

# Scikit-learn: структура библиотеки и принципы проектирования

---

Першин Антон Юрьевич, Ph.D.

Никольская Анастасия Николаевна

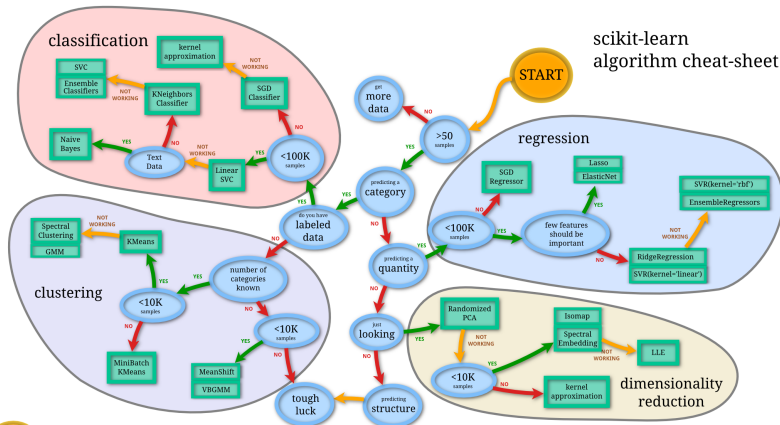
Программа «Большие данные и распределенная цифровая платформа»

Санкт-Петербургский государственный университет

Практика по дисциплине «Технологии ИИ»  
9 марта 2023 г.

# Обзор Scikit-learn

Scikit-learn является easy-to-use библиотекой для машинного обучения, реализующей множество полезных методов.



Scikit-learn обеспечивает следующую функциональность:

- Препроцессинг (обработка и преобразование признаков, их отбор)
- Понижение размерности (PCA, LLE, t-SNE etc.)
- Решение задач:
  - регрессии (линейные модели, деревья решений, SVM, GP etc.)
  - классификации (логистическая регрессия, деревья решений, SVM, etc.)
  - кластеризации (k-means, MeanShift, BIRCH, DBSCAN etc.)
- Создание мета-моделей (ансамбли: бэггинг, бустинг, стэкинг)
- Выбор моделей (кросс-валидация, поиск гиперпараметров)

Все объекты в `Scikit-learn` находятся в рамках общего API, который строится вокруг трех дополняющих друг друга интерфейсов:

- **Estimator** (строит и обучает модели)
- **Predictor** (выполняет предсказание/inference)
- **Transformer** (преобразует данные)

Общие принципы построения API `Scikit-learn`:

- **Consistency:** все объекты (простые и сложные) используют общий интерфейс, состоящий из ограниченного набора методов
- **Inspection:** все параметры, передаваемые на вход моделям и полученные во время обучения, сохраняются и доступны как открытые атрибуты класса
- **Non-proliferation of classes:** специализированные классы используются только для моделей, остальные объекты представлены стандартными типами/классами (например, датасетами могут быть `numpy` массивы, разреженные `scipy` матрицы или датафреймы `pandas`)
- **Composition:** везде, где это возможно, модели реализованы так, чтобы им можно было использовать как “строительные блоки” других композиционных моделей
- **Sensible defaults:** все параметры моделей должны иметь “разумные” значения по умолчанию, чтобы модель можно было использовать для получения baseline решения

Любая модель `Model`, имеющая параметры, обучаемые на входных данных  $X$  и выходных  $y$ , должна реализовать интерфейс `Estimator`:

1. Отнаследоваться от `sklearn.base.BaseEstimator` (под параметрами здесь понимаются гиперпараметры, передаваемые в конструктор)

```
1 class BaseEstimator:
2     def get_params(deep=True):
3         ...
4
5     def set_params(**params):
6         ...
```

2. Реализовать `fit(X, y, **kwargs)`

Пример использования:

```
1 X = np.random.randint(10, size=(1000, 2))
2 y = np.ones((1000, 1))
3 model = Model(hyperparameter=1e-3)
4 model.fit(X, y)
```

## Инициализация и обучение модели строго разделены:

- `__init__()` принимает только гиперпараметры модели как явно заданные ключевые слова
- `fit(X, y)` принимает данные для обучения и возвращает `self`

## Параметры модели:

- Гиперпараметры хранятся как атрибуты класса
- Обучаемые параметры хранятся как атрибуты с “\_” на конце

## Пример:

```
1 class SubtractMeanAndShiftEstimator(BaseEstimator):
2     def __init__(self, shift=0.):
3         self.shift: float = shift
4         self.means_: NDArray = None
5
6     def fit(self, X: NDArray, y: Optional[NDArray]):
7         self.means_ = X.mean(axis=0)
8         return self
```

Практически все объекты в `scikit-learn`, преобразующие данные, являются Estimators

# Интерфейс Predictor

Интерфейс Predictor расширяет интерфейс Estimator, добавляя к нему два метода:

- `predict(X_test)` для предсказания по входным данным `X_test`
- `score(X_test, y_test)` принимает данные для обучения

**Пример:**

```
1 class SubtractMeanAndShiftEstimator(BaseEstimator):
2     def __init__(self, shift=0.):
3         self.shift: float = shift
4         self.means_: NDArray = None
5
6     def predict(self, X: NDArray) -> NDArray:
7         e = np.ones((X.shape[0], 1))
8         return X - e @ self.means_.reshape(-1, 1).T + self.
          shift
9
10    def score(self, X: NDArray, y: NDArray) -> float:
11        return r2_score(y, self.predict(X))
```

Практически все объекты в `scikit-learn`, преобразующие данные, являются Estimators

Интерфейс Transformer расширяет интерфейс Estimator:

- Обязательно: добавить метод `transform(X)` для преобразования данных `X`
- Опционально для работы с pipelines:
  - добавить метод `get_feature_names_out(input_features=None)` для возвращения имен преобразованных признаков
  - установить атрибут `feature_names_in_` во время вызова `fit()`

Для создания Transformer полезными оказываются стандартные миксины:

- `sklearn.base.TransformerMixin`
- `sklearn.base.OneToOneFeatureMixin`
- `sklearn.base.ClassNamePrefixFeaturesOutMixin`



## FYI: миксины (mixins)

**Миксином** называют класс, расширяющий функциональность дочернего класса через ограниченную форму множественного наследования. Он не содержит собственных атрибутов и не предназначен для создания экземпляров.

Рассмотрим пример базового Estimator:

```
1 class MyEstimator:
2     def __init__(self, mean=0.):
3         self.mean: float = mean
4
5     def get_params(deep=True):
6         return {"mean": self.mean}
7
8     def set_params(mean=0.):
9         self.mean = mean
```

## FYI: миксины (mixins)

Создадим миксин для вывода гиперпараметров:

```
1 class PrinterMixin:
2     def __repr__(self):
3         return "\n".join([f"{att_name}: {att_value}" for
            att_name, att_value in vars(self).items()])
```

Тогда Estimator с выводом гиперпараметров будет иметь следующий вид:

```
1 class PrintableEstimator(MyEstimator, PrinterMixin):
2     def fit(X, y):
3         return self
```

Порядок базовых классов в множественном наследовании важен: он влияет на MRO (method resolution order).

# Композиция Estimators

**Последовательное объединение** Estimators: `sklearn.pipeline.Pipeline`

**Параллельное объединение** Estimators: `sklearn.pipeline.FeatureUnion`

Все промежуточные шаги в Pipeline являются Transformers, а последний шаг – любой Estimator

```
1 from sklearn.pipeline import Pipeline
2 from sklearn.preprocessing import StandardScaler
3 from sklearn.linear_model import LinearRegression
4
5
6 pipeline = Pipeline([
7     ("scaler", StandardScaler()),
8     ("regressor", LinearRegression()),
9 ])
10 ...
11 pipeline.fit(X, y)
12 y_pred = pipeline.predict(X_test) # mimics the last
    estimator interface
```

Доступ к параметрам индивидуальных Estimators организуется с помощью синтаксиса `<estimator>__<parameter>`

# Композиция Estimators через Pipeline

