Nearest Neighbors

CS771: Introduction to Machine Learning

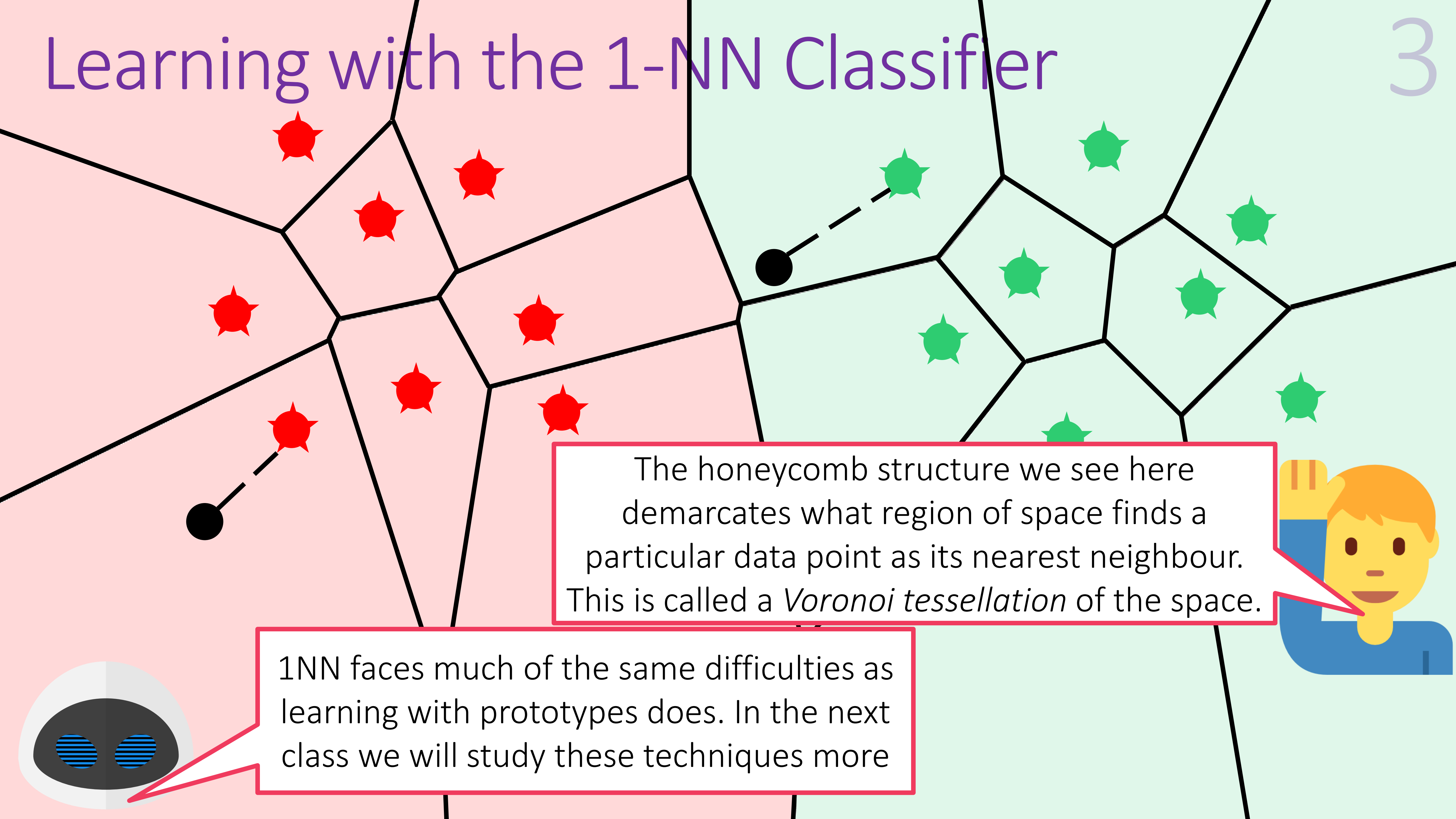
Purushottam Kar

Nearest Neighbour Classifier

- A really extreme form of learning with prototypes
- Recall that we commented that having multiple (say 2-3) prototypes may help when there is lots of diversity in a class
- With NN, every single training point becomes a prototype ☺
- The basic mantra here seems to be

“ *If the training email that most resembles a new email was spam, maybe the new email is spam too. Else, if a normal training email most resembles the new email, maybe the new email is normal* ”

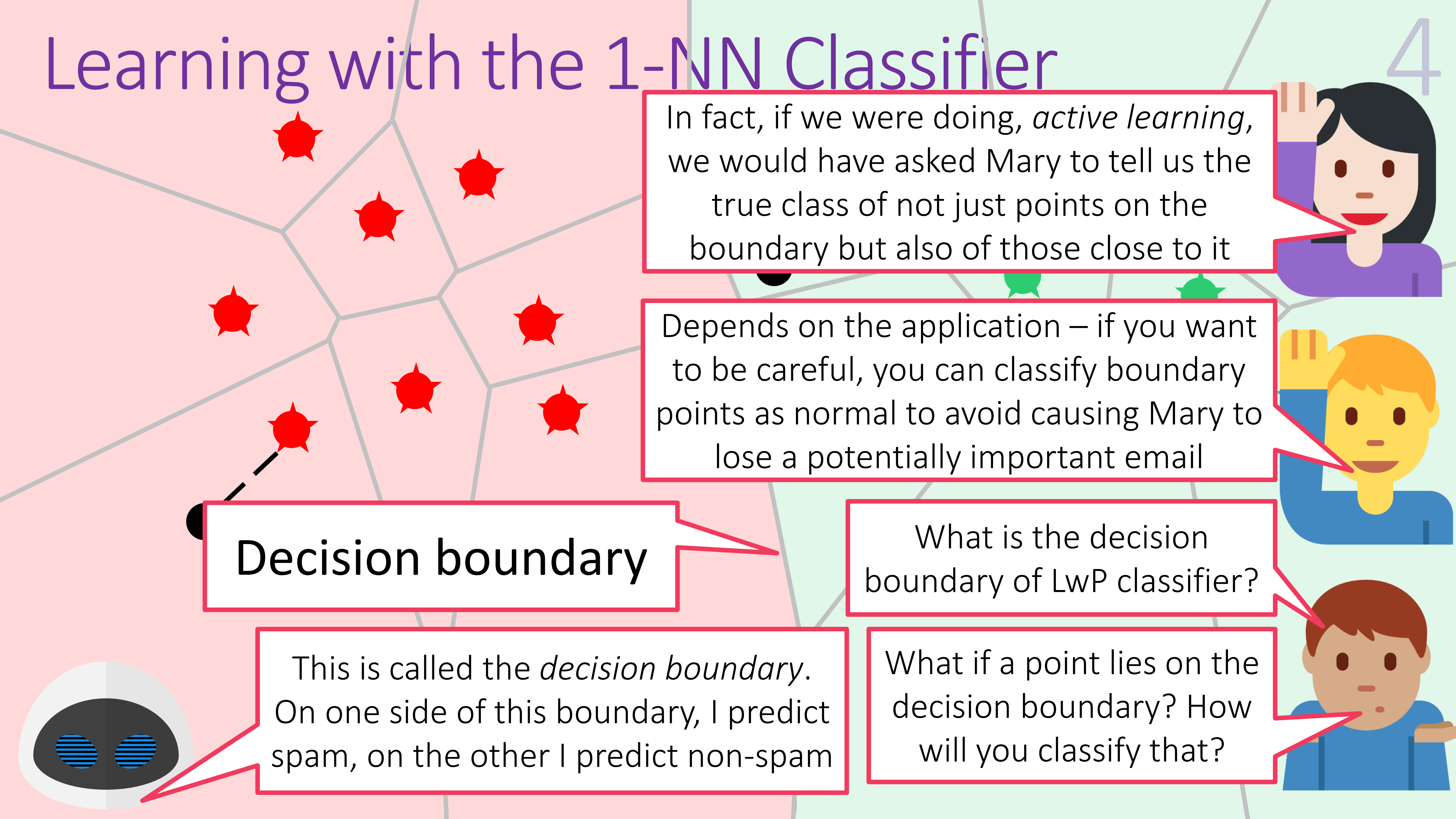
Learning with the 1-NN Classifier



The honeycomb structure we see here demarcates what region of space finds a particular data point as its nearest neighbour. This is called a *Voronoi tessellation* of the space.

1NN faces much of the same difficulties as learning with prototypes does. In the next class we will study these techniques more

Learning with the 1-NN Classifier



This is called the *decision boundary*. On one side of this boundary, I predict spam, on the other I predict non-spam

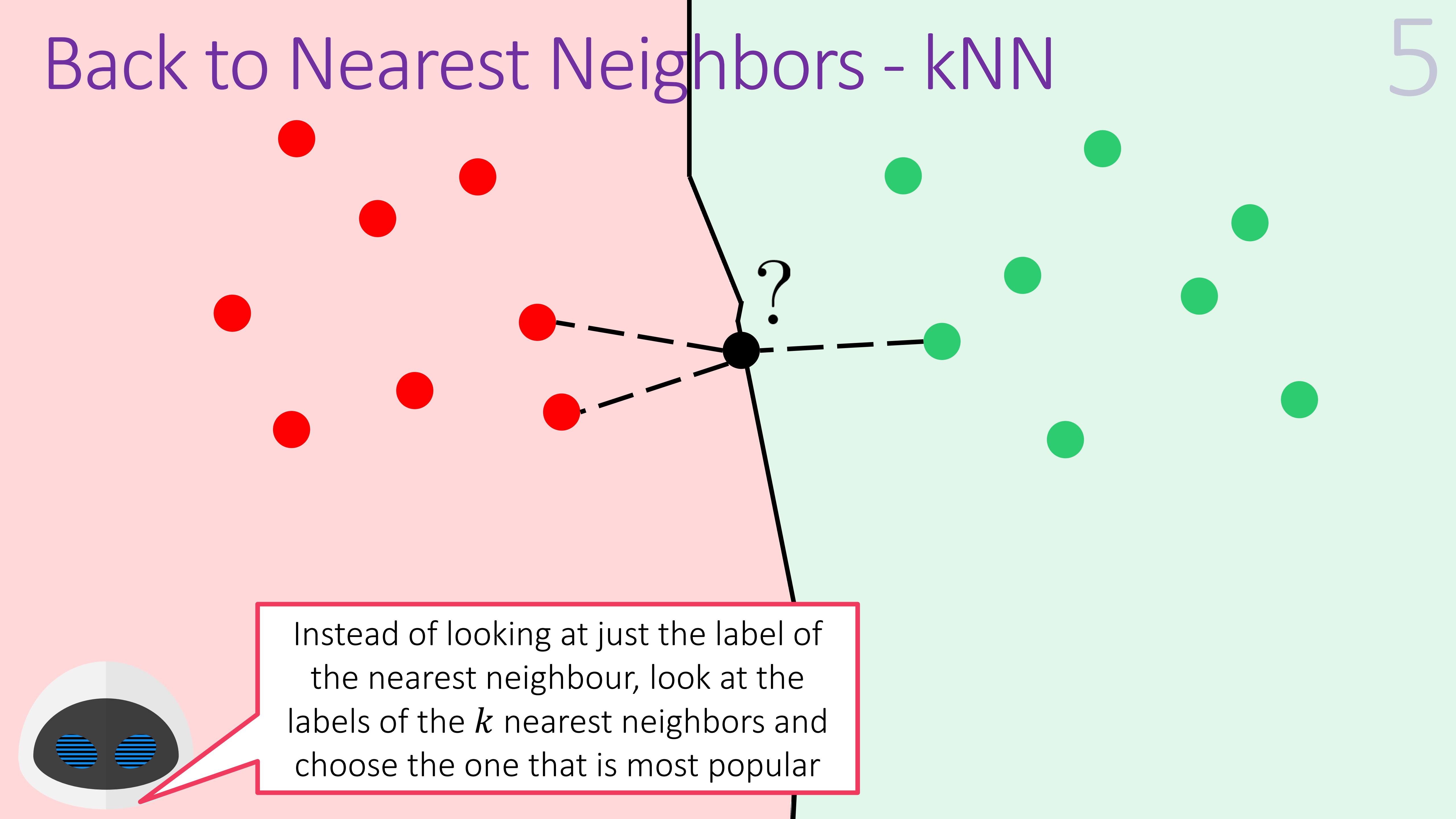
What if a point lies on the decision boundary? How will you classify that?

Depends on the application – if you want to be careful, you can classify boundary points as normal to avoid causing Mary to lose a potentially important email

In fact, if we were doing, *active learning*, we would have asked Mary to tell us the true class of not just points on the boundary but also of those close to it

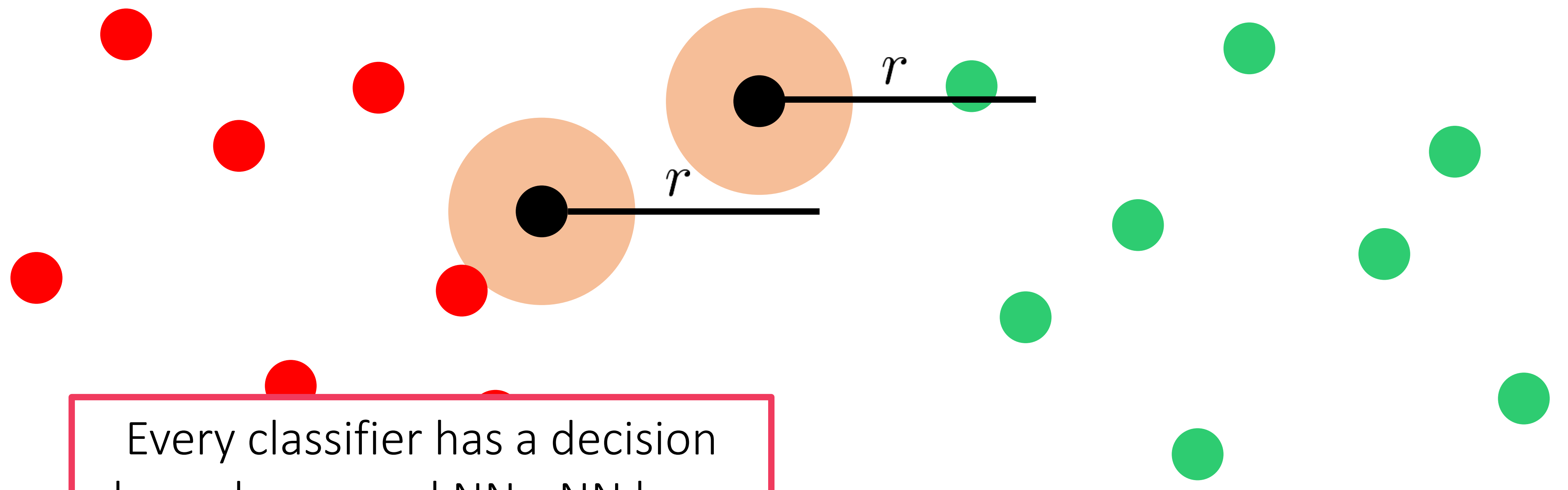
What is the decision boundary of LwP classifier?

Back to Nearest Neighbors - kNN

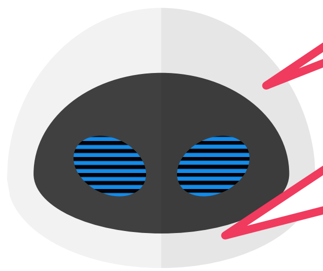


Instead of looking at just the label of the nearest neighbour, look at the labels of the k nearest neighbors and choose the one that is most popular

Back to Nearest Neighbors - rNN

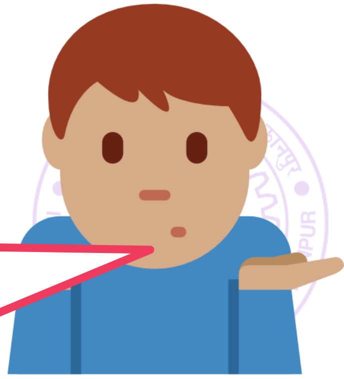


Every classifier has a decision boundary even kNN, rNN have some decision boundary (which we hope are better than 1NN's)



Look at all neighbors who are within an r radius of the test point and choose the label most popular among the neighbors

How should I decide which value of k or r to use? Also, should I use kNN or rNN?



Hyperparameter Tuning in ML

Constants like k in kNN, r in rNN, or even the metric to use in LwP are called *hyperparameters* of an ML algorithm

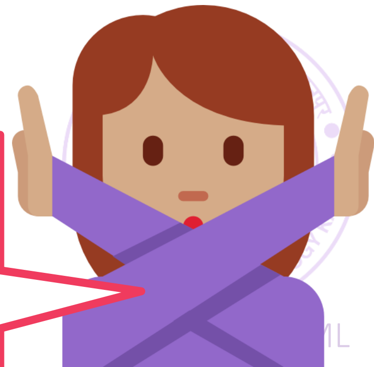
Just a fancy name (model vectors like $\mathbf{w}$ are called *parameters*)

We usually tune these hyperparameters by setting them to a value that gives us highest test accuracy

Take out a part of the training data and pretend it is test data for the purpose of hyperparameter tuning ☺

This part of data that is a mock test for us is called *validation data*

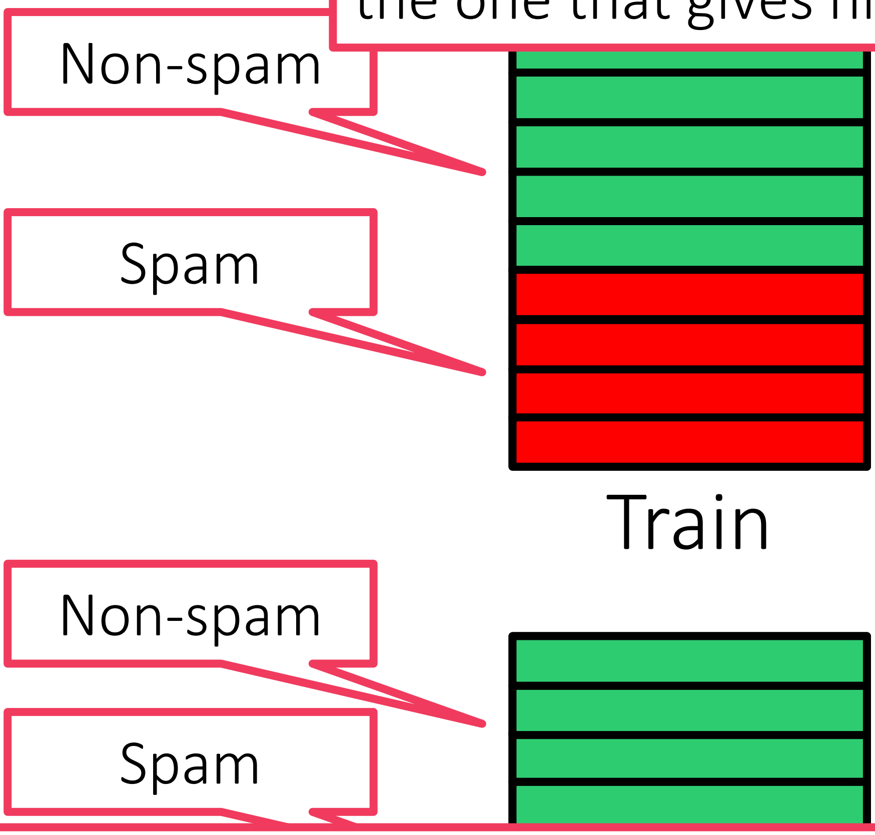
Let us look at the two most popular ways of creating such validation datasets



Looking at test data during training is an execution-worthy crime!

Held-out Val

If you wish to tune k for example, go over all values of k you think are valid, take only the “train” set as training set (i.e. don’t use validation data to train) and choose the one that gives highest accuracy on the validation set



Train

Train

Instead, pretend this is test

Validation

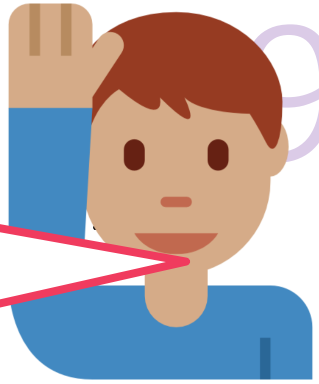
Never touch this during training

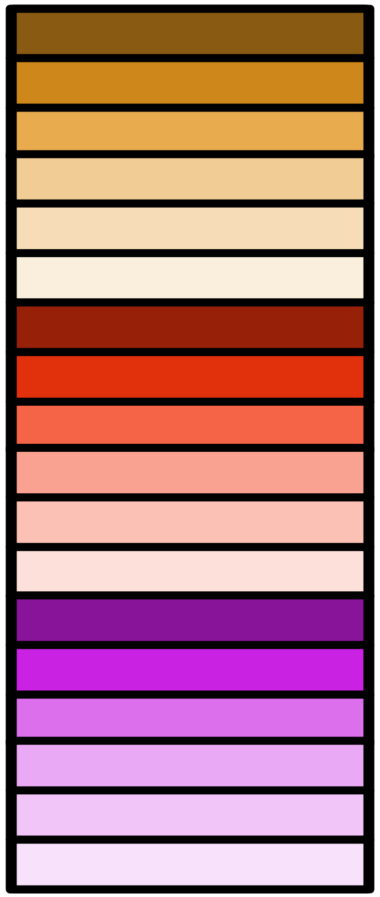
st

If you are concerned that relying on only one split may cause you to choose a wrong value of the hyperparameter (if the split is unlucky). If you feel this is happening, use multi-fold cross validation

Multi-*fol*

In this case, we choose the hyperparameter value (e.g. value of k) which gives the highest average validation accuracy across all splits. Clearly this is more expensive than simple held-out validation but offers less chance of an unlucky split





Train Train Train

Validation Validation Validation

Learning with NN – Lessons

One of the oldest learning algorithms 1)

Very in possible

In prac data

However, a

s

Also notice, with NN, all training need to be stored. In LwP, I could throw away training points – lightweight!

Compare this to LwP where model had 2 vectors no matter how many training points – such models are called *parametric models*

Correct! NN requires huge storage too! Note that the model size goes up with the amount of training data ☹

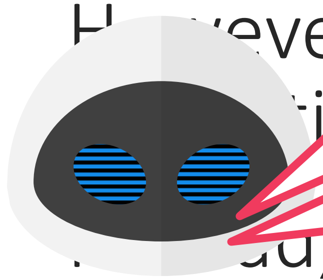
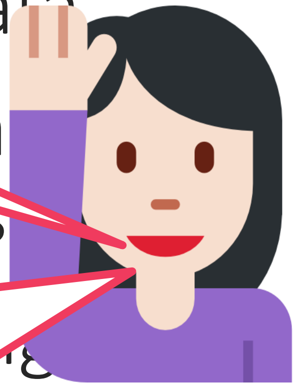
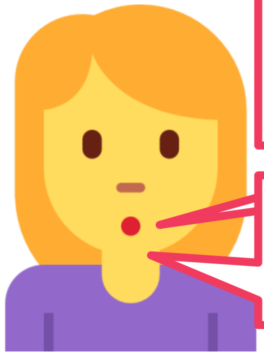
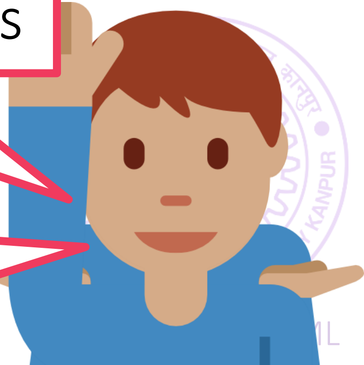
Models whose size depends on the amount of training data are called *non-parametric* models

Makes sense. If there are 2M training points, each a 10K-dim vector, then naively finding nearest neighbor takes 20B operations i.e. ~20 seconds @ 2GHz

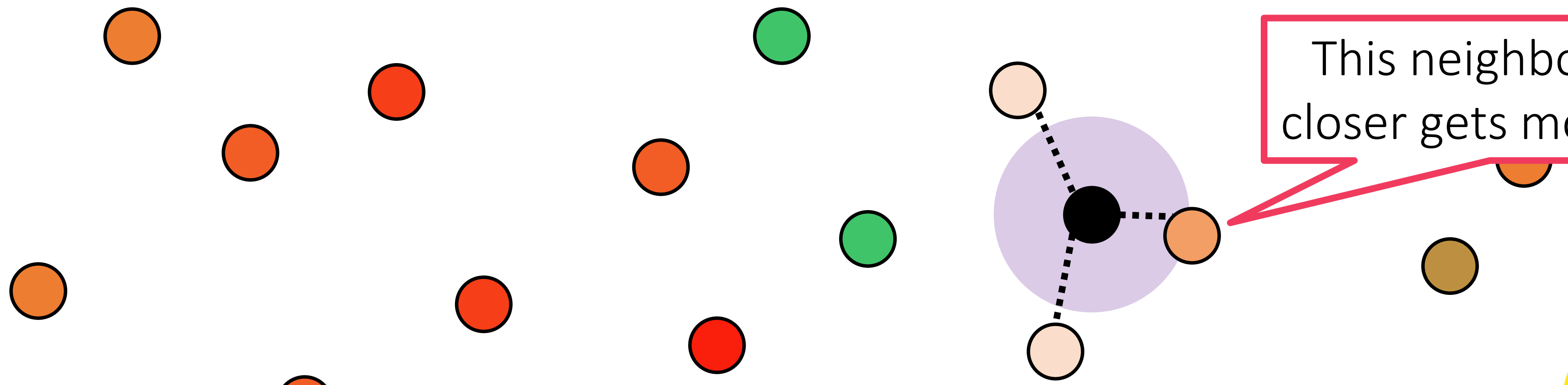
The entire training set is the model – NNs have huge models

Wait! So what is the "model" for NN?

Imagine if a bank website took 20 seconds to verify if a credit transaction is valid!

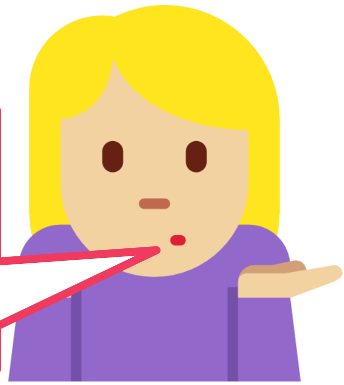





Regression with rNN

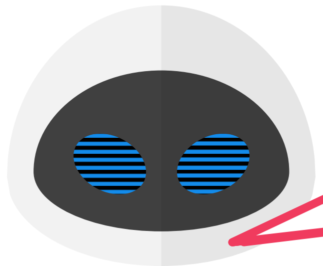


This neighbour being closer gets more weight

$$\bullet = \left\{ \frac{1}{3} \circ + \frac{3}{3} \circ + \frac{1}{3} \circ \right\}$$



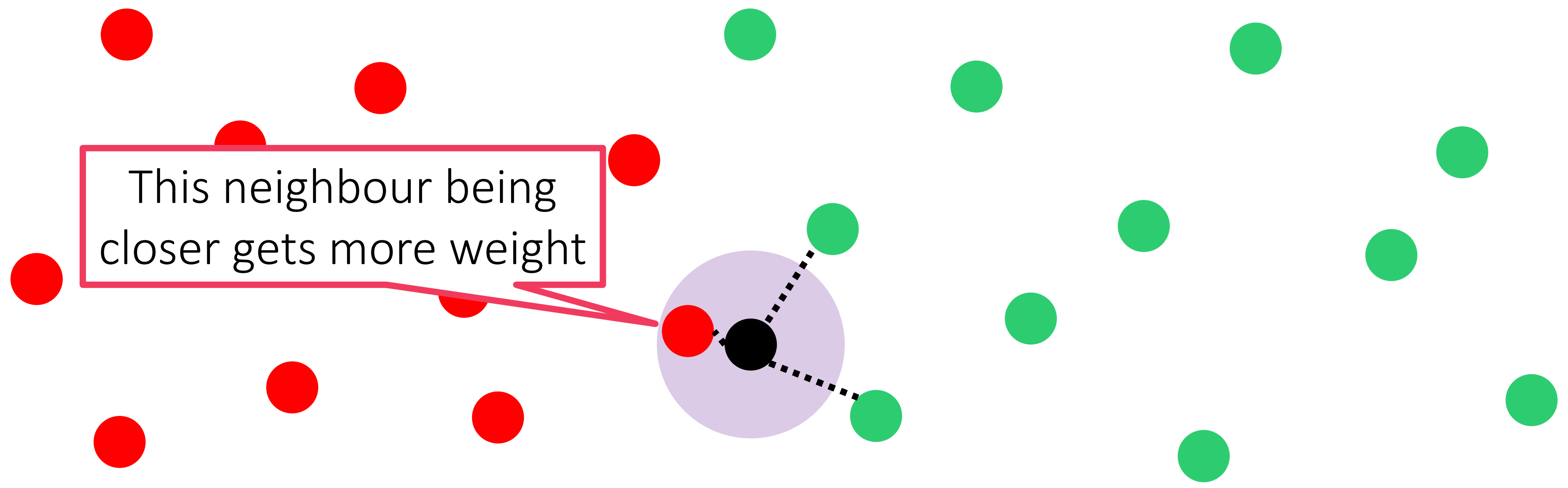
Can I use this trick with kNN? Can I use it while doing classification?



Yes, and yes. Need to be a bit careful while doing weighted classification since adding labels makes no sense

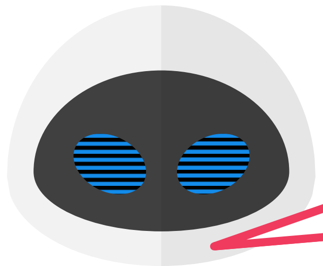
E.g. when data point is a student and we wish to predict their marks

Classification with Weighted rNN



This neighbour being closer gets more weight

● = { $\frac{1}{5}$ 🟢 $\frac{3}{5}$ 🔴 $\frac{1}{5}$ 🟢 }



Green gets 1/5 + 1/5 = 2/5 votes whereas Red gets 3/5 votes – Red wins!!

This method elegantly works even if there are more than 2 classes!