Subject: Image classification pipeline / /

- An image is just a grid of numbers.

First classifier: Nearest Neighbor

- Memorize all data & labels
- Predict the label of the most similar training image

Example dataset: CIFAR10

↳ 10 classes
50,000 training images
10,000 testing images

How to compare images?

↳ Distance metric to compare images:

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

test image — training image — Absolute diff

$$\begin{bmatrix} 56 & 32 & 10 & 18 \\ 90 & 23 & 128 & 133 \\ 24 & 26 & 178 & 200 \\ 2 & 0 & 255 & 220 \end{bmatrix} - \begin{bmatrix} 10 & 20 & 24 & 17 \\ 8 & 10 & 89 & 100 \\ 12 & 16 & 178 & 170 \\ 4 & 32 & 233 & 112 \end{bmatrix} = \begin{bmatrix} 46 & 12 & 14 & 1 \\ 82 & 13 & 39 & 33 \\ 12 & 10 & 0 & 30 \\ 2 & 32 & 22 & 108 \end{bmatrix} \overset{\text{sum}}{=} 456$$

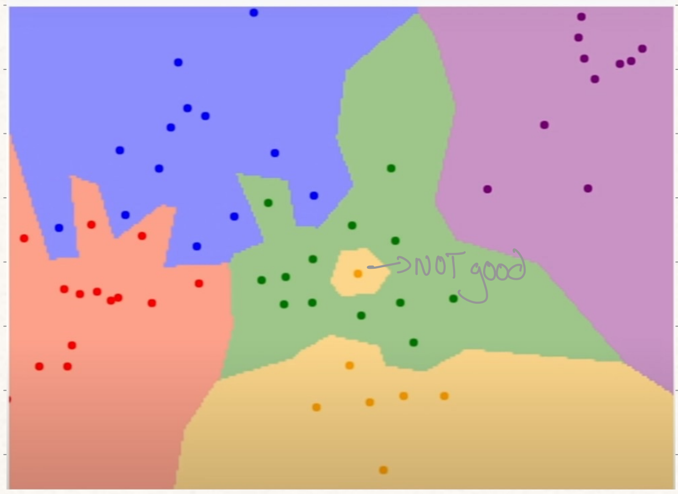
Time complexity:

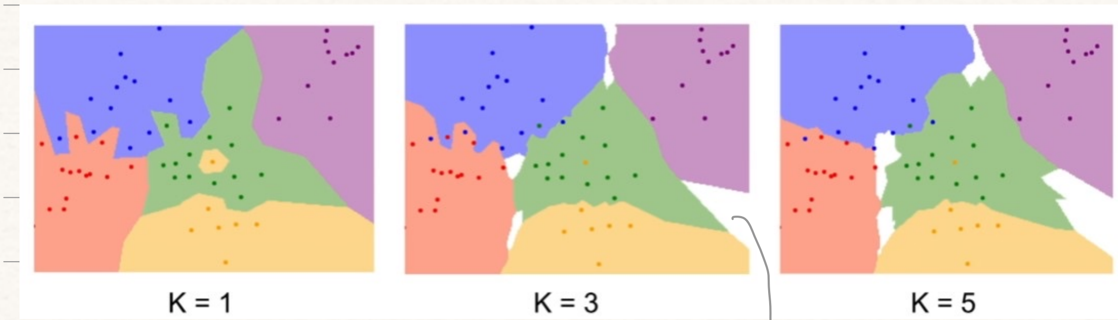
Training: $O(1)$

Predicting: $O(n)$ ⇒ slow

Subject: / /

```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
```





There was no nearest neighbor

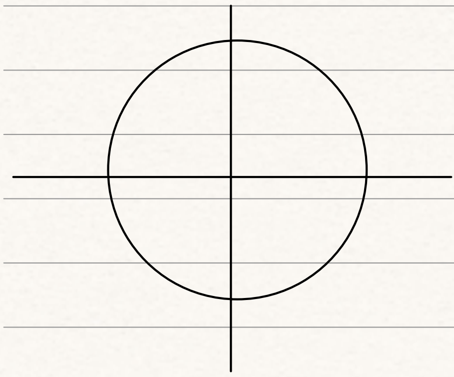
• Take majority vote while predicting

Subject: / /

- For a new point x, compute L1.
- Sort by decreasing order
- Choose k smallest distances
- Choose the category that appears majorily in those k points.

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

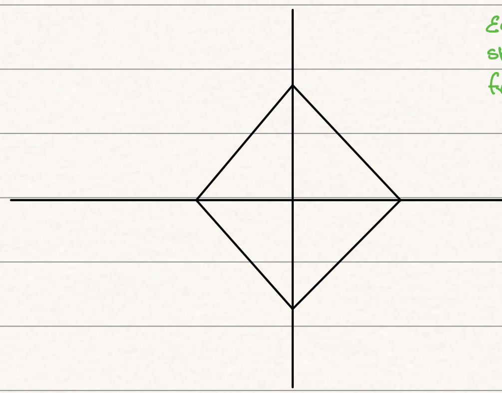


- Doesn't depend on coordinate system

Manhattan/L1 distance:

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

Each point on the shape is equidistant from origin



- Depends on coordinate system

Hyperparameters:

- Set them ahead of time
- Try different values

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

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

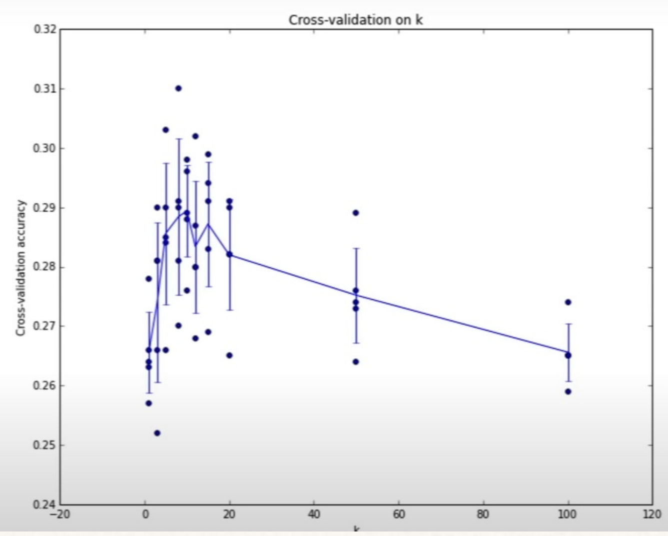
1) Choose hyperparametres
2) Perform 5 iterations:
   i) Train on folds 1,2,3,4; Test on fold 5
   ii) Train on folds 1,2,3,5; Test on fold 4

Subject: iii) Train on folds 1,2,4,5; Test on fold 3

iv) Train on folds 1,3,4,5; Test on fold 2

v) Train on folds 2,3,4,5; Test on fold 1

3. Calculate avg across all 5 iterations



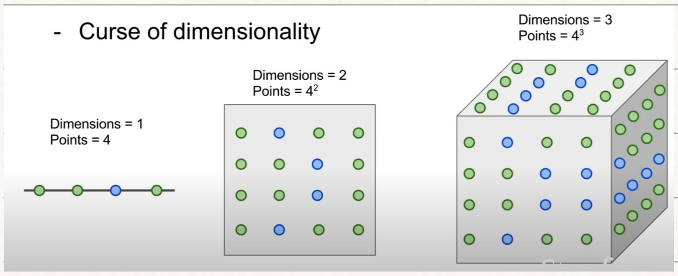
• k = 7 works best for this problem.

Problems of k-nearest:

• All 3 images to the left of original have same L2 distance

↳ Fails to capture preceptional differences

• Curse of dimensionality:



• For good predictions, we need points to cover the space densly. As dimensions increases, the points needed to cover the space densly increases exponentially

Subject:

Linear classification:

- Parametric model:

$$\underset{32\times32\times3}{\text{image}} \longrightarrow f(x, W) \longrightarrow \text{10 numbers giving class scores}$$

$$\underbrace{f(x, W)}_{10\times1} = \overbrace{W}^{10\times3072}\underbrace{x}_{3072\times1} + b \rightarrow 10\times1$$