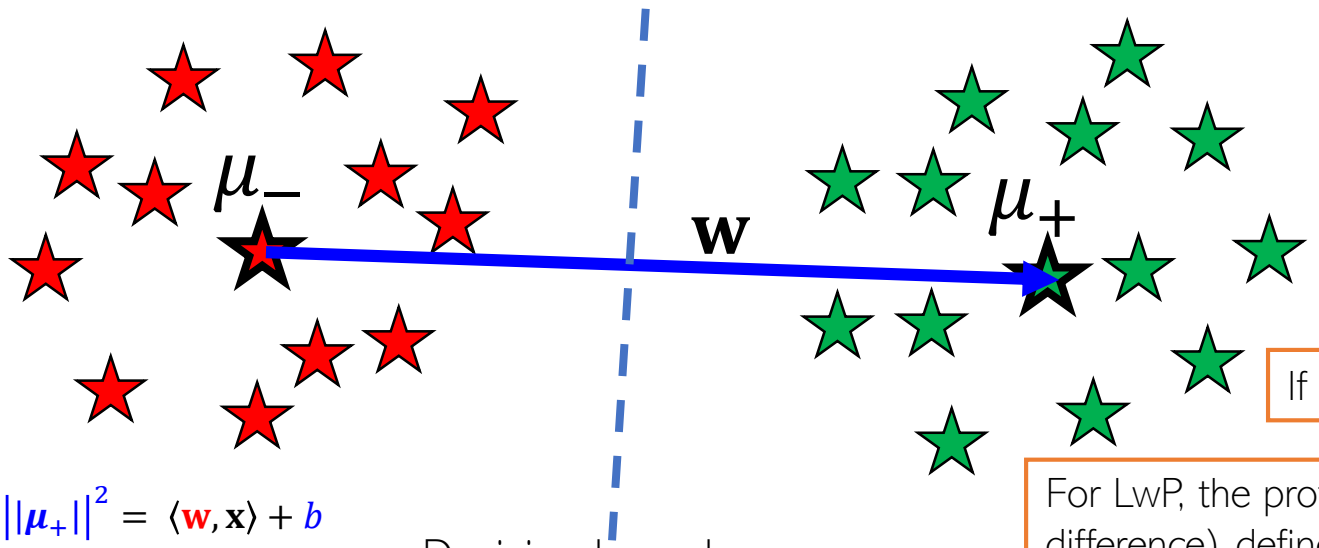
Learning by Computing Distances (2): Wrapping-up LwP, Nearest Neighbors

CS771: Introduction to Machine Learning

Piyush Rai

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).

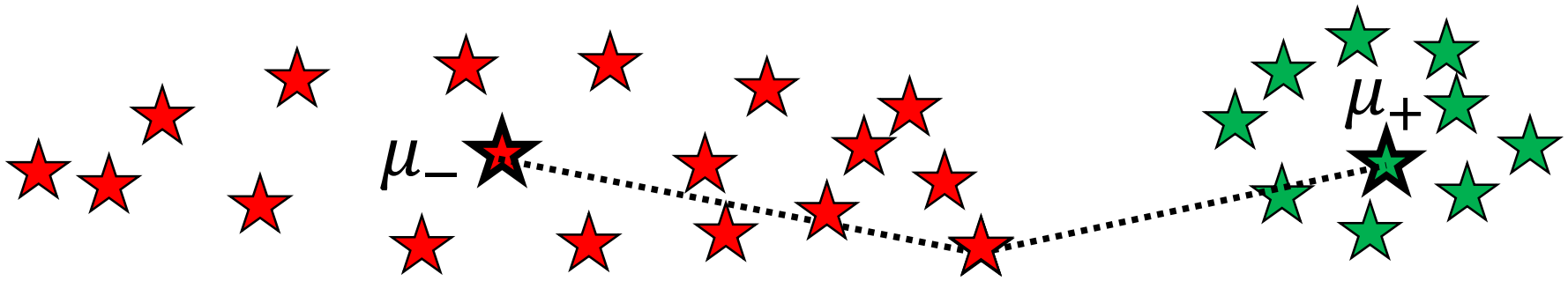
Can throw away training data after computing the prototypes and just need to keep the model parameters for the test time in such "parametric" models

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



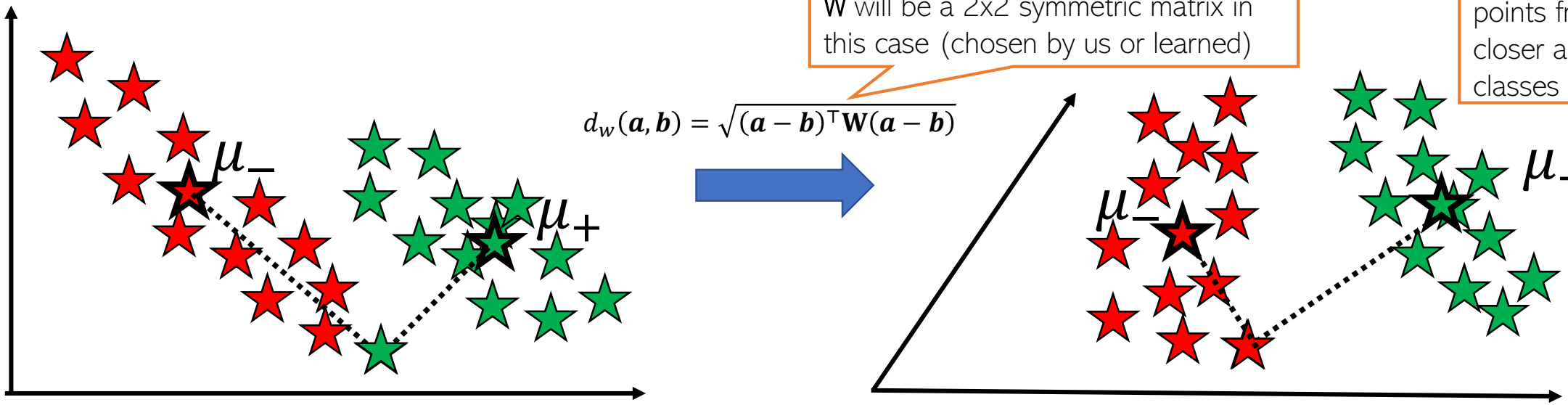
$$d_w(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{D} w_i (a_i - b_i)^2}$$

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)

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

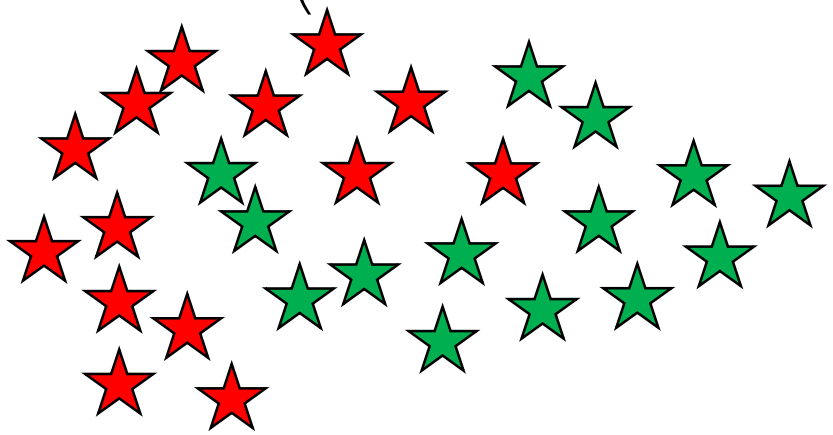


A good W will help bring points from same class closer and move different classes apart

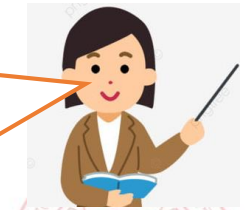
Improving LwP when classes are complex-shaped

- 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

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 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

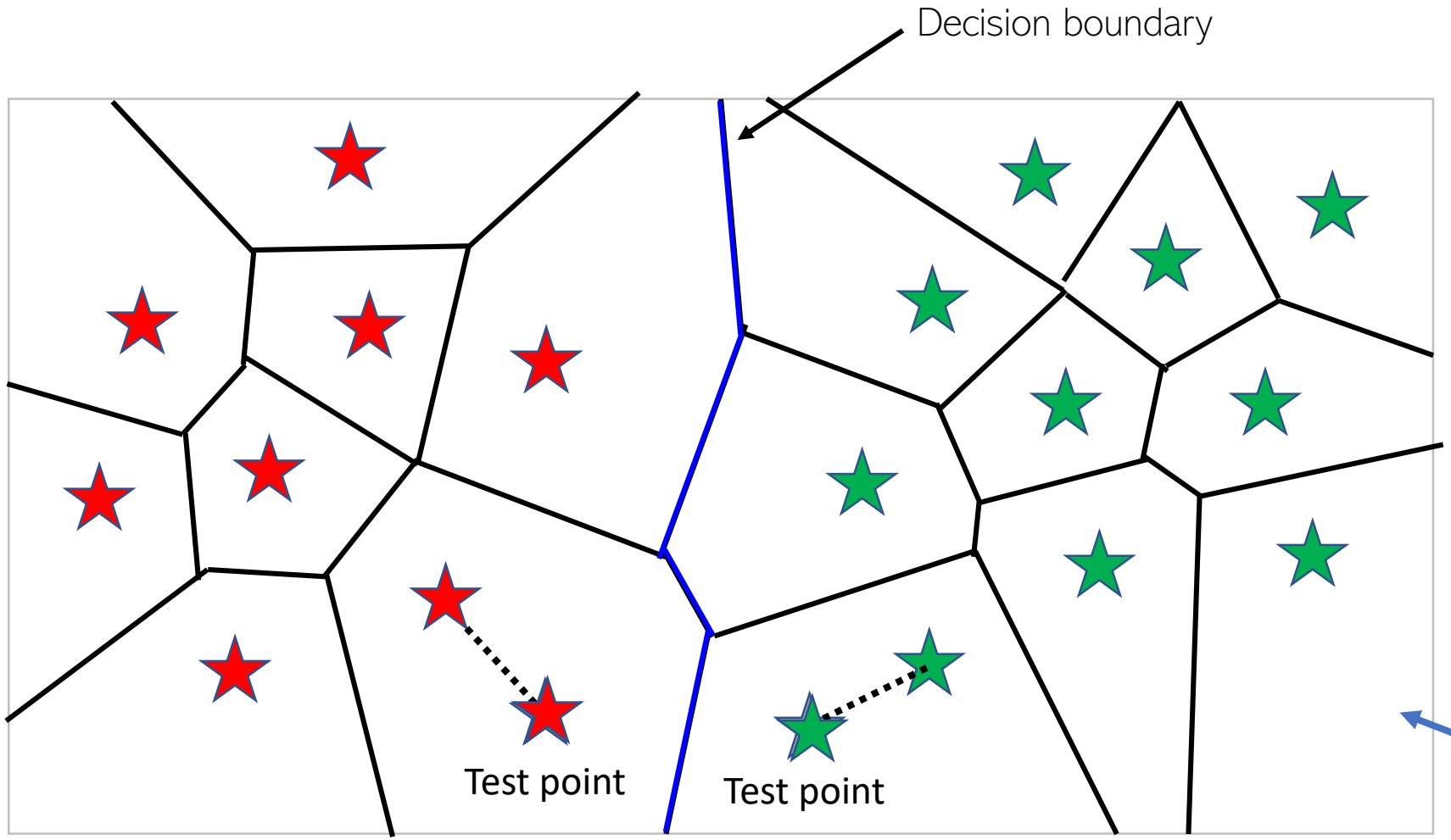
Wait. Did you say distance from ALL the training points? That's gonna be sooooo expensive! ☹

Yes, but let's not worry about that at the moment. There are ways to speed up this step

- Can use Euclidean or other dist (e.g., Mahalanobis). Choice imp 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 classifn as well as regression (LwP only works for classifn)

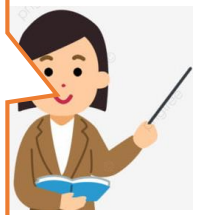
Nearest Neighbors for Classification

Nearest Neighbor (or "One" Nearest Neighbor)



Interesting. Even with Euclidean distances, it can learn nonlinear decision boundaries?

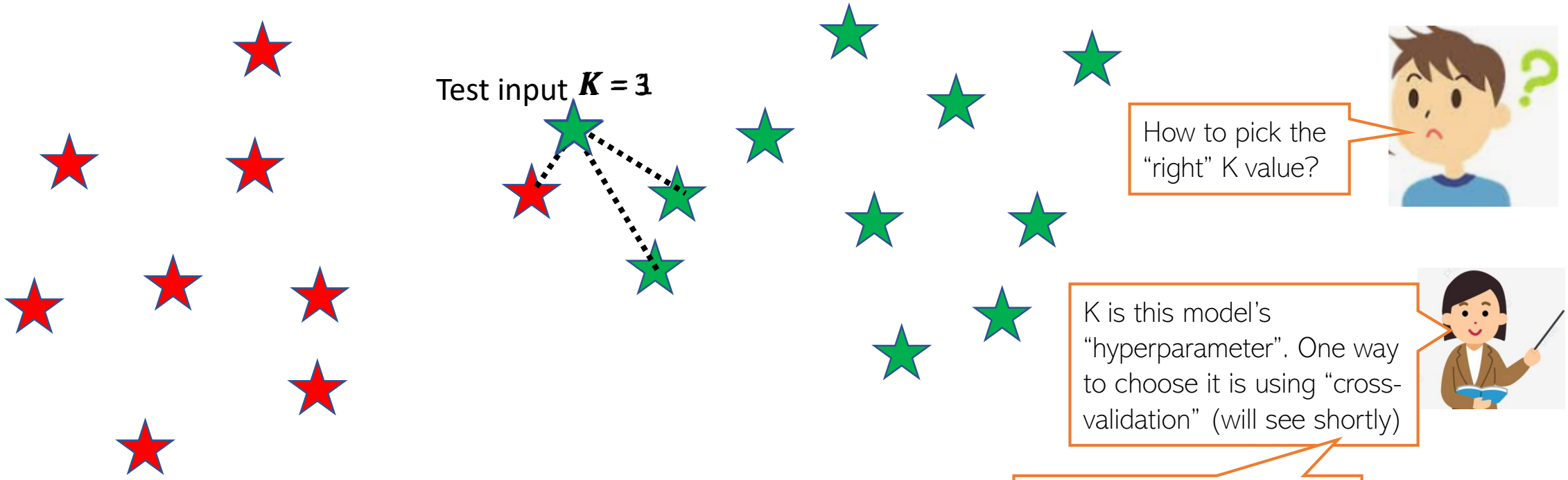
Indeed. And that's possible since it is a "local" method (looks at a local neighborhood of the test point to make prediction)



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)

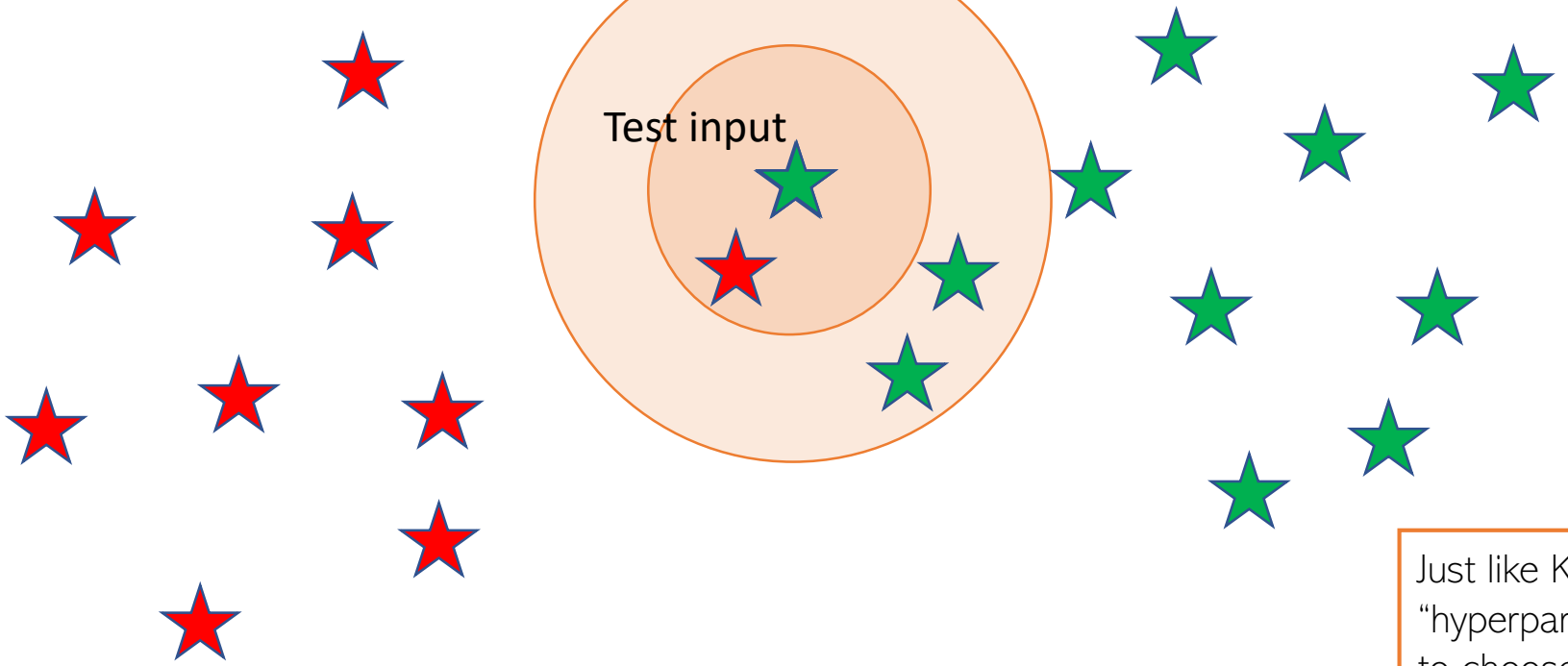
- In many cases, it helps to look at not one but $K > 1$ nearest neighbors



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

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

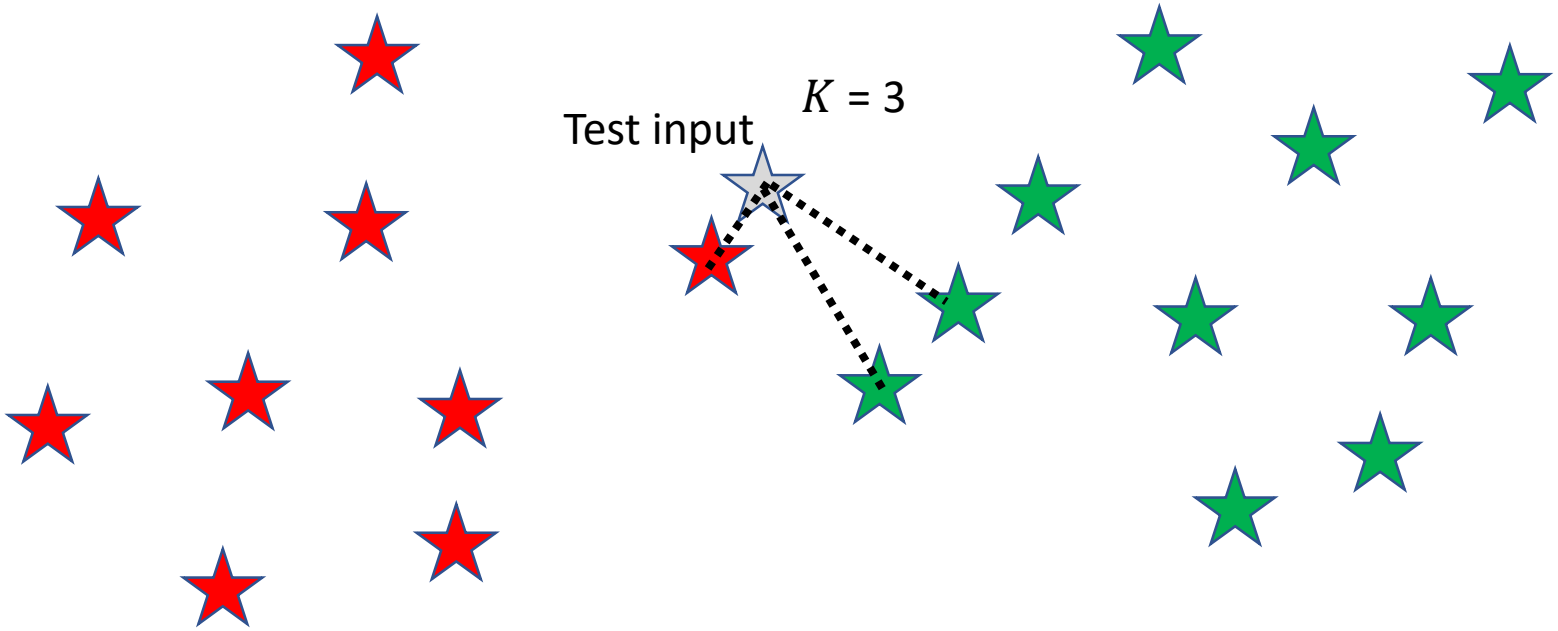


So changing ϵ may change the prediction. How to pick the "right" ϵ value?

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 classfn 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

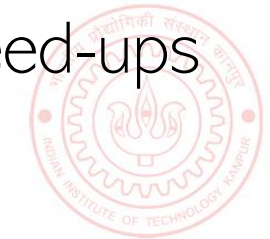
$$\mathbf{y} = \frac{1}{K} \sum_{i \in N_K(\mathbf{x})} \mathbf{y}_i$$

Assuming discrete labels with 5 possible values, the one-hot representation will be a 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 Neighbors: Some Comments

- An old, classic but still very widely used algorithm
  - Can sometimes give deep neural networks a run for their money ☺
- Can work very well in practical with the right distance function
- Comes with very nice theoretical guarantees
- Also called a memory-based or instance-based or non-parametric method
  - No "model" is learned here (unlike LwP). Prediction step uses all the training data
- 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



Next Lecture

- Hyperparameter/model selection via cross-validation
- Learning with Decision Trees