Machine Learning

Python Machine Learning

- The version numbers of the major Python packages that were used throughout this tutorial are listed below:
  - NumPy 1.9.1
  - SciPy 0.14.0
  - scikit-learn 0.15.2
  - matplotlib 1.4.0
  - pandas 0.15.2

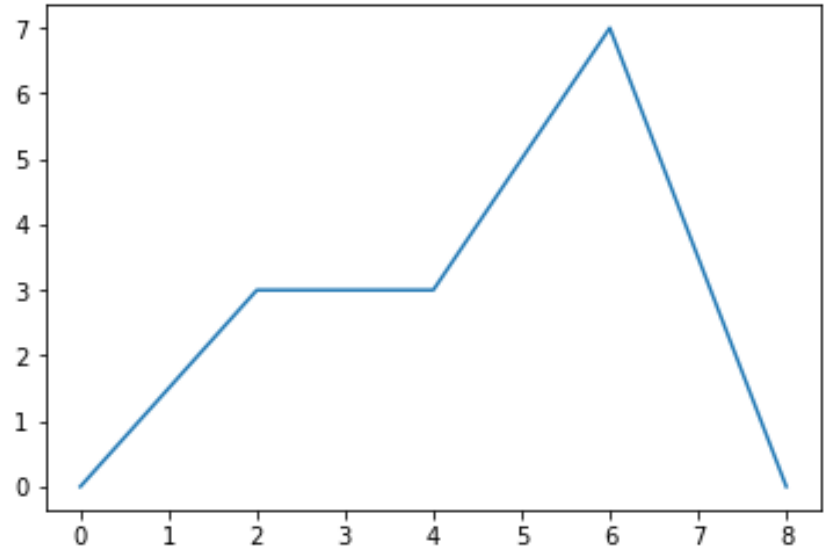
INSTALL ANACONDA

- **DOWNLOAD ANACONDA**
- https://www.continuum.io/downloads

Matplotlib

- The **pyplot** interface is a function-based interface that uses the **Matlab-like** conventions.
- However, it does not include the **NumPy** functions. So, if we want to use NumPy, it must be imported separately.

```
In [1]: import matplotlib.pyplot as pyp

In [2]: x = [0, 2, 4, 6, 8]

In [3]: y = [0, 3, 3, 7, 0]

In [4]: pyp.plot(x, y)
Out[4]: [<matplotlib.lines.Line2D at 0xc0c48d0>]
```




```
In [5]: pyp.savefig("MyFirstPlot.png")
<matplotlib.figure.Figure at 0xbd5cb00>
```

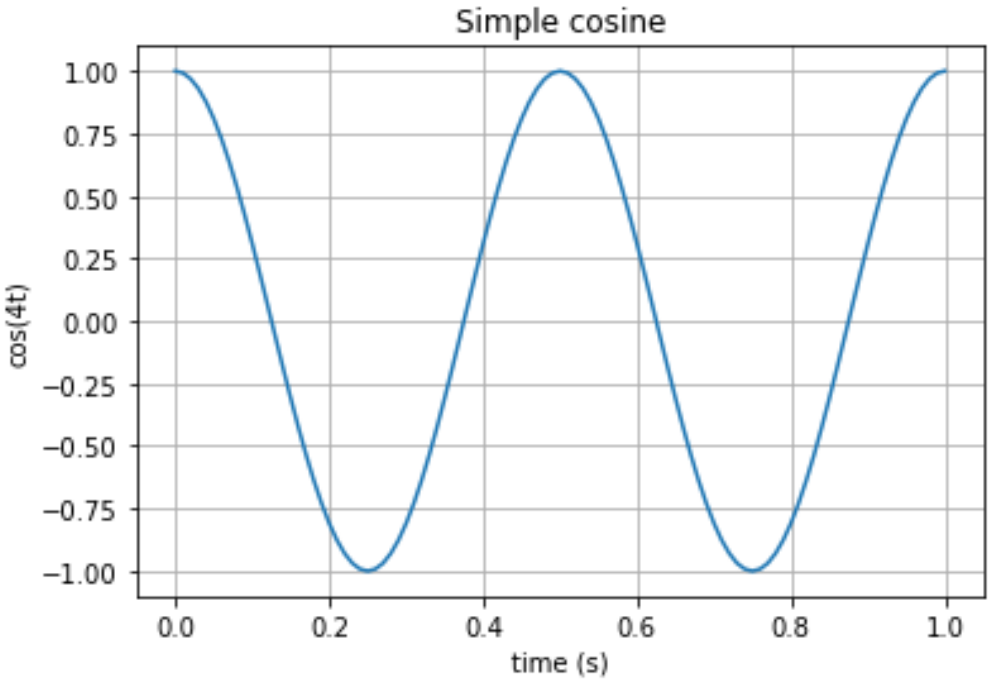
NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Another plot using Matplotlib

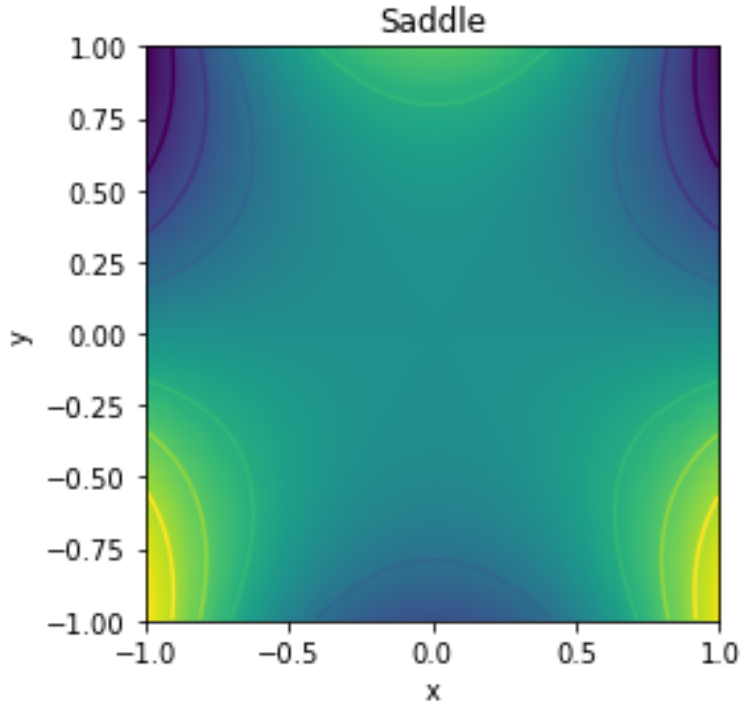
- Here is another simple Matplotlib code.

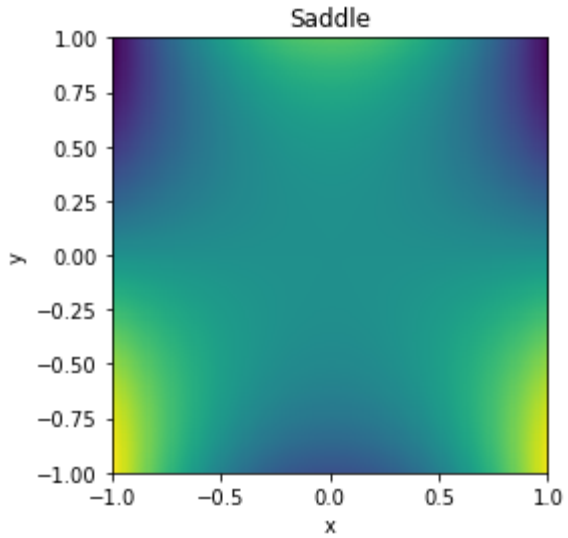
```
import numpy
import pylab

t = numpy.arange(0.0, 1.0+0.01, 0.01)
s = numpy.cos(numpy.pi*4*t)
pylab.plot(t, s)

pylab.xlabel('time (s)')
pylab.ylabel('cos(4t)')
pylab.title('Simple cosine')
pylab.grid(True)
pylab.savefig('simple_cosine')

pylab.show()
```

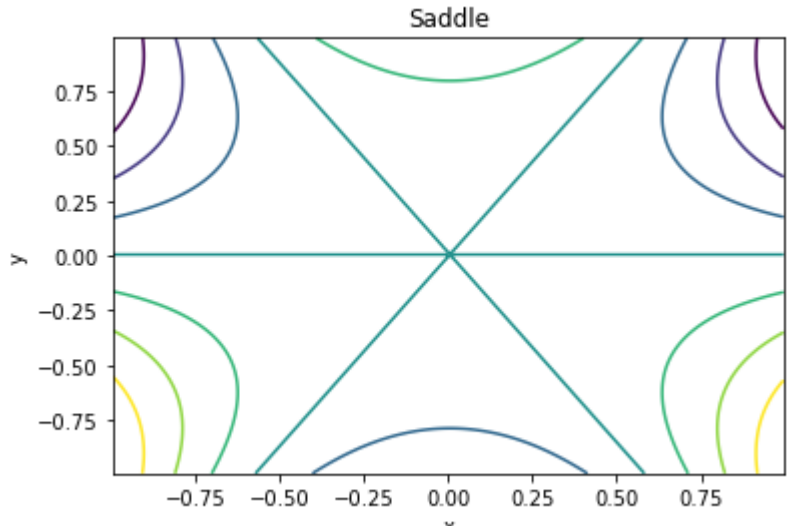


Contour plot using Matplotlib

```
import scipy
import pylab
import matplotlib.pyplot as plt

x,y = scipy.ogrid[-1.:1.:.01, -1.:1.:.01]
z = x**3-3*x*y**2
pylab.xlabel('x')
pylab.ylabel('y')
pylab.title('Saddle')
pylab.savefig('Saddle')
plt.show()
```







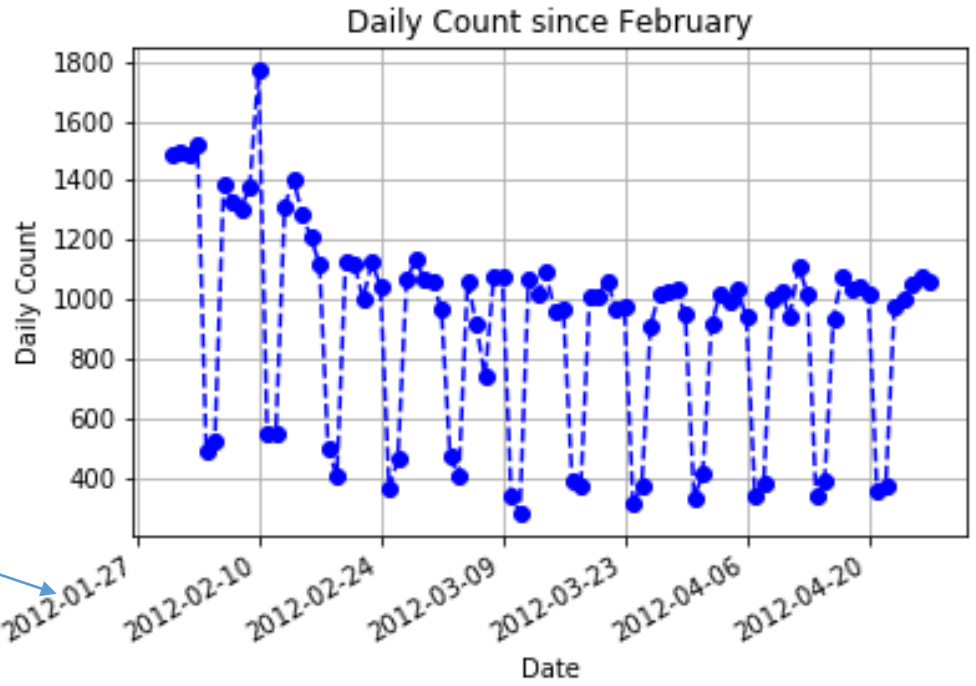
Plot using Matplc h csv data input

- The following example show t en x-axis is date string.

```
import numpy as np
import matplotlib.pyplot as plt
import datetime as DT

data= np.loadtxt('daily_count.csv', delimiter=',',
         dtype={'names': ('date', 'count'),'formats': ('S10', 'i4')} )

x = [DT.datetime.strptime(key,"%Y-%m-%d") for (key, value) in data ]
y = [value for (key, value) in data]

fig = plt.figure()
ax = fig.add_subplot(111)
ax.grid()

fig.autofmt_xdate()    # 自動格式日期標籤

plt.plot(x,y,'b--o--')    藍色
plt.xlabel('Date')
plt.ylabel('Daily Count')
plt.title('Daily Count since February')
plt.show()
```

string int

1	31/1/2012	1490
2	1/2/2012	1495
3	2/2/2012	1486
4	3/2/2012	1518
5	4/2/2012	492
6	5/2/2012	525
7	6/2/2012	1389
8	7/2/2012	1332
9	8/2/2012	1307
10	9/2/2012	1380
11	10/2/2012	1772
12	11/2/2012	547
13	12/2/2012	551
14	13/2/2012	1313
15	14/2/2012	1405
16	15/2/2012	1289
17	16/2/2012	1208
18	17/2/2012	1120
19	18/2/2012	495
20	19/2/2012	407
21	20/2/2012	1128
22	21/2/2012	1122
23	22/2/2012	1000
24	23/2/2012	1124
25	24/2/2012	1046
26	25/2/2012	364
27	26/2/2012	463
28	27/2/2012	1066
29	28/2/2012	1132
30	29/2/2012	1072
31	1/3/2012	1064



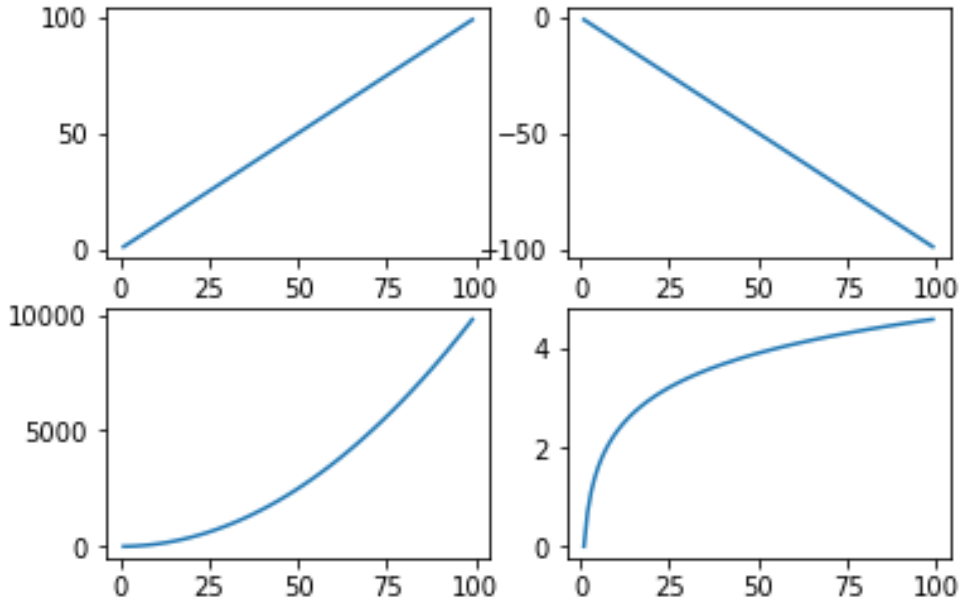
Subplot

```
subplot(nrows, ncols, plot_number)
```

- Where *nrows* and *ncols* are used to notionally split the figure into nrows * ncols sub-axes, and *plot_number* is used to identify the particular subplot that this function is to create within the notional grid.
- *plot_number* starts at 1, increments across rows first and has a maximum of nrows * ncols.

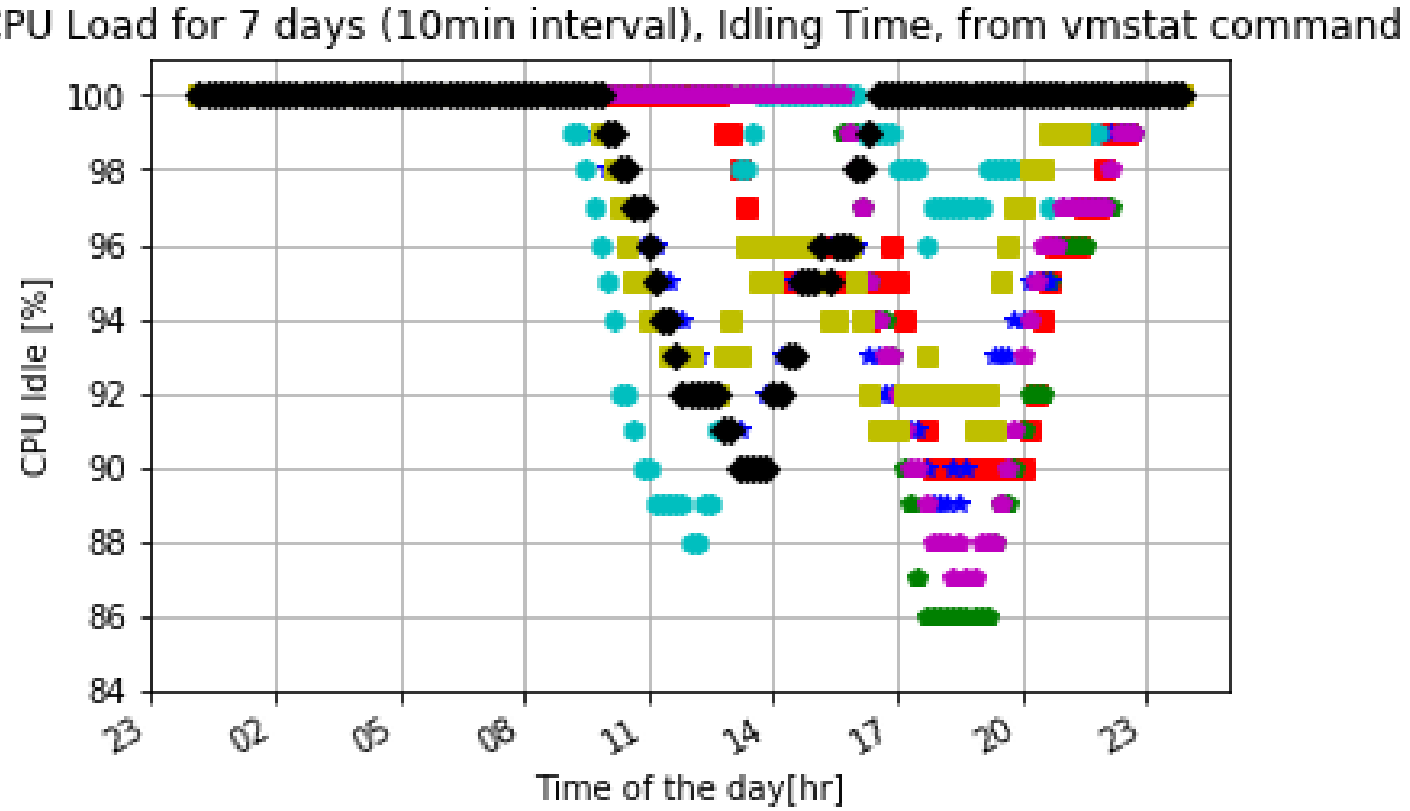
Subplot

```
import matplotlib.pyplot as plt
import numpy as np
import  pandas as pd
x=np.arange(1,100)
fig=plt.figure()
ax1=fig.add_subplot(221) #2*2的圖形 在第一個位置
ax1.plot(x,x)
ax2=fig.add_subplot(222)
ax2.plot(x,-x)
ax3=fig.add_subplot(223)
ax3.plot(x,x**2)
ax3=fig.add_subplot(224)
ax3.plot(x,np.log(x))
plt.show()
```



Plot using Matplotlib with legend

- The following example show the case when we have several columns of data.



```python
import numpy as np
import matplotlib.pyplot as plt
import datetime as dt
import matplotlib.dates as md

data= np.loadtxt('vmstat_7days_without_header.csv', delimiter=',',
    dtype={'names': ['time', 'mon','tue','wed','thrs','fri','sat','sun'],
           'formats': ['S8','i4','i4','i4','i4','i4','i4','i4']} )

x,y1,y2,y3,y4,y5,y6,y7 = [],[],[],[],[],[],[],[]

for z in data:
# 10 minute span
    if int((z[0].split(':',2))[1]) % 10 == 0:
        xc = dt.datetime.strptime(z[0],"%H:%M:%S")
        x.append(xc)
        y1.append(z[1])
        y2.append(z[2])
        y3.append(z[3])
        y4.append(z[4])
        y5.append(z[5])
        y6.append(z[6])
        y7.append(z[7])

fig = plt.figure()
ax = fig.add_subplot(111)
xfmt = md.DateFormatter('%H')
ax.xaxis.set_major_formatter(xfmt)
ax.grid()

# slanted x-axis tick label
fig.autofmt_xdate()

p1 = plt.plot(x,y1,'rs')
p2 = plt.plot(x,y2,'gp')
p3 = plt.plot(x,y3,'b*')
p4 = plt.plot(x,y4,'ch')
p5 = plt.plot(x,y5,'mp')
p6 = plt.plot(x,y6,'ys')
p7 = plt.plot(x,y7,'kD')

plt.ylabel("CPU Idle [%]")
plt.xlabel("Time of the day[hr]")

plt.ylim(84.0, 101)

plt.title("CPU Load for 7 days (10min interval), Idling Time, from vmstat command")

#let python select the best position for legend
plt.legend([p1[0],p2[0],p3[0],p4[0],p5[0],p6[0],p7[0]],
           ['Mon','Tue','Wed','Thu','Fri','Sat','Sun'], 'best', numpoints=1)

plt.show()
```

use `strftime()` format strings

`strftime_pre_1900(dt, fmt=None)`
Call time.strftime for years before 1900 by rolling forward a multiple of 28 years.

	A	B	C	D	E	F	G	H
1	0:02:00	1	2	3	4	100	100	100
2	0:03:00	99	99	99	100	100	99	99
3	0:04:00	100	100	100	100	100	100	100
4	0:05:00	100	100	100	100	100	100	100
5	0:06:00	100	100	100	100	100	100	100
6	0:07:00	100	100	100	100	100	100	100
7	0:08:00	99	99	99	100	100	99	99
8	0:09:00	100	100	100	100	100	100	100
9	0:10:00	100	100	100	100	100	100	100
10	0:11:00	100	100	100	100	100	100	100

#藍色:'b' | 綠色: 'g' | 紅色: 'r' | 藍綠色:'c' | 紅紫色:'m' | 黃色:'y' | 黑色:'k' | 白色:'w' #實線:'-' | 虛線: '--' | 虛點線:'-.' | 點線:':' | 點:'.' | #圓形:'o' | 上三角:'^' | 下三角:'v' | 左三角:'<' | 右三角:'>' | 方形:'s' | 加號:'+' | 叉形:'x' | 棱形:'D' | 細棱形:'d' # 三腳朝下:'1' | 三腳朝上:'2' | 三腳朝左:'3' | 三腳朝右:'4' | 六角形:'h' | 旋轉六角形:'H' | 五角形:'p' | 垂直線:'|'

21	0:22:00	100	100	100	100	100	100	100
22	0:23:00	99	99	99	100	100	99	99
23	0:24:00	100	100	100	100	100	100	100
24	0:25:00	100	100	100	100	100	100	100
25	0:26:00	100	100	100	100	100	100	100
26	0:27:00	100	100	100	100	100	100	100
27	0:28:00	99	99	99	100	100	99	99
28	0:29:00	100	100	100	100	100	100	100
29	0:30:00	100	100	100	100	100	100	100
30	0:31:00	100	100	100	100	100	100	100
31	0:32:00	100	100	100	100	100	100	100

The function gca() returns the current axes
(a matplotlib.axes.Axes instance)

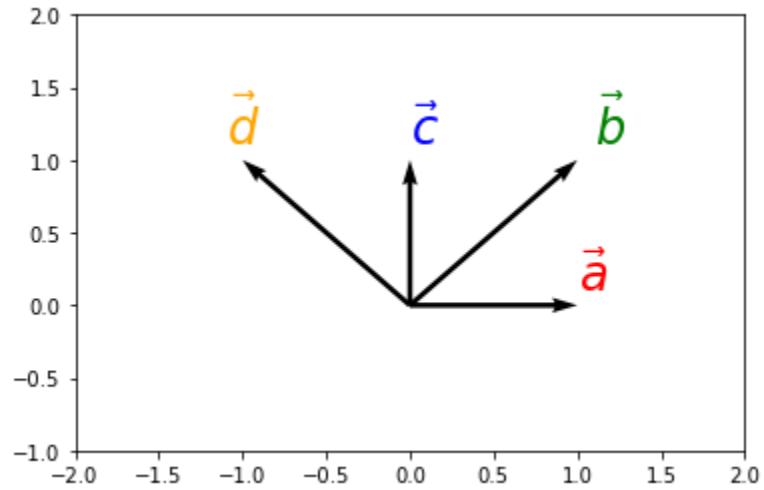
Vector Plot

quiver(**args*, ***kw*) Plot a 2-D field of arrows.

```
quiver(U, V, **kw)
quiver(U, V, C, **kw)
quiver(X, Y, U, V, **kw)
quiver(X, Y, U, V, C, **kw)
```

U and *V* are the arrow data, *X* and *Y* set the locaiton of the arrows, and *C* sets the color of the arrows. These arguments may be 1-D arrays or sequences.

```python
import numpy as np
import matplotlib.pyplot as plt
soa =np.array( [ [0,0,1,0], [0,0,1,1],[0,0,0,1], [0,0,-1,1]])
X,Y,U,V = zip(*soa)
plt.figure()
ax = plt.gca()
ax.quiver(X,Y,U,V,angles='xy',scale_units='xy',scale=1)
ax.set_xlim([-2,2])
ax.set_ylim([-1,2])
plt.text(1.0, 0.1, r'$\vec a$', fontsize=24, color='red', fontweight='bold')
plt.text(1.1, 1.1, r'$\vec b$', fontsize=24, color='green', fontweight='bold')
plt.text(0.0, 1.1, r'$\vec c$', fontsize=24, color='blue', fontweight='bold')
plt.text(-1.1, 1.1, r'$\vec d$', fontsize=24, color='orange', fontweight='bold')
plt.draw()
plt.show()
```



zip() 是 Python 的一個內建函數，它接受一系列可迭代的對象作為參數，將對象中對應的元素打包成一個個 tuple（元組），然後返回由這些 tuples 組成的 list（列表）。若傳入參數的長度不等，則返回list的長度和參數中長度最短的對象相同。利用*號操作符，可以將 list unzip（解壓）。

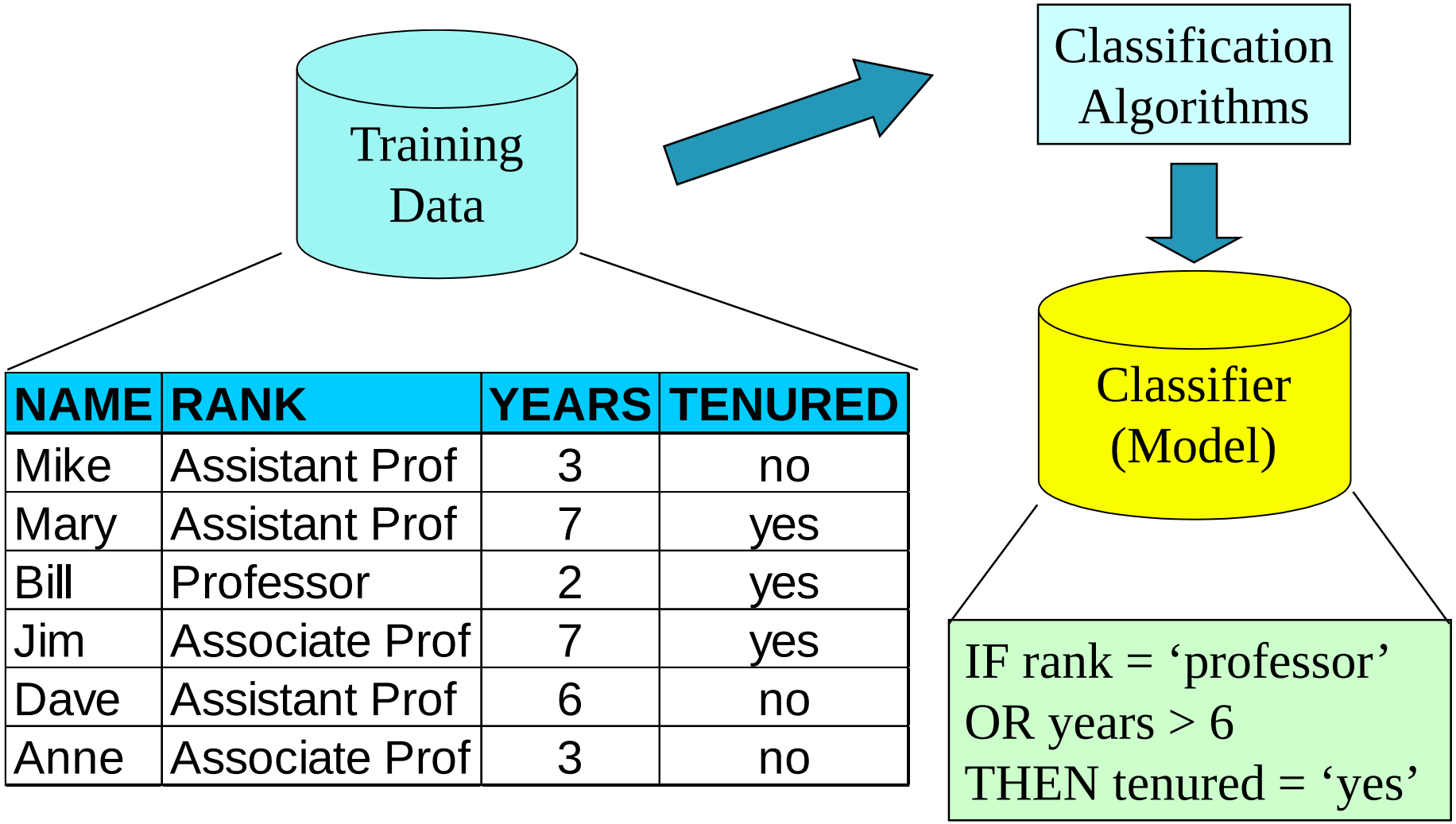
Classification vs. Prediction

- Classification
  - predicts categorical class labels (discrete or nominal)
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- Prediction
  - models continuous-valued functions, i.e., predicts unknown or missing values
- Typical applications
  - Credit approval
  - Target marketing
  - Medical diagnosis
  - Fraud detection

Classification—A Two-Step Process

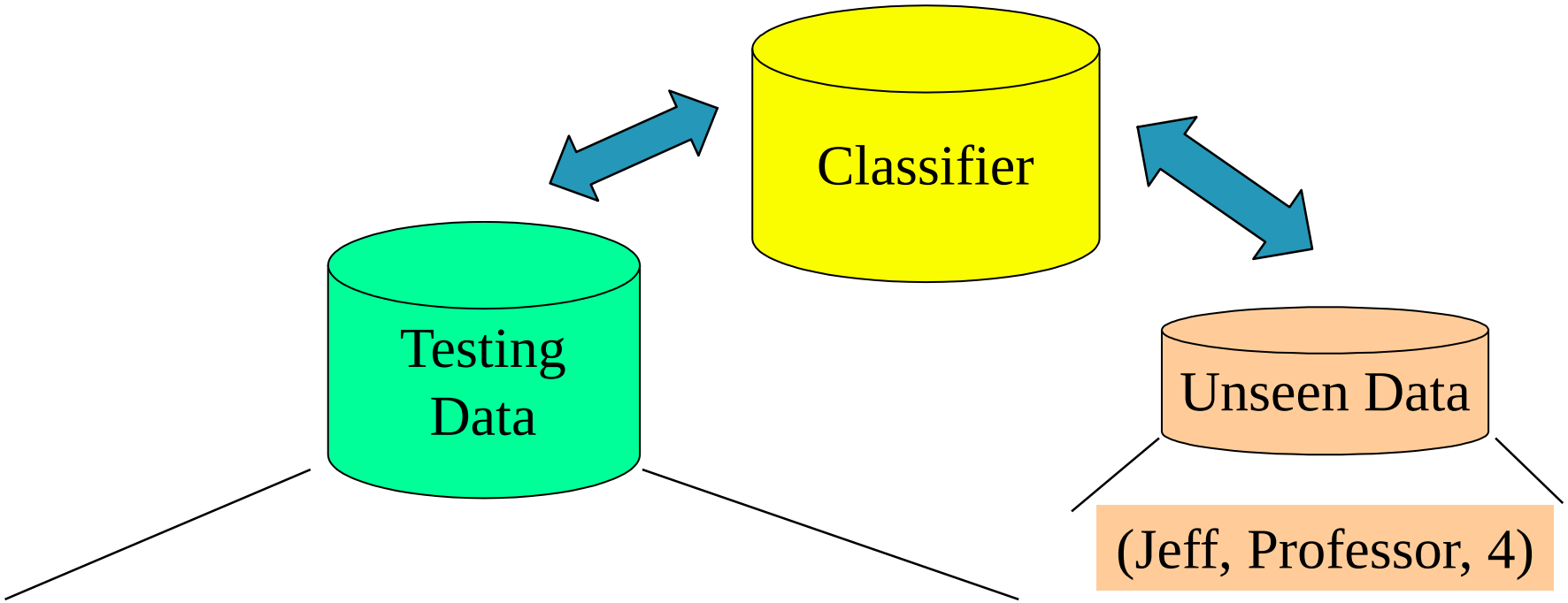
- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as classification rules, decision trees, or mathematical formulae
- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set, otherwise over-fitting will occur
  - If the accuracy is acceptable, use the model to classify data tuples whose class labels are not known

Process (1): Model Construction



NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Process (2): Using the Model in Prediction



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

Supervised vs. Unsupervised Learning

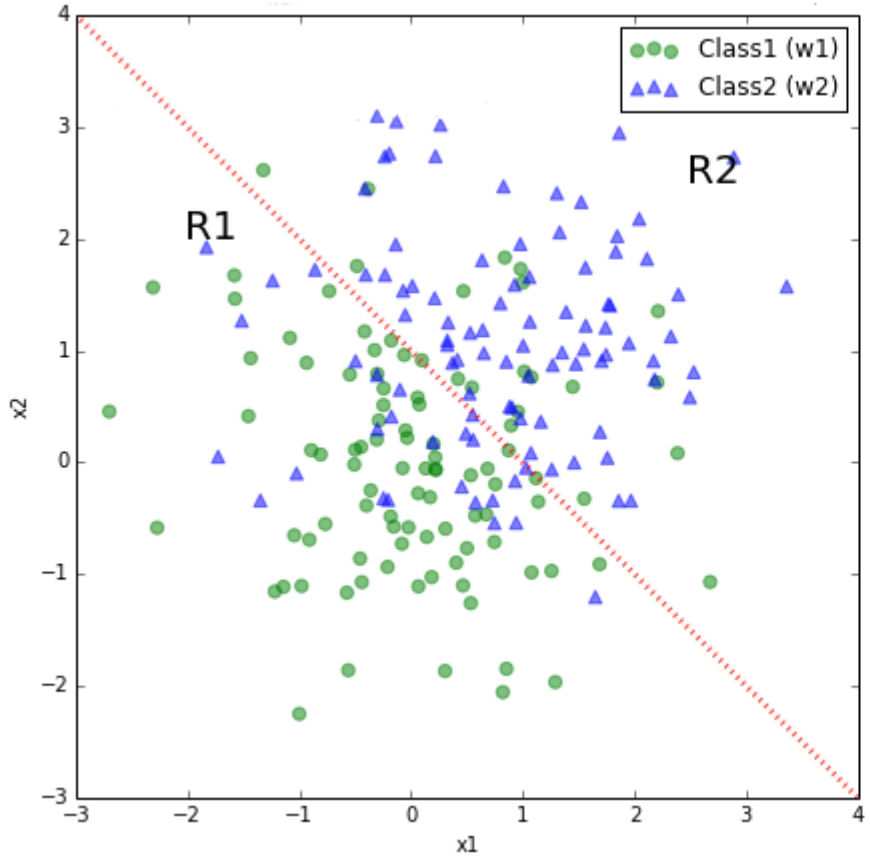
- Supervised learning (classification)
  - Supervision: The training data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations
  - New data is classified based on the training set
- Unsupervised learning (clustering)
  - The class labels of training data is unknown
  - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

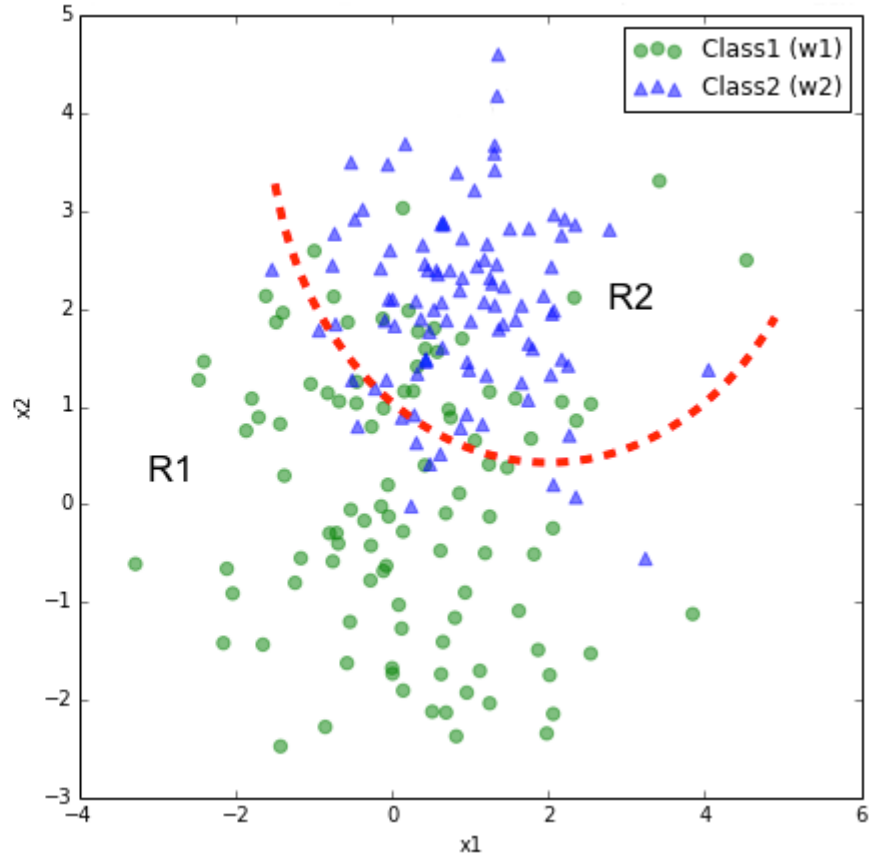
Supervised Learning

- The class labels in the dataset, which is used to build the classification model, are known.
- For example, a dataset for spam filtering would contain spam messages as well as “ham” (= not-spam) messages.
- We would know which message in the training set is spam or ham, and we’d use this information to train our model in order to classify new unseen messages.

Supervised Learning

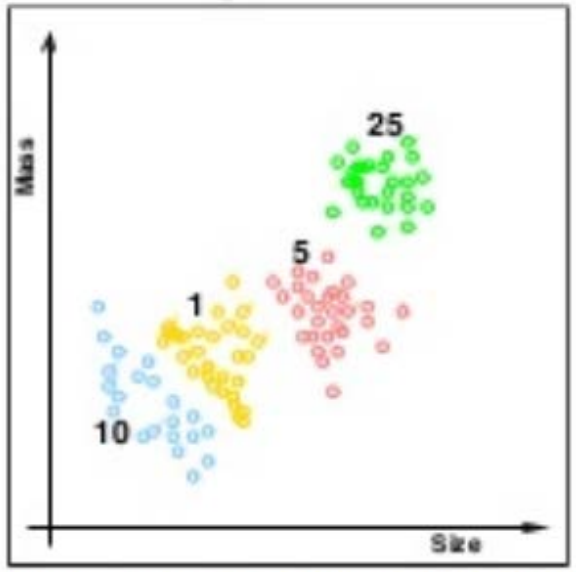
- The figure above shows an exemplary classification task for samples with two random variables; the training data (with class labels) are shown in the scatter plots.
- The red-dotted lines symbolize linear (left) or quadratic (right) decision boundaries that are used to define the decision regions R1 and R2.
- New observations will be assigned the class labels “w1” or “w2” depending on in which decision region they will fall into.
- We can already assume that our classification of unseen instances won’t be “perfect” and some percentage samples will likely be mis-classified.


Class1 (w1)
Class2 (w2)
R1
R2
x1
x2


Class1 (w1)
Class2 (w2)
R1
R2
x1
x2

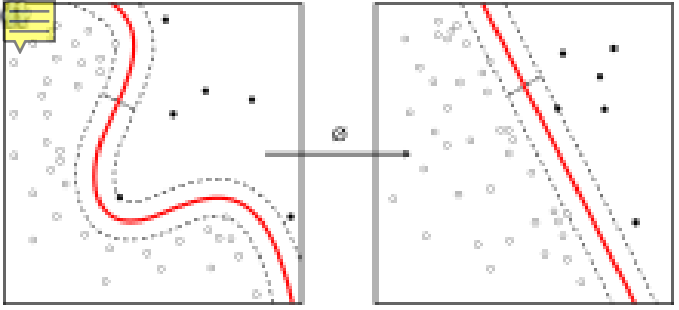
Supervised Learning

- Supervised learning is concerned with learning a model from **labeled data (training data)** which has the correct answer.
- This allows us to make predictions about future or unseen data.
- It's collections of scattered points whose coordinates are size and weight. Supervised learning gives us not only the sample data but also correct answers, for this case, it's the colors or the values of the coin.



Picture source : Lecture 01 - The Learning Problem, Caltech

Regression and **classification** are the most common types of problems in supervised learning.

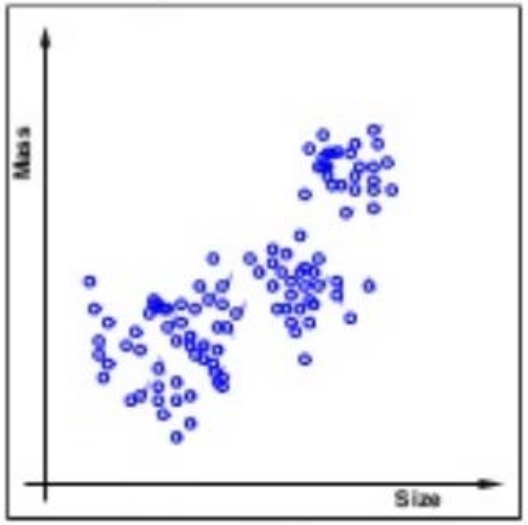


Support Vector Machine

- **Support vector machines** (**SVMs**) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.
- Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.
- An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.
- New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Unsupervised Learning

- "An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a '**reasonable**' way."
- Unsupervised Learning's task is to construct an estimator which is able to predict the label of an object given the set of features.
- Unsupervised learning does not give us the color or the value information. In other words, it only gives us sample data but not the data for correct answers:



Picture source : Lecture 01 - The Learning Problem, Caltech

Unsupervised Learning

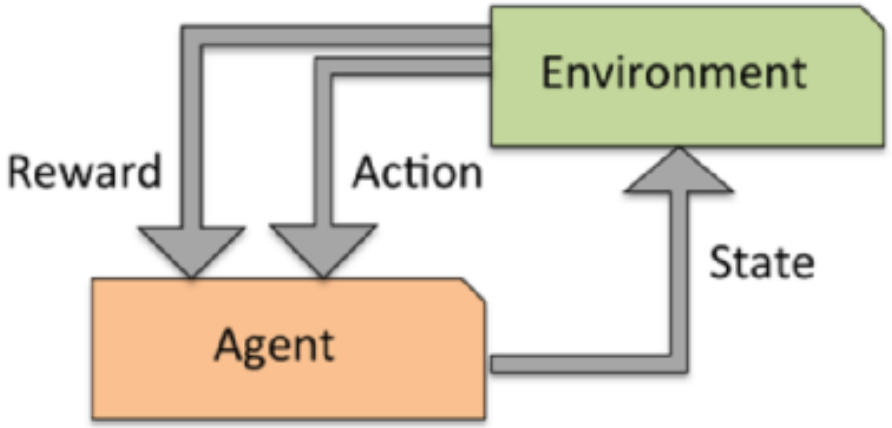
- For unsupervised learning we get: `(input, ?)` instead of the following for supervised learning: `(input, correct output)`
- Unsupervised Learning problem is "trying to find **hidden structure** in **unlabeled data**. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution.
  - This distinguishes unsupervised learning from supervised learning and reinforcement learning."
- Simply put, the goal of unsupervised learning is to **find structure in the unlabeled data**.
  - **Clustering** is probably the most common technique.

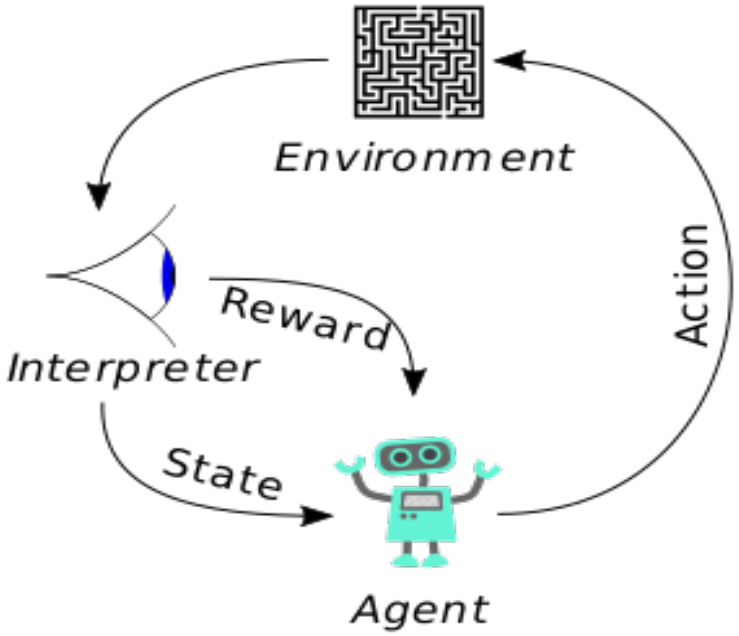
Reinforcement learning (強化學習)

- The goal is to develop a system that improves its performance based on interactions with the environment.
- We could think of reinforcement learning as a **supervised learning**, however, in reinforcement learning the **feedback (reward)** from the environment is not the label or value, but a measure of how well the action was measured by the reward function.
- Via the interaction with the environment, our system (agent) can then use reinforcement learning to learn a series of actions that maximizes this reward via an exploratory **trial-and-error** approach.

Reinforcement learning

- A popular example of reinforcement learning is a chess engine.
- Here, the agent decides upon a series of moves depending on the state of the board (the environment), and the reward can be defined as win or lose at the end of the game:

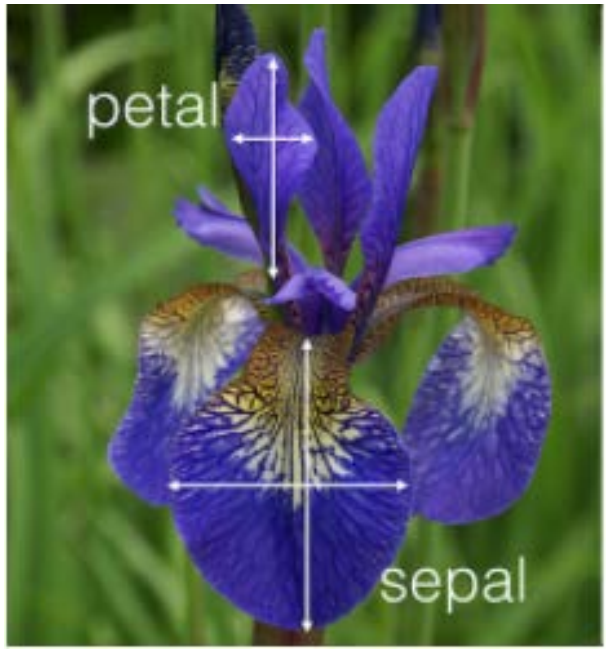




Credit: Python Machine Learning by Sebastian Raschka, 2015

Supervised - Classification with iris dataset

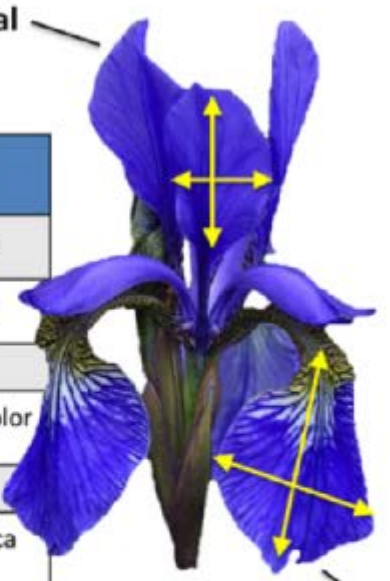
- The following table is iris dataset, which is a classic example in the field of machine learning.



花瓣

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



萼片

Class labels
(targets)

Features
(attributes, measurements, dimensions)

iris dataset

- Iris dataset contains the measurements of 150 iris flowers from three different species: *Setosa*, *Versicolor*, and *Viriginica*: it can then be written as a 150 x 3 matrix.
- Each flower sample represents one row in our data set, and the flower measurements in centimeters are stored as columns, which we also call the **features** of the dataset.
- We are given the measurements of petals and sepals. The task is to guess the class of an individual flower. It's a classification task.

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> X = iris.data
>>> y = iris.target
```

iris dataset

- It is trivial to train a classifier once the data has this format. A support vector machine (SVM), for instance, with a linear kernel:

```
In [5]: from sklearn.svm import LinearSVC

In [6]: LinearSVC()
Out[6]:
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)

In [7]: clf = LinearSVC
```

- **clf** is a statistical model that has hyperparameters that control the learning algorithm.
- Those hyperparameters can be supplied by the user in the constructor of the model.

iris dataset

- By default the real model parameters are not initialized. The model parameters will be automatically tuned from the data by calling the **fit()** method:

```
In [61]: X = iris.data

In [62]: y = iris.target

In [63]: clf.fit(X,y)
Out[63]:
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)

In [64]: clf.coef_
Out[64]:
array([[ 0.18424289,  0.45122875, -0.80793655, -0.45071061],
       [ 0.05326679, -0.89082157,  0.40466505, -0.94060226],
       [-0.85068118, -0.98664802,  1.38091056,  1.86530344]])

In [65]: clf.intercept_
Out[65]: array([ 0.10956102,  1.66146465, -1.70959045])
```

coef_ : array, shape = [n_class-1, n_features]

Weights assigned to the features (coefficients in the primal problem). This is only available in the case of a linear kernel.

coef_ is a read only property derived from dual_coef_ and support_vectors_.

intercept_ : array, shape = [n_class * (n_class-1) / 2]

Constants in decision function.

`fit` (*X*, *y*, *sample_weight=None*) [source]

Fit the SVM model according to the given training data.

Parameters:	**X** : {array-like, sparse matrix}, shape (n_samples, n_features) Training vectors, where n_samples is the number of samples and n_features is the number of features. For kernel="precomputed", the expected shape of X is (n_samples, n_samples). **y** : array-like, shape (n_samples,) Target values (class labels in classification, real numbers in regression) **sample_weight** : array-like, shape (n_samples,) Per-sample weights. Rescale C per sample. Higher weights force the classifier to put more emphasis on these points.
Returns:	**self** : object Returns self.

iris dataset

- Once the model is **trained**, it can be used to **predict** the most likely outcome on **unseen data**.

```
In [26]: iris.data
Out[26]:
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       ...,
       [ 6.5,  3. ,  5.2,  2. ],
       [ 6.2,  3.4,  5.4,  2.3],
       [ 5.9,  3. ,  5.1,  1.8]])

In [27]: X_new = [[ 5.9,  3. ,  5.1,  1.8]]

In [28]: clf.predict(X_new)
Out[28]: array([2])

In [29]: iris.target_names
Out[29]:
array(['setosa', 'versicolor', 'virginica'],
      dtype='|S10')
```

- The result is **2**, and the id of the 3rd iris class, namely '**virginica**'.

Supervised - Logistic regression models

- scikit-learn logistic regression models can further predict probabilities of the outcome.
- We continue to use the data from the previous section.

```
In [30]: from sklearn.linear_model import LogisticRegression

In [31]: clf2 = LogisticRegression().fit(X, y)

In [32]: clf2
Out[32]:
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)

In [33]: clf2.predict_proba(X_new)
Out[33]: array([[ 0.00168398,  0.2810578 ,  0.71725822]])
```

- This means that the model estimates that the sample in X_new has:
  - 0.1% likelyhood to belong to the 'setosa' class
  - 28% likelyhood to belong to the 'versicolor' class
  - 71% likelyhood to belong to the 'virginica' class

Logistic regression model (邏輯回歸)

- Actually, the model can predict using predict() method which is based on the probability output from **predict_proba()**:

```
In [34]: clf2.predict(X_new)
Out[34]: array([2])
```

- The **logistic regression** is not a regression method but a **classification method**.

- When do we use logistic regression?
  - In probabilistic setups - easy to incorporate prior knowledge
  - When the number of features is pretty small - The model will tell us which features are important.
  - When the training speed is an issue - training logistic regression is relatively fast.
  - When precision is not critical.

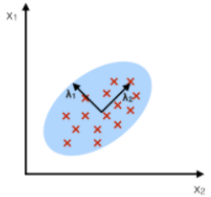
Unsupervised - Dimensionality Reduction

- We want to derive a set of new artificial features that is smaller than the original feature set while retaining most of the variance of the original data. We call this dimensionality reduction (維度縮減).

原本的Data寫在一個比較高的維度作標上，我們希望找到一個低維度的作標來描述它，但又不能失去Data本身的特質。

- Principal Component Analysis (PCA) is the most common technique for dimensionality reduction.
- PCA does it using linear combinations of the original features through a truncated Singular Value Decomposition of the matrix X so as to project the data onto a base of the top singular vectors.

PCA:
component axes that
maximize the variance



```
In [70]: from sklearn.decomposition import PCA

In [71]: pca = PCA(n_components=2, whiten=True).fit(X)
```

Feature Selection and Dimensionality Reduction

- Distinguishing between feature selection and dimensionality reduction might seem counter-intuitive at first, since feature selection will eventually lead (reduce dimensionality) to a smaller feature space.
- The key difference between the terms “feature selection” and “dimensionality reduction” is that in feature selection, we keep the “original feature axis”, whereas dimensionality reduction usually involves a transformation technique.
- The main purpose of those two approaches is to remove noise, increase computational efficiency by retaining only “useful” (discriminatory) information, and to avoid overfitting (“curse of dimensionality”).

Feature Selection and Dimensionality Reduction

- In feature selection, we are interested in retaining only those features that are “meaningful” - features that can help to build a “good” classifier.
- For example, if we’d have a whole bunch of attributes that describe our Iris flowers (color, height, etc.), feature selection could involve the reduction of the available data to the 4 measurements that describe the petal and sepal dimensions.
- Or, if we’d start with those 4 attributes (sepal and petal lengths and widths), we could further narrow down our selection to petal lengths and widths and thereby reduce our feature space from 4 to 2 dimensions

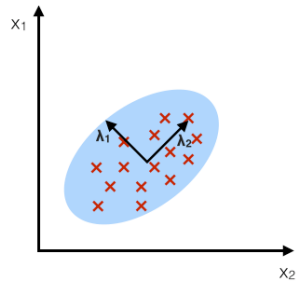
Feature Selection and Dimensionality Reduction

- Feature selection is often based on domain knowledge (note that it is always helpful to consult a domain expert) or exploratory analyses, such as histograms or scatterplots as we have seen earlier.
- Finding the feature subset of a certain size that optimizes the performance of a classification model would require an exhaustive search - the sampling of all possible combinations.
- In practice, this approach might not be feasible because of computational limitations so that sequential feature selection (Feature Selection Algorithms in Python) or genetic algorithms are being used to select a sub-optimal feature subset.

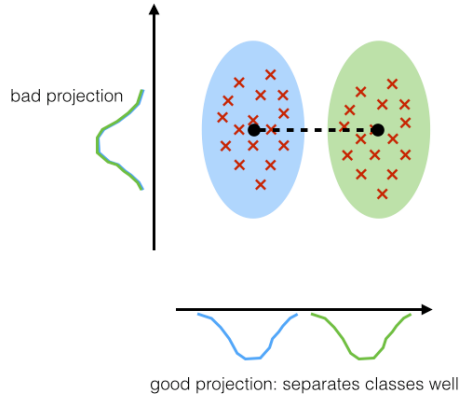
Dimensionality Reduction

- Commonly used dimensionality reduction techniques are linear transformations such as Principal Component Analyses (PCA) and Linear Discriminant Analysis (LDA).
- PCA can be described as an “unsupervised” algorithm, since it “ignores” class labels and its goal is to find the directions (the so-called principal components) that maximize the variance in a dataset.
- LDA is “supervised” and computes the directions (“linear discriminants”) that will represent the axes that maximize the separation between multiple classes.

PCA:
component axes that maximize the variance



LDA:
maximizing the component axes for class-separation



Unsupervised - Dimensionality Reduction

- After the **fit()**, the pca model exposes the singular vectors in the **components_ attribute**:

```
In [37]: pca.components_
Out[37]:
array([[ 0.36158968, -0.08226889,  0.85657211,  0.35884393],
       [ 0.65653988,  0.72971237, -0.1757674 , -0.07470647]])

In [38]: pca.explained_variance_ratio_
Out[38]: array([ 0.92461621,  0.05301557])

In [39]: pca.explained_variance_ratio_.sum()
Out[39]: 0.97763177502480336
```

components_ : array, [n_components, n_features]
Principal axes in feature space, representing the directions of maximum variance in the data. The components are sorted by `explained_variance_`

- Since the number of retained components is 2, we project the iris dataset along those first 2 dimensions:

```
X_pca = pca.transform(X)
```

explained_variance_ratio_ : array, [n_components]

Percentage of variance explained by each of the selected components.

If `n_components` is not set then all components are stored and the sum of explained variances is equal to 1.0.

Unsupervised - Dimensionality Reduction

- Normalized:

```
In [41]: import numpy as np

In [42]: np.round(X_pca.mean(axis=0), decimals=5)
Out[42]: array([ 0.,  0.])

In [43]: np.round(X_pca.std(axis=0), decimals=5)
Out[43]: array([ 1.,  1.])
```

```
>>> np.around(1.23456789)
1.0
>>> np.around(1.23456789, decimals=0)
1.0
>>> np.around(1.23456789, decimals=1)
1.2
>>> np.around(1.23456789, decimals=2)
1.23
>>> np.around(1.23456789, decimals=3)
1.2350000000000001
>>> np.around(1.23456789, decimals=4)
1.2345999999999999
```

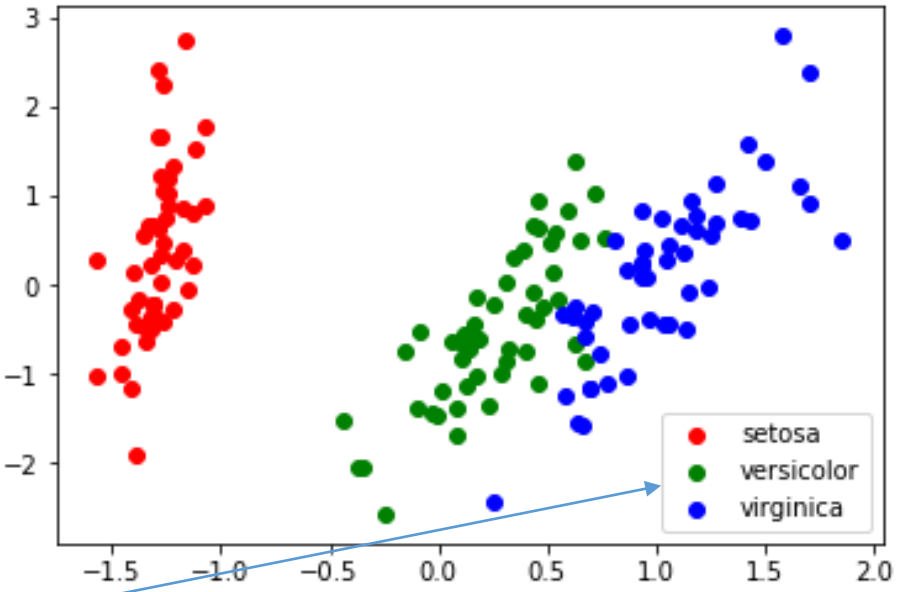
- Also note that the samples components do no longer carry any linear correlation:

```
In [44]: import numpy as np

In [45]: np.round(np.corrcoef(X_pca.T), decimals=5)
Out[45]:
array([[ 1.,  0.],
       [ 0.,  1.]])
```

- Now, we can visualize the dataset using **pylab**, for instance by defining the utility function:

```
1  from sklearn.datasets import load_iris
2  import pylab as pl
3  from itertools import cycle
4  from sklearn.decomposition import PCA
5
6  class pca_reduction:
7      def __init__(self):
8          iris = load_iris()
9          self.X = iris.data
10         self.y = iris.target
11         self.names = iris.target_names
12         self.plot()
13
14     def plot(self):
15         pca = PCA(n_components=2, whiten=True).fit(self.X)
16         X_pca = pca.transform(self.X)
17         plot_2D(X_pca, self.y, self.names)
18
19 def plot_2D(data, target, target_names):
20     colors = cycle('rgbcmykw')
21     target_ids = range(len(target_names))
22     pl.figure()
23     for i, c, label in zip(target_ids, colors, target_names):
24         pl.scatter(data[target == i, 0], data[target == i, 1],
25                    c=c, label=label)
26     pl.legend()
27     pl.show()
28
29 if __name__ == '__main__':
30     pr = pca_reduction()
31     print 'X = %s' %pr.X
32     print 'y = %s' %pr.y
33     print 'names = %s' %pr.names
```

rgbcmykw→ rgbcmykw…




```
X = [[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 ...,
 [ 6.5  3.   5.2  2. ]
 [ 6.2  3.4  5.4  2.3]
 [ 5.9  3.   5.1  1.8]]
y = [0 0 0 ..., 2 2 2]
names = ['setosa' 'versicolor' 'virginica']
```

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*)

Make a scatter plot of `x` vs `y`

Marker size is scaled by `s` and marker color is mapped to `c`

Parameters:

x, y : array_like, shape (n,)

Input data

s : scalar or array_like, shape (n,), optional

size in points^2. Default is `rcParams['lines.markersize'] ** 2`.

c : color, sequence, or sequence of color, optional, default: 'b'

`c` can be a single color format string, or a sequence of color specifications of length `N`, or a sequence of `N` numbers to be mapped to colors using the `cmap` and `norm` specified via kwargs (see below). Note that `c` should not be a single numeric RGB or RGBA sequence because that is indistinguishable from an array of values to be colormapped. `c` can be a 2-D array in which the rows are RGB or RGBA, however, including the case of a single row to specify the same color for all points.

marker : `MarkerStyle`, optional, default: 'o'

See `markers` for more information on the different styles of markers scatter supports. `marker` can be either an instance of the class or the text shorthand for a particular marker.

cmap : `Colormap`, optional, default: None

A `Colormap` instance or registered name. `cmap` is only used if `c` is an array of floats. If None, defaults to rc `image.cmap`.

norm : `Normalize`, optional, default: None

A `Normalize` instance is used to scale luminance data to 0, 1. `norm` is only used if `c` is an array of floats. If `None`, use the default `normalize()`.

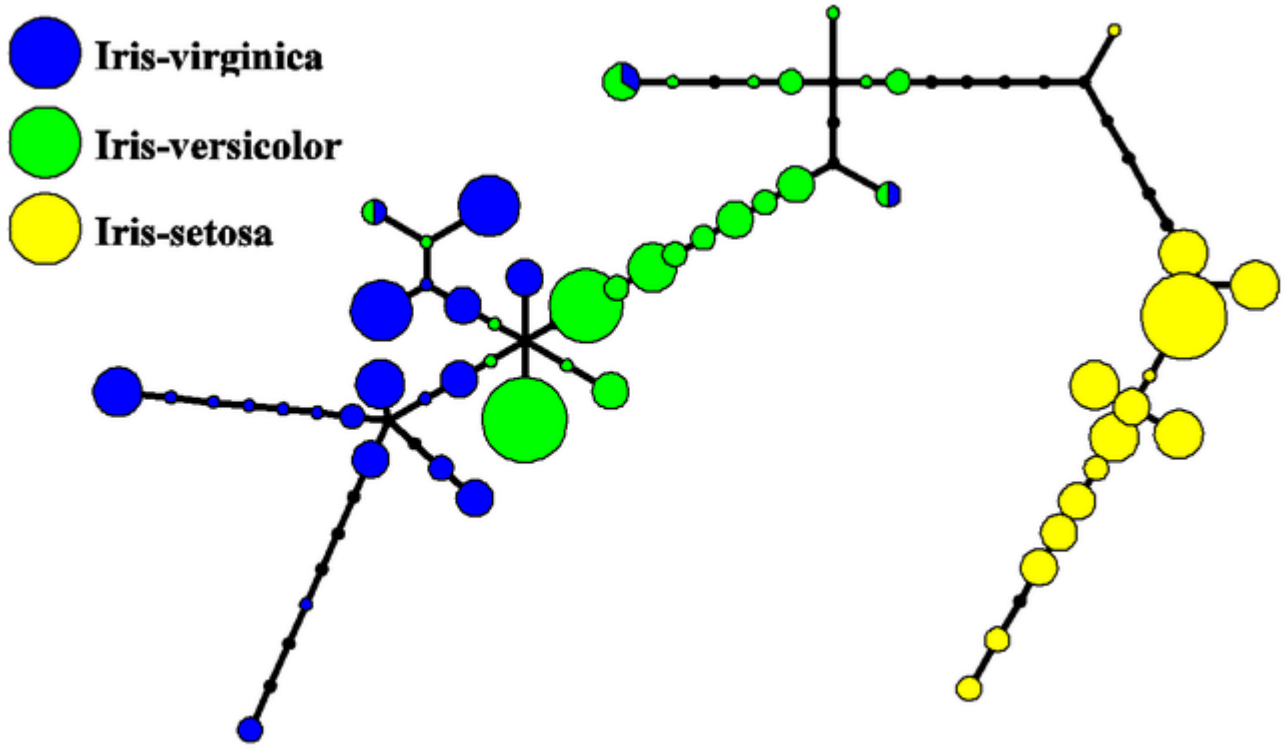
vmin, vmax : scalar, optional, default: None

`vmin` and `vmax` are used in conjunction with `norm` to normalize luminance data. If either are `None`, the min and max of the color array is used. Note if you pass a `norm` instance, your settings for `vmin` and `vmax` will be ignored.

alpha : scalar, optional, default: None

The alpha blending value, between 0 (transparent) and 1 (opaque)

- The projection was determined without any help from the labels (represented by the colors), which means this learning is **unsupervised**.
- Nevertheless, we see that the projection gives us insight into the distribution of the different flowers in parameter space: notably, iris setosa is much more distinct than the other two species as shown in the picture below:



Picture source - Iris flower data set.