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 $\mathbb{R}^d$ arrays of numbers

- E.g. $\mathbb{R}^3$ with integers between [0, 255], where $d=3$ represents 3 color channels (RGB)
Challenges: viewpoint variation

Michelangelo 1475-1564
Challenges: illumination
Challenges: scale
Challenges: deformation
Challenges: occlusion

Magritte, 1957
Challenges: background clutter

Kilmeny Niland. 1995
Challenges: intra-class variation
An image classifier

```python
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.
Data-driven approach:
1. Collect a dataset of images and label them
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images
First classifier: **Nearest Neighbor Classifier**

- Remember all training images and their labels
- Predict the label of the most similar training image

```python
def train(train_images, train_labels):
    # build a model of images -> labels

def predict(image):
    # evaluate the model on the image
    return class_label
```
Example dataset: CIFAR-10
10 labels
50,000 training images
10,000 test images.

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
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:**

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

Where \( I_1 \) denotes image 1, and \( p \) denotes each pixel.
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 ""
        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

```
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

Q: what is the complexity of the NN classifier w.r.t training set of N images and test set of M images?

1. at training time?

2. at test time?
Nearest Neighbor classifier

Q: what is the complexity of the NN classifier w.r.t training set of N images and test set of M images?

1. at training time? O(1)
2. at test time? O(NM)
Nearest Neighbor classifier

1. at training time?
\[ O(1) \]
2. at test time?
\[ O(NM) \]

This is **backwards**:
- test time performance is usually much more important.
- CNNs flip this: expensive training, cheap test evaluation
Aside: Approximate Nearest Neighbor
find approximate nearest neighbors quickly

ANN: A Library for Approximate Nearest Neighbor Searching
David M. Mount and Sunil Arya
Version 1.1.2
Release Date: Jan 27, 2010

What is ANN?

ANN is a library written in C++, which supports data structures and algorithms for both exact and approximate nearest neighbor searching in arbitrarily high dimensions.

In the nearest neighbor problem a set of data points in d-dimensional space is given. These points are preprocessed into a data structure, so that given any query point q, the nearest or generally k nearest points of p to q can be reported efficiently. The distance between two points can be defined in many ways. ANN assumes that distances are measured using any class of distance functions called Minkowski metrics. These include the well known Euclidean distance, Manhattan distance, and max distance.

Based on our own experience, ANN performs quite efficiently for point sets ranging in size from thousands to hundreds of thousands, and in dimensions as high as 20. (For applications in significantly higher dimensions, the results are rather spotty, but you might try it anyway.)

The library implements a number of different data structures, based on kd-trees and box-decomposition trees, and employs a couple of different search strategies.

The library also comes with test programs for measuring the quality of performance of ANN on any particular data sets, as well as programs for visualizing the structure of the geometric data structures.
The choice of distance is a hyperparameter

- **L1 (Manhattan) distance**
  \[ d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p| \]

- **L2 (Euclidean) distance**
  \[ d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2} \]

- Two most commonly used special cases of p-norm
  \[ ||x||_p = \left( |x_1|^p + \cdots + |x_n|^p \right)^{\frac{1}{p}} \quad p \geq 1, x \in \mathbb{R}^n \]
k-Nearest Neighbor
find the k nearest images, have them vote on the label

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.
Trying out what hyperparameters work best on test set: Very bad idea. The test set is a proxy for the generalization performance.
Validation data
use to tune hyperparameters
evaluate on test set ONCE at the end
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)
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.