Data and Features, Some Simple Supervised Learners (LwP, Nearest Neighbors)

CS771: Introduction to Machine Learning

Announcements

- Please join on Piazza
  - Joining link already shared in lecture 1 slides
  - If you don't join by Aug 10, there will be some penalty
  - If you have some issue joining Piazza, let us know and we can add you

- Some bonus marks will be reserved for (constructive) Piazza participation
  - Insightful questions and helpful answers/discussions from students
  - Amount of bonus at instructor's discretion (e.g., if you are falling below the threshold of a grade, you may get a "bump up")

- Project groups should be declared by Aug 11
  - Will share a Google form to enter details

Plan today

- Feature extraction from raw data
  - How to convert raw inputs into “features” that our ML algos can understand/use?

Images as inputs

Text docs as inputs

- How to transform some given features to make them more useful?
- Supervised learning methods that only require computing distances/similarities between inputs
  - Learning with Prototypes (LwP)
  - Nearest Neighbors
- How to “tune” our ML models (selecting hyperparameters using cross-validation)

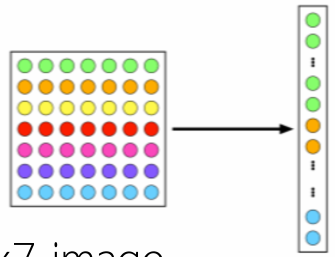
Data and Features

Features represent semantics of the inputs. Being able to extract good features is key to the success of ML algos

- ML algos require a numeric feature representation of the inputs
- Features can be obtained using one of the two approaches
  - Approach 1: Extracting/constructing features manually from raw inputs
  - Approach 2: Learning the features from raw inputs
- Approach 1 is what we will assume/focus on primarily for now
  - For the first few lectures, we will assume that someone has already given us the features
- Approach 2 is what is followed in Deep Learning based models and other ML models that learn a "code" (some optimal feature representation) for the inputs
- Approach 1 is not as powerful as Approach 2 but still used widely

Example: Feature Extraction for Image Data

- A very simple feature extraction approach for image data is flattening

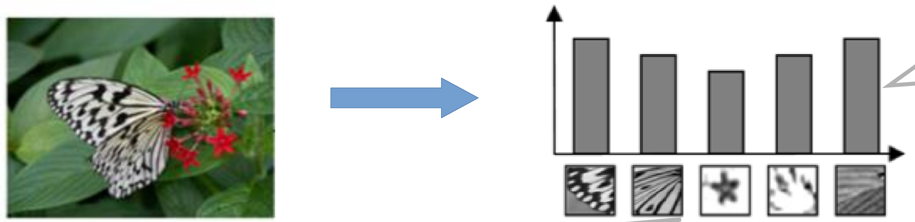


7x7 image (49 pixels)

Vector of pixel intensities

Flattening and histogram based methods destroy the spatial information in the image but often still work reasonably well

- Histogram of visual patterns is another popular feature extr. method for images



Bar heights in the histogram denote how the frequency of occurrence of each of the patterns in the given image (this vector of frequencies can be used as the extracted feature vector for this image)

These patterns can also be "discovered" by some feature learning algorithms (will see later)

Suppose these are typical patterns in the images in the dataset (some "domain expert" may have told us)

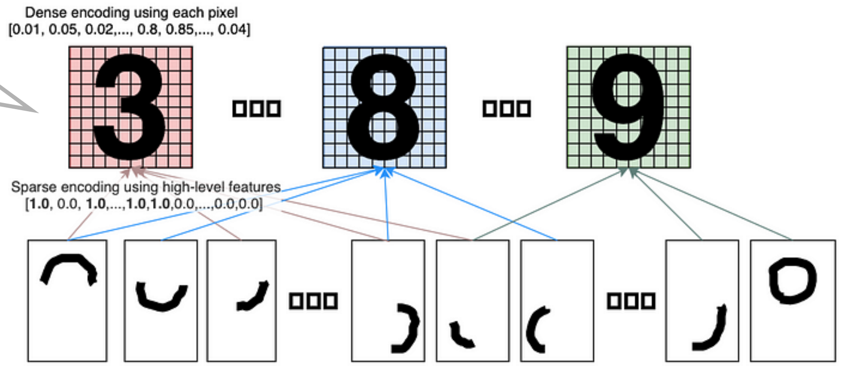
- Many other manual feature extraction techniques developed in computer vision and image processing communities (SIFT, HoG, and others)

Pic credit: cat.uab.cat/Research/object-recognition

Example: Feature Extraction for Image Data

- (Sparse) coding methods can also be used to <u>learn</u> good features

Each input represented using a binary feature vector denoting presence/absence of a large number of patterns (handwriting strokes)



- Deep Learning methods offer a very powerful way to <u>learn</u> good features

Several layers of feature representations learned for each input (that's why it's called "deep")

Last layer of features (the final supervised learning model uses these features)

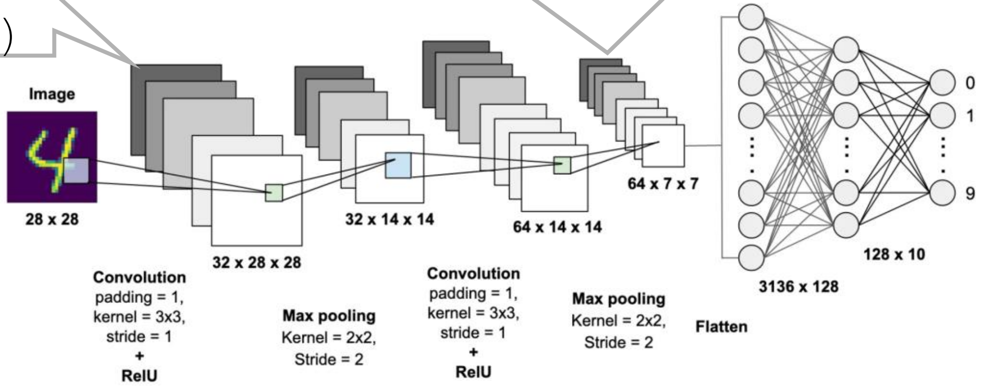


Image
28 x 28
32 x 28 x 28
32 x 14 x 14
64 x 14 x 14
64 x 7 x 7
3136 x 128
128 x 10
0
1
9
Convolution padding = 1, kernel = 3x3, stride = 1 + RelU
Max pooling Kernel = 2x2, Stride = 2
Convolution padding = 1, kernel = 3x3, stride = 1 + RelU
Max pooling Kernel = 2x2, Stride = 2
Flatten

Example: Feature Extraction for Text Data

- Consider some text data consisting of the following sentences:
  - John likes to watch movies
  - Mary likes movies too
  - John also likes football

BoW is just one of the many ways of doing feature extraction for text data. Not the most optimal one, and has various flaws (can you think of some?), but often works reasonably well

- Want to construct a feature representation for these sentences
- Here is a "bag-of-words" (BoW) feature representation of these sentences

Our "vocabulary" of 9 words

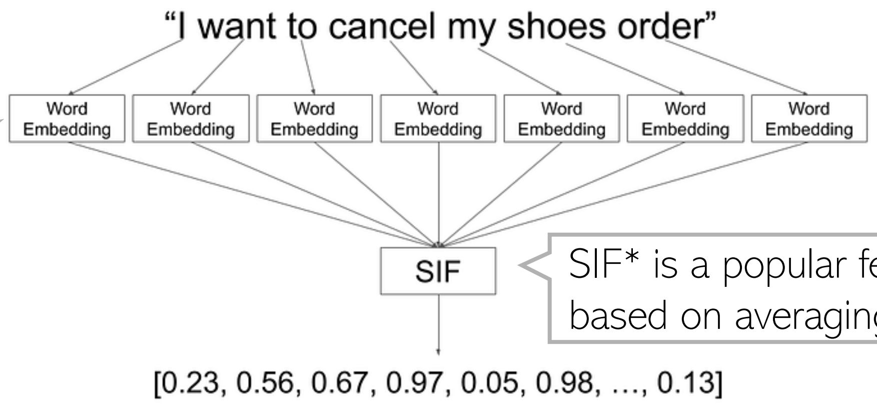
$$\begin{array}{c} \\ \text{Sentence 1} \\ \text{Sentence 2} \\ \text{Sentence 3} \end{array} \begin{pmatrix} \text{John} & \text{likes} & \text{to} & \text{watch} & \text{movies} & \text{Mary} & \text{too} & \text{also} & \text{football} \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

- Each sentence is now represented as a binary vector (each feature is a binary value, denoting presence or absence of a word). BoW is also called "unigram" rep.

Example: Feature Extraction for Text Data

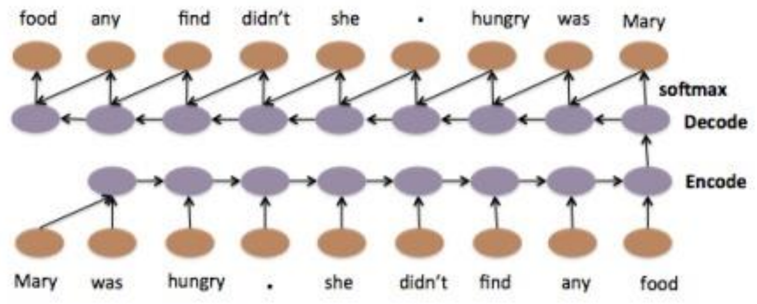
- For text data (like sentences or document), we can do feature extraction using other simple methods that also use explicit semantics of words

For each word, there exists an "embedding" (a vector) such that semantically similar words have similar embeddings



SIF* is a popular feature extraction method based on averaging of word embeddings

- Deep learning methods for sequence data (e.g., recurrent neural networks and transformers) are more powerful for feature extraction for text data (more on this later)



The state representation at the end of this sequence can be used as a feature for this sentence

*A Simple but Tough-to-Beat Baseline for Sentence Embeddings (Arora et al, 2017)

Don’t forget to apply common sense with features!

- Consider a problem of predicting some air-pollutant’s (e.g., PM2.5) concentration at various locations and at various times

- Here, in addition to other features, we also have features such as
  - Lat-long of each location
  - Time-stamp (e.g., hour of the day)

- Using raw values of these features may not be helpful (could even hurt) since
  - Earth is round ☺
  - Hours of the day have cyclical relationships (hour 23 is closer to hour 1 than hour 18)

- Thus we may need to transform such features using suitable techniques
  - Cyclical embeddings and other specific techniques for feature transformation can be used

- Caution: Don’t leave deep learning to take care of all the features. Do apply some common sense as well

Feature Learning = Distance Function Learning?

- It seems there are two ways to learn effectively from data
  - Extract (or rather "learn") features from the data + use standard distance function

Assume f represents is our feature extraction function

$$d(\boldsymbol{a},\boldsymbol{b}) = \sqrt{(f(\boldsymbol{a}) - f(\boldsymbol{b}))^\top (f(\boldsymbol{a}) - f(\boldsymbol{b}))}$$

  - Use a good distance/similarity function of the "basic" features, e.g.,

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

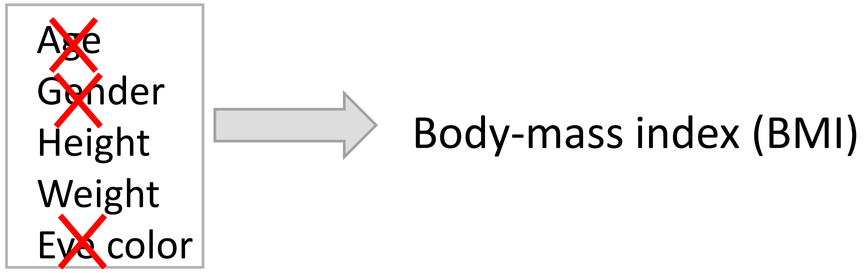
$$k(\boldsymbol{a},\boldsymbol{b}) = \exp(-\gamma||\boldsymbol{a}-\boldsymbol{b}||^2)$$

- Both approaches, at some level, can be seen doing the same thing, just in different ways
- Any feature learning method corresponds to learning some distance/similarity function
  - .. and vice-versa

Feature Selection

- Not all the extracted features may be relevant for learning the model (some may even confuse the learner)

- Feature selection (a step after feature extraction) can be used to identify the features that matter, and discard the others, for more effective learning



Calculating BMI from this data doesn't require ML but this simple example is just to illustrate the idea of feature selection ☺

- Many techniques exist – some based on intuition, some based on algorithmic principles (will visit feature selection later)

- More common in supervised learning but can also be done for unsup. Learning
- Many ML algorithms can automatically take care of feature extraction/selection
  - More on this later

Simple Supervised Learners

Supervised Learning: Regression and Classification

- Regression is when the output is real-valued, e.g.,

Other types of supervised learning problems also exist, such as ranking, which we will study later

Input = Output = Weight of the dog

- Classification is when the output is discrete, e.g.,

Binary classification

Input = Output = Is it a dog (1) or cat (0) ?

Label is a "one-hot" vector of length K

Multi-class classification

Label is a binary vector of length L

Input = Output = Breed (one of $K > 2$ possibilities)

$L > 1$ labels, each binary

Also called "tagging"

Multi-label classification

$L = 4$ in this case

Input = Output = Dog (1/0)? Fluffy (1/0)? Sitting (1/0)? Wearing collar (1/0)?

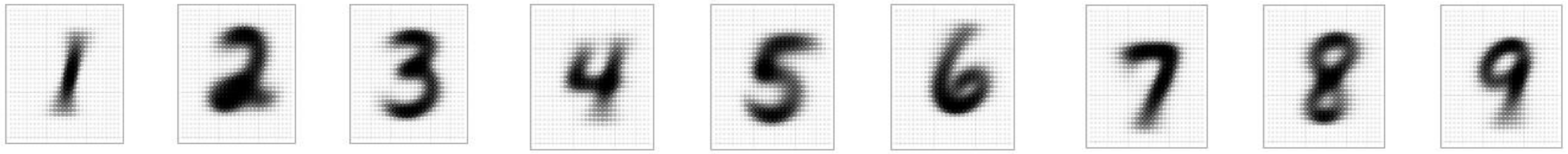
Image by Racool_studio on Freepik

Learning with Prototypes

Classification via 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

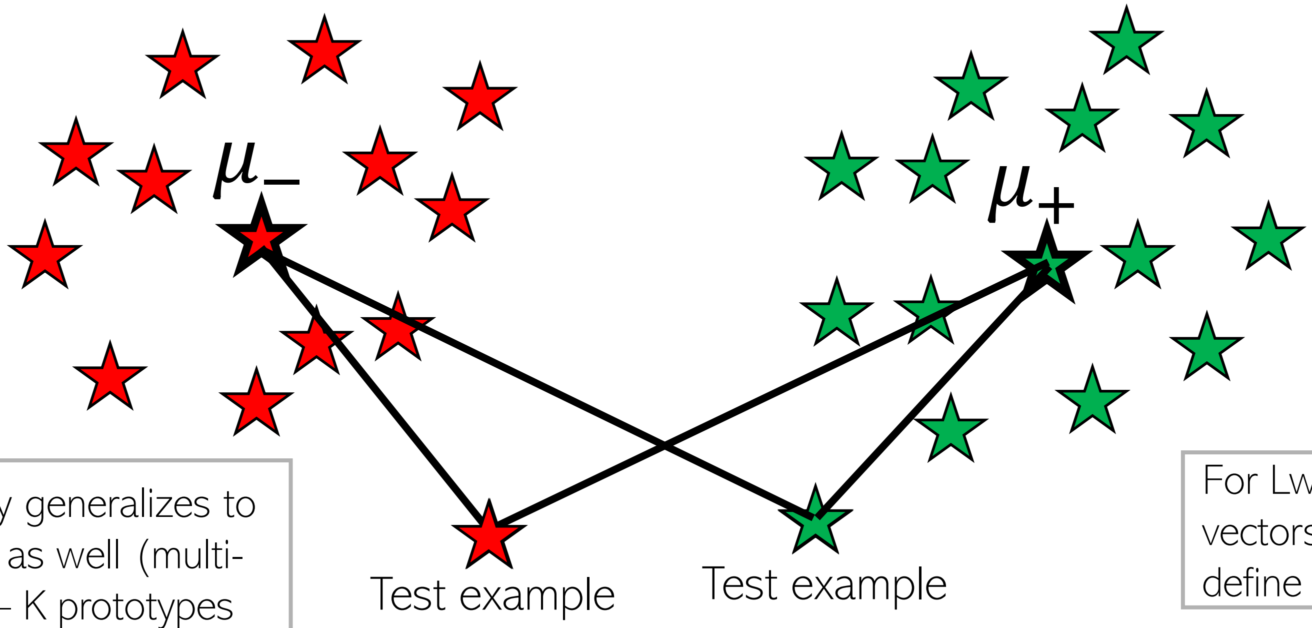
- How we compute distances can have an effect on the accuracy of this model (may need to try Euclidean, weight Euclidean, or something else)

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

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



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

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"

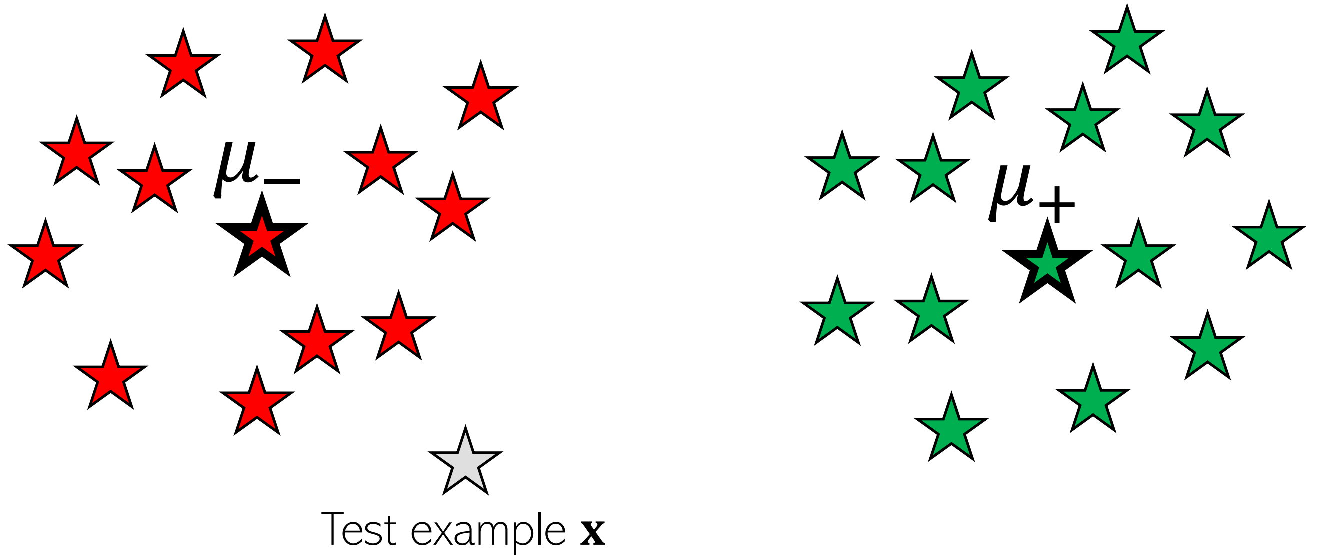
LwP: The Prediction Rule, Mathematically

- 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

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

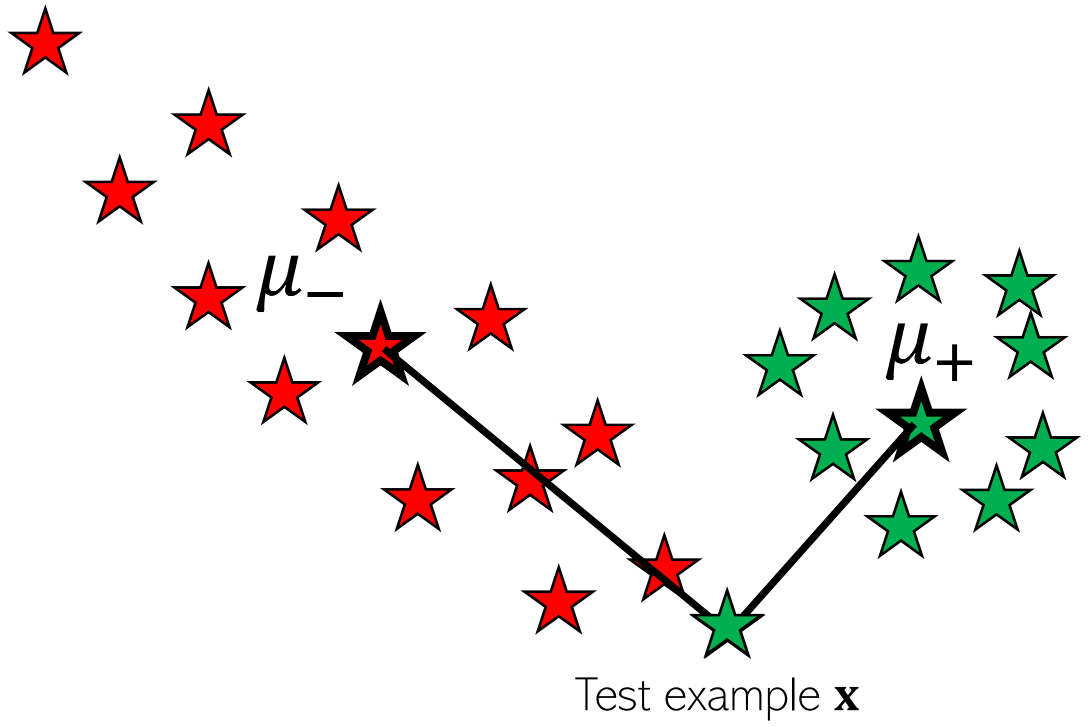
- Thus LwP with Euclidean distance is equivalent to a linear model with
  - Weight vector $\mathbf{w} = 2(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)$
  - Bias term $b = ||\boldsymbol{\mu}_-||^2 - ||\boldsymbol{\mu}_+||^2$

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. Can you think of how to fix this issue?

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")

Nearest Neighbors

Nearest Neighbors

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

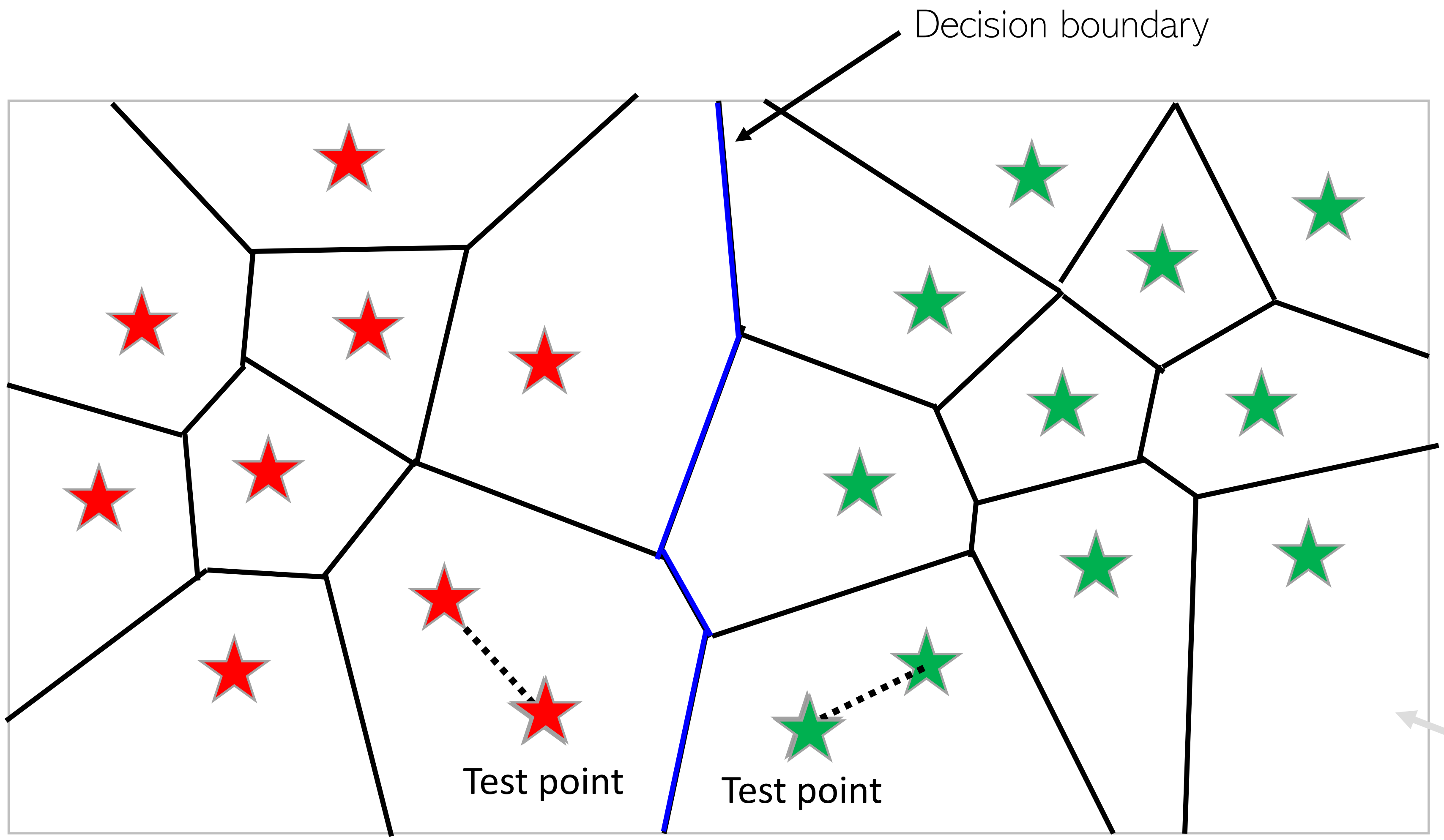
- Very simple idea. Simply do the following at test time
  - Compute distances of the test point from all the training inputs
  - Sort the distances to find the "nearest" input(s) in training data
  - Predict the label using majority or avg label of these inputs
  - Note: Can work with similarities as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

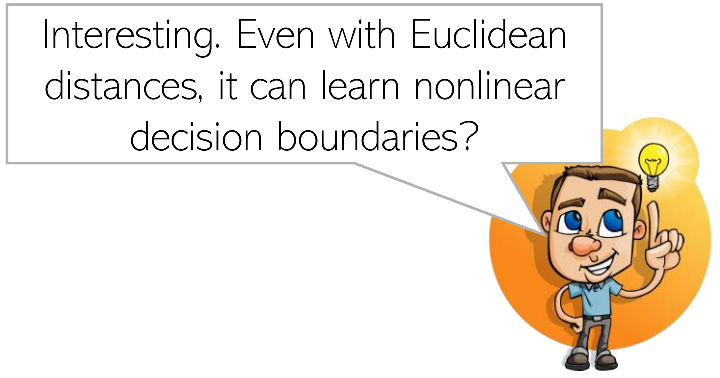
Yes, but let's not worry about that at the moment. There are ways to speed up this step

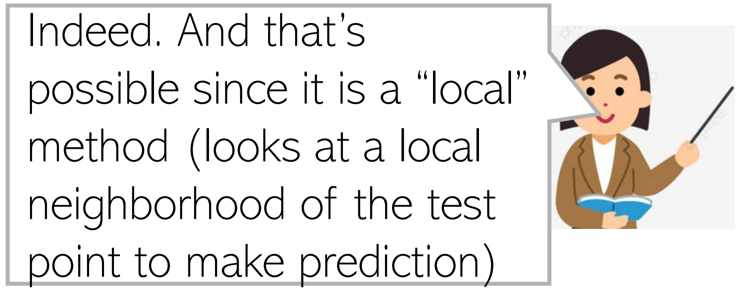
- Can use Euclidean or other dist (e.g., Mahalanobis). Choice imp just like LwP
- Unlike LwP which does prototype based comparison, nearest neighbors method looks at the labels of individual training inputs to make prediction
- Applicable to both classifn as well as regression (LwP only works for classifn)

Nearest Neighbors for Classification

Nearest Neighbor (or "One" Nearest Neighbor)



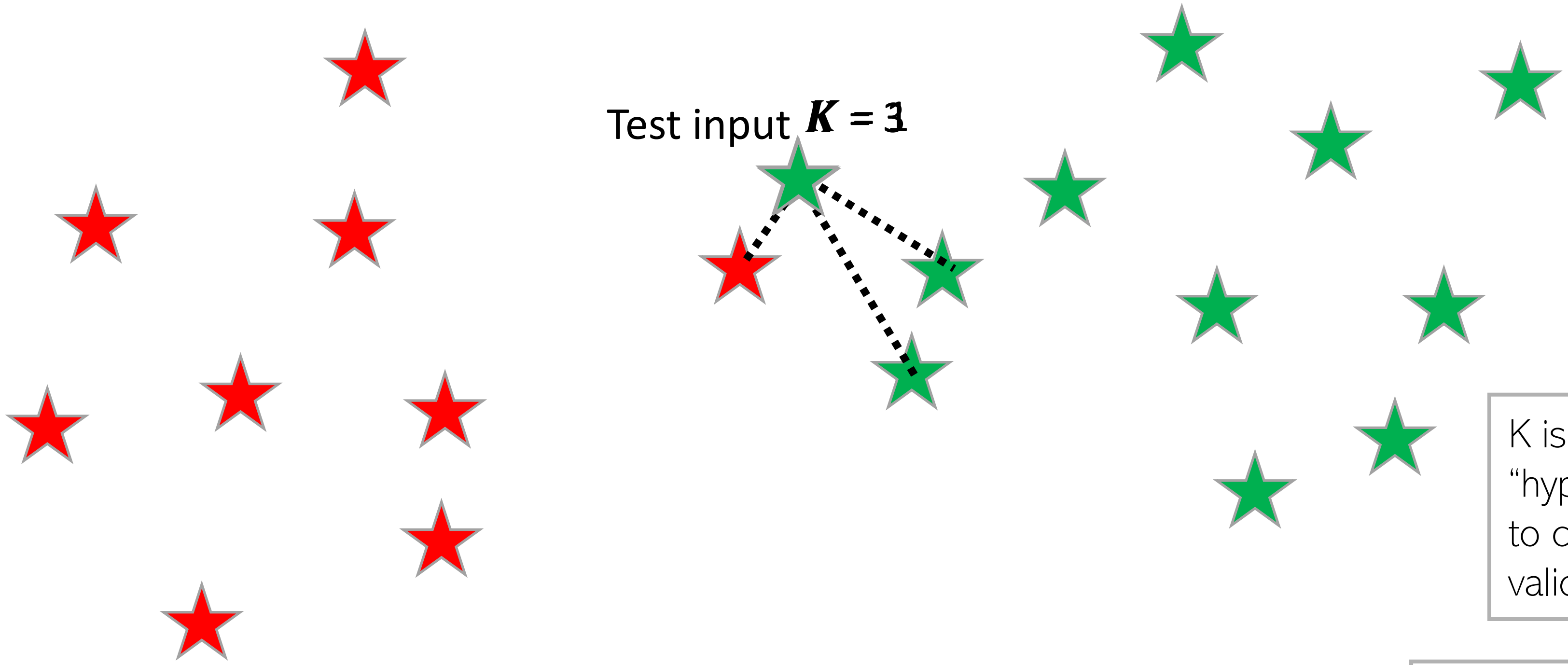

Interesting. Even with Euclidean distances, it can learn nonlinear decision boundaries?


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



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

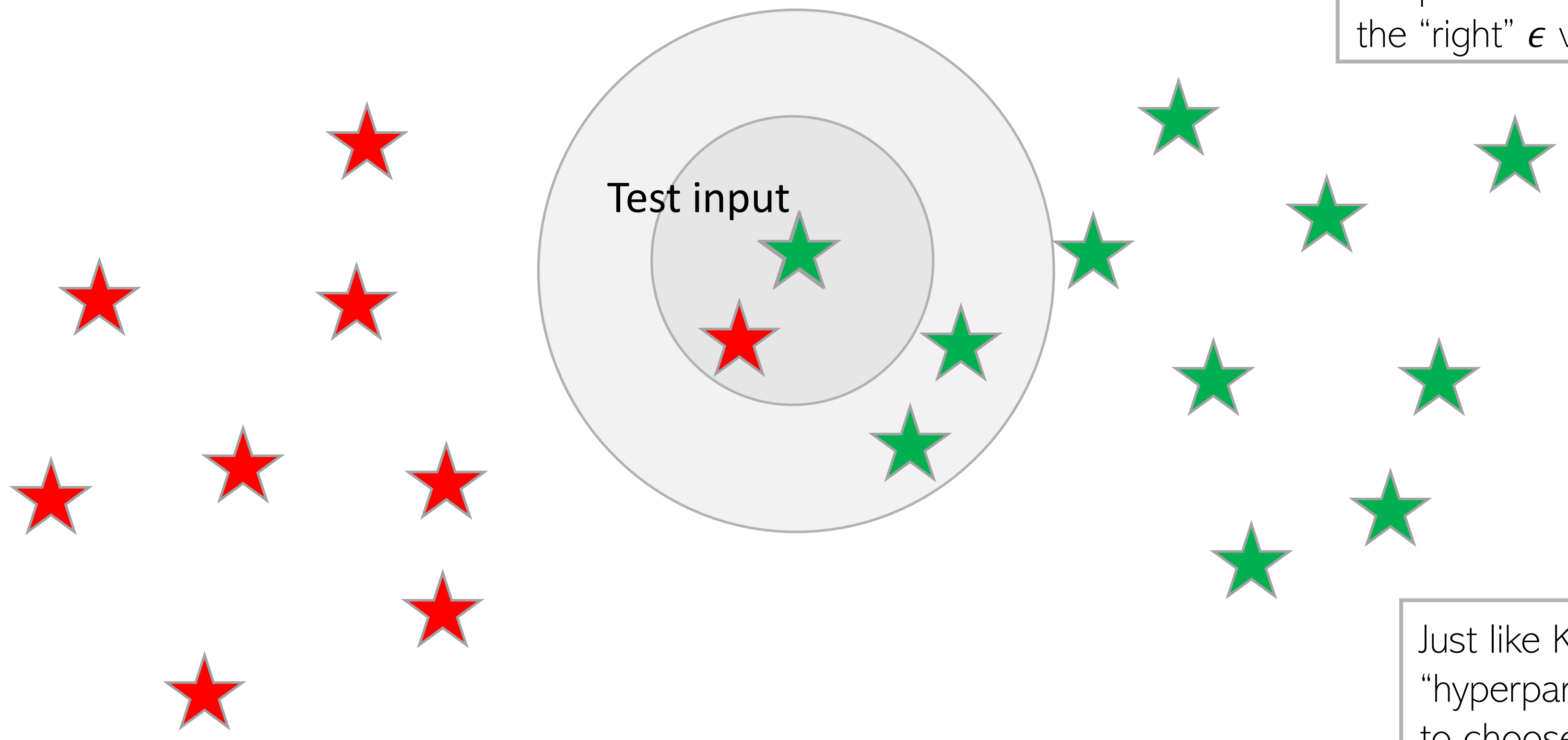
- Essentially, taking more votes helps!
  - Also leads to smoother decision boundaries (less chances of overfitting on training data)

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

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

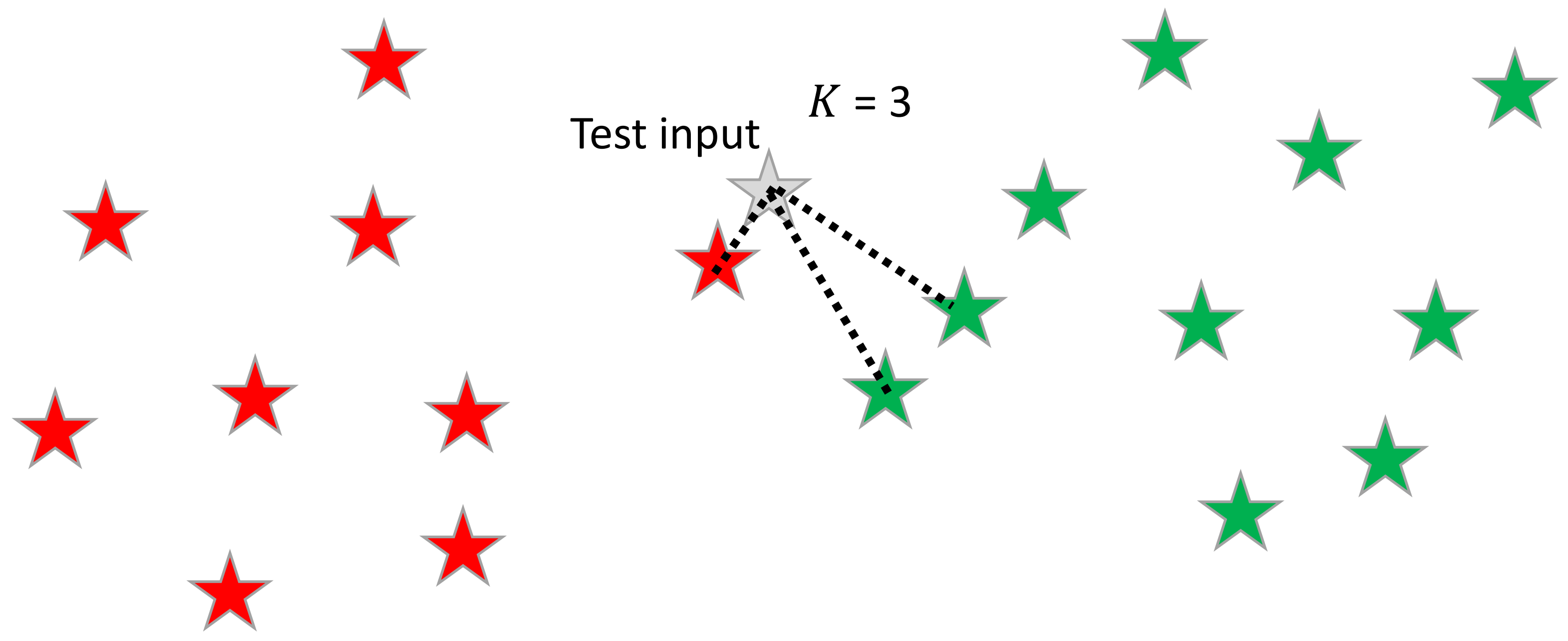
So changing ϵ may change the prediction. How to pick the “right” ϵ value?



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

*K*NN 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

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

- This form makes direct sense for 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?

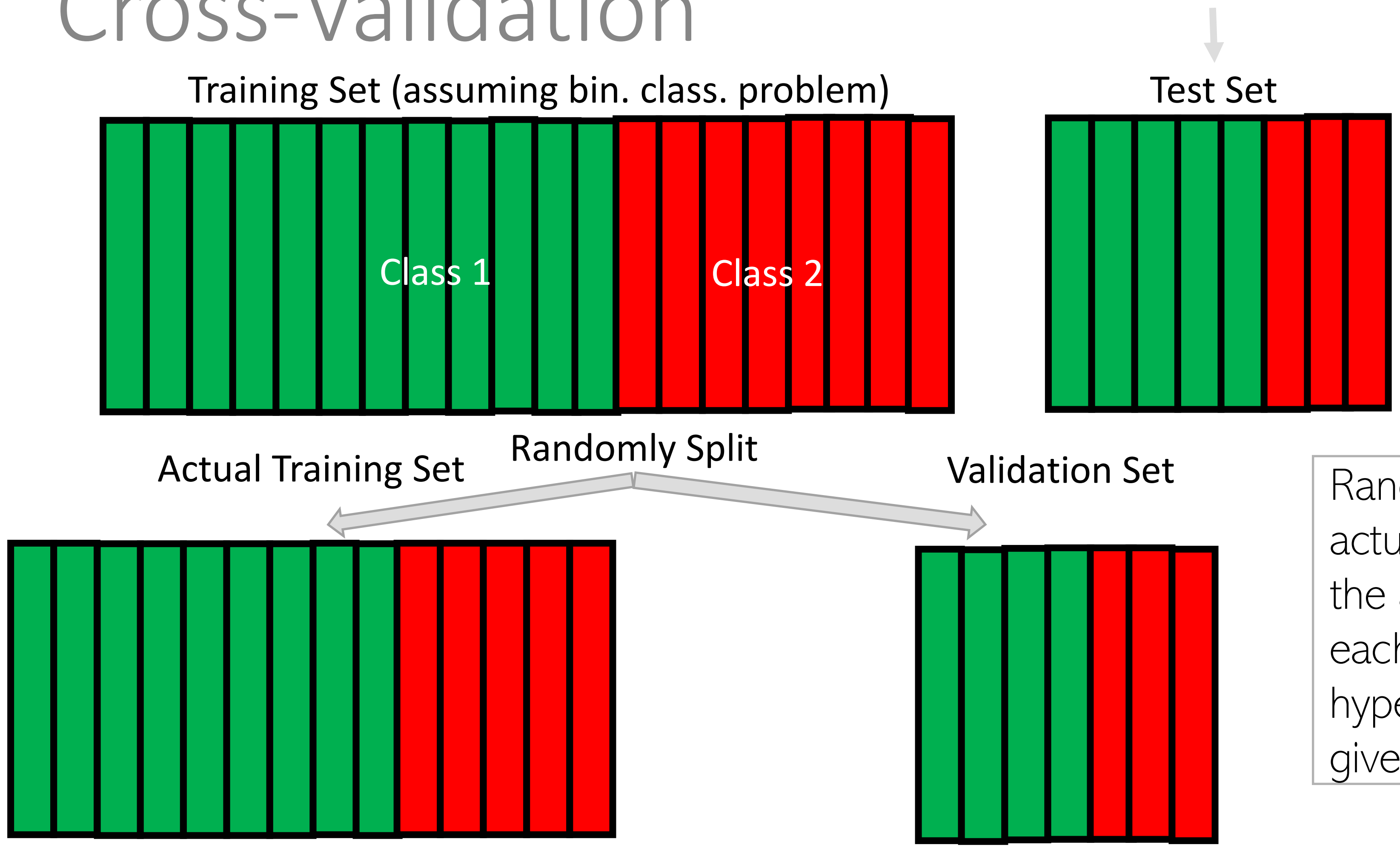
Beware. You are committing a crime. Never Ever touch your test data while building the model

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