Nearest Neighbors, Cross-Validation

CS771: Introduction to Machine Learning

Announcement

- Quiz 1 date shifted to Aug 20
    - Timing will be 7:00pm – 8:00pm
    - Venue is likely to be L7, L18, L20
    - More details to be shared soon

Nearest Neighbors

Nearest Neighbors

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

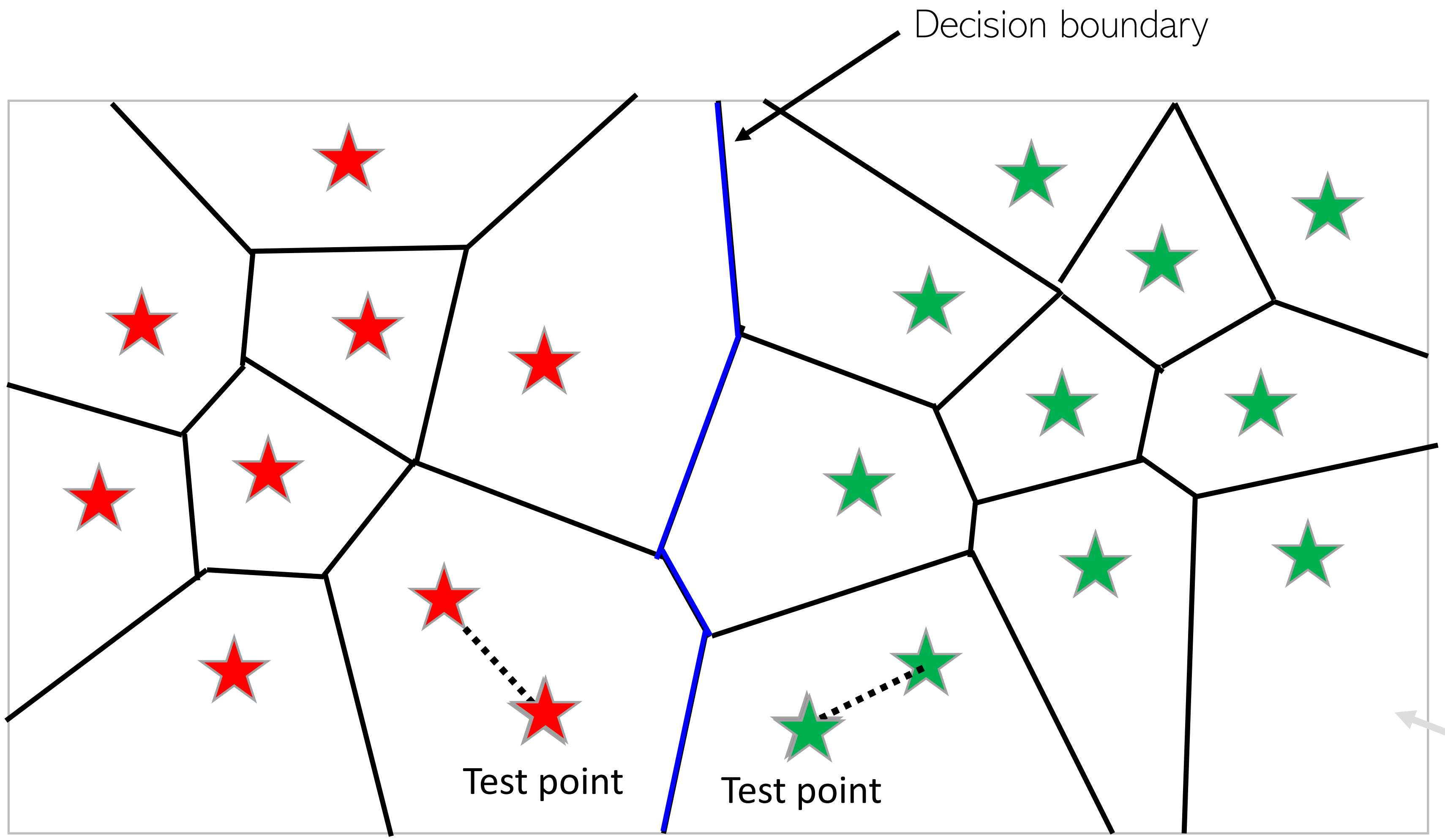
- Very simple idea for sup. learning. Simply do the following at test time
  - Compute distances of the test input from all the training inputs
  - Sort the distances to find the "nearest" input(s) in training data
  - Predict the label using majority or avg label of these inputs
  - Note: Can work with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



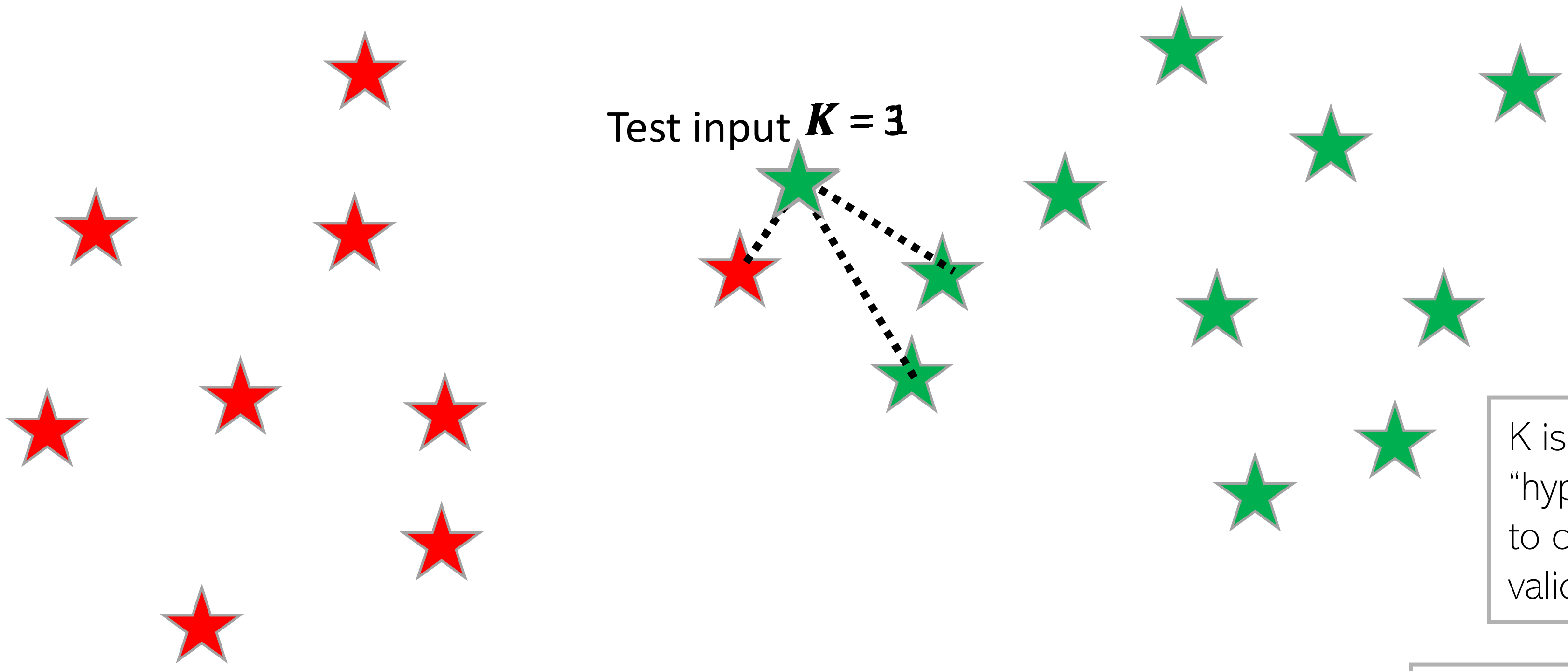
Interesting. Even with Euclidean distances, it can learn nonlinear decision boundaries?

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 (KNN)

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



How to pick the "right" K value?

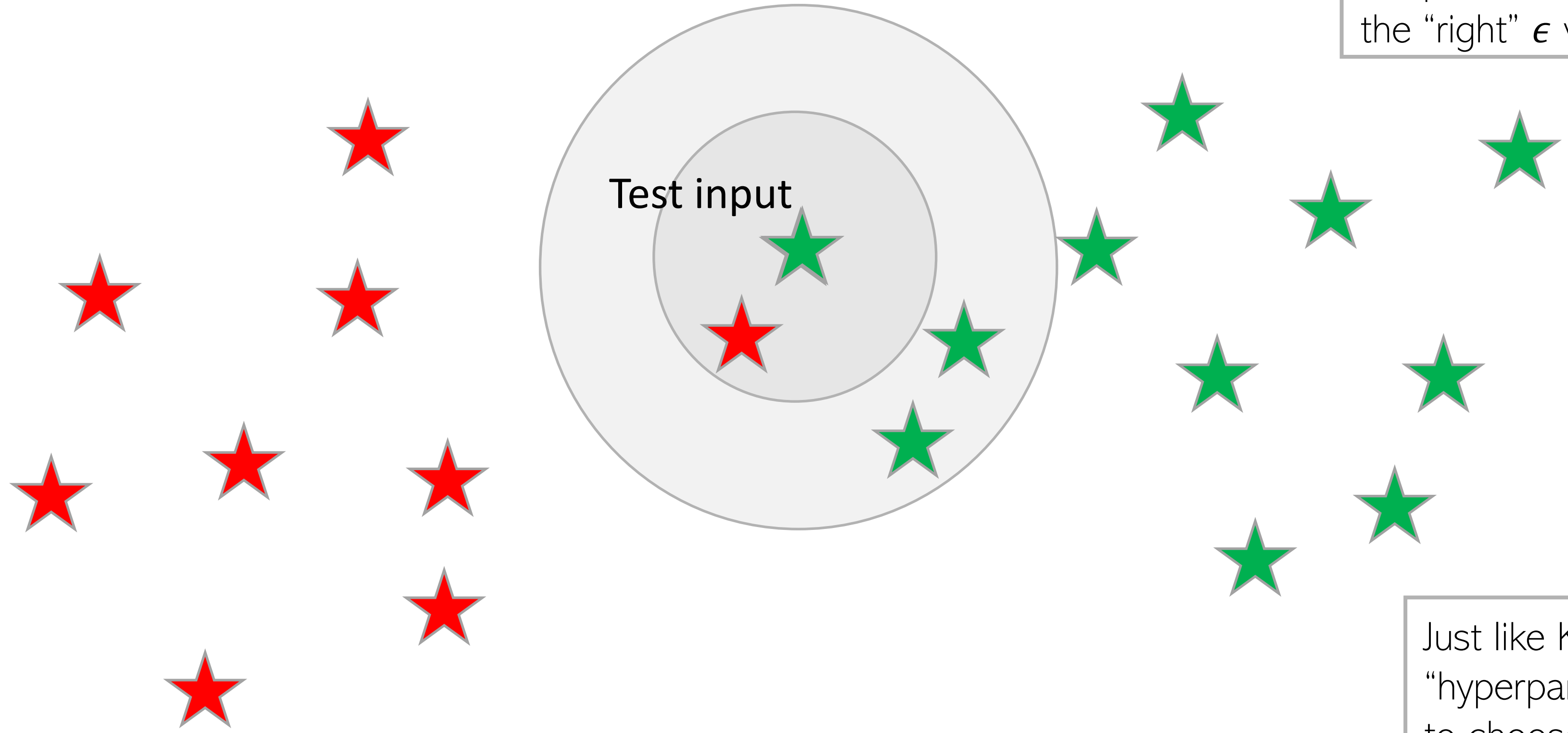
K is this model's "hyperparameter". One way to choose it is using "cross-validation" (will see shortly)

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)

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

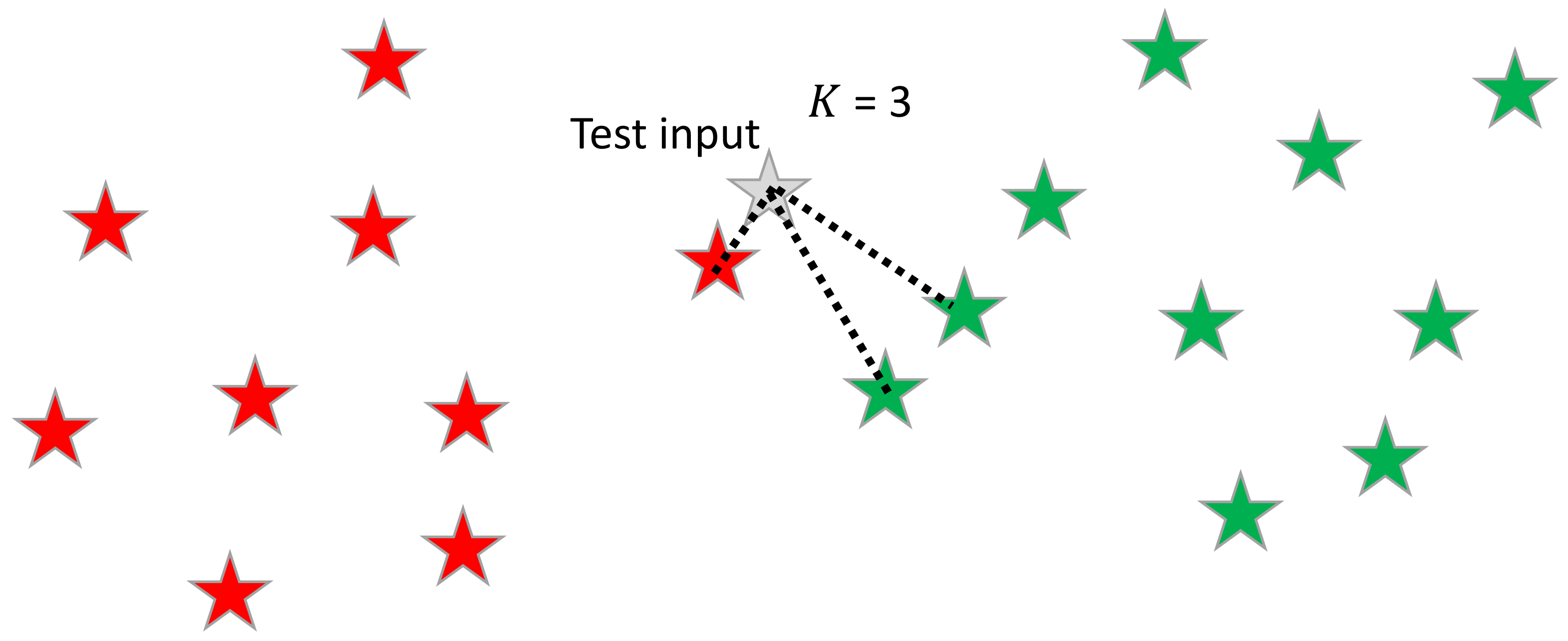


So changing ϵ may change the prediction. How to pick the "right" ϵ value?

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

Distance-weighted KNN 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 Prediction Rule, Mathematically

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

- For regression where output is real-valued, KNN prediction $\boldsymbol{y}$ for a test input $\boldsymbol{x}$

All neighbors given equal importance in averaging of labels

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

More similar neighbors given more importance

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

Computing using some appropriate similarity function

- For classification (e.g., binary/multi-class/multi-label classification)
  - Each label $\boldsymbol{y}_i$ may either be a binary label (0/1 or -1/+1), or one-hot, or binary vector
  - To get the prediction, the majority voting rule can be written as in terms of above $\boldsymbol{y}$ as

$$\hat{y} = \mathbb{I}[\boldsymbol{y} \geq 0.5]$$

If each $\boldsymbol{y}_i$ is 0/1

$$\hat{y} = \text{sign}[\boldsymbol{y}]$$

If each $\boldsymbol{y}_i$ is -1/+1

Most likely class in multi-class classification

$$\hat{y} = \text{argmax}(\boldsymbol{y})$$

If each $\boldsymbol{y}_i$ is a one-hot vector

Most likely subset of $\boldsymbol{m}$ labels in multi-label classification

$$\hat{y} = \text{top\_m\_indices}\ (\boldsymbol{y})$$

If each $\boldsymbol{y}_i$ is binary vector

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?

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

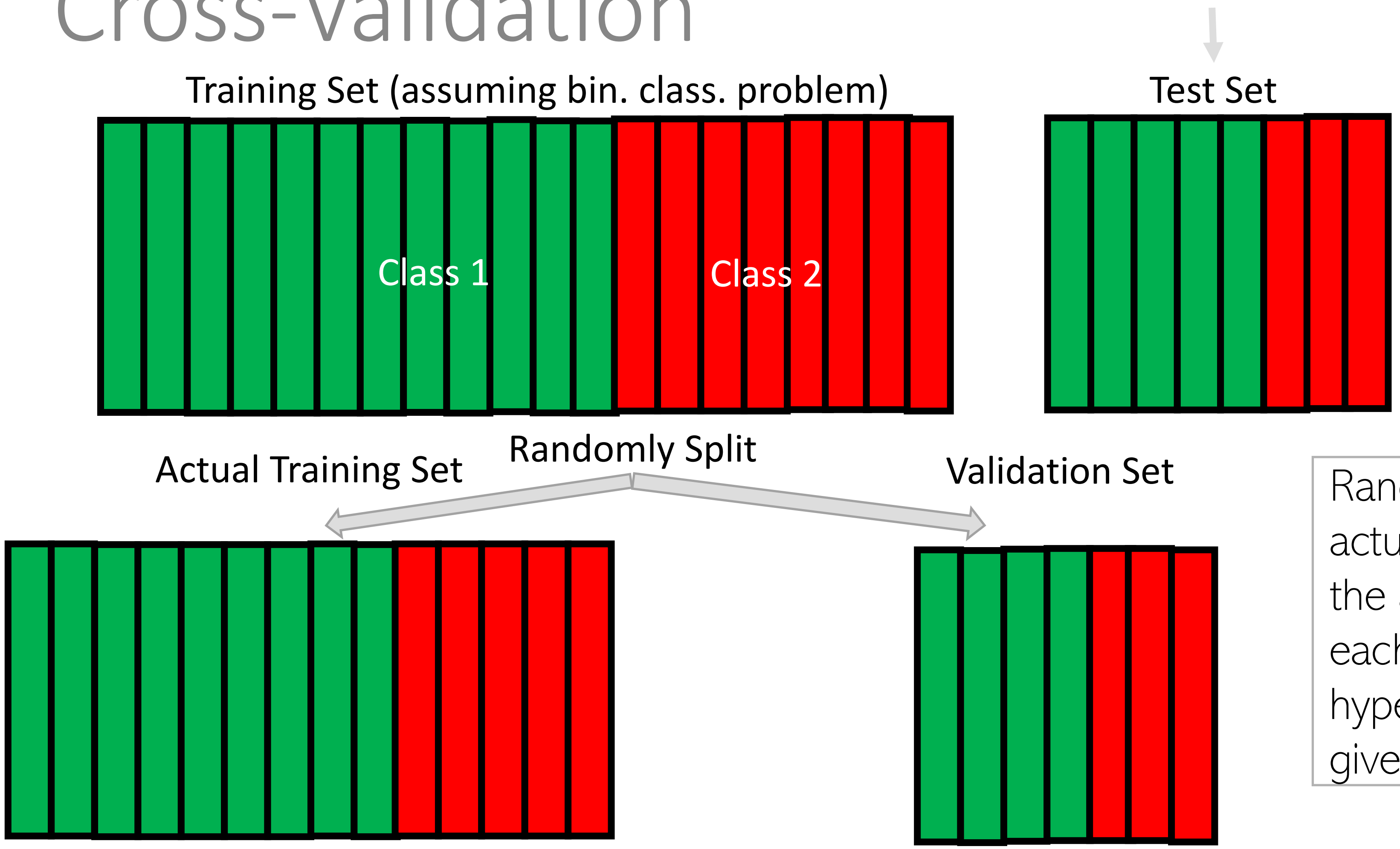
Use cross-validation – use a part of your training data (we will call it "validation/held-out set") to select best hyperparam values.

Is validation set a good proxy to test set?

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