Portfolio Question 1.2.2 - Classifiers

Daniel Lawson University of Bristol

Portfolio 01.2.2 (v4.0.0)

Portfolio Example Question

- Portfolio Question 1.2.2: What are the most famous classifiers, and which are competitive today?
  - We will cover trees and neural networks so focus on classical classifiers

Context

- We saw Logistic regression and k-NN
- We're talking about classifiers
- Got code from XXX

Some Important Classifiers

- Provided:
  - **Logistic Regression**
  - **K-Nearest Neighbours**
- Old:
  - **Linear Discriminant Analysis**
  - **Support Vector Machines**
- Trees
  - **Decision Trees**
  - **CART**: Classification and Regression Trees
  - **Random Forests**
  - **Gradient Boosting Trees**
- Other:
  - **Neural Networks**
  - **Naive Bayes**

Summary

- Will explain LDA and SVM
- Others are covered later in the course:
- **Naive Bayes:**
  - Assumes features independent, apply Bayes rule
  - Used in text mining due to sparse features
  - Simple baseline, otherwise not competitive
- Trees:
  - Decision trees are simple and interpretable
  - Boosted Trees and Random Forests are competitive extensions

Linear Discriminant Analysis

- Developed in **1936 by R. A. Fisher**[1] and extended to the current multi-class form in 1948[2].
- The goal is to **project** a high dimensional space into K dimensions, **maintaining** (linear) classification ability.
- Prediction benefit comes only from reducing overfitting
- Strong **relationship with PCA**, often used in tandem (PCA then LDA)
- Assumes that each class k has a different mean μ_k and a shared covariance matrix Σ
- Kernel Discriminant Analysis exists[3]

---

[1] Fisher R, "The Use of Multiple Measurements in Taxonomic Problems" (1936) Annals of eugenics (!), now "Annals of Human Genetics"

[2] Rao C, "Multiple Discriminant Analysis" (1948) JRSSB

[3] Mika, S et al "Fisher discriminant analysis with kernels" (1999) NIPS IX: 41-48

LDA algorithm

1. Compute the mean location μ_k for each class k and the overall mean μ, as well as the assignment sets D_k.
2. Compute the **within-class scatter matrix** S_W: $S_W = \sum_{k=1}^{K} S_k$ where

$$S_k = \sum_{i \in D_k} (\vec{x} - \vec{\mu}_k) (\vec{x} - \vec{\mu}_k)^T$$

3. Compute the **between-class scatter matrix** S_B:

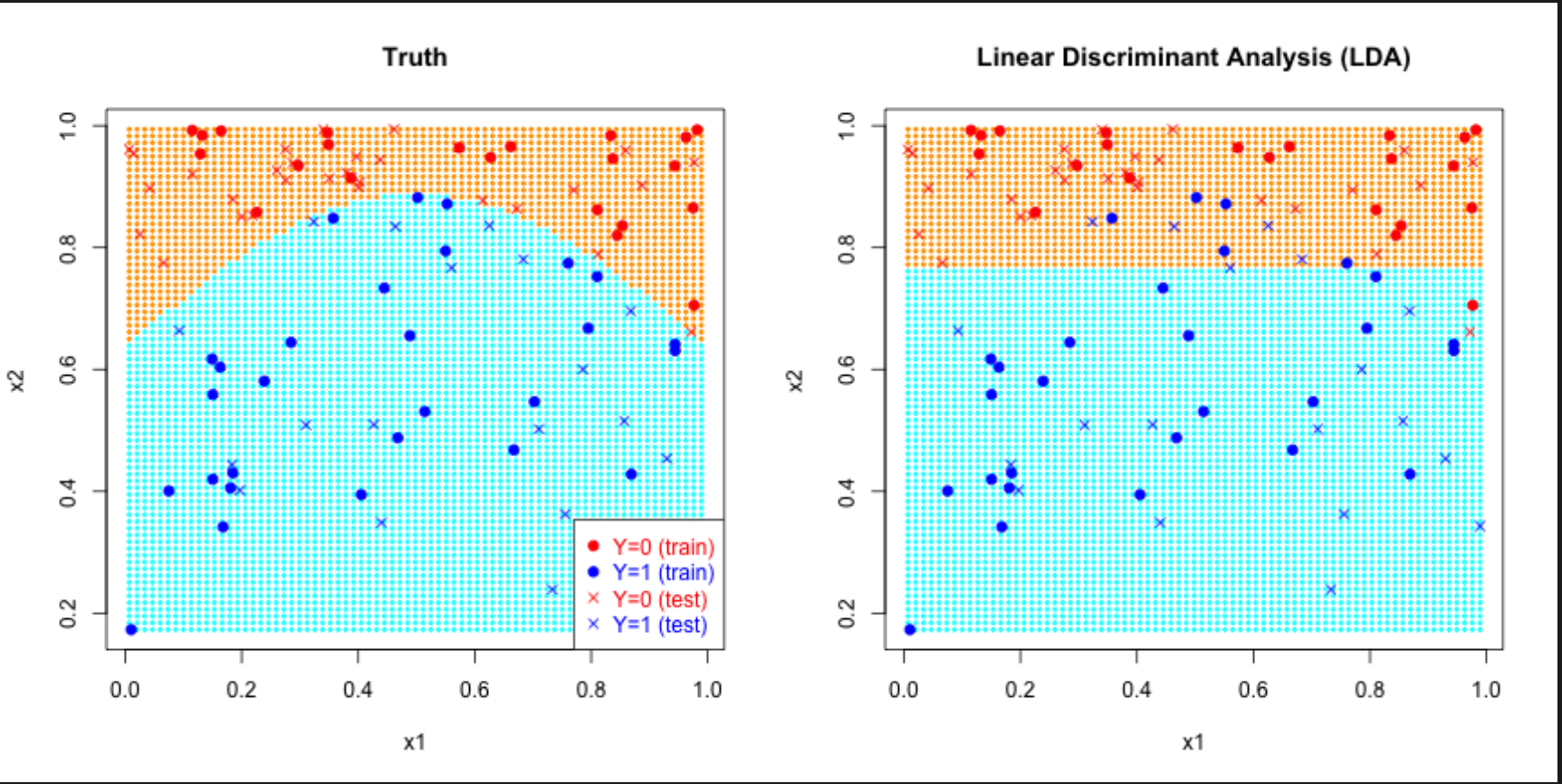
$$S_B = \sum_{k=1}^{K} n_i (\vec{\mu}_k - \vec{\mu}) (\vec{\mu}_k - \vec{\mu})^T$$

4. Solve for the **eigenvalues** λ_k and **eigenvectors** v_k of $S_W^{-1} S_B$
5. **Choose a dimension** threshold K^*, either using the same methods as for PCA, or cross-validation
6. **Predict** using μ_k ...

LDA prediction

- Class prediction can use any information in the LDA data summary. Options include:
  - Nearest cluster
  - **Likelihood**: $\Pr(\vec{x}|y_k = c) = \text{Normal}\,(\mu_k, \Sigma)$
  - **Posterior**: $\Pr(y_k = c|\vec{x}) \propto \Pr(\vec{x}|y_k = c)p(y_k = c)$; i.e. reweight classes according to their frequency

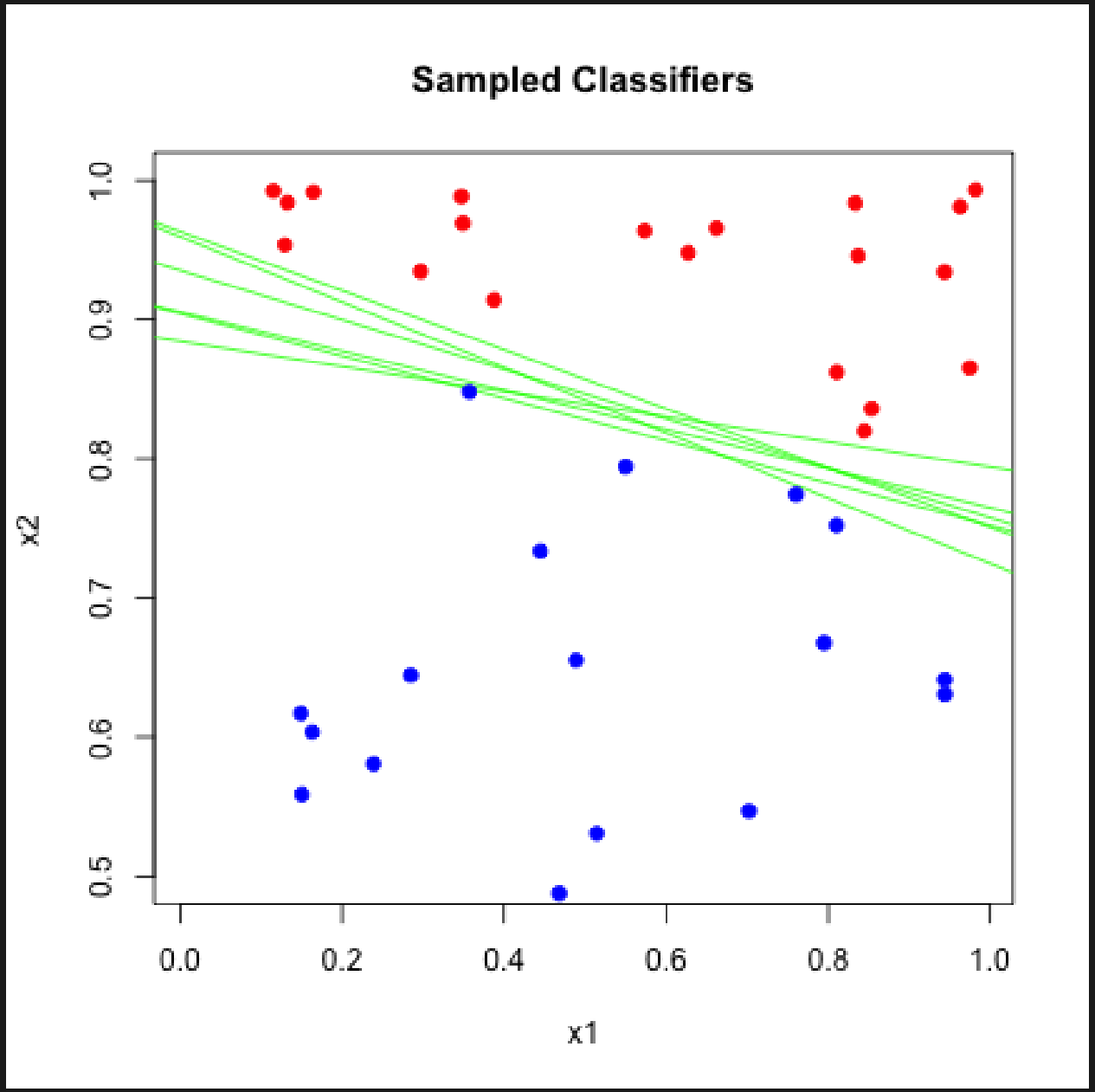
LDA example


Truth
Linear Discriminant Analysis (LDA)
x1
x2
0.0
0.2
0.4
0.6
0.8
1.0
Y=0 (train)
Y=1 (train)
Y=0 (test)
Y=1 (test)

Support Vector Machine overview

- Support Vector Machines (SVM) choose a decision boundary to maximally separate the classes
  - Uses Quadratic Programming, a widely useful tool
- Find the **maximum margin hyperplane** separating the classes **closest** points
- Allow soft margins: misclassified points are down-weighted
- Nonlinearity: express distances as **inner products**, allowing non-linearities via the "Kernel trick" (covered later)
- Algorithm: finding the hyperplane is a "quadratic optimisation problem".

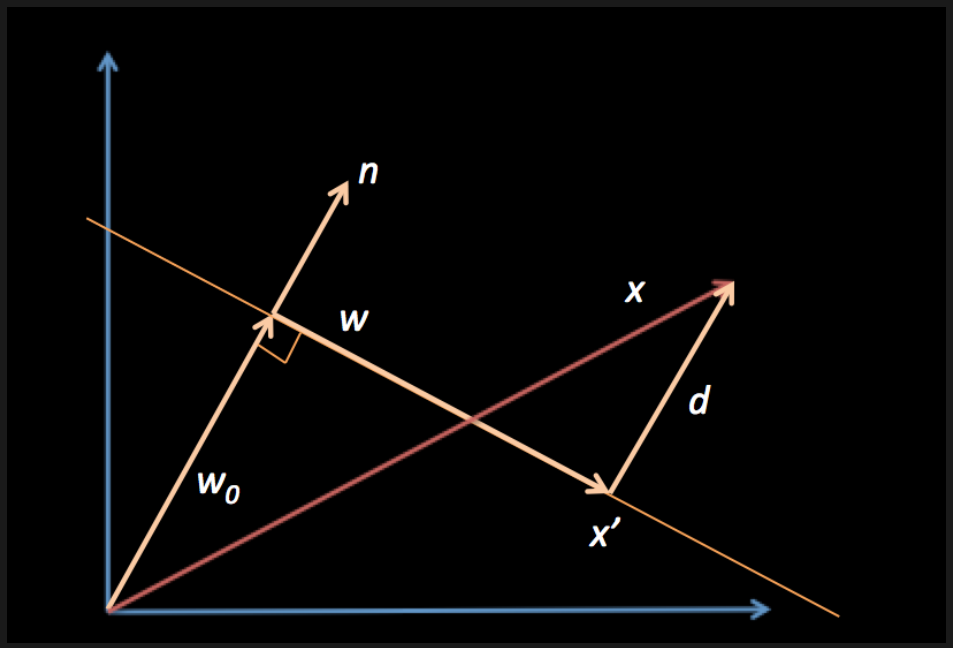
SVM illustration: solution space



Planar geometry

- The data are $\vec{x} \in D$ containing N examples
- The labels are $y_i \in (-1, 1)$
- A **hyperplane** is defined via:
  - $\vec{w}$, the coordinates of the plane
  - $\vec{w}_0$, a point on the plane chosen such that $\vec{w}_0$ is perpendicular to $\vec{w}$:

$$\vec{w} \cdot (\vec{x} - \vec{w}_0) = \vec{w} \cdot \vec{x} + b = 0$$



SVM margins

- The **distance of a point to the line** is the residual after the point is projected onto the line:

$$d_{\vec{w}}(\vec{x}) = \vec{n} \cdot (\vec{x} - \vec{x}') = \frac{|\vec{w} \cdot \vec{x} + b|}{|\vec{w}|}$$

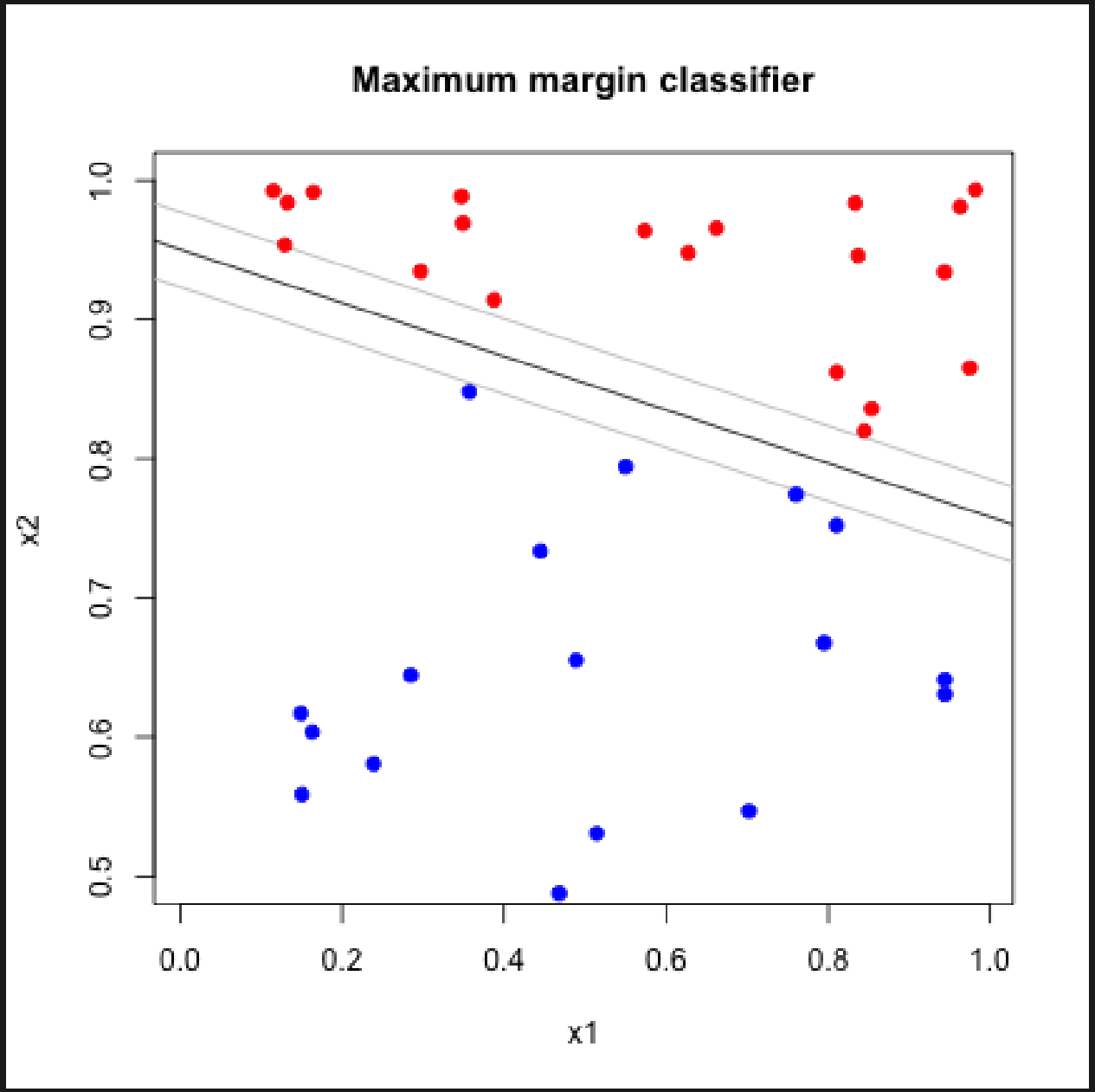
- For a given hyperplane, the **minimum margin** is

$$M_{\vec{w}} = \text{argmin}_{x \in D} d_{\vec{w}}(\vec{x})$$

- The **maximum margin hyperplane** is therefore:

$$\text{argmax}_{\vec{w}} \text{argmin}_{x \in D} d_{\vec{w}}(\vec{x})$$

SVM illustration: SVM solution



Computing the margins

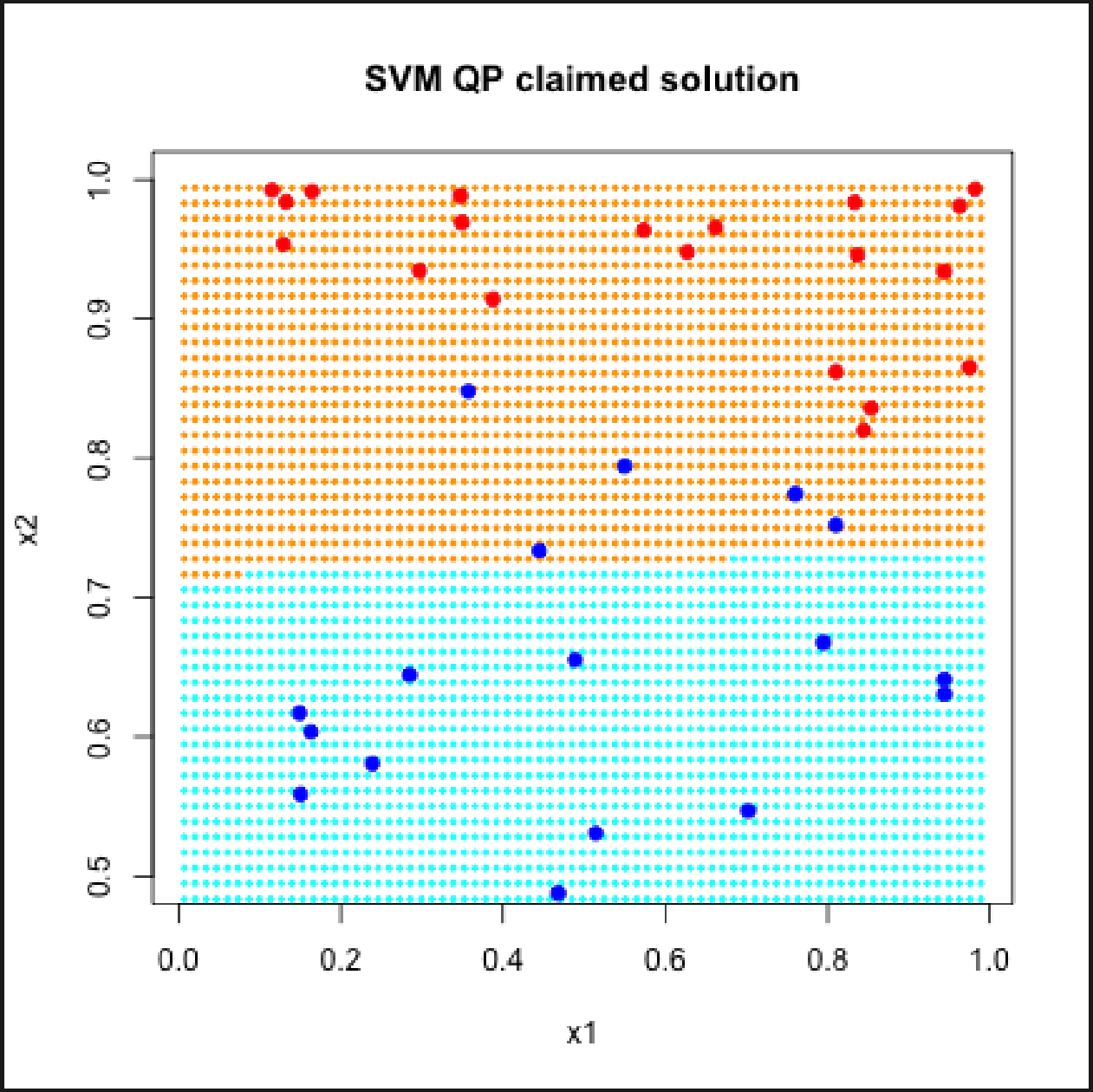
- This is a classic **Quadratic Programming** problem[4]
- Broadly:
  - quadratic penalty: distance to the plane $\propto$ squared norm of the hyperplane vector $\frac{1}{2}|\vec{w}|^2$
  - linear inequalities: none of the data are closer than $M_{\vec{w}}$. So $\forall i : y_i(\vec{w} \cdot \vec{x} + b) \geq 1$
- and pass these to a standard QP solver
- A computational trick: only evaluate the points on the margins

[4] For this course, you need to know what QP can do for you. You don't need to know how it works.

SVM problem: local optima

- Sometimes it gets stuck in a local optima
- How would we know this happened if we weren't able to visualise?
- Visualisation follows:

SVM problem: local optima



Imperfect classification with SVM

- To account for data the **wrong side of the margins**, the penalty is changed to:

$$\frac{1}{2}\left|\vec{w}\right|^2 + C\sum_{i=1}^{N}\epsilon_i$$

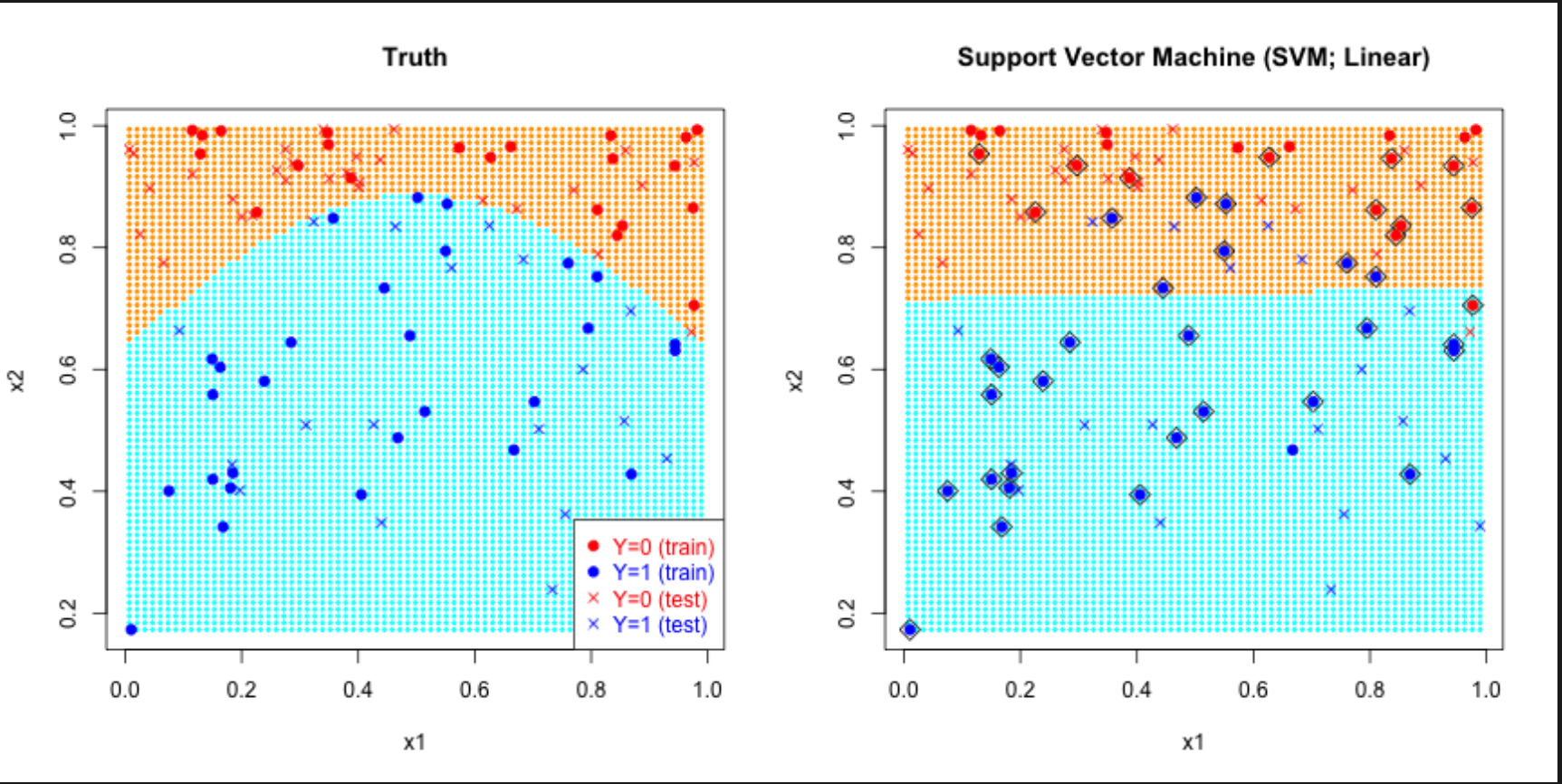
- where ϵ_i is the "distance" needed to move the point to the correct decision boundary, i.e.

$$\vec{w}\cdot\vec{x}_i + b \geq 1 - \epsilon_i \qquad \text{if :} \qquad y_i = 1 \quad (1)$$

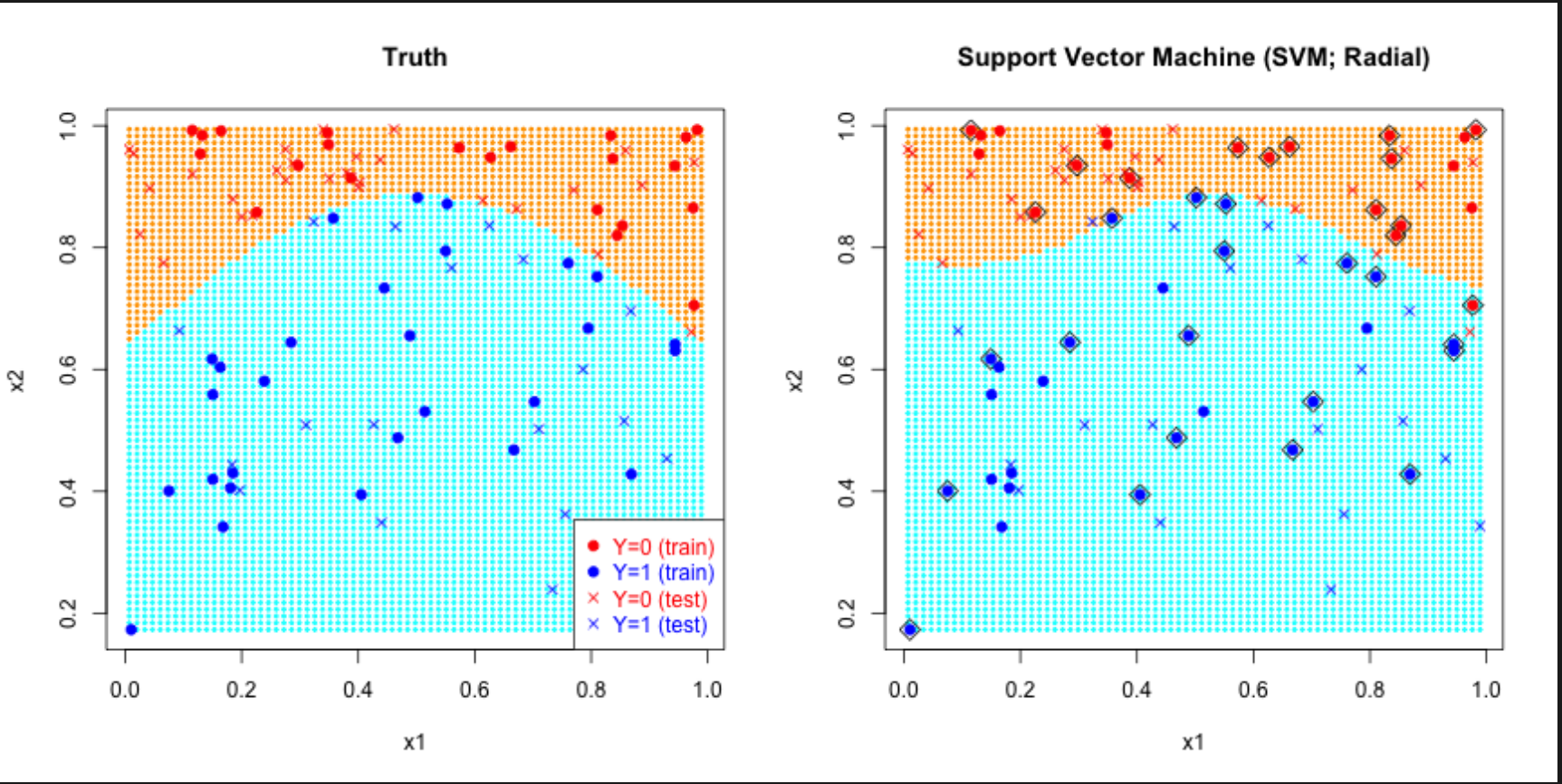
$$\vec{w}\cdot\vec{x}_i + b \leq -1 + \epsilon_i \qquad \text{if :} \qquad y_i = -1 \quad (2)$$

- and $\epsilon_i = 0$ if already inside it, so also requiring the constraint $\epsilon_i \geq 0$

SVM example


Truth
Support Vector Machine (SVM; Linear)
x1
x2
0.0
0.2
0.4
0.6
0.8
1.0
Y=0 (train)
Y=1 (train)
Y=0 (test)
Y=1 (test)

kernel SVM example



Competitiveness?

- kNN regarded as a good baseline
- Boosting and Trees regarded as competitive
- Linear approaches are only under special circumstances (low data)
- Kernel approaches hve n^2 cost, so are important when the dataset is not huge