- input: array of numbers
  e.g. 100x100x3 (RGB channels)
- challenges:
  - viewpoint
  - illumination
  - deformation
  - occlusion
  - background clutter
  - intraclass variation
- algorithm

```
def classify_image(image):
    # magic
    return class_label
```

- solution proposed:
  - edge ?
  - corners ?
- data-driven

```
def train(img, labels):
    # machine learning!
    return model

def predict(model, img):
    return class
```

- first classifier : Nearest Neighbor
  - memorize all data & labels
  - predict the label of the most similar training image
  - e.g. CIFAR10
    10 classes
    50,000 training images
    10,000 testing images
  - L1 distance $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$

```
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
```

  - Q: N examples
    how fast is it ?
    A: Train O(1)
    predict O(N)
    (bad classifier should be fast @ prediction
    slow @ training)

  - pts: training data
    region: classes
- K-NN

K-Nearest Neighbors

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

K = 1 K = 3 K = 5

  - difference metric
    L1 = manh, L2 = eucl
- hyperparameter
  - the parameters set for training
  - split data into
    - train
    - val
    - test

    } choose hyperparameters on val
    evaluate on test
  - cross-validation

Setting Hyperparameters

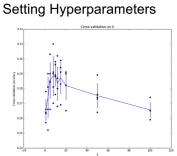
Your Dataset

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

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

Setting Hyperparameters



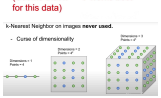
Example of 5-fold cross-validation for the value of **k**.

Each point: single outcome.

The line goes through the mean, bars indicated standard deviation

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

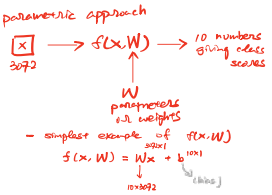
  - curse of dimensionality



- Linear classifiers
  - Neural Network

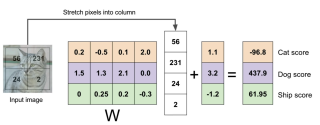
linear classifier

  - CIFAR10
    50,000 train (32x32x3)
    10,000 test
    3072
  - parametric approach

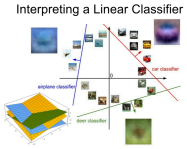


  - simplest example of f(x,W)
    $f(x, W) = Wx + b$ (10x1 = 10x3072 · 3072x1 + 10x1)

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Interpreting a Linear Classifier



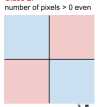
f(x,W) = Wx + b

Array of **32x32x3** numbers (3072 numbers total)

Hard cases for a linear classifier

Class 1:
number of pixels > 0 odd

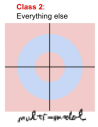
Class 2:
number of pixels > 0 even



parity problem (true/false)

Class 1:
1 <= L2 norm <= 2

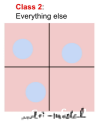
Class 2:
Everything else



multi-modal

Class 1:
Three modes

Class 2:
Everything else



multi-modal