EE 610, ST: ML Fundamentals

Nearest Neighbor Classifiers

Dr. W. David Pan

Dept. of ECE

UAH

Probability Density Estimation

- Pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories.
- Probability theory plays a central role in the solution of pattern recognition problems.
- Density Estimation
  - Model the probability distribution $p(x)$ of a random variable x, given a finite set $x_1, \ldots, x_N$ of observations.
  - Assume that the data points are independent and identically
  - distributed.
- The problem of density estimation is fundamentally ill-posed, because there are infinitely many probability distributions that could have given rise to the observed finite data set.
- The central issue of pattern recognition to choose an appropriate distribution relates to the problem of model selection.

Parametric Approaches

- Example models
  - Discrete random variables: binomial distribution and multinomial distribution
  - Continuous random variables: Gaussian distribution
- These are examples of *parametric* distributions, because they are governed by a small number of adaptive parameters, e.g., mean and variance in the case of a Gaussian.
- One limitation of the parametric approach is that it assumes a specific functional form for the distribution, which may turn out to be inappropriate for a particular application.
- The distribution chosen might be a poor model of the distribution that generates the data, which can result in poor predictive performance.
  - For example, if the process that generates the data is multimodal, then this aspect of the distribution can never be captured by a Gaussian, which is necessarily unimodal.

Nonparametric Methods

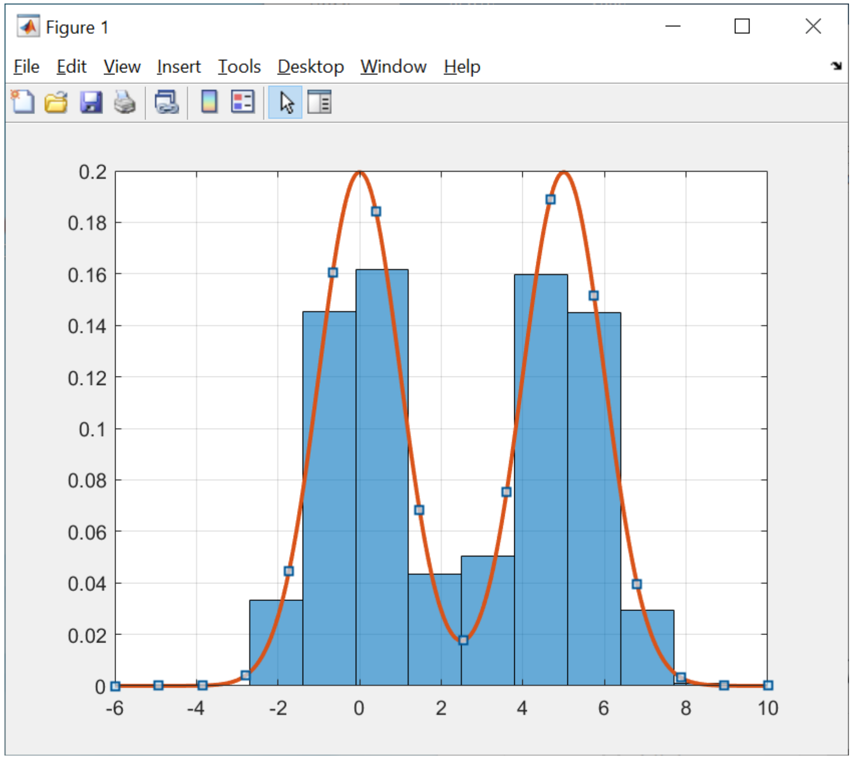
- An alternative approach is *nonparametric* density estimation methods, where the form of the distribution typically depends on the size of the dataset.
- Such models still contain parameters, but these control the model complexity rather than the form of the distribution.
- Examples
  - Histogram methods for density estimation
  - Nearest-Neighbor methods

Histogram Methods

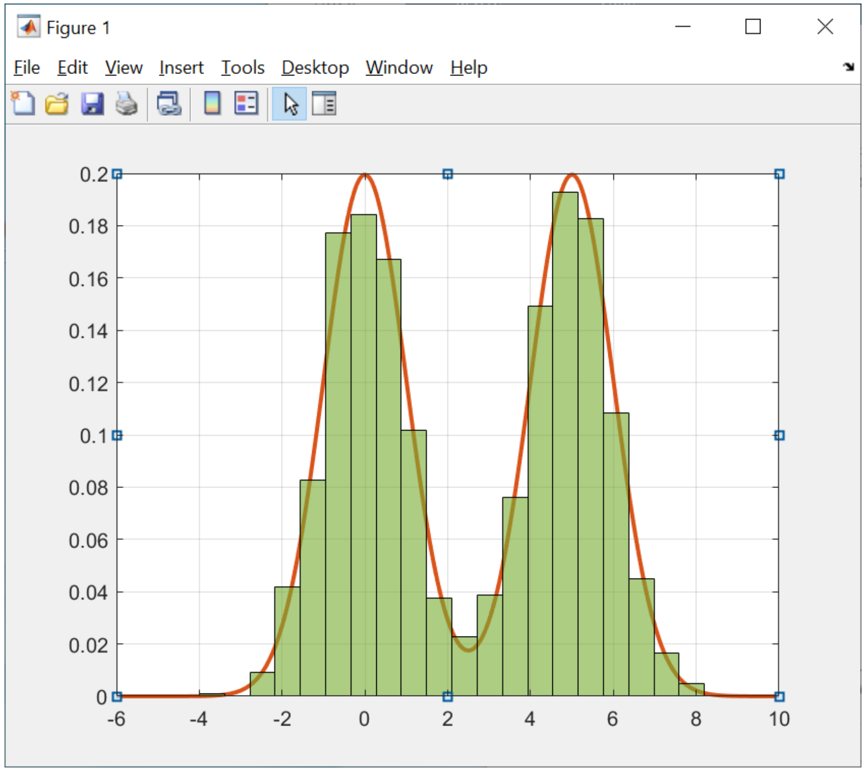
- Standard histograms simply partition the values (x) taken by a continuous random variable (X) into distinct bins of width Δ_i, and then count the number n_i of observations of x falling in bin i.
- In order to turn this count into a normalized probability density, we divide n_i by the total number (N) of observations and by the width (Δ_i) of the bins to obtain probability density values for each bin given by $f_i = \frac{n_i}{N\Delta_i}$.
- This gives a model for the density $f(x)$, which is constant over the width of each bin, and $\int f(x)dx = 1$
- Often the bins are chosen to have the same width $\Delta i = \Delta$.

‘hist_demo.m’

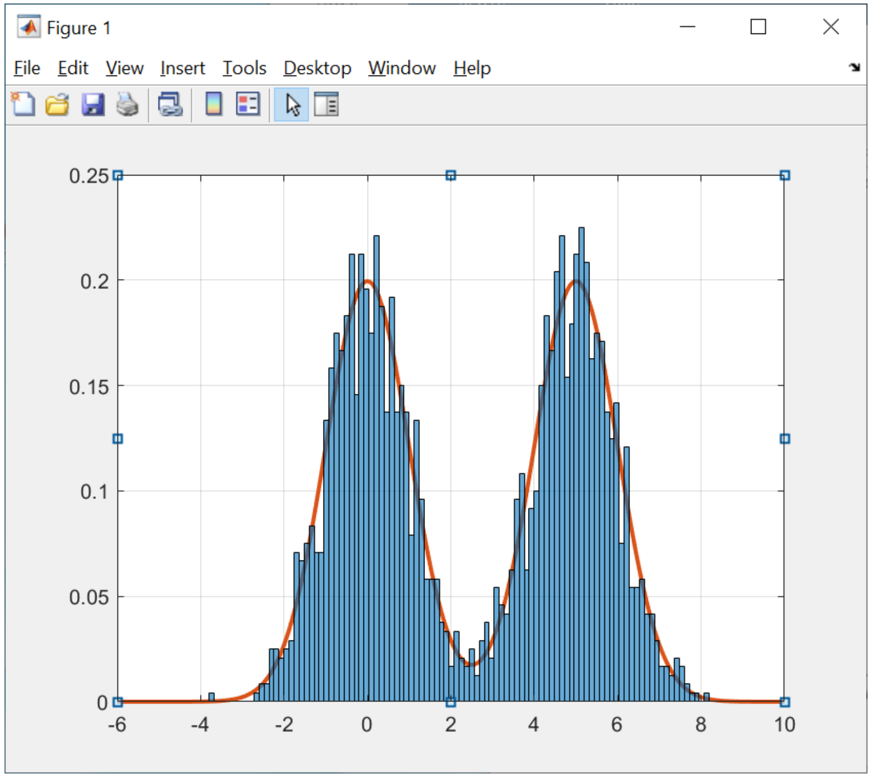
```
X = [randn(M, 1); 5+randn(M, 1)];
h = histogram(X, 10, 'Normalization',
'pdf');
h.BinLimits
sum(h.Values)
h.BinWidth
sum(h.Values)*h.BinWidth

% The true pdf
x = -6: 0.01: 10;
f = 1/2*(normpdf(x,0,1)+normpdf(x,5,1));

% Check the area under the curve
trapz(x,f)

hold on;
plot(x,f)
```



Keep reducing the bin size



Δ = 20



Δ = 100

Summary

- Δ being very large might cause over-smoothing and a washing out of structure that might otherwise be extracted from the data.
- Δ being too small can lead to noisy estimates, with the resulting density model being very spiky, with a lot of structure that is not present in the underlying distribution that generated the data set.
- The best results can be possibly obtained for some intermediate values of Δ.
- In principle, a histogram density model is also dependent on the choice of edge location for the bins, although this is typically much less significant than the value of Δ.

Advantages and Limitations

- The histogram method has the property (unlike the KNN method to be introduced next) that, once the histogram has been computed, the data set itself can be discarded, which can be advantageous if the data set is large.
- The histogram approach is easily applied if the data points are arriving sequentially.
- In practice, the histogram technique can be useful for obtaining a quick visualization of data in one or two dimensions but is *unsuited* to most density estimation applications.
  - The estimated density has discontinuities that are due to the bin edges rather than any property of the underlying distribution that generated the data.
  - Another major limitation is its scaling with dimensionality.
  - If we divide each variable in a D-dimensional space into M bins, then the total number of bins will be M^D -- an example of the **Curse of Dimensionality**.
  - In a space of high dimensionality, the quantity of data needed to provide meaningful estimates of local probability density would be prohibitive.

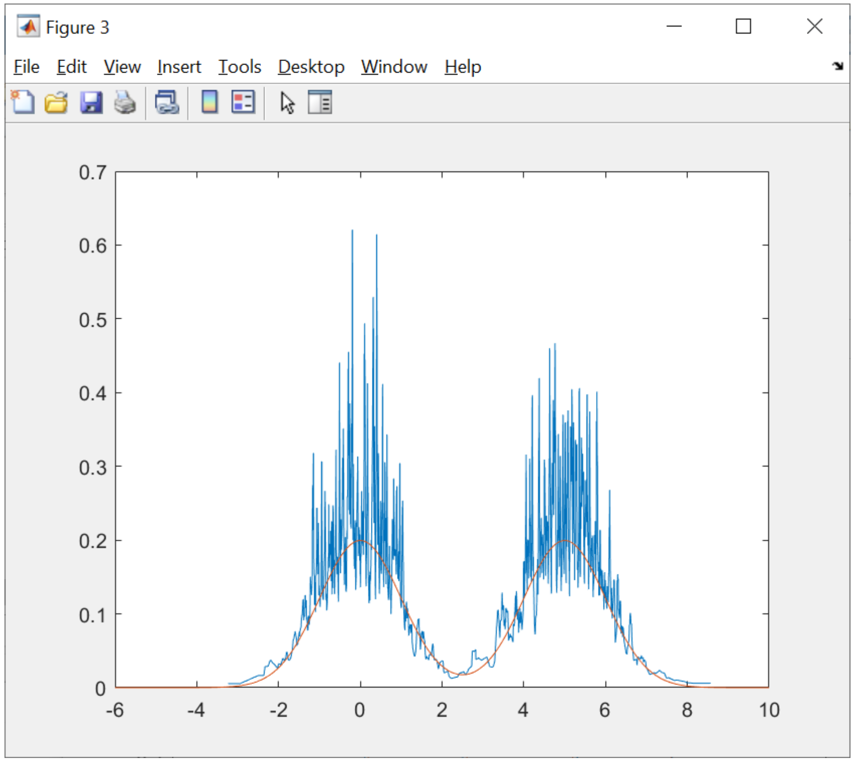
Lessons Learned

- to estimate the probability density at a particular location, we should consider the data points that lie within some local neighborhood of that point.
- This concept of *locality* requires that we assume some form of distance, e.g., the Euclidean distance.
- For histograms, this neighborhood property was defined by the bins, and there is a natural “smoothing” parameter (the bin width) describing the spatial extent of the local region.
- The value of the smoothing parameter should be neither too large nor too small in order to obtain good results.
- Nearest-neighbor methods are nonparametric techniques that have better scaling with dimensionality than the simple histogram model.

K-Nearest Neighbor Density Estimation

- Suppose we have collected a data set comprising N observations drawn from the distribution with PDF being $f(\boldsymbol{x})$.
- We consider a small sphere centered on the point $\boldsymbol{x}$ at which we want to estimate the probability density $f(\boldsymbol{x})$.
- We allow the radius of the sphere to grow until it contains precisely K data points.
- The estimate of the density $f(\boldsymbol{x})$ is then given by $f(\boldsymbol{x}) = \frac{K}{NV}$, where V is the volume of the resulting sphere.
  - For a simple 1-D dataset, V is the “bin” width, where the “bin” contains the nearest K data points.
- This technique is known as the K-nearest neighbors.

```
K = 5;
% For each unique data point, find the distance
% between this point and all other points
Y = unique(sort(X));

Density = zeros(length(Y), 1);
Dist = zeros(N, 1);

for i = 1: length(Y)
   Dist = (Y(i) - X).^2;
   [Dist_sorted, Index] = sort(Dist);

% Among the K nearest neighbors, find the range
% of these value
   V = range(X(Index(1:K)));
   Density(i) = K/(N*V);
end

figure;
plot(Y, Density)
```



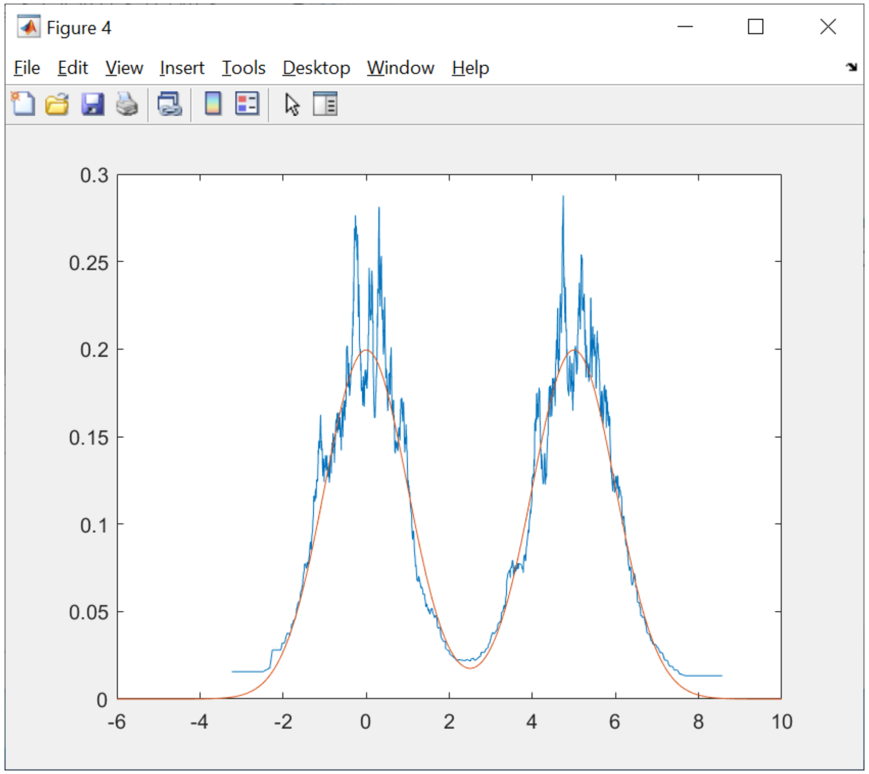


Figure 4
File Edit View Insert Tools Desktop Window Help
0.3
0.25
0.2
0.15
0.1
0.05
0
-6
-4
-2
0
2
4
6
8
10

K = 50

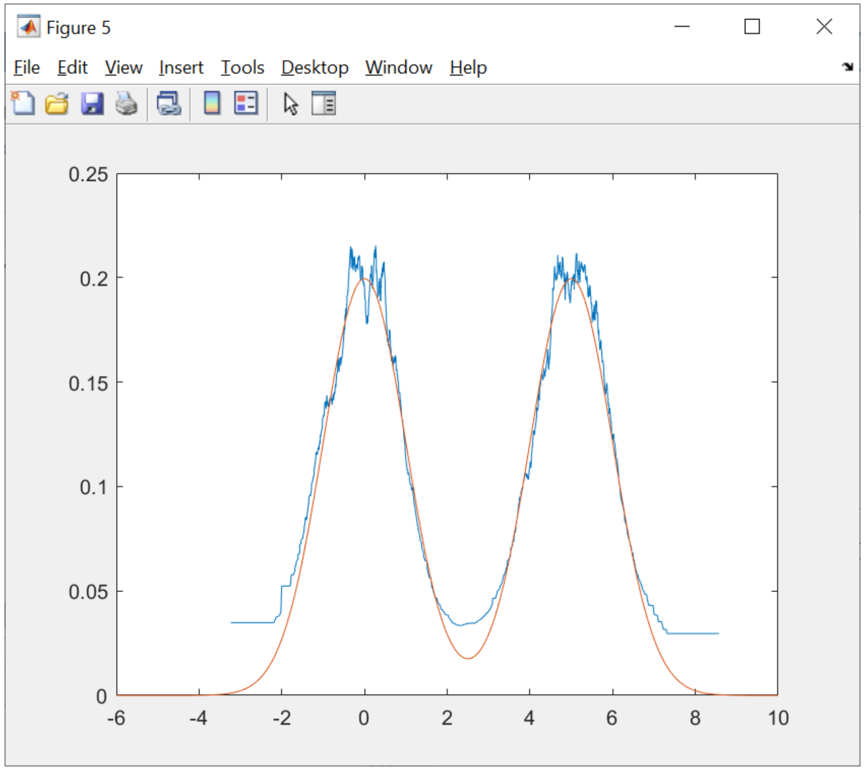


Figure 5
File Edit View Insert Tools Desktop Window Help
0.25
0.2
0.15
0.1
0.05
0
-6
-4
-2
0
2
4
6
8
10

K = 150

Extension to Classification

- The *K*-nearest-neighbor technique for density estimation can be extended to the problem of classification, where we apply the *K*-nearest-neighbor density estimation technique to each class separately and then use the Bayes' theorem.
- Suppose that we have a data set comprising N_i points in class C_i with N points in total, so that $\sum_i N_i = N$.
- The class priors (*a priori* probabilities) are given by $p(C_i) = \frac{N_i}{N}$.
- If we want to classify a new point $\boldsymbol{x}$, we draw a sphere centered on $\boldsymbol{x}$ containing precisely *K* points irrespective of their class.
- Suppose this sphere has volume V and contains K_i points from class C_i, then an estimate of the density associated with each class is $f(\boldsymbol{x}|C_i) = \frac{K_i}{N_i V}$, where $\sum_i K_i = K$.
- The unconditional density is given by $f(\boldsymbol{x}) = \frac{K}{NV}$.

Posterior Probability

Using Bayes’ theorem to obtain the posterior probability of class membership:

$$f(C_i|\boldsymbol{x}) = \frac{f(\boldsymbol{x}|C_i)p(C_i)}{f(\boldsymbol{x})} = \frac{K_i}{K}$$

where

$$f(\boldsymbol{x}|C_i) = \frac{K_i}{N_i V}, p(C_i) = \frac{N_i}{N}, \text{ and } f(\boldsymbol{x}) = \frac{K}{NV}$$

- In order to minimize the probability of misclassification, we assign the test point $\boldsymbol{x}$ to the class having the largest posterior probability, corresponding to the largest value of $\frac{K_i}{K}$.
- Thus to classify a new point, we identify the *K* nearest points from the training data set and then assign the new point to the class having the *largest* number of representatives amongst this set.
- The particular case of *K* = 1 is called the nearest-neighbor rule, because a test point is simply assigned to the same class as the nearest point from the training set.

Some Distance Metrics

Given two (1-by-n) row vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, the various distances between these two vectors are defined as follows:

- Minkowski distance (or L_p metric)

$$D_M = \left(\Sigma_{j=1}^{n}\left|x_j - y_j\right|^p\right)^{\frac{1}{p}}$$

- City block distance (or Manhattan distance)

$$D_{CB} = \Sigma_{j=1}^{n}\left|x_j - y_j\right|$$

  - A special case of Minkowski distance when $p = 1$.
- Euclidean distance

$$D_E = \left(\Sigma_{j=1}^{n}\left|x_j - y_j\right|^2\right)^{\frac{1}{2}}$$

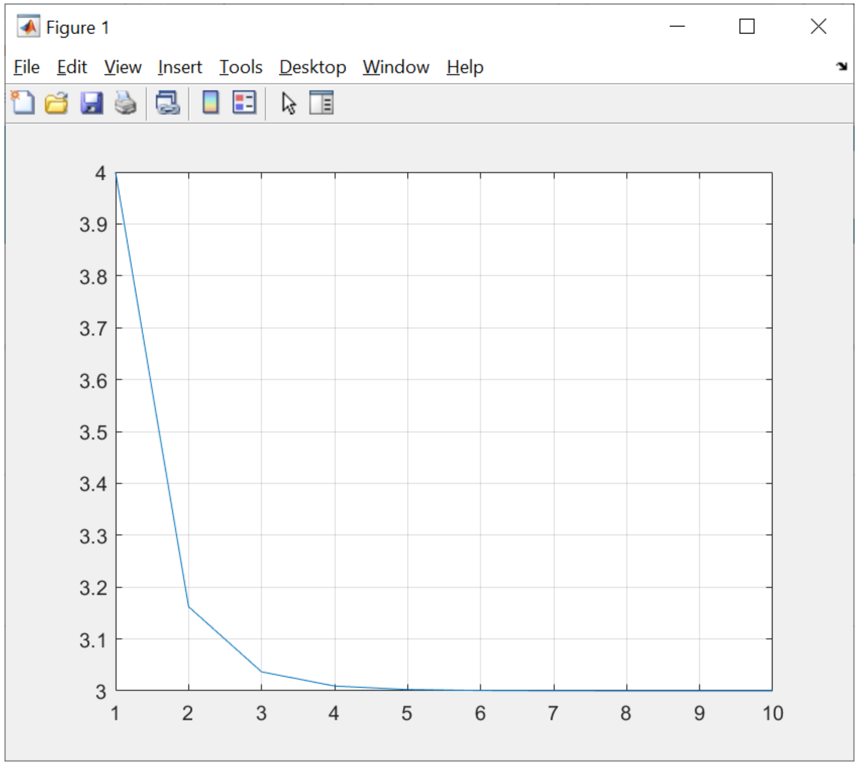
  - A special case of Minkowski distance when $p = 2$.
- Chebychev distance (maximum metric, or L_∞ metric)

$$D_C = \max_j\left|x_j - y_j\right|$$

  - A special case of Minkowski distance when $p = \infty$.

$p \to \infty$

```
>> x = [1 2]
>> y = [4 3]
>> pdist2(x,y,'minkowski',1)
>> pdist2(x,y,'minkowski',2)
>> norm(x-y)
>> pdist2(x,y,'minkowski',3)

>> for p=1:10
      d(p) = pdist2(x,y,'minkowski',p);
    end
>> plot(1:10, d); grid
```



Proof using the Squeeze Theorem

Show that

$\lim_{p\to\infty}(a^p + b^p)^{\frac{1}{p}} = \max(a, b)$, where $a \geq 0$, and $b \geq 0$.

Assume that $\max(a, b) = a$, without loss of generality.

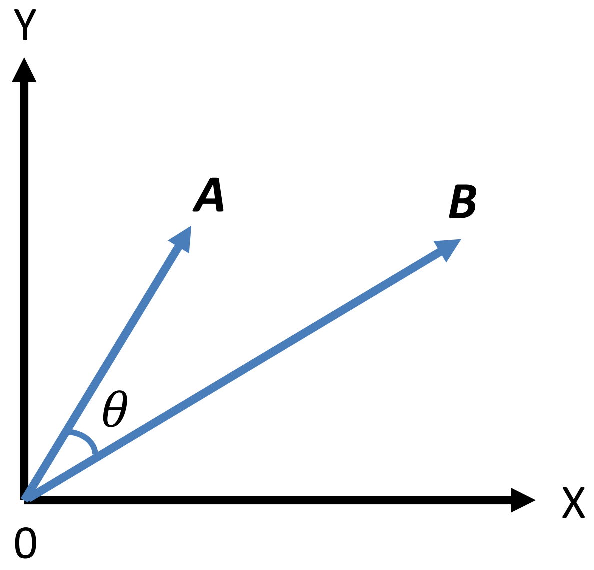
$$\lim_{p\to\infty}(a^p + b^p)^{1/p} \geq \lim_{p\to\infty}(a^p)^{1/p} = \max\{a, b\}$$

$$\begin{aligned}\lim_{p\to\infty}(a^p + b^p)^{1/p} \leq \lim_{p\to\infty}(a^p + a^p)^{1/p} &= a\lim_{p\to\infty}2^{1/p}\\ &= \max\{a, b\}\end{aligned}$$

Cosine Similarity

- The cosine of two non-zero vectors can be derived by using the Euclidean dot product formula:

$$\boldsymbol{A} \cdot \boldsymbol{B} = \|\boldsymbol{A}\|\|\boldsymbol{B}\| \cos(\theta)$$



- *Cosine Similarity* measures the similarity between two vectors of an inner product space as:

$$S_C(\boldsymbol{A}, \boldsymbol{B}) \triangleq \cos(\theta) = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{\|\boldsymbol{A}\|\|\boldsymbol{B}\|}$$

- The resulting similarity ranges from −1 (exactly opposite), to 1 (exactly the same), with 0 indicating orthogonality or decorrelation.
- In-between values indicate intermediate similarity/dissimilarity.

Cosine Distance

- Cosine Distance is used for the complement of cosine similarity in positive space.

$$D_C(\boldsymbol{A},\boldsymbol{B}) \triangleq 1 - \frac{\boldsymbol{A}\cdot\boldsymbol{B}}{\|\boldsymbol{A}\|\|\boldsymbol{B}\|} = 1 - \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

- Cosine Distance is a non-metric measure.
  - Coincidence axiom is violated:

$$D_C(\boldsymbol{A},\boldsymbol{B}) = 0 \text{ not} \leftrightarrow \boldsymbol{A} = \boldsymbol{B}$$

- Can be used to measure distance when the magnitude of the vectors does not matter.
- It is often used to measure document similarity in text analysis.

Matlab, numpy, scipy, sklearn

```
>> A = [1 0]
>> B = [2 2]
>> pdist2(A,B,'cosine')
>> 1 - dot(A,B)/(norm(A)*norm(B))
>> acos(dot(A,B)/(norm(A)*norm(B)))*180/pi
```

```
>> C = 2*B
C =
    4    4
>> pdist2(A,C,'cosine')
ans =
   0.2929
```

numpy:

```
from numpy import dot
from numpy.linalg import norm
A = [1, 0]
B = [2, 2]
1 - dot(A, B)/(norm(A) * norm(B))
```

sklearn:

```
from sklearn.metrics.pairwise import cosine_distances
cosine_distances([A], [B])
```

scipy:

```
from scipy import spatial
spatial.distance.cosine(A, B)
```

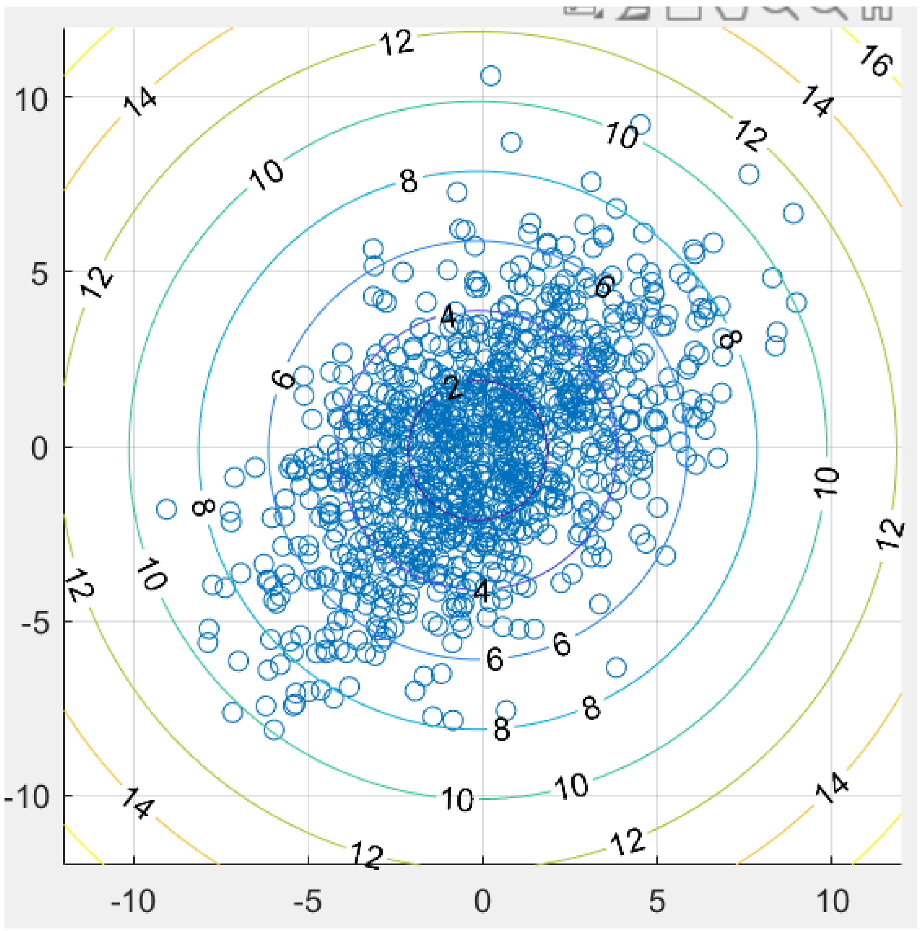
Mahalanobis Distance

- Mahalanobis distance between $\boldsymbol{x}$ and $\boldsymbol{y}$.

  $D_M = \sqrt{(\boldsymbol{x}-\boldsymbol{y})\boldsymbol{C}^{-1}(\boldsymbol{x}-\boldsymbol{y})^T},$ where $\boldsymbol{C}$ is the sample covariance matrix.
- $\boldsymbol{C}$ is a positive definite matrix
  - It is symmetric.
  - All of its eigenvalues are positive.
- The Mahalanobis distance can be used to measure the distance between a point *X* and the sample mean of a distribution *Q*, thus finds application in outlier analysis.

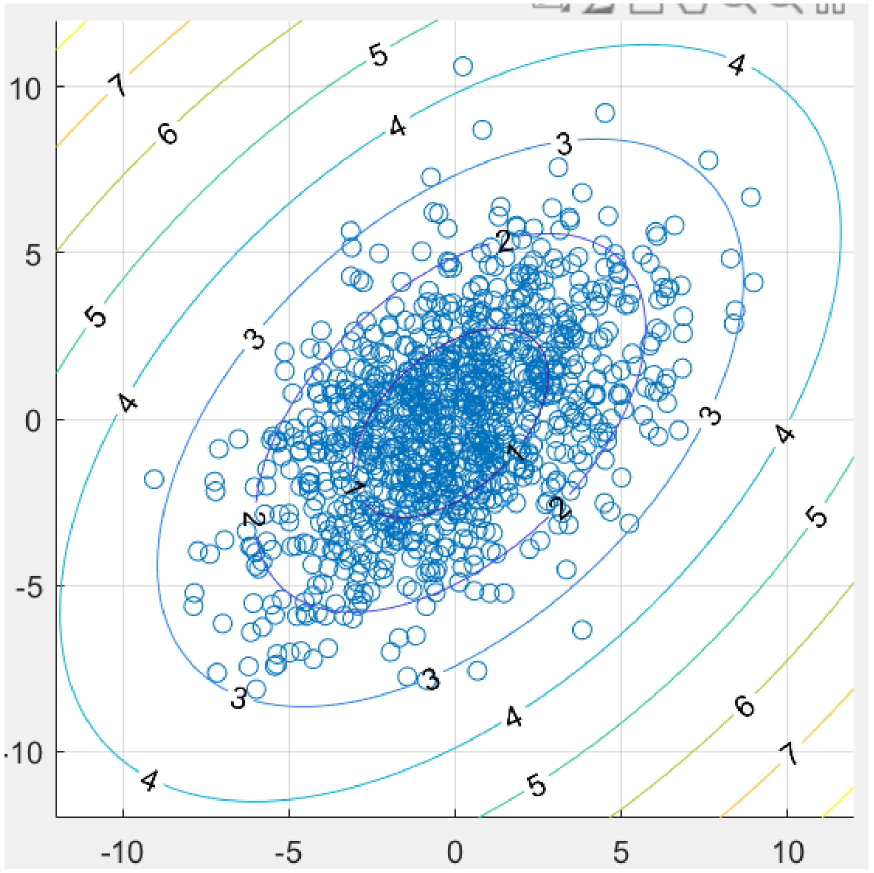
‘mahal_dist_demo.m’

```
m = [0, 0]
C =[8 4; 4 8]
eig(C)
Q = mvnrnd(m,C,1000);
figure; scatter(Q(:,1),Q(:,2))
grid; axis equal; axis square
smean = mean(Q)
scov = cov(Q)
hold on;
x = linspace(-12,12);
y = linspace(-12,12);
[X,Y] = meshgrid(x,y);
```



```
% Euclidean Distances
D_Euc = sqrt((X - smean(1)).^2 + (Y - smean(2)).^2);
contour(X,Y,D_Euc,'ShowText', 'on')
```

```
% Mahalanobis Distance from the sample mean vector
D_Mah = zeros(100, 100);
for i = 1: 100
    for j = 1: 100
        D_Mah(i,j) = pdist2([X(i,j),Y(i,j)], smean, 'mah', scov);
    end
End

contour(X,Y,D_Mah,'ShowText', 'on')
```



Matlab, numpy, scipy

```
>> sm = [-0.1235    0.0187]
>> x = [0 -10]
>> pdist2(x, sm, 'mah', scov)
ans =
   4.0930
```

```
>> sqrt((sm-x)*inv(scov)*(sm-x)')

>> scov
scov =
   8.6404    4.2092
   4.2092    8.1148
```

numpy:

```
x = np.array([0, -10])
sm = np.array([-0.1235, 0.0187])
scov = np.array([[8.6404, 4.2092],[4.2092, 8.1148]]);
d = x - sm
from numpy.linalg import multi_dot
np.sqrt(np.linalg.multi_dot([d, np.linalg.inv(scov), np.matrix.transpose(d)]))
```

scipy:

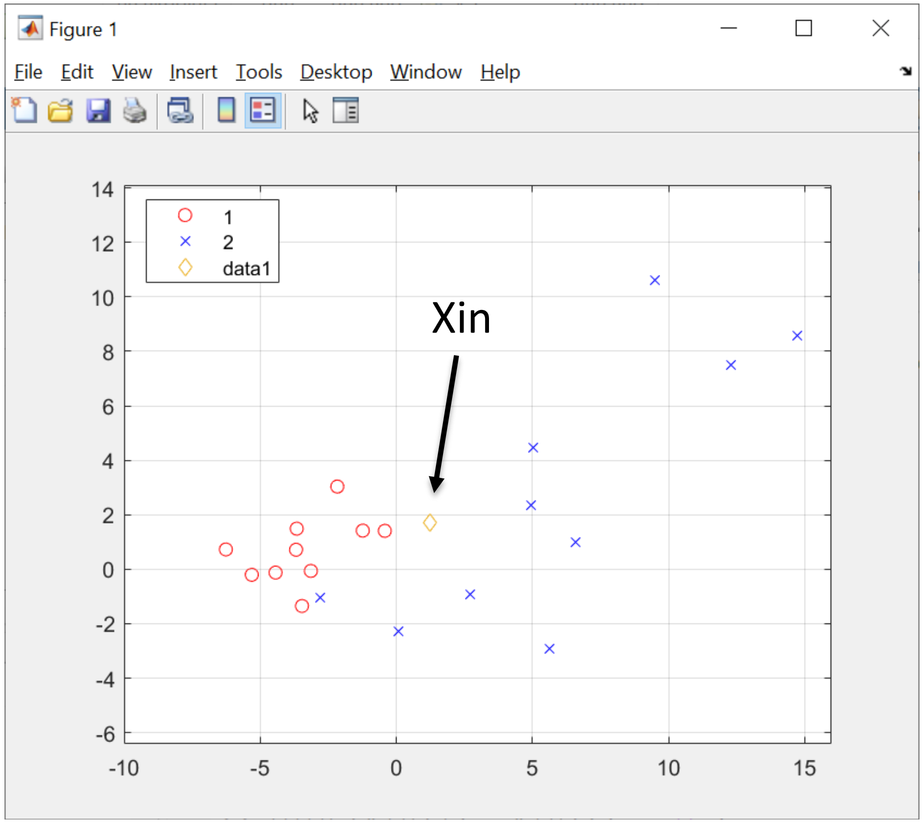
```
from scipy import spatial
spatial.distance.mahalanobis([-0.1235, 0.0187],[0, 10], np.linalg.inv(scov))
4.093030999968769
```

Dataset Generation

```
function [data, label] = dataset_generate(N)

close all
% Generate data entries for Class 1
m1 = [-4, 0];   % Mean vector
c1 = [1,0; 0,1]; % Covariance matrix
rng default  % For reproducibility
r1 = mvnrnd(m1,c1,N);

data_C1 = zeros(N, 2);
data_C1 = r1;
label_C1 = ones(N, 1);

% Generate data entries for Class 2
m2 = [4, 0];
c2 = [9,4; 4,9];
rng default  % For reproducibility
r2 = mvnrnd(m2,c2,N);

data_C2 = zeros(N, 2);
data_C2 = r2;
label_C2 = 2*ones(N, 1);

% Combine data of two classes
data = vertcat (data_C1, data_C2);
label = vertcat (label_C1, label_C2);

% Export the dataset to a csv file
% (to be imported by sklearn or other
apps)
data_label = horzcat(data, label);
writematrix(data_label, 'mvnrnd.csv');
```

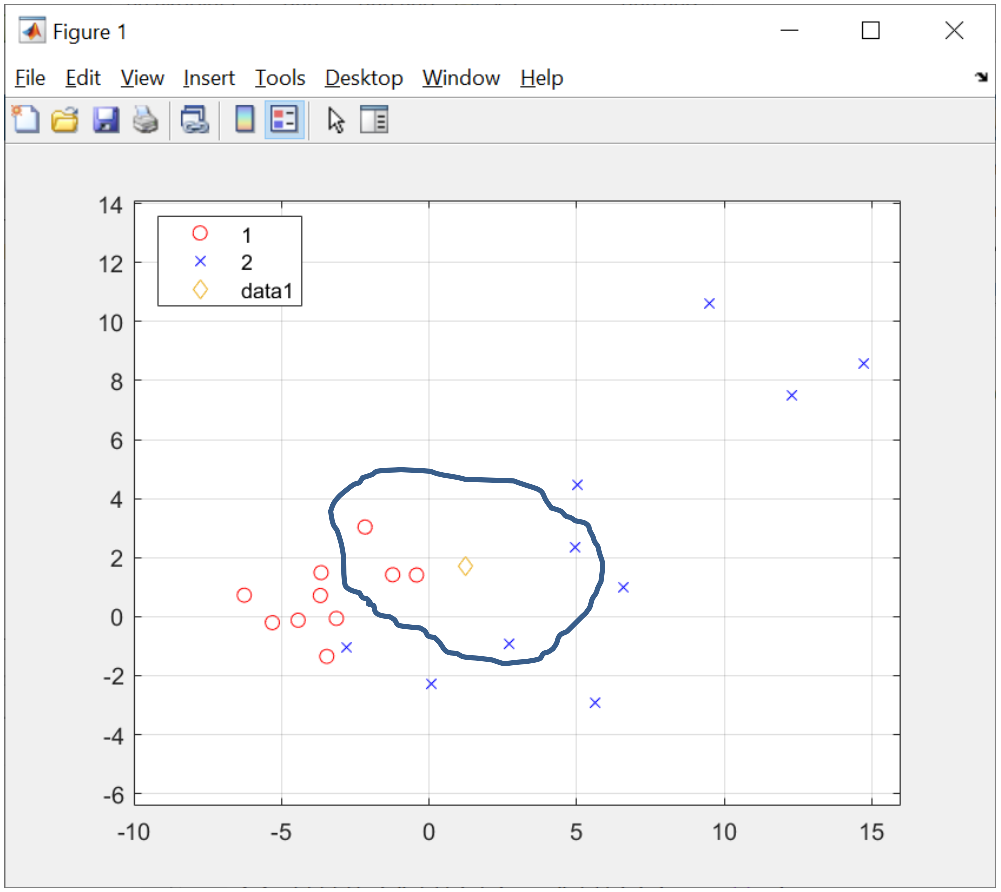
‘knn_demo.m’

```
N = 10;
% Number of samples per class

% X is the data array, C is the class label
vector
[X, C] = dataset_generate(N);

% Plot the data samples of the two classes
figure;
gscatter(X(:,1),X(:,2), C, 'rb','ox');
axis square
axis equal

% An example input query vector
Xin = mean(X)

Xin =
   1.2486    1.7158
```



```
K = 5;          % Look at the K nearest neighbors

L = length(X);

% Calculate the Euclidean distances
Dist = zeros(1, L);
for i = 1:L
   Dist(i) = norm(X(i,:) - Xin);
end

% Sort the distances
[D_sorted, Index] = sort(Dist);

% First 5 values of the index vector: Index[1:5]
% 9    10    17     2    15

% Look at the first K classes
Classes = C(Index(1:K));

% Classes = 1, 1, 2, 1, 2
% Posterior Prob. for Class 1: 3/5 = 60%

% The predicted label for Xin (majority vote)

label = mode(Classes)

% When there are multiple values occurring equally
frequently, % mode returns the smallest of those values
(tie breaker)
```

% Posterior Prob. for Class 1: 3/5 = 60% — "3/5 = 60%" printed in bold.



fitcknn ( )

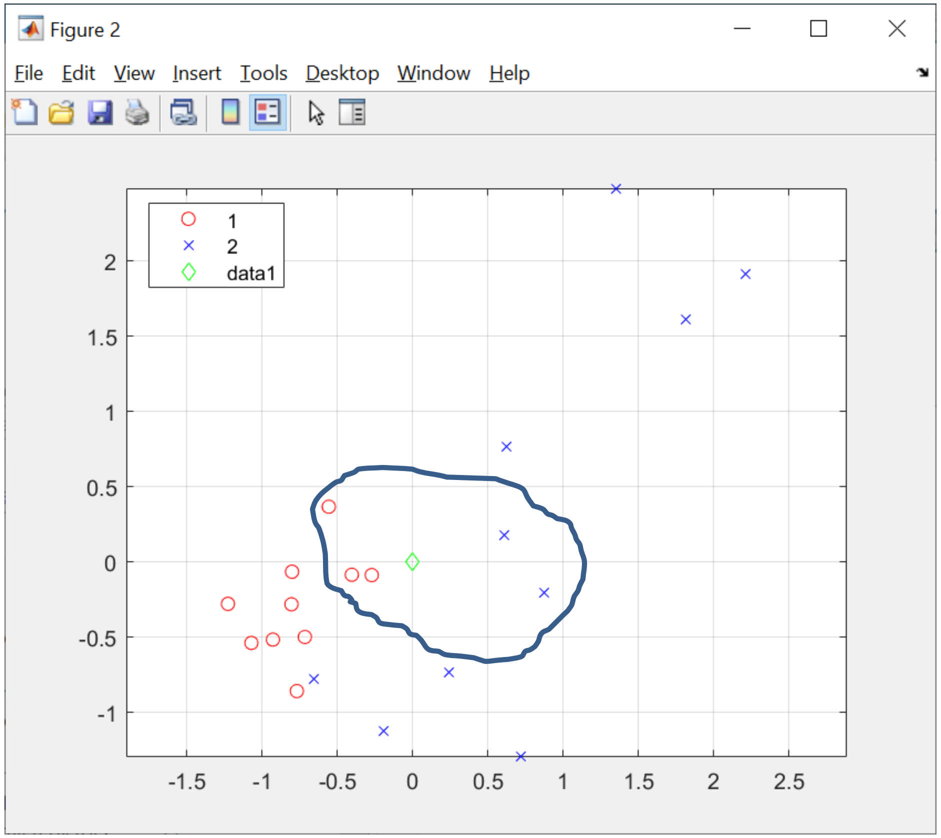
```
Model = fitcknn(X, C, 'NumNeighbors',K);
Model.Prior
%ans =
%    0.5000    0.5000
>> Model.Distance
ans =
    'euclidean‘
>> Model.BreakTies
ans =
    'smallest‘

[label,score,cost] = predict(Model, Xin)
label =
     1
score =
    0.6000    0.4000
cost =
    0.4000    0.6000
```

Data Preprocessing

- Standardization standards for mean removal and variance scaling
- Standardization of datasets is a common requirement for many machine learning algorithms, which might behave badly if the individual features do not more or less look like standard normally distributed data (Gaussian with zero mean and unit variance).
- For example, if a feature has a variance that is orders of magnitude larger than others, this dominating feature might make it difficult for the algorithm to learn from other features correctly as expected.
- In practice we often ignore the shape of the distribution and just transform the data to center it by removing the mean value of each feature, then scale it by dividing non-constant features by their standard deviation.

KNN Classification with Data Preprocessing / Standardization

```
% Plot the standardized samples after normalization
X_std_1 = zeros(L, 2);
X_std = zeros(L, 2);

Mu = zeros(1,2);
Std = zeros(1,2);

for i = 1:2
   Mu(i) = mean(X(:,i));
   Std(i) = std(X(:,i));
   X_std_1(:,i) = (X(:,i) - Mu(i))/Std(i);
end

>> % Alternatively, use the function
X_std = normalize(X);
>> isequal(X_std, X_std_1)
ans =
  logical
   1

figure; gscatter(X_std(:,1),X_std(:,2),C,'rb','ox');
axis square; axis equal; grid
```



Repeat the KNN Algorithm with the Standardized Data

```
% Normalize the Xin
for i = 1:2
   Xin_std(:,i) = (Xin(:,i) - Mu(i))/Std(i);
end
hold on;
plot(Xin_std(1), Xin_std(2), 'gd');


% Calculate the Euclidean distances
Dist_std = zeros(1, L);
for i = 1:L
   Dist_std(i) = norm(X_std(i,:) -
Xin_std);
end
```

```
% Sort the distances
[D_sorted_std, Index_std] = sort(Dist_std);
% First 5 values of the index vector are
different from above
%Index_std(1:5)
%ans =
%                 9   10   15    2   17
%    Previously,  9   10   17    2   15


% Look at the first K classes
Classes_std = C(Index_std(1:K));

% The predicted label for the input vector Xin
(majority vote)
label_std = mode(Classes_std)
```

fitcknn( ) with ‘standardize’ option

```
Model_std = fitcknn(X, C,
'NumNeighbors',K, 'Standardize', 1);
Mu =
   1.2486    1.7158
>> Model_std.Mu
ans =
   1.2486    1.7158
>> Std
Std =
   6.1113    3.5793
>> Model_std.Sigma
ans =
   6.1113    3.5793
```

```
>> [label,score,cost] = predict(Model_std,Xin)
label =
     1
score =
    0.6000    0.4000
cost =
    0.4000    0.6000
```

'preprocessing_demo.py'

```
import numpy as np
infile = r"C:\...\mvnrnd.csv"
dataset = np.loadtxt (infile, delimiter=',')
X = dataset[:, 0:2]
y = dataset[:,2]

from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(X)
```

```
# Mean
scaler.mean_
X.mean(axis = 0)

# Standard deviation
scaler.scale_
X.std(axis = 0)

X_scaled = scaler.transform(X)
X_scaled.mean(axis = 0)
X_scaled.var(axis = 0)
```

Pipeline in sklearn

from sklearn.neighbors import **KNeighborsClassifier**

neigh = KNeighborsClassifier(n_neighbors=5)

neigh.fit(X_scaled, y)

# Apply the same scaling on the test data Xin

Xin_scaled = scaler.transform([Xin])

neigh.predict(Xin_scaled)

neigh.predict_proba(Xin_scaled)

# Alternatively, use pipeline

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

pipe = make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors=5))

pipe.fit(X, y)
pipe.predict([Xin])
pipe.predict_proba([Xin])

Computational Complexity

- The *K*-nearest-neighbor method requires the entire training data set to be stored, leading to expensive computation if the data set is large.
- This effect can be offset by constructing tree-based search structures to allow (approximate) near neighbors to be found efficiently without doing an exhaustive search of the data set.
- fitcknn( ) function has the option ‘NSMethod’, or Nearest neighbor search method, specified as either 'kdtree' or 'exhaustive'.
  - 'kdtree' — Creates and uses a Kd-tree to find nearest neighbors.
  - 'exhaustive' — Uses the exhaustive search algorithm. When predicting the class of a new point xnew, the software computes the distance values from all points in X to xnew to find nearest neighbors.
- In computer science, a Kd-tree (short for K-dimensional tree) is a space-partitioning data structure for organizing points in a K-dimensional space.

Summary

- KNN categorizes objects based on the classes of their nearest neighbors in the dataset.
- KNN predictions assume that objects near each other are similar.
- Best
  - When you need a simple algorithm to establish benchmark learning rules
  - When memory usage of the trained model is a lesser concern
  - When prediction speed of the trained model is a lesser concern