LwP (contd), Nearest Neighbors

CS771: Introduction to Machine Learning

Piyush Rai

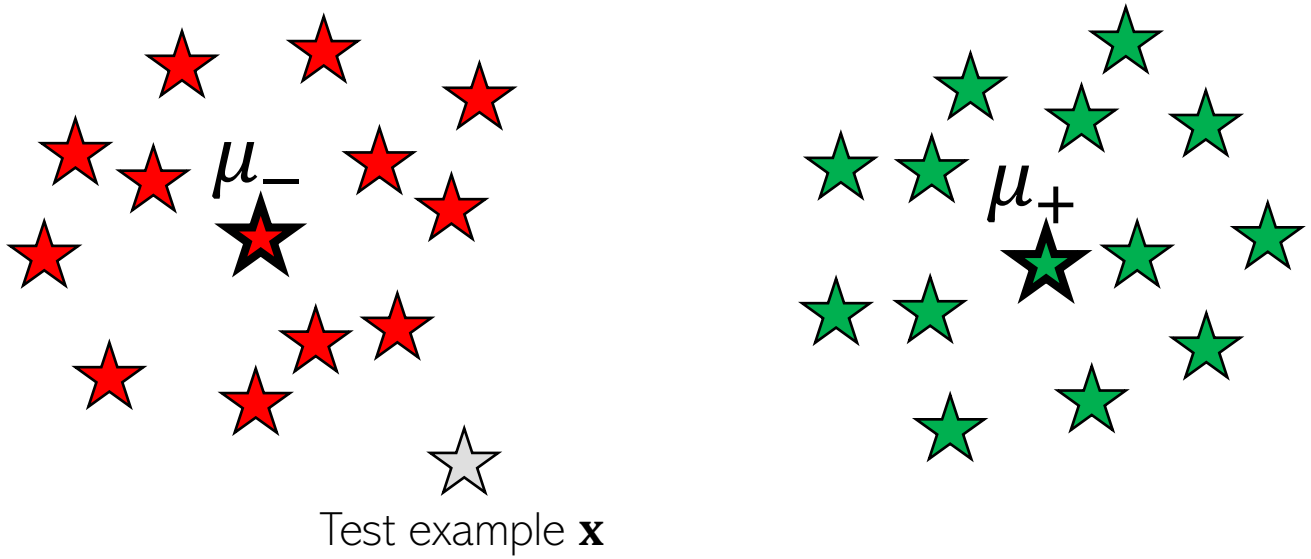
LwP: The Prediction Rule, Mathematically

- 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

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

Linear combination $\langle\mathbf{w}, \mathbf{x}\rangle$ of the D input features

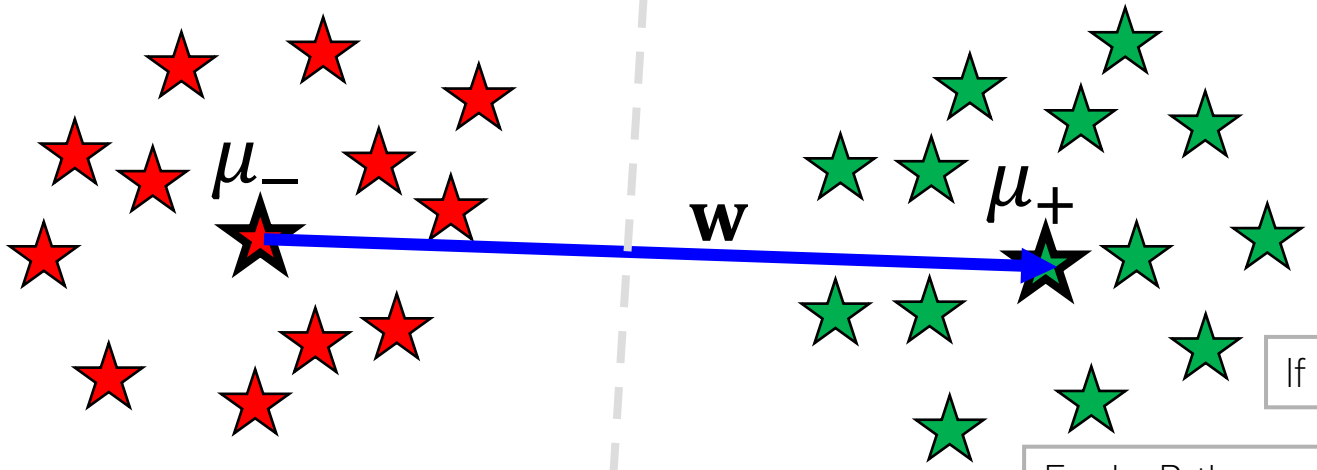
- Thus LwP with Euclidean distance is equivalent to a linear model with
  - A D-dim weight vector $\mathbf{w} = 2(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)$
  - A scalar bias term $b = \left|\left|\boldsymbol{\mu}_-\right|\right|^2 - \left|\left|\boldsymbol{\mu}_+\right|\right|^2$

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

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).

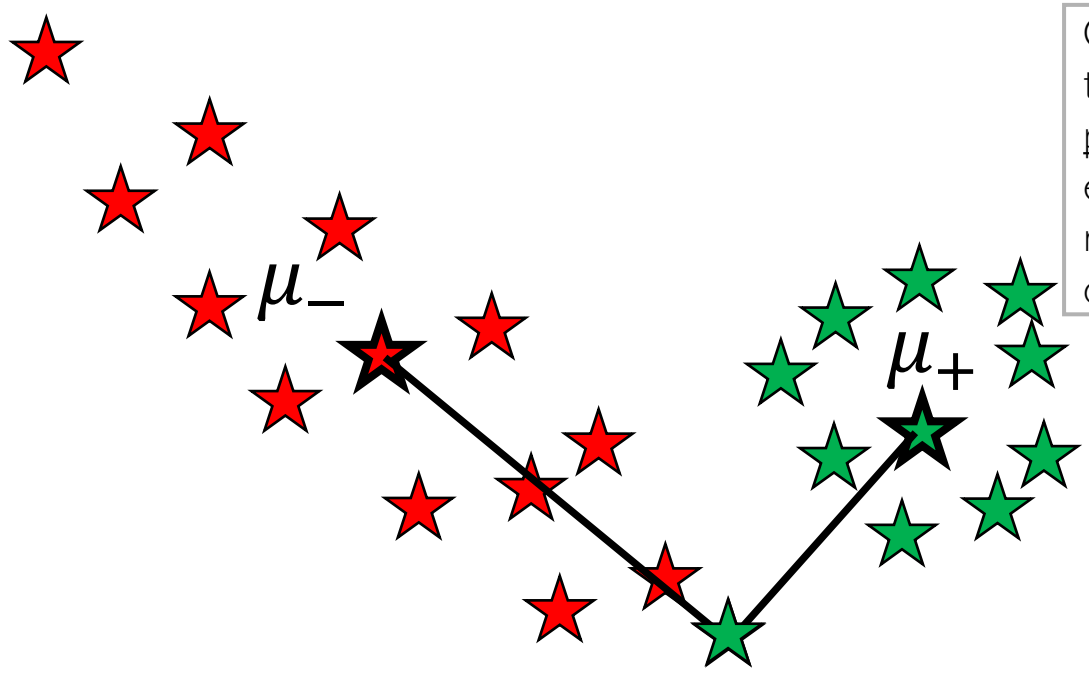
Can throw away training data after computing the prototypes and just need to keep the model parameters for the test time in such "parametric" models

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

LwP: Some Failure Cases

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



Test example $\mathbf{x}$

Can also use a Mahalanobis distance to handle such cases, or use probabilistic models that represent each class not by a "point" (its mean) but as a probability distribution like a Gaussian

Will study such approaches later

- In general, if classes are not equisized and spherical, LwP with Euclidean distance will usually not work well (but improvements possible if features are learned, e.g., using distance metric learning methods or deep learning)

LwP: Some Concluding Remarks

- 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)
- Only applicable for classification problems (not for regression)

Nearest Neighbors

Nearest Neighbors

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

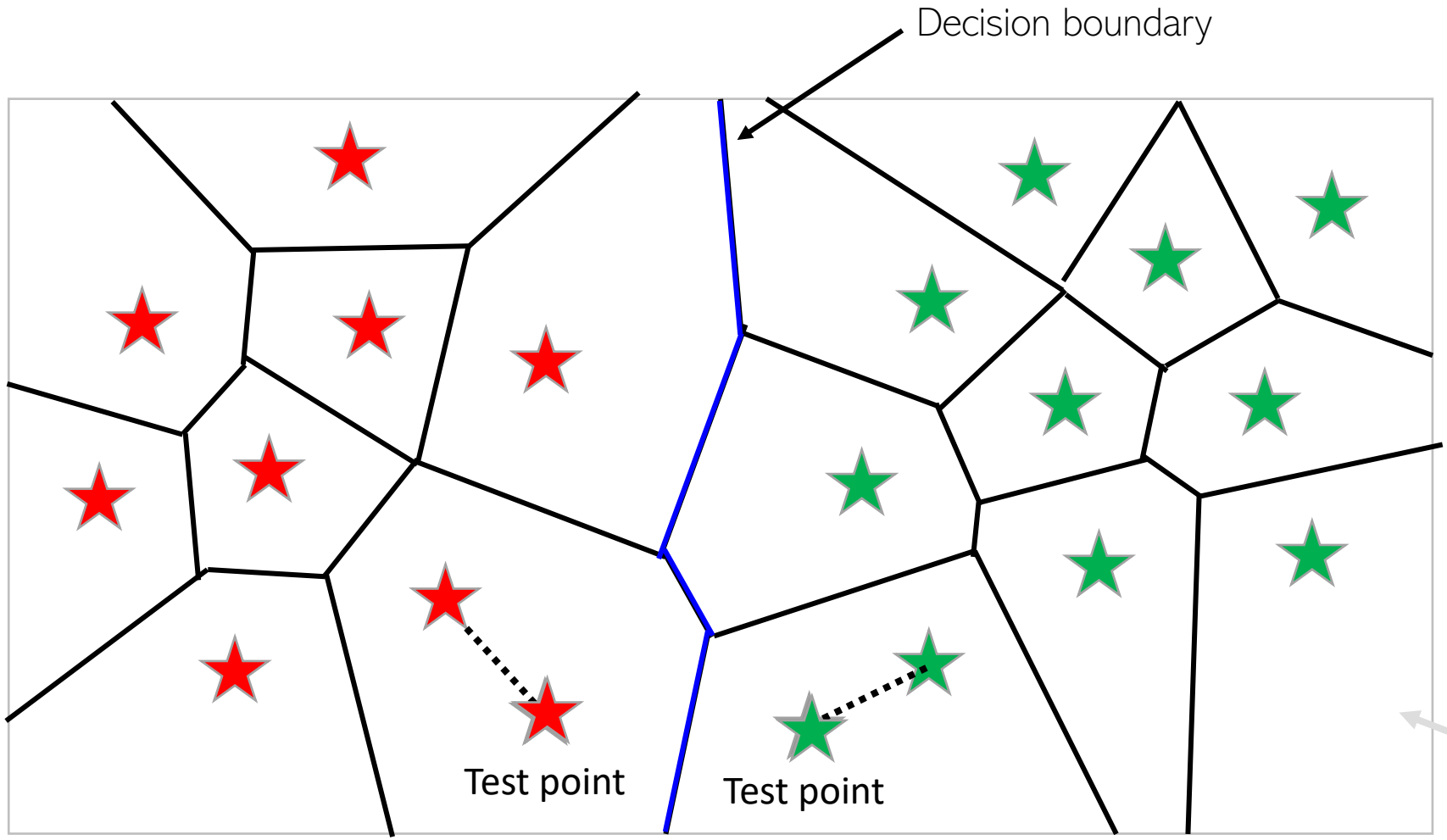
- Very simple idea. Simply do the following at test time
  - Compute distances of the test point from all the training inputs
  - Sort the distances to find the "nearest" input(s) in training data
  - Predict the label using majority or avg label of these inputs
  - Note: Can work with similarities as well instead of distances

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)



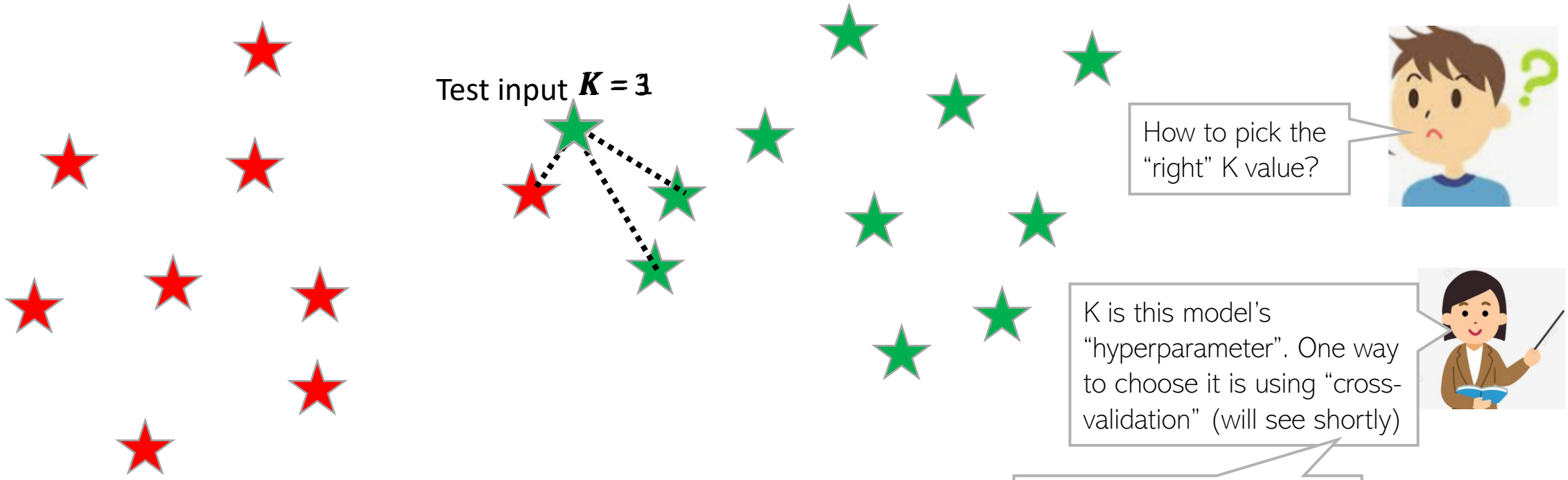
Interesting. Even with Euclidean distances, it can learn nonlinear decision boundaries?

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



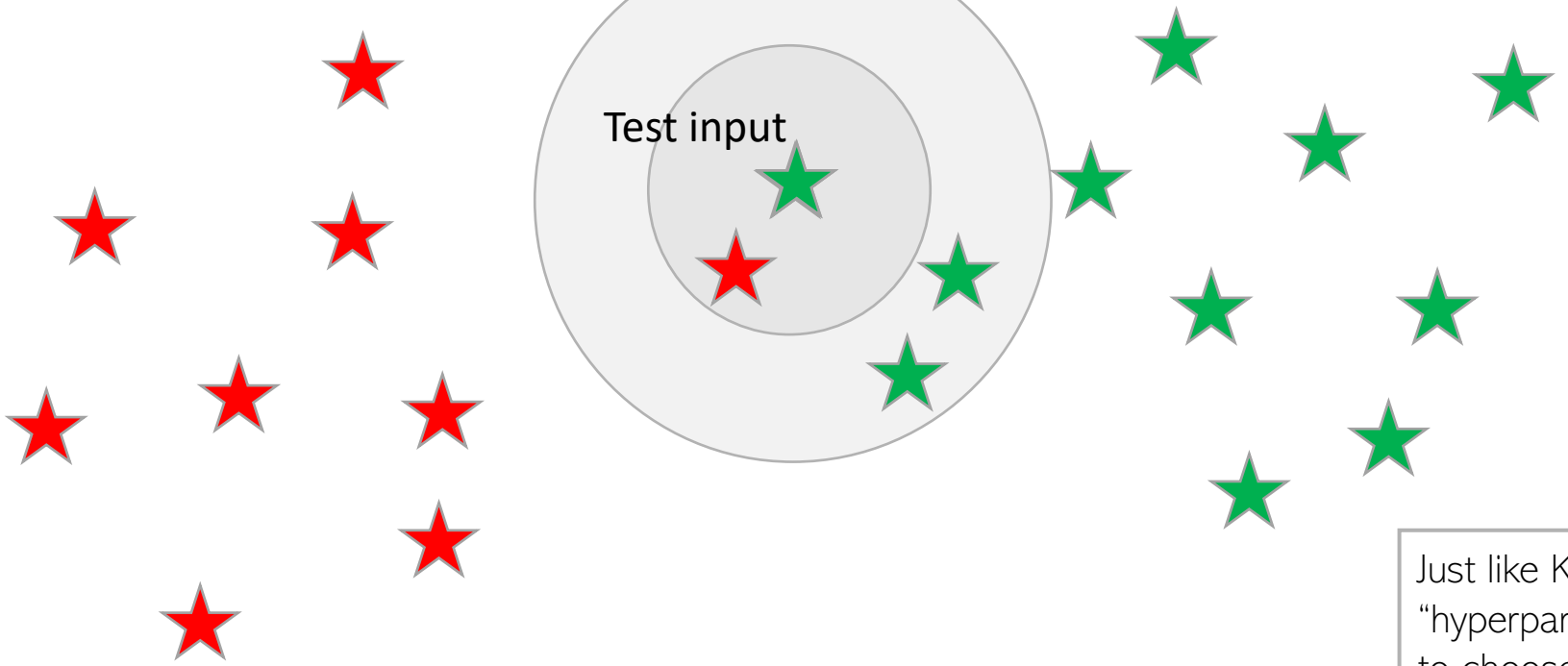
- Essentially, taking more votes helps!
  - Also leads to smoother decision boundaries (less chances of overfitting on training data)

*K*NN in Python (NumPy) Code

```python
import numpy as np

def k_nearest_neighbors(X, y, x_new, k):
  """
  This function predicts the class for a new point using k-nearest neighbors.

  Args:
    X: The training data.
    y: The class labels for the training data.
    x_new: The new point to predict the class for.
    k: The number of neighbors to use.

  Returns:
    The predicted class for the new point.
  """

  distances = np.linalg.norm(X - x_new, axis=1)
  nearest_neighbors = np.argsort(distances)[:k]
  class_labels = y[nearest_neighbors]
  majority_class = np.argmax(np.bincount(class_labels))

  return majority_class

if __name__ == "__main__":
  # Create the data
  X = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]])
  y = np.array([0, 0, 1, 1, 1])
  x_new = np.array([1, 1])

  # Predict the class for the new point
  predicted_class = k_nearest_neighbors(X, y, x_new, 3)

  print(predicted_class)
```

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

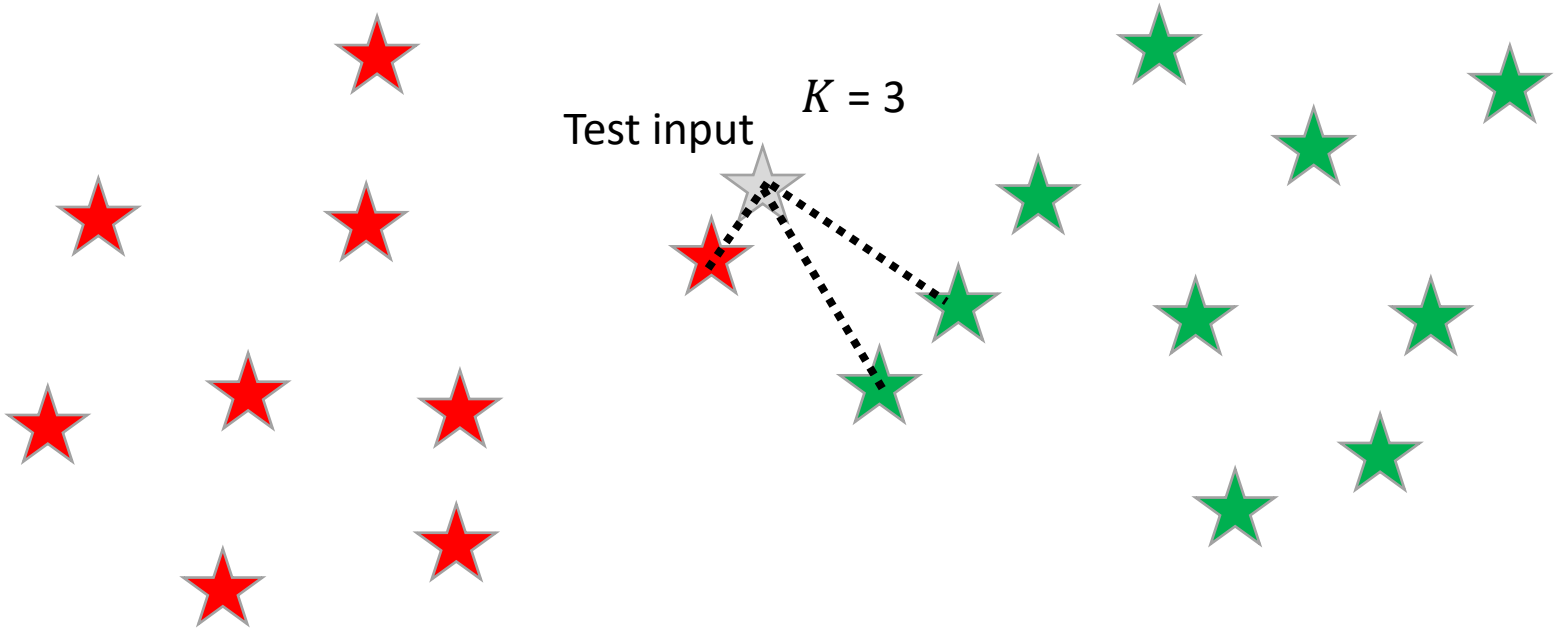


So changing ϵ may change the prediction. How to pick the "right" ϵ value?

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

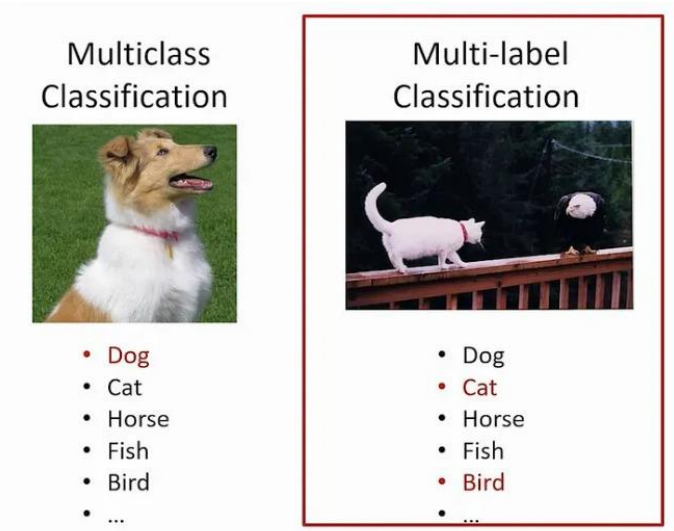
ϵ-NN can also be made weighted likewise

KNN/ϵ-NN for Other Supervised Learning Problems

- Can apply KNN/ϵ-NN for other supervised learning problems as well

Assuming discrete/categorical 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]

- Multi-class classification
  - Each input's label is categorical with K possible values (assuming total K classes)
  - Can also represent the label as a one-hot vector of length K



- Regression
  - Each input's label is a real number

- Tagging/multi-label learning
  - Each input's label is a binary vector of length L (L is the number of tags – the goal is to predict the presence/absence of each tag)

Pic source: https://www.microsoft.com/en-us/research/uploads/prod/2017/12/40250.jpg

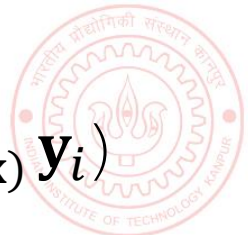
*K*NN Prediction Rule: The Mathematical Form

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

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

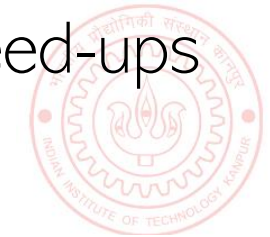
$$\mathbf{y} = \frac{1}{K} \sum_{i \in N_K(\mathbf{x})} \mathbf{y}_i$$

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



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



Speeding-up Nearest Neighbors

- Can use techniques to reduce the training set size
  - Several data summarization techniques exist that discard redundant training inputs
  - Now we will require fewer number of distance computations for each test input
- Can use techniques to reduce the data dimensionality (no. of features)
  - Won't reduce no. of distance computations but each distance computation will be faster
- Compressing each input into a small binary vector (a type of dim-red)
  - Distance/similarity computation between bin. vecs is very fast (can even be done in H/W)
- Various other techniques as well, e.g.,
  - Locality Sensitive Hashing (group training inputs into buckets)
  - Clever data structures (e.g., k-D trees) to organize training inputs
  - Use a divide-and-conquer type approach to narrow down the search region

We will look at Decision Trees which is also like a divide-and-conquer approach

Hyperparameter Selection

- Every ML model has some hyperparameters that need to be tuned, e.g.,
  - K in KNN or ϵ in ϵ-NN
  - Choice of distance to use in LwP or nearest neighbors

- Would like to choose h.p. values that would give best performance on test data

Oops, sorry! What to do then?

Okay. So I can try multiple hyperparam values and choose the one that gives the best accuracy on the test data. Simple, isn't it?

Is validation set a good proxy to test set?

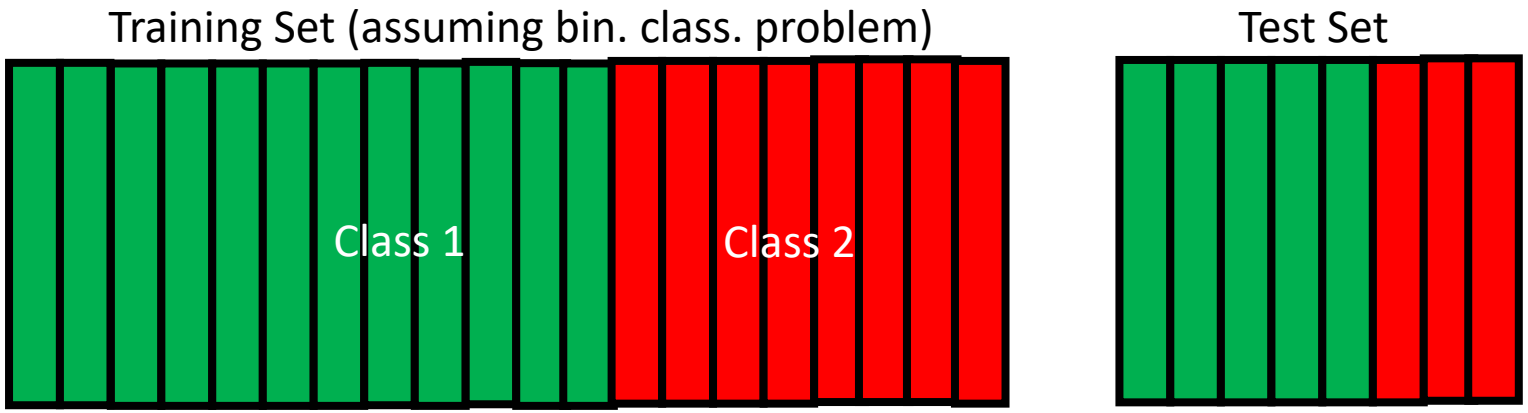
Beware. You are committing a crime. Never Ever touch your test data while building the model

Use cross-validation – use a part of your training data (we will call it "validation/held-out set") to select best hyperparam values.

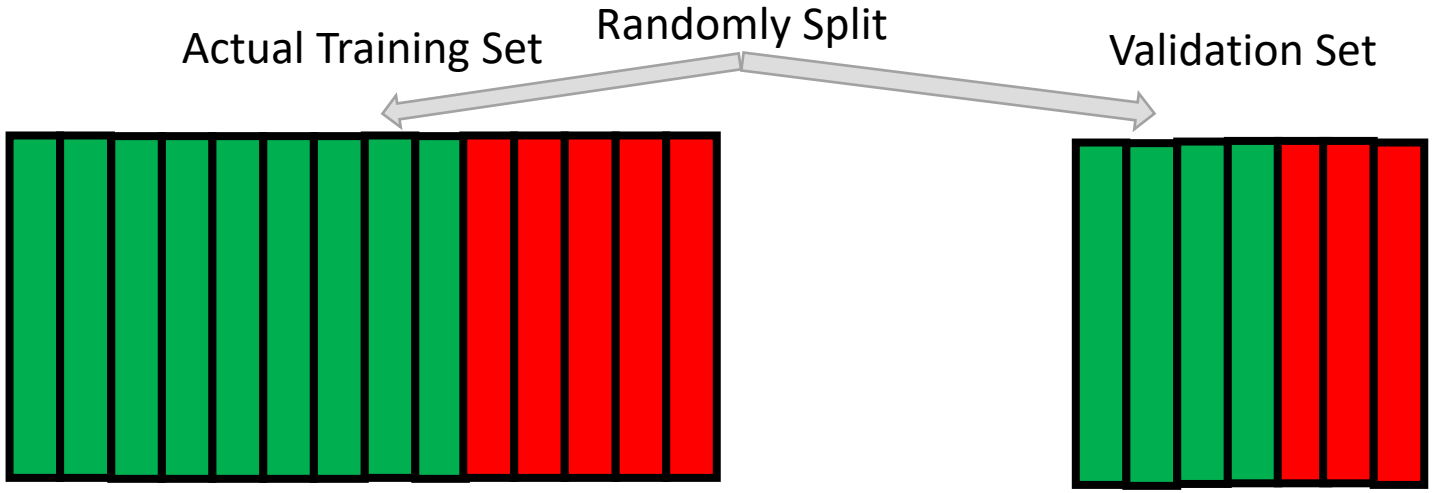
Usually yes since training set (from which the val set is taken) and test sets are assumed to have similar distribution)

Cross-Validation

No peeking while building the model



Note: Not just h.p. selection; we can also use CV to pick the best ML model from a set of different ML models (e.g., say we have to pick between two models we may have trained - LwP and nearest neighbors. Can use CV to choose the better one.



Randomly split the original training data into actual training set and validation set. Using the actual training set, train several times, each time using a different value of the hyperparam. Pick the hyperparam value that gives best accuracy on the validation set

What if the random split is unlucky (i.e., validation data is not like test data)?

If you fear an unlucky split, try multiple splits. Pick the hyperparam value that gives the best average CV accuracy across all such splits. If you are using N splits, this is called N–fold cross validation

Next Class

- Decision Trees and Forests