CPSC429/529: Machine Learning

Lecture 3: Intr. to Scikit-Learn

Dongsheng Che
Computer Science Department
East Stroudsburg University

Scikit-Learn

Data Representation in
Scikit-Learn

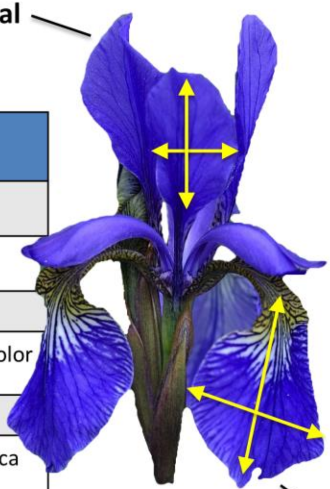
Iris Dataset

Samples
(instances, observations)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

Features
(attributes, measurements, dimensions)

Class labels
(targets)



Load Iris Dataset from Scikit-Learn

```
In [1]: from sklearn.datasets import load_iris

        iris = load_iris()

        print('Data type', type(iris),'\n')
        print('Data attributes: ', dir(iris),'\n')
```

```
Data type <class 'sklearn.utils._bunch.Bunch'>

Data attributes:  ['DESCR', 'data', 'data_module', 'feature_names',
'filename', 'frame', 'target', 'target_names']
```

Bunch data type

data : ***Bunch***

Dictionary-like object, with the following attributes.

data : *{ndarray, dataframe} of shape (150, 4)*

The data matrix. If `as_frame=True`, `data` will be a pandas DataFrame.

target: {ndarray, Series} of shape (150,)

The classification target. If `as_frame=True`, `target` will be a pandas Series.

feature_names: list

The names of the dataset columns.

target_names: list

The names of target classes.

frame: DataFrame of shape (150, 5)

Only present when `as_frame=True`. DataFrame with `data` and `target`.

New in version 0.23.

DESCR: str

The full description of the dataset.

filename: str

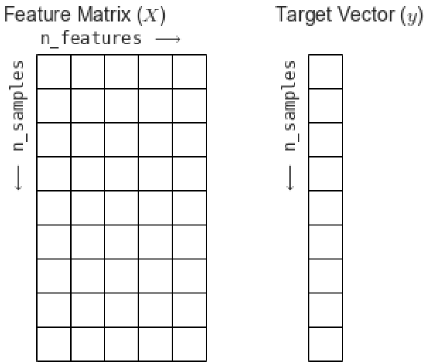
The path to the location of the data.

New in version 0.20.

(data, target) : *tuple if* `return_X_y` *is True*

A tuple of two ndarray. The first containing a 2D array of shape (n_samples, n_features) with each row representing one sample and each column representing the features. The second ndarray of shape (n_samples,) containing the target samples.

Feature Matrix (*X*) and Target Vector (*y*)



Important: If your input x is a vector (i.e., single feature), you need to make x a matrix.

In [8]:
```
X = x[:, np.newaxis]
X.shape
```

Out[8]: (50, 1)

Scikit-Learn

Data Representation in Scikit-Learn

Scikit-Learn

Scikit-Learn's Estimator
API

design principles

- **Consistency**: All objects share a common interface
- **Inspection**: All parameter values are exposed as public attributes.
- **Limited object hierarchy**: Only algorithms are represented by Python classes; datasets are represented in standard formats (NumPy arrays, Pandas DataFrames, SciPy sparse matrices) and parameter names use standard Python strings.
- **Composition**: Many machine learning tasks can be expressed as sequences of more fundamental algorithms, and Scikit-Learn makes use of this wherever possible.
- **Sensible defaults**: When models require user-specified parameters, the library defines an appropriate default value.

Steps of using the API

Step 0:

Split the dataset into training data and test data

Step 1:

Choose a class of model by importing the appropriate estimator class from Scikit-Learn.

Step 2:

Choose model hyperparameters by instantiating this class with desired values.

Step 3:

Arrange data into a features matrix and target vector.

Step 4:

Fit the model to your data by calling the fit() method of the model instance.

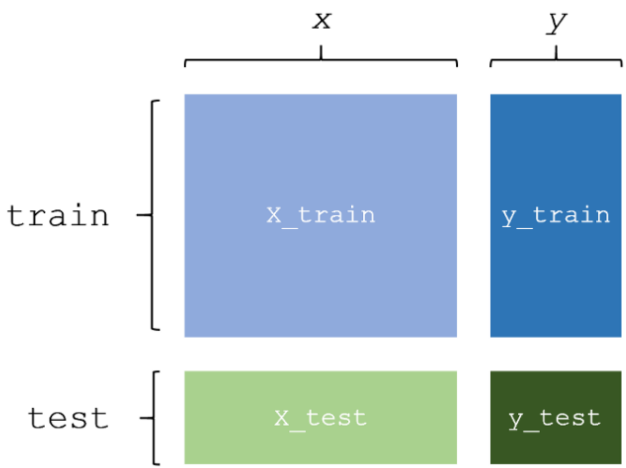
Step 5:

Apply the Model to new data:

- For supervised learning, often we predict labels for unknown data using the predict() method.
- For unsupervised learning, we often transform or infer properties of the data using the transform() or predict() method.

Template of using the API: Step 0

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3)
```



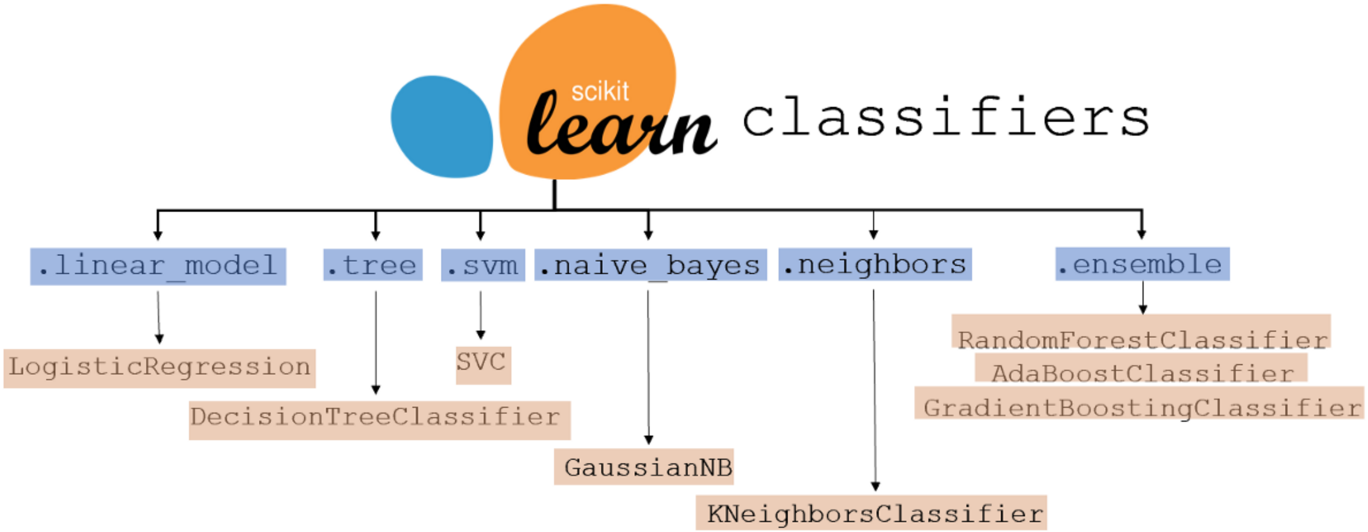
https://towardsdatascience.com/your-ultimate-data-mining-machine-learning-cheat-sheet-9fce3fa16

Template of using the API: Steps 1 - 4

```
#import
from sklearn.branch import model_name

#create instance
model = model_name()

#fit model
model.fit(X_train, y_train)
```

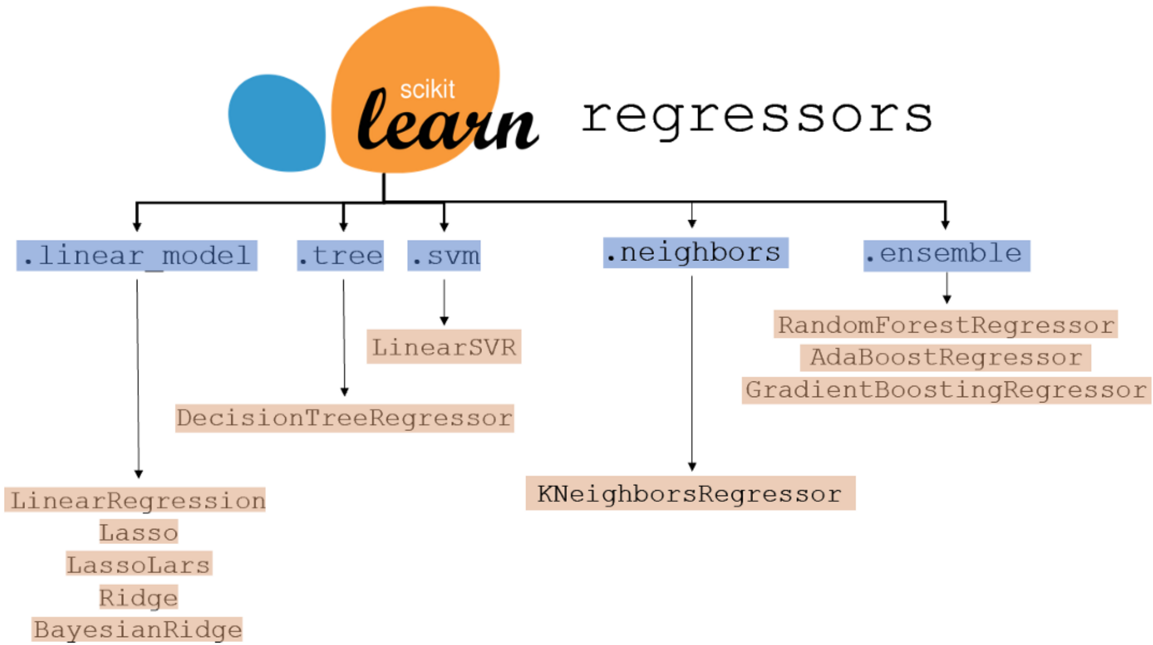
https://towardsdatascience.com/your-ultimate-data-mining-machine-learning-cheat-sheet-9fce3fa16

Scikit-Learn Classifiers



https://towardsdatascience.com/your-ultimate-data-mining-machine-learning-cheat-sheet-9fce3fa16

Scikit-Learn Regressors



https://towardsdatascience.com/your-ultimate-data-mining-machine-learning-cheat-sheet-9fce3fa16

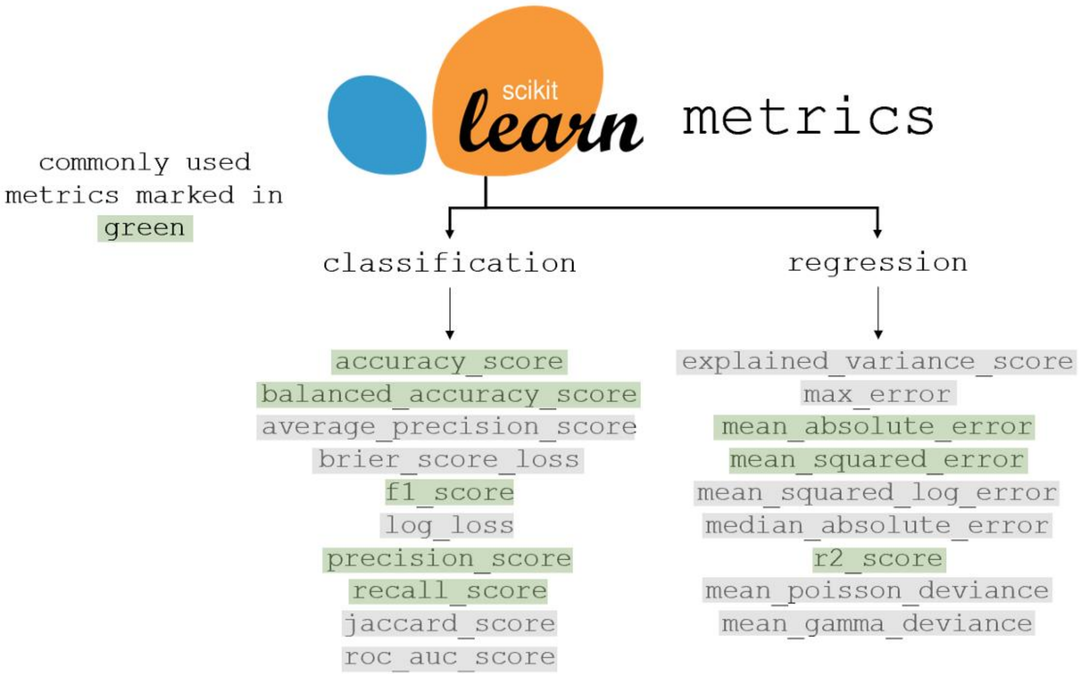
Template of using the API: Step 5

```
#import
from sklearn.metrics import metric_name

#create instance
metric_name(y_test, model.predict(X_test))
```

real target   predicted target

https://towardsdatascience.com/your-ultimate-data-mining-machine-learning-cheat-sheet-9fce3fa16

Scikit-Learn metrics



https://towardsdatascience.com/your-ultimate-data-mining-machine-learning-cheat-sheet-9fce3fa16

Scikit-Learn

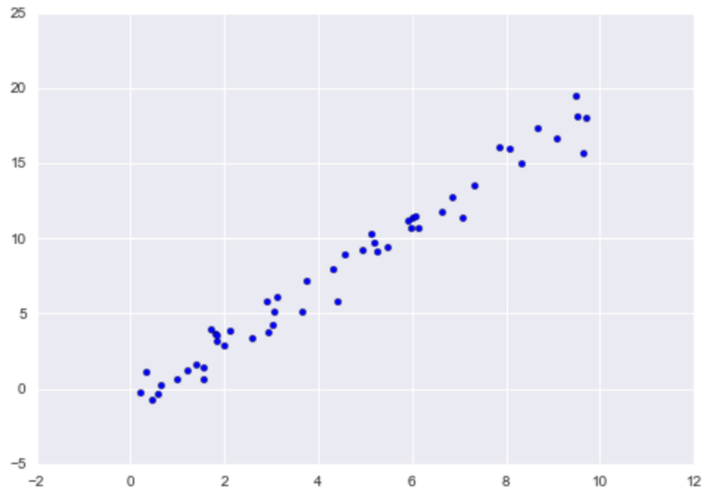
Scikit-Learn's Estimator
API

Scikit-Learn

Supervised Learning
Example: Simple Linear
Regression

Step 0: Dataset

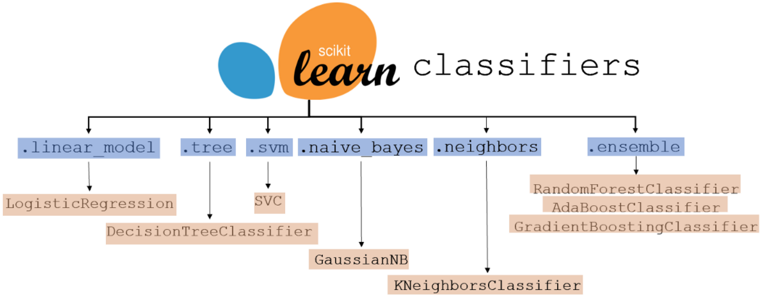
In [5]:
```
import matplotlib.pyplot as plt
import numpy as np

rng = np.random.RandomState(42)
x = 10 * rng.rand(50)
y = 2 * x - 1 + rng.randn(50)
plt.scatter(x, y);
```



Step 1. Choose a class of model

- In Scikit-Learn, every class of model is represented by a Python class.
- If we would like to compute a simple linear regression model, we can import the linear regression class

```
In [6]: from sklearn.linear_model import LinearRegression
```



Step 2. Choose model hyperparameters

- Depending on the model class we are working with, we might need to answer one or more questions like the following:
  - Would we like to fit for the offset (i.e., y-intercept)?
  - Would we like the model to be normalized?
  - Would we like to preprocess our features to add model flexibility?
  - What degree of regularization would we like to use in our model?
  - How many model components would we like to use?
- These choices are often represented as hyperparameters, or parameters that must be set before the model is fit to data

```
In [7]: model = LinearRegression(fit_intercept=True)
        model

Out[7]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Step 3. Arrange data into a features matrix and target vector

- Scikit-Learn requires a two-dimensional features matrix and a one-dimensional target array.
- Our target variable y is already in the correct form (a length-n_samples array),
- We need to make x a matrix of size [n_samples, n_features].

```
In [8]: X = x[:, np.newaxis]
        X.shape

Out[8]: (50, 1)
```

Step 4. Fit the model to your data

- The results of the computations (fit()) are stored in model-specific attributes.
- Model parameters that were learned during the fit() process have trailing underscores;

```
In [9]: model.fit(X, y)

Out[9]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [10]: model.coef_

Out[10]: array([ 1.9776566])

In [11]: model.intercept_

Out[11]: -0.903310725531116535
```

Step 5. Predict labels for unknown data

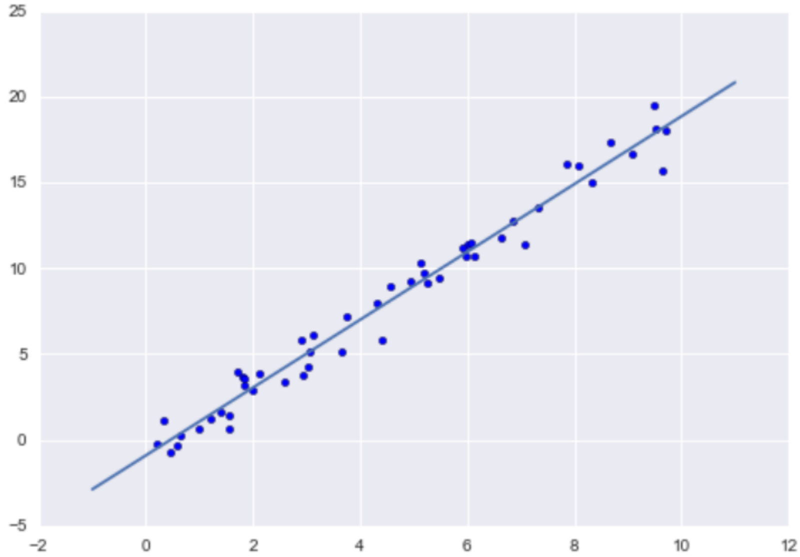
- In Scikit-Learn, this can be done using the predict() method.

In [12]:
```
xfit = np.linspace(-1, 11)
```

In [13]:
```
Xfit = xfit[:, np.newaxis]
yfit = model.predict(Xfit)
```

Plot of the raw data and predicted model

In [14]:
```
plt.scatter(x, y)
plt.plot(xfit, yfit);
```



Scikit-Learn

Supervised Learning
Example: Simple Linear
Regression

Scikit-Learn

More Examples

Supervised learning example: Iris classification

```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X_iris, y_iris,
                                                random_state=2)
```

In [16]:
```
from sklearn.naive_bayes import GaussianNB # 1. choose model class
model = GaussianNB()                       # 2. instantiate model
model.fit(Xtrain, ytrain)                  # 3. fit model to data
y_model = model.predict(Xtest)             # 4. predict on new data
```

In [17]:
```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

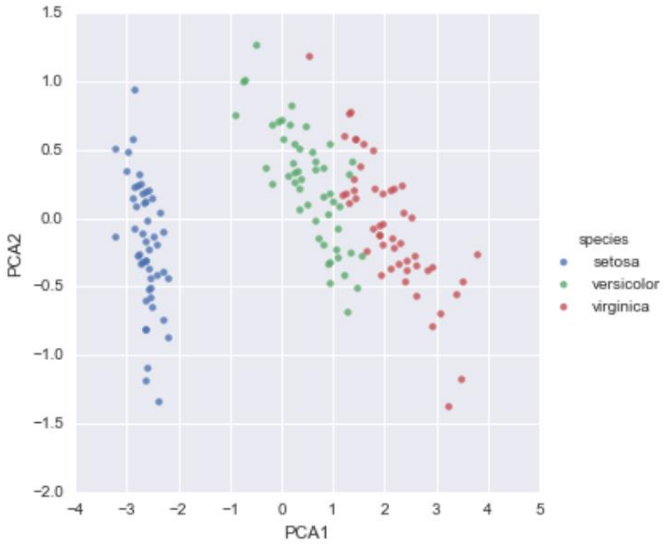
Out[17]: 0.9736842105263158

Unsupervised learning example: Iris dimensionality (1)

In [18]:
```
from sklearn.decomposition import PCA  # 1. Choose the model class
model = PCA(n_components=2)            # 2. Instantiate the model with hyper
model.fit(X_iris)                      # 3. Fit to data. Notice y is not spe
X_2D = model.transform(X_iris)         # 4. Transform the data to two dimens
```

Unsupervised learning example: Iris dimensionality (2)

In [19]:
```
iris['PCA1'] = X_2D[:, 0]
iris['PCA2'] = X_2D[:, 1]
sns.lmplot("PCA1", "PCA2", hue='species', data=iris, fit_reg=False);
```

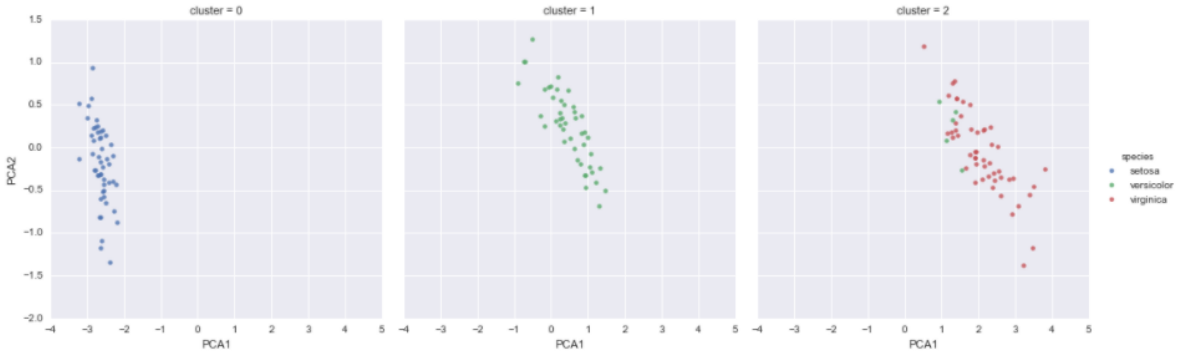


Unsupervised learning example: Iris clustering (1)

In [20]:
```
from sklearn.mixture import GMM      # 1. Choose the model class
model = GMM(n_components=3,
            covariance_type='full')  # 2. Instantiate the model with hyperpa
model.fit(X_iris)                    # 3. Fit to data. Notice y is not speci
y_gmm = model.predict(X_iris)        # 4. Determine cluster labels
```

Unsupervised learning example: Iris clustering (2)

In [21]:
```
iris['cluster'] = y_gmm
sns.lmplot("PCA1", "PCA2", data=iris, hue='species',
           col='cluster', fit_reg=False);
```



Scikit-Learn

More Examples

Scikit-Learn

Other Resources

scikit-learn: Python Machine Learning Library

- **scikit-learn Homepage**
  http://scikit-learn.org/
- **scikit-learn User Guide**
  http://scikit-learn.org/stable/user_guide.html
- **scikit-learn API reference**
  http://scikit-learn.org/stable/modules/classes.html
- **In Python, we typically import classes and functions we need like this:**

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test= train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

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

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning

```
>>> lr.fit(X, y)
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

Fit the model to the data

Unsupervised Learning

```
>>> k_means.fit(X_train)
>>> pca_model = pca.fit_transform(X_train)
```

Fit the model to the data

Fit to data, then transform it

Prediction

Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((2,5)))
>>> y_pred = lr.predict(X_test)
>>> y_pred = knn.predict_proba(X_test)
```

Predict labels

Predict labels

Estimate probability of a label

Unsupervised Estimators

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

Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Estimator score method

Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

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

Homogeneity

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

V-measure

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

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

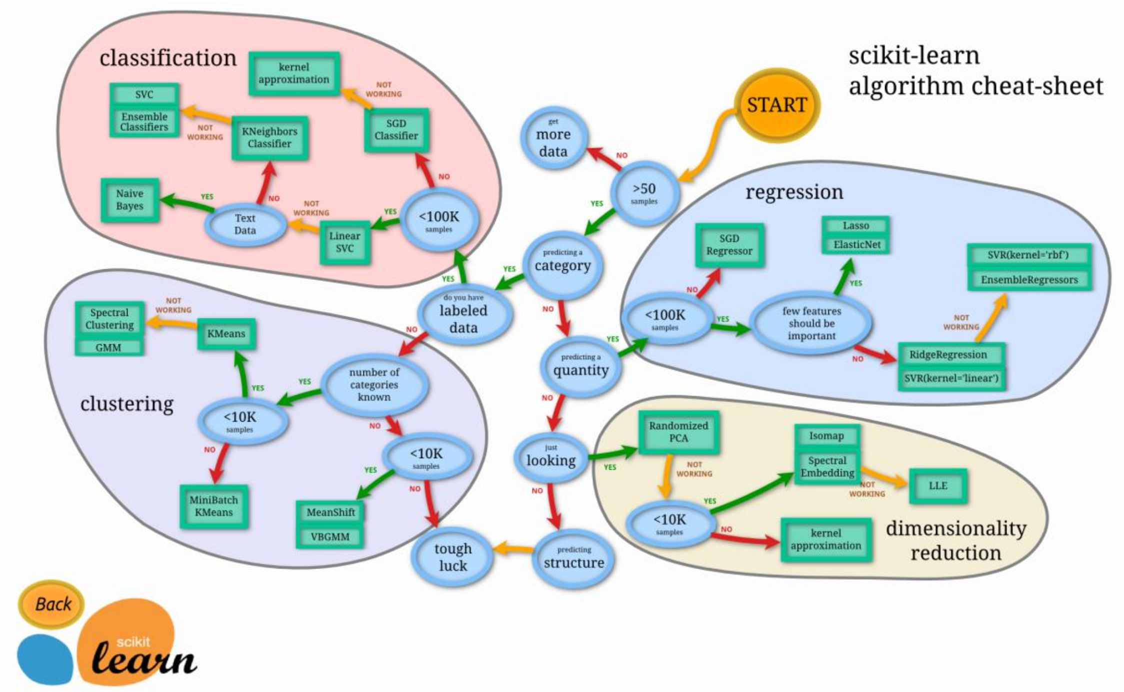
Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

Choosing the right estimator



Scikit-Learn

Other Resources