Scikit-learn

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

Professeur: Christian Gagné

Semaine 3

UNIVERSITÉ LAVAL

Travaux pratiques

- Travaux pratiques réalisés avec scikit-learn (et PyTorch) dans l'environnement PAX (notebooks Jupyter)
- Langage Python
  - Langage interprété, ouvert, d'usage général, facile à utiliser (pseudo-code exécutable)
  - Prototypage rapide, optimisation des performances par librairies performantes et/ou routines en C
  - Adoption généralisée dans le domaine de l'apprentissage automatique
- Scikit-learn (`https://scikit-learn.org`)
  - Librairie Python pour effectuer de l'apprentissage automatique
  - Inclut la plupart des méthodes vues en classe
  - Projet ouvert (*open source*)

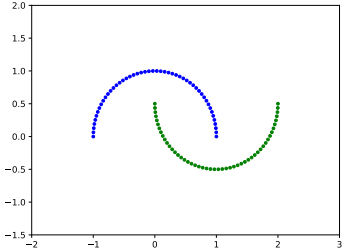
Environnements utilisés

- PAX (`https://pax.ulaval.ca/`)
  - Environnement d'apprentissage basé sur Jupyter Hub (notebooks fonctionnant sur serveur distant)
  - Cadre pour réaliser les exercices et faciliter la correction
- Sur une machine locale : Anaconda (`https://www.anaconda.com/`)
  - Distribution contenant un ensemble de 1000+ librairies de calcul scientifique et sciences des données
  - Disponible sous Windows, Mac et Linux
  - Inclus Jupyter Notebook
- Dans le nuage : Google Colab (`https://colab.research.google.com/`)
  - Environnement de notebooks offert par Google, basé/inspiré de Jupyter Notebook
  - Ressources de calcul dans le nuage (GPU de base)
  - D'abord développé pour promouvoir TensorFlow (de Google)
  - Autres librairies d'apprentissage automatique maintenant installées (scikit-learn, PyTorch, etc.)

Générer et visualiser des données

- Générer données 2D en demi-lunes

```
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()
```



Entraîner un classifieur

- Entraîner un classifieur (SVM linéaire) sur les demi-lunes

```
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)
```

- Sortie à la console

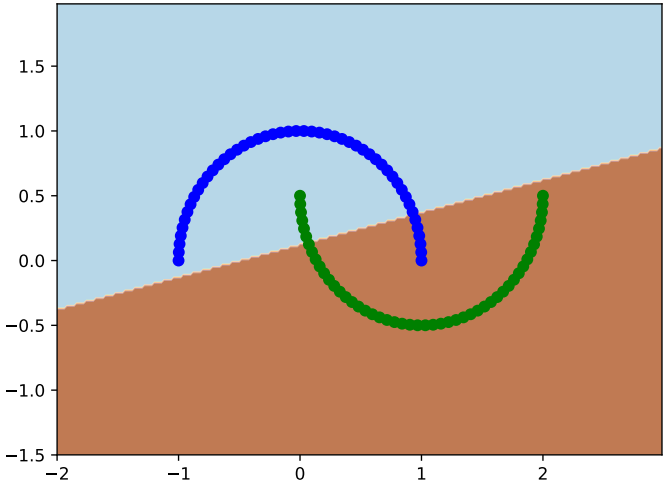
```
Train error: 0.120
```

Visualisation avec matplotlib

- Visualiser les frontières et régions de décision

```
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()
```

Visualisation avec matplotlib (résultat)



Estimation de l'erreur empirique

- Partition du jeu en ensemble d'entraînement et de test

```
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))
```

- Sortie à la console

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

Validation croisée à k-plis

- Partition du jeu en ensemble d'entraînement et de test

```
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)
```

- Sortie à la console

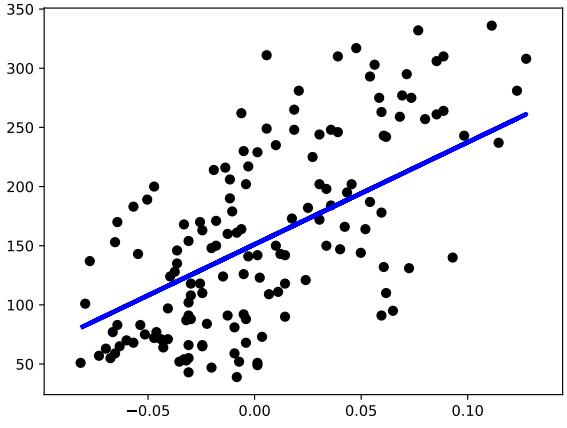
```
10-fold cross-validation error: 0.113
```

Régression linéaire

```
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()
```

Régression linéaire (résultats)

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



Jeux de données

- Données de scikit-learn en format array Numpy
  - Lecture de fichiers texte ou CSV possible avec fonctions `numpy.loadtxt`
- Nombreux jeux de données jouets disponibles dans scikit-learn
  - `sklearn.datasets.load_boston` (régression)
  - `sklearn.datasets.load_iris` (classement)
  - `sklearn.datasets.load_diabetes` (régression)
  - `sklearn.datasets.load_digits` (classement)
  - `sklearn.datasets.load_linnerud` (régression multivariée)
- Nombreux autres jeux de données disponibles
  - Images
  - Données en format svmlight / libsvm
  - Olivetti faces
  - 20 newsgroups text dataset
  - Forest covertypes
  - Etc.

Coder ses propres algorithmes (1/2)

- Objets scikit-learn doivent respecter un même interface précis
- Estimateur, méthode `fit` pour apprendre des données :

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

ou bien (cas non-supervisé) :

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

- Prédicteur, traiter de nouvelles données :

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

et lorsque la certitude des prédictions est quantifiée (ex. probabilités) :

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

Coder ses propres algorithmes (2/2)

- Transformateur, filtrer ou modifier les données :

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

lorsque l'apprentissage et la transformation se fait bien ensemble :

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

- Modèle, retourner un mesure de performance (ex. qualité d'apprentissage, vraisemblance) [valeurs élevées sont meilleures] :

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

- Pour plus d'information sur la programmation de classifieurs avec scikit-learn :

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

Documentation sur 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 sur scikit-learn

- Site Web : `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/`

Classement paramétrique multivarié avec scikit-learn

- `discriminant_analysis.QuadraticDiscriminantAnalysis`
  - Chaque classe représentée par une distribution normale $\mathcal{N}_D(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$
  - Matrices de covariance $\boldsymbol{\Sigma}_i$ distinctes et complètes pour chaque classe
- `discriminant_analysis.LinearDiscriminantAnalysis`
  - Partage d'une matrice de covariance complète entre les classes, $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$, $\forall i$
- `naive_bayes.GaussianNB`
  - Matrices de covariance $\boldsymbol{\Sigma}_i$ diagonales et distinctes pour chaque classe
- `neighbors.NearestCentroid`
  - Matrice de covariance partagée et isotropique (variances égales pour toutes les dimensions, covariance nulle), $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma} = \sigma \mathbf{I}$, $\forall i$
  - Probabilités a priori égales, $P(C_i) = P(C_j)$, $\forall i,j$
  - Correspond au classifieur à la plus proche moyenne

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