HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Machine Learning

IT3190E

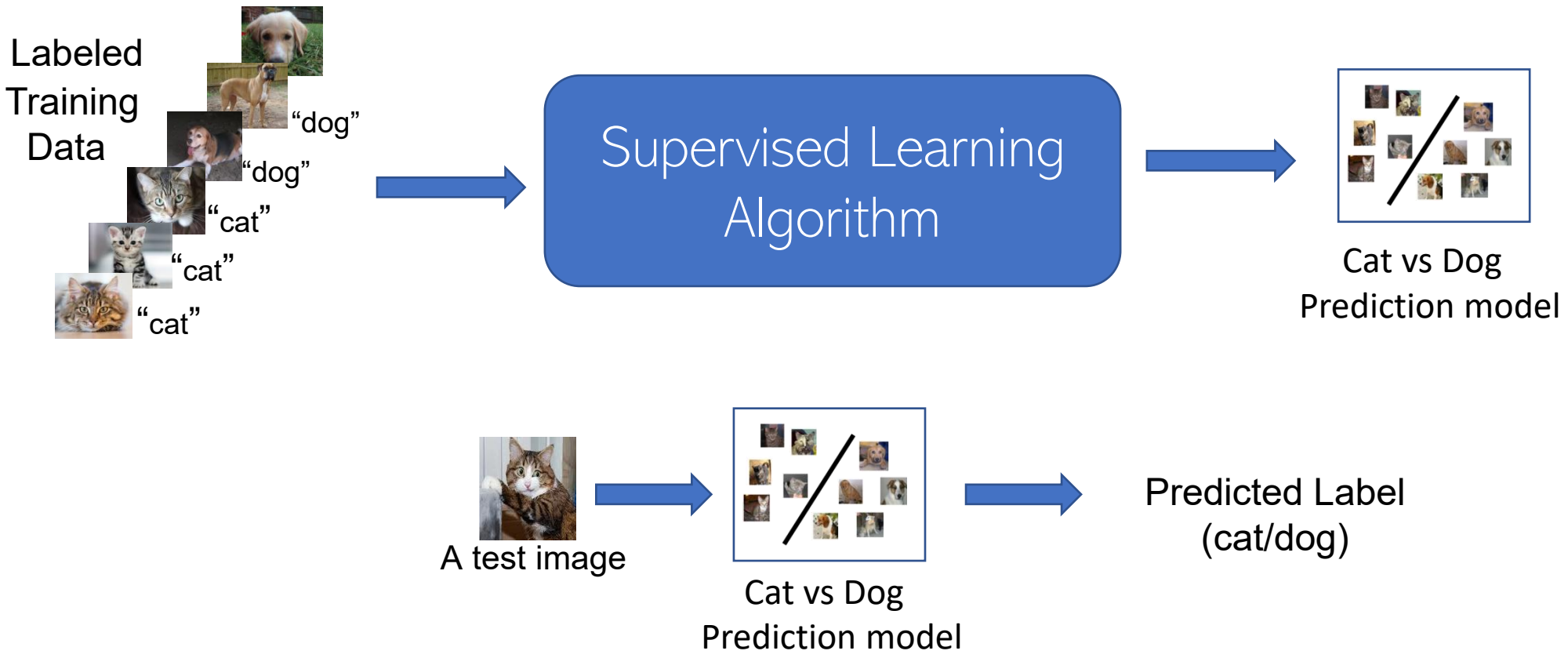
Lecture: Introduction

ONE LOVE. ONE FUTURE.

Contents

Learning with Prototypes

Supervised Learning



Some Types of Supervised Learning Problems

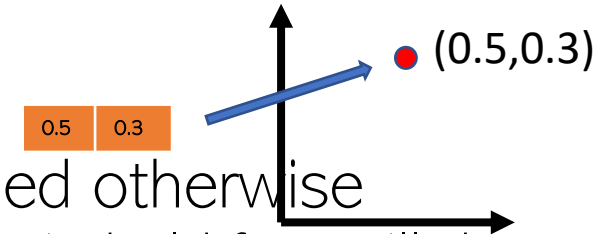
- Consider building an ML module for an e-mail client
- Some tasks that we may want this module to perform
  - Predicting whether an email of spam or normal: Binary Classification
  - Predicting which of the many folders the email should be sent to: Multi-class Classification
  - Predicting all the relevant tags for an email: Tagging or Multi-label Classification
  - Predicting what's the spam-score of an email: Regression
  - Predicting which email(s) should be shown at the top: Ranking
  - Predicting which emails are work/study-related emails: One-class Classification
- These predictive modeling tasks can be formulated as supervised learning problems
- Today: A very simple supervised learning model for binary/multi-class classification
  - This model doesn't require any fancy maths – just computing means and distances

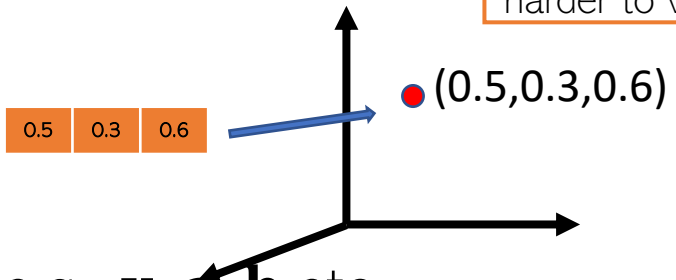
Some Notation and Conventions

- In ML, inputs are usually represented by vectors
- A vector consists of an array of scalar values

- Geometrically, a vector is just a point in a vector space, e.g.,
  - A length 2 vector is a point in 2-dim vector space
  - A length 3 vector is a point in 3-dim vector space

Likewise for higher dimensions, even though harder to visualize





- Unless specified otherwise
  - Small letters in bold font will denote vectors, e.g., $\mathbf{x}, \mathbf{a}, \mathbf{b}$ etc.
  - Small letters in normal font to denote scalars, e.g. x, a, b, etc
  - Capital letters in bold font will denote matrices (2-dim arrays), e.g., $\mathbf{X}, \mathbf{A}, \mathbf{B}$, etc

Some Notation and Conventions

- A single vector will be assumed to be of the form $\mathbf{x} = [x_1, x_2, \dots, x_D]$

- Unless specified otherwise, vectors will be assumed to be column vectors
  - So we will assume $\mathbf{x} = [x_1, x_2, \dots, x_D]$ to be a column vector of size $D \times 1$
  - Assuming each element to be real-valued scalar, $\mathbf{x} \in \mathbb{R}^{D \times 1}$ or $\mathbf{x} \in \mathbb{R}^{D}$ ($\mathbb{R}$: space of reals)

- If $\mathbf{x} = [x_1, x_2, \dots, x_D]$ is a feature vector representing, say an image, then
  - D denotes the dimensionality of this feature vector (number of features)
  - x_i (a scalar) denotes the value of i^{th} feature in the image

- For denoting multiple vectors, we will use a subscript with each vector, e.g.,
  - N images denoted by N feature vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_\mathrm{N}$, or compactly as $\{\mathbf{x}_n\}_{n=1}^{N}$
  - The vector $\mathbf{x}_n$ denotes the n^{th} image
  - x_{ni} (a scalar) denotes the i^{th} feature ($i = 1,2, \dots, D$) of the n^{th} image

Some Basic Operations on Vectors

- Addition/subtraction of two vectors gives another vector of the same size
- The mean $\boldsymbol{\mu}$ (average or centroid) of N vectors $\{\mathbf{x}_n\}_{n=1}^N$

$$\mu = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \in \mathbb{R}^D$$ (of the same size as each $\mathbf{x}_n$)

- The inner/dot product of two vectors $\boldsymbol{a} \in \mathbb{R}^D$ and $\boldsymbol{b} \in \mathbb{R}^D$

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a}^\top \boldsymbol{b} = \sum_{i=1}^{D} a_i b_i$$ (a real-valued number denoting how "similar" $\boldsymbol{a}$ and $\boldsymbol{b}$ are)

Assuming both $\boldsymbol{a}$ and $\boldsymbol{b}$ have unit Euclidean norm

- For a vector $\boldsymbol{a} \in \mathbb{R}^D$, its Euclidean norm is defined via its inner product with itself

$$\|\boldsymbol{a}\|_2 = \sqrt{\boldsymbol{a}^\top \boldsymbol{a}} = \sqrt{\sum_{i=1}^{d} a_i^2}$$

Computing Distances

- Euclidean (L2 norm) distance between two vectors $\boldsymbol{a} \in \mathbb{R}^D$ and $\boldsymbol{b} \in \mathbb{R}^D$

$$d_2(\boldsymbol{a}, \boldsymbol{b}) = ||\boldsymbol{a} - \boldsymbol{b}||_2 = \sqrt{\sum_{i=1}^{D} (a_i - b_i)^2} = \sqrt{(\boldsymbol{a} - \boldsymbol{b})^\top (\boldsymbol{a} - \boldsymbol{b})} = \sqrt{\boldsymbol{a}^\top \boldsymbol{a} + \boldsymbol{b}^\top \boldsymbol{b} - 2\boldsymbol{a}^\top \boldsymbol{b}}$$

Sqrt of Inner product of the difference vector

Another expression in terms of inner products of individual vectors

- Weighted Euclidean distance between two vectors $\boldsymbol{a} \in \mathbb{R}^D$ and $\boldsymbol{b} \in \mathbb{R}^D$

$$d_w(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{D} w_i (a_i - b_i)^2} = \sqrt{(\boldsymbol{a} - \boldsymbol{b})^\top \mathbf{W} (\boldsymbol{a} - \boldsymbol{b})}$$

$\mathbf{W}$ is a DxD diagonal matrix with weights w_i on its diagonals. Weights may be known or even learned from data (in ML problems)

- Absolute (L1 norm) distance between two vectors $\boldsymbol{a} \in \mathbb{R}^D$ and $\boldsymbol{b} \in \mathbb{R}^D$

L1 norm distance is also known as the Manhattan distance or Taxicab norm (it's a very natural notion of distance between two points in some vector space)

$$d_1(\boldsymbol{a}, \boldsymbol{b}) = ||\boldsymbol{a} - \boldsymbol{b}||_1 = \sum_{i=1}^{D} |a_i - b_i|$$

Our First Supervised Learner

Prelude: A Very Primitive Classifier

- Consider a binary classification problem – cat vs dog
- Assume training data with just 2 images – one and one 
- Given a new test image (cat/dog), how do we predict its label?
- A simple idea: Predict using its distance from each of the 2 training images

The idea also applies to multi-class classification: Use one image per class, and predict label based on the distances of the test image from all such images

$d($ Test image ,  $) < d($ Test image ,  $)$? Predict cat else dog

Wait. Is it ML? Seems to be like just a simple "rule". Where is the "learning" part in this?

Some possibilities: Use a feature learning/selection algorithm to extract features, and use a Mahalanobis distance where you learn the W matrix (instead of using a predefined W), using "distance metric learning" techniques

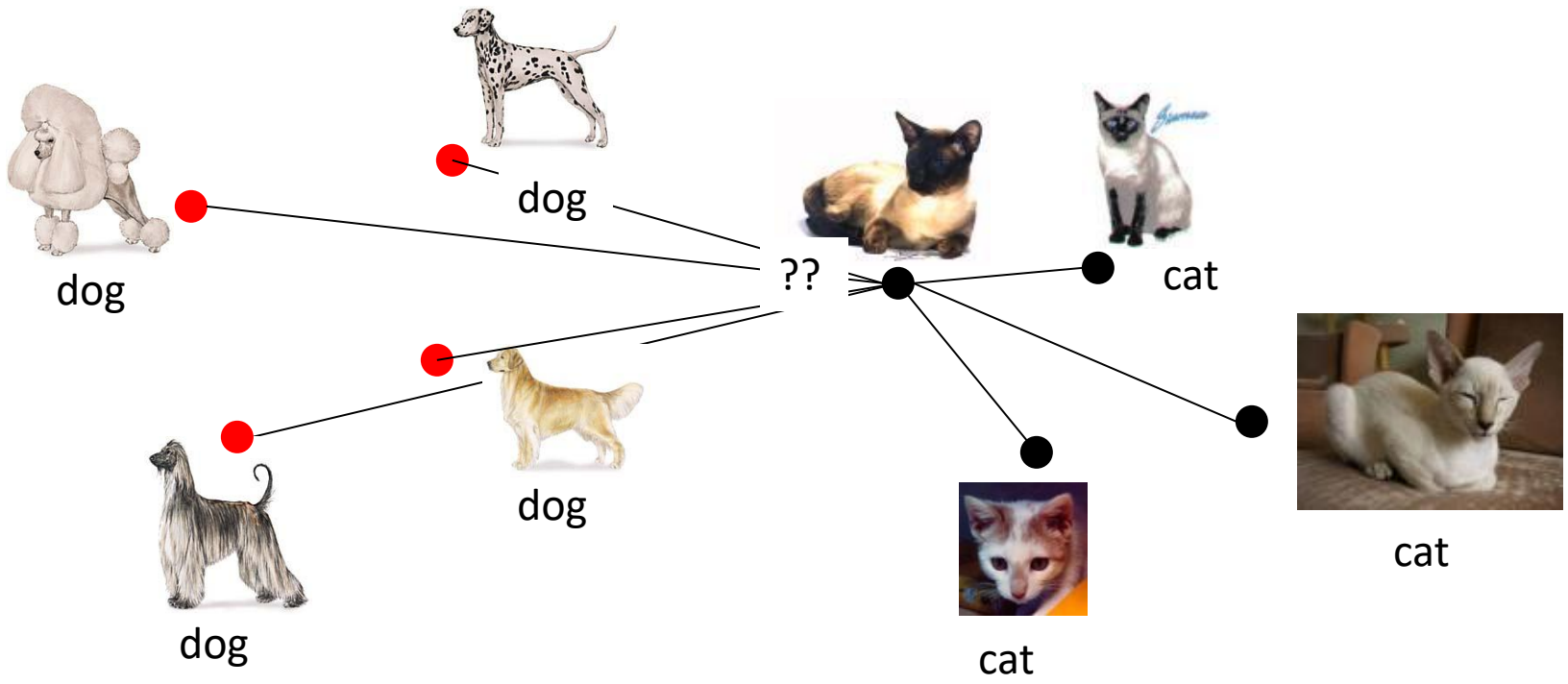
Even this simple model can be learned. For example, for the feature extraction/selection part and/or for the distance computation part

Improving Our Primitive Classifier

- Just one input per class may not sufficiently capture variations in a class
- A natural improvement can be by using more inputs per class

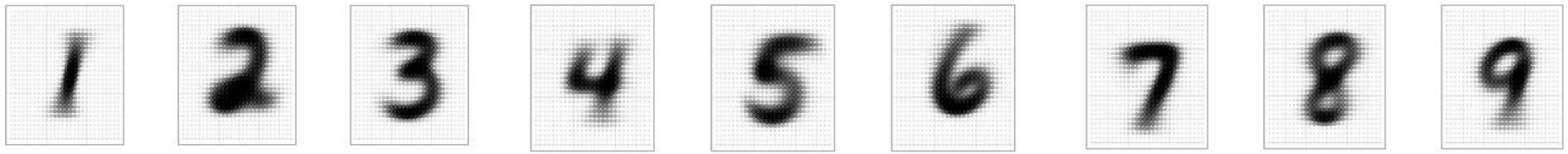
- We will consider two approaches to do th
  - Learning with Prototypes (LwP)
  - Nearest Neighbors (NN)
- Both LwP and NN will use multiple inputs per class but in different ways

Learning to predict categories



Learning with Prototypes (LwP)

- Basic idea: Represent each class by a "prototype" vector
- Class Prototype: The "mean" or "average" of inputs from that class



Averages (prototypes) of each of the handwritten digits 1-9

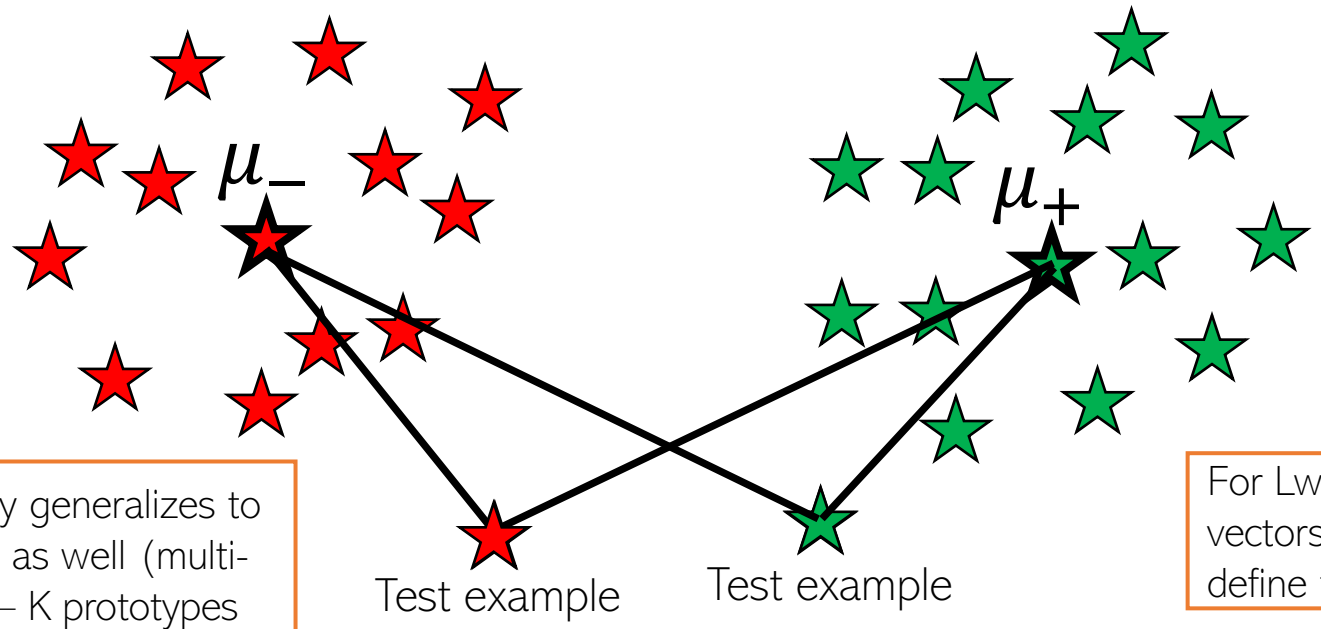
- Predict label of each test input based on its distances from the class prototypes
  - Predicted label will be the class that is the closest to the test input
- How we compute distances can have an effect on the accuracy of this model (may need to try Euclidean, weight Euclidean, Mahalanobis, or something else)

Pic from: https://www.reddit.com/r/dataisbeautiful/comments/3wgbv9/average_handwritten_digit_oc/

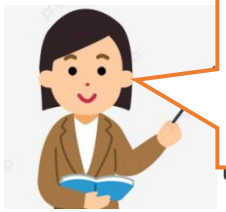
Learning with Prototypes (LwP): An Illustration

- Suppose the task is binary classification (two classes assumed pos and neg)
- Training data: N labelled examples $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$, $\mathbf{x}_n \in \mathbb{R}^D$, $y_n \in \{-1, +1\}$
  - Assume N_+ example from positive class, N_- examples from negative class
  - Assume green is positive and red is negative

$$\mu_- = \frac{1}{N_-} \sum_{y_n=-1} \mathbf{x}_n$$



$$\mu_+ = \frac{1}{N_+} \sum_{y_n=+1} \mathbf{x}_n$$



LwP straightforwardly generalizes to more than 2 classes as well (multi-class classification) – K prototypes for K classes

For LwP, the prototype vectors (μ_+ and μ_- here) define the "model"

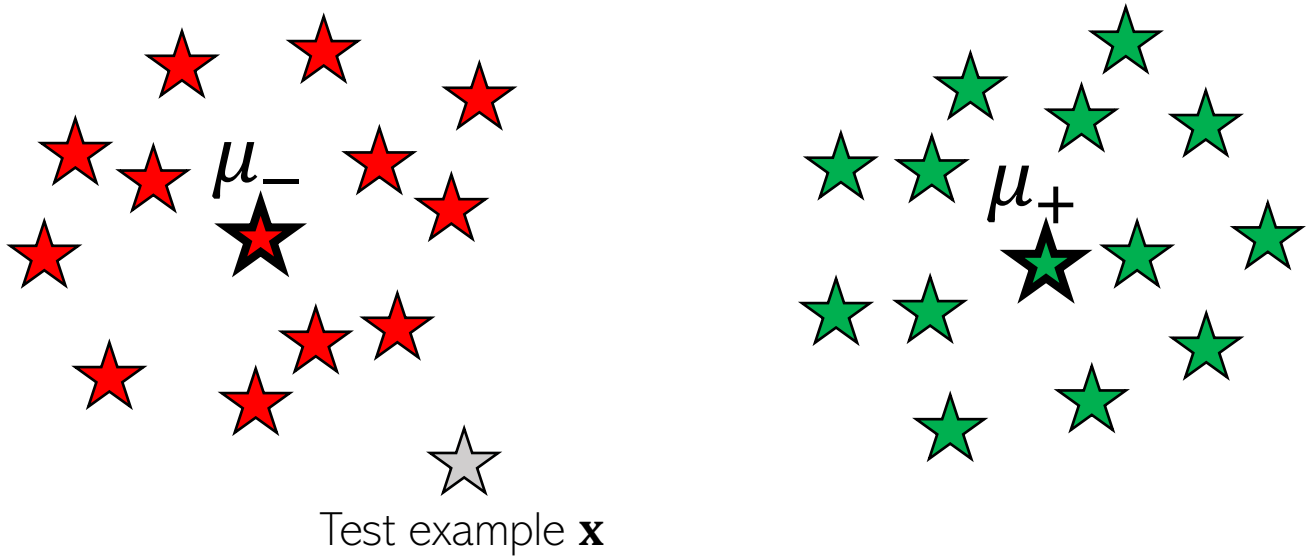
LwP: The Prediction Rule, Mathematically

- What does the prediction rule for LwP look like mathematically?

- Assume we are using Euclidean distances here

$$||\boldsymbol{\mu}_- - \mathbf{x}||^2 = ||\boldsymbol{\mu}_-||^2 + ||\mathbf{x}||^2 - 2\langle \boldsymbol{\mu}_-, \mathbf{x} \rangle$$

$$||\boldsymbol{\mu}_+ - \mathbf{x}||^2 = ||\boldsymbol{\mu}_+||^2 + ||\mathbf{x}||^2 - 2\langle \boldsymbol{\mu}_+, \mathbf{x} \rangle$$



Prediction Rule: Predict label as +1 if $f(\mathbf{x}) = ||\boldsymbol{\mu}_- - \mathbf{x}||^2 - ||\boldsymbol{\mu}_+ - \mathbf{x}||^2 > 0$ otherwise -1

LwP: The Prediction Rule, Mathematically

- Let's expand the prediction rule expression a bit more

$$
\begin{aligned}
f(\mathbf{x}) &= \left|\left|\boldsymbol{\mu}_- - \mathbf{x}\right|\right|^2 - \left|\left|\boldsymbol{\mu}_+ - \mathbf{x}\right|\right|^2 \\
&= \left|\left|\boldsymbol{\mu}_-\right|\right|^2 + \left|\left|\mathbf{x}\right|\right|^2 - 2\langle\boldsymbol{\mu}_-, \mathbf{x}\rangle - \left|\left|\boldsymbol{\mu}_+\right|\right|^2 - \left|\left|\mathbf{x}\right|\right|^2 + 2\langle\boldsymbol{\mu}_+, \mathbf{x}\rangle \\
&= 2\langle\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-, \mathbf{x}\rangle + \left|\left|\boldsymbol{\mu}_-\right|\right|^2 - \left|\left|\boldsymbol{\mu}_+\right|\right|^2 \\
&= \langle\mathbf{w}, \mathbf{x}\rangle + b
\end{aligned}
$$

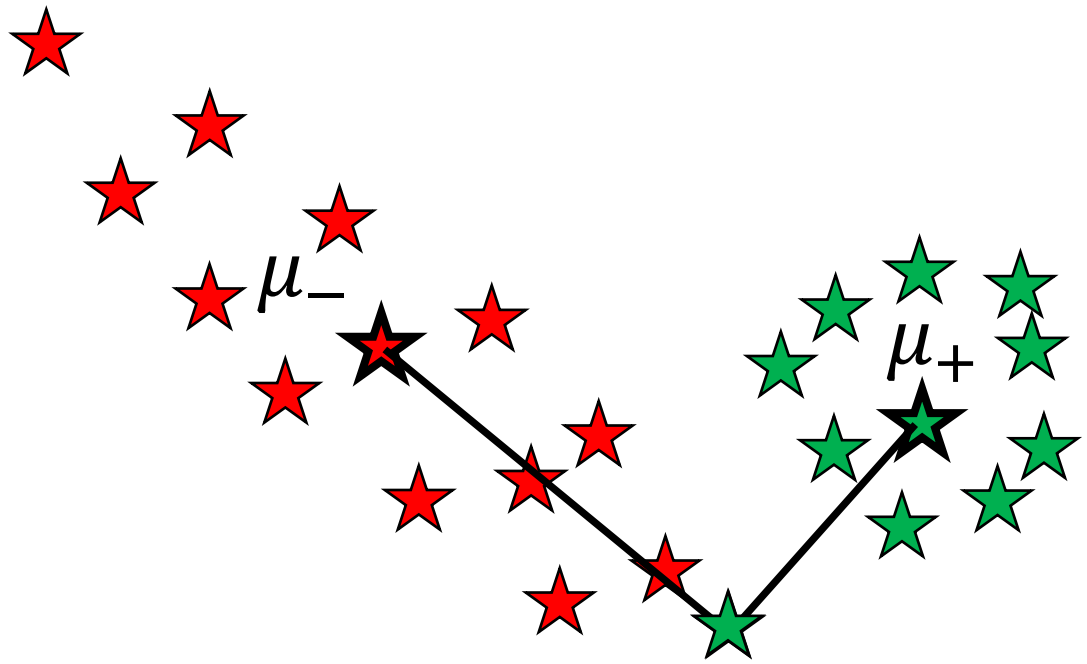
- Thus LwP with Euclidean distance is equivalent to a linear model with
  - Weight vector $\mathbf{w} = 2(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)$
  - Bias term $b = \left|\left|\boldsymbol{\mu}_-\right|\right|^2 - \left|\left|\boldsymbol{\mu}_+\right|\right|^2$

- Prediction rule therefore is: Predict +1 if $\langle\mathbf{w}, \mathbf{x}\rangle + b > 0$, else predict -1

Will look at linear models more formally and in more detail later

LwP: Some Failure Cases

- Here is a case where LwP with Euclidean distance may not work well



Can use feature scaling or use Mahalanobis distance to handle such cases (will discuss this in the next lecture)

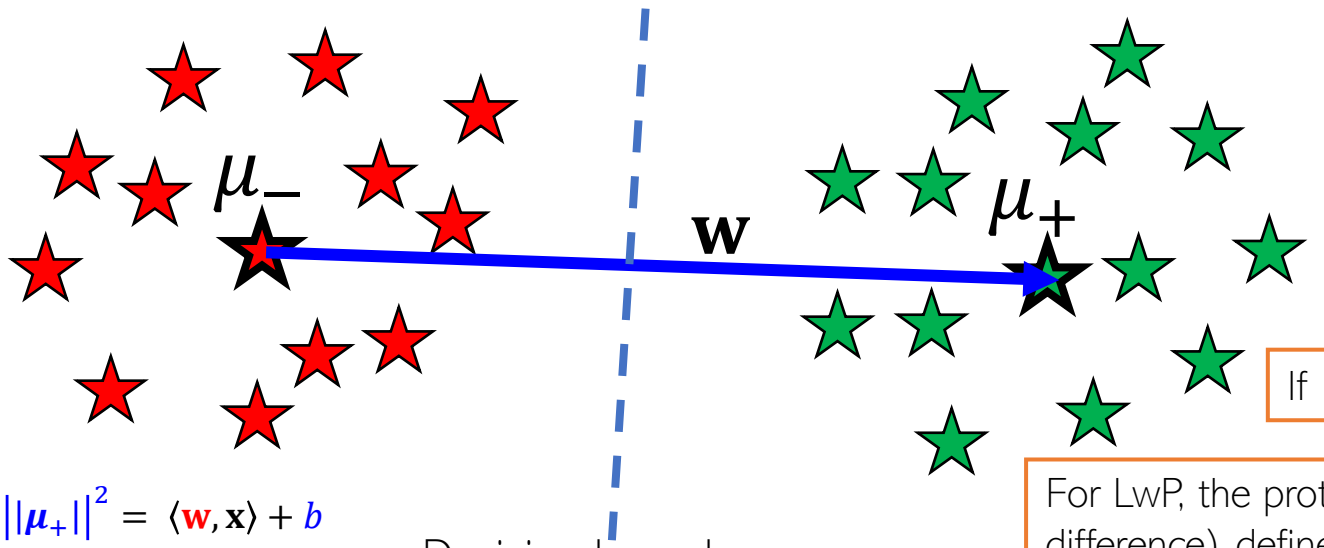
- In general, if classes are not equisized and spherical, LwP with Euclidean distance will usually not work well (but improvements possible; will discuss later)

LwP: Some Key Aspects

- Very simple, interpretable, and lightweight model
  - Just requires computing and storing the class prototype vectors
- Works with any number of classes (thus for multi-class classification as well)
- Can be generalized in various ways to improve it further, e.g.,
  - Modeling each class by a probability distribution rather than just a prototype vector
  - Using distances other than the standard Euclidean distance (e.g., Mahalanobis)
- With a learned distance function, can work very well even with very few examples from each class (used in some "few-shot learning" models nowadays – if interested, please refer to "Prototypical Networks for Few-shot Learning")

Learning with Prototypes (LwP)

$$\mu_- = \frac{1}{N_-} \sum_{y_n=-1} \mathbf{x}_n$$



$$\mu_+ = \frac{1}{N_+} \sum_{y_n=+1} \mathbf{x}_n$$

$$\mathbf{w} = \boldsymbol{\mu}_+ - \boldsymbol{\mu}_-$$

If Euclidean distance used

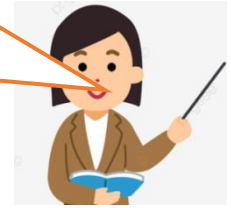
Prediction rule for LwP (for binary classification with Euclidean distance)

$$f(\mathbf{x}) = 2\langle \boldsymbol{\mu}_+ - \boldsymbol{\mu}_-, \mathbf{x} \rangle + ||\boldsymbol{\mu}_-||^2 - ||\boldsymbol{\mu}_+||^2 = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

($f(\mathbf{x}) > 0$ then predict +1 otherwise -1)

Decision boundary (perpendicular bisector of line joining the class prototype vectors)

For LwP, the prototype vectors (or their difference) define the "model". $\boldsymbol{\mu}_+$ and $\boldsymbol{\mu}_-$ (or just $\mathbf{w}$ in the Euclidean distance case) are the model parameters.



Exercise: Show that for the bin. classfn case

$$f(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n \langle \mathbf{x}_n, \mathbf{x} \rangle + b$$

Note: Even though $f(\mathbf{x})$ can be expressed in this form, if $N > D$, this may be more expensive to compute (O(N) time) as compared to $\langle \mathbf{w}, \mathbf{x} \rangle + b$ (O(D) time).

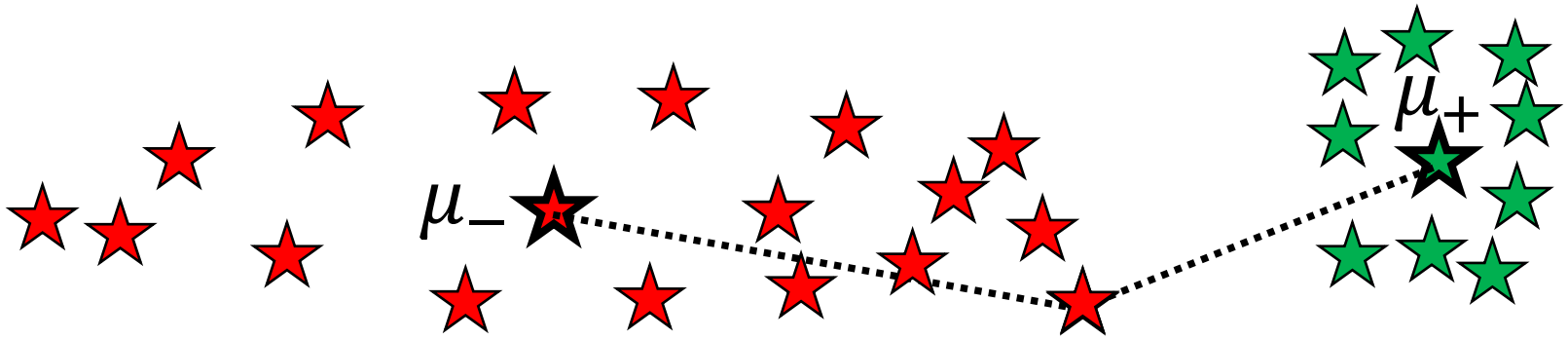
Can throw away training data after computing the prototypes and just need to keep the model parameters for the test time in such "parametric" models

So the "score" of a test point $\mathbf{x}$ is a weighted sum of its similarities with each of the N training inputs. Many supervised learning models have $f(\mathbf{x})$ in this form as we will see later

However the form $f(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n \langle \mathbf{x}_n, \mathbf{x} \rangle + b$ is still very useful as we will see later when we discuss kernel methods

Improving LwP when classes are complex-shaped

- Using weighted Euclidean or Mahalanobis distance can sometimes help



$$d_w(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{D} w_i (a_i - b_i)^2}$$

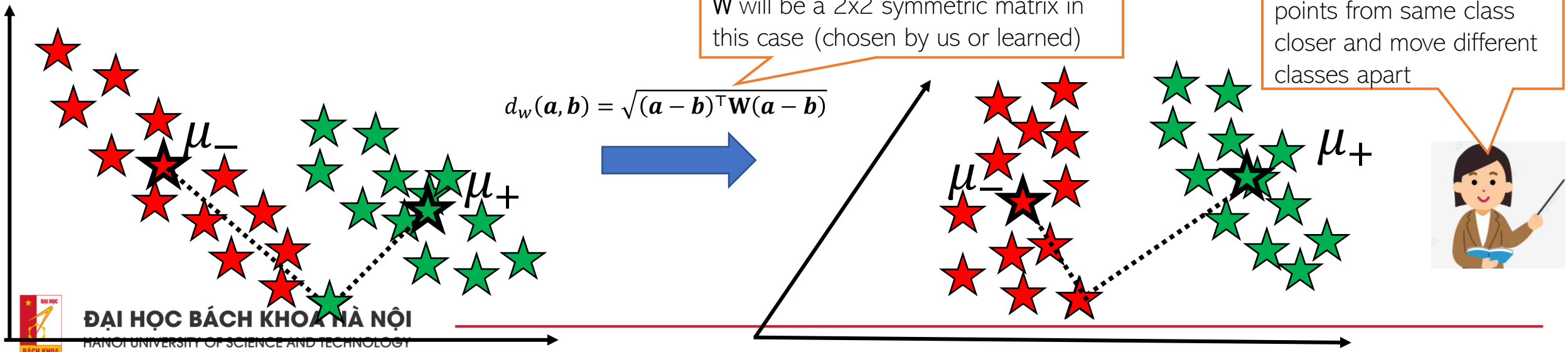
Use a smaller w_i for the horizontal axis feature in this example

- Note: Mahalanobis distance also has the effect of rotating the axes which helps

W will be a 2x2 symmetric matrix in this case (chosen by us or learned)

$$d_w(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{(\boldsymbol{a} - \boldsymbol{b})^\top \mathbf{W} (\boldsymbol{a} - \boldsymbol{b})}$$

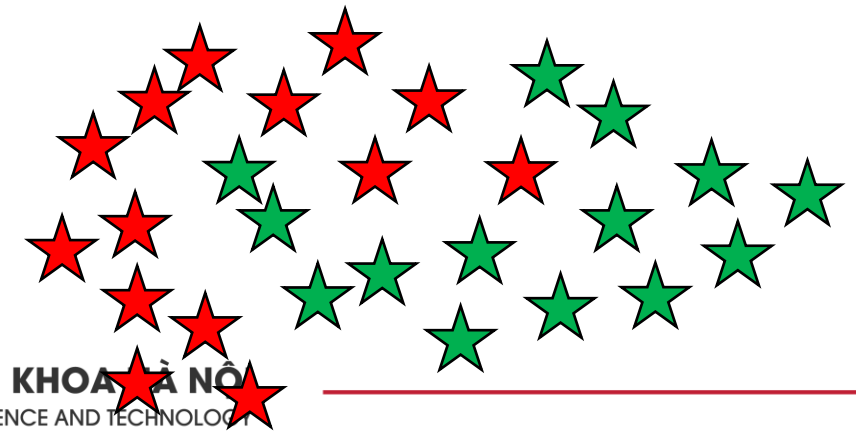
A good W will help bring points from same class closer and move different classes apart



Improving LwP when classes are complex-shaped

- Even with weighted Euclidean or Mahalanobis dist, LwP still a linear classifier ☹
- **Exercise:** Prove the above fact. You may use the following hint
  - Mahalanobis dist can be written as $d_{\boldsymbol{w}}(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{(\boldsymbol{a}-\boldsymbol{b})^{\top}\mathbf{W}(\boldsymbol{a}-\boldsymbol{b})}$
  - $\mathbf{W}$ is a symmetric matrix and thus can be written as $\mathbf{A}\mathbf{A}^{\top}$ for any matrix $\mathbf{A}$
  - Showing for Mahalabonis is enough. Weighted Euclidean is a special case with diag $\mathbf{W}$

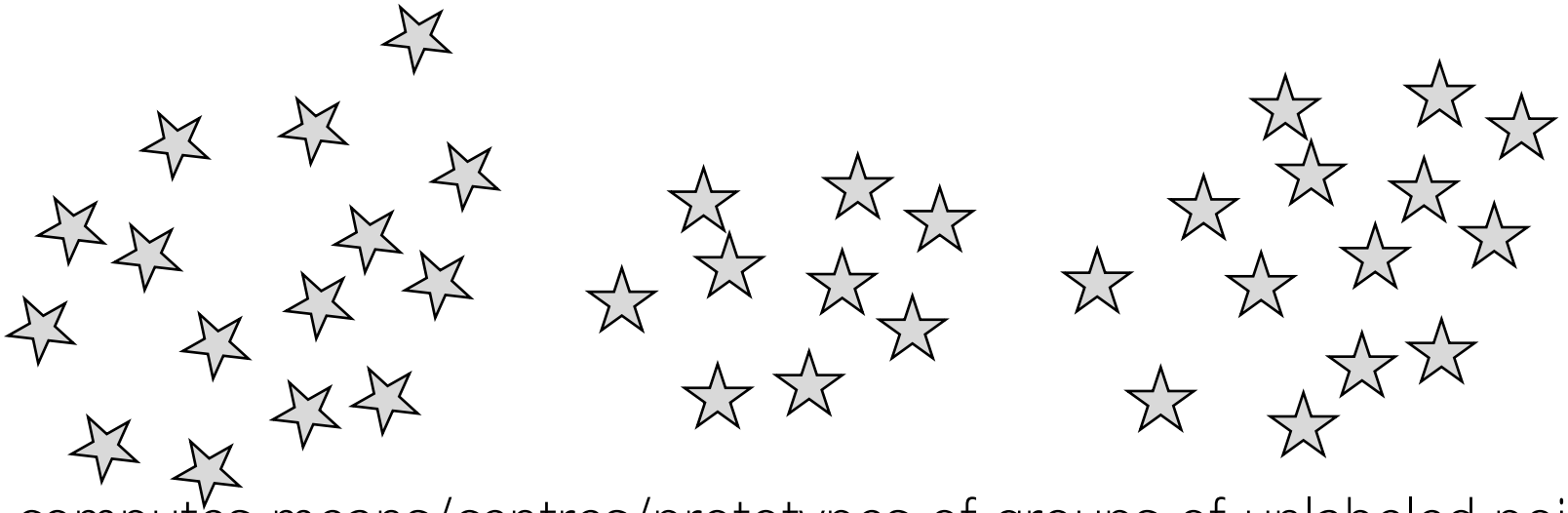
- LwP can be extended to learn nonlinear decision boundaries if we use nonlinear distances/similarities (more on this when we talk about kernels)



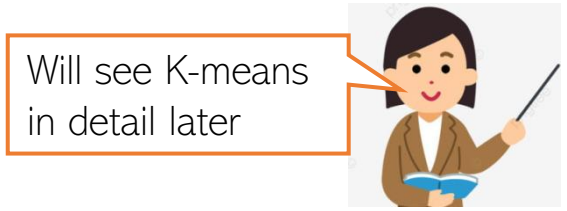
Note: Modeling each class by not just a mean by a probability distribution can also help in learning nonlinear decision boundaries. More on this when we discuss probabilistic models for classification

LwP as a subroutine in other ML models

- For data-clustering (unsupervised learning), *K*-means clustering is a popular algo



- *K*-means also computes means/centres/prototypes of groups of unlabeled points
- Harder than LwP since labels are unknown. But we can do the following
  - Guess the label of each point, compute means using guess labels
  - Refine labels using these means (assign each point to the current closest mean)
  - Repeat until means don't change anymore
- Many other models also use LwP as a subroutine



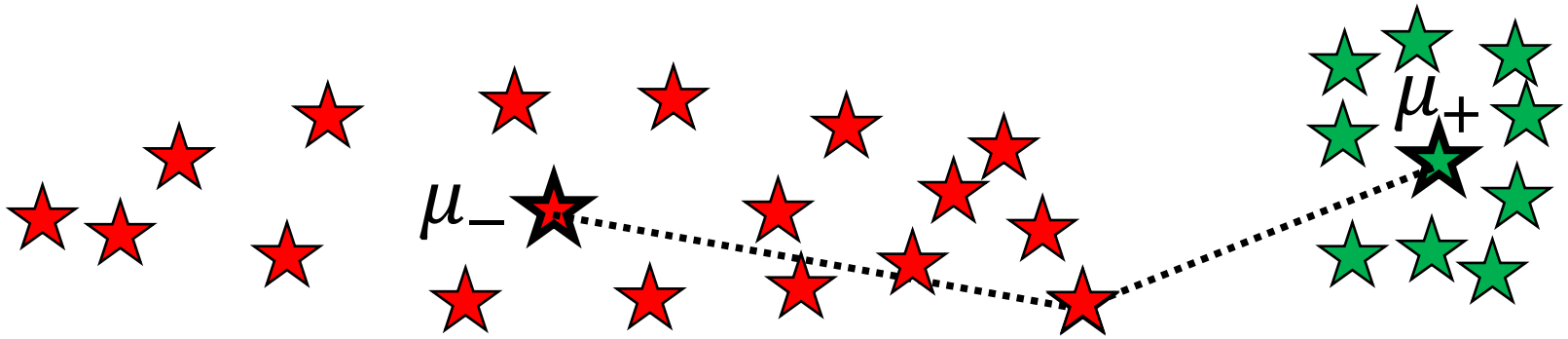
Next Lecture

- Nearest Neighbors

Exotic distances and nearest neighbors

Improving LwP when classes are complex-shaped

- Using weighted Euclidean or Mahalanobis distance can sometimes help



$$d_w(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{D} w_i (a_i - b_i)^2}$$

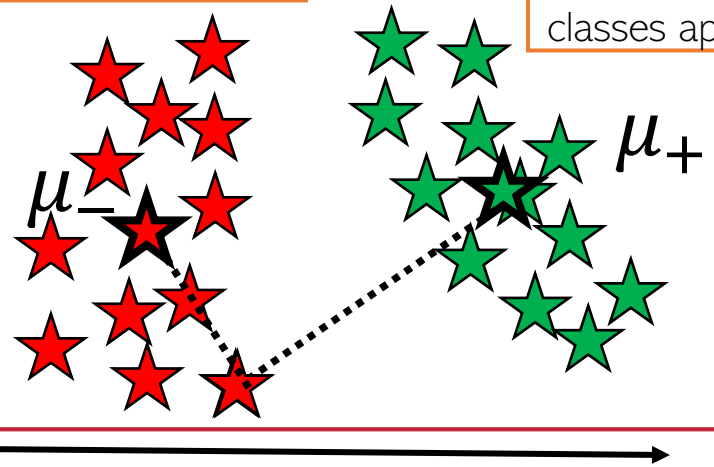
Use a smaller w_i for the horizontal axis feature in this example

- Note: Mahalanobis distance also has the effect of rotating the axes which helps

W will be a 2x2 symmetric matrix in this case (chosen by us or learned)

$$d_w(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{(\boldsymbol{a} - \boldsymbol{b})^\top \mathbf{W} (\boldsymbol{a} - \boldsymbol{b})}$$

A good W will help bring points from same class closer and move different classes apart



What is Mahalanobis Distance?

- Recall

$$\text{Euclidean } D(\mathbf{a},\mathbf{b}) = \sqrt{(\mathbf{a}-\mathbf{b})^T(\mathbf{a}-\mathbf{b})}$$

- And its generalization
  - Where W is a diagonal matrix

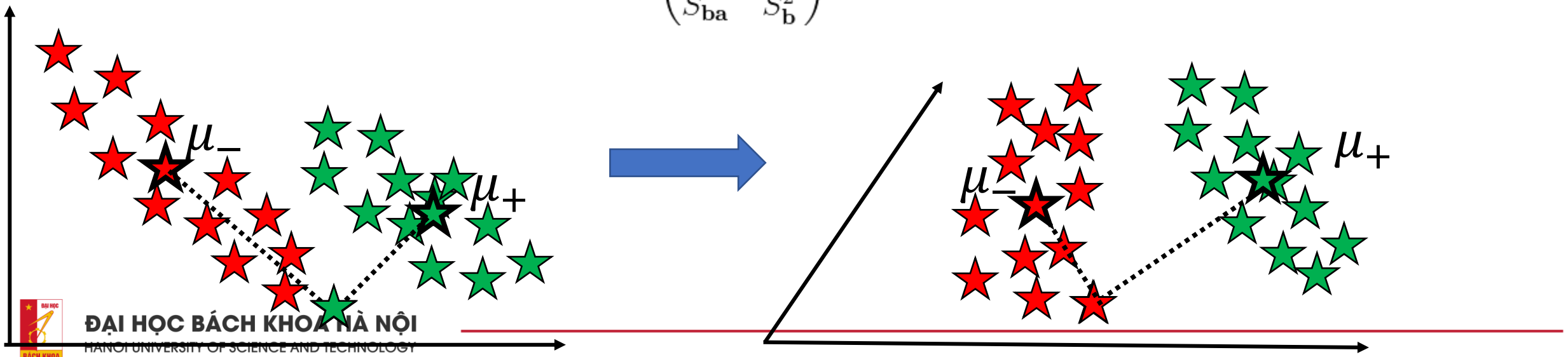
$$\text{Weighted Euclidean } D(\mathbf{a},\mathbf{b}) = \sqrt{(\mathbf{a}-\mathbf{b})^T\mathbf{W}(\mathbf{a}-\mathbf{b})}$$

- The Mahalanobis distance further generalizes the weighted Euclidean distance

- Here, S is the covariance ma

$$\text{Mahalanobis } D(\mathbf{a},\mathbf{b}) = \sqrt{(\mathbf{a}-\mathbf{b})^T\mathbf{S}^{-1}(\mathbf{a}-\mathbf{b})}$$

$$\begin{pmatrix} S_{\mathbf{a}}^2 & S_{\mathbf{ab}} \\ S_{\mathbf{ba}} & S_{\mathbf{b}}^2 \end{pmatrix}$$



Supervised Learning using Nearest Neighbors

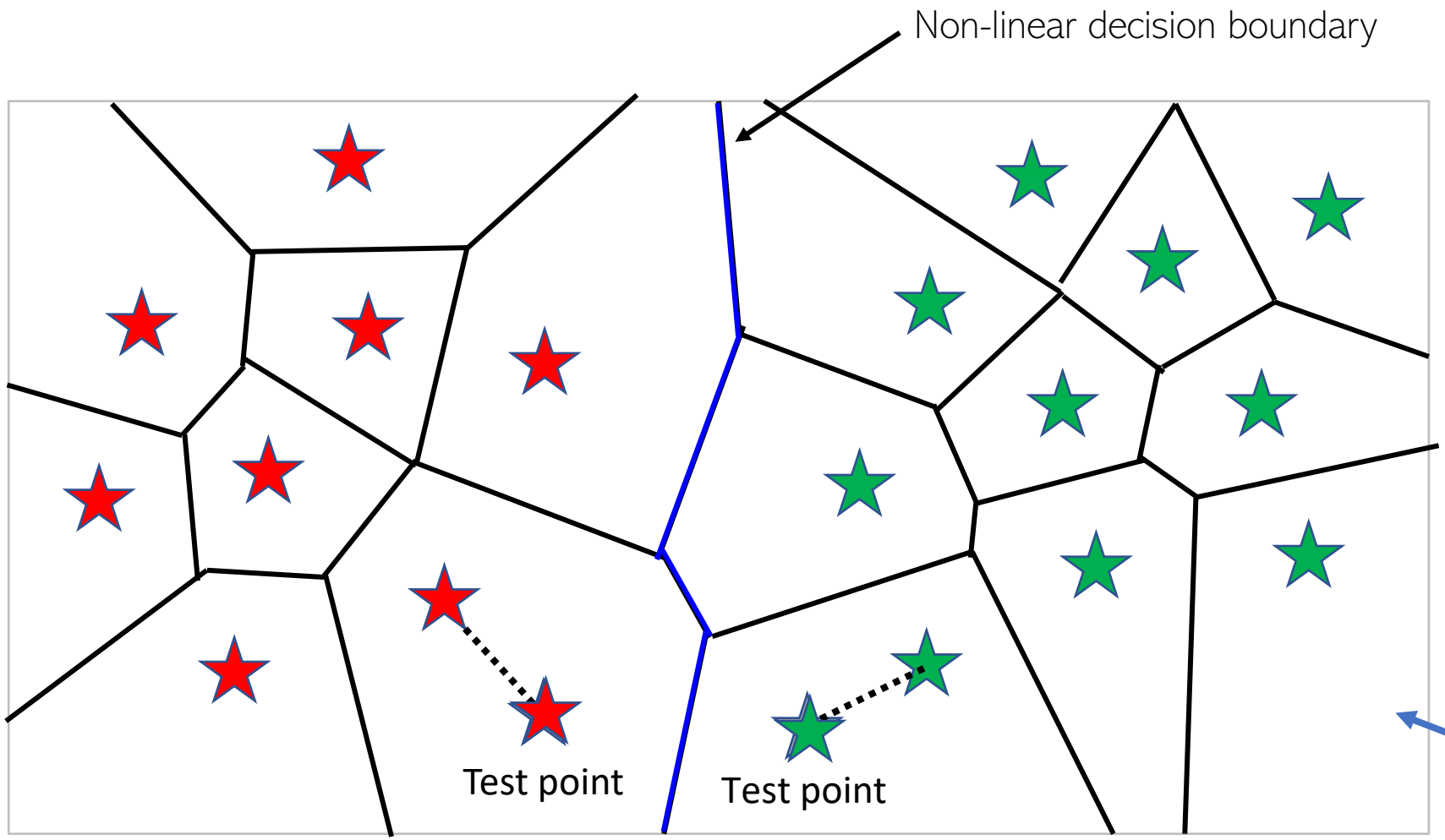
Nearest Neighbors

- Another supervised learning technique based on computing distances
- Very simple idea. Simply do the following at test time
  - Compute distance of the test point from all the training points
  - Sort the distances to find the "nearest" input(s) in training data
  - Predict the label using majority or avg label of these inputs
- Can use Euclidean or other dist (e.g., Mahalanobis). Choice important just like LwP
- Unlike LwP which does prototype based comparison, nearest neighbors method looks at the labels of individual training inputs to make prediction
- Applicable to both classification as well as regression (LwP only works for classification)

Nearest Neighbors for Classification

Reference material

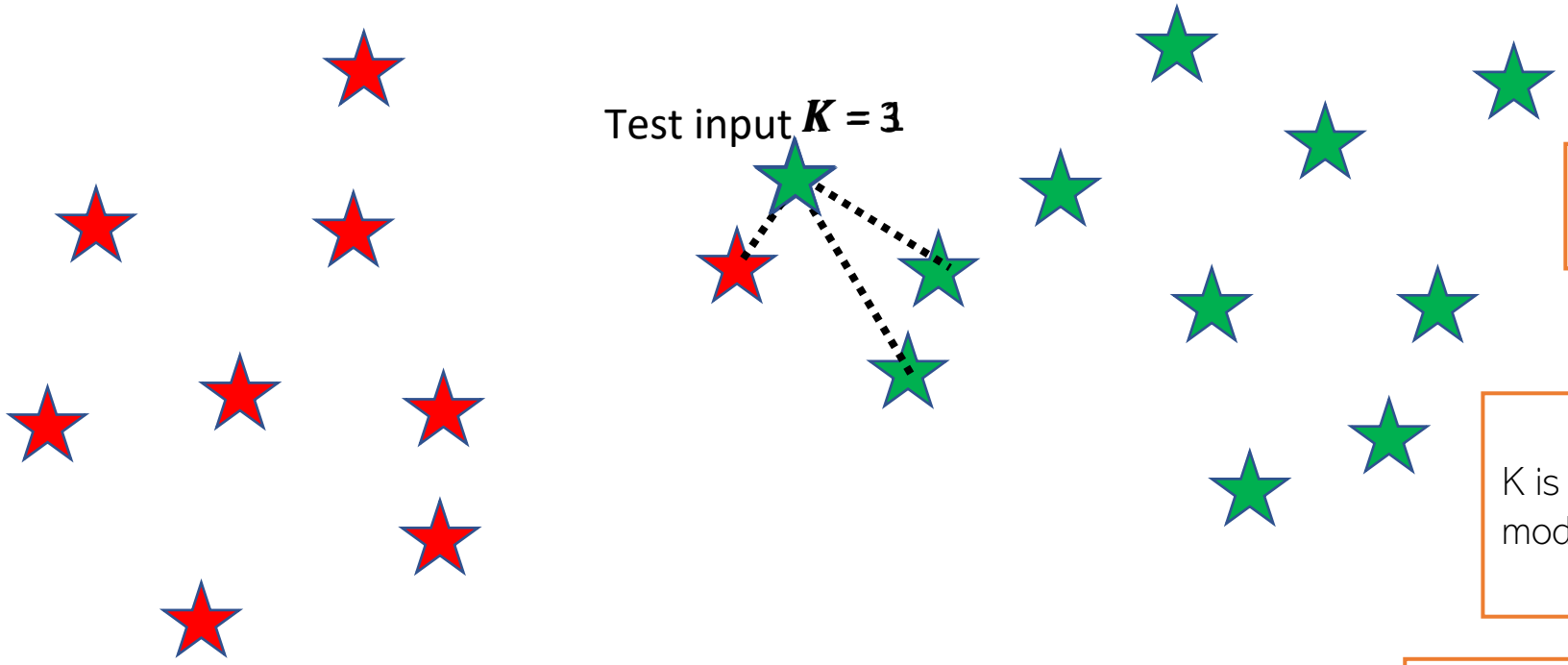
Nearest Neighbor (or "One" Nearest Neighbor)



Nearest neighbour approach induces a Voronoi tessellation/partition of the input space (all test points falling in a cell will get the label of the training input in that cell)

K Nearest Neighbors (*K*NN)

- In many cases, it helps to look at not one but $K > 1$ nearest neighbors



How to pick the "right" K value?

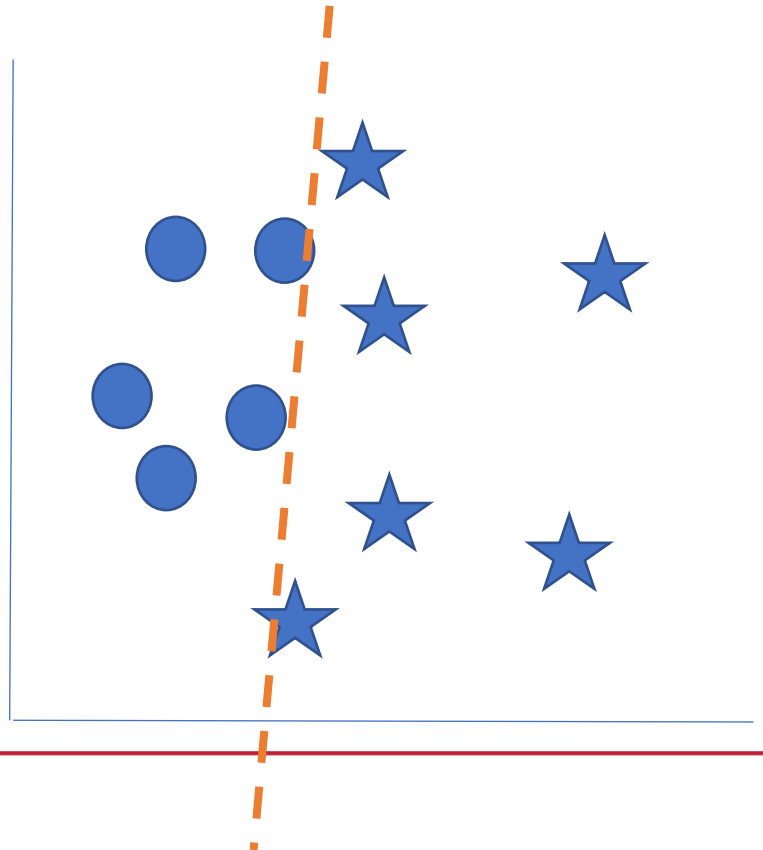
K is a free parameter in the model

Also, K should ideally be an odd number to avoid ties

- Essentially, taking more votes helps!
  - Also leads to smoother decision boundaries (less chances of overfitting on training data)

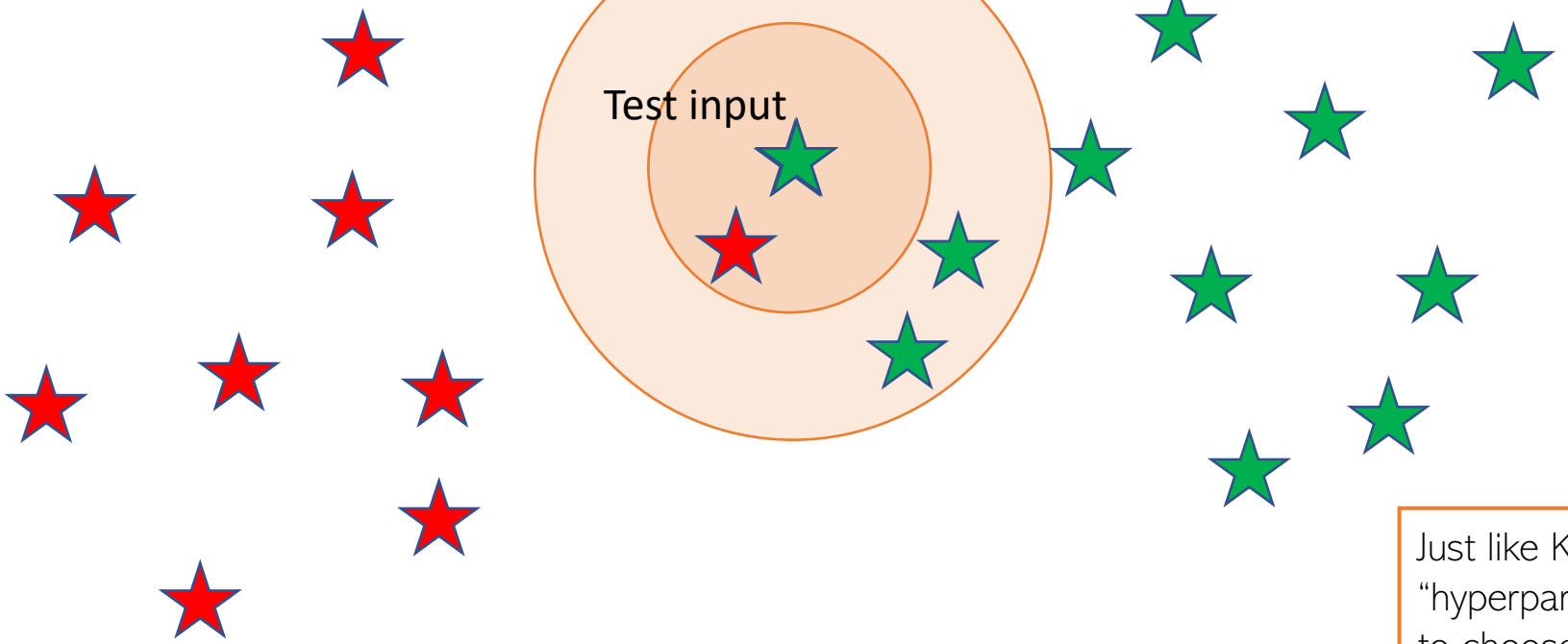
Setting parameter values

- The black magic of machine learning
  - “Predictions are hard, especially about the future” – Yogi Berra
- Basic idea: find parameter values for which you make the fewest possible mistakes on training data
- Pray that training data is representative
- If it’s not, your model will work badly
- We will see some nice math that helps
- Soon!



ϵ-Ball Nearest Neighbors (ϵ-NN)

- Rather than looking at a fixed number K of neighbors, can look inside a ball of a given radius ϵ, around the test input

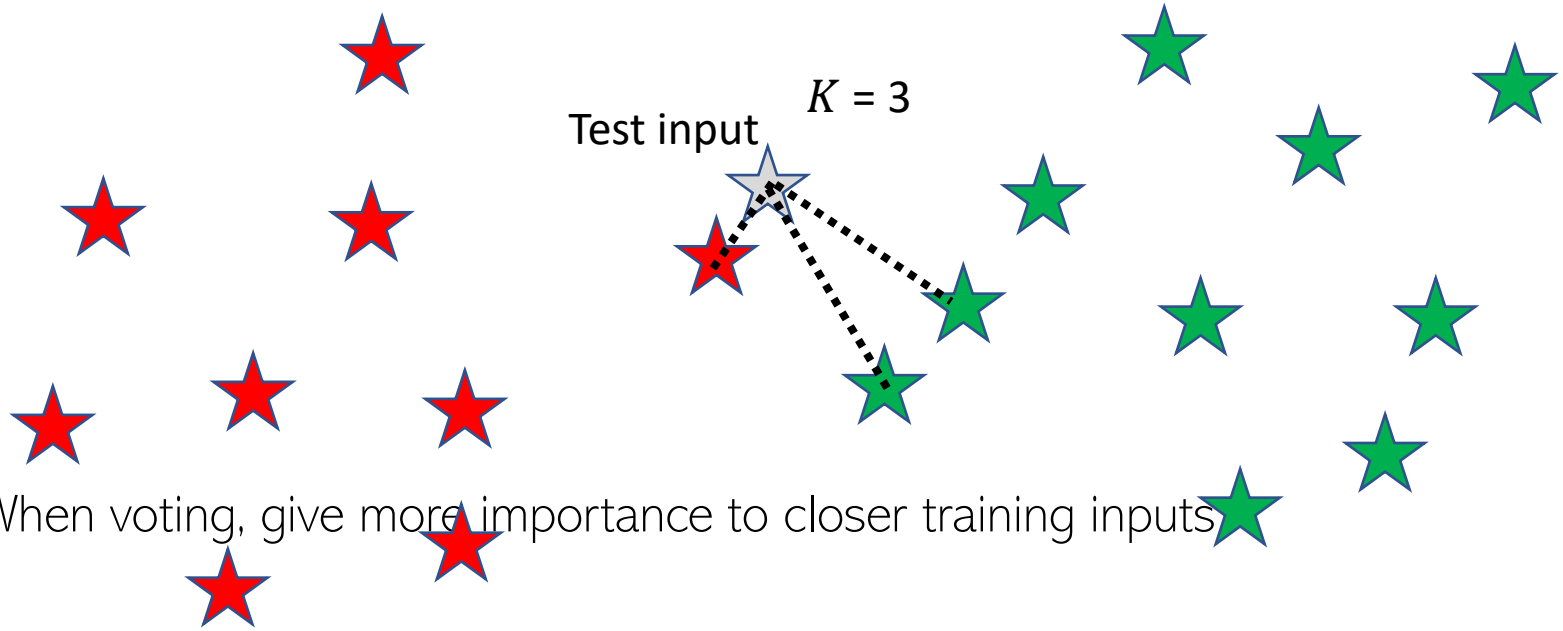


So changing ϵ may change the prediction. How to pick the "right" ϵ value?

Just like K, ϵ is also a "hyperparameter". One way to choose it is using "cross-validation" (will see shortly)

Distance-weighted *K*NN and ϵ-NN

- The standard KNN and ϵ-NN treat all nearest neighbors equally (all vote equally)

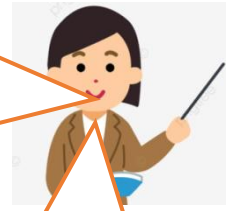


- An improvement: When voting, give more importance to closer training inputs

Unweighted KNN prediction: $\frac{1}{3}$ ★ + $\frac{1}{3}$ ★ + $\frac{1}{3}$ ★ = ★

Weighted KNN prediction: $\frac{3}{5}$ ★ + $\frac{1}{5}$ ★ + $\frac{1}{5}$ ★ = ★

In weighted approach, a single red training input is being given 3 times more importance than the other two green inputs since it is sort of "three times" closer to the test input than the other two green inputs

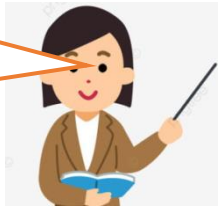


ϵ-NN can also be made weighted likewise

KNN/ϵ-NN for Other Supervised Learning Problems

- Can apply KNN/ϵ-NN for other supervised learning problems as well, such as
  - Multi-class classification
  - Regression
  - Tagging/multi-label learning

We can also try the weighted versions for such problems, just like we did in the case of binary classification



- For multi-class, simply used the same majority rule like in binary classfication case
  - Just a simple difference that now we have more than 2 classes
- For regression, simply compute the average of the outputs of nearest neighbors
- For multi-label learning, each output is a binary vector (presence/absence of tag)
  - Just compute the average of the binary vectors
  - Result won't be a binary vector but we can report the best tags based on magnitudes

KNN Prediction Rule: The Mathematical Form

- Let's denote the set of K nearest neighbors of an input $\mathbf{x}$ by $N_K(\mathbf{x})$

- The unweighted KNN prediction $\mathbf{y}$ for a test input $\mathbf{x}$ can be written as

$$\mathbf{y} = \frac{1}{K} \sum_{i \in N_K(\mathbf{x})} \mathbf{y}_i$$

For discrete labels with 5 possible values, the one-hot representation will be an all zeros vector of size 5, except a single 1 denoting the value of the discrete label, e.g., if label = 3 then one-hot vector = [0,0,1,0,0]

- This form makes direct sense of regression and for cases where the each output is a vector (e.g., multi-class classification where each output is a discrete value which can be represented as a one-hot vector, or tagging/multi-label classification where each output is a binary vector)
  - For binary classification, assuming labels as +1/-1, we predict $\mathbf{sign}(\frac{1}{K} \sum_{i \in N_K(\mathbf{x})} \mathbf{y}_i)$

Nearest Neighbours: Some Comments

- An old, classic but still very widely used algorithm
  - Competitive with state-of-the-art approaches most of the time
- Can work very well in practice with the right distance function
  - How do we pick the right distance function?
- Requires lots of storage (need to keep all the training data at test time)
- Prediction step can be slow at test time
  - For each test point, need to compute its distance from all the training points
- Clever data-structures or data-summarization techniques can provide speed-ups

Thanks