Machine Learning: Nearest Neighbors

ROB 102: Introduction to AI & Programming

Lecture 11

2021/11/22

Last time…

Image classification is a type of **supervised learning** where we predict the class of an image using labelled data.

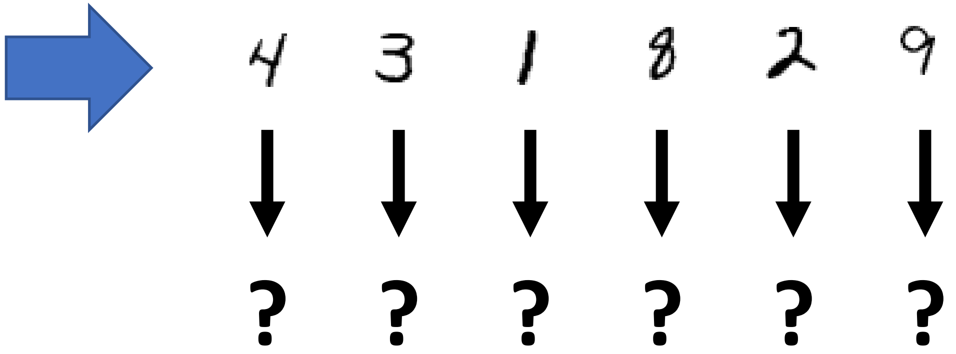


Image: CC4.0 (link)

Last time…

Machine Learning Algorithm:

Training time:

Learn a prediction model by optimizing over a labelled dataset.

Testing time:

Use the model to perform prediction on new data.

Data Split:

Training set: Labelled data used for training a machine learning algorithm.

Test set: Data used to test the accuracy of the machine learning algorithm.

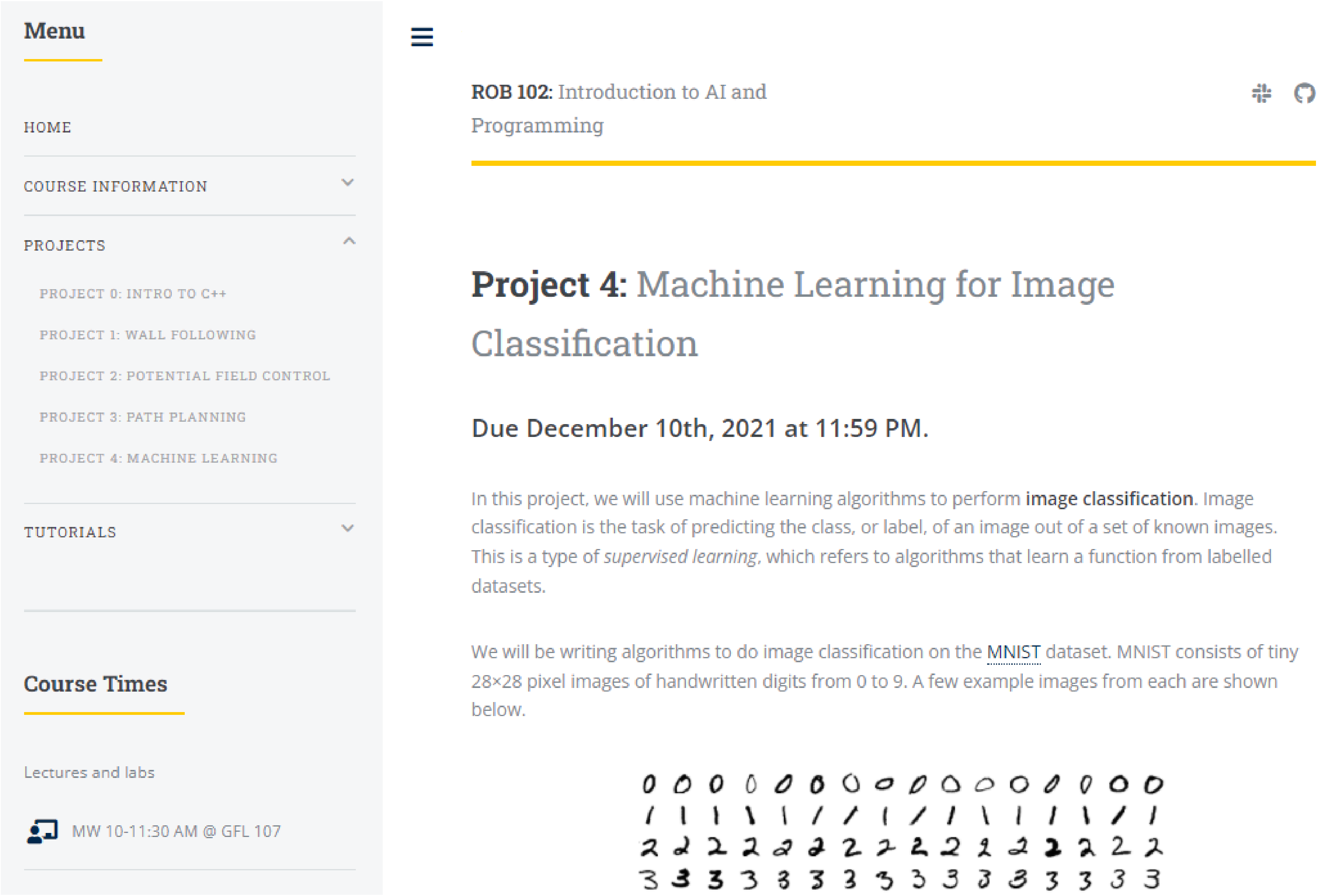
Project 4: Machine Learning

Implement three machine learning algorithms to classify images from the MNIST dataset.

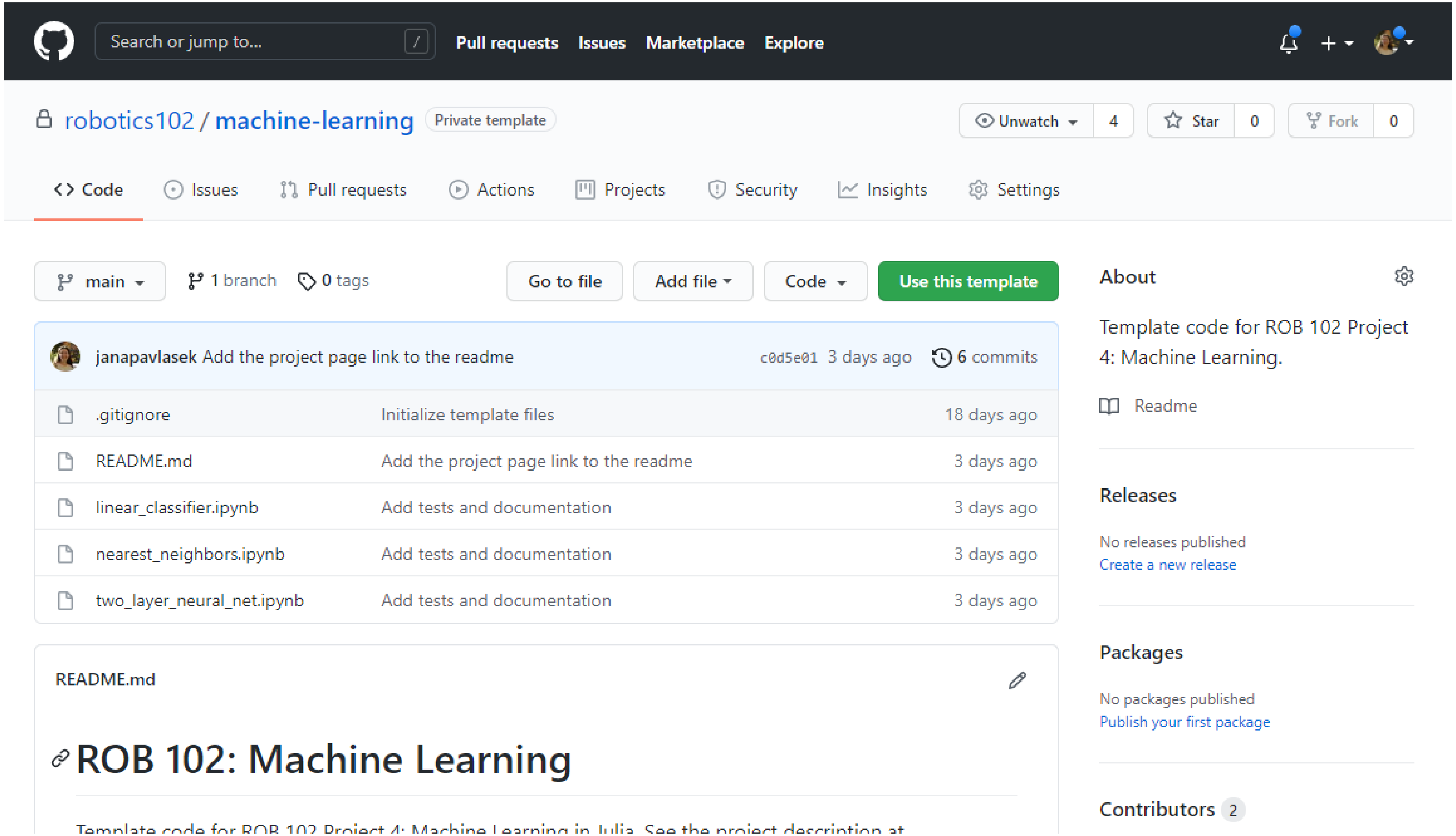
1. Nearest neighbors
2. Linear Classifier
3. Neural Network

The assignment instructions are available!

https://robotics102.github.io/projects/a4



The template code is available! Use the Github Classroom link to join.



Project 4: Machine Learning

Implement three machine learning algorithms to classify images from the MNIST dataset.

1. **Nearest neighbors** (Today!)
2. Linear Classifier
3. Neural Network

Image Classification on MNIST

Imagine we have 60k labelled images. How can we predict the class of a new image?

Image Classification on MNIST

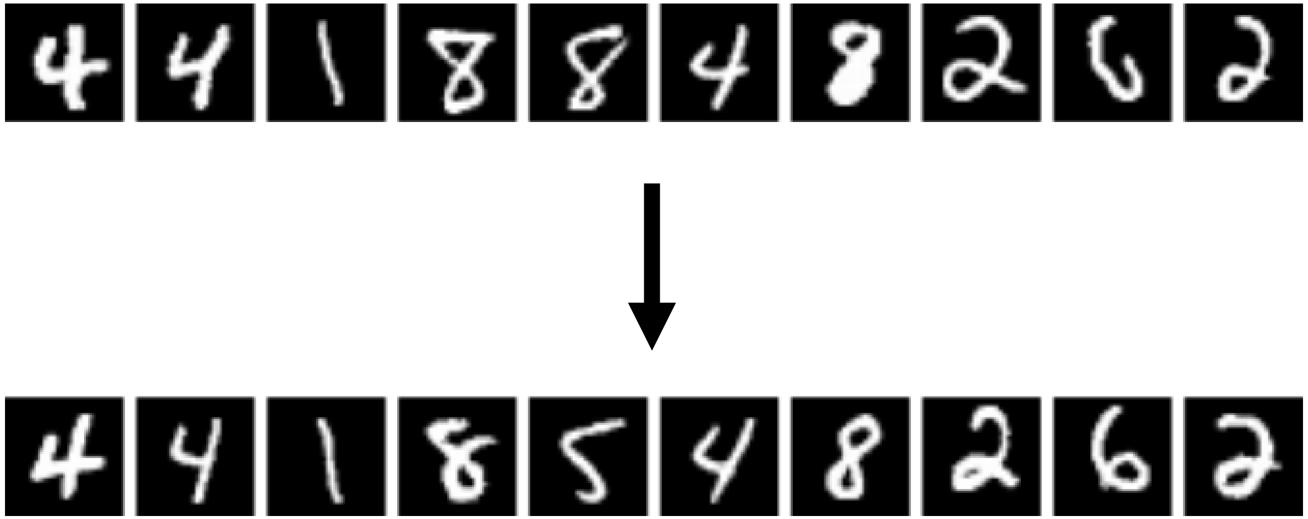
Imagine we have 60k labelled images. How can we predict the class of a new image?

Idea: This image of a two might be *numerically close* to other images of twos.

Nearest Neighbors

Idea: Given a new image, find the closest image in the training set. Then, assign the same label to the new image.

Test images



Nearest training images

Nearest Neighbors: Project 4.1

In Project 4 (Part 1), you will implement an algorithm to classify images using Nearest Neighbors.

jupyter nearest_neighbors Last Checkpoint: 5 hours ago (autosaved)

File Edit View Insert Cell Kernel Help

Run Code

Part I: Nearest Neighbors

This notebook implements a nearest neighbors classifier.

Imports

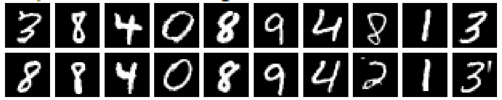
Some imports we'll need.

In [2]:
```
using MLDatasets
using Plots
using Images
using MosaicViews
using LinearAlgebra
using Random
using Printf
```

```
num_viz = 10
idx = shuffle(1:N_test)[1:num_viz]
nearest = x_train[indices[idx], :, :]

imgs = [x_test[idx, :, :]; nearest]
# imgs = reshape(imgs', width, height, num_viz * 2)
imgs = MNIST.convert2image(permutedims(imgs, (2, 3, 1)))

for i in 1:num_viz
    img = idx[i]
    @printf("Img %-2d -> Predicted: %-10s True: %s\n", i, y_pred[img], y_test[img])
end

println("\nTop row = test image, bottom row = nearest neighbor.")
mosaicview(imgs, fillvalue=1, nrow=2, npad=3, rowmajor=true)
```

```
Img 1  -> Predicted: 8          True: 3
Img 2  -> Predicted: 8          True: 8
Img 3  -> Predicted: 4          True: 4
Img 4  -> Predicted: 0          True: 0
Img 5  -> Predicted: 8          True: 8
Img 6  -> Predicted: 9          True: 9
Img 7  -> Predicted: 4          True: 4
Img 8  -> Predicted: 2          True: 8
Img 9  -> Predicted: 1          True: 1
Img 10 -> Predicted: 3          True: 3

Top row = test image, bottom row = nearest neighbor.
```

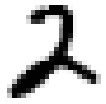
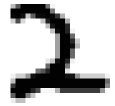
Out[36]:



Nearest Neighbors

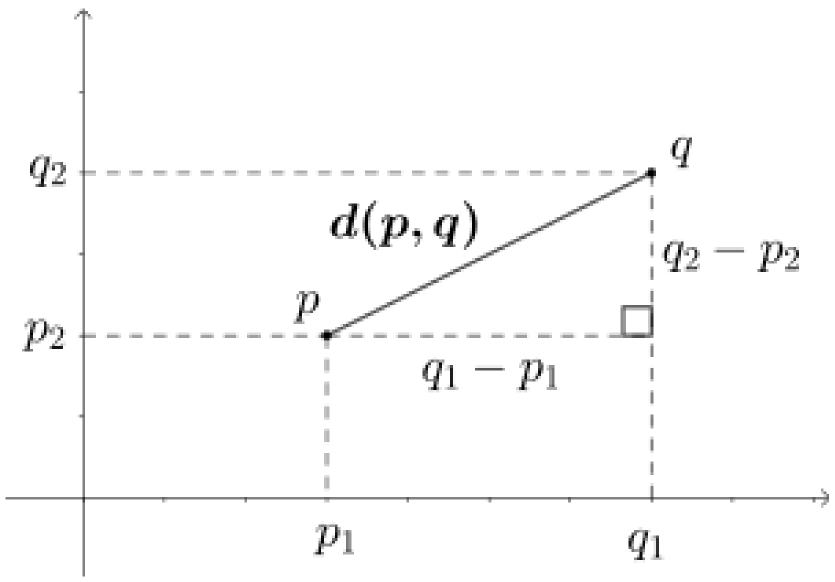
Idea: Given a new image, find the closest image in the training set. Then, assign the same label to the new image.

What does "nearest" mean?

distance( , )

Euclidean Distance

Recall: The Pythagorean Theorem gives us the distance:



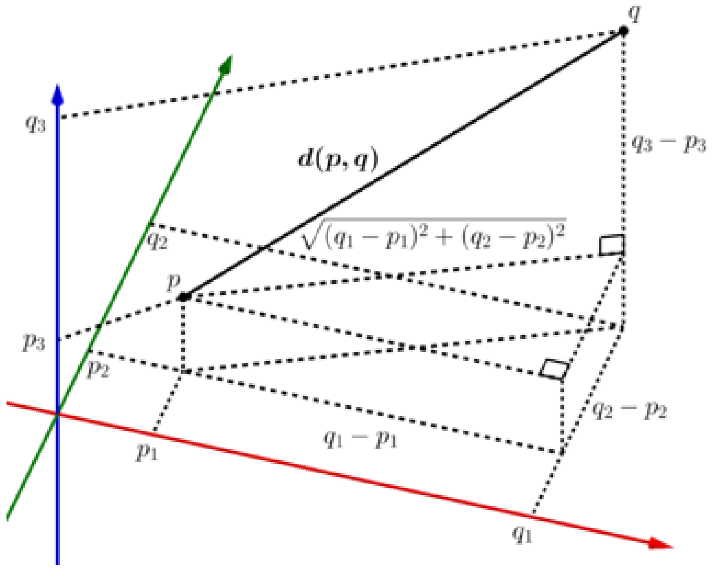
In 2D:

$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

distance

Euclidean Distance

Recall: The Pythagorean Theorem gives us the distance:



In 3D:

$$d(p, q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + (q_3 - p_3)^2}$$

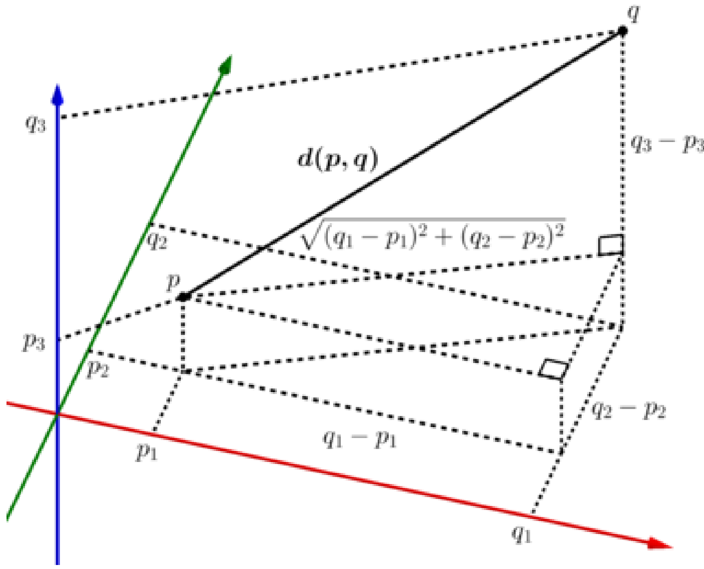
distance

Euclidean Distance

Recall: The Pythagorean Theorem gives us the distance:

In 3D:

$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + (q_3 - p_3)^2}$$



In N-D:

$$d(p,q) = \sqrt{\sum_{i=1}^{N} (q_i - p_i)^2}$$

distance

$$= \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_N - p_N)^2}$$

Euclidean Distance: Example

$$d(p,q) = \sqrt{\sum_{i=1}^{N} (q_i - p_i)^2}$$

Image A $\quad$ Image B

$$\begin{bmatrix} -1 & 2 & 0 \\ 3 & -3 & 1 \\ 0 & 1 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 2 & 0 \\ 1 & -2 & 2 \\ 1 & 1 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} -1 & 0 & 0 \\ 2 & -1 & -1 \\ -1 & 0 & -1 \end{bmatrix}$$

Euclidean Distance: Example

$$d(p,q) = \sqrt{\sum_{i=1}^{N} (q_i - p_i)^2}$$

Image A

-1	2	0
3	-3	1
0	1	2

-

Image B

0	2	0
1	-2	2
1	1	3

$$\left(\begin{array}{ccc} -1 & 0 & 0 \\ 2 & -1 & -1 \\ -1 & 0 & -1 \end{array} \right)^2$$

1	0	0
4	1	1
1	0	1

Euclidean Distance: Example

$$d(p,q) = \sqrt{\sum_{i=1}^{N} (q_i - p_i)^2}$$

Image A

Image B

$$\begin{bmatrix} -1 & 2 & 0 \\ 3 & -3 & 1 \\ 0 & 1 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 2 & 0 \\ 1 & -2 & 2 \\ 1 & 1 & 3 \end{bmatrix} \;\rangle\; \left(\begin{bmatrix} -1 & 0 & 0 \\ 2 & -1 & -1 \\ -1 & 0 & -1 \end{bmatrix} \right)^2 \;\rangle\; \sum \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

1 + 0 + 0 + 4 + 1 + 1 + 1 + 0 + 1 **= 9**

Euclidean Distance: Example

$$d(p,q) = \sqrt{\sum_{i=1}^{N} (q_i - p_i)^2}$$

Image A

$$\begin{bmatrix} -1 & 2 & 0 \\ 3 & -3 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

-

Image B

$$\begin{bmatrix} 0 & 2 & 0 \\ 1 & -2 & 2 \\ 1 & 1 & 3 \end{bmatrix}$$

$$\left(\begin{bmatrix} -1 & 0 & 0 \\ 2 & -1 & -1 \\ -1 & 0 & -1 \end{bmatrix}\right)^2$$

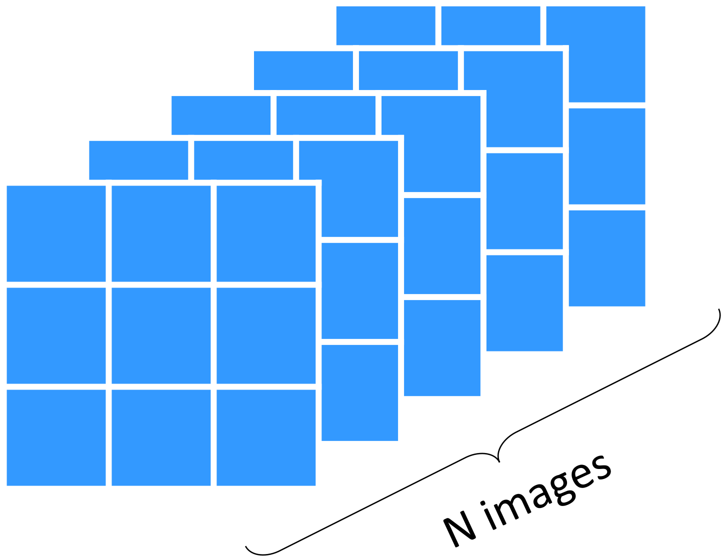
$$\sum \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

1 + 0 + 0 + 4 + 1 + 1 + 1 + 0 + 1 **= 9**

distance(A, B) = $\sqrt{9}$ = **3**

Images as Matrices

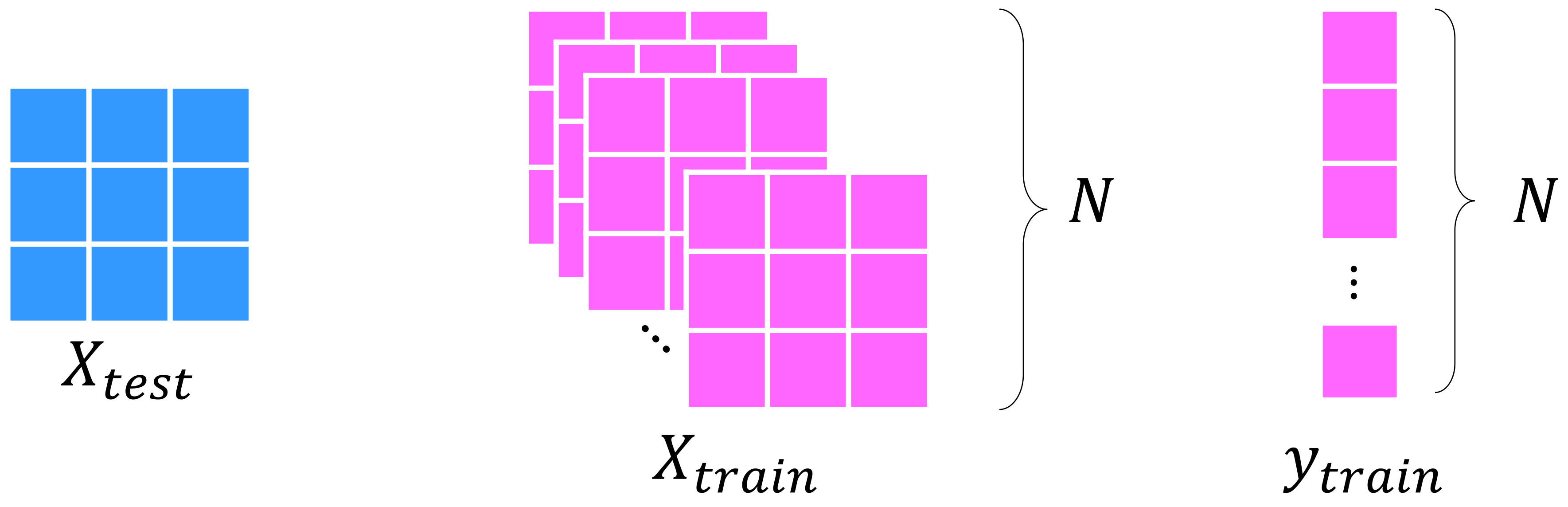
If we have many images, we will stack them up in a vector of matrices, or a 3D matrix of size NxWxH.



In Project 4, the data will be stored in matrices like these. This is a convenient representation, but we also like big matrices because computers are very good at dealing with them.

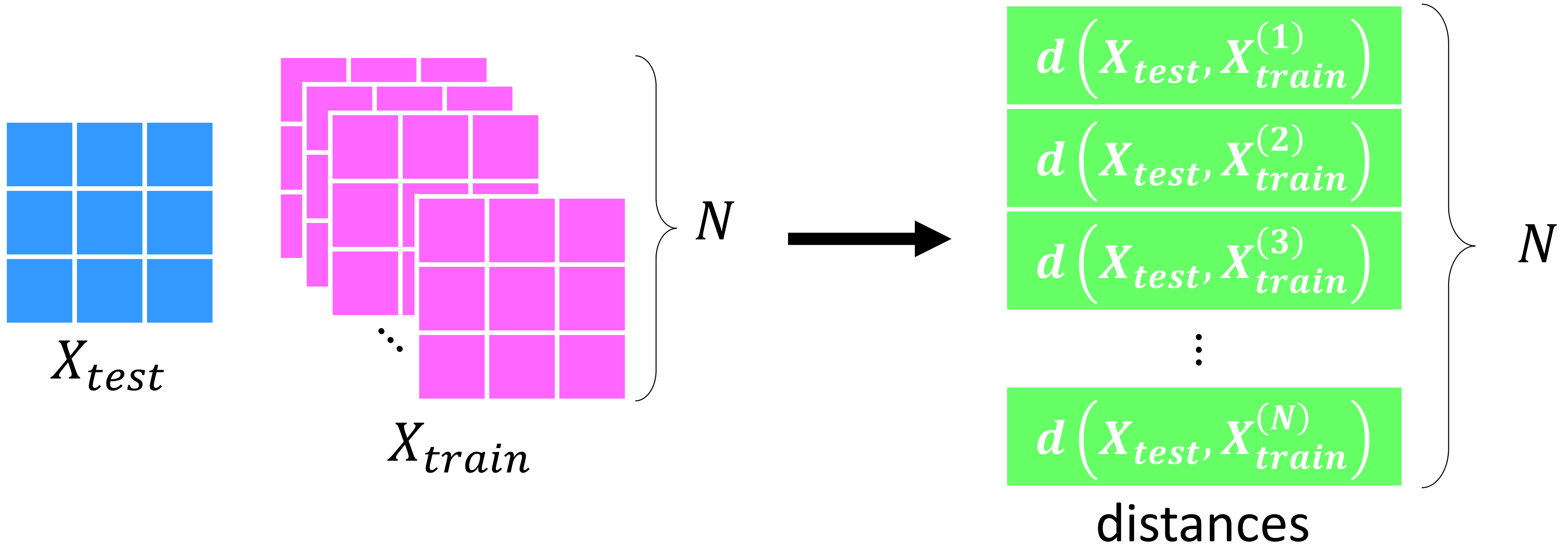
Nearest Neighbors

Back to the nearest neighbors algorithm. Say we have a matrix of N training images and a test image we want to classify.



Nearest Neighbors

Back to the nearest neighbors algorithm. Say we have a matrix of N training images and a test image we want to classify.



Nearest Neighbors

The smallest distance to the test image is given by:

```
minimum(distances)
```

Let's say the test image is closest to train image $X_{train}^{(7)}$. Then:

```
argmin(distances)=7
```

The function `argmin` gives the argument that minimizes `distances`. So, we can predict:

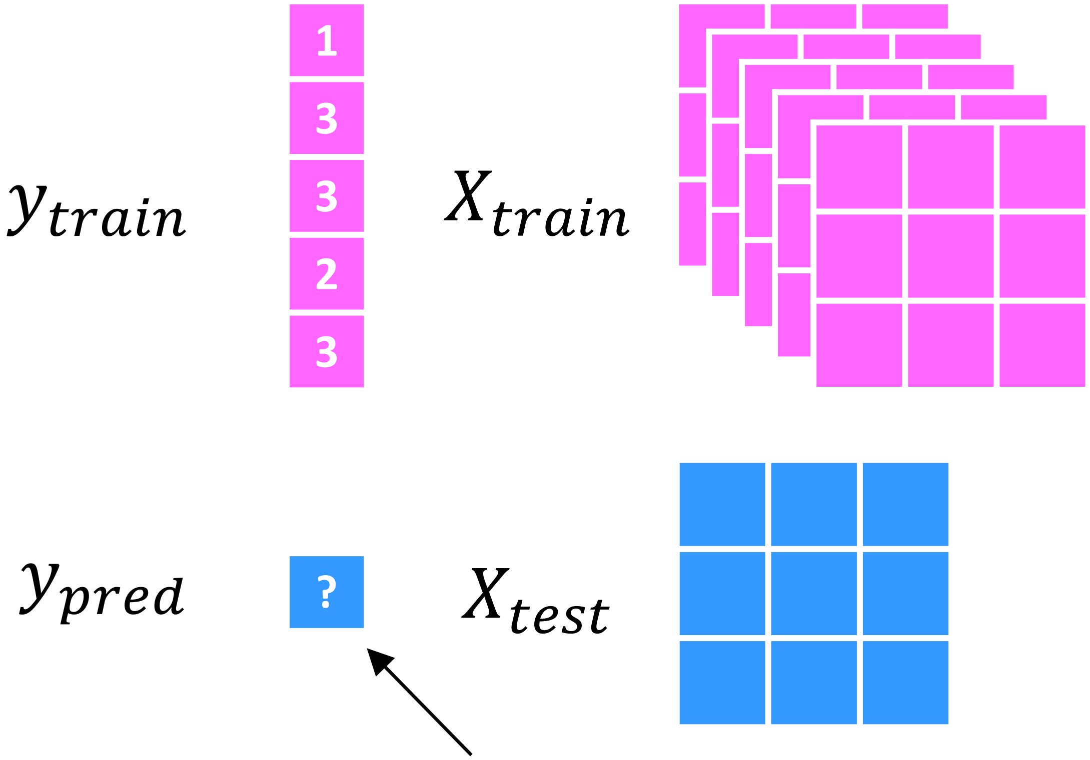
$$y_{pred} = y_{train}[7]$$

$$\left.\begin{array}{c} \boldsymbol{d\left(X_{test}, X_{train}^{(1)}\right)} \\ \boldsymbol{d\left(X_{test}, X_{train}^{(2)}\right)} \\ \boldsymbol{d\left(X_{test}, X_{train}^{(3)}\right)} \\ \vdots \\ \boldsymbol{d\left(X_{test}, X_{train}^{(N)}\right)} \end{array}\right\} N$$

distances

Nearest Neighbors

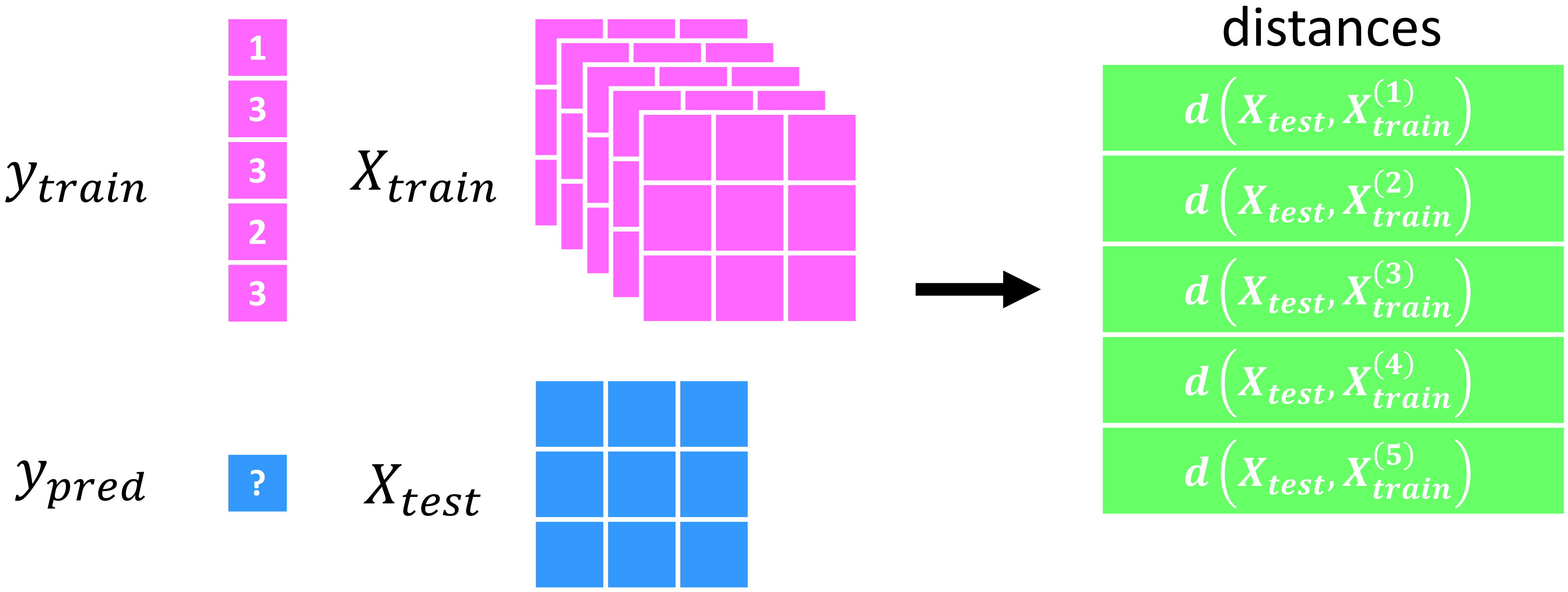
A small example:



Goal: Predict test image label, y_{pred}

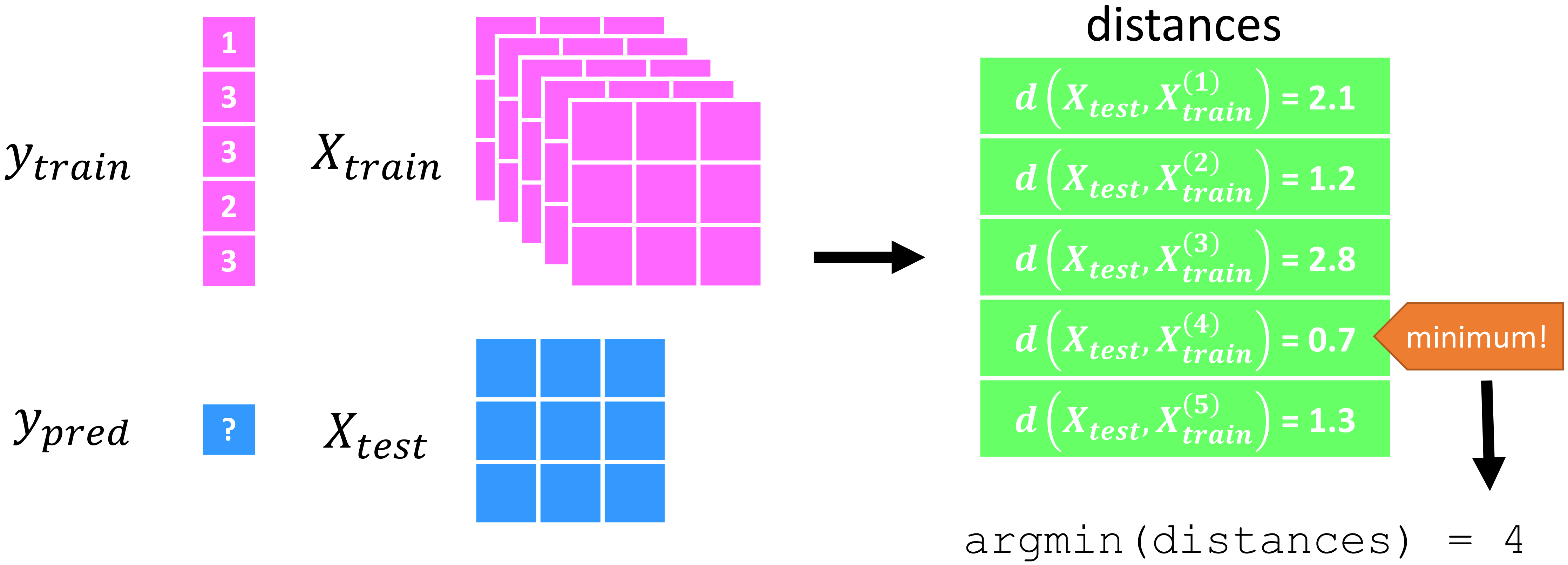
Nearest Neighbors

A small example:



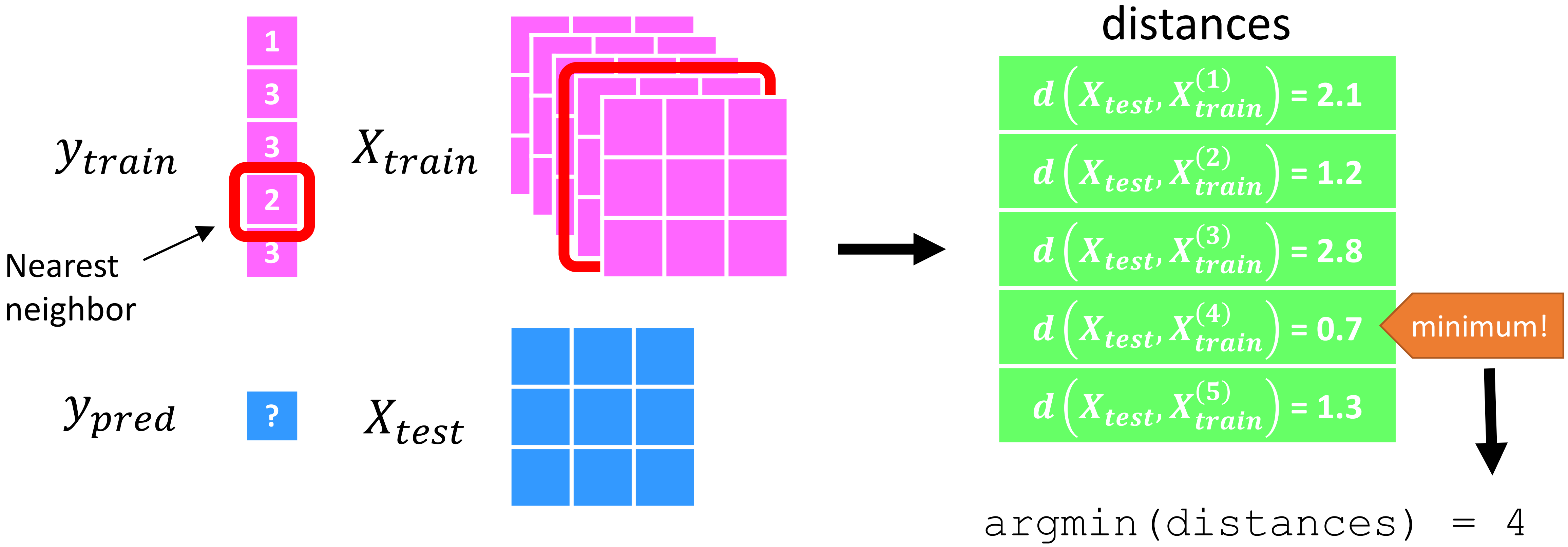
Nearest Neighbors

A small example:



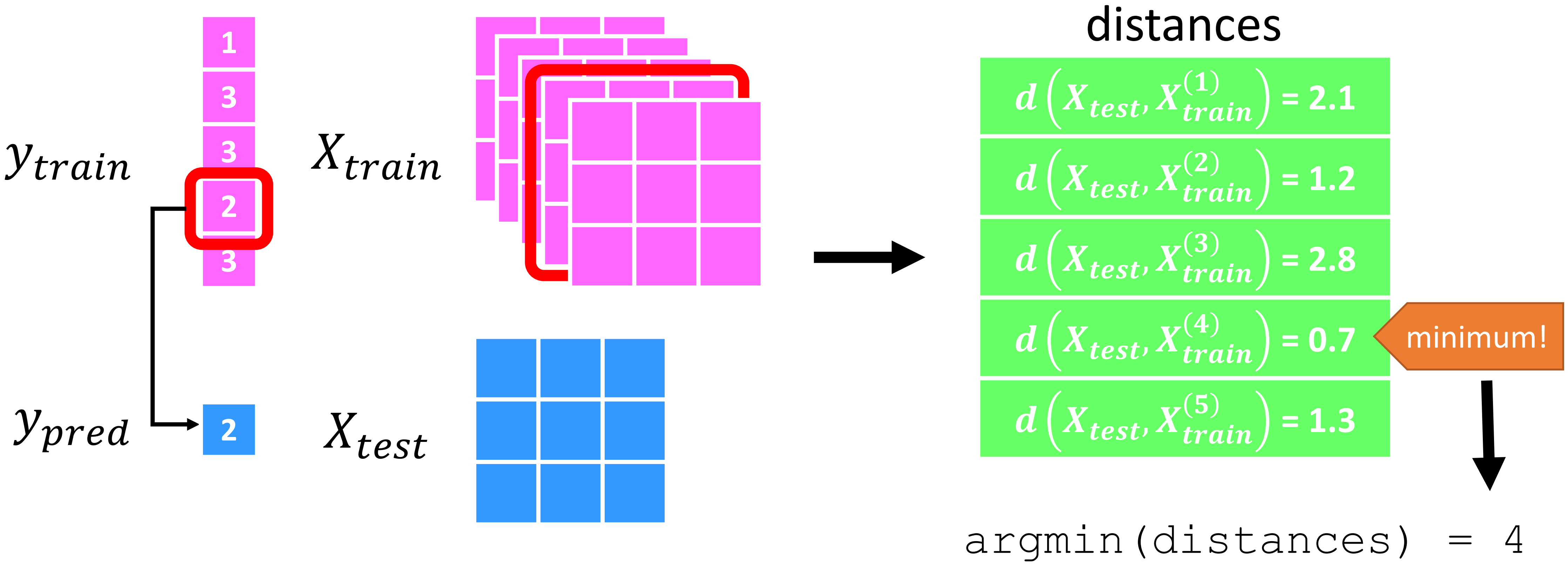
Nearest Neighbors

A small example:



Nearest Neighbors

A small example:



Nearest Neighbors: Algorithm

Training time:

Save the data, (X_{train}, y_{train}).

Testing time: Given N_{test} test images and N_{train} training images:

For each test image

```
for i = 1:N_test do:
    distances = [0,…,0]    (vector of length N_train)
    for j = 1:N_train do:
        distances[j] = distance(X_test[i], X_train[j])
    nearest_idx = argmin(distances)
    y_pred[i] = y_train[nearest_idx]
```

Nearest Neighbors: Algorithm

Training time:

Save the data, (X_{train}, y_{train}).

Testing time: Given N_{test} test images and N_{train} training images:

```
for i = 1:N_test do:
    distances = [0,…,0]   (vector of length N_train)
    for j = 1:N_train do:
        distances[j] = distance(X_test[i], X_train[j])
    nearest_idx = argmin(distances)
    y_pred[i] = y_train[nearest_idx]
```

Initialize distances to zero

Nearest Neighbors: Algorithm

Training time:

Save the data, (X_{train}, y_{train}).

Testing time: Given N_{test} test images and N_{train} training images:

for i = 1:N_{test} do:

distances = [0,…,0] (vector of length N_{train})

Calculate distance between current test image and each train image

for j = 1:N_{train} do:

distances[j] = distance(X_test[i], X_train[j])

nearest_idx = argmin(distances)

y_pred[i] = y_train[nearest_idx]

Nearest Neighbors: Algorithm

Training time:

Save the data, (X_{train}, y_{train}).

Testing time: Given N_{test} test images and N_{train} training images:

```
for i = 1:N_test do:
    distances = [0,…,0]    (vector of length N_train)
    for j = 1:N_train do:
        distances[j] = distance(X_test[i], X_train[j])
    nearest_idx = argmin(distances)
    y_pred[i] = y_train[nearest_idx]
```

Find the index of the nearest neighbor

Nearest Neighbors: Algorithm

Training time:

Save the data, (X_{train}, y_{train}).

Testing time: Given N_{test} test images and N_{train} training images:

```
for i = 1:N_test do:
    distances = [0,…,0]    (vector of length N_train)
    for j = 1:N_train do:
        distances[j] = distance(X_test[i], X_train[j])
    nearest_idx = argmin(distances)
    y_pred[i] = y_train[nearest_idx]
```

Assign the nearest neighbor's label to the current test image

Exercise!

0	1	0
2	-1	0
3	-1	1

Image A
label: 3

1	-4	2
0	1	0
-3	1	0

Image B
label: 1

-1	0	2
-2	3	0
3	-1	1

Image C
label: 2

Training data

1	0	4
-2	1	0
3	-1	3

Test Image

Exercise: Classify this image using nearest neighbors

Exercise: Solution

0	1	0
2	-1	0
3	-1	1

Image A
label: 3

1	-4	2
0	1	0
-3	1	0

Image B
label: 1

-1	0	2
-2	3	0
3	-1	1

Image C
label: 2

Training data

1	0	4
-2	1	0
3	-1	3

Test Image

distance(test, A) = $\sqrt{42} = 6.48$

distance(test, B) = $\sqrt{73} = 8.54$

distance(test, C) = $\sqrt{16} = 4$

Exercise: Solution

0	1	0
2	-1	0
3	-1	1

Image A
label: 3

1	-4	2
0	1	0
-3	1	0

Image B
label: 1

-1	0	2
-2	3	0
3	-1	1

Image C
label: 2

Training data

1	0	4
-2	1	0
3	-1	3

Test Image

distance(test, A) = $\sqrt{42} = 6.48$

distance(test, B) = $\sqrt{73} = 8.54$

distance(test, C) = $\boldsymbol{\sqrt{16} = 4}$

Nearest Neighbor: Image C

Predicted Label = 2

How do we evaluate our model?

Accuracy: Percentage of correct classifications made by the model.

$$\text{accuracy} = \frac{\#\ \text{correct predictions}}{N_{test}}$$

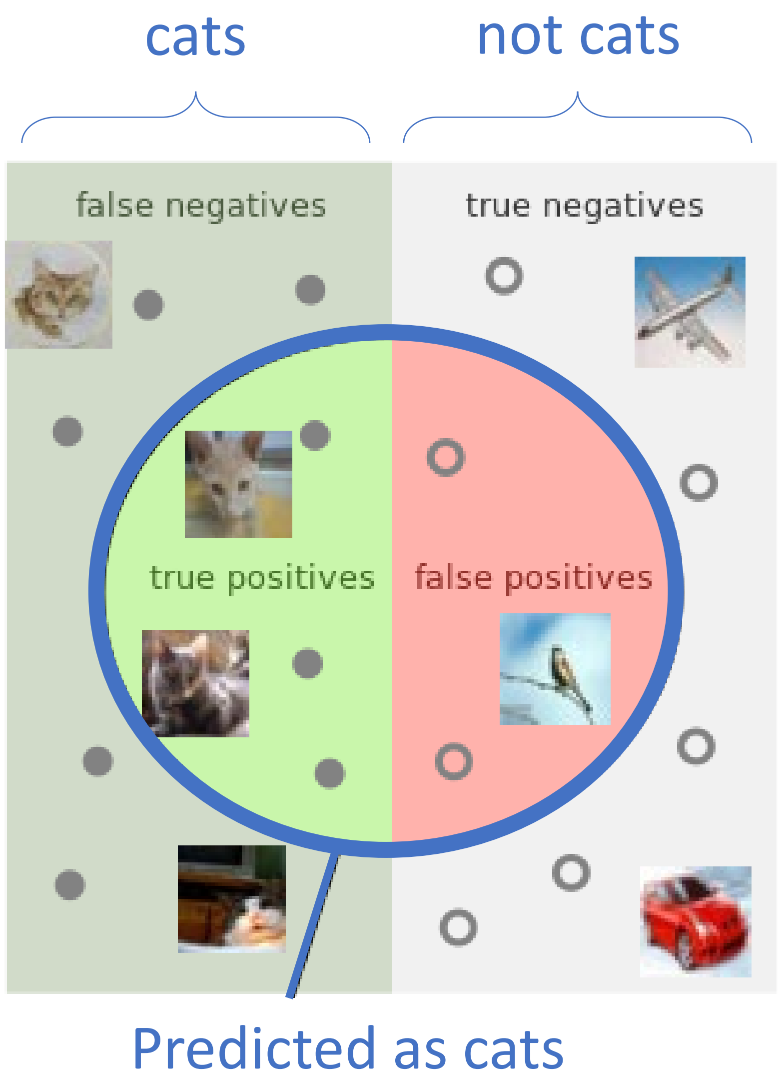
Total number of
data points tested

Quick, easy to interpret measure of how good the prediction is. Doesn't show why / how we're failing.

Evaluation: Types of Error

Say we have a binary classification problem, where a data point can be classified as one of two options.

Ex: Cat detector (1 = cat, 0 = not cat)



Evaluation: Types of Error

Say we have a binary classification problem, where a data point can be classified as one of two options.

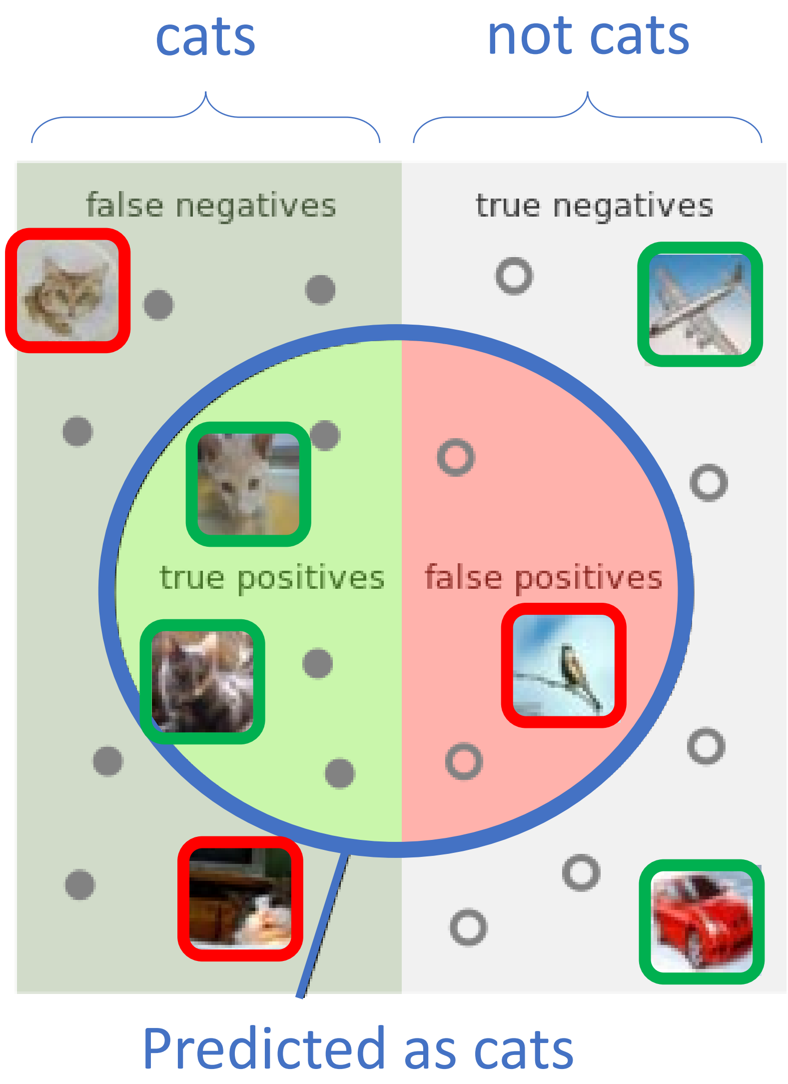
Ex: Cat detector (1 = cat, 0 = not cat)

True positive: Cat correctly classified as cat.

False positive: Non-cat incorrectly classified as cat.

True negative: Non-cat correctly classified as not cat.

False negative: Cat incorrectly classified as not cat.



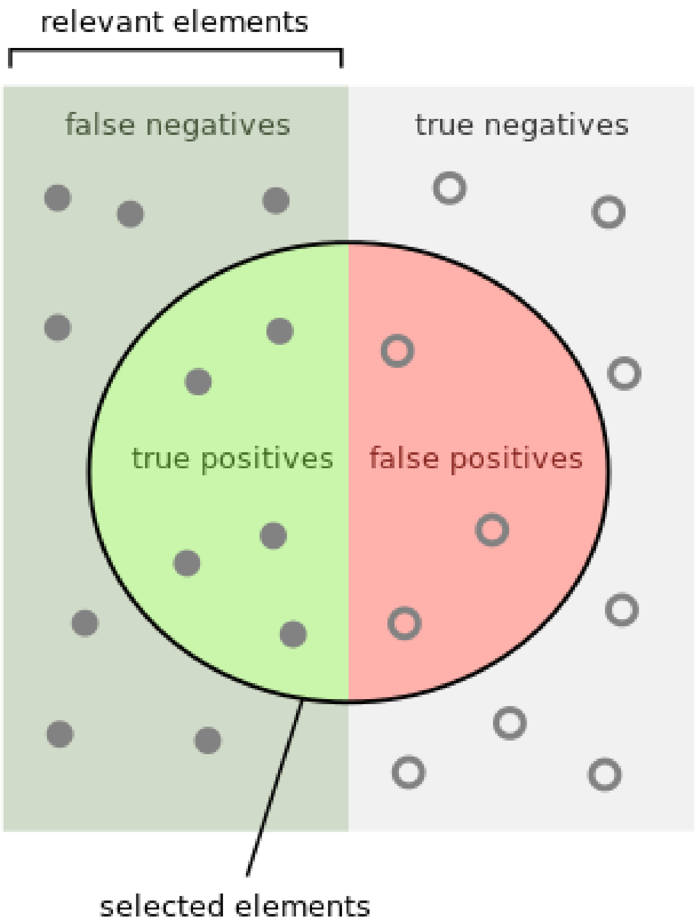
Evaluation: Precision & Recall

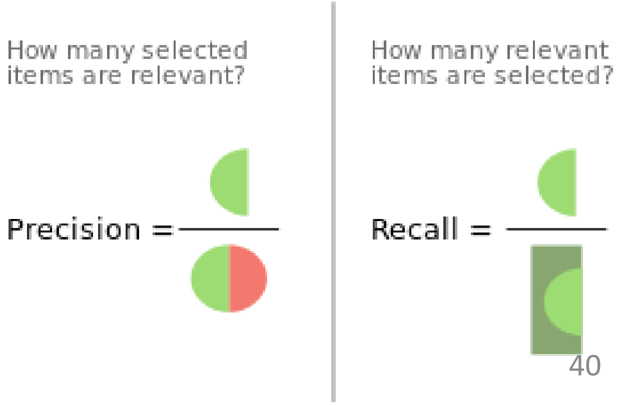
Precision: How valid the results are.

$$\text{precision} = \frac{\#\ \text{true positives}}{\#\ \text{true positives} + \#\ \text{false positives}}$$

Recall: How complete the results are.

$$\text{recall} = \frac{\#\ \text{true positives}}{\#\ \text{true positives} + \#\ \text{false negatives}}$$





Evaluation: Precision & Recall

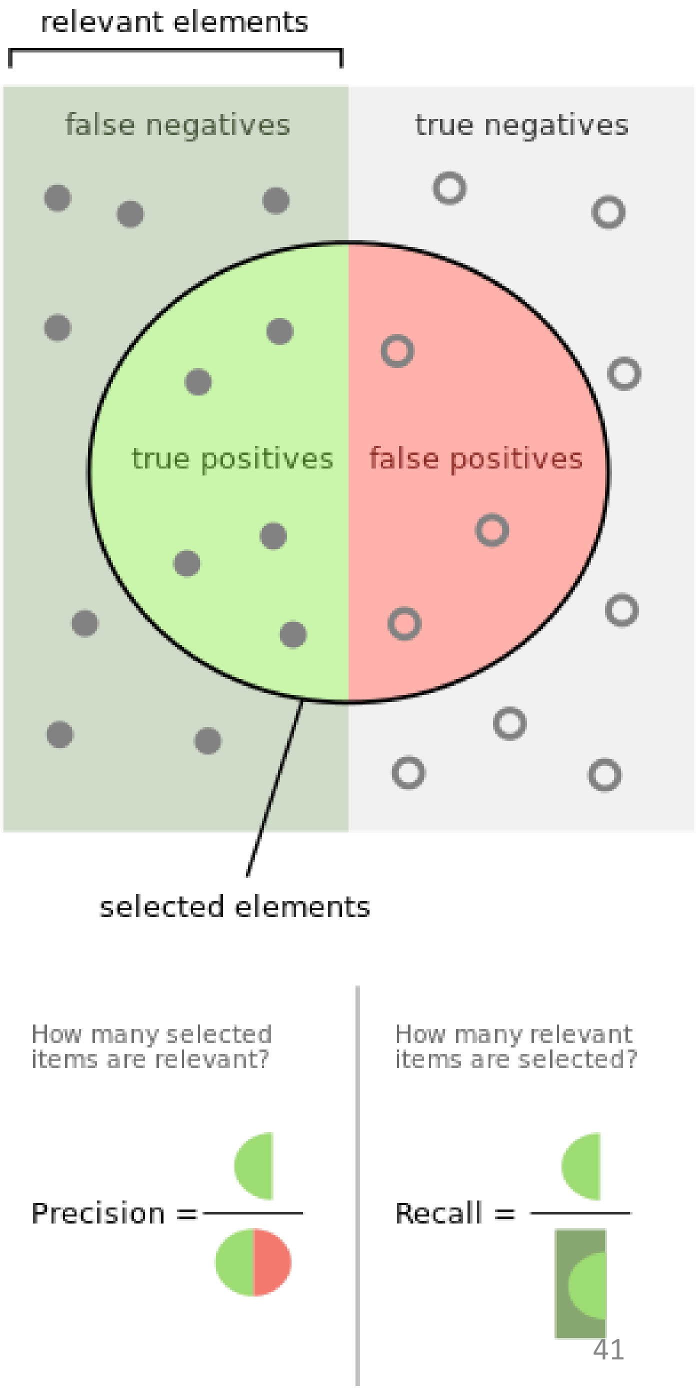
Precision: Helpful when it's important to have few false positives.

- Ex: A search engine should not show any irrelevant results, but it's okay to miss some relevant ones.

Recall: Helpful when it's important to have few false negatives.

- Ex: If a cancer detection algorithm gives a false negative, that's VERY bad! If there are some false positives, that's not so bad.

The metric chosen depends on the application!



Evaluation: Confusion Matrix

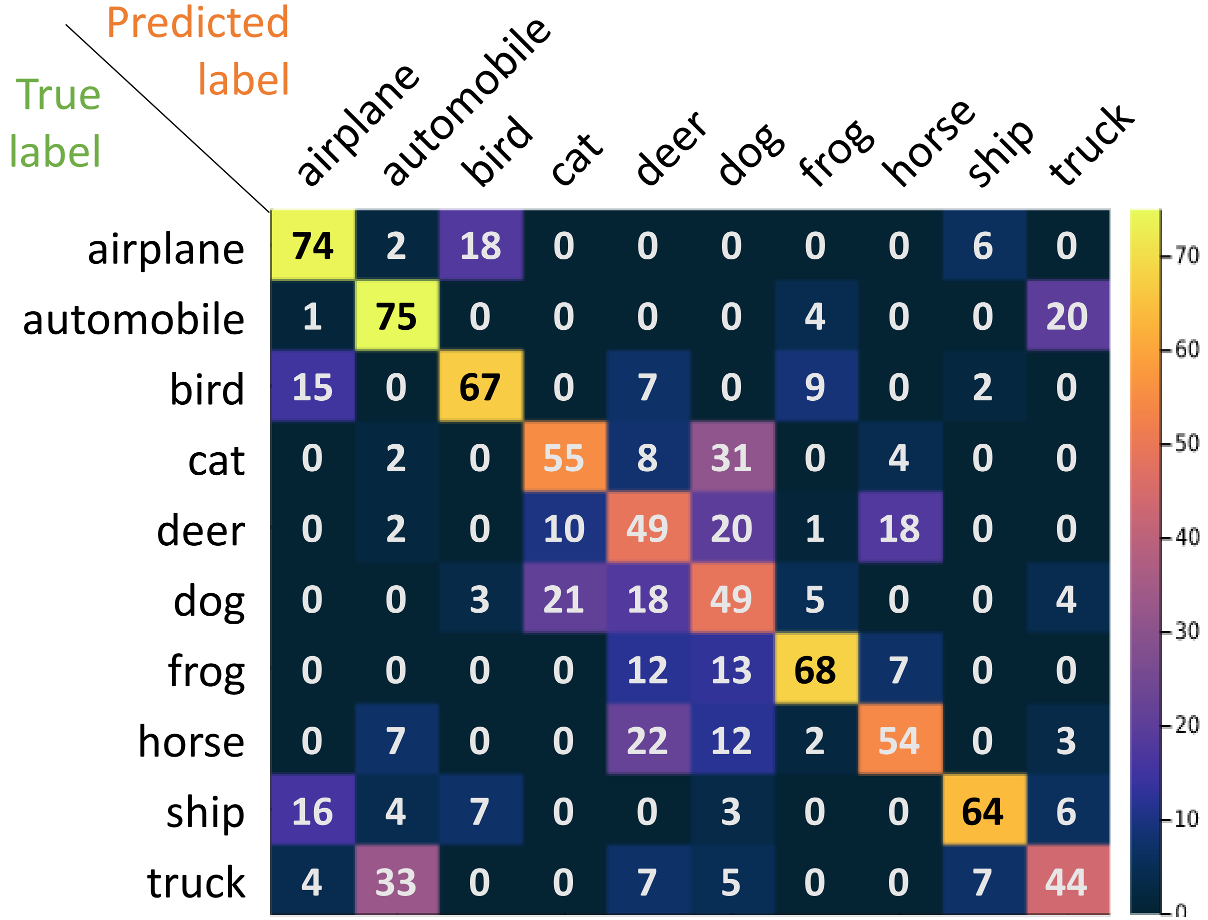
A heatmap of classification labels vs true labels.

For a perfect classifier, the confusion matrix looks like this.

True label \ Predicted label	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	100	0	0	0	0	0	0	0	0	0
automobile	0	100	0	0	0	0	0	0	0	0
bird	0	0	100	0	0	0	0	0	0	0
cat	0	0	0	100	0	0	0	0	0	0
deer	0	0	0	0	100	0	0	0	0	0
dog	0	0	0	0	0	100	0	0	0	0
frog	0	0	0	0	0	0	100	0	0	0
horse	0	0	0	0	0	0	0	100	0	0
ship	0	0	0	0	0	0	0	0	100	0
truck	0	0	0	0	0	0	0	0	0	100

Evaluation: Confusion Matrix

A heatmap of classification labels vs true labels.



Evaluation: Confusion Matrix

A heatmap of classification labels vs true labels.

airplanes classified as airplanes

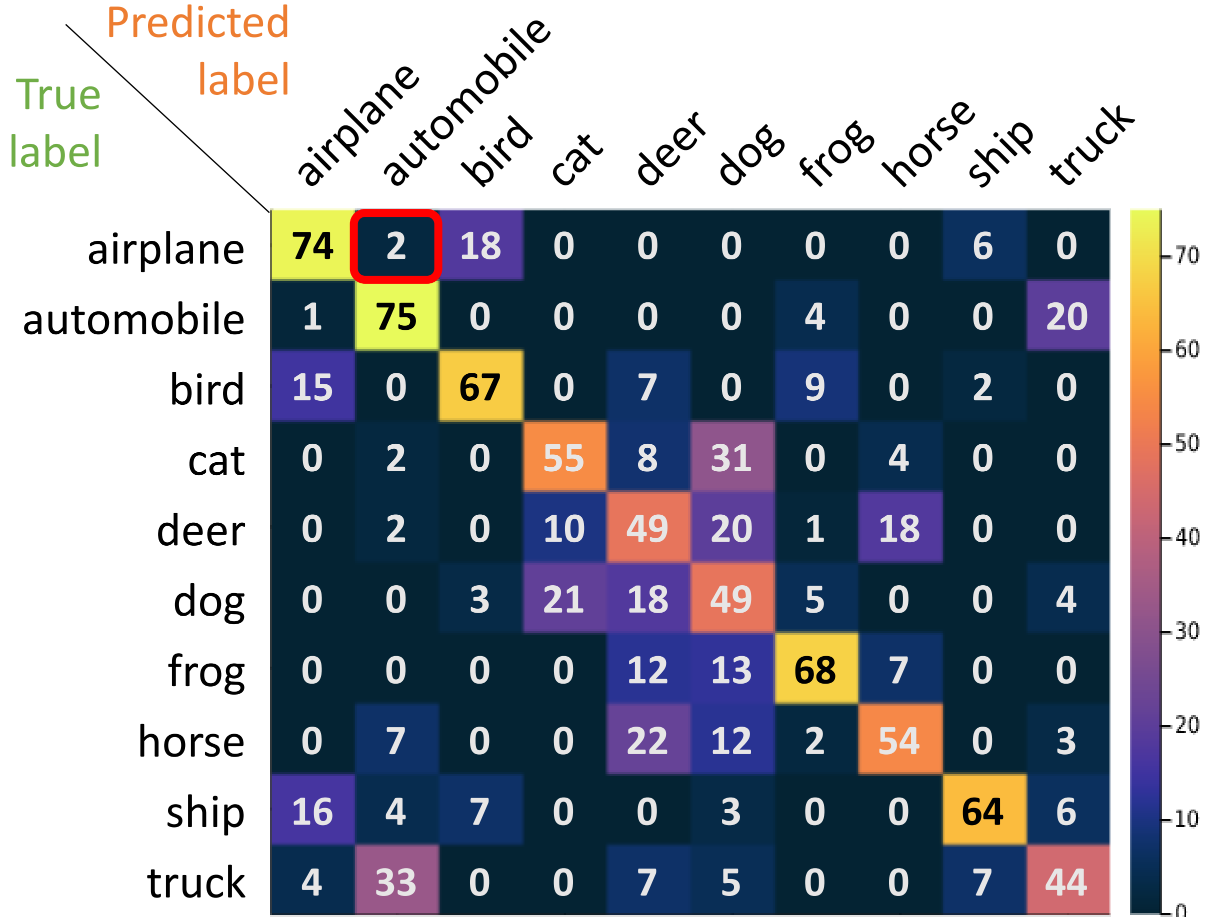
True label \ Predicted label	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	74	2	18	0	0	0	0	0	6	0
automobile	1	75	0	0	0	0	4	0	0	20
bird	15	0	67	0	7	0	9	0	2	0
cat	0	2	0	55	8	31	0	4	0	0
deer	0	2	0	10	49	20	1	18	0	0
dog	0	0	3	21	18	49	5	0	0	4
frog	0	0	0	0	12	13	68	7	0	0
horse	0	7	0	0	22	12	2	54	0	3
ship	16	4	7	0	0	3	0	0	64	6
truck	4	33	0	0	7	5	0	0	7	44

Evaluation: Confusion Matrix

A heatmap of classification labels vs true labels.

airplanes classified as airplanes

airplanes classified as automobiles



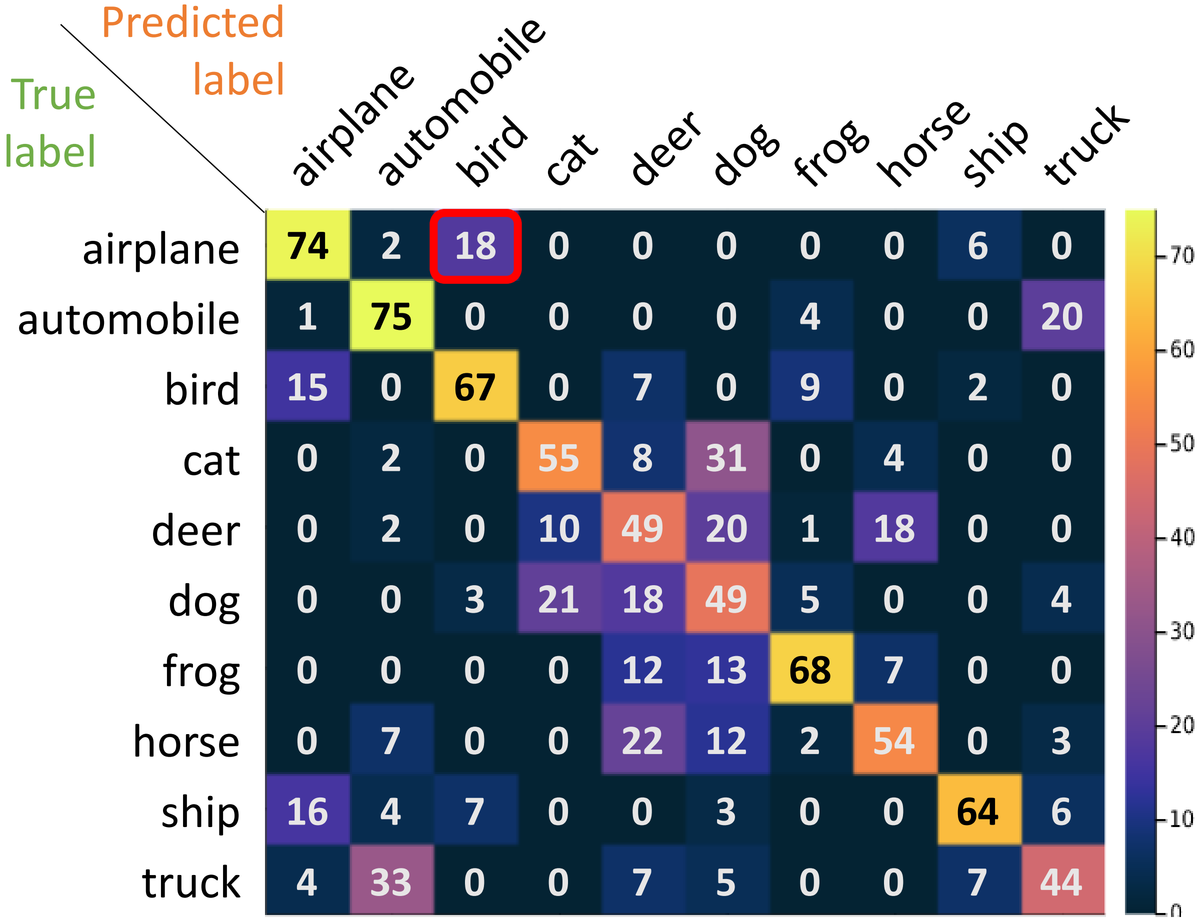
Evaluation: Confusion Matrix

A heatmap of classification labels vs true labels.

airplanes classified as airplanes

airplanes classified as automobiles

airplanes classified as birds



Evaluation: Confusion Matrix

A heatmap of classification labels vs true labels.

airplanes classified as airplanes

airplanes classified as automobiles

airplanes classified as birds

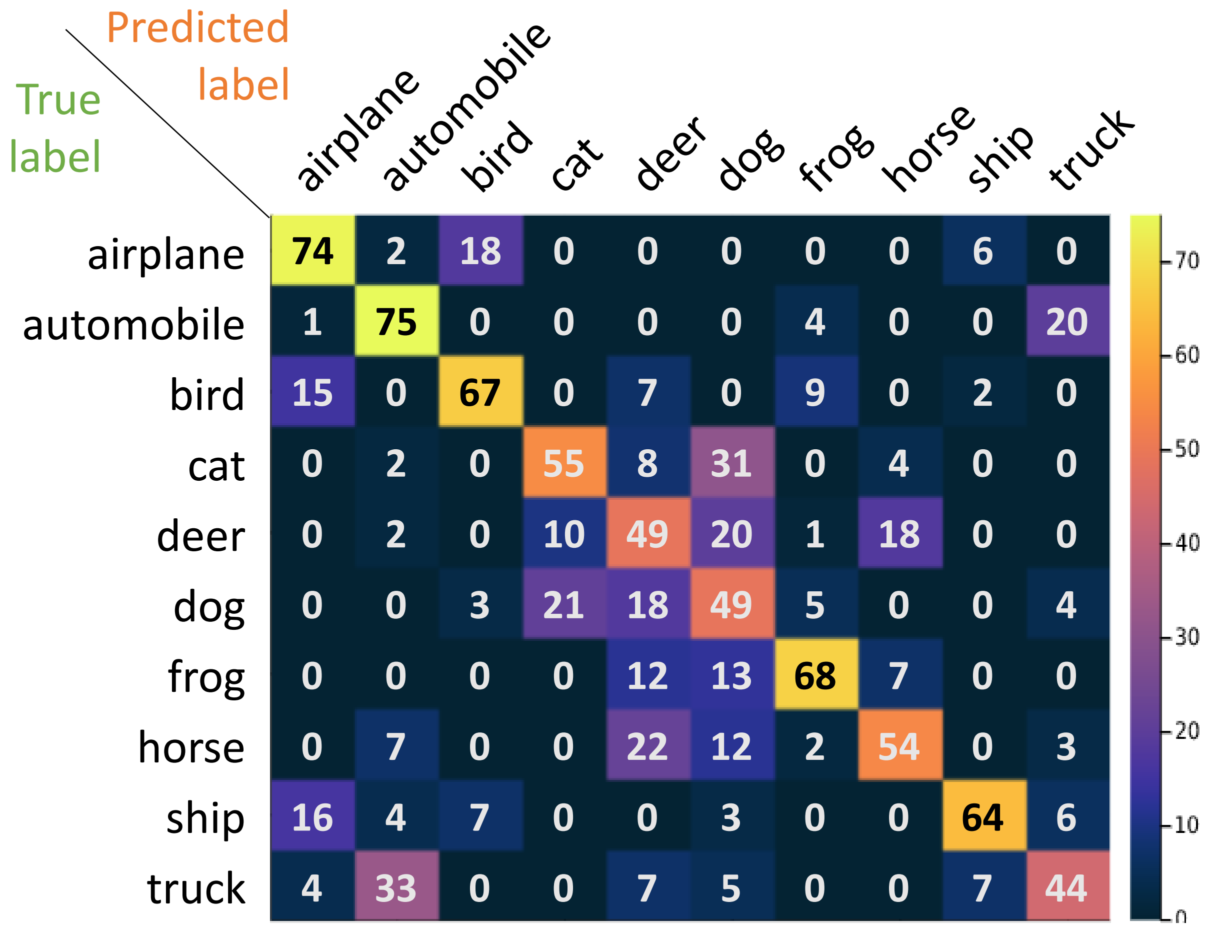
etc…

True label \ Predicted label	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	74	2	18	0	0	0	0	0	6	0
automobile	1	75	0	0	0	0	4	0	0	20
bird	15	0	67	0	7	0	9	0	2	0
cat	0	2	0	55	8	31	0	4	0	0
deer	0	2	0	10	49	20	1	18	0	0
dog	0	0	3	21	18	49	5	0	0	4
frog	0	0	0	0	12	13	68	7	0	0
horse	0	7	0	0	22	12	2	54	0	3
ship	16	4	7	0	0	3	0	0	64	6
truck	4	33	0	0	7	5	0	0	7	44

Evaluation: Confusion Matrix

A heatmap of classification labels vs true labels.

The confusion matrix gives us more insight about where the algorithm is failing.



Evaluation: Confusion Matrix

A heatmap of classification labels vs true labels.

The confusion matrix gives us more insight about where the algorithm is failing.

Airplanes are being misclassified as birds.

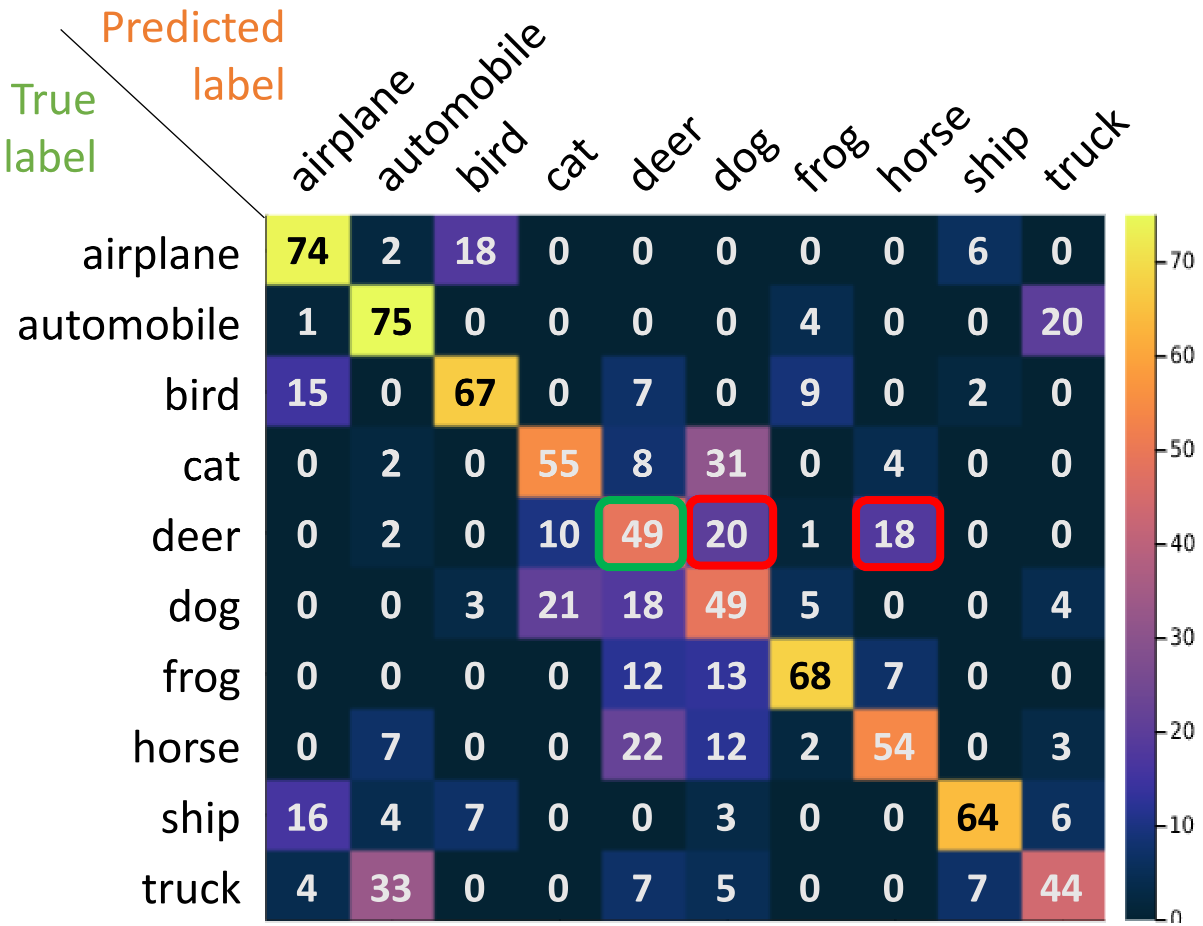
True label \ Predicted label	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	74	2	18	0	0	0	0	0	6	0
automobile	1	75	0	0	0	0	4	0	0	20
bird	15	0	67	0	7	0	9	0	2	0
cat	0	2	0	55	8	31	0	4	0	0
deer	0	2	0	10	49	20	1	18	0	0
dog	0	0	3	21	18	49	5	0	0	4
frog	0	0	0	0	12	13	68	7	0	0
horse	0	7	0	0	22	12	2	54	0	3
ship	16	4	7	0	0	3	0	0	64	6
truck	4	33	0	0	7	5	0	0	7	44

Evaluation: Confusion Matrix

A heatmap of classification labels vs true labels.

The confusion matrix gives us more insight about where the algorithm is failing.

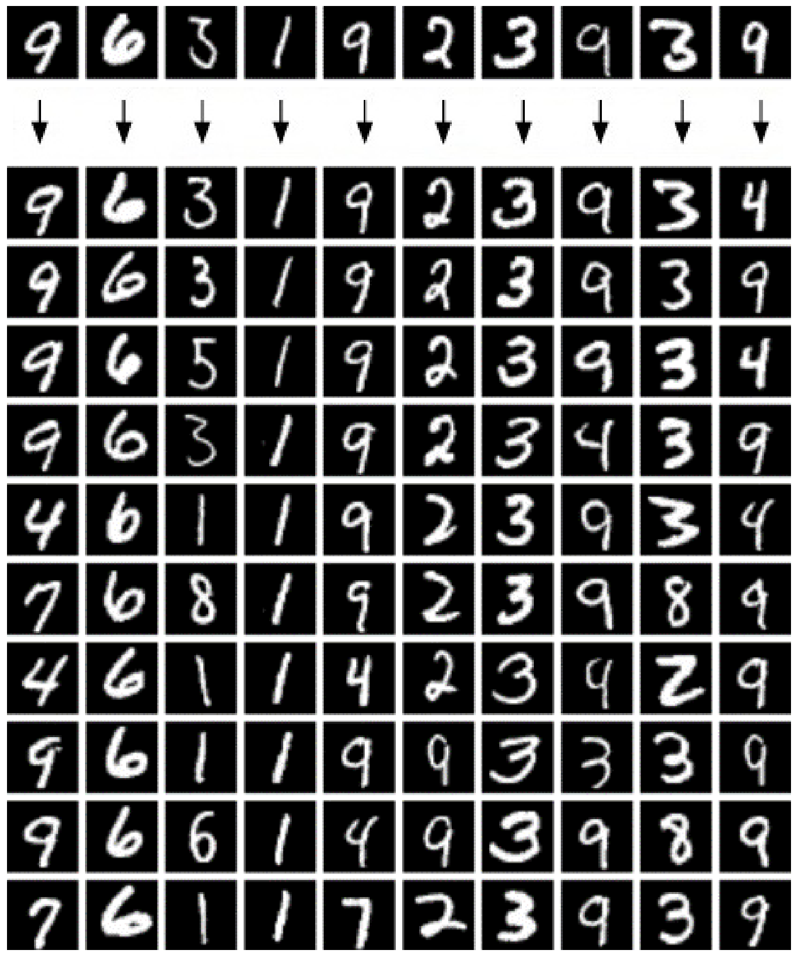
Deer are hard to classify. They are being labelled as dogs and horses.



k-Nearest Neighbors

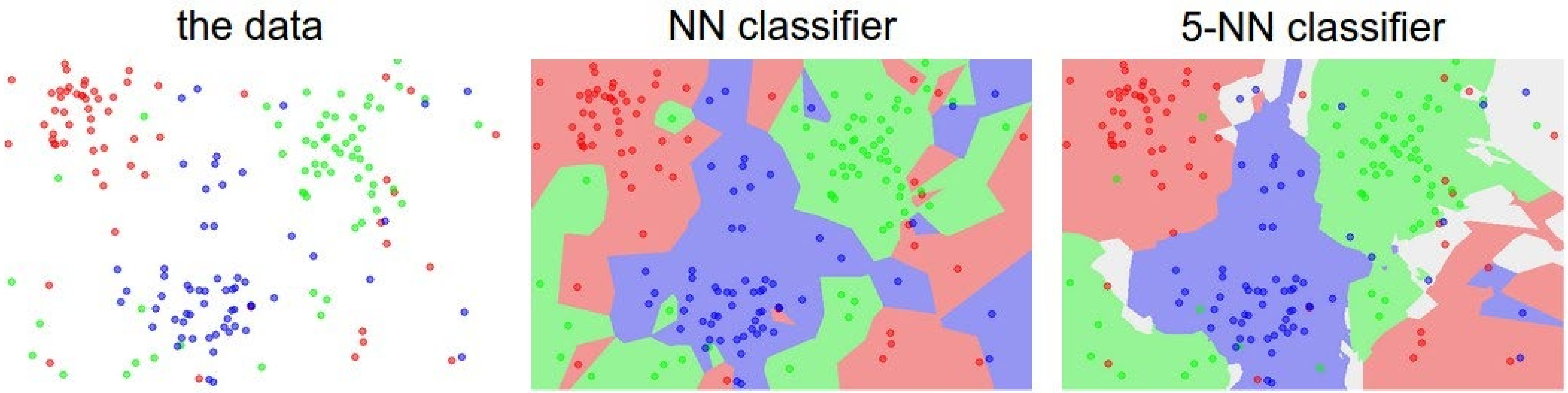
To make our algorithm more robust, we can let the k nearest neighbors vote on the label for the test image. This is the "k" in k-nearest neighbors (kNN).

Up until now, we were describing 1-NN.



k-Nearest Neighbors

The decision boundaries for our data change. We are <u>overfitting</u> less.



Experiment with kNN here: http://vision.stanford.edu/teaching/cs231n-demos/knn/

Hyperparameters

How do we pick k?

k is an example of a **hyperparameter**: a parameter we choose, which isn't learned.

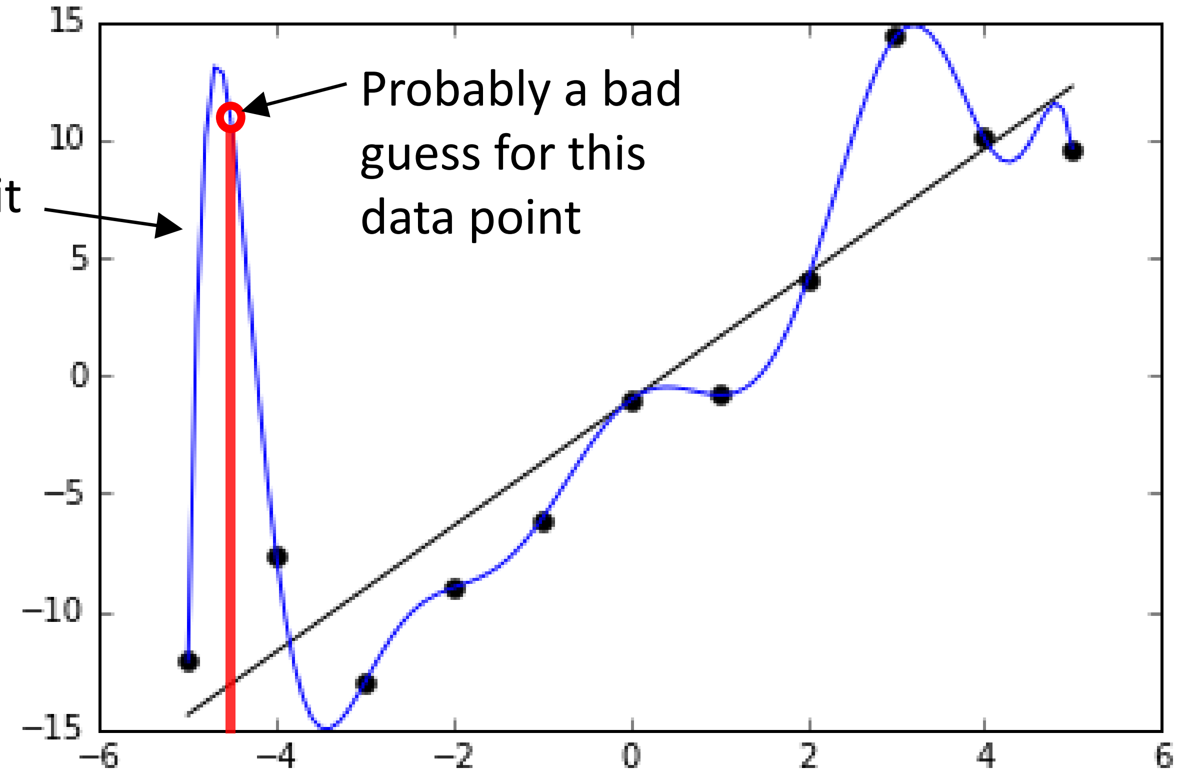
Generally, we need to tune these parameters by trying different values and selecting the best performing ones.

Overfitting

Overfitting happens when we fit a model that corresponds too closely to our data. Overfitting impacts performance on new data.

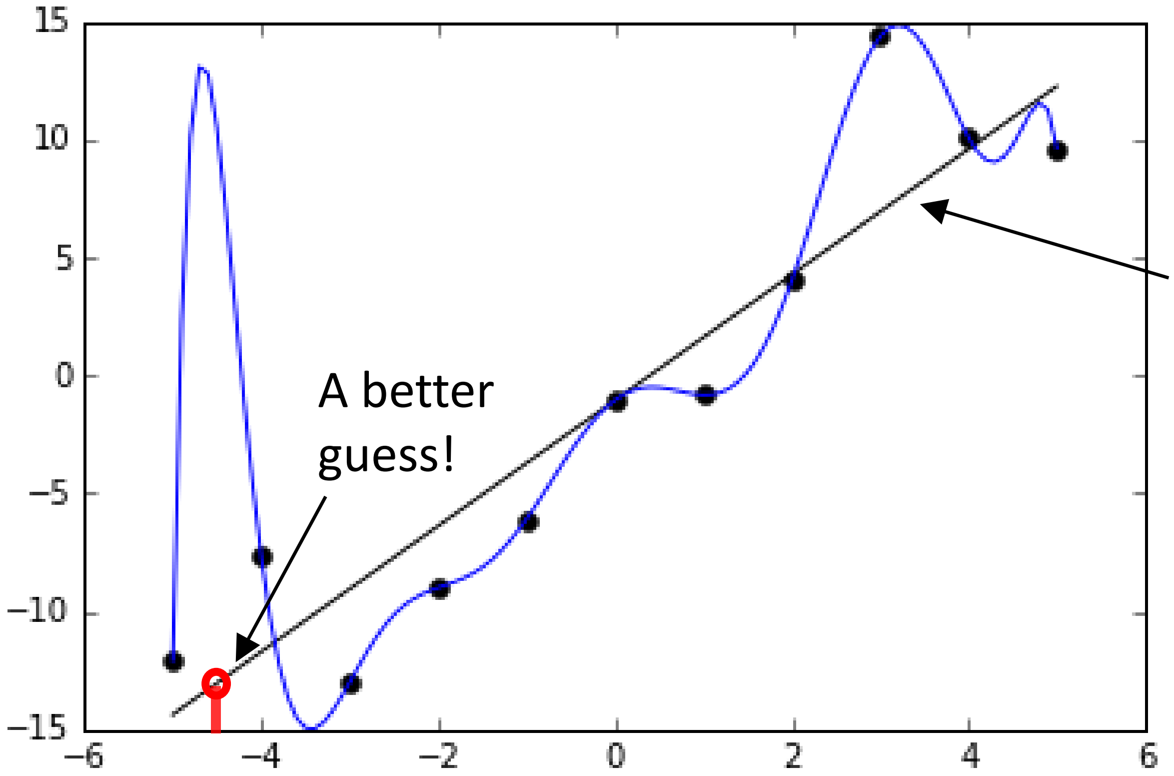
The blue line is a perfect fit for the given data.

But it's not a very good choice of model.



Overfitting

Overfitting happens when we fit a model that corresponds too closely to our data. Overfitting impacts performance on new data.



This linear fit is likely a better choice.

Avoids overfitting.

Image: CC 4.0 (link)

Overfitting

Overfitting happens when we fit a model that corresponds too closely to our data. Overfitting impacts performance on new data.

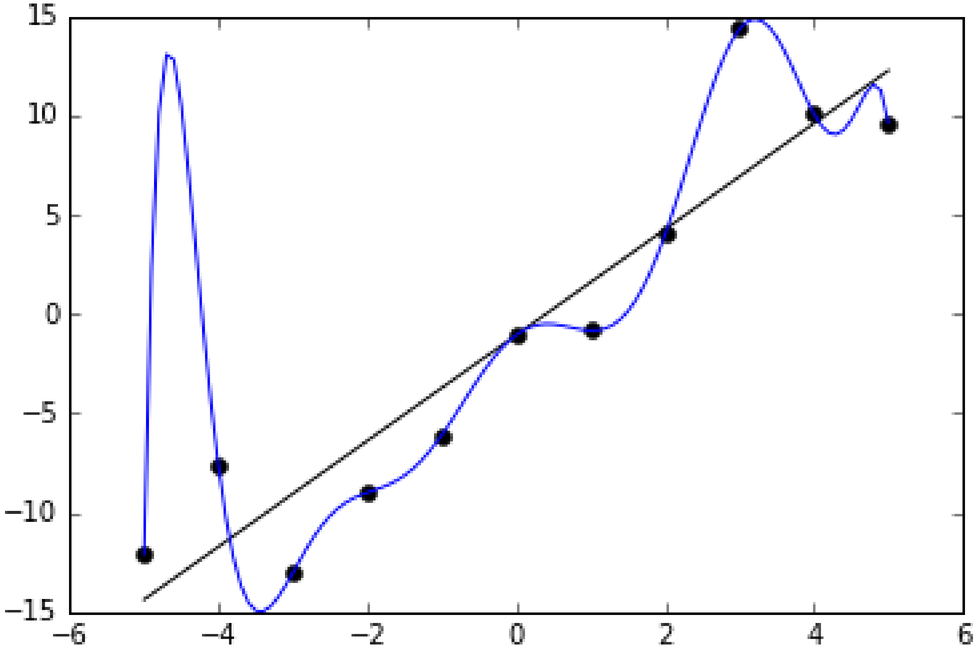


Image: CC 4.0 (link)

A classification example

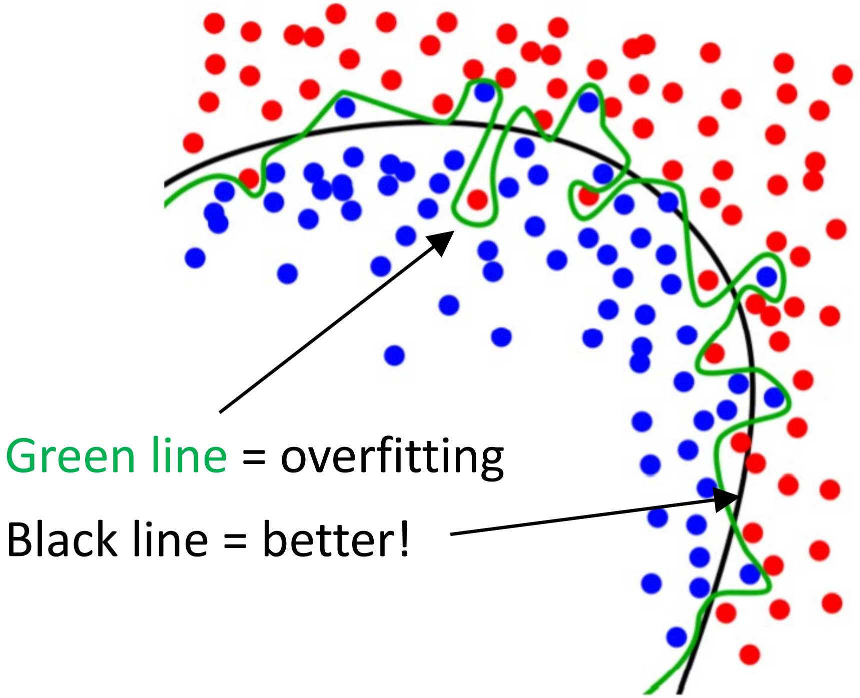


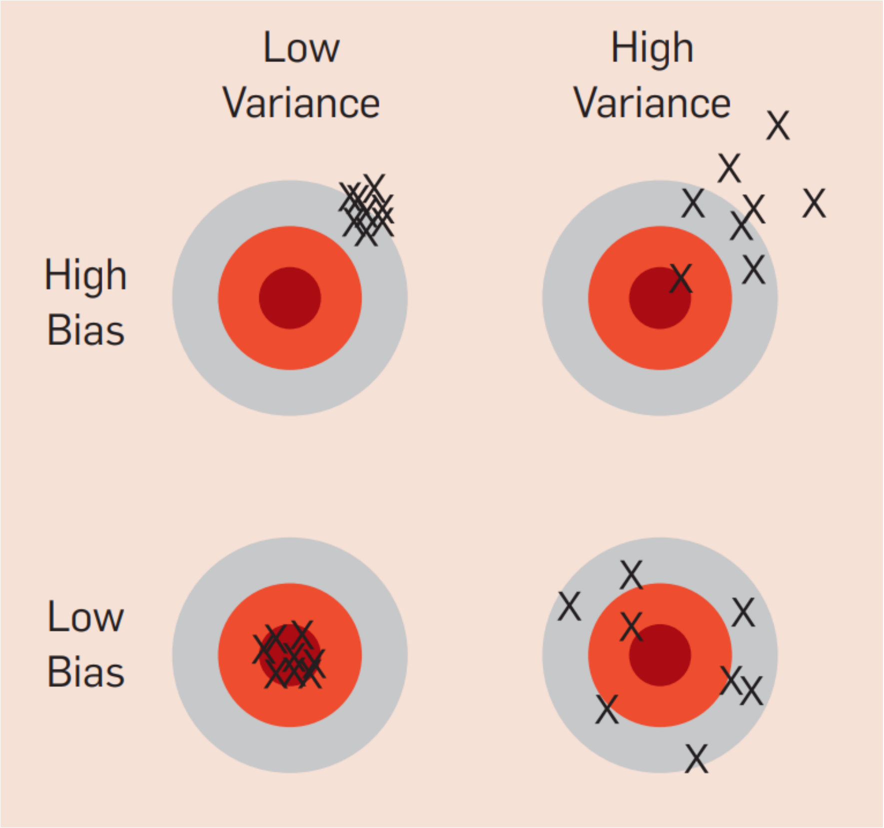
Image: CC 4.0 (link)

Bias-Variance Tradeoff

Bias is error due to deviation from the true value (underfitting).

Variance is error due to sensitivity to variations in the data (overfitting).

When choosing hyperparameters, we need to tradeoff between both.



Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

Bad idea ☹

We are minimizing the **training error**. This is basically just "memorizing" the training data (overfitting!).

The training error should be low. Ex: For nearest neighbor, k=1 will give zero training error. Training error should only be used as a sanity check.

Credit: Justin Johnson, EECS 498 F20 (link)

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

Bad idea ☹

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

Better idea ☺

train	test

We are minimizing the **testing error**.

This is better, but we still don’t know how we’ll do on new data.

Credit: Justin Johnson, EECS 498 F20 (link)

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

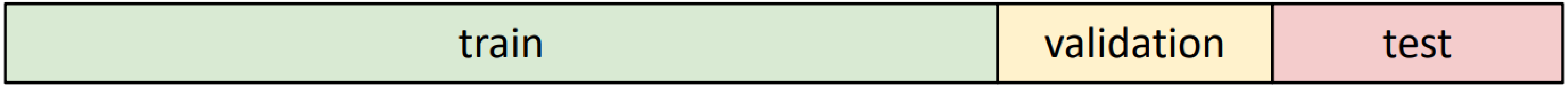
Bad idea ☹

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

Better idea ☺

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Good idea ☺



Credit: Justin Johnson, EECS 498 F20 (link)

Setting Hyperparameters

Your Dataset

Idea #4: **Cross-Validation**: Split data into **folds**, try each fold as validation and average the results

Best idea! ☺

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

Useful for small datasets, but (unfortunately) not used too frequently in deep learning

Credit: Justin Johnson, EECS 498 F20 (link)

Project 4: Machine Learning

Implement three machine learning algorithms to classify images from the MNIST dataset.

1. **Nearest neighbors** (Today!)
   - ✓ How to find the distance between images
   - ✓ The nearest neighbors algorithm
   - ✓ Evaluating classification algorithms
   - ✓ Setting hyperparameters
2. Linear Classifier ⟵ Next time!
3. Neural Network