Бесплатная электронная книга

УЧУСЬ

scikit-learn

Free unaffiliated eBook created from
Stack Overflow contributors.

#scikit-learn

Около

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

глава 1: Начало работы с scikit-learn

замечания

`scikit-learn` - универсальная библиотека с открытым исходным кодом для анализа данных, написанная на python. Он основан на других библиотеках python: NumPy, SciPy и matplotlib

`scikit-learn` содержит ряд реализаций для разных популярных алгоритмов машинного обучения.

Examples

Установка scikit-learn

Текущая стабильная версия scikit-learn требует :

- Python (> = 2.6 или> = 3.3),
- NumPy (> = 1,6,1),
- SciPy (> = 0,9).

Для большинства `pip` установки python менеджер пакетов может установить python и все его зависимости:

```
pip install scikit-learn
```

Однако для Linux-систем рекомендуется использовать диспетчер пакетов `conda` , чтобы избежать возможных процессов сборки

```
conda install scikit-learn
```

Чтобы проверить, что у вас есть `scikit-learn` , выполните в оболочке:

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

Установка Windows и Mac OSX:

Canopy и Anaconda поставляют новую версию *scikit-learn* в дополнение к большому набору научной библиотеки python для Windows, Mac OSX (также актуальной для Linux).

Обучить классификатор с перекрестной проверкой

Использование набора диафрагмы:

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

Данные разделяются на поезда и тестовые наборы. Для этого мы используем `train_test_split` утилиты `train_test_split` для разделения как `X` и `y` (данных и целевых векторов) случайным образом с опцией `train_size=0.75` (учебные наборы содержат 75% данных).

Учебные наборы данных подаются в классификатор k-ближайших соседей . Метод `fit` классификатора будет соответствовать модели данным.

```
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)
```

Наконец, предсказание качества тестового образца:

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

Используя одну пару поездов и тестовых наборов, мы можем получить предвзятую оценку качества классификатора из-за произвольного выбора разделения данных. Используя *кросс-валидацию,* мы можем поместить классификатор на разные поезда / тестовые подмножества данных и составить среднее значение по всем результатам точности. Функция `cross_val_score` классификатор входным данным с использованием кросс-валидации. Он может принимать в качестве входного количества различных разделов (сгибов), которые будут использоваться (5 в приведенном ниже примере).

```
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)
```

Создание трубопроводов

Поиск шаблонов в данных часто происходит в цепочке шагов обработки данных, например, при выборе, нормализации и классификации. В `sklearn` для этого используется трубопровод этапов.

Например, следующий код показывает конвейер, состоящий из двух этапов. Первый масштабирует функции, а второй обучает классификатор в полученном расширенном наборе данных:

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

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

После создания конвейера вы можете использовать его как обычный этап (в зависимости от его конкретных шагов). Здесь, например, конвейер ведет себя как классификатор. Следовательно, мы можем использовать его следующим образом:

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

Интерфейсы и условные обозначения:

Различные операции с данными выполняются с использованием специальных классов.

Большинство классов относятся к одной из следующих групп:

- алгоритмы классификации (полученные из `sklearn.base.ClassifierMixin`) для решения задач классификации
- регрессионные алгоритмы (полученные от `sklearn.base.RegressorMixin`) для решения проблемы восстановления непрерывных переменных (проблема регрессии)
- преобразование данных (полученное из `sklearn.base.TransformerMixin`), которое препроцессор данных

Данные хранятся в `numpy.array` s (но другие `pandas.DataFrame` объекты, такие как `pandas.DataFrame` s, принимаются, если они конвертируются в `numpy.array` s)

Каждый объект в данных описывается набором функций, общее соглашение состоит в том, что образец данных представлен массивом, где первое измерение является идентификатором выборки данных, второе измерение - идентификатором функции.

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

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

В `sklearn` выше наборе данных `sklearn` содержит 5 объектов, каждый из которых описывается двумя функциями.

Примеры наборов данных

Для удобства тестирования `sklearn` предоставляет некоторые встроенные наборы данных в модуле `sklearn.datasets` . Например, давайте загрузим набор данных диафрагмы Fisher:

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

Вы можете прочитать полное описание, имена функций и имена классов (`target_names`). Они хранятся в виде строк.

Нам интересны данные и классы, которые хранятся в `data` и `target` полях. По соглашению они обозначаются как `X` и `y`

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

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

Формы `X` и `y` говорят, что есть 150 образцов с четырьмя функциями. Каждый образец относится к одному из следующих классов: 0, 1 или 2.

Теперь `X` и `y` можно использовать для обучения классификатора, вызывая метод `fit()` классификатора.

Вот полный список наборов данных, предоставляемых модулем `sklearn.datasets` с их размером и предполагаемым использованием:

Загрузите	Описание	Размер	использование
`load_boston()`	Набор данных о ценах на жилье в Бостоне	506	регрессия
`load_breast_cancer()`	Рак молочной железы Висконсин	569	классификация (двоичная)
`load_diabetes()`	Набор данных диабета	442	регрессия
`load_digits(n_class)`	Цифры данных	1797	классификация
`load_iris()`	Набор данных Iris	150	классификация (многоклассовый)

Загрузите	Описание	Размер	использование
`load_linnerud()`	Набор данных Linnerud	20	многомерная регрессия

Обратите внимание, что (источник: http://scikit-learn.org/stable/datasets/) :

> Эти наборы данных полезны, чтобы быстро проиллюстрировать поведение различных алгоритмов, реализованных в scikit. Однако они часто слишком малы, чтобы быть репрезентативными для реальных задач машинного обучения.

В дополнение к этим встроенным наборам данных для образцов игрушек, `sklearn.datasets` также предоставляет служебные функции для загрузки внешних наборов данных:

- `load_mlcomp` для загрузки примерных наборов данных из репозитория mlcomp.org (обратите внимание, что наборы данных необходимо загрузить до этого). Вот пример использования.
- `fetch_lfw_pairs` и `fetch_lfw_people` для загрузки набора данных с маркированными лицами в Wild (LFW) из http://vis-www.cs.umass.edu/lfw/ , которые используются для проверки лица (соответственно распознавания лиц). Этот набор данных больше 200 МБ. Вот пример использования.

Прочитайте Начало работы с scikit-learn онлайн: https://riptutorial.com/ru/scikit-learn/topic/1035/начало-работы-с-scikit-learn

глава 2: Выбор модели

Examples

Перекрестная проверка

Изучение параметров функции прогнозирования и тестирование ее на одних и тех же данных является методологической ошибкой: модель, которая только что повторила бы метки образцов, которые она только что увидела, имела бы идеальный результат, но не смогла бы предсказать ничего полезного, невидимые данные. Эта ситуация называется **переопределением** . Чтобы избежать этого, обычно при проведении (контролируемого) **теста** машинного обучения удерживать часть доступных данных в качестве **тестового набора** `X_test, y_test` . Обратите внимание, что слово «эксперимент» не предназначено для обозначения академического использования, потому что даже в коммерческих условиях машинное обучение обычно начинается экспериментально.

В scikit-learn случайное разделение на тренировки и тестовые наборы можно быстро вычислить с помощью вспомогательной функции train_test_split . Давайте загрузим набор данных диафрагмы, чтобы он соответствовал линейной машине поддержки вектора:

```
>>> 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,))
```

Теперь мы можем быстро опробовать учебный набор, одновременно проведя 40% данных для тестирования (оценки) нашего классификатора:

```
>>> 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,))
```

Теперь, после того, как у нас есть тренировочные и тестовые наборы, давайте использовать его:

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

Проверка K-Fold Cross

Кросс-валидация K-fold - это систематический процесс повторения процедуры разделения поездов / испытаний несколько раз, чтобы уменьшить дисперсию, связанную с одним испытанием разделения поездов / испытаний. Вы по существу разделяете весь набор данных на K равными размерами «складки», и каждая сводка используется один раз для тестирования модели и K-1 раз для обучения модели.

В библиотеке scikit имеется несколько методов сгибания. Их использование зависит от характеристик входных данных. Некоторые примеры

K-Fold

Вы по существу разделяете весь набор данных на K равными размерами «складки», и каждая сводка используется один раз для тестирования модели и K-1 раз для обучения модели.

```
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` - это вариация k-fold, которая возвращает слоистые складки: каждый набор содержит примерно одинаковый процент выборок каждого целевого класса в качестве набора

ShuffleSplit

Используется для генерации определенного пользователем количества независимых разделов набора данных поезда / теста. Образцы сначала перетасовываются, а затем разбиваются на пару поездов и испытательных комплектов.

```
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` - это вариация ShuffleSplit, которая возвращает слоистые расщепления, то есть создает разделители, сохраняя одинаковый процент для каждого целевого класса, как в полном наборе.

Другие методы складывания, такие как Leave One / p Out и TimeSeriesSplit (вариация K-fold) доступны в библиотеке scicit model_selection.

Прочитайте Выбор модели онлайн: https://riptutorial.com/ru/scikit-learn/topic/4901/выбор-модели

глава 3: Выбор функции

Examples

Удаление функции с небольшим разбросом

Это очень простой метод выбора объектов.

Его основная идея заключается в том, что если функция постоянна (т. Е. Имеет 0 дисперсию), то она не может быть использована для поиска каких-либо интересных шаблонов и может быть удалена из набора данных.

Следовательно, эвристический подход к устранению признаков состоит в том, чтобы сначала удалить все функции, отклонения которых ниже некоторого (низкого) порога.

Исходя из примера в документации , предположим, что мы начинаем с

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

Здесь имеется *3* булевых функции, каждая из которых имеет *6* экземпляров. Предположим, мы хотим удалить те, которые являются постоянными, по крайней мере, в 80% случаев. Некоторые вероятностные вычисления показывают, что эти функции должны иметь дисперсию ниже *0,8 * (1 - 0,8)* . Следовательно, мы можем использовать

```
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]])
```

Обратите внимание, как первая функция была удалена.

Этот метод следует использовать с осторожностью, поскольку низкая дисперсия не обязательно означает, что функция «неинтересна». Рассмотрим следующий пример, где мы строим набор данных, который содержит *3* функции, первые два из которых состоят из случайно распределенных переменных и третьей равномерно распределенных переменных.

```
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 ]])
```

Теперь первая функция была удалена из-за ее низкой дисперсии, в то время как третья функция (то есть самая неинтересная) была сохранена. В этом случае было бы уместнее рассмотреть *коэффициент вариации,* поскольку он не зависит от масштабирования.

Прочитайте Выбор функции онлайн: https://riptutorial.com/ru/scikit-learn/topic/4909/выбор-функции

глава 4: классификация

Examples

Использование векторных машин поддержки

Поддерживающие векторные машины - это семейство алгоритмов, пытающихся передать (возможно, высокоразмерную) гиперплоскость между двумя помеченными наборами точек, так что расстояние точек от плоскости в каком-то смысле является оптимальным. SVM могут использоваться для классификации или регрессии (соответствующие `sklearn.svm.SVC` и `sklearn.svm.SVR` , соответственно.

Пример:

Предположим, что мы работаем в двумерном пространстве. Во-первых, мы создаем некоторые данные:

```
import numpy as np
```

Теперь мы создаем *x* и *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
```

Заметим, что *x* состоит из двух гауссианов: один с центром вокруг *(0, 0)* и один с центром вокруг *(1, 1)* .

Чтобы построить классификатор, мы можем использовать:

```
from sklearn import svm

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

Давайте проверим прогноз для *(0, 0)* :

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

Предсказание состоит в том, что класс равен 0.

Для регрессии мы можем аналогичным образом:

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

RandomForestClassifier

Случайный лес является мета-оценкой, которая соответствует ряду классификаторов дерева решений на разных подпакетах набора данных и использует усреднение для улучшения точности прогнозирования и контроля над переустановкой.

Простой пример использования:

Импортировать:

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

Определение данных поезда и целевых данных:

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

Значения в `target` представлении представляют собой метку, которую вы хотите предсказать.

Инициируйте объект RandomForest и выполните обучение (подойдет):

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

предсказать:

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

Анализ отчетов о классификации

Создайте текстовый отчет, показывающий основные показатели классификации, включая точность и отзыв , f1-score (среднее значение гармоник точности и отзыва) и поддержку (количество наблюдений этого класса в наборе обучения).

Пример из документов `sklearn` :

```
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))
```

Выход -

```
             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 для классификации. Классификатор усиления градиента представляет собой аддитивный ансамбль базовой модели, ошибка которой исправляется в последовательных итерациях (или этапах) путем добавления деревьев регрессии, которые корректируют остатки (ошибка на предыдущем этапе).

Импортировать:

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

Создайте некоторые данные по классификации игрушек

```
from sklearn.datasets import load_iris

iris_dataset = load_iris()

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

Разделим эти данные на набор для обучения и тестирования.

```
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)
```

Создайте модель `GradientBoostingClassifier` используя параметры по умолчанию.

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

Давайте забьем его на тестовом наборе

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

По умолчанию построено 100 оценок

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

Это можно контролировать, установив `n_estimators` на другое значение во время

инициализации.

Дерево решений

Дерево решений - это классификатор, который использует последовательность подробных правил (например, a> 7), которые могут быть легко поняты.

Приведенный ниже пример обучает классификатор дерева решений с использованием трех векторов признаков длины 3, а затем предсказывает результат для неизвестного четвертого вектора признаков, так называемого тестового вектора.

```
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
```

Выходные данные можно визуализировать, используя:

```
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")
```

Классификация с использованием логистической регрессии

В LR-классификаторе его вероятности, описывающие возможные результаты одного испытания, моделируются с использованием логистической функции. Он реализован в библиотеке `linear_model`

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

Реализация LL sklearn может соответствовать двоичной, однополярной логистической регрессии с одной или несколькими логическими системами с опциональной регуляцией L2 или L1. Например, рассмотрим двоичную классификацию в наборе данных образца

склеронов

```
from sklearn.datasets import make_hastie_10_2

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

Где X - массив `n_samples X 10` а y - метки-мишени -1 или +1.

Используйте разделение поездов для разделения входных данных на тренировочные и тестовые наборы (70% -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)
```

Использование LR-классификатора аналогично другим примерам

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

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

Используйте матрицу Confusion для визуализации результатов

```
from sklearn.metrics import confusion_matrix

confusion_matrix(predicted, labels_test)
```

Прочитайте классификация онлайн: https://riptutorial.com/ru/scikit-learn/topic/2468/классификация

глава 5: регрессия

Examples

Обычные наименьшие квадраты

Обычные наименьшие квадраты - это метод нахождения линейной комбинации признаков, которая наилучшим образом соответствует наблюдаемому результату в следующем смысле.

Если вектором прогнозируемых результатов является y , а объясняющие переменные образуют матрицу X , тогда OLS найдет решение β- вектора

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

где $y\hat{} = X\beta$ - линейное предсказание.

В sklearn это делается с использованием `sklearn.linear_model.LinearRegression` .

Контекст приложения

OLS следует применять только к регрессионным проблемам, обычно это непригодно для проблем классификации: Contrast

- Является ли спам электронной почты? (Classfication)
- Какова линейная зависимость между upvotes зависит от длины ответа? (Регрессия)

пример

Давайте `LinearRegression` линейную модель с некоторым шумом, а затем посмотрим, будет ли `LinearRegression` восстановлением линейной модели.

Сначала мы сгенерируем матрицу `X` :

```
import numpy as np

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

Теперь мы будем генерировать `y` как линейную комбинацию `X` с некоторым шумом:

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

Обратите внимание, что истинная линейная комбинация, генерирующая `y` , задается `beta.

Чтобы попытаться восстановить это только из `X` и `y` , давайте сделаем следующее:

```
>>> linear_model.LinearRegression().fit(x, y).coef_
array([  9.97768469e-01,   9.98237634e-01,   7.55016533e-04])
```

Обратите внимание, что этот вектор очень похож на `beta` .

Прочитайте регрессия онлайн: https://riptutorial.com/ru/scikit-learn/topic/5190/регрессия

глава 6: Уменьшение размерности (выбор функции)

Examples

Сокращение размера с помощью анализа основных компонентов

Анализ основных компонентов находит последовательности линейных комбинаций признаков. Первая линейная комбинация максимизирует дисперсию признаков (при условии ограничения единицы). Каждая из следующих линейных комбинаций максимизирует дисперсию признаков в подпространстве, ортогональном к тому, что натянуто на предыдущие линейные комбинации.

Общий метод понижения размерности является использование только K первых таких линейных комбинаций. Предположим, что функции представляют собой матрицу X из n строк и m столбцов. Первые k линейных комбинаций образуют матрицу β_k из m строк и k столбцов. Продукт $X\beta$ имеет n строк и k столбцов. Таким образом, полученную матрицу βk можно рассматривать как уменьшение от m до k размерностей, сохраняя части с высокой дисперсией исходной матрицы X.

В `scikit-learn` , PCA выполняется с помощью `sklearn.decomposition.PCA` . Например, предположим, что мы начинаем с матрицы 100 X 7, построенной так, чтобы дисперсия содержалась только в первых двух столбцах (путем уменьшения числа последних 5 столбцов):

```
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)))
```

Давайте сделаем сокращение до 2-х измерений:

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

Теперь давайте проверим результаты. Во-первых, вот линейные комбинации:

```
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]])
```

Обратите внимание, что первые два компонента в каждом векторе на несколько порядков больше других, что показывает, что СПС признал, что дисперсия содержится главным образом в первых двух столбцах.

Чтобы проверить соотношение дисперсии, объясненную этим СПС, мы можем рассмотреть `pca.explained_variance_ratio_` :

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

Прочитайте Уменьшение размерности (выбор функции) онлайн: https://riptutorial.com/ru/scikit-learn/topic/4829/уменьшение-размерности--выбор-функции-

глава 7: Эксплуатационная характеристика приемника (ROC)

Examples

Введение в РПЦ и АУК

Пример показателя эксплуатационной характеристики приемника (ROC) для оценки качества выходного сигнала классификатора.

Кривые ROC обычно имеют истинную положительную скорость по оси Y и ложную положительную скорость по оси X. Это означает, что верхний левый угол графика - это «идеальная» точка - ложная положительная норма нуля и истинная положительная скорость одного. Это не очень реалистично, но это означает, что большая площадь под кривой (AUC) обычно лучше.

«Крутизна» кривых ROC также важна, поскольку она идеальна для максимизации истинной положительной скорости при минимизации ложной положительной скорости.

Простой пример:

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

Произвольные значения `y` - в реальном случае это предсказанные целевые значения (`model.predict(x_test)`):

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

Оценки - это средняя точность данных данных и меток (`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])
```

Вычислите кривую ROC и AUC:

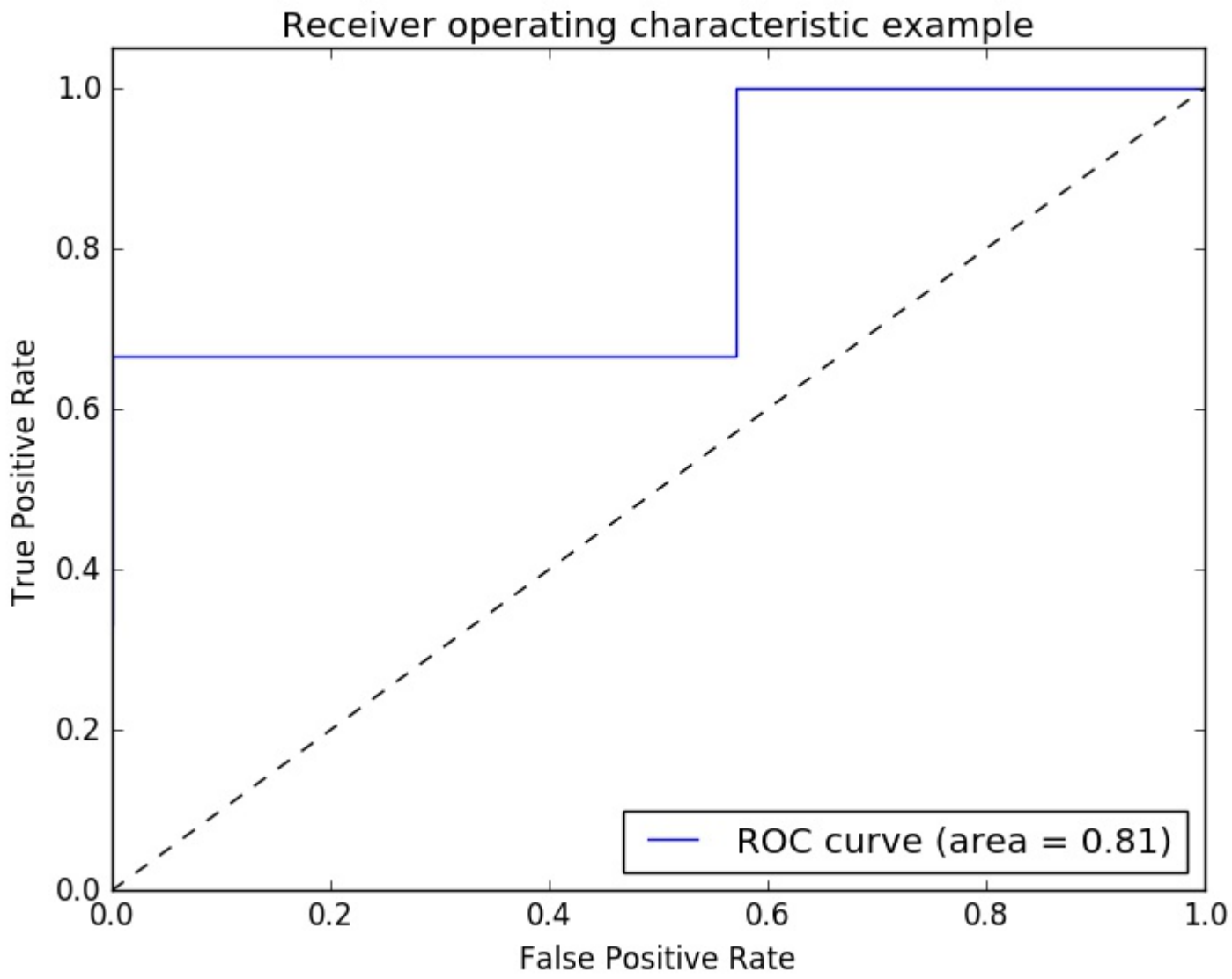
```
fpr, tpr, thresholds = metrics.roc_curve(y, scores, pos_label=2)
roc_auc = metrics.auc(fpr, tpr)
```

Заговор:

```
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()
```

Выход:



Примечание: источники были взяты из этих ссылок1 и link2

Показатель ROC-AUC с переопределением и перекрестной проверкой

Для расчета оценки ROC-AUC (площадь под кривой) нужны предсказанные вероятности. `cross_val_predict` использует методы `predict` классификаторов. Чтобы иметь возможность получить оценку ROC-AUC, можно просто подклассифицировать классификатор, переопределяя метод `predict` , чтобы он действовал как `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)))
```

выход:

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

Прочитайте Эксплуатационная характеристика приемника (ROC) онлайн: https://riptutorial.com/ru/scikit-learn/topic/5945/эксплуатационная-характеристика-приемника--roc-

кредиты

S. No	Главы	Contributors
1	Начало работы с scikit-learn	Alleo, Ami Tavory, Community, Gabe, Gal Dreiman, panty, Sean Easter, user2314737
2	Выбор модели	Gal Dreiman, Mechanic
3	Выбор функции	Ami Tavory, user2314737
4	классификация	Ami Tavory, Drew, Gal Dreiman, hashcode55, Mechanic, Raghav RV, Sean Easter, tfv, user6903745, Wayne Werner
5	регрессия	Ami Tavory, draco_alpine
6	Уменьшение размерности (выбор функции)	Ami Tavory, DataSwede, Gal Dreiman, Sean Easter, user2314737
7	Эксплуатационная характеристика приемника (ROC)	Gal Dreiman, Gorkem Ozkaya