Data Mining & Informatics Lecture #4

Amin Noroozi, Anirban Chakraborty
University of Wolverhampton
✉ a.noroozifakhabi@wlv.ac.uk, a.chakraborty@wlv.ac.uk

Unsupervised learning

Data mining

- **NOTE**
- For some of the examples in this lecture, we use the online editor as follows
- https://www.w3schools.com/python/trypython.asp?filename=demo_compiler

Data mining

- **Basics – Unsupervised learning**
- Unsupervised learning is the training of a machine using information that is neither classified nor labelled and allowing the algorithm to act on that information without guidance.
- Here the task of the machine is to group unsorted information according to similarities, patterns, and differences without any prior training of data.
- Unlike supervised learning, no teacher is provided, which means data are not labelled by a supervisor. Therefore the machine is restricted to finding the hidden structure in unlabeled data by itself and being trained without supervision.
- There are two main categories for unsupervised learning: **clustering** and **dimensionality reduction.**

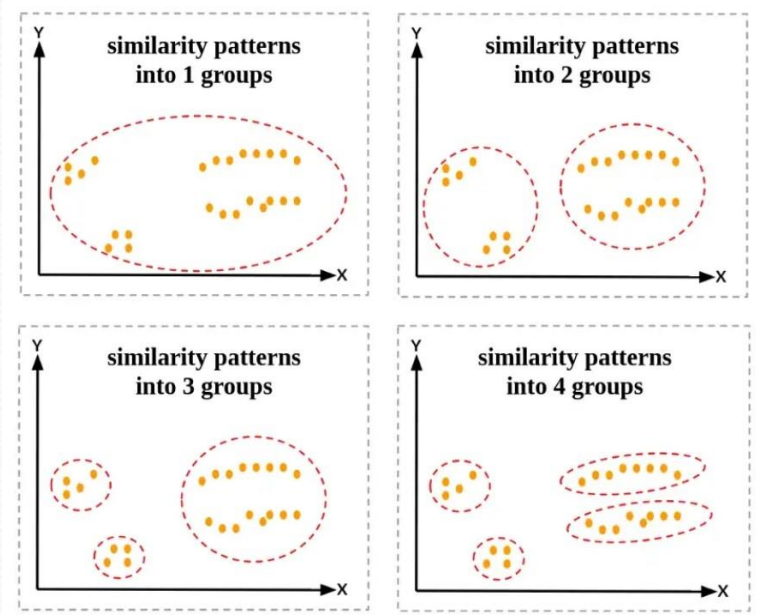
Unsupervised learning

- **Clustering**
- Clustering is a Machine Learning technique that involves the grouping of data.

- The machine has no idea about the features of dogs and cats because the data are not labeled by a supervisor. But it can categorize them according to their **similarities**, **patterns**, and **differences.**
- It can categorize the above picture into two parts. The first may contain all pics having dogs in them, and the second part may contain all pics having cats in them.

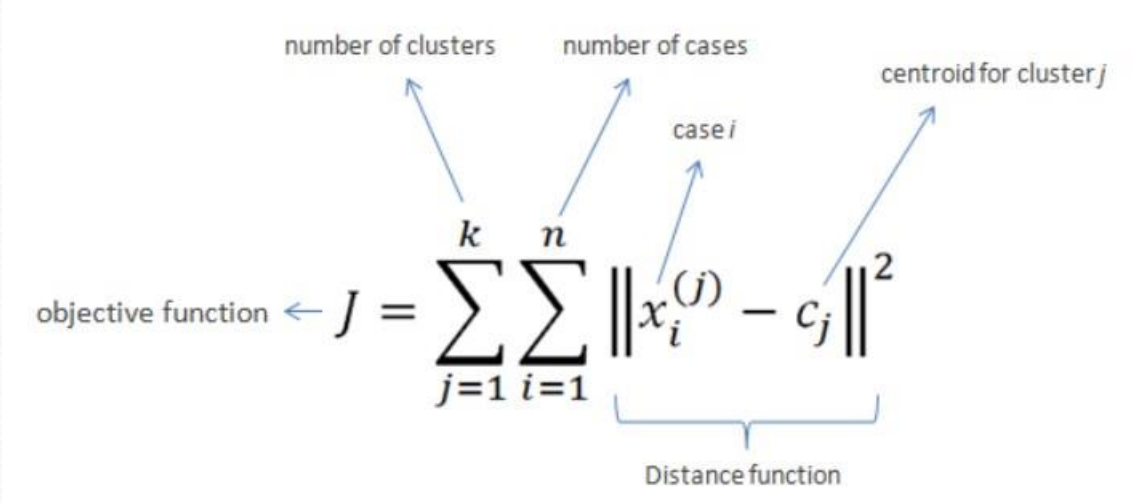
Unsupervised learning

- **Clustering methods – K-means**
- K-means clustering is a simple and elegant approach for partitioning a data set into K distinct, non-overlapping clusters.
- To perform K-means clustering, we must first specify the desired number of clusters K; then the K-means algorithm will assign each observation to exactly one of the K clusters. Therefore K is a hyperparameter.



Unsupervised learning

- **Clustering methods – K-means**
- The K-means algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares
- The K-means algorithm aims to choose centroids that minimise the inertia, or within-cluster sum-of-squares objective function as follows:



Unsupervised learning

- **Clustering methods – K-means**
- The K-means consists of the following steps:

1- Clusters the data into k groups where k is predefined.

2- Select k points at random as cluster centers.

3- Assign objects to their closest cluster center according to the Euclidean distance function.

4- Calculate the centroid or mean of all objects in each cluster.

5- Repeat steps 2,, 3 and 4 until the same points are assigned to each cluster in consecutive rounds (iterations), or the difference between the values of the objective function in two consecutive iterations drops below a threshold.

Unsupervised learning

- **Clustering methods – K-means**

Example

The age of 19 visitors to a website is as follows:

15,15,16,19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65

Cluster the visitors according to their age into 2 groups.

Initial clusters (random centroid or average):

$$k = 2$$

$$c_1 = 16$$

$$c_2 = 22$$

$$Distance\ 1 = |x_i - c_1|$$

$$Distance\ 2 = |x_i - c_2|$$

Unsupervised learning

- **Clustering methods – K-means**

Example

Iteration 1:

$c_1 = 15.33$

$c_2 = 36.25$

x_i	c_1	c_2	Distance 1	Distance 2	Nearest Cluster	New Centroid
15	16	22	1	7	1	
15	16	22	1	7	1	**15.33**
16	16	22	0	6	1	
19	16	22	3	3	2	
19	16	22	3	3	2	
20	16	22	4	2	2	
20	16	22	4	2	2	
21	16	22	5	1	2	
22	16	22	6	0	2	
28	16	22	12	6	2	
35	16	22	19	13	2	
40	16	22	24	18	2	**36.25**
41	16	22	25	19	2	
42	16	22	26	20	2	
43	16	22	27	21	2	
44	16	22	28	22	2	
60	16	22	44	38	2	
61	16	22	45	39	2	
65	16	22	49	43	2	

Unsupervised learning

- **Clustering methods – K-means**

Example

Iteration 2:

$c_1 = 18.56$

$c_2 = 45.90$

x_i	c_1	c_2	Distance 1	Distance 2	Nearest Cluster	New Centroid
15	15.33	36.25	0.33	21.25	1	
15	15.33	36.25	0.33	21.25	1	
16	15.33	36.25	0.67	20.25	1	
19	15.33	36.25	3.67	17.25	1	
19	15.33	36.25	3.67	17.25	1	**18.56**
20	15.33	36.25	4.67	16.25	1	
20	15.33	36.25	4.67	16.25	1	
21	15.33	36.25	5.67	15.25	1	
22	15.33	36.25	6.67	14.25	1	
28	15.33	36.25	12.67	8.25	2	
35	15.33	36.25	19.67	1.25	2	
40	15.33	36.25	24.67	3.75	2	
41	15.33	36.25	25.67	4.75	2	
42	15.33	36.25	26.67	5.75	2	
43	15.33	36.25	27.67	6.75	2	**45.9**
44	15.33	36.25	28.67	7.75	2	
60	15.33	36.25	44.67	23.75	2	
61	15.33	36.25	45.67	24.75	2	
65	15.33	36.25	49.67	28.75	2	

Unsupervised learning

- **Clustering methods – K-means**

Example

Iteration 3:

$c_1 = 19.50$

$c_2 = 47.89$

x_i	c_1	c_2	Distance 1	Distance 2	Nearest Cluster	New Centroid
15	18.56	45.9	3.56	30.9	1	
15	18.56	45.9	3.56	30.9	1	
16	18.56	45.9	2.56	29.9	1	
19	18.56	45.9	0.44	26.9	1	
19	18.56	45.9	0.44	26.9	1	
20	18.56	45.9	1.44	25.9	1	**19.50**
20	18.56	45.9	1.44	25.9	1	
21	18.56	45.9	2.44	24.9	1	
22	18.56	45.9	3.44	23.9	1	
28	18.56	45.9	9.44	17.9	1	
35	18.56	45.9	16.44	10.9	2	
40	18.56	45.9	21.44	5.9	2	
41	18.56	45.9	22.44	4.9	2	
42	18.56	45.9	23.44	3.9	2	
43	18.56	45.9	24.44	2.9	2	**47.89**
44	18.56	45.9	25.44	1.9	2	
60	18.56	45.9	41.44	14.1	2	
61	18.56	45.9	42.44	15.1	2	
65	18.56	45.9	46.44	19.1	2	

Unsupervised learning

- **Clustering methods – K-means**

Example

Iteration 4:

$c_1 = 19.50$

$c_2 = 47.89$

x_i	c_1	c_2	Distance 1	Distance 2	Nearest Cluster	New Centroid
15	19.5	47.89	4.50	32.89	1	
15	19.5	47.89	4.50	32.89	1	
16	19.5	47.89	3.50	31.89	1	
19	19.5	47.89	0.50	28.89	1	
19	19.5	47.89	0.50	28.89	1	
20	19.5	47.89	0.50	27.89	1	**19.50**
20	19.5	47.89	0.50	27.89	1	
21	19.5	47.89	1.50	26.89	1	
22	19.5	47.89	2.50	25.89	1	
28	19.5	47.89	8.50	19.89	1	
35	19.5	47.89	15.50	12.89	2	
40	19.5	47.89	20.50	7.89	2	
41	19.5	47.89	21.50	6.89	2	
42	19.5	47.89	22.50	5.89	2	
43	19.5	47.89	23.50	4.89	2	**47.89**
44	19.5	47.89	24.50	3.89	2	
60	19.5	47.89	40.50	12.11	2	
61	19.5	47.89	41.50	13.11	2	
65	19.5	47.89	45.50	17.11	2	

- No change between iterations 3 and 4 has been noted. By using clustering, 2 groups have been identified 15-28 and 35-65.

Unsupervised learning

- **Clustering methods – K-means**

Example - Python

```
# Visualisation


#Three lines to make the online compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')
###
import matplotlib.pyplot as plt
x = [15,15,16,19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65]
plt.plot(x,'o')
plt.show()
#Two lines to make the online compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

Unsupervised learning

- **Clustering methods – K-means**

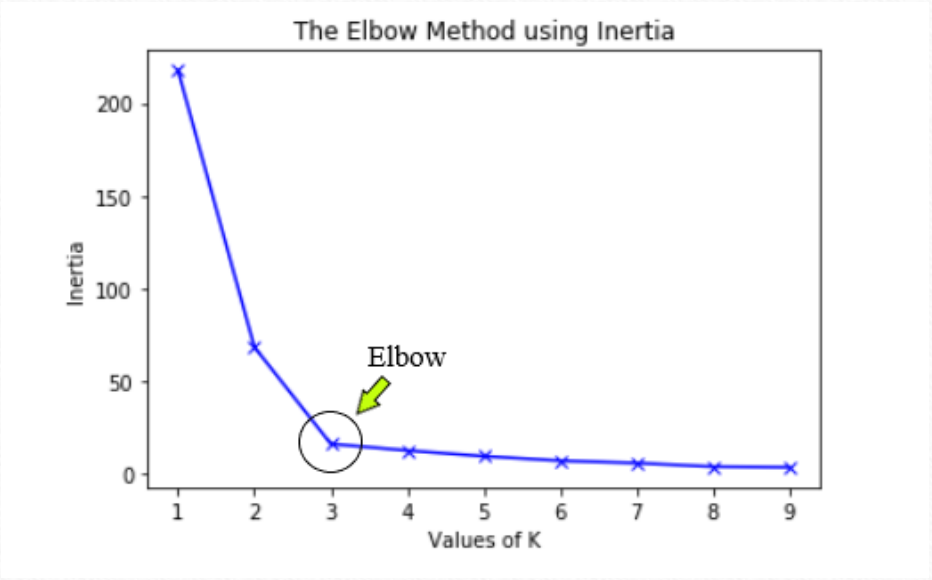
Example - Python

```
#Three lines to make the online compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')
##########
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
x=[15,15,16,19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65]
data=np.array([15,15,16,19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65]).reshape(-1, 1)
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)
plt.scatter(np.linspace(1,19,19),x, c=kmeans.labels_)
plt.show()
print(x)
print(kmeans.labels_)
print(kmeans.cluster_centers_)
print(kmeans.inertia_)
##########Two lines to make the online compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

Unsupervised learning

- **Clustering methods – K-means – elbow method for finding optimum k**

  - The number of clusters in K-means is a hyperparameter, i. e., it needs to be set manually.
  - We can use the elbow method to set k.
  - The elbow method runs k-means clustering on the dataset for a range of values for k (say from 1-10) and then for each value of k computes the inertia objective function. To determine the optimal number of clusters, we have to select the value of k at the "elbow" ie the point after which the inertia starts decreasing in a linear fashion



Unsupervised learning

- **Clustering methods – K-means – elbow method for finding optimum k**

Example

```
#Three lines to make the online compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')
##########
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
x=[15,15,16,19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65]
data=np.array([15,15,16,19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65]).reshape(-1, 1)
inertias = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
#########Two lines to make the online compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

Unsupervised learning

- **Clustering methods – Hierarchical clustering**
- Hierarchical clustering is one of the popular and easy-to-understand clustering techniques. This clustering technique is divided into two types: **Agglomerative**, **Divisive**

- **Agglomerative Hierarchical clustering Technique**: In this technique, initially each data point is considered as an individual cluster. At each iteration, the similar clusters merge with other clusters until one cluster or K clusters are formed.

- **Divisive Hierarchical clustering Technique**: In simple words, we can say that Divisive Hierarchical clustering is exactly the opposite of Agglomerative Hierarchical clustering. In Divisive Hierarchical clustering, we consider all the data points as a single cluster and in each iteration, we separate the data points from the cluster which are not similar. Each data point that is separated is considered an individual cluster. In the end, we'll be left with n clusters.

Unsupervised learning

- **Clustering methods – Hierarchical clustering - Agglomerative**

The basic algorithm of Agglomerative is straightforward.

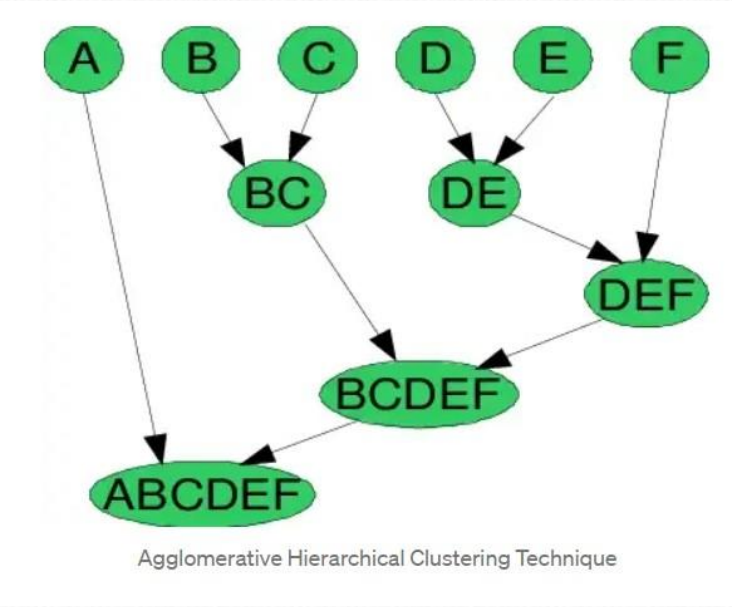
1- Let each data point be a cluster

2 - Compute the similarity of clusters

3 - Merge the two closest clusters and update the similarity for new clusters

4- Repeat this process until k clusters remain.

- To understand better let's see a pictorial representation of the Agglomerative Hierarchical clustering Technique. Lets say we have six data points {A,B,C,D,E,F}.



Agglomerative Hierarchical Clustering Technique

Unsupervised learning

- **Clustering methods – Hierarchical clustering - Agglomerative**

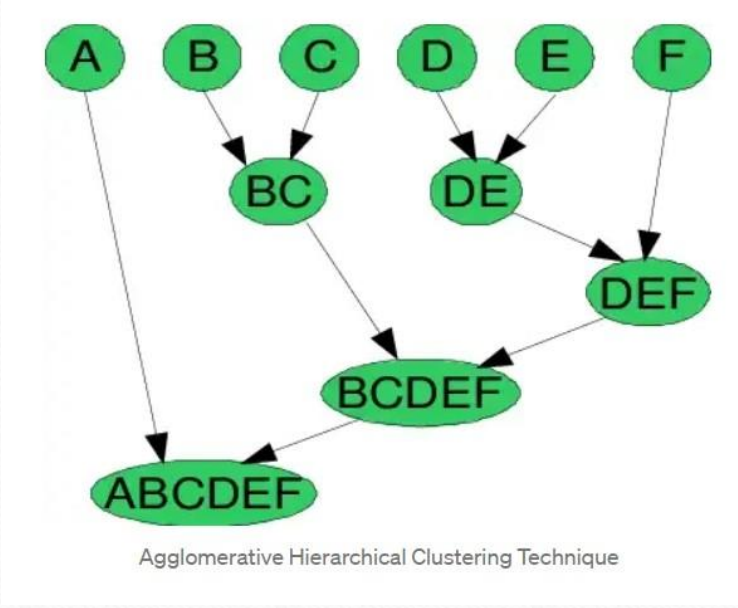
Step- 1: In the initial step, we calculate the similarity of individual points and consider all the six data points as individual clusters as shown in the image.

Step- 2: In step two, similar clusters are merged and formed as a single cluster. Let's consider B,C, and D,E are similar clusters that are merged in step two. Now, we're left with four clusters which are A, BC, DE, F.

Step- 3: We again calculate the proximity of new clusters and merge the similar clusters to form new clusters A, BC, DEF.

Step- 4: Calculate the proximity of the new clusters. The clusters DEF and BC are similar and merged to form a new cluster. We're now left with two clusters A, BCDEF.

Step- 5: Finally, all the clusters are merged and form a single cluster.



Agglomerative Hierarchical Clustering Technique

Unsupervised learning

- **Clustering methods – Hierarchical clustering - Agglomerative**
- Calculating the similarity between two clusters is important to merge or divide the clusters. Different techniques are used to calculate the similarity between two clusters:

1- **ward**: minimizes the variance of the clusters being merged.
2- **average**: uses the average of the distances of each observation of the two sets.
3- **complete or maximum linkage**: uses the maximum distances between all observations of the two sets.
4- **single linkage**: uses the minimum of the distances between all observations of the two sets

- All the approaches to calculating the similarity between clusters have their advantages and disadvantages.

- Limitations of Hierarchical clustering Technique:
- There is no mathematical objective for Hierarchical clustering.
- High space and time complexity for Hierarchical clustering. Hence this clustering algorithm cannot be used when we have huge data.

Unsupervised learning

- **Clustering methods – Hierarchical clustering - Agglomerative**

Example

```
#Three lines to make the online compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')
##########
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import AgglomerativeClustering
x=[15,15,16,19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65]
data=np.array([15,15,16,19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65]).reshape(-1, 1)
hierarchical_cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
labels = hierarchical_cluster.fit_predict(data)
plt.scatter(np.linspace(1,19,19),x, c=labels)
plt.show()
print(x)
print(hierarchical_cluster.labels_)
##########Two lines to make the online compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

Unsupervised learning

- **Clustering methods – Hierarchical clustering - Agglomerative**

NOTE: As there is no objective function in the hierarchical clustering, then we can't assign a score to the clustering. To use the elbow method we need to use other clustering scores such as the Silhouette Coefficient.

Example

```
#Three lines to make the online compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')
##########
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
x=np.array([15,15,16,19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65]).reshape(-1, 1)
y=np.linspace(1,19,19).reshape(-1, 1)
Z= np.concatenate((x,y))
inertias = []
k=[]
for i in range(2,7):
    k.append(i)
    ac = AgglomerativeClustering(n_clusters = i)
    labels = ac.fit_predict(Z)
    a=silhouette_score(Z, labels)
    inertias.append(a)
plt.bar(k, inertias)
plt.xlabel('Number of clusters', fontsize = 20)
plt.ylabel('S(i)', fontsize = 20)
plt.show()
#########Two lines to make the online compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

Unsupervised learning

- **Dimensionality reduction**

- If your data is represented using rows and columns, such as in a spreadsheet, then the input variables are the columns that are fed as input to a model to predict the target variable. Input variables are also called features.

- We can consider the columns of data representing dimensions on an n-dimensional feature space and the rows of data as points in that space. This is a useful geometric interpretation of a dataset.

- The performance of machine learning algorithms can degrade with too many input variables.

- This can dramatically impact the performance of machine learning algorithms fit on data with many input features, generally referred to as the "curse of dimensionality."

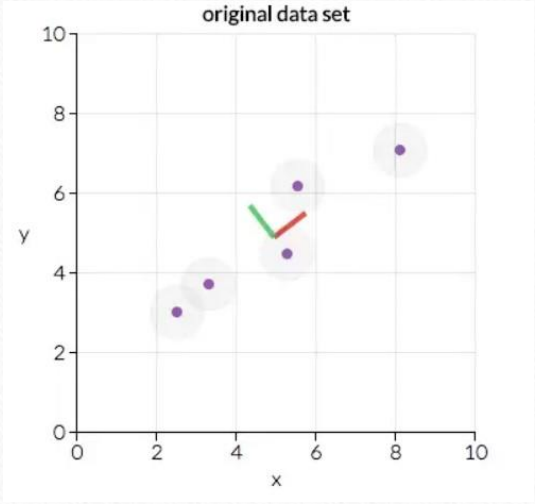
- Therefore, it is often desirable to reduce the number of input features.

- This reduces the number of dimensions of the feature space, hence the name "dimensionality reduction."

Unsupervised learning

- **Dimensionality reduction – Principle component analysis (PCA)**

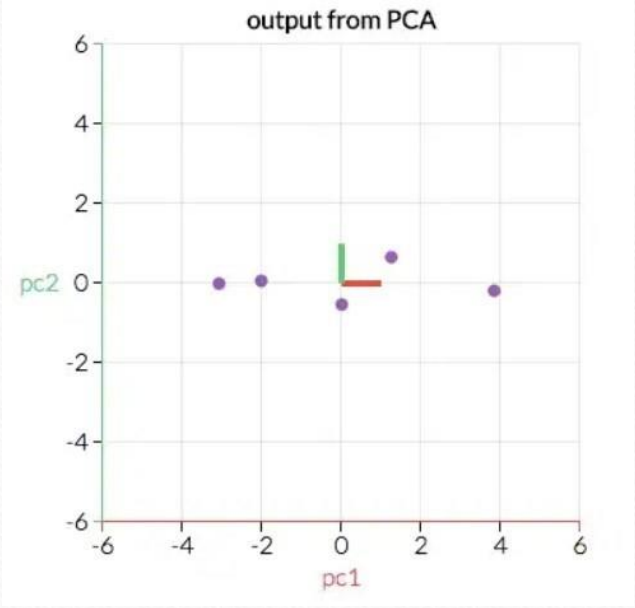
- PCA is one of the most favorite methods for reducing the number of input variables (features)
- To better understand PCA, let's consider the dataset as a matrix.
- We could break this matrix down into two separate components: direction and magnitude. We can then understand the "directions" of our data and its "magnitude" (or how "important" each direction is). For example, the screenshot below, from a dataset, displays the two main directions in the data: the "red direction" and the "green direction." In this case, the "red direction" is the more important one (why?)



Unsupervised learning

- **Dimensionality reduction – Principle component analysis (PCA)**

- We will transform our original data to align with these important directions (which are combinations of our original variables). The screenshot below is the same data, but transformed so that the x- and y-axes are now the "red direction" and "green direction."



Unsupervised learning

- **Dimensionality reduction – Principle component analysis (PCA)**
- While the visual example here is two-dimensional (and thus we have two "directions"), think about a case where our data has more dimensions. By identifying which "directions" are most "important," we can compress or project our data into a smaller space by dropping the "directions" that are the "least important."

NOTE: Always start PCS by Standardizing or normalizing the data since PCA's output is influenced based on the scale of the features of the data.

Unsupervised learning

- **Dimensionality reduction – Principle component analysis (PCA)**

Example

Performing PCA on a breast cancer dataset

```
from sklearn.datasets import load_breast_cancer
import numpy as np
import pandas as pd
pd.set_option('display.max_rows', 20)
pd.set_option('display.max_columns', 32)
breast = load_breast_cancer()
breast_data = breast.data
breast_labels = breast.target
labels = np.reshape(breast_labels,(569,1))
final_breast_data = np.concatenate([breast_data,labels],axis=1)
features = breast.feature_names
features_labels = np.append(features,'label')
breast_dataset = pd.DataFrame(final_breast_data,columns=features_labels)
print(breast_dataset.head())

# Test1
breast_dataset['label'].replace(0, 'Benign',inplace=True)
breast_dataset['label'].replace(1, 'Malignant',inplace=True)
print(breast_dataset.head())
```

Unsupervised learning

- **Dimensionality reduction – Principle component analysis (PCA)**

Example

Performing PCA on a breast cancer dataset

```
# preprocess data
from sklearn.preprocessing import StandardScaler
x = breast_dataset.loc[:, features].values
x = StandardScaler().fit_transform(x) # normalizing the features
print(np.mean(x),np.std(x))
normalised_breast = pd.DataFrame(x,columns=features)
print(normalised_breast.head())
#PCA
from sklearn.decomposition import PCA
pca_breast = PCA(n_components=2)
principalComponents_breast = pca_breast.fit_transform(x)
principal_breast_Df = pd.DataFrame(data = principalComponents_breast
        , columns = ['principal component 1', 'principal component 2'])
print(principal_breast_Df.tail())
```

Unsupervised learning

- **Dimensionality reduction – Principle component analysis (PCA)**

Example

Performing PCA on a breast cancer dataset

```
#Three lines to make the online compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')
##########
import matplotlib.pyplot as plt
plt.figure()
plt.figure(figsize=(10,10))
plt.xticks(fontsize=12)
plt.yticks(fontsize=14)
plt.xlabel('Principal Component - 1',fontsize=20)
plt.ylabel('Principal Component - 2',fontsize=20)
plt.title("Principal Component Analysis of Breast Cancer Dataset",fontsize=20)
targets = ['Benign', 'Malignant']
colors = ['r', 'y']
for target, color in zip(targets,colors):
    indicesToKeep = breast_dataset['label'] == target
    plt.scatter(principal_breast_Df.loc[indicesToKeep, 'principal component 1']
               , principal_breast_Df.loc[indicesToKeep, 'principal component 2'], c = color, s = 50)

plt.legend(targets,prop={'size': 15})
#########Two lines to make the online compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```