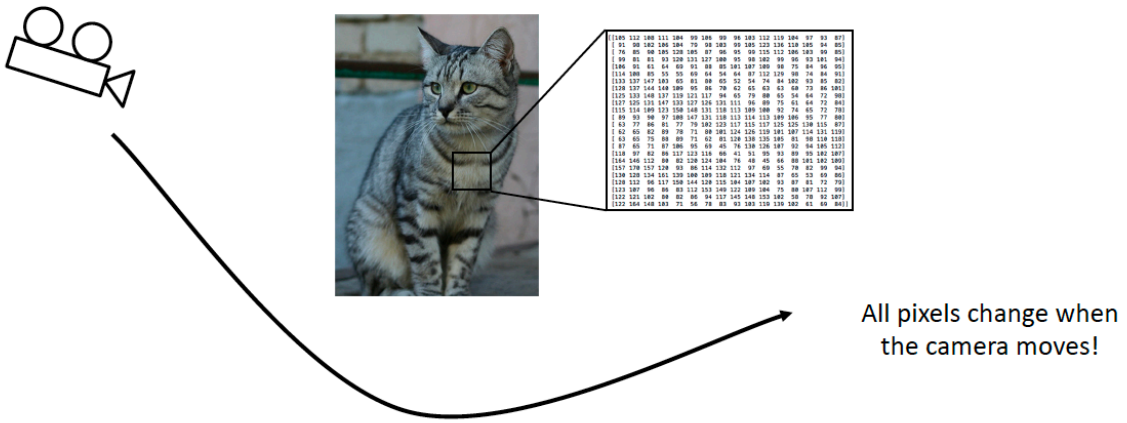
Lecture 2 - Image Classification

--> A core computer vision task

Input: image --> **Output:** Assign image to one of a fixed set of categories

1. Challenges (due to semantic gap)

- **Viewpoint Variation**



- **Intraclass Variation**

- **Fine-Grained Categories**

- **Background Clutter**
- **Illumination Changes**
- **Deformation**
- **Occusion** 遮挡

2. Image Classifier

- Data-Driven Approach
  - S1. Collect a dataset of images and labels
  - S2. Use Machine Learning to train a classifier
  - S3. Evaluate the classifier on new images
- Classifier

```
def train(images, labels):
  # Machine learning!
  return model
```

→ Memorize all data and labels

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

→ Predict the label of the most similar training image

  - **Nearest Neighbor**
    - Distance metric to compare images (p: each pixel)

L1 distance: $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$

test image					training image					pixel-wise absolute value differences				
56	32	10	18	-	10	20	24	17	=	46	12	14	1	add → 456
90	23	128	133		8	10	89	100		82	13	39	33	
24	26	178	200		12	16	178	170		12	10	0	30	
2	0	255	220		4	32	233	112		2	32	22	108	

Nearest Neighbor Classifier

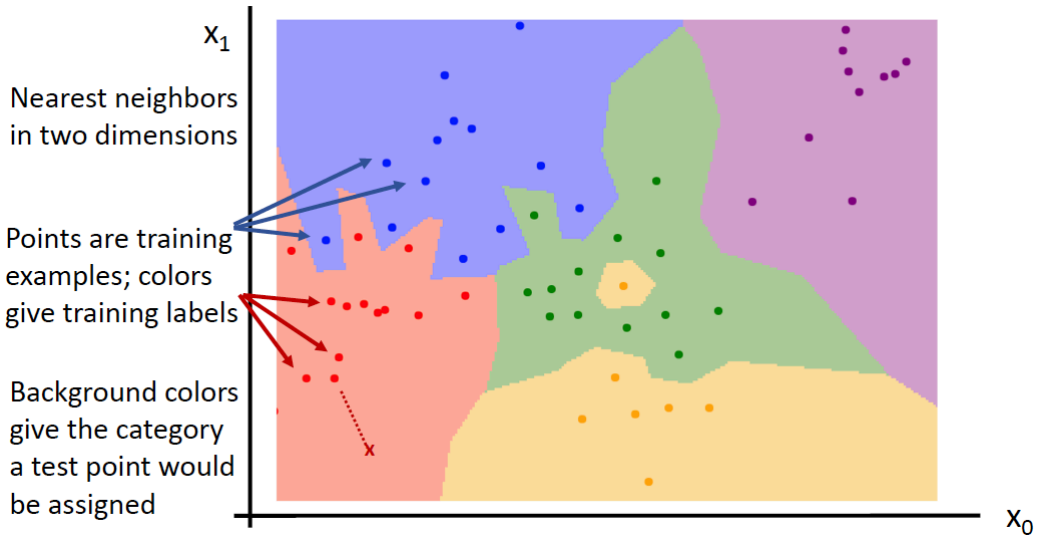
```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

For each test image:
Find nearest training image
Return label of nearest image

- Cons:
  - With N examples, training is O(1) while testing is O(N)

    --> bad: we can afford slow training, but need fast testing
- **Decision Boundary**

  --> the boundary between two classification regions

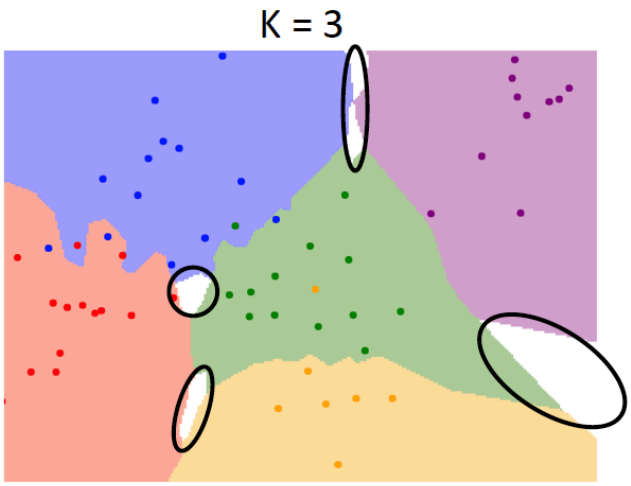


- Decision boundary can be noisy --> affected by outliers (yellow point in the graph)

- **K-Nearest Neighbors**

  --> use more neighbors and average to **smooth out** desicion boundaries!

  --> Instead of copying label from nearest neighbor, take majority vote from K closest points



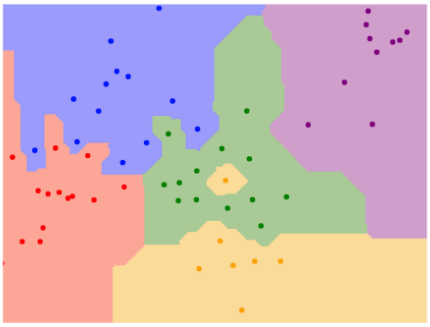
- Distance Metric

L1 (Manhattan) distance

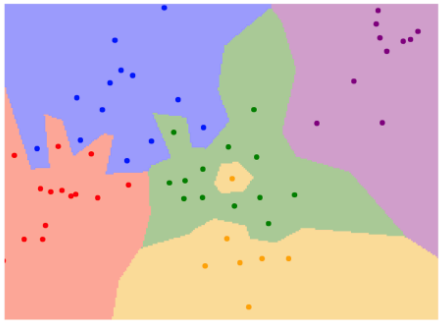
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

$$d_1(I_1, I_2) = \left(\sum_p (I_1^p - I_2^p)^2\right)^{\frac{1}{2}}$$



K = 1



- L1: all decision boundaries are 45°/90°/0°

  L2: decision boundary can be any angle

- With the right choice of distance metric, we can apply K-Nearest Neighbor to any type of data (image, similar articles…)

- KNN on raw pixels is seldom used
  - Very slow at test time
  - Distance metrics on pixels are not informative (not enough meaningful)

(all 3 images have same L2 distance to the one on the left)

3. Hyperparameters

Hyperparameters: choices about our learning algorithm that we **don't learn** from the training data; instead we **set them at the start of the learning process**

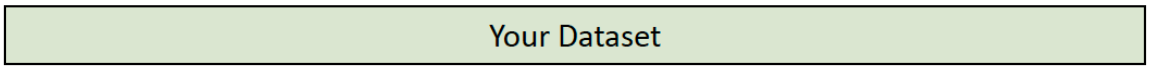
Eg. distance metric, k…

--> problem-dependent: in general, need to try them all and see what works best for the data / task

- Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data



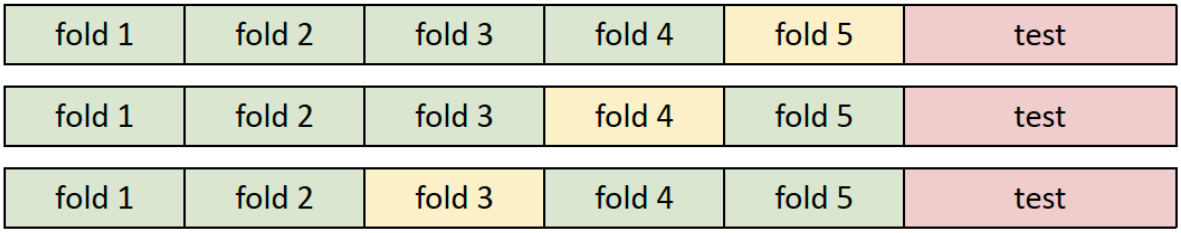
Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

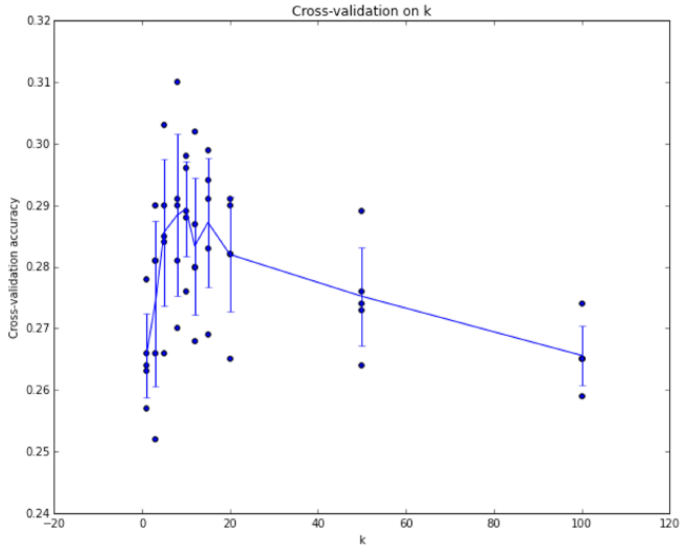
Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

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



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



Example of 5-fold cross-validation for the value of **k.**

Each point: single outcome.

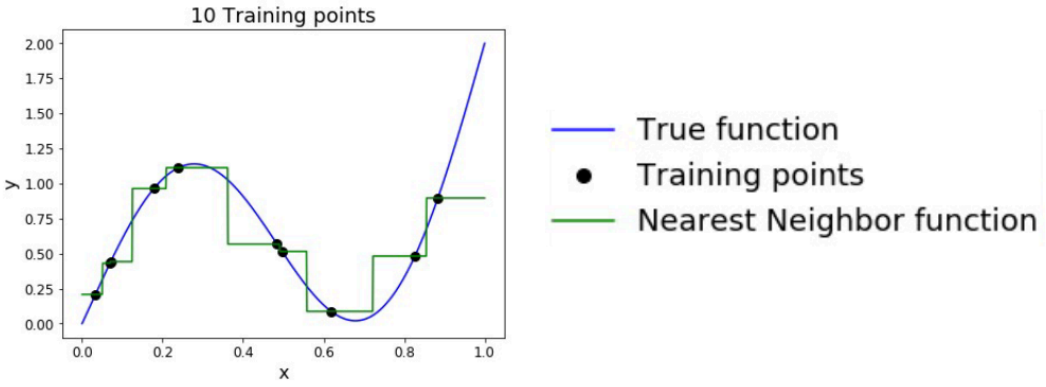
The line goes through the mean, bars indicated standard deviation

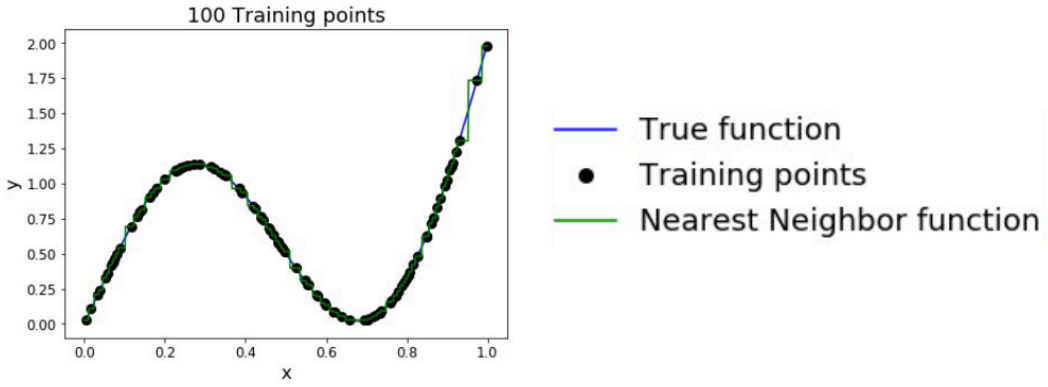
(Seems that k ~ 7 works best for this data)

Choose hyperparameters using the **validation set**, only run the test set at the very end!

4. Universal Approximation

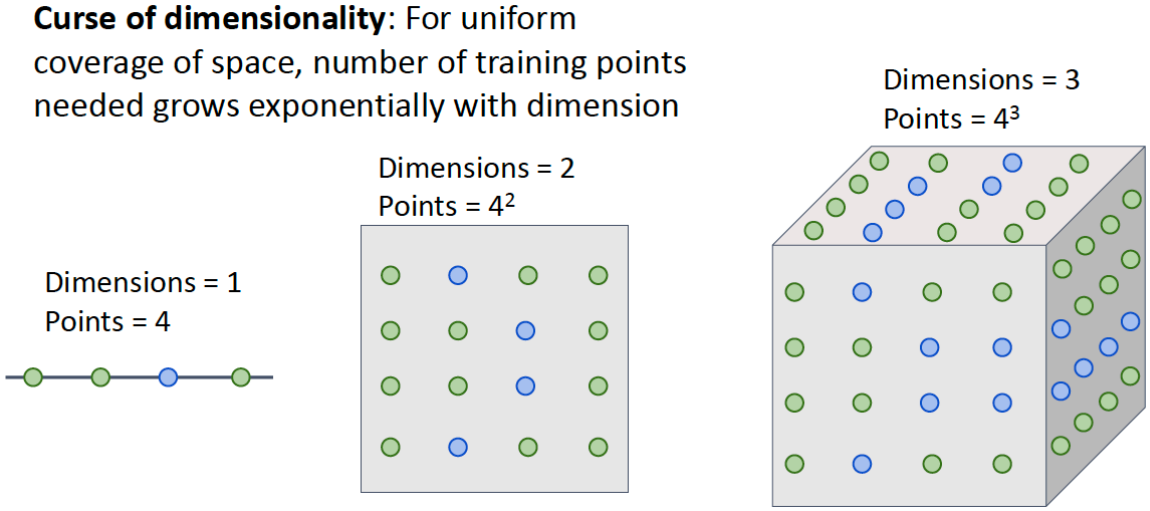
As the number of training samples goes to infinity, nearest neighbor can *represent any function*!





- Problem: **Curse of Dimensionality**

  当数据在更高维时，训练数据点会指数型增长，无法进行训练



eg. number of possible 32×32 binary images: $2^{32 \times 32} \approx 10^{308}$