МЕТОДЫ И ТЕХНОЛОГИИ МАШИННОГО ОБУЧЕНИЯ

Занятие 3. Среда Jupyter Notebook.

Введение в Scikit-Learn.

Анализ главных компонент.

Алексейчук Андрей Сергеевич,
ст. преп. каф. 805

2020 г.

Jupyter Notebook

Jupyter Notebook – интерактивная оболочка для интерпретируемых языков программирования – **Python**, **R**, **Ruby**,позволяющая объединить код, текст, диаграммы и распространять их для других пользователей. Язык Python и среда Jupyter Notebook широко используются в сфере Data Science.

Установка Jupyter Notebook

- Самый простой способ установки – с помощью пакета Anaconda Distribution. Доступны версии для Windows, Linux, macOS. https://www.anaconda.com/distribution/

- Можно запустить готовый образ Debian с установленным Jupyter при помощи Docker.

Запуск Jupyter Notebook

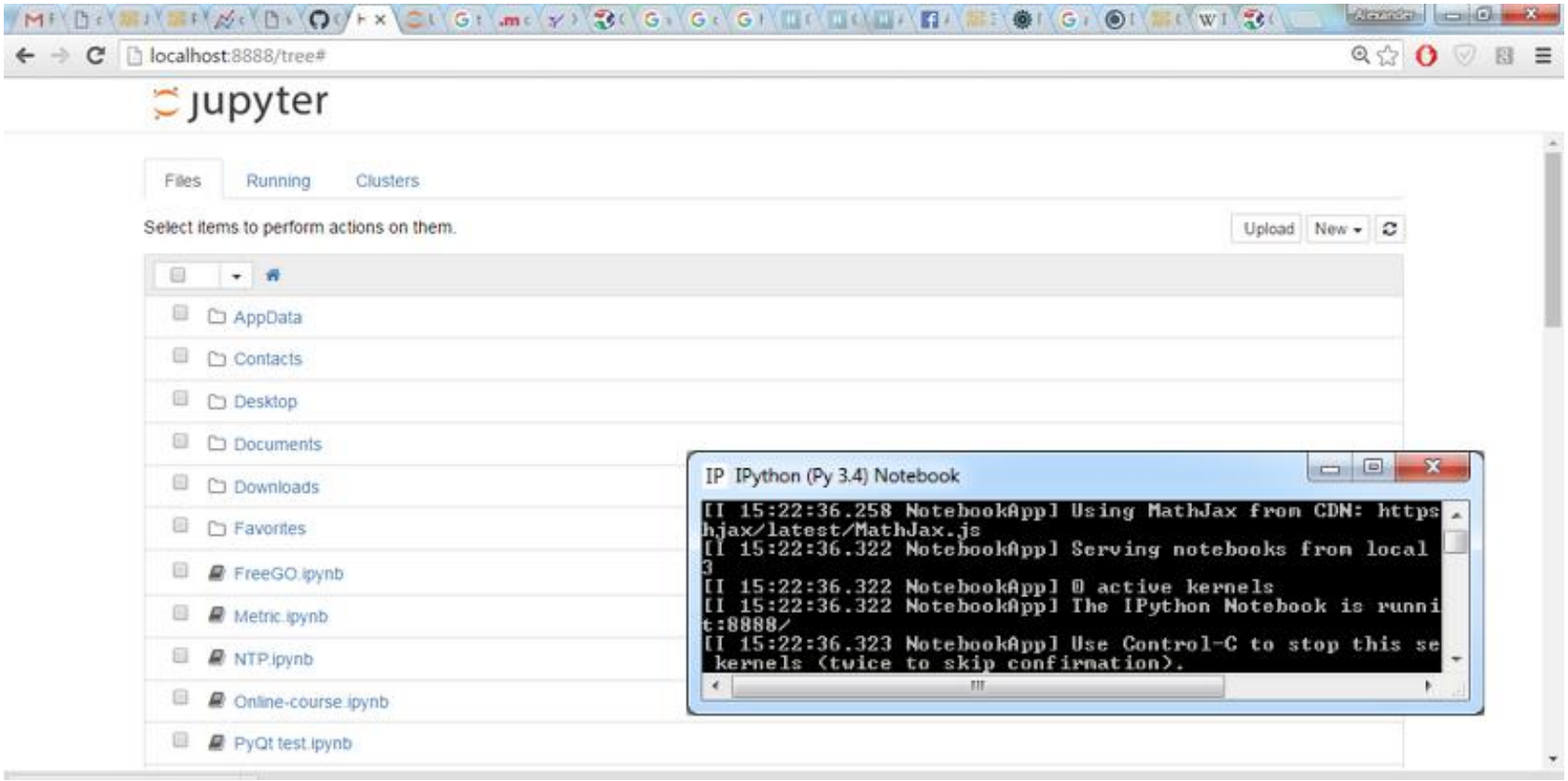
- На Linux: запуск командой

  ```
  jupyter notebook
  ```

- Команда откроет браузер на локальном хосте, по умолчанию http://127.0.0.1:8888.
- На Windows: через главное меню, пункт Программы > Anaconda > Jupyter Notebook.

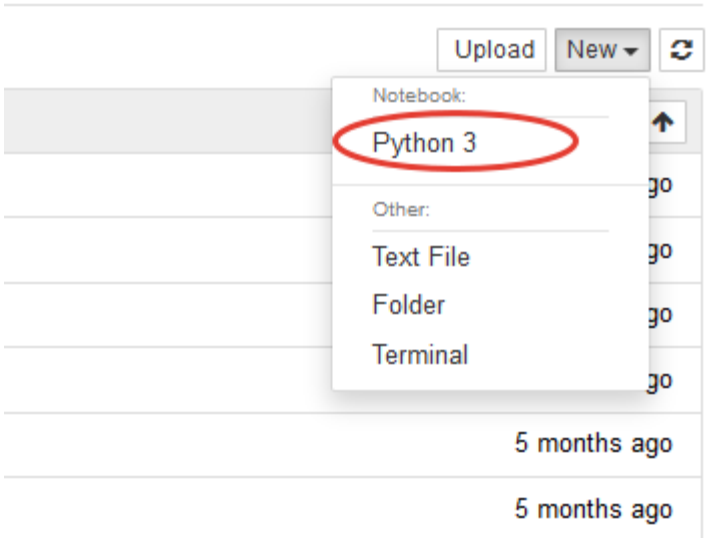
Запуск Jupyter Notebook

- В окне терминала отображается служебная информация. После запуска в окне браузера появляется панель со списком файлов. Преимущество файлов Jupyter (.ipynb) заключается в том, что они выглядят одинаково при написании кода и публикации. Есть все возможности для перемещения кода, запуска кода в ячейках, просмотра результатов.



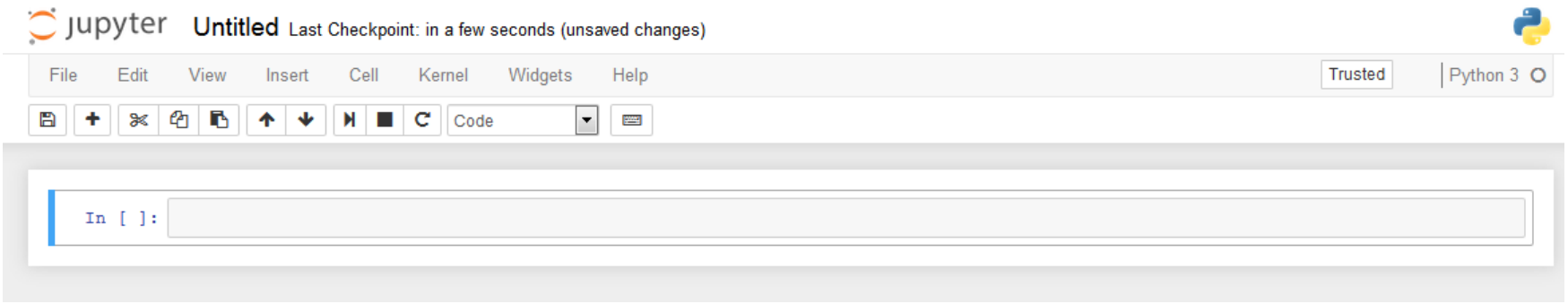
Запуск Jupyter Notebook

- Для создания нового ноутбука надо выбрать в меню «New» пункт «Python 3».



Запуск Jupyter Notebook

- В браузере открывается новая вкладка с ячейкой для ввода кода.



Запуск Jupyter Notebook

- В ячейках можно писать как код на языке Python, так и текстовое содержимое. Доступно форматирование и отображение формул, картинок и т.д.

Библиотеки ML на языке Python

- **NumPy** – библиотека для выполнения операций линейной алгебры и численных преобразований
- **Pandas** – библиотека для извлечения и подготовки данных для алгоритмов ML
- **Scikit-learn** – библиотека классических алгоритмов ML: регрессия, кластеризация, классификация, построение деревьев решений
- **Matplotlib** – библиотека для визуализации данных
- **TensorFlow, Theano, Keras, PyTorch** – библиотеки алгоритмов построения нейронных сетей глубокого обучения

Пакет NumPy

- **NumPy** - это пакет для научных вычислений на Python. Это библиотека Python, которая предоставляет объект многомерного массива *ndarray* и набор подпрограмм для быстрых операций над массивами, включая математические, логические, манипуляции с формами, сортировку, выбор, дискретные преобразования Фурье, операции линейной алгебры, основные статистические операции, случайное моделирование и многое другое.

Часто используемые функции NumPy

- **Создание массивов**

  arange, array, copy, empty, empty_like, eye, fromfile, fromfunction, identity, linspace, logspace, mgrid, ogrid, ones, ones_like, r, zeros, zeros_like

- **Манипуляции с массивами**

  array_split, column_stack, concatenate, diagonal, dsplit, dstack, hsplit, hstack, ndarray.item, newaxis, ravel, repeat, reshape, resize, squeeze, swapaxes, take, transpose, vsplit, vstack

- **Генераторы случайных чисел**

  random.rand, random.randn

- **Сортировка**

  argmax, argmin, argsort, max, min, ptp, searchsorted, sort

- **Агрегатные функции**

  choose, compress, cumprod, cumsum, inner, ndarray.fill, imag, prod, put, putmask, real, sum

- **Статистика**

  cov, mean, std, var

- **Операции линейной алгебры**

  cross, dot, outer, linalg.svd, vdot, @

Полный список:

https://docs.scipy.org/doc/numpy/reference/index.html

https://docs.scipy.org/doc/numpy/reference/routines.html

Примеры работы с NumPy

In [1]:
```
import numpy as np
```

In [11]:
```
# Создание массива NumPy из обычного массива
a = np.array([1,2,3])
a
```

Out[11]:
```
array([1, 2, 3])
```

In [12]:
```
# Создание многомерного массива NumPy, тип данных - float
b = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')
b
```

Out[12]:
```
array([[1., 2., 4.],
       [5., 8., 7.]])
```

In [16]:
```
# Создание двумерного массива из нулей
np.zeros((3, 4))
```

Out[16]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [17]:
```
# Создание двумерного массива из случайных чисел с распределением ~R(0,1)
np.random.random((2, 2))
```

Out[17]:
```
array([[0.53515076, 0.53905543],
       [0.55807025, 0.20695782]])
```

In [19]:
```
# Создание массива NumPy в заданных границах и с заданным шагом
np.arange(0, 20, 3)
```

Out[19]:
```
array([ 0,  3,  6,  9, 12, 15, 18])
```

Примеры работы с NumPy

In [20]:
```
# Создание массива NumPy в заданных границах с заданным количеством элементов
np.linspace(0, 5, 10)
```

Out[20]:
```
array([0.        , 0.55555556, 1.11111111, 1.66666667, 2.22222222,
       2.77777778, 3.33333333, 3.88888889, 4.44444444, 5.        ])
```

In [23]:
```
# Матрица случайных величин ~N(0,1)
np.random.randn(2, 2)
```

Out[23]:
```
array([[ 0.98833887,  1.23435556],
       [-1.50398899,  1.10516292]])
```

In [22]:
```
# То же, аргумент - кортеж
np.random.standard_normal((2, 2))
```

Out[22]:
```
array([[0.53587506, 0.24660406],
       [0.97454901, 0.95685792]])
```

In [25]:
```
# Матрица 4x2 случайных величин ~N(1/2, 3)
0.5 * np.random.randn(2, 4) + 3
```

Out[25]:
```
array([[3.77525847, 3.39517863, 2.61948809, 2.5058802 ],
       [2.51170106, 3.00217656, 1.71279075, 3.76501393]])
```

In [37]:
```
# Умножение матриц
a = np.array([[1, 2, 4], [5, 8, 7]])
b = np.array([[2, 1], [0, 3], [1, 2]])
print('a @ b=\n', a @ b)
```

```
a @ b=
 [[ 6 15]
 [17 43]]
```

Пакет Pandas

- **Pandas** - это пакет Python, предоставляющий быстрые, гибкие и выразительные структуры данных, предназначенные для того, чтобы сделать работу с «реляционными» или «помеченными» данными простой и интуитивно понятной.
- В отличие от NumPy, предназначенной для обработки массивов однородных данных (прежде всего числовых), Pandas предназначен для хранения и обработки произвольных разнородных данных в таблицах, похожих по структуре на таблицы реляционных БД.

Пакет Pandas

- В библиотеке имеются три встроенных типа данных: **Series**, **DataFrame** и **Panel** (для одномерных, двумерных и трехмерных данных соответственно). Наиболее часто используется **DataFrame**, представляющий собой двумерную таблицу.

Пакет Pandas

- Создание DataFrame из массива NumPy:

```
In [8]: import pandas as pd
        import numpy as np

In [10]: df = pd.DataFrame(np.random.randn(6, 4), columns=list('ABCD'))
         df
```

Out[10]:

	A	B	C	D
0	-0.095030	0.610688	1.524687	1.292590
1	-0.950675	-0.376972	1.498795	-0.005557
2	0.494650	1.380725	0.556052	-1.468297
3	0.392564	1.385386	-0.279894	1.762779
4	-0.600091	0.445455	-1.119313	0.240880
5	-0.743768	2.031123	1.763610	1.210546

Пакет Pandas

- Создание DataFrame из обычного словаря Python:

In [15]:
```
df2 = pd.DataFrame({'A': 1.,
                    'B': pd.date_range('20130101', periods=4),
                    'C': pd.Series([x for x in list(range(4))], dtype='float32'),
                    'D': np.random.randn(4),
                    'E': pd.Categorical(["test", "train", "test", "train"]),
                    'F': 'foo'})
df2
```

Out[15]:

	A	B	C	D	E	F
0	1.0	2013-01-01	0.0	0.863346	test	foo
1	1.0	2013-01-02	1.0	-1.686990	train	foo
2	1.0	2013-01-03	2.0	-0.615800	test	foo
3	1.0	2013-01-04	3.0	0.870753	train	foo

In [16]:
```
# Столбцы получившейся структуры содержат различные типы данных:
df2.dtypes
```

Out[16]:
```
A           float64
B    datetime64[ns]
C           float32
D           float64
E          category
F            object
dtype: object
```

Пакет Pandas

- Создание DataFrame из файла CSV:

```
In [2]: data = pd.read_csv("iris.csv")
        # Вывести первые 5 строк
        data.head()
```

Out[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

- Строки DataFrame идентифицируются индексами (обычно строки или числа), столбцы – именами.

Примеры работы с Pandas

- Вывод одного столбца DataFrame

```
In [17]: data = pd.read_csv("iris.csv")
         # Вывести содержимое столбца sepal_length
         data['sepal_length']

Out[17]: 0      5.1
         1      4.9
         2      4.7
         3      4.6
         4      5.0
         5      5.4
         6      4.6
         7      5.0
         8      4.4
         9      4.9
         10     5.4
```

```
In [18]: # Первое значение в столбце sepal_length
         data['sepal_length'][0]

Out[18]: 5.1
```

Примеры работы с Pandas

- Вывод нескольких столбцов DataFrame

```
In [25]: # Вывод столбцов sepal_length, sepal_width
         data[['sepal_length','sepal_width']]
```

Out[25]:

	sepal_length	sepal_width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6
5	5.4	3.9
6	4.6	3.4
7	5.0	3.4
8	4.4	2.9

Примеры работы с Pandas

- Выбор строк DataFrame
- Если индекс явно не задан, то можно использовать метод iloc[].

```
In [20]: first_row = data.iloc[0]
         first_row
```

```
Out[20]: sepal_length       5.1
         sepal_width        3.5
         petal_length       1.4
         petal_width        0.2
         species         setosa
         Name: 0, dtype: object
```

```
In [22]: first_three_rows = data.iloc[0:3]
         first_three_rows
```

Out[22]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa

Примеры работы с Pandas

- Выбор строк DataFrame по индексу

```
In [24]: # Выбор произвольных строк
         data.iloc[[1,2,3,5,8]]
```

Out[24]:

	sepal_length	sepal_width	petal_length	petal_width	species
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
8	4.4	2.9	1.4	0.2	setosa

Примеры работы с Pandas

- Выбор строк DataFrame по индексу
- Если индекс задан явно, то можно использовать метод loc[]. Метод iloc[] также доступен.

In [28]:
```
d = {'one' : pd.Series([1, 2, 3], index=['row_1', 'row_2', 'row_3']),
   'two' : pd.Series([1, 2, 3, 4], index=['row_1', 'row_2', 'row_3', 'row_4'])}

df = pd.DataFrame(d)
df
```

Out[28]:

	one	two
row_1	1.0	1
row_2	2.0	2
row_3	3.0	3
row_4	NaN	4

In [29]:
```
df.loc['row_2']
```

Out[29]:
```
one    2.0
two    2.0
Name: row_2, dtype: float64
```

In [32]:
```
df.iloc[1]
```

Out[32]:
```
one    2.0
two    2.0
Name: row_2, dtype: float64
```

Примеры работы с Pandas

- Выбор определенных строк и столбцов DataFrame

```
In [48]: # Выбор определенных строк и столбцов
         data.loc[[0,1,10,100],['sepal_length','petal_length']]
```

Out[48]:

	sepal_length	petal_length
0	5.1	1.4
1	4.9	1.4
10	5.4	1.5
100	6.3	6.0

Примеры работы с Pandas

- Выбор строк DataFrame по условию

```
In [33]: # Выбор строк по условию sepal_length=4.9
         data[data.sepal_length == 4.9]
```

Out[33]:

	sepal_length	sepal_width	petal_length	petal_width	species
1	4.9	3.0	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa
34	4.9	3.1	1.5	0.1	setosa
37	4.9	3.1	1.5	0.1	setosa
57	4.9	2.4	3.3	1.0	versicolor
106	4.9	2.5	4.5	1.7	virginica

Примеры работы с Pandas

- Выбор строк DataFrame по условию

```
In [34]: # Выбор строк по условию 4.7 <= sepal_length <= 4.9
         data[(data.sepal_length >= 4.7) & (data.sepal_length <= 4.9)]
```

Out[34]:

	sepal_length	sepal_width	petal_length	petal_width	species
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa
11	4.8	3.4	1.6	0.2	setosa
12	4.8	3.0	1.4	0.1	setosa
24	4.8	3.4	1.9	0.2	setosa
29	4.7	3.2	1.6	0.2	setosa
30	4.8	3.1	1.6	0.2	setosa
34	4.9	3.1	1.5	0.1	setosa
37	4.9	3.1	1.5	0.1	setosa
45	4.8	3.0	1.4	0.3	setosa
57	4.9	2.4	3.3	1.0	versicolor
106	4.9	2.5	4.5	1.7	virginica

Другие возможности Pandas

- Загрузка данных из файлов CSV, XLS, TXT и т.п., а также из баз данных
- Простая обработка отсутствующих данных (представленных как NaN)
- Добавление, удаление, перемещение столбцов
- Группировка данных
- Преобразование разнородных, по-разному проиндексированных данных в один DataFrame
- Конкатенация и объединение наборов данных
- Иерархическая маркировка осей (возможность иметь несколько меток)
- Функции обработки временных рядов: генерация диапазона дат, ресэмплинг и т.п.

Литература по Pandas

- http://pandas.pydata.org/pandas-docs/version/0.15.2/index.html
- https://www.tutorialspoint.com/python_pandas/python_pandas_quick_guide.htm
- https://nikgrozev.com/2015/12/27/pandas-in-jupyter-quickstart-and-useful-snippets/
- http://tomaugspurger.github.io/modern-1-intro

Пакет Scikit-learn

- Пакет Scikit-learn — самый распространенный выбор для решения задач классического машинного обучения на Python. Он предоставляет широкий выбор алгоритмов обучения с учителем (supervised learning) и без учителя (unsupervised learning). Библиотека тесно интегрирована с пакетами Pandas и NumPy.
- Официальный сайт: https://scikit-learn.org/

Пакет Scikit-learn

Библиотека Scikit-learn реализует следующие основные типы моделей машинного обучения:

- Линейные модели
- Метрические модели
- Деревья решений
- Ансамблевые методы
- Нейронные сети
- Машины опорных векторов

Также предоставляются методы понижения размерности и валидации моделей (кросс-валидация, ROC-кривые и т.д.)

Это — лишь базовый список. Помимо этого, Scikit-learn содержит большое число функций для расчета значений метрик, выбора моделей, препроцессинга данных и др.

Общая схема построения и обучения модели в Scikit-Learn

1. Загрузка данных: pd.read_from_csv(…)
2. Предобработка данных: Pandas.*, train_data=…, test_data=…
3. Создание модели: model = TheCoolClassifier(…)
4. Обучение модели: model.fit(train_data)
5. Тестирование модели: model.predict(test_data)
6. Оценка качества модели: confusion_matrix(…), classification_report(…)
7. Визуализация результатов: Matplotlib.plot(…)

Пакет Scikit-learn

В Scikit-learn приняты единые сигнатуры методов работы с моделями преобразования данных и методов работы с моделями ML.

Модели преобразования данных (понижение размерности и т.п.):

- `model.fit(X)` – обучение модели
- `model.transform(X)` – преобразование данных (обученной) моделью
- `model.fit_transform(X)` – обучение и преобразование одной командой

Модели ML (классификация, кластеризация):

- `model.fit(X, y)` – обучение с учителем
- `model.fit(X)` – обучение без учителя
- `model.predict(X_test)` – применение (обученной) модели к данным

Построение классификатора DecisionTreeClassifier для набора данных Iris

In [2]:
```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

In [3]:
```
# Считываем файл csv и загружаем его в объект DataFrame
data = pd.read_csv("iris.csv")
data.head()
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [4]:
```
# Выделяем признаки для обучения X и целевой признак y
X = data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = data['species']
```

In [5]:
```
# Разделяем выборку на обучающую и тестовую в соотношении 70% / 30%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)
```

In [6]:
```
# Инициализируем классификатор и обучаем его на полученной выборке
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train);
```

In [7]:
```
# Запускаем предсказание на тестовой выборке, чтобы сравнить
# полученные результаты с известными значениями целевого атрибута
prediction = classifier.predict(X_test)
```

Построение классификатора DecisionTreeClassifier для набора данных Iris

In [8]:
```
# Выведем бок-о-бок известные (ground truth) и предсказанные (prediction) значения
pd.DataFrame({'ground truth': y_test, 'prediction': prediction})
```

Out[8]:

	ground truth	prediction
14	setosa	setosa
98	versicolor	versicolor
75	versicolor	versicolor
16	setosa	setosa
131	virginica	virginica
56	versicolor	versicolor
141	virginica	virginica
44	setosa	setosa
29	setosa	setosa
120	virginica	virginica
94	versicolor	versicolor
5	setosa	setosa
102	virginica	virginica
51	versicolor	versicolor
78	versicolor	versicolor
42	setosa	setosa
92	versicolor	versicolor
66	versicolor	versicolor
31	setosa	setosa

Построение классификатора DecisionTreeClassifier для набора данных Iris

- Анализ качества модели:

In [9]:
```
# Матрица ошибок
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, prediction))
```

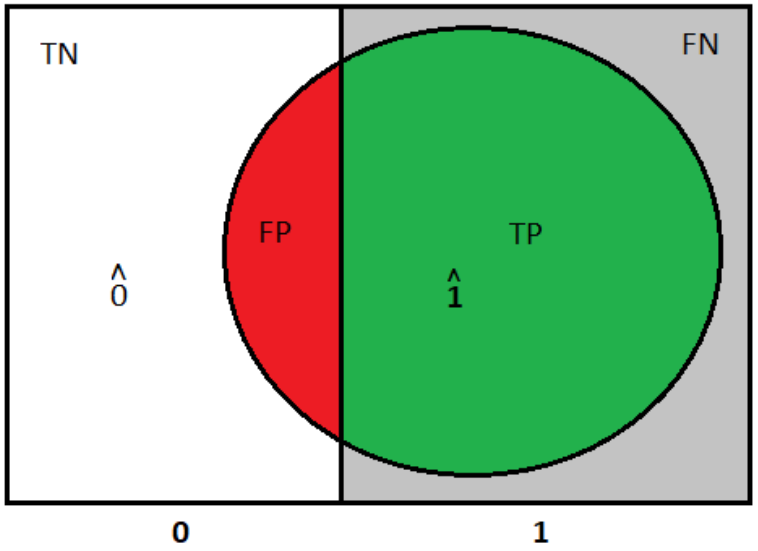
```
[[14  0  0]
 [ 0 17  1]
 [ 0  1 12]]
```

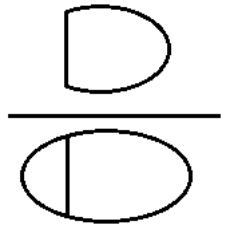
In [10]:
```
# Сводный отчет
print(classification_report(y_test, prediction))
```

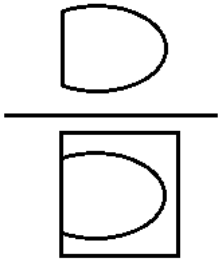
```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        14
  versicolor       0.94      0.94      0.94        18
   virginica       0.92      0.92      0.92        13

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```

Precision and recall

- Допустим, что классификатор предсказывает значение целевого атрибута label=$\hat{1}$ (область внутри круга) и label=$\hat{0}$ (область вне круга).

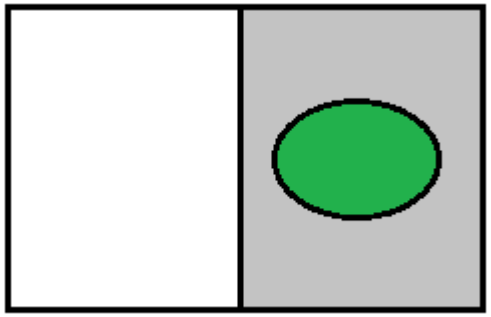


- $Precision = \frac{TP}{TP+FP} =$  (*точность*)

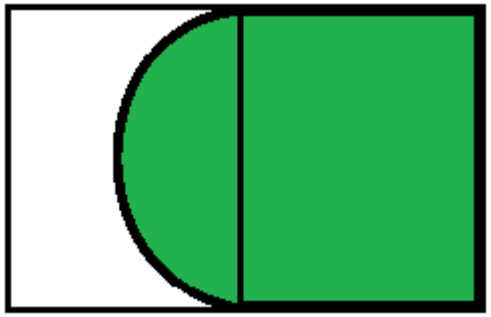
- $Recall = \frac{TP}{TP+FN} =$  (*полнота*)

Precision and recall

- Высокая точность, низкая полнота:



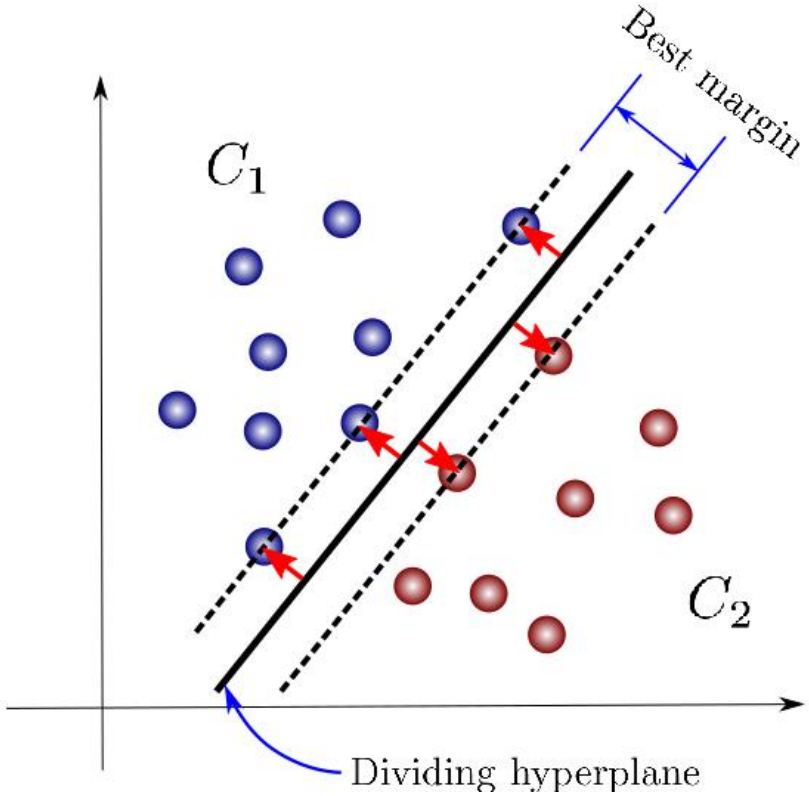
- Низкая точность, высокая полнота:



- На практике, чтобы учесть обе метрики, используют гармоническое среднее между чувствительностью и полнотой – F1-score.

SVC (метод опорных векторов) для Iris

- SVC (Support Vector Classification) – метод классификации, основанный на определении положения оптимальной разделяющей гиперплоскости, максимизирующей сумму расстояний от нее до ближайших к ней элементов выборки.



SVC (метод опорных векторов) для Iris

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import seaborn as sns
```

In [88]:
```
data = pd.read_csv('iris.csv')
```

In [6]:
```
from sklearn import svm
X = data[['sepal_length' , 'sepal_width' , 'petal_length' , 'petal_width']]
y = data['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)
```

In [14]:
```
clf = svm.SVC(gamma='auto')
clf.fit(X_train, y_train)
```

Out[14]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [15]:
```
prediction = clf.predict(X_test)
```

In [16]:
```
pd.DataFrame({'truth': y_test, 'prediction': prediction})
```

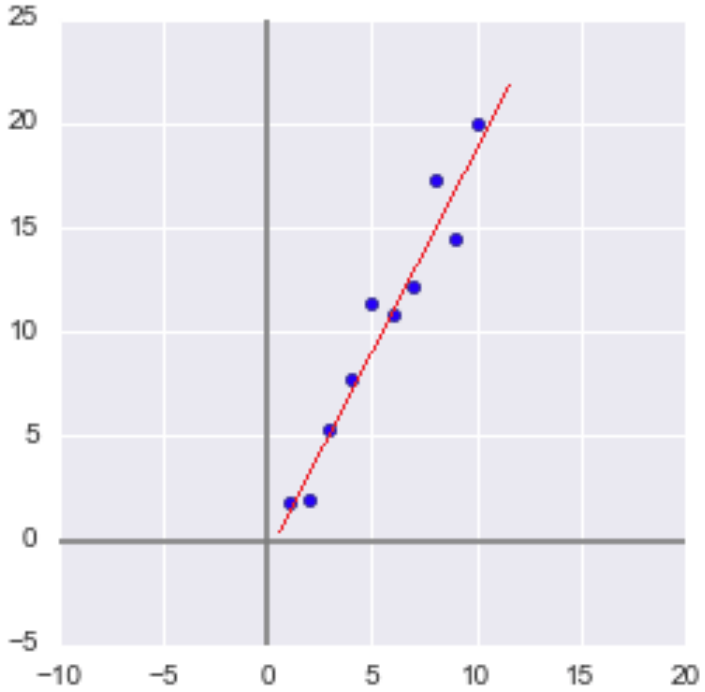
SVC (метод опорных векторов) для Iris

- Анализ метрик:

```
In [47]: from sklearn.metrics import classification_report, confusion_matrix

In [48]: print(confusion_matrix(y_test, prediction))

[[14  0  0]
 [ 0 17  1]
 [ 0  0 13]]

In [49]: print(classification_report(y_test, prediction))

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        14
  versicolor       1.00      0.94      0.97        18
   virginica       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

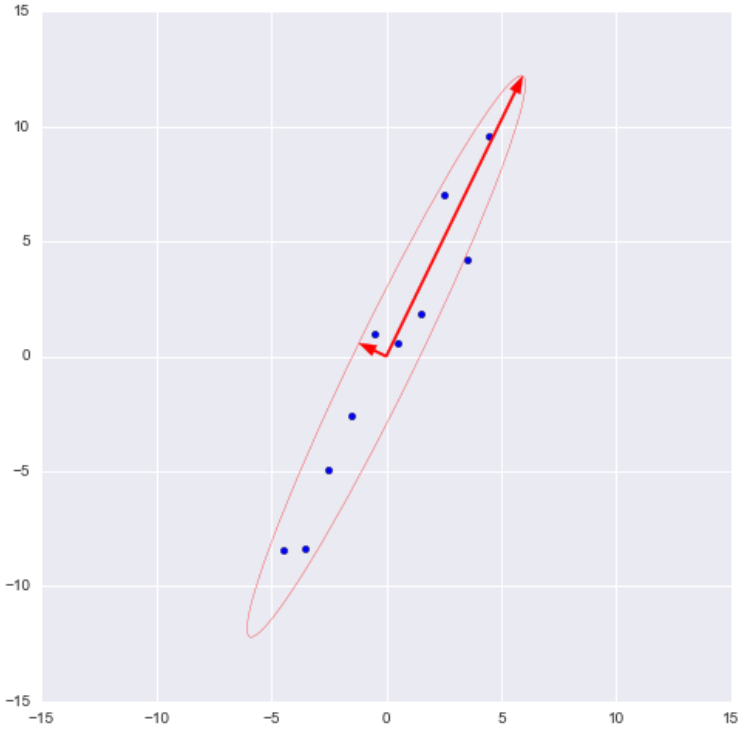
PCA (Principal Component Analysis)

- PCA – *метод главных компонент* – применяется для снижения размерности пространства признаков. Идея метода состоит в том, что зачастую исходные признаки сильно коррелируют между собой, и поэтому используемый алгоритм классификации занимается восстановлением тривиальных линейных зависимостей между признаками. Поэтому можно ввести новые оси координат, направленные вдоль направлений наибольшего разброса значений исходного набора данных, и спроецировать данные на новые оси.



PCA (Principal Component Analysis)

- Можно показать, что направления наибольшего разброса значений набора данных совпадают с направлениями собственных векторов ковариационной матрицы, которым соответствуют наибольшие по модулю собственные значения.
- Ковариационная матрица *K* - это матрица, у которой каждый элемент K_{ij} является корреляцией между соответствующими признаками X_i, X_j. Если собственные значения матрицы *K* сильно отличаются (на порядки), то имеет смысл использовать PCA.



PCA+SVC для Iris

- Строим матрицу корреляций признаков набора Iris. Видно, что многие элементы вне главной диагонали близки по модулю к 1, что говорит о сильной коррелированности соответствующих пар признаков.

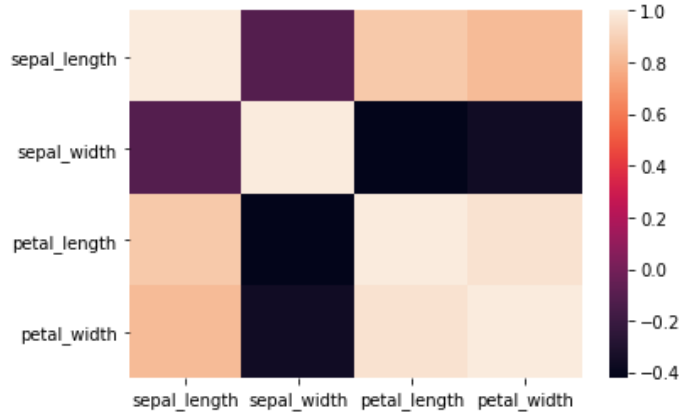
```
In [25]: from sklearn.decomposition import PCA
         from sklearn.covariance import empirical_covariance
         from numpy.linalg import eigvals
```

```
In [26]: X.corr()
```

Out[26]:

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.109369	0.871754	0.817954
sepal_width	-0.109369	1.000000	-0.420516	-0.356544
petal_length	0.871754	-0.420516	1.000000	0.962757
petal_width	0.817954	-0.356544	0.962757	1.000000

```
In [40]: sns.heatmap(X.corr());
```



PCA+SVC для Iris

- Вычисляем матрицу выборочных ковариаций и находим ее собственные значения. Видим, что первые два СЗ значительно больше остальных.
- Собственные значения матрицы ковариаций – это и есть дисперсии вдоль преобразованных осей. Поэтому мы оставляем только оси с наибольшей дисперсией, соответствующие первым двум СЗ.

```
In [28]: empirical_covariance(X)

Out[28]: array([[ 0.68112222, -0.03900667,  1.26519111,  0.51345778],
                [-0.03900667,  0.18675067, -0.319568  , -0.11719467],
                [ 1.26519111, -0.319568  ,  3.09242489,  1.28774489],
                [ 0.51345778, -0.11719467,  1.28774489,  0.57853156]])

In [29]: eigvals(empirical_covariance(X))

Out[29]: array([4.19667516, 0.24062861, 0.07800042, 0.02352514])
```

PCA+SVC для Iris

- Чтобы оценить объем потерянной информации, можно вычислить выборочную дисперсию по каждой из преобразованных осей. Атрибут explained_variance_ratio_ представляет долю дисперсии по каждой оси в процентах от суммы дисперсий по всем осям. В данном случае сохранено около 97,7% информации при сокращении количества признаков вдвое.

```
In [63]: pca.explained_variance_ratio_
Out[63]: array([0.92461621, 0.05301557])
```

- Недостатком PCA является то, что новые признаки могут не иметь физического смысла или какой-либо разумной интерпретации.

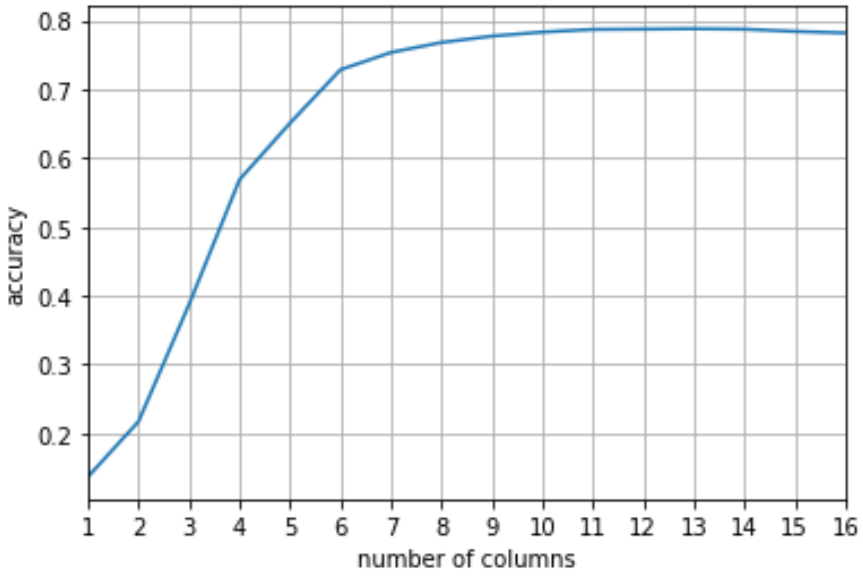
PCA+SVC для Iris

- Произведем обучение модели SVC на преобразованных признаках.
- Произошло небольшое (приемлемое) ухудшение точности.

```
In [46]: X_reduced = PCA(n_components=2).fit_transform(X)

In [47]: X_train, X_test, y_train, y_test = train_test_split(
                                X_reduced, y, test_size=0.3, random_state=3)

In [48]: clf = svm.SVC()
         clf.fit(X_train, y_train)

Out[48]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

In [49]: prediction = clf.predict(X_test)

In [50]: pd.DataFrame({'truth': y_test, 'prediction': prediction})

...

In [89]: print(confusion_matrix(y_test, prediction))

         [[17  0  0]
          [ 0 12  2]
          [ 0  0 14]]
```

PCA+SVC

- Выбор оптимальной размерности пространства производится исходя из баланса между размерностью пространства и точностью классификации. Обычно зависимость между размерностью и точностью имеет следующий вид:



- Оптимальная размерность начинается с момента «выхода на плато» графика.

Домашнее задание

- Выбрать произвольный набор данных. Реализовать понижение размерности исходных данных при помощи метода PCA и произвести классификацию при помощи SVC (или другого алгоритма, по желанию). Построить график зависимости точности (accuracy) от размерности преобразованного пространства. Выбрать оптимальную размерность преобразованного пространства.