Introduction to Machine Learning

Lecture 6 - Machine Learning with Scikit-learn

Guang Bing Yang, PhD

yguangbing@gmail.com, Guang.B@chula.ac.th February 26th, 2021

The problem setting

- A learning problem is about using a set of sample data instructively to construct a model to predict properties of unknown data.
- Features are defined as the attributes or fixed functions of data entries (dimensions)
- Supervised and unsupervised learning are covered by scikit-learn framework.
- Here is the link of scikit-learn for the models and algorithms in supervised learning: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
- Here is the link of scikit-learn for the models and algorithms in unsupervised learning: https://scikit-learn.org/stable/unsupervised_learning.html#unsupervised-learning
- Training set and testing set — ML commonly splits data samples into training and testing sets, while training set is used for learning some properties of data samples (parameters), the testing set is used for evaluation of the learned model or properties.

Load an example dataset

- Scikit-learn comes with several standard datasets for supervised and unsupervised learning, e.g., the iris and digits datasets for classification and diabetes datasets for regression.
- Here are python code to load the sample datasets:
  - from sklearn import datasets
  - iris = datasets.load_iris()
  - digits = datasets.load_digits()
- More details see Lab7:

Load from external datasets

- Scikit-learn works on any numeric data stored as numpy arrays or scipy sparse matrices.
- Other types that are convertible to numeric arrays such as pandas DataFrame are also acceptable.
- pandas.io provides tools to read data from common formats including CSV, Excel, JSON and SQL. DataFrames may also be constructed from lists of tuples or dicts. Pandas handles heterogeneous data smoothly and provides tools for manipulation and conversion into a numeric array suitable for scikit-learn.
- scipy.io specializes in binary formats often used in scientific computing context such as .mat and .arff
- numpy/routines.io for standard loading of columnar data into numpy arrays
- scikit-learn's datasets.load_svmlight_file for the svmlight or libSVM sparse format
- scikit-learn's datasets.load_files or directories of text files where the name of each directory is the name of each category and each file inside of each directory corresponds to one sample from that category
- For some miscellaneous data such as images, videos, and audio, you may wish to refer to
  - skimage.io or Imageio for loading images and videos into numpy arrays
  - scipy.io.wavfile.read for reading WAV files into a numpy array

Training and predicting

- In In the case of the digits dataset, the task is to predict, given an image, which digit it represents.
- We are given samples of each of the 10 possible classes (the digits zero through nine) on which we *fit* an estimator to be able to *predict* the classes to which unseen samples belong.
- In scikit-learn, an estimator for classification is a Python object that implements the methods fit(X, y) and predict(T).
- An example of an estimator is the class sklearn.svm.SVC, which implements support vector classification. The estimator's constructor takes as arguments the model's parameters.
- from sklearn import svm
- clf = svm.SVC(gamma=0.001, C=100.)

Training and predicting

- In In the case of the digits dataset, the task is to predict, given an image, which digit it represents.
- We are given samples of each of the 10 possible classes (the digits zero through nine) on which we *fit* an estimator to be able to *predict* the classes to which unseen samples belong.
- In scikit-learn, an estimator for classification is a Python object that implements the methods fit(X, y) and predict(T).
- An example of an estimator is the class sklearn.svm.SVC, which implements support vector classification. The estimator's constructor takes as arguments the model's parameters.
- from sklearn import svm
- clf = svm.SVC(gamma=0.001, C=100.)
- # To train the model clf
- clf.fit(digits.data[:-1], digits.target[:-1])
- # to predict new data after training
- clf.predict(digits.data[-1:])

Conventions

- scikit-learn estimators follow certain rules to make their behavior more predictive. These are described in more detail in the Glossary of Common Terms and API Elements
- Some important terms:
  - attribute(s)
  - estimator, estimators — An object which manages the estimation and decoding of a model. it must provide a fit function.
  - feature(s), feature vector, n_features
  - fitting, fit (to train a model)
  - hyperparameter, or hyper-parameter — parameters about parameters
  - parameter(s) — learned from data
  - impute or imputation — dealing with missing data— Algorithms that attempt to fill in (or impute) missing values are referred to as imputation algorithms.
  - n_features — the number of features
  - n_outputs — the number of outputs in the target (# of classes)
  - n_samples — the number of samples
  - n_targets == n_outputs

Type casting

- scikit-learn always uses float64 for its input unless otherwise specified.
- To cast data type, using
- X = np.array(X, dtype='float32')
- X.dtype —> 'float32'

Refitting and updating parameters

- Hyper-parameters of an estimator can be updated after it has been constructed via the set_params() method. Calling fit() more than once will overwrite what was learned by any previous fit():
- the default kernel rbf is first changed to linear via SVC.set_params() after the estimator has been constructed, and changed back to rbf to refit the estimator and to make a second prediction.

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.svm import SVC
X, y = load_iris(return_X_y=True)

clf = SVC()
clf.set_params(kernel='linear').fit(X, y)

clf.predict(X[:5])

# re-training the model based on the trained model parameters
clf.set_params(kernel='rbf').fit(X, y)

clf.predict(X[:5])
```

```
array([0, 0, 0, 0, 0])
```


Multiclass vs. multilabel fitting

- When using multi class classifiers the learning and prediction task that is performed is dependent on the format of the target data fit upon
- For example, a classifier fits on a id array of multi class labels. It can also fit to a 2d array of binary label classifier— similar to 1-to-k encoding scheme we learned before.

```
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import LabelBinarizer

X = [[1, 2], [2, 4], [4, 5], [3, 2], [3, 1]]
y = [0, 0, 1, 1, 2]

classif = OneVsRestClassifier(estimator=SVC(random_state=0))
classif.fit(X, y).predict(X)
```

```
array([0, 0, 1, 1, 2])
```

In the above case, the classifier is fit on a 1d array of multiclass labels and the predict() method therefore provides corresponding multiclass predictions. It is also possible to fit upon a 2d array of binary label indicators:

```
y = LabelBinarizer().fit_transform(y)
classif.fit(X, y).predict(X)
```

```
array([[1, 0, 0],
       [1, 0, 0],
       [0, 1, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

```
y
```

```
array([[1, 0, 0],
       [1, 0, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 0, 1]])
```


Supervised learning models in scikit-learn

- Scikit-learn provides a rich libraries for supervised learning algorithms and modelling solutions. You can refer them here:
- https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
  - the basic one is Ordinary Least Squares, which is the linear regression model. The non-negative least squares has some practical usage for frequency counts or prices of goods.
  - Elastic-Net is a powerful one which includes both L1 and L2 regularizations.
  - Multi-task Elastic-Net is an elastic-net model that estimates sparse coefficients for multiple regression problems jointly: is a 2D array of shape (n_samples, n_tasks). The selected features are the same for all the regression problems, also called tasks, so n_tasks == n_features.
  - Least Angle Regression (LARS) is a regression algorithm for high-dimensional data.

Supervised learning models in scikit-learn

- Scikit-learn provides a rich libraries for supervised learning algorithms and modelling solutions. You can refer them here:
- https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
  - Bayesian Regression is a regression techniques using uninformative priors over the hyper parameters of the model, or using gamma distributions,as Bayesian Ridge Regression. They are fully generative modelling approaches and output posterior probabilities.
  - Logistic regression, despite its name, is a linear model for classification rather than regression. It is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. Its implementation can fit binary, One-vs-Rest, or multinomial logistic regression with optional L1 and L2 or Elastic-Net regularization.

Supervised learning models in scikit-learn

- Scikit-learn provides a rich libraries for supervised learning algorithms and modelling solutions. You can refer them here:
- https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
  - Linear and Quadratic Discriminant Analysis, they are two classic classifiers using discriminant functions as the decision surface functions. These classes have closed-form solutions that can be easily computed, are inherently multiclass, have proven to work well in practice, and have no hyperparameters to tune.
  - Kernel ridge regression (KRR) combines Ridge regression and classification (linear least squares with l2-norm regularization) with the kernel trick.
  - Support Vector Machines are supervised methods both for classification and regression problems.
    - For classification, SVC, NuSVC, and LinearSVC for both binary and multi-class classifications
    - For regression problems: Support Vector Regression includes SVR, NuSVR and LinearSVR

Supervised learning models in scikit-learn

- Scikit-learn provides a rich libraries for supervised learning algorithms and modelling solutions. You can refer them here:
- https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
  - Stochastic Gradient Descent (SGD) is a simple but very efficient approach to fit linear classifier and regressor under convex loss functions such as SVMs and Logistic Regression. SGD is merely an optimization technique and does not correspond to a specific family of machine learning models. It is only a *way* to train a model.
  - For classification, there is SGDClassifier, implements a plain stochastic gradient descent learning routine which supports different loss functions and penalties for classification.
  - For regression, there is SGDRegressor, implements a plain stochastic gradient descent learning routine which supports different loss functions and penalties to fit linear regression models. It is good for a large number of training samples (> 10, 100).

Tips on practical use of SGDClassifier and SGDRegressor

- First, SGD is sensitive to feature scaling, so it is highly recommended to scale your data before bring them to training process.
- For example, scale each attribute on the input vector X to [0,1] or [-1,+1], or standardize it to have mean 0 and variance 1. Note that the *same* scaling must be applied to the test vector to obtain meaningful results. This can be easily done using StandardScaler:
- Finding a reasonable regularization term is best done using automatic hyper-parameter search, e.g. GridSearchCV or RandomizedSearchCV, usually in the range 10.0**-np.arange(1,7).
- Empirically, we found that SGD converges after observing approximately 10^6 training samples. Thus, a reasonable first guess for the number of iterations is max_iter = np.ceil(10**6 / n), where n is the size of the training set.
- If you apply SGD to features extracted using PCA we found that it is often wise to scale the feature values by some constant c such that the average L2 norm of the training data equals one.
- We found that Averaged SGD works best with a larger number of features and a higher eta0

Questions?

Lab6