Support Vector Machine(SVM)

葉建華

jhyeh@mail.au.edu.tw

http://jhyeh.csie.au.edu.tw/

Support Vector Machine

Support vector machines

Pros: Low generalization error, computationally inexpensive, easy to interpret results

Cons: Sensitive to tuning parameters and kernel choice; natively only handles binary classification

Works with: Numeric values, nominal values

Hard Problems

- Not linearly separable!

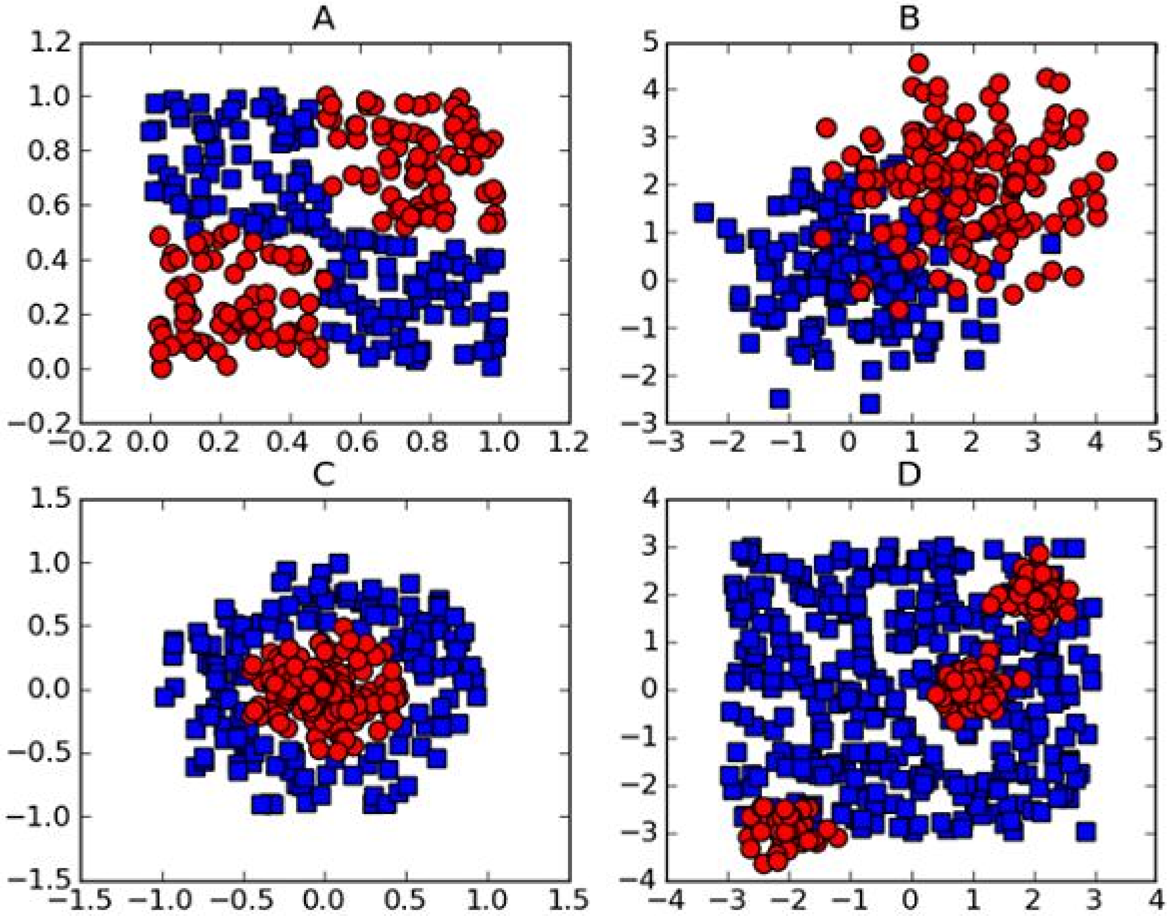


Figure 6.1 Four examples of datasets that aren't linearly separable

Separating Hyperplane

- The line used to separate the dataset
- If we have a dataset with N dimension, we need a plane with N-1 dimension to separate it
  - That's why it called hyper(plane)!

Linear Separable

- So which one is better?

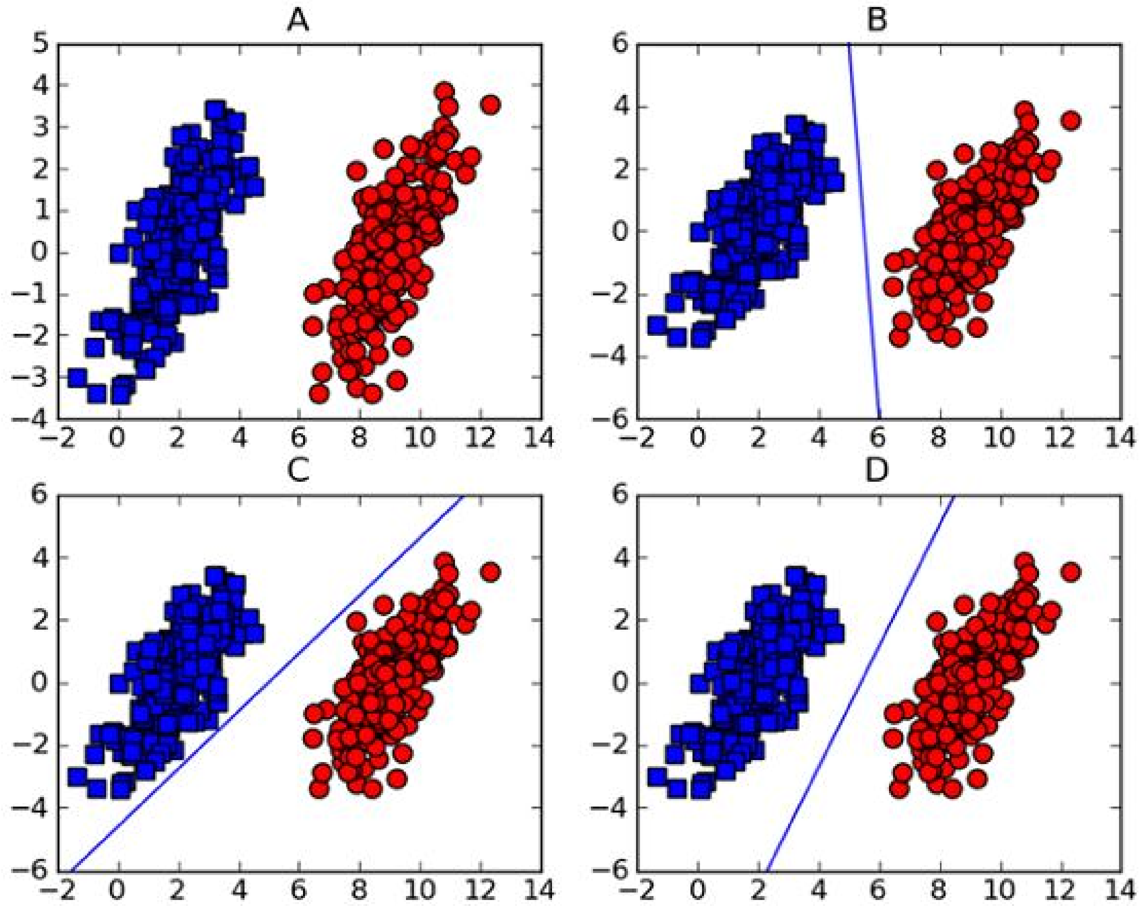


Figure 6.2 Linearly separable data is shown in frame A. Frames B, C, and D show possible valid lines separating the two classes of data.

Margin

- The distance between the hyperplane and the point closest to it
  - These points are called support vectors
  - Try to find the maximum margin as possible

Point Distance

- Separating hyperplane

$$\mathbf{w}^T\mathbf{x}+b$$

- Distance from point to hyperplane is measured by normal

$$|\mathbf{w}^T\mathbf{x}+b| / ||\mathbf{w}||$$

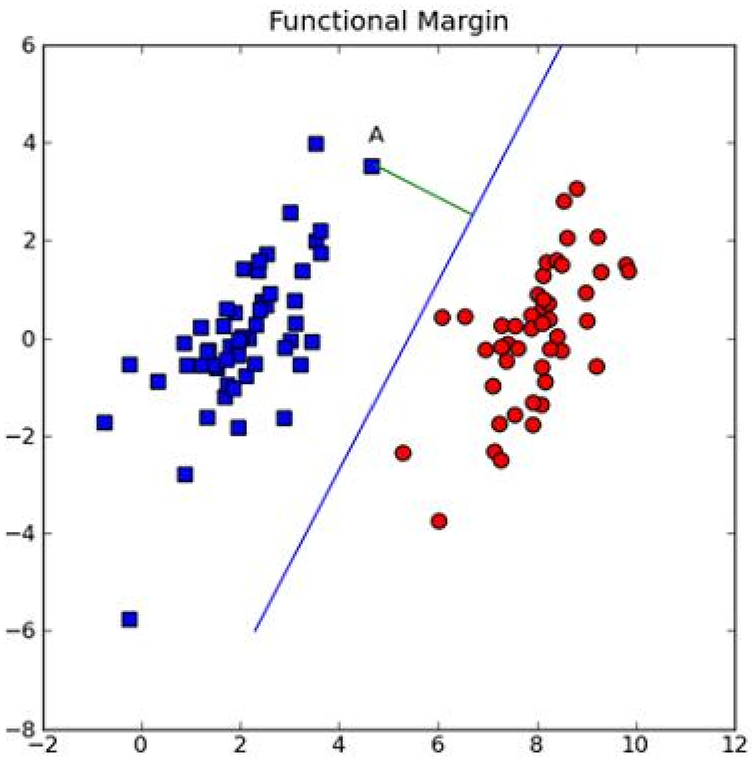


Figure 6.3 The distance from point A to the separating plane is measured by a line normal to the separating plane.

Separation as Optimization

- Use something like Heaviside step function but gives -1 and 1 rather than 0, 1
  - $f(u) = -1$ if $u<0$, 1 otherwise
  - Apply $f(w^Tx+b)$, so -1 and 1 represents each side of the hyperplane
- Margin is calculated by label*(w^Tx+b)
  - Label is the class label, -1 or 1
- Goal: find w and b
  - Optimization problem!

Separation as Optimization

- Find the points with the smallest margin
  - Support vectors
- Then, maximize the margin

$$\underset{w,b}{arg\max}\left\{\min_{n}\left(label\cdot(\boldsymbol{w}^{T}\boldsymbol{x}+b)\right)\cdot\frac{1}{\|\boldsymbol{w}\|}\right\}$$

Solving Tips

- Hold(set) label*(w^Tx+b) to be 1 for the support vectors, then maximize $||w||^{-1}$
- This is a constrained optimization problem
  - Constraints are data points
  - Find the best values w, b
- Using Lagrange multipliers to solve

$$\arg\max_{w,b}\left\{\min_{n}\left(label\cdot(\boldsymbol{w}^T\boldsymbol{x}+b)\right)\cdot\frac{1}{\|\boldsymbol{w}\|}\right\}$$

$$\max_{\alpha}\left[\sum_{i=1}^{m}\alpha-\frac{1}{2}\sum_{i,j=1}^{m} label^{(i)}\cdot label^{(j)}\cdot a_i\cdot a_j\langle x^{(i)},x^{(j)}\rangle\right]$$

$$\alpha\geq 0, and \sum_{i-1}^{m}\alpha_i\cdot label^{(i)}=0$$

Slack Variable

- Constant C controls weighting between our goal of making the margin large

$$c \geq \alpha \geq 0, and \sum_{i-1}^{m} \alpha_i \cdot label^{(i)} = 0$$

[1] Christopher M. Bishop, *Pattern Recognition and Machine Learning* (Springer, 2006).
[2] Bernhard Schlkopf and Alexander J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond* (MIT Press, 2001).

General Approach

General approach to SVMs

1. Collect: Any method.
2. Prepare: Numeric values are needed.
3. Analyze: It helps to visualize the separating hyperplane.
4. Train: The majority of the time will be spent here. Two parameters can be adjusted during this phase.
5. Test: Very simple calculation.
6. Use: You can use an SVM in almost any classification problem. One thing to note is that SVMs are binary classifiers. You'll need to write a little more code to use an SVM on a problem with more than two classes.

Platt's SMO

- SMO: Sequential Minimal Optimization
- John Platt, 1996
- Breaks large optimization problem into many small ones
  - Works to find a set of α and b
- SMO flow
  - Chooses two α to optimize on each cycle
  - One α is increased and one is decreased

SMO Helper Function

Listing 6.1 Helper functions for the SMO algorithm

```
def loadDataSet(fileName):
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr = line.strip().split('\t')
        dataMat.append([float(lineArr[0]), float(lineArr[1])])
        labelMat.append(float(lineArr[2]))
    return dataMat,labelMat

def selectJrand(i,m):
    j=i
    while (j==i):
        j = int(random.uniform(0,m))
    return j

def clipAlpha(aj,H,L):
    if aj > H:
        aj = H
    if L > aj:
        aj = L
    return aj
```

SMO Pseudocode

Create an alphas vector filled with 0s

While the number of iterations is less than MaxIterations:

    For every data vector in the dataset:

        If the data vector can be optimized:

            Select another data vector at random

            Optimize the two vectors together

            If the vectors can't be optimized → break

    If no vectors were optimized → increment the iteration count

Platt's SMO code

Listing 6.2 The simplified SMO algorithm

```
def smoSimple(dataMatIn, classLabels, C, toler, maxIter):
    dataMatrix = mat(dataMatIn); labelMat = mat(classLabels).transpose()
    b = 0; m,n = shape(dataMatrix)
    alphas = mat(zeros((m,1)))
    iter = 0
    while (iter < maxIter):
        alphaPairsChanged = 0
        for i in range(m):
            fXi = float(multiply(alphas,labelMat).T*\
                        (dataMatrix*dataMatrix[i,:].T)) + b
            Ei = fXi - float(labelMat[i])
            if ((labelMat[i]*Ei < -toler) and (alphas[i] < C)) or \
                ((labelMat[i]*Ei > toler) and \
                (alphas[i] > 0)):
                j = selectJrand(i,m)
                fXj = float(multiply(alphas,labelMat).T*\
                            (dataMatrix*dataMatrix[j,:].T)) + b
                Ej = fXj - float(labelMat[j])
                alphaIold = alphas[i].copy();
 alphaJold = alphas[j].copy();
                if (labelMat[i] != labelMat[j]):
                    L = max(0, alphas[j] - alphas[i])
                    H = min(C, C + alphas[j] - alphas[i])
                else:
                    L = max(0, alphas[j] + alphas[i] - C)
                    H = min(C, alphas[j] + alphas[i])
```

1 Enter optimization if alphas can be changed

2 Randomly select second alpha

3 Guarantee alphas stay between 0 and C

Platt's SMO code

```
            if L==H: print "L==H"; continue
            eta = 2.0 * dataMatrix[i,:]*dataMatrix[j,:].T - \
                  dataMatrix[i,:]*dataMatrix[i,:].T - \
                  dataMatrix[j,:]*dataMatrix[j,:].T
            if eta >= 0: print "eta>=0"; continue
            alphas[j] -= labelMat[j]*(Ei - Ej)/eta
            alphas[j] = clipAlpha(alphas[j],H,L)
            if (abs(alphas[j] - alphaJold) < 0.00001): print \
                "j not moving enough"; continue
            alphas[i] += labelMat[j]*labelMat[i]*\
                  (alphaJold - alphas[j])
            b1 = b - Ei- labelMat[i]*(alphas[i]-alphaIold)*\
                 dataMatrix[i,:]*dataMatrix[i,:].T - \
                 labelMat[j]*(alphas[j]-alphaJold)*\
                 dataMatrix[i,:]*dataMatrix[j,:].T
            b2 = b - Ej- labelMat[i]*(alphas[i]-alphaIold)*\
                 dataMatrix[i,:]*dataMatrix[j,:].T - \
                 labelMat[j]*(alphas[j]-alphaJold)*\
                 dataMatrix[j,:]*dataMatrix[j,:].T
            if (0 < alphas[i]) and (C > alphas[i]): b = b1
            elif (0 < alphas[j]) and (C > alphas[j]): b = b2
            else: b = (b1 + b2)/2.0
            alphaPairsChanged += 1
            print "iter: %d i:%d, pairs changed %d" % \
                                  (iter,i,alphaPairsChanged)
        if (alphaPairsChanged == 0): iter += 1
        else: iter = 0
        print "iteration number: %d" % iter
    return b,alphas
```

4 Update i by same amount as j in opposite direction

5 Set the constant term

Support Vectors

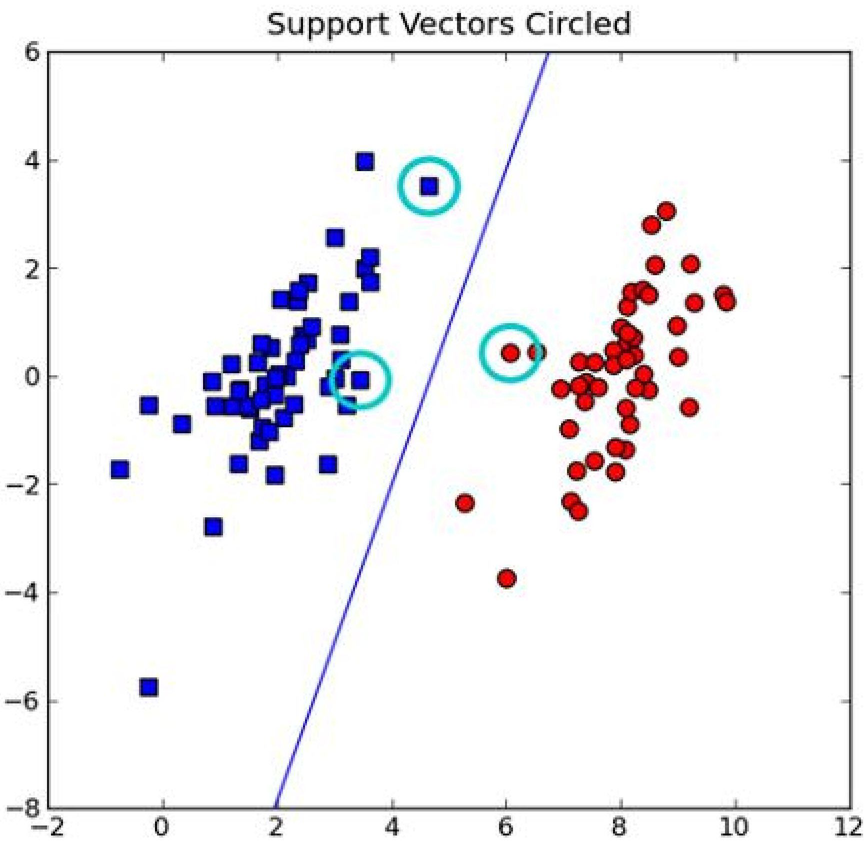


Figure 6.4 SMO sample dataset showing the support vectors circled and the separating hyperplane after the simplified SMO is run on the data

Full Platt's SMO: Speed Up

- Simplified SMO works OK on small datasets
- The only difference is how to select α
  - Use some heuristics

真理大學
Aletheia University

Support Functions

Listing 6.3 Support functions for full Platt SMO

```
class optStruct:
    def __init__(self,dataMatIn, classLabels, C, toler):
        self.X = dataMatIn
        self.labelMat = classLabels
        self.C = C
        self.tol = toler
        self.m = shape(dataMatIn)[0]
        self.alphas = mat(zeros((self.m,1)))
        self.b = 0
        self.eCache = mat(zeros((self.m,2)))

def calcEk(oS, k):
    fXk = float(multiply(oS.alphas,oS.labelMat).T*\
          (oS.X*oS.X[k,:].T)) + oS.b
    Ek = fXk - float(oS.labelMat[k])
    return Ek

def selectJ(i, oS, Ei):
    maxK = -1; maxDeltaE = 0; Ej = 0
    oS.eCache[i] = [1,Ei]
    validEcacheList = nonzero(oS.eCache[:,0].A)[0]
    if (len(validEcacheList)) > 1:
        for k in validEcacheList:
            if k == i: continue
            Ek = calcEk(oS, k)
            deltaE = abs(Ei - Ek)
            if (deltaE > maxDeltaE):
                maxK = k; maxDeltaE = deltaE; Ej = Ek
        return maxK, Ej
    else:
        j = selectJrand(i, oS.m)
        Ej = calcEk(oS, j)
    return j, Ej

def updateEk(oS, k):
    Ek = calcEk(oS, k)
    oS.eCache[k] = [1,Ek]
```

1 Error cache

2 Inner-loop heuristic

3 Choose j for maximum step size

Inner Flow

Listing 6.4 Full Platt SMO optimization routine

```
def innerL(i, oS):
    Ei = calcEk(oS, i)
    if ((oS.labelMat[i]*Ei < -oS.tol) and (oS.alphas[i] < oS.C)) or\
       ((oS.labelMat[i]*Ei > oS.tol) and (oS.alphas[i] > 0)):
        j,Ej = selectJ(i, oS, Ei)
        alphaIold = oS.alphas[i].copy(); alphaJold = oS.alphas[j].copy();
        if (oS.labelMat[i] != oS.labelMat[j]):
            L = max(0, oS.alphas[j] - oS.alphas[i])
            H = min(oS.C, oS.C + oS.alphas[j] - oS.alphas[i])
        else:
            L = max(0, oS.alphas[j] + oS.alphas[i] - oS.C)
            H = min(oS.C, oS.alphas[j] + oS.alphas[i])
        if L==H: print "L==H"; return 0
        eta = 2.0 * oS.X[i,:]*oS.X[j,:].T - oS.X[i,:]*oS.X[i,:].T - \
              oS.X[j,:]*oS.X[j,:].T
        if eta >= 0: print "eta>=0"; return 0
        oS.alphas[j] -= oS.labelMat[j]*(Ei - Ej)/eta
        oS.alphas[j] = clipAlpha(oS.alphas[j],H,L)
        updateEk(oS, j)
        if (abs(oS.alphas[j] - alphaJold) < 0.00001):
            print "j not moving enough"; return 0
        oS.alphas[i] += oS.labelMat[j]*oS.labelMat[i]*\
                        (alphaJold - oS.alphas[j])
        updateEk(oS, i)
        b1 = oS.b - Ei- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*\
             oS.X[i,:]*oS.X[i,:].T - oS.labelMat[j]*\
             (oS.alphas[j]-alphaJold)*oS.X[i,:]*oS.X[j,:].T
        b2 = oS.b - Ej- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*\
             oS.X[i,:]*oS.X[j,:].T - oS.labelMat[j]*\
             (oS.alphas[j]-alphaJold)*oS.X[j,:]*oS.X[j,:].T
        if (0 < oS.alphas[i]) and (oS.C > oS.alphas[i]): oS.b = b1
        elif (0 < oS.alphas[j]) and (oS.C > oS.alphas[j]): oS.b = b2
        else: oS.b = (b1 + b2)/2.0
        return 1
    else: return 0
```

1 Second-choice heuristic

2 Updates Ecache

2 Updates Ecache

Outer Loop

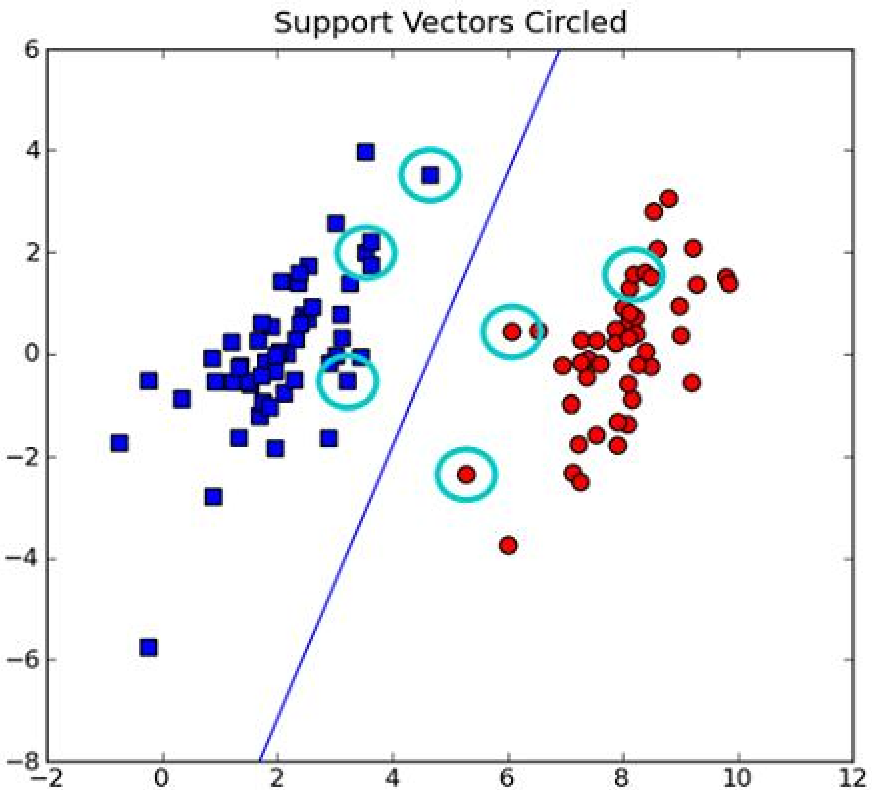
Listing 6.5 Full Platt SMO outer loop

```
def smoP(dataMatIn, classLabels, C, toler, maxIter, kTup=('lin', 0)):
    oS = optStruct(mat(dataMatIn),mat(classLabels).transpose(),C,toler)
    iter = 0
    entireSet = True; alphaPairsChanged = 0
    while (iter < maxIter) and ((alphaPairsChanged > 0) or (entireSet)):
        alphaPairsChanged = 0
        if entireSet:
            for i in range(oS.m):
                alphaPairsChanged += innerL(i,oS)
            print "fullSet, iter: %d i:%d, pairs changed %d" %\
    (iter,i,alphaPairsChanged)
            iter += 1
        else:
            nonBoundIs = nonzero((oS.alphas.A > 0) * (oS.alphas.A < C))[0]
            for i in nonBoundIs:
                alphaPairsChanged += innerL(i,oS)
                print "non-bound, iter: %d i:%d, pairs changed %d" % \
                (iter,i,alphaPairsChanged)
            iter += 1
        if entireSet: entireSet = False
        elif (alphaPairsChanged == 0): entireSet = True
        print "iteration number: %d" % iter
    return oS.b,oS.alphas
```

1 Go over all values

2 Go over non-bound values

Full Platt's Result



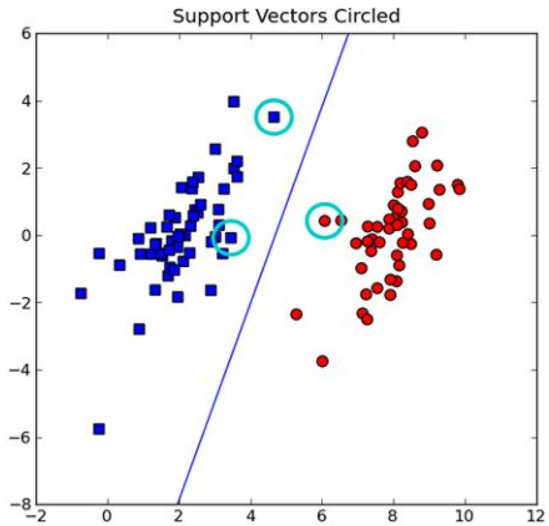


Figure 6.5 Support vectors shown after the full SMO algorithm is run on the dataset. The results are slightly different from those in figure 6.4.

Classification: Hyperplane from α

```
def calcWs(alphas,dataArr,classLabels):
    X = mat(dataArr); labelMat = mat(classLabels).transpose()
    m,n = shape(X)
    w = zeros((n,1))
    for i in range(m):
        w += multiply(alphas[i]*labelMat[i],X[i,:].T)
    return w
```

```
>>> ws=svmMLiA.calcWs(alphas,dataArr,labelArr)
>>> ws
array([[ 0.65307162],
       [-0.17196128]])
```

Now to classify something, say the first data point, type in this:

```
>>> datMat=mat(dataArr)
>>> datMat[0]*mat(ws)+b
matrix([[-0.92555695]])
```

Complex Data

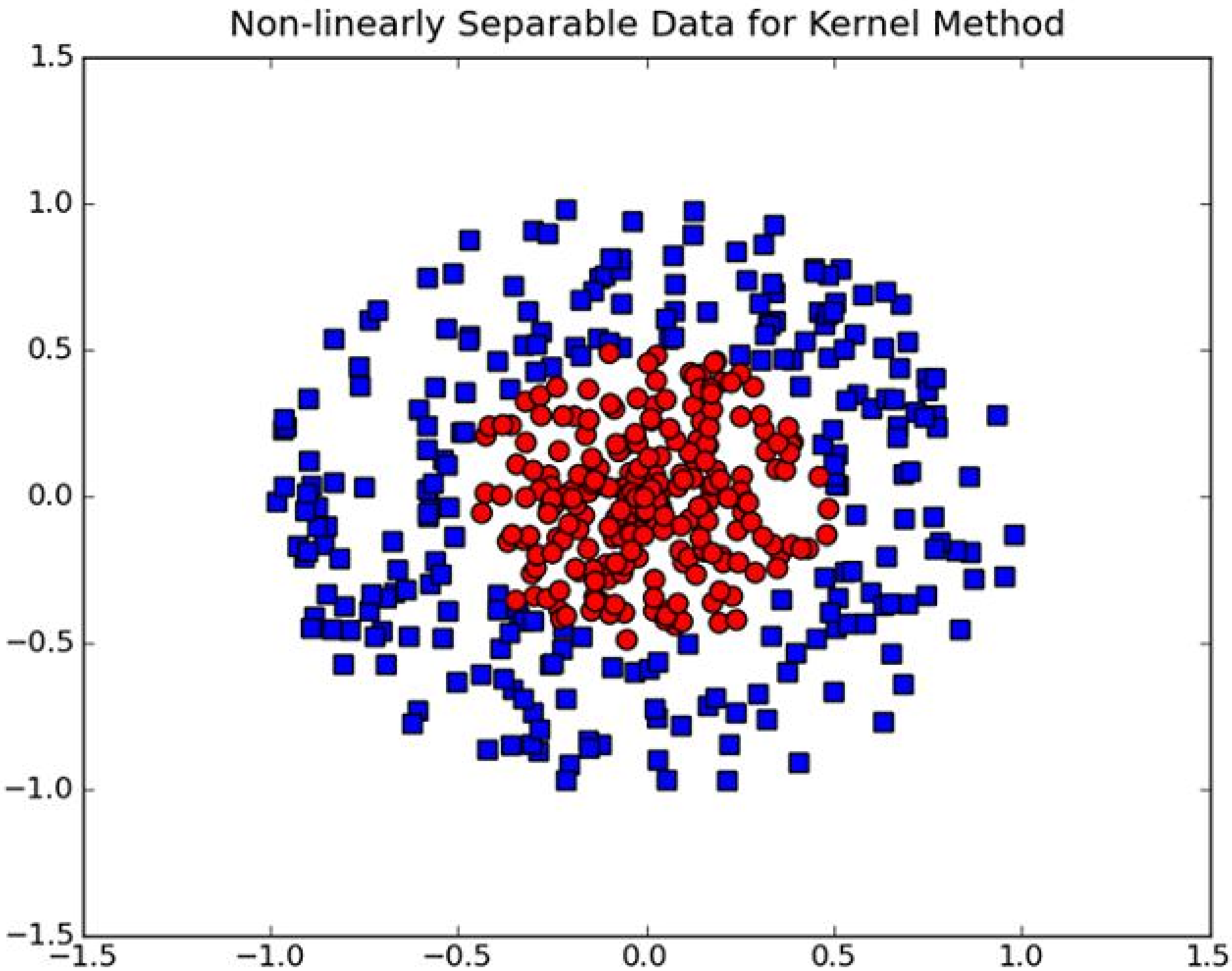


Figure 6.6 This data can't be easily separated with a straight line in two dimensions, but it's obvious that some pattern exists separating the squares and the circles.

Complex Data: Using Kernels

- Deal with data that are not linear separable
- Solution: use a function called kernel function to transform
  - Mapping from one feature space to another
  - Usually from lower-dimension to higher-dimension
- Kernels aren't unique to SVMs
- RBF: radial basis function, a popular kernel

Feature of RBF

- RBF takes a vector and outputs a scalar based on the vector's distance
- Gaussian version RBF

$$k(x,y) = exp\left(\frac{-\|x-y\|^2}{2\sigma^2}\right)$$

  - σ : define how quickly this falls off to 0

Kernel Transform

Listing 6.6 Kernel transformation function

```
def kernelTrans(X, A, kTup):
    m,n = shape(X)
    K = mat(zeros((m,1)))
    if kTup[0]=='lin': K = X * A.T
    elif kTup[0]=='rbf':
        for j in range(m):
            deltaRow = X[j,:] - A
            K[j] = deltaRow*deltaRow.T
        K = exp(K /(-1*kTup[1]**2))
    else: raise NameError('Houston We Have a Problem -- \
    That Kernel is not recognized')
    return K

class optStruct:
    def __init__(self,dataMatIn, classLabels, C, toler, kTup):
        self.X = dataMatIn
        self.labelMat = classLabels
        self.C = C
        self.tol = toler
        self.m = shape(dataMatIn)[0]
        self.alphas = mat(zeros((self.m,1)))
        self.b = 0
        self.eCache = mat(zeros((self.m,2)))
        self.K = mat(zeros((self.m,self.m)))
        for i in range(self.m):
            self.K[:,i] = kernelTrans(self.X, self.X[i,:], kTup)
```

1 Element-wise division

Platt's RBF Version

Listing 6.7 Changes to innerL() and calcEk() needed to user kernels

```
innerL():
                              .
                              .
                              .
eta = 2.0 * oS.K[i,j] - oS.K[i,i] - oS.K[j,j]
                              .
                              .
                              .
b1 = oS.b - Ei- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,i] -\
                oS.labelMat[j]*(oS.alphas[j]-alphaJold)*oS.K[i,j]
b2 = oS.b - Ej- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,j]-\
                oS.labelMat[j]*(oS.alphas[j]-alphaJold)*oS.K[j,j]
                              .
                              .
                              .
def calcEk(oS, k):
    fXk = float(multiply(oS.alphas,oS.labelMat).T*oS.K[:,k] + oS.b)
    Ek = fXk - float(oS.labelMat[k])
    return Ek
```

Test Function

Listing 6.8 Radial bias test function for classifying with a kernel

```
def testRbf(k1=1.3):
    dataArr,labelArr = loadDataSet('testSetRBF.txt')
    b,alphas = smoP(dataArr, labelArr, 200, 0.0001, 10000, ('rbf', k1))
    datMat=mat(dataArr); labelMat = mat(labelArr).transpose()
    svInd=nonzero(alphas.A>0)[0]
    sVs=datMat[svInd]
    labelSV = labelMat[svInd];
    print "there are %d Support Vectors" % shape(sVs)[0]
    m,n = shape(datMat)
    errorCount = 0
    for i in range(m):
        kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', k1))
        predict=kernelEval.T * multiply(labelSV,alphas[svInd]) + b
        if sign(predict)!=sign(labelArr[i]): errorCount += 1
    print "the training error rate is: %f" % (float(errorCount)/m)
    dataArr,labelArr = loadDataSet('testSetRBF2.txt')
    errorCount = 0
    datMat=mat(dataArr); labelMat = mat(labelArr).transpose()
    m,n = shape(datMat)
    for i in range(m):
        kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', k1))
        predict=kernelEval.T * multiply(labelSV,alphas[svInd]) + b
        if sign(predict)!=sign(labelArr[i]): errorCount += 1
    print "the test error rate is: %f" % (float(errorCount)/m)
```

Create matrix of support vectors

RBF Examples

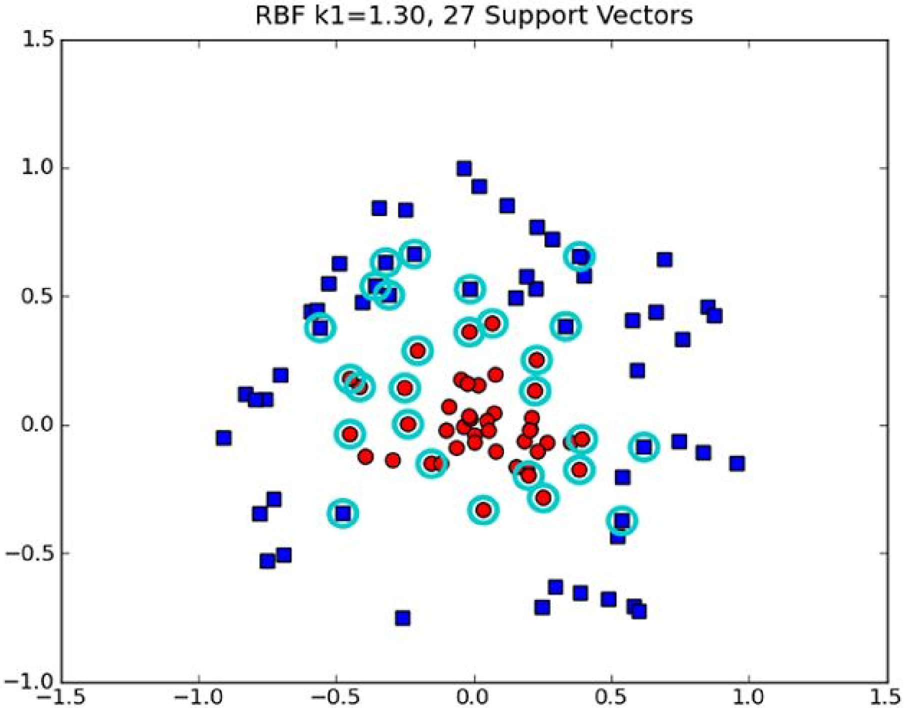


Figure 6.8 Radial bias kernel function with user parameter k1=1.3. Here we have fewer support vectors than in figure 6.7. The support vectors are bunching up around the decision boundary.

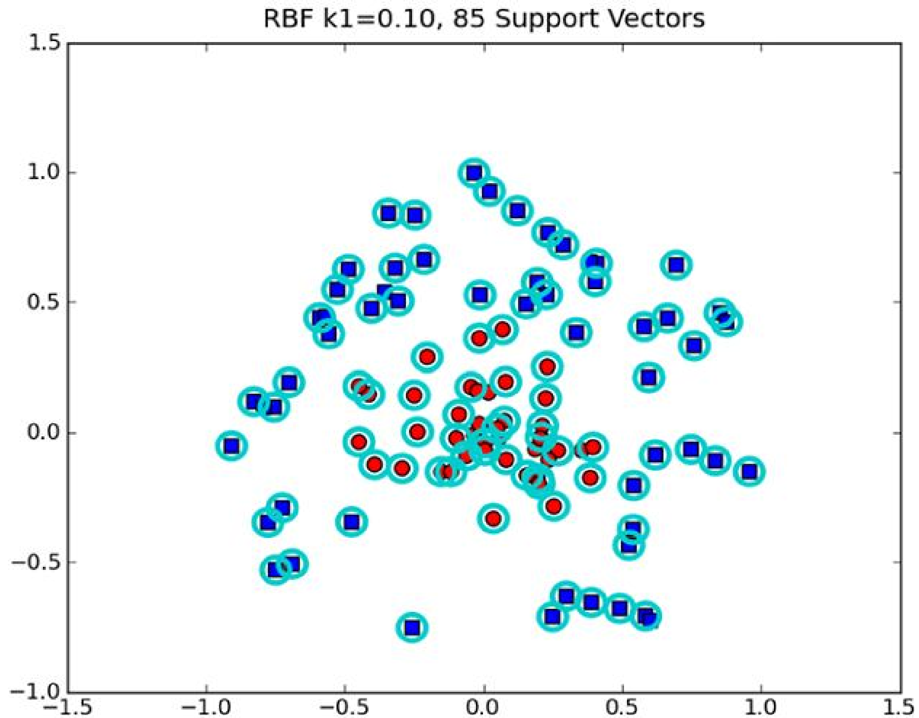


Figure 6.7 Radial bias function with the user-defined parameter k1=0.1. The user-defined parameter reduces the influence of each support vector, so you need more support vectors.

Summary

- SVM is a binary classification machine
- Support vectors have good generalization error
- Try to maximize margin by solving a quadratic optimization problem
  – John Platt speed up this
- Kernel methods (tricks) are helpful in non-linear separable problems
  – Usually from lower-dimension to higher-dimension
- RBF is a popular kernel that measures the distance between two vectors