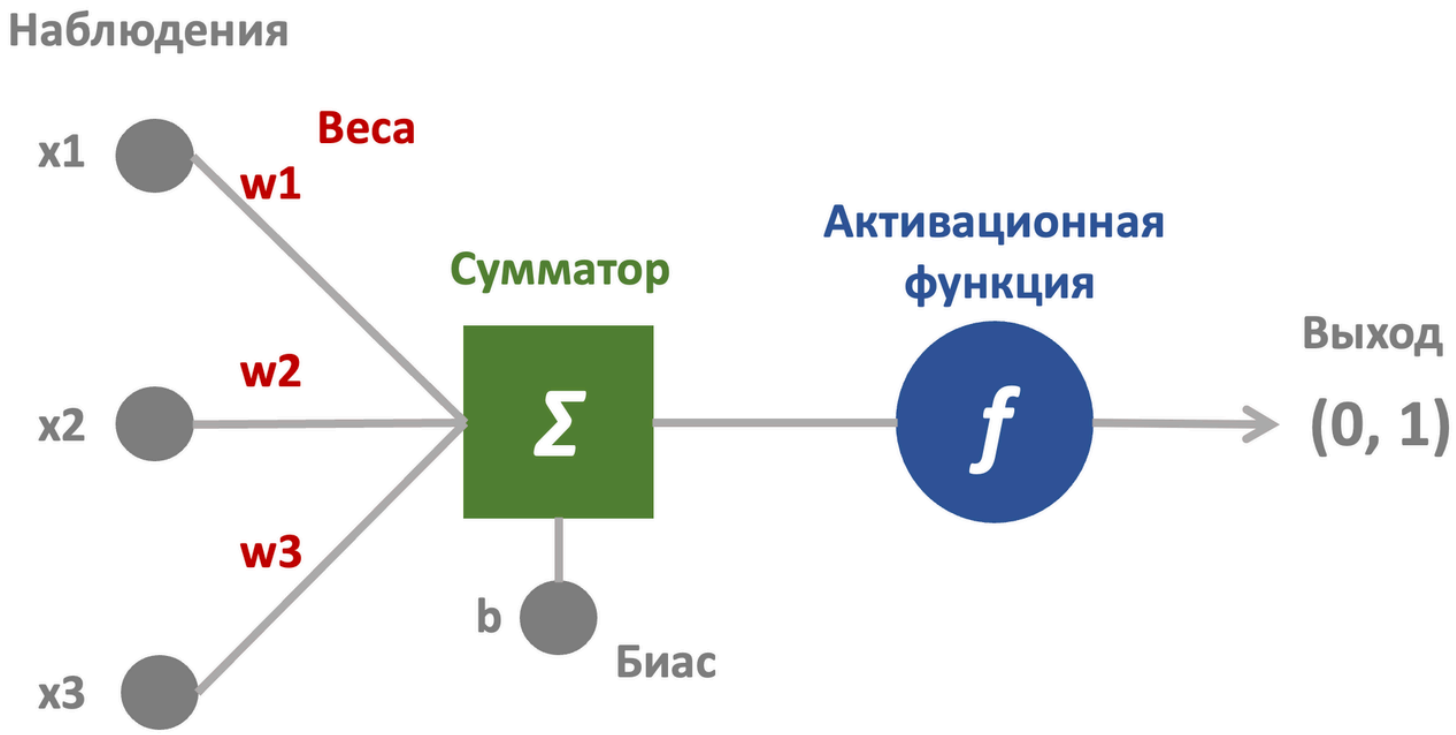
Персептроны и работа с ними в Python. Машинное обучение и анализ данных.

Перцептрон — это тип искусственной нейронной сети, который впервые был предложен Фрэнком Розенблаттом в 1958 году. Он представляет собой простейшую форму искусственного нейронного устройства и является основой для более сложных моделей нейронных сетей.

Перцептрон состоит из:

- Входные данные (input): Перцептрон принимает на вход набор значений, которые представляют собой признаки данных. Каждый вход соответствует одному из признаков.
- Весовые коэффициенты (weights): Каждому входному значению соответствует вес, который определяет его значимость для окончательного вывода. Эти веса настраиваются в процессе обучения.
- Сумматор (summation function): Все входные значения умножаются на свои соответствующие веса, и затем они суммируются. Это вычисление можно выразить формулой:
  z = sum(w_i*x_i) + b, где (x_i) — входные данные, (w_i) — веса, (b) — смещение (bias).
- Активационная функция (activation function): Результат сумматора проходит через активационную функцию, которая определяет выходной сигнал перцептрона. В простейших случаях используется пороговая функция или функция шагов, которая выдает 1, если сумма превышает определенный порог, и 0 в противном случае. Более сложные перцептроны могут использовать сигмоидальную, ReLU или другие активационные функции.
- Выход (output): После применения активационной функции перцептрон выдает выходное значение, которое может быть использовано для классификации или других задач.



(https://postimg.cc/NLGsSfdM)

Перцептрон может быть как однослойным (состоящим из одного слоя нейронов), так и многослойным, что позволяет решать более сложные задачи. Многослойные перцептроны (MLP) являются основой для более сложных нейронных сетей в глубоких обучениях.
Рассмотрим работу с перцептроном на примере:

In [1]:
```
import pandas as pd # Импорт библиотеки для работы с датафреймом
import numpy as np # Импорт библиотеки для работы с массивами
from sklearn.model_selection import train_test_split # Разделение выборки на тестовоую и тренировочную
from sklearn.linear_model import Perceptron # Модель перцептрона
from sklearn.metrics import accuracy_score # Метрика точности предсказания
import matplotlib.pyplot as plt # Библиотеки для работы с визуализацией
%matplotlib inline
import seaborn as sns

df = pd.read_csv('mushroom_cleaned.csv') # Загружаем датасет по классификации грибо на ядовитые и не ядовитые
# Ссылка на датасет: https://www.kaggle.com/datasets/prishasawhney/mushroom-dataset
df.head()
```

Out[1]:

	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	class
0	1372	2	2	10	3.807467	1545	11	1.804273	1
1	1461	2	2	10	3.807467	1557	11	1.804273	1
2	1371	2	2	10	3.612496	1566	11	1.804273	1
3	1261	6	2	10	3.787572	1566	11	1.804273	1
4	1305	6	2	10	3.711971	1464	11	0.943195	1

In [2]:
```
# Рассмотрим модель перцептрона из библиотеки sklearn
X, y = df[['cap-diameter','cap-shape', 'gill-attachment', 'gill-color', 'stem-height','stem-width', 'stem-color', 'season' ]], df['class'] # Разделяем переменные на зависимые и независимую
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # Разделяем выборку на тестовую и тренировочную
perceptron = Perceptron(max_iter=100, eta0=0.1, random_state=42) # Объявляем перцептрон
perceptron.fit(X_train, y_train) # Тренировка модели
y_pred = perceptron.predict(X_test) # Предсказываем значения
y_pred[:10]
```

Out[2]:
```
array([0, 0, 0, 1, 1, 1, 1, 0, 1, 0])
```

In [3]:
```
# Рассмотрим полученную точность
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
# 0.57 - удовлетворительное значение для задачи классификации
```

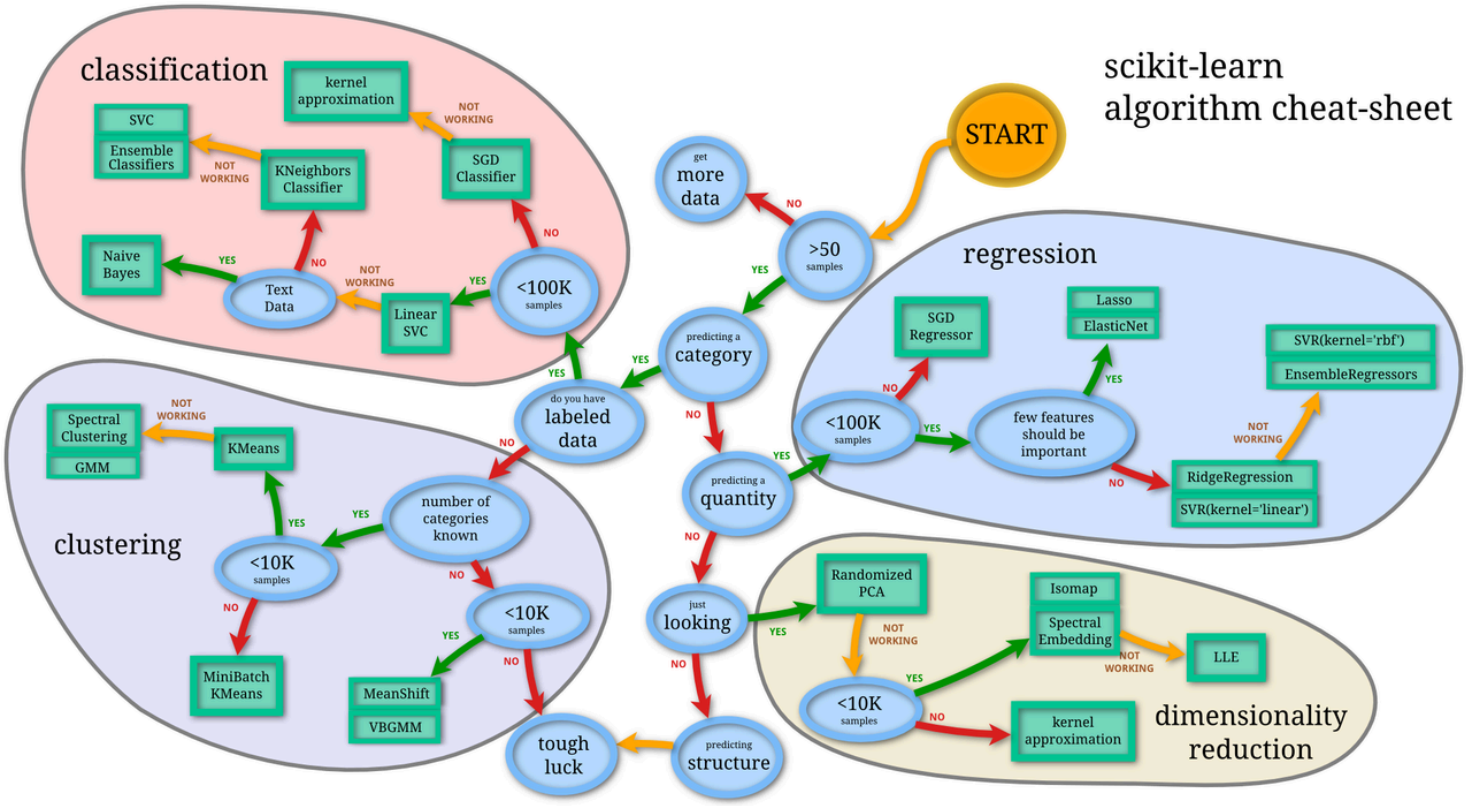
```
Accuracy: 0.5712038493568983
```

Машинное обучение — это совокупность методов искусственного интеллекта, с помощью которых можно создавать самообучающиеся компьютерные системы (в частности, нейросети).
Анализ данных и машинное обучение дополняют друг друга: анализ данных помогает подготовить и понять данные, в то время как машинное обучение позволяет извлечь более глубокие инсайты и построить предсказательные модели.

Типы машинного обучения

- Обучение с учителем: модель обучается на размеченных данных, где известны входные и выходные значения.
- Обучение без учителя: модель работает с неразмеченными данными и ищет скрытые закономерности.
- Обучение с подкреплением: модель обучается через взаимодействие с окружающей средой, получая награды или наказания за свои действия.
- Глубинное обучение: модель основана на обучении многослойных нейронных сетей и создается по аналогии с человеческим мозгом.

Машинное обучение решает задачи классификации, регрессии, кластеризации, снижения размерности, в каждую область входит большое количество алгоримов. Заранее невозможно определить какой алгоритм даст наибольшую эффективность, поэтому существуют следующие схемы работы с алгоритмами:



(https://postimg.cc/nXm0vm9w)
Рассмотрим на примере ниже как при помощи алгоритмов аналитики данных и машинного обучения можно повычить точность предсказания классов для предыдущего датафрейма:

In [4]:
```
# Исследуем массив данных на предмет пропущенных значений
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54035 entries, 0 to 54034
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   cap-diameter     54035 non-null  int64  
 1   cap-shape        54035 non-null  int64  
 2   gill-attachment  54035 non-null  int64  
 3   gill-color       54035 non-null  int64  
 4   stem-height      54035 non-null  float64
 5   stem-width       54035 non-null  int64  
 6   stem-color       54035 non-null  int64  
 7   season           54035 non-null  float64
 8   class            54035 non-null  int64  
dtypes: float64(2), int64(7)
memory usage: 3.7 MB
```

In [5]:
```
# Проверяем наличие дубликатов и удаляем их
print("Дублирующихся строк: ", df[df.duplicated()].shape[0])
df = df.drop_duplicates(keep = 'first')  # Перезаписываем датафрейм без дубликатов
```

```
Дублирующихся строк:  303
```

In [6]:
```
# Выведем описательную статистику для датафрейма
df.describe()
# По таблице можем заметить, что паказатели cap-diameter и stem-width имеют достаточно большое стандартное отклонение
# Такое значение могут оказывать выбросы, которые дополнительно влияют на точность работы алгоритмов машинного обучения
```

Out[6]:

	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	class
count	53732.000000	53732.000000	53732.000000	53732.000000	53732.000000	53732.000000	53732.000000	53732.000000	53732.000000
mean	568.629178	4.005900	2.142857	7.344599	0.752102	1057.000633	8.454013	0.952322	0.546639
std	360.384461	2.165188	2.232546	3.190447	0.645987	780.263033	3.235507	0.303795	0.497825
min	0.000000	0.000000	0.000000	0.000000	0.000426	0.000000	0.000000	0.027372	0.000000
25%	290.000000	2.000000	0.000000	5.000000	0.270146	430.000000	6.000000	0.888450	0.000000
50%	528.000000	5.000000	1.000000	8.000000	0.589316	929.000000	11.000000	0.943195	1.000000
75%	782.000000	6.000000	4.000000	10.000000	1.046900	1527.000000	11.000000	0.943195	1.000000
max	1891.000000	6.000000	6.000000	11.000000	3.835320	3569.000000	12.000000	1.804273	1.000000

In [7]:
```
# Удалим выбросы по методу межквартильного размаха
for i in df.columns: # В цикле для всех столбцов массива
    Q1 = df[i].quantile(0.25) # Расчитываем нижний квартиль
    Q3 = df[i].quantile(0.75) # Расчитываем верхний квартиль
    IQR = Q3 - Q1 # Межквартильный размах
    df = df[(df[i] < Q3 + 1.5 * IQR) & (df[i] > Q1 - 1.5 * IQR)] # Отбираем только те значения датафрейма, в которых
    # переменная меньше чем верхний квартиль + 1,5 * межквартильный размах и больше чем нижний квартиль + 1,5 * межквартильный размах
```

In [8]:
```
# Далее приведем все независимые переменные к стандартной шкале
from sklearn.preprocessing import StandardScaler # Импортируем метод StandardScaler
scaler = StandardScaler() # Создаем объект
X = scaler.fit_transform(df[['cap-diameter','cap-shape', 'gill-attachment', 'gill-color', 'stem-height','stem-width', 'stem-color', 'season' ]]) # Стандартизация
y = df['class'] # Переменную с классами объявляем отдельно
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # Разделяем выборку на тренировочную и тестовую
```

In [9]:
```
# По схеме приведенной выше видим, что если набор данных содержит менее 100 тысяч записей, то следует использовать метод LinearSVC – метод опорных векторов
# Суть алгоритма – найти гиперплоскость для максимизации расстояния между классифицированными образцами
from sklearn.svm import LinearSVC # Импортируем метод
svc = LinearSVC(dual="auto", random_state=42) # Создаем объект
svc.fit(X_train, y_train) # Обучаем модель
y_pred = svc.predict(X_test) # Предсказываем значения
```

In [10]:
```
# Проверим точность классификации методом опорных векторов
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
# Точность повысилась до 0,64, но все еще является удовлетворительной
```

```
Accuracy: 0.6367521367521367
```

In [11]:
```
# Далее по схеме используем классификатор к ближайших соседей
# Данный алгоритм определяет ближайшие точки данных (соседей) для точки запроса на основе вычисляемых расстояний
from sklearn.neighbors import KNeighborsClassifier # Импортируем метод
knn = KNeighborsClassifier(n_neighbors=5) # Создаем объект
knn.fit(X_train, y_train) # Обучаем модель
y_pred = knn.predict(X_test) # Предсказываем значения
```

In [12]:
```
# Проверяем значение точности
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
# Точность повысилась до 0,99, что является отличным значением
```

```
Accuracy: 0.9851551956815114
```

Задание 1

При помощи библиотеки sklearn и схемы выше решить задачи кластеризации, регрессии и снижения размерности для набора данных из ноутбука.

Задание 2 (*)

Написать алгоритм работы перцептрона