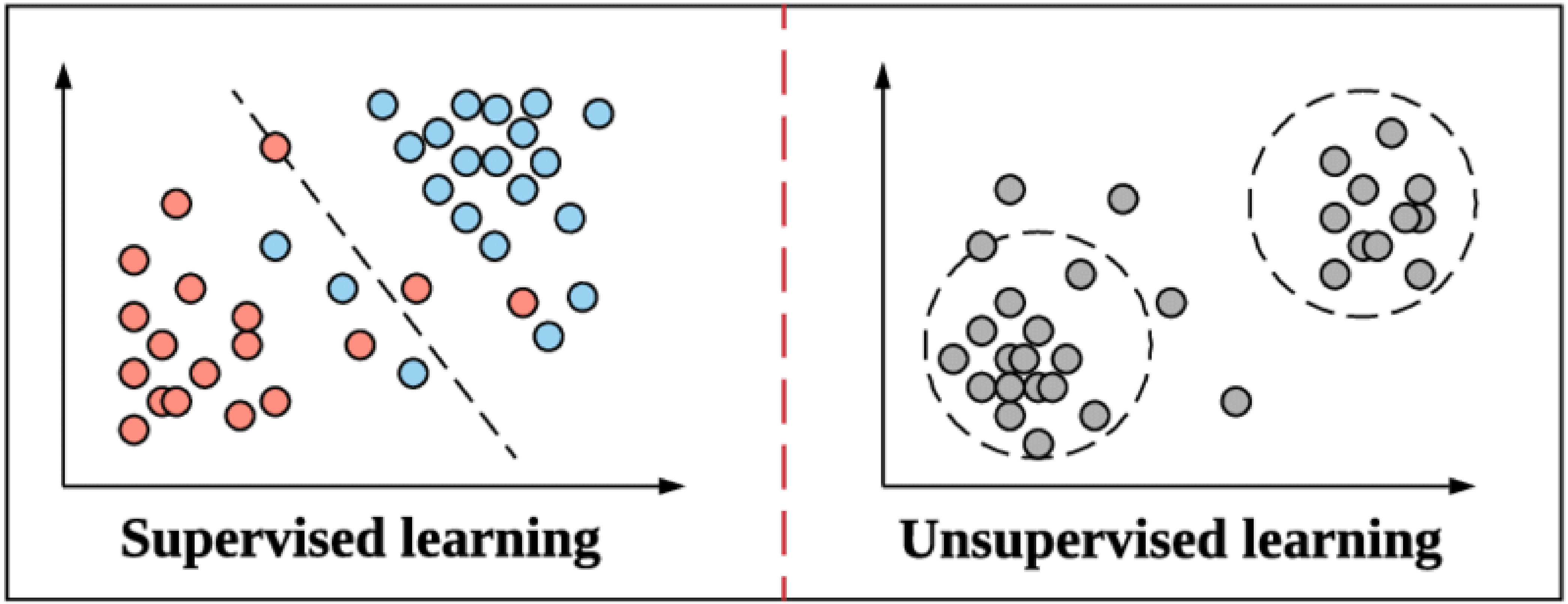
INTRODUCTION TO AI – UNSUPERVISED LEARNING

Philip Mortimer


Supervised learning
Unsupervised learning

RECAP

- Reinforced knowledge of supervised learning by building an ANN to identify cancerous tumours
- Discussed a modern variant of neural networks, CNN's
- CNN's use feature extraction to learn high level details
- This allows AI systems to be scaled for large inputs (e.g. high quality images)
- We discussed TensorFlow, a popular machine learning library
- Used TensorFlow to implement CNN and ANN to tackle handwritten digit recognition. We used MNIST dataset

UNSUPERVISED LEARNING

- Supervised learning involves labelled datasets. These labels often have to be produced by people
- As machine learning requires large amounts of data to be effective, it is often very expensive to label the data required
- This makes supervised learning challenging and costly to implement
- It also means that the strength of a model is limited by the accuracy of the labels
- This makes AI prone to human error and bias

UNSUPERVISED LEARNING

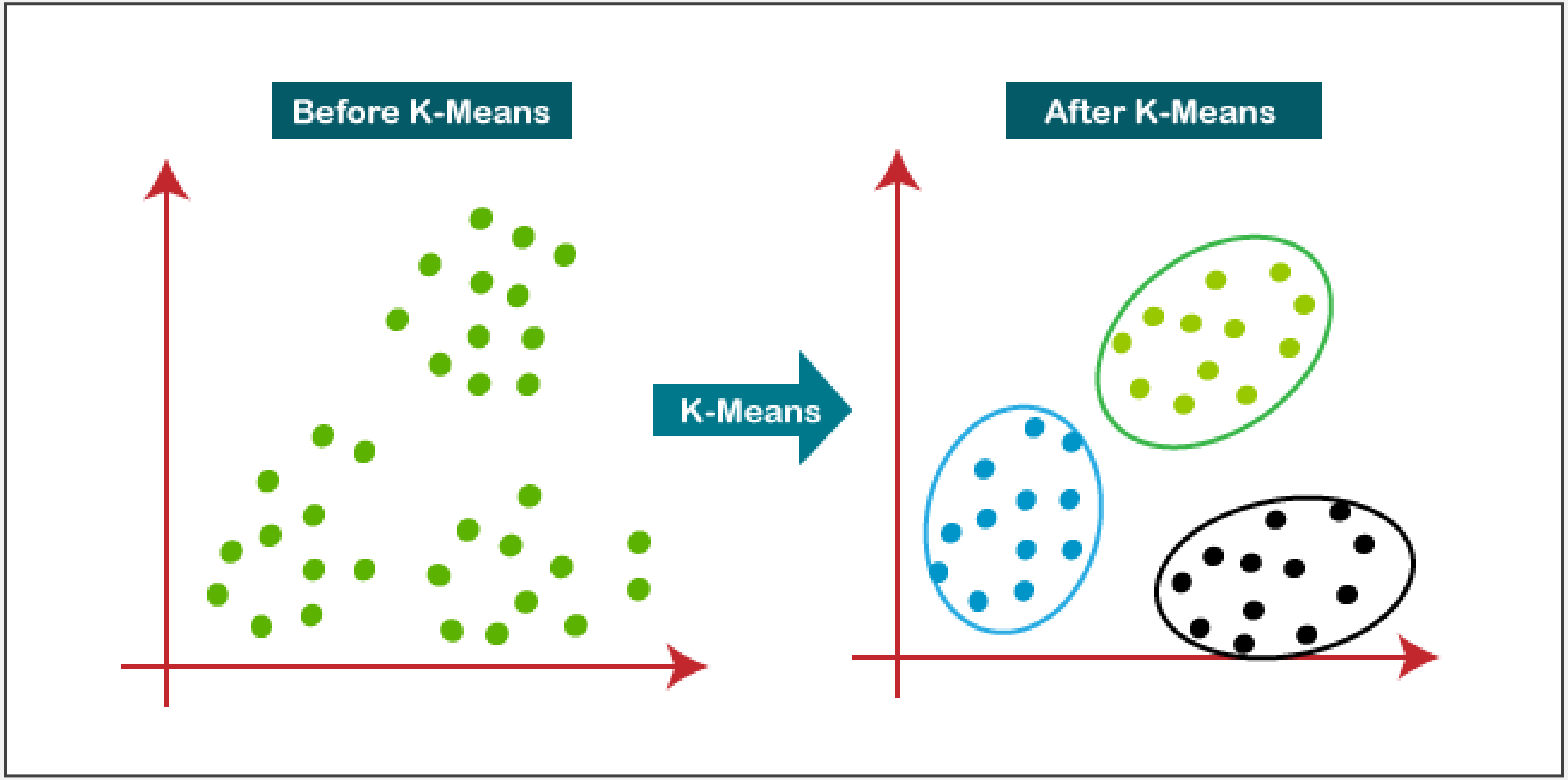
- Therefore, it is often much more appropriate to use unlabelled data
- This is often more cost effective
- There are also a wide range of systems were labelling data is simply inappropriate
- For example, me way have a social media website with a large number of users
- We may want to make an AI that recommends content that they are likely to find interesting
- We as outside observers have no efficient way to label these users interests (we don’t know what that person actually likes)
- However, a computer might be able to look at content that similar people like, and deduce that the user is likely to enjoy similar content

UNSUPERVISED LEARNING

- With supervised learning, there are a number of different algorithms to use (with gradient descent simply being the most common one)
- Unsupervised learning is a bucket term that encapsulates a range of possible algorithms
- Arguably the most popular one is k-means clustering

K-MEANS CLUSTERING

- The idea behind K-means clustering is that we have a set of datapoints and we went to group these data points into k clusters.
- The idea is that each of these clusters should represent similar datapoints
- The programmer species the number of different clusters that the data should be broken into (i.e. the value of k)

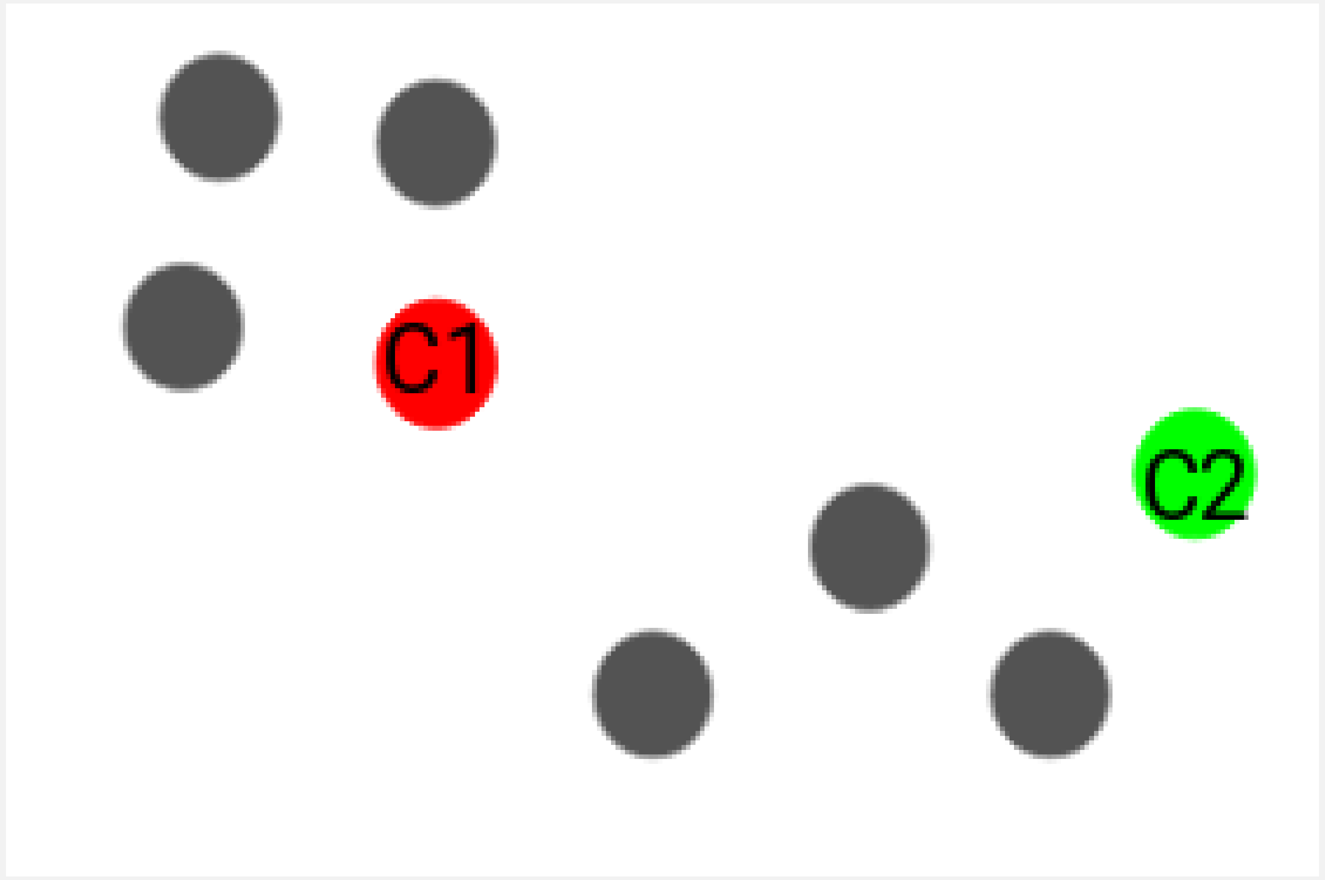

Before K-Means
K-Means
After K-Means

ALGORITHM

- We select k random points to be the initial "centroids" of each cluster
- We assign all points to one of these clusters by choosing the closest centroid for each point
- Now we can recompute the centroid of the cluster by aggregating the distance of all points
- We repeat this process with the newly calculated centroid
- This process occurs until the centroids no longer change (or until a specified number of iterations has passed)

K-MEANS

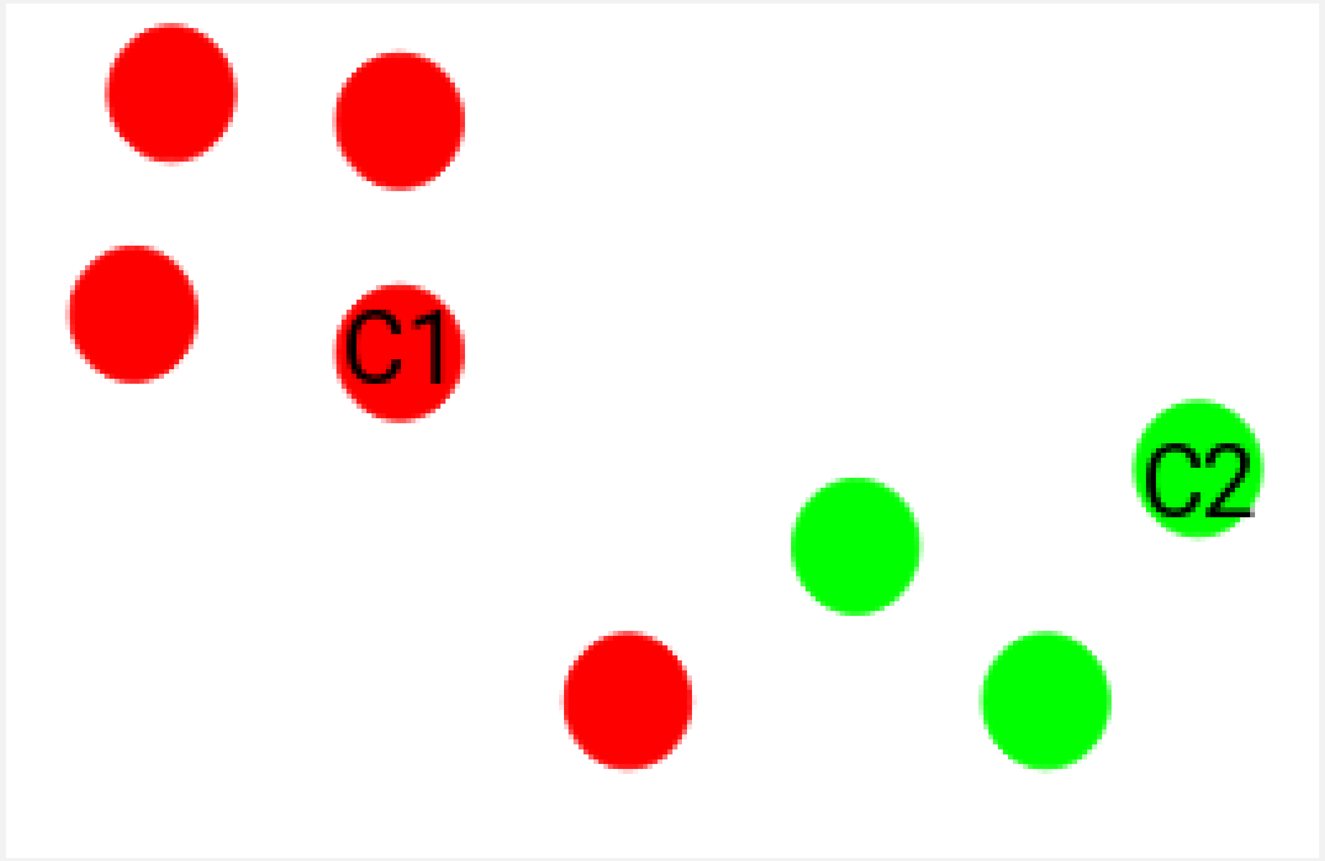
- STEP 1 – Choose K random centroids
- In this case, k = 2



Credit:
https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/

K-MEANS

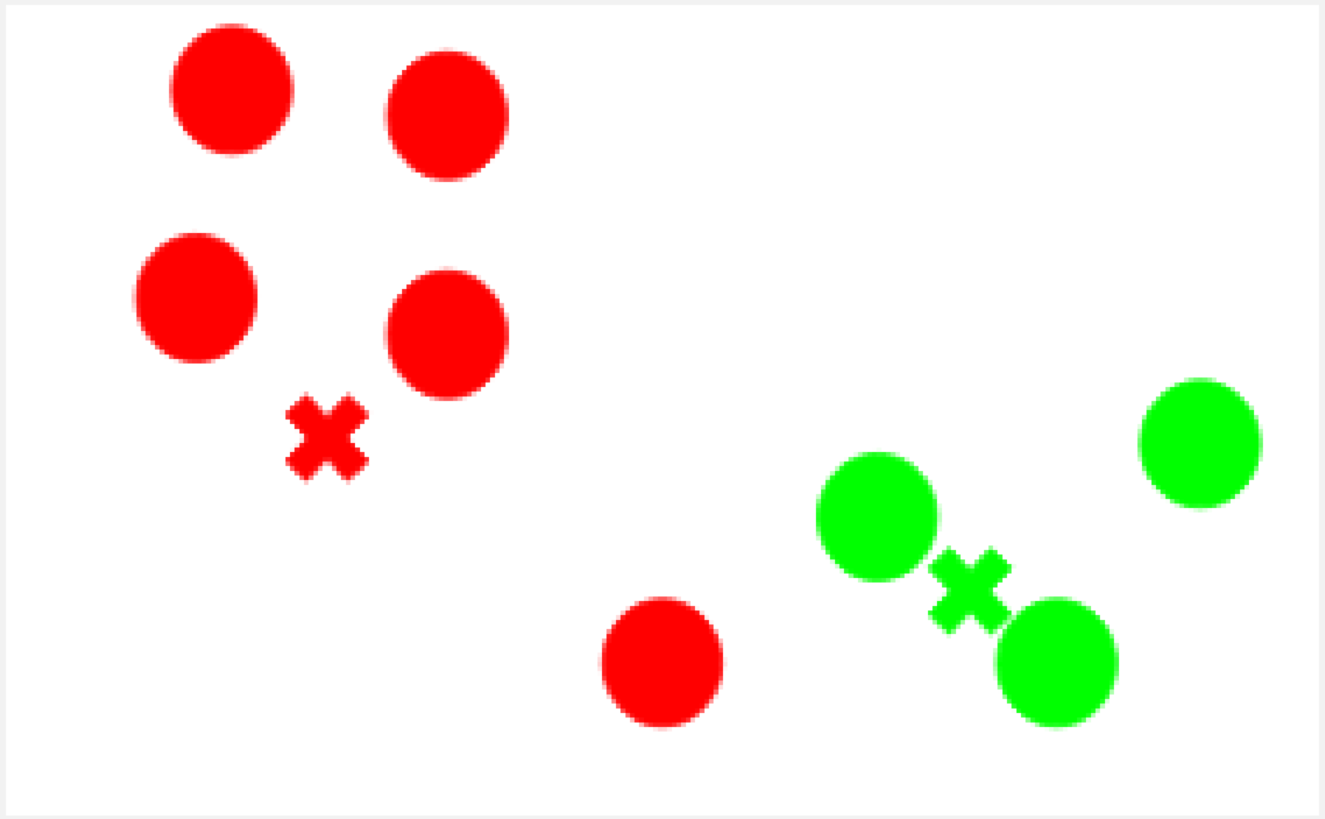
- STEP 2 – Assign all points to one of the k clusters based on which centroid they are closest to



Credit:
https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/

K-MEANS

- STEP 3 – Calculate new centroids by taking that the aggregate location of the points within each cluster



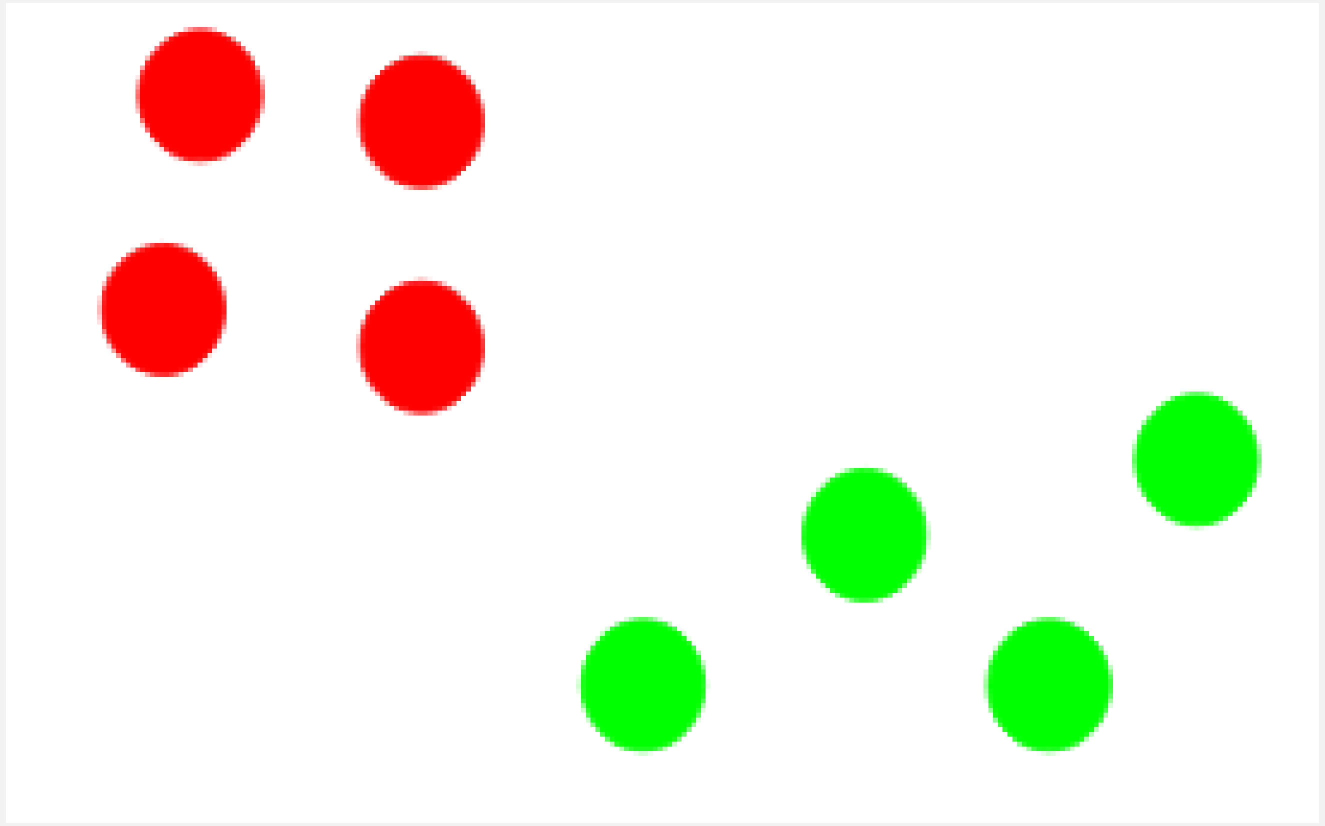
Credit:
https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/

K-MEANS

- Repeat Steps 2 and 3 until :

1. The recalculated centroids are the same as the current centroids
2. A fixed number of iterations has passed(i.e., we've spent enough time computing the solution)
3. All points remain in same cluster over multiple iterations

- This gives us our clusters



Credit: https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/

K-MEANS

- K-Means enables us to group unlabelled data into categories that share common traits
- K-means essentially draws k hyperspheres which have the same dimensionality as the number of datapoint attributes used
- This has a very large number of practical applications

K-MEANS

- Imagine we are a bank with 5 million customers
- Let’s say we wish to give each customer an offer to acquire a credit card
- Clearly, different customers will want different things from a credit card
- We can’t sit down and decide what offer to make to each customer individually
- Hence, we could make 3 different offer packages targeted at different customers
- We may want one package to target people based on income. So we could split our packages into three classes: “high income, medium income, low income”
- Using records of our customers, we may decide to cluster them based on the following attributes: “amount in bank, value of outstanding loans and bank balance five years ago”
- This gives us a three dimensional space of five million datapoints

K-MEANS

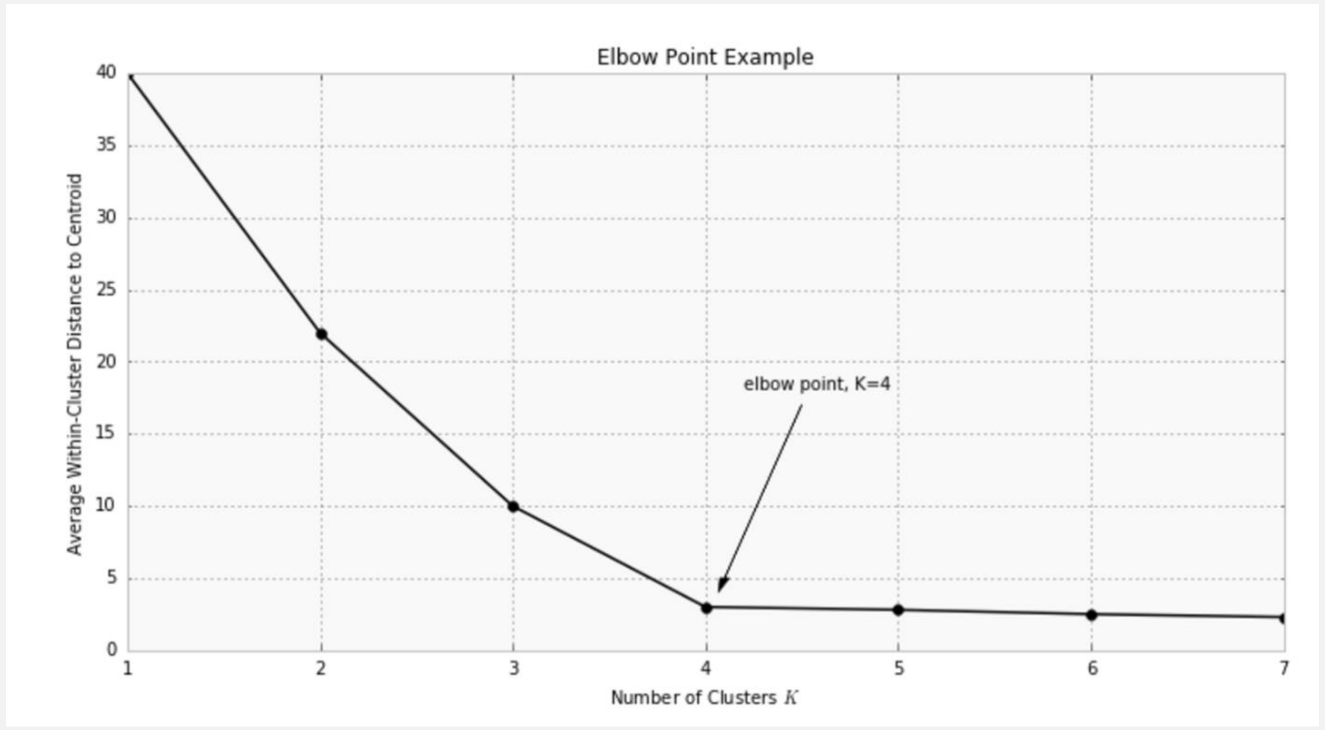
- We then run the k means algorithm using k = 3 to cluster our data
- We know have 5 million customers neatly divided and can target out offers accordingly
- Note of course, that k means tells us nothing about **what** each cluster actually means. We need some way of figuring this out ourselves
- This could be achieved by human verification, our smart algorithms

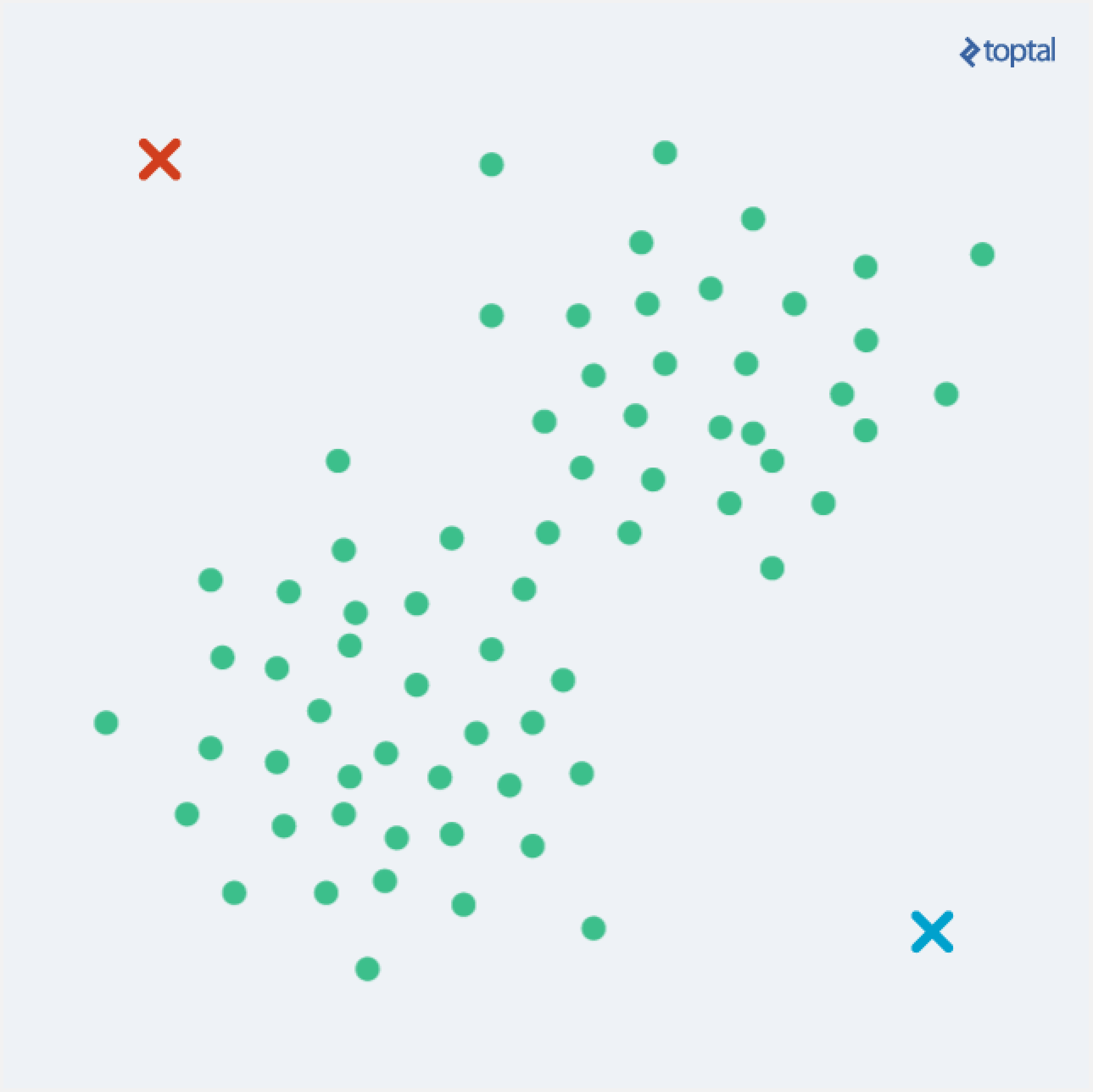
K-MEANS

- In many cases, we won’t know what value of k is most suitable
- A method known as the “elbow method” is often used. It involves choosing the value of k at which the distance between points in the cluster stops decreasing quickly

ELBOW METHOD

- We can see that at k = 4, the average distance within a cluster stops decreasing as rapidly
- At this points, the points have been grouped effectively
- We don't want k to be too large, as more groups often means more unneeded complexity





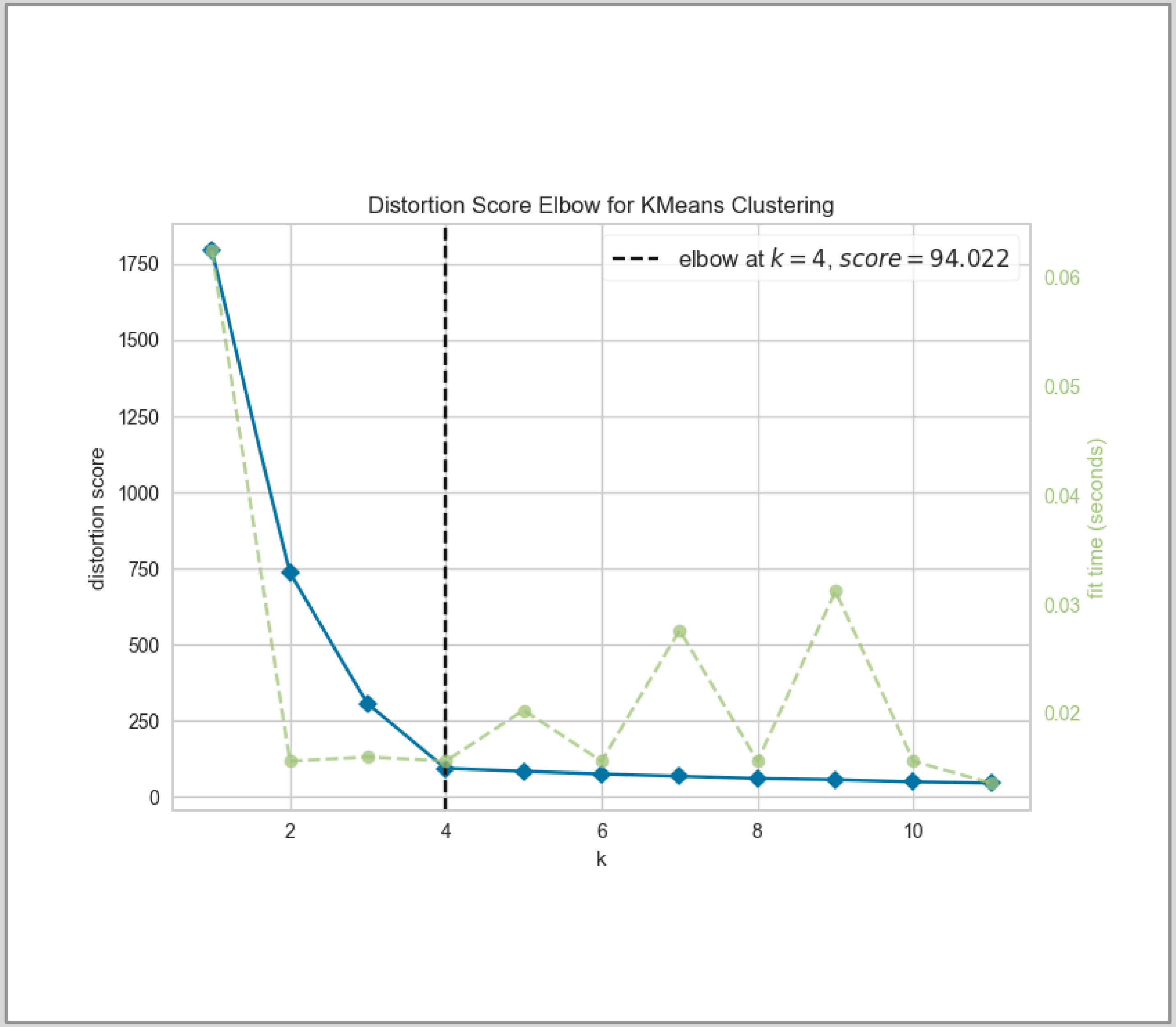
K-MEANS

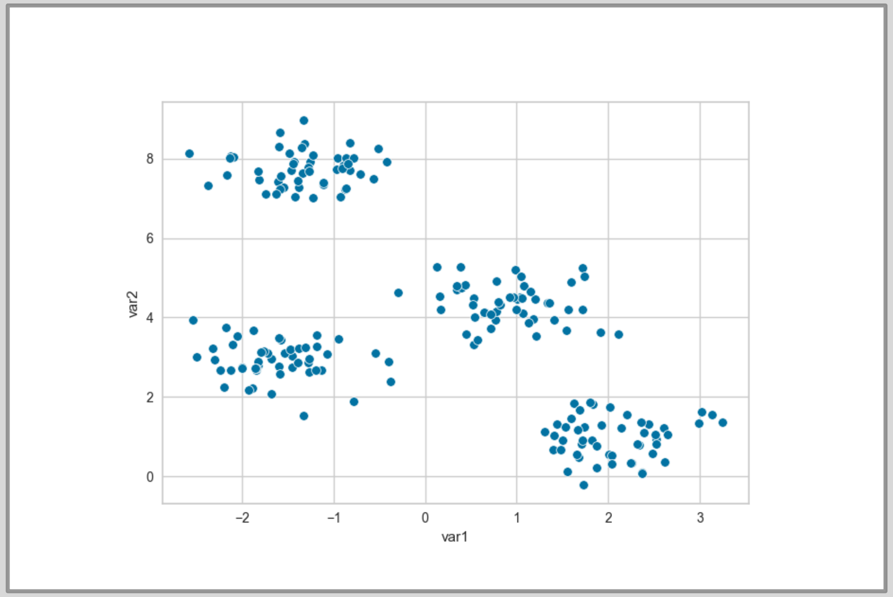
- Many algorithms have already been written and tested by other programmers. We can simply reuse these if we wish (which saves us developing from scratch).

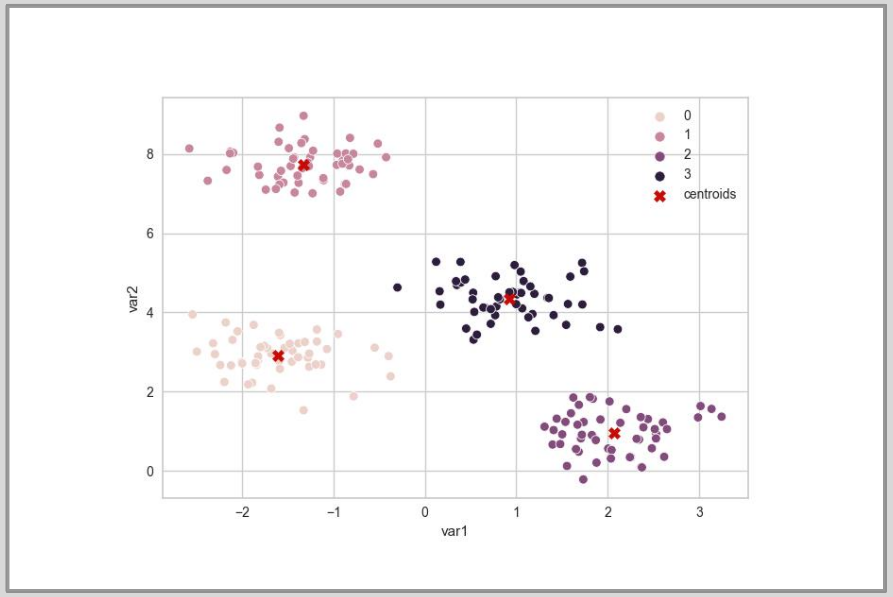
```
from sklearn.cluster import KMeans


def kMeansModel(X):
  #X is data
  model = KMeans(n_clusters=5)
  model.fit(X)
  return model
```

K-MEANS

- I've coded my own k-means learning model that clusters data into groups
- Let's look at the code. The code can be found here: https://github.com/philipmortimer/AI-Course/blob/main/Programs/Part%206/kMeans/main.py


Distortion Score Elbow for KMeans Clustering
elbow at $k = 4$, $score = 94.022$
distortion score
fit time (seconds)
k
0
250
500
750
1000
1250
1500
1750
0.02
0.03
0.04
0.05
0.06
2
4
6
8
10


var1
var2

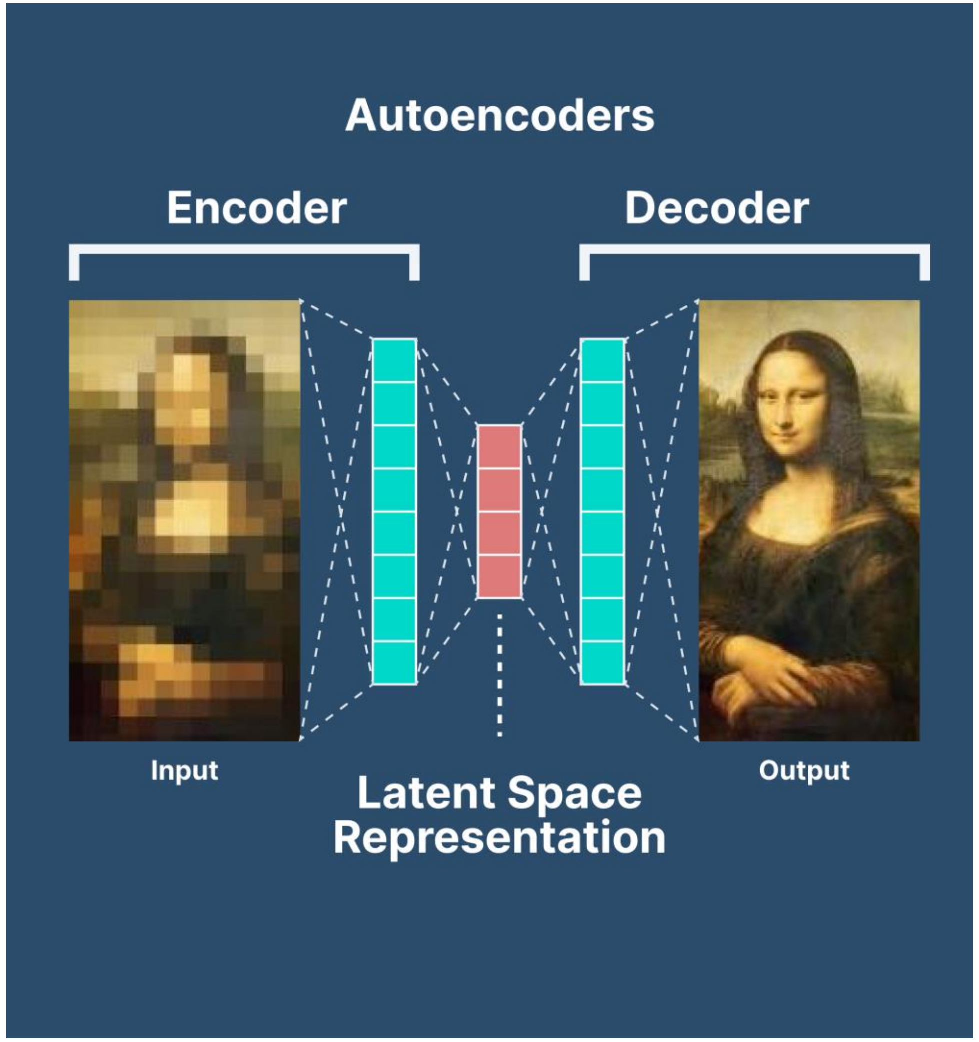

0
1
2
3
centroids
var1
var2

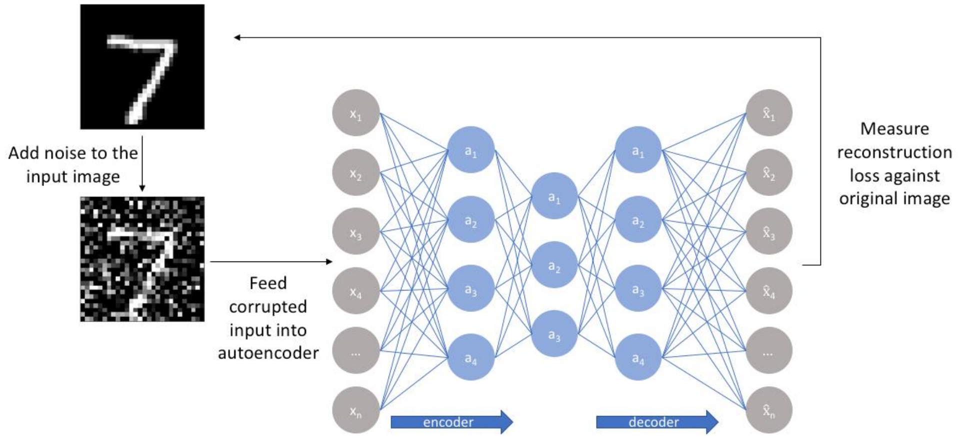
AUTO-ENCODERS

- Unsupervised learning covers a wide range of techniques and algorithms
- Image compression often uses a neural network structure known as an autoencoder
- These networks compress images by reducing their dimensionality by extracting the key features
- Auto-encoders are often trained on specific image types (e.g. only images of cats) in order to make them more effective at dimensionality reduction

AUTO-ENCODERS

- Auto-encoders require an unlabelled dataset (e.g. of images)
- We use supervised learning techniques to train the network (i.e. gradient descent)
- We have two networks, namely an “encoder” and a “decoder”
- The encoder network converts the image into an abstract representation that uses less data than the original image
- The decoder converts the compressed format into the original image
- By combining the two components, we have a network that can use data both as input and as a label
- Sometimes the input image is blurred slightly to add “noise” to the input


Autoencoders
Encoder
Decoder
Input
Latent Space Representation
Output


Add noise to the input image
Feed corrupted input into autoencoder
Measure reconstruction loss against original image
encoder
decoder

AUTO-ENCODER

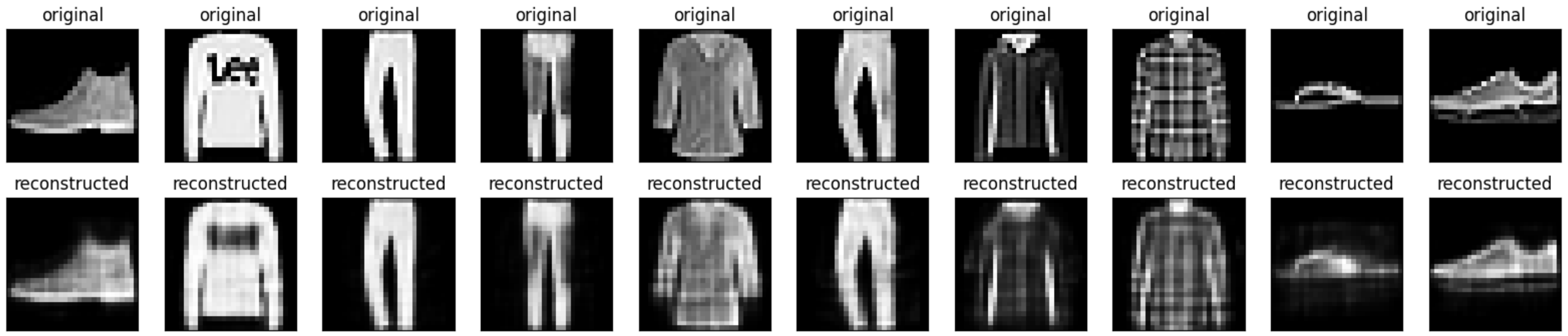
- As the compressed representation is dimensionally smaller than the input, the network must attempt to find a smaller representation of the image that allows the image to be best recreated
- Autoencoders are an example of lossy compression
- Autoencoders are often used to reduce the dimensionality of large datasets to make them easier to use (e.g. for machine learning techniques)
- For example, autoencoders have been applied to allow for more efficient representations of chess boards

AUTO-ENCODERS

- Training of auto-encoders is expensive
- The problem with auto-encoders is choosing the correct network layout (and other relevant supervised learning hyperparameters) to maximise compression, reconstruction accuracy and to minimise compute time
- This is often very difficult and time consuming

AUTO-ENCODERS

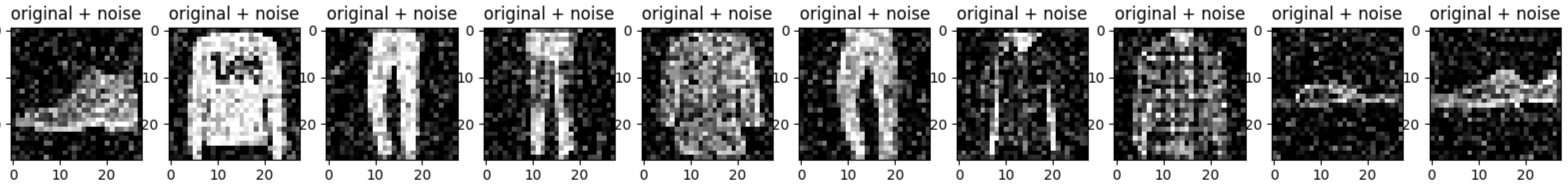
- I've trained my own autoencoder using TensorFlow
- This autoencoder takes in a 784 pixel image of a clothing item. This is then compressed to 64 pixels using the auto-encoder
- First, lets discuss the encoder. The encoder must take a 784 pixel input and convert it to a 64 value output. To achieve this, I have used no hidden layers (to maximise speed)
- The output layer contains 64 neurons.
- I have chosen to use the sigmoid activation
- For the decoder network, we must convert 64 values into 784 values
- I have once more elected to use no hidden layers to speed up computation
- Let's look at code! https://github.com/philipmortimer/AI-Course/blob/main/Programs/Part%206/autoencoder/main.py



AUTO-ENCODERS

AUTO-ENCODERS

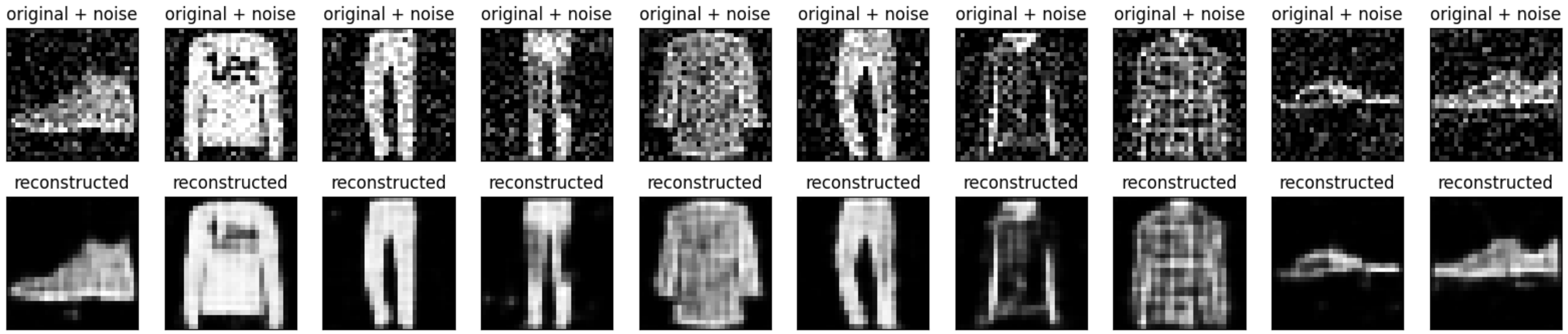
- Next, let's attempt to train an auto-encoder that removes noise from images
- To do this, we programmatically generate "noise" which has the effect of blurring our images somewhat
- We want to take this blurred image as input and return the unblurred image
- This is often used for post-processing of photos
- Additionally, it is used to increase the dataset size for traditional compression based autoencoders



NOISY IMAGES

DE-NOISING AUTOENCODER

- Notice, how we are mimicking supervised learning techniques.
- This system is unsupervised however, as it is using unlabelled data as both input and output
- For our de-noising autoencoder, I have decided to use convolutional layers in order to enable effective feature extraction
- Typically, convolutional approaches are more versatile and effective than just using ANN's
- Let's look at code! https://github.com/philipmortimer/AI-Course/blob/main/Programs/Part%206/autoencoder/denoiseAutencoder.py



DE-NOISED IMAGES

PRINCIPAL COMPONENT ANALYSIS (PCA)

- Principal component analysis (PCA) is another technique commonly used for dimension reduction
- Let's say we have a database containing millions of entries
- Each record stores three data attributes: x, y and z.
- Thus, each record is three dimensional
- PCA allows us to compress this dataset to 1d, 2d or 3d
- It does this by exploiting relationships between x, y and z.
- If it turns out that $x = 0.5y + 2z$, then we can reduce dimensionality as we clearly don't actually need to store x. It can simply be calculated from y and z.

PCA

- Before, doing anything else, it is very important that we standardise all of our attributes so that they have the same range
- This is important as PCA essentially captures maximum variance
- If one variable has range [0, 255] whilst another has [0, 1], the first variable will typically produce a much larger variance. This is misleading and hence both would need to be altered to fit same range (e.g. [0, 1])

PCA

- Say we have records with n variables. This data is said to be n-dimensional
- We can then compute the covariance matrix for all possible combination of variable pairs.
- If the covariance is positive, it suggests that as one variable, increases, so does the other
- Similarly, if it's negative, it suggests that they are inversely correlated
- Small covariances suggest minimal correlation

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

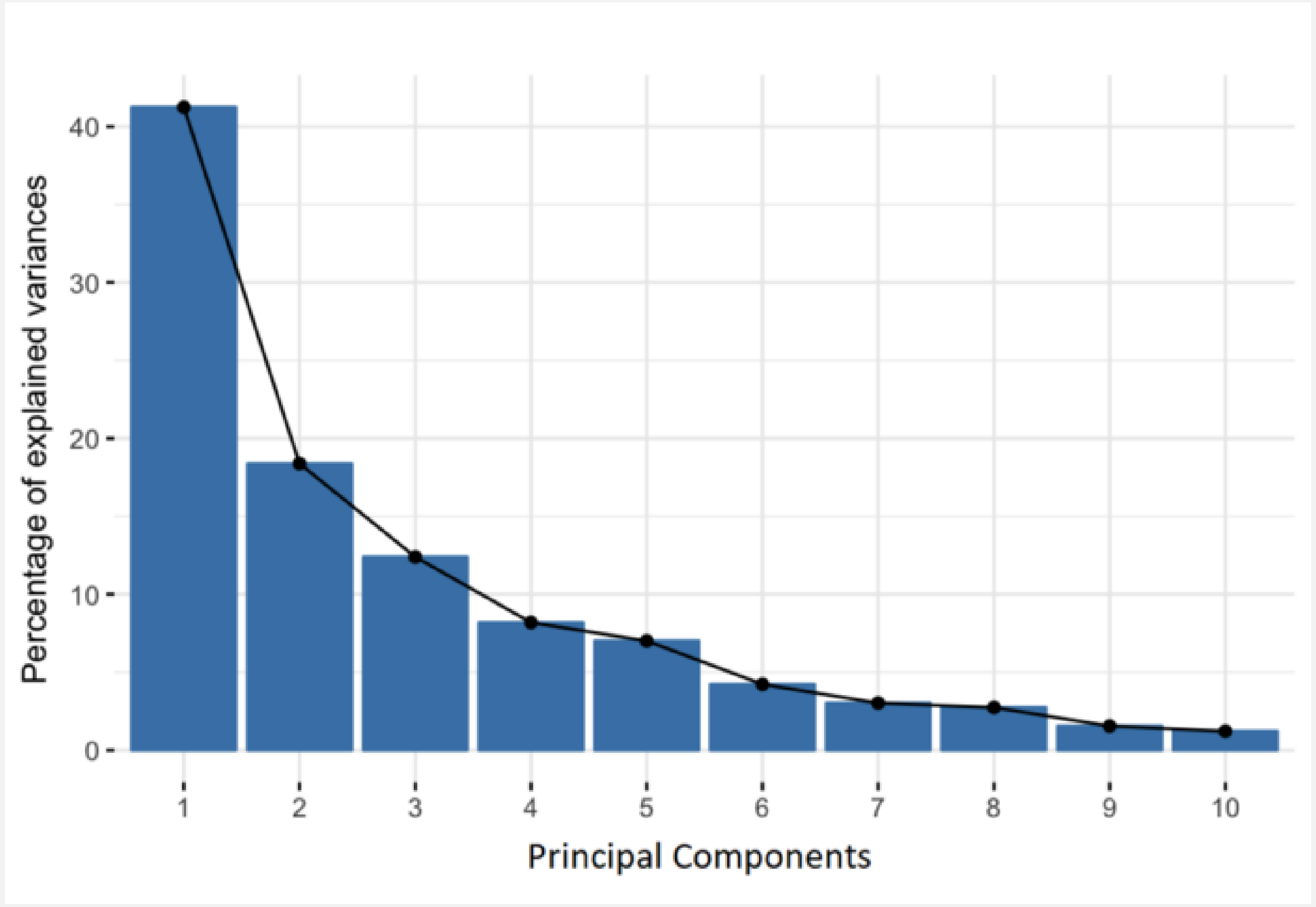
Covariance matrix for 3d data

PCA

- The idea of PCA is to convert n variables into k variables where $k <= n$
- To do this we will create k principal components. The idea is that we fit the most amount of information possible into the first principal component. Then as much into the second component and so on
- This allows us to use the first k principal components and ignore the rest, as these first k components will encode the most information

PCA

- This is an example for data which has 10 attributes
- One principal component attribute (which we must calculate) can explain about 40% of the data variance



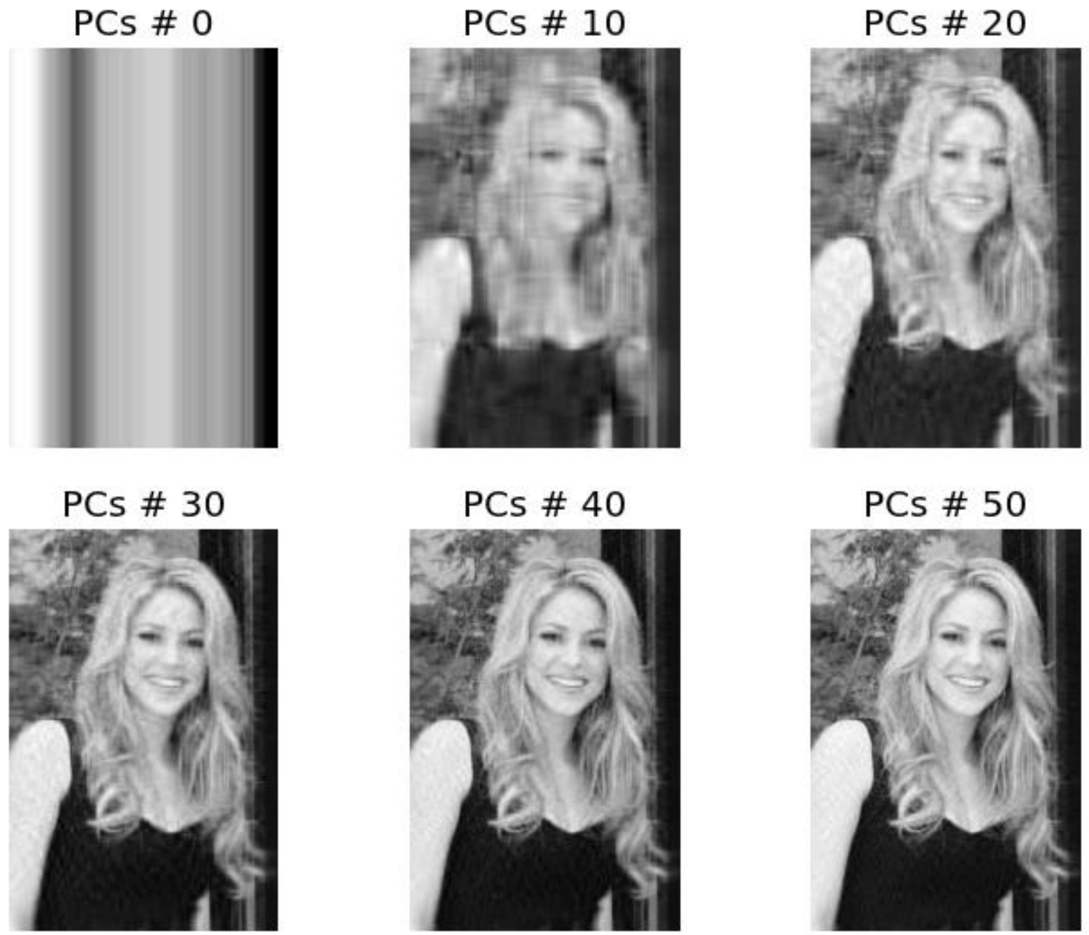
Credit: https://builtin.com/data-science/step-step-explanation-principal-component-analysis

PCA

- If we calculate all of the eigenvectors of our covariance matrix(and normalise them where we needed), we get a set of vectors
- These vectors form an orthonormal basis
- Additionally, the corresponding eigenvalues indicate order of importance of each eigenvector (/principal component).
- This allows us to rank each principal component as seen on the previous page
- You don't have to understand the maths behind this, just that it allows us to effectively reduce dimensionality
- We can select the number of dimensions / accuracy we want
- It's important to note that the principal components lack interpretability

```
from sklearn.decomposition import PCA


def prin_comps(data, n_comps):
  pca = PCA(n_components=n_comps) #Reduces dimensionaility to n_comps
  principalComponents = pca.fit_transform(data)
  return principalComponents
```

PCA


PCs # 0
PCs # 10
PCs # 20
PCs # 30
PCs # 40
PCs # 50

SUMMARY

- Unsupervised learning is a machine learning technique applied to datasets which are unlabelled
- There are a very wide range of algorithms used in unsupervised learning
- K-means clustering focuses on grouping datapoints into a certain number of groups. These clusters will group datapoints that are most similar to them
- Similarity between datapoints is calculated via the Euclidian distance in the vector space formed by the attributes of the points
- Autoencoders are often used to reduce dimensionality of data
- Autoencoders consist of an **encoder** neural network and a **decoder** neural network

SUMMARY

- The encoder network reduces the dimensionality of the network to a compressed format
- The decoder network takes this compressed format and attempts to convert it back to the original datapoint
- This can often be thought of as a feature extraction network
- Principal Component Analysis (PCA) is another dimensionality reduction technique
- The idea is that data may consists of n attributes. In the case of an image, this may mean n pixels
- PCA uses complicated maths to create an orthonormal basis with n attributes.

SUMMARY

- PCA essentially converts a data item of n attributes and calculates n different attributes to use.
- As much information as possible is put into the first attribute. As much information as possible is put into the second attribute
- This allows us to reduce the dimensionality of data by dropping as many of the least important attributes as we wish
- We can balance the accuracy of the compression with the reduction in data size as we choose
- There are many other useful unsupervised techniques

QUESTIONS

FURTHER READING

- K-means clustering: https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/
- Autoencoder: https://www.jeremyjordan.me/autoencoders/ https://www.v7labs.com/blog/autoencoders-guide
- Principal Component Analysis (PCA): https://builtin.com/data-science/step-step-explanation-principal-component-analysis