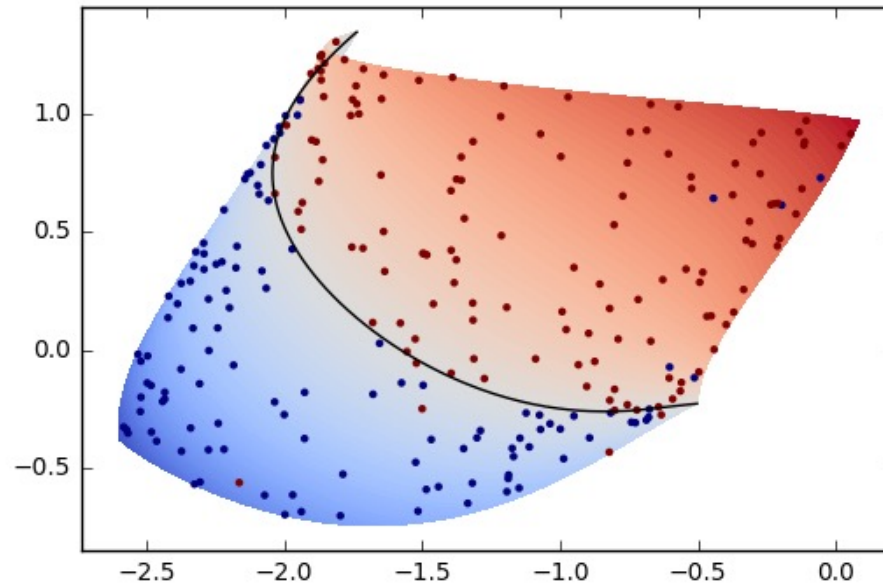


# Lecture 22: kMeans Clustering



Gavin Kerrigan  
Spring 2023

Some slides adapted from Padhraic Smyth, Alex Ihler

# Announcements

---

- HW4 due Friday (5/26)
  - Implementing & experimenting with decision trees
- Projects due in 3 weeks (6/12)
  - Week 9 discussion (6/1) is a project workshop
  - Good idea to have significant progress by then

Projects

K-Means Clustering Example

Properties of K-Means

Data Compression with K-Means

# Class Projects: Data Sets

---

Dataset	Type	#Instances	#Labels	Each Instance
Diabetes 130-US Hospitals	Tabular/ Classification	100K	3	49 features
Fashion-MNIST	Image/ Classification	70k	10	28 x 28 pixels
CIFAR-10	Image/ Classification	60k	10	32 x 32 pixels
IMDB Movie reviews	Text/Classification	50k	2	variable length text

# Goals of the Class Projects

---

Main goals:

- > for you to learn about applying ML models to real datasets
- > to gain insight and understanding about strengths/weaknesses of models

The goal is not to build the world's most accurate classifier for your dataset

You should....

1. Understand your dataset in full detail, do exploratory data analysis (Lec01-Lec02)
2. Evaluate at least 4 different classification models
  - kNN, logistic regression, feedforward neural nets, & one of your choice
  - Can try complex models for image/text datasets
  - e.g. convolutional neural networks on images, recurrent nets on text
3. Try to get the best possible performance out of each model
  - i.e. through hyperparameter tuning and experimentation
4. Most importantly: derive **insights** about your models and data from your experiments

# Goals of the Class Projects

---

Main goals:

- > for you to learn about applying ML models to real datasets
- > to gain insight and understanding about strengths/weaknesses of models

The goal is not to build the world's most accurate classifier for your dataset

You should.... most importantly: derive **insights** about your models and data from your experiments.

For example:

- Are there particular classes or datapoints that are hard to classify?
- Are certain types of models better than others here? **Why?**
- What are the limitations of your models and how might you overcome them?

Be creative! Ask and answer questions about and using your data.

# Ideas for your Projects

---

- Look at metrics beyond just classification accuracy
  - Confusion matrices
  - Ranking-based metrics like precision, recall, F1, AUC
  - Do error analysis: what types of errors does the classifier make?
- Look at learning curves
  - Learning curve = accuracy/error as a function of training set size
  - You can subsample your dataset to create smaller-sized datasets
  - Differences between classifiers can change as dataset size changes

# Ideas for your Projects

---

- Investigate complexity-error tradeoffs
  - See if you can create graphs like we showed in class
  - X-axis = complexity of your model
  - Y-axis = error on training and on test
  - Note: you may need to vary (e.g., reduce) training set size
- Interpretability
  - Investigate which features the model is relying on
    - E.g., remove features one by one and measure increase (if any) in accuracy
  - For neural networks, can you understand what the hidden units are doing? Are there visualization tools to interpret neural nets?



# Project FAQs

---

Q: How should I handle categorical features?

Idea: somehow represent the categories numerically, and then plug into model as with any other numerical feature

- *Binary encoding*: e.g. employed = {'yes', 'no'}
  - Map 'yes' to 0, 'no' to 1
- *Ordinal encoding*: e.g. income = {'low', 'medium', 'high'}
  - Map 'low' to 0, 'medium' to 1, 'high' to 2
  - Only makes sense when the categories have some natural ordering
- *Categorical encoding*: e.g. state = {'AK', 'AZ', ..., 'WY'}
  - Create a new binary feature per each possible category
  - e.g. new features state\_AK, state\_AZ, ..., state\_WY
  - Then if state='WY', we have state\_AK=0, state\_AZ=0, ..., state\_WY=1

# Project FAQs

---

Q: How should I handle categorical features?

Idea: somehow represent the categories numerically, and then plug

A very useful tool for categorical encoding in Pandas:

## pandas.get\_dummies

```
pandas.get_dummies(data, prefix=None, prefix_sep='_', dummy_na=False,  
columns=None, sparse=False, drop_first=False, dtype=None) \[source\]
```

Convert categorical variable into dummy/indicator variables.

Each variable is converted in as many 0/1 variables as there are different values. Columns in the output are each named after a value; if the input is a DataFrame, the name of the original variable is prepended to the value.

# Project FAQs

---

Q: What is a “good enough” error rate?

Depends!

- Some datasets are inherently more difficult to classify
- You should put a good-faith effort into tuning your models to get best performance you can from your tools
- Goal of project is *insights and understanding*; not necessarily highest performance

# Project FAQs

---

Q: How should I handle missing data?

	species	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
0	Adelie	39.1	18.7	181.0	3750.0
1	Adelie	39.5	17.4	186.0	3800.0
2	Adelie	40.3	18.0	195.0	3250.0
3	Adelie	NaN	NaN	NaN	NaN
4	Adelie	36.7	19.3	193.0	3450.0
..	...	...	...	...	...
339	Gentoo	NaN	NaN	NaN	NaN
340	Gentoo	46.8	14.3	215.0	4850.0
341	Gentoo	50.4	15.7	222.0	5750.0
342	Gentoo	45.2	14.8	212.0	5200.0
343	Gentoo	49.9	16.1	213.0	5400.0

"NaN" = "not a number"  
(standard notation for  
missing data)

- Easiest option: drop rows with any missingness
- You could explore *imputation*: try to fill in missing values based on rest of data
- Make sure to write in your report what you choose to do, and justify *why* you chose to do it

Questions?

Projects

K-Means Clustering Example

Properties of K-Means

Data Compression with K-Means

# Unsupervised Learning

---

## Supervised learning:

The learning algorithm is provided with target values  $y_i$

Examples: classification, regression, time-series prediction

## Unsupervised learning:

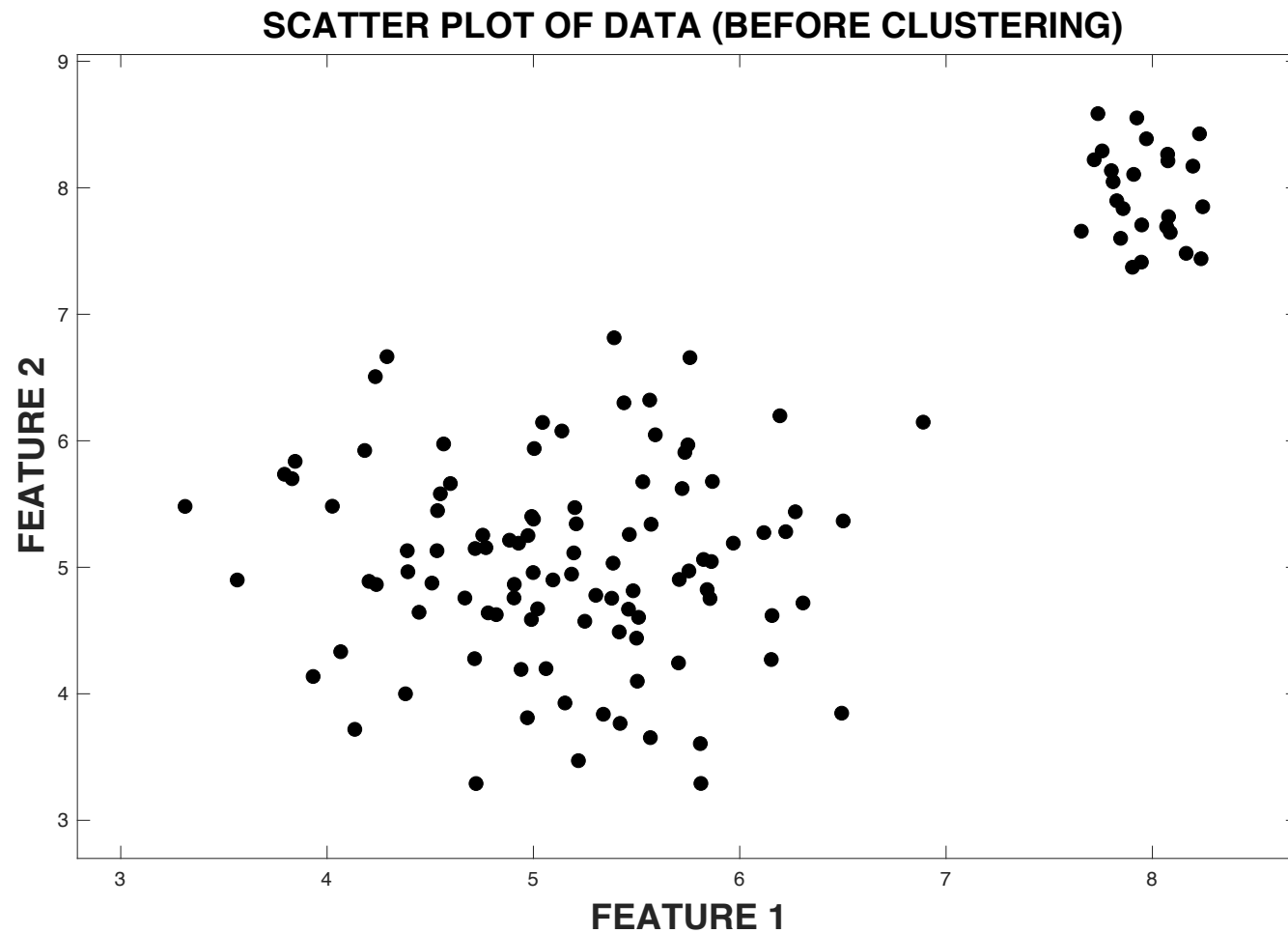
The learning algorithm has no target values

Examples: clustering, dimensionality reduction

Similar to human learning in many respects,  
e.g., how do we form categories for the world around us?

Can be difficult to obtain large, labeled datasets

# Example: K-Means Clustering, 2 Clusters

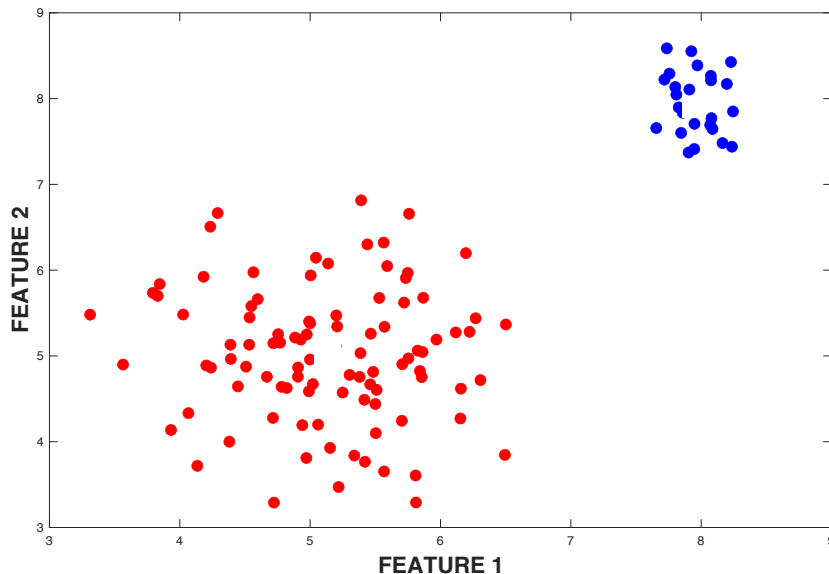




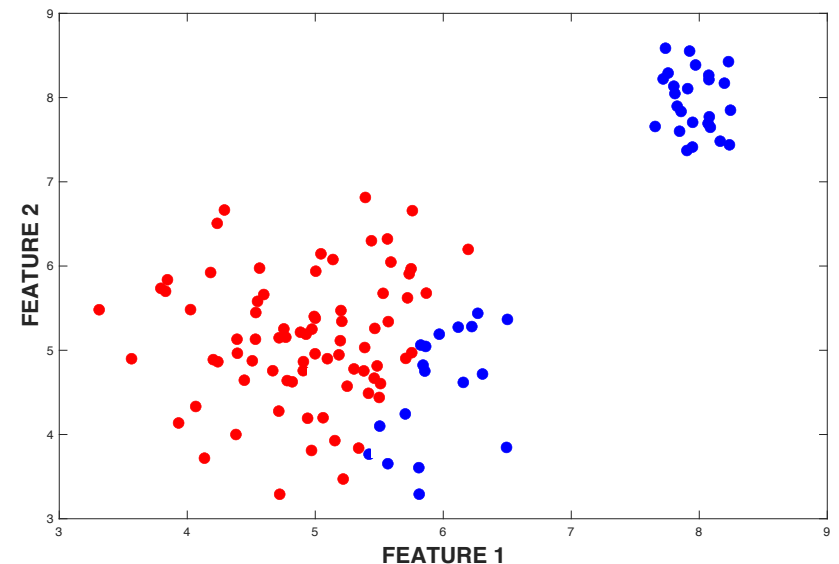
# Clustering of Data

Blue, red indicate which clusters points are assigned to

An Example of Good Clustering



An Example of Bad Clustering



## Questions:

1. How can we quantify what is a good or bad clustering of the data?
2. Can we develop algorithms to automatically find good clusterings?

# Notation

---

**Data:**  $\mathbf{X} = \mathbf{x}_1, \dots, \mathbf{x}_n$ , each  $\mathbf{x}_i$  is a  $d$ -dimensional vector

**Cluster assignment with  $K$  clusters:**

$\mathbf{Z} = z_1, \dots, z_n$ : set of cluster assignments

$z_i \in \{1, \dots, K\}$  is the cluster assignment of vector  $\mathbf{x}_i$

Each  $z_i$  is an integer between 1 and  $K$ : indicates the cluster  $\mathbf{x}_i$  is assigned to

**Clustered Data:**  $\{(\mathbf{x}_i, z_i)\}$ ,  $i = 1, \dots, n$   
(but unlike supervised learning, the  $z_i$ 's are not provided)

# Cluster Centroids

---

$K$  cluster centroids;  $\mathbf{c}_1, \dots, \mathbf{c}_K$

Each centroid  $\mathbf{c}_k$  is a  $d$ -dimensional vector


$$\mathbf{c}_k = (c_{k1}, c_{k2}, \dots, c_{kd})$$

Given cluster assignments  $z_1, \dots, z_n$ , cluster centroid for cluster  $k$  is

$$\text{Centroid}_k = \mathbf{c}_k = \frac{1}{n_k} \sum_{i: z_i=k} \mathbf{x}_i$$

i.e., the average of the vectors  $\mathbf{x}_i$  assigned to cluster  $k$   
where  $n_k$  = number of vectors assigned to cluster  $k$ .

Sum is just over the  
points assigned to  
cluster  $k$



In effect:  $\mathbf{c}_k$  is “center of mass” of points assigned to cluster  $k$

# The K-Means Clustering Algorithm

---

## K-Means Algorithm

Iterates between two steps:

Initialized randomly (e.g., pick  $K$  datapoints randomly as centroids)

### Step 1:

given centroids: compute cluster assignments for each data point  $\mathbf{x}_i$

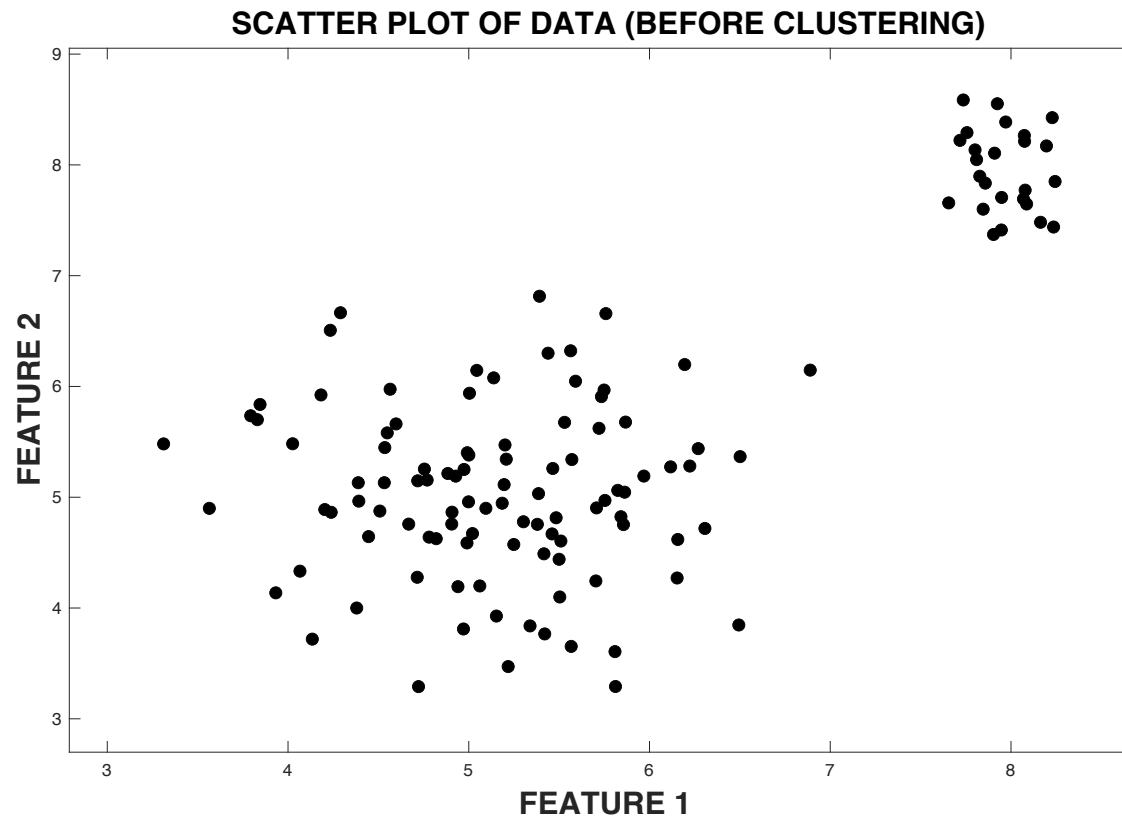
### Step 2:

given assignments: compute cluster centroid for each cluster

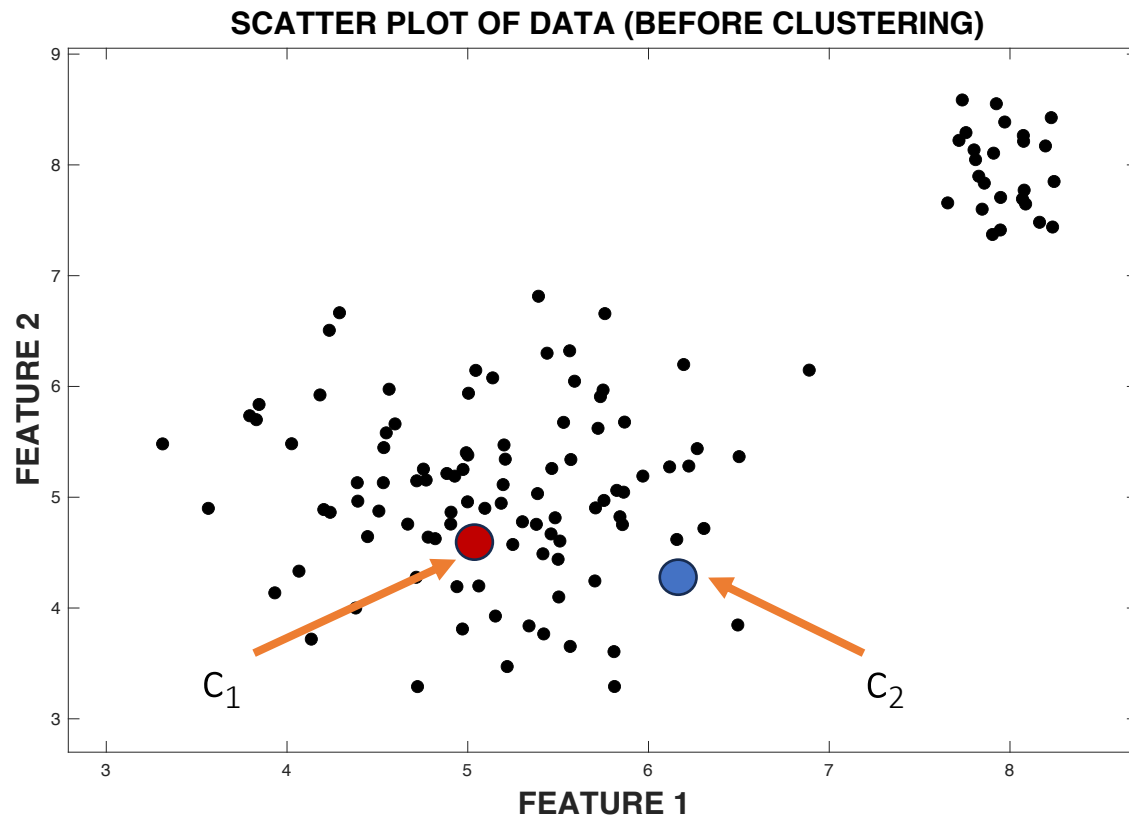
$k = 1, \dots, K$

Guaranteed to converge to at least a local minimum of the Squared Error

# Example: K-Means Clustering, 2 Clusters

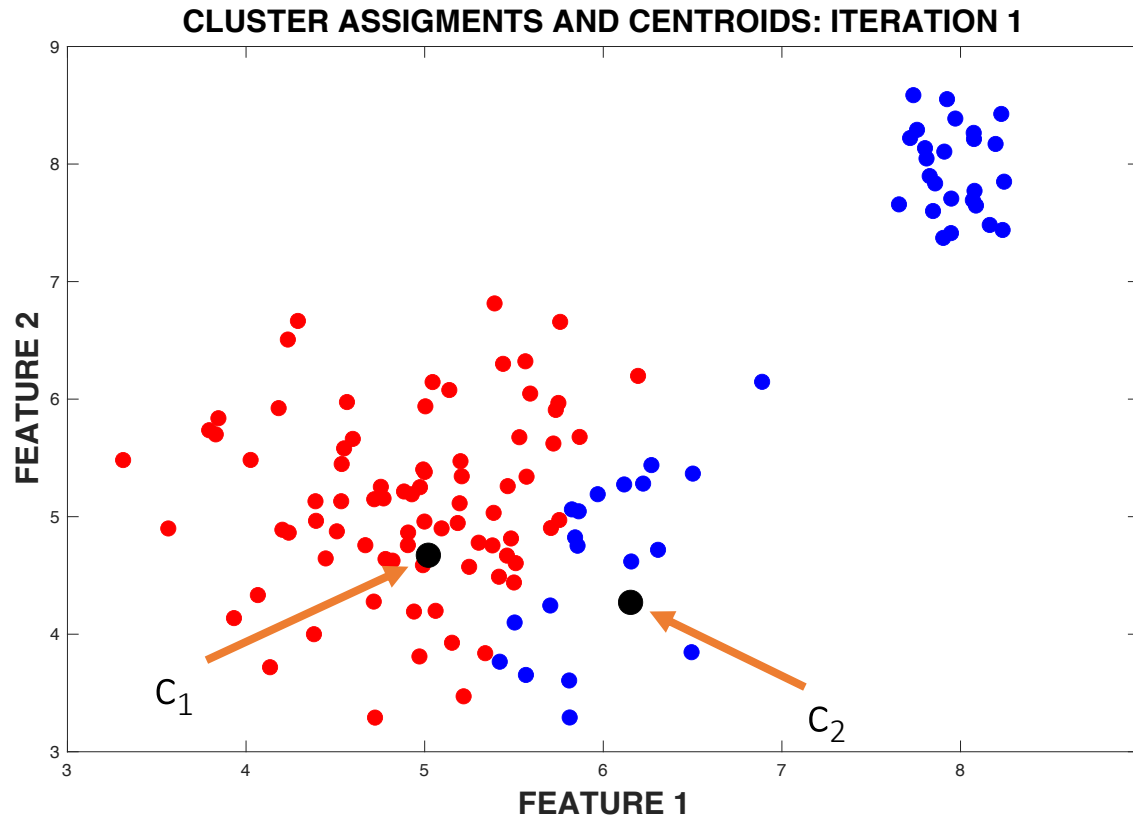


# Example: K-Means Clustering, 2 Clusters



Initialization: Randomly select two datapoints as cluster centroids  $c_1$  and  $c_2$

# Example: K-Means Clustering, 2 Clusters



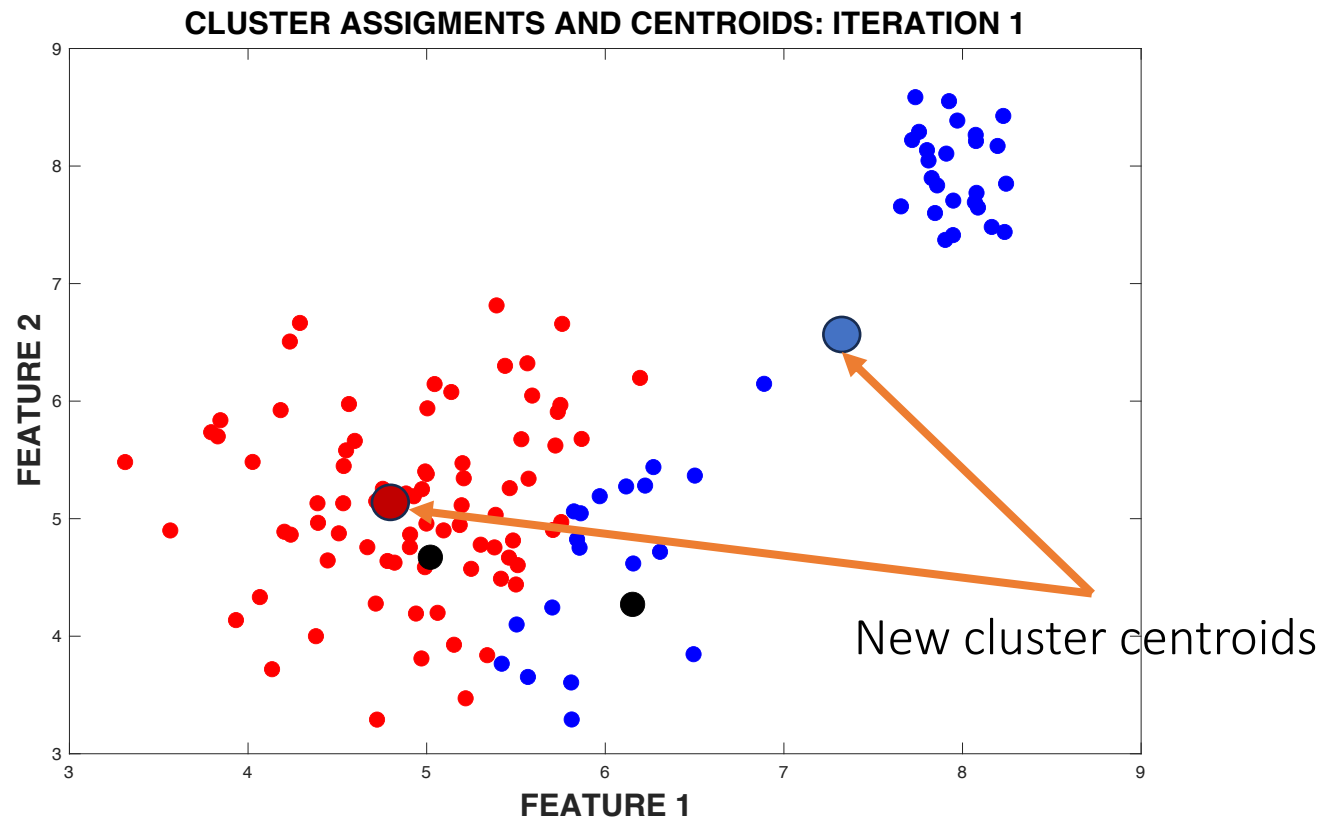
Step 1: Assign *every* datapoint  $\mathbf{x}_i$  a cluster label by:

- Compute the distances  $d(\mathbf{x}_i, \mathbf{c}_1)$  and  $d(\mathbf{x}_i, \mathbf{c}_2)$
- Set  $z_i = 1$  if  $d(\mathbf{x}, \mathbf{c}_1) < d(\mathbf{x}, \mathbf{c}_2)$ ; set  $z_i = 2$  otherwise

**Squared Distance** between centroid  $\mathbf{c}_k$  and data point  $\mathbf{x}_i$ :

$$\text{dist}(\mathbf{c}_k, \mathbf{x}_i) = \sum_{j=1}^d (c_{kj} - x_{ij})^2$$

# Example: K-Means Clustering, 2 Clusters



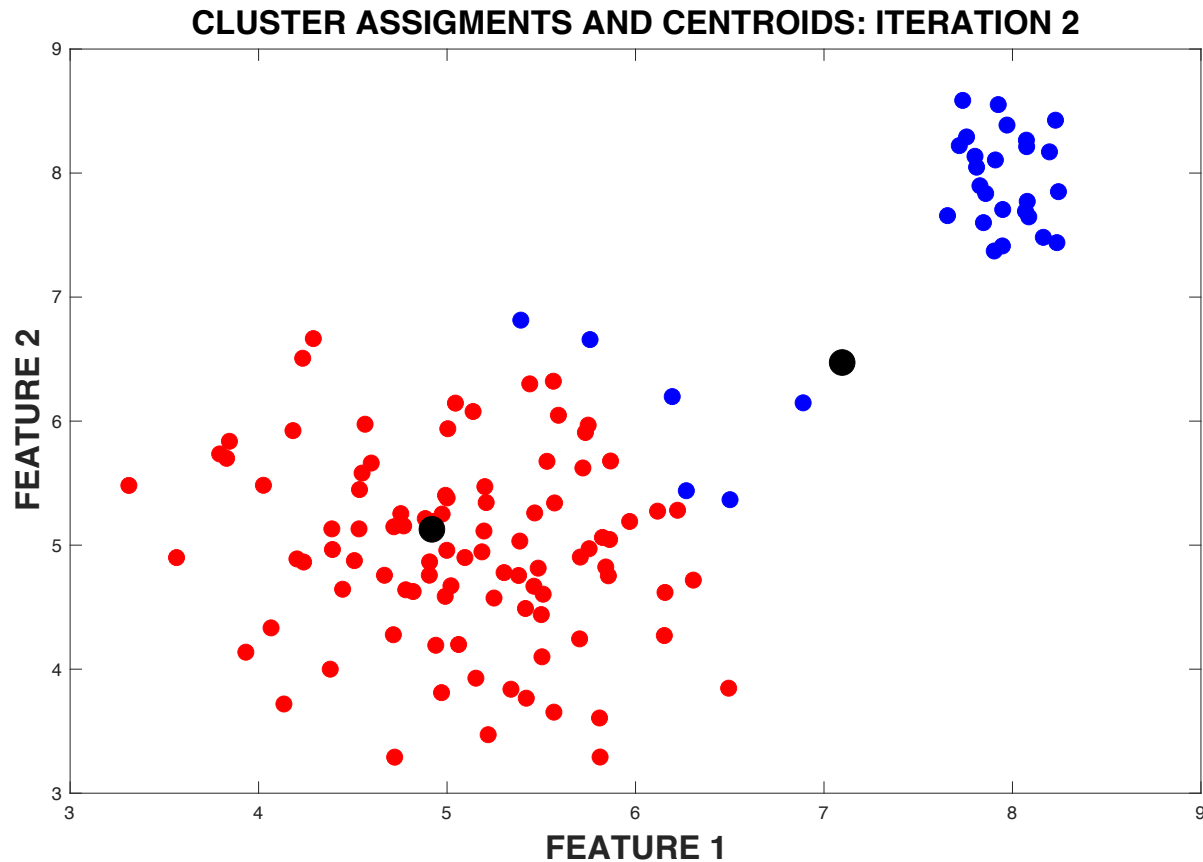
Step 2: Compute new centroids for each cluster:

- Take the mean of all datapoints assigned to that cluster

$$\mathbf{c}_k = \frac{1}{n_k} \sum_{i: z_i=k} \mathbf{x}_i$$

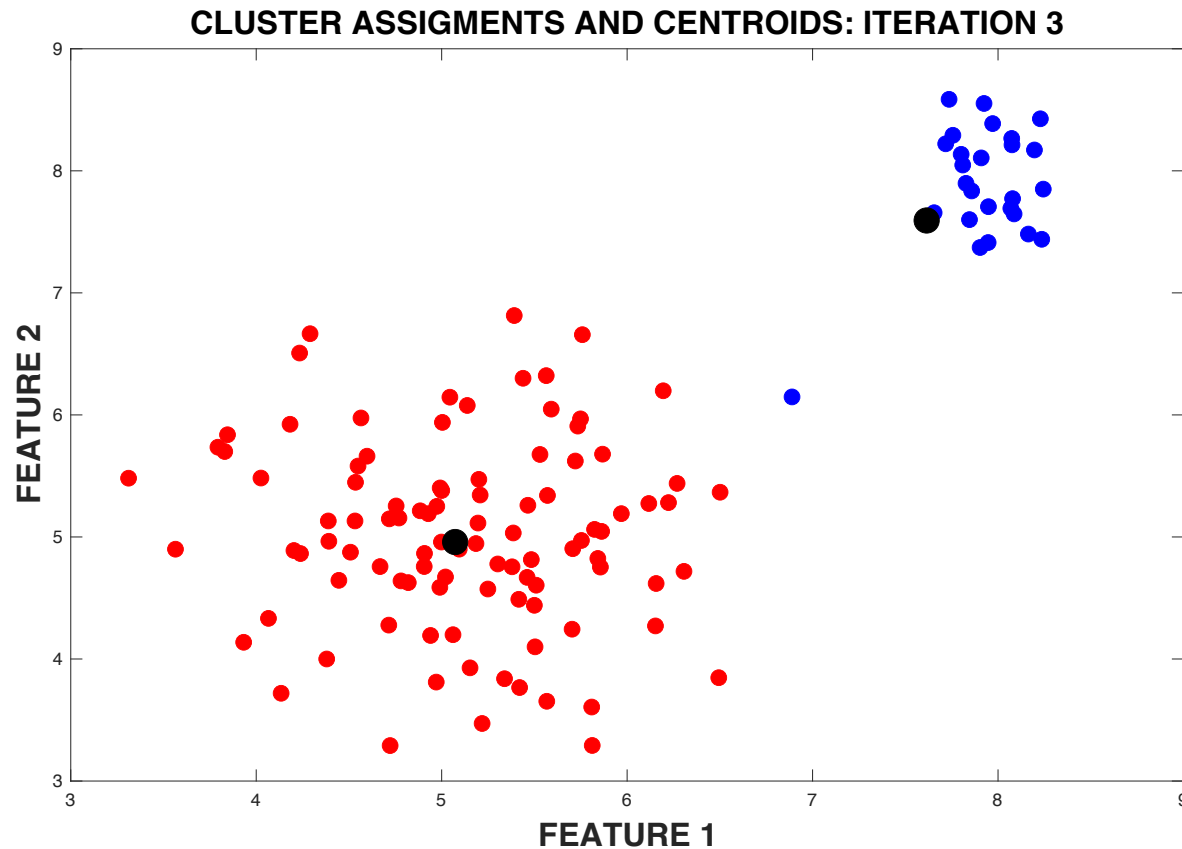


# Example: K-Means Clustering, 2 Clusters



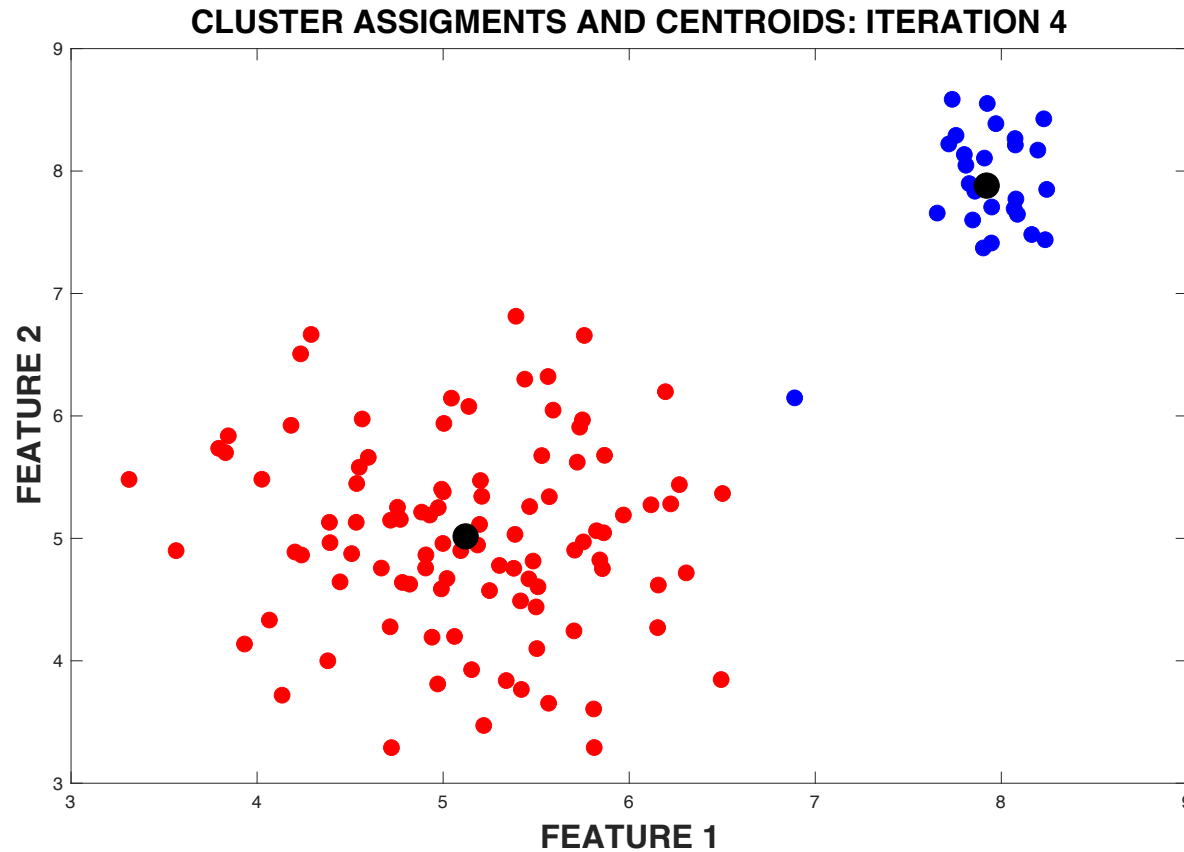
Now iterate Step 1 & Step 2 until convergence

# Example: K-Means Clustering, 2 Clusters



Now iterate Step 1 & Step 2 until convergence

# Example: K-Means Clustering, 2 Clusters



Now iterate Step 1 & Step 2 until convergence

Questions?

Projects

K-Means Clustering Example

Properties of K-Means

Data Compression with K-Means

# K-Means Cluster Boundaries

---

Between any 2 clusters: linear boundary, halfway between cluster centers (similar to nearest-centroid classifier)

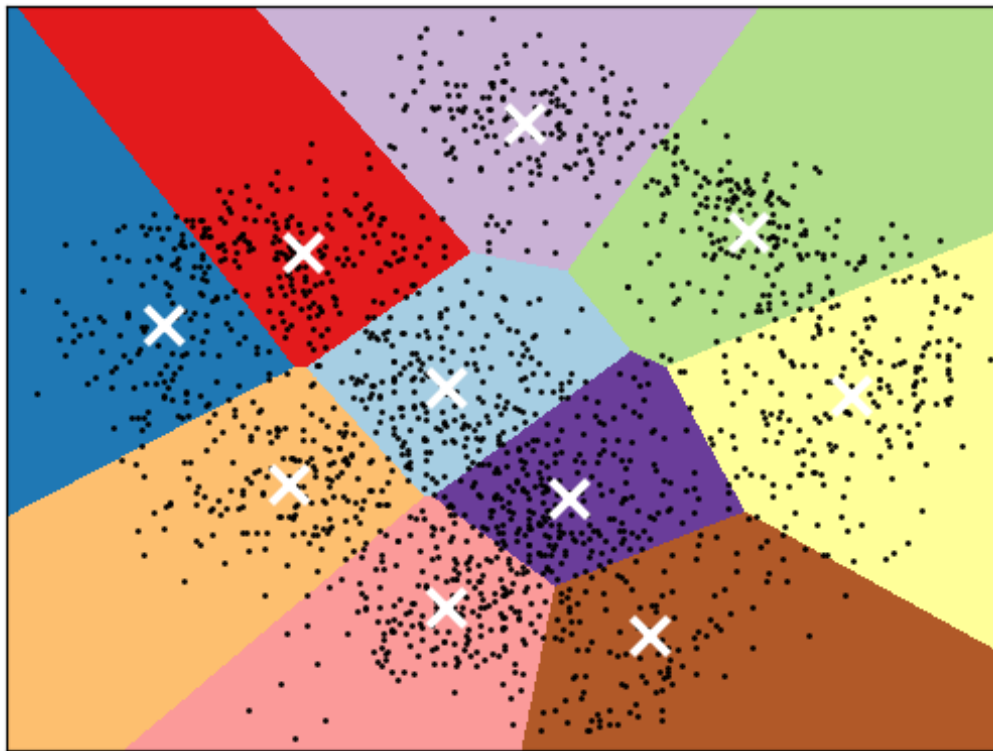
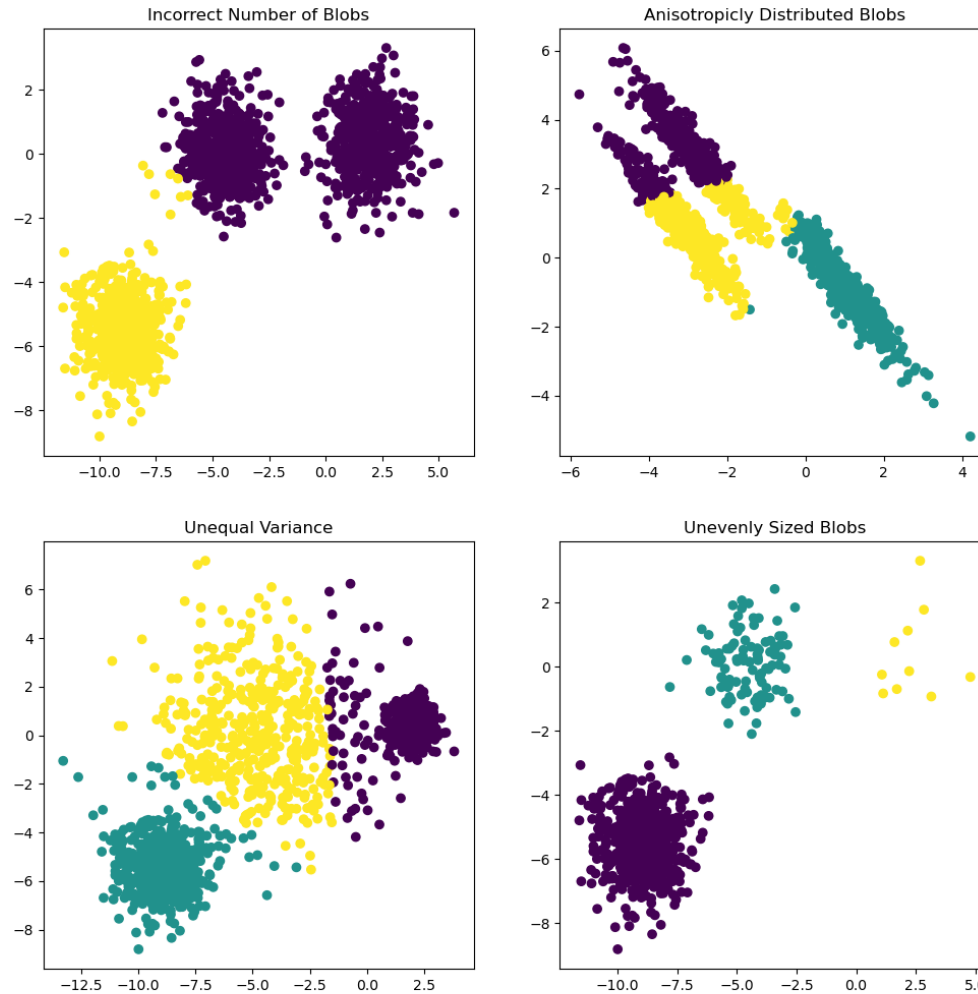


Figure from [https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_kmeans\\_digits.html](https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html)

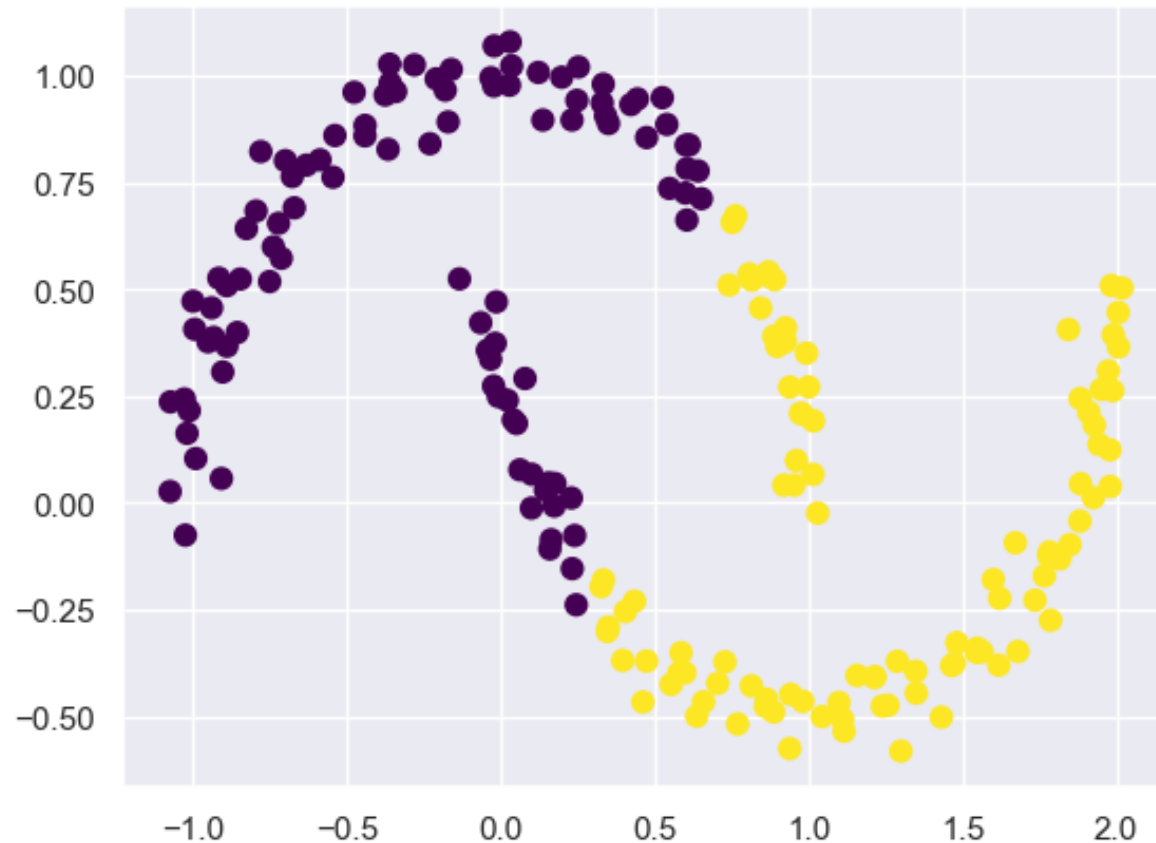
# K-Means on Different Cluster Shapes



From: [https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_kmeans\\_assumptions.html](https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html)

# K-Means on “Non-Compact” Clusters

---



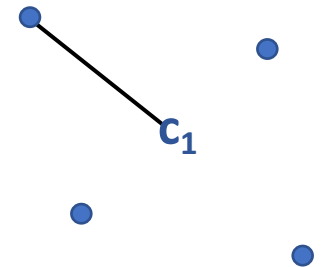


# Squared Distance and Error

---

**Squared Distance** between centroid  $\mathbf{c}_k$  and data point  $\mathbf{x}_i$ :

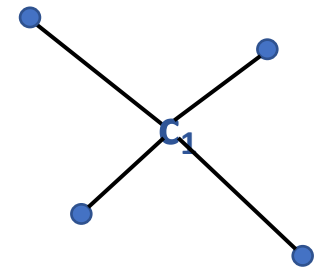
$$\text{dist}(\mathbf{c}_k, \mathbf{x}_i) = \sum_{j=1}^d (c_{kj} - x_{ij})^2$$



**Squared Error for Cluster  $k$** : sum of squared distances of datapoints to centroid:

$$SE_k = \sum_{i: z_i=k} \text{dist}(\mathbf{c}_k, \mathbf{x}_i)$$

i.e. the sum total of how far away each datapoint in cluster  $k$  is from the centroid



# Squared Error for Clustering

**Squared Error (SE):** sum of squared errors across clusters

$$SE = \sum_{k=1}^K SE_k = SE_1 + \dots + SE_K$$

## Clustering Problem:

Find cluster assignments  $z_i$  and cluster centroids  $\mathbf{c}_k$  that minimize:

$$SE = \sum_{k=1}^K SE_k = \sum_{k=1}^K \left( \sum_{i: z_i=k} \text{dist}_S(\mathbf{c}_k, \mathbf{x}_i) \right)$$

i.e. we are quantifying the error of a clustering by the total distance between each datapoint and its corresponding cluster centroid

# Does K-Means always converge? Yes

At each iteration:

- either (a) assignments have not changed (convergence), or
- (b) we do the 2-step iteration:
  - (1) compute centroids: provably reduces the SE
  - (2) update assignments: provably reduces the SE

So, if we do an iteration, the squared error (SE) is guaranteed to decrease

This means its impossible to “revisit” an earlier clustering  
(to do so the SE would need to increase in an iteration: which it can't)

So, since there's a finite number of possible clusterings,  
and each iteration visits a different one,  
=> the algorithm will converge in finite iterations....to at least a local minimum of SE

# Time Complexity?

---

Time Complexity per iteration:

1. **Compute memberships:**

- > compute distances of  $n$   $d$ -dim vectors to  $K$  centroids
- >  $O(n d K)$

2. **Update centroids**

- > compute  $K$  sums, total of  $n$  datapoints summed, each  $d$ -dimensional
- >  $O(n d)$

⇒ Total complexity per iteration is  $O(n d K)$

Number of iterations? depends on the dataset.....

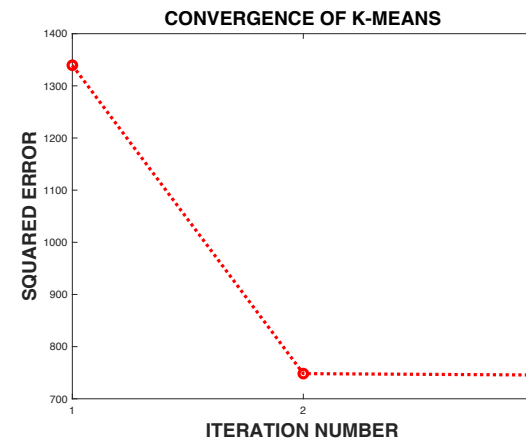
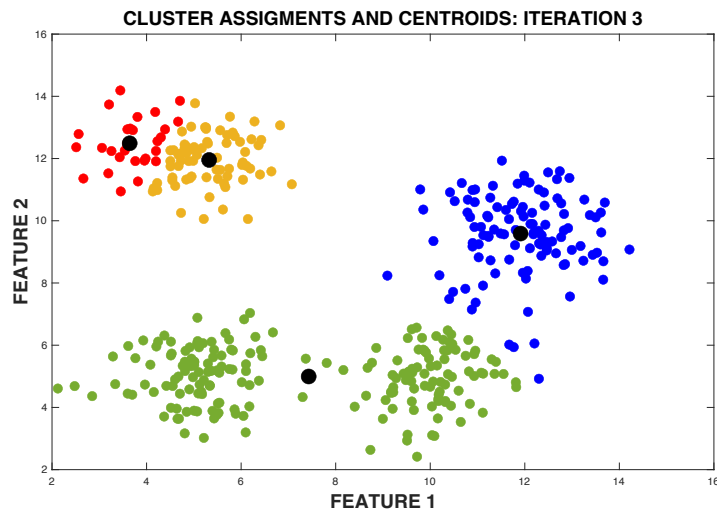
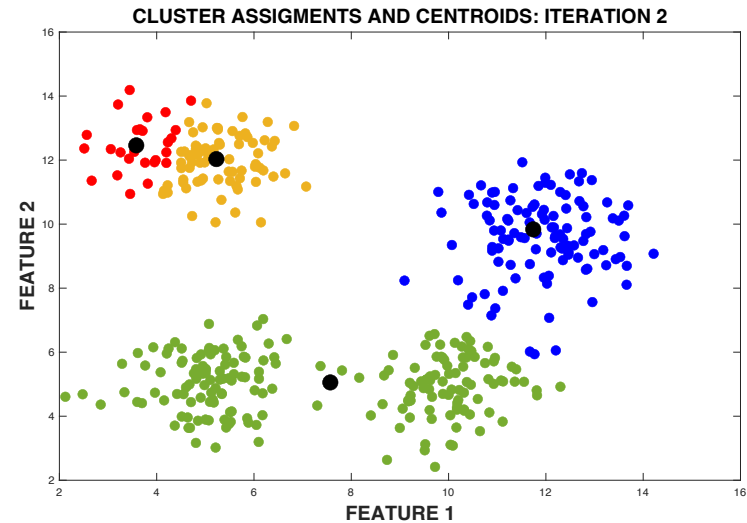
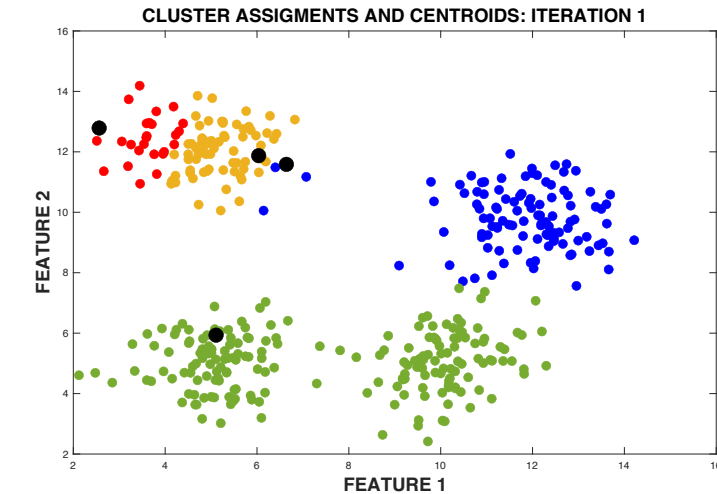
# Local Minima?

---

K-means is sensitive to initial conditions

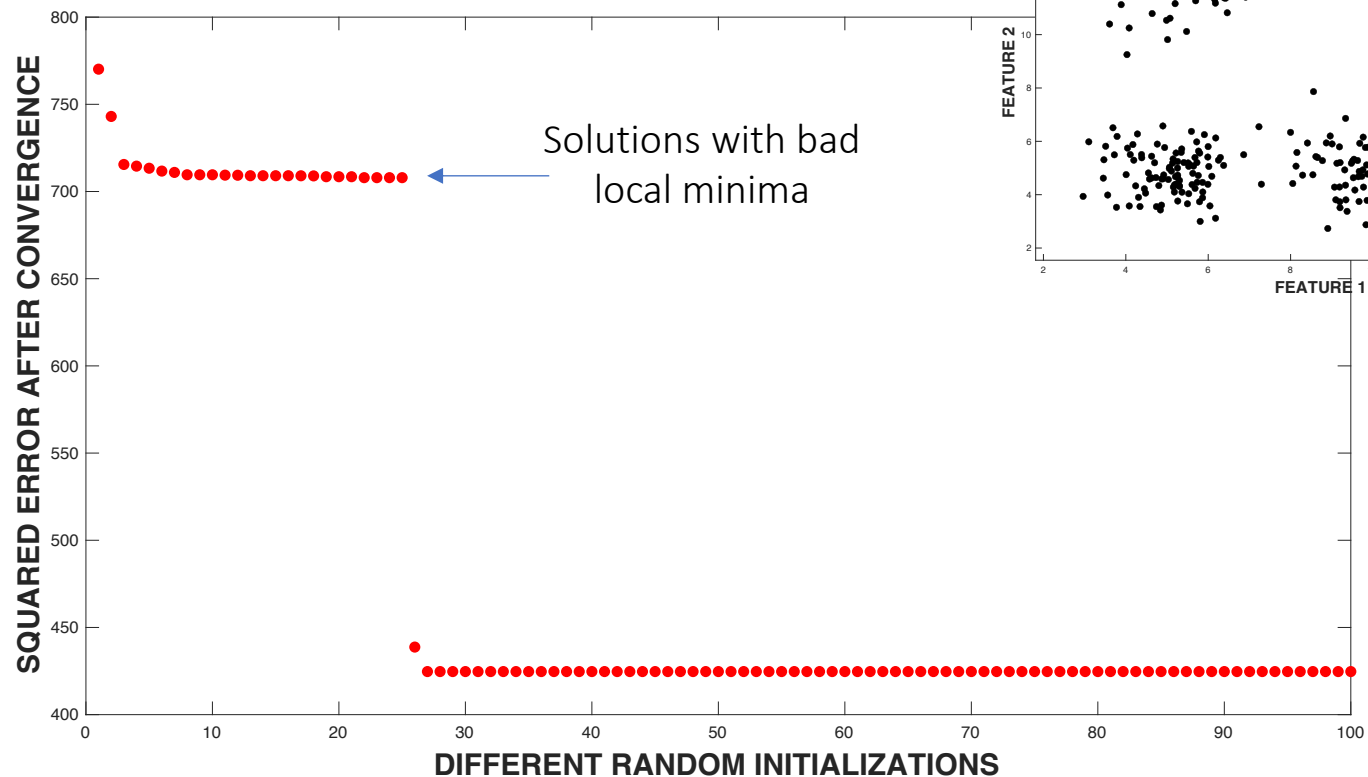
Only guaranteed to converge to a local minimum of Squared Error

# Example: Local Minimum



# Local Minima with $K = 4$

Each red dot is the result of running k-means with a different set of randomly selected initial centroids



# Avoiding Local Minima?

---

Various strategies (can be combined), e.g.,

1. better initializations (e.g., using k-means++ , as in HW5)
2. Run k-means multiple times and select solution with minimum SE

However, finding the centroids/assignments that achieve the global minimum of SE is provably NP-hard

So we cannot guarantee that we can find the optimal global solution unless we do exhaustive search over all possible clusterings

.....and there are  $O(K^n)$  clusterings (huge!) for  $K$  clusters and  $n$  datapoints  
e.g. tiny problem:  $K=4$ ,  $n=100$  has over  $10^{60}$  clusterings



## sklearn.cluster.KMeans

```
class sklearn.cluster.KMeans(n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, verbose=0,
                             random_state=None, copy_x=True, algorithm='lloyd')
```

[\[source\]](#)

K-Means clustering.

Read more in the [User Guide](#).

**Parameters::** **n\_clusters : int, default=8**

The number of clusters to form as well as the number of centroids to generate.

**init : {'k-means++', 'random'}, callable or array-like of shape (n\_clusters, n\_features), default='k-means++'**

Method for initialization:

'k-means++': selects initial cluster centroids using sampling based on an empirical probability distribution of the points' contribution to the overall inertia. This technique speeds up convergence, and is theoretically proven to be  $\mathcal{O}(\log k)$ -optimal. See the description of `n_init` for more details.

'random': choose `n_clusters` observations (rows) at random from data for the initial centroids.

If an array is passed, it should be of shape (n\_clusters, n\_features) and gives the initial centers.

If a callable is passed, it should take arguments X, n\_clusters and a random state and return an initialization.

**n\_init : int, default=10**

Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n\_init consecutive runs in terms of inertia.

**max\_iter : int, default=300**

Maximum number of iterations of the k-means algorithm for a single run.

# Speeding Up KMeans: Minibatch K-means

## `sklearn.cluster.MinibatchKMeans`

```
class sklearn.cluster.MinibatchKMeans(n_clusters=8, *, init='k-means++', max_iter=100, batch_size=1024, verbose=0,
compute_labels=True, random_state=None, tol=0.0, max_no_improvement=10, init_size=None, n_init=3,
reassignment_ratio=0.01)
```

[\[source\]](#)

Mini-Batch K-Means clustering.

Read more in the [User Guide](#).

**Parameters::** `n_clusters : int, default=8`

The number of clusters to form as well as the number of centroids to generate.

**`init : {'k-means++', 'random'}, callable or array-like of shape (n_clusters, n_features), default='k-means++'`**

Method for initialization:

'k-means++' : selects initial cluster centroids using sampling based on an empirical probability distribution of the points' contribution to the overall inertia. This technique speeds up convergence, and is theoretically proven to be  $\mathcal{O}(\log k)$ -optimal. See the description of `n_init` for more details.

'random': choose `n_clusters` observations (rows) at random from data for the initial centroids.

If an array is passed, it should be of shape (n\_clusters, n\_features) and gives the initial centers.

If a callable is passed, it should take arguments X, n\_clusters and a random state and return an initialization.

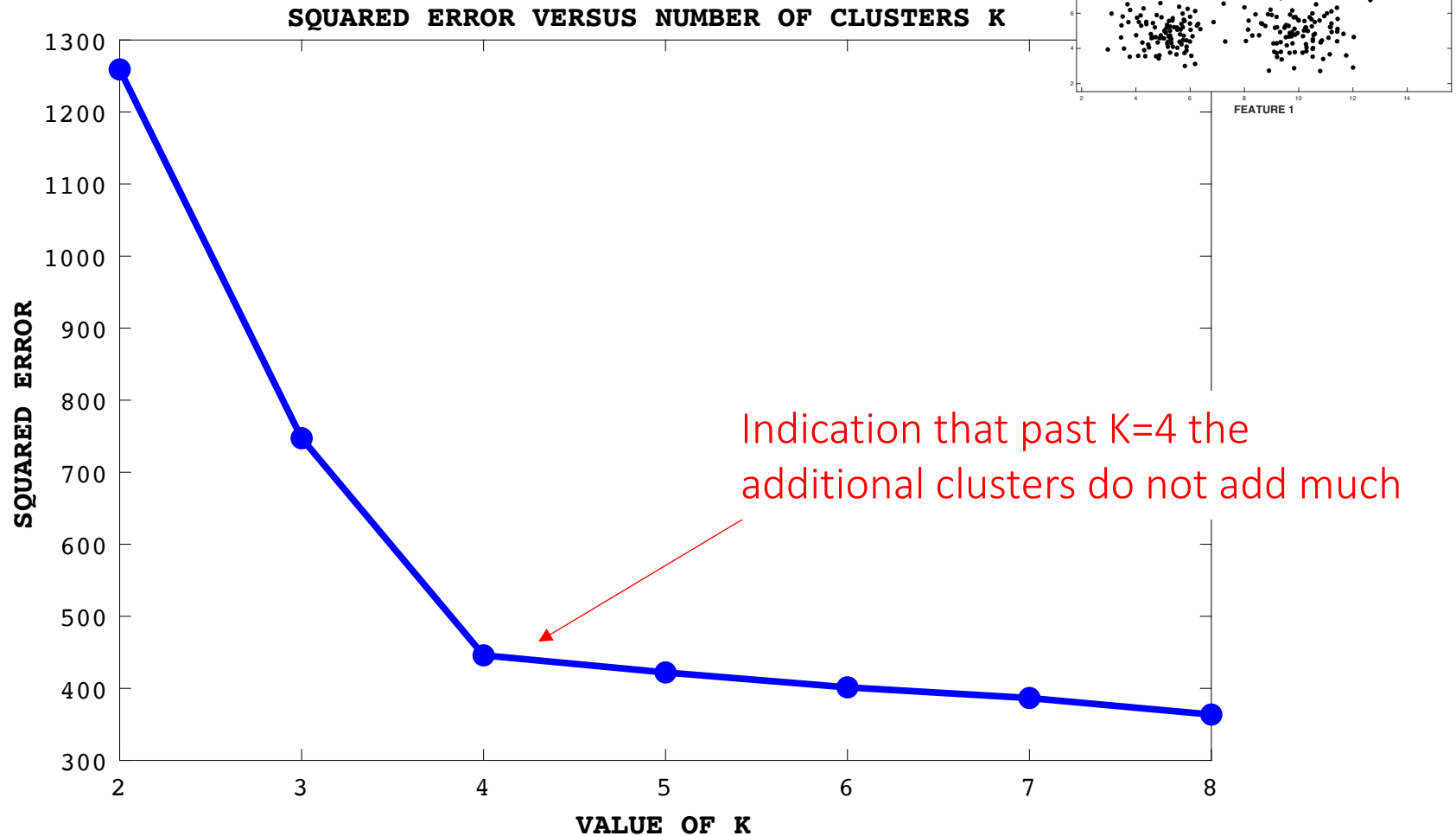
**`max_iter : int, default=100`**

Maximum number of iterations over the complete dataset before stopping independently of any early stopping criterion heuristics.

**`batch_size : int, default=1024`**

Size of the mini batches. For faster computations, you can set the `batch_size` greater than 256 \* number of cores to enable parallelism on all cores.

# How many Clusters?



Questions?

Projects

K-Means Clustering Example

Properties of K-Means

Data Compression with K-Means

# Clustering for Data Compression

---

Motivation for “Lossy” Data Compression:

-> can represent original data approximately with fewer bits

⇒ Compressed data requires less memory, less bandwidth  
at the cost of losing some fidelity to the original data

Compression is widely used in practice e.g., JPEG, MPEG, etc

Compression algorithms and clustering algorithms are often very similar

# Original Color Image

---

3 channels (red, green, blue)

427 x 640 pixel values per channel

Each pixel represented by 24 bits (3 integers, each from 0 to 255)

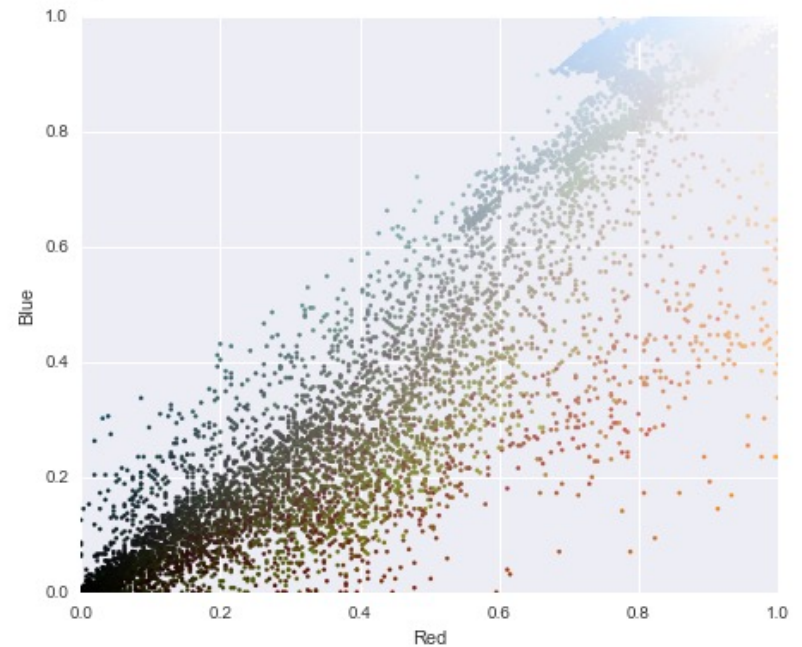
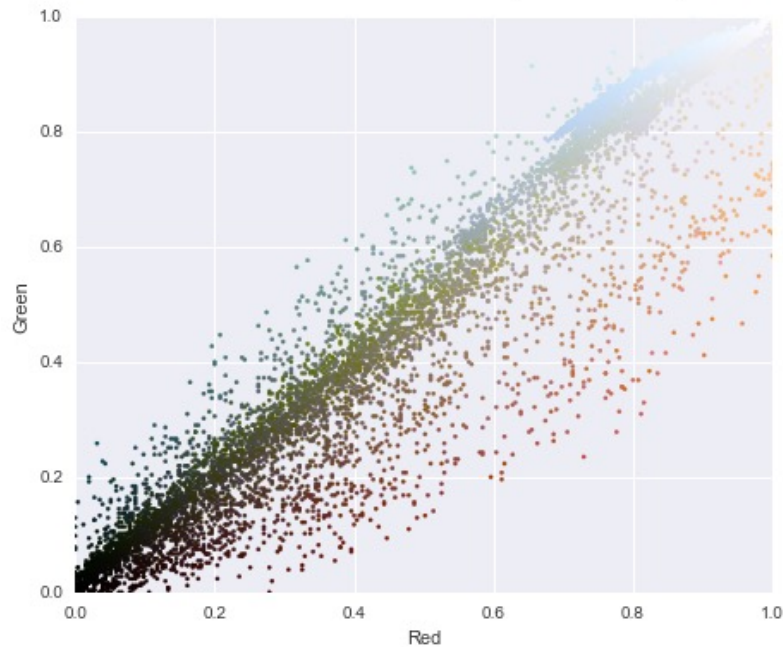


Total number of possible colors =  $2^{24} = 16$  million possible colors  
Let's use clustering to compress the image to far fewer colors!

# Pixels in Original Color Space

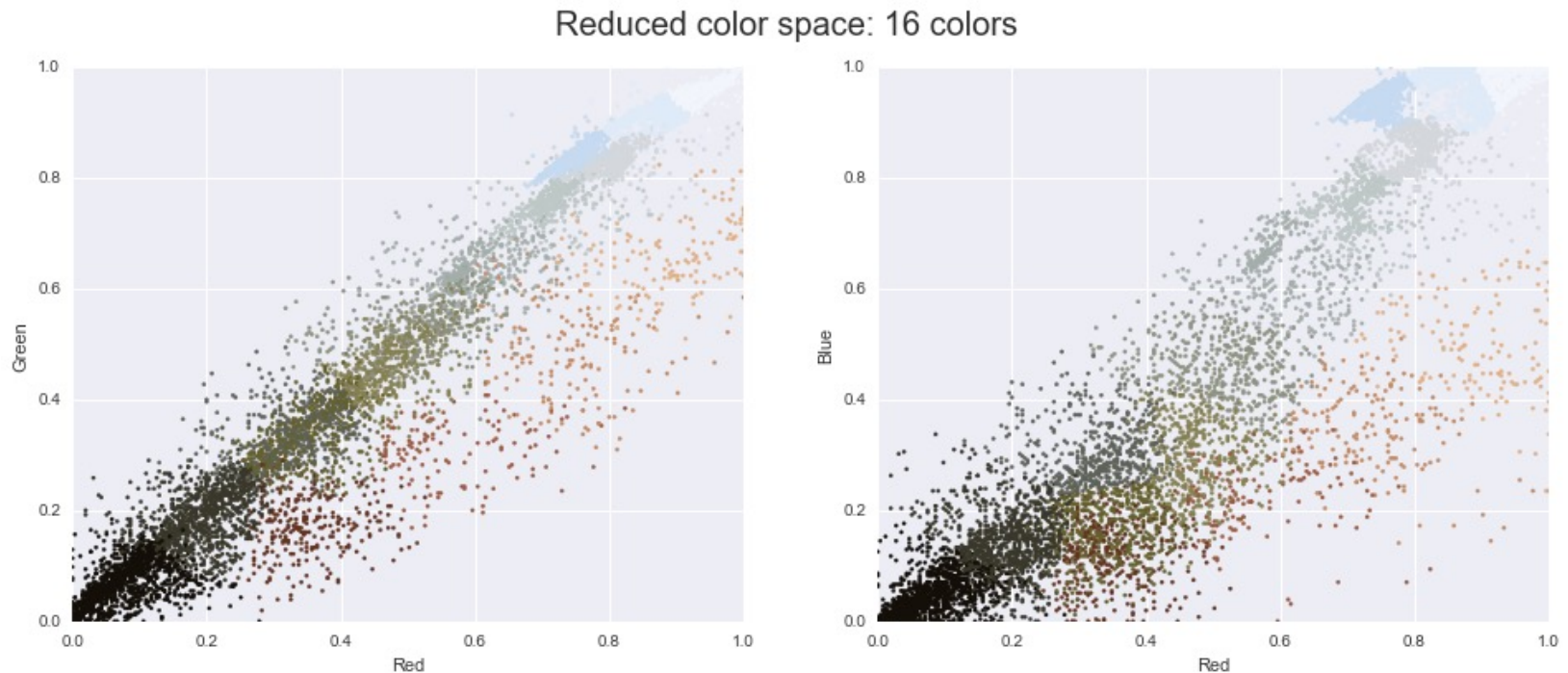
---

Input color space: 16 million possible colors





# Pixels in Quantized Color Space (after K-means)



Results after running K-means,  $K=16$ , on pixel data in 3d (r,g,b) space

Number of bits? Need to specify cluster label per pixel  $\Rightarrow \log_2(K) = 4$  bits  
(also need to transmit cluster centers...but relative to whole image this is small)

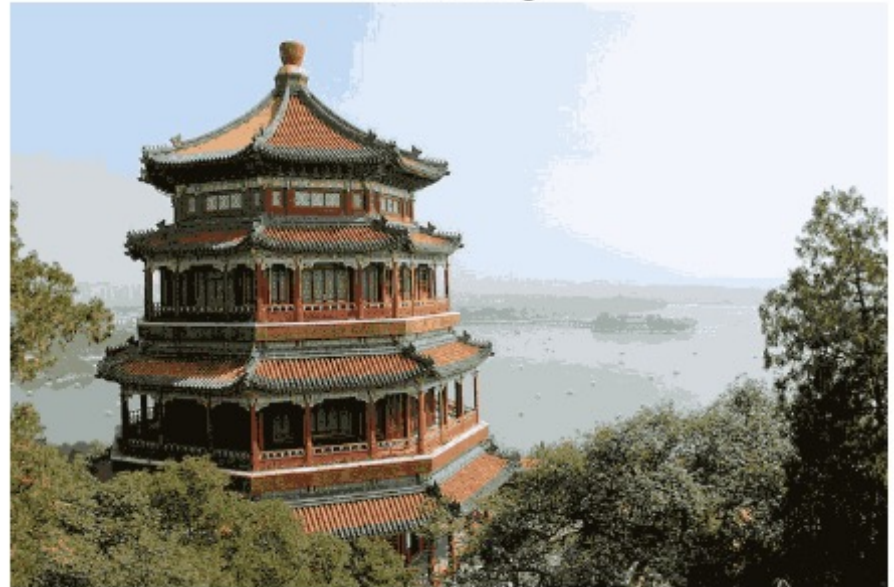
# Compression Results with K-means

Original Image



24 bits per pixel

16-color Image



4 bits per pixel  
(6 times faster to store and transmit!)

See [https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_color\\_quantization.html#sphx-glr-auto-examples-cluster-plot-color-quantization-py](https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html#sphx-glr-auto-examples-cluster-plot-color-quantization-py)

# Compression Results with K-means (K=64)

Original Image



24 bits per pixel

Quantized image (64 colors, K-Means)



6 bits per pixel  
(4 times faster to store and transmit)

See [https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_color\\_quantization.html#sphx-glr-auto-examples-cluster-plot-color-quantization-py](https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html#sphx-glr-auto-examples-cluster-plot-color-quantization-py)

# Wrapup: kMeans Clustering

---

- Useful and simple clustering algorithm
  - Seeks centroids and assignments that minimize Squared Error
- Iterations consist of 2 steps
  - Update the centroids given assignments
  - Update the assignments given centroids
- Converges to a local minimum of the Squared Error
  - Can be sensitive to local minima
  - Various heuristics for improving initialization
- Next Lecture
  - Hierarchical clustering