Classification: SVM

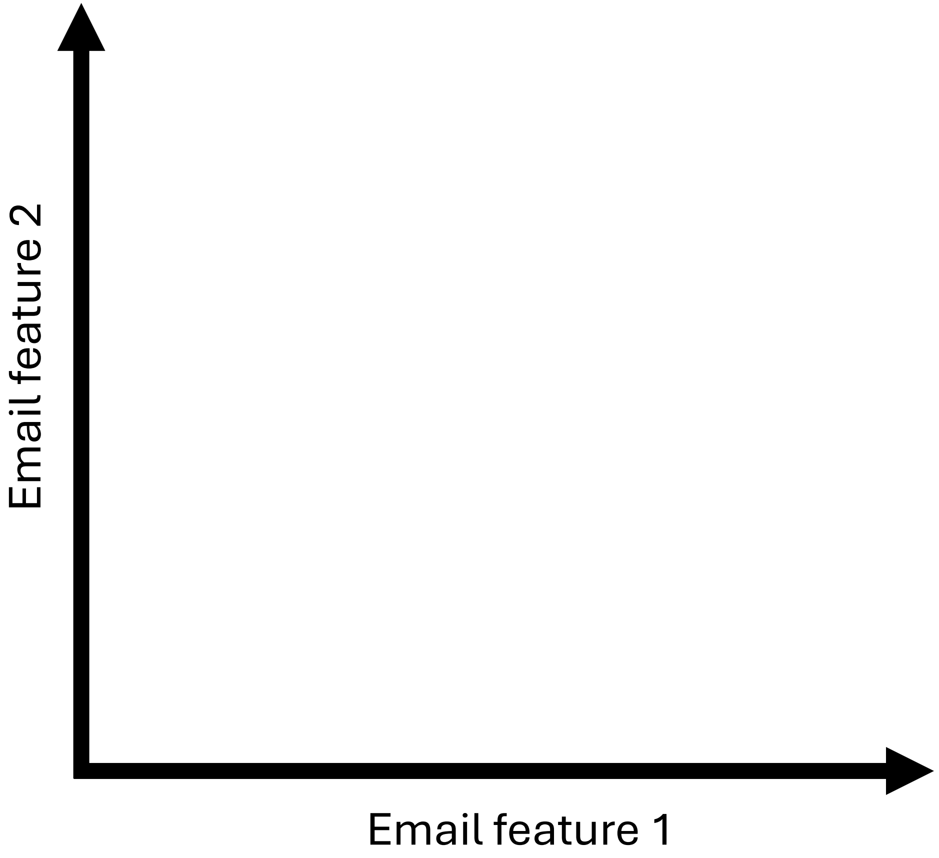
Intro to ML, 2024

Rahul Shome
School of Computing
Australian National University

Based on content from UT Austin, Cornell, scikit-learn.

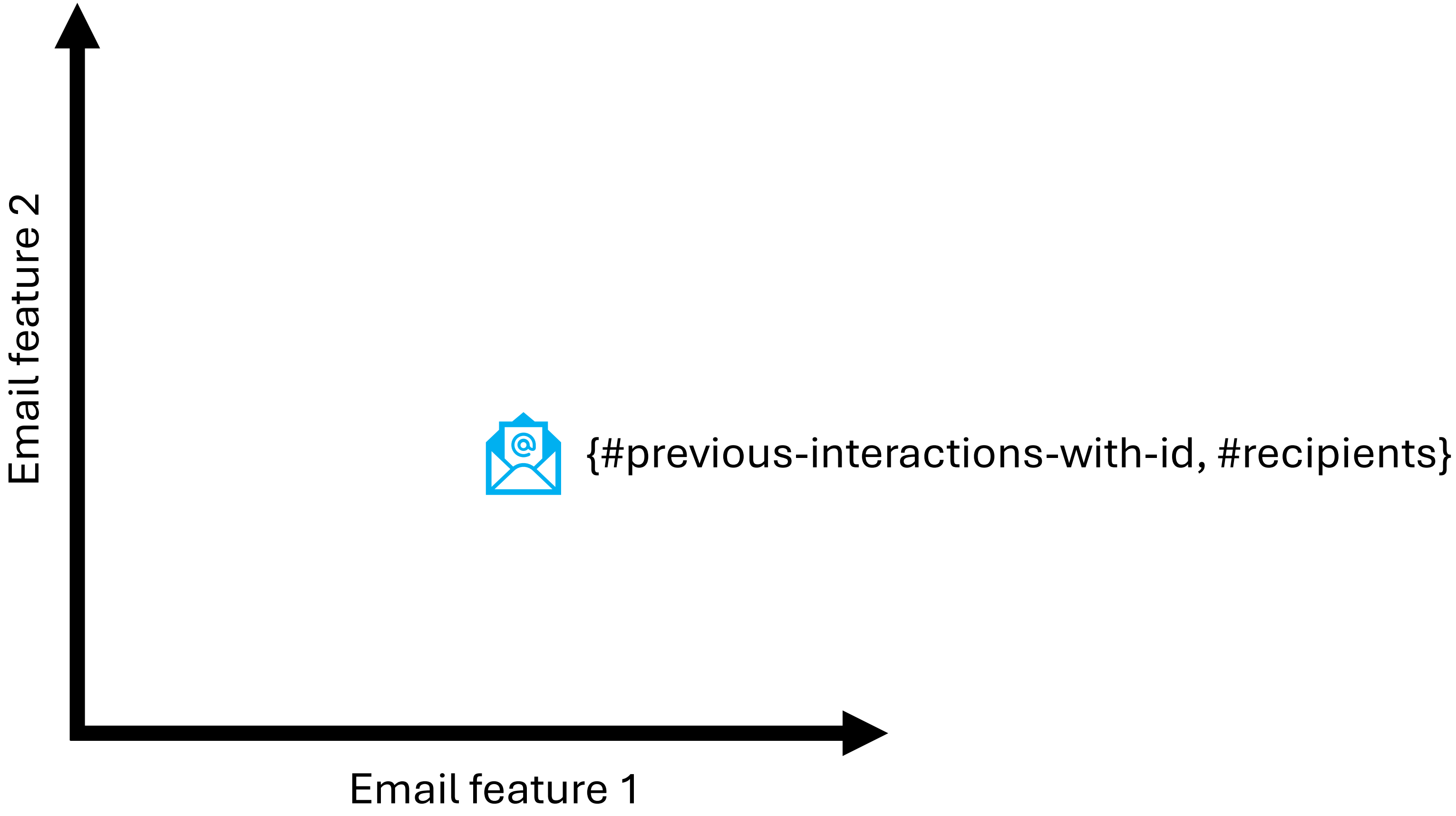
Classification

- Predict label of new data from features of data.



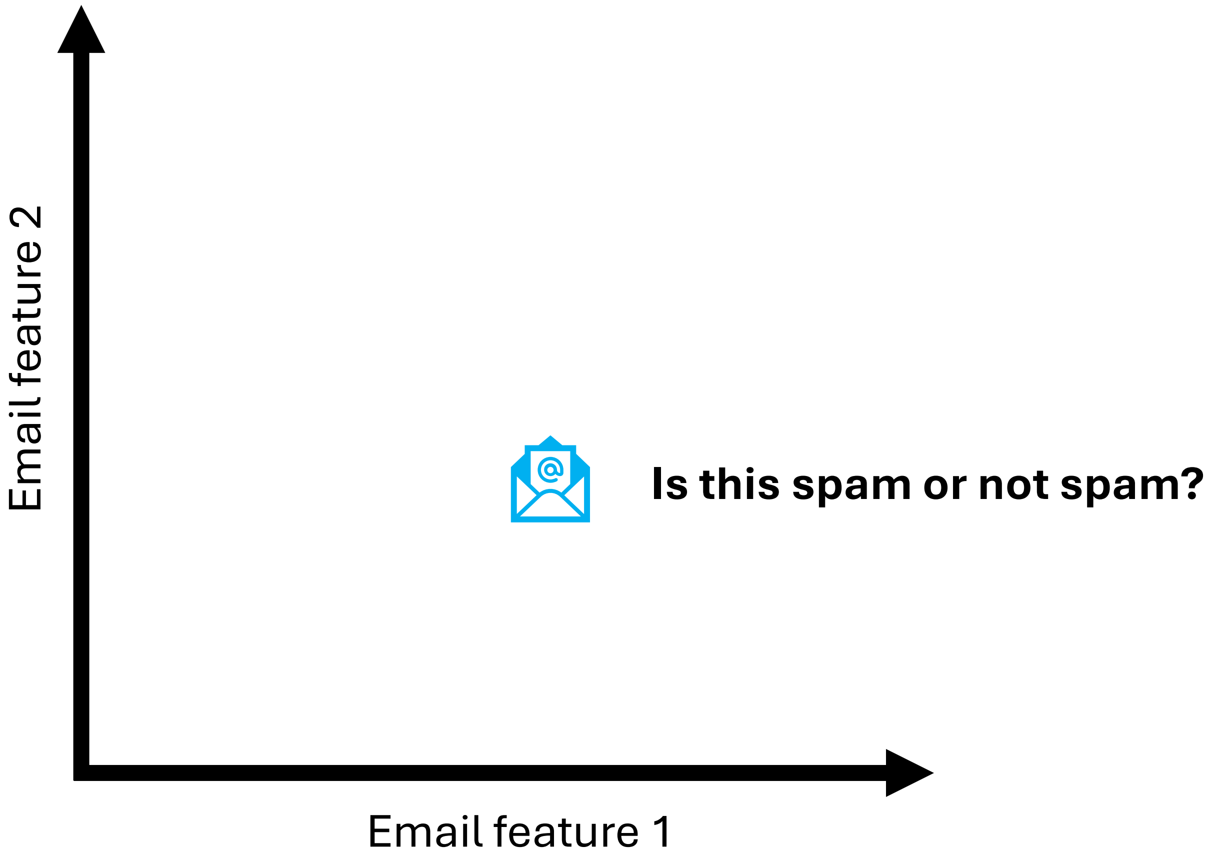
Classification

- Predict label of new data from features of data.



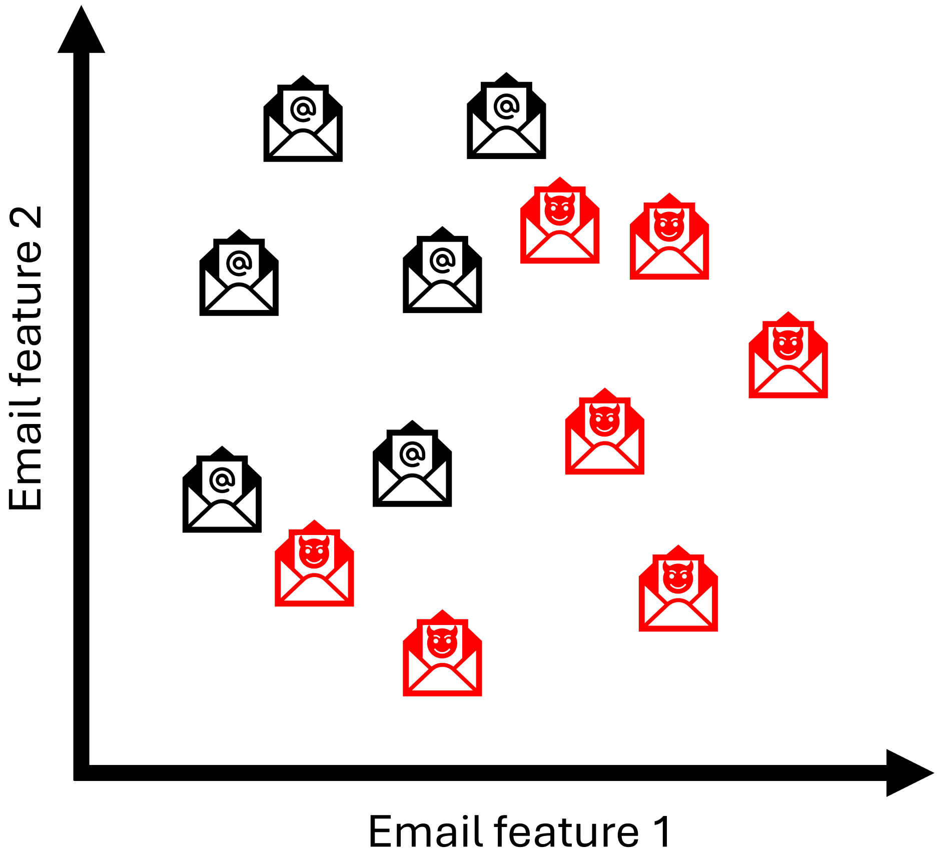
Classification

- Predict label of new data from features of data.



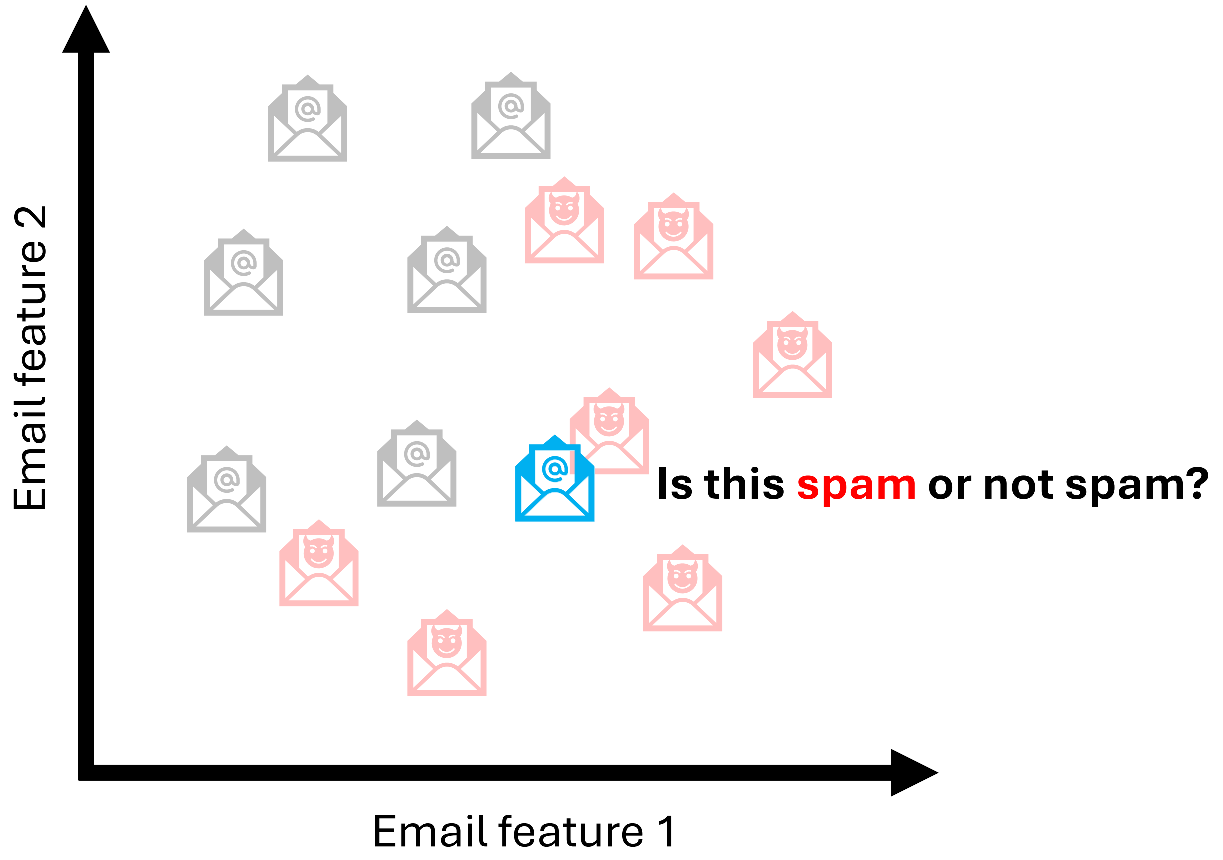
Classification

- Learn a classifier from training data



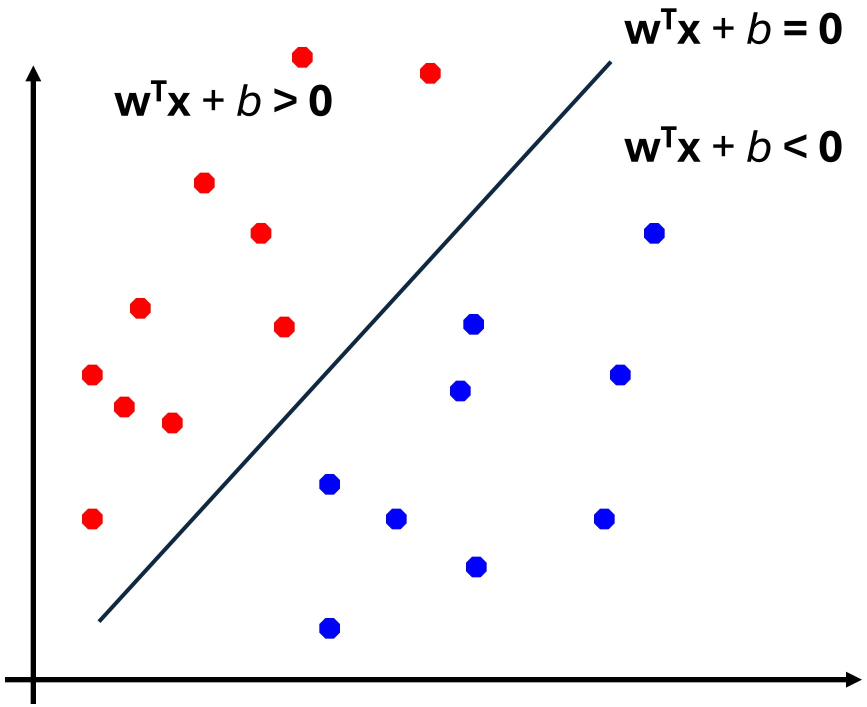
Classification

- Use the learned model to predict label



Linear Separators

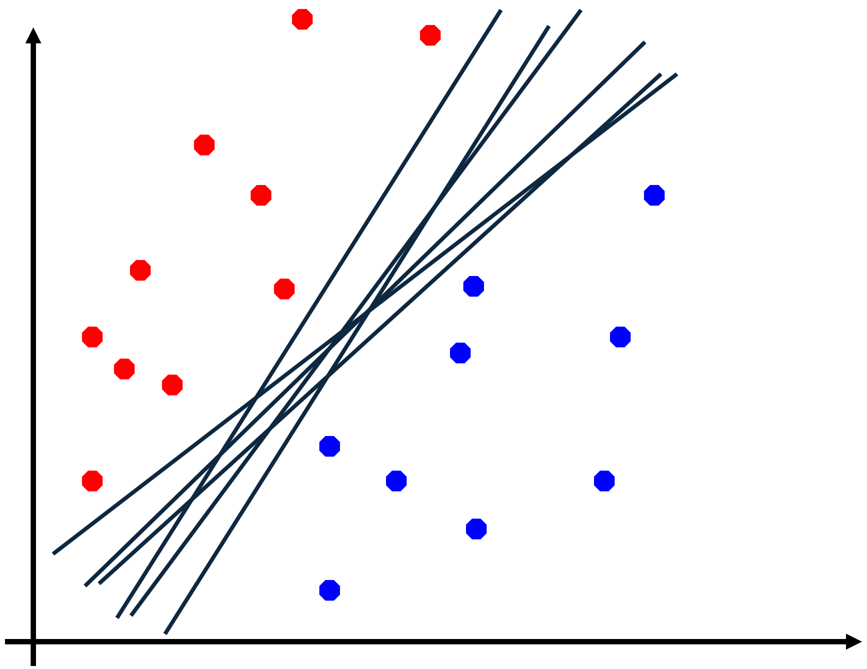
- Binary classification can be viewed as the task of separating classes in feature space:



$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$$

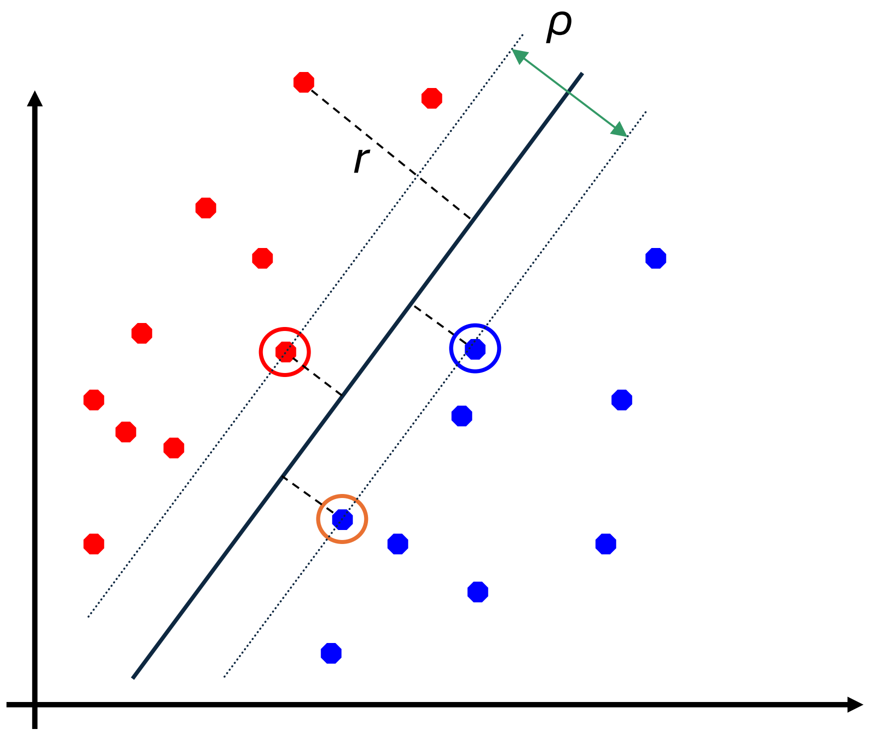
Linear Separators

- Which of the linear separators is optimal?



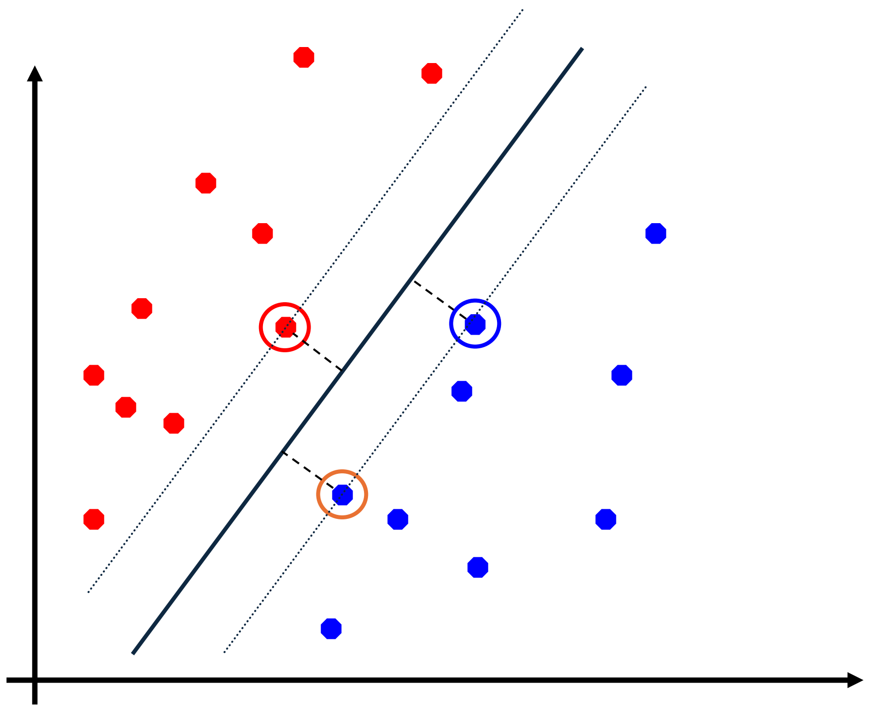
Margin

- Distance from example $\mathbf{x}_i$ to the separator is $r = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are ***support vectors***.
- ***Margin*** ρ of the separator is the distance between support vectors.



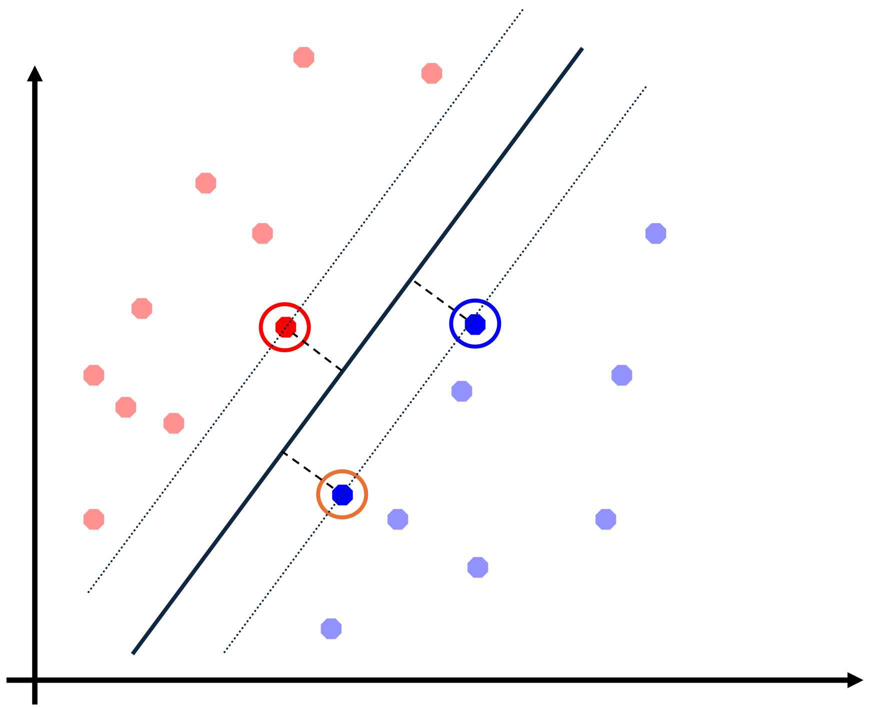
Maximum Margin

- Maximizing the margin is good according to intuition and PAC theory.
- Implies that only support vectors matter; other training examples can be ignored.



Maximum Margin Classification

- Maximizing the margin is good according to intuition and PAC theory.
- Implies that only support vectors matter; other training examples **can be ignored**.



Linear SVM

- Let training set $\{(\mathbf{x}_i, y_i)\}_{i=1..n}$, $\mathbf{x}_i \in \mathbf{R}^d$, $y_i \in \{-1, 1\}$ be separated by a hyperplane with margin ρ. Then for each training example $(\mathbf{x}_i, y_i)$:

$$\begin{array}{ll} \mathbf{w}^\mathbf{T}\mathbf{x}_i + b \leq -\rho/2 & \text{if } y_i = -1 \\ \mathbf{w}^\mathbf{T}\mathbf{x}_i + b \geq \rho/2 & \text{if } y_i = 1 \end{array} \quad \Leftrightarrow \quad y_i(\mathbf{w}^\mathbf{T}\mathbf{x}_i + b) \geq \rho/2$$

- For every support vector $\mathbf{x}_s$ the above inequality is an equality. After rescaling **w** and b by $\rho/2$ in the equality, we obtain that distance between each $\mathbf{x}_s$ and the hyperplane is

$$r = \frac{y_s(\mathbf{w}^T\mathbf{x}_s + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

- Then the margin can be expressed through (rescaled) **w** and b as:

$$\rho = 2r = \frac{2}{\|\mathbf{w}\|}$$

Linear SVM

- Then we can formulate the *quadratic optimization problem:*

Find **w** and *b* such that $\rho = \dfrac{2}{\|\mathbf{w}\|}$ is maximized

and for all ($\mathbf{x}_i$, y_i), i=1..n : $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Which can be reformulated as:

Find **w** and *b* such that

$\boldsymbol{\Phi}(\mathbf{w}) = ||\mathbf{w}||^2 = \mathbf{w}^T\mathbf{w}$ is minimized

and for all ($\mathbf{x}_i$, y_i), i=1..n : $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Solving the Optimization Problem

Find $\mathbf{w}$ and b such that
$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $\quad y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Optimize a *quadratic* function subject to *linear* constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist.

Find $\alpha_1 \ldots \alpha_n$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

The Optimization Problem Solution

- Given a solution $\alpha_1 \dots \alpha_n$ to the dual problem, solution to the primal is:

$$\mathbf{w} = \Sigma \alpha_i y_i \mathbf{x}_i \qquad b = y_k - \Sigma \alpha_i y_i \mathbf{x}_i^{\mathbf{T}} \mathbf{x}_k \quad \text{for any } \alpha_k > 0$$

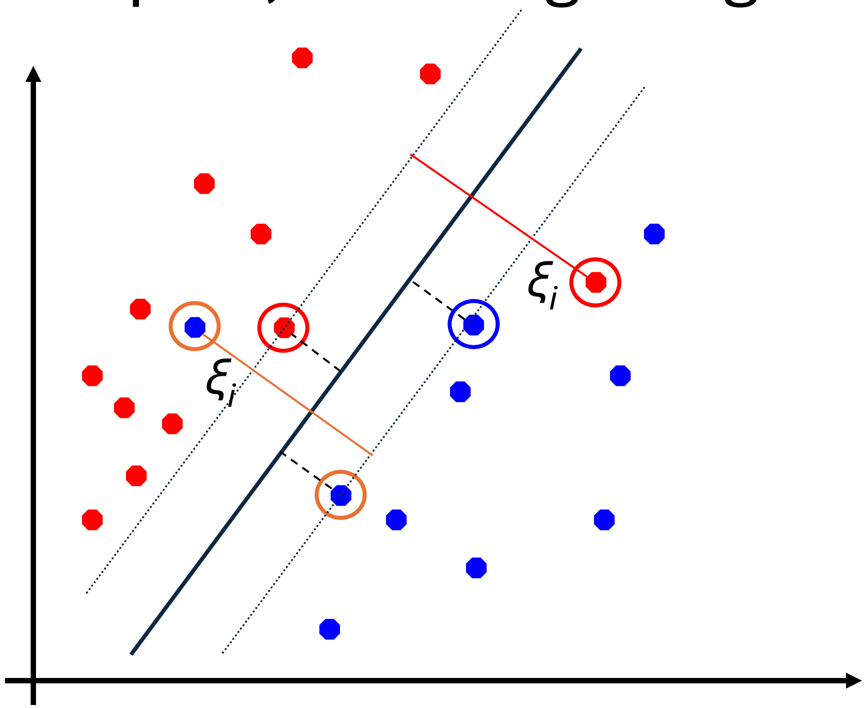
- Each non-zero α_i indicates that corresponding $\mathbf{x}_i$ is a support vector.
- Then the classifying function is (note that we don't need **w** explicitly):

$$f(\mathbf{x}) = \Sigma \alpha_i y_i \mathbf{x}_i^{\mathbf{T}} \mathbf{x} + b$$

- Notice that it relies on an *inner product* between the test point **x** and the support vectors $\mathbf{x}_i$. Solving the optimization problem involved computing the inner products $\mathbf{x}_i^{\mathbf{T}} \mathbf{x}_j$ between all training points.

Soft Margin Classification

- What if the training set is not linearly separable?
- *Slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called *soft*.



Soft Margin Classification Mathematically

- The old formulation:

Find **w** and b such that
$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Modified formulation incorporates slack variables:

Find **w** and b such that
$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$, $\xi_i \geq 0$

- Parameter *C* can be viewed as a way to control overfitting: it “trades off” the relative importance of maximizing the margin and fitting the training data.

Theoretical Justification for Maximum Margins

- Vapnik has proved the following:

  The class of optimal linear separators has VC dimension h bounded from above as

  $$h \leq \min\left\{\left\lceil \frac{D^2}{\rho^2} \right\rceil, m_0\right\} + 1$$

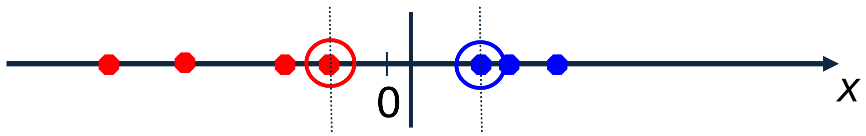
  where ρ is the margin, D is the diameter of the smallest sphere that can enclose all of the training examples, and m_0 is the dimensionality.

- Intuitively, this implies that regardless of dimensionality m_0 we can minimize the VC dimension by maximizing the margin *ρ*.

- Thus, complexity of the classifier is kept small regardless of dimensionality.

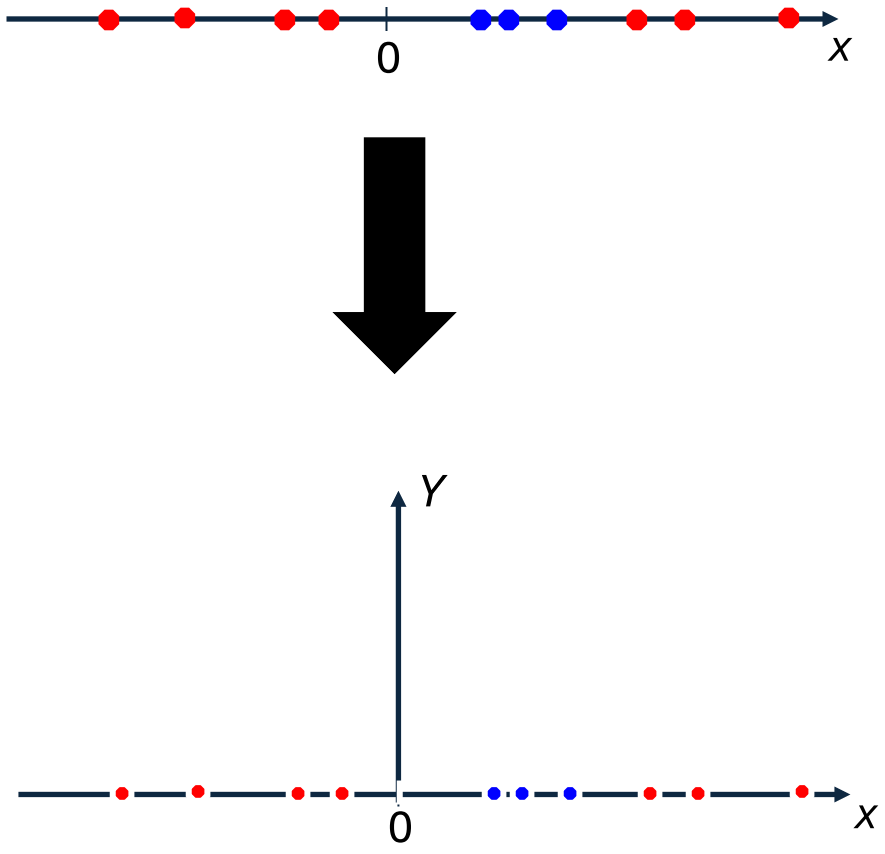
Beyond Linearly Separable Problems

- Datasets that are linearly separable with some noise.

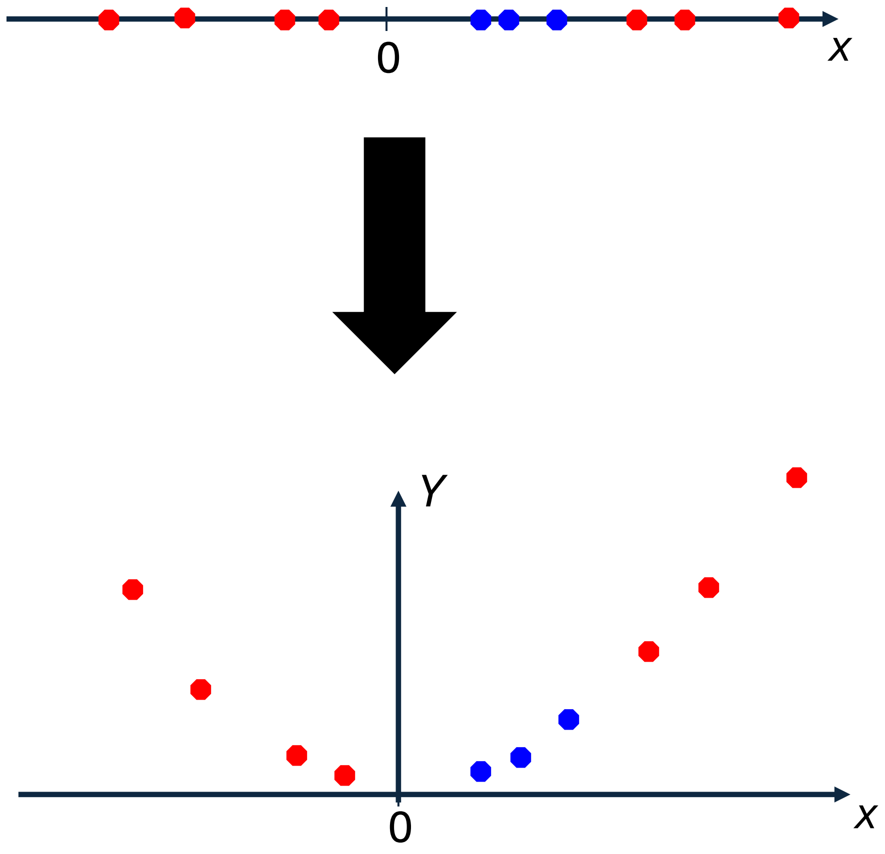


- Data might not be linearly separable.

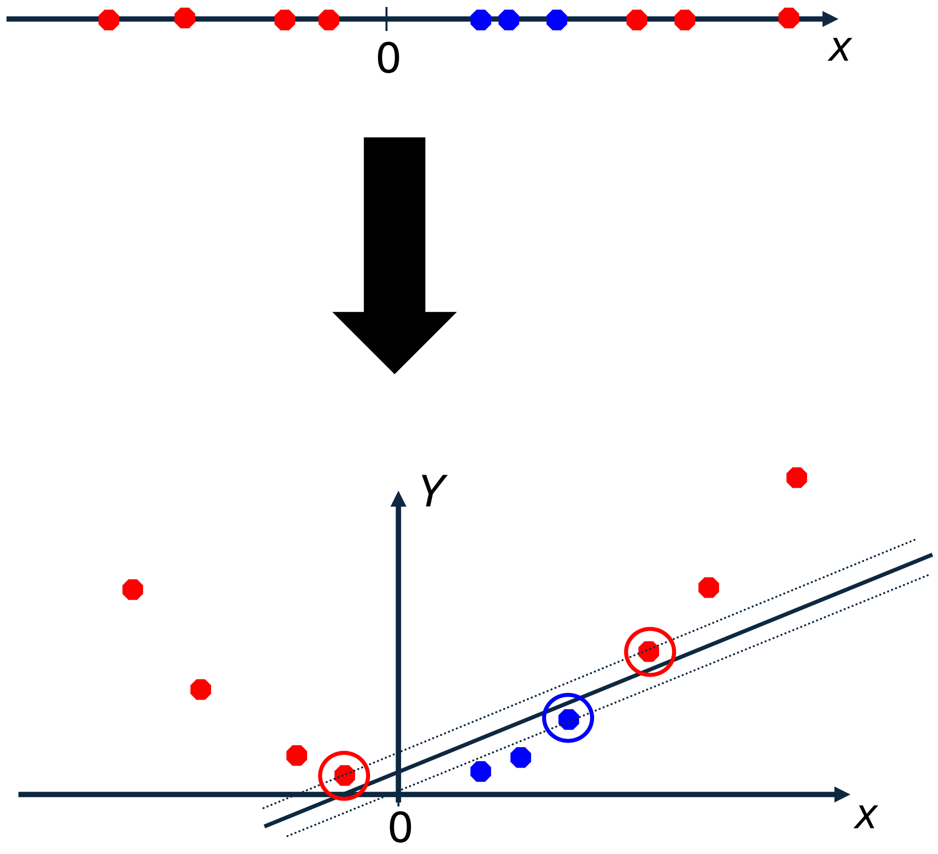
Beyond Linearly Separable Problems



Beyond Linearly Separable Problems

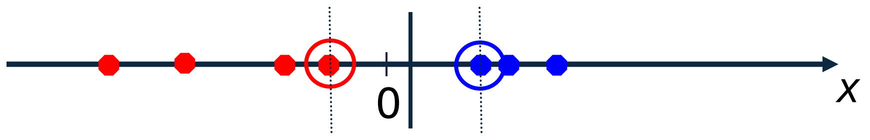


Beyond Linearly Separable Problems



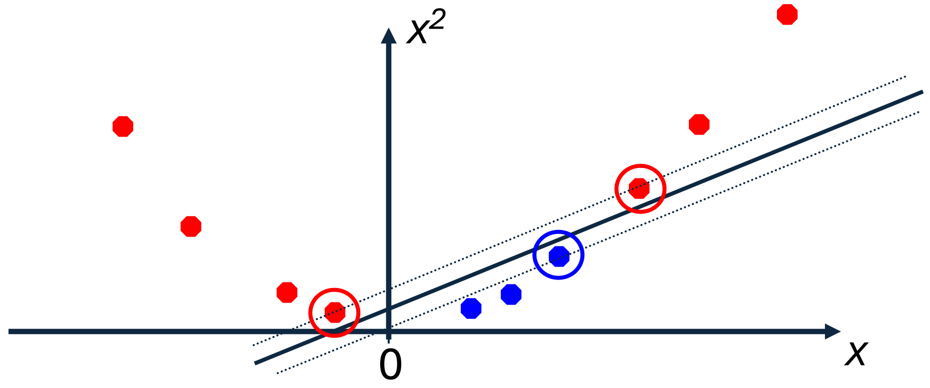
Beyond Linearly Separable Problems

- Datasets that are linearly separable with some noise.



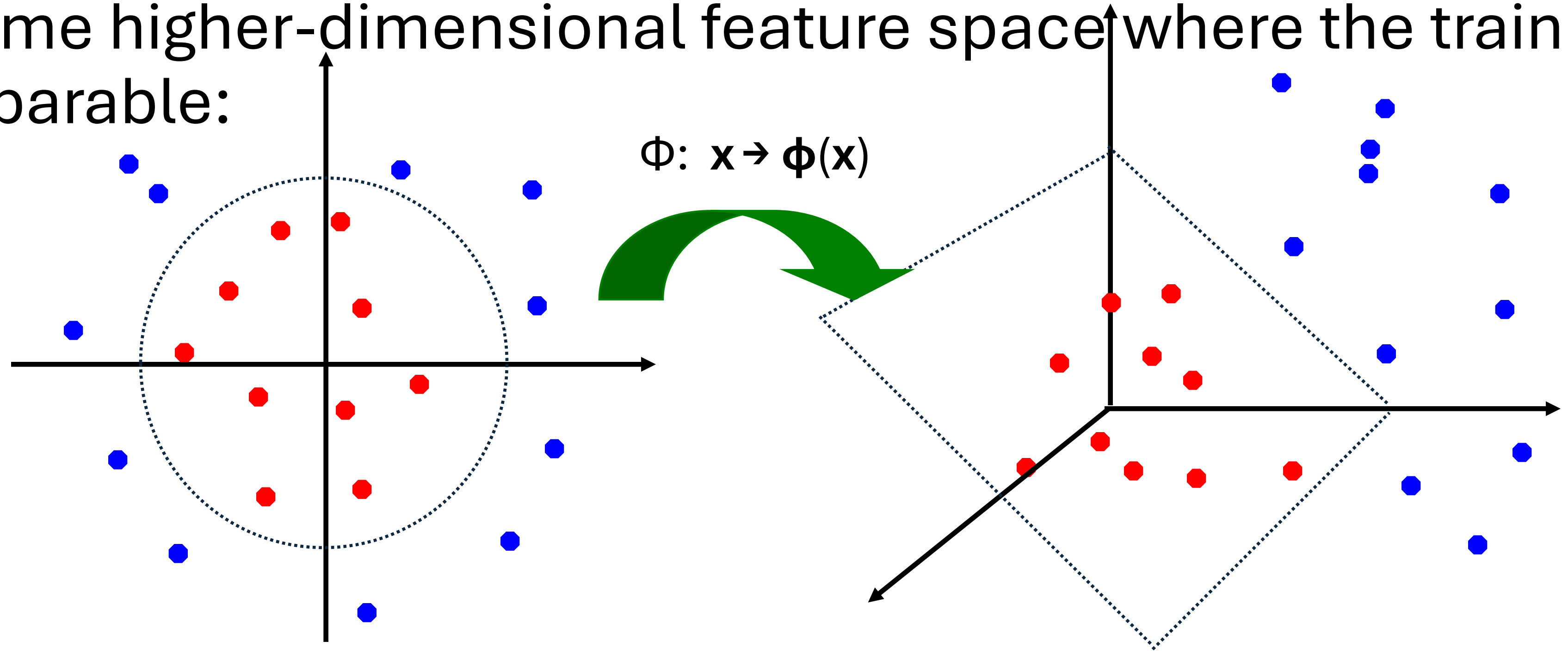
- Data might not be linearly separable.

- Map the data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



The “Kernel Trick”

- The linear classifier relies on inner product between vectors $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j$
- If every datapoint is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner product becomes:

$$K(\mathbf{x}_i,\mathbf{x}_j)= \boldsymbol{\phi}(\mathbf{x}_i)^{\mathbf{T}}\boldsymbol{\phi}(\mathbf{x}_j)$$

- A *kernel function* is a function that is eqiuvalent to an inner product in some feature space.
- Example:

  2-dimensional vectors $\mathbf{x}=[x_1 \;\; x_2]$; let $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j)^2$,

  Need to show that $K(\mathbf{x}_i,\mathbf{x}_j)= \boldsymbol{\phi}(\mathbf{x}_i)^{\mathbf{T}}\boldsymbol{\phi}(\mathbf{x}_j)$:

  $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j)^2 = 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}=$

  $= [1 \;\; x_{i1}^2 \;\; \sqrt{2}\, x_{i1}x_{i2} \;\; x_{i2}^2 \;\; \sqrt{2}x_{i1} \;\; \sqrt{2}x_{i2}]^{\mathbf{T}} [1 \;\; x_{j1}^2 \;\; \sqrt{2}\, x_{j1}x_{j2} \;\; x_{j2}^2 \;\; \sqrt{2}x_{j1} \;\; \sqrt{2}x_{j2}] =$

  $= \boldsymbol{\phi}(\mathbf{x}_i)^{\mathbf{T}}\boldsymbol{\phi}(\mathbf{x}_j)$, where $\boldsymbol{\phi}(\mathbf{x}) = [1 \;\; x_1^2 \;\; \sqrt{2}\, x_1x_2 \;\; x_2^2 \;\; \sqrt{2}x_1 \;\; \sqrt{2}x_2]$

- Thus, a kernel function *implicitly* maps data to a high-dimensional space (without the need to compute each $\boldsymbol{\phi}(\mathbf{x})$ explicitly).

What Functions are Kernels?

- For some functions $K(\mathbf{x}_i,\mathbf{x}_j)$ checking that $K(\mathbf{x}_i,\mathbf{x}_j)= \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ can be cumbersome.
- Mercer's theorem:

Every semi-positive definite symmetric function is a kernel

- Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

K=

$K(\mathbf{x}_1,\mathbf{x}_1)$	$K(\mathbf{x}_1,\mathbf{x}_2)$	$K(\mathbf{x}_1,\mathbf{x}_3)$	…	$K(\mathbf{x}_1,\mathbf{x}_n)$
$K(\mathbf{x}_2,\mathbf{x}_1)$	$K(\mathbf{x}_2,\mathbf{x}_2)$	$K(\mathbf{x}_2,\mathbf{x}_3)$		$K(\mathbf{x}_2,\mathbf{x}_n)$
…	…	…	…	…
$K(\mathbf{x}_n,\mathbf{x}_1)$	$K(\mathbf{x}_n,\mathbf{x}_2)$	$K(\mathbf{x}_n,\mathbf{x}_3)$	…	$K(\mathbf{x}_n,\mathbf{x}_n)$

Examples of Kernel Functions

- Linear: $K(\mathbf{x}_i,\mathbf{x}_j)= \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x})$, where $\boldsymbol{\phi}(\mathbf{x})$ is $\mathbf{x}$ itself

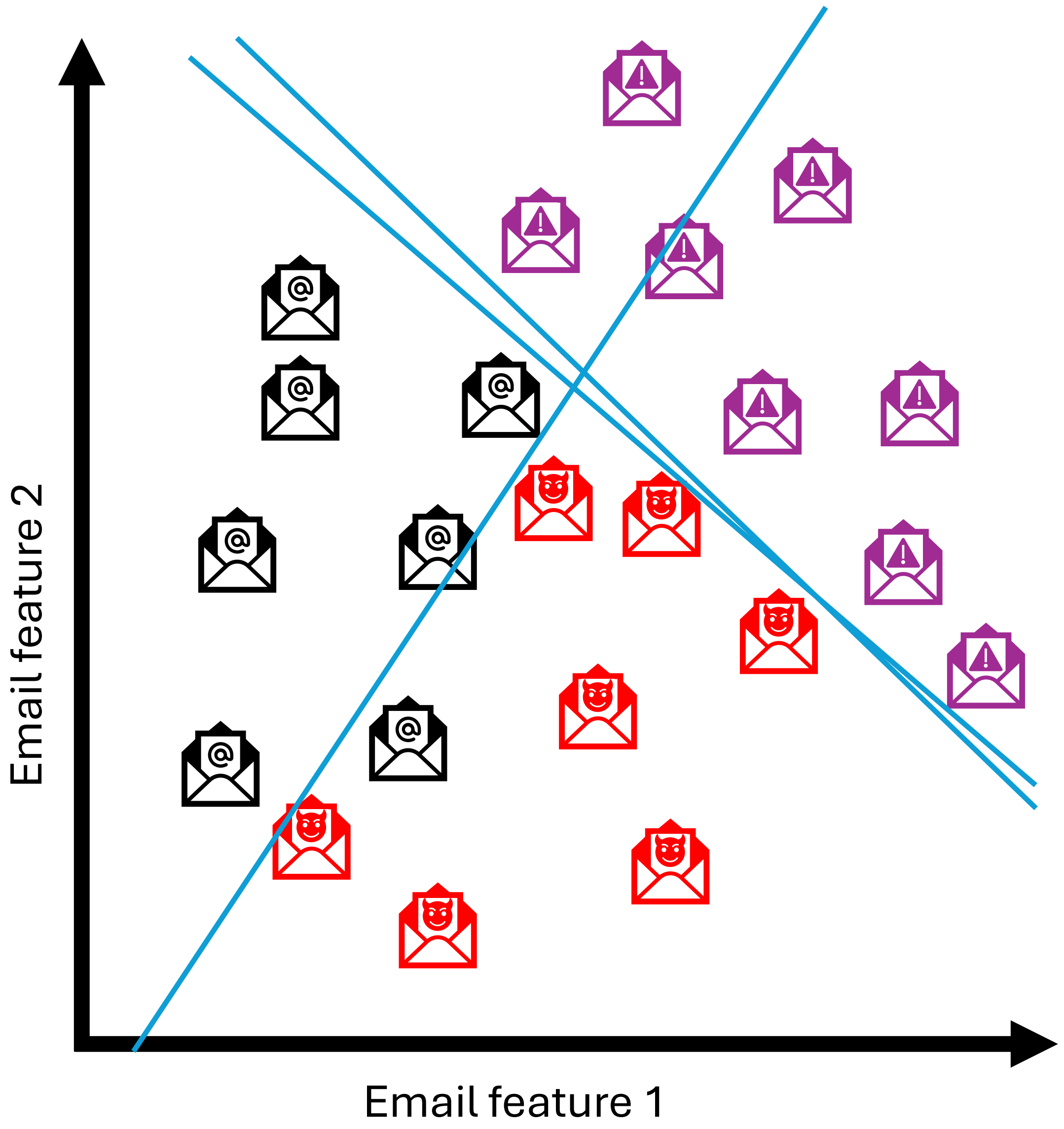
- Polynomial of power p: $K(\mathbf{x}_i,\mathbf{x}_j)= (1+ \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j)^p$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x})$, where $\boldsymbol{\phi}(\mathbf{x})$ has $\binom{d+p}{p}$ dimensions

- Gaussian (radial-basis function): $K(\mathbf{x}_i,\mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2\sigma^2}}$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x})$, where $\boldsymbol{\phi}(\mathbf{x})$ is *infinite-dimensional*: every point is mapped to *a function* (a Gaussian); combination of functions for support vectors is the separator.

- Higher-dimensional space still has *intrinsic* dimensionality d (the mapping is not *onto*), but linear separators in it correspond to *non-linear* separators in original space.

Multiple Classes

- Combining multiple SVMs
- Each SVM partitions (classifies) the feature space into two



Metrics

		True label		
		$y = -1$	$y = 1$	Total
Predicted label	$\hat{y} = -1$	True negative (TN)	False negative (FN)	$\hat{N}_n$
	$\hat{y} = 1$	False positive (FP)	True positive (TP)	$\hat{N}_p$
	Total	N_n	N_p	N

Accuracy = $\frac{TP + TN}{N}$

Precision = $\frac{TP}{TP + FP}$

True positive rate (TPR) = $\frac{TP}{TP + FN}$

Error = $\frac{FP + FN}{N}$

Recall = $\frac{TP}{TP + FN}$

False positive rate (FPR) = $\frac{FP}{FP + TN}$

Binary classification

Questions: compute these metrics for

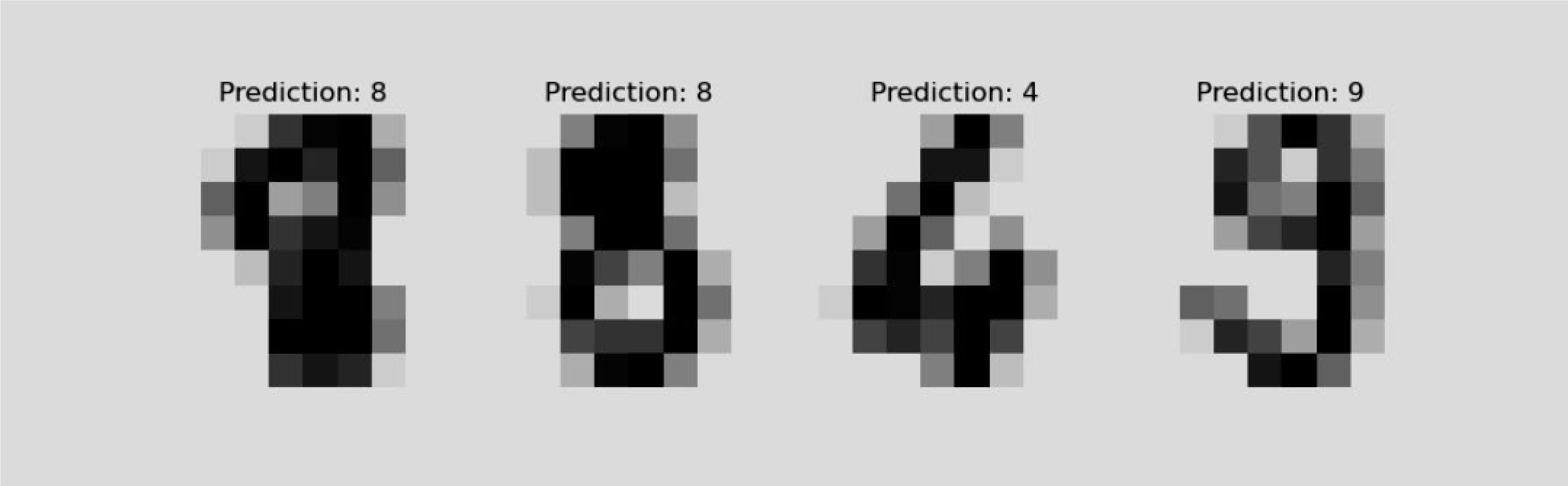
1. A perfect classifier
2. All classified as positive
3. All classified as negative
4. Random guessing

SVM applications

- SVMs were originally proposed by Boser, Guyon and Vapnik in 1992 and gained increasing popularity in late 1990s.
- SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
- SVM techniques have been extended to a number of tasks such as regression [Vapnik *et al*. ’97], principal component analysis [Schölkopf *et al*. ’99], etc.
- Most popular optimization algorithms for SVMs use *decomposition* to hill-climb over a subset of α_i’s at a time, e.g. SMO [Platt ’99] and [Joachims ’99]
- Tuning SVMs remains a black art: selecting a specific kernel and parameters is usually done in a try-and-see manner.

Applications: MNIST Dataset

- https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html



Rosenblatt

Frank Rosenblatt
1928–1969

Rosenblatt's perceptron played an important role in the history of machine learning. Initially, Rosenblatt simulated the perceptron on an IBM 704 computer at Cornell in 1957, but by the early 1960s he had built special-purpose hardware that provided a direct, parallel implementation of perceptron learning. Many of his ideas were encapsulated in "Principles of Neuro-dynamics: Perceptrons and the Theory of Brain Mechanisms" published in 1962. Rosenblatt's work was criticized by Marvin Minksy, whose objections were published in the book "Perceptrons", co-authored with Seymour Papert. This book was widely misinterpreted at the time as showing that neural networks were fatally flawed and could only learn solutions for linearly separable problems. In fact, it only proved such limitations in the case of single-layer networks such as the perceptron and merely conjectured (incorrectly) that they applied to more general network models. Unfortunately, however, this book contributed to the substantial decline in research funding for neural computing, a situation that was not reversed until the mid-1980s. Today, there are many hundreds, if not thousands, of applications of neural networks in widespread use, with examples in areas such as handwriting recognition and information retrieval being used routinely by millions of people.

Model - *step* `squashing` function and criterion

Linear mapping:

$$f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$$

Reminder - logistic regression:

$$p(y_n | x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y = 1 \\ 1 - g_\theta(x_n), & \text{if } y = -1 \end{cases}$$

$g_\theta(x) = \sigma(f_\theta(x))$, σ is logistic sigmoid

Perceptron model: $y_n = \begin{cases} 1, & \text{if } \theta^\intercal x_n \geq 0 \\ -1 & \text{if } \theta^\intercal x_n < 0 \end{cases}$

Perceptron criterion: Want $y_n \theta^\intercal x_n > 0$ for all n, $L(\theta) = -\sum_{n \in \mathcal{M}} y_n \theta^T x_n$

$\mathcal{M}$: all mis-classified examples

Perceptron - learning algorithm

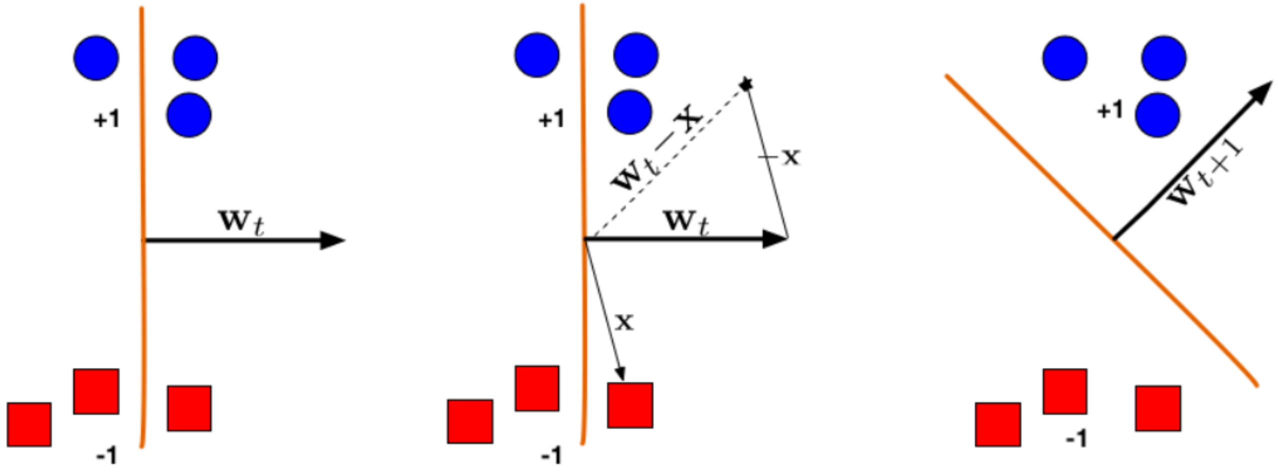
Perceptron criterion: $y_n \theta^\intercal x_n > 0$

for all n, $L(\theta) = -\sum_{n \in \mathcal{M}} y_n \theta^T x_n$

$\mathcal{M}$: all mis-classified examples

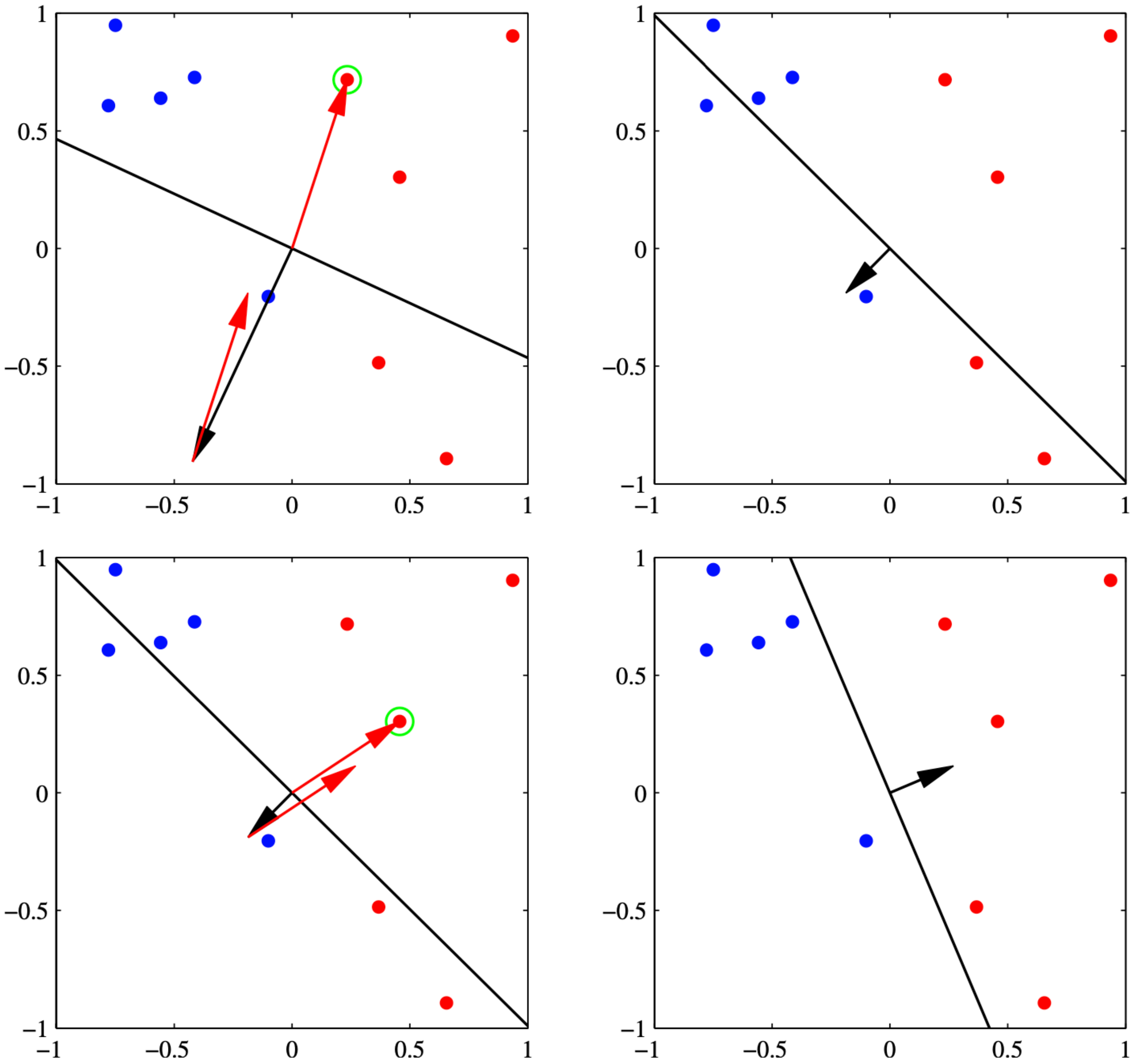
SGD update: *for each misclassified example,*

$\theta^{t+1} = \theta^t + x_n y_n.$



Initialize $\vec{w} = \vec{0}$
while TRUE **do**
    $m = 0$
    for $(x_i, y_i) \in D$ **do**
        if $y_i(\vec{w}^T \cdot \vec{x_i}) \leq 0$ **then**
            $\vec{w} \leftarrow \vec{w} + y\vec{x}$
            $m \leftarrow m + 1$
        end if
    end for
    if $m = 0$ **then**
        break
    end if
end while

Perceptron algorithm - an example



PRML Bishop

Perceptron - potential issues

Perceptron convergence theorem (Rosenblatt 1962): if there exists an exact solution (aka. if the training data set is linearly separable), then the perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps.

But the number of steps required to achieve convergence could still be *substantial*.

Even when the data set is linearly separable, there may be *many* solutions, and which one is found will depend on the *initialisation* of the parameters and on the order of presentation of the data points. Furthermore, for data sets that are not linearly separable, the perceptron learning algorithm will **never** converge.

The perceptron *does not* provide probabilistic outputs, nor does it generalise readily to K > 2 class

PRML Bishop

Multiple Perceptrons?

- What functions can it approximate?
- What should be the structure and connectivity?