

• imports: array of numbers
e.g. 300x600x3 (3D channels)

- challenges:
 - visual pattern
 - illumination
 - deformation
 - occlusion
 - background clutter
 - intraclass variation

• algorithm:
def classifyImage(image):
 # logic
 return class_label

- solution proposal:
- edge?
- corners?

• data-driven:
def train(img, labels):
 # Machine learning!
 return model
def predict(model, img):
 return class

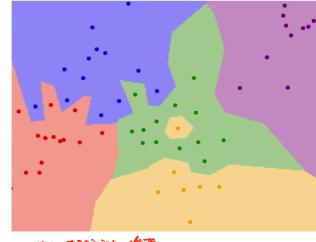
- first classifier: Nearest Neighbors
 - memorize all
 - store & labels
 - predict the label of the most similar training image

- e.g. CIFAR10
 - 10 classes
 - 50,000 training images
 - 10,000 testing images
- L1 distance

$$| \begin{matrix} 4 \\ 5 \\ 3 \end{matrix} - | \begin{matrix} 1 \\ 1 \\ 1 \end{matrix} | = | \begin{matrix} 6 \\ 2 \\ 2 \end{matrix} |_1 = 15$$

```
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-dimensional 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]  
        # let's 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 range(num_test):  
            # find the nearest training image to the i'th test image  
            # using the L1 distance  
            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
```

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



- K-NN K-Nearest Neighbors

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



- hyperparameters
 - the parameters set during training
 - split data into { choose hyperparameters on val
- train val
- test } evaluate on test
- cross-validation

Setting Hyperparameters

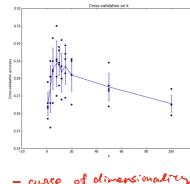
Your Dataset

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



- curse of dimensionality

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)

K-Nearest Neighbor on images never used.



• Linear classifiers

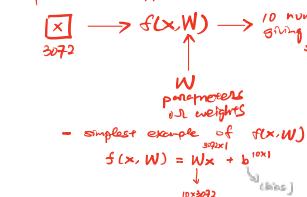
- Neural Network



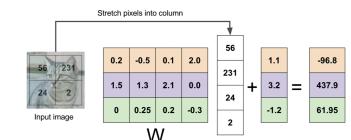
- CIFAR10

50,000 train (32x32x3)
10,000 test 3072

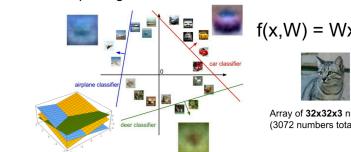
- parametric approach



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



Interpreting a Linear Classifier



Array of 32x32x3 numbers
(3072 numbers total)

Hard cases for a linear classifier

