Lecture 2:
Image Classification pipeline

Image Classification: a core task in Computer Vision

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

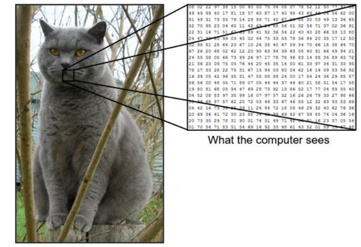
→ cat

The problem:
semantic gap

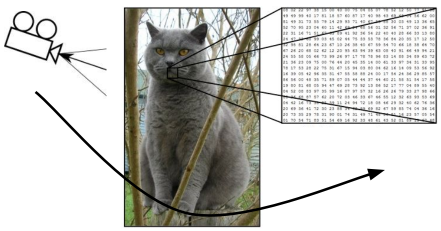
Images are represented as 3D arrays of numbers, with integers between [0, 255].

E.g.
300 x 100 x 3

(3 for 3 color channels RGB)



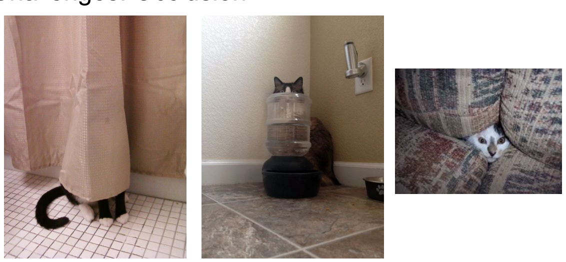
Challenges: Viewpoint Variation



Challenges: Illumination

Challenges: Deformation

Challenges: Occlusion



Challenges: Background clutter

Challenges: Intraclass variation

An image classifier

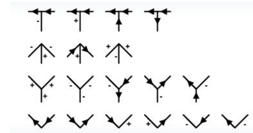
```
def predict(image):
    # ????
    return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

Attempts have been made

???

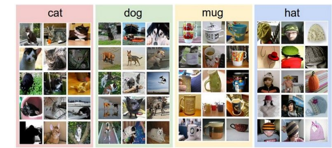


Data-driven approach:

1. Collect a dataset of images and labels
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images

```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

Example training set



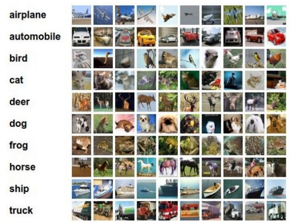
First classifier: **Nearest Neighbor Classifier**

```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

Remember all training images and their labels

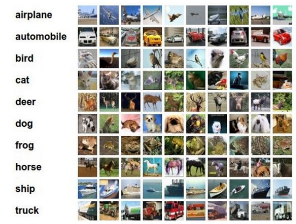
Predict the label of the most similar training image

Example dataset: **CIFAR-10**
10 labels
50,000 training images, each image is tiny: 32x32
10,000 test images.



Example dataset: **CIFAR-10**
10 labels
50,000 training images
10,000 test images.

For every test image (first column), examples of nearest neighbors in rows



How do we compare the images? What is the **distance metric**?

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

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

\-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

$\xrightarrow{\text{add}}$ 456


```
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
```

Nearest Neighbor classifier


```
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
```

Nearest Neighbor classifier

remember the training data

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 every test image:
- find nearest train image with L1 distance
- predict the label of nearest training image

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
```

Q: how does the classification speed depend on the size of the training data?

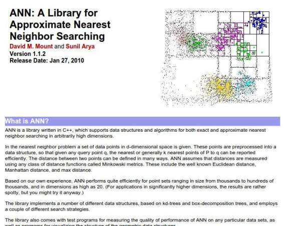
Nearest Neighbor classifier

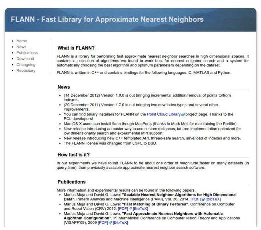
```python
import numpy as np

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
```

Q: how does the classification speed depend on the size of the training data?
linearly :(

This is **backwards**:
- test time performance is usually much more important in practice.
- CNNs flip this: expensive training, cheap test evaluation

Aside: Approximate Nearest Neighbor
find approximate nearest neighbors quickly





The choice of distance is a **hyperparameter**
common choices:

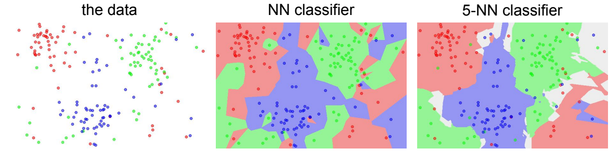
L1 (Manhattan) distance

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

L2 (Euclidean) distance

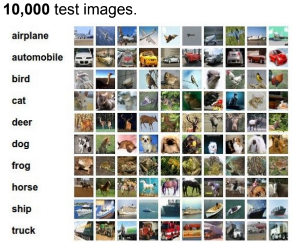
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$

k-Nearest Neighbor
find the k nearest images, have them vote on the label

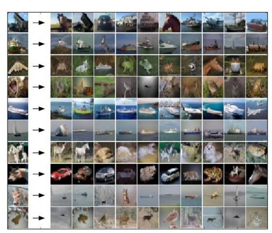


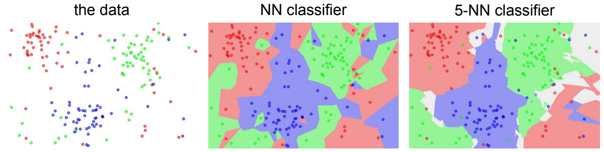
http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Example dataset: **CIFAR-10**
10 labels
50,000 training images
10,000 test images.

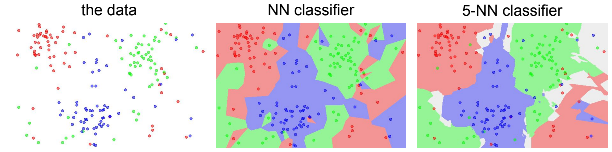


For every test image (first column), examples of nearest neighbors in rows





Q: what is the accuracy of the nearest neighbor classifier on the training data, when using the Euclidean distance?



Q2: what is the accuracy of the **k-**nearest neighbor classifier on the training data?

What is the best **distance** to use?
What is the best value of **k** to use?

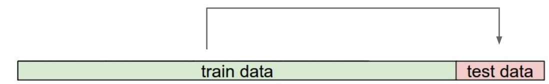
i.e. how do we set the **hyperparameters**?

What is the best **distance** to use?
What is the best value of **k** to use?

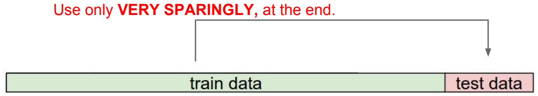
i.e. how do we set the **hyperparameters**?

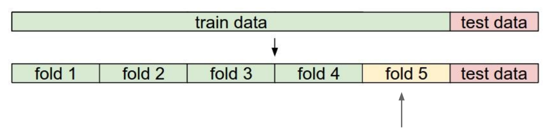
Very problem-dependent.
Must try them all out and see what works best.

Try out what hyperparameters work best on test set.

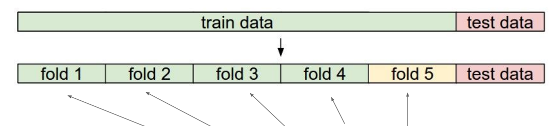


Trying out what hyperparameters work best on test set:
Very bad idea. The test set is a proxy for the generalization performance!
Use only **VERY SPARINGLY,** at the end.

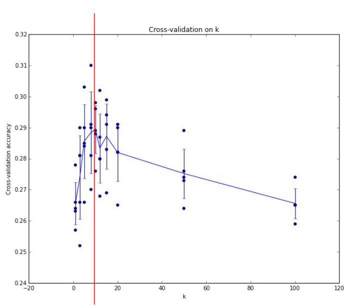




Validation data
use to tune hyperparameters



Cross-validation
cycle through the choice of which fold is the validation fold, average results.



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

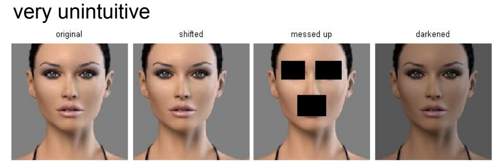
Each point: single outcome.

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.**

- terrible performance at test time
- distance metrics on level of whole images can be very unintuitive

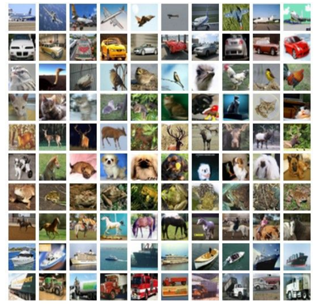


(all 3 images have same L2 distance to the one on the left)

Summary

- **Image Classification:** We are given a **Training Set** of labeled images, asked to predict labels on **Test Set**. Common to report the **Accuracy** of predictions (fraction of correctly predicted images)
- We introduced the **k-Nearest Neighbor Classifier**, which predicts the labels based on nearest images in the training set
- We saw that the choice of distance and the value of k are **hyperparameters** that are tuned using a **validation set**, or through **cross-validation** if the size of the data is small.
- Once the best set of hyperparameters is chosen, the classifier is evaluated once on the test set, and reported as the performance of kNN on that data.

Linear Classification



Example dataset: **CIFAR-10**
10 labels
50,000 training images
each image is **32x32x3**
10,000 test images.

Parametric approach

image parameters

f(**x**,**W**)

10 numbers, indicating class scores

[32x32x3]
array of numbers 0...1
(3072 numbers total)

Parametric approach: **Linear classifier**

$$f(x, W) = Wx$$

10 numbers, indicating class scores

[32x32x3]
array of numbers 0...1

Parametric approach: **Linear classifier**

$$\boxed{f(x, W)} = \boxed{W}\boxed{x}$$

3072x1

10x1 **10x3072**

10 numbers, indicating class scores

[32x32x3]
array of numbers 0...1

parameters, or "weights"

Parametric approach: **Linear classifier**

$$\boxed{f(x, W)} = \boxed{W}\boxed{x} \quad \boxed{(+b)}$$

3072x1 **10x1**

10x1 **10x3072**

10 numbers, indicating class scores

[32x32x3]
array of numbers 0...1

parameters, or "weights"

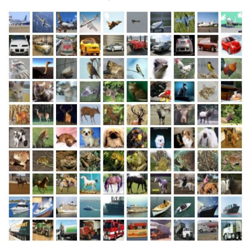
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

stretch pixels into single column

input image

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{x_i} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} \longrightarrow \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{f(x_i; W, b)} \begin{array}{l} \text{cat score} \\ \text{dog score} \\ \text{ship score} \end{array}$$

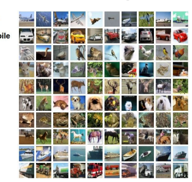
Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

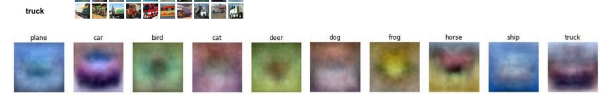
Q: what does the linear classifier do, in English?

Interpreting a Linear Classifier

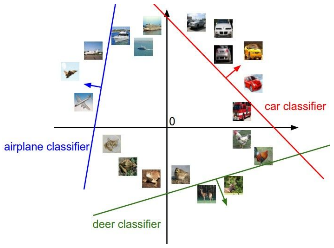


$$f(x_i, W, b) = Wx_i + b$$

Example trained weights of a linear classifier trained on CIFAR-10:



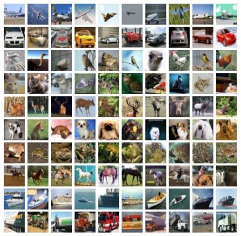
Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

[32x32x3]
array of numbers 0...1
(3072 numbers total)

Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

Q2: what would be a very hard set of classes for a linear classifier to distinguish?

So far: We defined a (linear) **score function**: $f(x_i, W, b) = Wx_i + b$

Example class scores for 3 images, with a random W:

airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

$$f(x, W) = Wx$$

Coming up:

- Loss function (quantifying what it means to have a "good" W)
- Optimization (start with random W and find a W that minimizes the loss)
- ConvNets! (tweak the functional form of f)