# Design Principles of Scikit-learn

Antonio Prgomet
www.linkedin.com/in/antonioprgomet

# Content

1. Background
2. General Design Principles
3. Estimators, Predictors, Transformers
4. Test Set Creation, Pipelines and Model Selection through Grid Search
5. References and Reading Suggestions

- Code examples that I am going to demonstrate in this video are available at GitHub. Find the link in the video description.

# Scikit-learn - Background

- Scikit-learn is an open source ML library for Python. It has various alghoritms for classification, regression and clustering.
- First public release 2010.
- Scientific toolboxes built around Scikit are called "scikits", hence the name.
- Widely used in commercial and academic settings.
- Designed to be simple, efficient, and usable to non-experts.

# General Design Principles

Some important principles that the design of Scikit-learn follows:

- **Consistency.** All objects share a consistent interface and have a limited number of methods. The documentation is consistent as well.
- **Inspection.** Constructor parameters and learned parameters are stored and accessible as public attributes. Learned parameters are accessed by using an underscore suffix.
- **Non-proliferation of classes.** Only learning algorithms are represented using custom classes. Datasets are represented as NumPy arrays or SciPy sparse matrices. Hyperparameters are represented as regular Python strings or numbers whenever possible.
- **Sensible defaults.** When user-defined parameters are required, Scikit-Learn provides appropriate default values. This makes it easy to test and implement different models to get a base-line.

# Estimators, Predictors, and Transformers

The central object in Scikit-learn are Estimators. Estimators that have a predict method are called Predictors. Estimators that have a transform method are called Transformers.

- Estimators are used to build and fit models. Ex:
  `my_estimator = estimator.fit(data, targets)`
- Predictors are used for making predictions. Ex:
  `my_prediction = predictor.predict(data)`
- Transformers are used for transforming data. Ex:
  `my_new_data = transformer.fit_transform(data)`

# Estimators

Estimators are (1) initialized before they are (2) fitted to learn from data.

- In the first step we attach some hyperparameters to control the learning process.
- In the second step we use the fit method where we run a learning algorithm for determining model-specific parameters based on the training data. These model specific parameters are available as attributes on the estimator object (using a underscore suffix).

- Example:

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression(fit_intercept=True)
lin_reg.fit(X, y)

lin_reg.intercept_  # Returns the fitted intercept.
```

# Predictors

Estimators that can make predictions through the predict method are called predictors.

- Predictors must provide a score function which quantifies the quality of its predictions. Higher scores means it is better.

- Example:

```
# lin_reg is now used as a predictor.
y_pred = lin_reg.predict(X_new)
```

# Transformers

Estimators that can transform a dataset through the transform method are called transformers.

- The transform() method takes the dataset you want to transform as an argument and returns the transformed dataset.

- Example:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X)
X_transformed = scaler.transform(X)

# Fit and transform in one step with fit_transform().
scaler_2 = StandardScaler()
X_transformed_2 = scaler_2.fit_transform(X)
```

# Train & Test Set Data

- A **Test Set** is often around 20% of your data (or less if it is a big data set) that is "set aside" and not touched until you finish your supervised ML model.
- When finished with your model, evaluate it on the test set to get an estimate of your models future performance. Generally, good performance on the training set and and bad on the test set indicates overfitting.

- Easily created:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2)
```

# Gridsearch

**GridSearchCV** allows you to do an exhaustive search over specified parameter values for an estimator and then selects the best one by cross validation. The "best one" is decided based on a score function.

- For large hyperparameter spaces, RandomizedSearchCV can be useful (see documentation for details).

# Pipelines

- In ML data is often going through a sequence of transformations to for instance handle missing values and scale data. Through **pipelines** we can chain these steps together so that they run sequentially.
- All estimators in a pipeline, except the last one, must be transformers (i.e. must have a transform method). The last estimator may be any type (transformer, classifier, etc.).

Two benefits of pipelines:

1. **Convenience/Efficiency** You only have to call fit and predict once on your data to fit a whole sequence of estimators.
2. **Joint parameter selection** You can grid search over parameters of all estimators in the pipeline at once.

- To handle different columns (e.g. numeric and categorical) jointly use **ColumnTransformer** (see documentation for details).

# References & Reading Suggestions

This presentation is much based on the following two **references**:

- Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.
- API design for machine learning software: experiences from the scikit-learn project, Buitinck *et al.*, 2013.

**Reading Suggestions**

- Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow by the author Aurélien Géron.
- At the homepage of Scikit-learn you can check out "Getting Started", "Tutorial" and the "User Guide".