Introduction à `scikit-learn`

INF8460 - TP1

Polytechnique Montréal

Automne 2020

scikit-learn

Présentation

LA librairie d'apprentissage automatique en Python.

La documentation est très détaillée, et contient notamment des **guides** sur tous les aspects du machine learning couverts par la librairie, parmi lesquels *Naive Bayes*, la *régression logistique*, l'évaluation des modèles ou la validation croisée.

Naive Bayes

Tous les classifieurs de scikit-learn exposent les mêmes méthodes `fit` (pour l'entraînement) et `predict` (pour la prédiction), et éventuellement une méthode `transform`.

Ici, on va prendre l'exemple du classifieur Naive Bayes multinominal, que vous aurez à utiliser dans le TP :

```
>>> from sklearn.naive_bayes import MultinomialNB
```

Naive Bayes (suite)

Supposons qu'on ait des données $X \in \mathbb{R}^{m \times n}$ où m est le nombre d'échantillons et n le nombre de *features* (chaque ligne est un échantillon), et un vecteur $y \in \{0, 1\}^m$ qui contient les labels de chaque échantillon. On peut entraîner un classifieur Naive Bayes avec :

```
>>> clf = MultinomialNB(alpha=0.5)
>>> clf.fit(X, y)
```

Pour prédire les classes d'un jeu de test $X_{\text{test}} \in \mathbb{R}^{m' \times n}$:

```
>>> y_pred = clf.predict(X_test)
```

Régression logistique

Un autre modèle très utile lorsque des données binaires sont prédites est la régression logistique qui dans scikit-learn provient de

```
>>> from sklearn.linear_model import LogisticRegression
```

Ce modèle est utilisé de la même façon que Naive Bayes, avec entres autres les méthodes `fit` et `predict`:

```
>>> model = LogisticRegression(C=1.0)
>>> model.fit(X, y)
>>> y_pred = model.predict(X_test)
```

Le paramètre `C` est un paramètre de régularisation correspondant à $C = 1/\lambda$. Plus une valeur est petite, plus la régularisation sera grande et une très grande valeur fera en sorte qu'il n'y aura aucune régularisation.

Pipeline

Il y a parfois plusieurs étapes entre les données originales et les prédictions. Pour ces cas-là, scikit-learn possède un Pipeline qui permet de combiner différentes parties du modèle ensemble pour pouvoir les entraîner et utiliser ensemble.
Par exemple, pour standardiser les données avant de faire une régression logistique, on ferait

```
>>> pipeline = Pipeline(
...     [
...         ("scaler", StandardScaler()),
...         ("regression", LogisticRegression)
...     ]
... )
>>> pipeline.fit(X, y)
>>> pipeline.predict(X)
```

Métriques

Le module `sklearn.metrics` fournit de nombreux outils pour évaluer la qualité d'un modèle ou d'une prédiction : précision et rappel, F-mesure, courbe ROC, etc.

On peut se reporter à la section Classification Metrics du guide utilisateur pour une liste plus complète.

Métriques (suite)

Si on reprend l'exemple précédent, dans le contexte de l'apprentissage supervisé, on dispose d'un vecteur $y_{\text{true}} \in \{0, 1\}^{m'}$ qui contient les vrais labels du jeu de test.

On peut alors évaluer la qualité de notre prédiction avec (ici on prend la F-mesure comme métrique) :

```
>> from sklearn.metrics import f1_score
>> f1_score(y_true, y_pred)
```

Métriques (suite et fin)

On peut aussi obtenir un résumé des performances de classification du modèle avec la fonction `classification_report` :

```
>>> from sklearn.metrics import classification_report
>>> y_true = [0, 1, 2, 2, 2]
>>> y_pred = [0, 0, 2, 2, 1]
>>> print(classification_report(y_true, y_pred))
              precision    recall  f1-score

           A       0.50      1.00      0.67
           B       0.00      0.00      0.00
           C       1.00      0.67      0.80

    accuracy                           0.60
   macro avg       0.50      0.56      0.49
weighted avg       0.70      0.60      0.61
```

exemple tiré de la documentation scikit-learn