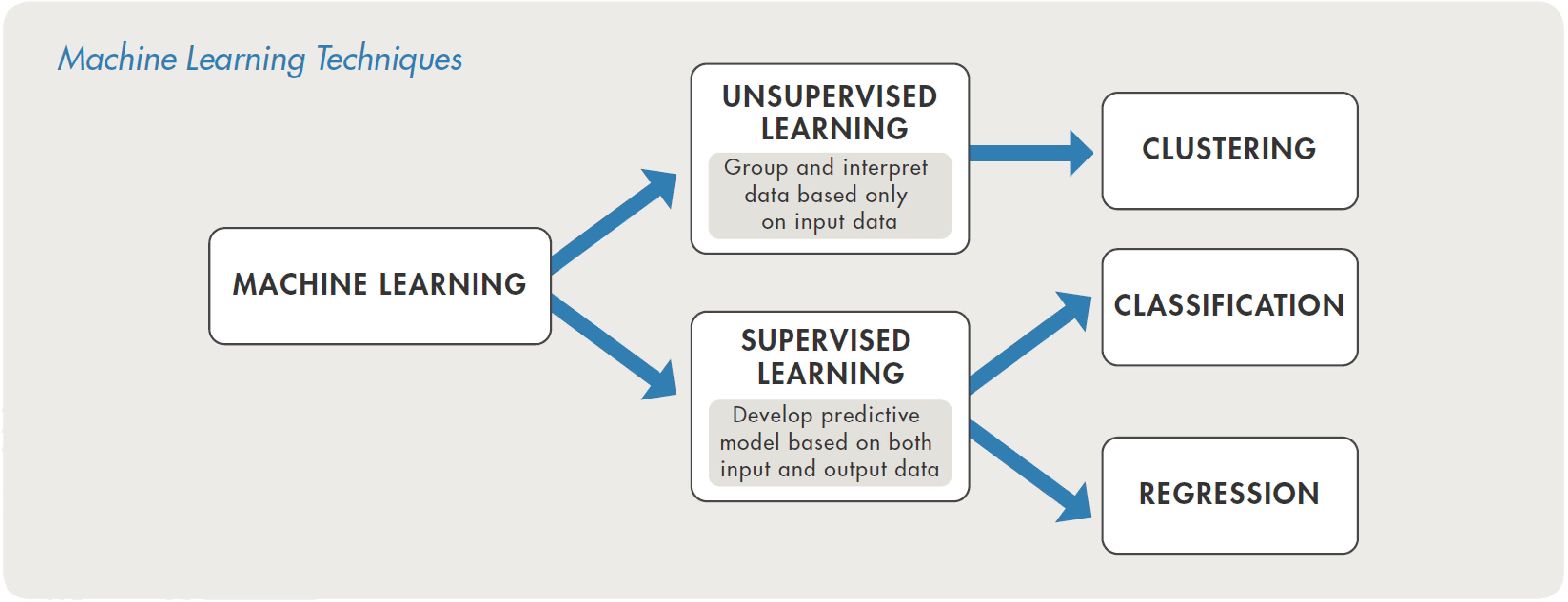
Lecture 10. Clustering and Classification

Alexander Zhigalov / Dept. of CS, University of Helsinki and Dept. of NBE, Aalto University

Outline / overview

Section 1. **Machine learning**

Machine learning


Machine Learning Techniques
MACHINE LEARNING
UNSUPERVISED LEARNING
Group and interpret data based only on input data
SUPERVISED LEARNING
Develop predictive model based on both input and output data
CLUSTERING
CLASSIFICATION
REGRESSION

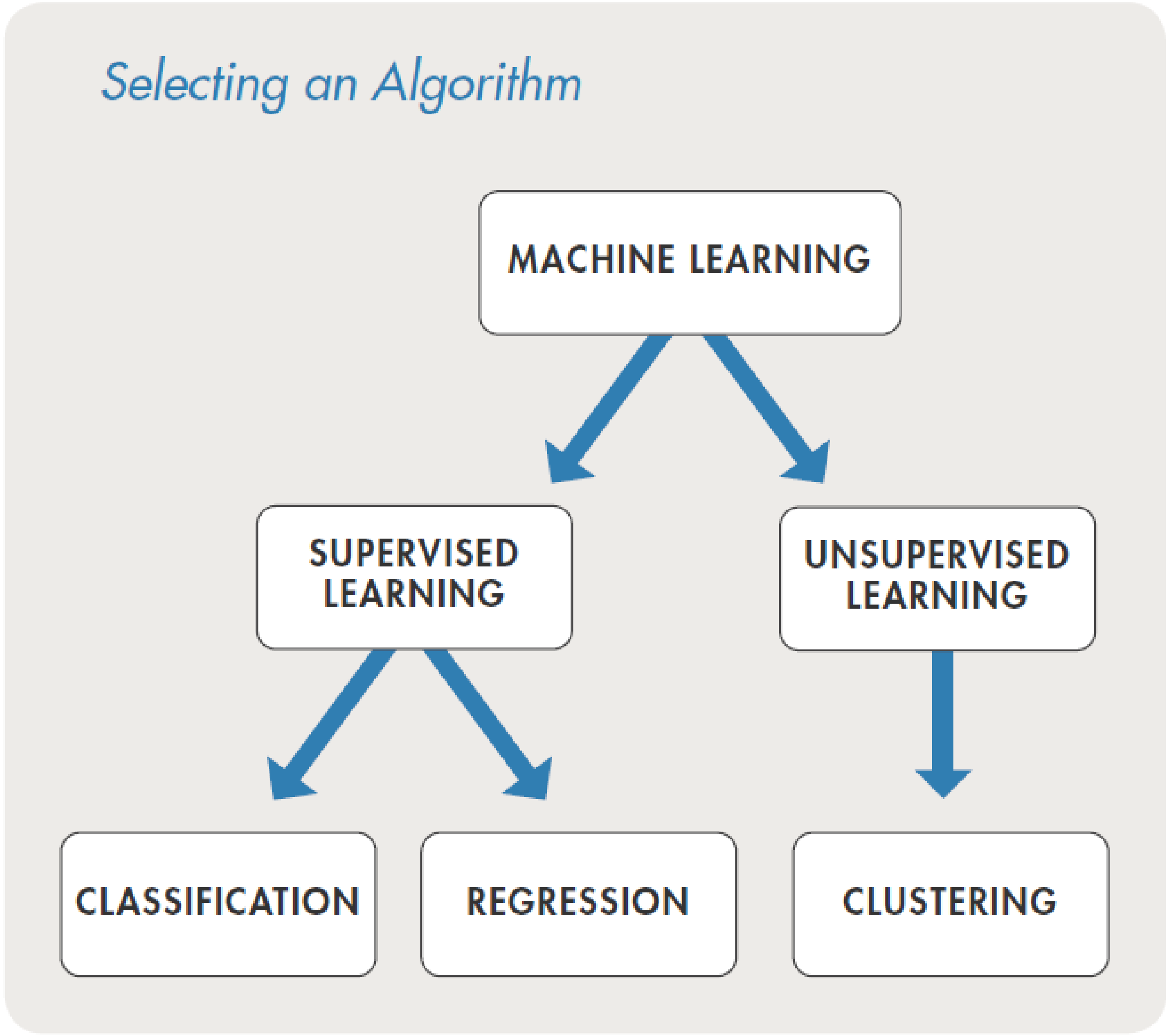
Introducing Machine Learning - MathWorks

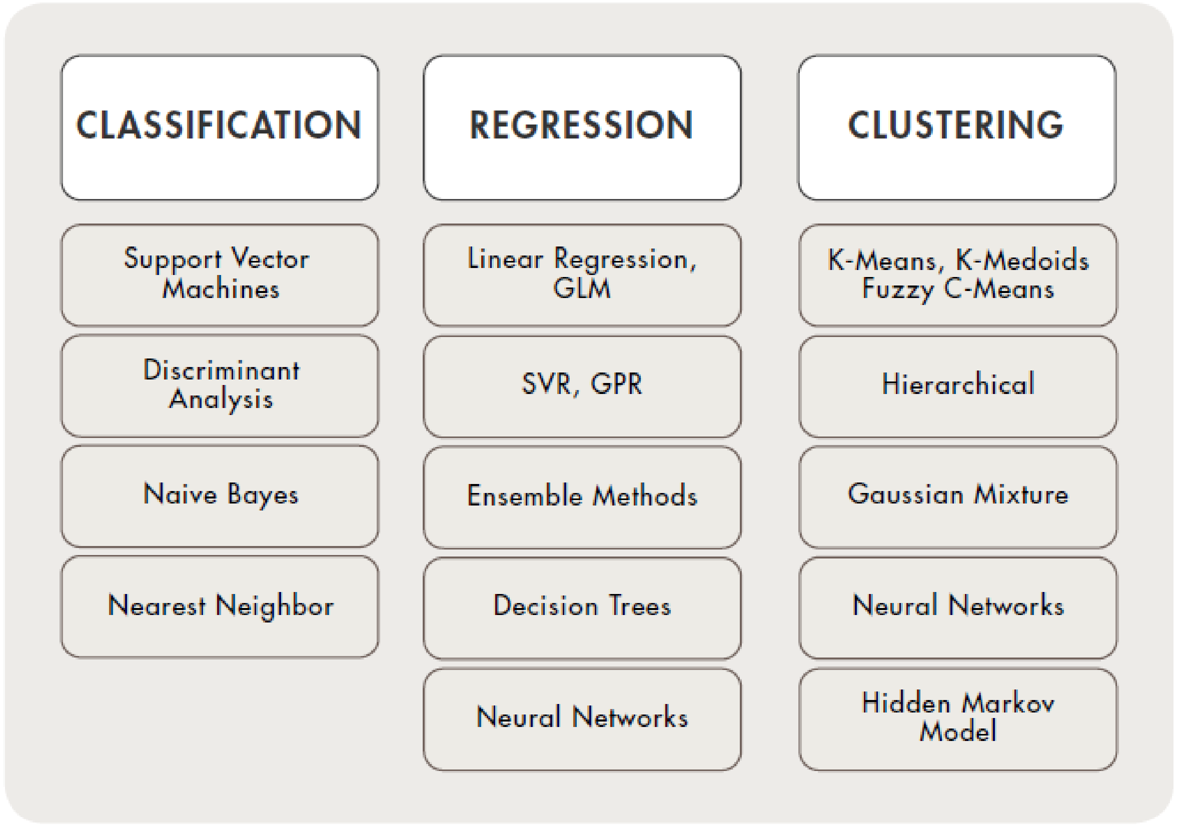
Machine learning

Unsupervised learning is useful when you want to explore your data but do not yet have a specific goal or are not sure what information the data contains.

A supervised learning algorithm takes a known set of input data (the training set) and known responses to the data (output), and trains a model to generate reasonable predictions for the response to new input data.

Machine learning





CLASSIFICATION	REGRESSION	CLUSTERING
Support Vector Machines	Linear Regression, GLM	K-Means, K-Medoids Fuzzy C-Means
Discriminant Analysis	SVR, GPR	Hierarchical
Naive Bayes	Ensemble Methods	Gaussian Mixture
Nearest Neighbor	Decision Trees	Neural Networks
	Neural Networks	Hidden Markov Model

Introducing Machine Learning - MathWorks

Section 2. Clustering

Clustering

In cluster analysis, data is partitioned into groups based on some measure of similarity or shared characteristic.

Clusters are formed so that objects in the same cluster are very similar and objects in different clusters are very distinct.

K-means clustering

How it works

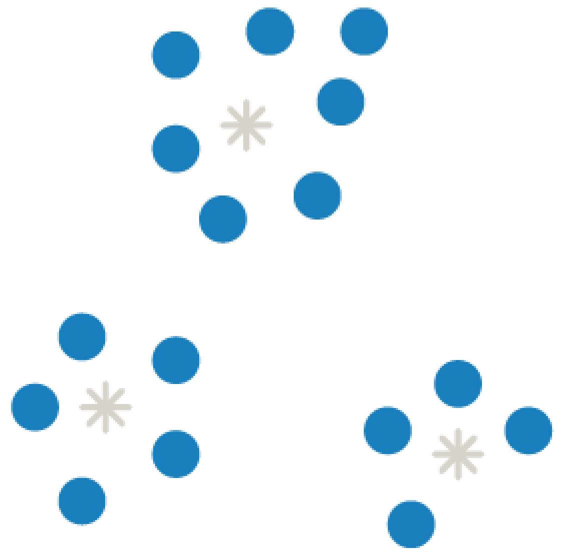
Partitions data into *k* number of mutually exclusive clusters. How well a point fits into a cluster is determined by the distance from that point to the cluster's center.

Best used ...

- When the number of clusters is known
- For fast clustering of large data sets

Result

Cluster centers



K-means clustering (1/2)

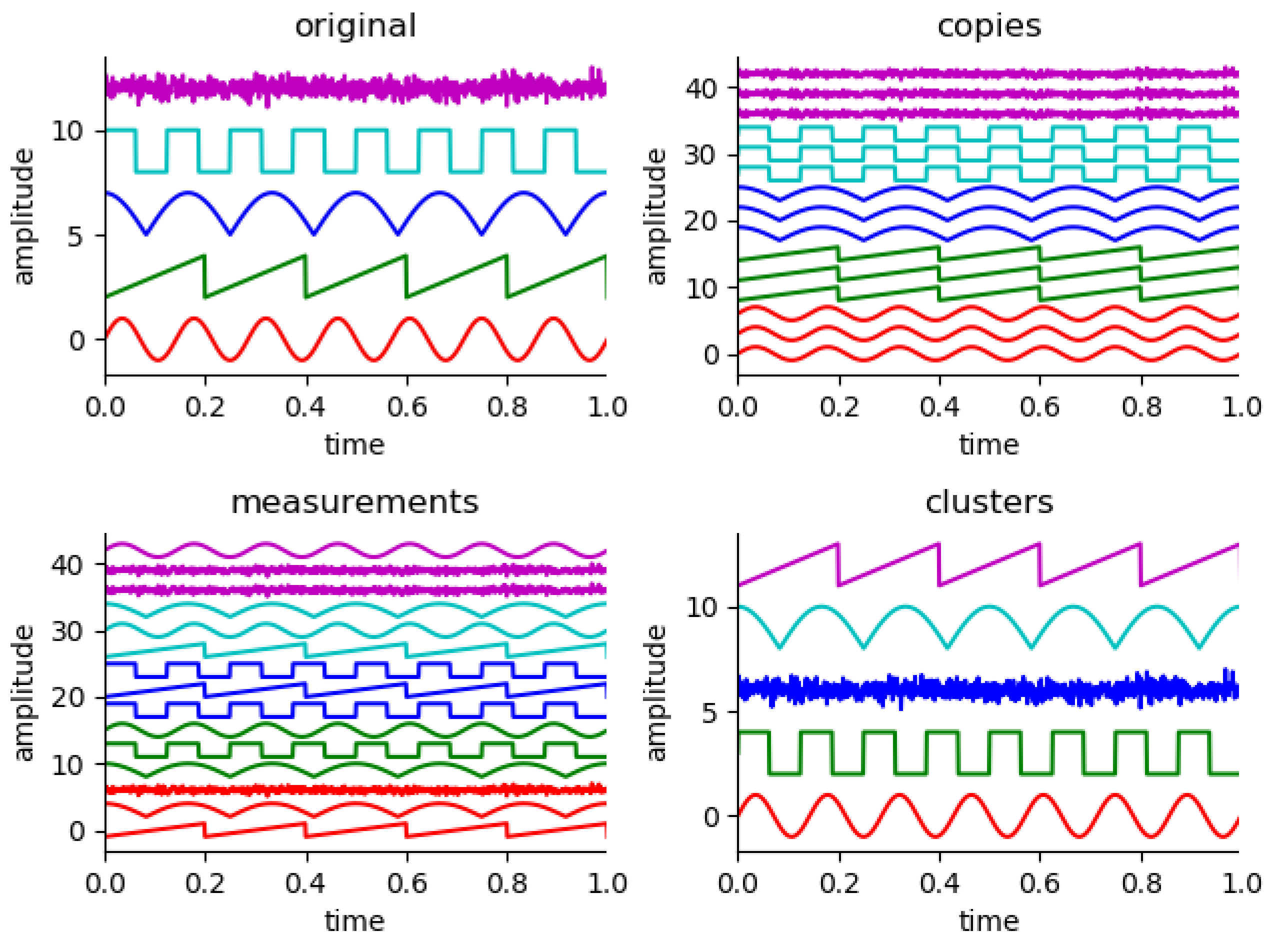
The number of clusters (K) is equal to the number of sources.

```
# create copies
X[i] = np.tile(S[i, :], (R, 1)) +
       np.random.randn(R, N) * SNR

# measurements
Y = X[np.random.permutation(M*R), :]


# clustering using sklearn
model = cluster.KMeans(n_clusters=K)
model.fit(Y)

# clustering outcome
labels = model.labels_
Z = model.cluster_centers_
inertia = model.inertia_
print(inertia) # within-cluster sum-of-squares
```



inertia = 0.0

See, “L10_clustering_kmeans.py”

K-means clustering (1/2)

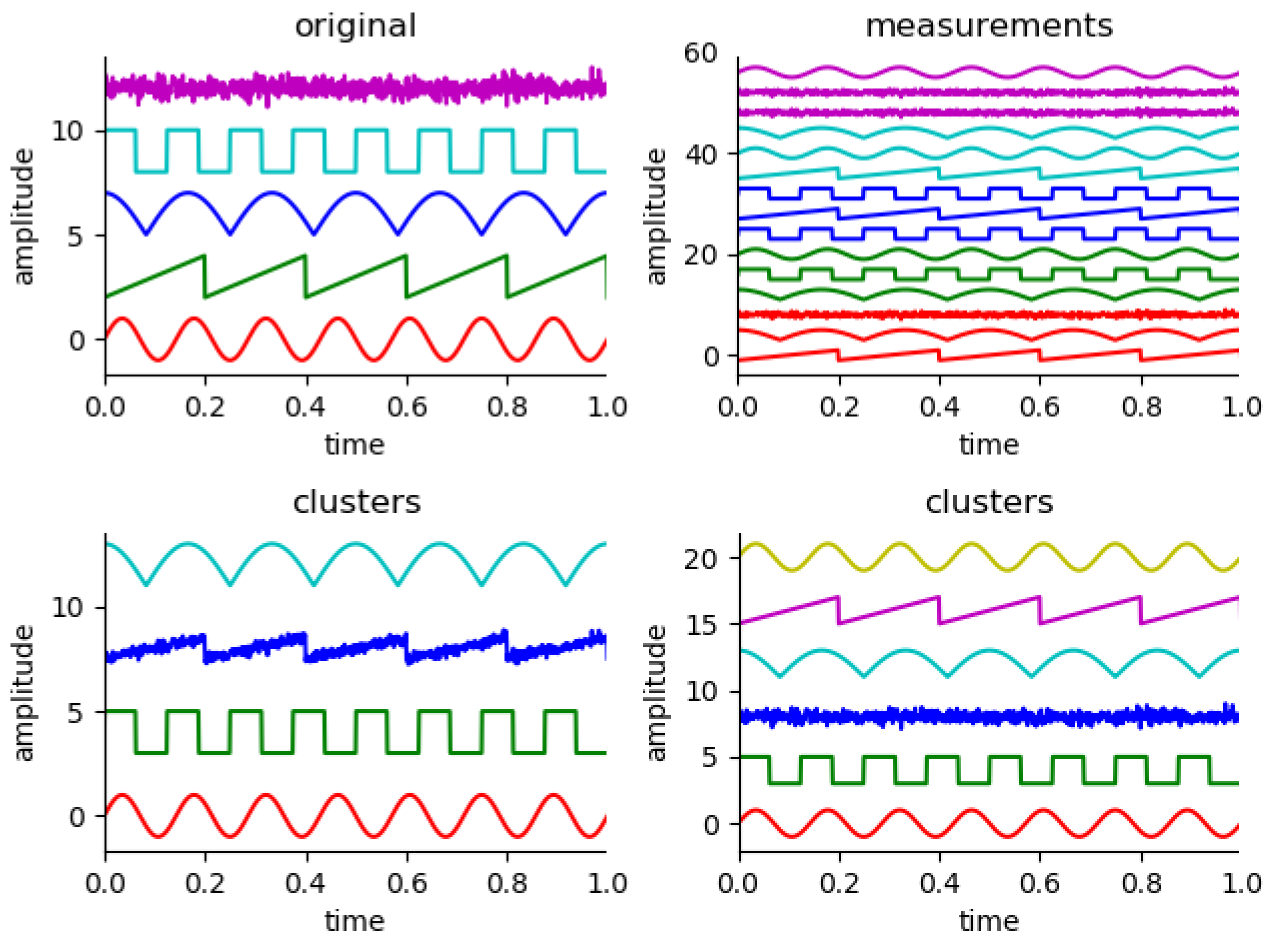
The number of clusters (K) is greater or less than the number of sources.

```
# create copies
X[i] = np.tile(S[i, :], (R, 1)) +
       np.random.randn(R, N) * SNR

# measurements
Y = X[np.random.permutation(M*R), :]


# clustering using sklearn
model = cluster.KMeans(n_clusters=K)
model.fit(Y)

# clustering outcome
labels = model.labels_
Z = model.cluster_centers_
inertia = model.inertia_
print(inertia)
```



inertia = 603.5 inertia = 0.0

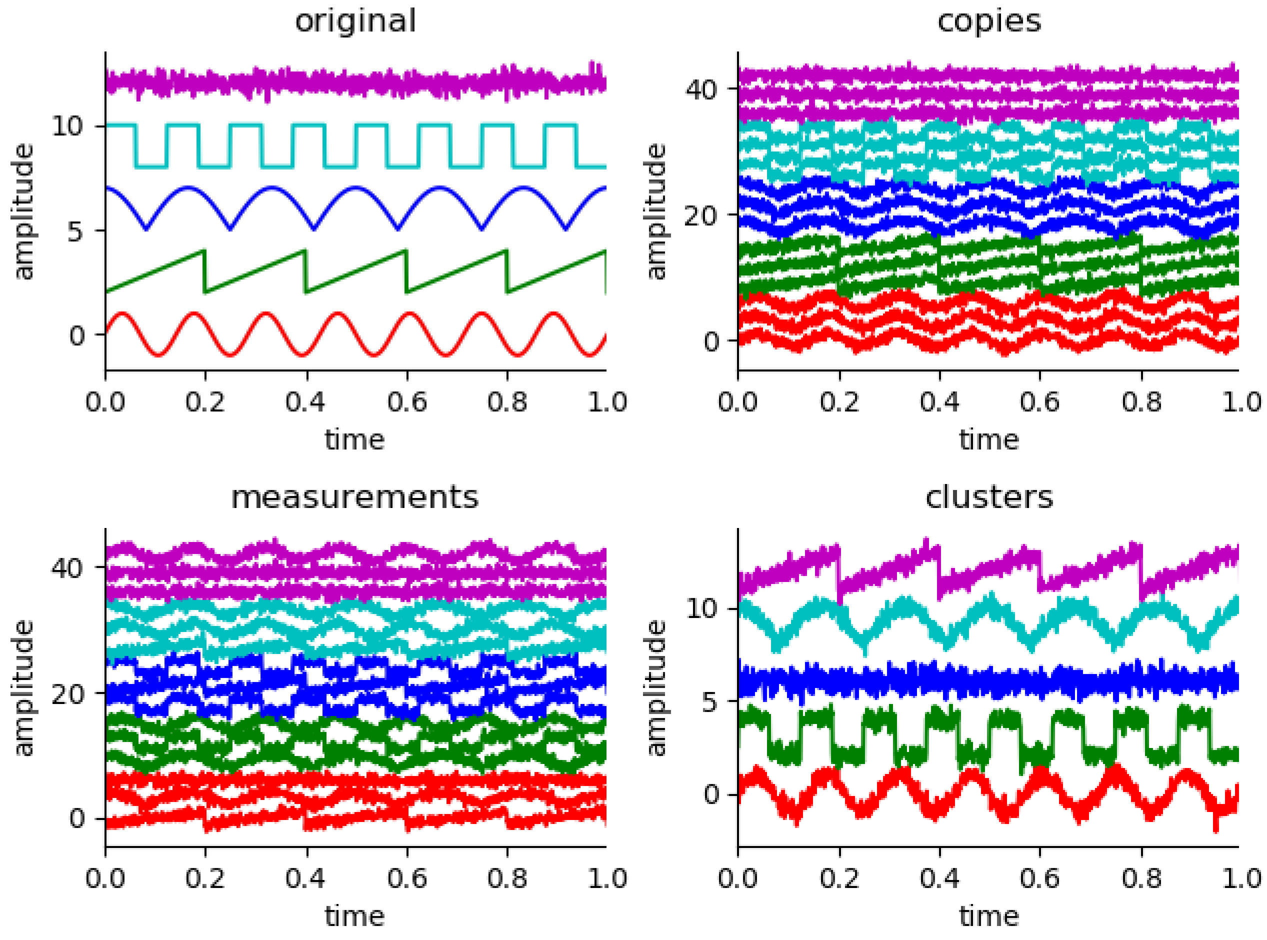
See, “L10_clustering_kmeans.py”

Noisy measurements (1/2)

How does it work in presence of noise?

```
# create copies
X[i] = np.tile(S[i, :], (R, 1)) +
       np.random.randn(R, N) * 0.5

# measurements
Y = X[np.random.permutation(M*R), :]
```



inertia = 2550.0

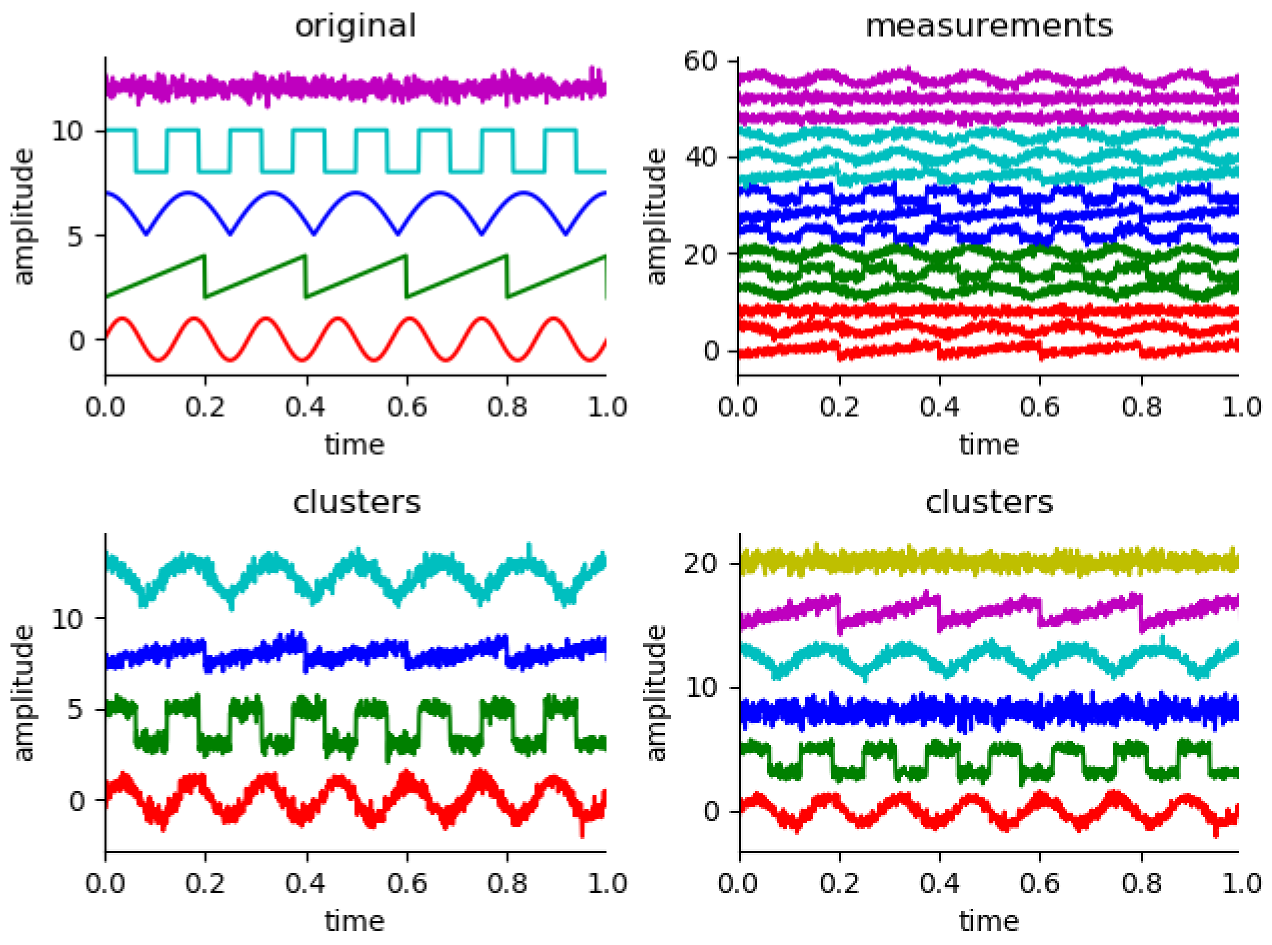
See, “L10_clustering_kmeans.py”

Noisy measurements (2/2)

Can the algorithm put noise to a single cluster?

```
# create copies
X[i] = np.tile(S[i, :], (R, 1)) +
       np.random.randn(R, N) * 0.5

# measurements
Y = X[np.random.permutation(M*R), :]
```



inertia = 3381.0

inertia = 2279.0

See, “L10_clustering_kmeans.py”

Hierarchical clustering

How it works

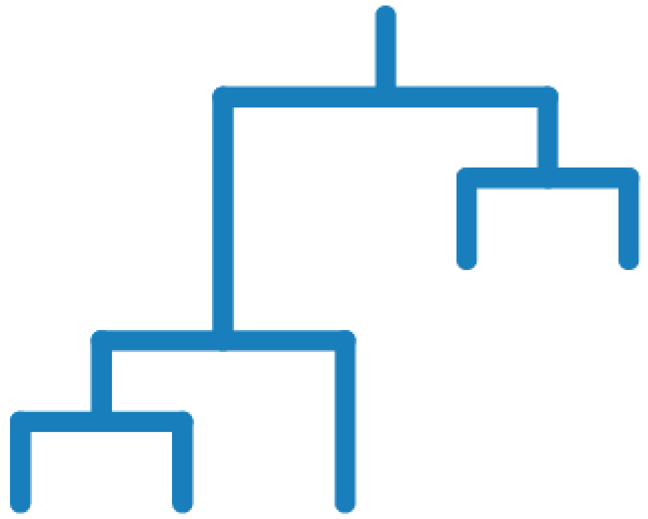
Produces nested sets of clusters by analyzing similarities between pairs of points and grouping objects into a binary, hierarchical tree.

Best used ...

- When you do not know in advance how many clusters are in your data
- You want visualization to guide your selection

Result

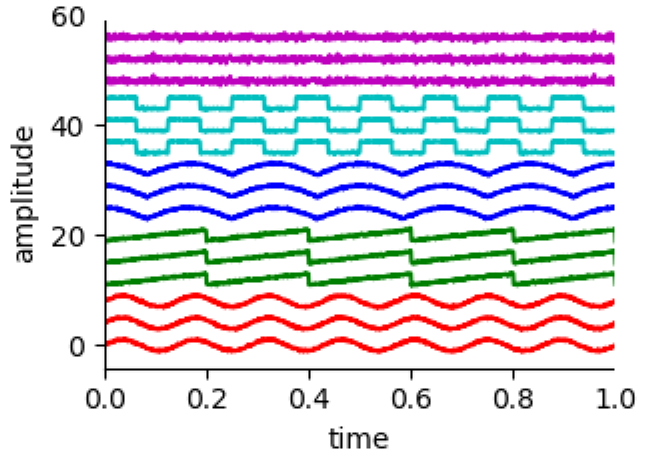
Dendrogram showing the hierarchical relationship between clusters

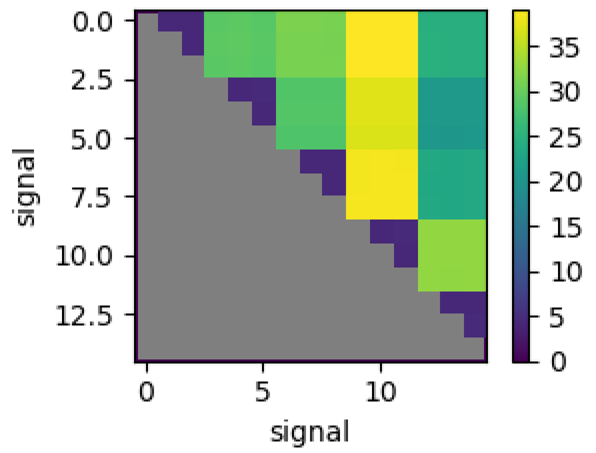


Hierarchical clustering (1/2)

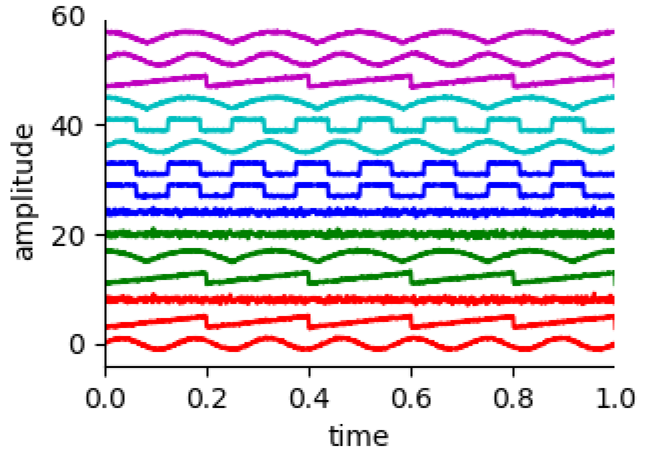
What are the distance measures between signals?

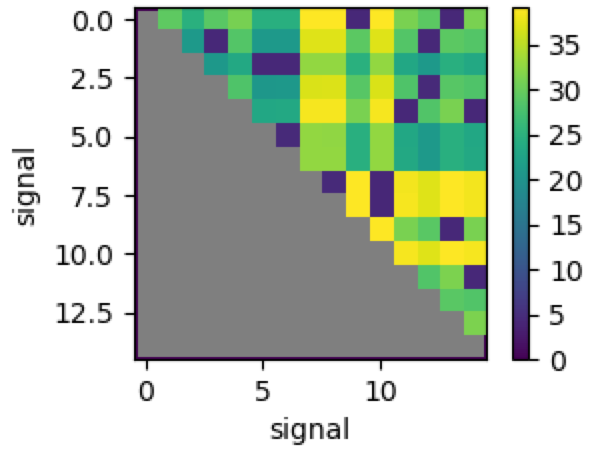
```
# pair-wise distance between signals
PX = np.zeros((MR, MR))
PX[np.triu_indices(MR, 1)] = pdist(X,
                                    'euclidean')
```





```
# distance after permutation
PY = np.zeros((MR, MR))
    PY[np.triu_indices(MR, 1)] = pdist(Y,
                                        'euclidean')
```



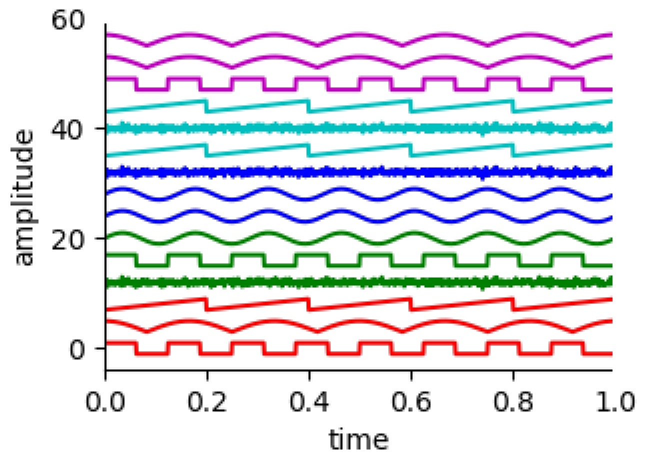


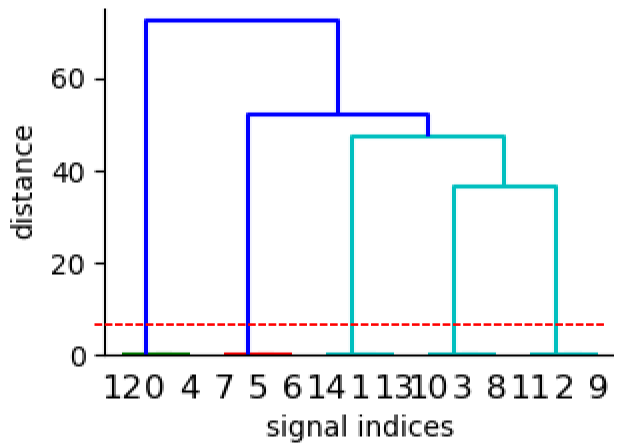
See, "L10_clustering_hierarchical.py"

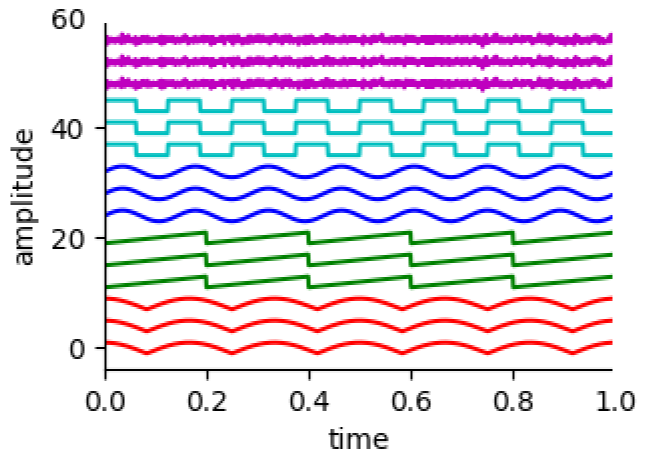
Hierarchical clustering (2/2)

How does it work?

```
# clustering
model =
  cluster.AgglomerativeClustering()
model.fit(Y)
labels = model.labels_
children = model.children_
```







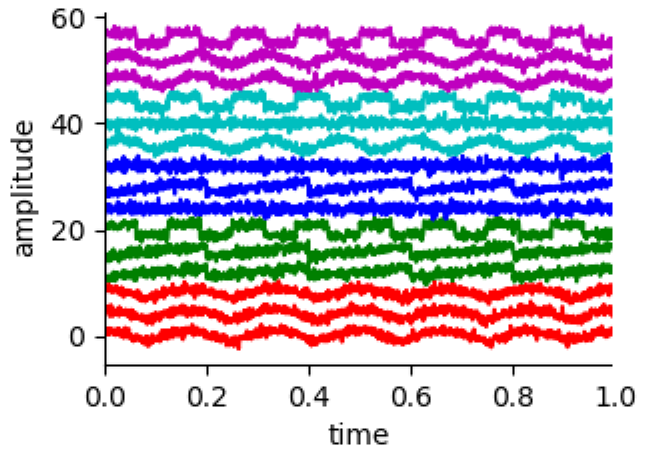
See, “L10_clustering_hierarchical.py”

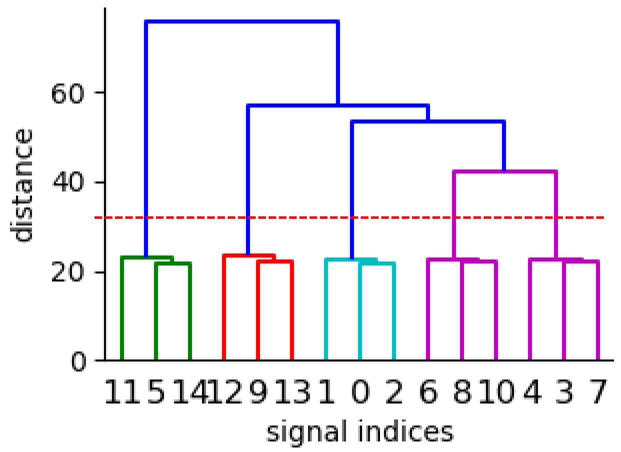
Noisy measurements

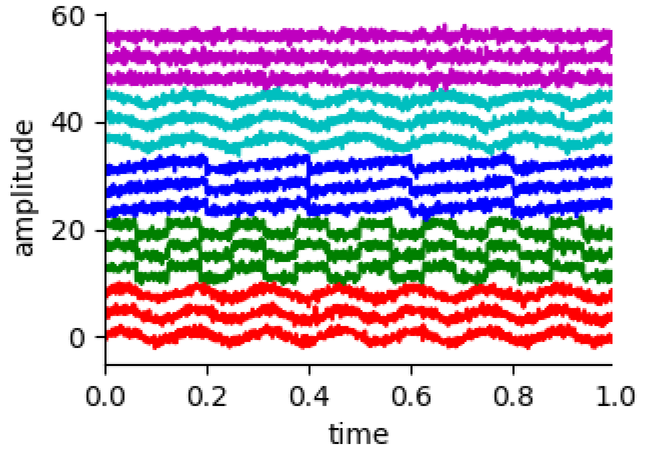
How does noise affect the clustering results?

```
# create copies
X[i] = np.tile(S[i, :], (R, 1)) +
       np.random.randn(R, N) * 0.5

# measurements
Y = X[np.random.permutation(M*R), :]
```





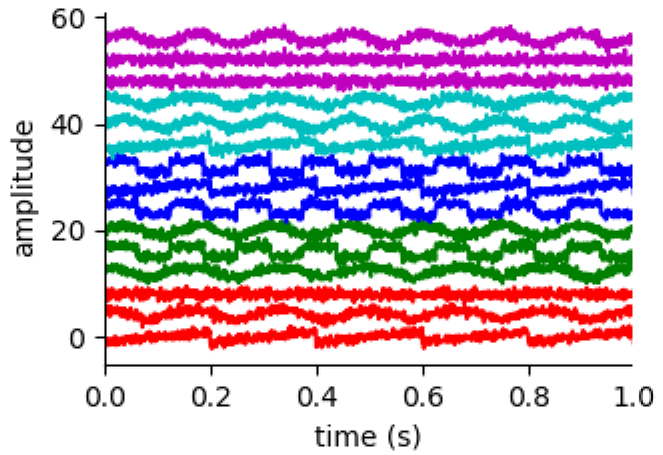


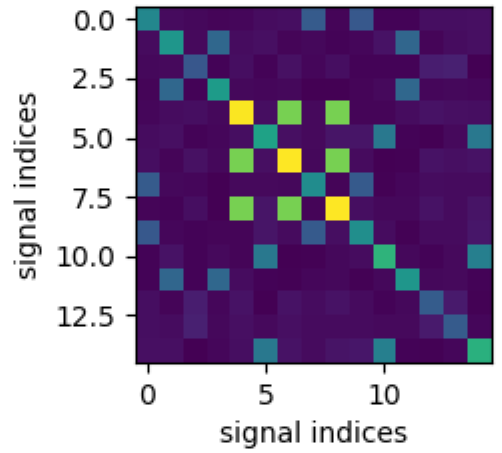
See, “L10_clustering_hierarchical.py”

Hierarchical clustering 2D (1/2)

Could we cluster the covariance matrix instead of sources?

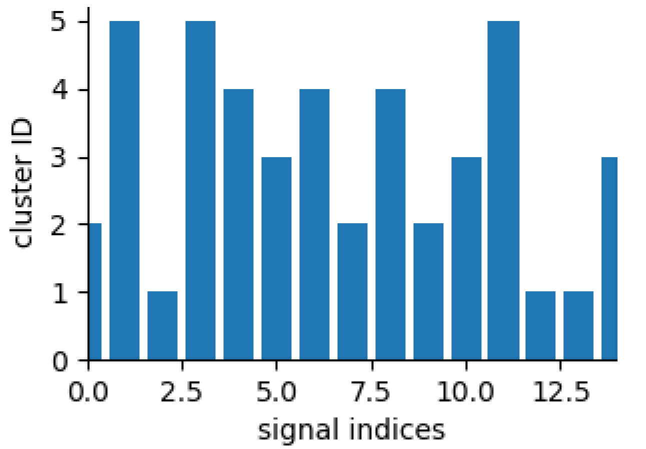
```
# covariance
CX = np.cov(X)
CY = np.cov(Y)
```

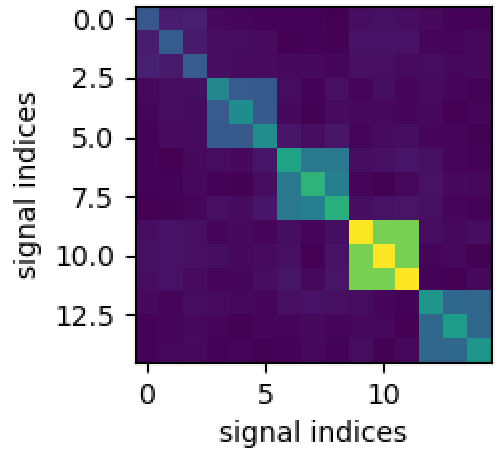




```
# clustering
model =
cluster.AgglomerativeClustering(n_clusters=K)
model.fit(Y)
labels = model.labels_

# sort matrix
indices = np.squeeze(np.argsort(labels))
CZ = CY
CZ = CZ[indices, :]
CZ = CZ[:, indices]
```





See, “L10_clustering_hierarchical_2D.py”

Hierarchical clustering 2D (2/2)

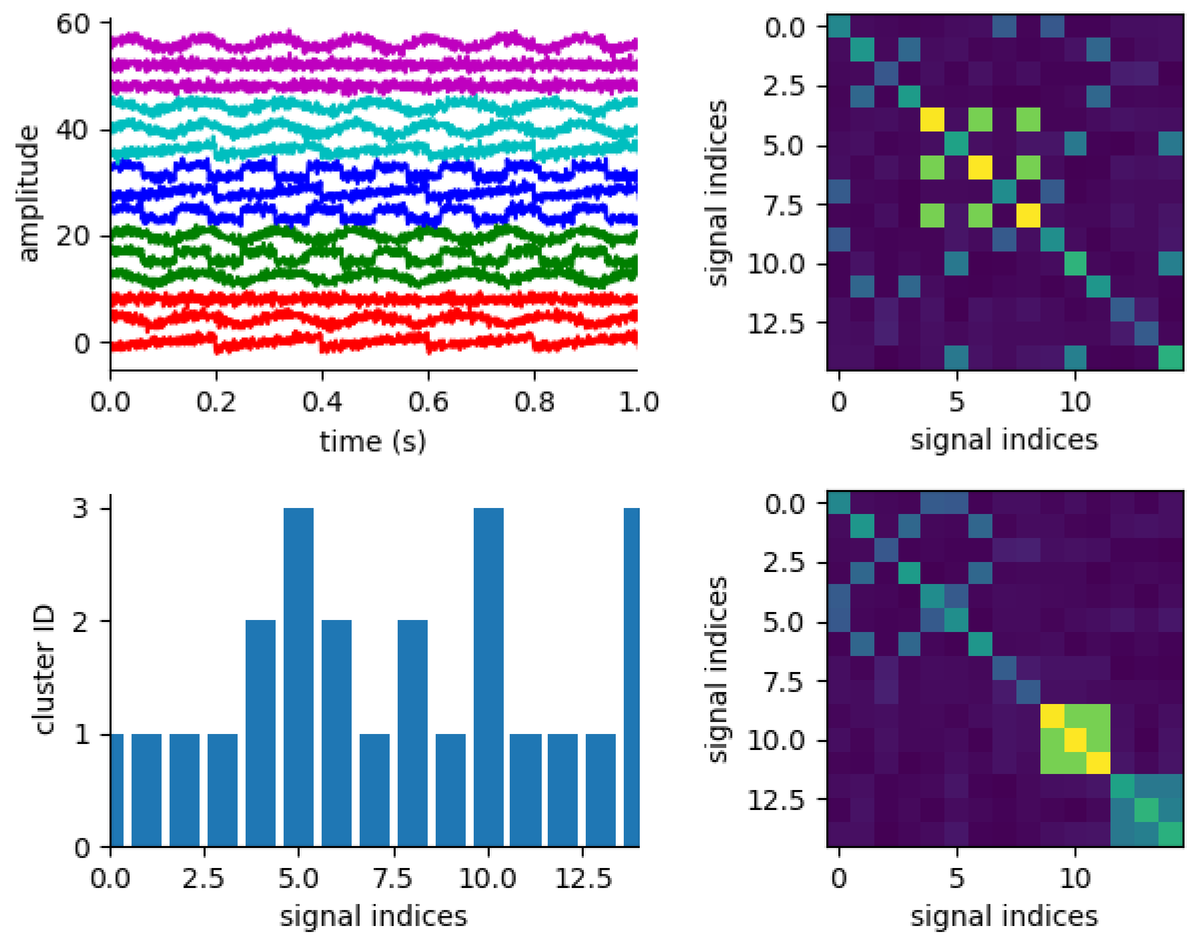
Sub-optimal number of clusters …

```
# covariance
CX = np.cov(X)
CY = np.cov(Y)



# clustering
model =
cluster.AgglomerativeClustering(n_clusters=K)
model.fit(Y)
labels = model.labels_

# sort matrix
indices = np.squeeze(np.argsort(labels))
CZ = CY
CZ = CZ[indices, :]
CZ = CZ[:, indices]
```



See, “L10_clustering_hierarchical_2D.py”

Gaussian Mixture Model

How it works

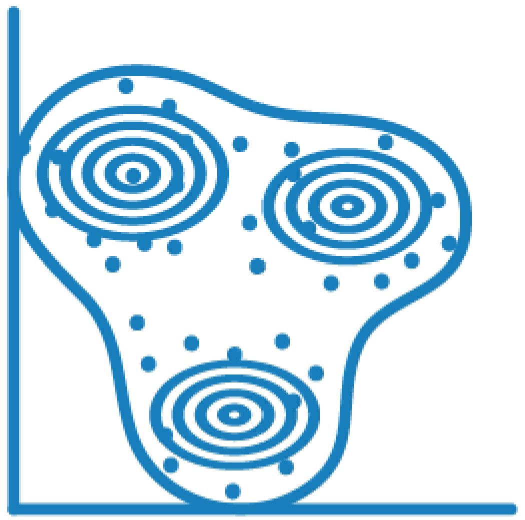
Partition-based clustering where data points come from different multivariate normal distributions with certain probabilities.

Best used ...

- When a data point might belong to more than one cluster
- When clusters have different sizes and correlation structures within them

Result

A model of Gaussian distributions that give probabilities of a point being in a cluster



Gaussian Mixture Model (1/2)

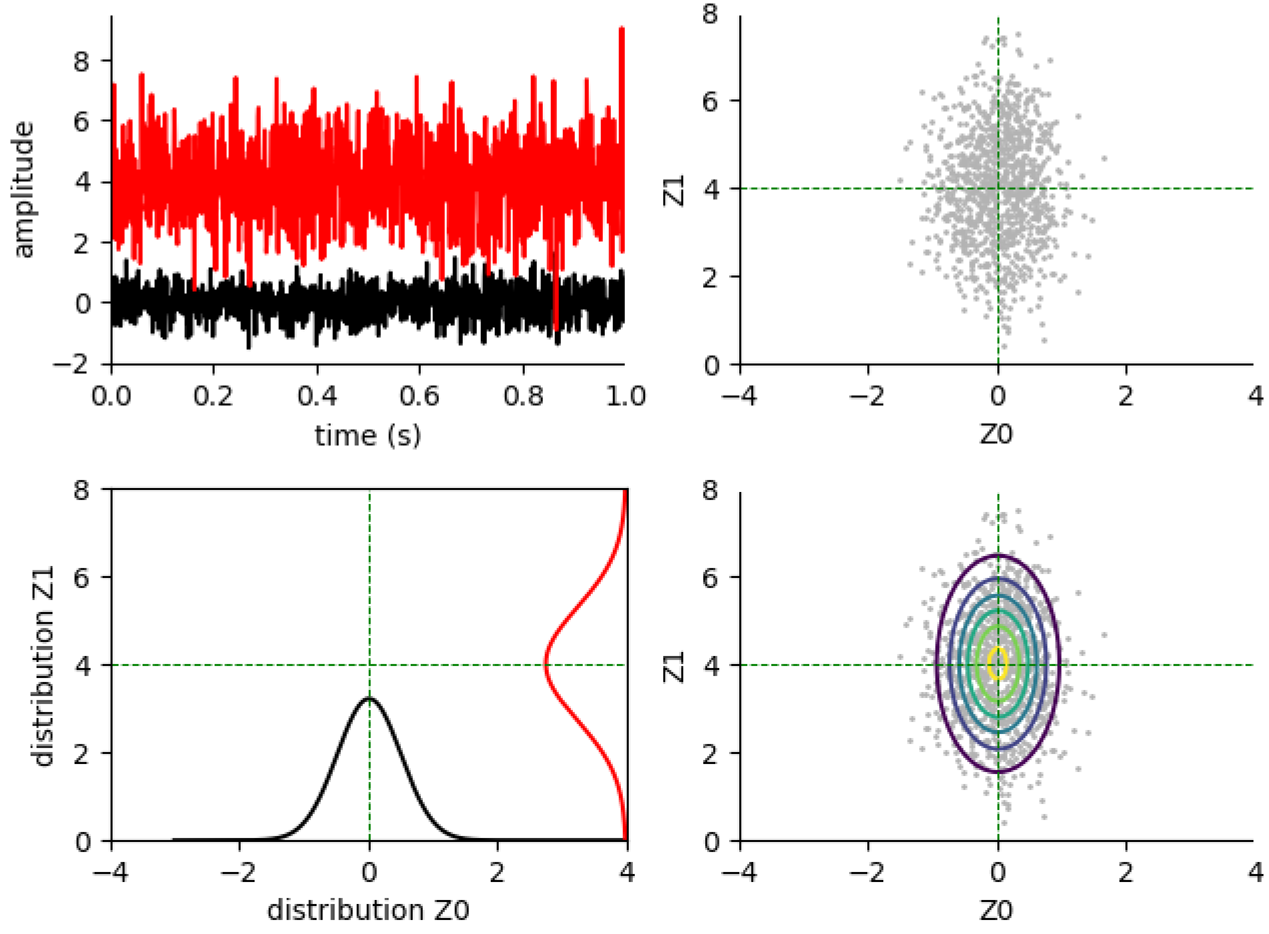
Why does it work?

```
# generate data
Z = [np.random.randn(1, N) * 0.5 + 0.0,
     np.random.randn(1, N) * 1.25 + 4.0]

# scatter plot (joint distribution)
plt.scatter(Z[0], Z[1])


# gaussian PDF
b = np.linspace(-3, 10, 1000)
p0 = norm.pdf(b, np.mean(Z[0]), np.std(Z[0]))
p1 = norm.pdf(b, np.mean(Z[1]), np.std(Z[1]))

# multivariate gaussian PDF
x, y = np.meshgrid(np.arange(-10.0, 10.0,
delta), np.arange(-10.0, 10.0, delta))
z = mlab.bivariate_normal(x, y, np.std(Z[0]),
np.std(Z[1]), np.mean(Z[0]), np.mean(Z[1]))
plt.contour(x, y, z)
```



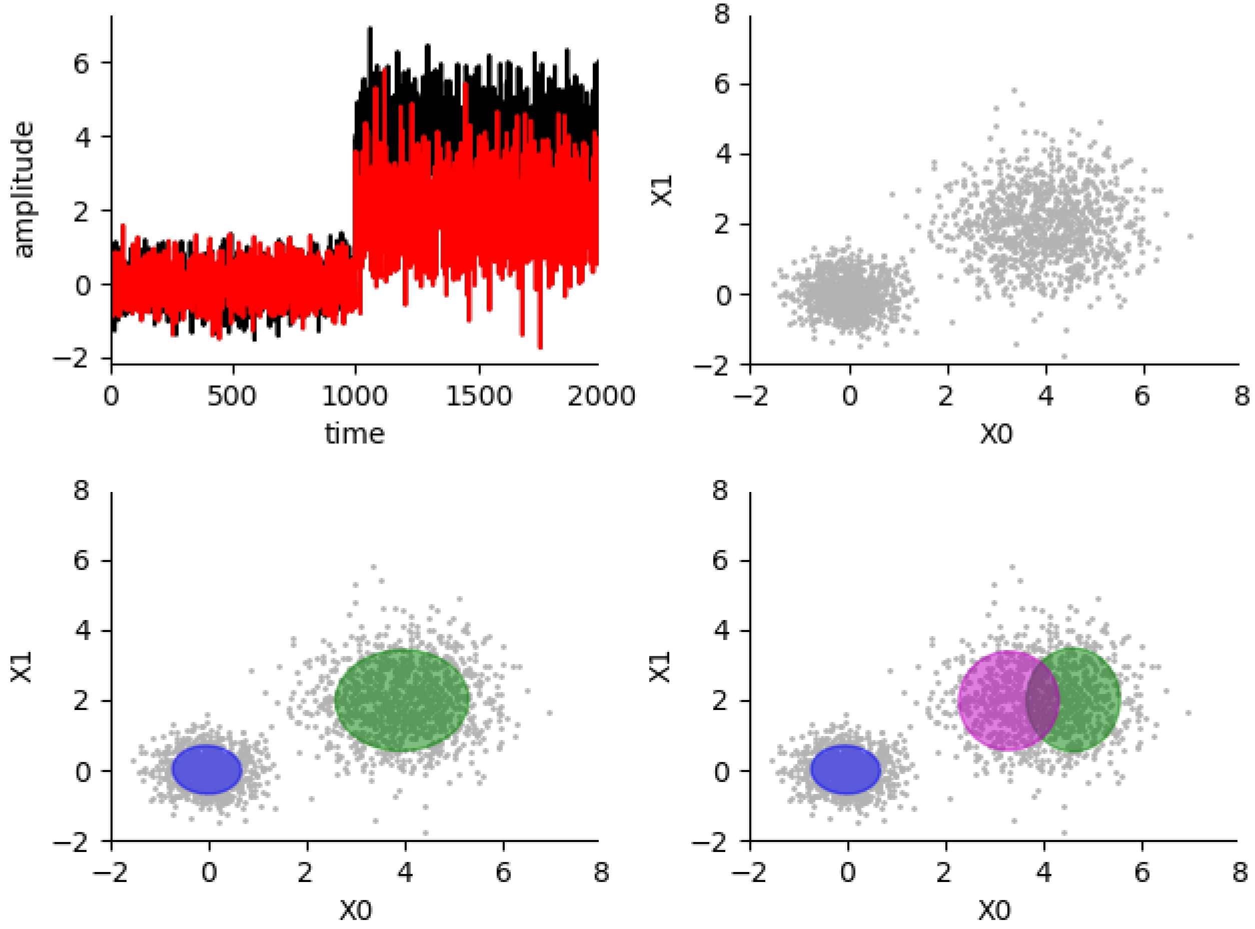
See, “L10_clustering_gmm.py”

Gaussian Mixture Model (2/2)

How does it work?

```
# fit model
model = mixture.GaussianMixture(n_components=K)
model.fit(X)

# model properties
Y = model.predict(X)
model_mu = model.means_
model_cov = model.covariances_
```



See, "L10_clustering_gmm.py"

Section 3. **Classification**

Classification

Classification techniques predict discrete responses, for example, whether an email is genuine or spam.

Classification models are trained to classify data into categories.

Support Vector Machine

How it works

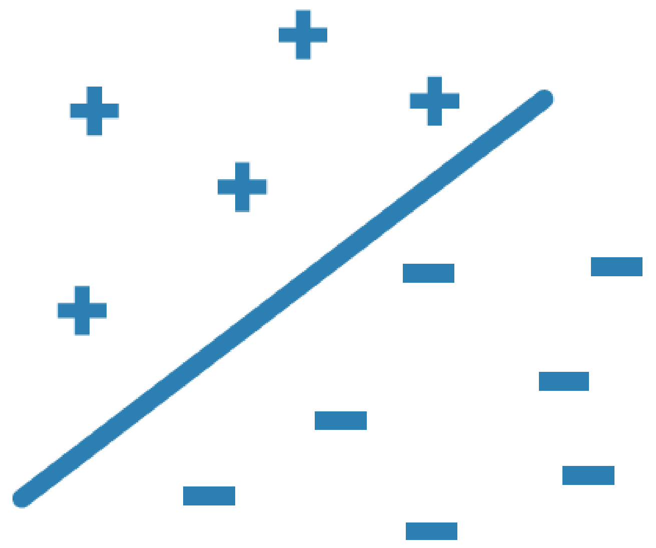
Classifies data by finding the linear decision boundary (hyperplane) that separates all data points of one class from those of the other class. The best hyperplane for an SVM is the one with the largest margin between the two classes.

Best used ...

- For data that has exactly two classes
- For high-dimensional, nonlinearly separable data
- When you need a classifier that is simple, easy to interpret and accurate

Result

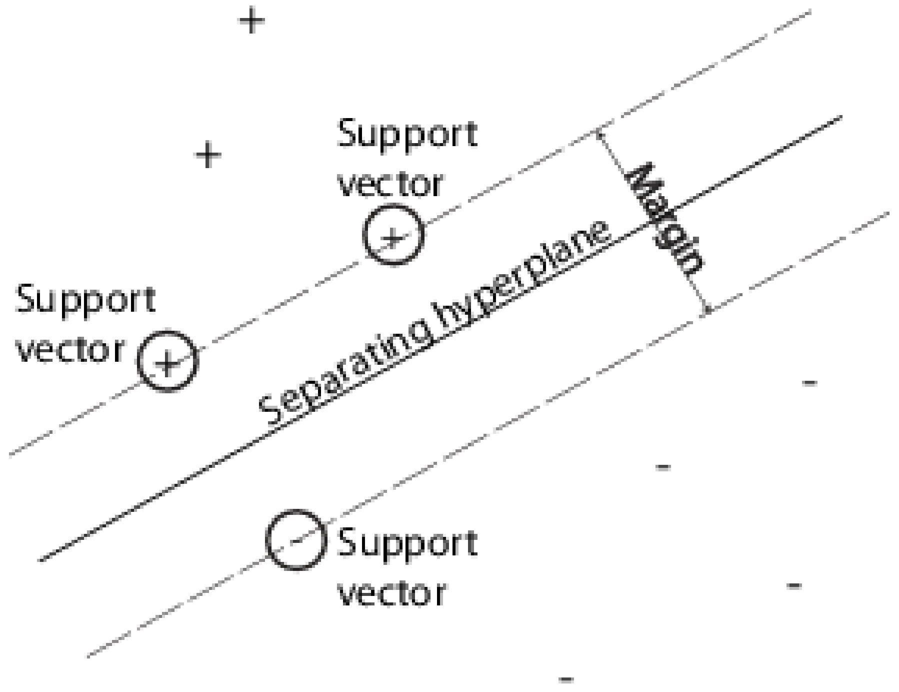
Training/fitting transforms data and labels to coefficients, while testing/prediction transforms data and coefficients to labels.



Support Vector Machine

When it works

SVM works if data has exactly two classes.



Support Vector Machine (1/2)

SVM as any other classification approach consists of two stages: training and testing.

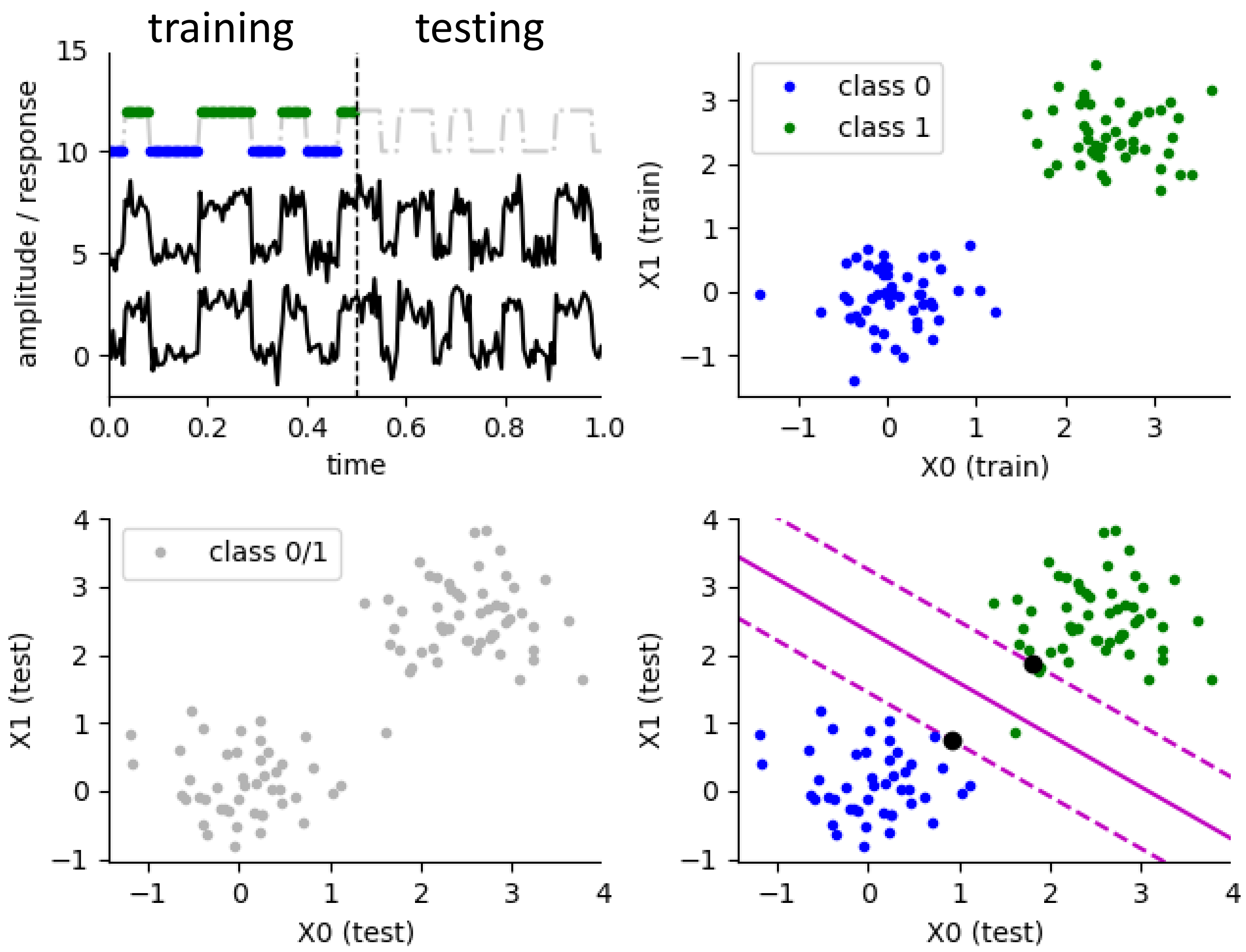
```
# data
X = np.random.randn(M, N)

# binary labels
y = get_sequence(5, 0.8, N)

# induce some correlation between X and y
X = X + 2.0 * np.tile(y, (M, 1))

# training and testing datasets
L = N // 2
Y = y[:L] # training labels
U = y[L:] # testing labels
XY = X[:, :L] # training data
XU = X[:, L:] # testing data

# train classifier
model = SVC(kernel='linear')
model.fit(XY.T, Y)
```



See, “L10_classification_svm_2_signals.py”

Support Vector Machine (2/2)

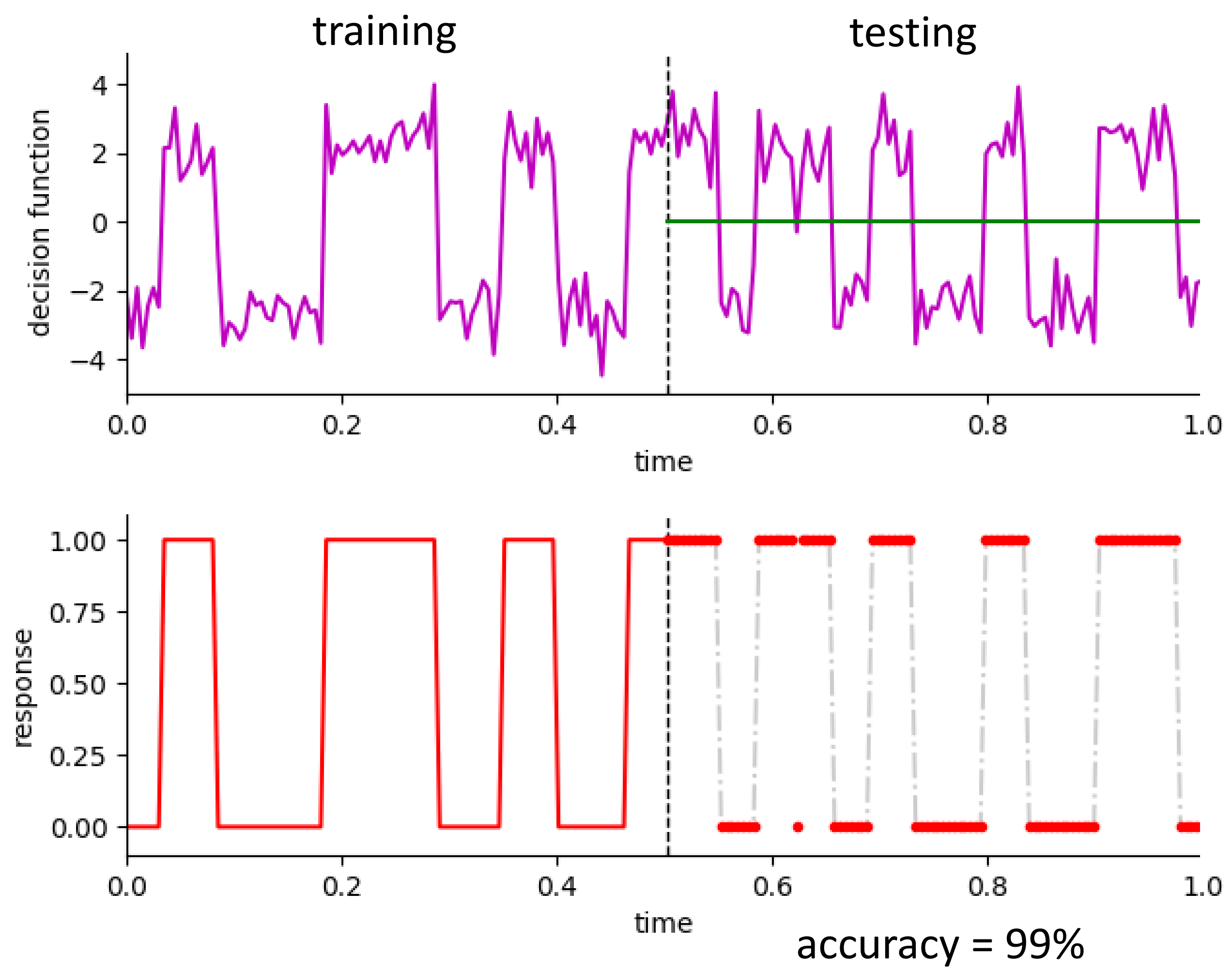
The classifier gives the coefficients that can be converted to the decision function.

```
# classifier outcome
coef = model._get_coef()
intercept = model.intercept_

# decision function
Z = np.zeros(N)
for i in range(0, N):
  Z[i] = np.sum(X[:, i] * coef) + intercept

# testing
v = U
u = model.predict(XU.T)
u = u > 0.5

# accuracy
a = np.mean(v == u)
print('accuracy: %1.2f' % (a))
```



See, “L10_classification_svm_2_signals.py”

Classification accuracy (1/2)

What if two datasets cannot be clearly separated?

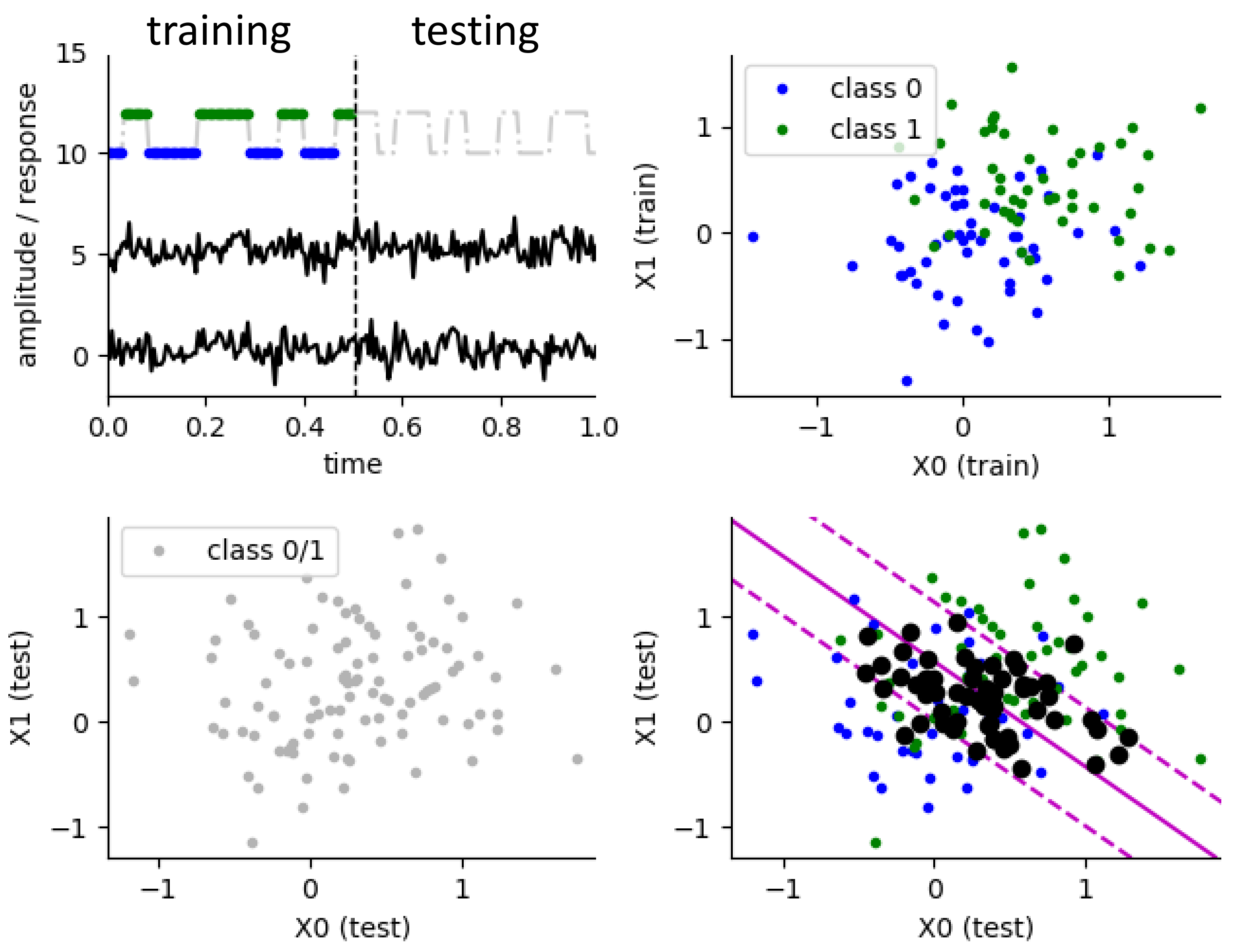
```
# data
X = np.random.randn(M, N)

# binary labels
y = get_sequence(5, 0.8, N)

# induce some correlation between X and y
X = X + 2.0 * np.tile(y, (M, 1))

# training and testing datasets
L = N // 2
Y = y[:L] # training labels
U = y[L:] # testing labels
XY = X[:, :L] # training data
XU = X[:, L:] # testing data

# train classifier
model = SVC(kernel='linear')
model.fit(XY.T, Y)
```



See, “L10_classification_svm_2_signals.py”

Classification accuracy (2/2)

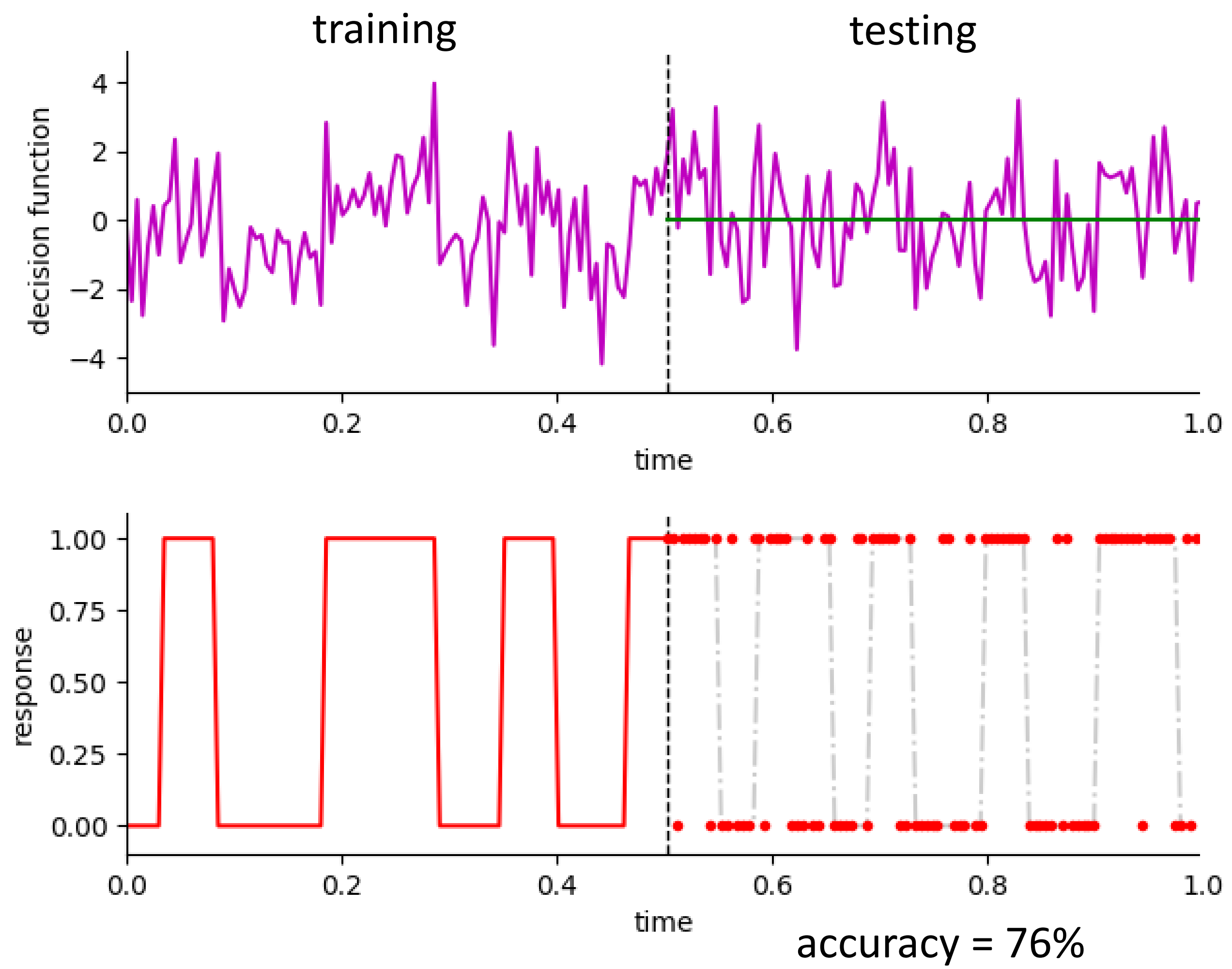
The decision function looks like a random noise.

```
# classifier outcome
coef = model._get_coef()
intercept = model.intercept_

# decision function
Z = np.zeros(N)
for i in range(0, N):
  Z[i] = np.sum(X[:, i] * coef) + intercept

# testing
v = U
u = model.predict(XU.T)
u = u > 0.5

# accuracy
a = np.mean(v == u)
print('accuracy: %1.2f' % (a))
```



See, “L10_classification_svm_2_signals.py”

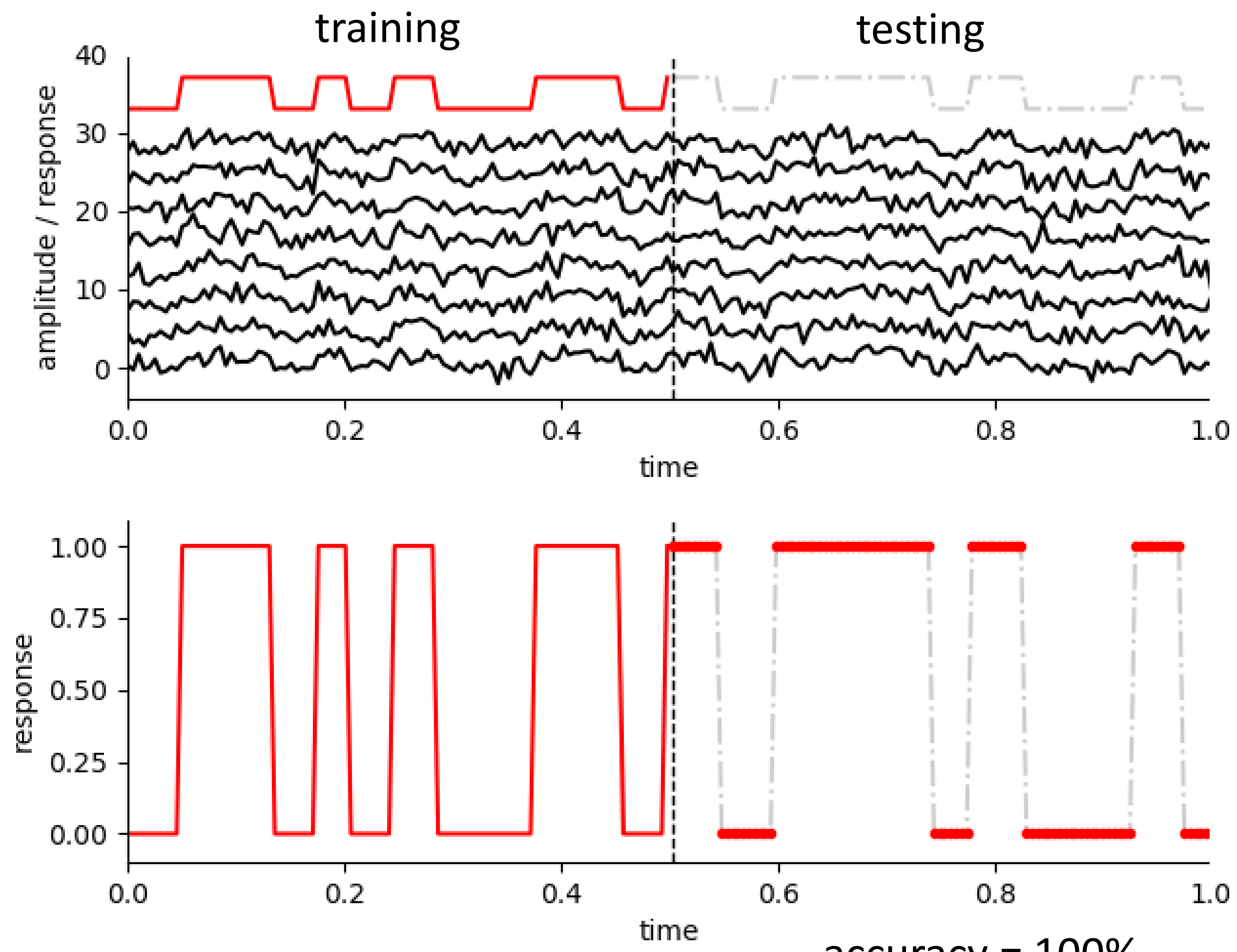
Multichannel recordings (1/2)

More channels could provide a better fit of the model but also overfitting may occur.

```
# data
X = np.random.randn(M, N)

# binary labels
y = get_sequence(5, 0.8, N)

# induce some correlation between X and y
X = X + 2.0 * np.tile(y, (M, 1))
```



accuracy = 100%

See, “L10_classification_svm.py”

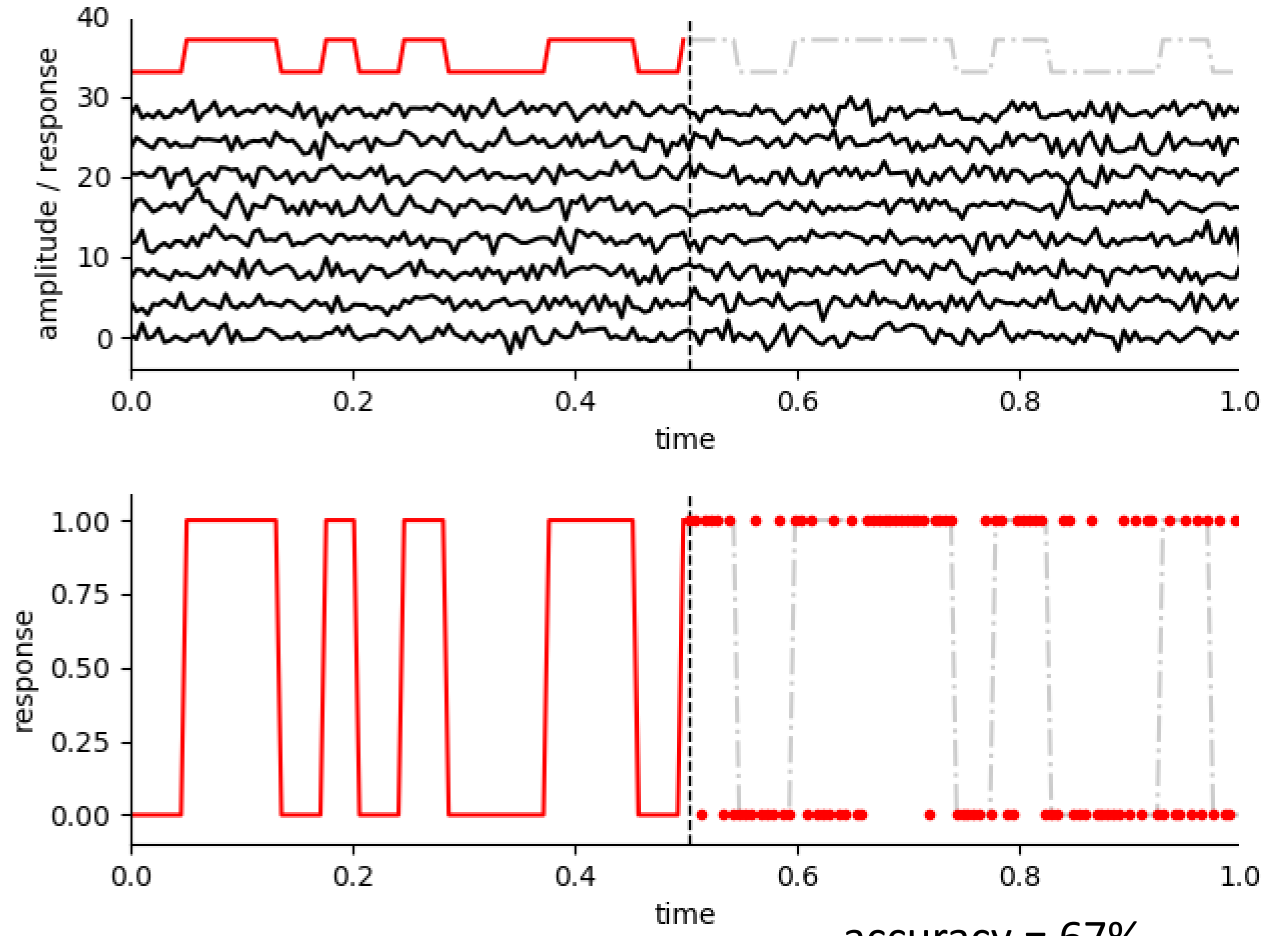
Classification accuracy (2/3)

In case of weak correlation between data and labels …

```
# data
X = np.random.randn(M, N)

# binary labels
y = get_sequence(5, 0.8, N)

# induce some correlation between X and y
X = X + 0.5 * np.tile(y, (M, 1))
```



accuracy = 67%

See, “L10_classification_svm.py”

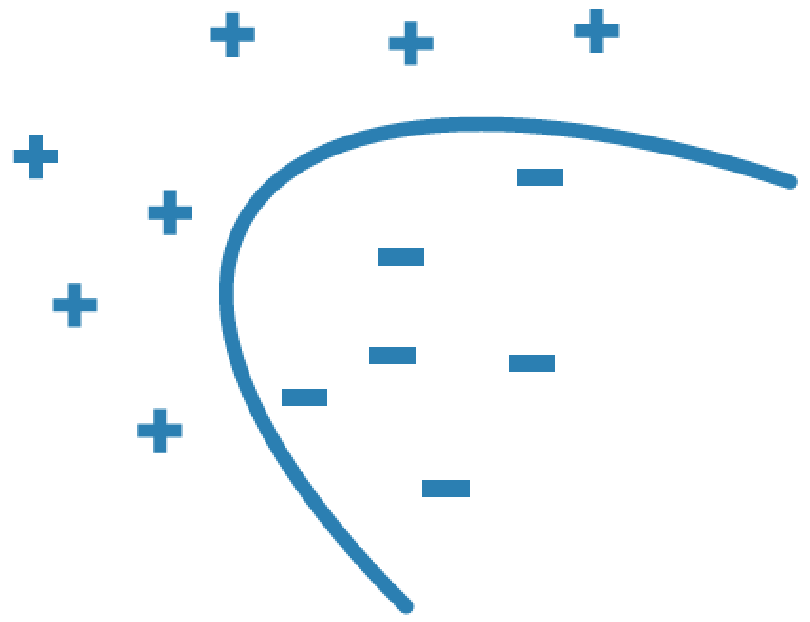
Discriminant Analysis

How it works

Discriminant analysis classifies data by finding linear combinations of features. Discriminant analysis assumes that different classes generate data based on Gaussian distributions.

Best used ...

- When you need a simple model that is easy to interpret
- When you need a model that is fast to predict



Result

Coefficients

Logistic regression

How it works

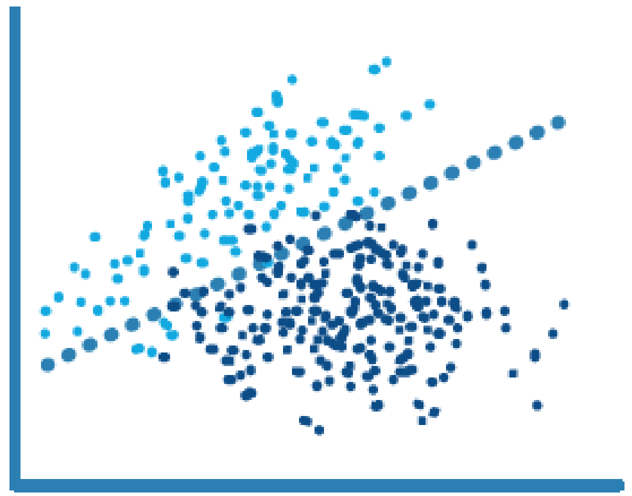
Fits a model that can predict the probability of a binary response belonging to one class or the other. Because of its simplicity, logistic regression is commonly used as a starting point for binary classification problems.

Best used ...

- When data can be clearly separated by a single, linear boundary
- As a baseline for evaluating more complex classification methods

Result

Coefficients



Section 4. **Regression**

Regression

Regression techniques predict continuous responses, for example, changes in temperature or fluctuations in electricity demand.

Linear regression

How it works

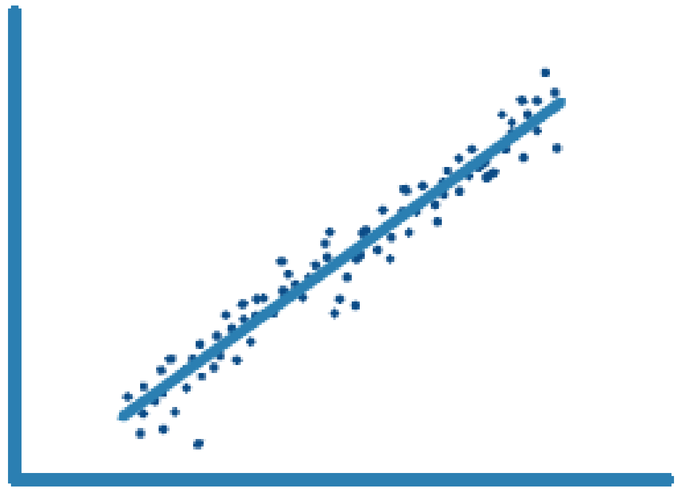
Linear regression is a statistical modeling technique used to describe a continuous response variable as a linear function of one or more predictor variables.

Best used ...

- When you need an algorithm that is easy to interpret and fast to fit
- As a baseline for evaluating other, more complex, regression models

Result

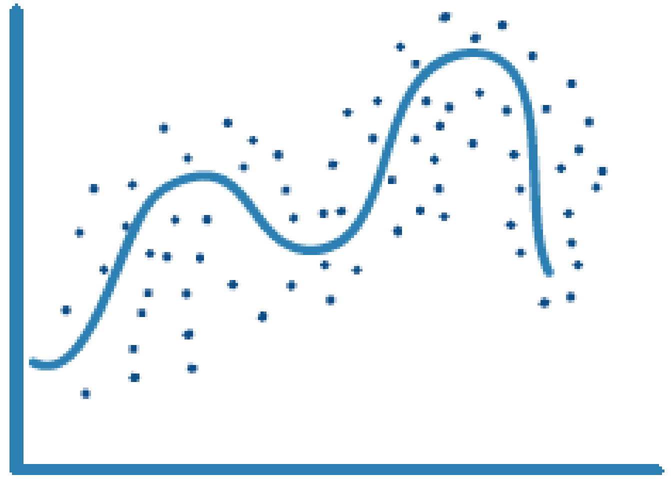
Coefficients



Nonlinear regression

How it works

Nonlinear regression is a statistical modeling technique that helps describe nonlinear relationships in experimental data.



Best used ...

- When data has strong nonlinear trends and cannot be easily transformed into a linear space
- For fitting custom models to data

Result

Coefficients

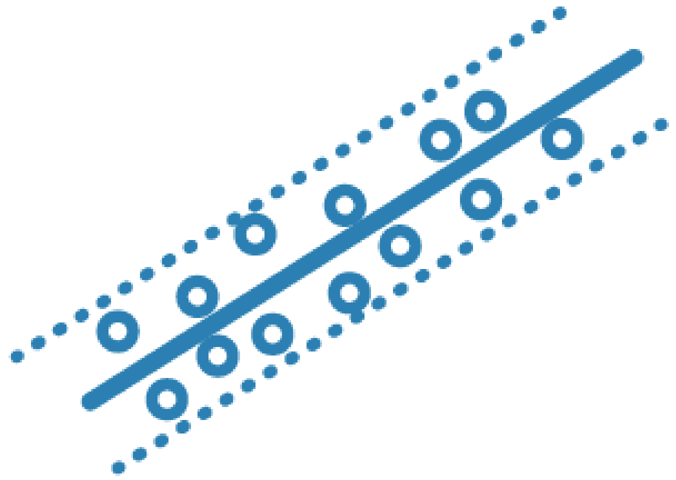
SVM regression

How it works

SVM regression algorithms work like SVM classification algorithms, but are modified to be able to predict a continuous response.

Best used ...

- For high-dimensional data with a large number of predictor variables



Result

Coefficients

Literature

- **Python programming language**

- http://www.scipy-lectures.org/, see “materials/L02_ScipyLectures.pdf”

- **Data analysis**

- Andreas Müller and Sarah Guido “**Introduction to Machine Learning with Python: A Guide for Data Scientists**”