

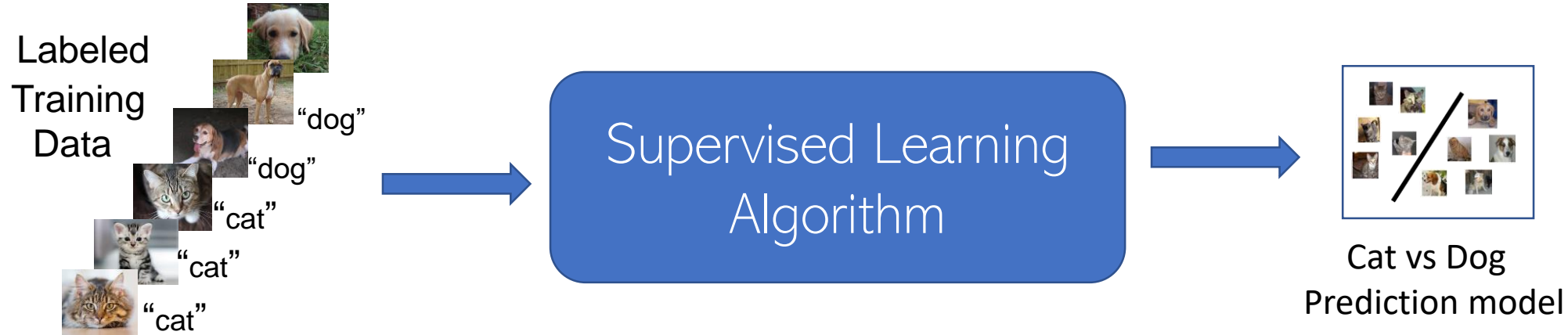
Getting Started with Supervised Learning, Learning by Computing Distances (1)

CS771: Introduction to Machine Learning

Piyush Rai

Supervised Learning

2



Important: In ML (not just sup. learning but also unsup. and RL), training and test datasets should be “similar” (we don’t like “out-of-syllabus” questions in exams 😊)



A test image



Cat vs Dog Prediction model

Predicted Label
(cat/dog)

In the above example, it means that we can’t have test data with BnW images or sketches of cats and dogs

More formally, the train and test data distributions should be the same

Of course not. 😊 Many ML techniques exist to handle such situations (a bit advanced but will touch upon those later)

Does it mean ML is useless if this assumption is violated?

Will give you Just the names for now – **domain adaptation**, **covariate shift**, **transfer learning**, etc



Some Types of Supervised Learning Problems

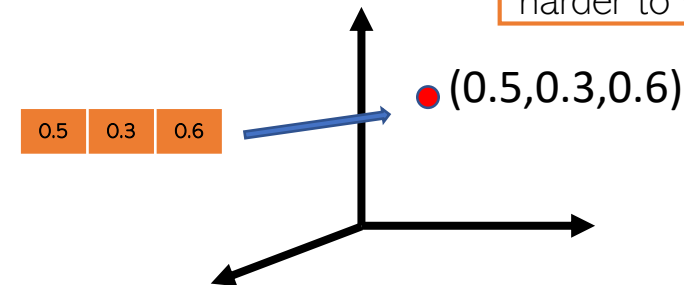
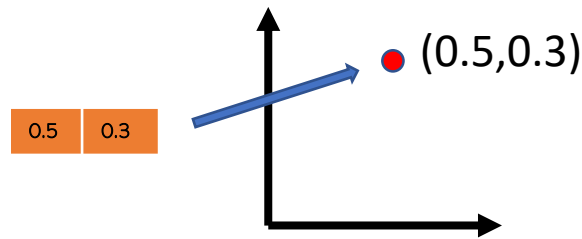
- Consider building an ML module for an e-mail client
- Some tasks that we may want this module to perform
 - Predicting whether an email is spam or normal: [Binary Classification](#)
 - Predicting which of the many folders the email should be sent to: [Multi-class Classification](#)
 - Predicting all 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

0.5 0.3 0.6 0.1 0.2 0.5 0.9 0.2 0.1 0.5



Likewise for higher dimensions, even though harder to visualize



- Unless specified otherwise
 - Small letters in bold font will denote vectors, e.g., **x**, **a**, **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., **X**, **A**, **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 $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$, 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 μ (average or centroid) of N vectors $\{\mathbf{x}_n\}_{n=1}^N$

$$\mu = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \quad (\text{of the same size as each } \mathbf{x}_n)$$

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

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

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

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

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

- Also the Euclidean distance of \mathbf{a} from origin
- Note: Euclidean norm is also called L2 norm



Computing Distances

7

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

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

Sqrt of Inner product of the difference vector!

Another expression in terms of inner products of individual vectors

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



Useful tip: Can achieve the effect of feature scaling (recall last lecture) by using weighted Euclidean distances!

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

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

Note: If \mathbf{W} is a $D \times D$ symmetric matrix then it is called the **Mahalanobis distance** (more on this later)

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



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)

Yes. Another, although less commonly used, distance is the L-infinity distance (equals to max of abs-value of element-wise difference between two vectors)

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

Apart from L2 and L1, there other ways of defining distances?





Our First Supervised Learner



Prelude: A Very Primitive Classifier

9

- 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



$d(\text{Test image}, \text{cat}) < d(\text{Test image}, \text{dog}) ?$ 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

Excellent question! Glad you asked! 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

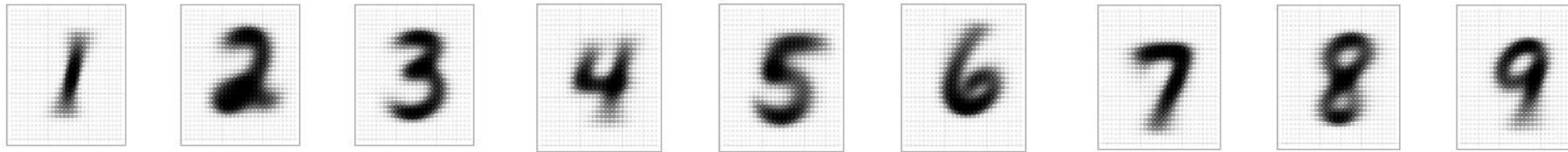


- We will consider two approaches to do this
 - Learning with Prototypes (LwP)
 - Nearest Neighbors (NN – not “neural networks”, at least not for now 😊)
- Both LwP and NN will use multiple inputs per class but in different ways



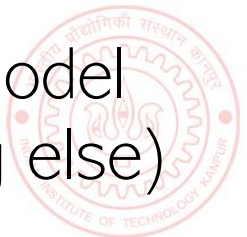
Learning with Prototypes (LwP)

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



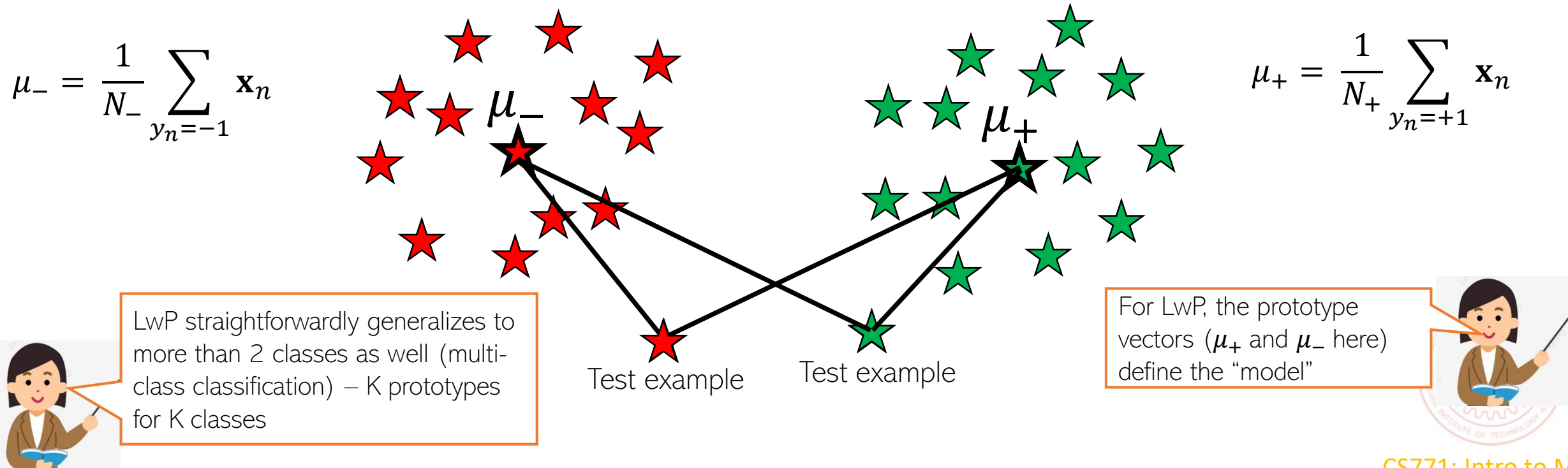
Averages (prototypes) of each of the handwritten digits 1-9

- 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)



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

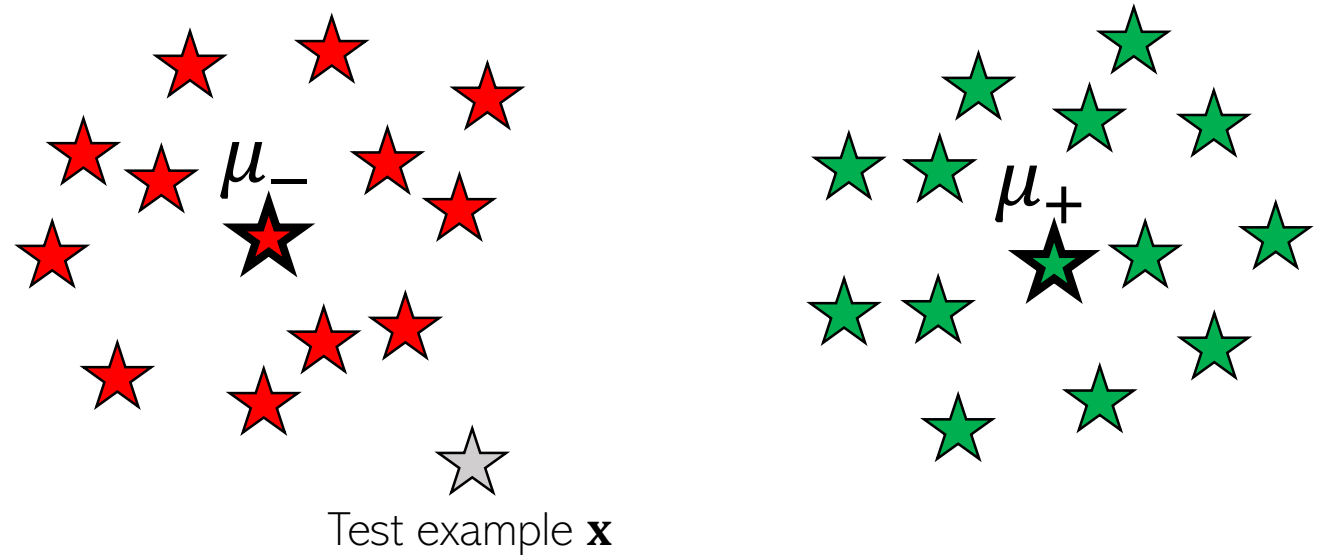


LwP: The Prediction Rule, Mathematically

- What does the prediction rule for LwP look like mathematically?
- Assume we are using Euclidean distances here

$$||\mu_- - \mathbf{x}||^2 = ||\mu_-||^2 + ||\mathbf{x}||^2 - 2\langle \mu_-, \mathbf{x} \rangle$$

$$||\mu_+ - \mathbf{x}||^2 = ||\mu_+||^2 + ||\mathbf{x}||^2 - 2\langle \mu_+, \mathbf{x} \rangle$$



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



LwP: The Prediction Rule, Mathematically

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

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

- Thus LwP with Euclidean distance is equivalent to a linear model with

- Weight vector $\mathbf{w} = 2(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)$
- Bias term $b = ||\boldsymbol{\mu}_-||^2 - ||\boldsymbol{\mu}_+||^2$

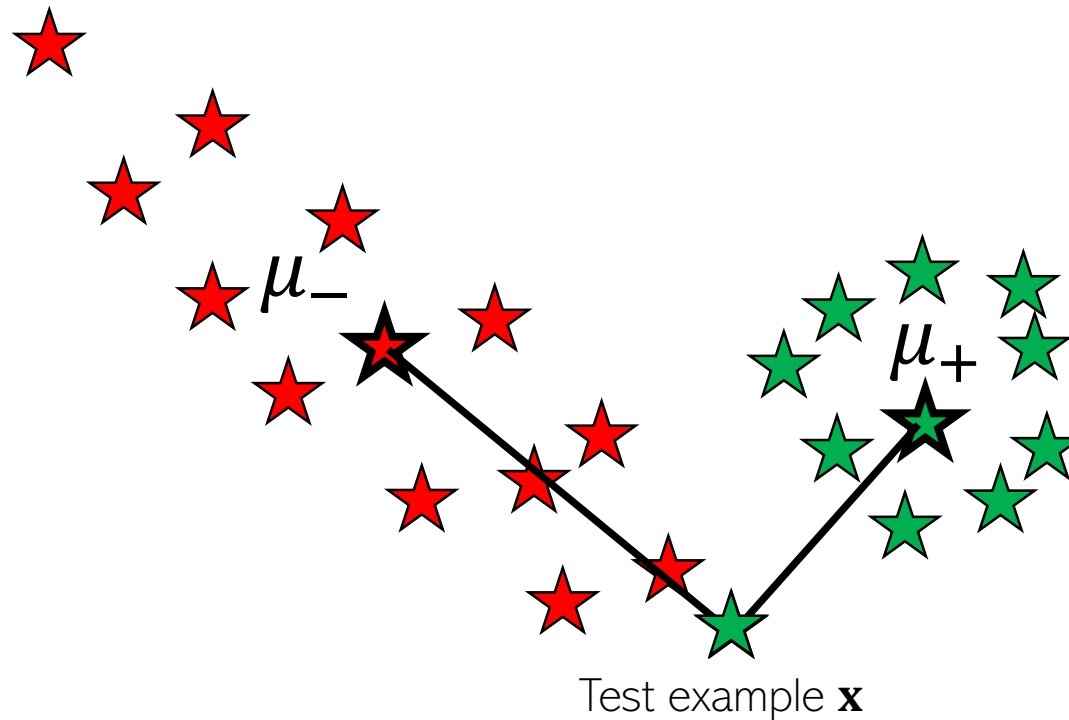
Will look at linear models more formally and in more detail later

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



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”)



Next Lecture

- Fixing LwP
- Nearest Neighbors

