Support Vectors

Machine Learning

Motivation

- Can you recognise handwritten digits?

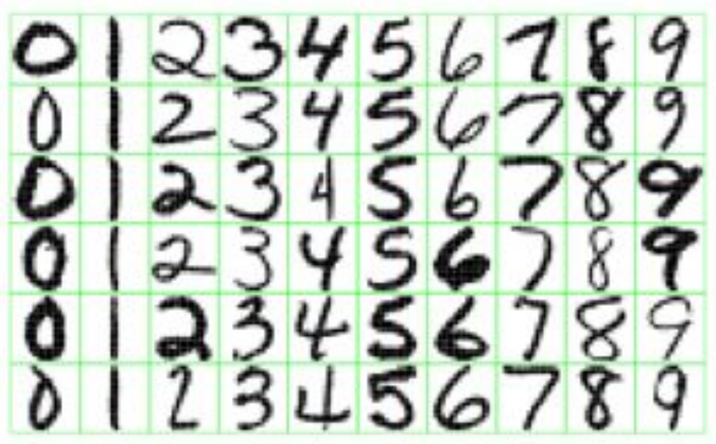


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

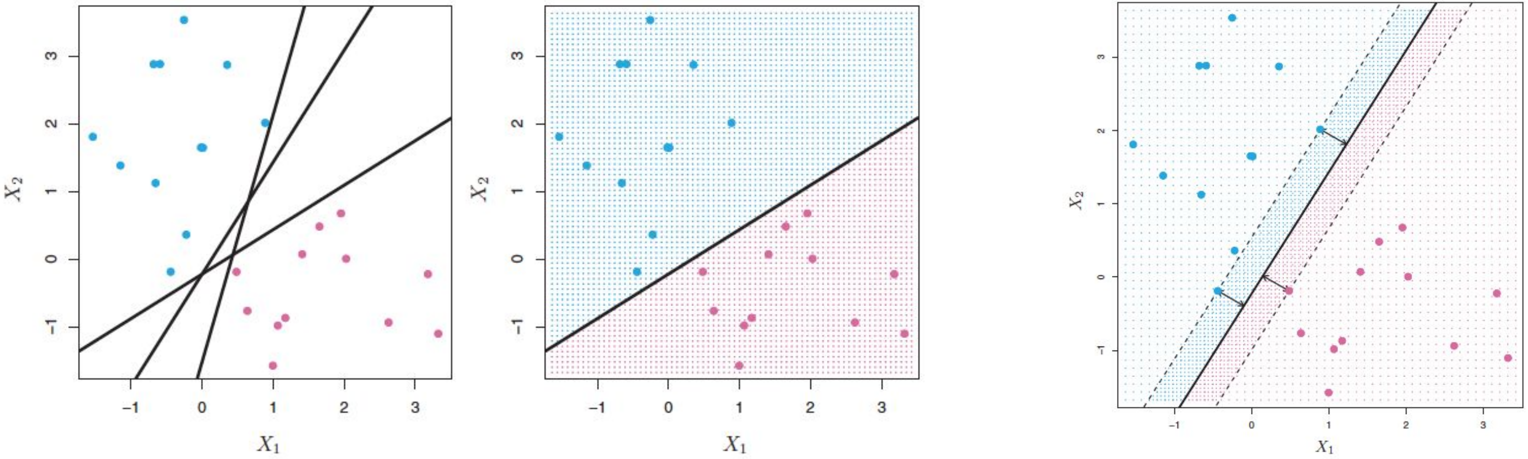
- Let’s first discuss a popular classifier: Support Vector Machines

Why Support Vectors?

- Pros
  - Memory and computationally inexpensive because it uses a subset of data
  - Easy to interpret results
  - Makes few errors
- Cons
  - Sensitive to configuration and outliers — needs small-ish dataset
  - Natively sensitive to class imbalances
  - Inherently expects pairs of classes
- Other stuff
  - Popular implementations in scikit-learn and (highly optimized) libsvm
  - Some division of opinion: considered old by some, powerful by others

Drawing a Line

- Assume: linearly separable classes
- Goal: Find a *hyperplane* (decision boundary with p-1 dimensions)
  - Of the infinite possible hyperplanes, find the one with the largest width
  - Maximize the margin: gap between hyperplane and nearest observation



Class Labels & Hyperplane

- Two classes (1, -1) — assumes points equidistant from margin
- Hyperplane defined by:

$$f(x) = \omega^{T}x + \mathrm{b}$$

… where:

ω^{T} is a vector of weights

x is the input data

b is an offset weight

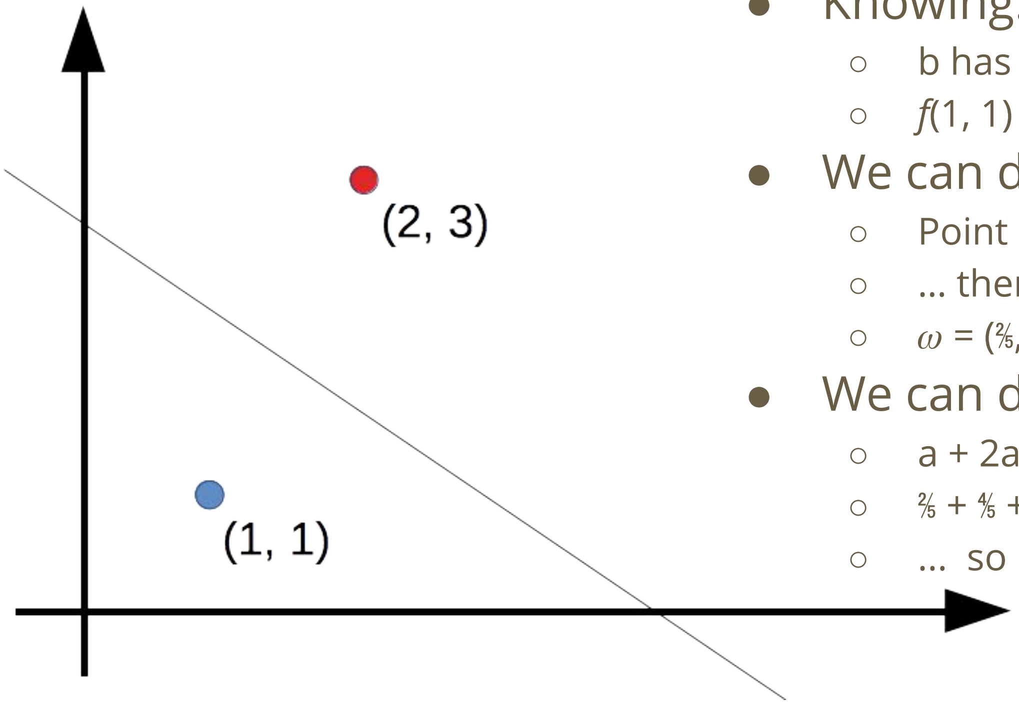
- Goals:
  - Find support vectors — “extreme” instances close to frontier but not in (or past) margin
  - $f(\mathrm{x}) \geq 1$ for all support vectors in class 1; $f(\mathrm{x}) \leq 1$ for all support vectors in class 2
  - Maximise the margin [$z = f(x) / \|\omega^{T}\|$] by minimising weight vector (ω^{T})

Two Points ⇒ Hyperplane (1)

- Goal: find the values for ω, b
- Using 2 points:
  - (1, 1) to class 1 ⇒ $f(1, 1) = -1$
  - (2, 3) to class 2 ⇒ $f(2, 3) = 1$
- Plane is equidistant from points:
  - $\omega = (2, 3) - (1, 1) = (a, 2a)$
  - $f(1, 1) = -1 = a + 2a + b$
  - $f(2, 3) = 1 = 2a + 6a + b$
- So b has the value [1 - 8a]

From Thales Sehn Korting's "How SVM (Support Vector Machine) algorithm works"

Two Points ⇒ Hyperplane (2)



- Knowing:
  - b has the value [1 - 8a]
  - $f(1, 1) = a + 2a + b = -1$
- We can determine ω:
  - Point (1, 1) = a + 2a + 1 - 8a = -1
  - … therefore a = ⅖
  - ω = (⅖, ⅘)
- We can determine b:
  - a + 2a + b = -1
  - ⅖ + ⅘ + b = -1
  - … so b = -11/5

From Thales Sehn Korting's "How SVM (Support Vector Machine) algorithm works"

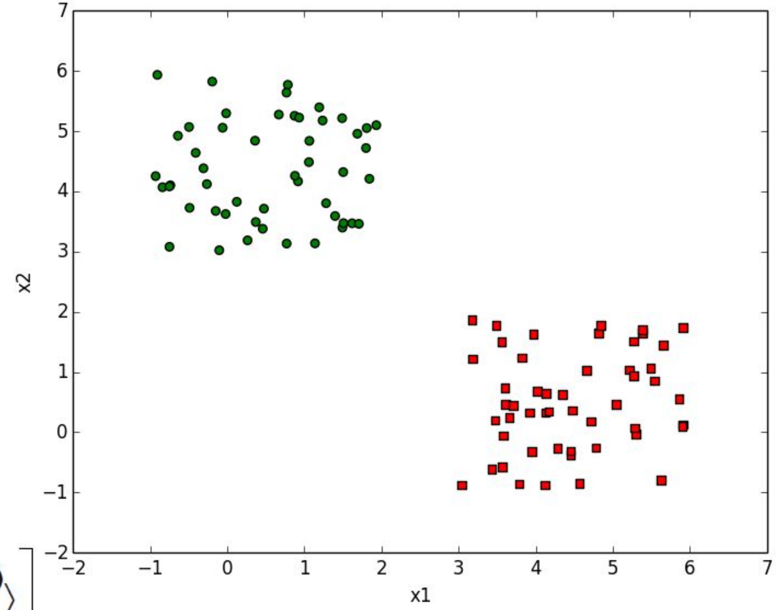
More Than Two Points?

- Use support vectors (α = “alphas”)
  - … i.e. instances closest to the margin
  - … to maximise margin, given by:

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

  - Problem: this is difficult to optimise
- Solution:
  - Constrained Optimisation
  - Set [label * $\omega^{\mathrm{T}}x$ + b] to 1*

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



Slack Variables

- Problems:
  - What if an instance appears within the margin?
  - Worse, what if an instance appears on the wrong side of the margin? (Data is not always linearly separable)
- Solution: Further constrain the solution to exclude these problem points

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

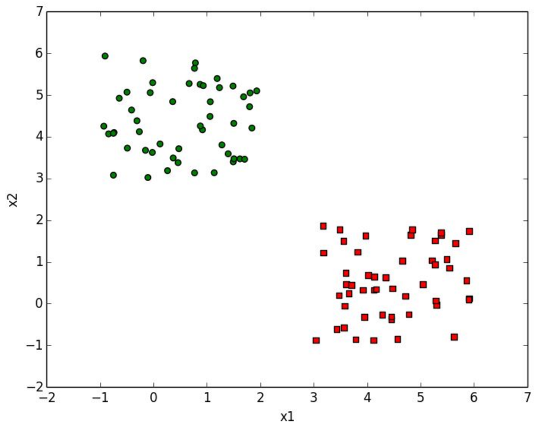
  - "c" controls weighting between two goals:
    - Ensuring most instances have a margin of 1.0 or greater
    - Making the margin large

SMO

- Platt, 1996: SMO (Sequential Minimal Optimization)
    - Was it conceived before?
    - Breaks optimisation into several smaller problems, each with relatively easy solutions
- Basic algorithm
    - Choose a (the best?) pair of alphas
        - Alphas must be outside margin
        - Alphas cannot be “clamped” or “bounded”
    - Determine b
    - Use alphas & b to find weights
    - Harrington’s implementation is on github, here
- Advanced
    - Heuristics to select good candidates for alpha pairs
    - Classes for holding data structures

Lab 2

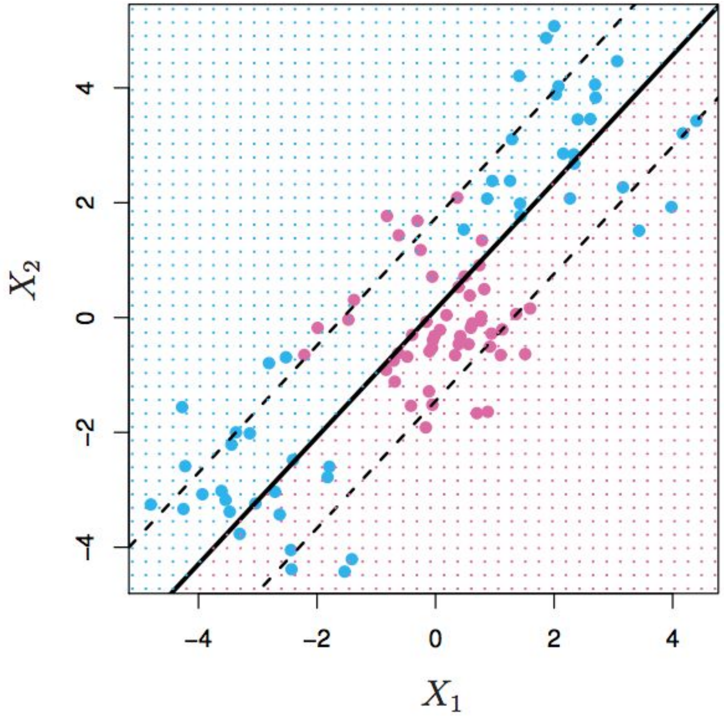
- Get Harrington's SMO implementation (**Non-Kernel version**)
  - https://github.com/pbharrin/machinelearninginaction3x/blob/master/Ch06/svmMLiA.py
  - Note that Harrington conflates loading the data (loadDataSet) with the classifier; you should NOT do this
- Get the data
  - Download from linearly_separable.csv
  - Create a figure to ensure classes are linearly separable, eg:



- Change Harrington's code to a classifier
  - Implement "fit" and "predict"
  - Use defaults for C: 1.0; toler: 0.001; maxIter: 50
  - Ignore

Nonlinearities and Kernels

- Problem: linear separation
  - Sometimes class separations would give poor performance
  - Tuning "C" would not help in these cases
- Solution:
  - Move to a higher-dimension space
  - Scalar from two (feature) vectors <x_i, x_i'>
  - Linear support vector classifier —>
    - Adds non-linear kernels (Polynomial and RBF = radial basis function)
    - Example: d-dimensional polynomials



SVM vs. Logistic Regression

- Options for $K > 2$ classes
  - OVA (one vs. all): fit K different 2-class SVM classifiers; classify x^* to the class which generates the max score
  - OVO: (one vs. one): Fit all K choose 2 pairwise classifiers; classify x^* to the class which wins the most pairwise competitions
- When classes are (nearly) separable, SVM generally performs better
- When classes are not linearly separable, both behave similarly
  - LR must have *ridge penalty*
  - Need to estimate probabilities? LR is the better choice
- SVMs are popular for non-linear boundaries
  - Implemented as kernels
  - Can also apply kernels ot LR or LDA… but don’t

Next Time

- Handwritten digits practical