Scikit-Learn: Machine Learning in Python

CSE 5449
Autumn ‘21

Arpan Jain
The Ohio State University
E-mail: jain.575@osu.edu
https://go.osu.edu/aj-prime
Outline

• Introduction
• Principles
• API
• Advanced API
Machine Learning Frameworks

- **Scikit-learn (one of the oldest Machine Learning framework):**
  - Built using SciPy, NumPy, Matplotlib, Pandas, SymPy, and others
  - No support for GPUs
    - [https://scikit-learn.org/](https://scikit-learn.org/)

- **XGBoost (Extreme Gradient Boosting):**
  - Optimized gradient boosting library

- **cuML:**
  - Aimed to accelerate ML on GPUs
  - A key component of the NVIDIA RAPIDS project

- **TensorFlow, PyTorch, Apache MXNet, Deeplearning4j, and other DL frameworks** also support classical ML algorithms and techniques
Scikit-learn

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on Numpy, Scipy, and matplotlib
- Open source, commercial usable – BSD license
- First version released in June 2007; 14 years ago!!!
- Includes classical learning algorithms, model evaluation, selection tools, and data preprocessing procedures
“Provide efficient and well-established machine learning tools within a programming environment that is accessible to non-machine learning experts and reusable in various scientific areas”
Outline

• Introduction

• Principles

• API

• Advanced API
General Principles

• **Consistency:** All objects share a consistent interface composed of a limited set of methods.

• **Inspection:** Constructor parameters and parameter values determined by learning algorithms are stored and exposed as public attributes.

• **Non-proliferation of classes:** Learning algorithms are the only objects to be represented using custom classes. Datasets are represented as NumPy arrays or SciPy sparse matrices. Hyper-parameter names and values are represented as standard Python strings or numbers whenever possible. This keeps scikit learn easy to use and easy to combine with other libraries.
General Principles

• **Composition:** Many machine learning tasks are expressible as sequences or combinations of transformations to data. Some learning algorithms are also naturally viewed as meta-algorithms parametrized on other algorithms. Whenever feasible, such algorithms are implemented and composed from existing building blocks.

• **Sensible defaults:** Whenever an operation requires a user-defined parameter, an appropriate default value is defined by the library. The default value should cause the operation to be performed in a sensible way (giving a baseline solution for the task at hand).
Outline

• Introduction
• Principles

• API

• Advanced API
Data Representation

• In machine learning tasks, data is modeled as a set of variables.

• **Input Variables:** $X_1, X_2, \ldots, X_n$ (also known as features)

• **Output variables:** $Y$

• A sample is defined as a pair of values $([x_1, \ldots, x_n]^T, y)$

• **Dataset:** A pair of matrices
  • Each row of input and output matrices represents a sample
  • Each column corresponds to one variable
Estimators

- Core functionality of Scikit-learn.
- Defines instantiation mechanisms of objects and exposes a ‘fit’ method for learning a model from training data.
- All supervised and unsupervised learning algorithms are offered as objects.
- Machine learning tasks like feature extraction, feature selection, and dimensionality reduction are also provided as estimators.
- Constructor of an estimator does not see any actual data, nor does it perform any actual learning. All it does is attach the given parameters.
Estimators

- Actual learning is performed by the `fit` method

- This method is called with training data (X_train and Y_train)

- Its task is to run a learning algorithm and to determine model-specific parameters from training data

```python
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(penalty="l1")
clf.fit(X_train, y_train)
```
Predictors

• Extends the notion of an estimator by adding a predict method that take an array X_test

• Outputs predictions for X_test

```python
from sklearn.cluster import KMeans
km = KMeans(n_clusters=10)
km.fit(X_train)
clust_pred = km.predict(X_test)
```
Transformers

• Some estimators implement a transformer interface that defines a transform method

• It takes as input some new data $X_{\text{test}}$ and yields as output a transformed version of $X_{\text{test}}$

• Preprocessing, feature selection, feature extraction, and dimensionality reduction algorithms are all provided as transformers in Scikit-learn.

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
```
Outline

• Introduction
• Principles
• API

• Advanced API
Advanced API: Pipelines and Feature Unions

• A distinguishing feature of the scikit-learn API is its ability to compose new estimators from several base estimators.

• Pipeline objects chain multiple estimators into a single one

• `FeatureUnion` objects combine multiple transformers into a single one that concatenates their outputs

• `Pipeline` and `FeatureUnion` can be combined to create complex and nested workflows.
```python
from sklearn.pipeline import FeatureUnion, Pipeline
from sklearn.decomposition import PCA, KernelPCA
from sklearn.feature_selection import SelectKBest

union = FeatureUnion([(
    "pca", PCA()
),
    (
    "kpca", KernelPCA(kernel="rbf")
)])

Pipeline([(
    "feat_union", union
),
    ("feat_sel", SelectKBest(k=10))
    ("log_reg", LogisticRegression(penalty="l2")
)]).fit(X_train, y_train).predict(X_test)
```
Advanced API: Model Selection

• The problem of model selection is to find, within some hyper-parameter space, the best combination of hyper-parameters, with respect to some user-specified criterion.

• Model selection is supported in two distinct meta-estimators, `GridSearchCV` and `RandomizedSearchCV`.

• *Take input as an estimator (basic or composite), whose hyper-parameters must be optimized, and a set of hyperparameter settings to search through.*
from sklearn.grid_search import GridSearchCV
from sklearn.svm import SVC

param_grid = [
    {
        "kernel": ["linear"],
        "C": [1, 10, 100, 1000]
    },
    {
        "kernel": ["rbf"],
        "C": [1, 10, 100, 1000],
        "gamma": [0.001, 0.00001]
    }
]

clf = GridSearchCV(SVC(), param_grid, scoring="f1", cv=10)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
Scikit-learn – choosing the right estimator

Classification
- SVC
- Ensemble Classifiers
- KNighbors Classifier
- SGD Classifier
- Naive Bayes
- Text Data
- Linear SVC
- <100K samples
- NOT WORKING

Regression
- SGD Regressor
- Lasso
- ElasticNet
- <100K samples
- few features should be important
- NOT WORKING

Clustering
- Spectral Clustering
- GMM
- kMeans
- Minibatch kMeans
- MeanShift
- VBGMM
- <10K samples
- NOT WORKING

Dimensionality Reduction
- Isomap
- Spectral Embedding
- LLE
- <10K samples
- kernel approximation

Data
- more data
- >50 samples
- just looking
- structure
- predicting a quantity
- predicting a category
- you have labeled data
- number of categories known

Not Working
- START
XGBoost

CSE 5449
Autumn ‘21

Arpan Jain
The Ohio State University
E-mail: jain.575@osu.edu
https://go.osu.edu/aj-prime
Outline

• Ensemble Learning

• XGBoost

• Using XGBoost

• Example
**Ensemble Learning**

- Combines several classifiers to produce better predictive performance than a single classifier.

- Main principle: a group of weak learners can come together to form a strong learner
  - Thus, increasing the accuracy

*Source: https://www.pluralsight.com/guides/ensemble-methods:-bagging-versus-boosting*
Bagging and Boosting

Source: https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/
Bagging and Boosting

Source: https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/
## Bagging and Boosting

<table>
<thead>
<tr>
<th>Similarities</th>
<th>Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both are ensemble methods to get N learners from 1 learner ...</td>
<td>... but, while they are built independently for Bagging, Boosting tries to add new models that do well where previous models fail.</td>
</tr>
<tr>
<td>Both generate several training data sets by random sampling...</td>
<td>... but only Boosting determines weights for the data to tip the scales in favor of the most difficult cases.</td>
</tr>
<tr>
<td>Both make the final decision by averaging the N learners (or taking the majority of them)...</td>
<td>... but it is an equally weighted average for Bagging and a weighted average for Boosting, more weight to those with better performance on training data.</td>
</tr>
<tr>
<td>Both are good at reducing variance and provide higher stability...</td>
<td>... but only Boosting tries to reduce bias. On the other hand, Bagging may solve the over-fitting problem, while Boosting can increase it.</td>
</tr>
</tbody>
</table>

Outline

• Ensemble Learning

• XGBoost

• Using XGBoost

• Example
XGBoost

• eXtreme Gradient Boosted trees

• Boosting is an ensemble method
  
  • Each tree boosts attributes that led to misclassification of previous tree

• It is Amazing!!!
  
  • Routinely wins Kaggle competitions
  • Easy to use
  • Fast
  • A good choice for an algorithm to start with
Features of XGBoost

• Regularized boosting (prevents overfitting)

• Can handle missing values automatically

• Parallel processing

• Can cross-validate at each iteration
  • Enables early stopping, finding optimal number of iterations

• Incremental training

• Tree pruning
  • Generally, results in deeper, but optimized, trees
Outline

• Ensemble Learning

• XGBoost

• **Using XGBoost**

• Example
Using XGBoost

• pip install xgboost

• Supports: CLI, C++, R, Julia, and JVM interfaces

• It’s not just made for scikit-learn, so it has its own interface
  • Uses Dmatrix structure to hold features and labels
    • Can create this easily from a numpy array though

• All parameters passed in via a dictionary

• Call train, then predict. It’s easy!!!
XGBoost Hyperparameters

• Booster
  • gbtree or gblinear

• Objective (can be multi:softmax, multi:softprob)

• Eta (learning rate – adjusts weights on each step)

• Max_depth (depth of the tree)

• Min_child_weight (can control overfitting, but too high will underfit)

• ...and many others
Outline

• Ensemble Learning
• XGBoost
• Using XGBoost
• Example
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)

# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
Thank You!

Jain.575@osu.edu

Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

The MVAPICH2 Project
http://mvapich.cse.ohio-state.edu/

The High-Performance Deep Learning Project
http://hidl.cse.ohio-state.edu/