UNIVERSITY OF AGDER

ICT Seminar 2
Support Vector Machines

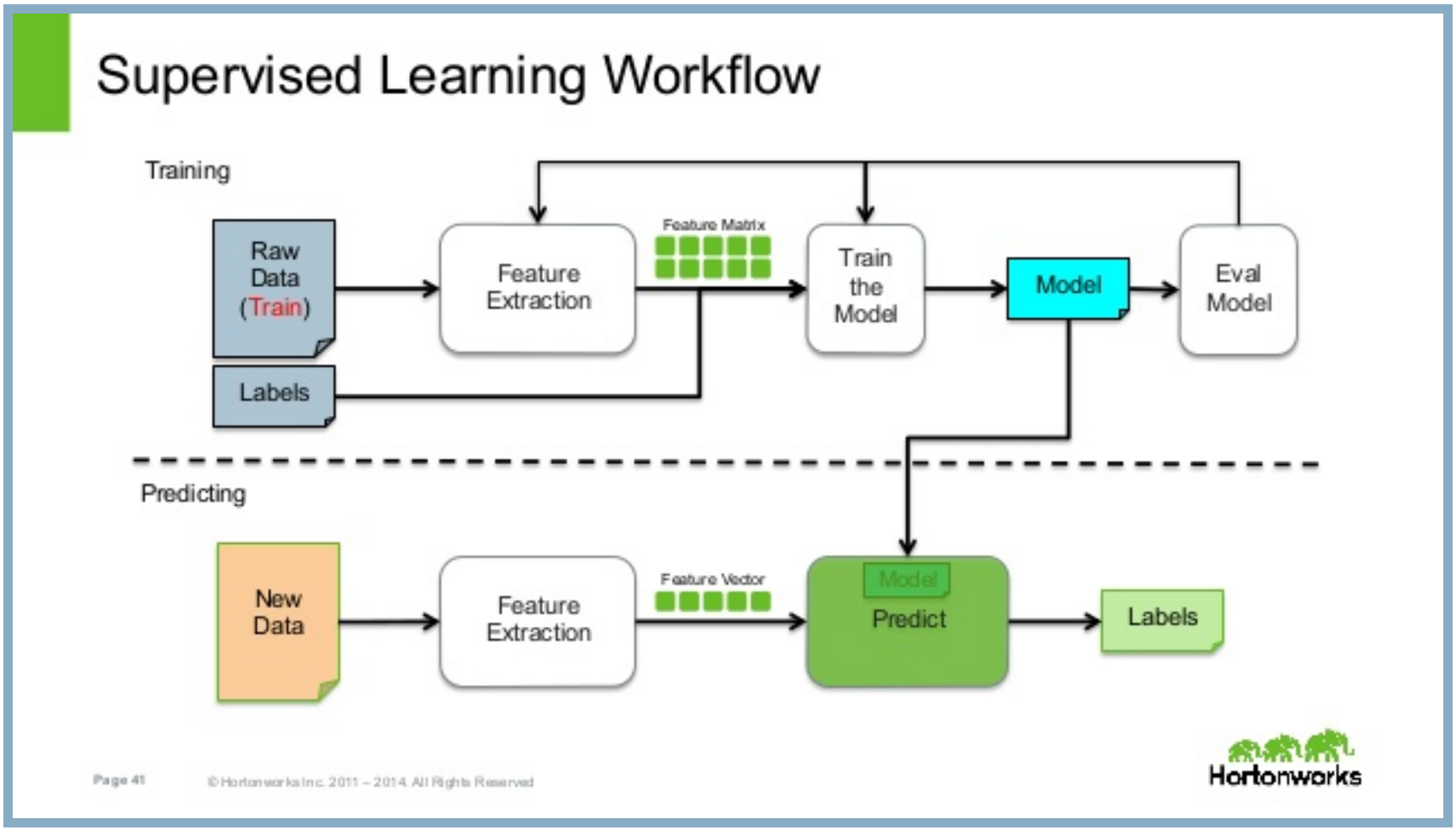
Associate Professor Dr. Morten Goodwin

2020

Overview

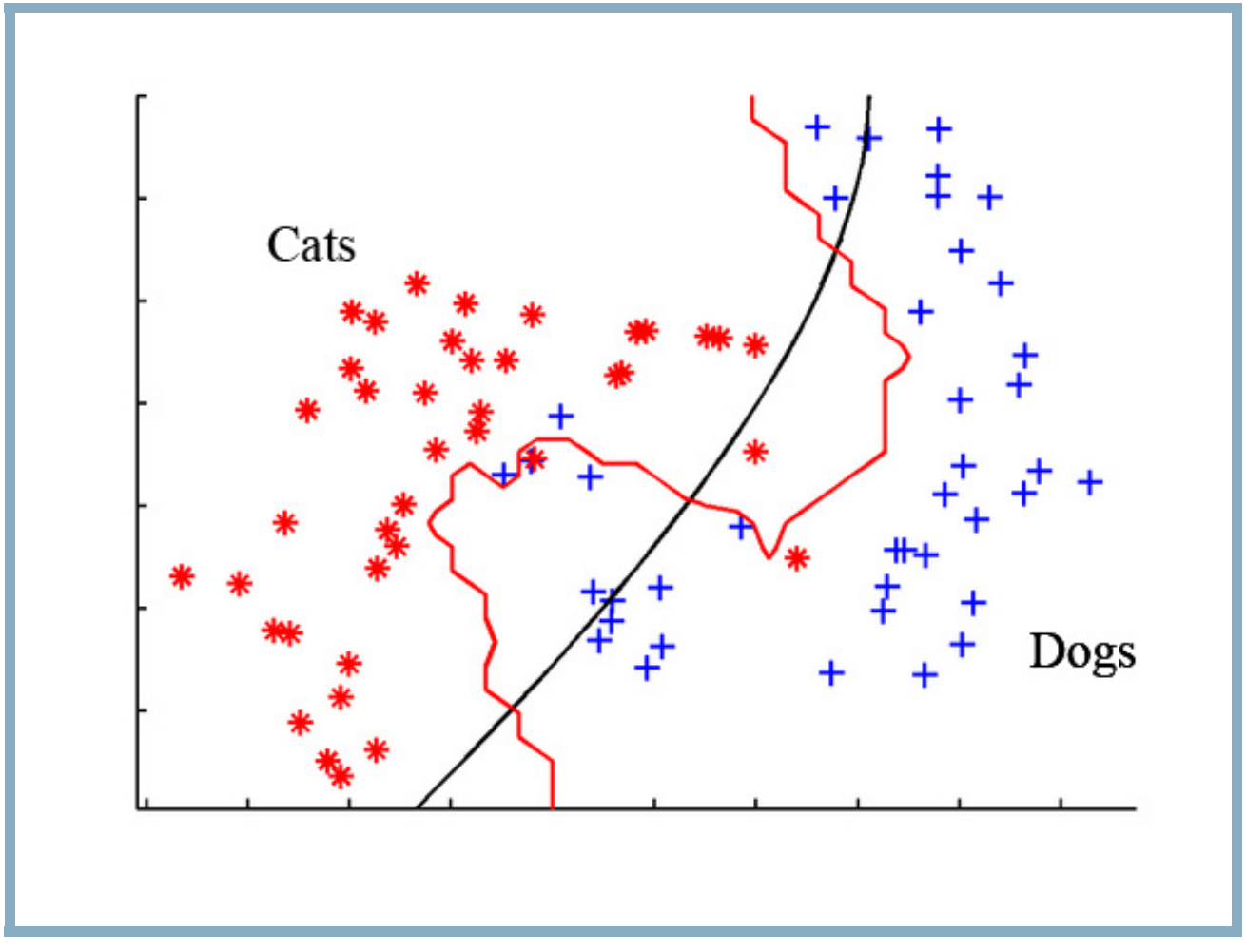
Introduction

Supervised Learning



Source of image: http://en.proft.me/2015/12/24/types-machine-learning-algorithms/

Classification



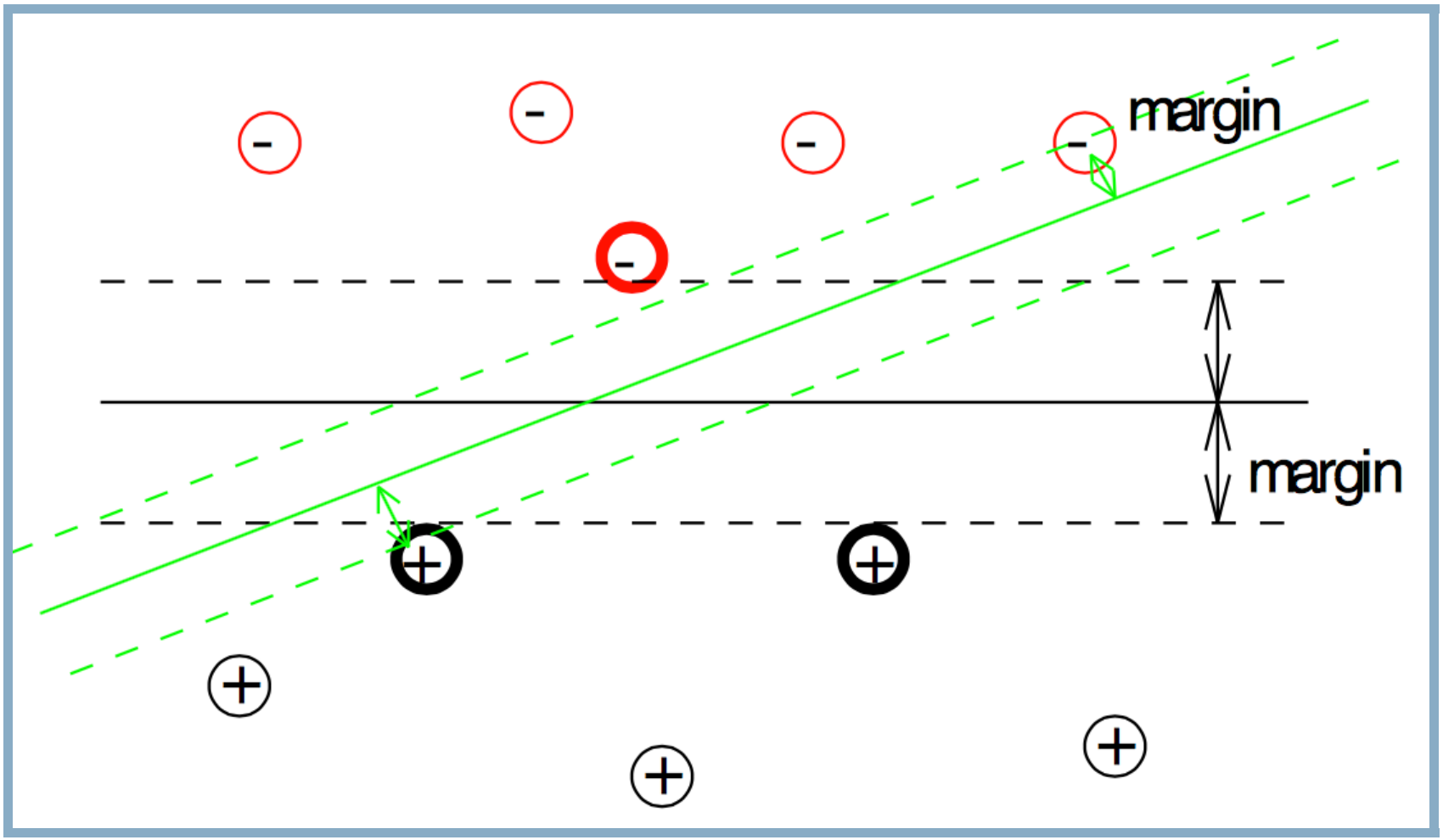
Source of image: http://en.proft.me/2015/12/24/types-machine-learning-algorithms/

Support Vector Machine

History

- In the 1960s: Statistical Learning Theory (Vapnik & Chervonenkis)
- In 1992 Boser, Guyon & Vapnik introduces Support Vector Machine.
- Positives: Theoretically and analytically correct.
- Empirically good performance.
  - Bioinformatics,
  - Text classification.
  - Image recognition.
  - ...

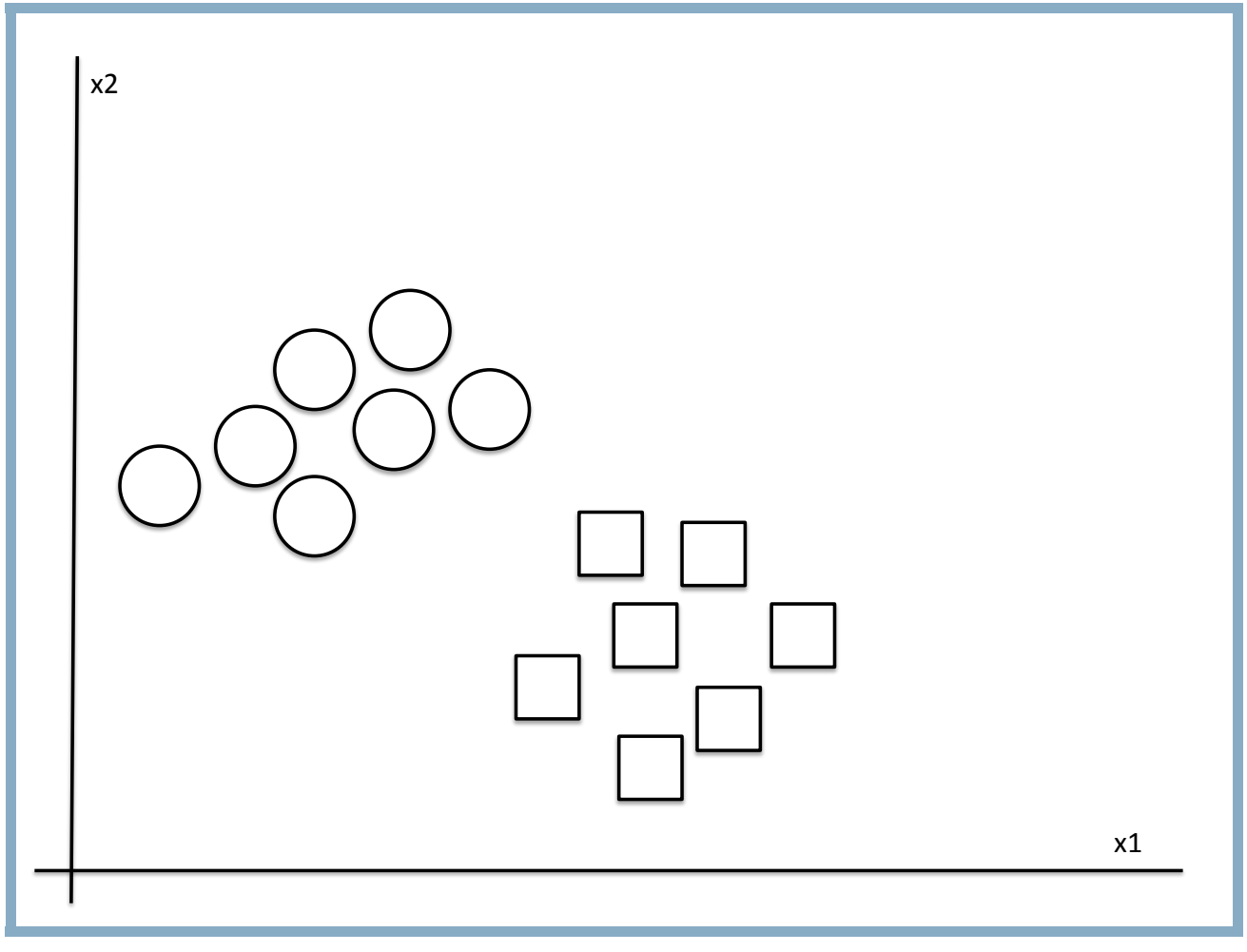
SVM Basics



Linearly separable binary set

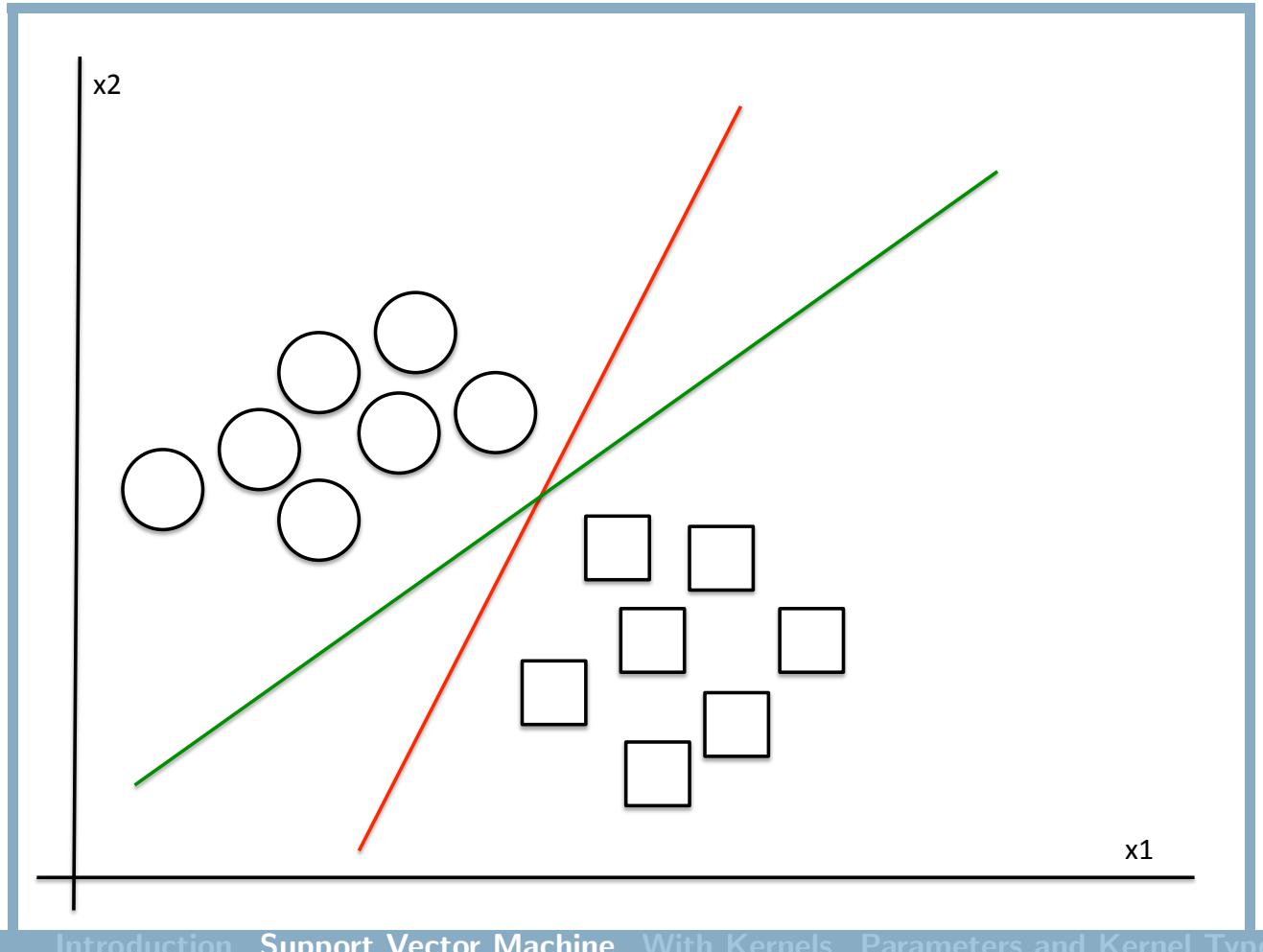
- Two types of items.
- Can be perfectly separated with a straight line.
- Goal define a hyperplane that classifies all training vectors in two classes

Linearly separable items



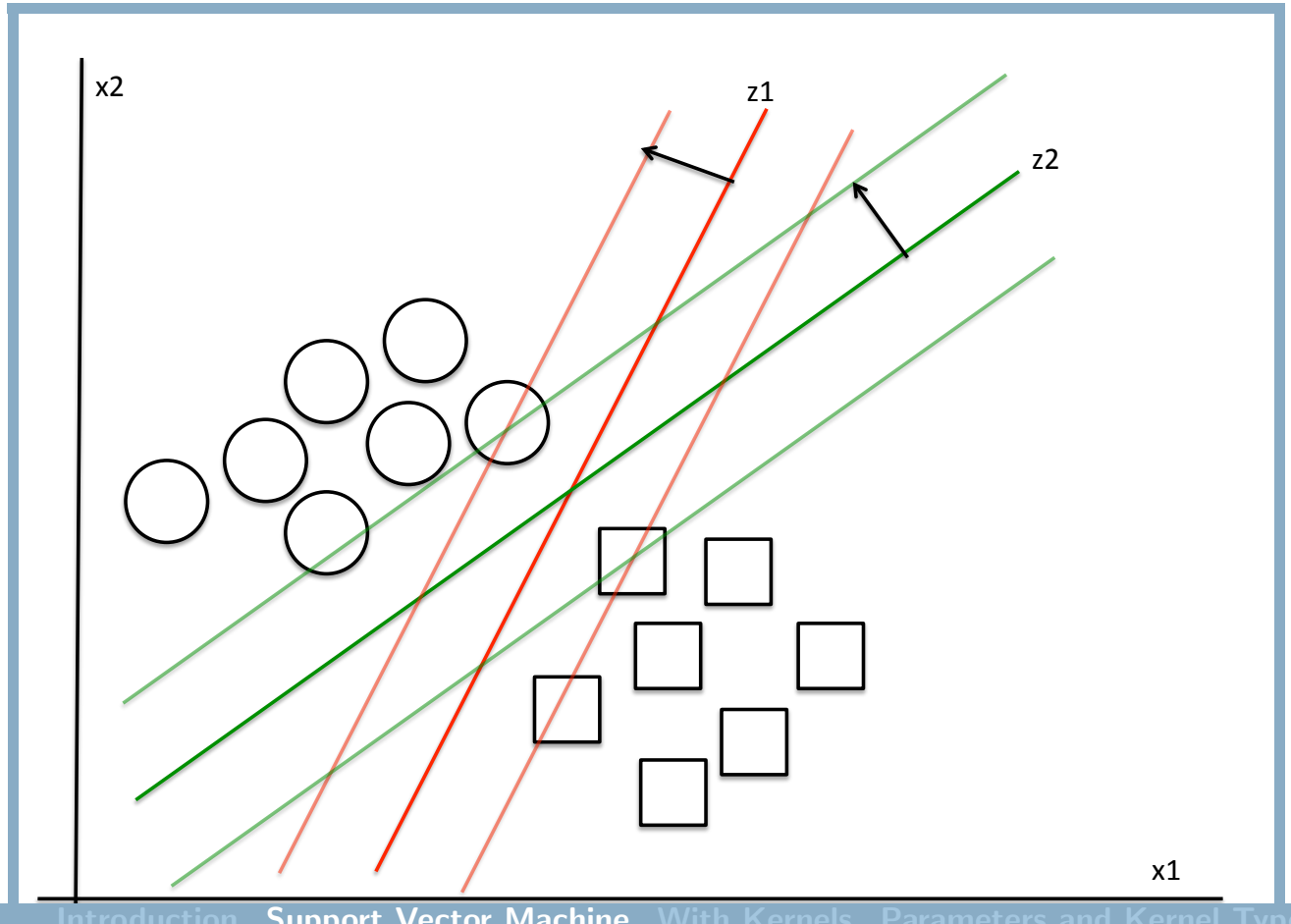
Assignment

- Which is the best hyperplane?

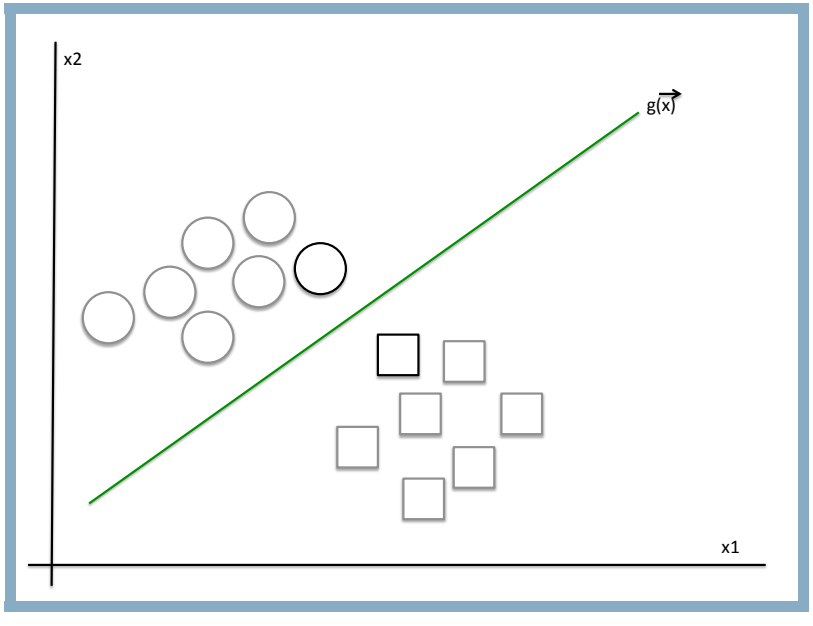


Solution

- The one with the highest margin?
- $z2 > z1$

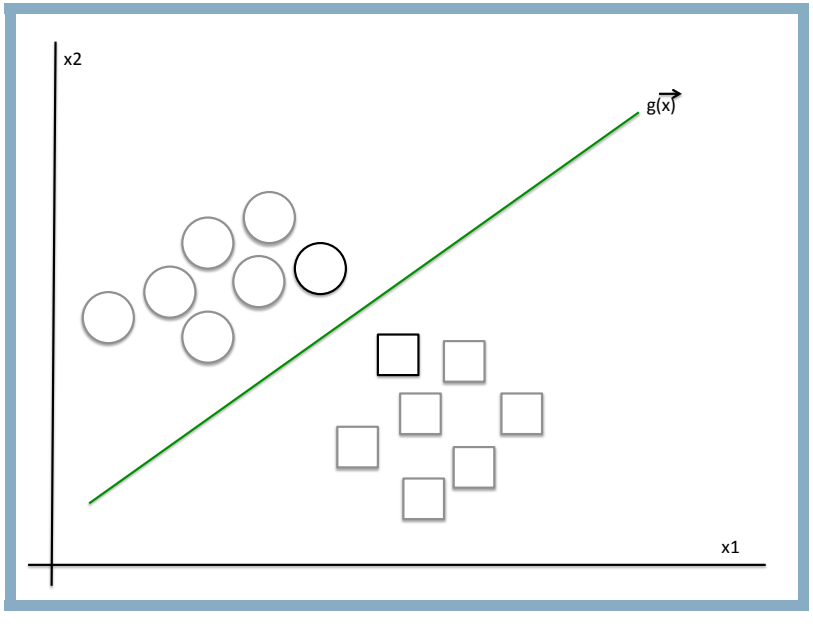


Define the classifier



- Equation: $g(\vec{x})$ = Vector of weights*vectorx+start weight.
- Equation: $g(\vec{x}) = \vec{w}^T \vec{x} + w_0$
- Equation for two dimensions: $g(\vec{x}) = \theta_1 * x_1 + \theta_2 * x_2 + w_0$
- What should $\vec{w}^T$ be?
- What should w_0 be?

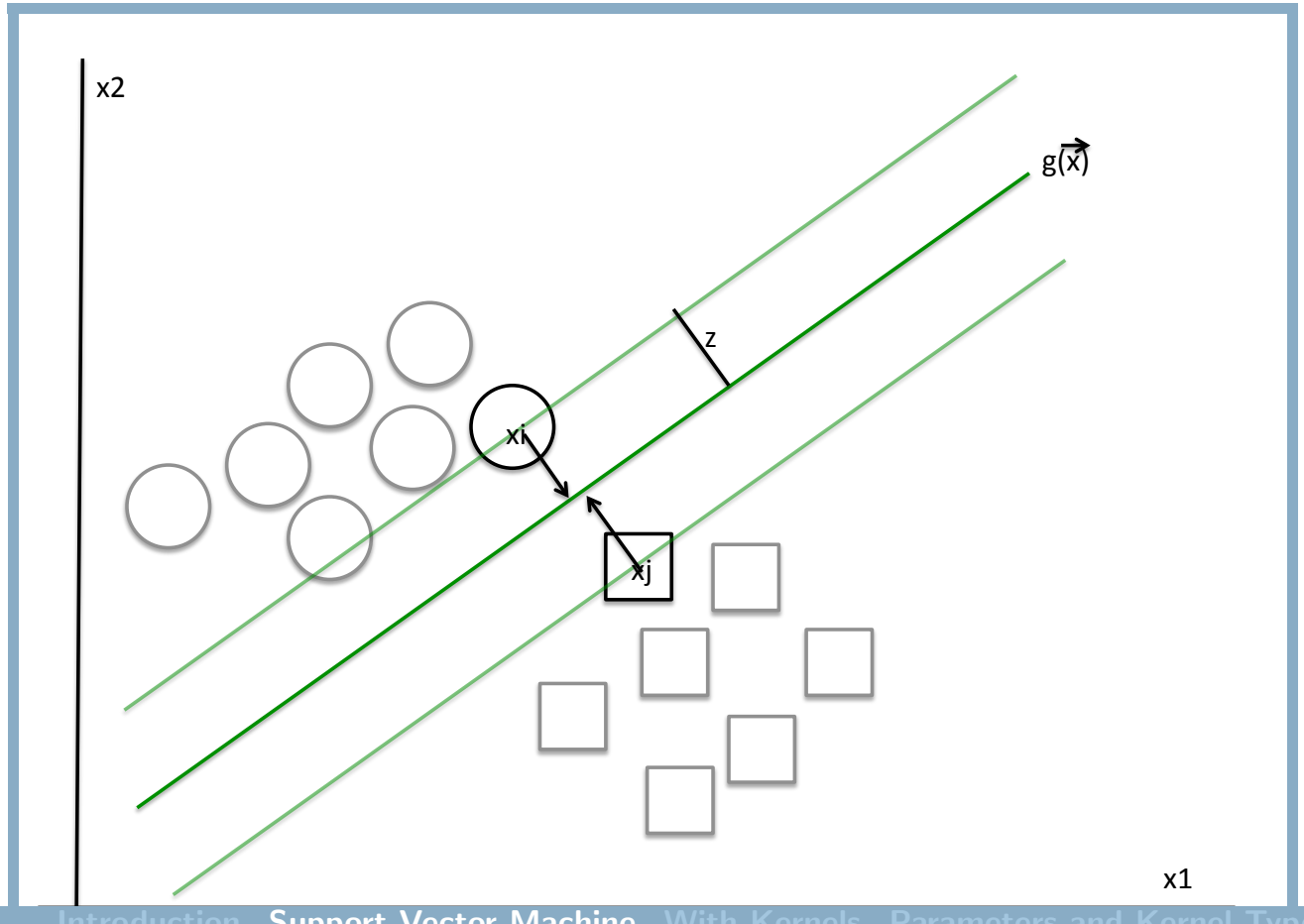
Define the classifier



- $g(\vec{x}) \leq 0 \Rightarrow$ Circle class
- $g(\vec{x}) \geq 0 \Rightarrow$ Square class

Compute and maximize the margins

- Choose some Vectors to Support the Classifier
  - Or let the Machine be Supported by Vectors

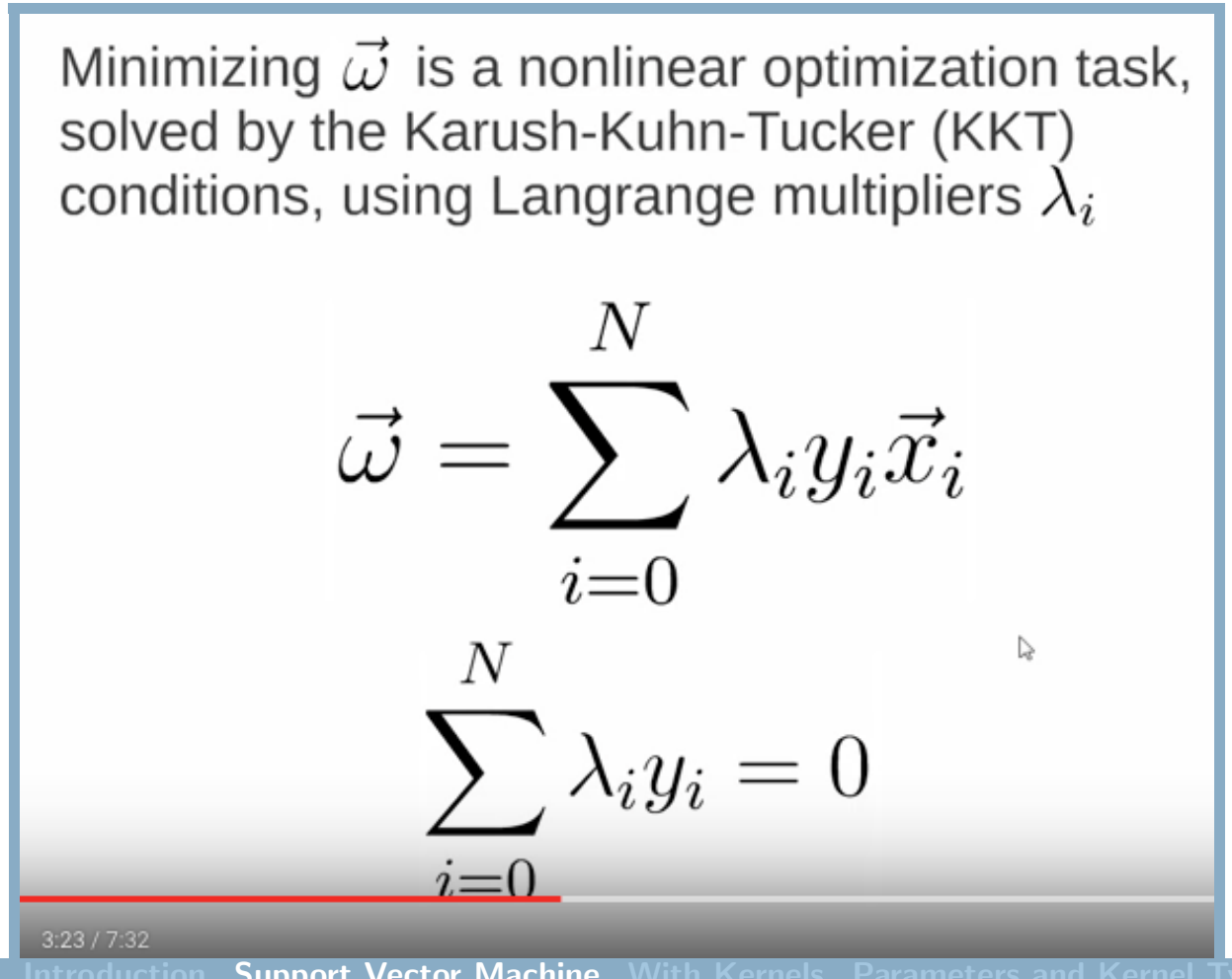


Minimize the margins

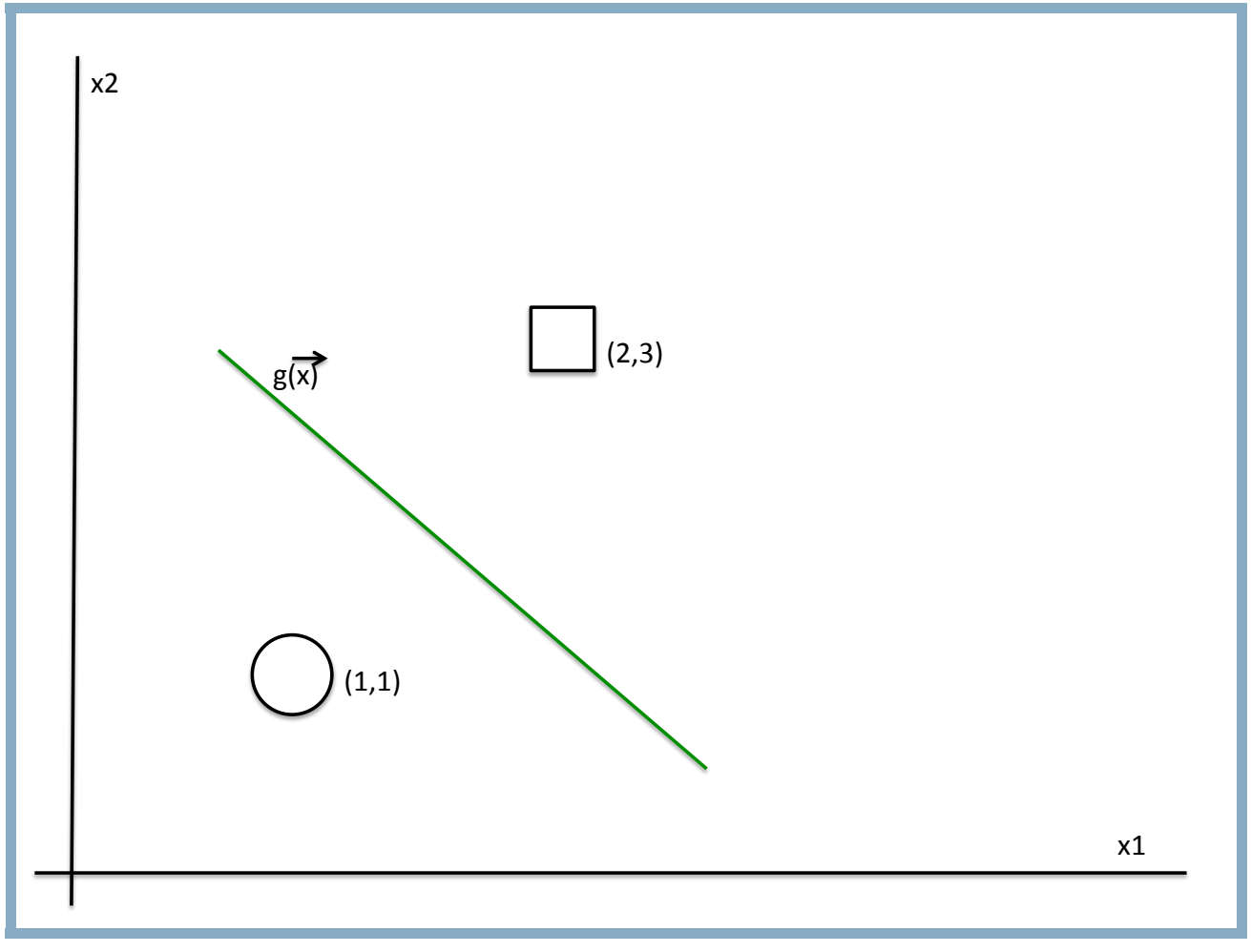
- Classifier Equation: $g(\vec{x}) = \vec{w}^T \vec{x} + w_0$
- $z = \frac{|g(\vec{x_i})|}{||\vec{w}||} = \frac{|g(\vec{x_j})|}{||\vec{w}||} = \frac{1}{||\vec{w}||}$
- Total margin is: $\frac{1}{||\vec{w}||} + \frac{1}{||\vec{w}||} = \frac{2}{||\vec{w}||}$.
- By minimizing the weight $\vec{w}$, we maximize the separability (maximizing z).

Minimizing

- Non-linear optimization task



Simple example



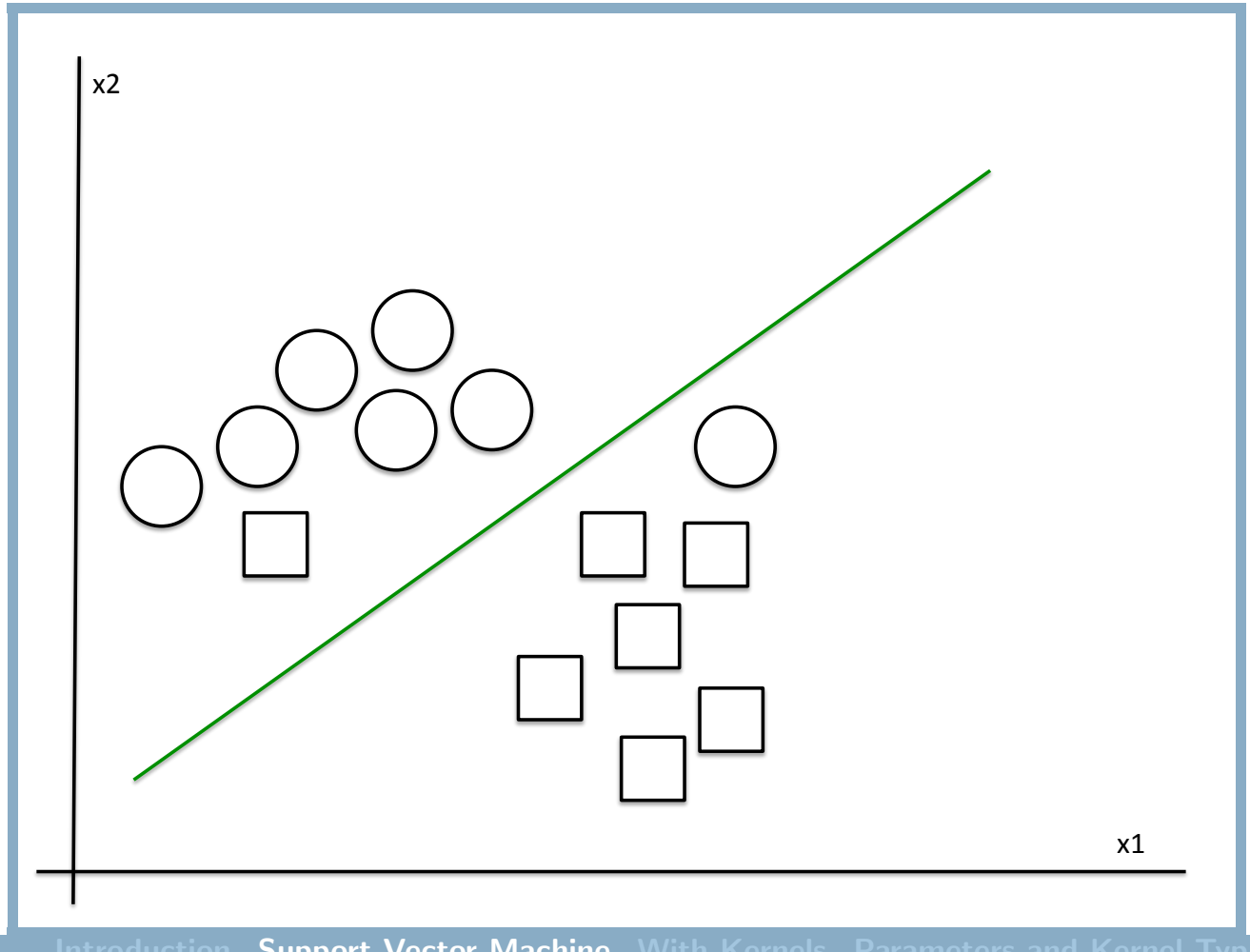
- Weights: $\vec{w} = (2, 2) - (1, 1) = (a, 2a)$

Solving the equation

- Weight: $\vec{w} = (2,3) - (1,1) = (a, 2a)$
- Classifier Equation:
  $g(\vec{x}) = \vec{w}^T\vec{x} + w_0 = (\theta_1 x1 + \theta_2 x2) + w_0 = a * x1 + 2 * a * x2 + w_0$
- Equation 1, using (1,1): $g(1,1) = a + 2a + w_0 = 1$
  - We known that $g(1,1) = 1$
- Equation 2, using (2,3): $g(2,3) = 2a + 6a + w_0 = -1$
  - We known that $g(2,3) = -1$
- Solve the questions for a and w_0.
- $a = \frac{-2}{5}, w_0 = \frac{11}{5}$
- The weight will then be: $\vec{w} = (a, 2a) = (\frac{-2}{5}, \frac{-4}{5})$
- The support vectors are:$(\frac{-2}{5}, \frac{-4}{5})$.
- Equation: $g(\vec{x}) = \vec{w}^T\vec{x} + w_0$
- The support vector machine: $g(\vec{x}) = \frac{-2}{5}x1 + \frac{-4}{5}x2 + \frac{11}{5}$
- Check 1: $g(1,1) = \frac{-2}{5} * 1 + \frac{-4}{5} * 1\frac{11}{5} = 1$
- Check 2: $g(2,3) = \frac{-2}{5} * 2 + \frac{-4}{5} * 3\frac{11}{5} = -1$

Soft margins

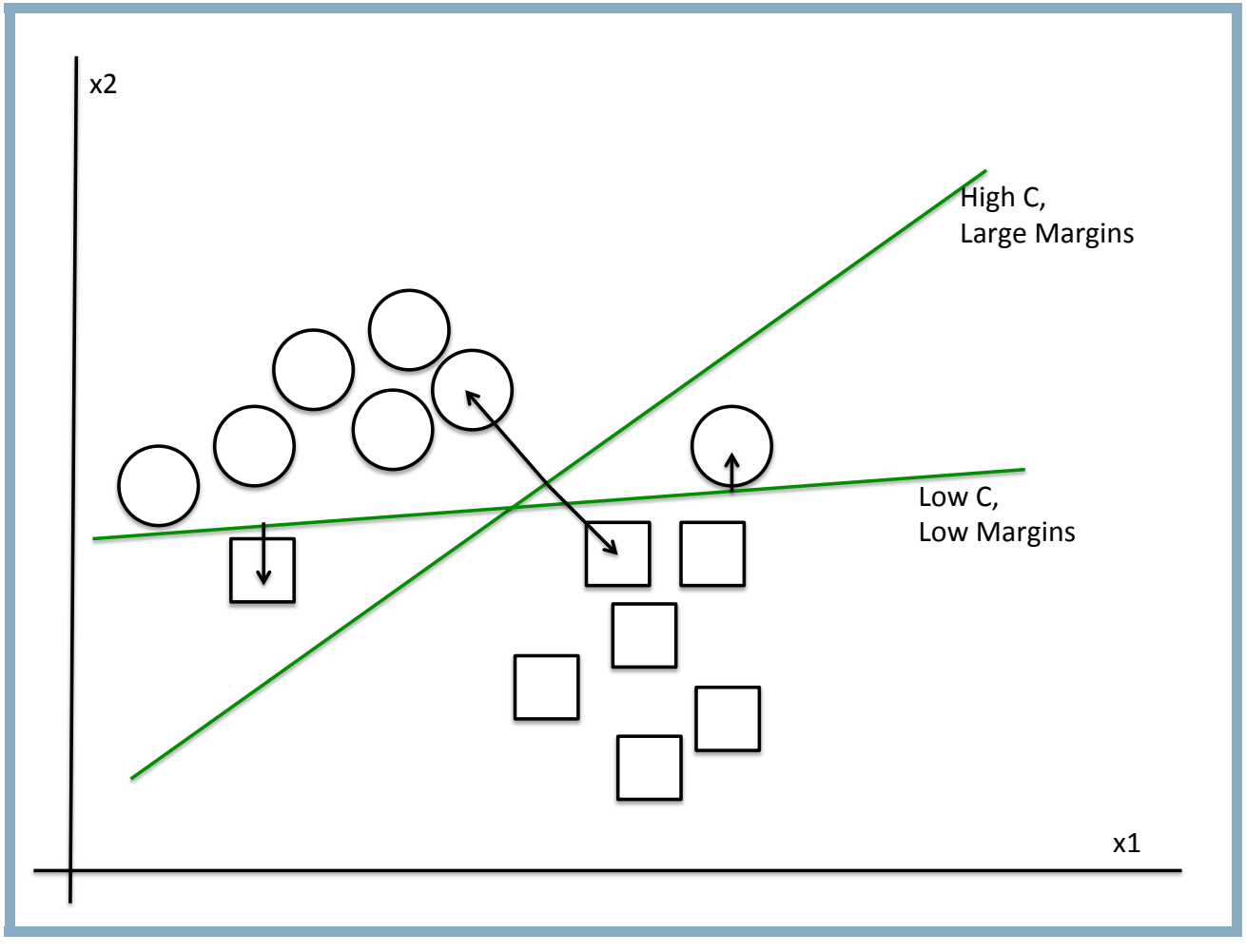
- How to do it now?



Hard-margins vs soft-margins

- The previous example have a hard margin.
- Soft margins means introducing a parameter C saying how much accept mis-classifications versus large margins.

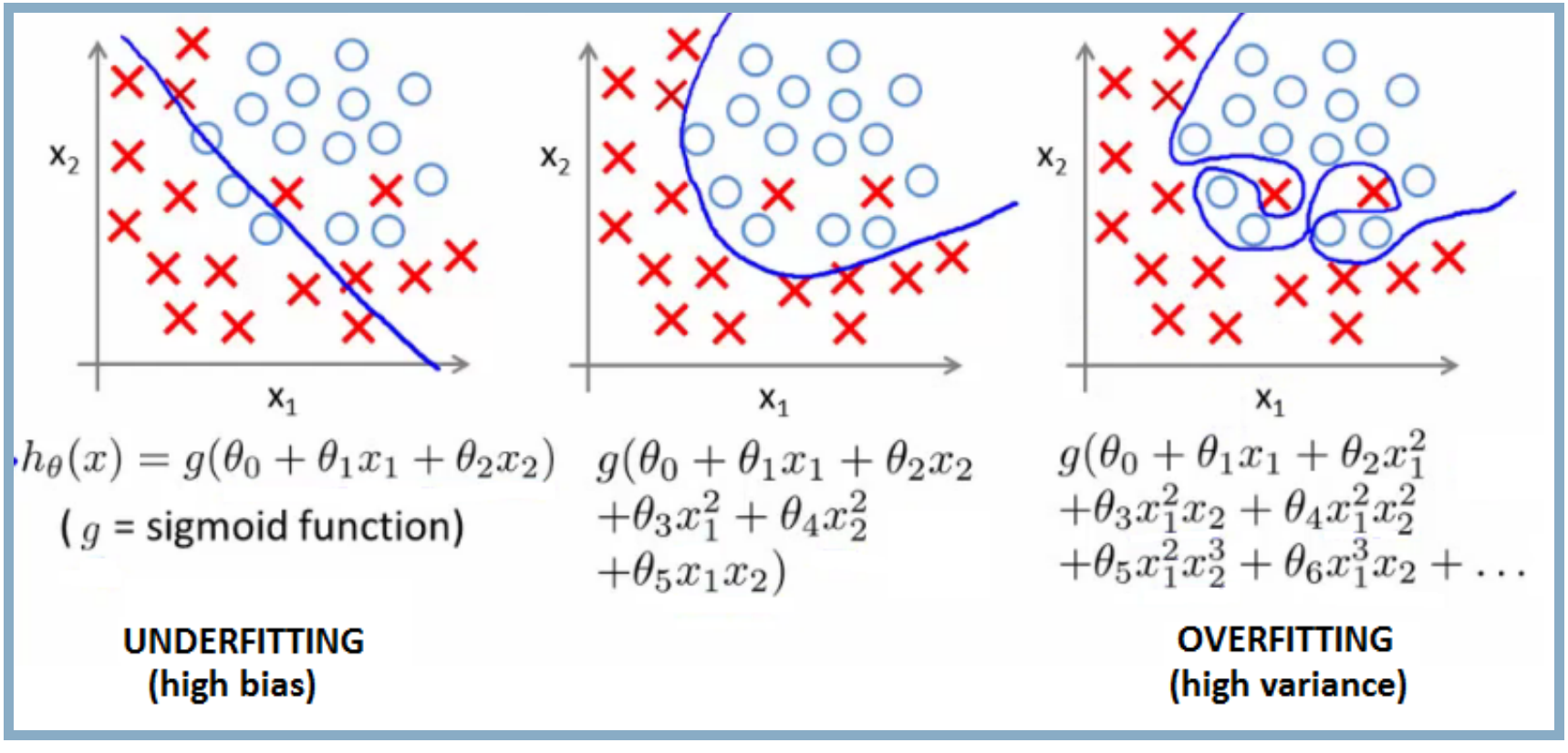
Varying C



Objective Function

- Intead of minimizing the weight, you minimise $f(\vec{w})$.
- $f(\vec{w}) = \frac{||\vec{w}||^2}{2} + C(\sum_{i=1}^{N} \xi_i)^k$

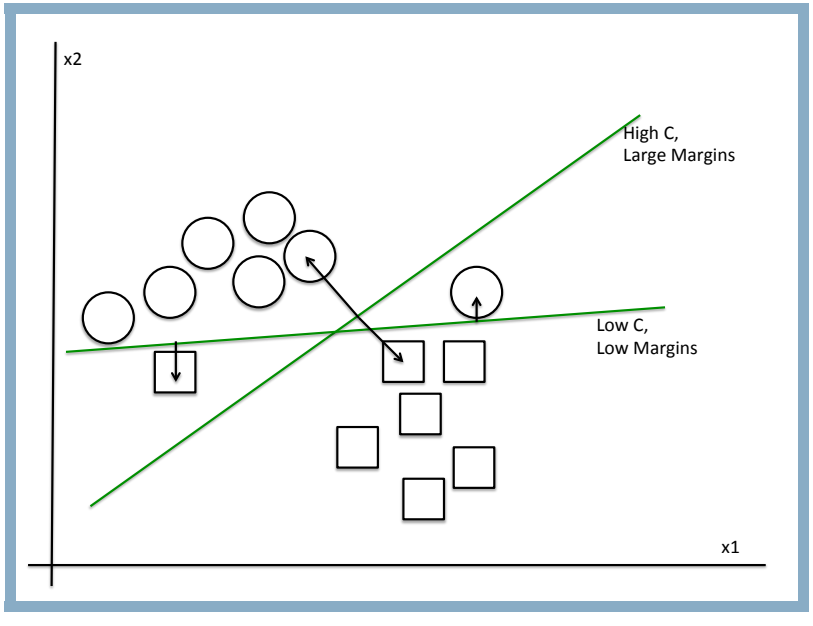
Overfitting versus Underfitting



Source of image: `http://www.holehouse.org/mlclass/07_Regularization.html`

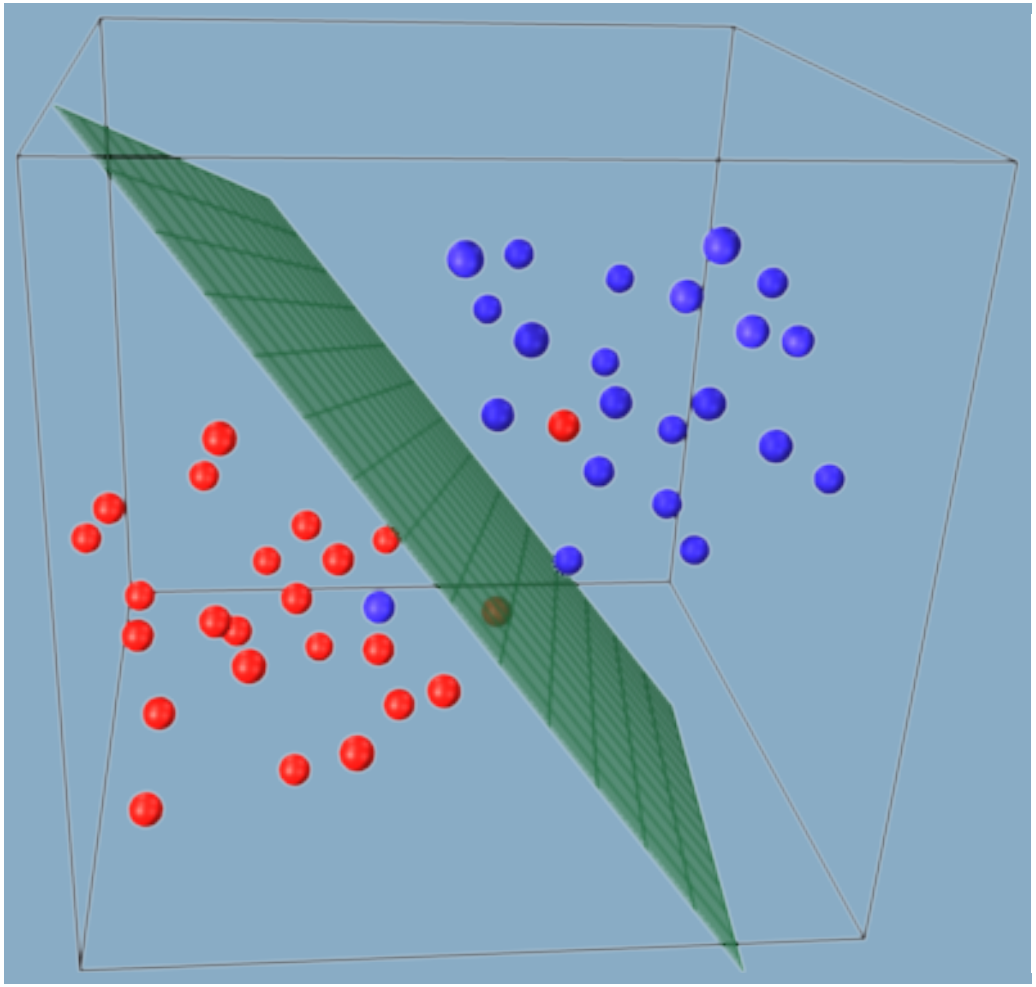
Overfitting in practice

Varying C



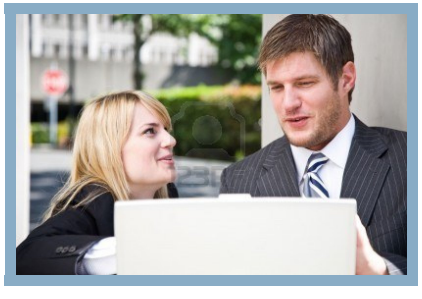
- High C
  - large margins
  - tends to underfit
- Low C
  - small margins
  - tends to overfit

Multiple dimensions



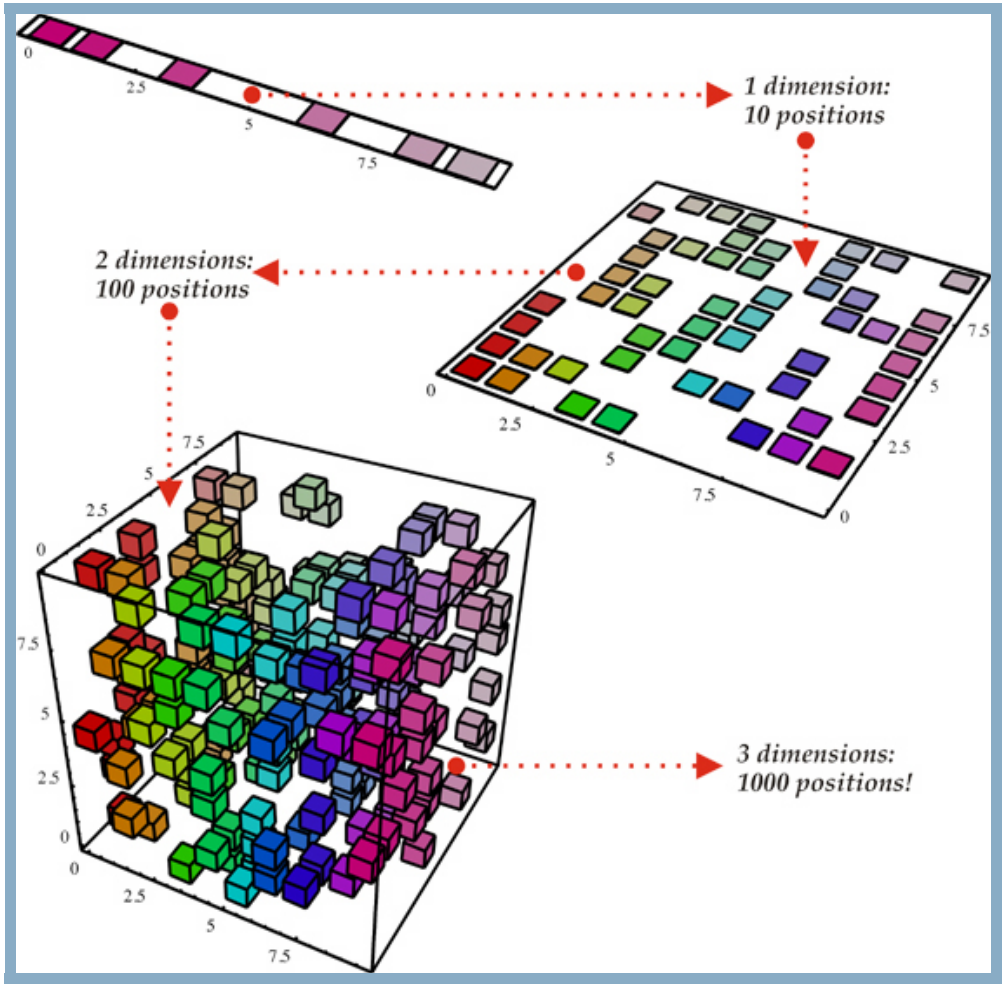
Source of image: http://stackoverflow.com/questions/9480605/what-is-the-relation-between-the-number-of-support-vectors-and-training-data-and

Discussion



- Any potential problem by having many dimension?

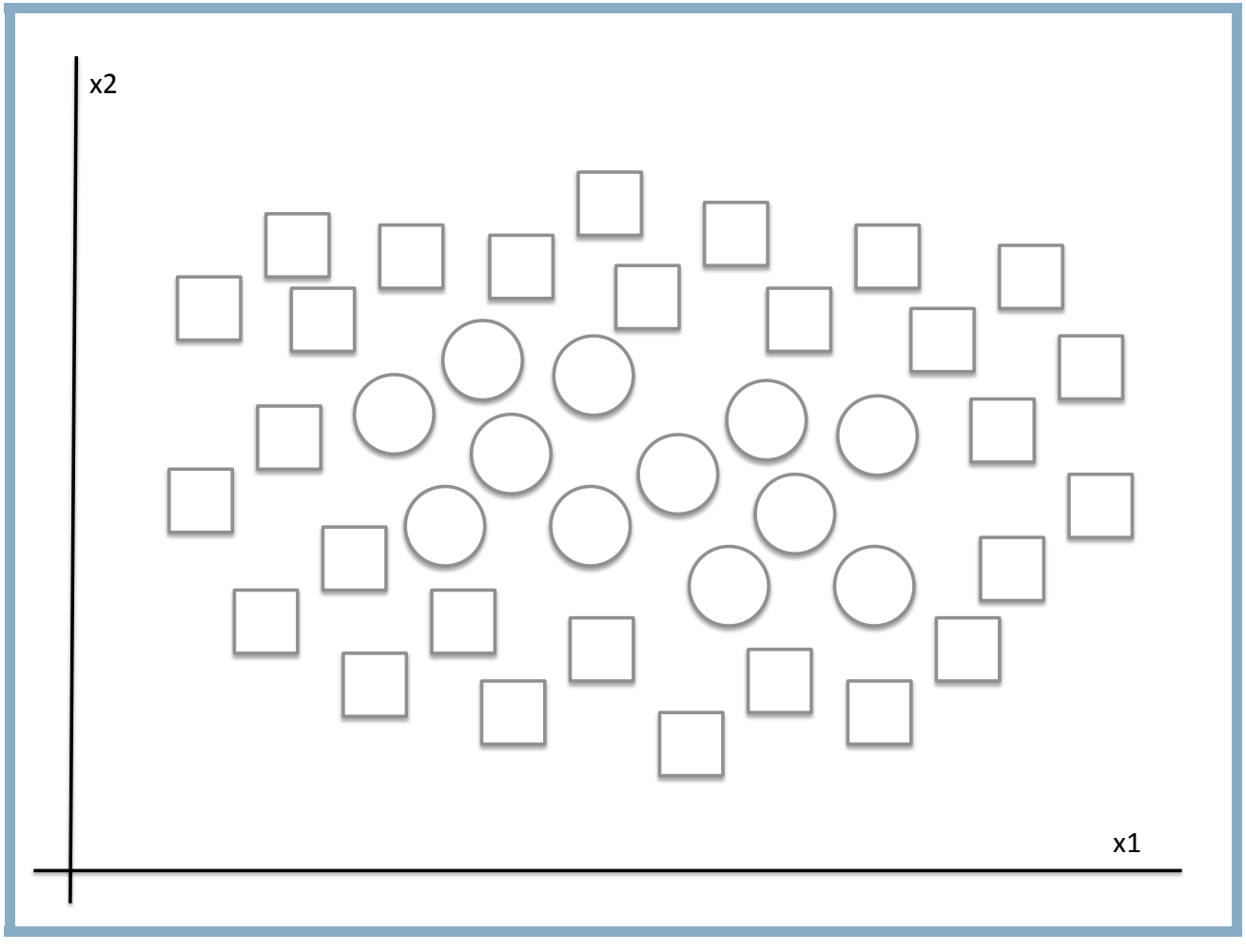
Curse of dimensionality



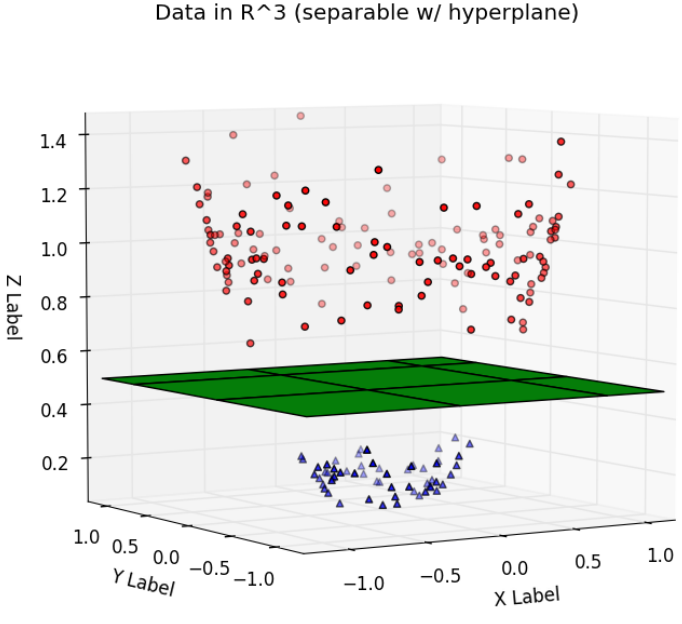
Source of image: http://www.iro.umontreal.ca/~bengioy/yoshua_en/research.html

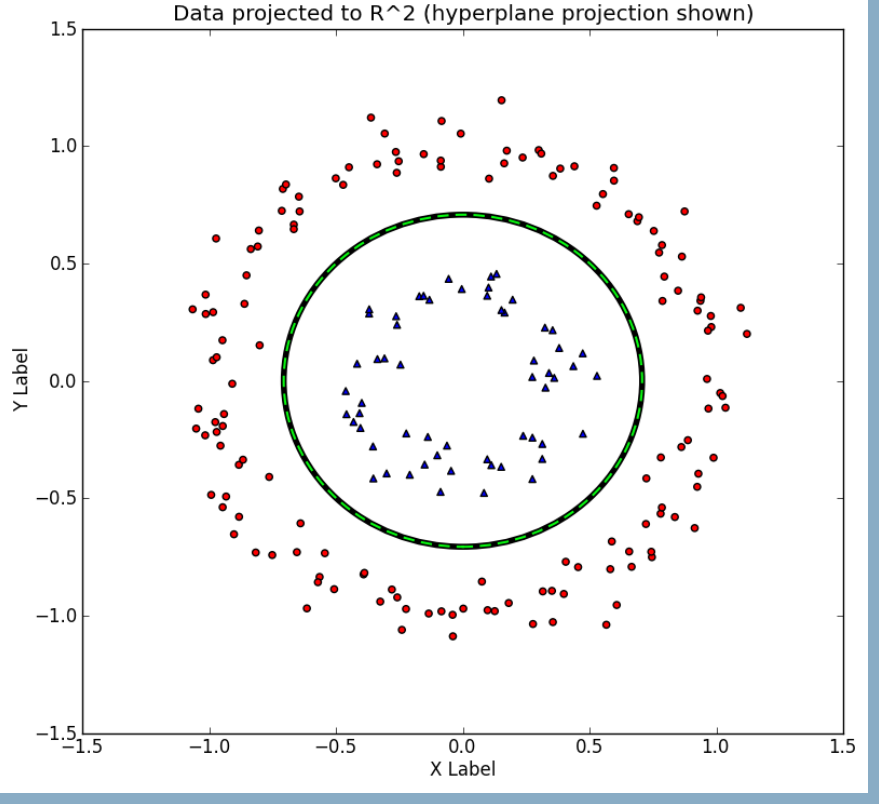
With Kernels

Non-linear decision boundary



Move to multiple dimensions



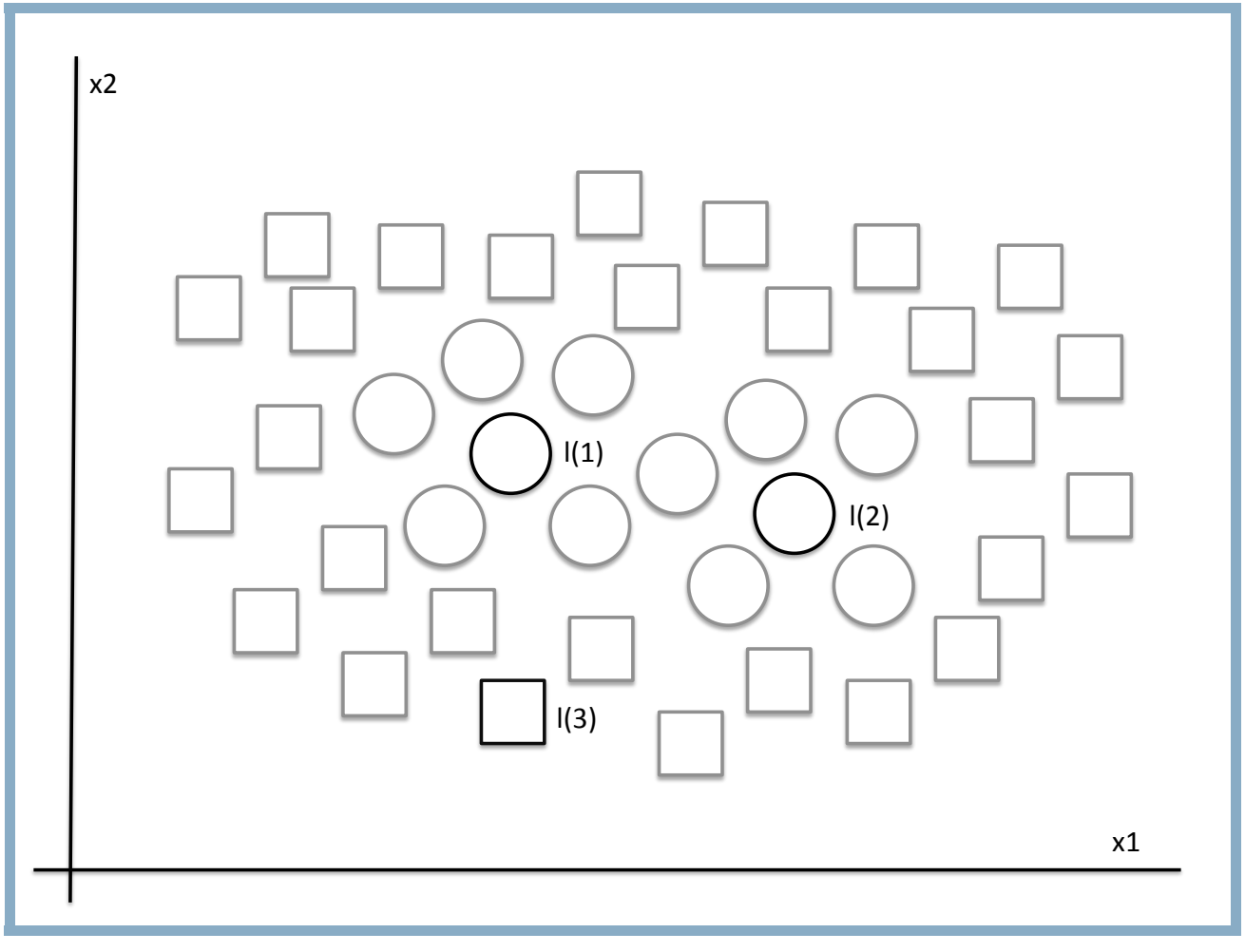


Source of image: `http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html/Pa`

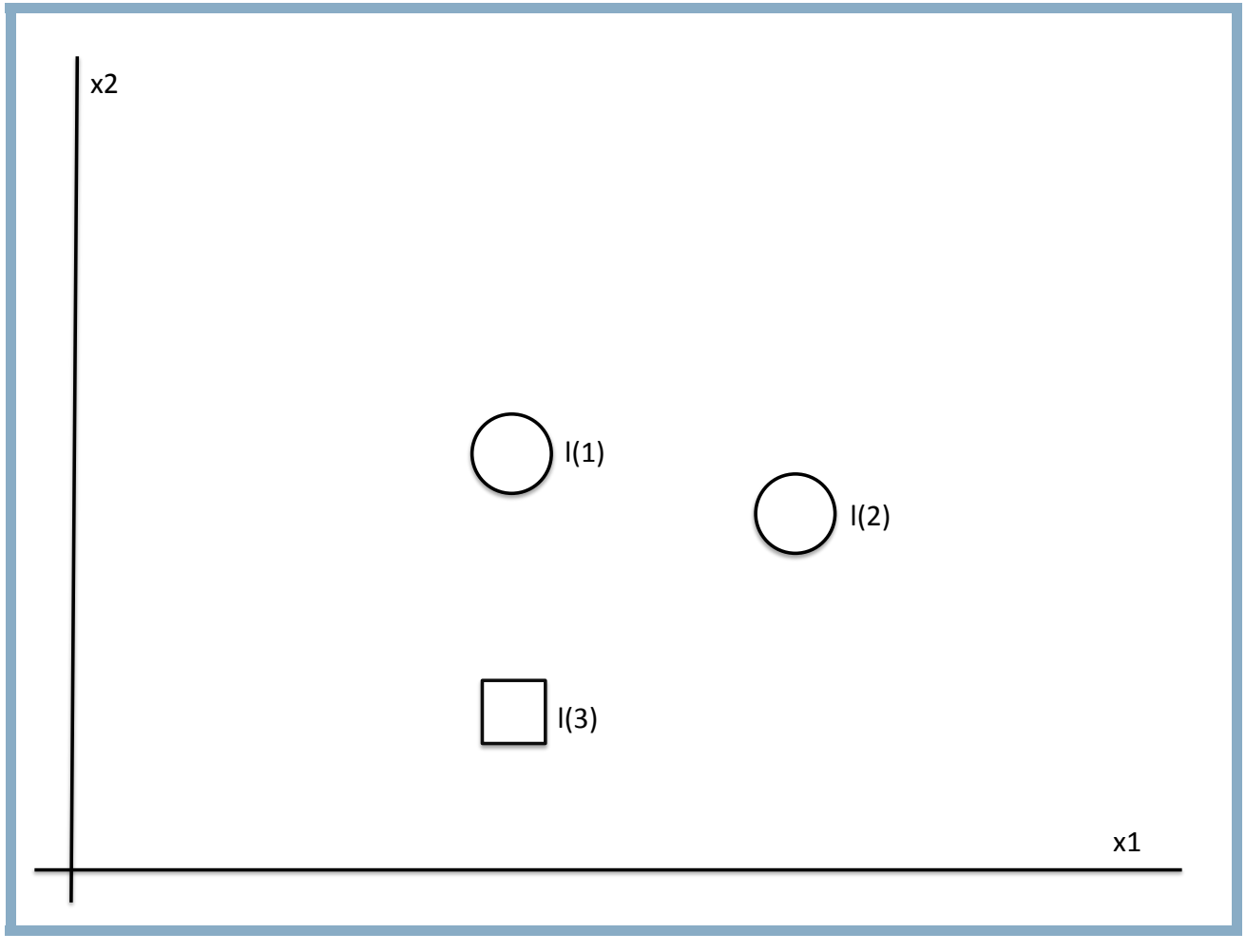
Mapping to multiple dimensions

- http://crsouza.com/2010/03/kernel-functions-for-machine-learning-applications/

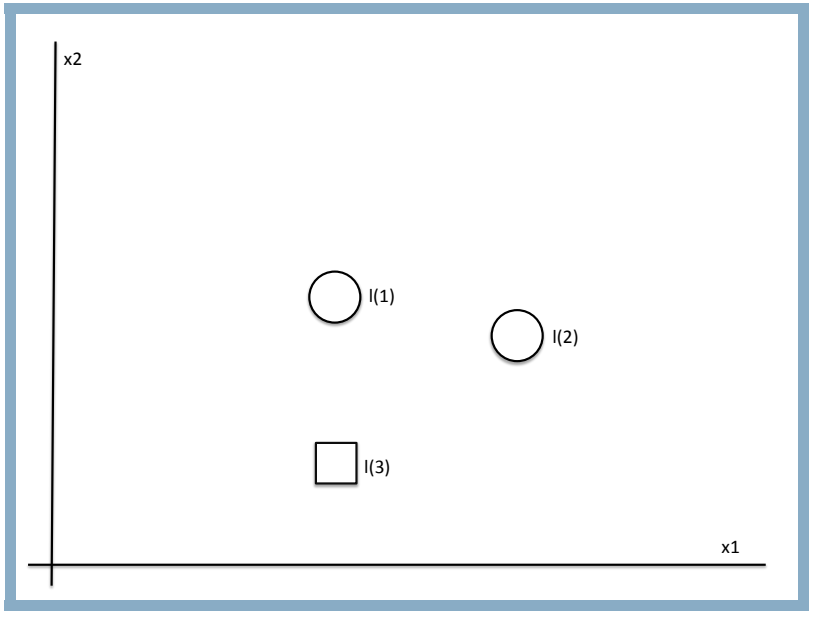
Choose some landmarks



Choose some landmarks



Add a kernel

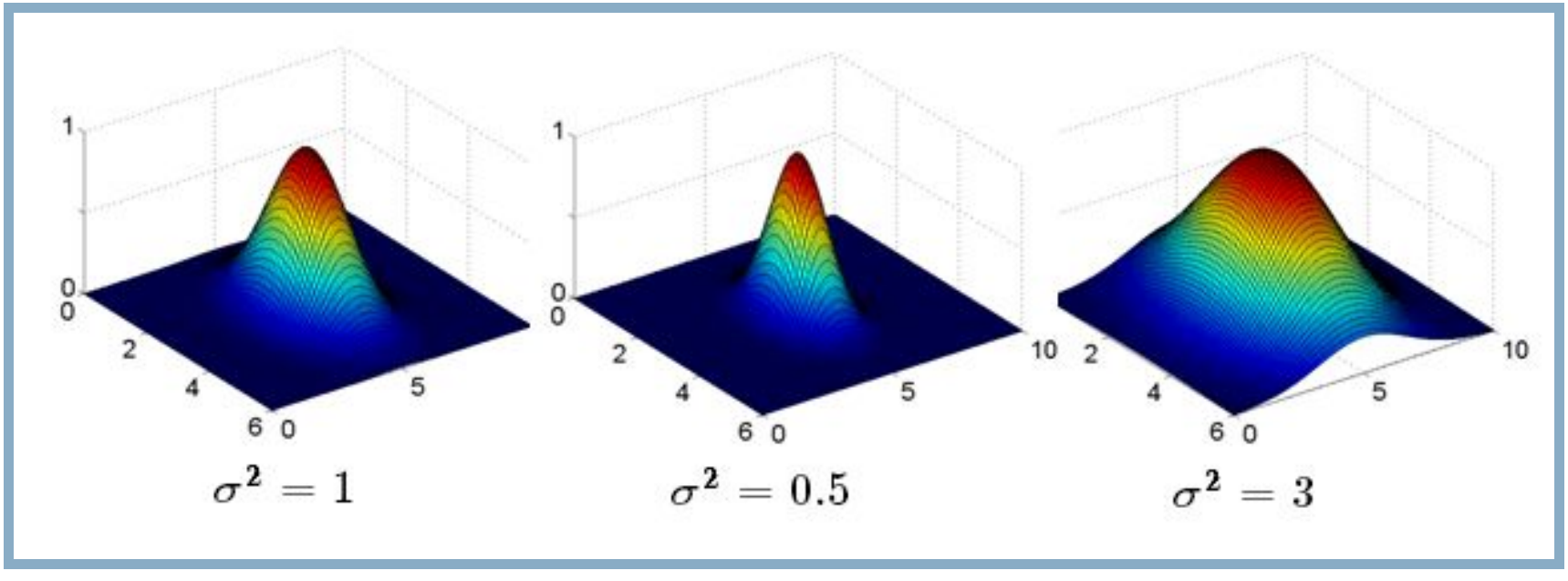


- Gaussian kernel: $similarity(\vec{a}, \vec{b}) = exp(-\frac{||\vec{a}-\vec{b}||}{2*\sigma^2})$
- For many x:
  - $f_1(\vec{x}) = similarity(\vec{x}, l(1))$
  - $f_2(\vec{x}) = similarity(\vec{x}, l(2))$
  - $f_2(\vec{x}) = similarity(\vec{x}, l(3))$
- $g(\vec{x}) = w_0 + \vec{w_1} * f_1(x) + \vec{w_2} * f_2(x) + \vec{w_3} * f_3(x)$
- $g(\vec{x}) \geq 0 \Rightarrow$ inside.
- $g(\vec{x}) < 0 \Rightarrow$ outside.

Values of x

- If $x \approx l(1)$
  - $f_1 \approx exp(-\frac{0^2}{2*\sigma^2}) \approx 1$
- If x is far away from $l(1)$
  - $f_1 \approx exp(-\frac{(largenumber)^2}{2*\sigma^2}) \approx 0$

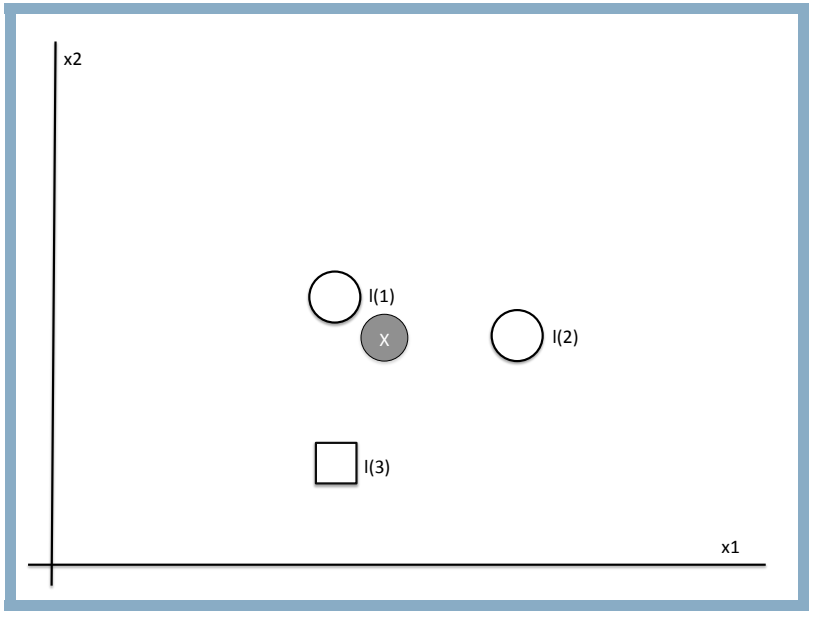
Choosing σ^2



- Choosing σ^2

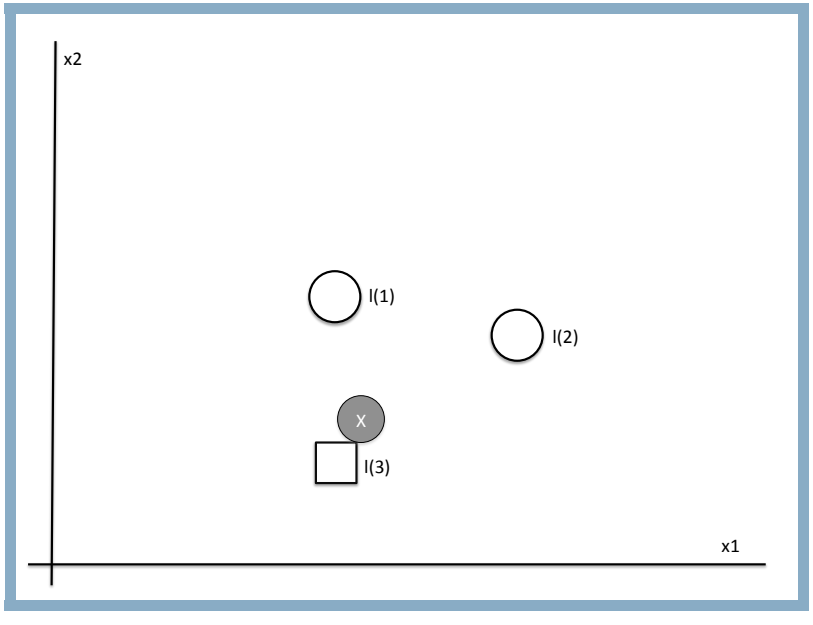
Source of image:

Predicting (classifying)



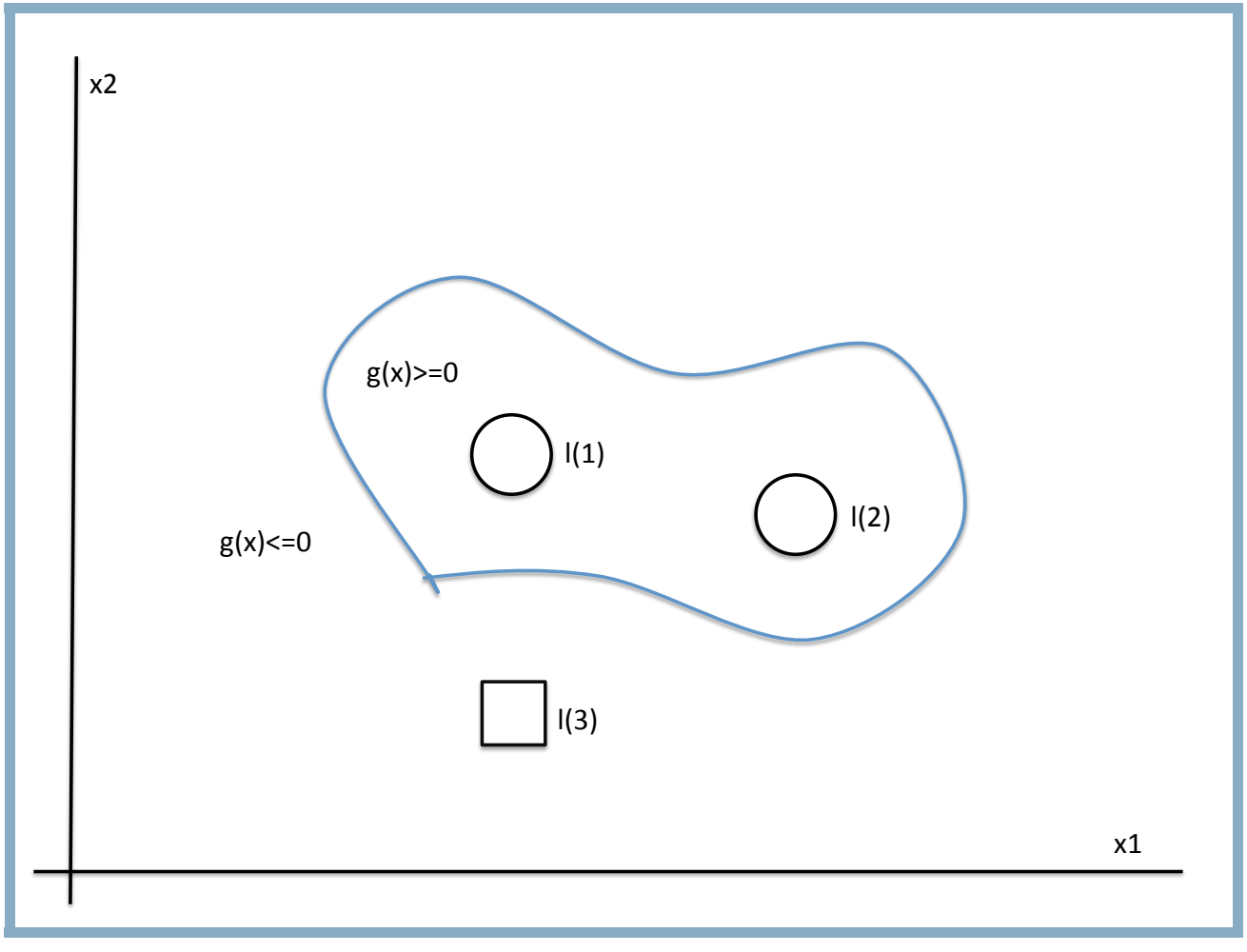
- $g(\vec{x}) = w_0 + \theta_1 * f_1(\vec{x}) + \theta_2 * f_2(\vec{x}) + \theta_3 * f_3(\vec{x})$
- Assume: $w_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0$
- $g(\vec{x}) = -0.5 + 1 * f_1(\vec{x}) + 1 * f_2(\vec{x}) * 0 * f_3(\vec{x}) \geq 0$
- Here: $f_1 \approx 0, f_2 \approx 1, f_3 \approx 1$
- $g(\vec{x}) = -0.5 + 1 * 1 + 1 * 0 + 0 = 0.5 \geq 0$ Inside

Predicting (classifying)

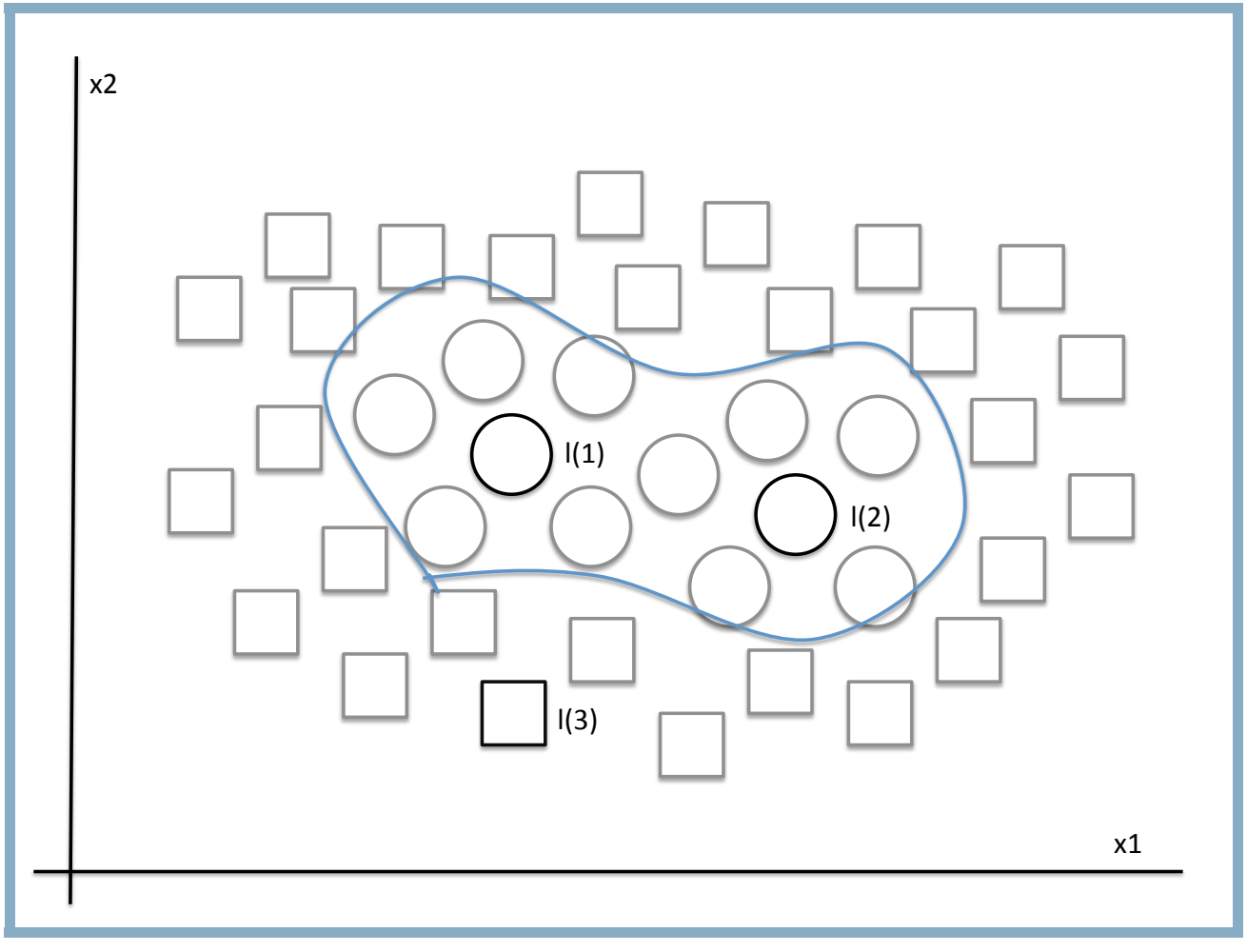


- $g(\vec{x}) = w_0 + \theta_1 * f_1(\vec{x}) + \theta_2 * f_2(\vec{x}) + \theta_3 * f_3(\vec{x}) \geq 0$
- Assume: $w_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0$
- $g(x) = -0,5 + 1 * f_1(\vec{x}) + 1 * f_2(\vec{x}) + 0 * f_3(\vec{x}) \geq 0$
- Here: $f_1 \approx 0, f_2 \approx 0, f_3 \approx 1$
- $g(X) = -0.5 + 0 * 1 + 0 * 1 + 1 * 0 = -0.5 \leq 0$ Outide

Predicting (classifying)



Predicting (classifying)



Choosing landmarks

- Different for each kernel type.
- Gaussian kernel:
  - One landmark per location of the training example

Parameters and Kernel Types

Paramters

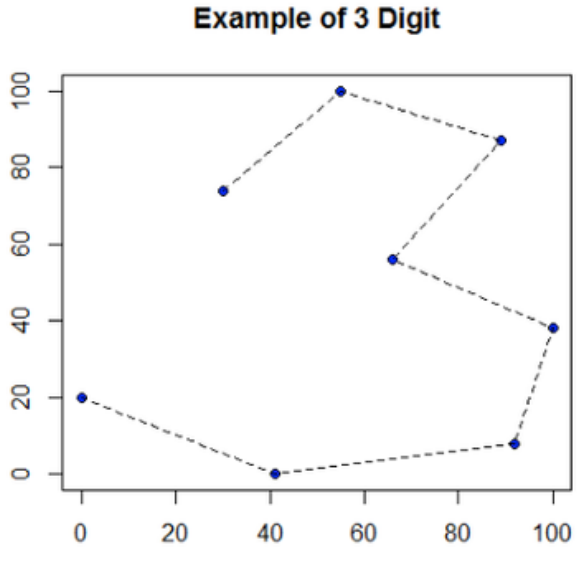
- A bit of black art.
- Sigma: (see Choosing σ^2)
- C: What do you favour: Large margins, or strict accuracy for the training data.

Kernel types

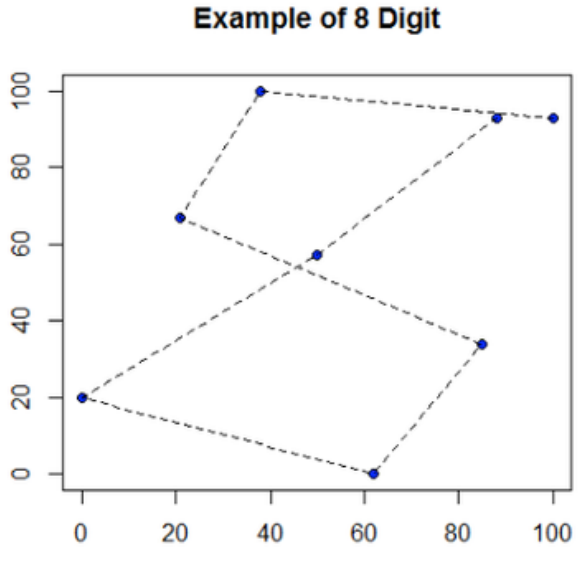
- Simple kernel:
  - Linear $g(x) = w^t x + w_o$
  - Polynomial $g(x) = (\alpha * w^T * x + w_o)^d$
- Gaussian Radial Basis Function (RBF) $g(x) = exp(-\frac{||x-y||}{2*\sigma^2})$
- Hyperbolic Tangent Kernel (Sigmoid) — comes from Neural Networks $g(x) = tanh(\alpha * w^T * x + w_0)$
- Rational Quadratic Kernel $g(x) = 1 - \frac{||x-w||}{||x-y||^2+w_0}$
- +++ 25 kernel types commonly in use (ref. `http://crsouza.com/2010/03/kernel-functions-for-machine-learning-applications/`).
- Kernels mush satisfy Mercer's Theorem: The kernel function between a pair of vectors is equivalent to the dot product in the transformed space.

Coding example

Digits



(a) Sample Points w/ Ordered Line Segments ('3')



(b) Sample Points w/ Ordered Line Segments ('8')

Source of image:

http://cs229.stanford.edu/proj2013/AragonLaneZhang-ClassifyHandWrittenNumericDigits.pdf

Classifying pen digits I

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  from sklearn import svm
4
5  import warnings
6  warnings.simplefilter("ignore")
7
8  training = [[int(i) for i in i.split(",")] for i in open("pendigits.tra").readlines↩
       () if i.strip()]
9  testing = [[int(i) for i in i.split(",")] for i in open("pendigits.tes").readlines↩
       () if i.strip()]
10
11 training_0 = [i[:-1] for i in training if i[-1]==0]
12 training_1 = [i[:-1] for i in training if i[-1]==1]
13
```

Classifying pen digits II

```
14 testing_0 = [i[:-1] for i in testing if i[-1]==0]
15 testing_1 = [i[:-1] for i in testing if i[-1]==1]
16
17
18 #Mapping to 2d --- for plot purposes
19
20 def mapTo2D(data):
21     retval = []
22     for i in range(0,len(data),2):
23         x = data[i]
24         y = data[i+1]
25         retval.append((x,y))
26     return retval
27
28 training_2d_0 = []
29 training_2d_1 = []
```

Classifying pen digits III

```
30
31 for d in training_0:
32     training_2d_0 += mapTo2D(d)
33
34
35 for d in training_1:
36     training_2d_1 += mapTo2D(d)
37
38 #Plotting 2d
39 plt.subplot(2,2,1)
40 plt.plot([i[0] for i in training_2d_0],[i[1] for i in training_2d_0], "-o", color="↩
       green")
41
42 plt.subplot(2,2,2)
43 plt.plot([i[0] for i in training_2d_1],[i[1] for i in training_2d_1], "-o", color="↩
       green")
```

Classifying pen digits IV

```
44
45
46  plt.subplot(2,2,3)
47  plt.plot([i[0] for i in training_2d_0][:8],[i[1] for i in training_2d_0][:8], "-o", ↩
        color="green")
48
49  plt.subplot(2,2,4)
50  plt.plot([i[0] for i in training_2d_1][:8],[i[1] for i in training_2d_1][:8], "-o", ↩
        color="green")
51  plt.show()
52
53  #2d classification
54  X = np.array(training_2d_0[:8]+training_2d_1[:8])
55  Y = np.array([0 for i in training_2d_0][:8] + [1 for i in training_2d_1][:8])
56
57  C=1.0
```

Classifying pen digits V

```
58  gamma = 0.5
59  svm_linear = svm.SVC(kernel='linear',C=C,gamma=gamma).fit(X,Y)
60  #svm_polynomial = svm.SVC(kernel='poly',C=C,gamma=gamma).fit(X,Y)
61  svm_rbf = svm.SVC(kernel='rbf',C=C,gamma=gamma).fit(X,Y)
62  svm_sigmoid = svm.SVC(kernel='sigmoid',C=C,gamma=gamma).fit(X,Y)
63
64  h = 0.2 #Mesh step
65  x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
66  y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
67  xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
68                       np.arange(y_min, y_max, h))
69
70  def plotSVM(svm,n,title):
71      plt.subplot(2,2,n)
72      Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
73
```

Classifying pen digits VI

```
74      Z = Z.reshape(xx.shape)
75      plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
76      plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.Paired)
77      plt.title(title)
78
79  plotSVM(svm_linear,1,"Linear")
80  plotSVM(svm_rbf,2,"RBF")
81  plotSVM(svm_sigmoid,3,"Sigmoid")
82
83  plt.show()
84
85
86  #Testing 2d
87
88  testing_2d_0 = []
89  testing_2d_1 = []
```

Classifying pen digits VII

```
90
91  for d in testing_0:
92      testing_2d_0 += mapTo2D(d)
93
94
95  for d in testing_1:
96      testing_2d_1 += mapTo2D(d)
97
98  def testSVM(svm,zero,one):
99      numcorrect = 0
100     numwrong = 0
101     for correct,testing in ((0,zero),(1,one)):
102         for d in testing:
103             r = svm.predict(d)[0]
104             if(r==correct):
105                 numcorrect += 1
```

Classifying pen digits VIII

```
106                 else:
107                     numwrong += 1
108         print "Correct",numcorrect
109         print "Wrong",numwrong
110
111     print "Testing 2d"
112
113     print "Linear"
114     testSVM(svm_linear,testing_2d_0,testing_2d_1)
115     #import pdb;pdb.set_trace()
116
117     print "RBF"
118     testSVM(svm_rbf,testing_2d_0,testing_2d_1)
119     #import pdb;pdb.set_trace()
120
121     print "Sigmoid"
```

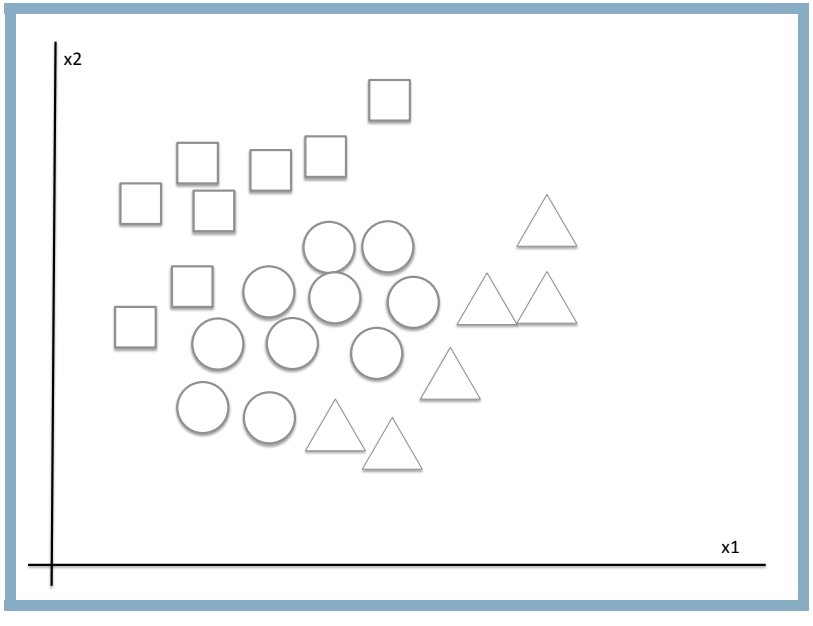
Classifying pen digits IX

```
122 testSVM(svm_sigmoid,testing_2d_0,testing_2d_1)
123 #import pdb;pdb.set_trace()
124
125 #8d classification
126 X = np.array(training_0+training_1)
127 Y = np.array([0 for i in training_0] + [1 for i in training_1])
128
129
130 svm_linear = svm.SVC(kernel='linear',C=C,gamma=gamma).fit(X,Y)
131 svm_poly = svm.SVC(kernel='poly',C=C,gamma=gamma).fit(X,Y)
132 svm_rbf = svm.SVC(kernel='rbf',C=C,gamma=gamma).fit(X,Y)
133 svm_sigmoid = svm.SVC(kernel='sigmoid',C=C,gamma=gamma).fit(X,Y)
134
135
136
137 print "Testing 8d"
```

Classifying pen digits X

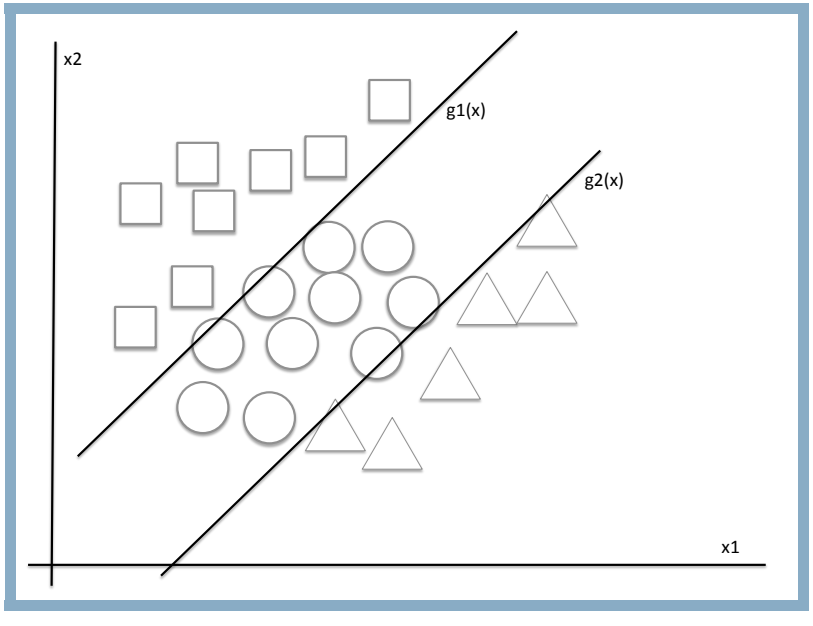
```
138
139 print "Linear"
140 testSVM(svm_linear,testing_0,testing_1)
141
142 print "RBF"
143 testSVM(svm_rbf,testing_0,testing_1)
144
145 print "Sigmoid"
146 testSVM(svm_sigmoid,testing_0,testing_1)
147
148 print "Poly"
149 testSVM(svm_poly,testing_0,testing_1)
```

Discussion



- How to apply SVM for more than two classes?

Solution



- $g_1(x)$: For separating squares from all.
- $g_2(x)$: For separating triangles from all.
- For classificiation, in case of ambiguity use if $g_1(x) > g_2(x)$, use $g_1(x)$ otherwise $g_2(x)$.

Libraries

- Sklearn uses libsvm. Libsvm is not Python specific.
- Other libraries:
  `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`
- Example of usages: `http://chicago05.mlss.cc/tiki/tiki-read_article.php?articleId=2`

Resources

- Non-linear SVM Kernels 1:
  https://www.youtube.com/watch?v=HwQQXs4Nyjo
- Non-linear SVM Kernels 2:
  https://www.youtube.com/watch?v=LHzqW0EolzE
- http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf

References