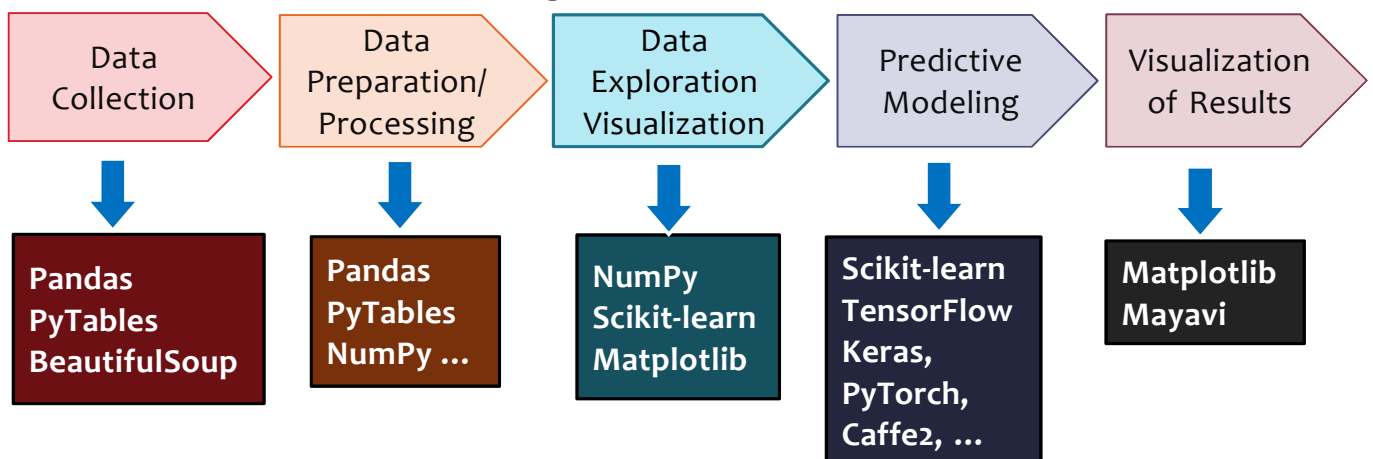


MACHINE LEARNING

Data Analysis Process

- All these activities can be grouped as ...



What Is Machine Learning?

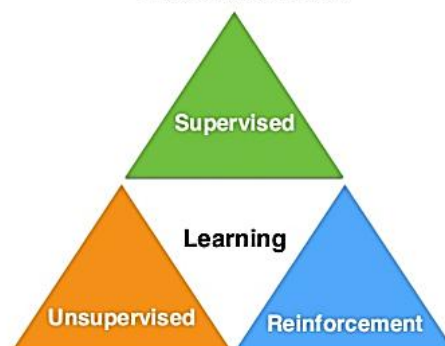
3

- Machine learning is often categorized as a subfield of artificial intelligence.
- In the data science application of machine learning methods, it's more helpful to think of machine learning as a means of *building models of data*.
- Fundamentally, machine learning involves building mathematical models to help understand data.

Categories of Machine Learning

4

- Supervised learning
 - Labeled data
 - Direct feedback
 - Predict outcome/future
- Unsupervised learning
- Reinforcement Learning



- No labels
- No feedback
- “Find hidden structure”
- Decision process
- Reward system
- Learn series of actions

Supervised Learning

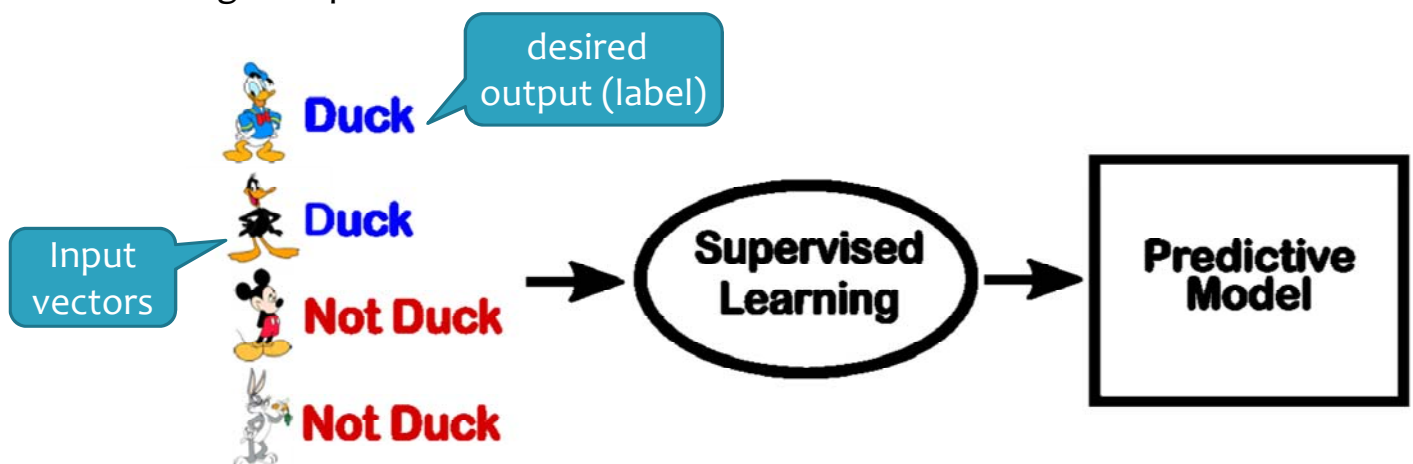
5

- Supervised learning
 - ▣ The training data consist of a set of *training examples*.
 - ▣ Each example is a *pair* consisting of an input object (typically a vector) and a desired output.
 - ▣ A supervised learning algorithm analyzes the training data and produces an inferred model.
 - ▣ Classification problems
 - Convolutional Neural Networks are a great example of this, as the images are the inputs and the outputs are the classifications of the images (dog, cat, etc).

Supervised Learning

6

training data pairs



Unsupervised Learning

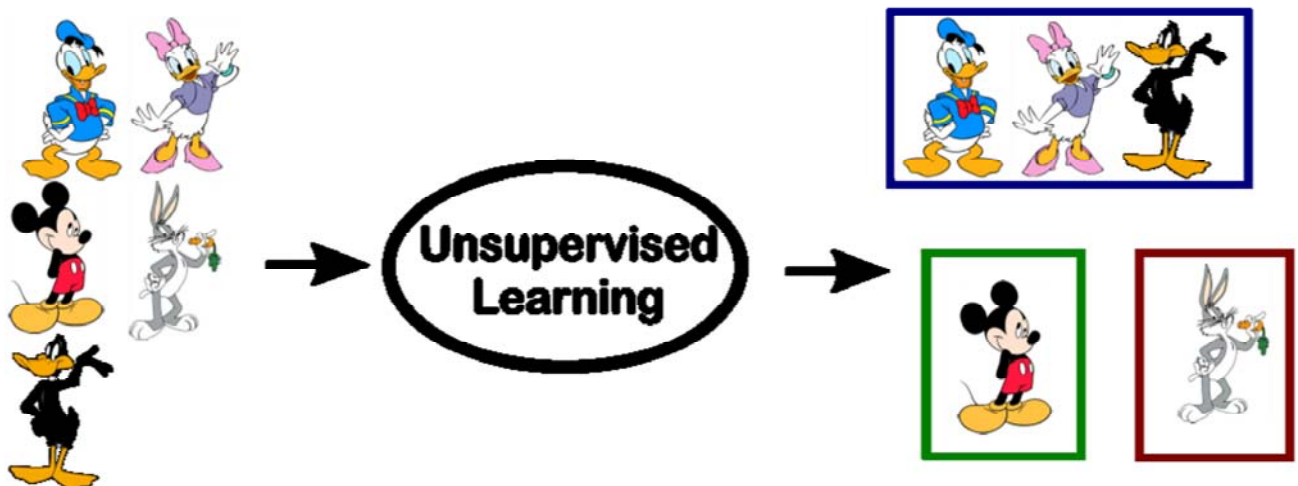
7

- Unsupervised learning
 - ▣ Training data consist of a set of input vectors x without any corresponding target values.
 - ▣ The goal in such unsupervised learning problems is to discover groups of similar examples within the data
 - ▣ Clustering problems
 - K-Means, is an example of unsupervised learning.

Unsupervised learning

8

It can be regarded as a clustering problem.



Reinforcement Learning

9

- Reinforcement Learning
 - ▣ RL is the task of learning what actions to take, given a certain situation/environment, so as to maximize a reward signal.
 - ▣ This reward signal simply tells you whether the action (or input) that the agent takes is good or bad.
 - ▣ It doesn't tell you anything about what the *best* action is.
 - ▣ Another unique component of RL is that an agent's actions will affect the subsequent data it receives.
 - For example, an agent's action of moving left instead of right means that the agent will receive different input from the environment at the next time step.

Reinforcement Learning

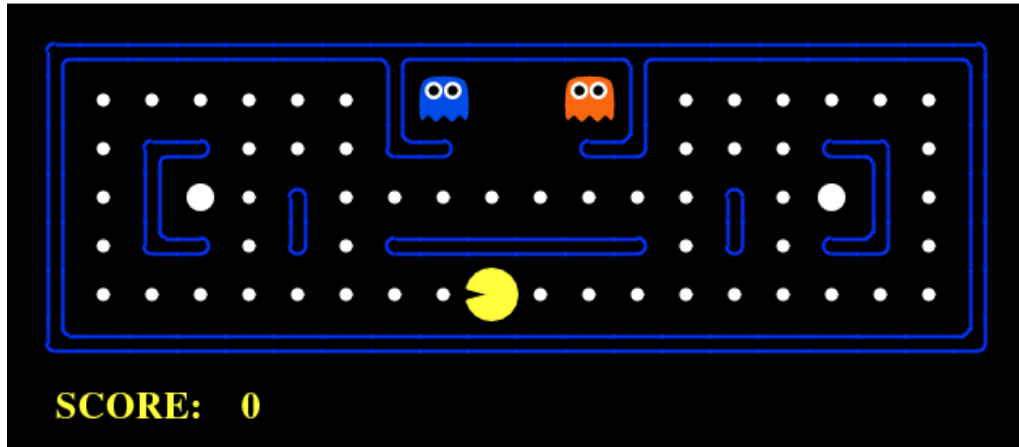
10

- The major components of reinforcement learning
 - ▣ **Set of Environment States** : Such as the different states in the game at a point in time.
 - ▣ **Set of Actions** :Such as Up , Down ,Left ,Right and a Fire button.
 - ▣ **Rules of transitioning between states** : We need to keep track of the best next state we can go to.
 - ▣ **Rules that determine the *scalar immediate reward* of a transition:** For every transition that the algorithm decides to take there is an associated reward associated with that step.
 - For example when you kill an opponent you get a positive reward and when you get hurt you get a negative reward.

Reinforcement Learning

11

PacMan game



12

INTRODUCING SCIKIT-LEARN

Introducing Scikit-Learn

13

- There are several Python libraries that provide solid implementations of a range of machine learning algorithms.
- One of the best known is [Scikit-Learn](#), a package that provides efficient versions of a large number of common algorithms.
- <http://scikit-learn.org/stable/>

Introducing Scikit-Learn

14

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.

Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning

Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.

Modules: preprocessing, feature extraction. — Examples

Data Representation in Scikit-Learn

15

□ Data as table

- For example, consider the Iris (鳶尾花) dataset, famously analyzed by Ronald Fisher in 1936. We can download this dataset in the form of a Pandas DataFrame using the [Seaborn](#) library:

□ Seaborn

- Seaborn is a Python visualization library based on matplotlib.
- It provides a high-level interface for drawing attractive statistical graphics.
- <https://seaborn.pydata.org/>

Iris (鳶尾花)



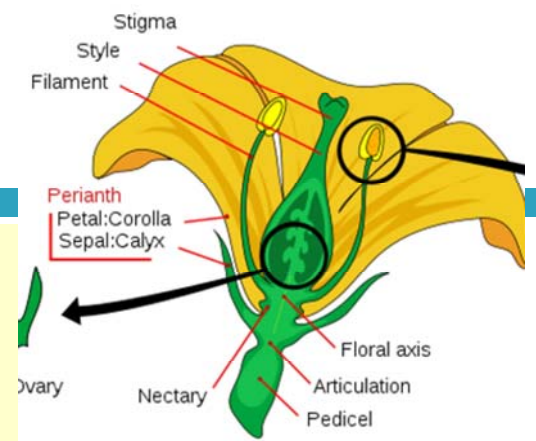
Iris dataset

16

```
import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```

Out[1]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa



Data Representation

17

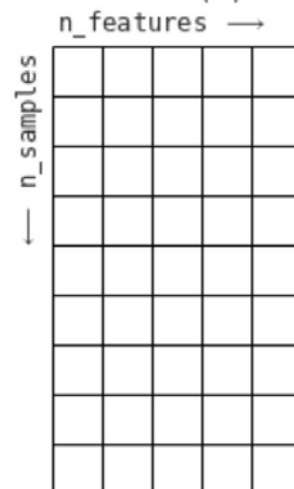
- In general, we will refer to the rows of the matrix as *samples*, and the number of rows as $n_samples$.
- We refer to the columns of the matrix as *features*, and the number of columns as $n_features$.
- **Features matrix**
 - ▣ The features matrix is assumed to be two-dimensional, with shape $[n_samples, n_features]$

Data Representation

18

- **Target array**
 - ▣ The target array is usually one dimensional, with length $n_samples$.
 - ▣ It represents the desired output (label).
 - ▣ It is usually the quantity we want to *predict from the data*: in statistical terms, it is the dependent variable.
 - ▣ In the iris case, the species column would be considered the target.

Feature Matrix (X)



Target Vector (y)

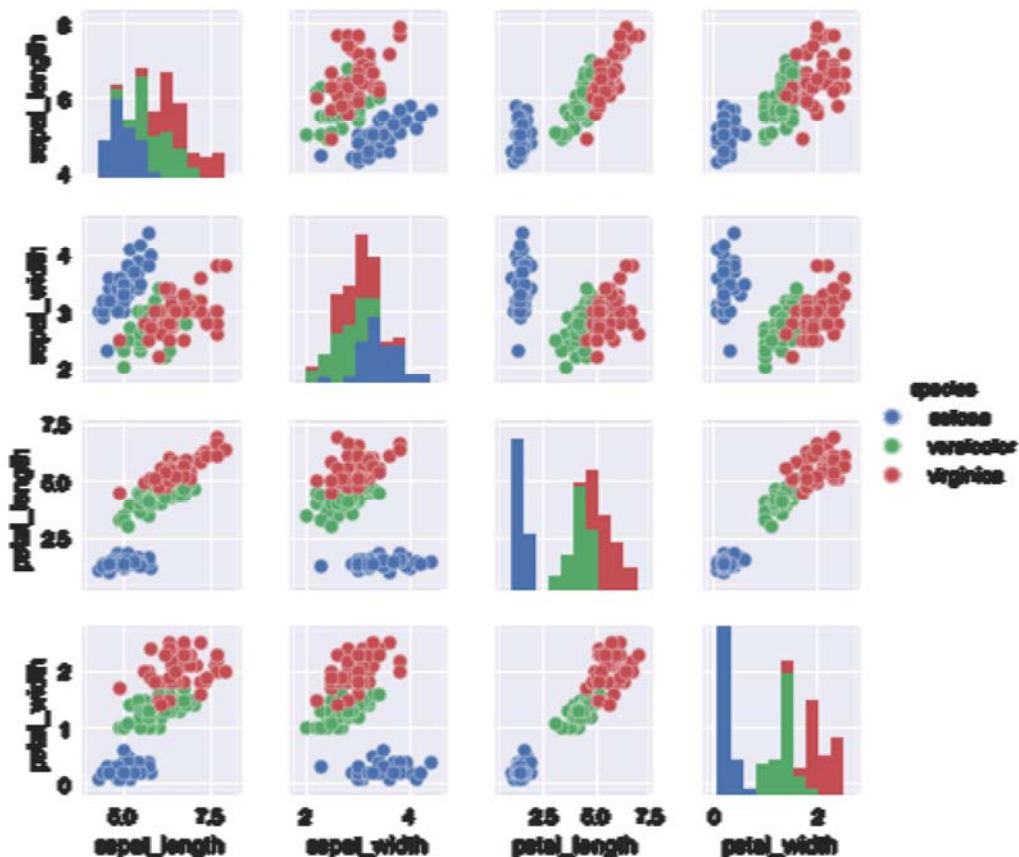


Data Representation

19

```
%matplotlib inline
import seaborn as sns;
sns.set()
sns.pairplot(iris, hue='species', size=1.5);
```

- `seaborn.set(context='notebook', style='darkgrid', palette='deep', font='sans-serif', font_scale=1, color_codes=False, rc=None)`
 - ▣ Set aesthetic parameters in one step.
 - ▣ <https://seaborn.pydata.org/generated/seaborn.set.html#seaborn.set>



Data Representation

21

- For use in Scikit-Learn, we will extract the features matrix and target array from the DataFrame

```
X_iris = iris.drop('species', axis=1)
X_iris.shape
Out[11]: (150, 4)
```

```
y_iris = iris['species']
y_iris.shape
Out[13]: (150,)
```

Steps in using the Scikit-Learn

22

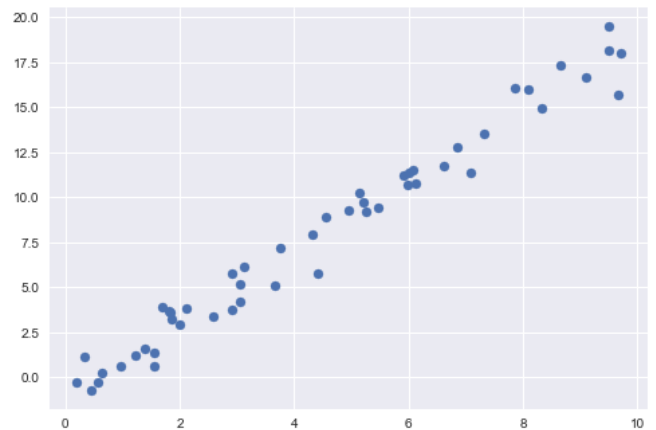
1. Choose a class of model from Scikit-Learn.
2. Choose model parameters.
3. Arrange data into a features matrix and target vector.
4. Fit the model to your data by calling the `fit()` method.
5. Apply the model to new data:
 - For supervised learning, often we predict labels for unknown data using the `predict()` method.
 - For unsupervised learning, we often transform or infer properties of the data using the `transform()` or `predict()` method.

Supervised learning example

23

- Simple linear regression
 - ▣ The common case of fitting a line to (x, y) data.

```
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)
```



Supervised learning example

24

- Step 1. Choose a class of model
 - ▣ In Scikit-Learn, every class of model is represented by a Python class. We can import the linear regression class:

```
from sklearn.linear_model import LinearRegression
```

Supervised learning example

25

- Step 2. Choose model parameters.
 - ▣ For our linear regression example, we can instantiate the LinearRegression class and specify that we would like to fit the intercept using the fit_intercept parameter:

```
model = LinearRegression(fit_intercept=True)
model
```

```
Out[17]: LinearRegression(copy_X=True, fit_intercept=True,
n_jobs=1, normalize=False)
```

Supervised learning example

26

- Step 3. Arrange data into a features matrix and target vector.
 - ▣ We need to make the data x to a matrix of size [n_samples, n_features].

```
x
Out[18]:
array([ 3.74540119,  9.50714306, ...
```

```
X = x[:, np.newaxis]
X.Shape
Out[22]: (50, 1)
```

- numpy.newaxis
The newaxis object can be used in all slicing operations to create an axis of length one.

Supervised learning example

27

- Step 4. Fit the model to your data

```
model.fit(X, y)
Out[24]: LinearRegression(copy_X=True, fit_intercept=True,
n_jobs=1, normalize=False)
```

- This `fit()` command causes a number of model-dependent internal computations to take place, and the results of these computations are stored in model-specific attributes that the user can explore.

Supervised learning example

28

- Step 4. Fit the model to your data

- In Scikit-Learn, all model parameters that were learned during the `fit()` process have trailing underscores; for example, in this linear model, we have the following:

```
model.coef_
Out[25]: array([ 1.9776566])

model.intercept_
Out[26]: -0.90331072553111635
```

These two parameters represent the slope and intercept of the simple linear fit to the data.

Supervised learning example

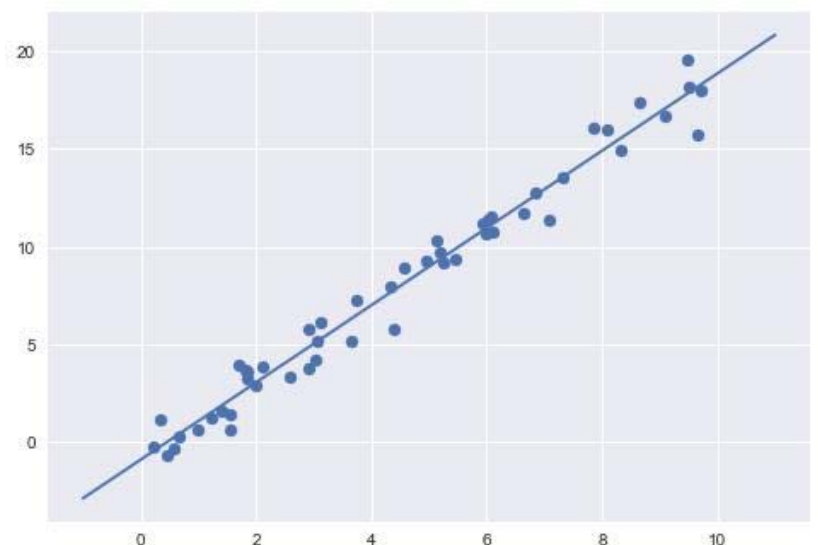
29

- Step 5. Predict labels for unknown data
 - ▣ Once the model is trained, the main task of supervised machine learning is to evaluate it based on what it says about new data that was not part of the training set.
 - ▣ In Scikit-Learn, we can do this using the `predict()` method.

Supervised learning example

30

```
xfit = np.linspace(-1, 11)  
Xfit = xfit[:, np.newaxis]  
yfit = model.predict(Xfit)  
plt.scatter(x, y)  
plt.plot(xfit, yfit);
```



Supervised learning example: Iris classification

31

- We will use [Gaussian naive Bayes](#), which proceeds by assuming each class is drawn from a Gaussian distribution.
- Because it is so fast and has no parameters to choose, Gaussian naive Bayes is often a good model to use as a baseline classification, before you explore whether improvements can be found through more sophisticated models.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

Likelihood (points to $P(x|c)$)
Class Prior Probability (points to $P(c)$)
Posterior Probability (points to $P(c|x)$)
Predictor Prior Probability (points to $P(x)$)

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

Supervised learning example: Iris classification

32

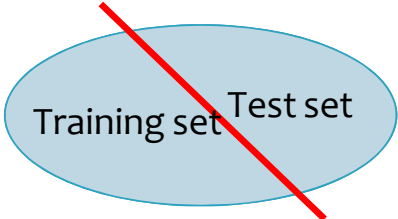
- We would like to evaluate the model on data it has not seen before, and so we will split the data into a *training set* and a *testing set*.
 - ▣ This could be done by hand, but it is more convenient to use the [train_test_split](#) utility function:

Supervised learning example: Iris classification

33

- `sklearn.model_selection.train_test_split(*arrays, **options)`
 - Split arrays or matrices into random train and test subsets
 - **shuffle** : boolean, optional (default=True)
 - **random_state** : int, RandomState instance or None, optional (default=None)
 - http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X_iris,
                                              y_iris, random_state=1)
```



Training set Test set

Supervised learning example: Iris classification

34

```
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
```

- Finally, we can use the `accuracy_score` utility to see the fraction of predicted labels that match their true value:

```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
# or acc = model.score(Xtest, ytest)
Out[38]: 0.97368421052631582
```

With an accuracy topping 97%, we see that even this very naive classification algorithm is effective for this particular dataset!

Unsupervised learning example: Iris dimensionality

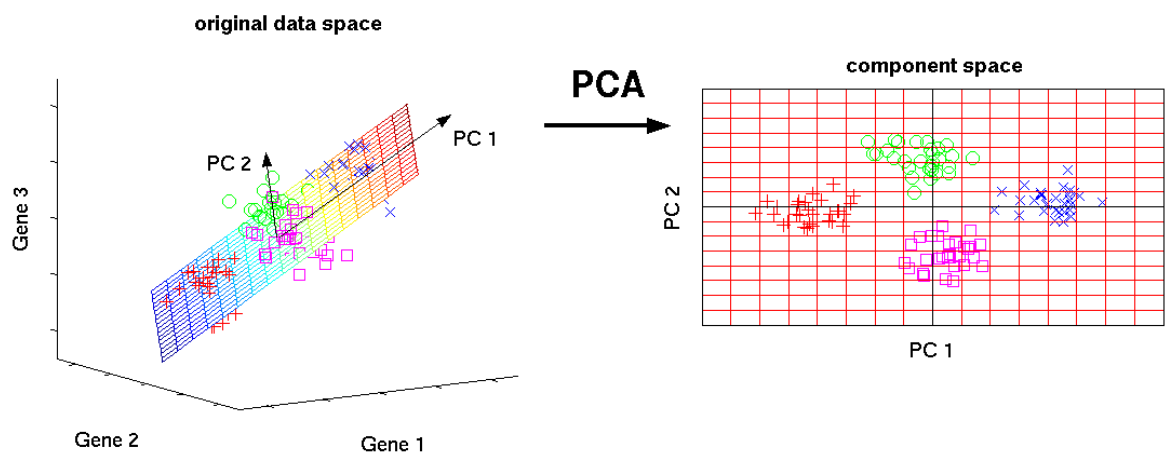
35

- Let's take a look at reducing the dimensionality of the Iris data so as to more easily visualize it.
- Recall that the Iris data is four dimensional: there are four features recorded for each sample.
- Often dimensionality reduction is used as an aid to visualizing data; after all, it is much easier to plot data in two dimensions than in four dimensions or higher!

Unsupervised learning example: Iris dimensionality

36

- Principal component analysis (PCA): It is a fast linear dimensionality reduction technique.



Unsupervised learning example: Iris dimensionality

37

```
import seaborn as sns
iris = sns.load_dataset('iris')
X_iris = iris.drop('species', axis=1)
y_iris = iris['species']
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X_iris, y_iris,
random_state=1)

from sklearn.decomposition import PCA # 1. Choose the model class
model = PCA(n_components=2) # 2. Choose the model parameters
model.fit(X_iris) # 3. Fit to data. Notice y is not specified!
X_2D = model.transform(X_iris) # 4. Transform the data to two dimensions
```

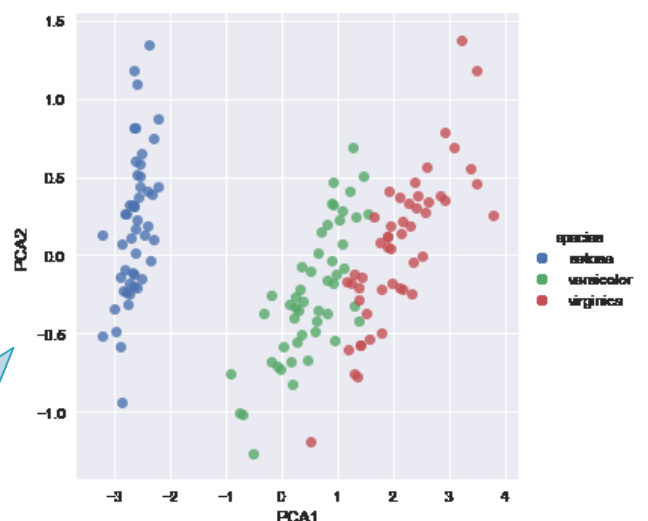
Unsupervised learning example: Iris dimensionality

38

- Now use Seaborn's Implot to show the results

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

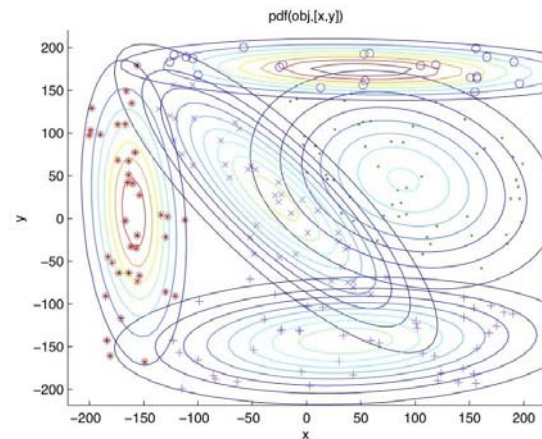
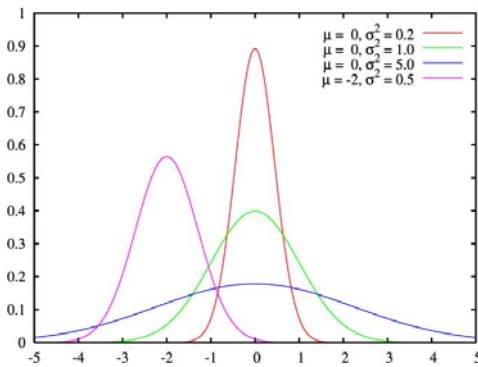
We see that in the two-dimensional representation, the species are fairly well separated.



Unsupervised learning: Iris clustering

39

- A clustering algorithm attempts to find distinct groups of data without reference to any labels.
- Here we will use a powerful clustering method called a Gaussian mixture model (GMM).



Unsupervised learning: Iris clustering

40

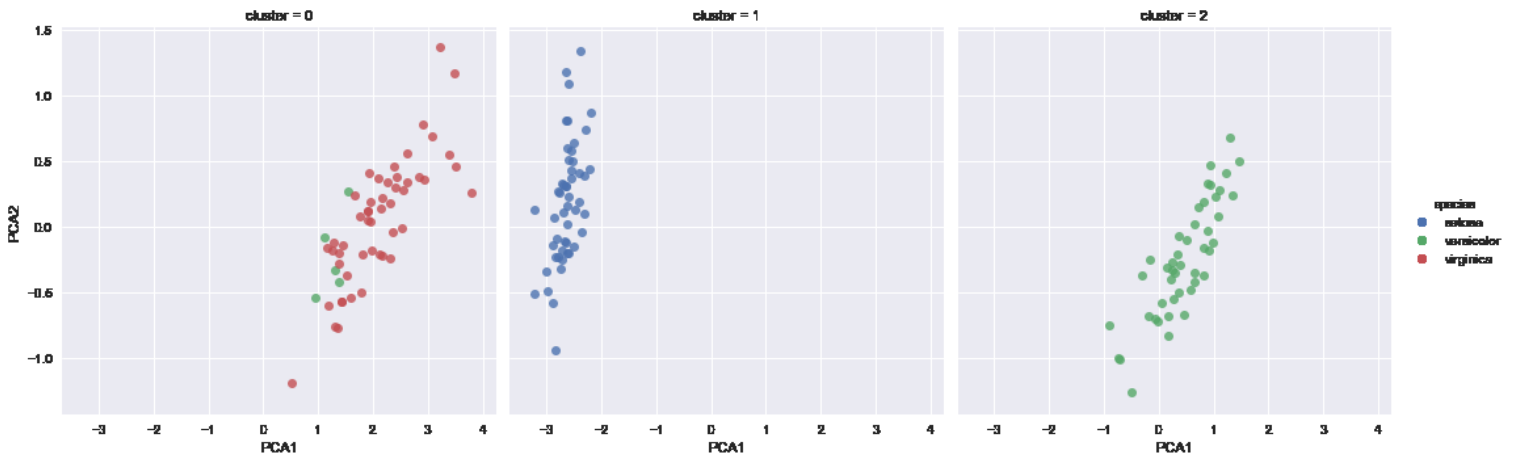
```
# 1. Choose the model class
from sklearn.mixture import GaussianMixture
# 2. Choose model parameters
model = GaussianMixture(n_components=3, covariance_type='full')
# 3. Fit to data. Notice y is not specified!
model.fit(X_iris)
# 4. Determine cluster labels
y_gmm = model.predict(X_iris)

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

Unsupervised learning: Iris clustering

41

□ Plot the results



APPLICATION: EXPLORING HANDWRITTEN DIGITS

Exploring Handwritten Digits

44

- Let's consider one piece of the optical character recognition problem:
 - ▣ The identification of handwritten digits.
 - ▣ In the wild, this problem involves both locating and identifying characters in an image.

Loading and visualizing the digits data

45

```
from sklearn.datasets import load_digits
digits = load_digits()
digits.images.shape
```

```
Out[1]: (1797, 8, 8)
```

- The images data is a three-dimensional array:
 - ▣ 1,797 samples
 - ▣ each consisting of an 8×8 grid of pixels.

Loading and visualizing the digits data

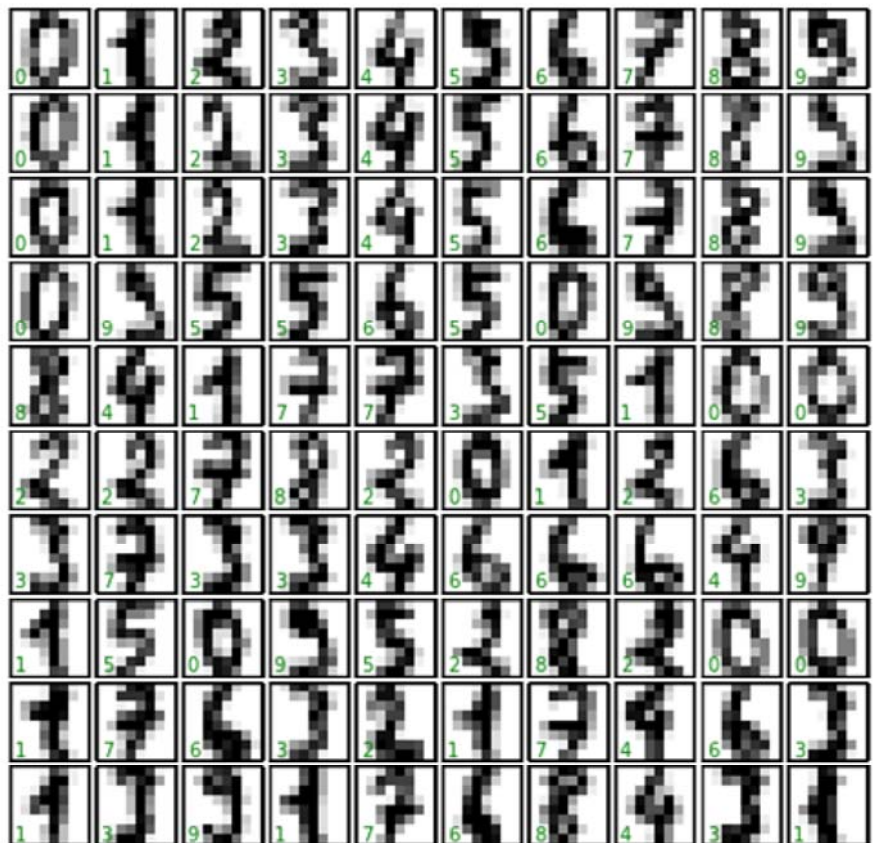
46

- Let's visualize the first hundred of these

```
%matplotlib inline
import matplotlib.pyplot as plt
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                        subplot_kw={'xticks':[], 'yticks':[]},
                        gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
    ax.imshow(digits.images[i], cmap='binary',
              interpolation='nearest')
    ax.text(0.05, 0.05, str(digits.target[i]),
           transform=ax.transAxes, color='green')
```

The handwritten digits data.

Each sample is represented by one 8×8 grid of pixels.



Data Preparation

48

- In order to work with this data within Scikit-Learn, we need a two-dimensional, $[n_samples, n_features]$ representation.
 - ▣ We can accomplish this by treating each pixel in the image as a feature — that is, by flattening out the pixel arrays so that we have a length-64 array of pixel values representing each digit.
- Additionally, we need the target array, which gives the previously determined label for each digit.
- These two quantities are built into the digits dataset under the **data** and **target** attributes, respectively:

Data Preparation

49

- We see here that there are 1,797 samples and 64 features.

```
X = digits.data
X.shape
Out[6]: (1797, 64)

y = digits.target
y.shape
Out[7]: (1797,)
```

Unsupervised learning: Dimensionality reduction

50

- We'd like to visualize our points within the 64-dimensional parameter space, but it's difficult to effectively visualize points in such a high-dimensional space.
- Instead we'll reduce the dimensions to 2, using an unsupervised method.
- Here, we'll make use of a [manifold learning algorithm](#) called *Isomap* and transform the data to two dimensions.

Unsupervised learning: Dimensionality reduction

51

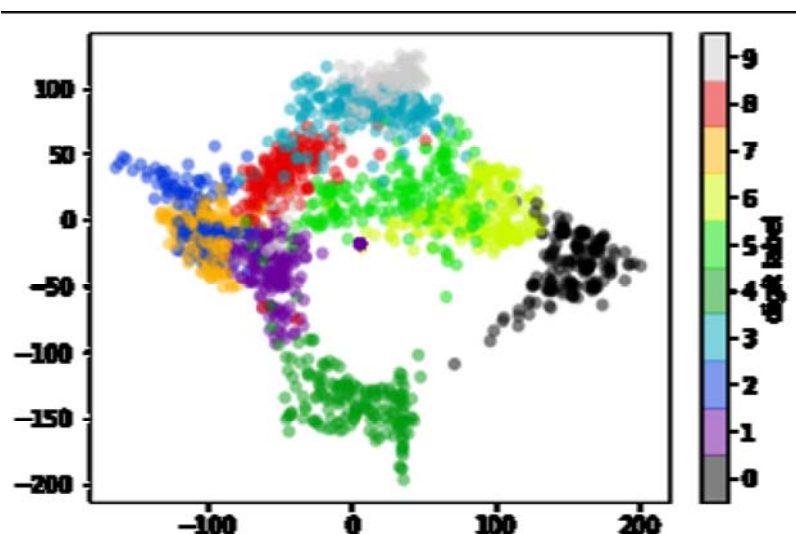
```
from sklearn.manifold import Isomap
iso = Isomap(n_components=2)
iso.fit(digits.data)
data_projected = iso.transform(digits.data)
data_projected.shape
Out[8]: (1797, 2)
```

```
plt.scatter(data_projected[:, 0], data_projected[:, 1],
            c=digits.target,
            edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('nipy_spectral', 10))
plt.colorbar(label='digit label', ticks=range(10))
plt.clim(-0.5, 9.5);
```

Unsupervised learning: Dimensionality reduction

52

- The different groups appear to be fairly well separated in the parameter space:
- This tells us that even a very straightforward supervised classification algorithm should perform suitably on this data.



matplotlib.pyplot.scatter()

53

- `matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, hold=None, data=None, **kwargs)`
 - **c**: color, sequence, or sequence of color, optional, default: 'b'
 - **alpha**: alpha value, between 0 (transparent) and 1 (opaque)
 - **cmap**: [Colormap](#), optional, default: None
 - `matplotlib.cm.get_cmap()`: Get a colormap instance

matplotlib.pyplot.colorbar(), clim()

54

- `matplotlib.pyplot.colorbar(mappable=None, cax=None, ax=None, **kw)`
 - Add a colorbar to a plot.
 - https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.colorbar
- `matplotlib.pyplot.clim(vmin=None, vmax=None)`
 - Set the color limits of the current image.
 - https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.clim

Classification on digits

55

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0)
model = GaussianNB()
model.fit(Xtrain, ytrain)
y_model = model.predict(Xtest)
accuracy_score(ytest, y_model)

Out[9]: 0.8333333333333333
```

Confusion Matrix

56

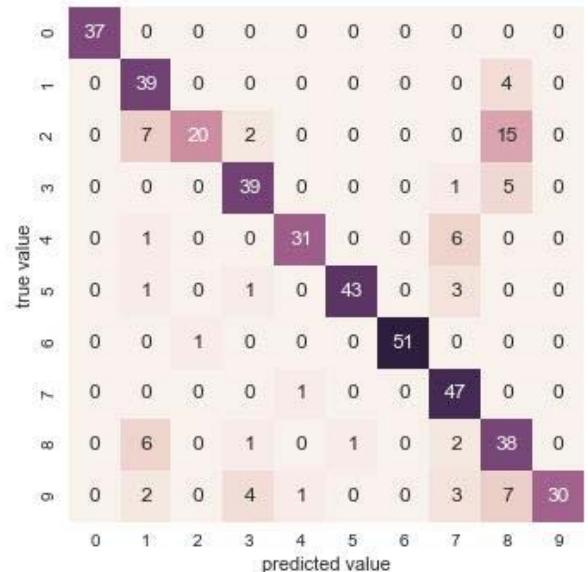
- The single accuracy number doesn't tell us *where* we've gone wrong.
- One nice way to do this is to use the *confusion matrix*.

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(ytest, y_model)
sns.heatmap(mat, square=True, annot=True,
            cbar=False, fmt="d")
plt.xlabel('predicted value')
plt.ylabel('true value');
plt.show()
```

Confusion Matrix

57

- This shows us where the mislabeled points tend to be: for example, a large number of twos here are misclassified as either ones or eights.



Inputs with their predicted labels

58

- Plot the inputs again with their predicted labels.
 - We'll use green for correct labels, and red for incorrect labels.

```
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                        subplot_kw={'xticks':[], 'yticks':[]},
                        gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
    ax.imshow(digits.images[i], cmap='binary',
              interpolation='nearest')
    ax.text(0.05, 0.05, str(y_model[i]),
           transform=ax.transAxes, color='green' if (ytest[i] ==
y_model[i]) else 'red')
```

green for correct labels,
and red for incorrect
labels

