

Working with Image Data

Bootcamp Section 1

Joseph Adler, Drew Conway, Jake Hofman, Hilary Mason

February 1, 2011



Creative Commons Attribution-Share Alike 3.0

Acquiring image data

YAHOO! DEVELOPER NETWORK jake.hofman | Sign Out | Help

Home Documentation My Projects

Developer > APIs and Tools > Yahoo! Query Language > Console

YOUR YQL STATEMENT [permalink](#) [Create Query Alias](#)

```
select * from flickr.photos.search(100) where tags="kittens" and sort="interestingness-desc"
```

XML JSON cbfunc Diagnostics [TEST](#)

FORMATTED [TREE](#) Wrap Text [flickr.photos.search](#)

```
{
  "results": [
    {
      "farm": "3",
      "id": "4220856234",
      "isfamily": "0",
      "isfriend": "0",
      "ispublic": "1",
      "owner": "36587311@N08",
      "secret": "029e5b8348",
      "server": "2570",
      "title": "Cuba Gallery: Cat / evil / kitten / pets / animal / cute / photography"
    },
    {
      "farm": "4",
      ...
    }
  ]
}
```

THE REST QUERY [How do I use this?](#) [hide](#)

```
http://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20flickr.photos.search(100)%20where%20tags%3D%22kitte
```

EXAMPLE QUERIES

- get my friends sorted by nickname
- get 10 flickr "cat" photos
- get a flickr photo by photo ID
- get the flickr image url by photo ID
- get san francisco geo data
- get san francisco woeid
- get geo details about san francisco

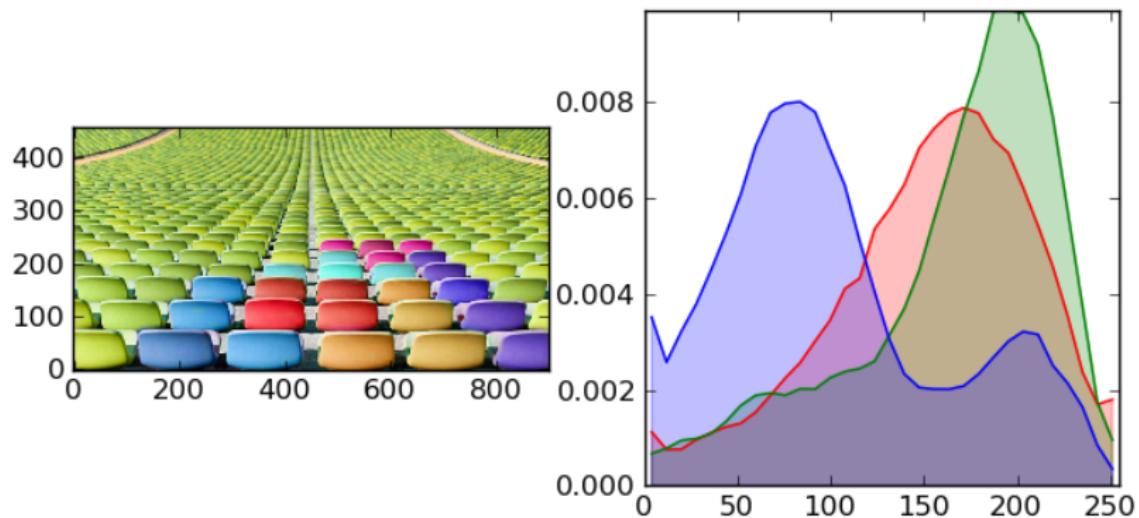
DATA TABLES (162)

Show Community Tables [What's this?](#)

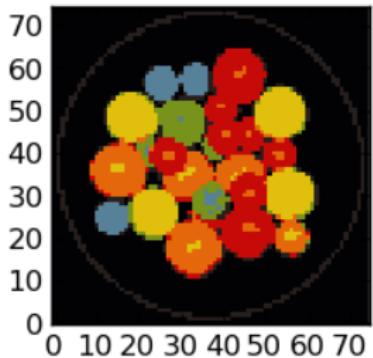
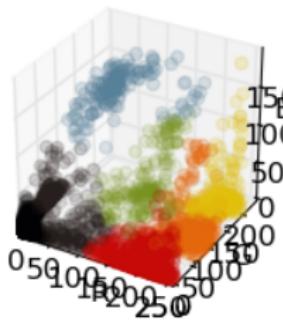
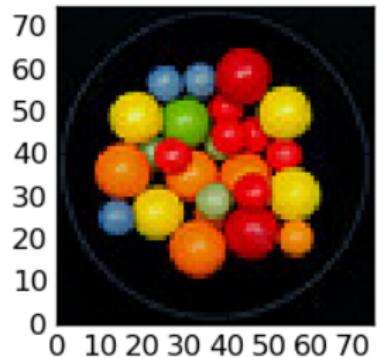
[Filter Tables](#)

- flickr.people.info2
- flickr.people.publicphotos
- flickr.photos.exif
- flickr.photos.info
- flickr.photos.interestingness
- flickr.photos.recent
- flickr.photos.search
- flickr.photos.size
- flickr.photosets.info

Image features



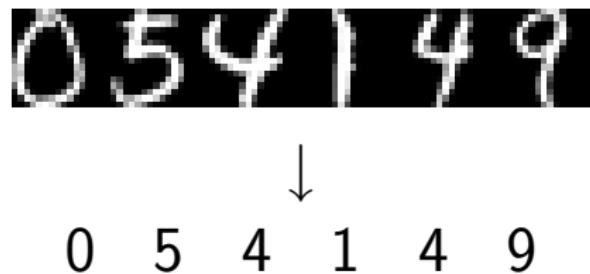
Clustering pixels



Clustering images



Classifying images



Outline

1 Acquiring image data

2 Image features

3 Clustering

4 Classification

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® from YAHOO!

You aren't signed in [Sign In](#) [Help](#)

Home The Tour Sign Up Explore | [Upload](#)

Search | ▾

Explore / Interestingness / January 2011

[◀ December 2010](#) [January 2011](#) [▶ February 2011](#)

SUN	MON	TUE	WED	THU	FRI	SAT
						
						
						
						
						
						



[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 Internet¹

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

YAHOO! DEVELOPER NETWORK jake.hofman | Sign Out | Help

Home Documentation My Projects

Developer > APIs and Tools > Yahoo Query Language > Console

YOUR YQL STATEMENT [permalink](#) [Create Query Alias](#)

```
select * from flickr.photos.search(100) where tags="kittens" and sort="interestingness-desc"
```

XML JSON cbfunc Diagnostics **TEST**

FORMATTED **TREE** Wrap Text **flickr.photos.search**

```
},
  "results": [
    "photo": [
      {
        "farm": "3",
        "id": "4220856234",
        "isfamily": "0",
        "isfriend": "0",
        "ispublic": "1",
        "owner": "36587311@N08",
        "secret": "029e5b8348",
        "server": "2570",
        "title": "Cuba Gallery: Cat / evil / kitten / pets / animal / cute / photography"
      },
      {
        "farm": "4".
```

THE REST QUERY [How do I use this?](#) [hide](#)

```
http://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20flickr.photos.search(100)%20where%20tags%3D%22kitten%22&sort=interestingness-desc
```

EXAMPLE QUERIES

- get my friends sorted by nickname
- get 10 flickr "cat" photos
- get a flickr photo by photo ID
- get the flickr image url by photo ID
- get san francisco geo data
- get san francisco woeid
- get geo details about san francisco

DATA TABLES (162)

Show Community Tables [What's this?](#)

[Filter Tables](#)

- flickr.people.info2
- flickr.people.publicphotos
- flickr.photos.exif
- flickr.photos.info
- flickr.photos.interestingness
- flickr.photos.recent
- flickr.photos.search
- flickr.photos.sizes
- flickr.photosets.info

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

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']
```

²See <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

1 Acquiring image data

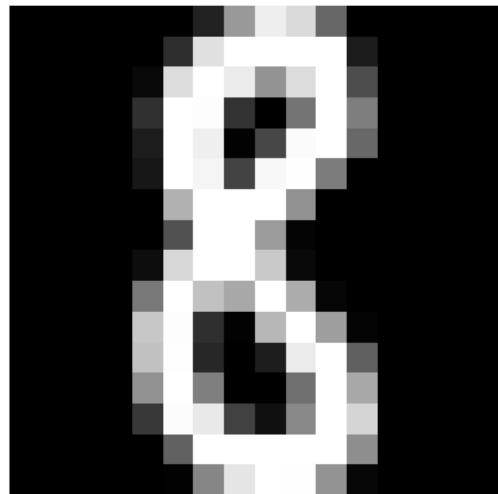
2 Image features

3 Clustering

4 Classification

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

```
array([[ -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. ,
       -0.643,  0.625,  0.815, -0.533, -1. , -1. , -1. , -1. , -1. , -1. ],
      [-1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -0.441,
       0.951,  1. ,  0.902, -0.111, -1. , -1. , -1. , -1. , -1. , -1. ],
      [-1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -0.724,  0.776,
        1. ,  0.622, -0.736, -1. , -1. , -1. , -1. , -1. , -1. , -1. ],
      [-1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. ,  0.389,  1. ,
        0.739, -0.908, -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. ],
      [-1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -0.387,  0.985,
       0.943, -0.598, -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. ],
      [-1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -0.824,  0.728,
        1. ,  -0.044, -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. ],
      [-1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. ,  0.364,  1. ,
        0.54,  -0.972, -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. ],
      [-1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -0.441,  0.999,
       0.854, -0.675, -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. ],
      [-1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -0.925,  0.8 ,
       0.995, -0.121, -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. ],
      [-1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -0.495,  0.999,
       0.782, -0.99, -0.852, -0.172,  0.208, -0.038, -0.516, -1. ,
       -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. , -1. ],
      [-1. , -1. , -0.179,  1. ,  0.125, -0.242,  0.773,  1. ,
        1. ,  1. ,  0.998,  0.534, -0.628, -1. , -1. , -1. , -1. ],
      [-1. , -1. , -0.339,  1. ,  0.756,  0.993,  0.99,  0.571,
       -0.05, -0.317, -0.106,  0.666,  0.845, -0.699, -1. , -1. ],
      [-1. , -1. , -0.773,  0.945,  1. ,  0.858, -0.304, -1. ,
       -1. , -1. , -0.967,  0.327,  1. ,  -0.079, -1. , -1. , -1. ],
      [-1. , -1. , -1. ,  0.418,  1. ,  -0.126, -0.995, -1. ,
       -0.986, -0.57,  0.494,  1. ,  0.957, -0.737, -1. , -1. , -1. ],
      [-1. , -1. , -1. , -0.752,  0.297,  0.995,  0.698,  0.646,
       0.724,  1. ,  1. ,  0.912, -0.358, -1. , -1. , -1. , -1. ],
      [-1. , -1. , -1. , -1. , -1. , -0.607,  0.351,  0.585,
       0.841,  0.595,  0.312, -0.797, -1. , -1. , -1. , -1. ]])
```

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]],

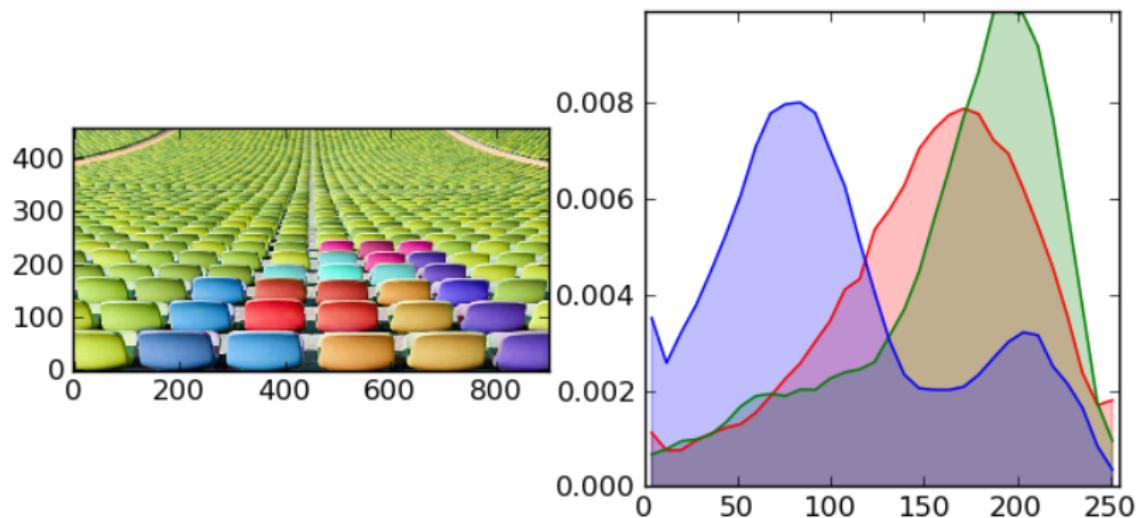
      [[124, 131,  87],
       [121, 129,  78],
       [116, 126,  63],
       ...,
       [ 31,   31,  31],
       [ 31,   31,  31],
       [ 31,   31,  31]],

      ...,
      [[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')
```

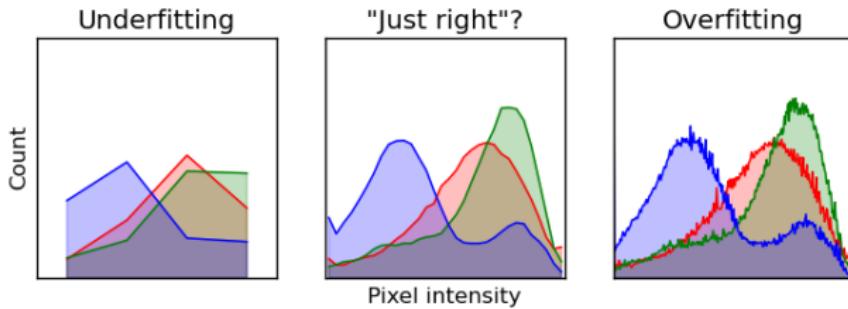
Intensity histograms

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

1 Acquiring image data

2 Image features

3 Clustering

4 Classification

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

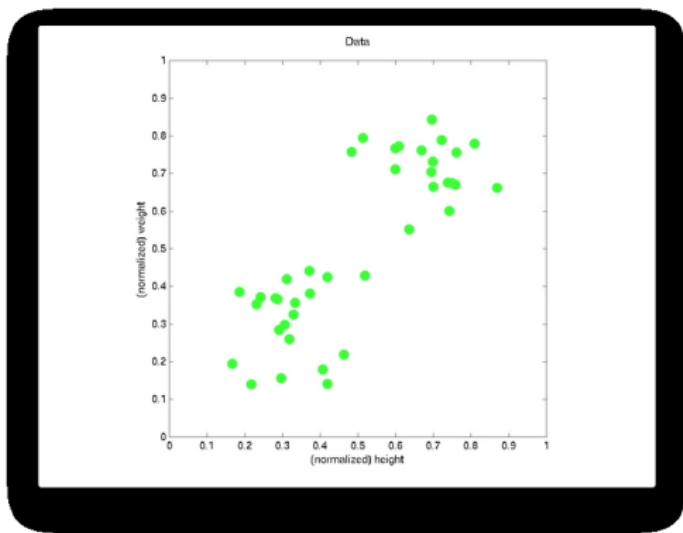
Clustering is an *unsupervised* learning task by which we look for structure in the data, grouping similar examples together



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 assignments

K-means clustering

K-means:

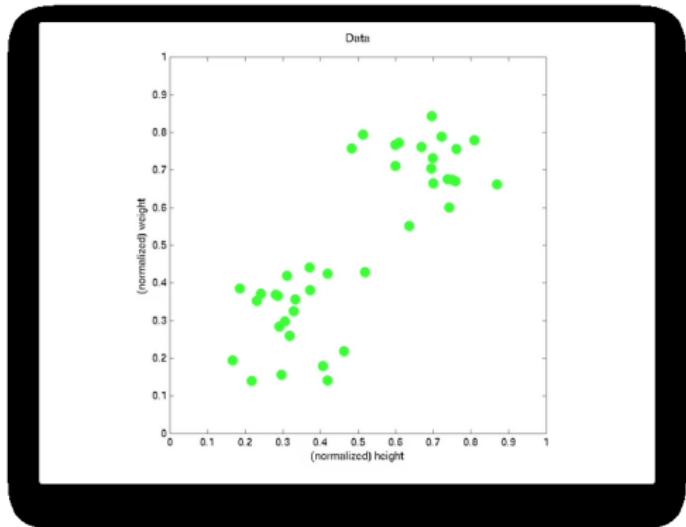
Choose number of clusters

Initialize cluster centers

While not converged:

 Assign each point to closest cluster

 Update cluster centers



K-means clustering

K-means:

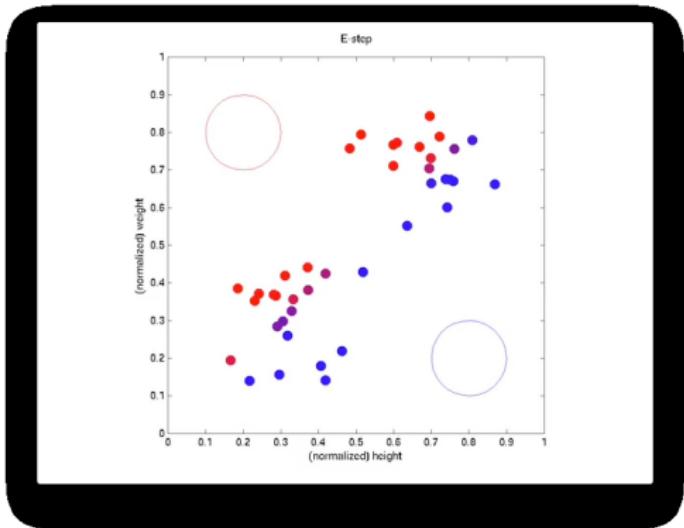
Choose number of clusters

Initialize cluster centers

While not converged:

Assign each point to closest cluster

Update cluster centers



K-means clustering

K-means:

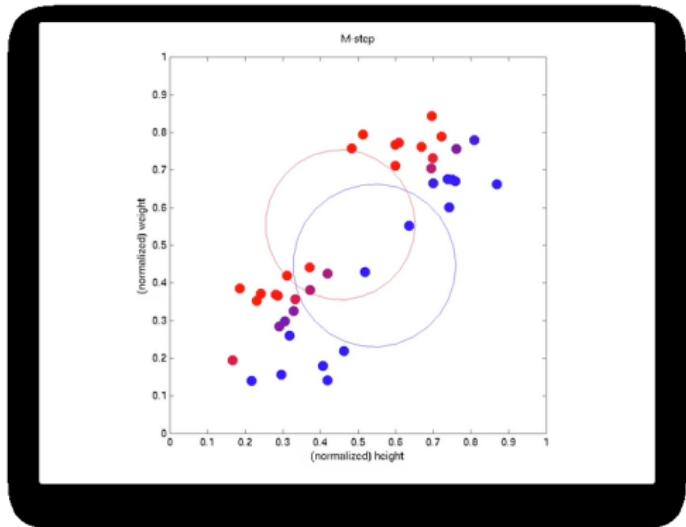
Choose number of clusters

Initialize cluster centers

While not converged:

 Assign each point to closest cluster

 Update cluster centers



K-means clustering

K-means:

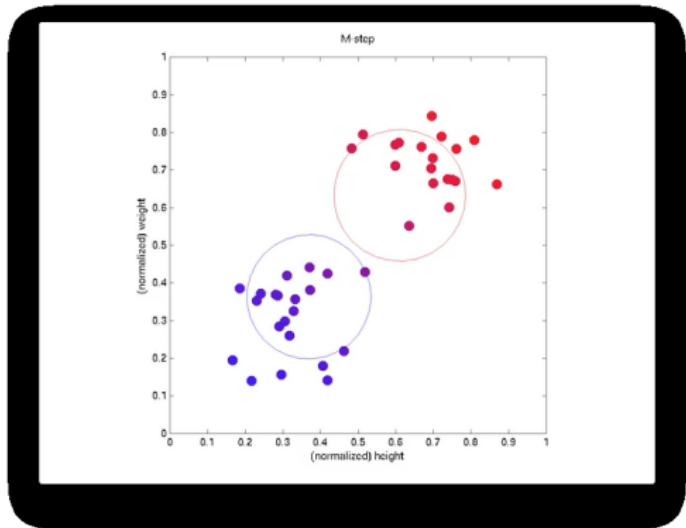
Choose number of clusters

Initialize cluster centers

While not converged:

 Assign each point to closest cluster

 Update cluster centers



K-means clustering

K-means:

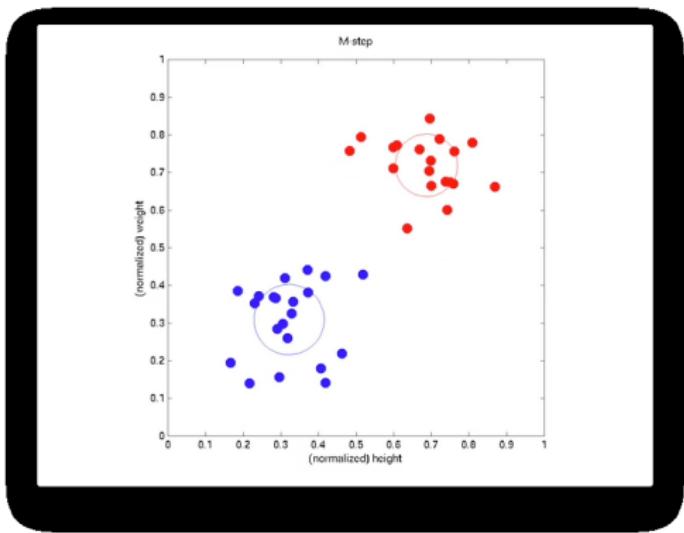
Choose number of clusters

Initialize cluster centers

While not converged:

 Assign each point to closest cluster

 Update cluster centers





[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”)

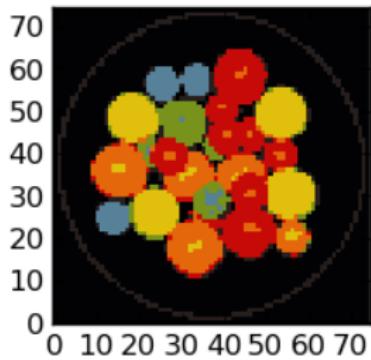
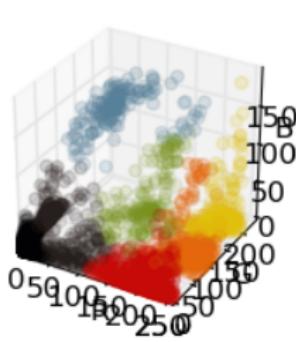
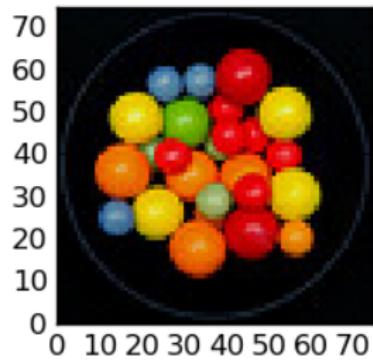


Represent each pixel as a separate example
with its (R,G,B) value as a 3-d feature vector

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

1 Acquiring image data

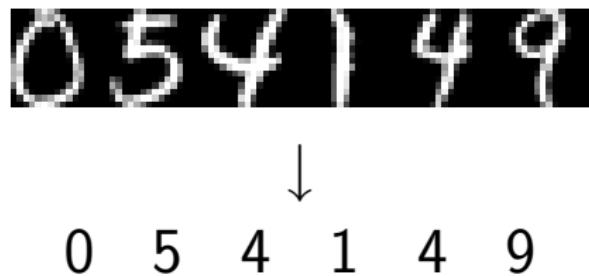
2 Image features

3 Clustering

4 Classification

Classifying images

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



e.g. determine which digit $\{0, 1, \dots, 9\}$ 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'

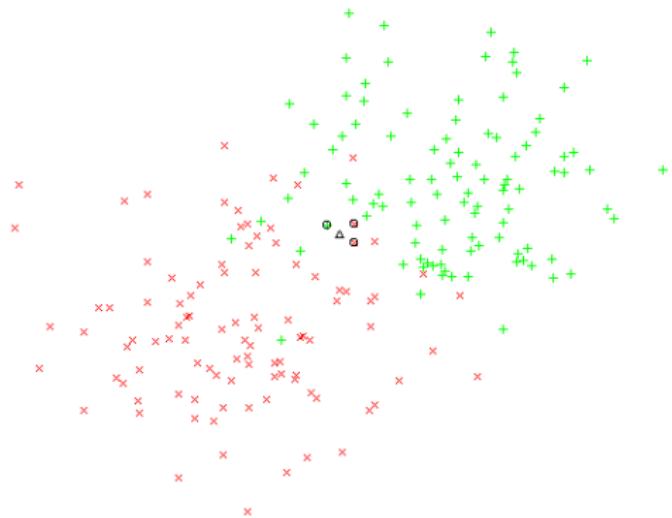


↓
'headshot'

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

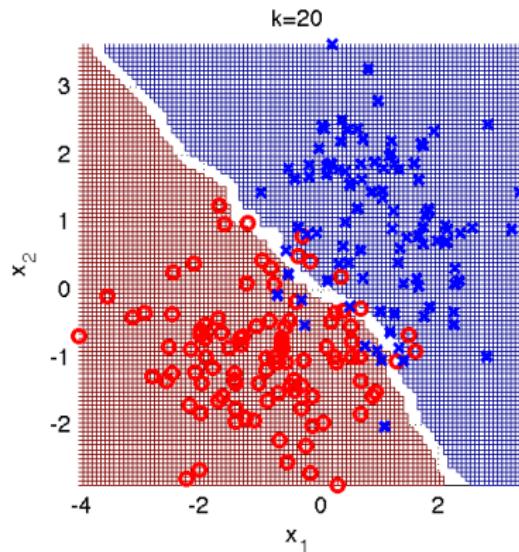
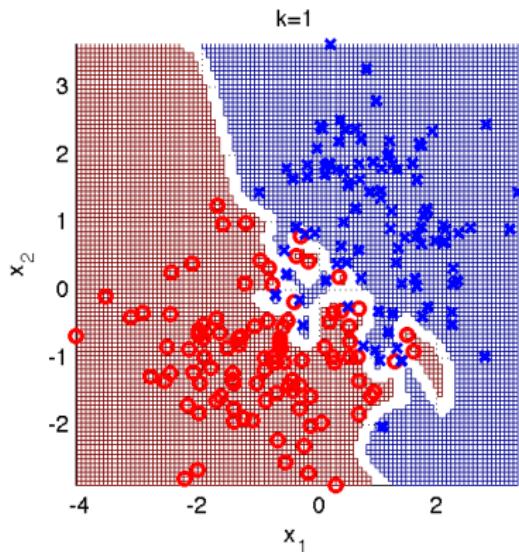
k-nearest neighbors classification

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

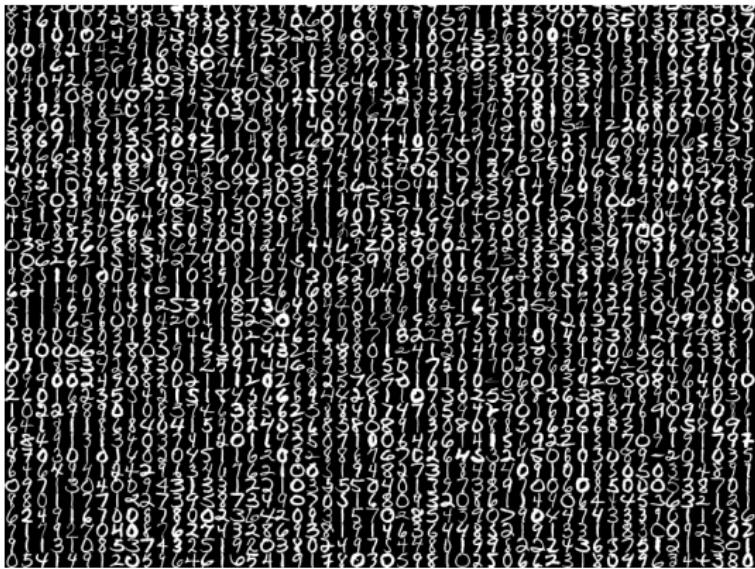
# use the cdist function to quickly compute distances
# between all test and training examples
D = spat.distance.cdist(X, self.X, 'euclidean')

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 $\{0, 1, \dots, 9\}$ 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 classifier 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
```