Функционал библиотеки Numpy

In [1]:
```
import numpy as np
list = [1,2,3,4]
arr = np.array(list)
arr
```

Out[1]:
```
array([1, 2, 3, 4])
```

Другие способы создания массивов данных

Матрица с нулями, где 1 аргумент - ранг, 2 - размер

In [3]:
```
np.zeros((2, 3))
```

Out[3]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

Матрица с единицами

In [4]:
```
np.ones((2, 3))
```

Out[4]:
```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

Массив с регулярно увеличивающимися значениями

In [5]:
```
np.arange(7)
```

Out[5]:
```
array([0, 1, 2, 3, 4, 5, 6])
```

In [6]:
```
arr = np.arange(2, 10, dtype = np.float)
print(arr, arr.dtype)
```

```
[2. 3. 4. 5. 6. 7. 8. 9.] float64

<ipython-input-6-a5835e5a3114>:1: DeprecationWarning: `np.float` is a depr
ecated alias for the builtin `float`. To silence this warning, use `float`
by itself. Doing this will not modify any behavior and is safe. If you spe
cifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.or
g/devdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdoc
s/release/1.20.0-notes.html#deprecations)
  arr = np.arange(2, 10, dtype = np.float)
```

Массив равномерно распределенных чисел от начального до конечного значений

In [8]:
```
np.linspace(1., 9., 5)
```

Out[8]:
```
array([1., 3., 5., 7., 9.])
```

Двумерный массив и функция транспонирования

In [13]:
```
a = np.matrix('1 2; 3 4')
print(a)
print('\nТранспонированная:\n')
print(a.T)
```

```
[[1 2]
 [3 4]]

Транспонированная:

[[1 3]
 [2 4]]
```

SciPy

Полное описание: https://coderlessons.com/tutorials/python-technologies/uchitsia-stsipi/scipy-kratkoe-rukovodstvo (https://coderlessons.com/tutorials/python-technologies/uchitsia-stsipi/scipy-kratkoe-rukovodstvo) или https://docs.scipy.org/doc/scipy/tutorial/index.html (https://docs.scipy.org/doc/scipy/tutorial/index.html)

KMeans кластеризация

Алгоритм k-средних принимает в качестве входных данных набор данных X, содержащий N точек, а также параметр K, задающий требуемое количество кластеров. На выходе получаем набор из K центроидов кластеров, кроме того, всем точкам множества X присваиваются метки, относящие их к определенному кластеру. Все точки в пределах данного кластера расположены ближе к своему центроиду, чем к любому другому центроиду.

In [16]:
```
from scipy.cluster.vq import kmeans,vq,whiten

from numpy import vstack,array
from numpy.random import rand

# data generation with three features
data = vstack((rand(100,3) + array([.2,.2,.2]),rand(100,3)))
data
```

Out[16]:
```
array([[0.83968555, 1.05814207, 0.53029056],
       [0.33120461, 0.91014495, 0.48790525],
       [0.95569004, 0.58260196, 0.99288384],
       [0.86185456, 0.21283568, 0.69822969],
       [0.54057025, 1.1867456 , 0.5109787 ],
       [0.90347081, 1.06186335, 0.45731062],
       [1.14770216, 0.67402981, 0.74885802],
       [0.94429674, 0.38298163, 0.69963151],
       [0.58105479, 0.54748007, 0.56797454],
       [0.93564778, 0.49313739, 0.58960095],
       [1.02316039, 0.32090271, 0.40770517],
       [0.3532336 , 1.052895  , 0.88309134],
       [0.72284542, 1.11162755, 0.70737789],
       [1.05374206, 0.50928374, 0.48016941],
       [0.54991277, 0.49021971, 0.72865534],
       [0.38789691, 0.87759911, 0.79216375],
       [0.20865108, 0.27725194, 0.50192597],
       [0.85437617, 0.63810378, 0.97893803],
       [0.53895965, 0.82602089, 0.55569011],
```

"Отбеливаем" данные - нормализуем их и масштабируем

In [17]:
```
data = whiten(data)
```

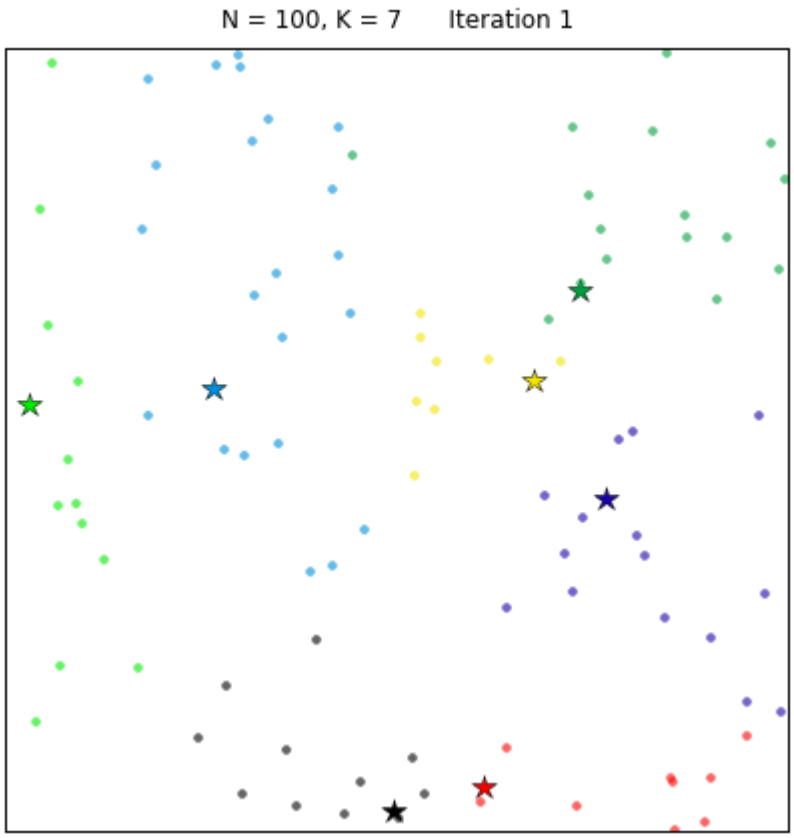
Вычислим средние значения для 3 кластеров:

In [19]:
```
centroids,_ = kmeans(data,3)
print(centroids)
```

```
[[2.82387221 2.4106987  2.62716369]
 [0.90728849 2.41372322 1.9270272 ]
 [1.88750176 1.0544804  1.34781327]]
```

In [1]:
```
from IPython import display
display.Image("part/k-means.gif")
```

Out[1]:



vq сравнивает каждый вектор наблюдений данных из аргумента 1 с центроидами в аргументе 2

In [21]:
```
clx,_ = vq(data,centroids)
clx
```

Out[21]:
```
array([0, 1, 0, 2, 1, 0, 0, 0, 2, 0, 2, 1, 0, 0, 2, 1, 2, 0, 1, 0, 0, 0,
       0, 2, 0, 1, 0, 1, 1, 1, 1, 0, 0, 2, 0, 2, 0, 2, 1, 1, 1, 1, 0, 1,
       0, 2, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 2, 2, 0, 0, 0, 1, 2, 0, 2, 0,
       0, 2, 0, 2, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 2, 1, 0, 2, 0,
       1, 0, 0, 2, 1, 1, 0, 1, 0, 2, 1, 2, 2, 2, 2, 1, 2, 2, 0, 2, 1, 1,
       2, 1, 2, 0, 1, 2, 2, 2, 1, 2, 2, 2, 0, 1, 2, 1, 2, 0, 2, 2, 1, 1,
       1, 2, 1, 1, 2, 1, 1, 2, 2, 1, 1, 1, 2, 0, 2, 0, 1, 2, 2, 2, 2, 2,
       2, 1, 1, 1, 2, 2, 2, 1, 2, 1, 0, 1, 0, 1, 2, 2, 2, 1, 2, 2, 2, 0,
       0, 1, 0, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 2, 2, 2, 0,
       2, 1])
```

Из Scipy можно вызвать различные константы

In [29]:
```
import scipy
print(dir(scipy.constants))
```

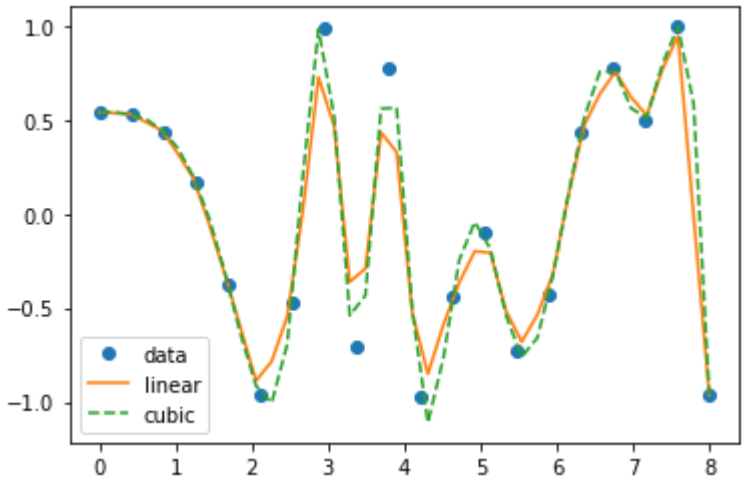
```
['Avogadro', 'Boltzmann', 'Btu', 'Btu_IT', 'Btu_th', 'ConstantWarning',
'G', 'Julian_year', 'N_A', 'Planck', 'R', 'Rydberg', 'Stefan_Boltzmann',
'Wien', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '_
_loader__', '__name__', '__package__', '__path__', '__spec__', '_obsolete_
constants', 'acre', 'alpha', 'angstrom', 'arcmin', 'arcminute', 'arcsec',
'arcsecond', 'astronomical_unit', 'atm', 'atmosphere', 'atomic_mass', 'att
o', 'au', 'bar', 'barrel', 'bbl', 'blob', 'c', 'calorie', 'calorie_IT', 'c
alorie_th', 'carat', 'centi', 'codata', 'constants', 'convert_temperatur
e', 'day', 'deci', 'degree', 'degree_Fahrenheit', 'deka', 'dyn', 'dyne',
'e', 'eV', 'electron_mass', 'electron_volt', 'elementary_charge', 'epsilon
_0', 'erg', 'exa', 'exbi', 'femto', 'fermi', 'find', 'fine_structure', 'fl
uid_ounce', 'fluid_ounce_US', 'fluid_ounce_imp', 'foot', 'g', 'gallon', 'g
allon_US', 'gallon_imp', 'gas_constant', 'gibi', 'giga', 'golden', 'golden
_ratio', 'grain', 'gram', 'gravitational_constant', 'h', 'hbar', 'hectar
e', 'hecto', 'horsepower', 'hour', 'hp', 'inch', 'k', 'kgf', 'kibi', 'kil
o', 'kilogram_force', 'kmh', 'knot', 'lambda2nu', 'lb', 'lbf', 'light_yea
r', 'liter', 'litre', 'long_ton', 'm_e', 'm_n', 'm_p', 'm_u', 'mach', 'meb
i', 'mega', 'metric_ton', 'micro', 'micron', 'mil', 'mile', 'milli', 'minu
te', 'mmHg', 'mph', 'mu_0', 'nano', 'nautical_mile', 'neutron_mass', 'nu2l
ambda', 'ounce', 'oz', 'parsec', 'pebi', 'peta', 'physical_constants', 'p
i', 'pico', 'point', 'pound', 'pound_force', 'precision', 'proton_mass',
'psi', 'pt', 'short_ton', 'sigma', 'slinch', 'slug', 'speed_of_light', 'sp
eed_of_sound', 'stone', 'survey_foot', 'survey_mile', 'tebi', 'tera', 'tes
t', 'ton_TNT', 'torr', 'troy_ounce', 'troy_pound', 'u', 'unit', 'value',
'week', 'yard', 'year', 'yobi', 'yotta', 'zebi', 'zepto', 'zero_Celsius',
'zetta']
```

SciPy — FFTpack реализует преобразование Фурье. Вычисляется на сигнале временной области, чтобы проверить его поведение в частотной области.

Scipy умеет в преобразования - преобразование Фурье (fft), дискретное косинусное преобразование (dct) Scipy умеет в интегралы - integrate

Scipy умеет в интерполяцию (процесс нахождения значения между двумя точками на линии или кривой).

In [2]:
```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt
x = np.linspace(0, 8, 20)
y = np.cos(x**3/5+1)

f1 = interpolate.interp1d(x, y,kind = 'linear')
f2 = interpolate.interp1d(x, y, kind = 'cubic')

xnew = np.linspace(0, 8, 40)

plt.plot(x, y, 'o', xnew, f1(xnew), '-', xnew, f2(xnew), '--')

plt.legend(['data', 'linear', 'cubic','nearest'], loc = 'best')

plt.show()
```



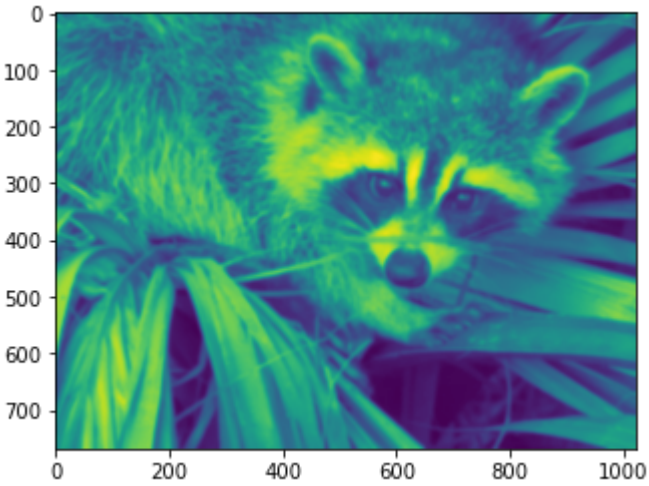
Scipy используют для обработки изображений - удаление шумов, фильтрация, кадрирование, классификация...

In [3]:
```
from scipy import misc
f = misc.face()
try:
    misc.imsave('face.png', f) # uses the Image module (PIL)
except AttributeError:
    !conda install pillow

import matplotlib.pyplot as plt
plt.imshow(f)
plt.show()
```

```
^C
```

Размываем изображение (убираем шумы)

In [40]:
```
from scipy import misc
from  scipy import ndimage
face = misc.face(gray = True)
blurred_face = ndimage.gaussian_filter(face, sigma=3)
import matplotlib.pyplot as plt
plt.imshow(blurred_face)
plt.show()
```



Scipy умеет в оптимизацию

Решить нелинейную задачу наименьших квадратов с оценками переменных. Учитывая невязки f (x) (m-мерная вещественная функция от n вещественных переменных) и функцию потерь rho (s) (скалярная функция), наименьшие квадраты находят локальный минимум функции стоимости F (x). Давайте рассмотрим следующий пример.

В этом примере мы находим минимум функции Розенброка без границ для независимых переменных.

In [42]:
```
#Rosenbrock Function
def fun_rosenbrock(x):
   return np.array([10 * (x[1] - x[0]**2), (1 - x[0])])

from scipy.optimize import least_squares
input = np.array([2, 2])
res = least_squares(fun_rosenbrock, input)

print(res)
```

```
 active_mask: array([0., 0.])
       cost: 9.866924291084687e-30
        fun: array([4.44089210e-15, 1.11022302e-16])
       grad: array([-8.89288649e-14,  4.44089210e-14])
        jac: array([[-20.00000015,  10.        ],
       [ -1.        ,   0.        ]])
    message: '`gtol` termination condition is satisfied.'
       nfev: 3
       njev: 3
  optimality: 8.892886493421953e-14
     status: 1
    success: True
          x: array([1., 1.])
```

Обратите внимание, что мы предоставляем только вектор невязок. Алгоритм строит функцию стоимости как сумму квадратов невязок, которая дает функцию Розенброка (https://ru.wikipedia.org/wiki/Функция_Розенброка (https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8 Точный минимум находится при x = [1,0,1,0].

Нахождение корней системы линейных уравнений

In [43]:
```
import numpy as np
from scipy.optimize import root
def func(x):
   return x*2 + 2 * np.cos(x)
sol = root(func, 0.3)
print(sol)
```

```
    fjac: array([[-1.]])
     fun: array([0.])
 message: 'The solution converged.'
    nfev: 10
     qtf: array([-2.77666778e-12])
       r: array([-3.3472241])
  status: 1
 success: True
       x: array([-0.73908513])
```

Scipy умеет в статистику:

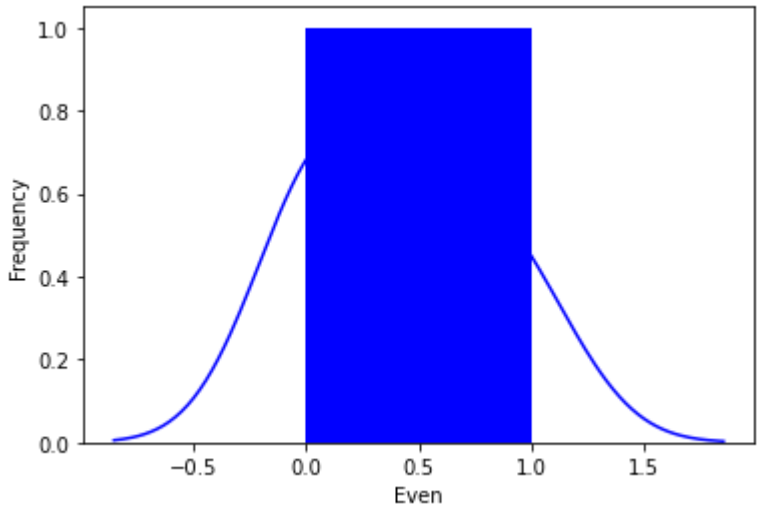
In [45]:
```
# Нормальная непрерывная случайная величина
from scipy.stats import norm
import numpy as np
print(norm.cdf(np.array([1,-1., 0, 1, 3, 4, -2, 6])))
# медиана распределения
print(norm.ppf(0.5))
# последовательность случайных переменных
print(norm.rvs(size = 5))
```

```
[0.84134475 0.15865525 0.5        0.84134475 0.9986501  0.99996833
 0.02275013 1.        ]
0.0
[-2.1641282   1.85044868 -0.52765039 -0.27209008  0.15796567]
```

In [58]:
```
# Равномерное распределение
from scipy.stats import uniform
ravn_data = uniform.cdf([0, 1, 2, 3, 4, 5], loc = 1, scale = 4)
# loc - среднее, scale - стандартное отклонение
ax = sb.distplot(ravn_data,
                 kde=True,
                 color='blue',
                 hist_kws={"linewidth": 25,'alpha':1})
ax.set(xlabel='Even', ylabel='Frequency')
```

```
D:\Users\glebk\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

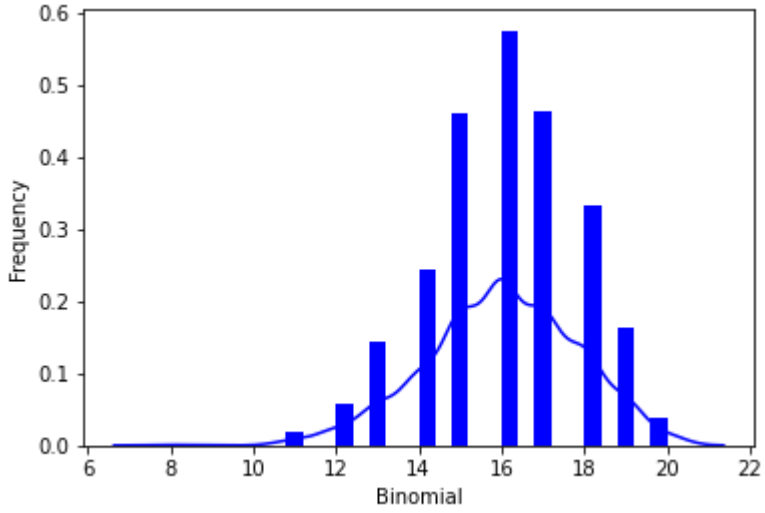
Out[58]:
```
[Text(0.5, 0, 'Even'), Text(0, 0.5, 'Frequency')]
```



In [48]:
```
# Биномиальное распределение
from scipy.stats import binom
import seaborn as sb

binom.rvs(size=10,n=20,p=0.8)

data_binom = binom.rvs(n=20,p=0.8,loc=0,size=1000)
ax = sb.distplot(data_binom,
                 kde=True,
                 color='blue',
                 hist_kws={"linewidth": 25,'alpha':1})
ax.set(xlabel='Binomial', ylabel='Frequency')
```

```
D:\Users\glebk\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[48]:
```
[Text(0.5, 0, 'Binomial'), Text(0, 0.5, 'Frequency')]
```



Библиотека sklearn

Почитать тут: https://neerc.ifmo.ru/wiki/index.php?title=Обзор_библиотек_для_машинного_обучения_на_Python (https://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D0%B1%D0%B7%D0%BE%D1%80_%D0%B1%D0%B8%D0%B1%D0%BB

In [24]:
```
# Add required imports
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

In [26]:
```
diabetes = datasets.load_diabetes()
# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]
```

Разбиваем датасет на тренировочный и тестовый

In [30]:
```
# Split the data into training/testing sets
x_train = diabetes_X[:-20]
x_test = diabetes_X[-20:]

# Split the targets into training/testing sets
y_train = diabetes.target[:-20]
y_test = diabetes.target[-20:]
```

Модель Линейной регрессии

In [31]:
```
lr = LinearRegression()
lr.fit(x_train, y_train)
predictions = lr.predict(x_test)
```

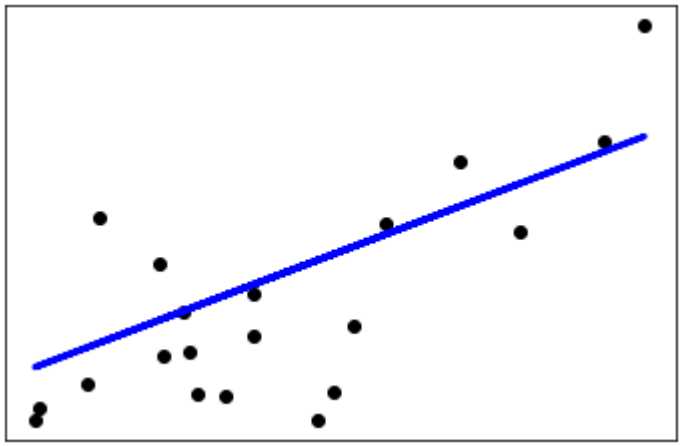
Оцениваем модель

In [32]:
```
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(y_test, predictions))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, predictions))
```

```
Mean squared error: 2548.07
Variance score: 0.47
```

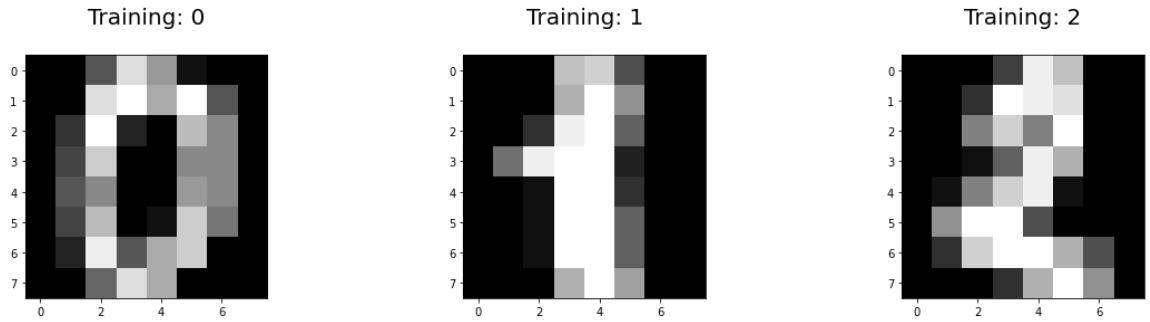
Отображаем графически

In [33]:
```
plt.scatter(x_test, y_test,  color='black')
plt.plot(x_test, predictions, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```



Модель логистической регрессии

In [34]:
```
from sklearn.datasets import load_digits
digits = load_digits()

import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(digits.data[0:3], digits.target
  plt.subplot(1, 3, index + 1)
  plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)
  plt.title('Training: %i\n' % label, fontsize = 20)

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.targ
```



Строим саму модель и обучаем её

In [35]:
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(solver='lbfgs', max_iter=10000)
# Подробнее об оптимизаторе https://en.wikipedia.org/wiki/Limited-memory_BFG
lr.fit(x_train, y_train)
predictions = lr.predict(x_test)
```

In [36]:
```
score = lr.score(x_test, y_test)
print("Score: %.3f" % score)
```

```
Score: 0.956
```

Обучение Перцептрона https://ru.wikipedia.org/wiki/Перцептрон (https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%86%D0%B5%D0%BF%D1%8

In [37]:
```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

In [38]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

Трансформируем признаки

In [39]:
```
from sklearn.preprocessing import StandardScaler
# Стандартизирует функции, удалив среднее значение и масштабируя до единично
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Строим и обучаем модель

In [40]:
```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
mlp.fit(X_train, y_train.ravel())
predictions = mlp.predict(X_test)
```

Оцениваем модель

In [41]:
```
from sklearn.metrics import classification_report, confusion_matrix
# confusion_matrix - вычисляет матрицу неточностей для определения точности
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[11  0  0]
 [ 0  9  1]
 [ 0  1  8]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       0.90      0.90      0.90        10
           2       0.89      0.89      0.89         9

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

Распознавание текста

In [42]:
```
from sklearn.datasets import fetch_20newsgroups
twenty_train = fetch_20newsgroups(subset='train', shuffle=True, random_state
```

In [43]:
```
print("\n".join(twenty_train.data[0].split("\n")[:3]))
print(twenty_train.target_names[twenty_train.target[0]])
```

```
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
rec.autos
```

Построение и обучение двух моделей. Первая на основе Байесовской классификации, а вторая использует метод опорных векторов:

In [44]:
```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransforme

from sklearn.naive_bayes import MultinomialNB
text_clf1 = Pipeline([
('vect', CountVectorizer()),
('tfidf', TfidfTransformer()),
('clf', MultinomialNB()),
])

from sklearn.linear_model import SGDClassifier
text_clf2 = Pipeline([
('vect', CountVectorizer()),
('tfidf', TfidfTransformer()),
('clf', SGDClassifier(loss='hinge', penalty='l2',
                      alpha=1e-3, random_state=42,
                      max_iter=5, tol=None)),
])

text_clf1.fit(twenty_train.data, twenty_train.target)
text_clf2.fit(twenty_train.data, twenty_train.target)
```

Out[44]:
```
Pipeline(steps=[('vect', CountVectorizer()), ('tfidf', TfidfTransformer
()),
                ('clf',
                 SGDClassifier(alpha=0.001, max_iter=5, random_state=42,
                               tol=None))])
```

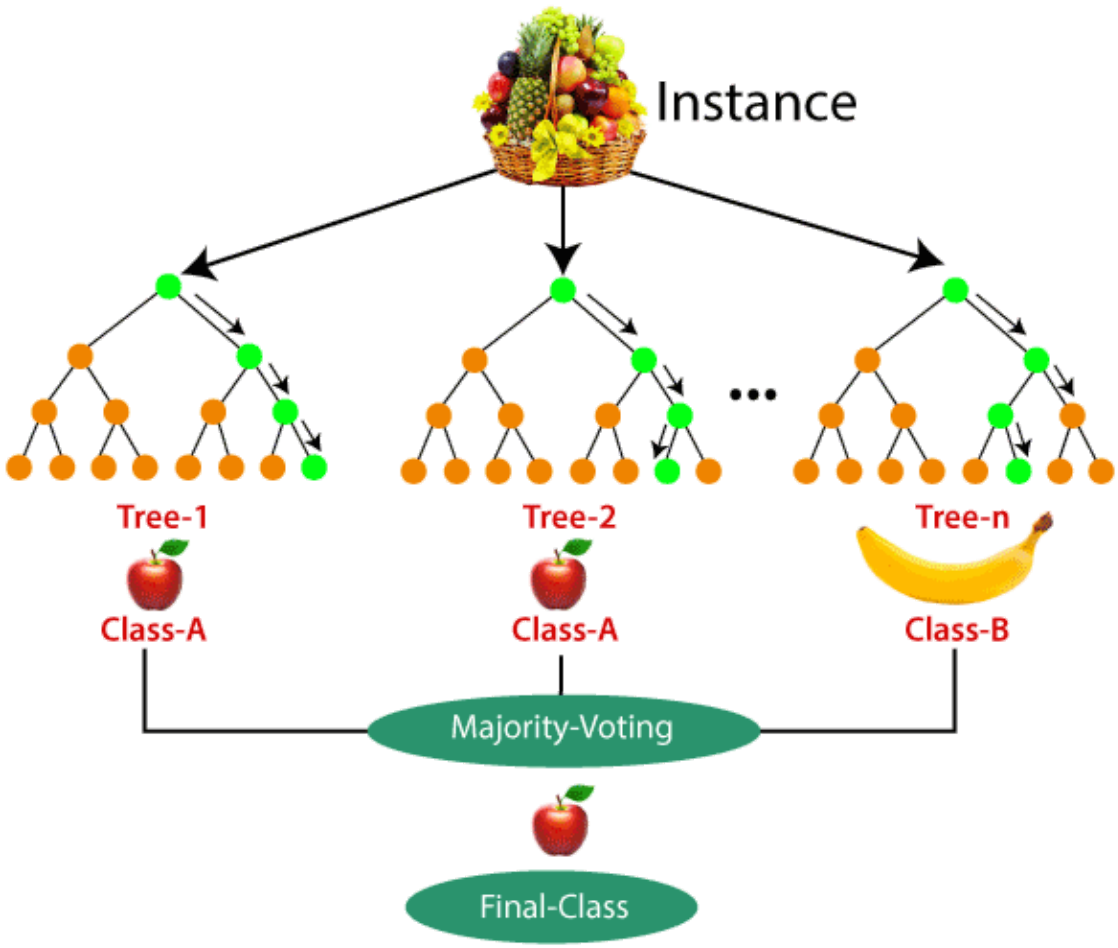
Оцениваем алгоритмы

In [45]:
```
twenty_test = fetch_20newsgroups(subset='test', shuffle=True, random_state=
docs_test = twenty_test.data
predicted1 = text_clf1.predict(docs_test)
predicted2 = text_clf2.predict(docs_test)
print("Score: %.3f" % np.mean(predicted1 == twenty_test.target))
print("Score: %.3f" % np.mean(predicted2 == twenty_test.target))
```

```
Score: 0.774
Score: 0.825
```

In [46]:
```
from IPython import display
display.Image("part/example-of-random-forest-classifier.png")
```

Out[46]:



In [47]:
```
from sklearn.datasets import load_boston
from sklearn.ensemble import RandomForestRegressor

x, y = load_boston(return_X_y=True)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, ra

regressor = RandomForestRegressor(n_estimators=10, random_state=0)
regressor.fit(x_train, y_train)
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=0, verbose=
                      warm_start=False)

regressor.score(x_train, y_train)
```

Out[47]:
```
0.9692280622549937
```

In [48]:
```
regressor.score(x_test, y_test)
```

Out[48]:
```
0.8244855276351775
```

In [ ]: