KNN

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

$$\|x_1 - x_2\|_2 = \sqrt{\sum_{i=1}^{3}(x_{1i} - x_{2i})^2}$$

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

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
```

```
number_of_neighbors = np.arange(3, 10 ,25) #количество соседей для перебора
model_KNN = KNeighborsClassifier() #инициализация модели
params = {"n_neighbors": number_of_neighbors}

grid_search = GridSearchCV(estimator = model_KNN,
                           param_grid = params, cv = 6) #задание параметров для поиска по сетке
```

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

```
grid_search.fit(x_train, y_train) #обучение модели
```

```
...
```

```
grid_search.best_score_ #лучшее значение macro-average
```

```
0.9693627450980392
```

```
grid_search.best_estimator_ # лучшая модель получается при k = 3
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
knn_preds = grid_search.predict(x_test) # результат работы модели для тестовых данных
```

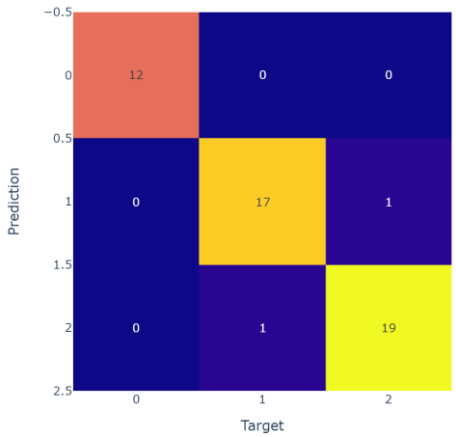
```
print(classification_report(knn_preds,y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.94      0.94      0.94        18
           2       0.95      0.95      0.95        20

    accuracy                           0.96        50
   macro avg       0.96      0.96      0.96        50
weighted avg       0.96      0.96      0.96        50
```

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). Также сравнить время работы алгоритмов. Сделать выводы.
6. Оформить отчет о проделанной работе.