eBook Gratuit

APPRENEZ scikit-learn

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#scikit-learn

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: scikit-learn

It is an unofficial and free scikit-learn ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official scikit-learn.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Commencer avec scikit-learn

Remarques

`scikit-learn` est une bibliothèque open source polyvalente pour l'analyse de données écrite en python. Il est basé sur d'autres bibliothèques python: NumPy, SciPy et matplotlib

`scikit-learn` contient un certain nombre d'implémentations pour différents algorithmes populaires d'apprentissage automatique.

Examples

Installation de scikit-learn

La version stable actuelle de scikit-learn nécessite :

- Python (> = 2.6 ou> = 3.3),
- NumPy (> = 1.6.1),
- SciPy (> = 0.9).

Pour la plupart l' installation `pip` gestionnaire de paquets python peut installer python et toutes ses dépendances:

```
pip install scikit-learn
```

Cependant, pour les systèmes Linux, il est recommandé d'utiliser le `conda` paquets `conda` pour éviter les processus de génération possibles.

```
conda install scikit-learn
```

Pour vérifier que vous avez `scikit-learn` , exécutez en shell:

```
python -c 'import sklearn; print(sklearn.__version__)'
```

Installation de Windows et Mac OSX:

Canopy et Anaconda proposent tous deux une version récente de *scikit-learn* , en plus d'un grand ensemble de bibliothèques scientifiques Python pour Windows, Mac OSX (également pertinentes pour Linux).

Former un classificateur avec validation croisée

Utilisation du jeu de données iris:

```
import sklearn.datasets
iris_dataset = sklearn.datasets.load_iris()
X, y = iris_dataset['data'], iris_dataset['target']
```

Les données sont divisées en trains et ensembles de test. Pour ce faire, nous utilisons la fonction utilitaire `train_test_split` pour séparer de manière aléatoire `X` et `y` (vecteurs de données et cibles) avec l'option `train_size=0.75` (les ensembles d'apprentissage contiennent 75% des données).

Les jeux de données d'entraînement sont introduits dans un classificateur de voisins les plus proches . L' `fit` de la méthode du classificateur adaptera le modèle aux données.

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75)
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)
```

Enfin, prédire la qualité sur l'échantillon de test:

```
clf.score(X_test, y_test) # Output: 0.94736842105263153
```

En utilisant une paire de trains et de jeux de tests, nous pouvons obtenir une estimation biaisée de la qualité du classificateur en raison du choix arbitraire de la division des données. En utilisant la *validation croisée,* nous pouvons adapter le classificateur à différents sous-ensembles train / test des données et effectuer une moyenne sur tous les résultats de précision. La fonction `cross_val_score` adapte un classificateur aux données d'entrée en utilisant une validation croisée. Il peut prendre en entrée le nombre de fractionnements (plis) à utiliser (5 dans l'exemple ci-dessous).

```
from sklearn.cross_validation import cross_val_score
scores = cross_val_score(clf, X, y, cv=5)
print(scores)
# Output: array([ 0.96666667,  0.96666667,  0.93333333,  0.96666667,  1.        ])
print "Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() / 2)
# Output: Accuracy: 0.97 (+/- 0.03)
```

Créer des pipelines

La recherche de modèles dans les données passe souvent par une chaîne d'étapes de traitement des données, par exemple la sélection des fonctionnalités, la normalisation et la classification. Dans `sklearn` , un pipeline d'étapes est utilisé pour cela.

Par exemple, le code suivant montre un pipeline composé de deux étapes. Le premier met à l'échelle les entités et le second forme un classificateur sur le jeu de données augmenté résultant:

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

pipeline = make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors=4))
```

Une fois le pipeline créé, vous pouvez l'utiliser comme une étape normale (en fonction de ses étapes spécifiques). Ici, par exemple, le pipeline se comporte comme un classificateur. Par conséquent, nous pouvons l'utiliser comme suit:

```
# fitting a classifier
pipeline.fit(X_train, y_train)
# getting predictions for the new data sample
pipeline.predict_proba(X_test)
```

Interfaces et conventions:

Différentes opérations avec des données sont effectuées à l'aide de classes spéciales.

La plupart des classes appartiennent à l'un des groupes suivants:

- algorithmes de classification (dérivés de `sklearn.base.ClassifierMixin`) pour résoudre les problèmes de classification
- algorithmes de régression (dérivés de `sklearn.base.RegressorMixin`) pour résoudre le problème de la reconstruction de variables continues (problème de régression)
- Transformations de données (dérivées de `sklearn.base.TransformerMixin`) qui prétraitent les données

Les données sont stockées dans `numpy.array` s (mais les autres objets de type `pandas.DataFrame` tels que `pandas.DataFrame` s sont acceptés si ceux-ci sont convertibles en `numpy.array` s)

Chaque objet dans les données est décrit par un ensemble de caractéristiques. La convention générale est que l'échantillon de données est représenté avec un tableau, où la première dimension est l'identifiant de l'échantillon de données, la seconde est l'identifiant de la caractéristique.

```
import numpy
data = numpy.arange(10).reshape(5, 2)
print(data)

Output:
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
```

Dans les conventions `sklearn` , le jeu de données ci-dessus contient 5 objets, chacun décrit par 2 entités.

Jeux de données d'échantillons

Pour faciliter les tests, `sklearn` fournit des jeux de données `sklearn.datasets` dans le module `sklearn.datasets` . Par exemple, chargez le jeu de données iris de Fisher:

```
import sklearn.datasets
```

```
iris_dataset = sklearn.datasets.load_iris()
iris_dataset.keys()
['target_names', 'data', 'target', 'DESCR', 'feature_names']
```

Vous pouvez lire la description complète, les noms des entités et les noms des classes (`target_names`). Ceux-ci sont stockés sous forme de chaînes.

Nous nous intéressons aux données et aux classes stockées dans `target` champs de `data` et `target` . Par convention, ceux-ci sont notés `X` et `y`

```
X, y = iris_dataset['data'], iris_dataset['target']
X.shape, y.shape
((150, 4), (150,))
```

```
numpy.unique(y)
array([0, 1, 2])
```

Les formes de `X` et `y` indiquent qu'il y a 150 échantillons avec 4 caractéristiques. Chaque échantillon appartient à l'une des classes suivantes: 0, 1 ou 2.

`X` et `y` peuvent maintenant être utilisés pour former un classificateur, en appelant la méthode `fit()` du classificateur.

Voici la liste complète des ensembles de données fournis par le module `sklearn.datasets` avec leur taille et leur utilisation prévue:

Charger avec	La description	Taille	Usage
`load_boston()`	Ensemble de données sur les prix de l'immobilier à Boston	506	régression
`load_breast_cancer()`	Ensemble de données sur le cancer du sein du Wisconsin	569	classification (binaire)
`load_diabetes()`	Ensemble de données sur le diabète	442	régression
`load_digits(n_class)`	Jeu de données Digits	1797	classification
`load_iris()`	Ensemble de données Iris	150	classification (multi-classes)
`load_linnerud()`	Ensemble de données Linnerud	20	régression multivariée

Notez que (source: http://scikit-learn.org/stable/datasets/) :

> Ces jeux de données sont utiles pour illustrer rapidement le comportement des différents algorithmes implémentés dans le scikit. Ils sont toutefois souvent trop petits

pour être représentatifs des tâches d'apprentissage réel dans le monde réel.

En plus de ces jeux de données d'échantillons de jouets `sklearn.datasets` , `sklearn.datasets` fournit également des fonctions utilitaires pour charger des jeux de données externes:

- `load_mlcomp` pour charger des exemples de jeux de données à partir du référentiel mlcomp.org (notez que les jeux de données doivent être téléchargés auparavant). Voici un exemple d'utilisation.
- `fetch_lfw_pairs` et `fetch_lfw_people` pour le chargement du jeu de données de paires de visages étiquetés dans la nature (LFW) à partir de http://vis-www.cs.umass.edu/lfw/ , utilisé pour la vérification du visage (resp. reconnaissance du visage). Ce jeu de données est supérieur à 200 Mo. Voici un exemple d'utilisation.

Lire Commencer avec scikit-learn en ligne: https://riptutorial.com/fr/scikit-learn/topic/1035/commencer-avec-scikit-learn

Chapitre 2: Caractéristique d'exploitation du récepteur (ROC)

Examples

Introduction à ROC et AUC

Exemple de mesure ROC (Receiver Operating Characteristic) pour évaluer la qualité de sortie du classificateur.

Les courbes ROC comportent généralement un taux de vrais positifs sur l'axe Y et un taux de faux positifs sur l'axe X. Cela signifie que le coin supérieur gauche de l'intrigue est le point «idéal» - un taux de faux positif de zéro et un taux positif réel de un. Ce n'est pas très réaliste, mais cela signifie qu'une plus grande surface sous la courbe (AUC) est généralement meilleure.

La «raideur» des courbes ROC est également importante, car il est idéal pour maximiser le taux positif réel tout en minimisant le taux de faux positifs.

Un exemple simple:

```
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
```

Valeurs `y` arbitraires - en réalité, il s'agit des valeurs cibles prévues (`model.predict(x_test)`):

```
y = np.array([1,1,2,2,3,3,4,4,2,3])
```

Les scores sont la précision moyenne sur les données de test et les étiquettes données (`model.score(X,Y)`):

```
scores = np.array([0.3, 0.4, 0.95,0.78,0.8,0.64,0.86,0.81,0.9, 0.8])
```

Calculez la courbe ROC et l'ASC:

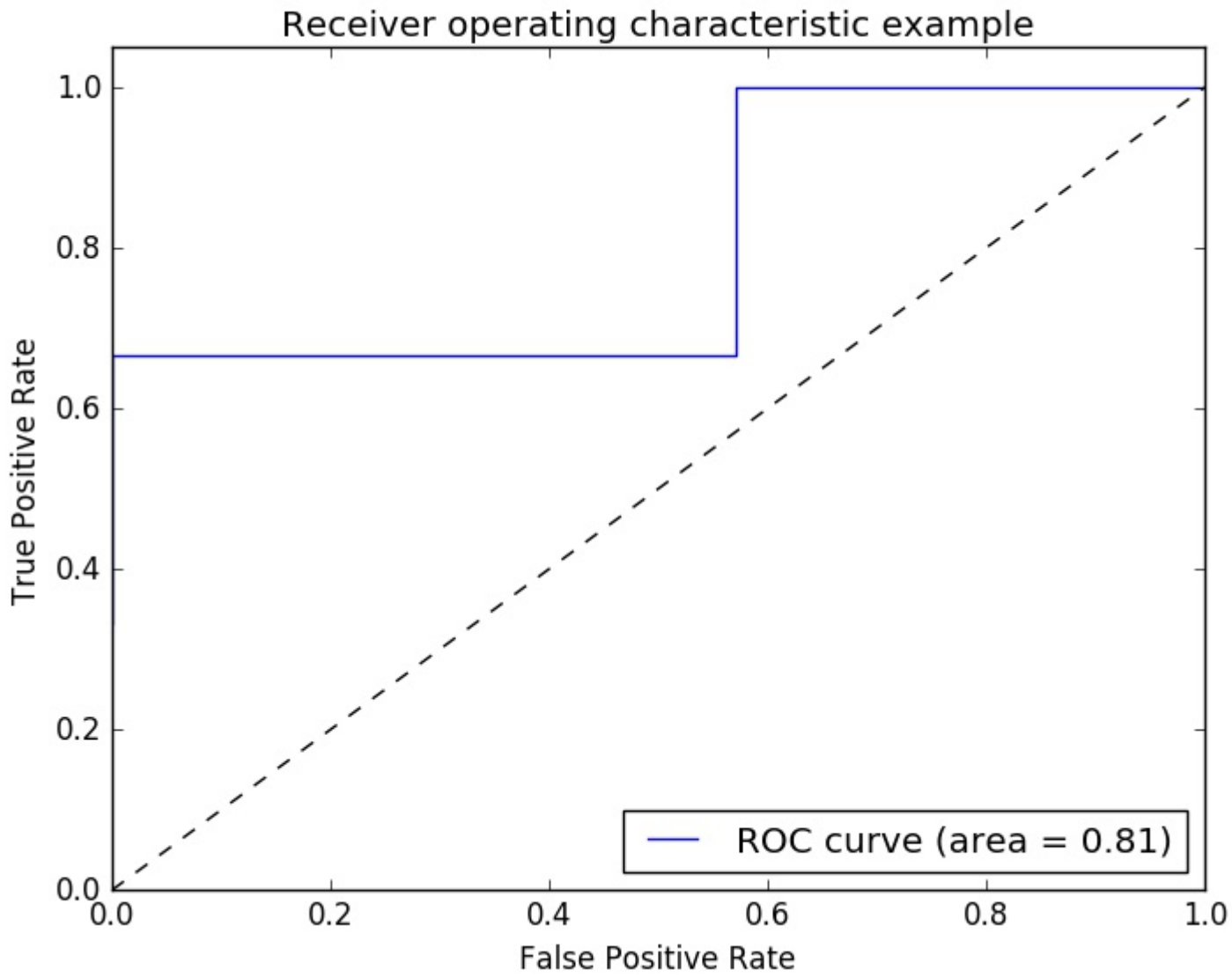
```
fpr, tpr, thresholds = metrics.roc_curve(y, scores, pos_label=2)
roc_auc = metrics.auc(fpr, tpr)
```

Traçage:

```
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

Sortie:



Note: les sources ont été prises à partir de ces link1 et lien2

Score ROC-AUC avec validation et validation croisée

On a besoin des probabilités prévues pour calculer le score ROC-AUC (aire sous la courbe). `cross_val_predict` utilise les méthodes de `predict` des classificateurs. Pour pouvoir obtenir le score ROC-AUC, il suffit de sous-classer le classificateur, en écrasant la méthode `predict` , de manière à ce qu'il agisse comme `predict_proba` .

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import cross_val_predict
```

```
from sklearn.metrics import roc_auc_score

class LogisticRegressionWrapper(LogisticRegression):
    def predict(self, X):
        return super(LogisticRegressionWrapper, self).predict_proba(X)

X, y = make_classification(n_samples = 1000, n_features=10, n_classes = 2, flip_y = 0.5)

log_reg_clf = LogisticRegressionWrapper(C=0.1, class_weight=None, dual=False,
             fit_intercept=True)

y_hat = cross_val_predict(log_reg_clf, X, y)[:,1]

print("ROC-AUC score: {}".format(roc_auc_score(y, y_hat)))
```

sortie:

```
ROC-AUC score: 0.724972396025
```

Lire Caractéristique d'exploitation du récepteur (ROC) en ligne: https://riptutorial.com/fr/scikit-learn/topic/5945/caracteristique-d-exploitation-du-recepteur--roc-

Chapitre 3: Classification

Examples

Utilisation de machines à vecteurs de support

Les machines à vecteurs de support sont une famille d'algorithmes qui tentent de faire passer un hyperplan (éventuellement de grande dimension) entre deux ensembles de points étiquetés, de sorte que la distance des points par rapport au plan est optimale dans un certain sens. Les SVM peuvent être utilisés pour la classification ou la régression (correspondant respectivement à `sklearn.svm.SVC` et à `sklearn.svm.SVR`).

Exemple:

Supposons que nous travaillions dans un espace 2D. Tout d'abord, nous créons des données:

```
import numpy as np
```

Maintenant, nous créons *x* et *y* :

```
x0, x1 = np.random.randn(10, 2), np.random.randn(10, 2) + (1, 1)
x = np.vstack((x0, x1))

y = [0] * 10 + [1] * 10
```

Notez que *x* est composé de deux gaussiennes: une centrée autour de *(0, 0)* et une centrée autour de *(1, 1)* .

Pour construire un classificateur, on peut utiliser:

```
from sklearn import svm

svm.SVC(kernel='linear').fit(x, y)
```

Vérifions la prédiction pour *(0, 0)* :

```
>>> svm.SVC(kernel='linear').fit(x, y).predict([[0, 0]])
array([0])
```

La prédiction est que la classe est 0.

Pour la régression, nous pouvons faire de même:

```
svm.SVR(kernel='linear').fit(x, y)
```

RandomForestClassifier

Une forêt aléatoire est un méta-estimateur qui correspond à un certain nombre de classificateurs d'arbres de décision sur divers sous-échantillons du jeu de données et utilise la moyenne pour améliorer la précision prédictive et le sur-ajustement du contrôle.

Un exemple d'utilisation simple:

Importer:

```
from sklearn.ensemble import RandomForestClassifier
```

Définir les données de train et les données cibles:

```
train = [[1,2,3],[2,5,1],[2,1,7]]
target = [0,1,0]
```

Les valeurs dans la `target` représentent l'étiquette que vous souhaitez prédire.

Initier un objet RandomForest et effectuer un apprentissage (ajustement):

```
rf = RandomForestClassifier(n_estimators=100)
rf.fit(train, target)
```

Prédire:

```
test = [2,2,3]
predicted = rf.predict(test)
```

Analyse des rapports de classification

Construisez un rapport de texte montrant les principales métriques de classification, y compris la précision et le rappel , le score f1 (la moyenne harmonique de précision et le rappel) et le support (le nombre d'observations de cette classe dans l'ensemble d'apprentissage).

Exemple de `sklearn` docs :

```
from sklearn.metrics import classification_report
y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))
```

Sortie -

```
             precision    recall  f1-score   support

    class 0       0.50      1.00      0.67         1
    class 1       0.00      0.00      0.00         1
    class 2       1.00      0.67      0.80         3

avg / total       0.70      0.60      0.61         5
```

GradientBoostingClassifier

Gradient Boosting pour la classification. Le classificateur à graduation est un ensemble additif d'un modèle de base dont l'erreur est corrigée par itérations successives (ou étapes) par l'ajout d'arbres de régression qui corrigent les résidus (l'erreur de l'étape précédente).

Importer:

```
from sklearn.ensemble import GradientBoostingClassifier
```

Créer des données de classification de jouets

```
from sklearn.datasets import load_iris

iris_dataset = load_iris()

X, y = iris_dataset.data, iris_dataset.target
```

Partageons ces données en un ensemble de formation et de test.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)
```

Instanciez un modèle `GradientBoostingClassifier` utilisant les paramètres par défaut.

```
gbc = GradientBoostingClassifier()
gbc.fit(X_train, y_train)
```

Laissez-nous le marquer sur l'ensemble de test

```
# We are using the default classification accuracy score
>>> gbc.score(X_test, y_test)
1
```

Par défaut, il y a 100 estimateurs construits

```
>>> gbc.n_estimators
100
```

Cela peut être contrôlé en définissant `n_estimators` sur une valeur différente pendant le temps d'initialisation.

Un arbre de décision

Un arbre de décision est un classificateur qui utilise une séquence de règles verbeuses (comme a> 7) qui peuvent être facilement comprises.

L'exemple ci-dessous forme un classificateur d'arbre de décision à l'aide de trois vecteurs de

caractéristiques de longueur 3, puis prédit le résultat pour un quatrième vecteur d'entité encore inconnu, appelé vecteur de test.

```
from sklearn.tree import DecisionTreeClassifier

# Define training and target set for the classifier
train = [[1,2,3],[2,5,1],[2,1,7]]
target = [10,20,30]

# Initialize Classifier.
# Random values are initialized with always the same random seed of value 0
# (allows reproducible results)
dectree = DecisionTreeClassifier(random_state=0)
dectree.fit(train, target)

# Test classifier with other, unknown feature vector
test = [2,2,3]
predicted = dectree.predict(test)

print predicted
```

La sortie peut être visualisée en utilisant:

```
import pydot
import StringIO

dotfile = StringIO.StringIO()
tree.export_graphviz(dectree, out_file=dotfile)
(graph,)=pydot.graph_from_dot_data(dotfile.getvalue())
graph.write_png("dtree.png")
graph.write_pdf("dtree.pdf")
```

Classification à l'aide de la régression logistique

Dans LR Classifier, les probabilités décrivant les résultats possibles d'un seul essai sont modélisées en utilisant une fonction logistique. Il est implémenté dans la bibliothèque `linear_model`

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

L'implémentation de sklearn LR peut s'adapter à une régression logistique binaire, un contre un ou multinomiale avec une régularisation facultative L2 ou L1. Par exemple, considérons une classification binaire sur un exemple de jeu de données sklearn

```
from sklearn.datasets import make_hastie_10_2

X,y = make_hastie_10_2(n_samples=1000)
```

Où X est un `n_samples X 10` et y les étiquettes cibles -1 ou +1.

Utiliser la séparation des tests de train pour diviser les données d'entrée en jeux de formation et de test (70% à 30%)

```
from sklearn.model_selection import train_test_split
#sklearn.cross_validation in older scikit versions

data_train, data_test, labels_train, labels_test = train_test_split(X,y, test_size=0.3)
```

L'utilisation du classificateur LR est similaire à d'autres exemples

```
# Initialize Classifier.
LRC = LogisticRegression()
LRC.fit(data_train, labels_train)

# Test classifier with the test data
predicted = LRC.predict(data_test)
```

Utiliser la matrice de confusion pour visualiser les résultats

```
from sklearn.metrics import confusion_matrix

confusion_matrix(predicted, labels_test)
```

Lire Classification en ligne: https://riptutorial.com/fr/scikit-learn/topic/2468/classification

Chapitre 4: Réduction de dimensionnalité (sélection de fonctionnalités)

Examples

Réduction de la dimension avec l'analyse en composantes principales

L'analyse en composantes principales trouve des séquences de combinaisons linéaires des entités. La première combinaison linéaire maximise la variance des entités (soumise à une contrainte d'unité). Chacune des combinaisons linéaires suivantes maximise la variance des entités dans le sous-espace orthogonal à celles générées par les combinaisons linéaires précédentes.

Une technique commune de réduction de dimension consiste à utiliser uniquement les *k* premières combinaisons linéaires. Supposons que les entités soient une matrice *X* de *n* lignes et *m* colonnes. Les *k* premières combinaisons linéaires forment une matrice β_k de *m* lignes et *k* colonnes. Le produit *X* β comporte *n* lignes et *k* colonnes. Ainsi, la matrice résultante βk peut être considérée comme une réduction de *m* à *k* dimensions, en conservant les parties à forte variance de la matrice d'origine *X.*

Dans `scikit-learn` , PCA est exécuté avec `sklearn.decomposition.PCA` . Par exemple, supposons que nous commençons par une matrice 100 X 7, construite de telle sorte que la variance ne soit contenue que dans les deux premières colonnes (en réduisant les 5 dernières colonnes):

```
import numpy as np
np.random.seed(123) # we'll set a random seed so that our results are reproducible
X = np.hstack((np.random.randn(100, 2) + (10, 10), 0.001 * np.random.randn(100, 5)))
```

Réalisons une réduction à 2 dimensions:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
```

Maintenant vérifions les résultats. Tout d'abord, voici les combinaisons linéaires:

```
pca.components_
# array([[ -2.84271217e-01,  -9.58743893e-01,  -8.25412629e-05,
#           1.96237855e-05,  -1.25862328e-05,   8.27127496e-05,
#          -9.46906600e-05],
#        [ -9.58743890e-01,   2.84271223e-01,  -7.33055823e-05,
#          -1.23188872e-04,  -1.82458739e-05,   5.50383246e-05,
#           1.96503690e-05]])
```

Notez que les deux premières composantes de chaque vecteur sont plusieurs fois plus grandes que les autres, ce qui montre que la PCA a reconnu que la variance se trouvait principalement

dans les deux premières colonnes.

Pour vérifier le rapport de la variance expliquée par cette PCA, on peut examiner `pca.explained_variance_ratio_` :

```
pca.explained_variance_ratio_
# array([ 0.57039059,  0.42960728])
```

Lire Réduction de dimensionnalité (sélection de fonctionnalités) en ligne: https://riptutorial.com/fr/scikit-learn/topic/4829/reduction-de-dimensionnalite--selection-de-fonctionnalites-

Chapitre 5: Régression

Examples

Moindres Carrés Ordinaires

Les moindres carrés ordinaires est une méthode permettant de trouver la combinaison linéaire des caractéristiques qui correspond le mieux au résultat observé dans le sens suivant.

Si le vecteur des résultats à prédire est y et que les variables explicatives forment la matrice X , alors OLS trouvera le vecteur β résolvant

$min_{\beta} | y\hat{} - y |_2^2$

où $y\hat{} = X\beta$ est la prédiction linéaire.

Dans sklearn, cela se fait en utilisant `sklearn.linear_model.LinearRegression` .

Contexte d'application

OLS ne devrait être appliqué qu'aux problèmes de régression, il est généralement inadapté aux problèmes de classification: Contraste

- Un spam est-il un email? (Classfication)
- Quelle est la relation linéaire entre les votes accrus dépend de la longueur de la réponse? (Régression)

Exemple

`LinearRegression` un modèle linéaire avec du bruit, puis voyons si `LinearRegression` gère pour reconstruire le modèle linéaire.

Tout d'abord, nous générons la matrice `X` :

```
import numpy as np

X = np.random.randn(100, 3)
```

Nous allons maintenant générer le `y` sous forme de combinaison linéaire de `X` avec du bruit:

```
beta = np.array([[1, 1, 0]])
y = (np.dot(x, beta.T) + 0.01 * np.random.randn(100, 1))[:, 0]
```

Notez que la vraie combinaison linéaire générant `y` est donnée par `beta.

Pour essayer de le reconstruire à partir de `X` et `y` seul, faisons:

```
>>> linear_model.LinearRegression().fit(x, y).coef_
```

```
array([  9.97768469e-01,   9.98237634e-01,   7.55016533e-04])
```

Notez que ce vecteur est très similaire à la `beta` .

Lire Régression en ligne: https://riptutorial.com/fr/scikit-learn/topic/5190/regression

Chapitre 6: Sélection de fonctionnalité

Examples

Suppression de fonctions à faible variance

C'est une technique de sélection de fonctionnalités très basique.

Son idée sous-jacente est que si une entité est constante (c'est-à-dire qu'elle a 0 variance), elle ne peut pas être utilisée pour trouver des modèles intéressants et peut être supprimée de l'ensemble de données.

Par conséquent, une approche heuristique de l'élimination des entités consiste à supprimer d'abord toutes les entités dont la variance est inférieure à un seuil (bas).

En prenant l' exemple de la documentation , supposons que nous commencions par

```
X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]
```

Il y a *3* caractéristiques booléennes ici, chacune avec *6* instances. Supposons que nous souhaitons supprimer celles qui sont constantes dans au moins 80% des instances. Certains calculs de probabilité montrent que ces caractéristiques devront présenter une variance inférieure à *0,8 * (1 - 0,8)* . Par conséquent, nous pouvons utiliser

```
from sklearn.feature_selection import VarianceThreshold
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
sel.fit_transform(X)
# Output: array([[0, 1],
                 [1, 0],
                 [0, 0],
                 [1, 1],
                 [1, 0],
                 [1, 1]])
```

Notez comment la première fonctionnalité a été supprimée.

Cette méthode doit être utilisée avec précaution car une faible variance ne signifie pas nécessairement qu'une fonctionnalité est «inintéressante». Considérons l'exemple suivant où nous construisons un jeu de données qui contient *3* entités, les deux premières étant constituées de variables distribuées aléatoirement et la troisième de variables uniformément distribuées.

```
from sklearn.feature_selection import VarianceThreshold
import numpy as np

# generate dataset
np.random.seed(0)

feat1 = np.random.normal(loc=0, scale=.1, size=100) # normal dist. with mean=0 and std=.1
feat2 = np.random.normal(loc=0, scale=10, size=100) # normal dist. with mean=0 and std=10
```

```
feat3 = np.random.uniform(low=0, high=10, size=100) # uniform dist. in the interval [0,10)
data = np.column_stack((feat1,feat2,feat3))

data[:5]
# Output:
# array([[  0.17640523,  18.83150697,   9.61936379],
#        [  0.04001572, -13.47759061,   2.92147527],
#        [  0.0978738 , -12.70484998,   2.4082878 ],
#        [  0.22408932,   9.69396708,   1.00293942],
#        [  0.1867558 , -11.73123405,   0.1642963 ]])

np.var(data, axis=0)
# Output: array([  1.01582662e-02,   1.07053580e+02,   9.07187722e+00])

sel = VarianceThreshold(threshold=0.1)
sel.fit_transform(data)[:5]
# Output:
# array([[ 18.83150697,   9.61936379],
#        [-13.47759061,   2.92147527],
#        [-12.70484998,   2.4082878 ],
#        [  9.69396708,   1.00293942],
#        [-11.73123405,   0.1642963 ]])
```

Désormais, la première fonctionnalité a été supprimée en raison de sa faible variance, tandis que la troisième fonctionnalité (la moins intéressante) a été conservée. Dans ce cas, il aurait été plus approprié de prendre en compte un *coefficient de variation* car celui-ci est indépendant de la mise à l'échelle.

Lire Sélection de fonctionnalité en ligne: https://riptutorial.com/fr/scikit-learn/topic/4909/selection-de-fonctionnalite

Chapitre 7: Sélection du modèle

Examples

Validation croisée

Apprendre les paramètres d'une fonction de prédiction et les tester sur les mêmes données est une erreur méthodologique: un modèle qui ne ferait que répéter les étiquettes des échantillons qu'il vient de voir aurait un score parfait mais ne pourrait rien prédire d'utile- données invisibles Cette situation s'appelle **overfitting** . Pour l'éviter, il est courant lors d'une expérience d'apprentissage automatique (supervisé) de conserver une partie des données disponibles sous la forme d'un **ensemble de test** `X_test, y_test` . Notez que le mot «expérience» n'est pas destiné à désigner uniquement une utilisation académique, car même dans les environnements commerciaux, l'apprentissage automatique commence expérimentalement.

Dans scikit-learn, la fonction helper train_test_split permet de calculer rapidement un fractionnement aléatoire en jeux d'entraînement et en tests. Chargez le jeu de données de l'iris pour l'adapter à une machine à vecteur de support linéaire:

```
>>> import numpy as np
>>> from sklearn import cross_validation
>>> from sklearn import datasets
>>> from sklearn import svm

>>> iris = datasets.load_iris()
>>> iris.data.shape, iris.target.shape
((150, 4), (150,))
```

Nous pouvons maintenant échantillonner rapidement un ensemble d'entraînement en conservant 40% des données pour tester (évaluer) notre classificateur:

```
>>> X_train, X_test, y_train, y_test = cross_validation.train_test_split(
...     iris.data, iris.target, test_size=0.4, random_state=0)

>>> X_train.shape, y_train.shape
((90, 4), (90,))
>>> X_test.shape, y_test.shape
((60, 4), (60,))
```

Maintenant, après avoir des ensembles de train et de test, utilisons-les:

```
>>> clf = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)
>>> clf.score(X_test, y_test)
```

Validation croisée K-Fold

La validation croisée K-fold est un processus systématique de répétition répétée de la procédure de fractionnement train / test afin de réduire la variance associée à un seul essai de séparation

train / test. Vous divisez essentiellement le jeu de données entier en K de taille égale "plis", et chaque fois est utilisé une fois pour tester le modèle et K-1 fois pour entraîner le modèle.

Plusieurs techniques de pliage sont disponibles avec la bibliothèque scikit. Leur utilisation dépend des caractéristiques des données d'entrée. Certains exemples sont

K-Fold

Vous divisez essentiellement le jeu de données entier en K de taille égale "plis", et chaque fois est utilisé une fois pour tester le modèle et K-1 fois pour entraîner le modèle.

```
from sklearn.model_selection import KFold
X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
y = np.array([1, 2, 1, 2])
cv = KFold(n_splits=3, random_state=0)

for train_index, test_index in cv.split(X):
...    print("TRAIN:", train_index, "TEST:", test_index)

TRAIN: [2 3] TEST: [0 1]
TRAIN: [0 1 3] TEST: [2]
TRAIN: [0 1 2] TEST: [3]
```

`StratifiedKFold` est une variation de k-fold qui renvoie des plis stratifiés: chaque ensemble contient approximativement le même pourcentage d'échantillons de chaque classe cible que l'ensemble complet

ShuffleSplit

Utilisé pour générer un nombre défini par l'utilisateur de groupes de données de train / test indépendants. Les échantillons sont d'abord mélangés puis divisés en une paire de trains et de jeux de test.

```
from sklearn.model_selection import ShuffleSplit
X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
y = np.array([1, 2, 1, 2])
cv = ShuffleSplit(n_splits=3, test_size=.25, random_state=0)

for train_index, test_index in cv.split(X):
...    print("TRAIN:", train_index, "TEST:", test_index)

TRAIN: [3 1 0] TEST: [2]
TRAIN: [2 1 3] TEST: [0]
TRAIN: [0 2 1] TEST: [3]
```

`StratifiedShuffleSplit` est une variante de ShuffleSplit, qui renvoie des divisions stratifiées, c'est-à-dire qui créent des divisions en conservant le même pourcentage pour chaque classe cible que dans l'ensemble complet.

D'autres techniques de pliage telles que Leave One / p Out et TimeSeriesSplit (une variante de K-

fold) sont disponibles dans la bibliothèque scikit model_selection.

Lire Sélection du modèle en ligne: https://riptutorial.com/fr/scikit-learn/topic/4901/selection-du-modele

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec scikit-learn	Alleo, Ami Tavory, Community, Gabe, Gal Dreiman, panty, Sean Easter, user2314737
2	Caractéristique d'exploitation du récepteur (ROC)	Gal Dreiman, Gorkem Ozkaya
3	Classification	Ami Tavory, Drew, Gal Dreiman, hashcode55, Mechanic, Raghav RV, Sean Easter, tfv, user6903745, Wayne Werner
4	Réduction de dimensionnalité (sélection de fonctionnalités)	Ami Tavory, DataSwede, Gal Dreiman, Sean Easter, user2314737
5	Régression	Ami Tavory, draco_alpine
6	Sélection de fonctionnalité	Ami Tavory, user2314737
7	Sélection du modèle	Gal Dreiman, Mechanic