データマイニング（１）

matsuda@symbolics.jp (2010.1.27)

【データマイニング(Data Mining)】

構造・パターンを持たないデータからある種の**構造あるいはパターン（法則）を見つける**こと。データという鉱脈から構造という金を見つける（発掘する mininig)。

１．ルールの発見

学習データにより一定のルール（決定木の場合が多い）を構成。ついで実データにより必要ならば決定木の枝別れ（ルールそのもの）を変更する。クラス判別ルールの発見と利用、あるいはある種の予測（確率を含む）。
長所：ノイズの影響少ない。決定的。高速、短所：融通性がない。分割データの粒度が小さくなりすぎる。

２．クラスタ化

何らかの指標をベースにデータを複数のグループに分類する。この場合、ＡとＢが遠い、近いということを区別するために両者の距離を計算する必要が出てくる。多様な距離関数が存在。クラスタ化されればＯＫの場合とクラスタ間の関係をさらに必要とする場合がある。
長所：学習データが不要（近傍型の場合、大まかに複数の中心が必要）、短所：遅い。ノイズによる精度への影響大。

３．確定的な手法

通常、これらの手法が使えるのであればデータマイニングは不要。それでもクラスタリングの前段階としてラフに分類する場合に有効。
多くの統計解析手法。Fitting関数(線形、非線形）。特に時系列データの傾向を見るのにFItting関数を使用することができる。

クラスタ分析

データ分類手法の一つ。教師無し学習。データ要素に対し距離関数に基づいた分類（クラスタリング）を行う。
ペア要素が同一の場合、距離はゼロ。それ以外は正。データ要素として可能なもの：数値のリスト・行列・テンソル、True/False要素のリスト、文字列のリスト。
（参考）tutorial/PartitioningDataIntoClusters

In[1]:=
```
Clear["Global`*"]
```

In[2]:=
```
data = {1.2, 9.1, 2.3, 15.4, 71.8};
```

数の近似に基づく分類。ただしデフォルトの距離関数はユークリッド距離。

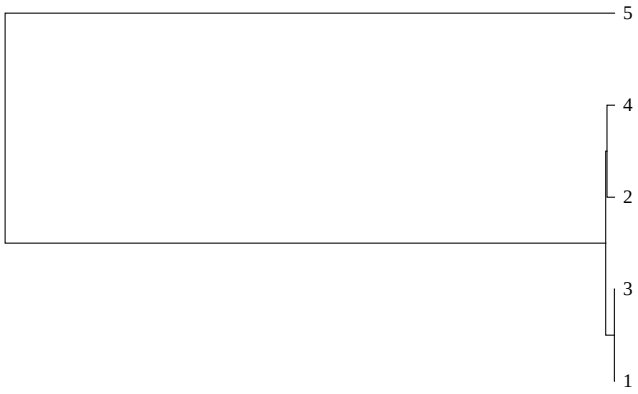
In[3]:=
```
FindClusters[data]
```

Out[3]=
```
{{1.2, 2.3}, {9.1, 15.4}, {71.8}}
```

In[4]:=
```
Needs["HierarchicalClustering`"]
```

In[5]:=
```
DendrogramPlot[data, LeafLabels → Automatic, Orientation → Left]
```

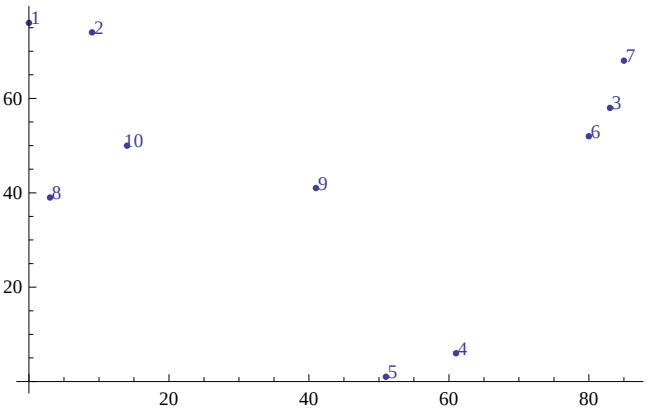
Out[5]=



■ ラベル付された10個の点に対しクラスタ分類を行う

In[6]:=
```
data1 = Table[{Random[Integer, 100], Random[Integer, 100]}, {10}];
p = Apply[Text, #] & /@ Thread[{Range[Length[data1]], data1 + 1}];
SetOptions[ListPlot, PlotStyle → p];
ListPlot[data1]
```

Out[9]=



In[10]:=
```
FindClusters[data1 -> Range[Length[data1]]](*ラベルとして1から個数までの整数を使用 *)
```

Out[10]=
```
{{1, 2, 8, 9, 10}, {3, 6, 7}, {4, 5}}
```

In[11]:=
```
SetOptions[ListPlot, PlotStyle → Automatic]; Clear[p]
```

■ 数字、文字列が混在したデータに対するクラスタ分類

In[12]:=
```
datarecords =
  {{"Joe", "Smith", 158, 64.4}, {"Mary", "Davis", 137, 64.4}, {"Bob", "Lewis", 141, 62.8},
   {"John", "Thompson", 235, 71.1}, {"Lewis", "Black", 225, 71.4},
   {"Sally", "Jones", 168, 62.}, {"Tom", "Smith", 243, 70.9}, {"Jane", "Doe", 225, 71.4}};
TableForm[datarecords]
```

Out[13]//TableForm=

Joe	Smith	158	64.4
Mary	Davis	137	64.4
Bob	Lewis	141	62.8
John	Thompson	235	71.1
Lewis	Black	225	71.4
Sally	Jones	168	62.
Tom	Smith	243	70.9
Jane	Doe	225	71.4

In[14]:=
```
FindClusters[datarecords](* 失敗する *)
```

FindClusters::amtd :

FindClusters is unable to automatically select an appropriate dissimilarity function for the input data {{Joe, Smith, 158, 64.4}, {Mary, Davis, 137, 64.4}, {Bob, Lewis, 141, 62.8}, {John, Thompson, 235, 71.1}, {Lewis, Black, 225, 71.4}, {Sally, Jones, 168, 62.}, {Tom, Smith, 243, 70.9}, {Jane, Doe, 225, 71.4}}. ≫

Out[14]=
```
FindClusters[{{Joe, Smith, 158, 64.4}, {Mary, Davis, 137, 64.4},
  {Bob, Lewis, 141, 62.8}, {John, Thompson, 235, 71.1}, {Lewis, Black, 225, 71.4},
  {Sally, Jones, 168, 62.}, {Tom, Smith, 243, 70.9}, {Jane, Doe, 225, 71.4}}]
```

数値部分のみを使いクラスタ分類し、出力としては元データをマッピングして使用。

In[15]:=
```
FindClusters[Drop[datarecords, None, {1, 2}] → datarecords]
```

Out[15]=
```
{{{Joe, Smith, 158, 64.4}, {Mary, Davis, 137, 64.4},
  {Bob, Lewis, 141, 62.8}, {Sally, Jones, 168, 62.}}, {{John, Thompson, 235, 71.1},
  {Lewis, Black, 225, 71.4}, {Tom, Smith, 243, 70.9}, {Jane, Doe, 225, 71.4}}}
```

In[16]:=
```
Map[TableForm[#] &, %]
```

Out[16]=
```
{ Joe   Smith 158 64.4   John  Thompson 235 71.1
  Mary  Davis 137 64.4   Lewis Black    225 71.4
  Bob   Lewis 141 62.8 , Tom   Smith    243 70.9
  Sally Jones 168 62.    Jane  Doe      225 71.4 }
```

In[17]:=
```
Drop[datarecords, None, {1, 2}] // TableForm
```

Out[17]//TableForm=
```
158 64.4
137 64.4
141 62.8
235 71.1
225 71.4
168 62.
243 70.9
225 71.4
```

■ 一般的なデータを想定したクラスタ分類

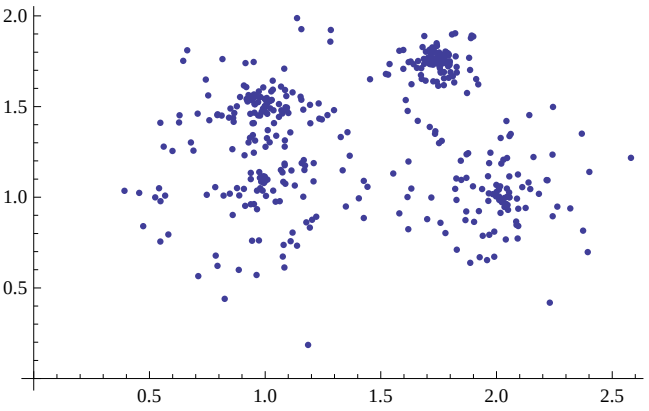
いくつかの点({2,1},{1,1.5},{1,1.1},{1.75,1.75})を決め、その周囲に正規分布密度関数(平均0、分散sigma)で求められる点を配置する。二次元方向へ分散を広げるために$e^{i\,\text{RandomReal}[\{0,2\pi\}]}$を使用。

In[18]:=
```
GaussianRandomData[n_Integer, p_, sigma_] := Table[
   p + {Re[#], Im[#]} &[RandomReal[NormalDistribution[0, sigma]] e^(i RandomReal[{0, 2 π}])], {n}];
datapairs = BlockRandom[
   SeedRandom[1234];
   Join[GaussianRandomData[100, {2, 1}, .3], GaussianRandomData[100, {1, 1.5}, .2],
    GaussianRandomData[100, {1, 1.1}, .4], GaussianRandomData[100, {1.75, 1.75}, 0.1]]];
```

In[20]:=
```
r = Thread[{{0, 0}, {datapairs[[All, 1]] // Max, datapairs[[All, 2]] // Max}}];
```

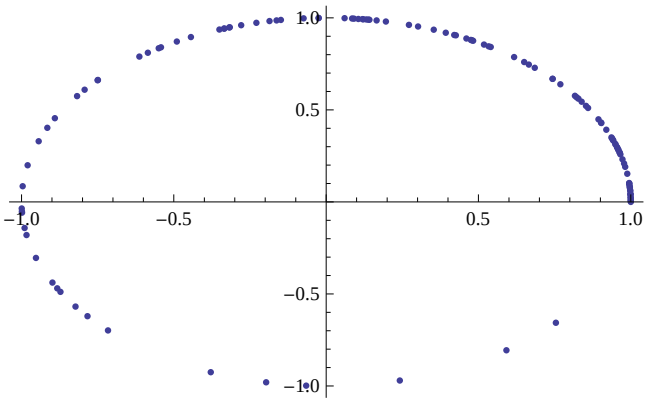
In[21]:=
```
ListPlot[datapairs]
```

Out[21]=



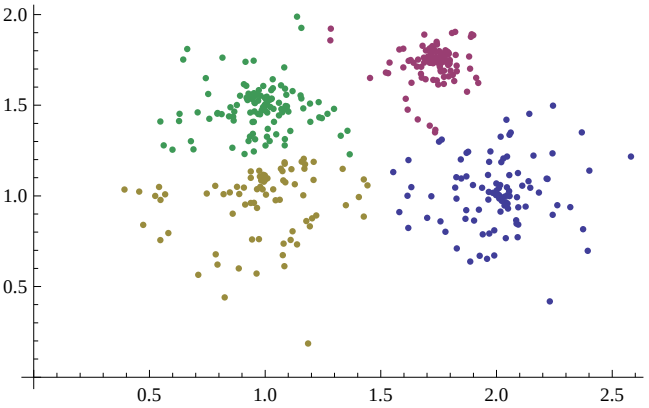
In[22]:=
```
Table[{Re[#], Im[#]} &[e^(i RandomReal[i])], {i, 0, 2 π, π / 60}] // ListPlot
```

Out[22]=



In[23]:=
```
ListPlot[FindClusters[datapairs]]
```

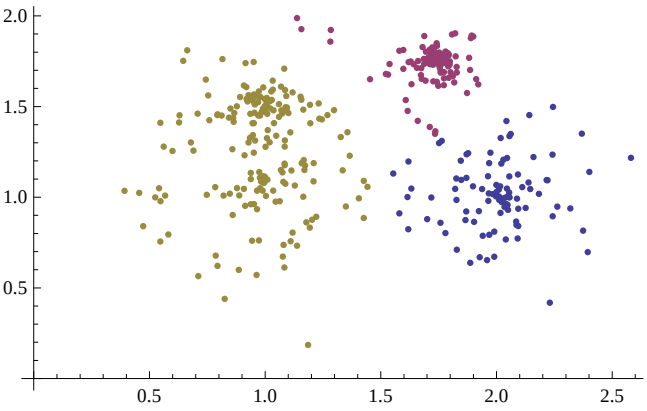
Out[23]=



デフォルトクラスタ数=4

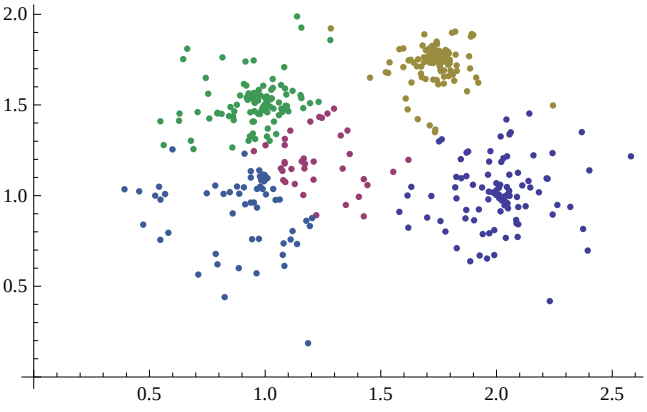
In[24]:=
```
ListPlot[FindClusters[datapairs, 3]]
```

Out[24]=



In[25]:=
```
ListPlot[FindClusters[datapairs, 5]]
```

Out[25]=



■ 距離関数の指定

In[26]:=
```
Options[FindClusters]
```

Out[26]=
```
{DistanceFunction → Automatic, Method → Automatic}
```

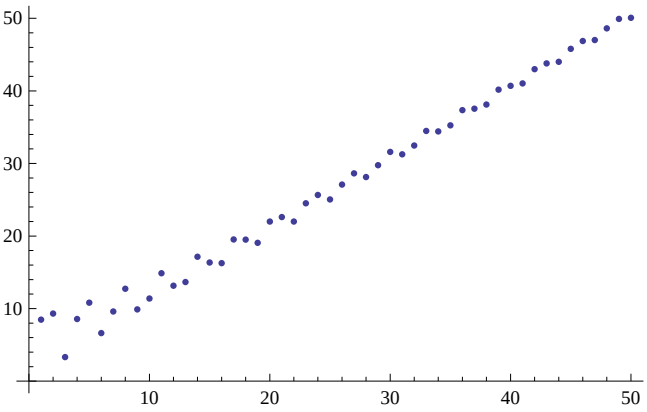
In[27]:=
```
points = Table[{i, Sin[i] * 10.}, {i, 50}];
```

In[28]:=
```
EuclideanDistance[{a, b, c}, {x, y, z}]
```

Out[28]=
$$\sqrt{\text{Abs}[a-x]^2+\text{Abs}[b-y]^2+\text{Abs}[c-z]^2}$$

In[29]:=
```
EuclideanDistance[{0, 0}, #] & /@ points;
ListPlot[%]
```

Out[30]=

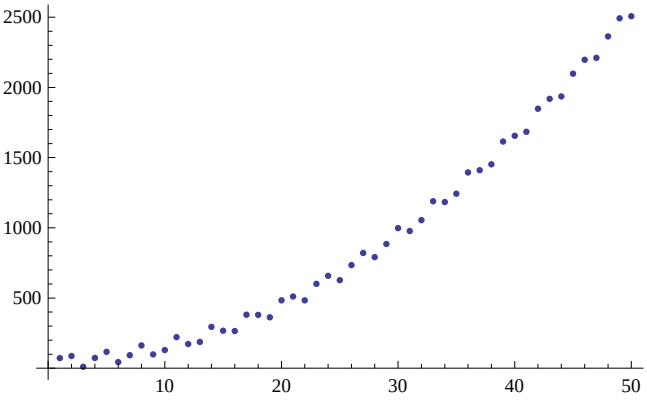


In[31]:=
```
SquaredEuclideanDistance[{a, b, c}, {x, y, z}]
```

Out[31]=
$$\text{Abs}[a - x]^2 + \text{Abs}[b - y]^2 + \text{Abs}[c - z]^2$$

In[32]:=
```
SquaredEuclideanDistance[{0, 0}, #] & /@ points;
ListPlot[%]
```

Out[33]=

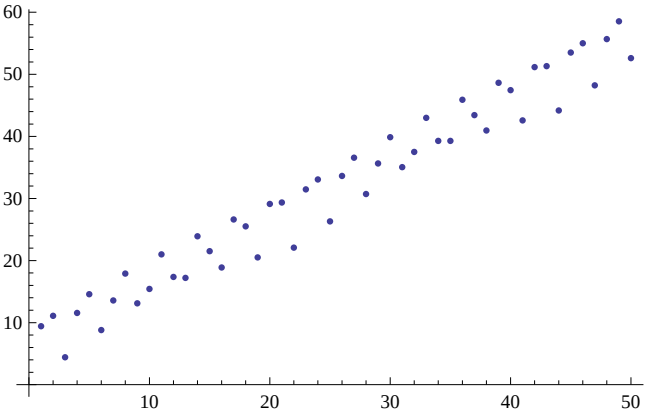


In[34]:=
```
ManhattanDistance[{a, b, c}, {x, y, z}]
```

Out[34]=
$$\text{Abs}[a - x] + \text{Abs}[b - y] + \text{Abs}[c - z]$$

In[35]:=
```
ManhattanDistance[{0, 0}, #] & /@ points;
ListPlot[%]
```

Out[36]=

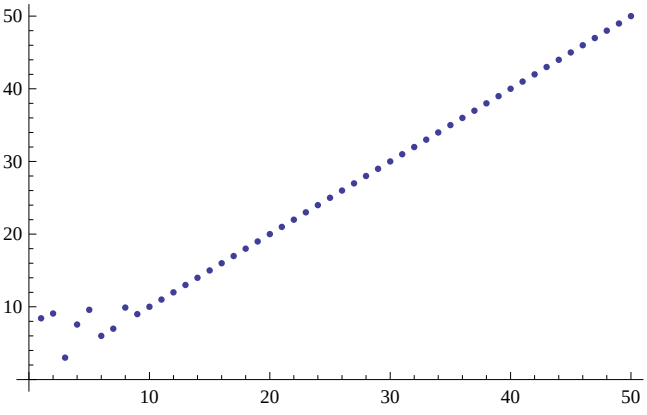


In[37]:=
```
ChessboardDistance[{a, b, c}, {x, y, z}] (* チェビシェフ距離 *)
```

Out[37]=
$$\text{Max}[\text{Abs}[a - x], \text{Abs}[b - y], \text{Abs}[c - z]]$$

In[38]:=
```
ChessboardDistance[{0, 0}, #] & /@ points;
ListPlot[%]
```

Out[39]=

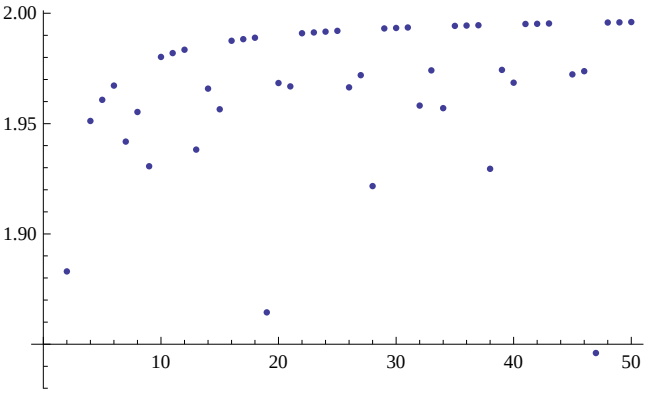


In[40]:=
```
CanberraDistance[{a, b, c}, {x, y, z}]
```

Out[40]=
$$\frac{\text{Abs}[a - x]}{\text{Abs}[a] + \text{Abs}[x]} + \frac{\text{Abs}[b - y]}{\text{Abs}[b] + \text{Abs}[y]} + \frac{\text{Abs}[c - z]}{\text{Abs}[c] + \text{Abs}[z]}$$

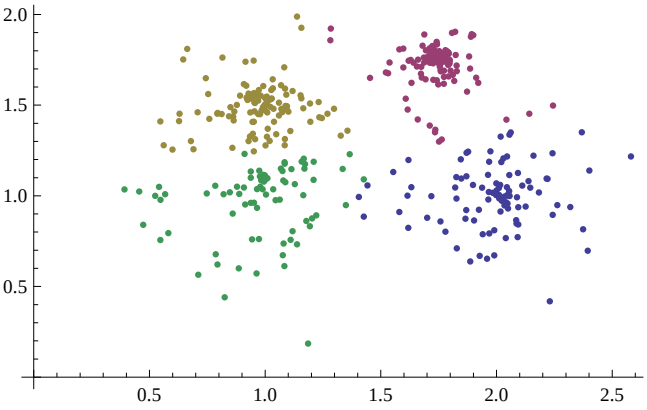
In[41]:=
```
CanberraDistance[{0.1, 0.1}, #] & /@ points;
ListPlot[%]
```

Out[42]=

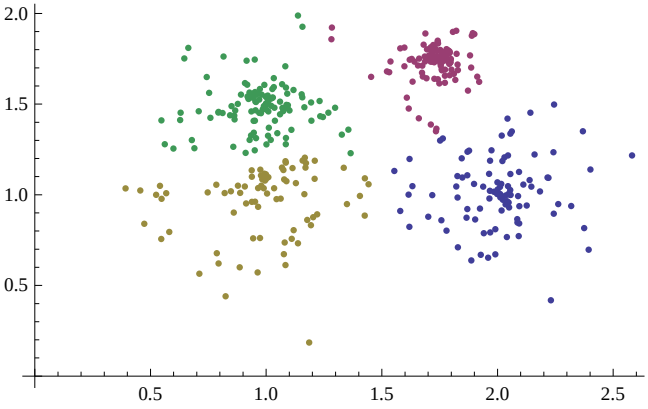


In[43]:=
```
ListPlot[FindClusters[datapairs, DistanceFunction → CanberraDistance]]
```

Out[43]=



EuclideanDistance

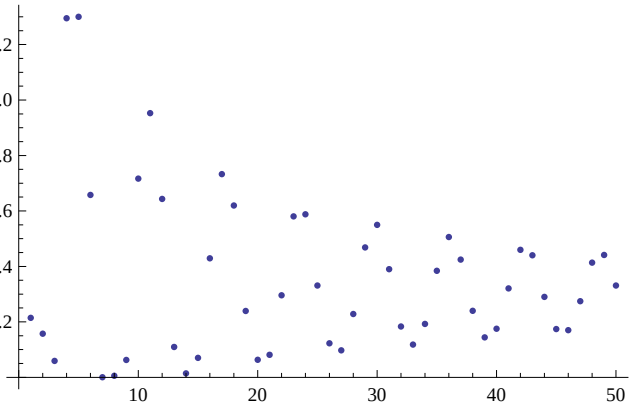


In[44]:=
```
CosineDistance[{a, b, c}, {x, y, z}]
```

Out[44]=
$$1 - \frac{a\,x + b\,y + c\,z}{\sqrt{\text{Abs}[a]^2 + \text{Abs}[b]^2 + \text{Abs}[c]^2}\ \sqrt{\text{Abs}[x]^2 + \text{Abs}[y]^2 + \text{Abs}[z]^2}}$$

In[45]:=
```
CosineDistance[{0.1, 0.1}, #] & /@ points;
ListPlot[%]
```

Out[46]=



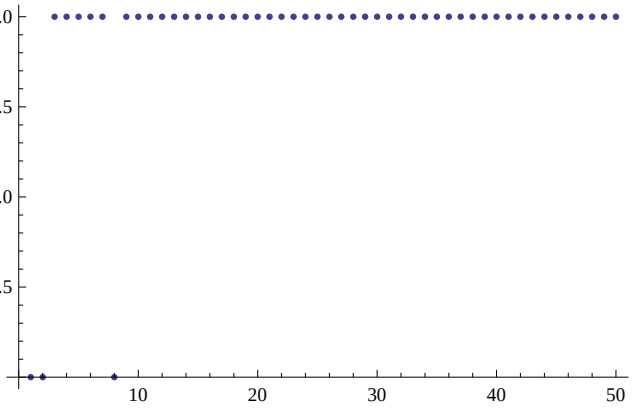
In[47]:=
```
CorrelationDistance[{a, b, c}, {x, y, z}]
```

Out[47]=

$$1 - \left(\left(a + \frac{1}{3}(-a-b-c)\right)\left(x + \frac{1}{3}(-x-y-z)\right) + \left(b + \frac{1}{3}(-a-b-c)\right)\left(y + \frac{1}{3}(-x-y-z)\right) + \left(\frac{1}{3}(-a-b-c) + c\right)\left(\frac{1}{3}(-x-y-z) + z\right)\right) \Big/ \left(\sqrt{\text{Abs}\left[a + \frac{1}{3}(-a-b-c)\right]^2 + \text{Abs}\left[b + \frac{1}{3}(-a-b-c)\right]^2 + \text{Abs}\left[\frac{1}{3}(-a-b-c) + c\right]^2}\ \sqrt{\text{Abs}\left[x + \frac{1}{3}(-x-y-z)\right]^2 + \text{Abs}\left[y + \frac{1}{3}(-x-y-z)\right]^2 + \text{Abs}\left[\frac{1}{3}(-x-y-z) + z\right]^2}\right)$$

In[48]:=
```
CorrelationDistance[{0.1, 0.5}, #] & /@ points;
ListPlot[%]
```

Out[49]=



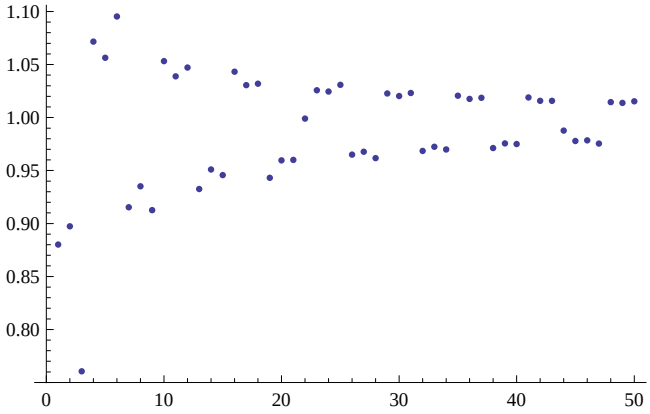
In[50]:=
```
BrayCurtisDistance[{a, b, c}, {x, y, z}]
```

Out[50]=

$$\frac{\text{Abs}[a-x] + \text{Abs}[b-y] + \text{Abs}[c-z]}{\text{Abs}[a+x] + \text{Abs}[b+y] + \text{Abs}[c+z]}$$

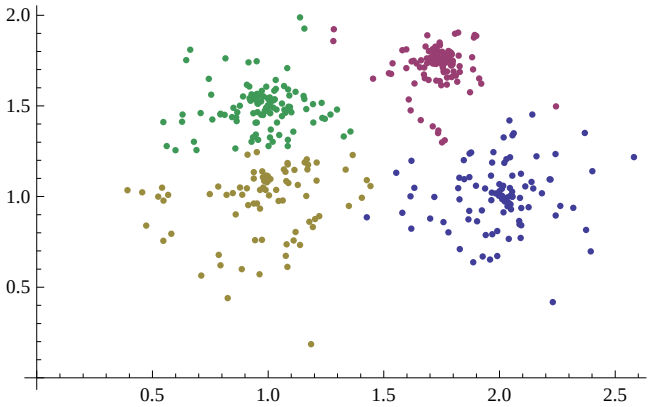
In[51]:=
```
BrayCurtisDistance[{0.1, 0.5}, #] & /@ points;
ListPlot[%]
```

Out[52]=



In[53]:=
```
ListPlot[FindClusters[datapairs, DistanceFunction → BrayCurtisDistance]]
```

Out[53]=



■ 論理値データの場合

In[54]:=
```
bdata = {{False, False, False, False, False, True, False, False, True, True},
    {True, False, False, False, False, False, False, False, False, True},
    {True, False, False, True, False, False, True, False, True, True},
    {True, True, False, False, True, False, False, False, True, True},
    {True, True, False, False, True, True, True, True, True, True}};
```

In[55]:=
```
bdata = bdata /. {True → 1, False → 0}
```

Out[55]=
```
{{0, 0, 0, 0, 0, 1, 0, 0, 1, 1}, {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},
 {1, 0, 0, 1, 0, 0, 1, 0, 1, 1}, {1, 1, 0, 0, 1, 0, 0, 0, 1, 1}, {1, 1, 0, 0, 1, 1, 1, 1, 1, 1}}
```

In[56]:=
```
TableForm /@ FindClusters[bdata]
```

Out[56]=
```
{ 0 0 0 0 0 1 0 0 1 1          1 1 0 0 1 0 0 0 1 1 }
{ 1 0 0 0 0 0 0 0 0 1 ,        1 1 0 0 1 1 1 1 1 1 }
  1 0 0 1 0 0 1 0 1 1
```

`MatchingDissimilarity[u, v]` simple matching $(n_{10}+n_{01}) \,/\, \text{Length}[u]$

`JaccardDissimilarity[u, v]` the Jaccard dissimilarity $(n_{10}+n_{01}) / (n_{11}+n_{10}+n_{01})$

`RussellRaoDissimilarity[u, v]` the Russell–Rao dissimilarity $(n_{10}+n_{01}+n_{00}) \,/\, \text{Length}[u]$

`SokalSneathDissimilarity[u, v]` the Sokal–Sneath dissimilarity $2\,(n_{10}+n_{01}) / (n_{11}+2\,(n_{10}$

`RogersTanimotoDissimilarity[u, v]` the Rogers–Tanimoto dissimilarity $2\,(n_{10}+n_{01})/(n_{11}+2\,(n_{10}+n_{01})+n_{00})$

`DiceDissimilarity[u, v]` the dice dissimilarity $(n_{10}+n_{01}) / (2\,n_{11}+n_{10}+n_{01})$

`YuleDissimilarity[u, v]` the Yule dissimilarity $2\,n_{10}\,n_{01} / (n_{11}\,n_{00}+n_{10}\,n_{01})$

In[57]:=
```
TableForm /@ FindClusters[bdata, DistanceFunction → MatchingDissimilarity]
```

Out[57]=
$$\left\{\begin{matrix}0&0&0&0&0&1&0&0&1&1\\1&0&0&0&0&0&0&0&0&1\\1&0&0&1&0&0&1&0&1&1\end{matrix},\ \begin{matrix}1&1&0&0&1&0&0&0&1&1\\1&1&0&0&1&1&1&1&1&1\end{matrix}\right\}$$

In[58]:=
```
TableForm /@ FindClusters[bdata, DistanceFunction → JaccardDissimilarity]
```

Out[58]=
$$\left\{\begin{matrix}0&0&0&0&0&1&0&0&1&1\end{matrix},\ \begin{matrix}1&0&0&0&0&0&0&0&0&1\\1&0&0&1&0&0&1&0&1&1\\1&1&0&0&1&0&0&0&1&1\\1&1&0&0&1&1&1&1&1&1\end{matrix}\right\}$$

In[59]:=
```
TableForm /@ FindClusters[bdata, DistanceFunction → RussellRaoDissimilarity]
```

Out[59]=
$$\left\{\begin{matrix}0&0&0&0&0&1&0&0&1&1\\1&0&0&1&0&0&1&0&1&1\\1&1&0&0&1&0&0&0&1&1\\1&1&0&0&1&1&1&1&1&1\end{matrix},\ \begin{matrix}1&0&0&0&0&0&0&0&0&1\end{matrix}\right\}$$

In[60]:=
```
TableForm /@ FindClusters[bdata, DistanceFunction → SokalSneathDissimilarity]
```

Out[60]=
$$\left\{\begin{matrix}0&0&0&0&0&1&0&0&1&1\end{matrix},\ \begin{matrix}1&0&0&0&0&0&0&0&0&1\\1&0&0&1&0&0&1&0&1&1\\1&1&0&0&1&0&0&0&1&1\\1&1&0&0&1&1&1&1&1&1\end{matrix}\right\}$$

In[61]:=
```
TableForm /@ FindClusters[bdata, DistanceFunction → RogersTanimotoDissimilarity]
```

Out[61]=
$$\left\{\begin{matrix}0&0&0&0&0&1&0&0&1&1\\1&0&0&0&0&0&0&0&0&1\\1&0&0&1&0&0&1&0&1&1\end{matrix},\ \begin{matrix}1&1&0&0&1&0&0&0&1&1\\1&1&0&0&1&1&1&1&1&1\end{matrix}\right\}$$

In[62]:=
```
TableForm /@ FindClusters[bdata, DistanceFunction → DiceDissimilarity]
```

Out[62]=
$$\left\{\begin{matrix}0&0&0&0&0&1&0&0&1&1\end{matrix},\ \begin{matrix}1&0&0&0&0&0&0&0&0&1\\1&0&0&1&0&0&1&0&1&1\\1&1&0&0&1&0&0&0&1&1\\1&1&0&0&1&1&1&1&1&1\end{matrix}\right\}$$

In[63]:=
```
TableForm /@ FindClusters[bdata, DistanceFunction → YuleDissimilarity]
```

Out[63]=
$$\left\{\begin{matrix}0&0&0&0&0&1&0&0&1&1\\1&1&0&0&1&1&1&1&1&1\end{matrix},\ \begin{matrix}1&0&0&0&0&0&0&0&0&1\\1&0&0&1&0&0&1&0&1&1\\1&1&0&0&1&0&0&0&1&1\end{matrix}\right\}$$

`EditDistance[u, v]`
文字の順序を変えないで，1つの文字列を別の文字列に変換するのに必要な削除，挿入，代入の回数
`DamerauLevenshteinDistance[u, v]` 削除，挿入，代入，転置の数
`HammingDistance[u, v]` 代入の数だけ

In[64]:=
```
EditDistance["abcd", "dcba"]
```

Out[64]= 4

In[65]:=
```
DamerauLevenshteinDistance["abcd", "dcba"]
```

Out[65]= 3

In[66]:=
```
HammingDistance["abcd", "dcba"]
```

Out[66]= 4

In[67]:=
```
sdata = {"The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"};
```

In[68]:=
```
FindClusters[sdata]
```

Out[68]=
```
{{The, fox, over, the, lazy, dog}, {quick, brown, jumps}}
```

■ メソッドの指定

メソッドは2種類："Agglomerate"と"Optimize"。デフォルトメソッド"Optimize"は k 個の代表的オブジェクトの集合を構築してから，（局所的に）最適なクラスタリングが見付かるまで繰り返しそのオブジェクト付近をクラスタ化していくこ。"Agglomerate"はクラスタの集合それぞれから始め，k 個のクラスタが残るまで最近傍のクラスタを融合していく。

In[69]:=
```
FindClusters[sdata, 3, Method → "Agglomerate"]
```

Out[69]=
```
{{The, brown, fox, over, the, lazy, dog}, {quick}, {jumps}}
```

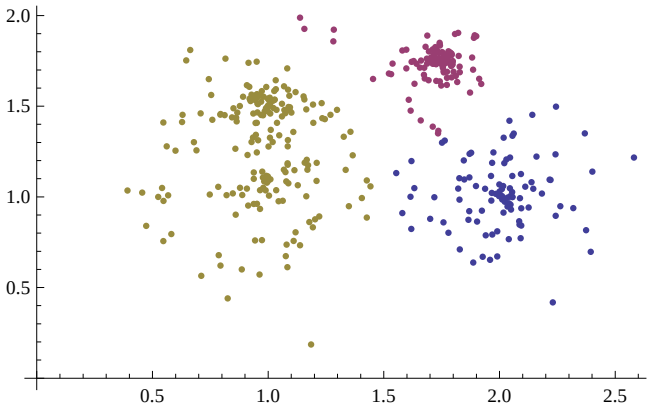
In[70]:=
```
FindClusters[sdata, 3, Method → "Optimize"]
```

Out[70]=
```
{{The, over, the}, {quick, jumps}, {brown, fox, lazy, dog}}
```

最適なクラスタ数を決めるためのMethodオプションとしてSigfinicanceTestが用意されている。

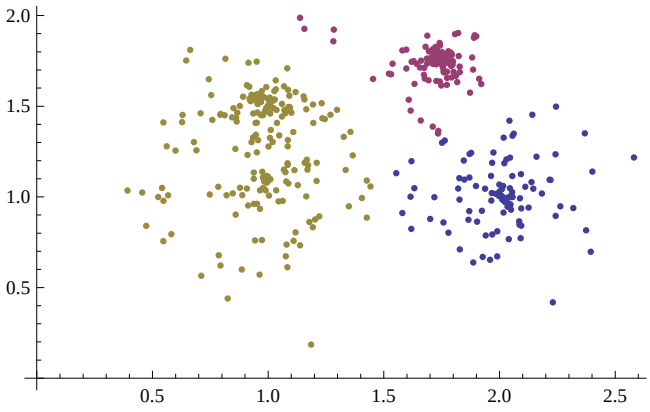
In[71]:=
```
ListPlot[
 FindClusters[datapairs, Method → {Automatic, "SignificanceTest" -> "Silhouette"}]]
```

Out[71]=



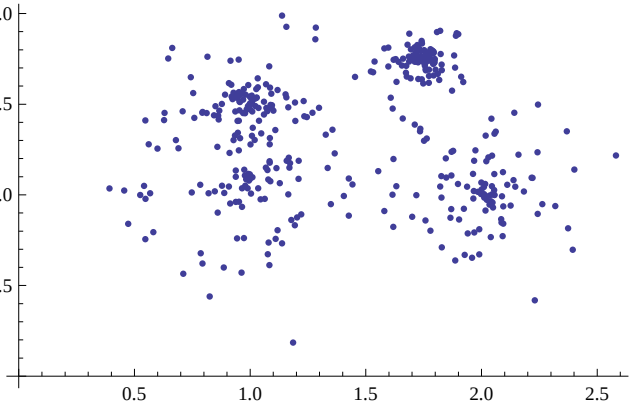
In[72]:=
```
ListPlot[FindClusters[datapairs,
  Method → {Automatic, "SignificanceTest" → {"Gap", "Tolerance" → 3}}]]
```

Out[72]=



In[73]:=
```
ListPlot[FindClusters[datapairs,
  Method → {Automatic, "SignificanceTest" → {"Gap", "Tolerance" → 6}}]]
```

Out[73]=

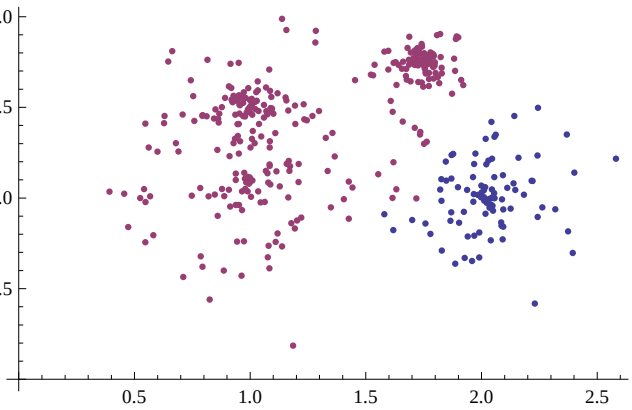


また次の関数によってクラスタ間の距離を縮めるあるいは広げる（つまりクラスタの結合、非決堂を決める）ことができる。

"Single"　　最短距離法
"Average"　　群平均法
"Complete"　　最長距離法
"WeightedAverage"　重み付き群平均法
"Centroid"　　クラスタの重心からの距離
"Median"　　クラスタのメジアンからの距離
"Ward"ウォード(Ward)の最小分散法
f　　純関数

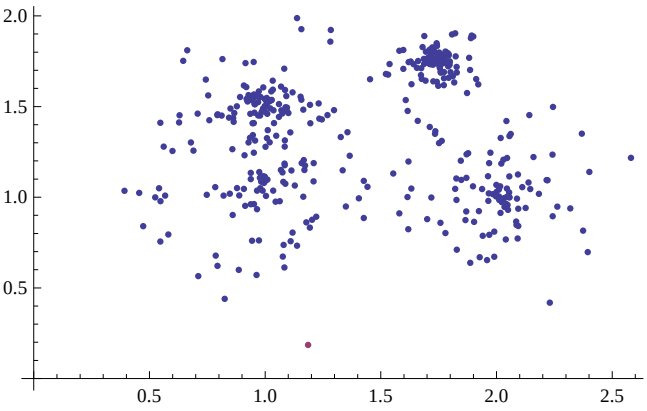
In[74]:=
```
ListPlot[FindClusters[datapairs, Method → {"Agglomerate", "Linkage" → "Complete"}]]
```

Out[74]=



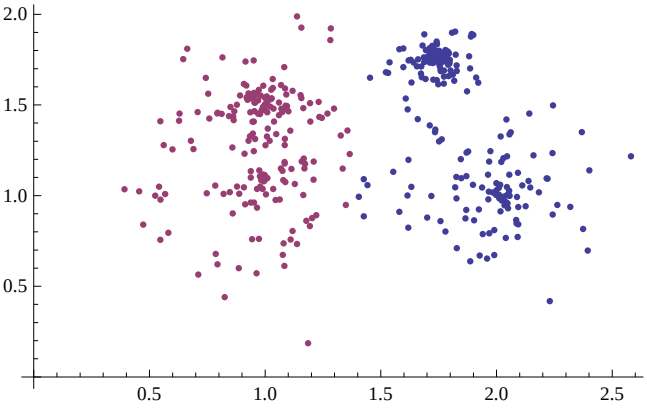
In[75]:=
```
ListPlot[FindClusters[datapairs, Method → {"Agglomerate", "Linkage" → "Centroid"}]]
```

Out[75]=



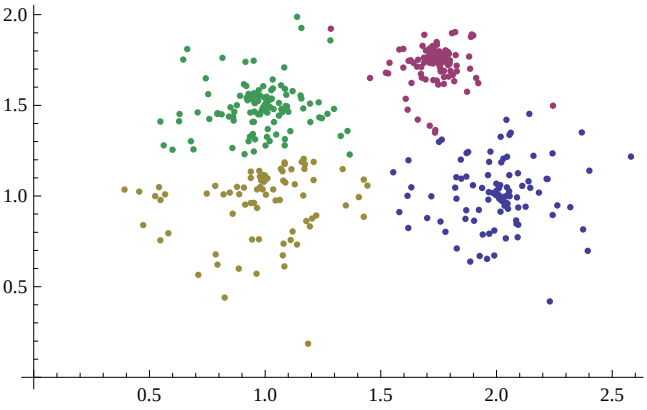
In[76]:=
```
ListPlot[FindClusters[datapairs, Method → {"Optimize", "Iterations" → 2}]]
```

Out[76]=



In[77]:=
```
ListPlot[FindClusters[datapairs, Method → {"Optimize", "Iterations" → 3}]]
```

Out[77]=



Fitting Noisy Data

参考：http://demonstrations.wolfram.com/FittingNoisyData/

ノイズデータのフィッティング。データマイニングで使用するリアルなデータに対し数学関数がフィットするケースはほとんどない。しかし、複数の非線形関数の組み合わせを使用することはありえる。

次の関数は多項式で表わされる関数fに対し一定のノイズを加えたものを初期データとして用意する。

In[78]:=
```
scatterdata[seed_, n_: 3, noise_: 100] := Module[{f}, SeedRandom[seed];
  f[x_, n] = Plus @@ Table[RandomReal[{-10, 10}, WorkingPrecision → 2] x^i, {i, 0, n}];
  Table[{i, f[i, n] + Random[Real, {-noise, noise}] * 10^n}, {i, -20, 20, 1}]]
```

■ 項の選択とフィッティング具合の視覚による確認

与えられたデータに対するフィッティング図示は関数Fitによって、一方、元データ自身のプロットはListPlotによって実行。

In[79]:=
```
seed = 1;
order = RandomInteger[{0, 10}];
vars = {1, x, x^2, x^3};
noise = 3;
scatterdata[seed, order, noise]
```

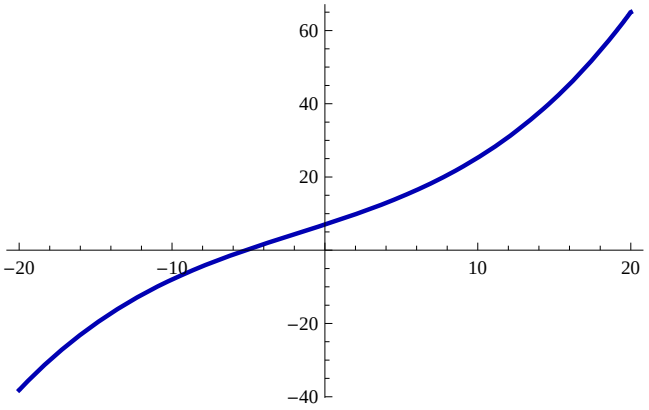
Out[83]=
```
{{-20, -33.6284}, {-19, -21.378}, {-18, -21.8416}, {-17, -58.7719}, {-16, -7.67776},
 {-15, -25.2349}, {-14, -13.2487}, {-13, 1.89534}, {-12, -46.208}, {-11, -16.7118},
 {-10, -3.44685}, {-9, 3.0363}, {-8, -7.47394}, {-7, -29.5821}, {-6, 0.688416},
 {-5, -2.67649}, {-4, 16.4538}, {-3, -21.6178}, {-2, 22.5161}, {-1, 30.4761},
 {0, 32.4809}, {1, -5.59498}, {2, 24.8873}, {3, 0.497904}, {4, 2.35936},
 {5, 14.533}, {6, 47.9788}, {7, 3.01985}, {8, 16.2871}, {9, 48.5179}, {10, 12.4775},
 {11, 14.8745}, {12, 18.7452}, {13, 23.9797}, {14, 37.1744}, {15, 35.5882},
 {16, 52.4691}, {17, 22.3119}, {18, 67.736}, {19, 72.0147}, {20, 72.2653}}
```

In[84]:=
```
Fit[scatterdata[seed, order, noise], vars, x]
```

Out[84]= $7.03849 + 1.35917\,x + 0.0159571\,x^2 + 0.00305221\,x^3$

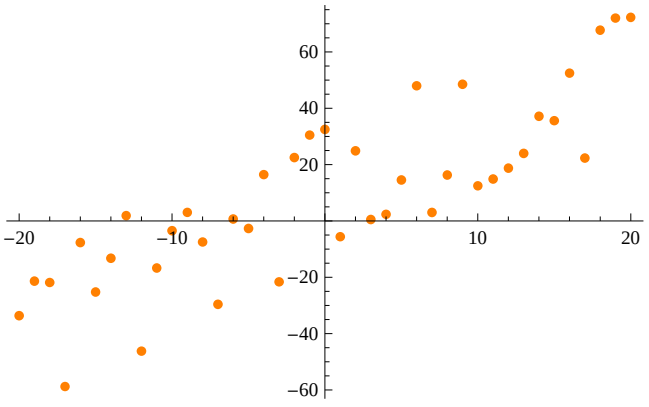
In[85]:=
```
Plot[Evaluate[Fit[scatterdata[seed, order, noise], vars, x]],
 {x, -20, 20}, PlotStyle → {Thickness[.007], Darker[Blue, .3]}]
```

Out[85]=



In[86]:=
```
ListPlot[scatterdata[seed, order, noise],
 PlotStyle → {RGBColor[1, .5, 0], PointSize[.015]}]
```

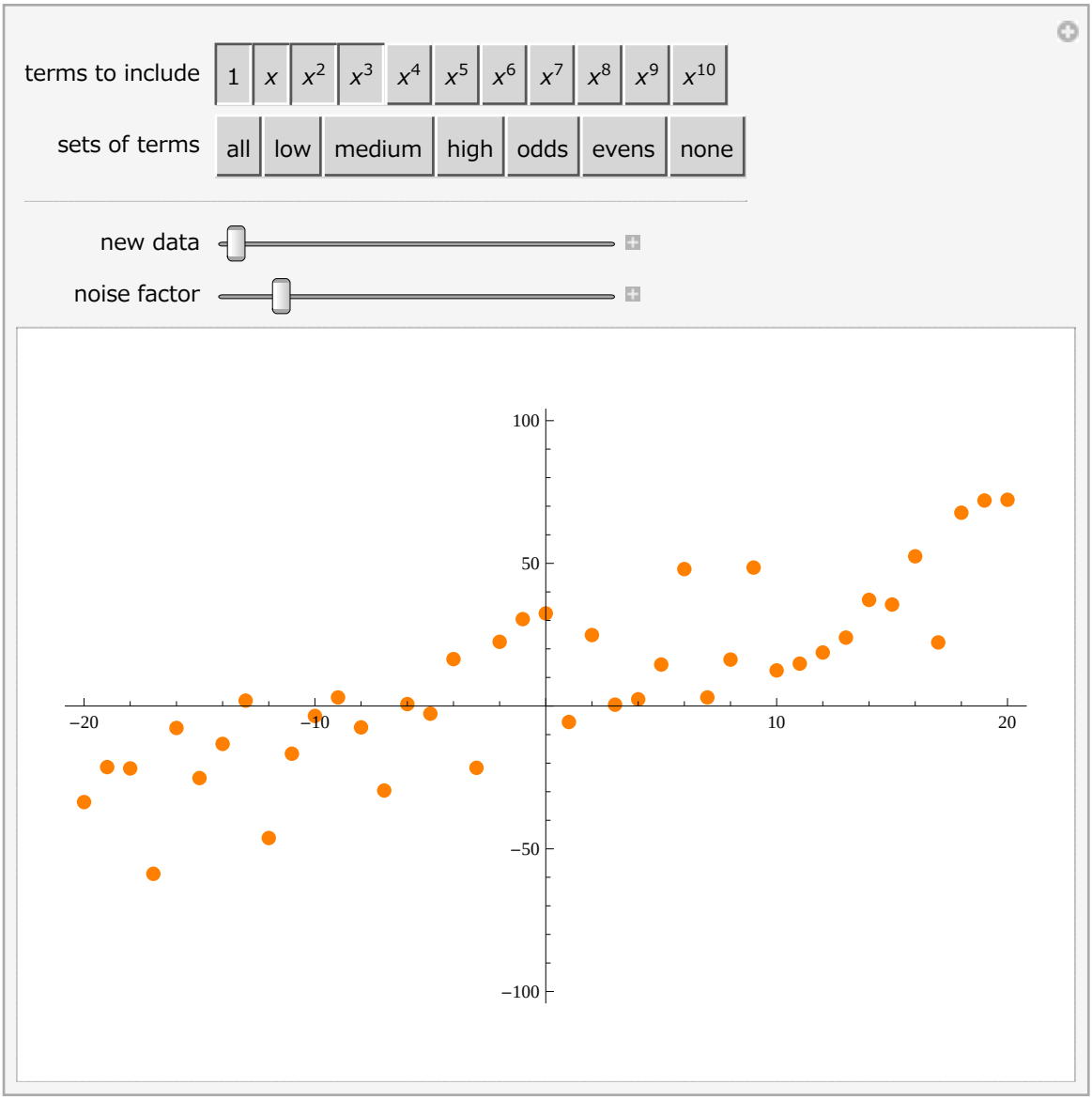
Out[86]=



In[87]:=
```
Manipulate[
 SeedRandom[seed];
 Module[{order = RandomInteger[{0, 10}]},

  Show[{Plot[Evaluate[Fit[scatterdata[seed, order, noise], vars, x]],
     {x, -20, 20}, PlotStyle → {Thickness[.007], Darker[Blue, .3]}], ListPlot[
     scatterdata[seed, order, noise], PlotStyle → {RGBColor[1, .5, 0], PointSize[.015]}]},
   PlotRange → {{-20, 20}, {-10^(order + 1), 10^(order + 1)}},
   ImageSize → {500, 350}, ImagePadding → {{10, 10}, {0, 0}}]],
 {{vars, {1, x, x^2}, "terms to include"}, {1 → ToString[1, TraditionalForm],
   x → ToString[x, TraditionalForm], x^2 → ToString[x^2, TraditionalForm],
   x^3 → ToString[x^3, TraditionalForm], x^4 → ToString[x^4, TraditionalForm],
   x^5 → ToString[x^5, TraditionalForm], x^6 → ToString[x^6, TraditionalForm],
   x^7 → ToString[x^7, TraditionalForm], x^8 → ToString[x^8, TraditionalForm],
   x^9 → ToString[x^9, TraditionalForm], x^10 → ToString[x^10, TraditionalForm]},
  ControlType → TogglerBar}, {{vars, {1, x, x^2, x^3}, "sets of terms"},
  {x^# & /@ Range[0, 10] → "all", x^# & /@ Range[0, 2] → "low", x^# & /@ Range[4, 6] → "medium",
   x^# & /@ Range[8, 10] → "high", x^# & /@ Select[Range[0, 10], OddQ] → "odds",
   x^# & /@ Select[Range[0, 10], EvenQ] → "evens", {} → "none"}, ControlType → SetterBar},
 Delimiter, {{seed, 1, "new data"}, 1, 1000, 1}, {{noise, 3, "noise factor"}, 2, 10},
 {seed, Range[1, 1000, 50], ControlType → None},
 AutorunSequencing → {5}, SaveDefinitions → True,
 TrackedSymbols → {vars, seed, noise}]
```

Out[87]=



決定木例題

■ 学習過程（ルール生成）

In[88]:=
```
ruleCreate[data1_, data2_] := Module[{sdata1, sdata2, sep},
  sdata1 = Sort[data1]; sdata2 = Sort[data2];
  If[Max[sdata1] < Min[sdata2],
   sep = Max[sdata1]
   ,
   sep = Max[sdata2]];
  Return[# > sep &]]
```

In[89]:=
```
(* 学習事例 *)
rule = ruleCreate[{23, 24, 16, 34}, {15, 9, 8, 3, 4, 11}];
```

■ 適用過程

In[90]:=
```
data = Table[Random[Integer, 40], {100}];
A = {}; B = {};
Do[
 If[rule[#], AppendTo[B, #], AppendTo[A, #]] &[data[[i]]],
 {i, 1, Length[data]}]
```

In[93]:=
```
A
```

Out[93]=
```
{1, 4, 8, 3, 10, 12, 6, 2, 10, 9, 12, 2, 7, 6, 3, 11, 7, 4, 15, 0, 1, 0, 3, 14, 4, 8, 0, 5, 11, 5}
```

In[94]:=
```
B
```

Out[94]=
```
{29, 18, 40, 40, 19, 22, 31, 36, 29, 25, 38, 37, 36, 18, 26, 24, 40, 27, 27, 39, 33, 22, 35, 30,
 17, 28, 38, 21, 17, 16, 25, 32, 19, 32, 20, 35, 38, 38, 35, 18, 36, 27, 27, 40, 19, 31, 16,
 21, 37, 34, 31, 34, 32, 17, 20, 24, 22, 29, 35, 17, 25, 29, 39, 33, 23, 26, 28, 35, 18, 31}
```

---

k-Nearest Neighbor (kNN) Classifier

参考 http://demonstrations.wolfram.com/KNearestNeighborKNNClassifier/

In[95]:=
```
Get["ComputationalGeometry`"];
```

■ サンプルデータ

kNN分類の場合、核になるデータがあらかじめ与えられている点が特徴的。

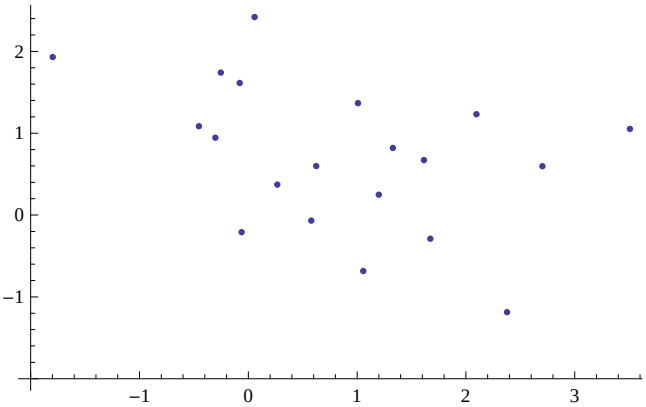
In[96]:=
```
means = {{-0.253433, 1.74148}, {0.266693, 0.371234}, {2.09647, 1.23336},
   {-0.0612727, -0.208679}, {2.70354, 0.596828}, {2.37721, -1.18641},
   {1.05691, -0.683894}, {0.578884, -0.0683458}, {0.624252, 0.598738},
   {1.67335, -0.289316}, {1.19937, 0.248409}, {-0.302561, 0.945419},
   {0.0572723, 2.41973}, {1.32932, 0.819226}, {-0.0793842, 1.6138}, {3.50793, 1.05299},
   {1.61392, 0.671738}, {1.00754, 1.36831}, {-0.454621, 1.08607}, {-1.79802, 1.92978}};
```

In[97]:=
```
means // Length
```

Out[97]=
```
20
```

In[98]:=
```
ListPlot[means, AxesOrigin → {-2, -2}]
```

Out[98]=



指定した点 （例。means[[4]]） を平均、σを分散とする正規分布に従うサンプルデータを用意 。関数PDは確率密度関数を求める。

In[99]:=
```
σ = Sqrt[1 / 5.0];
PDF[NormalDistribution[First[means[[4]]], σ], x]
```

Out[100]=
$$0.892062\, e^{-2.5\,(0.0612727+x)^2}$$

任意に選んだ点の周囲に上記正規分布によって得られる確率密度空間内の任意の座標を設定。なお元の中心座標のxとyの値をこの分散の平均値として使用。

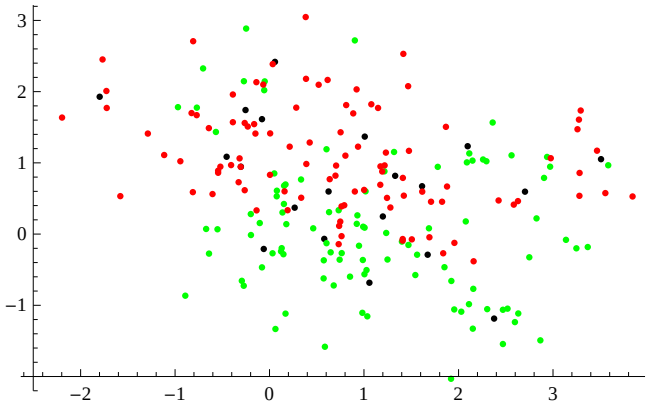
In[101]:=
```
greenRV[] :=
  Module[{i}, i = RandomInteger[{1, 10}]; {Random[NormalDistribution[First[means[[i]]], σ]],
    Random[NormalDistribution[Last[means[[i]]], σ]]}];
redRV[] := Module[{i}, i = RandomInteger[{11, 20}];
   {Random[NormalDistribution[First[means[[i]]], σ]],
    Random[NormalDistribution[Last[means[[i]]], σ]]}];
```

In[103]:=
```
greenRV[]
```

Out[103]=
```
{0.447906, 0.489674}
```

In[104]:=
```
SeedRandom[29 771 083]; n = 100;
y = Join[Array[0 &, n], Array[1 &, n]];
xG = Array[greenRV[] &, n];
xR = Array[redRV[] &, n];
x1x2y = Transpose[Join[Transpose[Join[xG, xR]], {y}]];
{x1, x2, y} = Transpose[x1x2y];
ListPlot[{means, xG, xR}, PlotStyle → {Black, Green, Red}, AxesOrigin → {-2.5, -2}]
```

Out[110]=



In[111]:=
```
x1x2y // Short[#, 4] &
```

Out[111]//Short=
```
{{2.14748, 1.03185, 0}, {3.36832, -0.181137, 0},
 {-0.247176, 2.88748, 0}, {1.00591, 0.0900456, 0}, <<193>>,
 {-0.519905, 0.94652, 1}, {0.761488, 0.388264, 1}, {1.07735, 1.82401, 1}}
```

■ 分類の準備

In[112]:=
```
EU[u_, v_] := EuclideanDistance[Most[u], Most[v]];
GetNearest[xy_, x0_, k_] := Nearest[xy, Append[x0, 0], k, DistanceFunction → EU];
```

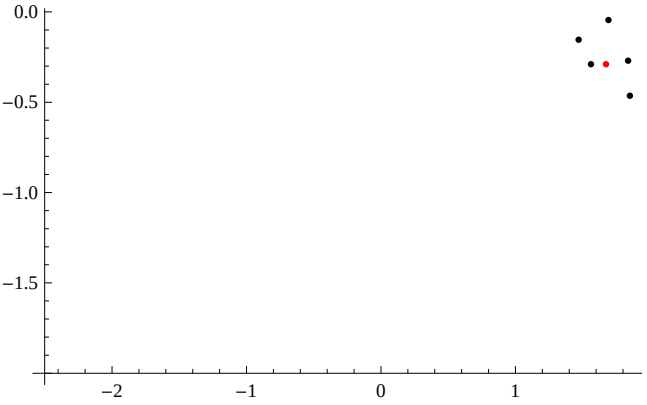
ユークリッド距離で測ったとして中心means[[10]]に最も近い５個の点を選択。

In[113]:=
```
GetNearest[x1x2y, means[[10]], 5](* 5-NN *)
```

Out[113]=
```
{{1.56166, -0.289222, 0}, {1.83769, -0.270449, 1},
 {1.46963, -0.153749, 0}, {1.69157, -0.0448326, 1}, {1.8508, -0.464317, 0}}
```

In[114]:=
```
ListPlot[{{means[[10]]}, Most /@ %}, PlotStyle → {Red, Black}, AxesOrigin → {-2.5, -2}]
```

Out[114]=



中心x0、近傍kを想定した場合、0側にある点の個数と1側にある点の個数が等しい時、ランダムにどちらかを選びそれ以外は多く存在する方を選ぶ。

In[115]:=
```
predictNNB[x0_, xy_, k_] := Module[{z},
  z = Tally[Last /@ GetNearest[xy, x0, k]];
  z = Sort[z, Last[#1] ≥ Last[#2] &];
  First[Transpose[Select[z, Last[First[z]] == Last[#] &]]] // RandomChoice
 ]
```

In[116]:=
```
GetNearest[x1x2y, means[[10]], 8]
predictNNB[means[[10]], x1x2y, 8]
```

Out[116]=
```
{{1.56166, -0.289222, 0}, {1.83769, -0.270449, 1},
 {1.46963, -0.153749, 0}, {1.69157, -0.0448326, 1}, {1.8508, -0.464317, 0},
 {1.50579, -0.0751388, 1}, {1.54365, -0.574626, 0}, {1.40967, -0.0931181, 1}}
```

Out[117]=
```
1
```

In[118]:=
```
GetNearest[x1x2y, means[[10]], 5]
predictNNB[means[[10]], x1x2y, 5]
```

Out[118]=
```
{{1.56166, -0.289222, 0}, {1.83769, -0.270449, 1},
 {1.46963, -0.153749, 0}, {1.69157, -0.0448326, 1}, {1.8508, -0.464317, 0}}
```

Out[119]=
```
0
```

In[120]:=
```
MapThread[f, {{a1, a2, a3}, {b1, b2, b3}}]
```

Out[120]=
```
{f[a1, b1], f[a2, b2], f[a3, b3]}
```

データ分布x1x2yの中のx1,x2の散らばり具合

In[121]:=
```
yp = MapThread[predictNNB[{#1, #2}, x1x2y, 5] &, {x1, x2}]
```

Out[121]=
```
{0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1}
```

In[122]:=
```
x1x2y2 = Transpose[{x1, x2, x1x2y, yp}]; x1x2y2[[19]]
```

Out[122]=
```
{0.168976, 0.696505, {0.168976, 0.696505, 0}, 0}
```

In[123]:=
```
If[Last[Dimensions[x1x2y2]] == 3,
      {x, y, indxy} = Transpose[x1x2y2];
      indxy = indxy + 1;
      indpoints = indxy;
  ,
      {x, y, indpoints, indxy} = Transpose[x1x2y2];
      indpoints = indpoints + 1;
      indxy = indxy + 1;
 ];
```

In[124]:=
```
indxy (* yp に対応 *)
```

Out[124]=
```
{1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1,
 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1,
 1, 1, 2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2,
 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2,
 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2}
```

In[125]:=
```
xy = Transpose[{x, y}];
{vorvert, vorval} = ComputationalGeometry`VoronoiDiagram[xy];
```

In[127]:=
```
vorvert // Short[#, 4] & (* Voronoi図頂点リスト。但しRay点は無限大に繋がる *)
```

Out[127]//Short=
```
{{-25.2069, -11.9505}, {-5.63458, 18.8295}, {-2.24876, 2.1848}, <<393>>,
 Ray[{12.313, 25.6219}, {13.6252, 28.5319}], Ray[{-2.24876, 2.1848}, {-3.06516, 2.61363}]}
```

In[128]:=
```
vorval // Short  (* Voronoi分割領域に入る点のリスト *)
```

Out[128]//Short=
```
{{1, {310, 321, 323, 319}}, {2, {386, 381, 378, 374, 372, 377}},
 <<196>>, {199, {161, 169, 189, 174}}, {200, {209, 211, 221, 246, 239}}}
```

In[129]:=
```
vorvert // Length
```

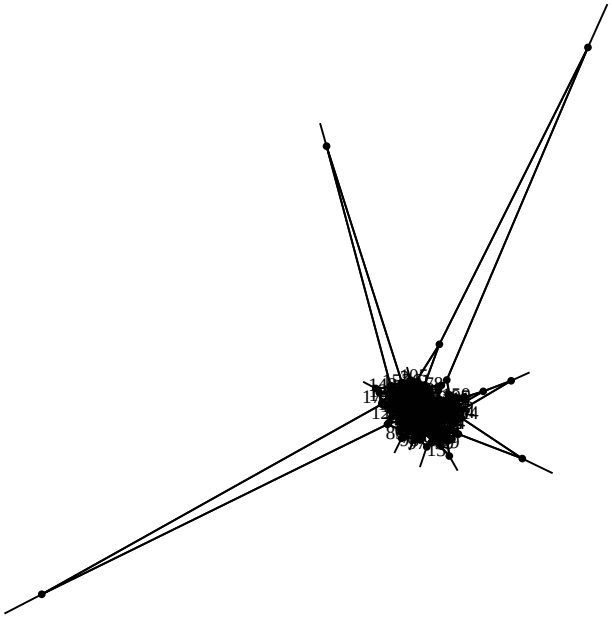
Out[129]= 398

In[130]:=
```
Select[vorvert, ListQ] // Length
```

Out[130]= 387

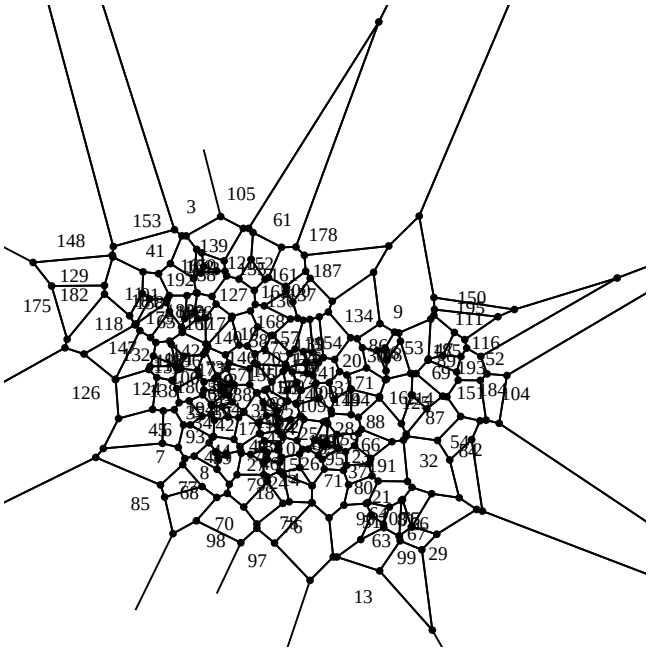
In[131]:=
```
DiagramPlot[xy, vorvert, vorval]
```

Out[131]=

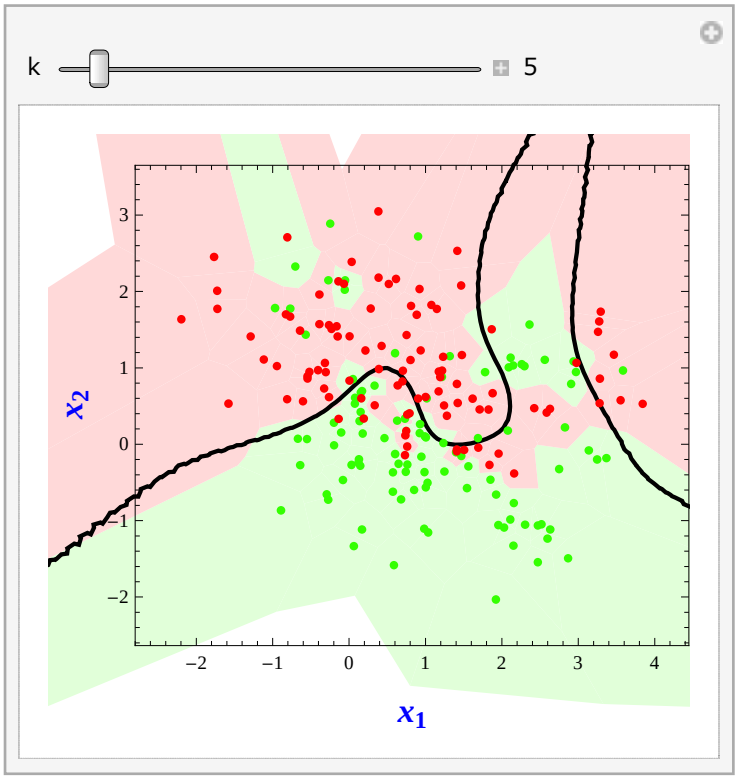


In[132]:=
```
DiagramPlot[xy, vorvert, vorval, TrimPoints → 6]
```

Out[132]=



修正が必要。参考http://demonstrations.wolfram.com/KNearestNeighborKNNClassifier/
近傍の数kを変えることで境界領域が変わることを見る。



リンク分析

クラスタ間の関係およびクラス内の影響力ある要素を見つけることで有効なアプローチ先を発見。

cf. http://econo.twinkle.cc/trackback/55
1. n 者間の互いの間に発生するトランザクション量をカウントする。
2.クラスタリングを用いてある一定の閾値以上の関係性を持つものだけでグルーピングする。
3.各々のグループ内において互いの関係の深さを表す量を基準化する。
4.各々のグループに対して、そのグループ内の全ての人物に対し、シャプレイ値 (グループ内へ与える影響度の度合) を計算し、スコアリングする。
5.グループとしての重要性 (例えば、そのグループが与えてくれる収益など) によって、キャンペーンを実施すべきグループを抽出。
6.抽出したグループのオピニオンリーダに対して、プロモーション活動を実施。