From Perceptron to SVM

Dr. Fayyaz Minhas

Department of Computer Science
University of Warwick
https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs909/
Classification

• Given
  – A set of labelled training examples

• Find
  – A mathematical function that **generalizes** well to unseen cases
    • Discriminant function

\[ f(\mathbf{x}) = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b = \mathbf{w}^T\mathbf{x} + b \]
REO For Perceptron

• Representation
  – Features
  – Discriminant
    • Linear
      \[ f(x) = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b = w^T x + b \]

• Evaluation
  – 0/1 (Step) Loss
    \[ l(f(x), y) = \begin{cases} 0 & yf(x) > 0 \\ 1 & yf(x) \leq 0 \end{cases} \]
  – Hinge Loss
    \[ l(f(x), y) = \begin{cases} 0 & yf(x) > 1 \\ 1 - yf(x) & yf(x) \leq 1 \end{cases} \]

• Optimization
  – Using Gradient Descent
    \[ \nabla L = \sum_{i=1}^{N} \nabla w l(f(x_i; w), y_i) \]
    \[ w^{(k)} \leftarrow w^{(k-1)} - \alpha \nabla L(w^{(k-1)}) \]
import numpy as np
import matplotlib.pyplot as plt
import itertools

class Perceptron:
    def __init__(self, alpha = 0.1, epochs = 200):
        self.alpha = alpha
        self.epochs = epochs
        self.W = np.array([0])
        self.bias = np.random.randn()
        self.Lambda = 0.5
    def fit(self, Xtr, Ytr):
        d = Xtr.shape[1]
        self.W = np.random.randn(d)
        for e in range(self.epochs):
            finished = True
            for i, x in enumerate(Xtr):
                if Ytr[i] != self.predict(np.atleast_2d(x)):
                    finished = False
                    self.W += self.alpha*Ytr[i]*x
                    self.bias += self.alpha*Ytr[i]
            if finished: break
    def score(self, x):
        return np.dot(x, self.W) + self.bias
    def predict(self, x):
        return np.sign(self.score(x))

if __name__ == '__main__':
    from plotit import plotit
    Xtr = np.array([[-1,0],[0,1],[4,4],[2,3]])
    ytr = np.array([-1,-1,1,1])
    clf = Perceptron()
    clf.fit(Xtr, ytr)
    z = clf.score(Xtr)
    print("Prediction Scores:", z)
    y = clf.predict(Xtr)
    print("Prediction Labels:", y)
    plotit(Xtr, ytr, clf=clf.score, contours=[0],
           extent=[-5,5,-5,5])

from sklearn.linear_model import Perceptron
clf = Perceptron()
clf.fit(X, y)
clf.predict(X)

Empirical Risk Minimization

• So far, our machine learning models look like the following (empirical error) minimization:

\[ f^* = \arg\min_f L(X_{\text{train}}, Y_{\text{train}}; f) \]

\[ w^* = \arg\min_w L(X_{\text{train}}, Y_{\text{train}}; f) \]

• This is called ERM:
  – Learning only from training data
Issues with empirical risk parameters

• There are a large number of lines (or in general ‘hyperplanes’) separating the two classes

\[ f(x) = w^T x + b = 0 \]

Which separator is the best?
Limitations of Empirical Risk Minimization

The boundary which lies closer to data points has low margin for error: A small change in the input can change the prediction label.

The boundary which lies at the maximum distance from data points of both classes gives better tolerance to noise and better generalization.
Margin of a linear classifier

• The width by which the boundary of a linear classifier can be increased before hitting a data point is called the margin of the linear classifier.

Linear Classifiers with larger margins are better.
Margin and Regularization

• Large Margin
• Classifiers with large margin have a property
  – Small changes in \( x \) should cause small changes in output: Regularization
• How can we achieve regularization?

\[
f(x) = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b = w^T x + b
\]
Structural Risk Minimization

• In order to produce better generalization, we need to do both empirical error minimization but also reduce “Structural Risk”

• Formally, minimizing “structural risk” puts an upper bound on your generalization error
  – Structural risk control, in essence, controls the structure of your prediction model in addition to empirical error minimization
Support Vector Machines (SVM)

• Support Vector Machines are linear classifiers that produce the optimal separating boundary (hyper-plane)
  – Find \( \mathbf{w} \) and \( b \) in a way so as to:
    • Minimize misclassification error over training data (Empirical Risk Minimization)
    • Maximize the margin
      – Or equivalently, maximize regularization
      – Or equivalently, minimize the individual absolute weights

\[
f(x) = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b = \mathbf{w}^T \mathbf{x} + b
\]
Understanding Regularization

- If $\mathbf{w}$ is too large (positive or negative)
  - Then a small change in $\mathbf{x}$ (e.g., due to noise) will cause a large change in the output $\mathbf{w}^T \mathbf{x} + b$
  - Can lead to errors
  - Controlling for this is called “Regularization”

- Achieved by minimizing:
  \[ R(f) = w_1^2 + w_2^2 + \cdots + w_d^2 = \mathbf{w}^T \mathbf{w} = \|\mathbf{w}\|^2 \]

However, this can cause a problem!
SRM to SVM

• Representation

\[ f(x) = w_1 x^{(1)} + w_2 x^{(2)} + \ldots + w_d x^{(d)} + b = \mathbf{w}^T \mathbf{x} + b \]

• Evaluation & Optimization

\[
\min_{\mathbf{w}} R(f) + CL(\mathbf{X}, \mathbf{Y}; \mathbf{w})
\]

- (Inverse of) Margin
  - AKA Regularization
  - \( C > 0 \) is a weighting factor that controls the relative contribution of both

\[
R(f) = \frac{1}{2} (w_1^2 + w_2^2 + \ldots + w_d^2) = \frac{1}{2} \mathbf{w}^T \mathbf{w} = \frac{1}{2} \| \mathbf{w} \|^2
\]

\[
L(\mathbf{X}, \mathbf{Y}; \mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \max\{0,1 - y_i f(x_i; \mathbf{w})\}
\]

\[
\min_{\mathbf{w}, b} \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{N} \sum_{i=1}^{N} \max\{0,1 - y_i f(x_i; \mathbf{w})\}
\]
SVM Optimization

\[
\min_{w,b} \frac{1}{2} w^T w + \frac{C}{N} \sum_{i=1}^{N} \max\{0,1- y_i f(x_i)\}
\]

\[
\min_w P(w) = \frac{\lambda}{2} w^T w + \frac{1}{N} \sum_{i=1}^{N} \max\{0,1- y_i f(x_i;w)\}
\]

\[
w_k \leftarrow w_{k-1} - \alpha \nabla P(w_{k-1})
\]

\[
\nabla P = \lambda w - \frac{\partial}{\partial w} \max\{0,1- y(w^T x)\}
\]

\[
\nabla P = \lambda w - 1(yf(x) < 1)(-yx) = \lambda w + 1(yf(x) < 1)(yx)
\]

\[
w_k \leftarrow w_{k-1} - \alpha \lambda w_{k-1} - \alpha 1(yf(x) < 1)(yx)
\]
Support Vector Machines

- Support Vector Machines, in their basic form, are linear classifiers that give maximum margin

- Principles of Operation
  - Minimize the number of training errors
    - Achieved by minimizing hinge loss
  - Maximize margin
    - Allows noise tolerance
    - Allows Regularization
  - Perform Nonlinear Classification
    - Achieved through feature transformations/kernels

- The points that determine the margin are called Support Vectors
Wanna Play?

• Use the Java Applet at:

• [https://www.csie.ntu.edu.tw/~cjlin/libsvm/](https://www.csie.ntu.edu.tw/~cjlin/libsvm/)

• Set “-t 0 -c 100”
SVMs up till now

\[
\min_{\mathbf{w}, b} \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{N} \sum_{i=1}^{N} \max\{0, 1 - y_i f(x_i)\}
\]

- **Vapnik and Chervonenkis:**
  - Theoretical foundations for SVMs
  - Structural Risk Minimization
- **Corinna Cortes**
  - Soft SVM (1995)
- **Bernard Scholkopf (1997)**
  - Representer Theorem
  - Complete Kernel trick!
  - Kernels not only allow nonlinear boundaries but also allow representation of non-vectoral data

http://www.svms.org/history.html
import numpy as np
import matplotlib.pyplot as plt
import itertools

class RegularizedPerceptron:
    def __init__(self, Lambda=0.0, margin=0.0, alpha=0.1, epochs=1000):
        self.alpha = alpha
        self.epochs = epochs
        self.W = np.array([0])
        self.bias = np.random.randn()
        self.Lambda = Lambda #not used in perceptron
        self.Margin = margin #0.0 in Perceptron
        def fit(self, Xtr, Ytr):
            d = Xtr.shape[1]
            self.W = np.random.randn(d)
            for e in range(self.epochs):
                finished = True
                for i, x in enumerate(Xtr):
                    if self.score(np.atleast_2d(x)) * Ytr[i] < self.Margin:
                        self.W += self.alpha * Ytr[i] * x
                        self.bias += self.alpha * Ytr[i]
        def score(self, x):
            return np.dot(x, self.W) + self.bias
        def predict(self, x):
            return np.sign(self.score(x))

if __name__ == '__main__':
    from plotit import plotit
    Xtr = np.array([[-1, 0], [0, 1], [4, 4], [2, 3]])
    ytr = np.array([-1, -1, 1, 1])
    clf = RegularizedPerceptron(Lambda=0.1, margin=1.0)
    clf.fit(Xtr, ytr)
    z = clf.score(Xtr)
    print("Prediction Scores:", z)
    y = clf.predict(Xtr)
    print("Prediction Labels:", y)
    plotit(Xtr, ytr, clf=clf.score, conts=0, extent=[-5, 5, -5, 5])

https://github.com/foxtrotmike/CS909/blob/master/regper.ipynb
Difference between Perceptron and SVM
import numpy as np
from sklearn.svm import LinearSVC as Classifier

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([-1,1,1,1])
clf = Classifier(class_weight='balanced',C=100)
clf.fit(X, y)
f = clf.predict(X)
print('Coefficients before adding additional feature:', clf.coef_, clf.intercept_)
print('Predictions before adding additional feature:', f)

plotit(X,y,clf=clf.decision_function,conts=[0],extent=[-2,+2,-2,+2])

C=1000
w=[2,2],b=-1

C=1
w = [0.84 0.84],b=-0.465
Finding Margin of a Linear Classifier

- Consider a linear classifier with the boundary
  \[ f(x) = w^T x + b = 0 \quad \text{for all } x \text{ on the boundary} \]

- We know that the vector \( w \) is perpendicular to the boundary
  - Consider two points \( x^{(1)} \) and \( x^{(2)} \) on the boundary
    - Subtracting (1) from (2)
      \[ w^T (x^{(2)} - x^{(1)}) = 0 \quad \Rightarrow \quad w \perp (x^{(2)} - x^{(1)}) \]
Distance of a line from a point

• Consider the line
  \[ x_1 + 2x_2 + 3 = 0 \]
• The distance of (4,2) is
  \[ r = 4.92 \]
Finding Margin of a Linear Classifier

- Let $x^{(s)}$ be a point in the feature space with its projection $x^{(p)}$ on the boundary.
  \[ f\left(x^{(p)}\right) = w^T x^{(p)} + b = 0 \]

- We know that:
  \[ x^{(s)} = x^{(p)} + r\hat{w} \]
  \[ \Rightarrow f\left(x^{(s)}\right) = w^T x^{(s)} + b \]
  \[ = w^T \left(x^{(p)} + r\hat{w}\right) + b \]
  \[ = w^T x^{(p)} + rw^T \hat{w} + b \]
  \[ = w^T x^{(p)} + b + rw^T \frac{w}{\|w\|} \]
  \[ = 0 + r\|w\| = r\|w\| \]

\[ \Rightarrow r = \frac{f\left(x^{(s)}\right)}{\|w\|} \]
Finding Margin of a Linear Classifier

- Let $x^{(s)}$ be a point in the feature space with its projection $x^{(p)}$ on the boundary
  
  $$f(x^{(p)}) = w^T x^{(p)} + b = 0$$

- We know that,
  
  $$x^{(s)} = x^{(p)} + r\hat{w}$$

  $$\Rightarrow f(x^{(s)}) = w^T x^{(s)} + b$$

  $$= w^T (x^{(p)} + r\hat{w}) + b$$

  $$= w^T x^{(p)} + r w^T \hat{w} + b$$

  $$= w^T x^{(p)} + b + r w^T \frac{w}{||w||}$$

  $$= 0 + r ||w|| = r ||w||$$

  $$\Rightarrow r = \frac{f(x^{(s)})}{||w||}$$

  $$r' = \frac{1}{||w||}$$

Since, based on the hinge loss constraint, we have $f(x^{(s)}) \geq 1$, so:
An alternate view of SVMs

• As an unconstrained optimization problem

\[
\min_w \frac{1}{2} w^T w + \frac{C}{N} \sum_{i=1}^{N} \max\{0, 1 - y_i f(x_i; w)\}
\]

• As a constrained optimization problem

\[
\min_w \frac{1}{2} w^T w + \frac{C}{N} \sum_{i=1}^{N} \xi_i
\]

Such that, for all \(i = 1 \ldots N\),

\[
y_i f(x_i; w) \geq 1 - \xi_i
\]

\[
\xi_i \geq 0
\]

Training examples should be correctly classified with minimum “slack”

Can be solved as a Quadratic Constraint Optimization Problem
Nonlinear Separation through Transformation

- Given a classification problem with a nonlinear boundary, we can, at times, find a mapping or transformation of the feature space which makes the classification problem linear separable in the transformed space.
Examples: Transformation

\[ f(x; \theta) = w_1 x^{(1)} + w_2 x^{(2)} + b = 0 \]

(0,0): \( b < 0 \)
(0,1): \( w_2 + b > 0 \)
(1,0): \( w_1 + b > 0 \)
(1,1): \( w_1 + w_2 + b < 0 \)

\( w_1 = 2, w_2 = 2, w_3 = -3, b = -1 \)

\[ \phi \left( \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} \right) = \begin{bmatrix} \sqrt{2} x^{(1)} \\ x^{(2)} \end{bmatrix} \]

<table>
<thead>
<tr>
<th>( x^{(1)} )</th>
<th>( x^{(2)} )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>+1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>+1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x^{(1)} )</th>
<th>( x^{(2)} )</th>
<th>( x^{(3)} )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>+1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>+1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>( \sqrt{2} )</td>
<td>-1</td>
</tr>
</tbody>
</table>
XOR: Linear Separability

![Linear Discrimination Boundary for XOR using Higher Subspace Projection](image)
import numpy as np
from sklearn.svm import LinearSVC as Classifier
from plotit import *

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([-1,1,1,-1])
clf2d = Classifier(C=1000).fit(X, y)
f = clf2d.predict(X)
print('Coefficients before adding additional feature:', clf2d.coef_,clf2d.intercept_)
print('Predictions before adding additional feature:', f)

transform = lambda x: np.hstack((x,np.atleast_2d(np.sqrt(2)*x[:,0]*x[:,1]).T))

Xt = transform(X)
print(Xt)
clf = Classifier(C=10).fit(Xt, y)
f = clf.predict(Xt)
print('Coefficients after adding additional feature:', clf.coef_, clf.intercept_)
print('Predictions after adding additional feature:', f)

# showing the plane in 3d
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

xx = np.arange(-1, 2, 0.01)
yy = np.arange(-1, 2, 0.01)
xx, yy = np.meshgrid(xx, yy)
zz = -(clf.coef_[0,0]*xx+clf.coef_[0,1]*yy+clf.intercept_)/clf.coef_[0,2]
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
pidx, nidx = y==1, y!=1
ax.scatter(Xt[pidx,0], Xt[pidx,1], Xt[pidx,2], 'y', alpha=1, s=100)
ax.scatter(Xt[nidx,0], Xt[nidx,1], Xt[nidx,2], 'b', alpha=1, s=100)
ax.plot_surface(xx, yy, zz, linewidth=0, antialiased=True)

# showing the boundary in 2d
plt.figure()
plotit(X,y,clf = clf.decision_function,transform = transform,conts=[0],extent=[-2,+2,-2,+2])

https://github.com/foxtrotmike/CS909/blob/master/transformations.ipynb
Examples: Transformation

- Does this mapping do it?

  \[ \phi \left( \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} \right) = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ 1 \end{bmatrix} \]

- What about this one?

  \[ \phi \left( \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} \right) = (x^{(1)} + x^{(2)} - 1)^2 \]
Transformation Examples

- Can you find a transform that makes the following classification problems linear separable? Can you draw the data points in the new transformed feature space?
Feature Transformation ↔ Distance Change

• Feature transformations change the concept of distance or dot product between two points

– Consider $d(a, b) = \|a - b\|^2 = (a - b)^T (a - b) = a^T a + b^T b - 2a^T b$

– However, after transformation $d_\phi(a, b) = \|\phi(a) - \phi(b)\|^2$

– This corresponds to a change of the feature space

\[
d_\phi(a, b) = \|\phi(a) - \phi(b)\|^2 = (\phi(a) - \phi(b))^T (\phi(a) - \phi(b)) \\
= \phi(a)^T \phi(a) + \phi(b)^T \phi(b) - 2\phi(a)^T \phi(b) = k(a, a) + k(b, b) - 2k(a, b)
\]

\[
k(a, b) = a^T b \\
k(a, b) = \phi(a)^T \phi(b)
\]
Feature Transformation ↔ Dot Product Change

– Let’s say, we have 2D data, then
  
  • \( a^T b = a^{(1)} b^{(1)} + a^{(2)} b^{(2)} \)
  
  • \( k(a, b) = (a^T b)^2 = (a^{(1)} b^{(1)} + a^{(2)} b^{(2)})^2 \)
  
  • Let’s use the transformation \( \phi(u) = \begin{bmatrix} u^{(1)} \\ u^{(2)} \\ \sqrt{2}u^{(1)}u^{(2)} \end{bmatrix} \)
  
  \[ \begin{align*}
  \phi(a)^T \phi(b) &= [a^{(1)} \ a^{(2)} \ \sqrt{2}a^{(1)}a^{(2)}] \begin{bmatrix}
  b^{(1)} \\
  b^{(2)} \\
  \sqrt{2}b^{(1)}b^{(2)}
  \end{bmatrix} \\
  &= a^{(1)}b^{(1)} + a^{(2)}b^{(2)} + 2a^{(1)}a^{(2)}b^{(1)}b^{(2)} \\
  &= (a^{(1)}b^{(1)} + a^{(2)}b^{(2)})^2 = (a^T b)^2 = k(a, b)
  \end{align*} \]
So How can we make a non-linear SVM?

• Transform the data and then apply the SVM!
  – Choosing a transformation

• Somehow change the definition of the dot-product so that the implicit transformation allows non-linear classification!
  – Choosing a kernel function

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>$K(x, y) = x \cdot y$</td>
</tr>
<tr>
<td>Sigmoid</td>
<td>$K(x, y) = \tanh(ax \cdot y + b)$</td>
</tr>
<tr>
<td>Polynomial</td>
<td>$K(x, y) = (1 + x \cdot y)^d$</td>
</tr>
<tr>
<td>KMOD</td>
<td>$K(x, y) = a \left[ \exp \left( \frac{-y}{|x-y|^2 + \sigma^2} \right) - 1 \right]$</td>
</tr>
<tr>
<td>RBF</td>
<td>$K(x, y) = \exp(-a |x - y|^2)$</td>
</tr>
<tr>
<td>Exponential RBF</td>
<td>$K(x, y) = \exp(-a |x - y|)$</td>
</tr>
</tbody>
</table>

Any function $k$ can be a kernel if its ‘Gram’ matrix

$$K = \begin{bmatrix}
    k(x_1, x_1) & k(x_1, x_2) & k(x_1, x_3) \\
    k(x_2, x_1) & k(x_2, x_2) & k(x_2, x_3) \\
    k(x_3, x_1) & k(x_3, x_2) & k(x_3, x_3)
\end{bmatrix}$$

is symmetric, positive semi-definite (for all given data).
Kernelized SVM

• We know that the discriminant function of the SVM can be written as:

\[ f(x) = w^T x + b \]

• The Representer theorem (Scholkopf 2001) allows us to represent the SVM weight vector as a linear combination of input vectors with each example’s contribution weighted by a factor \( \alpha_i \)

\[ w = \sum_{i=1}^{N} \alpha_i x_i \]
Kernelized SVM

• Thus, we can write

\[ f(x) = w^T x + b = b + \sum_{j=1}^{N} \alpha_j x_j^T x \]

• Notice, that there is a dot product of the test example with every given input example. Let’s denote the dot product as \( k(u, v) = u^T v \)

• This allows to write: \( f(x) = b + w^T x = b + \sum_{j=1}^{N} \alpha_j k(x_j, x) \)

• Now, \( w^T w = (\sum_{i=1}^{N} \alpha_i x_i)^T \sum_{j=1}^{N} \alpha_j x_j = \sum_{i,j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) \)

• Thus, the SVM can be rewritten as

\[
\begin{align*}
\min_w & \frac{1}{2} w^T w + \frac{C}{N} \sum_{i=1}^{N} \max\{0,1-y_i f(x_i; w)\} \\
\text{s.t.} & \quad w = \sum_{i=1}^{N} \alpha_i x_i \\
\min_{\alpha,b} & \quad \sum_{i=1}^{N} \alpha_i \alpha_j k(x_i, x_j) + \frac{C}{N} \sum_{i=1}^{N} \max\{0,1-y_i (b + \sum_{j=1}^{N} \alpha_j k(x_i, x_j))\}
\end{align*}
\]
Kernelized SVM

• In this formulation

\[
\min_{\alpha, b} \sum_{i,j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) + \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, 1 - y_i \left( b + \sum_{j=1}^{N} \alpha_j k(x_i, x_j) \right) \right\}
\]

• The weight vector is not present
• We know \( k(x_i, x_j) \) for any two given training examples
• All the dot products have been replaced with \( k(x_j, x_i) \)
• The optimization solution will be to obtain the \( \alpha \)

\[
f(x) = b + \sum_{j=1}^{N} \alpha_j k(x_j, x)
\]
Wanna Play?

• Use the Java Applet at:

• [https://www.csie.ntu.edu.tw/~cjlin/libsvm/](https://www.csie.ntu.edu.tw/~cjlin/libsvm/)

• Set "-t 0 -c 100"
import numpy as np
from sklearn.svm import SVC as Classifier

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([-1,1,1,1])
clf = Classifier(kernel = 'poly', degree = 1, C = 10).fit(X, y)
plotit(X,y,clf = clf.decision_function,conts=[0],extent=[-2,+2,-2,+2])
print("Alpha: ",clf.dual_coef_)
print(clf.support_vectors_)

\[ x^i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \]
\[ y = -1 + 1 + 1 + 1 \]
\[ \alpha_i = +4 + 2 + 2 \quad 0 \]
\[ w^* = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \]
\[ b^* = -1 \]

\[ k(a, b) = a^T b \]
import numpy as np
from sklearn.svm import SVC as Classifier

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y =np.array([-1,1,1,-1])
clf = Classifier(kernel = 'poly', degree = 2, C = 1).fit(X, y)
plotit(X,y,clf = clf.decision_function,conts=[0],extent=[-2,2,-2,2])
print("Alpha: ",clf.dual_coef_)
print(clf.support_vectors_)

\[ k(a, b) = (a^T b)^2 \]
Using the SVM

• Read:


• http://pyml.sourceforge.net/doc/howto.pdf
Steps for Feature based Classification

- Prepare the pattern matrix
- Select the kernel function to use
- Select the parameter of the kernel function and the value of $C$
  - You can use the values suggested by the SVM software, or you can set apart a validation set to determine the values of the parameter
- Execute the training algorithm and obtain the $\alpha_i$
- Unseen data can be classified using the $\alpha_i$ and the support vectors
Choosing the Kernel Function

• Probably the trickiest part of using SVM.
• The kernel function is important because it creates the kernel matrix, which summarizes all the data
• In practice, a low degree polynomial kernel or RBF kernel with a reasonable width is a good initial try
Handling data imbalance

- If the data is imbalanced (too much of one class and only a small number of examples from the other)
  - You can set an individual C for each example
  - Can also be used to reflect a priori knowledge

\[
\min_w \frac{1}{2} w^T w + \sum_{i=1}^{N} c_i \max\{0,1-y_i f(x_i;w)\}
\]
Strengths and Weaknesses of SVM

• Strengths
  – Only a few training points determine the final boundary
    • Support Vectors
  – Margin maximization and kernelized
  – Training is relatively easy
    • No local optimal, unlike in neural networks
  – It scales relatively well to high dimensional data
  – Tradeoff between classifier complexity and error can be controlled explicitly (through C)
  – Non-traditional data like strings and trees can be used as input to SVM, instead of feature vectors

• Weaknesses
  – Need to choose a “good” kernel function.
Advantages of kernels

• Once we replace the dot product with a kernel function (i.e., perform the kernel trick or ‘kernelize’ the formulation), the SVM formulation no longer requires any features!

• As long as you have a kernel function, everything works
  – Remember a kernel function is simply a mapping from two examples to a scalar
    • Tells us how similar the two examples are to each other
General Principle

• Each machine learning model should have:
  – Empirical Error Minimization
  – Regularization

• This gives an upper bound on the generalization error (Vapnik, 1974)
What can we do with SRM?

- The principal of SRM allows us to develop a family of large margin learning machines by changing its components

- Example
  - SVM: $\min_{w,b} \frac{\lambda}{2} \|w\|^2 + \sum_{i=1}^{N} [1 - y_i f(x_i)]_+$
  - Regularized least square regression
    - $\min_{w,b} \frac{\lambda}{2} \|w\|^2 + \sum_{i=1}^{N} (y_i - f(x_i))^2$
  - Support Vector Regression
    - $\min_{w,b} \frac{\lambda}{2} \|w\|^2 + \sum_{i=1}^{N} |y_i - f(x_i)| - \epsilon_+$
  - Feature selection
    - $\min_{w,b} \frac{\lambda}{2} \|w\|^2_1 + \sum_{i=1}^{N} [1 - y_i f(x_i)]_+$
Loss Functions

• Maps the extent of error of a machine learning model for an example or a set of examples on to a real number

\[ L(X, y; f) = \sum_{i=1}^{N} max(0, 1 - y_i f(x_i)) \]

– Or \( l(x, y; f) = max(0, 1 - y f(x)) \)

Regularizers

- Controls the complexity error of the classifier
- There are also other regularizers
  - $\|\mathbf{w}\|_2^2 = w_1^2 + w_2^2 + \cdots + w_d^2$
    - Convex, Smooth
  - $\|\mathbf{w}\|_1^1 = |w_1| + |w_2| + \cdots + |w_d|$
    - Used for feature reduction
  - $\|\mathbf{w}\|_0 = \text{number of non-zero elements in } \mathbf{w}$
    - Minimization of this norm will lead to feature selection
End of Lecture

We want to make a machine that will be proud of us.

- Danny Hillis