

Nonlinear Classification

INFO-4604, Applied Machine Learning
University of Colorado Boulder

October 5-10, 2017

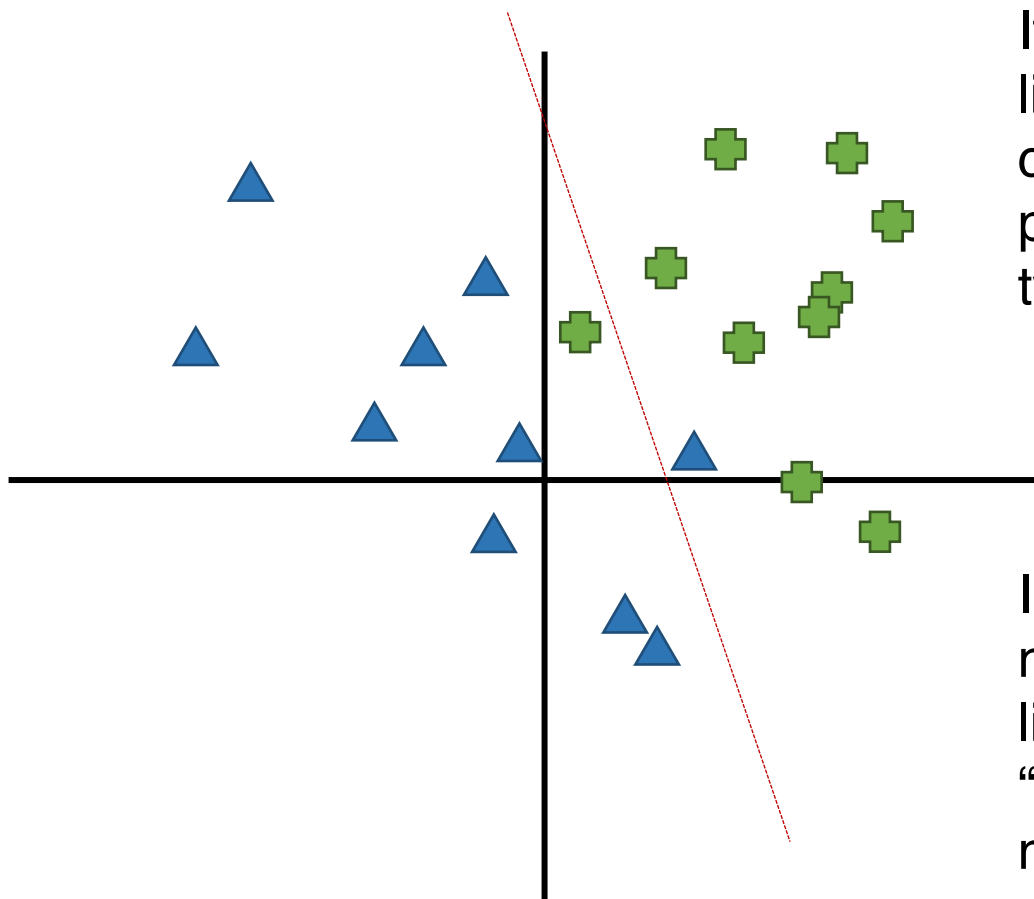
Prof. Michael Paul

Linear Classification

Most classifiers we've seen use **linear** functions to separate classes:

- Perceptron
- Logistic regression
- Support vector machines
(unless kernelized)

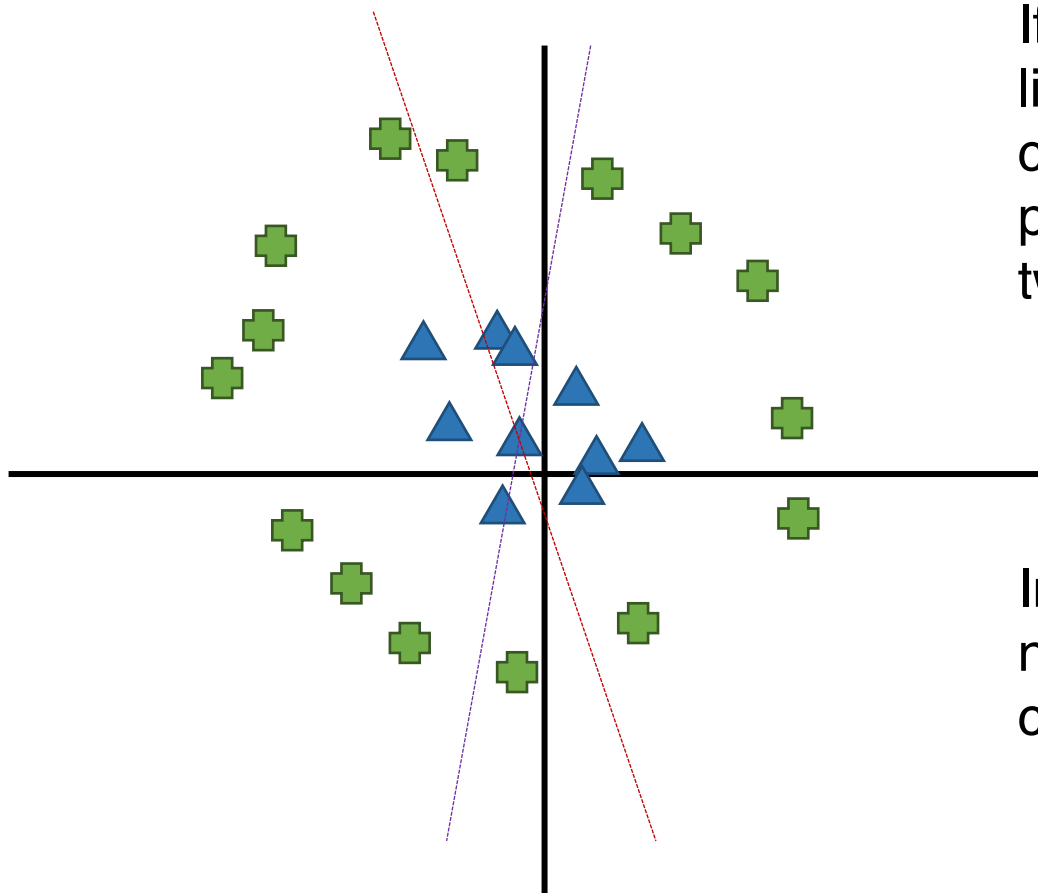
Linear Classification



If the data are not linearly separable, a linear classification cannot perfectly distinguish the two classes.

In many datasets that are not linearly separable, a linear classifier will still be “good enough” and classify most instances correctly.

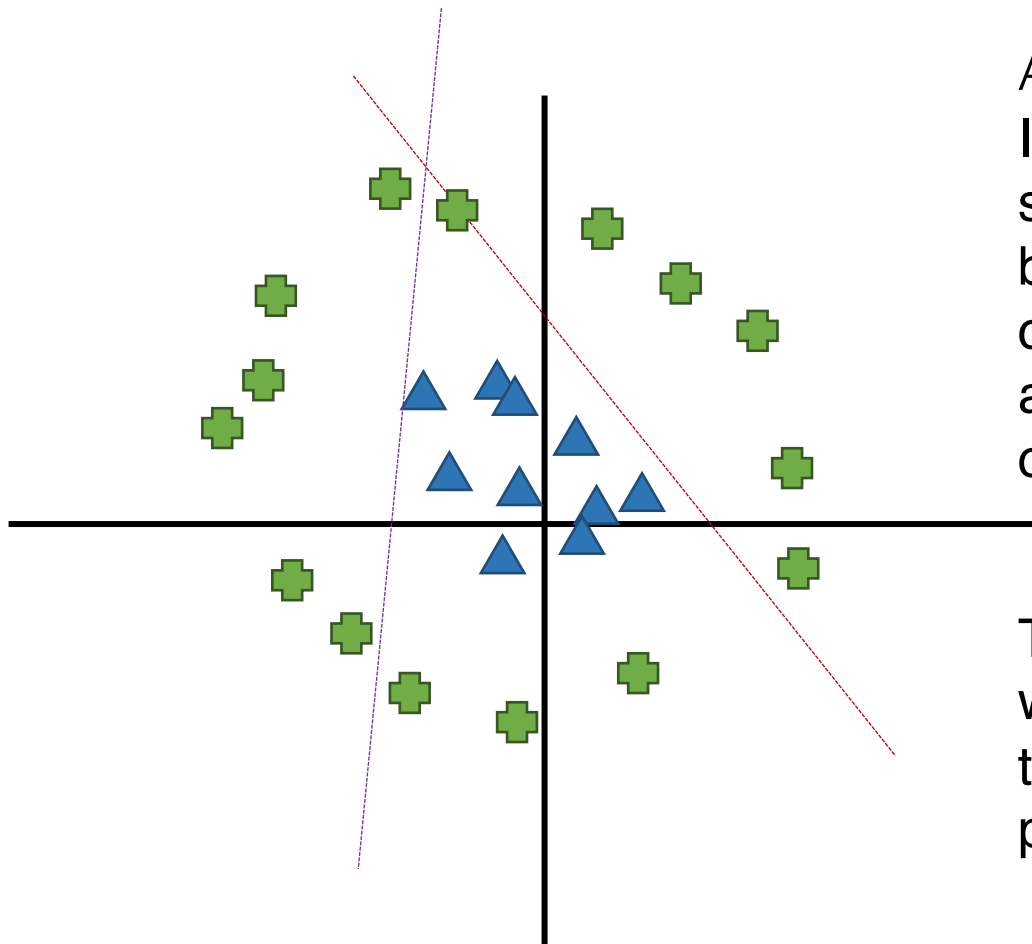
Linear Classification



If the data are not linearly separable, a linear classification cannot perfectly distinguish the two classes.

In some datasets, there is no way to learn a linear classifier that works well.

Linear Classification



Aside:

In datasets like this, it might still be possible to find a boundary that isolates one class, even if the classes are mixed on the other side of the boundary.

This would yield a classifier with decent *precision* on that class, despite having poor overall *accuracy*.

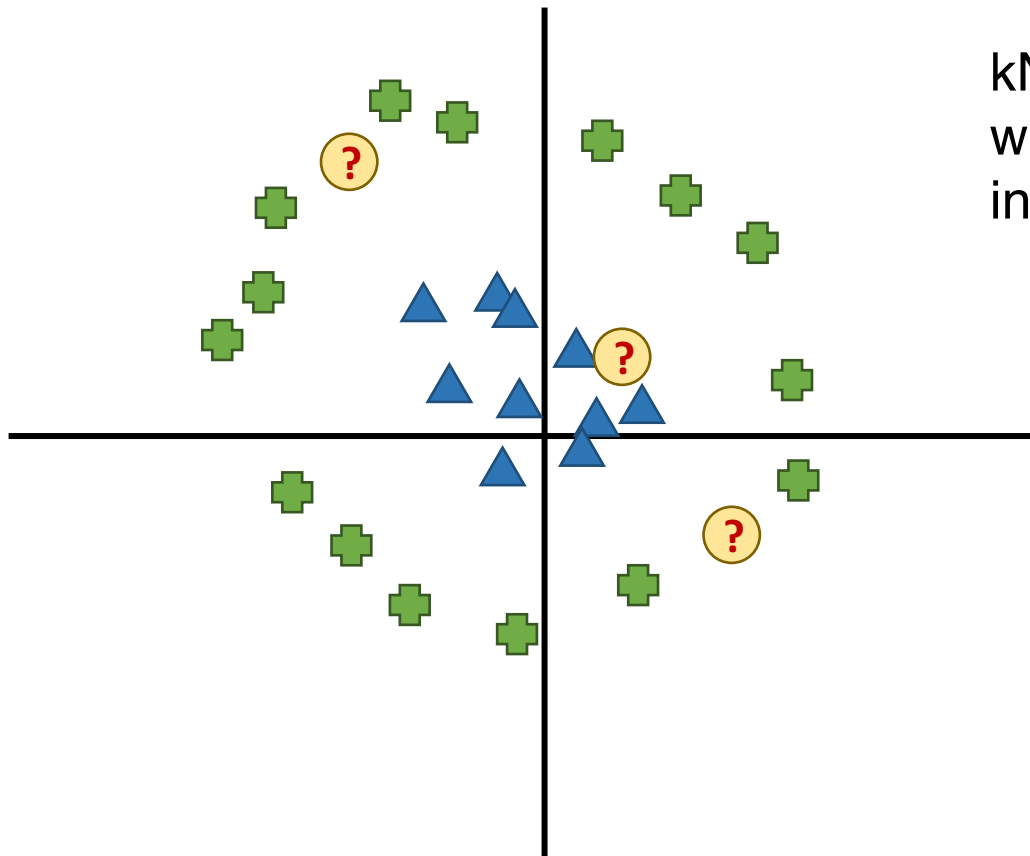
Nonlinear Classification

Nonlinear functions can be used to separate instances that are not linearly separable.

We've seen two nonlinear classifiers:

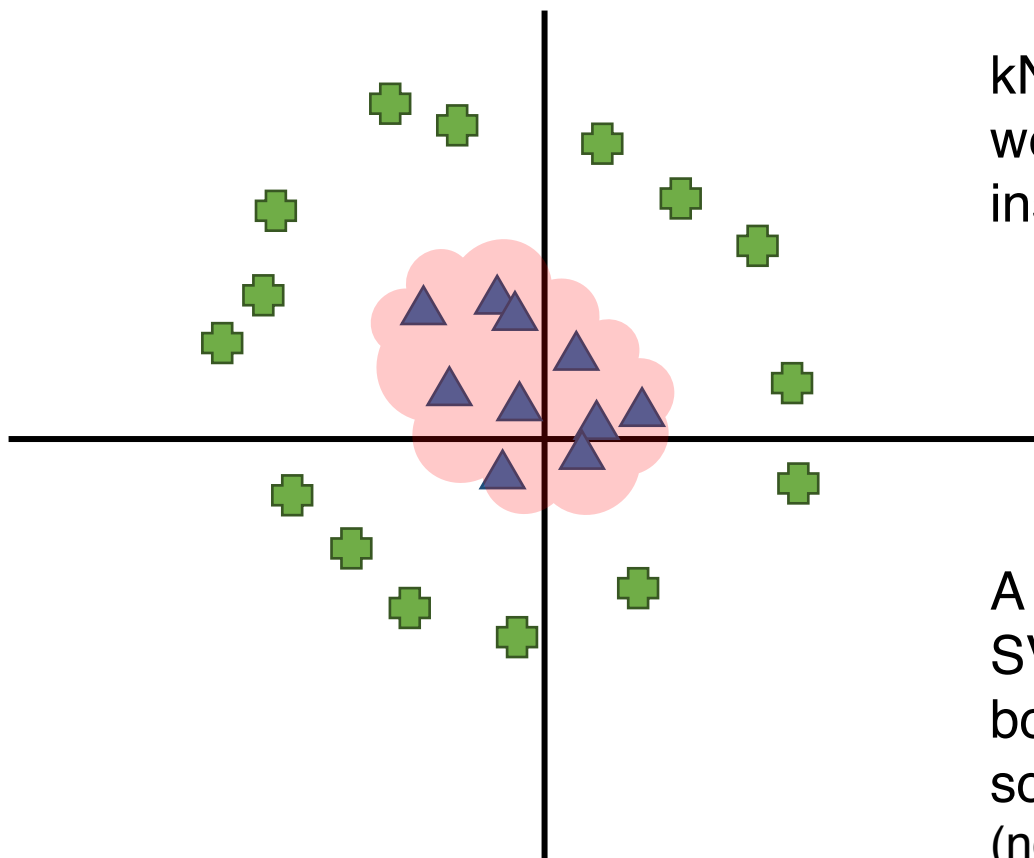
- k-nearest-neighbors (kNN)
- Kernel SVM
 - Kernel SVMs are still implicitly learning a linear separator in a higher dimensional space, but the separator is nonlinear in the original feature space.

Nonlinear Classification



kNN would probably work well for classifying these instances.

Nonlinear Classification



kNN would probably work well for classifying these instances.

A Gaussian/RBF kernel SVM could also learn a boundary that looks something like this.
(not exact; just an illustration)

Nonlinear Classification

Both kNN and kernel methods use the concept of distance/similarity to training instances

Next, we'll see two nonlinear classifiers that make predictions based on features instead of distance/similarity:

- Decision tree
- Multilayer perceptron
 - A basic type of *neural network*

Decision Trees

What color is the cat in this photo?



Calico

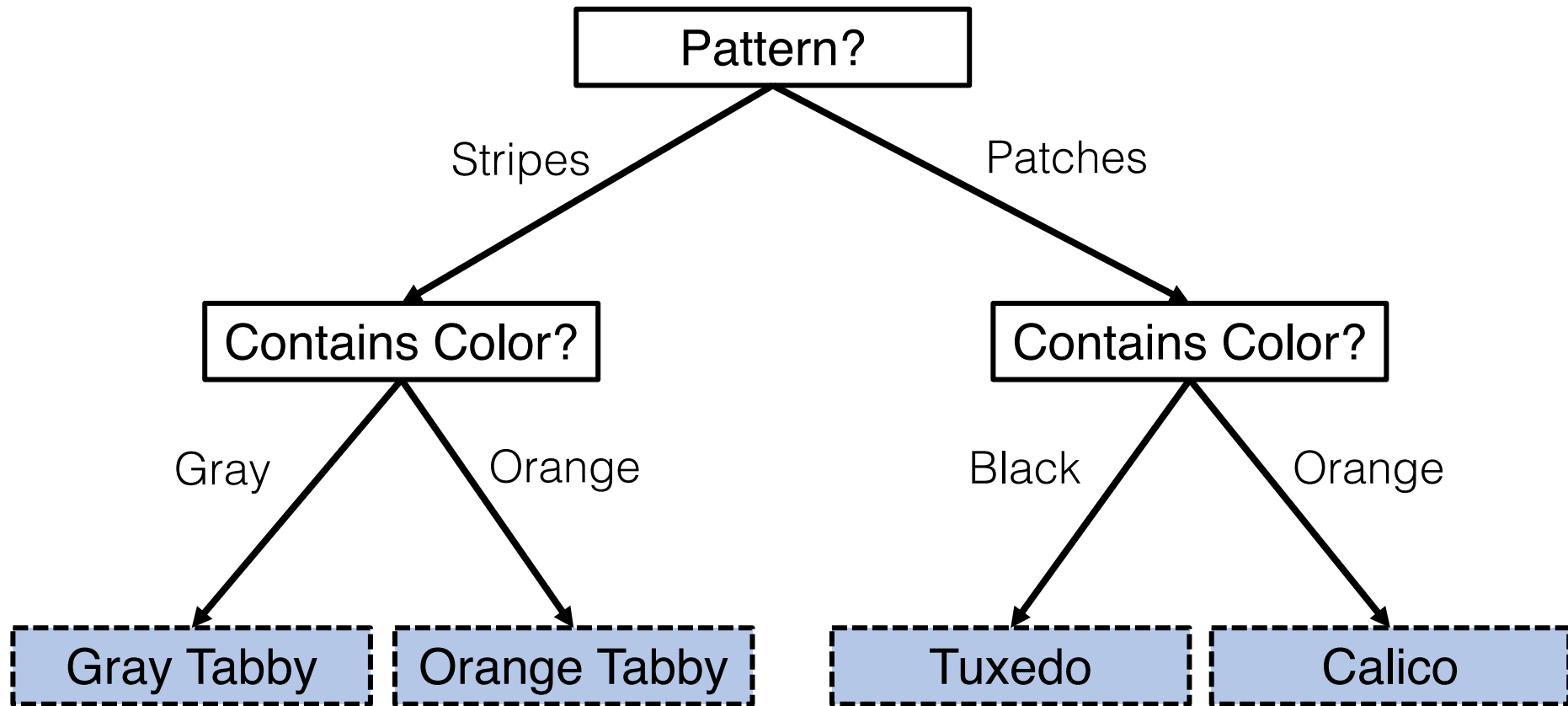


Orange Tabby

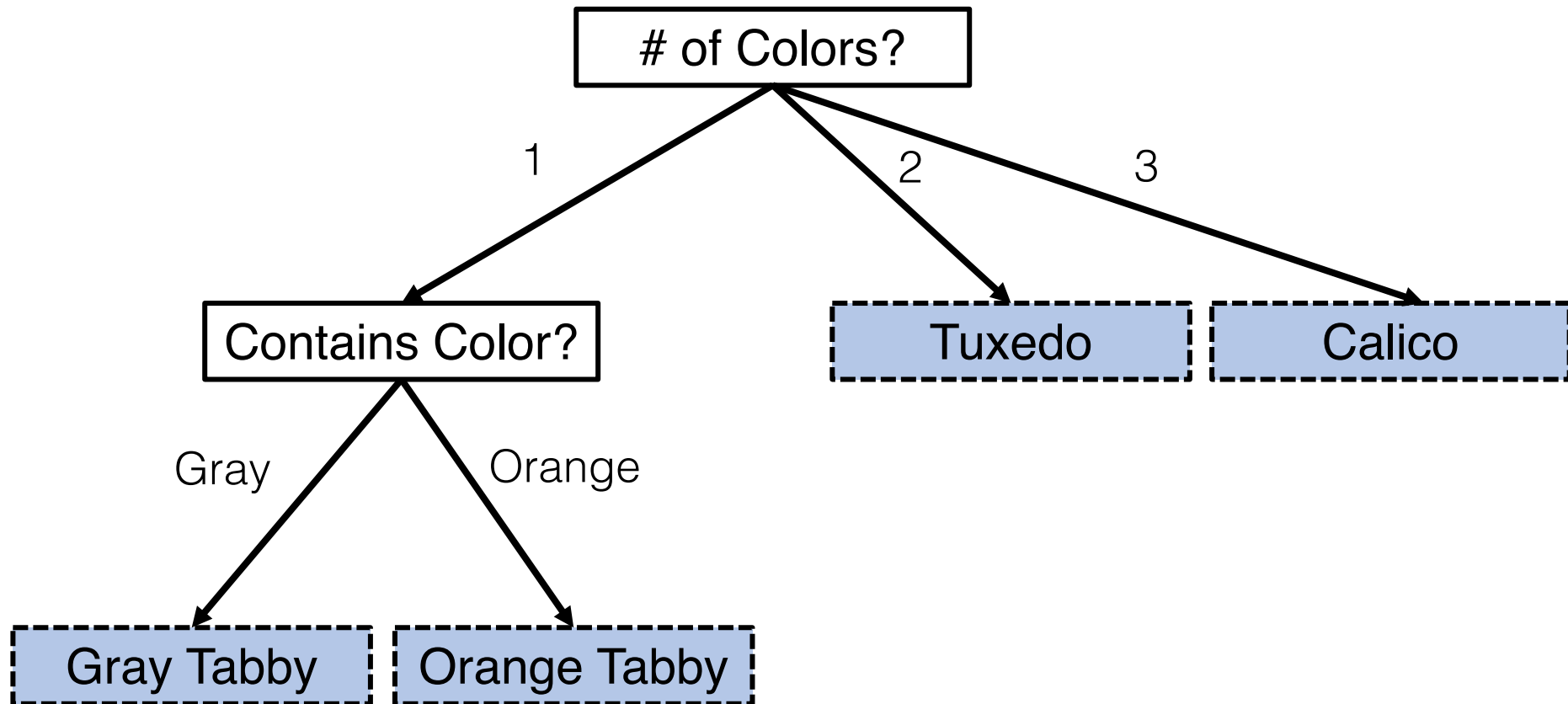


Tuxedo

Decision Trees



Decision Trees



Decision Trees

Decision tree classifiers are structured as a **tree**, where:

- *nodes* are features
- *edges* are feature values
 - If the values are numeric, an edge usually corresponds to a range of values (e.g., $x < 2.5$)
- *leaves* are classes

To classify an instance:

Start at the root of the tree, and follow the branches based on the feature values in that instance. The final node is the final prediction.

Decision Trees

We won't cover how to *learn* a decision tree in detail in this class (see book for more detail)

General idea:

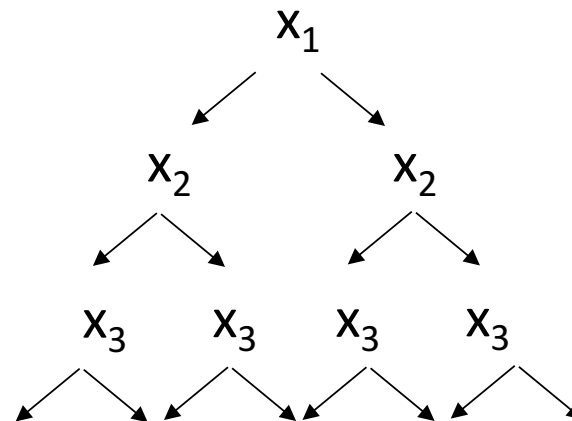
1. Pick the feature that best distinguishes classes
 - If you group the instances based on their value for that feature, some classes should become more likely
 - The distribution of classes should have low *entropy* (high entropy means the classes are evenly distributed)
2. Recursively repeat for each group of instances
3. When all instances in a group have the same label, set that class as the final node.

Decision Trees

Decision trees can easily overfit.

Without doing anything extra, they will literally memorize training data!

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



A tree can encode all possible combinations of feature values.

Decision Trees

Two common techniques to avoid overfitting:

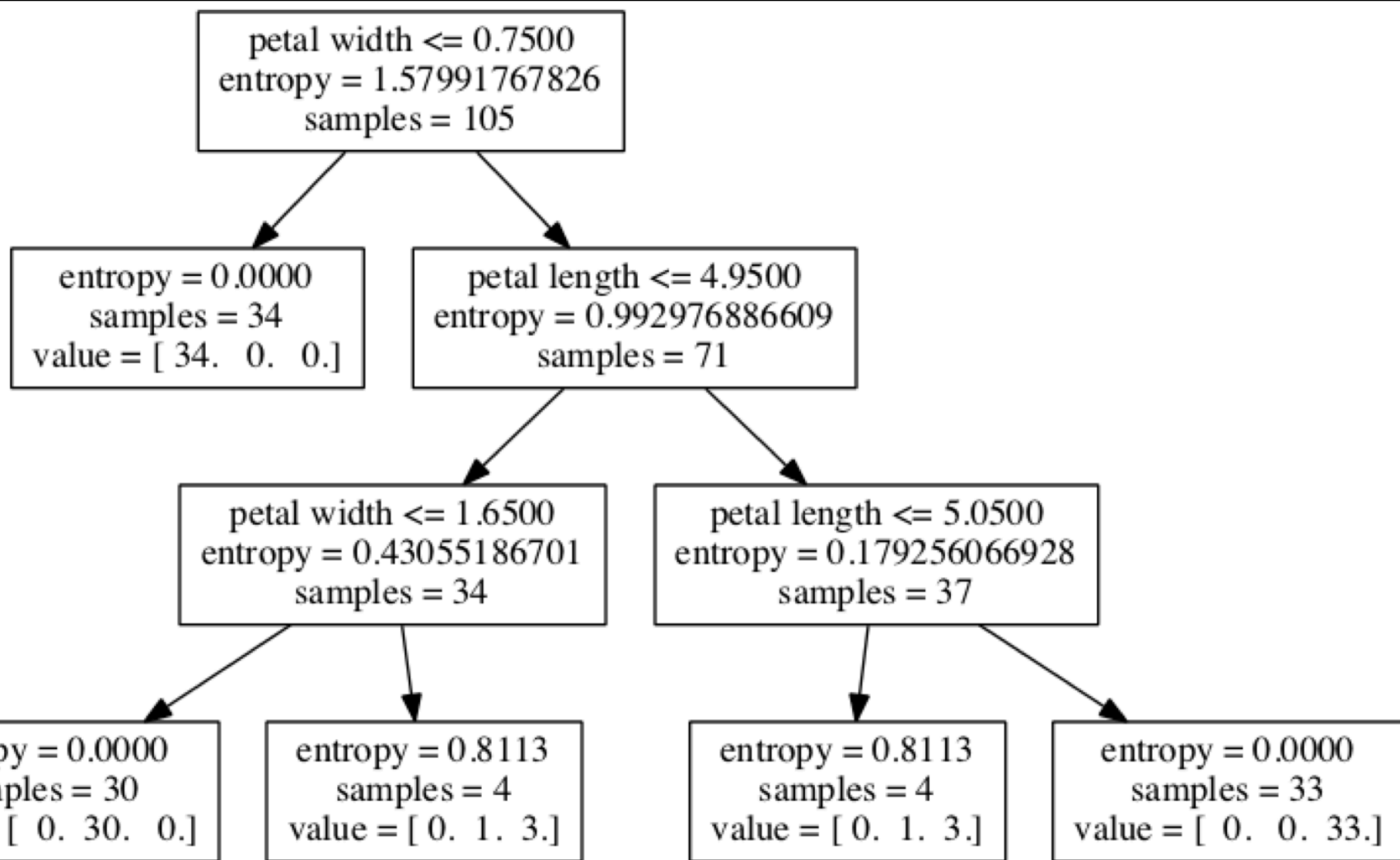
- Restrict the maximum depth of the tree
- Restrict the minimum number of instances that must be remaining before splitting further

If you stop creating a deeper tree before all instances at that node belong to the same class, use the majority class within that group as the final class for the leaf node.

Decision Trees

One reason decision trees are popular is because the algorithm is relatively easy to understand, and the classifiers are relatively **interpretable**.

- That is, it is possible to see how a classifier makes a decision.
- This stops being true if your decision tree becomes large; trees of depth 3 or less are easiest to visualize.



Decision Trees

Decision trees are powerful because they automatically use *conjunctions* of features (e.g., *Pattern*="striped" AND *Color*="Orange")

This gives *context* to the feature values.

- In a decision tree, *Color*="Orange" can lead to a different prediction depend on the value of the *Pattern* feature.
- In a linear classifier, only one weight can be given to *Color*="Orange"; it can't be associated with different classes in different contexts

Decision Trees

Decision trees naturally handle **multiclass** classification without making any modifications

Decision trees can also be used for **regression** instead of classification

- Common implementation: final prediction is the average value of all instances at the leaf

Random Forests

Random forests are a type of **ensemble** learning with decision trees (a *forest* is a set of trees)

We'll revisit this later in the semester, but it's useful to know that random forests:

- are one of the most successful types of ensemble learning
- avoid overfitting better than individual decision trees

Neural Networks

Recall from the book that perceptron was inspired by the way neurons work:

- a perceptron “fires” only if the inputs sum above a threshold (that is, a perceptron outputs a positive label if the score is above the threshold; negative otherwise)

Also recall that a perceptron is also called an *artificial neuron*.

Neural Networks

An **artificial neural network** is a collection of artificial neurons that interact with each other

- The outputs of some are used as inputs for others

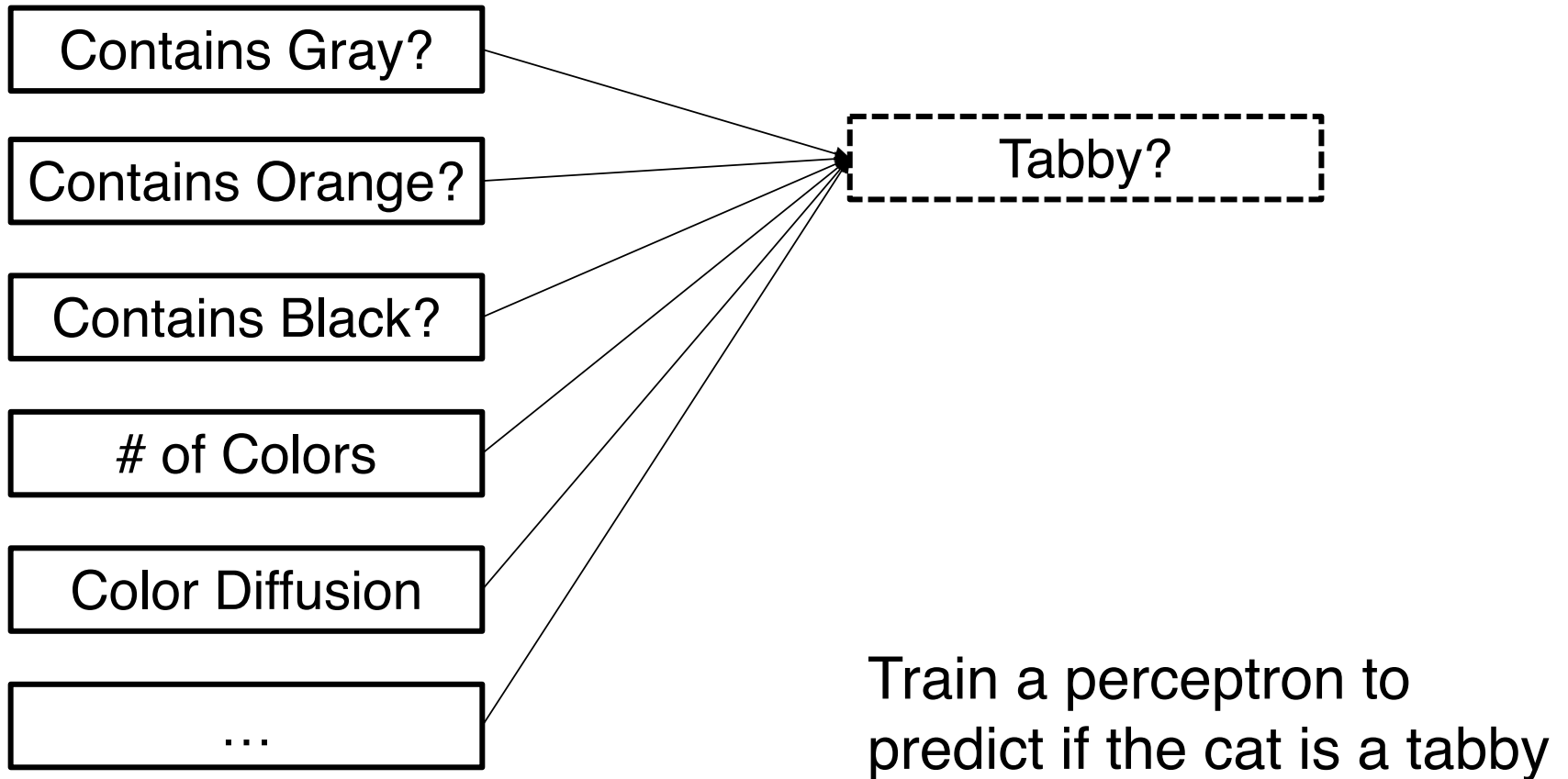
A **multilayer perceptron** is one type of neural network which combines multiple perceptrons

- Multiple *layers* of perceptrons, where each layer's output “feeds” into the next layer as input
- Called a *feed-forward* network

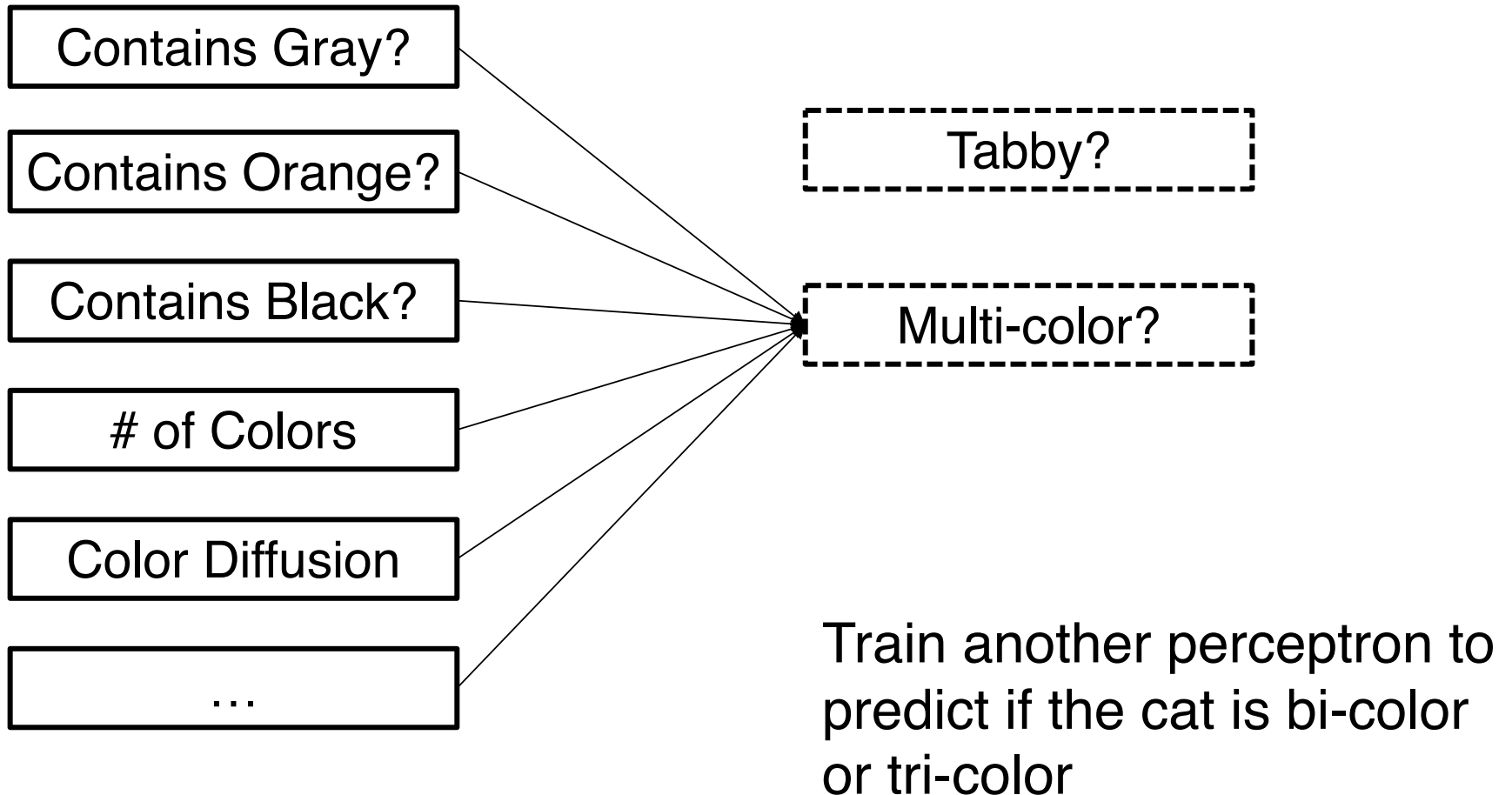
Multilayer Perceptron

Let's start with a cartoon about how a multilayer perceptron (MLP) works conceptually.

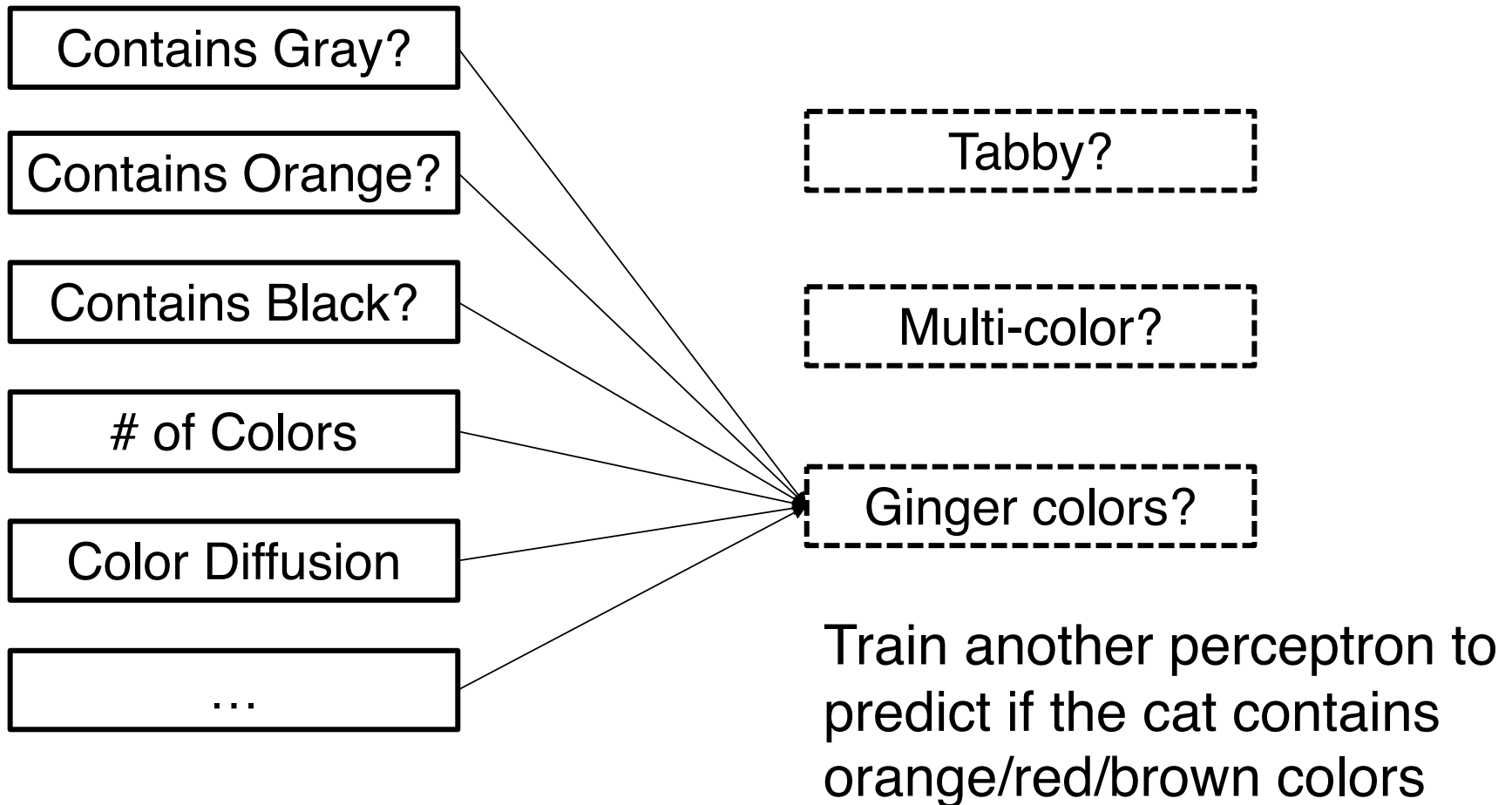
Multilayer Perceptron



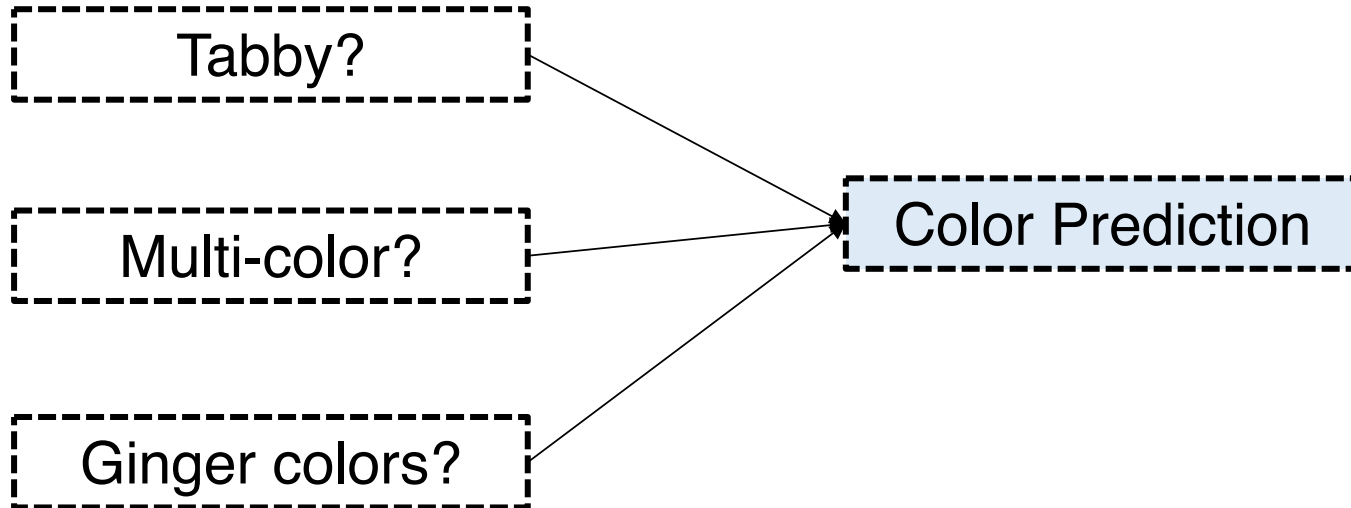
Multilayer Perceptron



Multilayer Perceptron



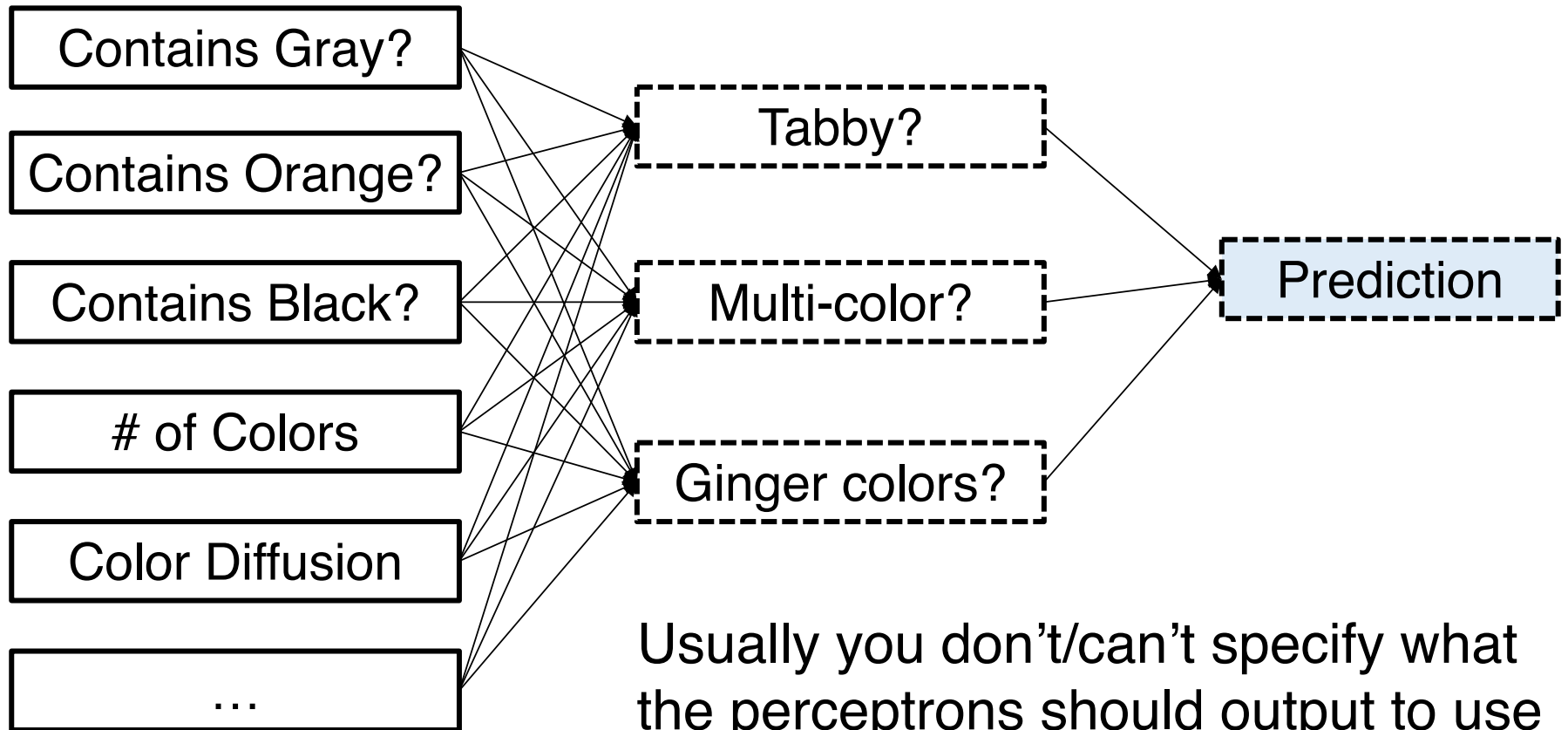
Multilayer Perceptron



Treat the outputs of your perceptrons as new features

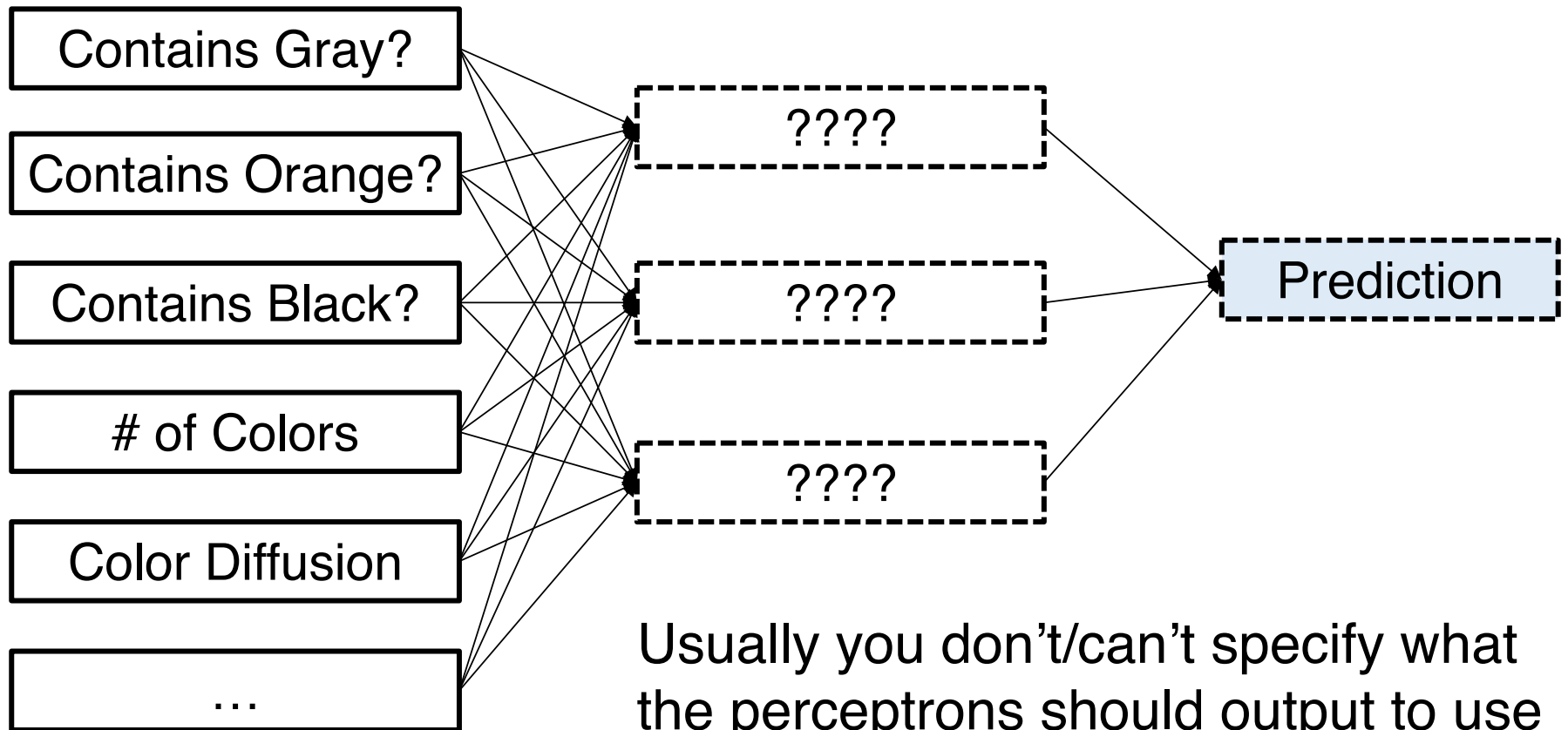
Train another perceptron on these new features

Multilayer Perceptron



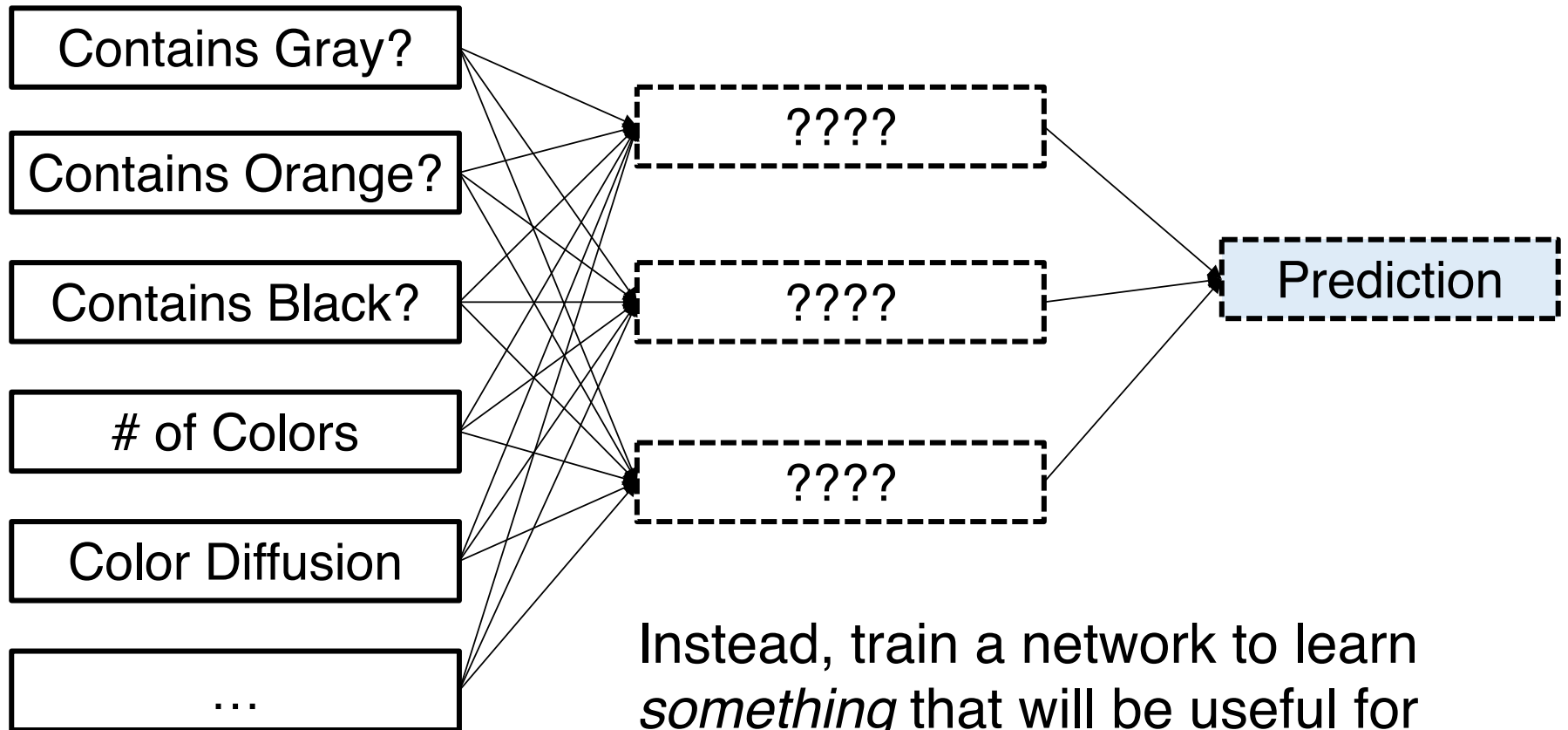
Usually you don't/can't specify what the perceptrons should output to use as new features in the next layer.

Multilayer Perceptron



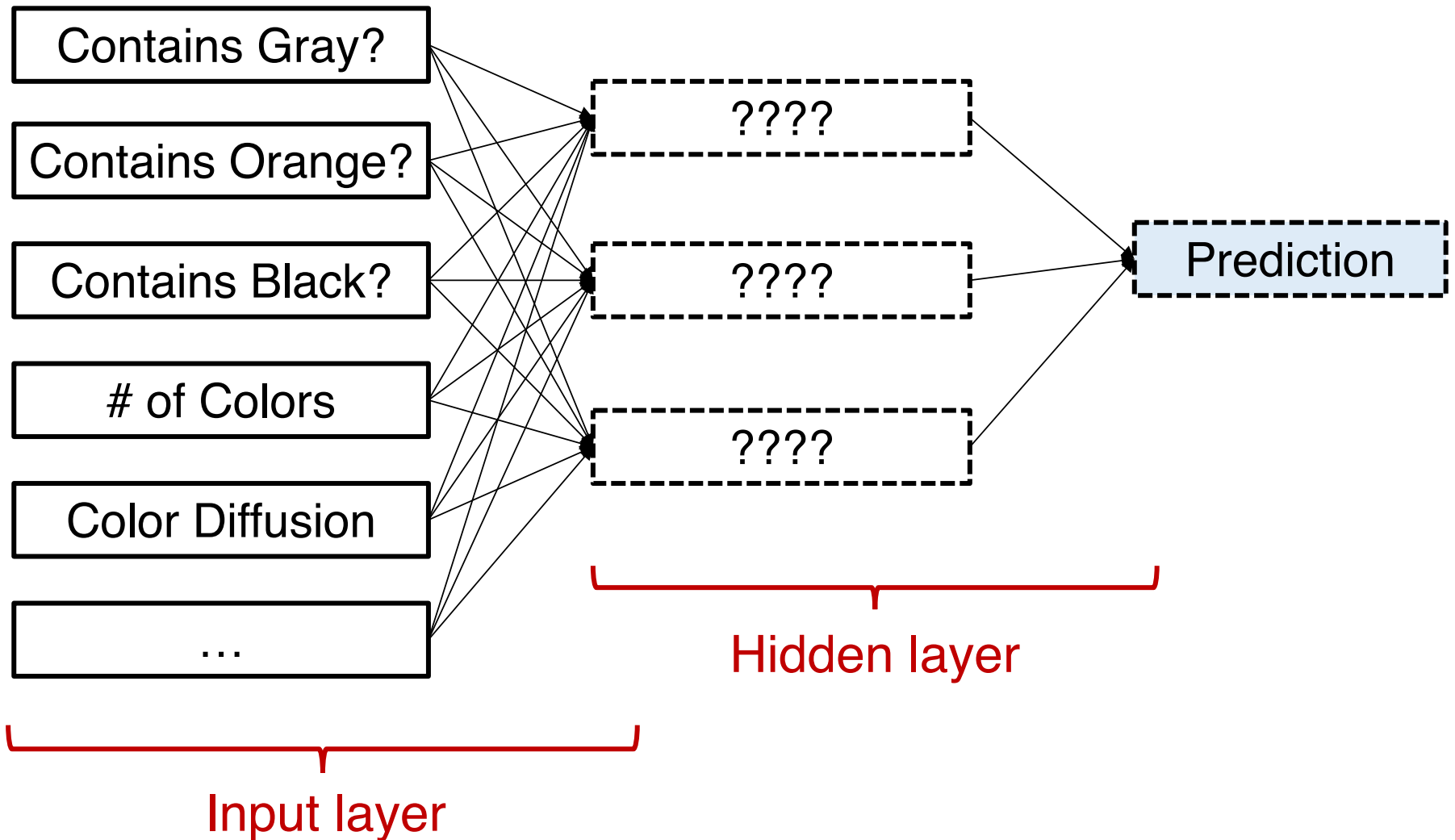
Usually you don't/can't specify what the perceptrons should output to use as new features in the next layer.

Multilayer Perceptron



Instead, train a network to learn *something* that will be useful for prediction.

Multilayer Perceptron



Multilayer Perceptron

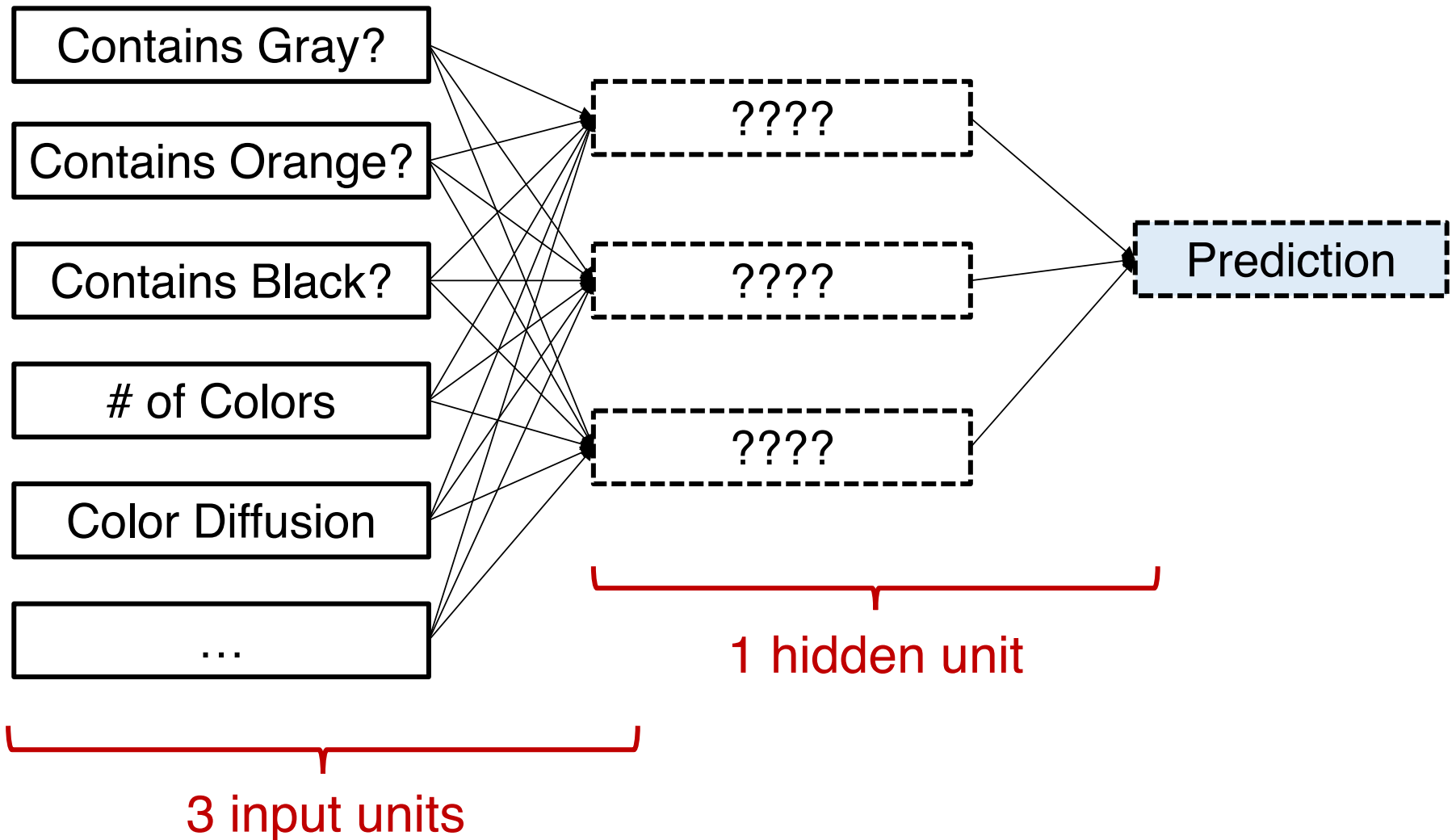
The **input layer** is the first set of perceptrons which output positive/negative based on the observed features in your data.

A **hidden layer** is a set of perceptrons that uses the outputs of the previous layer as inputs, instead of using the original data.

- There can be multiple hidden layers!
- The final hidden layer is also called the **output layer**.

Each perceptron in the network is called a **unit**.

Multilayer Perceptron



Activation Functions

Remember, perceptron defines a *score* $\mathbf{w}^T \mathbf{x}$, then the score is input into an *activation function* which converts the score into an output:

$$\phi(\mathbf{w}^T \mathbf{x}) = 1 \text{ if above } 0, -1 \text{ otherwise}$$

Logistic regression used the logistic function to convert the score into an output between 0 and 1:

$$\phi(\mathbf{w}^T \mathbf{x}) = 1 / (1 + \exp(-\mathbf{w}^T \mathbf{x}))$$

Activation Functions

Neural networks usually use also use the logistic function (or another sigmoid function) as the activation function

This is true even in multilayer perceptron

- Potentially confusing terminology:
The “units” in a multilayer perceptron aren’t technically perceptrons!

The reason is that calculating the perceptron threshold is not differentiable, so can’t calculate the gradient for learning.

Multilayer Perceptron

The final classification can be written out in full:

$$\phi(w_{211}(\phi(\mathbf{w}_{11}^T \mathbf{x})) + w_{212}(\phi(\mathbf{w}_{12}^T \mathbf{x})) + w_{213}(\phi(\mathbf{w}_{13}^T \mathbf{x})))$$

Equivalent to writing:

$$\phi(w_{21}^T \mathbf{y}), \text{ where } \mathbf{y} = \langle \phi(\mathbf{w}_{11}^T \mathbf{x}), \phi(\mathbf{w}_{12}^T \mathbf{x}), \phi(\mathbf{w}_{13}^T \mathbf{x}) \rangle$$

(above, the dot product has been expanded out)

Multilayer Perceptron

The final classification can be written out in full:

$$\phi(w_{211}(\underbrace{\phi(\mathbf{w}_{11}^T \mathbf{x})}_{\text{Scores of the three "perceptron" units in the first layer}})) + w_{212}(\underbrace{\phi(\mathbf{w}_{12}^T \mathbf{x})}_{\text{Scores of the three "perceptron" units in the first layer}}) + w_{213}(\underbrace{\phi(\mathbf{w}_{13}^T \mathbf{x})}_{\text{Scores of the three "perceptron" units in the first layer}}))$$

Scores of the three “perceptron” units in the first layer

Multilayer Perceptron

The final classification can be written out in full:

$$\phi(w_{211}(\underbrace{\phi(\mathbf{w}_{11}^T \mathbf{x}))}_{\text{Outputs of the three "perceptron" units in the first layer}}) + w_{212}(\underbrace{\phi(\mathbf{w}_{12}^T \mathbf{x}))}_{\text{Scores of the three "perceptron" units in the first layer}}) + w_{213}(\underbrace{\phi(\mathbf{w}_{13}^T \mathbf{x}))}_{\text{Scores of the three "perceptron" units in the first layer}})$$

Outputs of the three “perceptron” units in the first layer
(passing the three scores through the activation function)

{ Scores of the three “perceptron” units in the first layer

Multilayer Perceptron

The final classification can be written out in full:

$$\underbrace{\phi(w_{211}(\underbrace{\phi(\mathbf{w}_{11}^T \mathbf{x}))}_{\text{purple}}) + w_{212}(\underbrace{\phi(\mathbf{w}_{12}^T \mathbf{x}))}_{\text{purple}} + w_{213}(\underbrace{\phi(\mathbf{w}_{13}^T \mathbf{x}))}_{\text{purple}})}_{\text{orange}}$$

Score of the one “perceptron” unit in the second layer
(which uses the three outputs from the last layer as “features”)

{ Outputs of the three “perceptron” units in the first layer

{ Scores of the three “perceptron” units in the first layer

Multilayer Perceptron

The final classification can be written out in full:

$$\underbrace{\phi\left(w_{211}\underbrace{\underbrace{\phi(\mathbf{w}_{11}^T \mathbf{x})}_{\text{Outputs of the three "perceptron" units in the first layer}}}_{\text{Scores of the three "perceptron" units in the first layer}}\right) + w_{212}\underbrace{\underbrace{\phi(\mathbf{w}_{12}^T \mathbf{x})}_{\text{Outputs of the three "perceptron" units in the first layer}}}_{\text{Scores of the three "perceptron" units in the first layer}} + w_{213}\underbrace{\underbrace{\phi(\mathbf{w}_{13}^T \mathbf{x})}_{\text{Outputs of the three "perceptron" units in the first layer}}}_{\text{Scores of the three "perceptron" units in the first layer}}\right)}_{\text{Final output (passing the final score through the activation function)}}$$

Final output (passing the final score through the activation function)

{ Score of the one “perceptron” unit in the second layer

{ Outputs of the three “perceptron” units in the first layer

{ Scores of the three “perceptron” units in the first layer

Multilayer Perceptron

The final classification can be written out in full:

$$\phi(w_{211}(\phi(\mathbf{w}_{11}^T \mathbf{x})) + w_{212}(\phi(\mathbf{w}_{12}^T \mathbf{x})) + w_{213}(\phi(\mathbf{w}_{13}^T \mathbf{x})))$$

Can then define a loss function $L(\mathbf{w})$ based on the difference between classifications and true labels

- Can minimize with gradient descent (or related methods)
 - Algorithm called **backpropagation** makes gradient calculations more efficient
- Loss function is **non-convex**
 - Lots of local minima make it hard to find good solution

Multilayer Perceptron

How many hidden layers should there be?

How many units should there be in each layer?

You have to specify these when you run the algorithm.

- You can think of these as yet more *hyperparameters* to tune.

Nonlinear Example: XOR

Consider two binary features x_1 and x_2 and you want to learn to output the function, $x_1 \text{ XOR } x_2$.

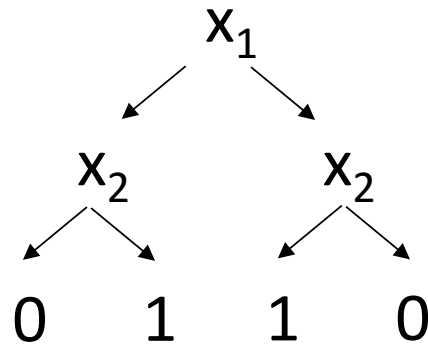
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

A linear classifier would not be able to learn this, but decision trees and neural networks can.

Nonlinear Example: XOR

A decision tree can simply learn that:

- if $x_1=0$, output 0 if $x_2=0$ and output 1 if $x_2=1$
- if $x_1=1$, output 1 if $x_2=0$ and output 0 if $x_2=1$



Nonlinear Example: XOR

We can learn this with a MLP with 2 input units and 1 hidden unit.

Input units:

$$w_{11} = \langle 0.6, 0.6 \rangle, b_{11} = -1.0$$

$$w_{12} = \langle 1.1, 1.1 \rangle, b_{12} = -1.0$$

- Unit 1 will only output 1 if both $x_1=1$ and $x_2=1$
- Unit 2 will only output 1 if either $x_1=1$ or $x_2=1$

Nonlinear Example: XOR

Input units:

- Unit 1 will only output 1 if both $x_1=1$ and $x_2=1$
- Unit 2 will only output 1 if either $x_1=1$ or $x_2=1$

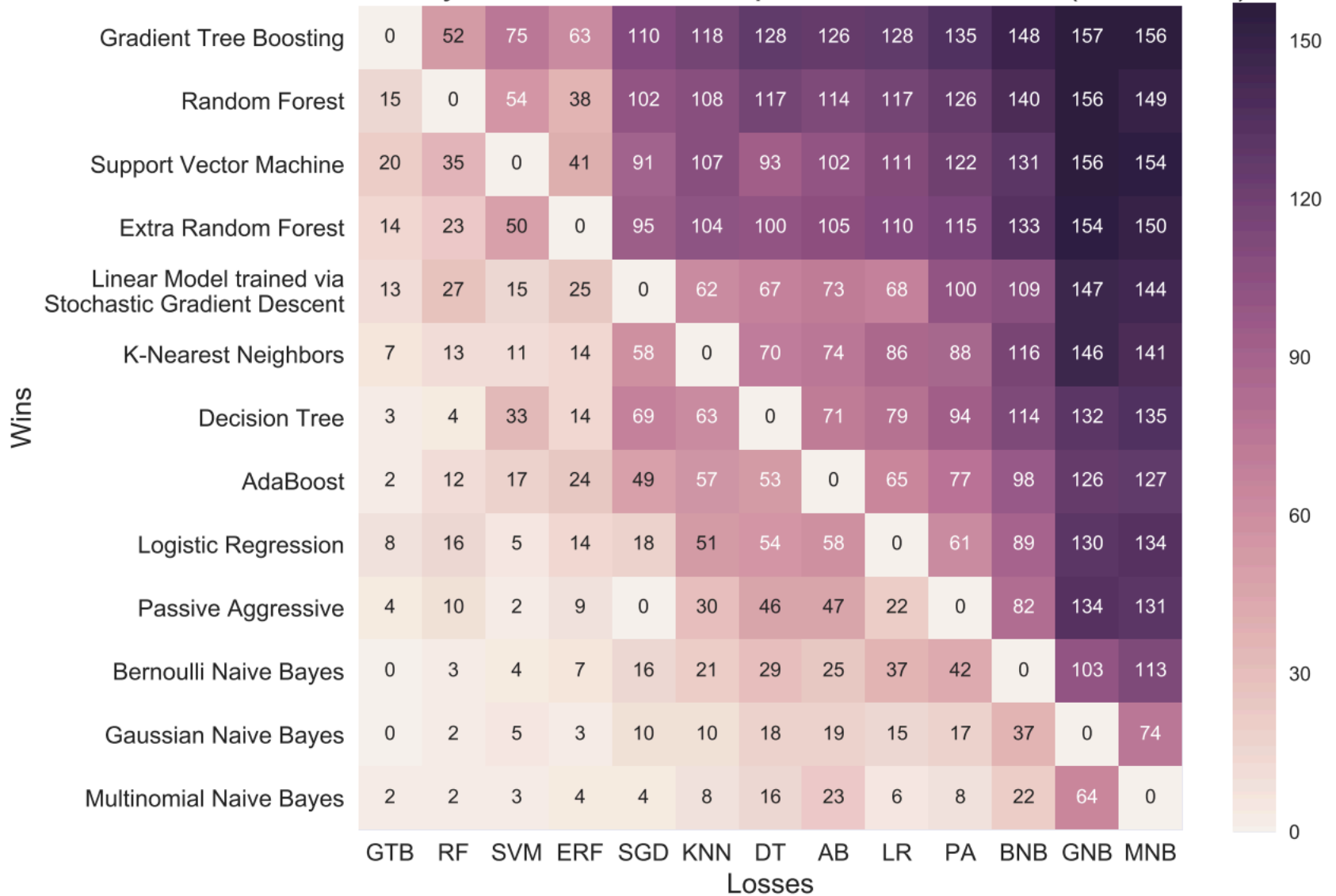
Hidden unit:

$$w_2 = \langle -2.0, 1.1 \rangle, b_2 = -1.0$$

- If feature 1 (Unit 1 output) is 1, this will output 0
- If both features (Units 1 and 2) are 0, this will output 0
- If only feature 2 (Unit 2 output) is 1, this will output 1

What Classifier to Use?

How many times model X outperformed model Y (out of 165)



Data-driven Advice for Applying Machine Learning to Bioinformatics Problems.

Randal S. Olson, William La Cava, Zairah Mustahsan, Akshay Varik, Jason H. Moore.