NEUROTECHUOFT: INTERMEDIATE WORKSHOPS
MACHINE LEARNING IN NEUROTECHNOLOGY

Machine Learning II

NeurotechUofT

Agenda

Topics for this week:

1. K Nearest Neighbors (KNN)
2. Decision Trees
3. Logistic Regression
4. Support Vector Machines (SVM)
5. Clustering

*Slides from today are mainly from CSC311 Machine Learning - Winter 2020 (Taught by Prof. Amir Farahmand)

Nearest Neighbors

Adapted from CSC311, 2020

- Suppose we're given a novel input vector $\mathbf{x}$ we'd like to classify.
- The idea: find the nearest input vector to $\mathbf{x}$ in the training set and copy its label.
- Can formalize "nearest" in terms of Euclidean distance

$$||\mathbf{x}^{(a)} - \mathbf{x}^{(b)}||_2 = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - x_j^{(b)})^2}$$

Algorithm:

1. Find example $(\mathbf{x}^*, t^*)$ (from the stored training set) closest to $\mathbf{x}$. That is:

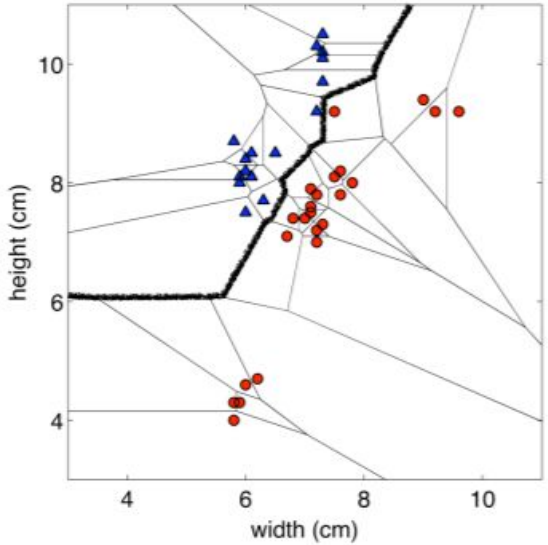
$$\mathbf{x}^* = \underset{\mathbf{x}^{(i)} \in \text{train. set}}{\text{argmin}} \text{ distance}(\mathbf{x}^{(i)}, \mathbf{x})$$

2. Output $y = t^*$

- Note: we do not need to compute the square root. Why?

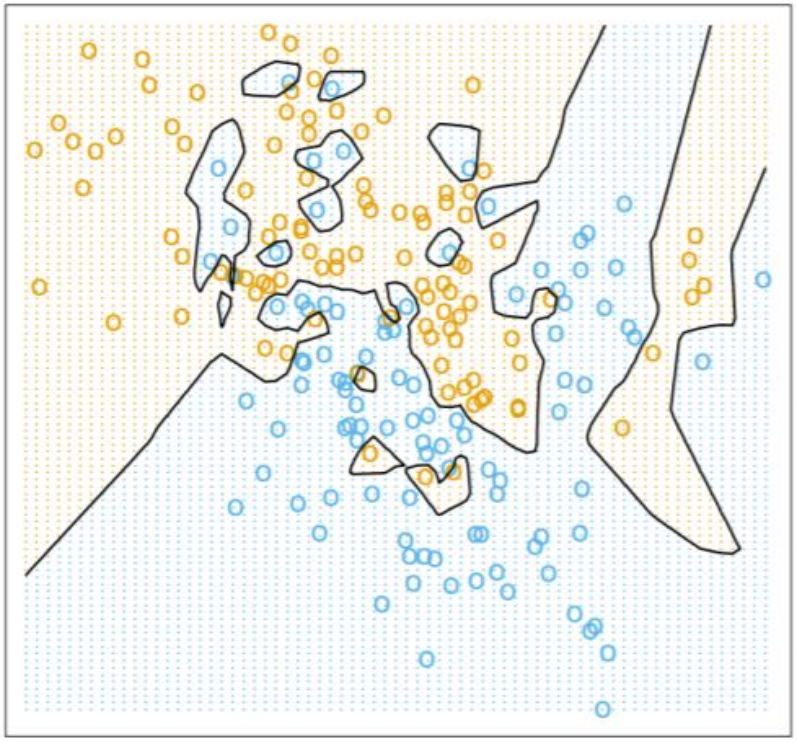
Nearest Neighbors: Decision Boundaries

Decision boundary: the boundary between regions of input space assigned to different categories.



K-Nearest neighbors

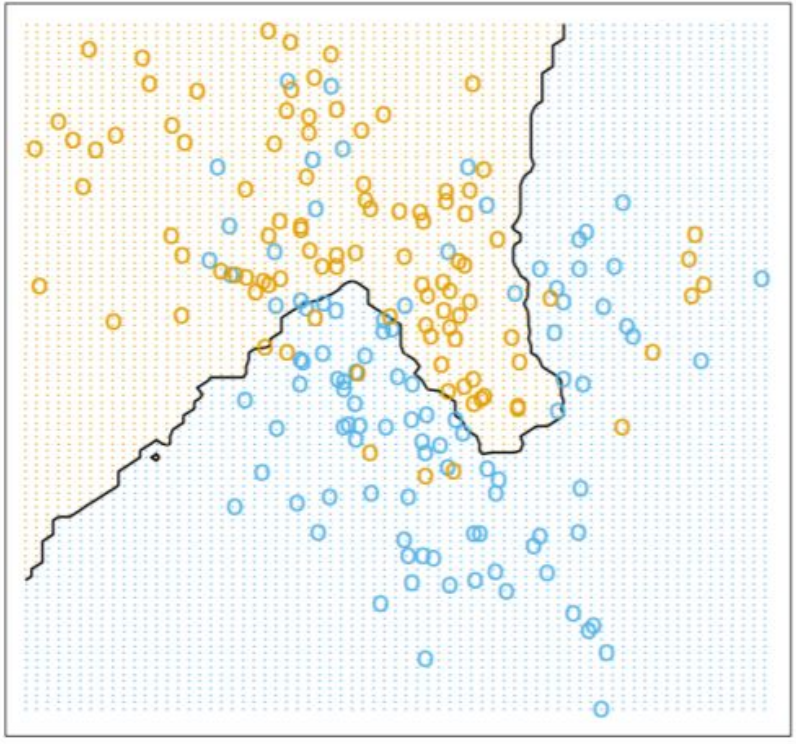
k=1



[Image credit: "The Elements of Statistical Learning"]

K-Nearest neighbors

k=15



[Image credit: "The Elements of Statistical Learning"]

k-Nearest Neighbors

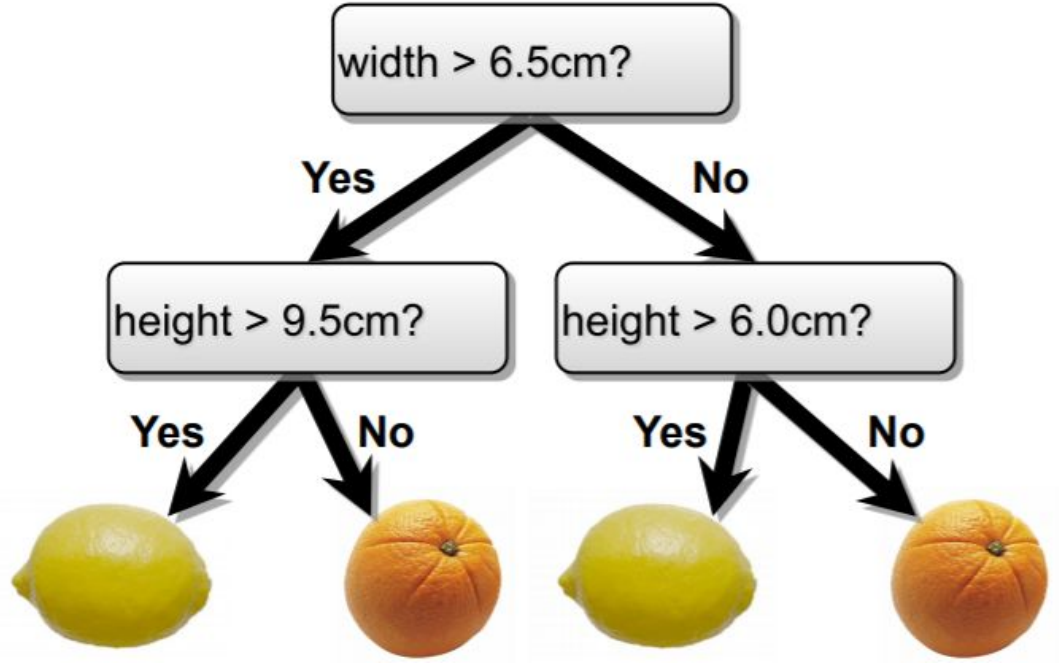
Tradeoffs in choosing k?

- Small k
  - Good at capturing fine-grained patterns
  - May overfit, i.e. be sensitive to random idiosyncrasies in the training data
- Large k
  - Makes stable predictions by averaging over lots of examples
  - May underfit, i.e. fail to capture important regularities
- Balancing k:
  - The optimal choice of k depends on the number of data points n.
  - Nice theoretical properties if $k \to \infty$ and $\frac{k}{n} \to 0$.
  - Rule of thumb: Choose $k = n^{\frac{2}{2+d}}$.
  - We explain an easier way to choose k using data.

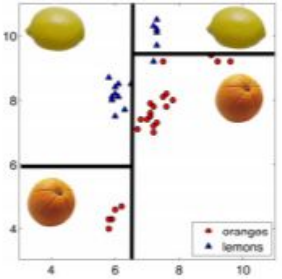
Decision Trees

Decision Trees

- **Decision trees** make predictions by recursively splitting on different attributes according to a tree structure.
- Example: classifying fruit as an orange or lemon based on height and width



Decision Tree: Classification and Regression



- Each path from root to a leaf defines a region R_m of input space
- Let $\{(x^{(m_1)}, t^{(m_1)}), \ldots, (x^{(m_k)}, t^{(m_k)})\}$ be the training examples that fall into R_m
- **Classification tree**:
  - discrete output
  - leaf value y^m typically set to the most common value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$
- **Regression tree**:
  - continuous output
  - leaf value y^m typically set to the mean value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$

Note: We will focus on classification

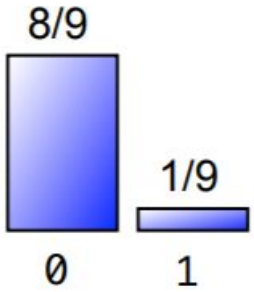
How do we choose a good split?

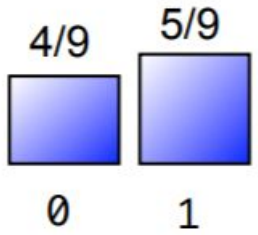
- With many features, there are many possible splits
- One way is to choose a split that maximizes **information gain**
- What do we mean by information gain? Find out very soon...

Quantifying Uncertainty

Entropy is a measure of expected "surprise": How uncertain are we of the value of a draw from this distribution?

$$H(X) = -\mathbb{E}_{X\sim p}[\log_2 p(X)] = -\sum_{x\in X} p(x)\log_2 p(x)$$





$$-\frac{8}{9}\log_2\frac{8}{9} - \frac{1}{9}\log_2\frac{1}{9} \approx \frac{1}{2}$$

$$-\frac{4}{9}\log_2\frac{4}{9} - \frac{5}{9}\log_2\frac{5}{9} \approx 0.99$$

- Averages over information content of each observation
- Unit = **bits** (based on the base of logarithm)
- A fair coin flip has 1 bit of entropy

Modular Approach for ML **IMPORTANT**

- KNN and decision trees are procedures for learning, we want a **more general approach**

1. Choose a **model** - eg. logistic regression
2. Define the **loss function -** how bad is the model compared to the ground truth?
3. Choose a **regularizer -** think of it like a constraint
4. Fit the model (minimize loss function) - **training**
    a. **Optimization algorithm eg. gradient descent**

Linear Regression – Model

- Model: In linear regression, we use linear functions of the inputs $\mathbf{x} = (x_1, \ldots, x_D)$ to make predictions y of the target value t:

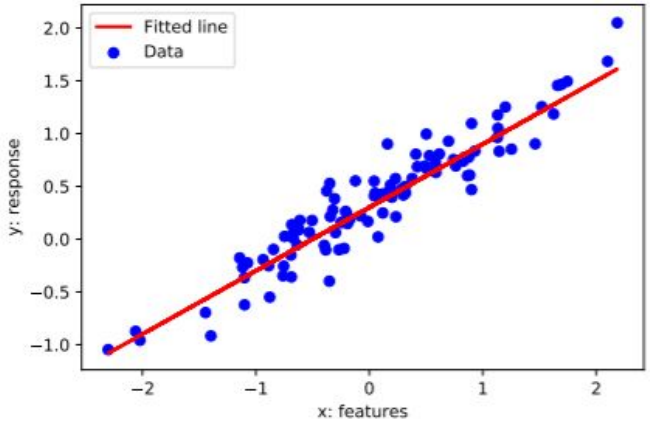
$$y = f(\mathbf{x}) = \sum_j w_j x_j + b$$

  - y is the prediction
  - $\mathbf{w}$ is the weights
  - b is the bias (or intercept) (do not confuse with the bias-variance tradeoff in the next lecture)
- $\mathbf{w}$ and b together are the parameters
- We hope that our prediction is close to the target: $y \approx t$.

Linear Regression

We have a dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, t^{(i)}) \text{ for } i = 1, 2, ..., N\}$ where,

- $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_D^{(i)})^\top \in \mathbb{R}^D$ are the inputs, e.g., age, height.
- $t^{(i)} \in \mathbb{R}$ is the target or response (e.g. income),
- predict $t^{(i)}$ with a linear function of $\mathbf{x}^{(i)}$:



- $t^{(i)} \approx y^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + b$
- Find the "best" line $(\mathbf{w}, b)$.
- minimize$_{(\mathbf{w},b)} \sum_{i=1}^{N} \mathcal{L}(y^{(i)}, t^{(i)})$

Linear Regression – Loss Function

- How to quantify the quality of the fit to data?
- A loss function $\mathcal{L}(y, t)$ defines how bad it is if, for some example $\mathbf{x}$, the algorithm predicts y, but the target is actually t.
- Squared error loss function:

$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$$

- $y - t$ is the residual, and we want to make its magnitude small
- The $\frac{1}{2}$ factor is just to make the calculations convenient.
- Cost function: loss function averaged over all training examples

$$\begin{aligned} \mathcal{J}(\mathbf{w}, b) &= \frac{1}{2N} \sum_{i=1}^{N} \left(y^{(i)} - t^{(i)} \right)^2 \\ &= \frac{1}{2N} \sum_{i=1}^{N} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - t^{(i)} \right)^2 \end{aligned}$$

- The terminology is not universal. Some might call "loss" *pointwise loss* and the "cost function" the *empirical loss* or *average loss*.

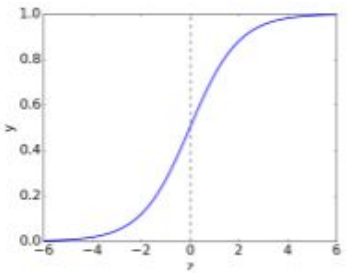
Gradient Descent

- How do we minimize the cost $\mathcal{J}$ in this case? No direct solution.
  - Taking derivatives of $\mathcal{J}$ w.r.t. $\mathbf{w}$ and setting them to 0 doesn't have an explicit solution.
- Now let's see a second way to minimize the cost function which is more broadly applicable: gradient descent.
- Gradient descent is an iterative algorithm, which means we apply an update repeatedly until some criterion is met.
- We initialize the weights to something reasonable (e.g. all zeros) and repeatedly adjust them in the direction of steepest descent.

Logistic Regression

- The logistic function is a kind of sigmoid, or S-shaped function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



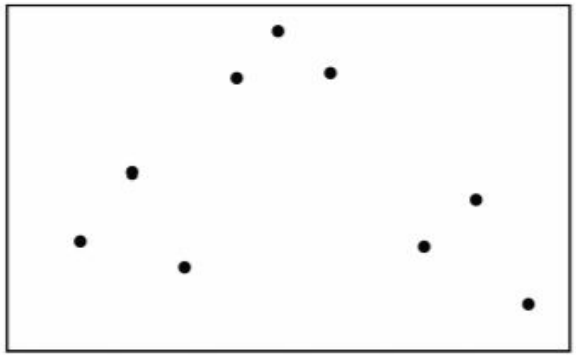
- $\sigma^{-1}(y) = \log(y/(1-y))$ is called the logit.
- A linear model with a logistic nonlinearity is known as log-linear:

$$z = \mathbf{w}^\top \mathbf{x} + b$$
$$y = \sigma(z)$$
$$\mathcal{L}_{\text{SE}}(y, t) = \frac{1}{2}(y - t)^2.$$

- Used in this way, σ is called an activation function.

Clustering

Clustering problem

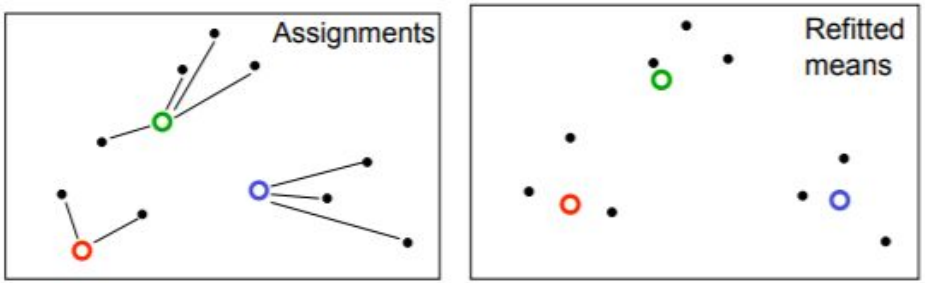


- Assume that the data points $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$ live in an Euclidean space, i.e., $\mathbf{x}^{(n)} \in \mathbb{R}^D$.
- Assume that each data point belongs to one of K clusters
- Assume that the data points from same cluster are similar, i.e., close in Euclidean distance.
- How can we identify those clusters and the data points that belong to each cluster?

K-means Algorithm

High level overview of algorithm:

- Initialization: randomly initialize cluster centres
- The algorithm iteratively alternates between two steps:
  - Assignment step: Assign each data point to the closest cluster
  - Refitting step: Move each cluster centre to the mean of the data assigned to it



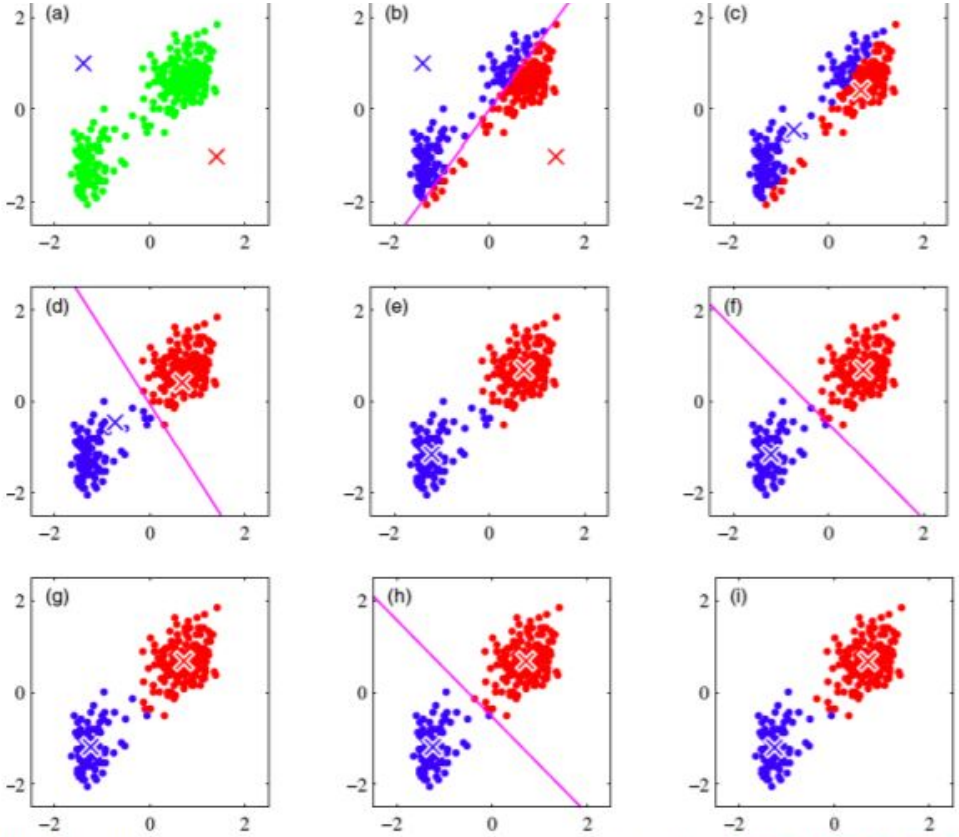


Figure from Bishop

Simple demo: http://syskall.com/kmeans.js/

The K-means Algorithm

- Initialization: Set K cluster means $\mathbf{m}_1, \ldots, \mathbf{m}_K$ to random values
- Repeat until convergence (until assignments do not change):
  - Assignment: Optimize J w.r.t. $\{\mathbf{r}\}$: Each data point $\mathbf{x}^{(n)}$ assigned to nearest centre

$$\hat{k}^{(n)} = \arg\min_k ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$$

and Responsibilities (1-hot or 1-of-K encoding)

$$r_k^{(n)} = \mathbb{I}\{\hat{k}^{(n)} = k\} \quad \text{for} \quad k = 1, .., K$$

  - Refitting: Optimize J w.r.t. $\{\mathbf{m}\}$: Each centre is set to mean of data assigned to it

$$\mathbf{m}_k = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{\sum_n r_k^{(n)}}.$$

SVM

Binary Classification with a Linear Model

- Classification: Predict a discrete-valued target
- Binary classification: Targets $t \in \{-1, +1\}$
- Linear model:

$$z = \mathbf{w}^\top \mathbf{x} + b$$
$$y = \text{sign}(z)$$

- Question: How should we choose $\mathbf{w}$ and b?

Zero-One Loss

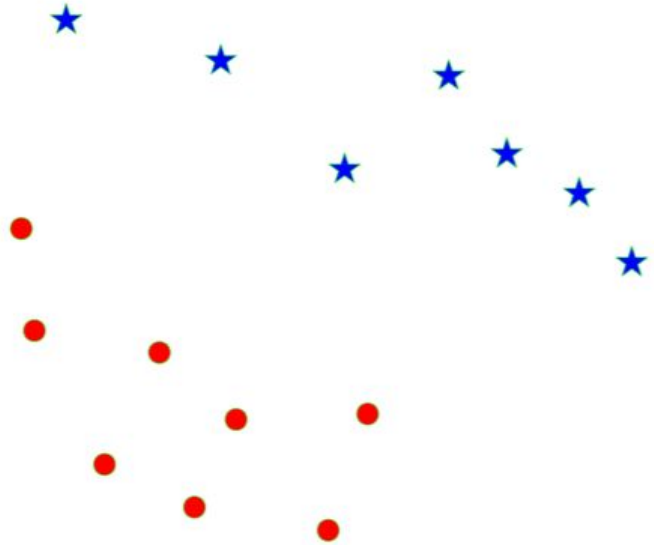
- We can use the $0-1$ loss function, and find the weights that minimize it over data points

$$\begin{aligned}\mathcal{L}_{0-1}(y,t) &= \begin{cases} 0 & \text{if } y = t \\ 1 & \text{if } y \neq t \end{cases} \\ &= \mathbb{I}\{y \neq t\}.\end{aligned}$$

- But minimizing this loss is computationally difficult, and it can't distinguish different hypotheses that achieve the same accuracy.
- We investigated some other loss functions that are easier to minimize, e.g., logistic regression with the cross-entropy loss $\mathcal{L}_{\text{CE}}$.
- Let's consider a different approach, starting from the geometry of binary classifiers.

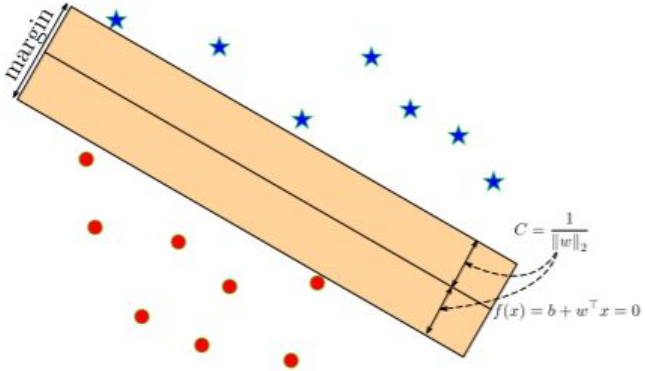
Separating Hyperplanes

Suppose we are given these data points from two different classes and want to find a linear classifier that separates them.



Optimal Separating Hyperplane

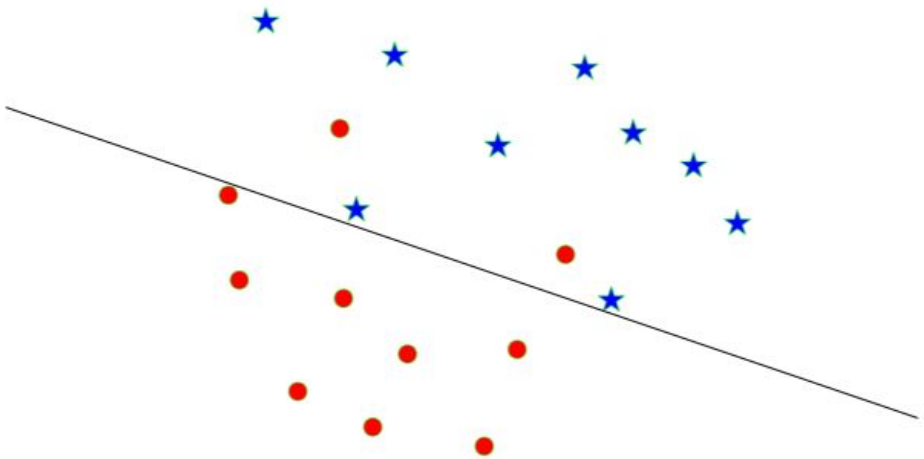
Optimal Separating Hyperplane: A hyperplane that separates two classes and maximizes the distance to the closest point from either class, i.e., maximize the margin of the classifier.



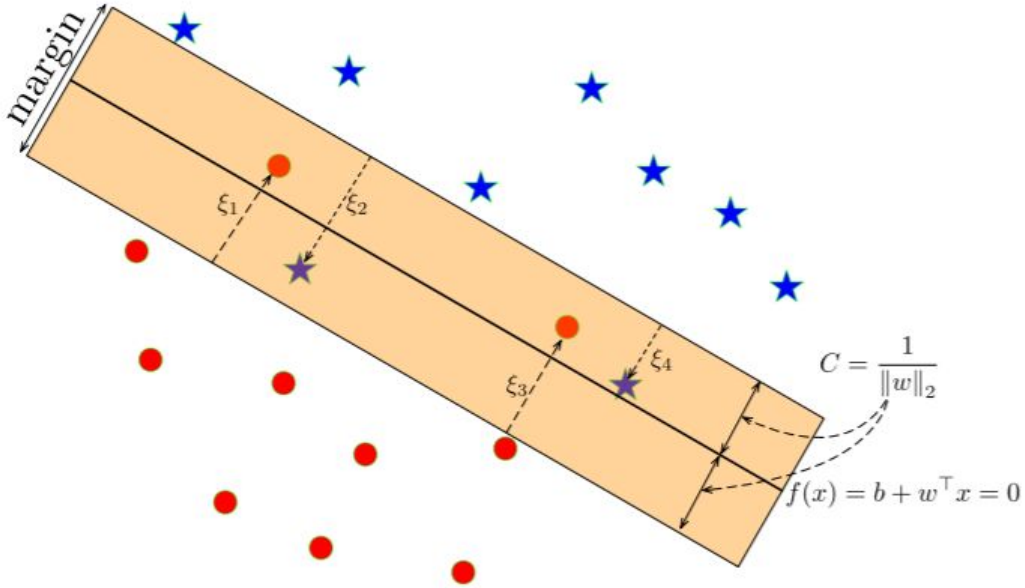
Intuitively, ensuring that a classifier is not too close to any data points leads to better generalization on the test data.

Non-Separable Data Points

How can we apply the max-margin principle if the data are **not** linearly separable?



Maximizing Margin for Non-Separable Data Points



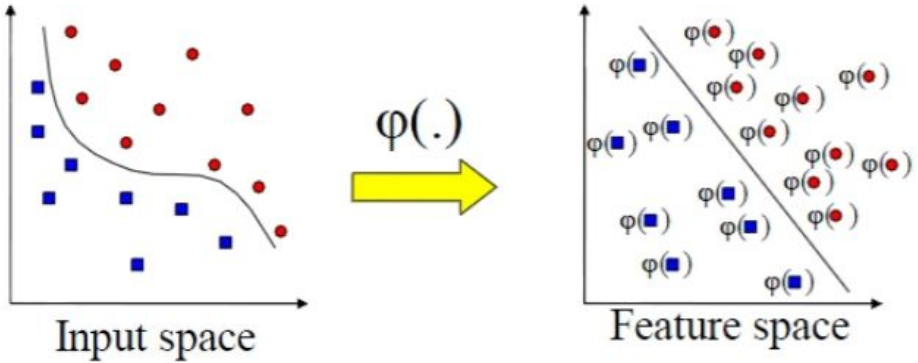
Main Idea:

- Allow some points to be within the margin or even be misclassified; we represent this with slack variables ξ_i.
- But constrain or penalize the total amount of slack.

Kernel Methods

Non-Linear Decision Boundaries

- We talk about SVM: Max margin linear classifier
- Linear is limiting, how do we get non-linear decision boundaries?
- Feature mapping $\mathbf{x} \mapsto \phi(\mathbf{x})$



- How do we find good features?
- If features are in a high dimension, the computation cost might be large. Can we avoid it?