Working with Image Data
Bootcamp Section 1

Joseph Adler, Drew Conway, Jake Hofman, Hilary Mason

February 1, 2011

Acquiring image data

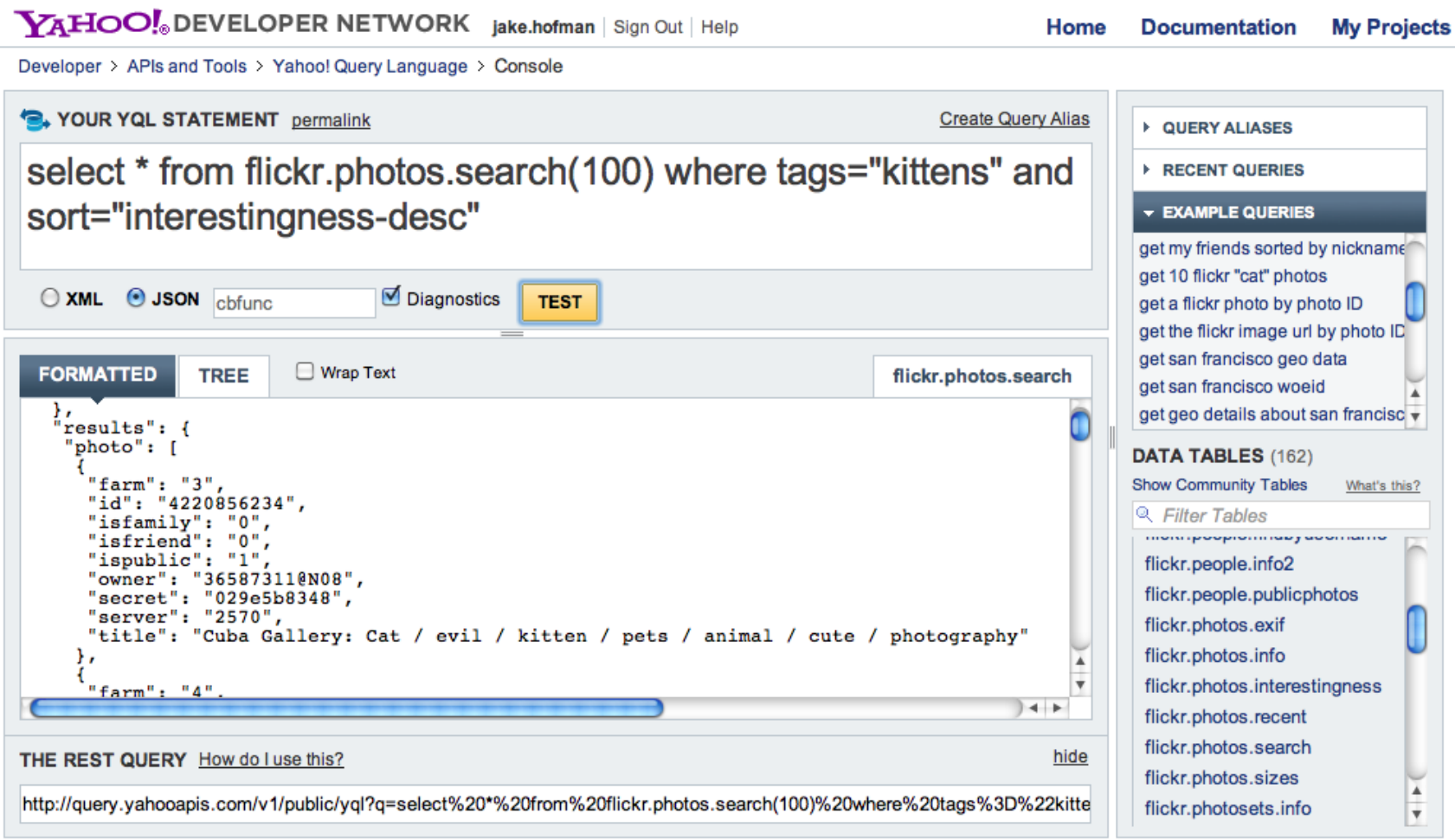
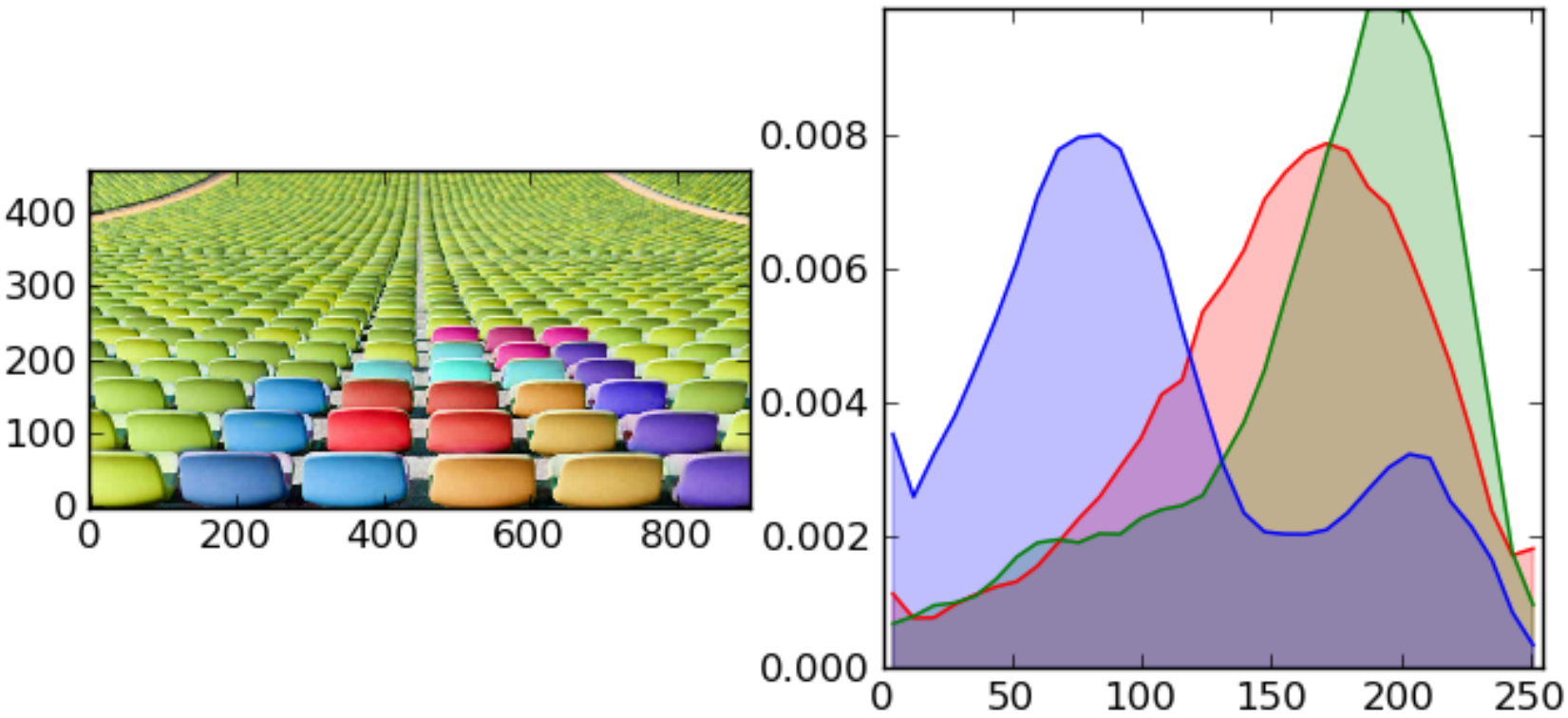
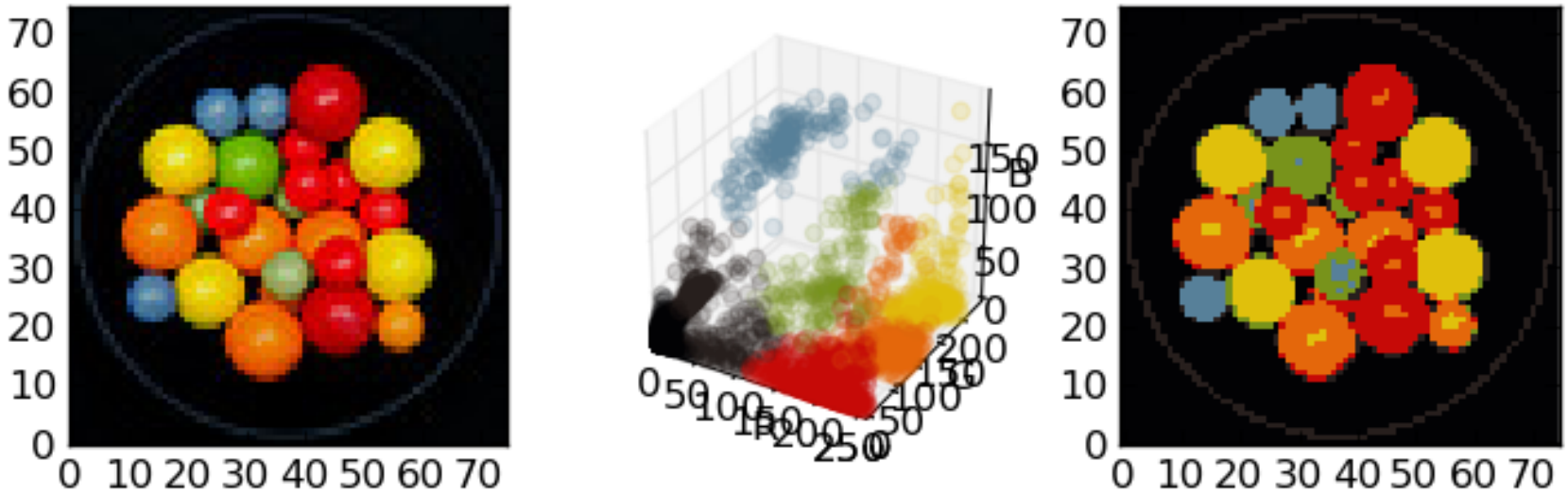


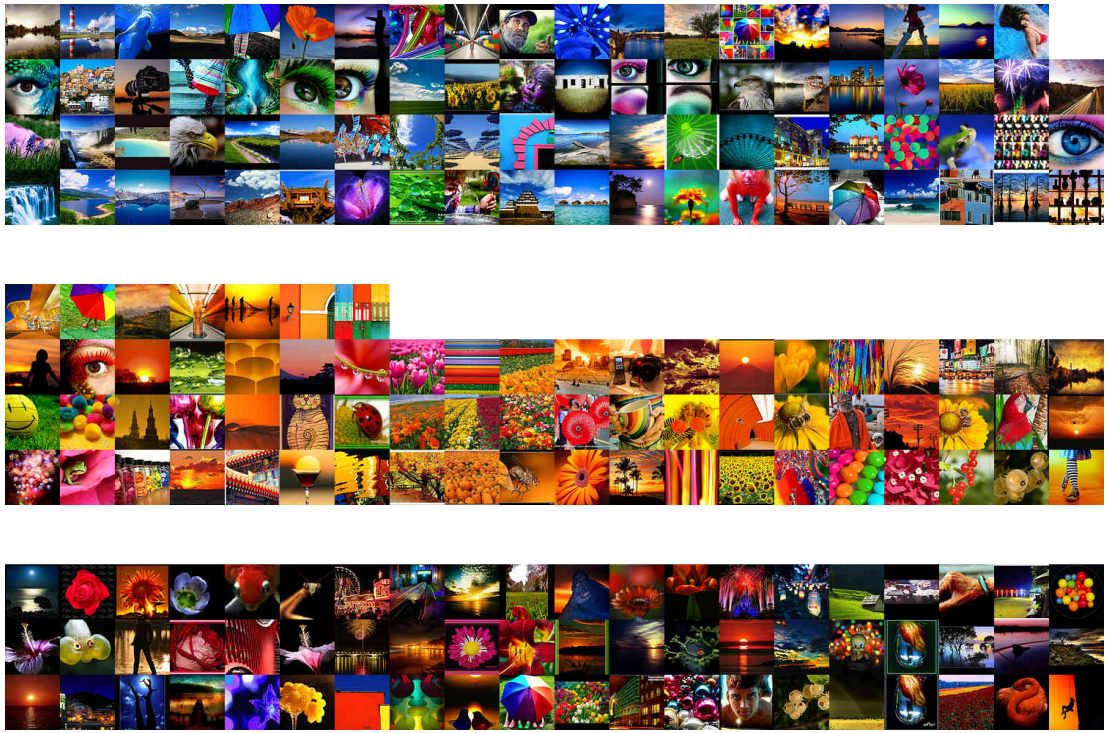
Image features



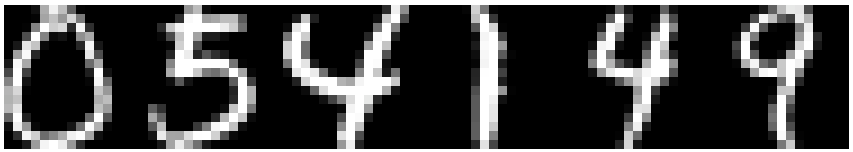
Clustering pixels



Clustering images



Classifying images



0 5 4 1 4 9

Outline

Simple screen scraping

One-liner to scrape images from a webpage

```
wget -O- http://bit.ly/gpCSQi |
  tr ''\''"=' '\n' |
  egrep '^http.*(png|jpg|gif)' |
  xargs wget
```

Simple screen scraping

One-liner to scrape images from a webpage

```
wget -O- http://bit.ly/gpCSQi |
  tr ''\''"=' '\n' |
  egrep '^http.*(png|jpg|gif)' |
  xargs wget
```

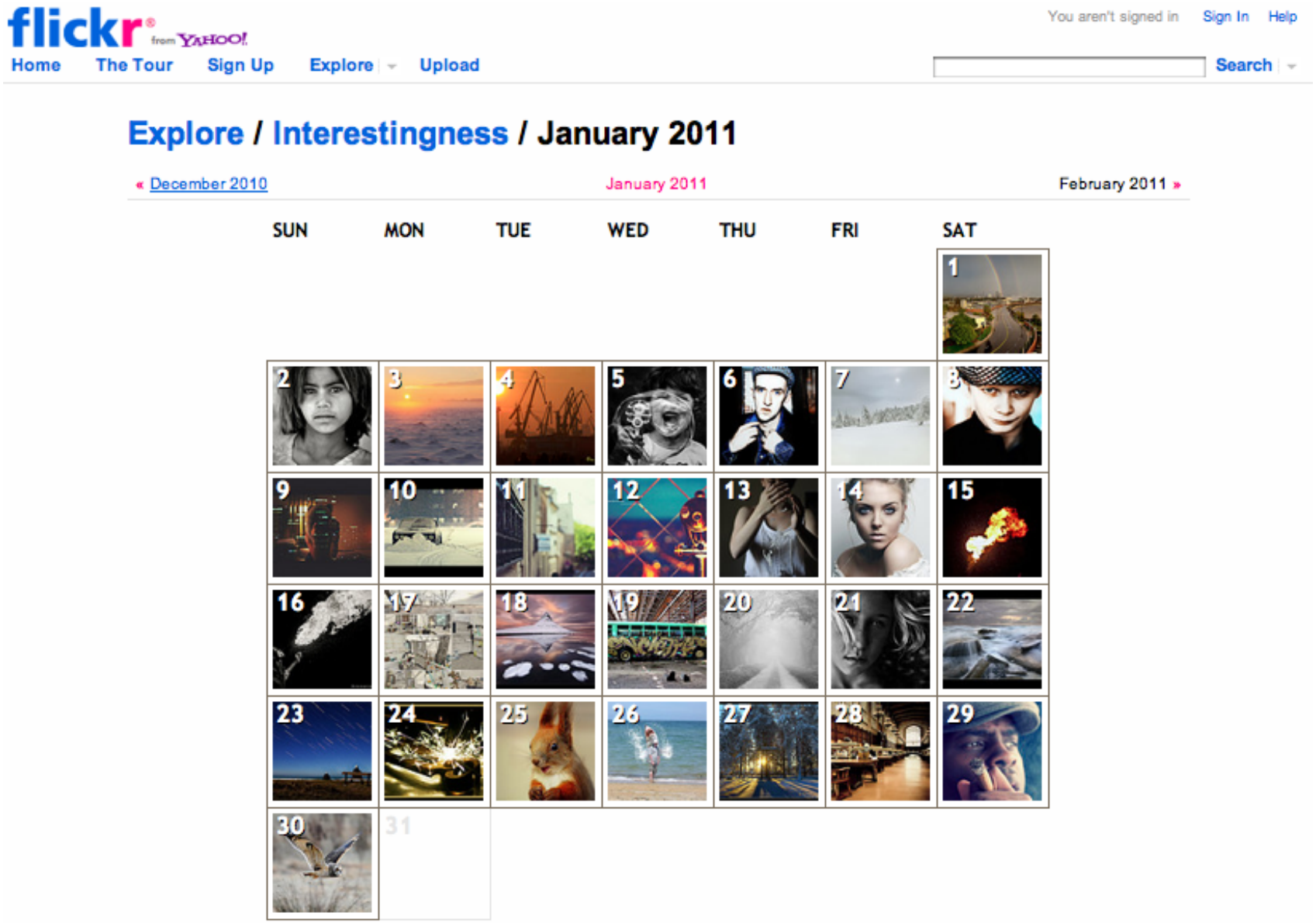
- get page source
- translate quotes and = to newlines
- match urls with image extensions
- download qualifying images

Simple screen scraping

One-liner to download ESL digit data

```
wget -Nr --level=1 --no-parent http://bit.ly/fsymq6
```

“cat flickr | xargs wget”?



Flickr API

Home | API | Community | Business | Attributions

Flickr API Changelog

Getting Started

To begin using the Flickr API:

1. Request an API key, to sign your API requests.
2. Read the Community Guidelines and the API Terms of Use.
3. Build, build, build. Test, test, test.
4. Launch (and if it's an app of interest to the Flickr community, create a profile for your app in the Flickr App Garden).

YQL: SELECT * FROM Internet[1]

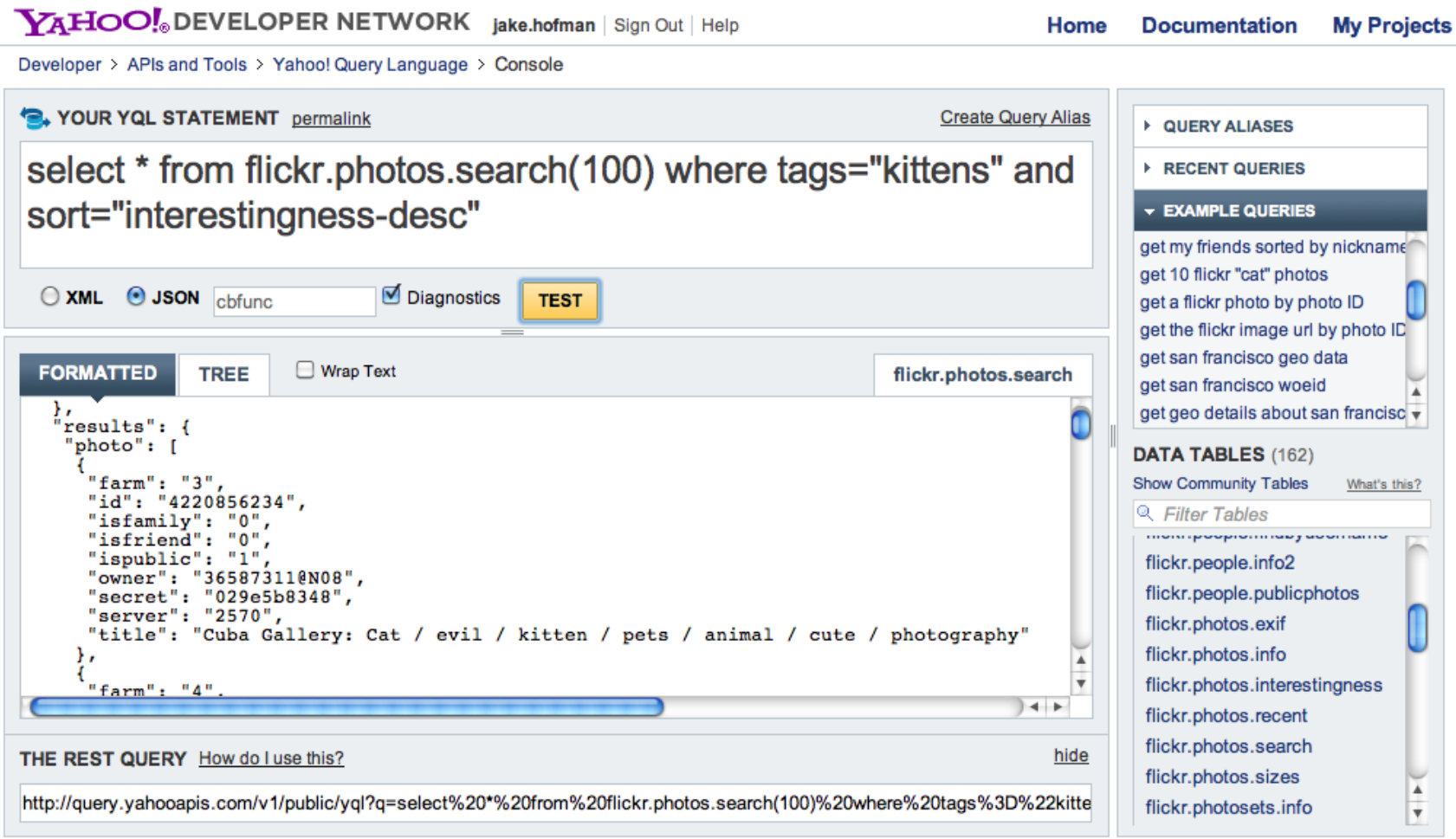
What is YQL?

The Yahoo! Query Language is an expressive SQL-like language that lets you query, filter, and join data across Web services. With YQL, *apps run faster with fewer lines of code and a smaller network footprint.*

`http://developer.yahoo.com/yql`

[1] `http://oreillynet.com/pub/e/1369`

YQL: Console



http://developer.yahoo.com/yql/console

YQL + Python[2]

Python function for public YQL queries

```
YQL_PUBLIC = 'http://query.yahooapis.com/v1/public/yql'

def yql_public(query):
  # escape query
  query_str = urlencode({'q': query, 'format': 'json'})

  # fetch results
  url = '%s?%s' % (YQL_PUBLIC, query_str)
  result = urlopen(url)

  # parse json and return
  return json.load(result)['query']['results']
```

[2] See `http://python-yql.org/` for a more robust client

YQL + Python + Flickr

Fetch info for "interestingness" photos

```
./simpleyql.py ''select * from
  flickr.photos.interestingness(100)''
```

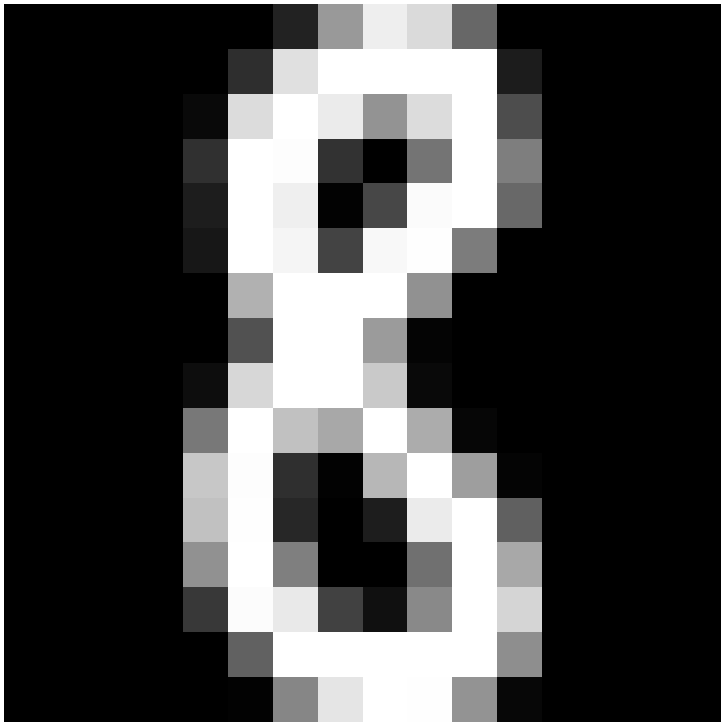
Download thumbnails for photos tagged with "vivid"

```
./download_flickr.py vivid 500
```

Outline

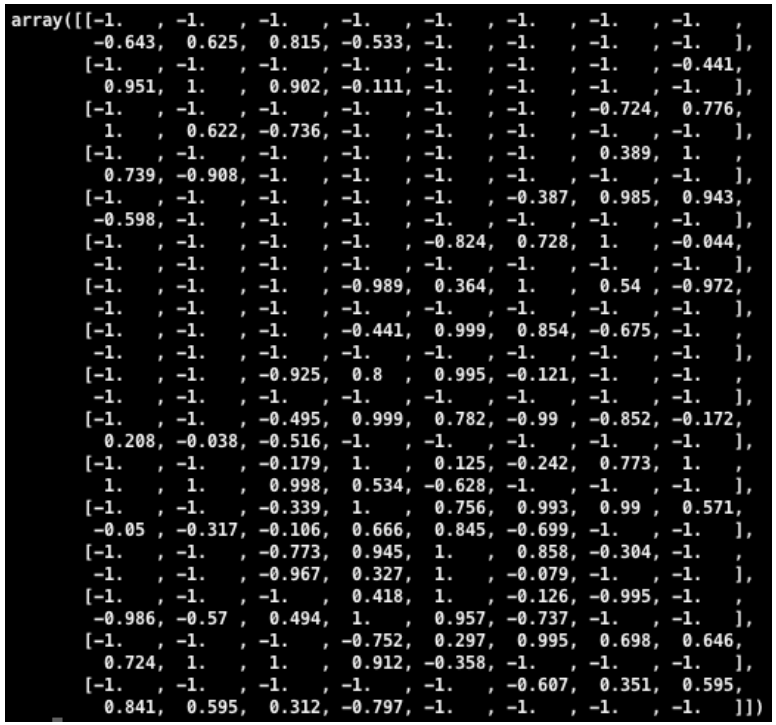
Images as arrays

Grayscale images $\leftrightarrow$ 2-d arrays of $M \times N$ pixel intensities



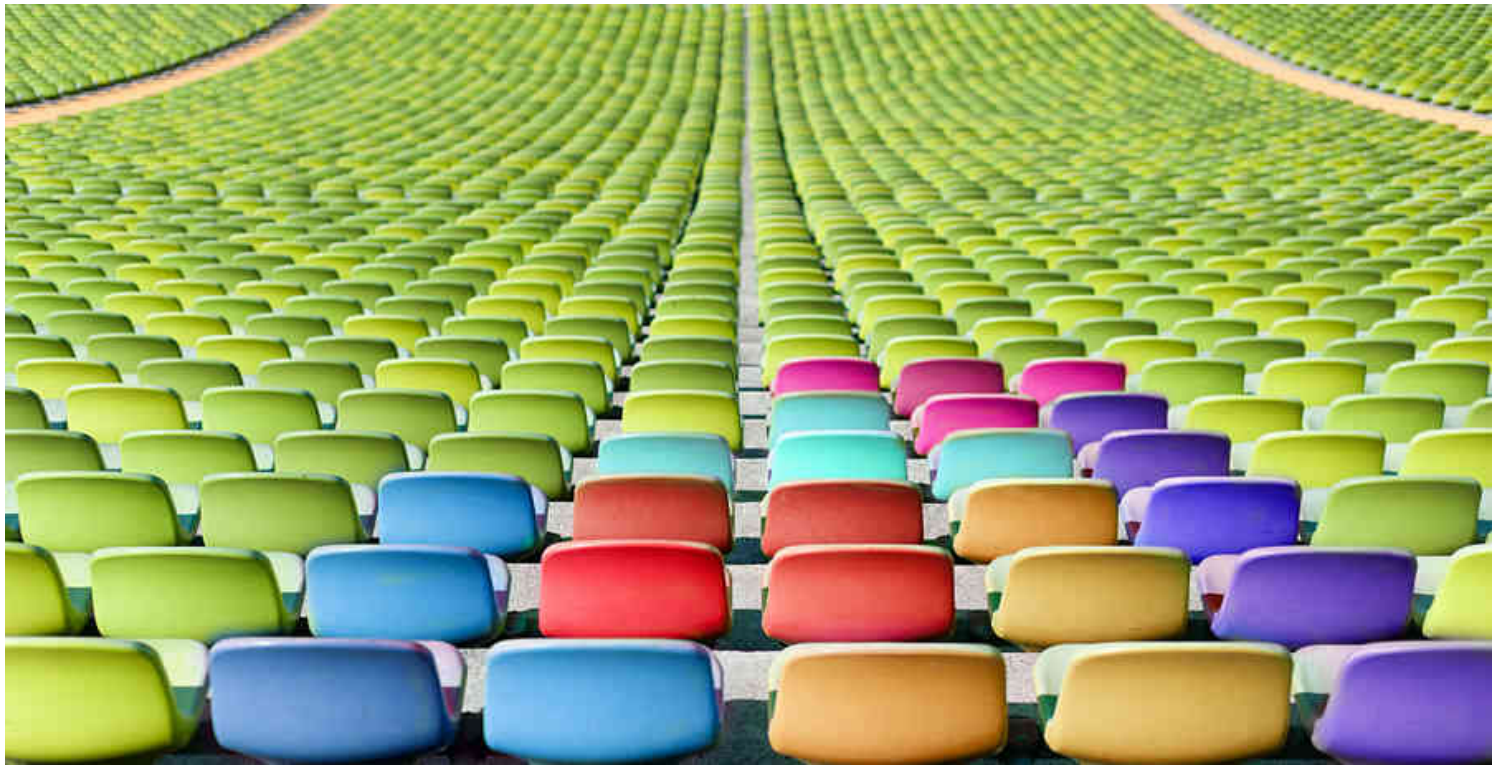
Images as arrays

Grayscale images $\leftrightarrow$ 2-d arrays of $M \times N$ pixel intensities



Images as arrays

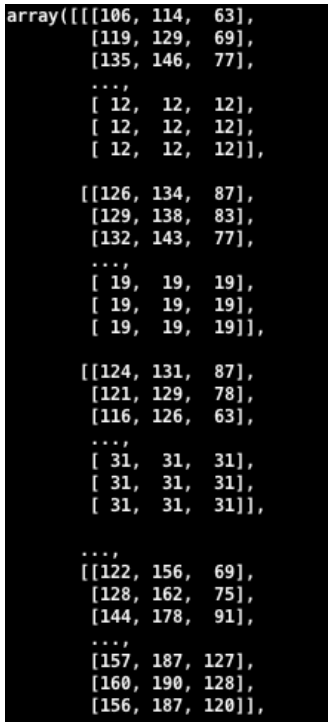
Color images $\leftrightarrow$ 3-d arrays of $M \times N \times 3$ RGB pixel intensities



```
import matplotlib.image as mpimg
I = mpimg.imread('chairs.jpg')
```

Images as arrays

Color images $\leftrightarrow$ 3-d arrays of $M \times N \times 3$ RGB pixel intensities

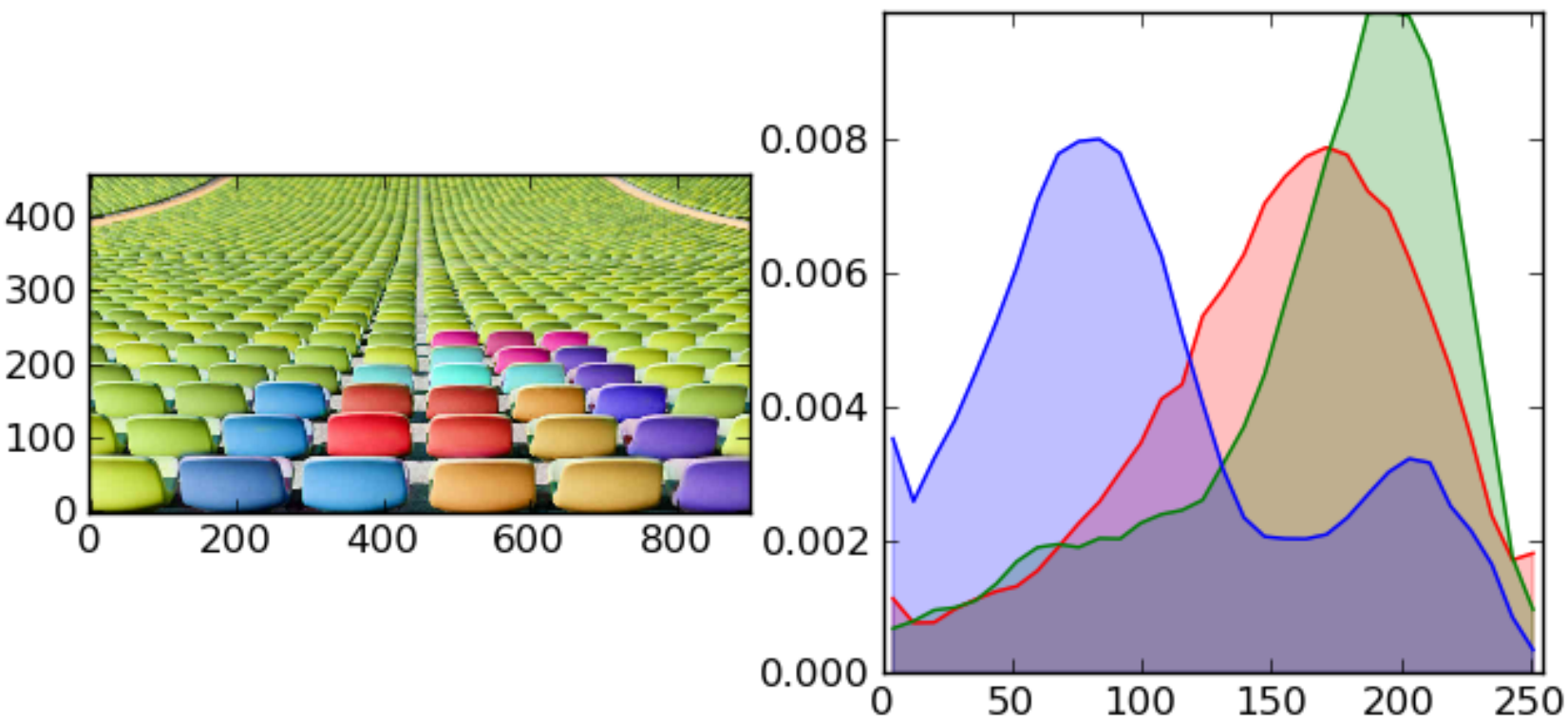


```
array([[[106, 114,  63],
        [119, 129,  69],
        [135, 146,  77],
        ...,
        [ 12,  12,  12],
        [ 12,  12,  12],
        [ 12,  12,  12]],

       [[126, 134,  87],
        [129, 138,  83],
        [132, 143,  77],
        ...,
        [ 19,  19,  19],
        [ 19,  19,  19],
        [ 19,  19,  19]],

       [[124, 131,  87],
        [121, 129,  78],
        [116, 126,  63],
        ...,
        [ 31,  31,  31],
        [ 31,  31,  31],
        [ 31,  31,  31]],

       ...,
       [[122, 156,  69],
        [128, 162,  75],
        [144, 178,  91],
        ...,
        [157, 187, 127],
        [160, 190, 128],
        [156, 187, 120]],
```

```
import matplotlib.image as mpimg
I = mpimg.imread('chairs.jpg')
```

Intensity histograms

Disregard all spatial information, simply count pixels by intensities
(e.g. lots of bright green and dark blue pixels)



Intensity histograms

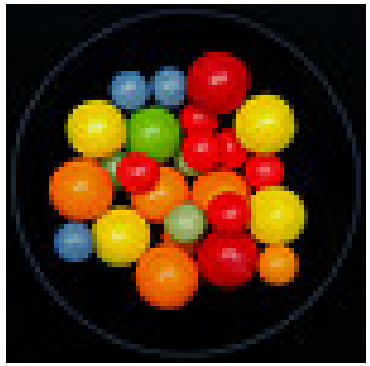
How many bins for pixel intensities?

Too many bins gives a noisy, **overly complex** representation of the data, while using **too few** bins results in an **overly simple** one

Outline

Clustering images

Clustering is an *unsupervised* learning task by which we look for structure in the data, grouping similar examples together



e.g., find groups of similar pixels within a single image

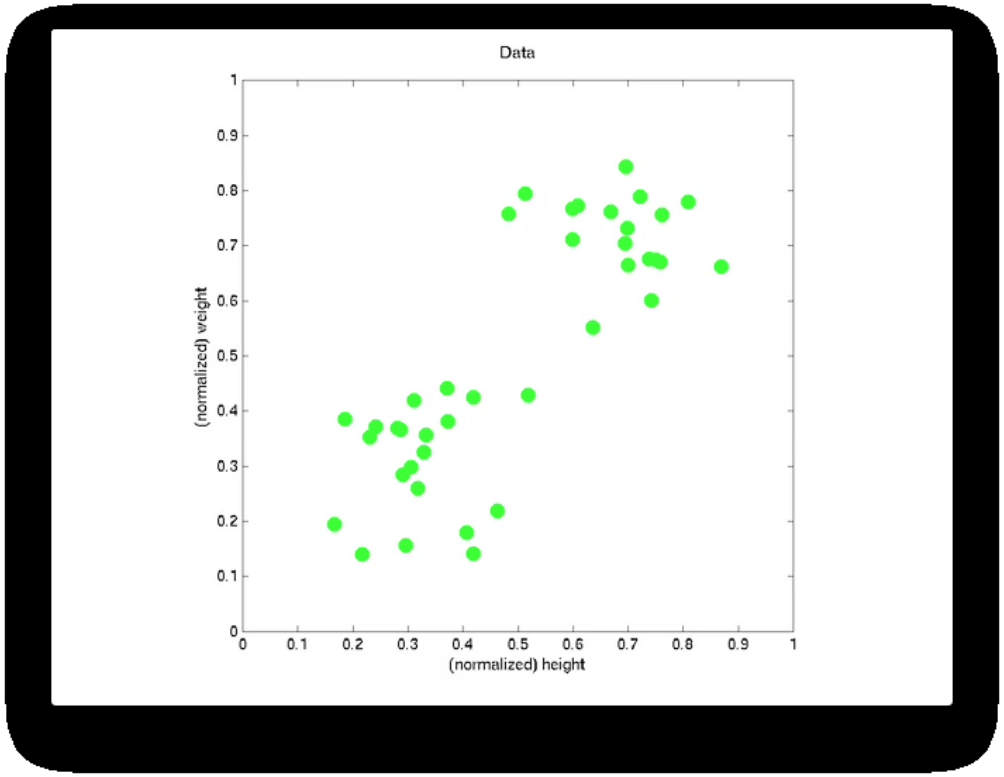
Clustering images

Clustering is an *unsupervised* learning task by which we look for structure in the data, grouping similar examples together

e.g., find groups of similar images across a collection of images

K-means clustering

K-means: represent each cluster by the average of its points



Learn by iteratively updating cluster means and point assigments

K-means clustering

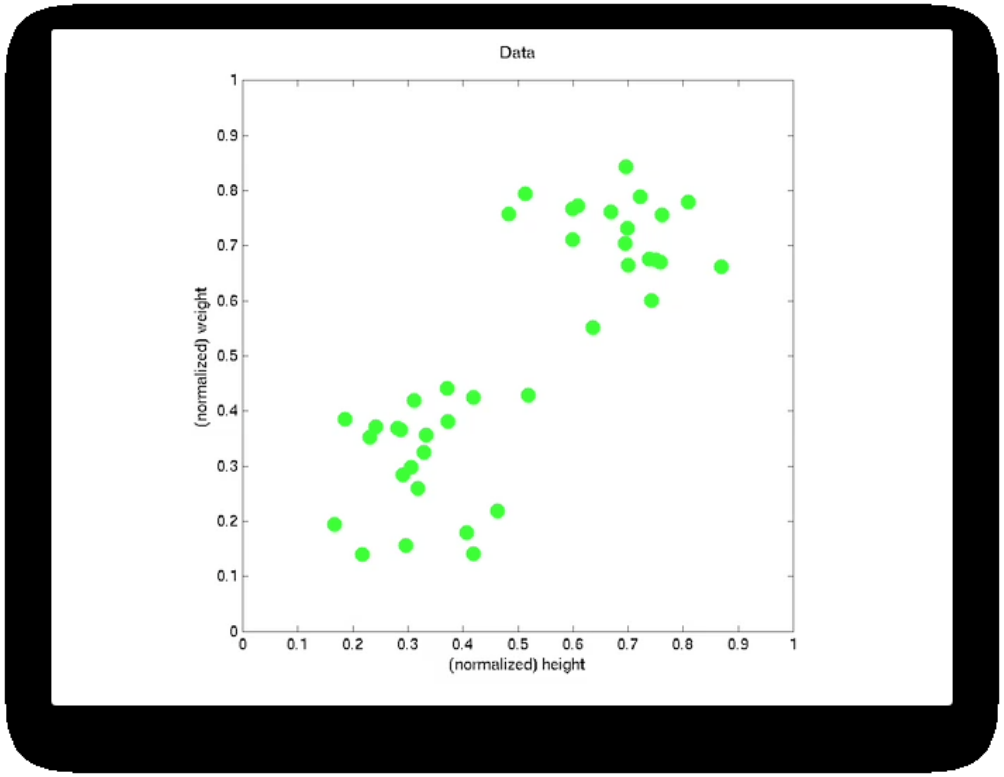
K-means:

Choose number of clusters

Initialize cluster centers

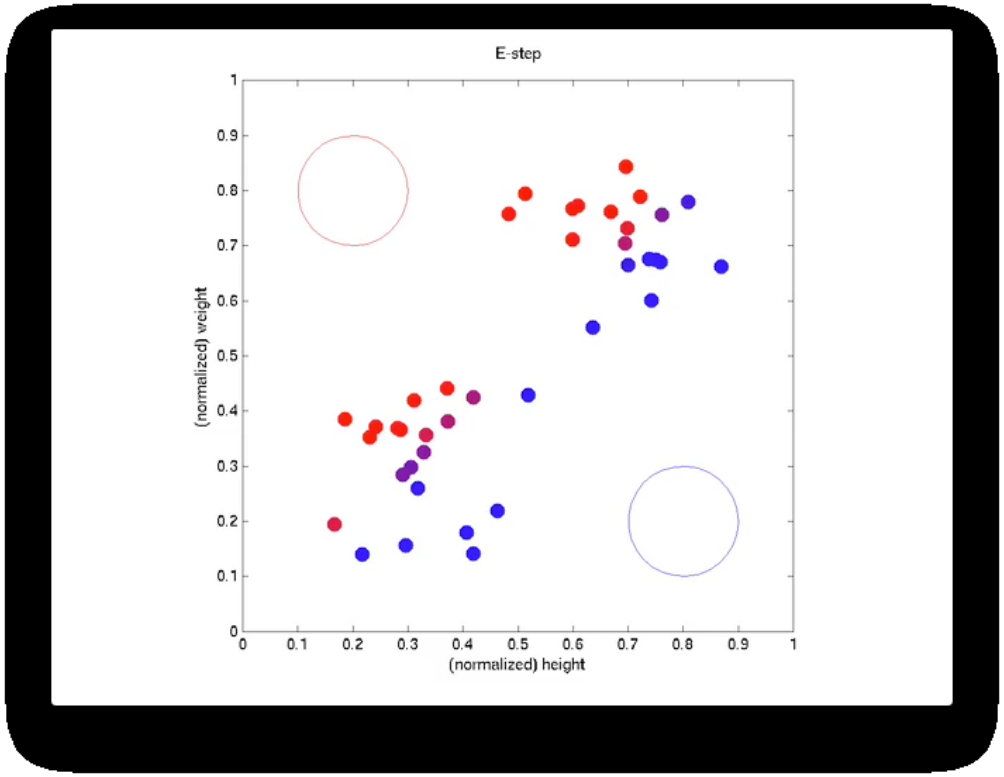
While not converged:

- Assign each point to closest cluster
- Update cluster centers



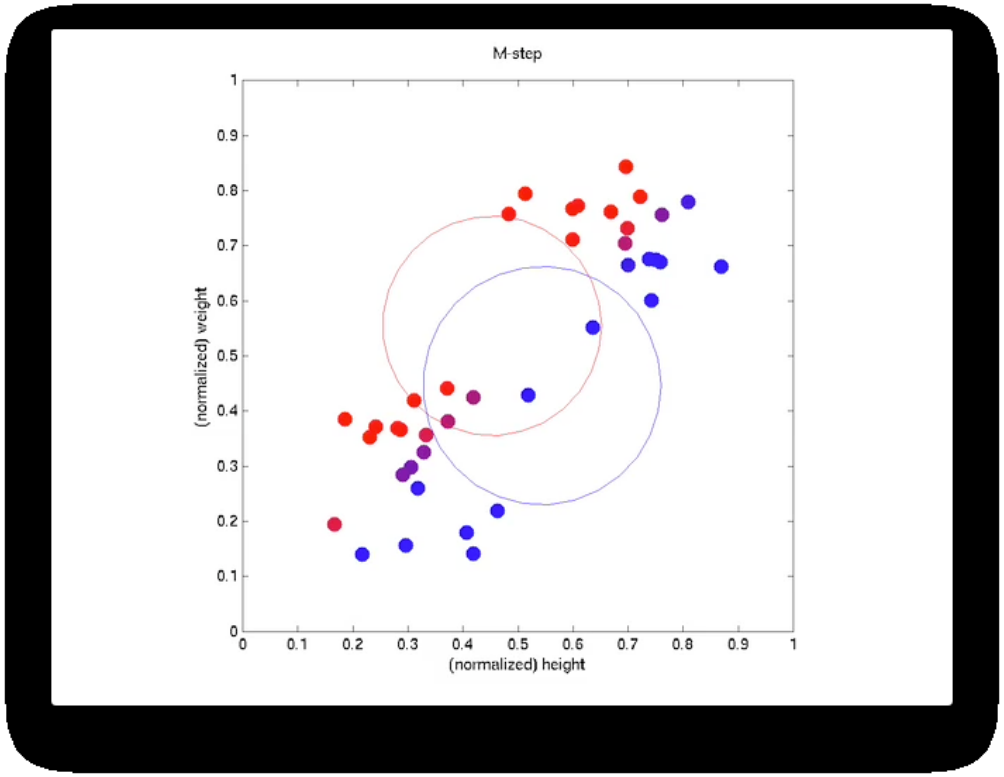
K-means clustering

K-means:
Choose number of clusters
Initialize cluster centers
While not converged:
Assign each point to closest cluster
Update cluster centers



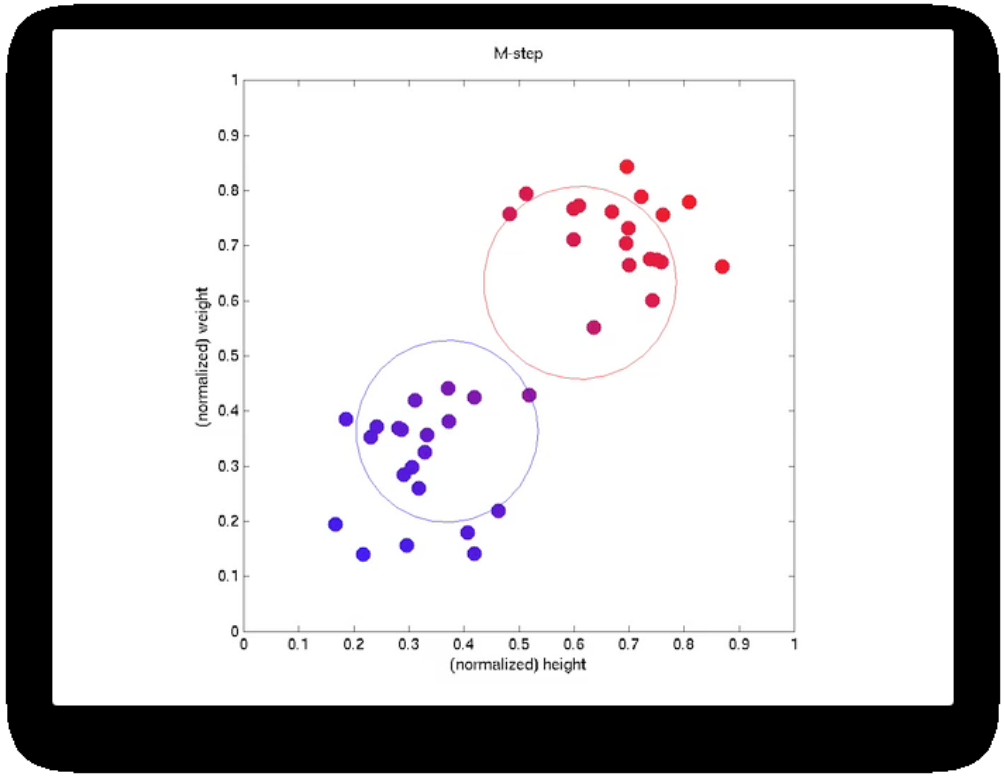
K-means clustering

K-means:
Choose number of clusters
Initialize cluster centers
While not converged:
    Assign each point to closest cluster
    Update cluster centers



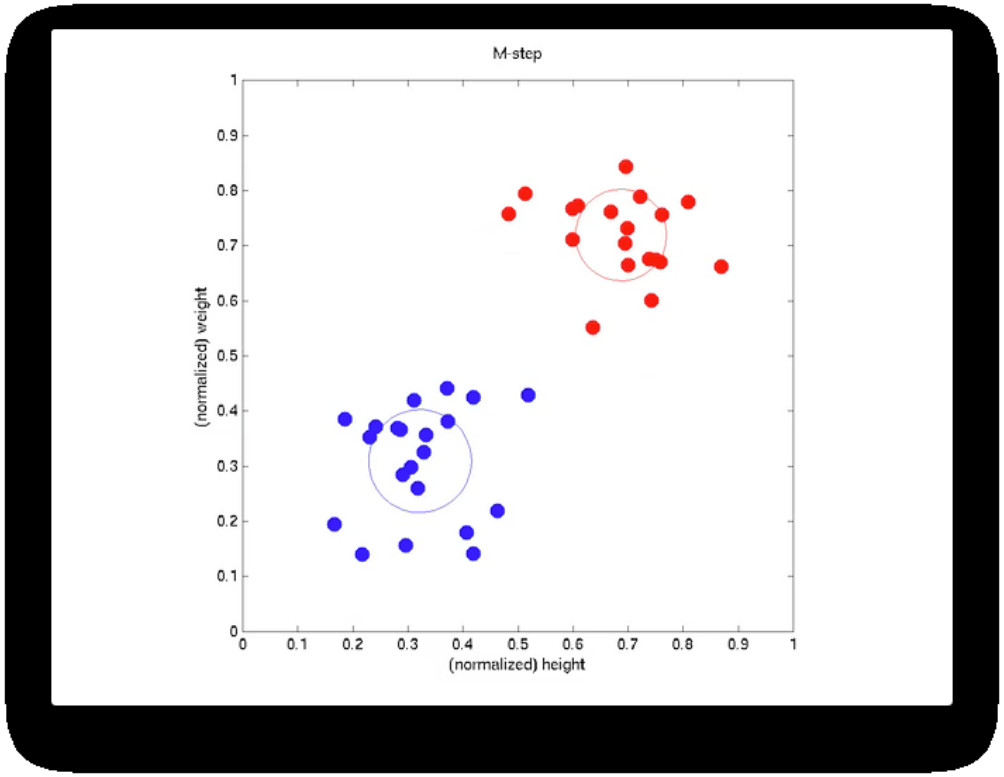
K-means clustering

K-means:
Choose number of clusters
Initialize cluster centers
While not converged:
    Assign each point to closest cluster
    Update cluster centers



K-means clustering

K-means:
Choose number of clusters
Initialize cluster centers
While not converged:
    Assign each point to closest cluster
    Update cluster centers



scipy.cluster.vq

Table Of Contents

K-means clustering and vector quantization (scipy.cluster.vq)

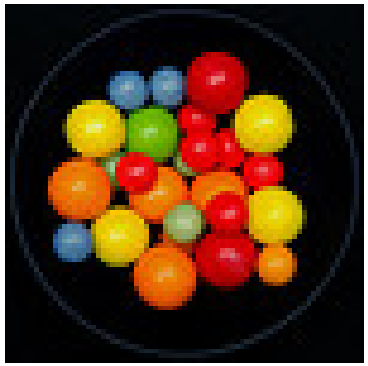
K-means Clustering and Vector Quantization Module

Provides routines for k-means clustering, generating code books from k-means models, and quantizing vectors by comparing them with centroids in a code book.

The k-means algorithm takes as input the number of clusters to generate, k, and a set of observation vectors to cluster. It returns a set of centroids, one for each of the k clusters. An observation vector is classified with the cluster number or centroid index of the centroid closest to it.

Clustering pixels

Find groups of similar pixels within a single image
(e.g. "the bright red circles")

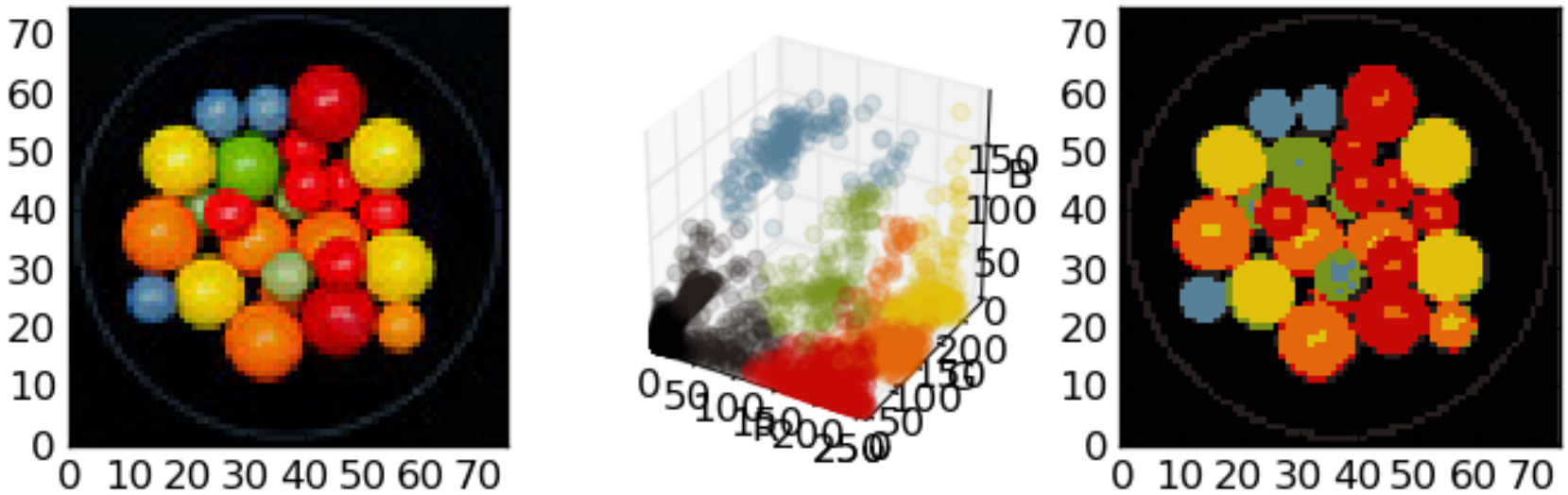


Represent each pixel as a separate example
with its (R,G,B) value as a 3-d feature vector

Clustering pixels

Group pixels within candy.jpg into 7 clusters

```
./cluster_pixels.py candy.jpg 7
```



Clustering images

Find groups of similar images within a collection of images
(e.g. "warm" photos)

Represent each image with a binned RGB intensity histogram

Clustering images

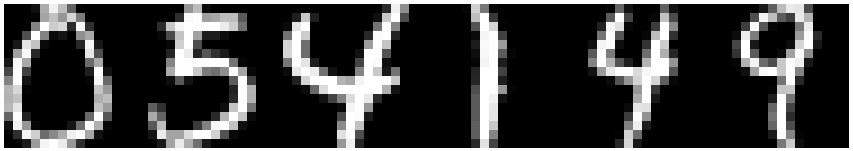
Group 'vivid' images into 3 clusters

```
./cluster_flickr.py flickr_vivid 7 10
```

Outline

Classifying images

Classification is an *supervised* learning task by which we aim to predict the correct label for an example given its features

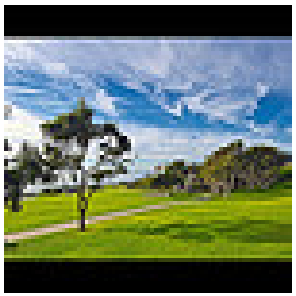


0 5 4 1 4 9

e.g. determine which digit $\{0, 1, \ldots, 9\}$ is in depicted in each image

Classifying images

Classification is an *supervised* learning task by which we aim to predict the correct label for an example given its features



↓

'landscape'

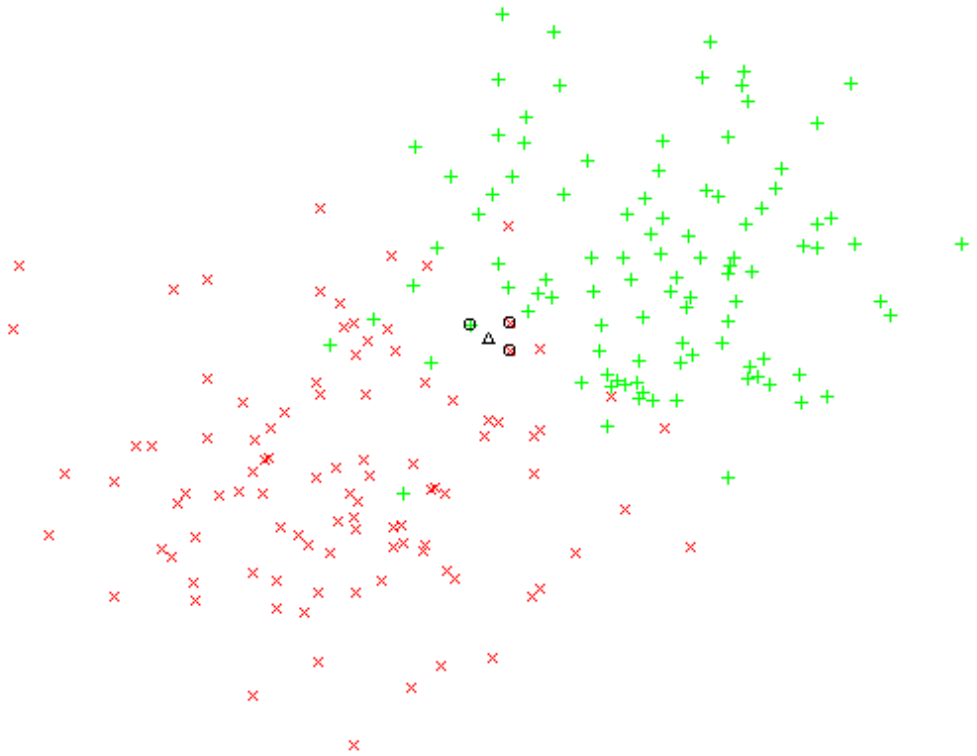
↓

'headshot'

e.g. determine if an image is a landscape or headshot

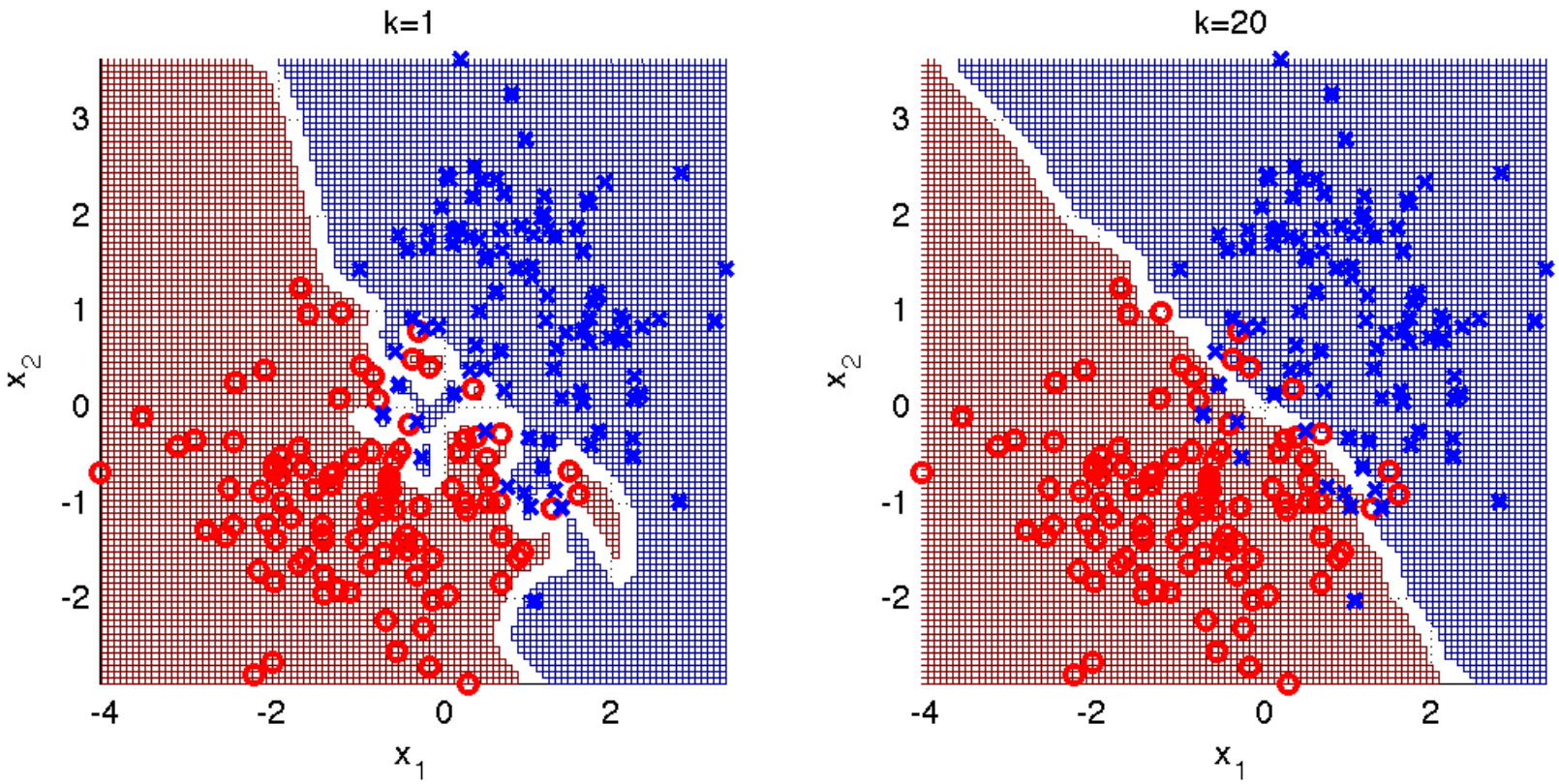
k-nearest neighbors classification

k-nearest neighbors: memorize training examples, predict labels using labels of the k closest training points



Intuition: nearby points have similar labels

k-nearest neighbors classification



Small k gives a **complex** boundary, **large k** results in **coarse** averaging

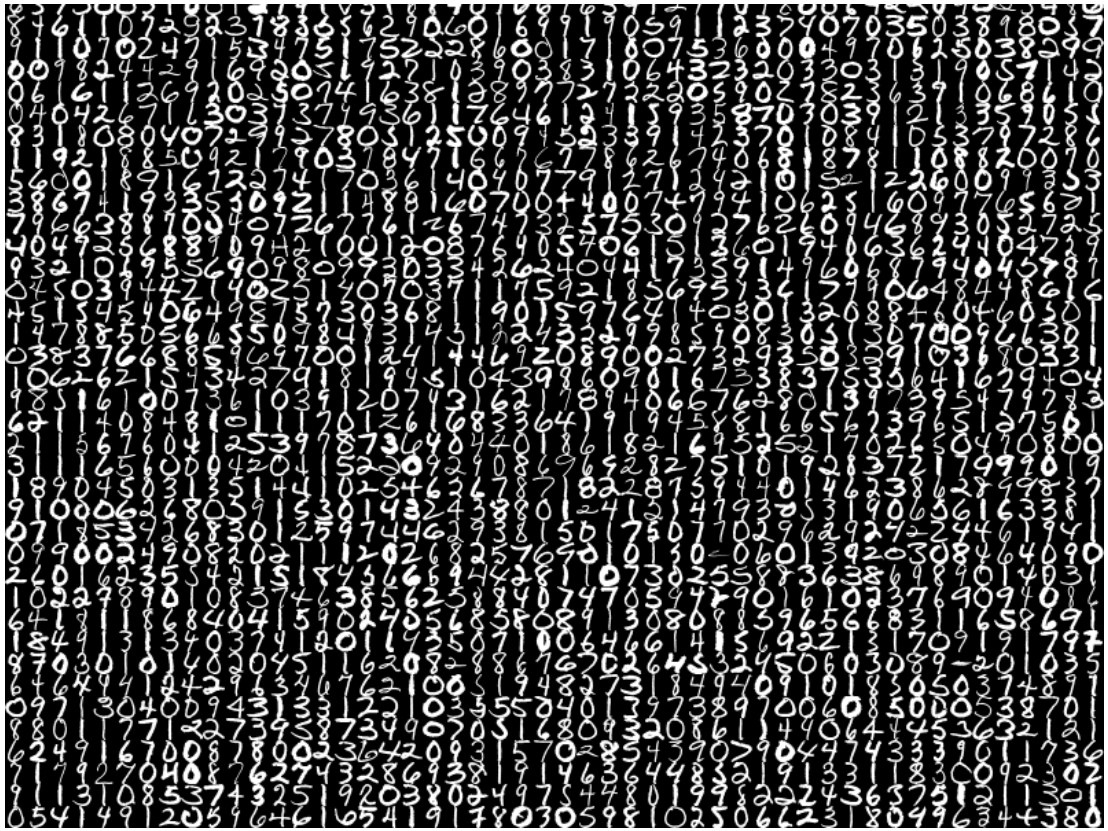
k-nearest neighbors classification

A simple kNN implementation with SciPy

```
import scipy.spatial as spat

# use the cdist function to quickly compute distances
# between all test and training examples
D = spat.distance.cdist(X, self.X, 'euclidean')

for i in range(N):
    # grab the indices of the k closest points
    ndx = D[i,:].argsort()[:k]

    # take a majority vote over the nearest points'
    # labels
    yhat[i] = mode(self.y[ndx])[0]
```

Digit recognition

Determine which digit $\{0, 1, \ldots, 9\}$ is in depicted in each image



Represent each image as a "bag of pixels", flattening the 2-d array of pixels to a 1-d vector

Digit recognition

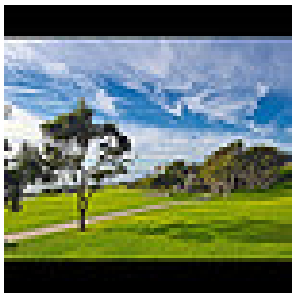
Simple digit classifer with k=1 nearest neighbors

```
./classify_digits.py
```

```
confusion matrix:
[[ 221.    0.    1.    0.    0.    2.    0.    0.    0.    0.]
 [   0.  202.    0.    0.    1.    0.    0.    1.    0.    0.]
 [   1.    0.  126.    0.    0.    0.    0.    2.    0.    0.]
 [   1.    0.    2.  121.    0.    3.    0.    0.    0.    0.]
 [   0.    0.    0.    0.  151.    0.    0.    0.    0.    4.]
 [   2.    1.    1.    4.    0.  102.    2.    0.    0.    0.]
 [   2.    0.    0.    0.    0.    0.  136.    0.    1.    0.]
 [   0.    1.    0.    0.    0.    1.    0.  120.    1.    1.]
 [   0.    1.    0.    3.    3.    0.    0.    1.  113.    1.]
 [   0.    0.    0.    0.    1.    0.    0.    4.    0.  118.]]
accuracy: 0.966
```

Image classification

Determine if an image is a landscape or headshot



↓

'landscape'

↓

'headshot'

Represent each image with a binned RGB intensity histogram

Image classification

Classify

```
./classify_flickr.py 16 9
   flickr_headshot flickr_landscape
```

Change in performance with number of neighbors

```
k = 1, accuracy = 0.7125
k = 3, accuracy = 0.7425
k = 5, accuracy = 0.7725
k = 7, accuracy = 0.7650
k = 9, accuracy = 0.7500
```