Problem Set 4

Boris Ivanovic
CS 231A

May 26, 2017

Outline

Part 1: Facial Detection via HoG Features + SVM Classifier

Part 2: Image Segmentation with K-Means and Meanshift

Part 1(a): Facial Detection via HoG Features + SVM Classifier

Main Idea

We have a “good” facial detector (ie. classifier) which operates on HoG features of small images and we want to apply it to larger images from the real world.

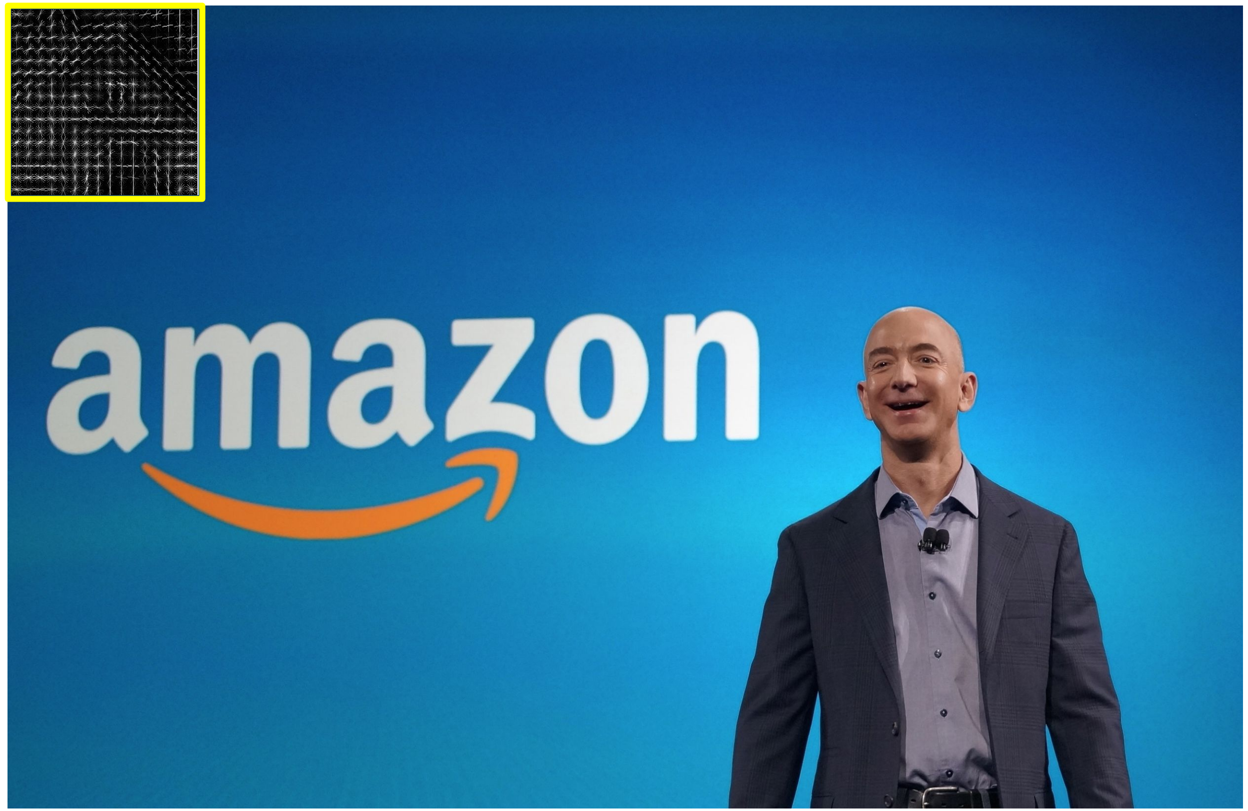
How do we do this? **In a sliding window fashion!**

We slide a window over the image and…

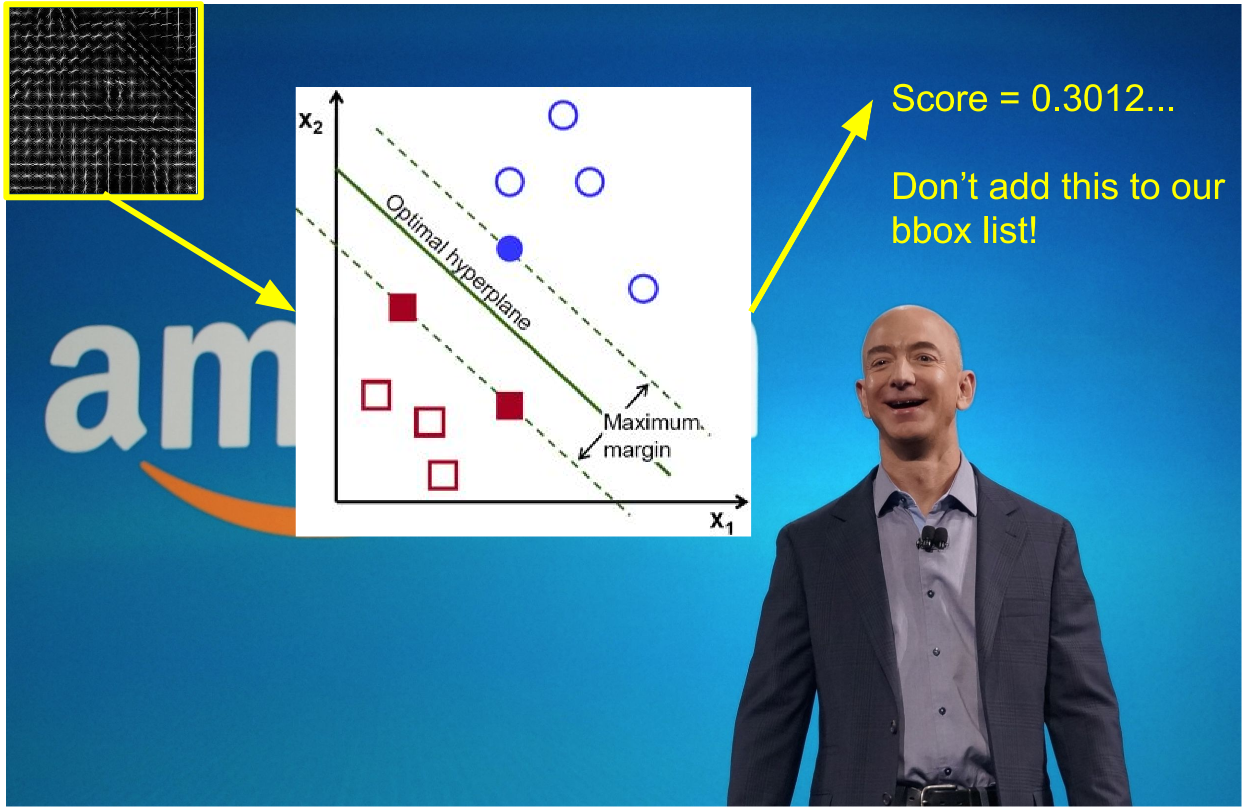
- Compute the HoG features of the window,
- Calculate the classifier score on those features
    - If the score is > the threshold we add the current window to our bounding box list.

Pictorally this looks like...

Pictorally this looks like...



Pictorally this looks like...



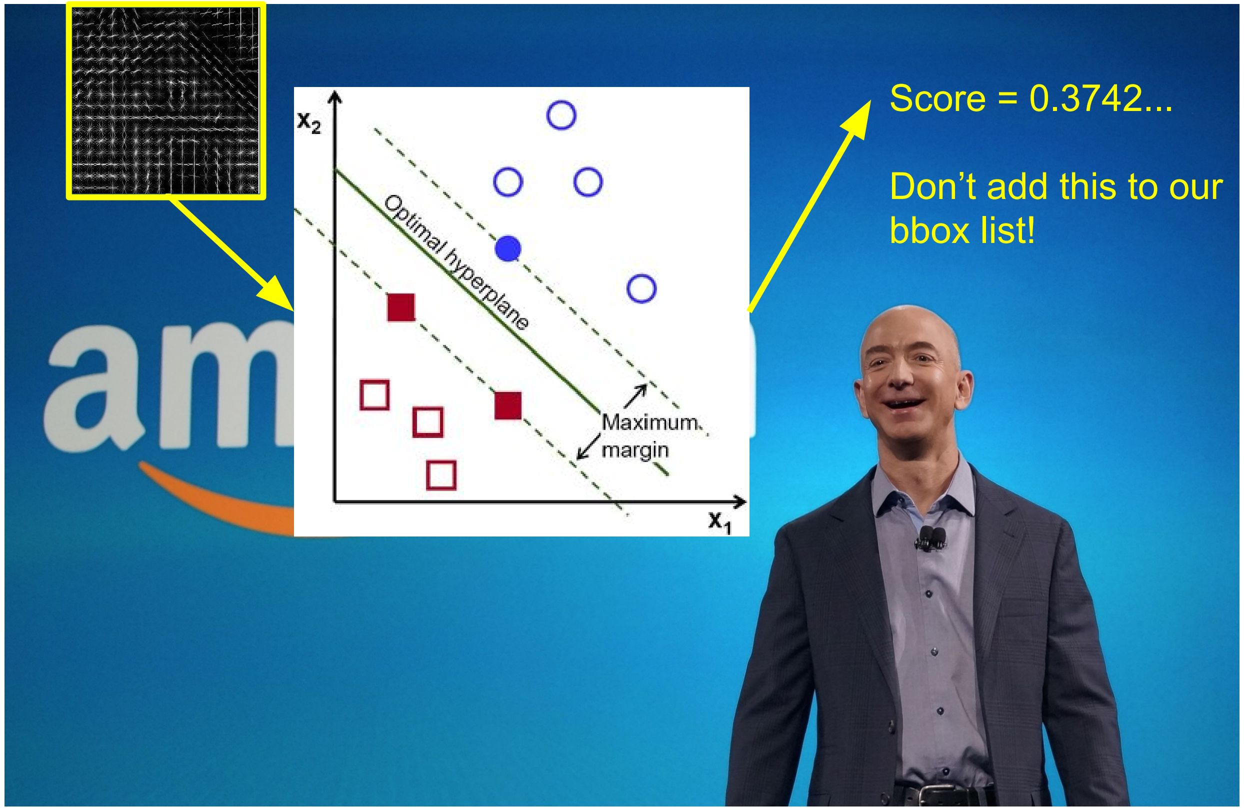
Score = 0.3012...

Don’t add this to our bbox list!

Pictorally this looks like...

Pictorally this looks like...

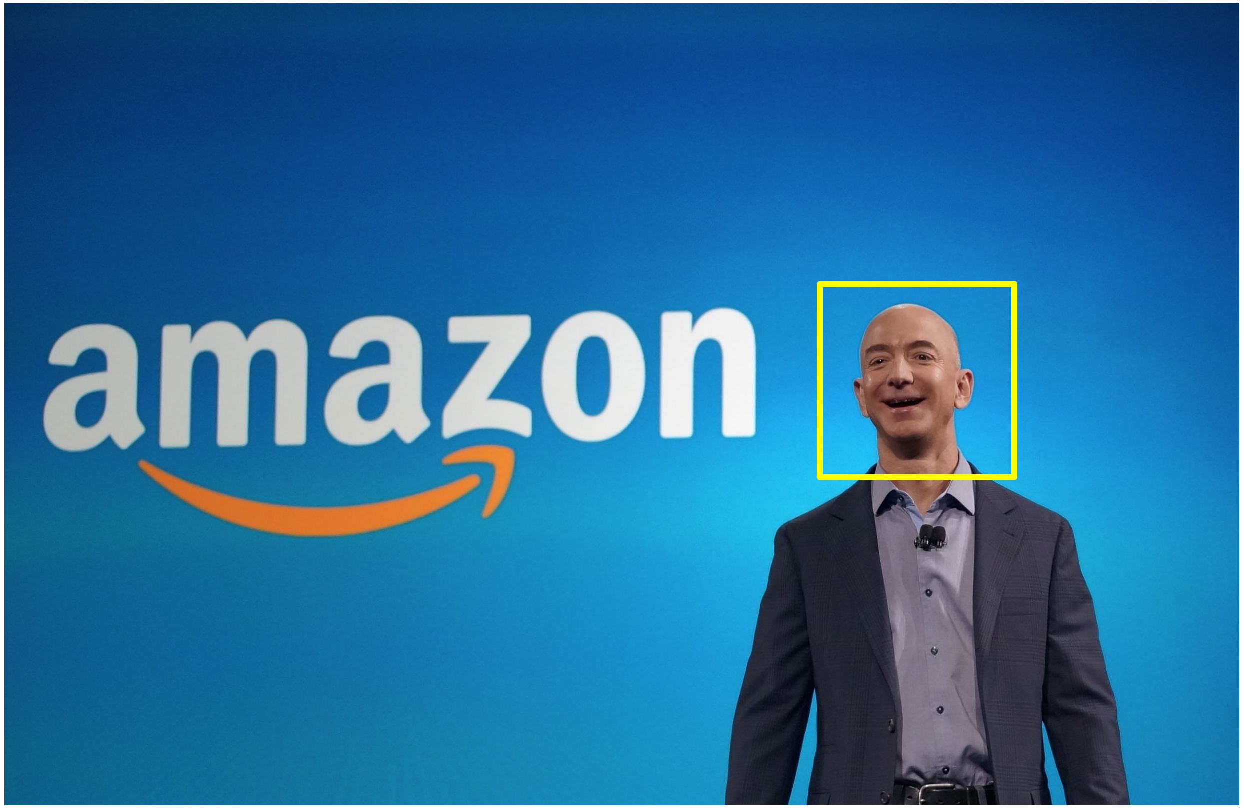
Pictorally this looks like...



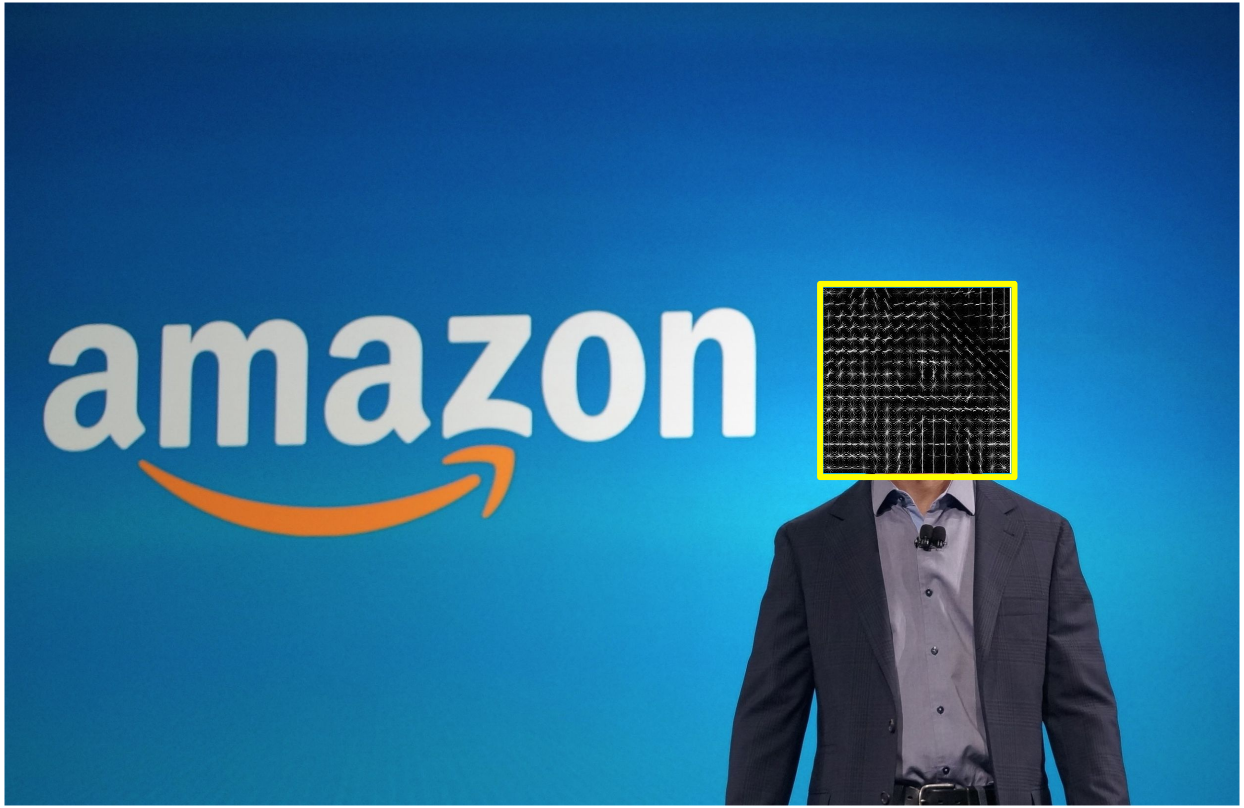
Pictorally this looks like...

...

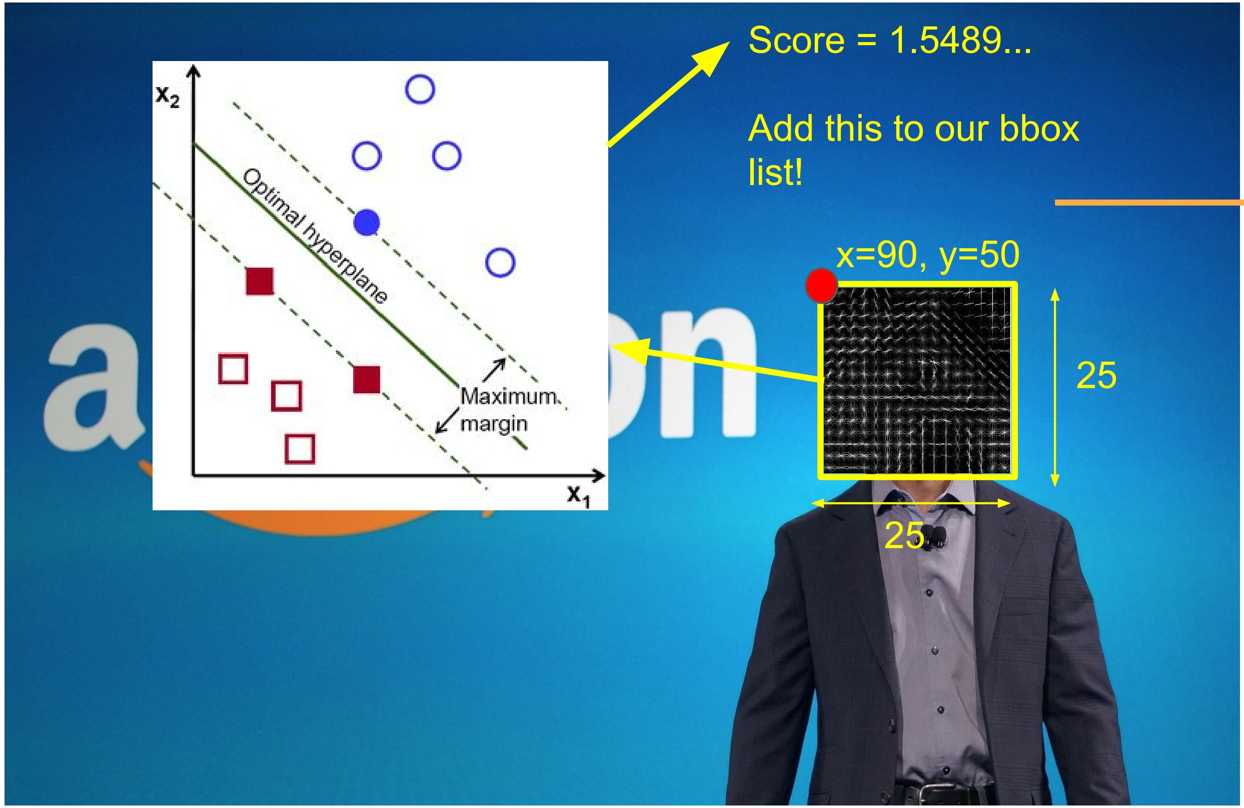
Pictorally this looks like...



Pictorally this looks like...



Pictorally this looks like...



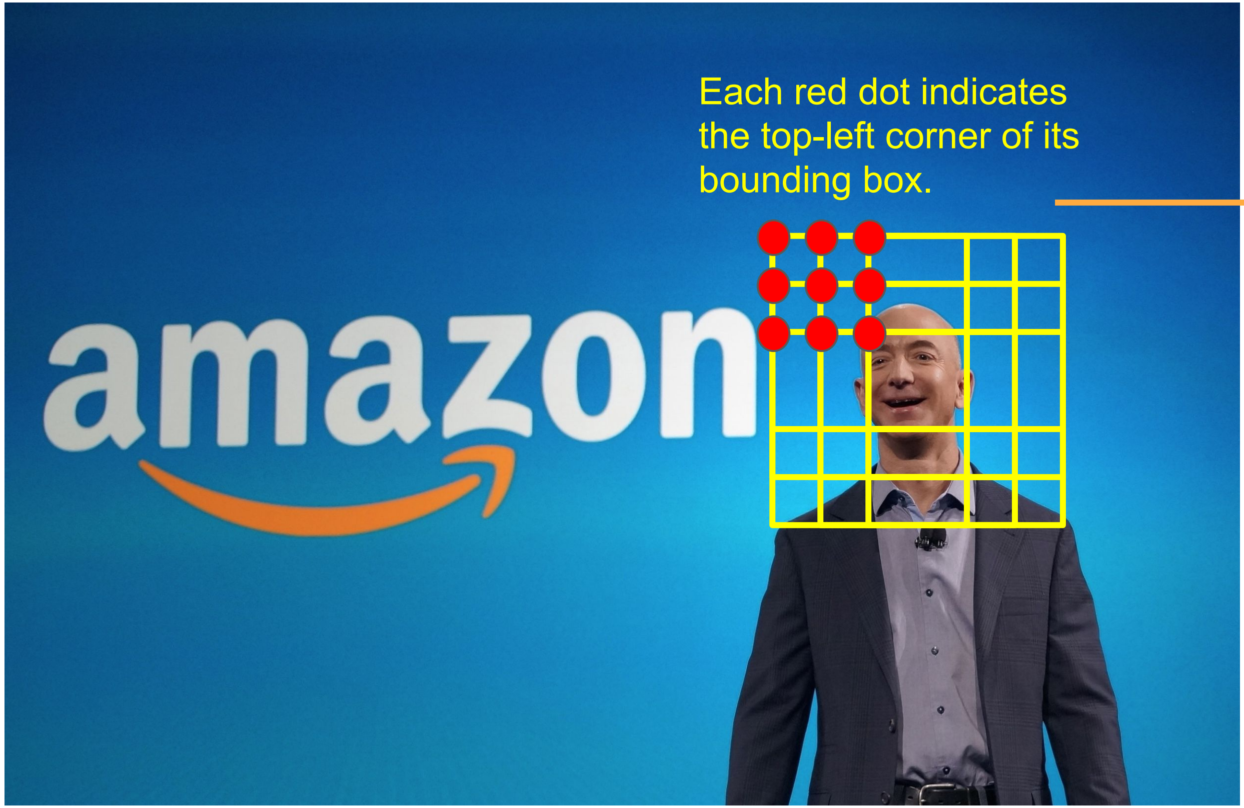
<u>Bounding Boxes</u>

1) [90, 50, 25, 25]

<u>Scores</u>

1) 1.5489...

Pictorally this looks like...


Each red dot indicates the top-left corner of its bounding box.

<u>Bounding Boxes</u>

1)	[86, 46, 25, 25]
2)	[90, 46, 25, 25]
3)	[94, 46, 25, 25]
4)	[86, 50, 25, 25]
5)	[90, 50, 25, 25]
6)	[94, 50, 25, 25]
7)	[86, 54, 25, 25]
8)	[90, 54, 25, 25]
9)	[94, 54, 25, 25]

<u>Scores</u>

1)	1.2348...
2)	1.2837...
3)	1.2352...
4)	1.4302...
5)	1.5489...
6)	1.4319...
7)	1.2397...
8)	1.3092...
9)	1.2345...

Part 1(b): Non-Maxima Suppression

Sadly, we have way too many bounding boxes.

This is a consequence of us having a good facial detector, it works even if parts of the face aren’t visible in the window.

To fix this, we’ll perform a post-processing step: **Non-maxima Suppression**

What we do is:

- Sort the bounding boxes by their classifier score (in reverse order!!)
- Loop through this reversely-sorted list and add the bbox to a final list if:
    - The center of the current bbox is NOT within any other bbox in the final list.

Pictorally this looks like...

Bounding Boxes

1) [86, 46, 25, 25]
2) [90, 46, 25, 25]
3) [94, 46, 25, 25]
4) [86, 50, 25, 25]
5) [90, 50, 25, 25]
6) [94, 50, 25, 25]
7) [86, 54, 25, 25]
8) [90, 54, 25, 25]
9) [94, 54, 25, 25]

Final List

Scores

1) 1.2348...
2) 1.2837...
3) 1.2352...
4) 1.4302...
5) 1.5489...
6) 1.4319...
7) 1.2397...
8) 1.3092...
9) 1.2345...

Pictorally this looks like...

Bounding Boxes

1) [90, 50, 25, 25]
2) [94, 50, 25, 25]
3) [86, 50, 25, 25]
4) [90, 54, 25, 25]
5) [90, 46, 25, 25]
6) [86, 54, 25, 25]
7) [94, 46, 25, 25]
8) [86, 46, 25, 25]
9) [94, 54, 25, 25]

Scores

1) 1.5489…
2) 1.4319…
3) 1.4302…
4) 1.3092…
5) 1.2837…
6) 1.2397…
7) 1.2352...
8) 1.2348...
9) 1.2345...

Final List

Reverse Sort

Pictorally this looks like...

Bounding Boxes

1) [90, 50, 25, 25]
2) [94, 50, 25, 25]
3) [86, 50, 25, 25]
4) [90, 54, 25, 25]
5) [90, 46, 25, 25]
6) [86, 54, 25, 25]
7) [94, 46, 25, 25]
8) [86, 46, 25, 25]
9) [94, 54, 25, 25]

Final List

1) [90, 50, 25, 25]

Scores

1) 1.5489…
2) 1.4319…
3) 1.4302…
4) 1.3092…
5) 1.2837…
6) 1.2397…
7) 1.2352...
8) 1.2348...
9) 1.2345...

Is bbox 1 within any bbox in the final list?

No. Add it to the final list.

Pictorally this looks like...

<u>Bounding Boxes</u>

1) [90, 50, 25, 25] → <u>Final List</u> 1) [90, 50, 25, 25]

→ 2) [94, 50, 25, 25] X
3) [86, 50, 25, 25]
4) [90, 54, 25, 25]
5) [90, 46, 25, 25]
6) [86, 54, 25, 25]
7) [94, 46, 25, 25]
8) [86, 46, 25, 25]
9) [94, 54, 25, 25]

<u>Scores</u>

1) 1.5489…
2) 1.4319…
3) 1.4302…
4) 1.3092…
5) 1.2837…
6) 1.2397…
7) 1.2352...
8) 1.2348...
9) 1.2345...

Is bbox 2 within any bbox in the final list?

Yes. Don’t add it to the final list.

How do we calculate this? Exercise for you to decide!

Pictorally this looks like...

Bounding Boxes

1)	[90, 50, 25, 25]	→
2)	[94, 50, 25, 25]	X
3)	[86, 50, 25, 25]	X
4)	[90, 54, 25, 25]	X
5)	[90, 46, 25, 25]	X
6)	[86, 54, 25, 25]	X
7)	[94, 46, 25, 25]	X
8)	[86, 46, 25, 25]	X
→ 9)	[94, 54, 25, 25]	X

Final List

1)	[90, 50, 25, 25]

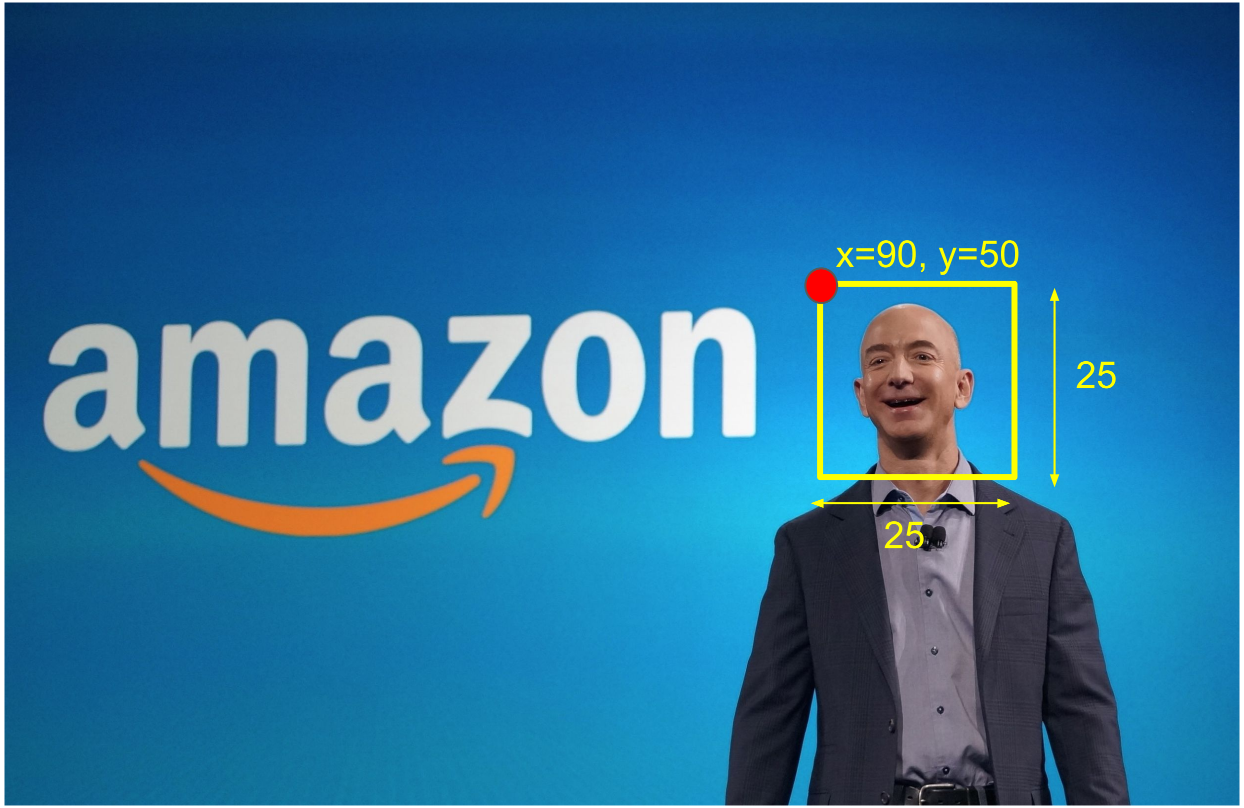
Scores

1)	1.5489…
2)	1.4319…
3)	1.4302…
4)	1.3092…
5)	1.2837…
6)	1.2397…
7)	1.2352...
8)	1.2348...
9)	1.2345...

Is bbox 9 within any bbox in the final list?

Yes. Don’t add it to the final list.

Pictorally this looks like...



<u>Final List</u>

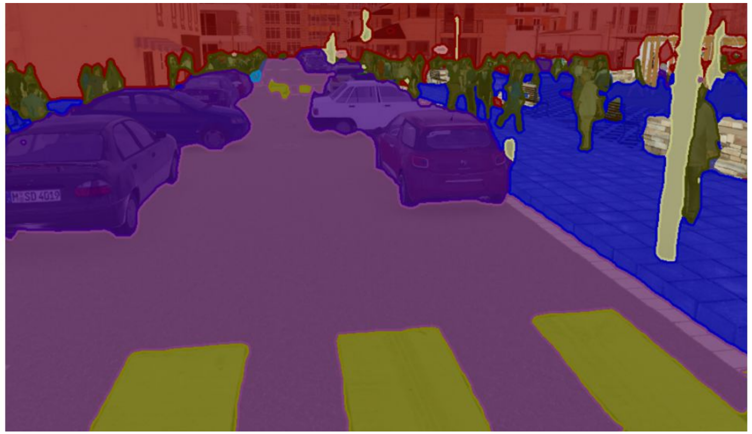
1) [90, 50, 25, 25]

Questions for Part 1?

Part 2(a): Image Segmentation with K-Means

Main Idea

Segment an image into its semantic components, specifically using K-Means.



Hold up, what’s K-Means? We haven’t seen this yet!

You’re right, and you actually won’t see K-Means in this class, hence this session.

For part (a), you’ll be implementing the K-Means algorithm.

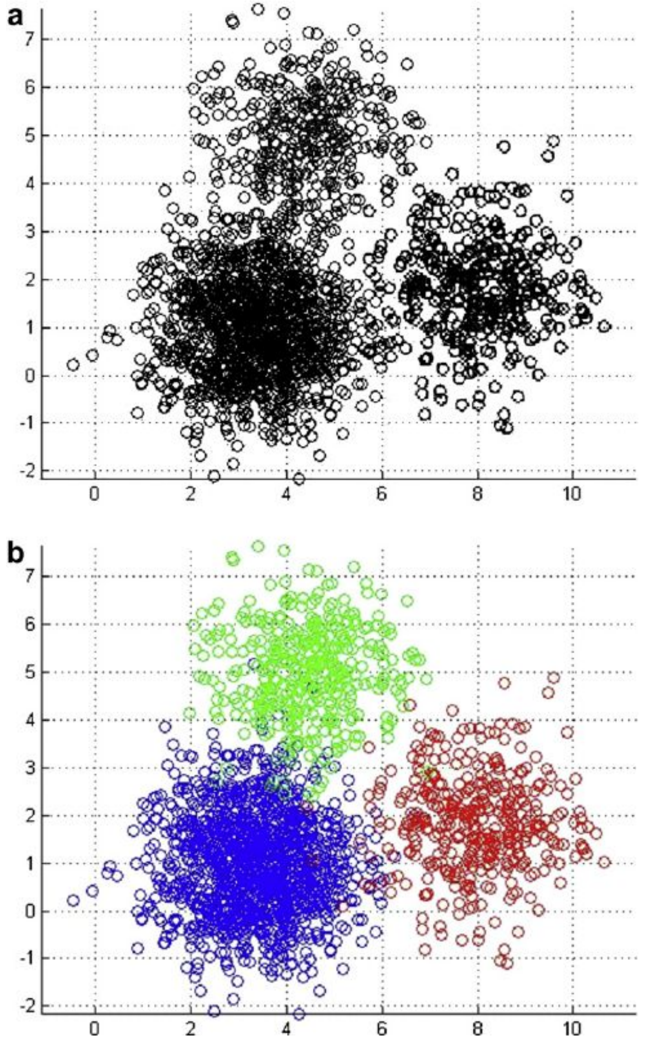
K-Means Clustering

“Unsupervised Learning”

A technique to cluster data for which you have no labels.

For us: A method of grouping together “like” features in feature space.

Called “K”-means because there’s K clusters (a hyperparameter that you choose before running the algorithm).



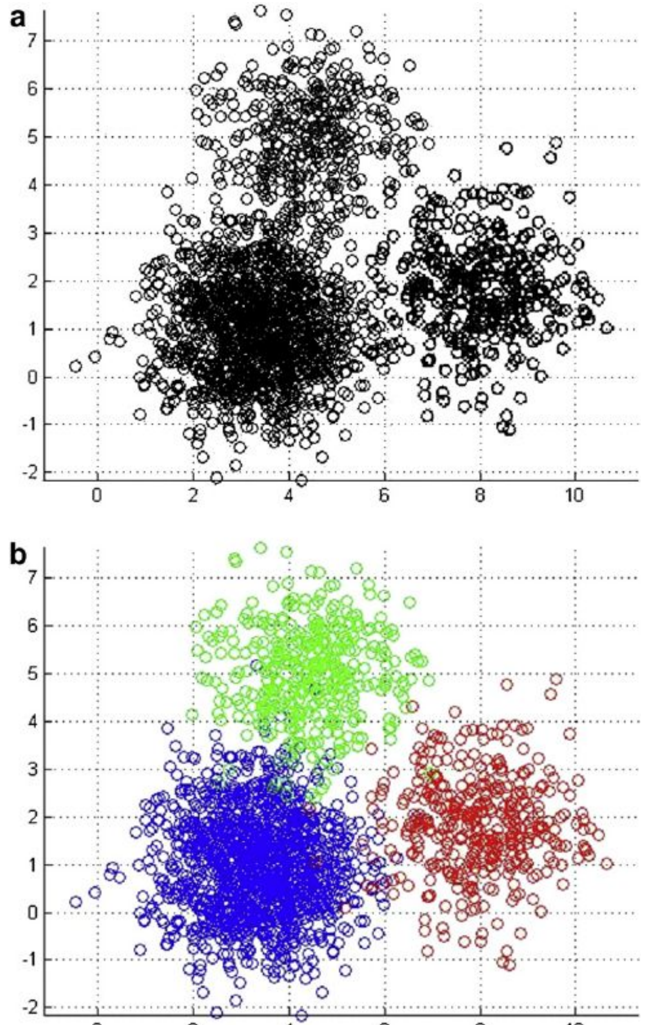
K-Means Clustering

<u>Algorithm:</u>

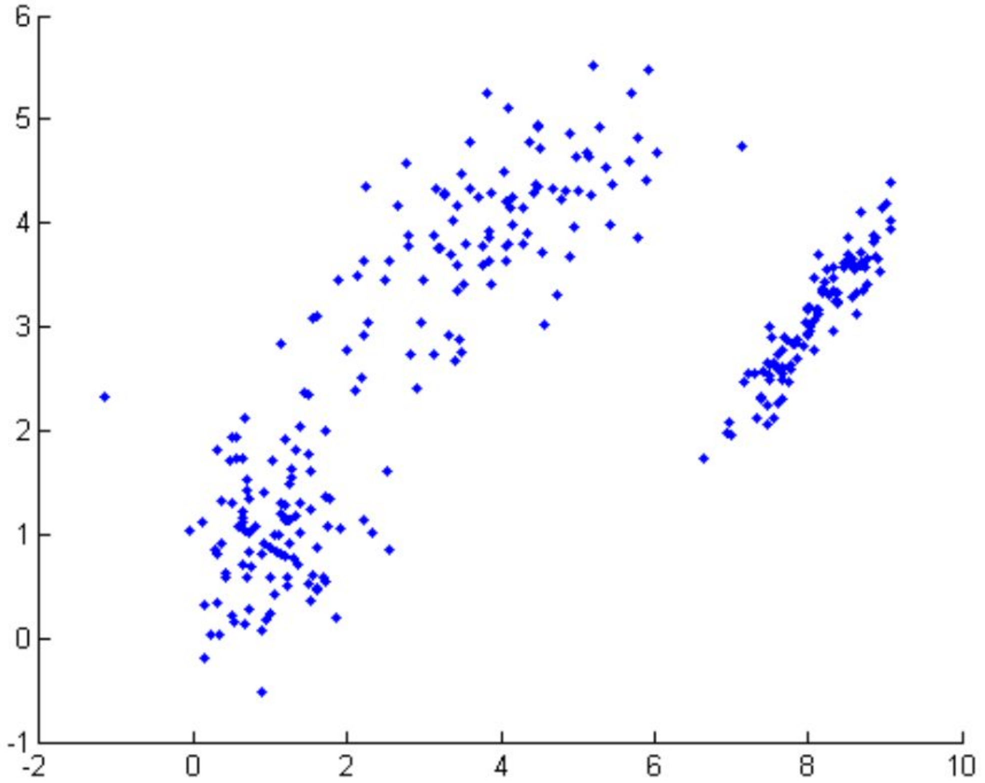
Initialization: Choose K random points to act as cluster centers

Iterate until convergence (ie. the centroids don’t change by much between iterations):

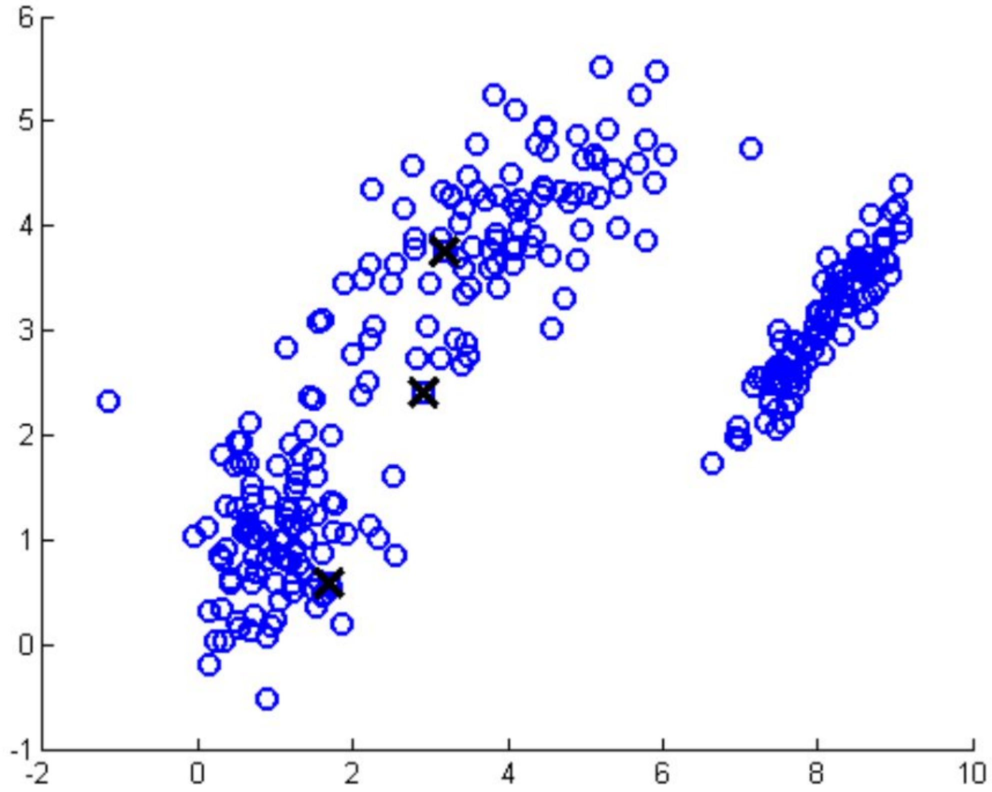
1. Assign points to closest center (forming K groups).
2. Reset the centers to be the mean of the points in their respective groups.



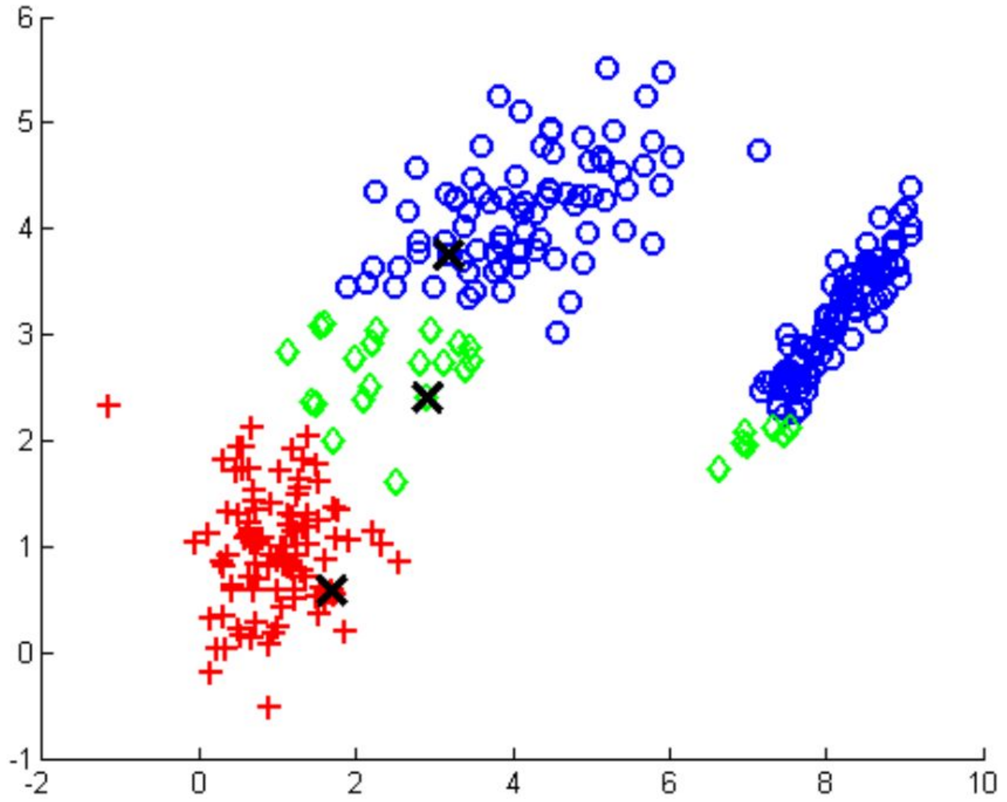
K-Means Clustering - Example Data



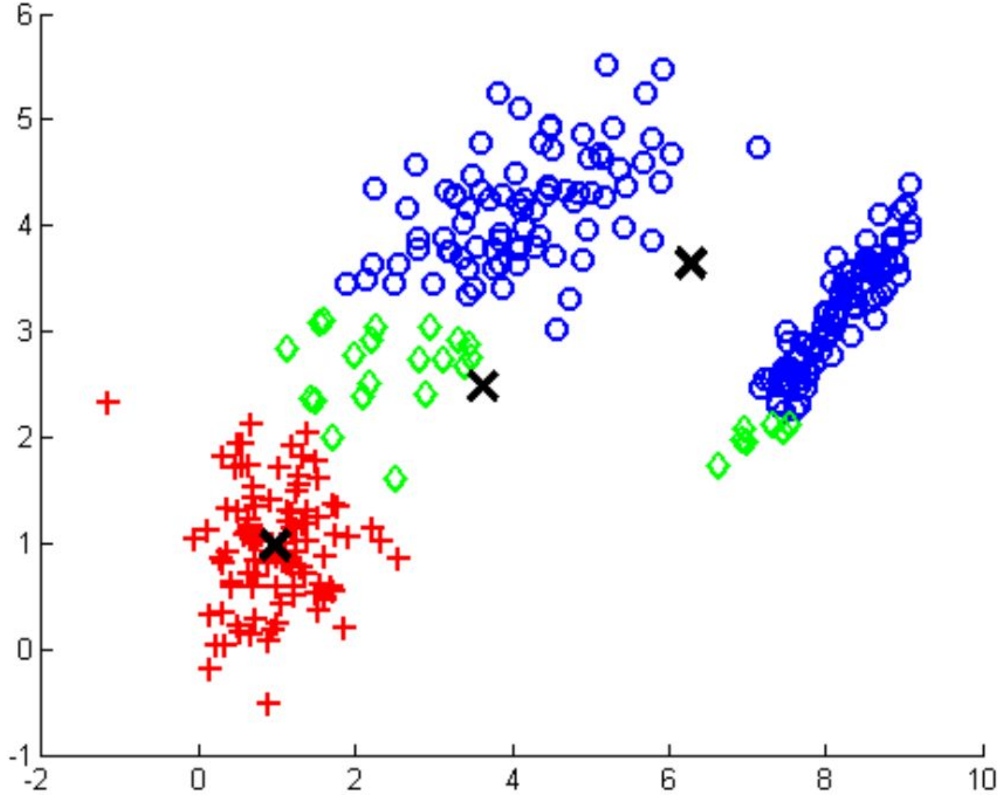
K-Means Clustering - Initialization



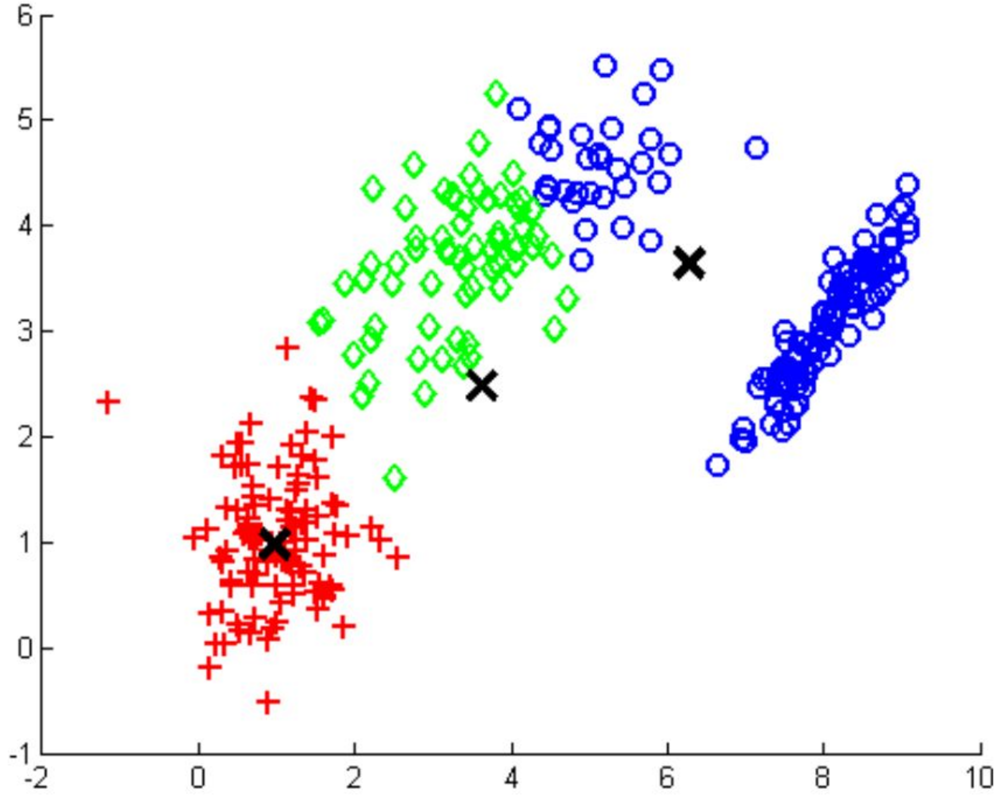
K-Means Clustering - Step 1



K-Means Clustering - Step 2



K-Means Clustering - Step 1



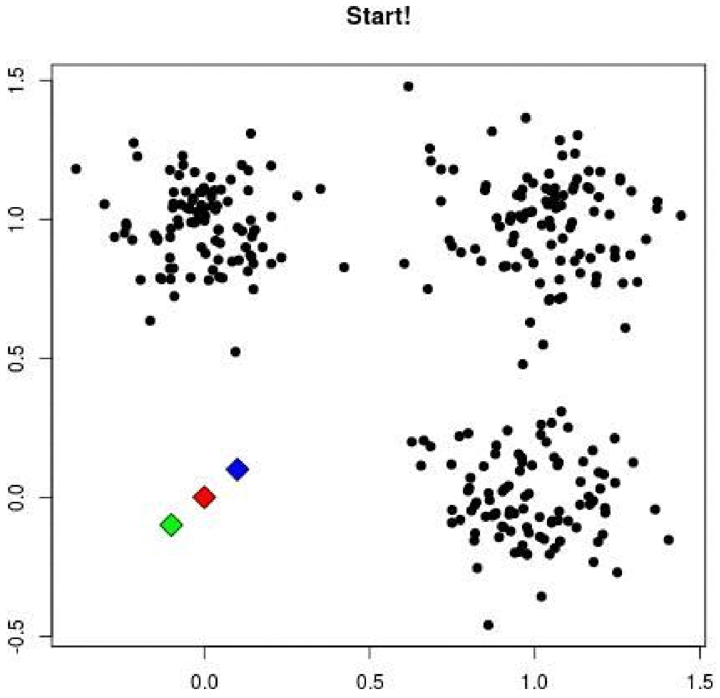
Repeat until Convergence...

Convergence:

When the centroids don't change anymore between iterations.

For this assignment you can literally do:

```
if np.all(new_centroids == centroids):
    break
```



Now we know what K-Means is, how do we use it?

We now know how to cluster things in feature space, but how do we go from an image to feature space?

For this assignment we'll be using [r, g, b, x, y] as the feature vector **per pixel** (since we want a pixel-wise segmentation).

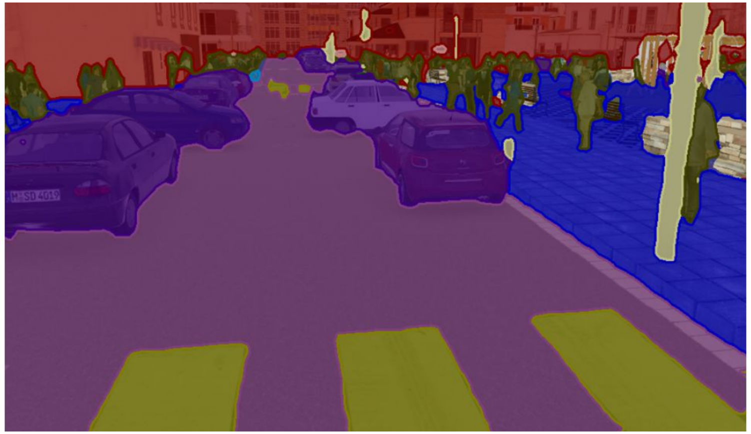
[81, 125, 19, 500, 130]

[r, g, b, x, y] is a pretty standard choice when using K-Means, since you're encoding a belief in **spatial locality** as well as **colour similarity**.

Part 2(b): Image Segmentation with Meanshift

Main Idea

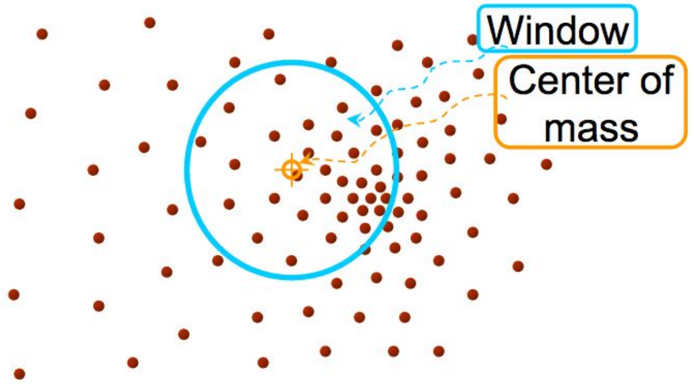
Segment an image into its semantic components, specifically using Meanshift.



Thankfully, you have seen Meanshift in class

The mean shift algorithm seeks the *modes* or local maximums of density of a given distribution

- Choose a search window (size and location)
- Compute the mean of the data in the search window
- Center the search window at the new mean location
- Repeat until convergence



The way you’ll implement it is slightly different...

The meanshift algorithm can be done in the following steps:

(1) Keep track of an array whether we have seen each pixel or not. Initialize it such that we haven't seen any.

The way you'll implement it is slightly different...

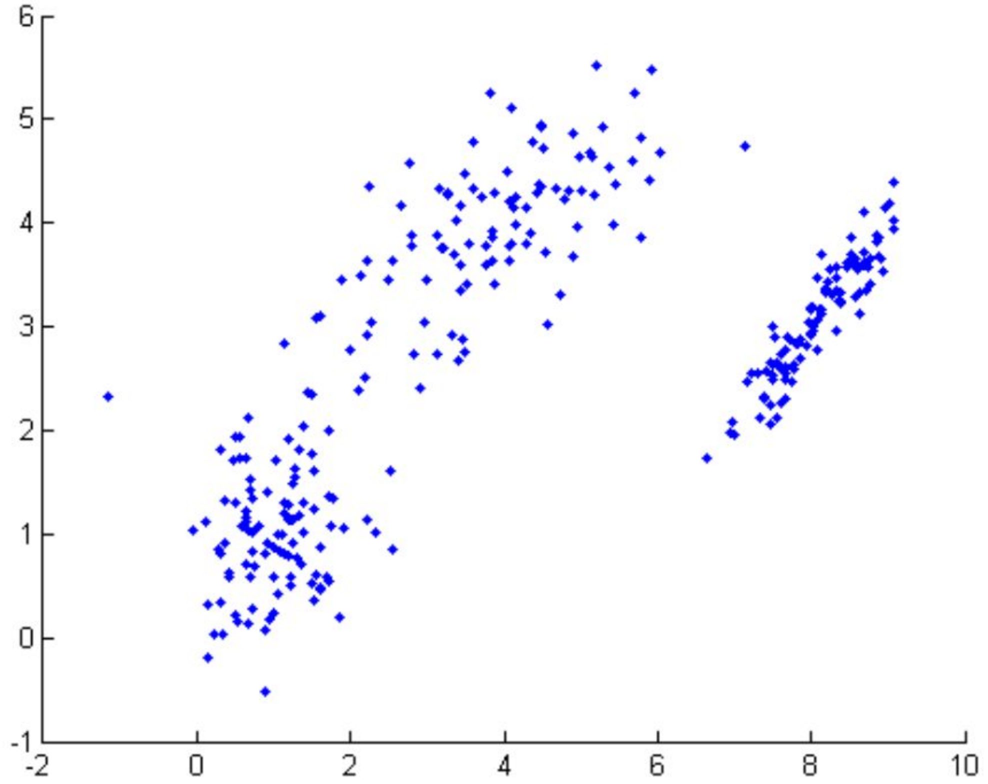
(2) While there are still pixels we haven't seen do the following:

- Pick a random pixel we haven't seen
- Until convergence (mean is within 1% of the bandwidth of the old mean), mean shift. The output of this step will be a mean vector.
    - For each iteration of the meanshift, if another pixel is within the bandwidth circle (in feature space), then that pixel should also be marked as seen
- If the output mean vector from the mean shift step is sufficiently close (within half a bandwidth) to another cluster center, say it's part of that cluster
- If it's not sufficiently close to any other cluster center, make a new cluster

The way you’ll implement it is slightly different...

(3) After finding all clusters, assign every pixel to the nearest cluster
in feature space.

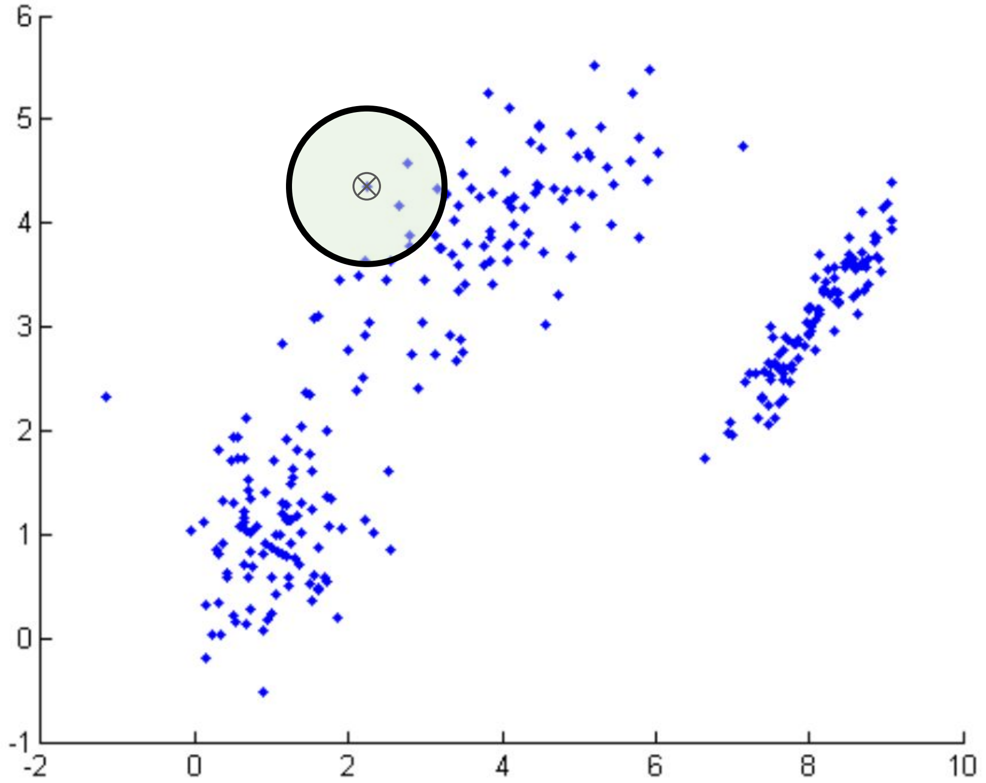
Meanshift Segmentation (Pictorally)



Clusters

Meanshift Segmentation (Pictorally)

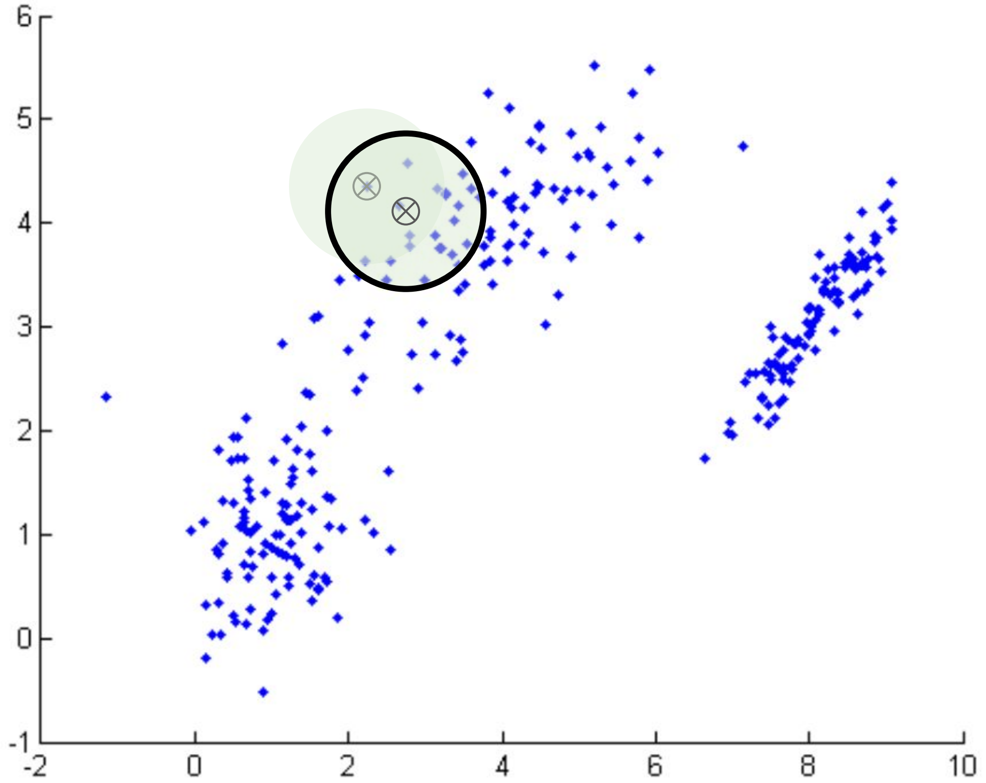
Points within the circle (and green shaded area) are “seen”



Clusters

Meanshift Segmentation (Pictorally)

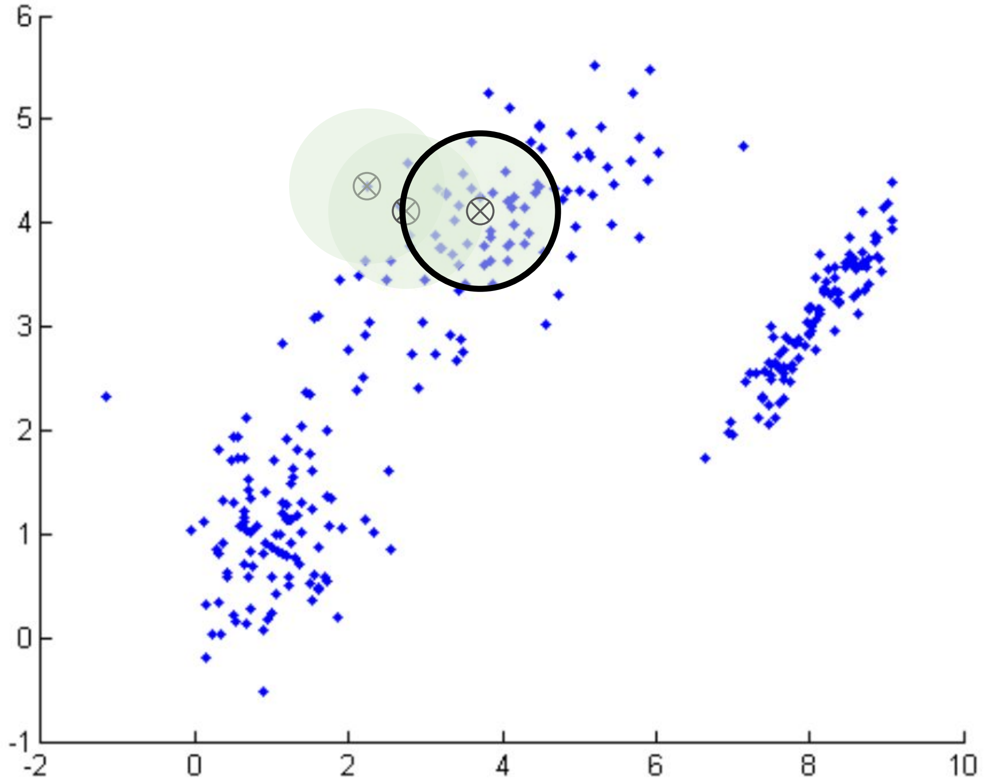
Points within the circle (and green shaded area) are "seen"



Clusters

Meanshift Segmentation (Pictorally)

Points within the circle (and green shaded area) are "seen"

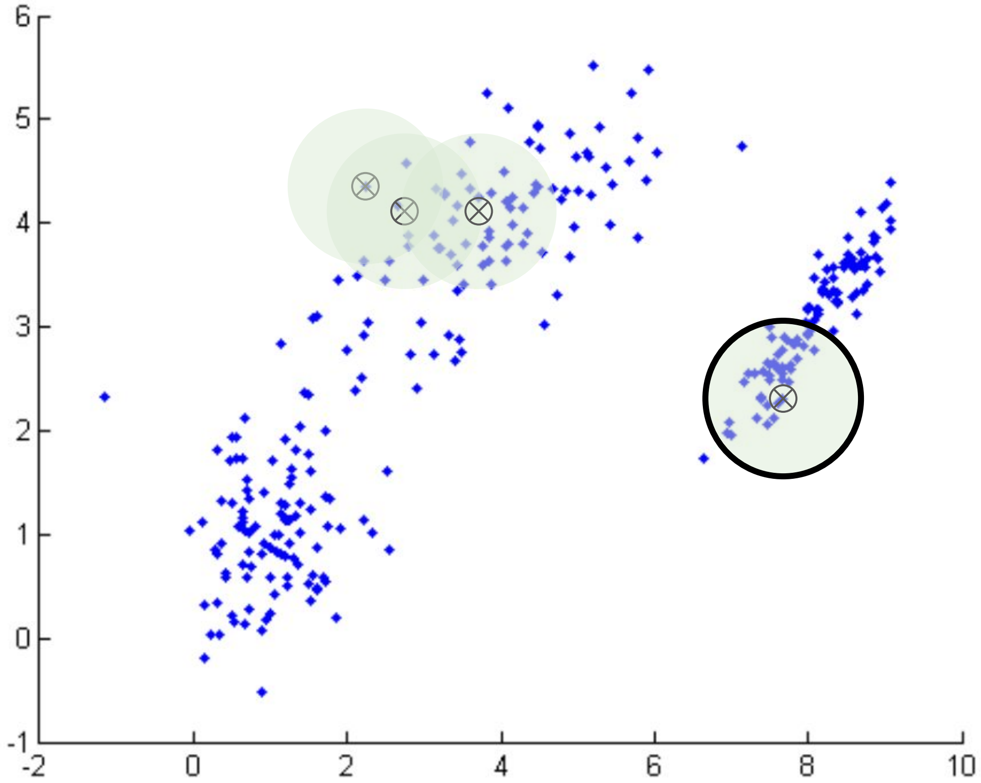


Clusters

(3.8, 4.0)

Meanshift Segmentation (Pictorally)

Points within the circle (and green shaded area) are “seen”

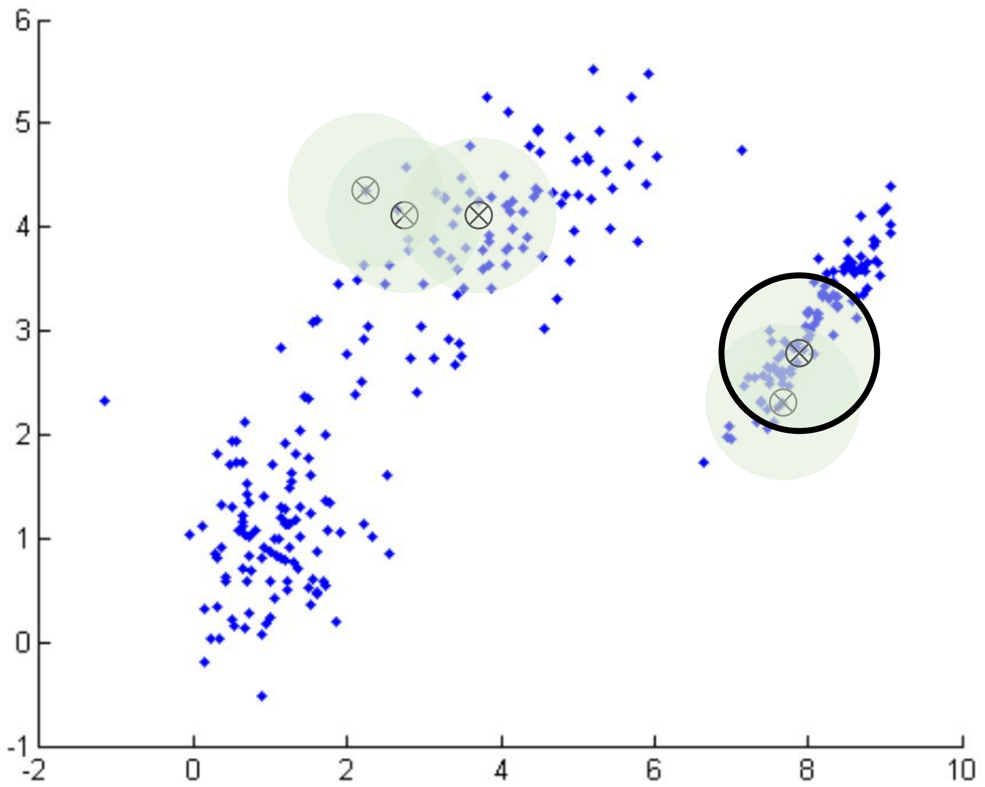


Clusters

(3.8, 4.0)

Meanshift Segmentation (Pictorally)

Points within the circle (and green shaded area) are “seen”

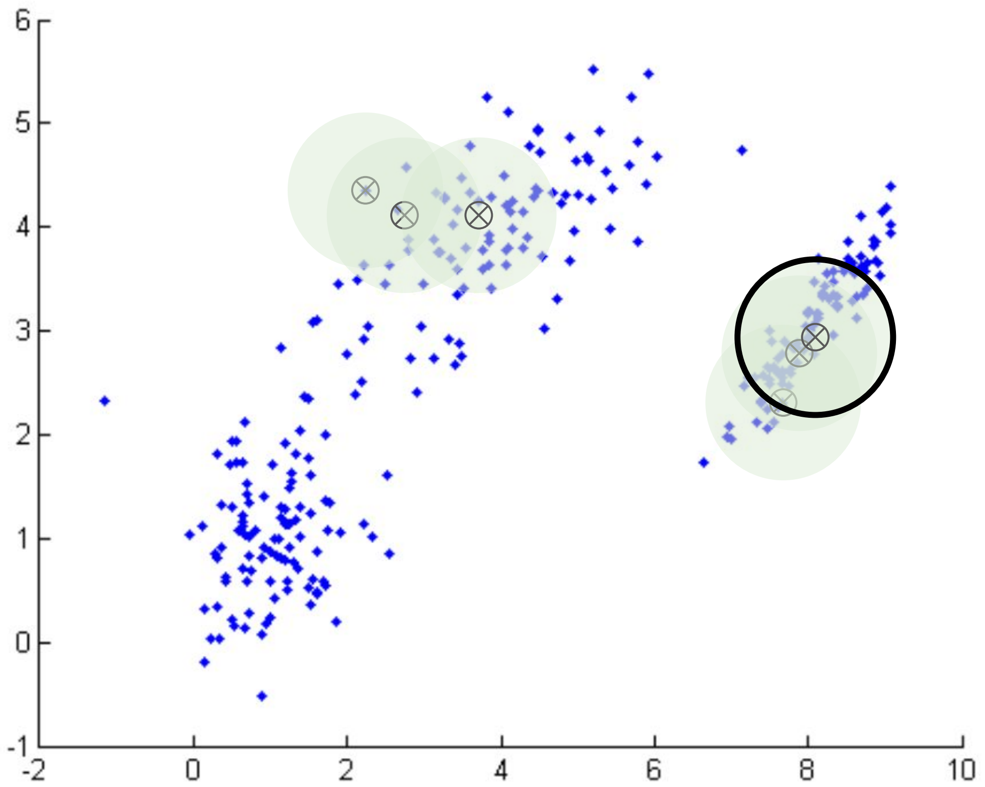


Clusters

(3.8, 4.0)

Meanshift Segmentation (Pictorally)

Points within the circle (and green shaded area) are “seen”



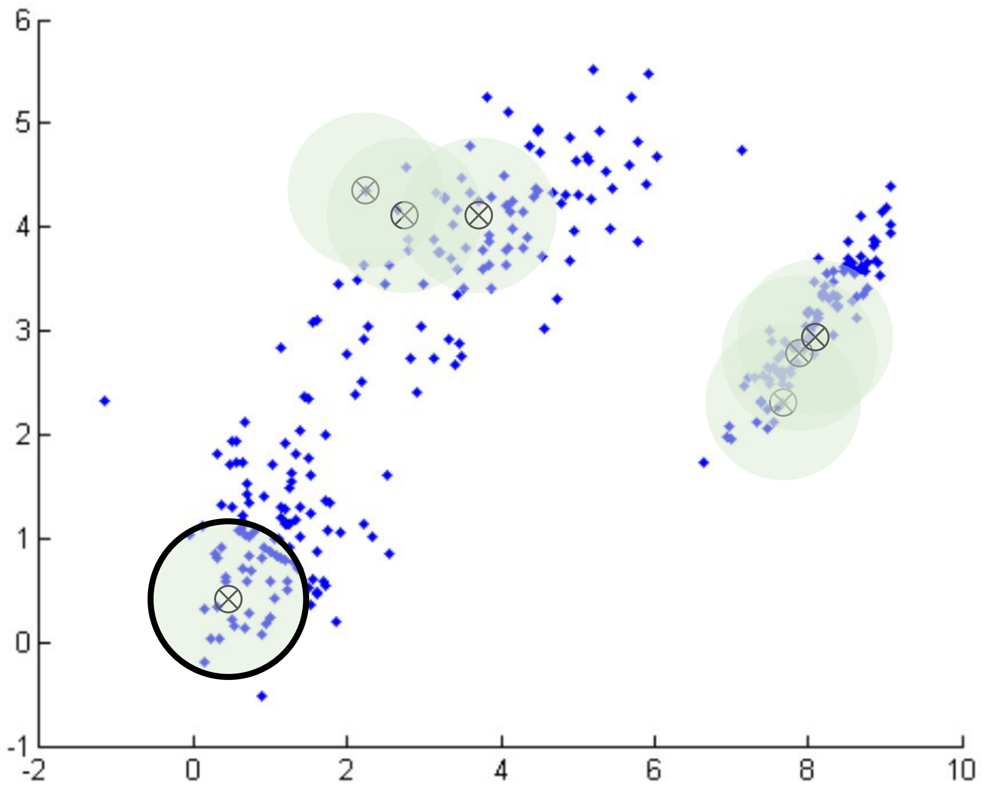
Clusters

(3.8, 4.0)

(8.0, 3.0)

Meanshift Segmentation (Pictorally)

Points within the circle (and green shaded area) are "seen"



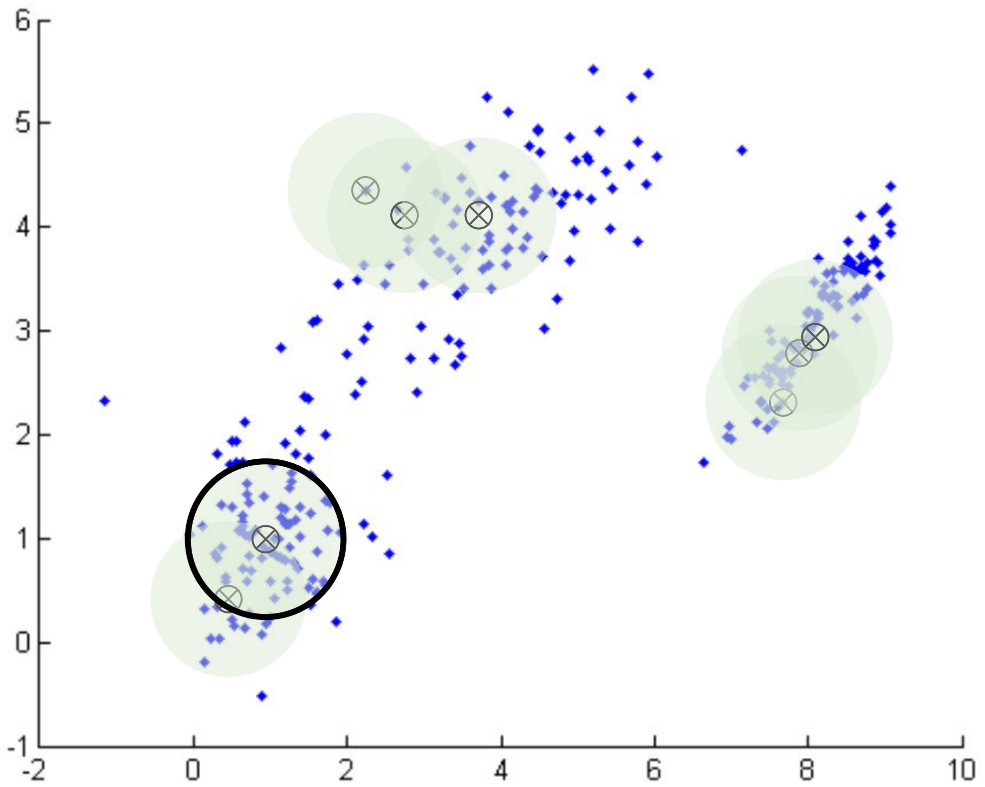
Clusters

(3.8, 4.0)

(8.0, 3.0)

Meanshift Segmentation (Pictorally)

Points within the circle (and green shaded area) are "seen"



Clusters

(3.8, 4.0)

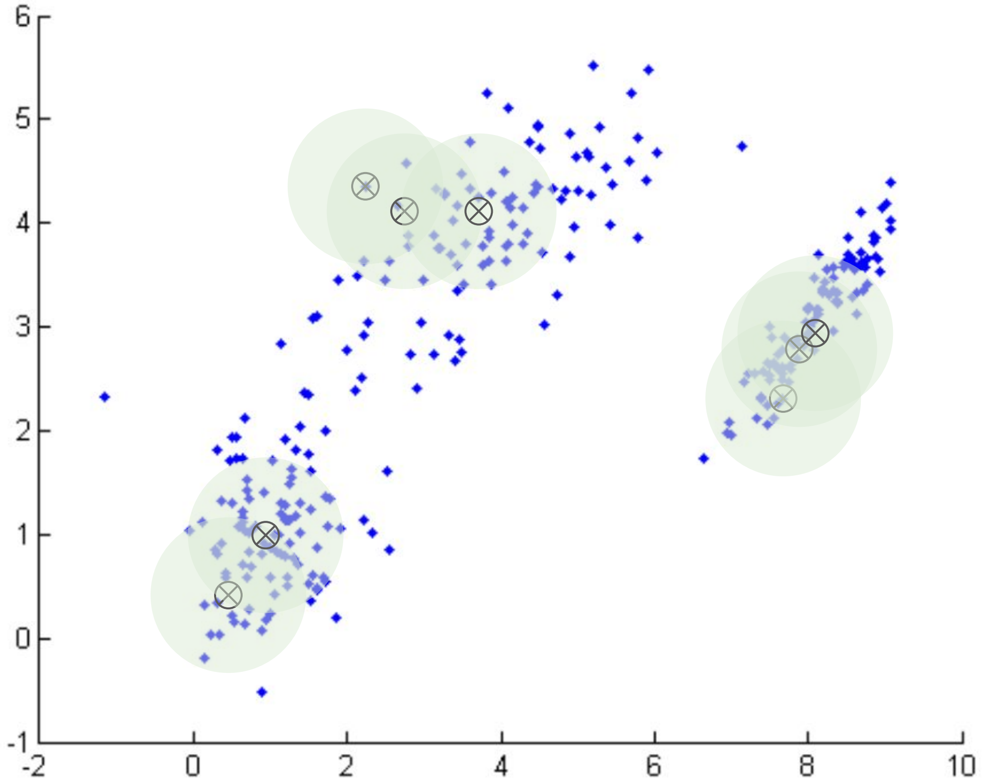
(8.0, 3.0)

(1.5, 1.0)

Meanshift Segmentation (Pictorally)

Points within the circle (and green shaded area) are “seen”

Note, this continues until every point is seen, meaning we’ll have cases where clusters overlap (shown in the next slides).



Clusters

(3.8, 4.0)

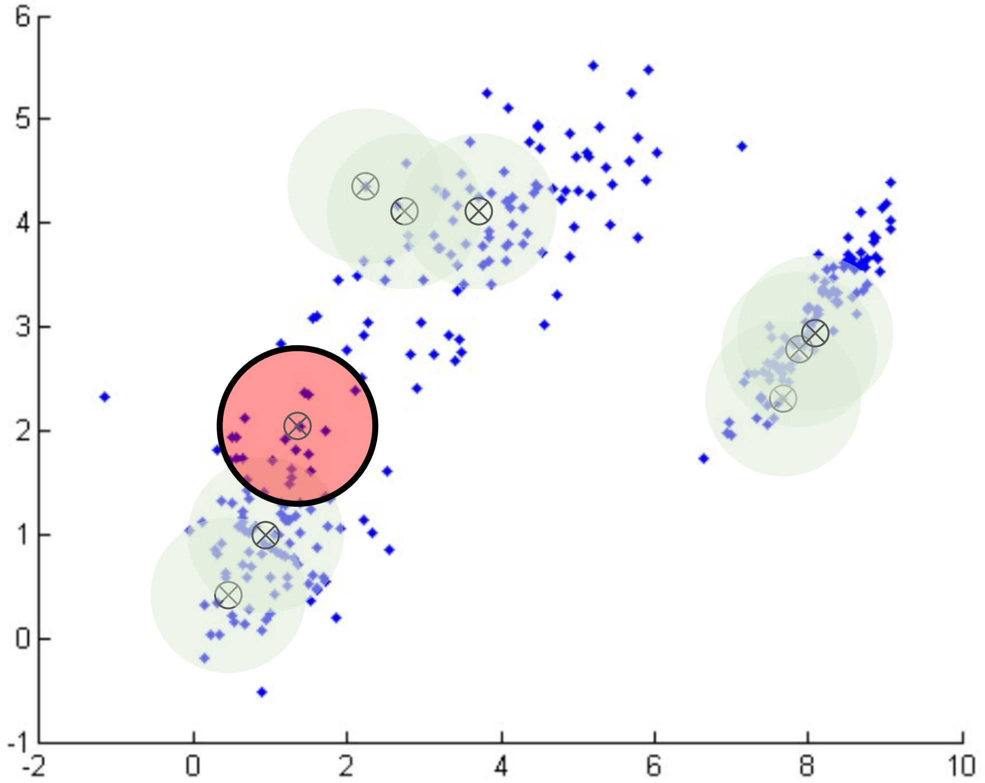
(8.0, 3.0)

(1.5, 1.0)

Meanshift Segmentation (Pictorally)

Points within the circle (and green shaded area) are "seen"

Note, this continues until every point is seen, meaning we'll have cases where clusters overlap.



Clusters

(3.8, 4.0)

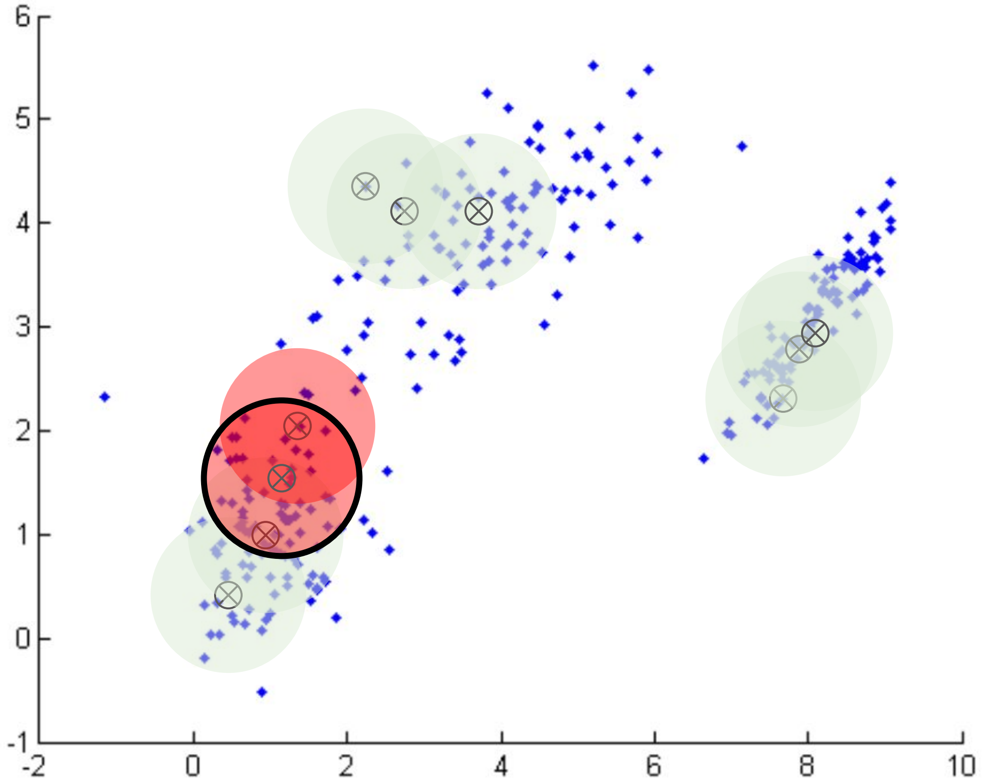
(8.0, 3.0)

(1.5, 1.0)

Meanshift Segmentation (Pictorally)

Points within the circle (and green shaded area) are “seen”

Note, this continues until every point is seen, meaning we’ll have cases where clusters overlap.



Clusters

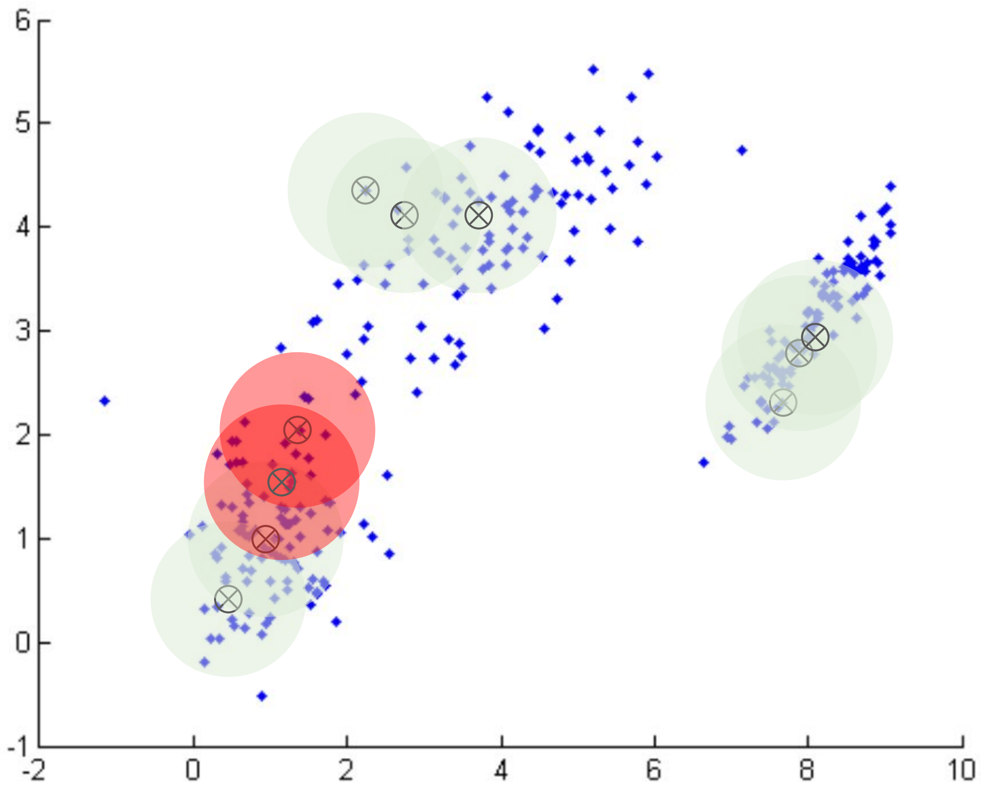
(3.8, 4.0)

(8.0, 3.0)

(1.5, 1.0)

Meanshift Segmentation (Pictorally)

As said in the algorithm, If the output mean vector from the meanshift step is sufficiently close (within half a bandwidth) to another cluster center, say it's part of that cluster.



Clusters

(3.8, 4.0)

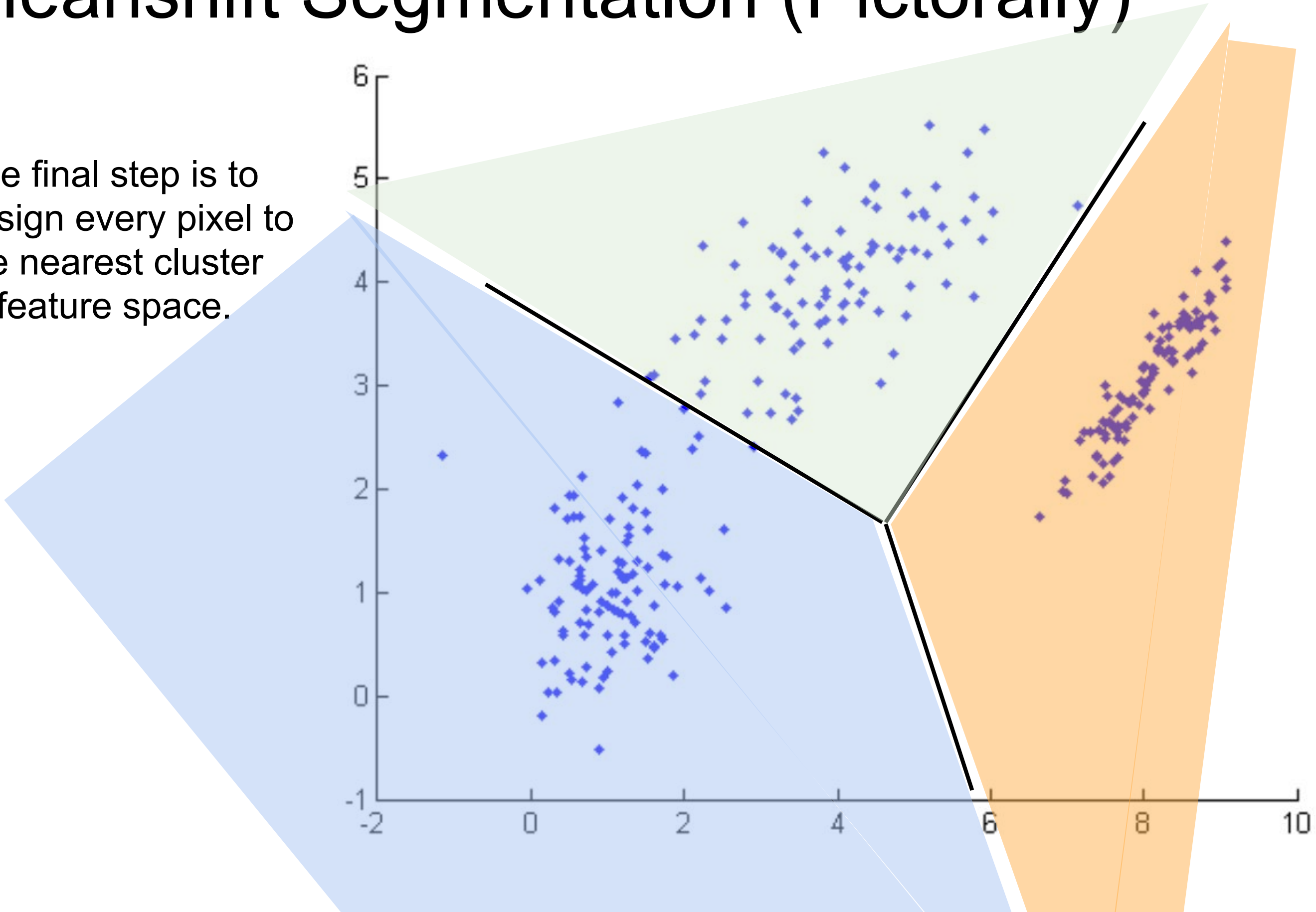
(8.0, 3.0)

(1.5, 1.0)

Note we don’t add the red cluster to this list since it overlaps with another on the list!

Meanshift Segmentation (Pictorally)

The final step is to assign every pixel to the nearest cluster in feature space.



Clusters

(3.8, 4.0)

(8.0, 3.0)

(1.5, 1.0)

Questions for Part 2?

Thanks!