Scikit-learn

Apprentissage et reconnaissance – GIF-4101 / GIF-7005
Professeur : Christian Gagné

UNIVERSITÉ LAVAL

Semaine 3 : 21 septembre 2016

Travaux pratiques

- Travaux pratiques réalisés avec scikit-learn et scikit-neuralnetwork, en langage Python
  - Installation de Python et des librairies dans les laboratoires d'informatiques du département de génie électrique et de génie informatique (PLT-0103 et PLT-0105)
  - Utiliser l'image Linux
- 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
  - Librairie Python pour effectuer de l'apprentissage automatique
  - Inclut la plupart des méthodes vues en classe, excepté pour les réseaux de neurones (solution : scikit-neuralnetwork)
  - Projet ouvert (*open source*)
  - Site Web : `http://scikit-learn.org`

Installation des librairies

- Option 1 (**privilégiée**) : Anaconda (`https://www.continuum.io`)
  - Distribution contenant un ensemble de 100 librairies de calcul scientifique
  - Disponible sous Windows, Mac et Linux
  - Inclut Numpy, Scipy, matplotlib, scikit-learn, etc.
  - N'inclut pas scikit-neuralnetwork
  - Solution pour ceux moins à l'aise avec Python et la ligne de commande
- Option 2 (**à vos risques**) : avec installateur du système et/ou pip
  - Installer Python 3 par l'installateur du système
    - Apt-get sous Ubuntu, Homebrew sous Mac
  - Installer Numpy, matplotlib, scikit-learn et scikit-neuralnetwork avec pip
- Scikit-neuralnetwork ne fonctionne pas sous Windows, comme il dépend de Theano et Lasagne
  - Très difficile de faire de l'apprentissage profond avec Windows.

Exemple d'installation sur macOS

- Installation sur macOS avec Homebrew et pip (Python 3)

```
# Si Python 3 n'est pas encore installé
brew install python3

# Installation à partir des sources, peut prendre du temps
brew install openblas
brew install numpy --with-openblas --with-python3
brew install scipy --with-openblas --with-python3

# Installation du reste avec pip
pip3 install matplotlib
pip3 install scikit-learn
pip3 install scikit-neuralnetwork
```

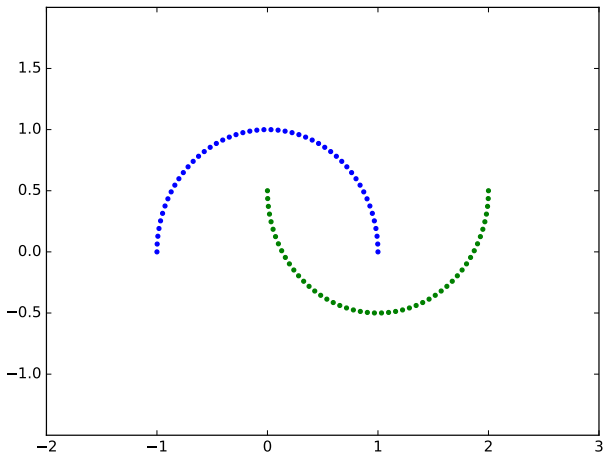
Générer et visualiser des données

- Générer données 2D en demi-lunes

```
from sklearn import datasets
X, R = datasets.make_moons(100)
```

- Tracer les données avec matplotlib

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

# Générer données 2D en demi-lunes
X, R = datasets.make_moons(100)

# Entraîner classifieur de type SVM linéaire
linsvc = svm.LinearSVC(C=1.0)
linsvc.fit(X, R)

# Évaluer performance en entraînement
acc_train = metrics.accuracy_score(linsvc.predict(X), R)
print("Train accuracy: %.3f" % acc_train)
```

- Sortie à la console

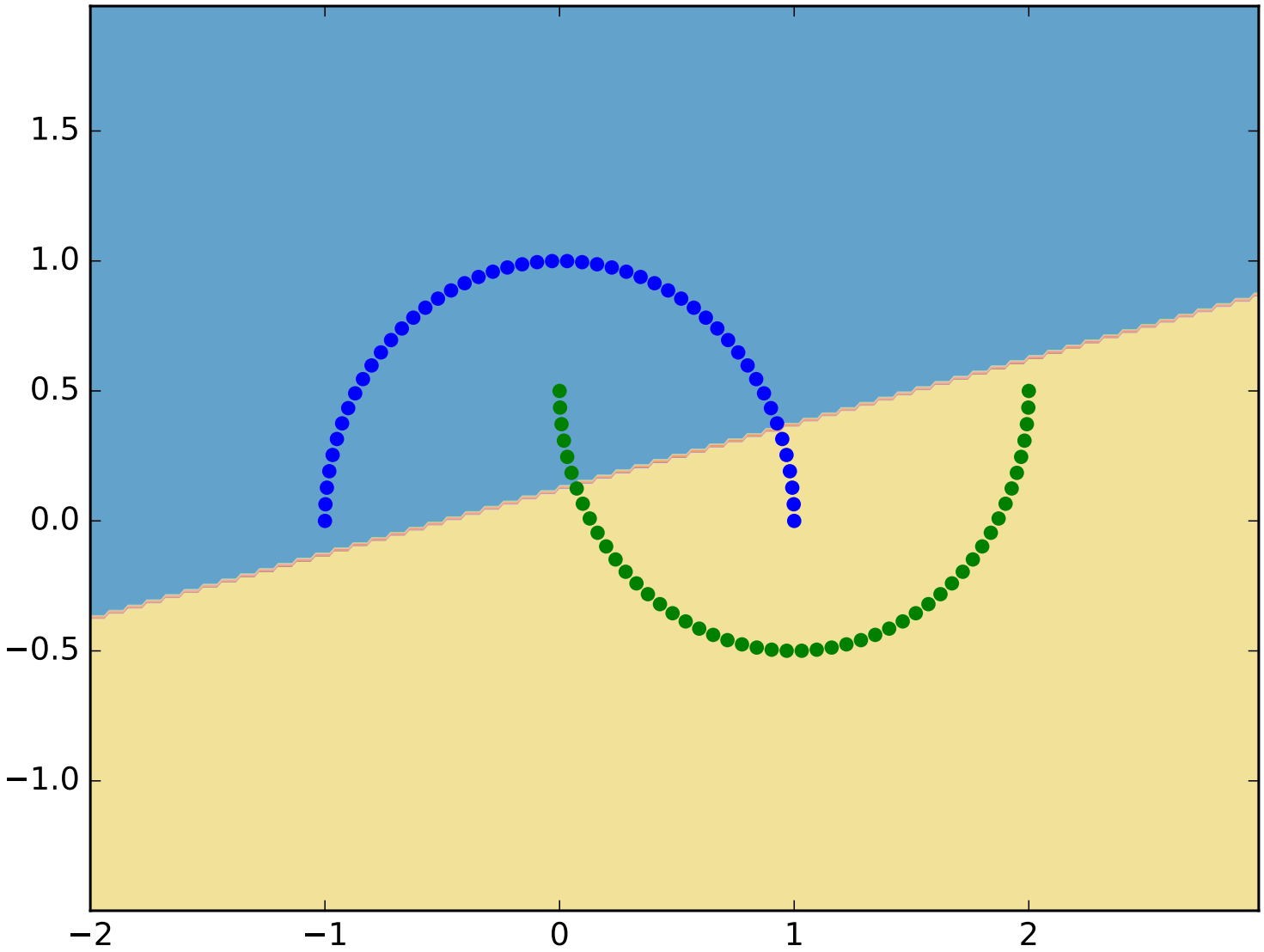
```
Train accuracy: 0.880
```

Visualisation avec matplotlib

- Visualiser les frontières et régions de décision

```
# Créer un mesh pour faire le tracé
h = .02       # Espacement du mesh
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()])

# Mettre le résultat dans un tracé couleur
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 numpy
import matplotlib.pyplot
from sklearn import datasets, svm, cross_validation, metrics

# Générer données 2D en demi-lunes
X, R = datasets.make_moons(300)
X_train, X_test, R_train, R_test = \
  cross_validation.train_test_split(X, R, test_size=0.33)

# Entraîner classifieur de type SVM linéaire
linsvc = svm.LinearSVC(C=1.0)
linsvc.fit(X_train, R_train)

# Évaluer erreur moyenne en entraînement
acc_train = metrics.accuracy_score(linsvc.predict(X_train), R_train)
acc_test = metrics.accuracy_score(linsvc.predict(X_test), R_test)
print("Train accuracy: %.3f, test accuracy: %.3f" % (acc_train, acc_test))
```

- Sortie à la console

```
Train accuracy: 0.881, test accuracy: 0.889
```

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, cross_validation

# Générer données 2D en demi-lunes
X, R = datasets.make_moons(300)

# Entraîner classifieur de type SVM linéaire
linsvc = svm.LinearSVC(C=1.0)
scores = cross_validation.cross_val_score(linsvc, X, R, cv=10)

# Évaluer erreur moyenne
acc_crossval = numpy.mean(scores)
print("10-fold cross-validation accuracy: %.3f" % acc_crossval)
```

- Sortie à la console

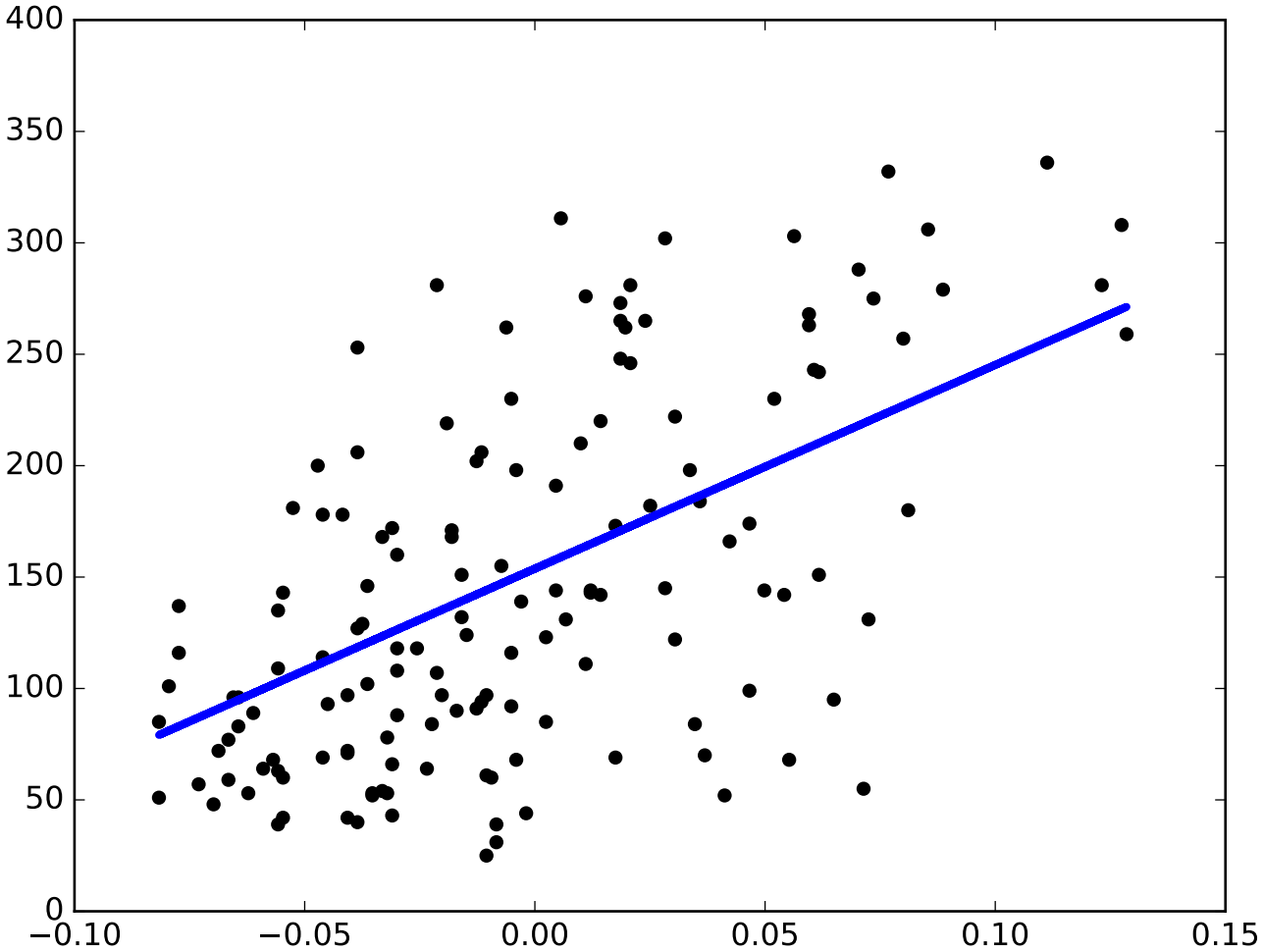
```
10-fold cross-validation accuracy: 0.883
```

Régression linéaire

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

# Lire jeu de données diabetes
diabetes = datasets.load_diabetes()

# Utiliser une seule variable et séparer entraînement/test
X = diabetes.data[:, numpy.newaxis, 2]
r = diabetes.target
X_train, X_test, r_train, r_test = cross_validation.train_test_split(X, r, test_size=0.33)

# Créer objet de régression linéaire et entraîner le modèle
regr = linear_model.LinearRegression()
regr.fit(X_train, r_train)

# Poids du modèle
print('w_1: %.3f, w_0: %.3f' % (regr.coef_[0], regr.intercept_))

# Erreurs quadratiques moyennes
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)

# Afficher résultats
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: 878.484, w_0: 148.580
Mean square error (train): 3985.306
Mean square error (test): 3770.875
```



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

- Objets scikit-learn suivent l'interface suivant
  - 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)
    ```
  - 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 : `http://scikit-learn.org/stable/developers/contributing.html#apis-of-scikit-learn-objects`

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* : `http://proquestcombo.safaribooksonline.com.acces.bibl.ulaval.ca/9781449355722?uicode=univlaval`
- *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* : `http://proquestcombo.safaribooksonline.com.acces.bibl.ulaval.ca/9781783281930`

Accès au laboratoire informatique GEL-GIF

- Laboratoire informatique GEL-GIF : PLT-0103 et PLT-0105, ouvert 24/7
- Stations de travail Windows et Linux (dual boot)
- Installation sur image Linux de Python 3 / Numpy / Scipy / matplotlib / scikit-learn / scikit-neuralnetwork
- Vous devez activer votre compte avant d'accéder au laboratoire (une seule fois) :
  `https://sechuron.gel.ulaval.ca/auth/comptes_analyse.php`
- Requiert une combinaison pour accéder aux locaux (donnée lors de l'activation du compte)