Scikit-learn

Introduction à l'apprentissage automatique – GIF-4101 / GIF-7005

Professor: Christian Gagné

Week 3

UNIVERSITÉ LAVAL

Homeworks

- The homeworks are done with scikit-learn (and PyTorch) in the PAX environment (Jupyter notebooks)
- Python language
  - Interpreted, open, general purpose, easy-to-use language (executable pseudo-code)
  - Rapid Prototyping, performance optimization with high-performance libraries and/or C routines
  - Widespread adoption in the field of machine learning
- Scikit-learn (`https://scikit-learn.org`)
  - Python library to perform machine learning
  - Includes most of the methods seen in class
  - Open source

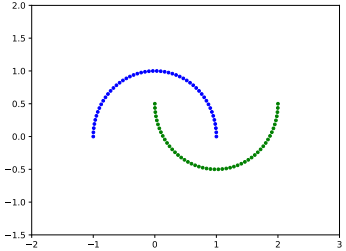
Environments used

- PAX (`https://pax.ulaval.ca/`)
  - Learning environment based on Jupyter Hub (notebooks running on a remote server)
  - Framework to perform the exercises and facilitate correction
- On a local machine: Anaconda (`https://www.anaconda.com/`)
  - Distribution containing a set of 1000+ scientific computing and data science libraries
  - Available on Windows, Mac and Linux
  - Includes Jupyter Notebook
- In the cloud: Google Colab (`https://colab.research.google.com/`)
  - Notebook environment offered by Google, based/inspired by Jupyter Notebook
  - Computing resources in the cloud (basic GPU)
  - First developed to promote TensorFlow (from Google)
  - Other machine learning libraries have now been installed (scikit-learn, PyTorch, etc.)

Generate and visualize data

- Generate half-moon 2D data

```
from sklearn import datasets
X, R = datasets.make_moons(100)
import numpy
from matplotlib import pyplot
colors = numpy.array([x for x in "bgrcmyk"])
pyplot.scatter(X[:, 0], X[:, 1], color=colors[R].tolist(), s=10)
pyplot.show()
```



Train a classifier

- Train a classifier (linear SVC) on the half-moons

```
import numpy
from matplotlib import pyplot
from sklearn import datasets, svm, metrics

X, R = datasets.make_moons(100)

linsvc = svm.LinearSVC(C=1.0)
linsvc.fit(X, R)

err_train = 1 - metrics.accuracy_score(linsvc.predict(X), R)
print("Train error: %.3f" % err_train)
```

- Console output

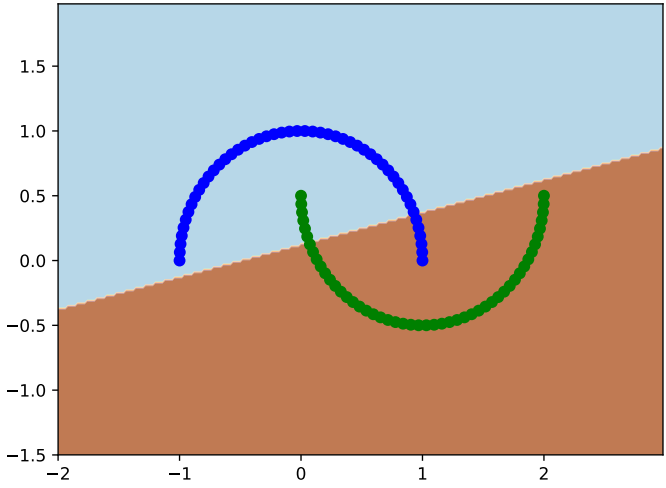
```
Train error: 0.120
```

Visualization with matplotlib

- Visualize decision boundaries and regions

```
h = .02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = numpy.meshgrid(numpy.arange(x_min, x_max, h),
                        numpy.arange(y_min, y_max, h))
Y = linsvc.predict(numpy.c_[xx.ravel(), yy.ravel()])

colors = numpy.array([x for x in "bgrcmyk"])
Y = Y.reshape(xx.shape)
pyplot.contourf(xx, yy, Y, cmap=pyplot.cm.Paired, alpha=0.8)
pyplot.scatter(X[:, 0], X[:, 1], cmap=pyplot.cm.Paired,\
  color=colors[R].tolist())
pyplot.xlim(xx.min(), xx.max())
pyplot.ylim(yy.min(), yy.max())
pyplot.show()
```

Visualization with matplotlib (result)



Estimate the empirical error

- Partition the dataset into training and test sets

```
import matplotlib.pyplot
from sklearn import datasets, svm, model_selection, metrics

X, R = datasets.make_moons(300)
X_train, X_test, R_train, R_test = \
  model_selection.train_test_split(X, R, train_size=0.67, test_size=0.33)

linsvc = svm.LinearSVC(C=1.0)
linsvc.fit(X_train, R_train)

err_train = 1 - metrics.accuracy_score(linsvc.predict(X_train), R_train)
err_test = 1 - metrics.accuracy_score(linsvc.predict(X_test), R_test)
print("Train error: %.3f, test error: %.3f" % (err_train, err_test))
```

- Console output

```
Train error: 0.114, test error: 0.111
```

k-fold cross-validation

- Partition the dataset into training and test sets

```
import numpy
import matplotlib.pyplot
from sklearn import datasets, svm, model_selection

X, R = datasets.make_moons(300)

linsvc = svm.LinearSVC(C=1.0)
scores = model_selection.cross_val_score(linsvc, X, R, cv=10)

err_crossval = 1 - numpy.mean(scores)
print("10-fold cross-validation error: %.3f" % err_crossval)
```

- Console output

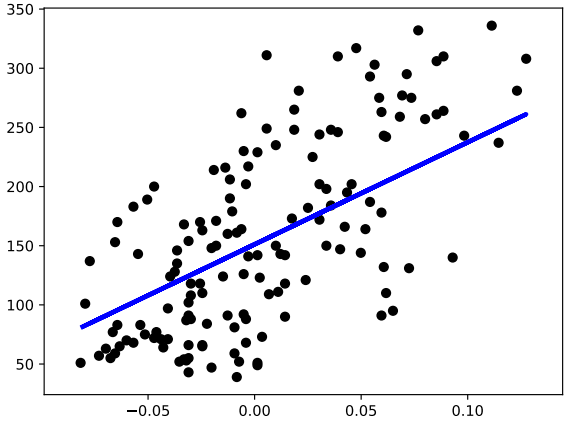
```
10-fold cross-validation error: 0.113
```

Linear regression

```
import numpy
from matplotlib import pyplot
from sklearn import datasets, linear_model, model_selection, metrics

diabetes = datasets.load_diabetes()
X = diabetes.data[:, numpy.newaxis, 2]
r = diabetes.target
X_train, X_test, r_train, r_test = \
    model_selection.train_test_split(X, r, train_size=0.67, test_size=0.33)

regr = linear_model.LinearRegression()
regr.fit(X_train, r_train)

print("w_1: %.3f, w_0: %.3f" % (regr.coef_[0], regr.intercept_))

mse_train = metrics.mean_squared_error(regr.predict(X_train), r_train)
print("Mean square error (train): %.3f" % mse_train)
mse_test = metrics.mean_squared_error(regr.predict(X_test), r_test)
print("Mean square error (test): %.3f" % mse_test)

pyplot.scatter(X_test, r_test, color="black")
X_test_sorted = numpy.sort(X_test)
pyplot.plot(X_test_sorted, regr.predict(X_test_sorted), color="blue", linewidth=3)
pyplot.show()
```

Linear regression (results)

```
w_1: 862.933, w_0: 151.150
Mean square error (train): 4054.312
Mean square error (test): 3612.435
```



Datasets

- Scikit-learn data in Numpy array format
  - Reading text or CSV files is possible with `numpy.loadtxt` functions
- Many "toy" datasets available in scikit-learn
  - `sklearn.datasets.load_boston` (regression)
  - `sklearn.datasets.load_iris` (classification)
  - `sklearn.datasets.load_diabetes` (regression)
  - `sklearn.datasets.load_digits` (classification)
  - `sklearn.datasets.load_linnerud` (multivariate regression)
- Many other data sets available
  - Images
  - Data in svmlight / libsvm format
  - Olivetti faces
  - 20 newsgroups text dataset
  - Forest covertypes
  - Etc.

Code your own algorithms (1/2)

- scikit-learn objects must respect the same specific interface
- Estimator, method `fit` to learn from data:

```
estimator = obj.fit(data, targets)
```

or (unsupervised case):

```
estimator = obj.fit(data)
```

- Predictor, to process new data:

```
prediction = obj.predict(data)
```

and when the certainty of predictions is quantified (e.g. probabilities):

```
prediction = obj.predict_proba(data)
```

Code your own algorithms (2/2)

- Transform, filter or modify data:

```
new_data = obj.transform(data)
```

when learning and transformation work well together:

```
new_data = obj.fit_transform(data)
```

- Model, return a performance measure (e.g., learning quality, likelihood) [high values are better]:

```
score = obj.score(data)
```

- For more information about programming classifiers with scikit-learn:

https://scikit-learn.org/stable/developers/develop.html

Documentation about Python

- *Apprendre à programmer avec Python 3*: `https://inforef.be/swi/python.htm`
- *Think Python: How to Think Like a Computer Scientist*: `http://www.greenteapress.com/thinkpython/html/`
- *Learning Python*: `https://learning.oreilly.com/library/view/learning-python-5th/9781449355722/`
- *Dive into Python 3*: `http://link.springer.com.acces.bibl.ulaval.ca/book/10.1007%2F978-1-4302-2416-7`
- Matplotlib: `http://matplotlib.org/`

Documentation about scikit-learn

- Website: `http://scikit-learn.org/`
- Scikit-learn tutorials: `http://scikit-learn.org/stable/tutorial/index.html`
- Scikit-learn user guide: `http://scikit-learn.org/stable/user_guide.html`
- *Learning scikit-learn: Machine Learning in Python*: `https://learning.oreilly.com/library/view/python-machine-learning/9781787125933/`

Multivariate parametric classification with scikit-learn

- `discriminant_analysis.QuadraticDiscriminantAnalysis`
  - Each class is represented by a normal distribution $\mathcal{N}_D(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$
  - Separate and complete covariance matrices $\boldsymbol{\Sigma}_i$ for each class
- `discriminant_analysis.LinearDiscriminantAnalysis`
  - Sharing a complete covariance matrix between the classes, $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$, $\forall i$
- `naive_bayes.GaussianNB`
  - Diagonal and distinct covariance matrices $\boldsymbol{\Sigma}_i$ for each class
- `neighbors.NearestCentroid`
  - Isotropic and shared covariance matrix (equal variances for all dimensions, covariance $= 0$), $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma} = \sigma \mathbf{I}$, $\forall i$
  - Equal a priori probabilities, $P(C_i) = P(C_j)$, $\forall i,j$
  - Corresponds to the classifier with the closest mean

$$\mathrm{h}_i(\mathbf{x}) = -\|\mathbf{x} - \mathbf{m}_i\|^2$$