Advanced Machine Learning

CSC 722 Spring 2014

Colin Potts, Devendra Chavan

Department of Computer Science
North Carolina State University

Support Vector Machines

SVMs for regression

In simple regression, the objective is to minimize a regularized error function

$$\frac{1}{2}\sum_{n=1}^{N}\{y_n - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2 \tag{1}$$

To obtain sparse solutions, the above equation can be replaced by a ϵ - *insensitive error function*

$$E_\epsilon(y(\mathbf{x}) - t) = \begin{cases} 0 & \text{if } |y(\mathbf{x}) - t| < \epsilon \\ |y(\mathbf{x}) - t| - \epsilon & \text{otherwise} \end{cases} \tag{2}$$

Minimizing the error function

To obtain the solution, minimize the regularized error function

$$C \sum_{n=1}^{N} E_\epsilon(y(\mathbf{x}_n) - t_n) + \frac{1}{2}\|\mathbf{w}\|^2 \tag{3}$$

where C is the (inverse) regularization parameter

Error function with slack variables

By introducing slack variables for each data point x_n

- $\xi_n \geq 0$ where $t_n > y(\mathbf{x}_n) + \epsilon$
- $\widehat{\xi}_n \geq 0$ where $t_n < y(\mathbf{x}_n) + \epsilon$

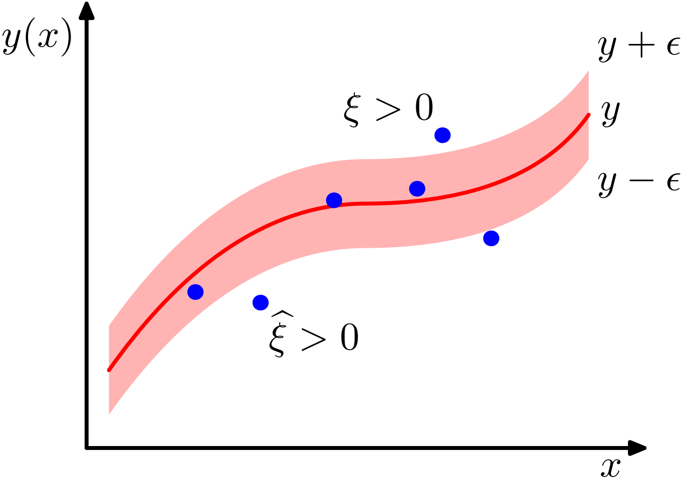
the error function can be written as

$$C \sum_{n=1}^{N} (\xi_n + \widehat{\xi}_n) + \frac{1}{2} \|\mathbf{w}\|^2 \tag{4}$$

This is to be minimized subject to the constraints

- $\widehat{\xi}_n \geq 0$ and $\xi_n \geq 0$
- $t_n \leq y(\mathbf{x}_n) + \epsilon + \xi_n$
- $t_n \geq y(\mathbf{x}_n) - \epsilon - \widehat{\xi}_n$

SVM Regression *tube*



Applying Lagrange multipliers

Plugging in the Lagrange multipliers and simplifying,

$$\widetilde{L}(\mathbf{a}, \widehat{\mathbf{a}}) = -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n - \widehat{a}_n)(a_m - \widehat{a}_m)k(\mathbf{x}_n, \mathbf{x}_m) - \epsilon\sum_{n=1}^{N}(a_n - \widehat{a}_n) + \sum_{n=1}^{N}(a_n - \widehat{a}_n)t_n \tag{5}$$

where $a_n \geq 0$ and $\widehat{a_n} \geq 0$ are the Lagrange multipliers; and
$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}')$ is the kernel.

Predicting new inputs

The prediction can be made for new inputs using

$$y(\mathbf{x}) = \sum_{n=1}^{N} (a_n - \widehat{a}_n) k(\mathbf{x}, \mathbf{x}_n) + b \tag{6}$$

ν-SVMs

- An alternative for of SVM for regression
- Use a parameter ν that bounds the fraction of points lying *outside* the tube instead of fixing the width ϵ of the *insensitive* region

Maximizing the dual

$$\widetilde{L}(\mathbf{a}, \widehat{\mathbf{a}}) = -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n - \widehat{a}_n)(a_m - \widehat{a}_m)k(\mathbf{x}_n, \mathbf{x}_m) + \sum_{n=1}^{N}(a_n - \widehat{a}_n)t_n \tag{7}$$

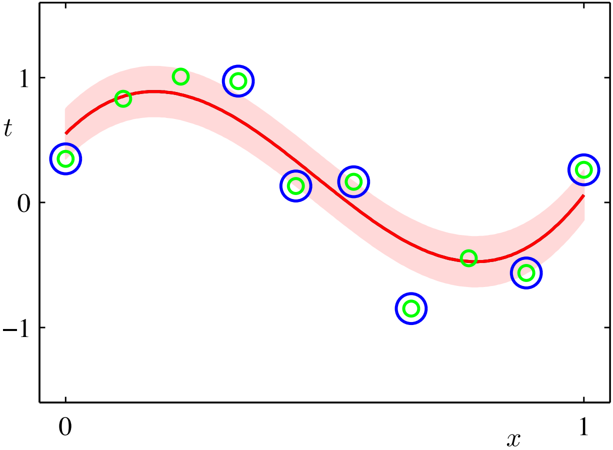
subject to the constraints

- $0 \leq a_n \leq C/N$
- $0 \leq \widehat{a}_n \leq C/N$
- $\sum_{n=1}^{N}(a_n - \widehat{a}_n) = 0$
- $\sum_{n=1}^{N}(a_n + \widehat{a}_n) \leq \nu C$

Observations

- At most νN data points fall outside the insensitive tube, while at least νN data points are the support vectors that lie on or outside the tube
- Parameters ν and C are determined by cross-validation

ν-SVM regression



Relevance Vector Machines

- Bayesian sparse kernel technique similar to SVM
- Leads to much sparser models resulting in faster performance on test data while maintaining the same generalization error

Limitations of SVM

- Outputs of SVM represent decisions rather than posterior probabilities
- Extension to $K > 2$ classes is problematic
- Complexity parameter C or ν to be found out using cross validation
- Kernels used are centered on training data points and are required to be *positive definite* (Mercer kernels)

RVM for regression

Consider the model defining a conditional distribution for a real-valued target variable t, given an input vector $\mathbf{x}$

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}), \beta^{-1}) \tag{8}$$

where $\beta = \sigma^{-2}$ is the noise precision and mean is of the form

$$y(\mathbf{x}) = \sum_{i=1}^{M} w_i \phi_i(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) \tag{9}$$

Using kernel for each data point in place of the the bias function $\phi_i(\mathbf{x})$

$$y(\mathbf{x}) = \sum_{n=1}^{N} w_n k(\mathbf{x}, \mathbf{x}_n) + b \tag{10}$$

Given a set of set of N observations of the input vector $\mathbf{x}$, which can be denote collectively by data matrix $\mathbf{X}$ whose n^{th} row is $\mathbf{x}_n^T$ with $n = 1, \ldots, N$ and corresponding target values $\mathbf{t} = (t_1, \ldots, t_N)^T$, the *likelihood* function is

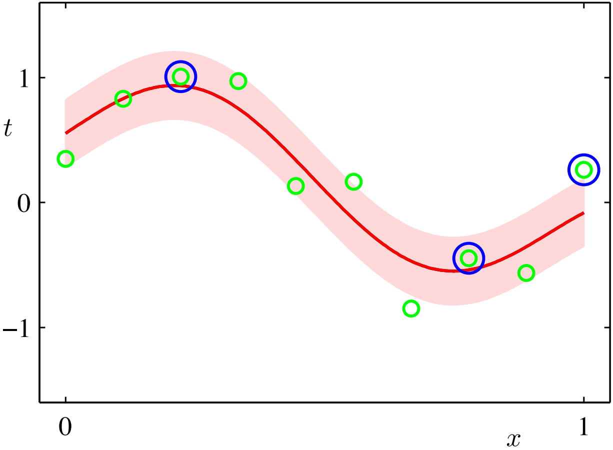
$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta^{-1}) \tag{11}$$

Simplifying, the predictive distribution over t for a new input x is

$$p(t|\mathrm{x}, \mathbf{X}, \mathbf{t}, \alpha^*, \beta^*) = \mathcal{N}(t|\mathbf{m}^T\phi(\mathrm{x}), \sigma^2(\mathrm{x})) \qquad (12)$$

where $\alpha^* and \beta^*$ are hyperparameters that maximize the marginal likelihood and
m is the posterior mean and variance of the predictive distribution is $\sigma^2(\mathrm{x})$

RVM vs ν-SVM regression



RVM vs SVM

- Training involves optimizing a non convex function $\Rightarrow$ longer training time $O(N^3)$ as compared to SVM $O(N^2)$
- Parameters governing the complexity and noise variance are determined automatically from a single run, as compared to SVM which requires multiple training runs (cross validation) for determining C and ϵ (or ν)

SVM vs Logistic regression

Given the number of features n and the number of training samples m

- n is large relative to m: Logistic regression or SVM with linear kernel
- n is small and m is intermediate: SVM with Gaussian kernel
- n is small and m is large: Create/add more features, then use logistic regression or SVM with linear kernel

SVM vs ANN

- ANNs can be used when there is a large number of training sample available
- SVMs have a better generalization than ANNs due to Structural Risk Minimization (SRM) principle

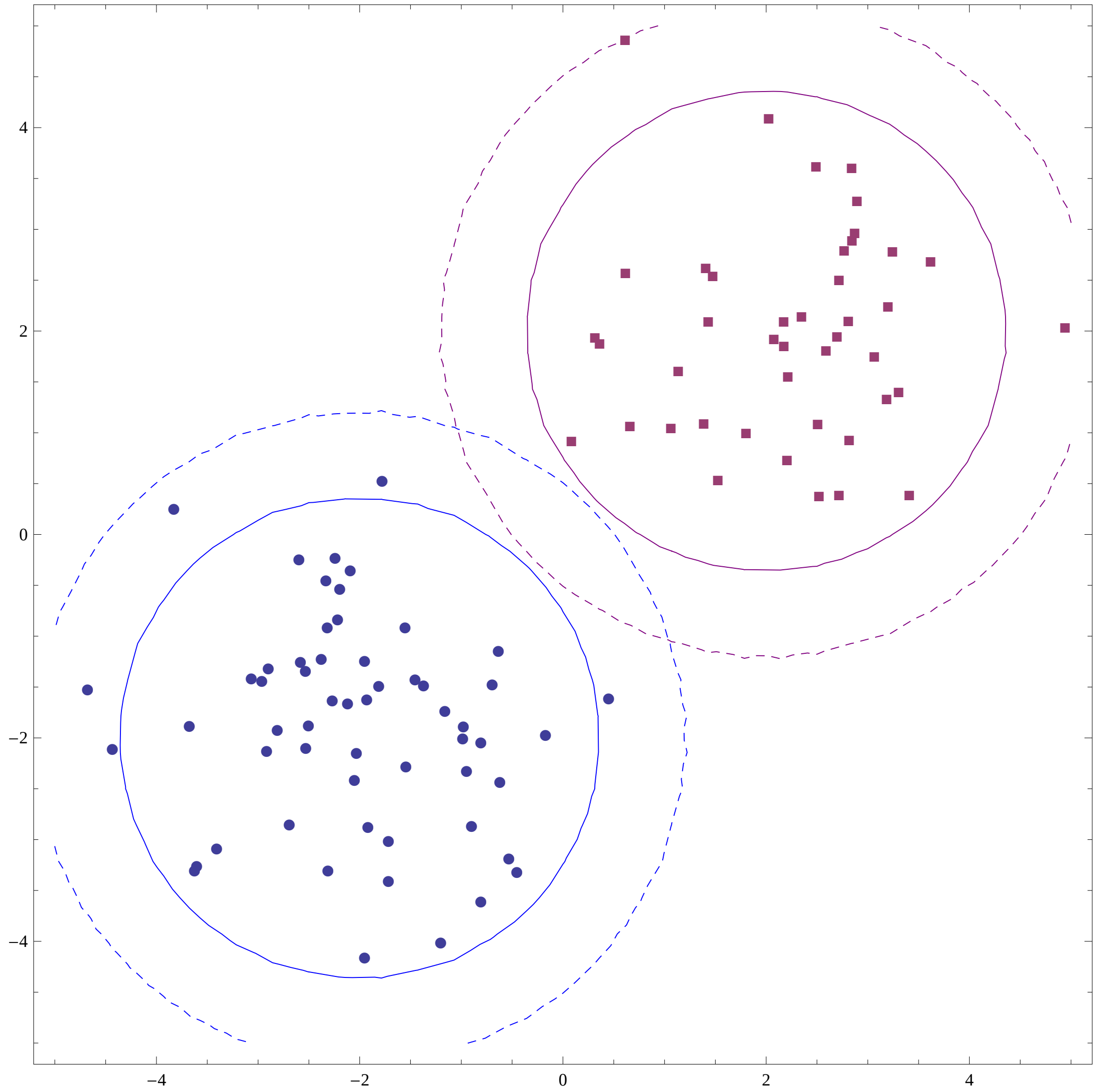
Demo of SVM in Mathematica

Prelims

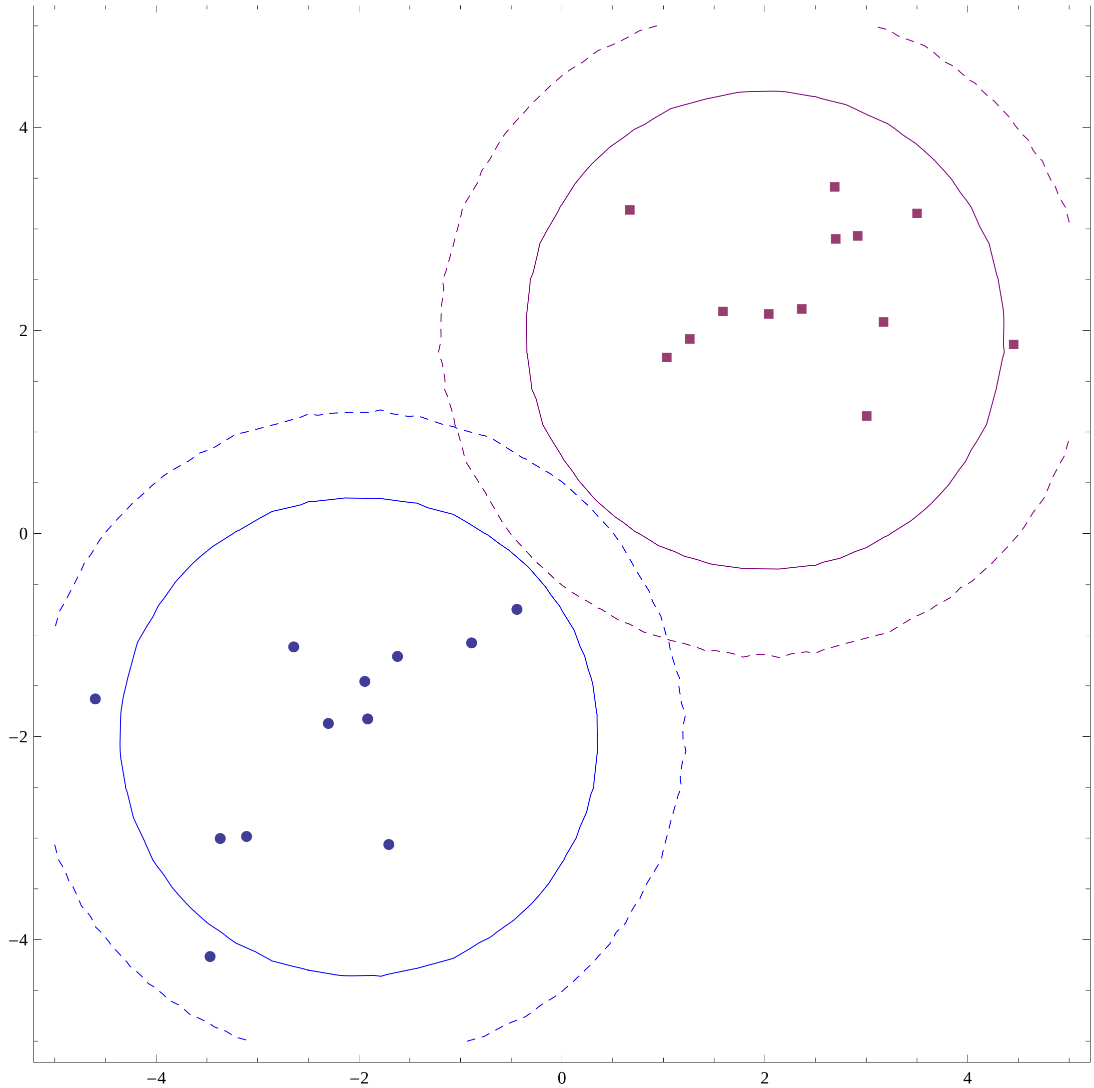
Linearly Seperable Data

```
GenData[n_] := Module[{X, y, pos, neg, npos, nneg},
   {npos} = Ceiling[n RandomVariate[NormalDistribution[0.5, 0.25], 1]];
   nneg = n - npos;
   X = Join[
     RandomVariate[NormalDistribution[-2, 1], {npos, 2}],
     RandomVariate[NormalDistribution[2, 1], {nneg, 2}]];
   y = Join[Table[1, {npos}], Table[-1, {nneg}]];
   pos = Take[X, npos];
   neg = Take[X, -nneg];
   {X, y, pos, neg}];

dlen = 100;
{X, y, pos, neg} = GenData[dlen];

tlen = 25;
{tpos, tneg} = Take[GenData[tlen], -2];

distro = ContourPlot[
   {PDF[NormalDistribution[-2, 1], x] PDF[NormalDistribution[-2, 1], y] == 0.001,
    PDF[NormalDistribution[2, 1], x] PDF[NormalDistribution[2, 1], y] == 0.001,
    PDF[NormalDistribution[-2, 1], x] PDF[NormalDistribution[-2, 1], y] == 0.01,
    PDF[NormalDistribution[2, 1], x] PDF[NormalDistribution[2, 1], y] == 0.01},
   {x, -5, 5}, {y, -5, 5}, ContourStyle →
    {Directive[Dashed, Blue], Directive[Dashed, Purple], Blue, Purple}];
```

```
Show[distro, dataPlot = ListPlot[{pos, neg}, PlotMarkers → Automatic]]
```



```
Show[distro, testPlot = ListPlot[{tpos, tneg}, PlotMarkers → Automatic]]
```




```
τ = 0.01;
α = SeparableSVM[X, y, τ]
```

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.530622, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.281524, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0.812146, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

```
w = {WeightVector[α, X, y]}
```

```
{{-0.884729, -0.921494}}
```

```
b = Bias[α, X, y]
```

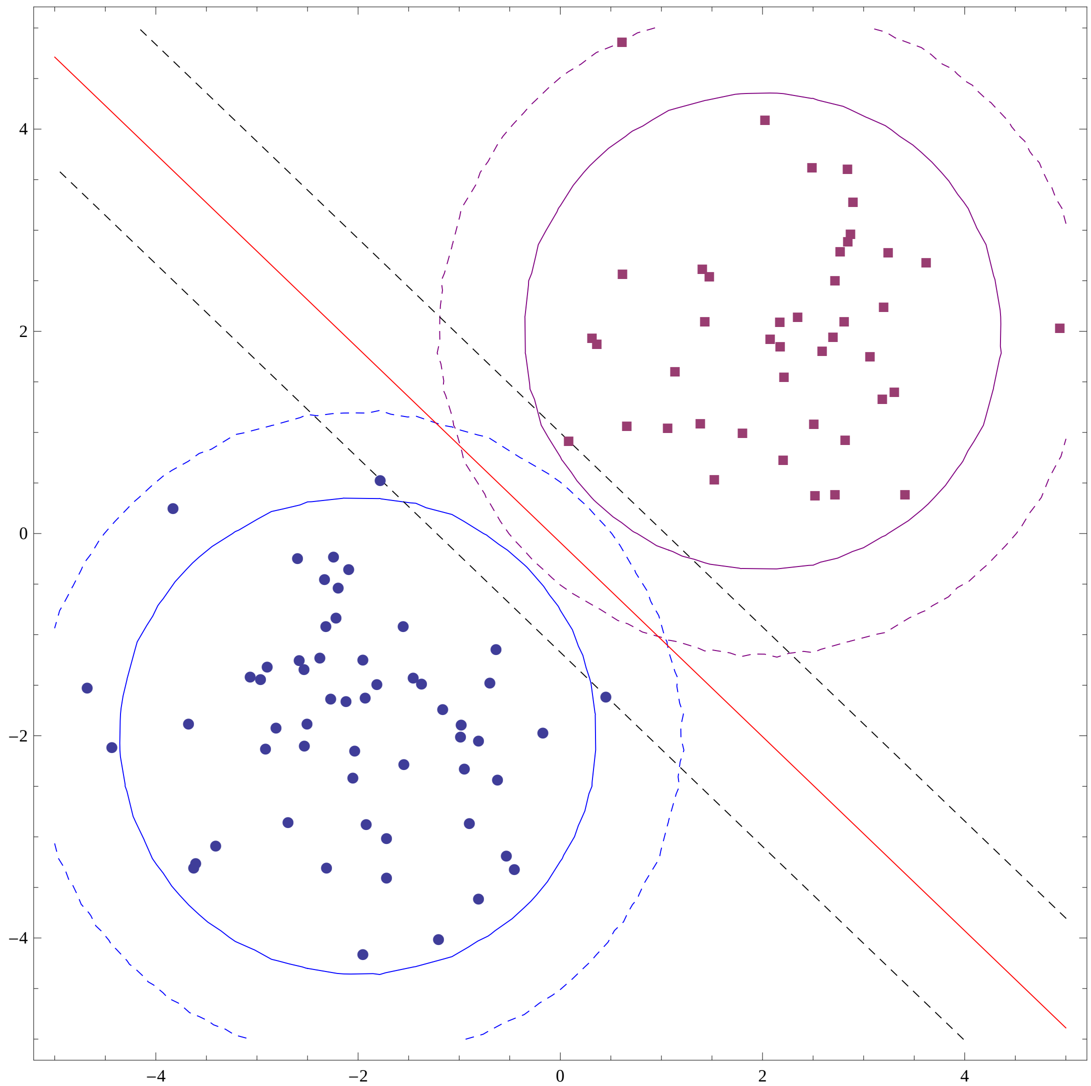
```
-0.0816545
```

```
g[x_, y_] := w.{{x}, {y}} + b
```

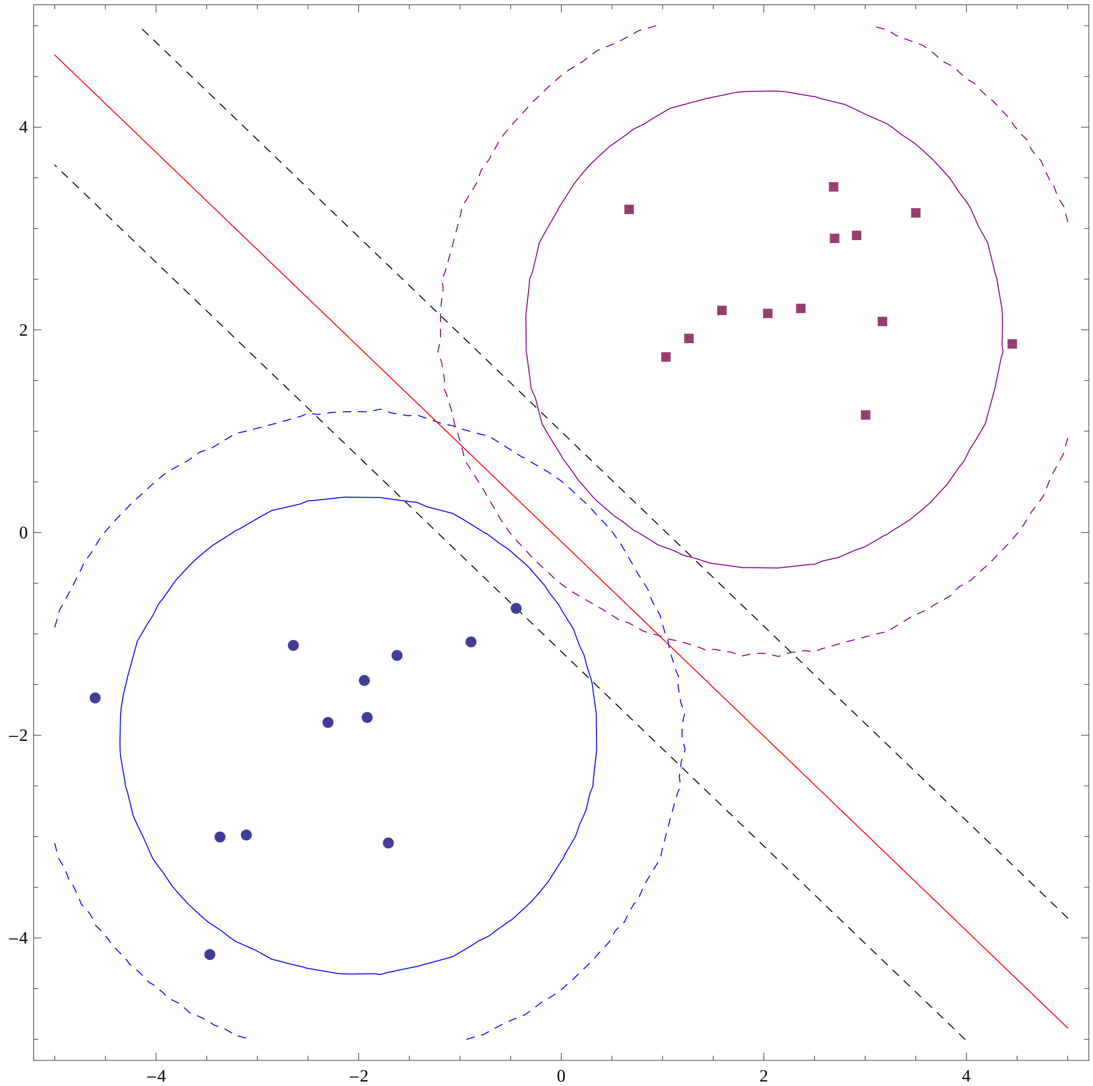
```
contour = ContourPlot[{g[x, y] == -1, g[x, y] == 0, g[x, y] == 1},
   {x, -5, 5}, {y, -5, 5}, ContourStyle → {Dashed, Red, Dashed}];
PDF[NormalDistribution[2, 1], x]
```

$$\frac{e^{-\frac{1}{2}(-2+x)^2}}{\sqrt{2\pi}}$$

```
Show[distro, contour, dataPlot]
```



```
Show[distro, contour, testPlot]
```



Spiral Data

$$\texttt{func[t\_, y\_] := } \frac{2}{17\,\mathrm{Pi}} \texttt{\{y t Sin[t], y t Cos[t]\}}$$

```
Clear[next];
```

$$\texttt{next[a\_, n\_] := next[a, n] = a + NArgMin}\left[\left\{\frac{1}{n}\,\texttt{Sum}\left[\texttt{Norm}\left[\texttt{D}\left[\texttt{func}\left[\texttt{a + i}\,\frac{b}{n}\texttt{, 1}\right]\right]\right]\texttt{, \{i, 0, n\}}\right]\texttt{,}\right.\right.$$
$$\left.\left.\frac{1}{n}\,\texttt{Sum}\left[\texttt{Norm}\left[\texttt{D}\left[\texttt{func}\left[\texttt{a + i}\,\frac{b}{n}\texttt{, 1}\right]\right]\right]\texttt{, \{i, 0, n\}}\right] > \frac{1}{20}\texttt{, b > 0}\right\}\texttt{, \{b\}}\right]\texttt{[[1]];}$$

```
Clear[next];
next[a_, n_] := next[a, n] = a + NArgMin[{Norm[func[a, 1] - func[a + b, 1]],
    Norm[func[a, 1] - func[a + b, 1]] > 1 / 20, b > 0}, {b}][[1]];
```

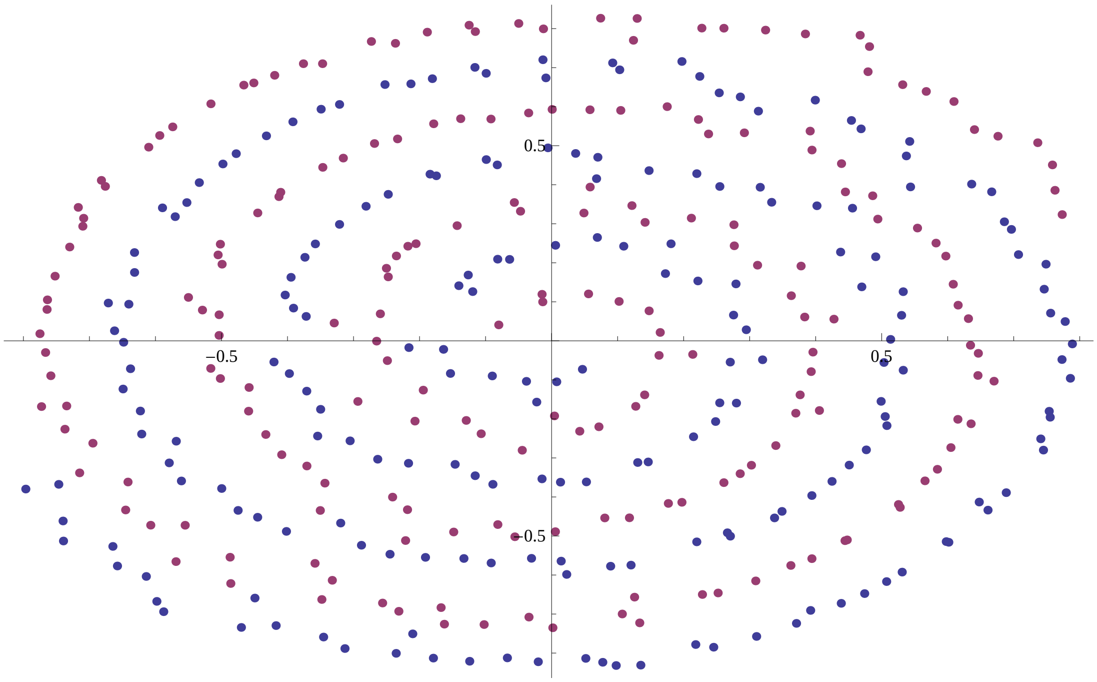
```
indices = Table[Nest[next[#, 10] &, 13 / 25 Pi, i], {i, 1, 200}]
```

```
{2.25815, 2.75919, 3.18869, 3.57022, 3.91676, 4.23637, 4.53442, 4.81472, 5.08008,
 5.33265, 5.57409, 5.80576, 6.02873, 6.24392, 6.45208, 6.65386, 6.8498, 7.04039,
 7.22603, 7.40709, 7.5839, 7.75673, 7.92584, 8.09147, 8.25382, 8.41307, 8.56939,
 8.72295, 8.87388, 9.02231, 9.16836, 9.31214, 9.45376, 9.59331, 9.73087, 9.86653,
 10.0004, 10.1325, 10.2629, 10.3916, 10.5188, 10.6445, 10.7688, 10.8916, 11.0131,
 11.1333, 11.2522, 11.3698, 11.4863, 11.6016, 11.7157, 11.8288, 11.9408, 12.0518,
 12.1617, 12.2707, 12.3788, 12.4859, 12.5921, 12.6974, 12.8018, 12.9054, 13.0082,
 13.1102, 13.2114, 13.3118, 13.4115, 13.5105, 13.6087, 13.7063, 13.8031, 13.8993,
 13.9948, 14.0897, 14.184, 14.2776, 14.3706, 14.463, 14.5549, 14.6461, 14.7368, 14.827,
 14.9166, 15.0057, 15.0942, 15.1822, 15.2698, 15.3568, 15.4434, 15.5294, 15.615,
 15.7001, 15.7848, 15.869, 15.9528, 16.0361, 16.119, 16.2015, 16.2836, 16.3653,
 16.4465, 16.5274, 16.6078, 16.6879, 16.7676, 16.8469, 16.9259, 17.0045, 17.0827,
 17.1606, 17.2381, 17.3153, 17.3921, 17.4686, 17.5447, 17.6206, 17.6961, 17.7713,
 17.8461, 17.9207, 17.995, 18.0689, 18.1426, 18.2159, 18.289, 18.3617, 18.4342,
 18.5064, 18.5783, 18.65, 18.7213, 18.7924, 18.8633, 18.9338, 19.0041, 19.0742,
 19.144, 19.2135, 19.2828, 19.3518, 19.4206, 19.4892, 19.5575, 19.6256, 19.6934,
 19.761, 19.8284, 19.8955, 19.9625, 20.0292, 20.0957, 20.1619, 20.228, 20.2938,
 20.3594, 20.4248, 20.49, 20.555, 20.6198, 20.6844, 20.7488, 20.813, 20.877, 20.9407,
 21.0044, 21.0678, 21.131, 21.194, 21.2569, 21.3195, 21.382, 21.4443, 21.5064,
 21.5683, 21.6301, 21.6917, 21.7531, 21.8143, 21.8754, 21.9363, 21.997, 22.0576,
 22.118, 22.1782, 22.2383, 22.2982, 22.3579, 22.4175, 22.477, 22.5362, 22.5953,
 22.6543, 22.7131, 22.7718, 22.8303, 22.8887, 22.9469, 23.0049, 23.0629, 23.1206}
```

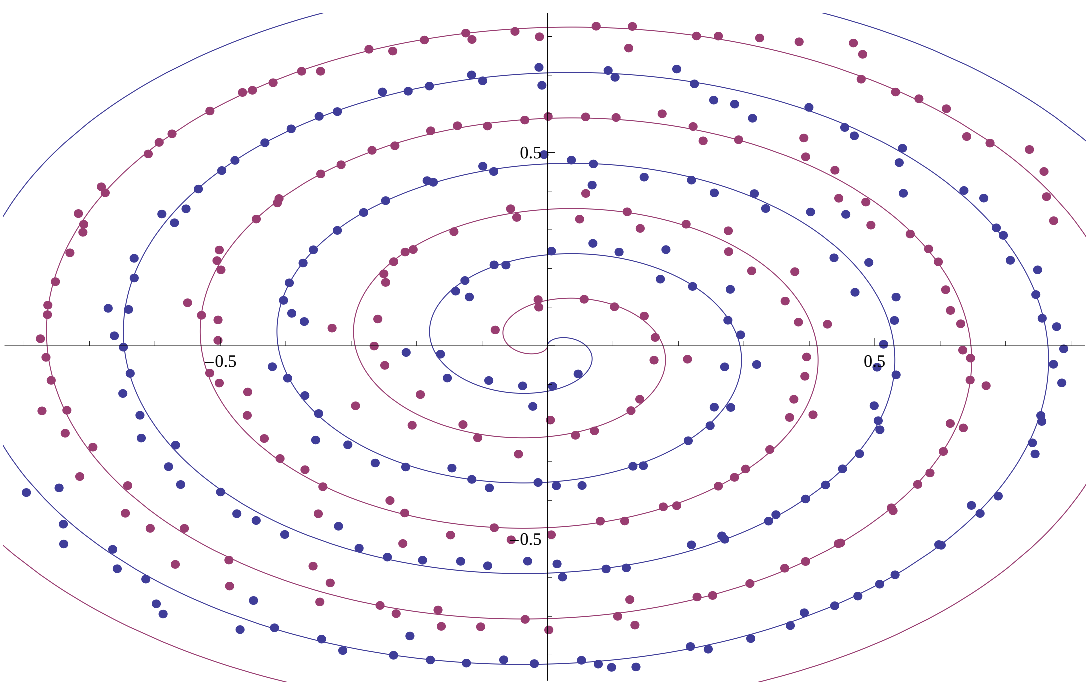
```
funcPlot = ParametricPlot[{func[t, 1], func[t, -1]}, {t, 0, (8 + 1 / 2) Pi}];
```

```
Nd = 200;
(*data=Table[Join[func[(i+12)/25Pi,(-1)^i+RandomReal[]/25]
     (*+{RandomReal[],RandomReal[]}/15*),{(-1)^i}],{i,1,Nd}]//N;*)
(*data=Import["spiral-data.csv"];*)
dist = NormalDistribution[0, 0.02];
RandomIndices[n_, k_] := Intersection[Round /@ Table[RandomReal[] * n, {k}]];
genPoint[t_, c_] :=
  Join[func[t + RandomReal[] / 25, c] + RandomVariate[dist] {1, 1}, {c}];
data = Join[Table[genPoint[indices[[i]], 1], {i, 1, Length[indices]}],
   Table[genPoint[indices[[i]], -1], {i, 1, Length[indices]}]];
test = Join[Table[genPoint[indices[[i]], 1],
     {i, RandomIndices[Length[indices], Ceiling[0.2 Length[indices]]]}],
   Table[genPoint[indices[[i]], -1],
     {i, RandomIndices[Length[indices], Ceiling[0.2 Length[indices]]]}]];
Nd = Length[data];
xdata = Table[{data[[i, 1]], data[[i, 2]]}, {i, 1, Nd}];
ydata = Table[data[[i, 3]], {i, 1, Nd}];
pos = Table[{item[[1]], item[[2]]}, {item, Select[data, #[[3]] > 0 &]}];
neg = Table[{item[[1]], item[[2]]}, {item, Select[data, #[[3]] < 0 &]}];
tpos = Table[{item[[1]], item[[2]]}, {item, Select[test, #[[3]] > 0 &]}];
tneg = Table[{item[[1]], item[[2]]}, {item, Select[test, #[[3]] < 0 &]}];
```

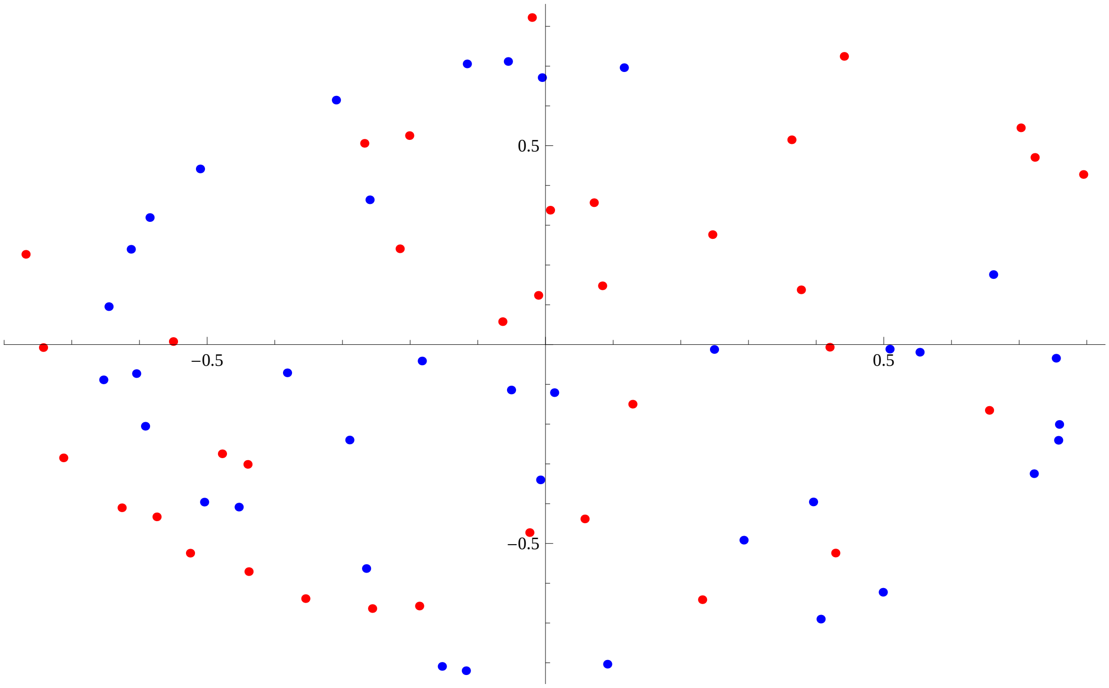
```
dPlot = ListPlot[{pos, neg}, PlotStyle → PointSize[Medium]]
```




```
Show[dPlot, funcPlot]
```



```
tPlot = ListPlot[{tpos, tneg},
  PlotStyle → {Directive[PointSize[Medium], Blue], Directive[PointSize[Medium], Red]}]
```



■ As a separable dataset

```
? SeparableSVM
```

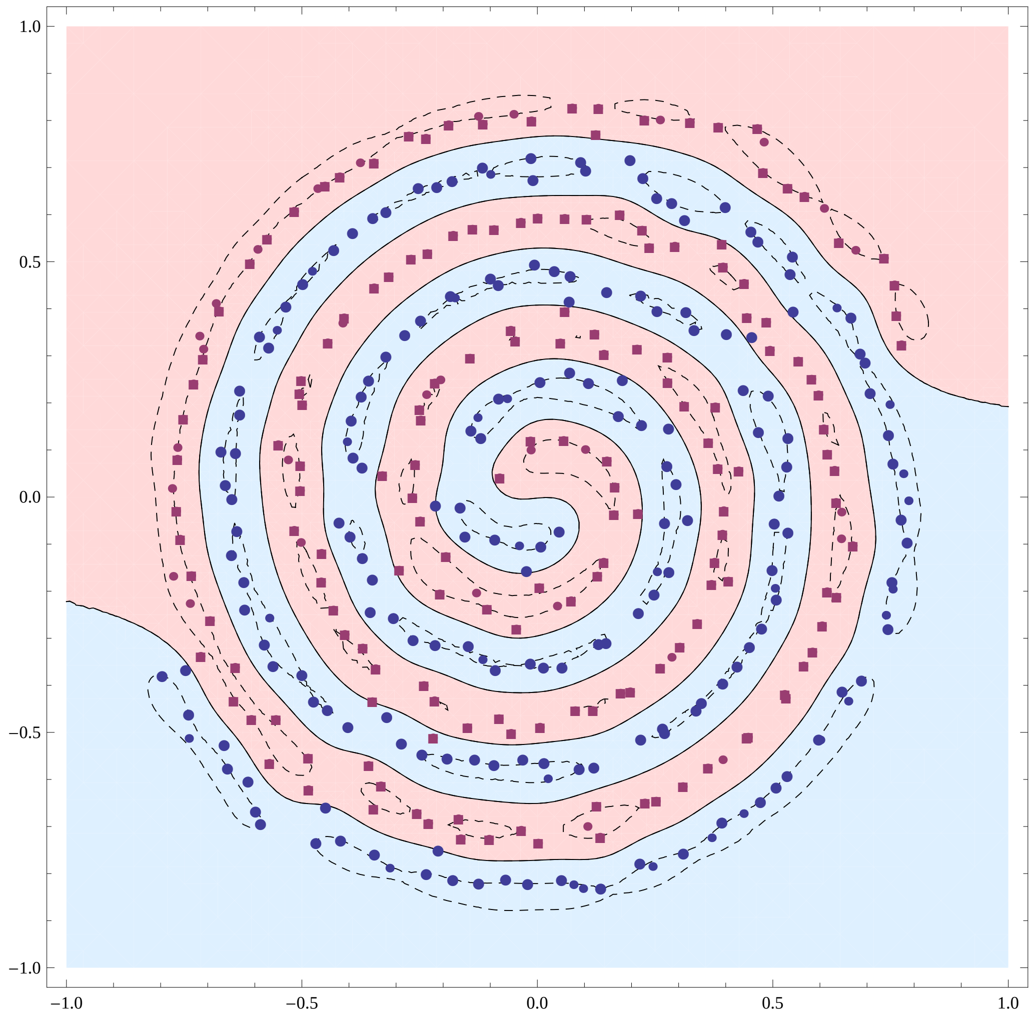
SeparableSVM[X,y,τ] trains a separable SVM on data X, labels y, and solution tolerance τ (see QPSolve). Returns the multiplier vector α. Option KernelFunction determines the kernel to use; default is IdentityKernel (i.e., no kernel).

```
krbf[x_, y_] := RBFKernel[x, y, 100];

α = SeparableSVM[xdata, ydata, 0.01, KernelFunction → krbf];

ynew[α_, X_, y_, k_] := Sum[α[[i]] y[[i]] k[#, X[[i]]], {i, 1, Length[X]}] &

rbfY = ynew[α, xdata, ydata, krbf];

rbfPlot = ContourPlot[rbfY[{x, y}], {x, -1, 1}, {y, -1, 1}, Contours → {-1, 0, 1},
   ContourStyle → {Dashed, Black, Dashed}, ContourShading → None];

rbfPlotShading = ContourPlot[rbfY[{x, y}], {x, -1, 1}, {y, -1, 1}, Contours → {0},
   ContourStyle → {Dashed, Black, Dashed}, ContourShading → {LightRed, LightBlue}];

SVs = ListPlot[{({#[[1]], #[[2]]}) & /@ #1[[1]], ({#[[1]], #[[2]]}) & /@ #2[[1]]} & @@
    SplitBy[Table[data[[i]], {i, SupportVectors[α, ydata]}], #[[3]] &],
   PlotMarkers → Automatic];
```

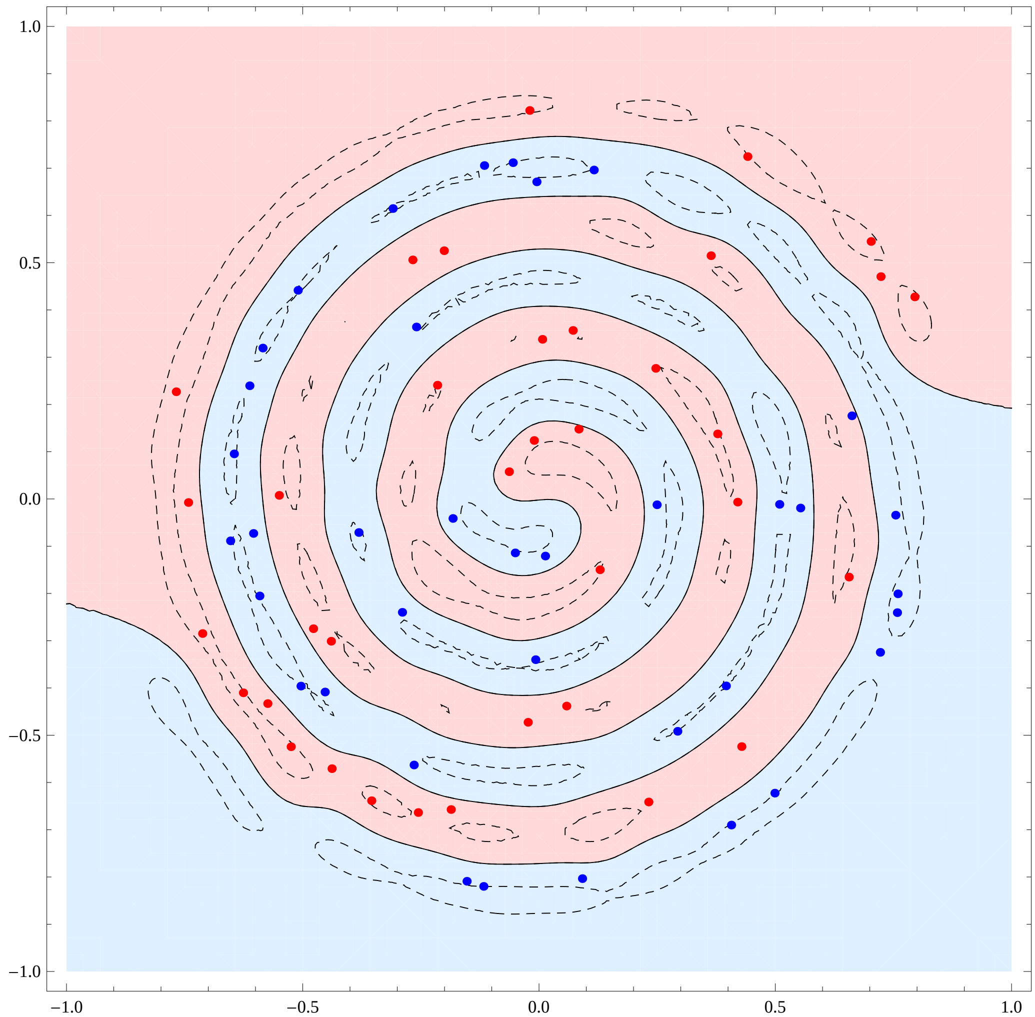
```
SupportVectors[α, ydata]
```

```
{{1, 2, 3, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27, 28, 29,
  30, 31, 32, 33, 34, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52,
  53, 54, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
  75, 76, 77, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 92, 93, 94, 95, 96, 97,
  98, 99, 100, 101, 102, 103, 104, 105, 107, 108, 109, 110, 111, 112, 113, 114, 115,
  116, 117, 119, 120, 121, 123, 124, 125, 126, 127, 128, 129, 130, 132, 133, 134, 135,
  136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 148, 149, 150, 151, 153, 154,
  157, 158, 159, 162, 163, 164, 166, 168, 169, 170, 173, 174, 176, 177, 180, 181, 182,
  183, 184, 185, 186, 188, 189, 190, 191, 192, 193, 194, 195, 196, 198, 199, 200},
 {201, 202, 204, 206, 207, 208, 209, 210, 211, 212, 214, 215, 216, 218, 219, 220, 221,
  222, 223, 224, 225, 226, 228, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240,
  241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 253, 254, 255, 256, 257, 258,
  259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 273, 274, 275, 277,
  278, 279, 280, 281, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295,
  296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 314,
  315, 316, 317, 318, 319, 320, 321, 322, 323, 325, 326, 327, 328, 329, 331, 332, 333,
  334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350,
  352, 354, 355, 357, 359, 360, 361, 364, 366, 368, 369, 371, 372, 374, 375, 376, 377,
  379, 381, 382, 383, 384, 385, 387, 388, 389, 391, 392, 393, 395, 397, 398, 399, 400}}
```

```
Show[rbfPlotShading, rbfPlot, dPlot, SVs]
```



```
Show[rbfPlotShading, rbfPlot, tPlot]
```



- **As a nonseparable dataset**

```
? NonseparableSVM
```

NonSeparableSVM[X, y, C, τ] trains a non-separable SVM on data X, labels y, penalty term C and solution tolerance τ (see QPSolve). Returns the multiplier vector α. Option KernelFunction determines the kernel to use; default is IdentityKernel (i.e., no kernel)

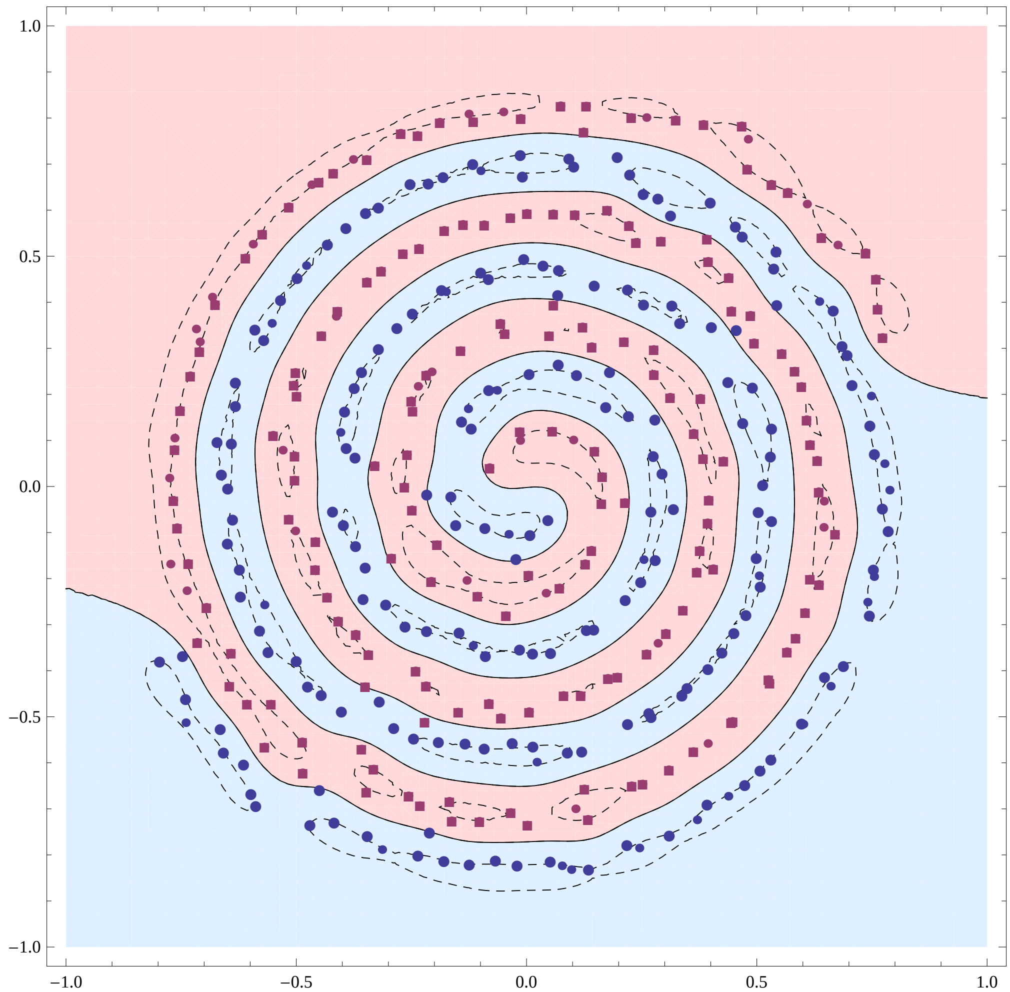
```
α2 = NonseparableSVM[xdata, ydata, 1.0, 0.01, KernelFunction → krbf];

rbfY2 = ynew[α, xdata, ydata, krbf];

rbfPlot2 = ContourPlot[rbfY2[{x, y}], {x, -1, 1}, {y, -1, 1}, Contours → {-1, 0, 1},
   ContourStyle → {Dashed, Black, Dashed}, ContourShading → None];

rbfPlotShading2 = ContourPlot[rbfY2[{x, y}], {x, -1, 1}, {y, -1, 1}, Contours → {0},
   ContourStyle → {Dashed, Black, Dashed}, ContourShading → {LightRed, LightBlue}];
```

```
SVs2 = ListPlot[{({#[[1]], #[[2]]}) & /@ #1[[1]], ({#[[1]], #[[2]]}) & /@ #2[[1]]} & @@
    SplitBy[Table[data[[i]], {i, SupportVectors[α2, ydata]}], #[[3]] &],
   PlotMarkers → Automatic];
```

```
SupportVectors[α2, ydata]
```

```
{{1, 2, 3, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27, 28, 29,
  30, 31, 32, 33, 34, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52,
  53, 54, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
  75, 76, 77, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 92, 93, 94, 95, 96, 97,
  98, 99, 100, 101, 102, 103, 104, 105, 107, 108, 109, 110, 111, 112, 113, 114, 115,
  116, 117, 119, 120, 121, 123, 124, 125, 126, 127, 128, 129, 130, 132, 133, 134, 135,
  136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 148, 149, 150, 151, 153, 154,
  157, 158, 159, 162, 163, 164, 166, 168, 169, 170, 173, 174, 176, 177, 180, 181, 182,
  183, 184, 185, 186, 188, 189, 190, 191, 192, 193, 194, 195, 196, 198, 199, 200},
 {201, 202, 204, 206, 207, 208, 209, 210, 211, 212, 214, 215, 216, 218, 219, 220, 221,
  222, 223, 224, 225, 226, 228, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240,
  241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 253, 254, 255, 256, 257, 258,
  259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 273, 274, 275, 277,
  278, 279, 280, 281, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295,
  296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 314,
  315, 316, 317, 318, 319, 320, 321, 322, 323, 325, 326, 327, 328, 329, 331, 332, 333,
  334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350,
  352, 354, 355, 357, 359, 360, 361, 364, 366, 368, 369, 371, 372, 374, 375, 376, 377,
  379, 381, 382, 383, 384, 385, 387, 388, 389, 391, 392, 393, 395, 397, 398, 399, 400}}
```

```
Show[rbfPlotShading2, rbfPlot2, dPlot, SVs2]
```



```
Show[rbfPlotShading2, rbfPlot2, tPlot]
```