Week 9: Unsupervised learning: dimensionality reduction and clustering

Introduction

This week you will learn about the curse of dimensionality and how to overcome this problem. The term curse of dimensionality refers to the issues arising when dealing with high dimensional datasets. You will learn techniques to reduce the dimensions of datasets. You will learn about unsupervised learning that will help you to categorise data by identifying the common features. You will then practise unsupervised learning methods and dimensionality reduction techniques using Python or R and machine learning libraries.

Recommended reading

Before completing this week's lessons and activities, you should read the following two chapters from Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Ed.). O'Reilly Media, Inc.

- Chapter 8: Dimensionality Reduction (all sections)
- Chapter 9: Unsupervised Learning Techniques (up to the section "Other Clustering Algorithms")

All readings are available from the course Leganto reading list.

In the following slides, we summarise some of the important points discussed in the above chapters. We will then focus on Ed exercises based on the content discussed in this week.

Linear Algebra and Stats Recap

Source of Slides: "A Tutorial on Data Reduction":
http://www.sci.utah.edu/~shireen/pdfs/tutorials/Elhabian_PCA09.pdf

We see implementation in Python with an example below.

Run PYTHON

```python
import numpy as np

a=np.array([[1,2],[3,4]])
b=np.array([[11,12],[13,14]])
print(np.dot(a,b), ' dot product')
#array([[37, 40],
 #       [85, 92]])

print(np.inner(a,b), ' inner product')
#array([[35, 41],
 #       [81, 95]])
```

If we consider Matrix a

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

and Matrix b

$$a = \begin{bmatrix} 11 & 12 \\ 13 & 14 \end{bmatrix}$$

The dot product would be

$$\begin{bmatrix} 1*11+2*13 & 1*12+2*14 \\ 3*11+4*13 & 3*12+4*14 \end{bmatrix}$$

$$= \begin{bmatrix} 37 & 40 \\ 85 & 92 \end{bmatrix}$$

and inner product would be

$$\begin{bmatrix} 1*11+2*12 & 1*13+2*14 \\ 3*11+4*12 & 3*13+4*14 \end{bmatrix}$$

$$= \begin{bmatrix} 35 & 41 \\ 81 & 95 \end{bmatrix}$$

i Code and example source: https://www.bogotobogo.com/python/python_numpy_matrix_tutorial.php

Determinant of Matrix

Run PYTHON

```python
# importing Numpy package
#Source https://www.geeksforgeeks.org/how-to-calculate-the-determinant-o
import numpy as np

# creating a 2X2 Numpy matrix
n_array = np.array([[50, 29], [30, 44]])

# Displaying the Matrix
print("Numpy Matrix is:")
print(n_array)

# calculating the determinant of matrix
det = np.linalg.det(n_array)

```

Code Source. https://www.geeksforgeeks.org/how-to-calculate-the-determinant-of-a-matrix-using-numpy/

$|A| = ad - bc$

$|A| = a(ei - fh) - b(di - fg) + c(dh - eg)$

Source: https://www.mathsisfun.com/algebra/matrix-determinant.html

Covariance Matrix

Source of Slides: "A Tutorial on Data Reduction":
http://www.sci.utah.edu/~shireen/pdfs/tutorials/Elhabian_PCA09.pdf

Below is an example of a Covariance matrix from Iris dataset.

Source: https://www.kaggle.com/dronio/iris-plots-correlation-matrix

```python
from sklearn.datasets import load_iris

iris = load_iris()
iris_X = iris.data


# Source: https://www.geeksforgeeks.org/python-numpy-cov-function/
import numpy as np

x = [1.23, 2.12, 3.34, 4.5]
y = [2.56, 2.89, 3.76, 3.95]

# find out covariance with respect  columns
cov_mat = np.stack((x, y), axis = 0)
```

Code Source: https://www.geeksforgeeks.org/python-numpy-cov-function/

Run PYTHON

```python

from numpy import array
from numpy import cov
from numpy import var

M = array([[1,2,3,4,5,6],[1,2,3,4,5,6]])
print(M)
col_mean = var(M, ddof=1, axis=0)
print(col_mean)
row_mean = var(M, ddof=1, axis=1)
print(row_mean)

print(' now we see covariance example')
x = array([1,2,3,4,5,6,7,8,9])
```

Code Source. https://machinelearningmastery.com/introduction-to-expected-value-variance-and-covariance/

Orthogonal and Orthonormal Vectors

Orthogonal: If two vectors are perpendicular to each other, i.e if their dot product is 0.

Orthonormal: A set of vectors if orthonormal if every pair is orthogonal and given that they are normalised, i.e add to 1.

Source and more information: https://www.ucl.ac.uk/~ucahmdl/LessonPlans/Lesson10.pdf
https://www.geeksforgeeks.org/orthogonal-and-orthonormal-vectors-in-linear-algebra/

Eigenvector and Eigenvalues

Implementation in Python: https://numpy.org/doc/stable/reference/generated/numpy.linalg.eigh.html

Figure Source. Lui Serrano, Principal Component Analysis (PCA): https://www.youtube.com/watch?v=g-Hb26agBFg

Below an example of eigen decomposition for 3x3 Matrix that gives eigen values and eigen vectors.

Run PYTHON

```python
# eigendecomposition
#source: https://machinelearningmastery.com/introduction-to-eigendecompo
from numpy import array
from numpy.linalg import eig
# define matrix
A = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(A)
# calculate eigendecomposition
values, vectors = eig(A)
print(values)
print(vectors)
```

Code Source: https://machinelearningmastery.com/introduction-to-eigendecomposition-eigenvalues-and-eigenvectors/

To confirm that a vector is indeed an eigenvector of a matrix, we multiply the candidate eigenvector by the eigenvector and compare the result with the eigenvalue as shown below.

▸ Run PYTHON

```python
# confirm eigenvector
# Source: https://machinelearningmastery.com/introduction-to-eigendecomp
from numpy import array
from numpy.linalg import eig
# define matrix
A = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# calculate eigendecomposition
values, vectors = eig(A)
# confirm first eigenvector
B = A.dot(vectors[:, 0])
print(B)
C = vectors[:, 0] * values[0]
print(C)
```

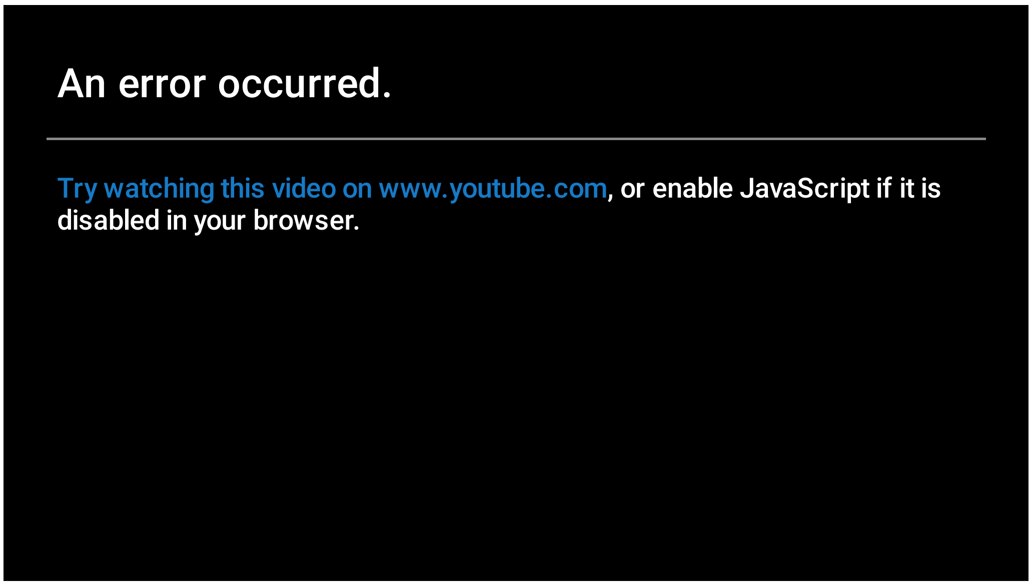
Code Source: https://machinelearningmastery.com/introduction-to-eigendecomposition-eigenvalues-and-eigenvectors/

Next, we reconstruct the original matrix.

▸ Run PYTHON

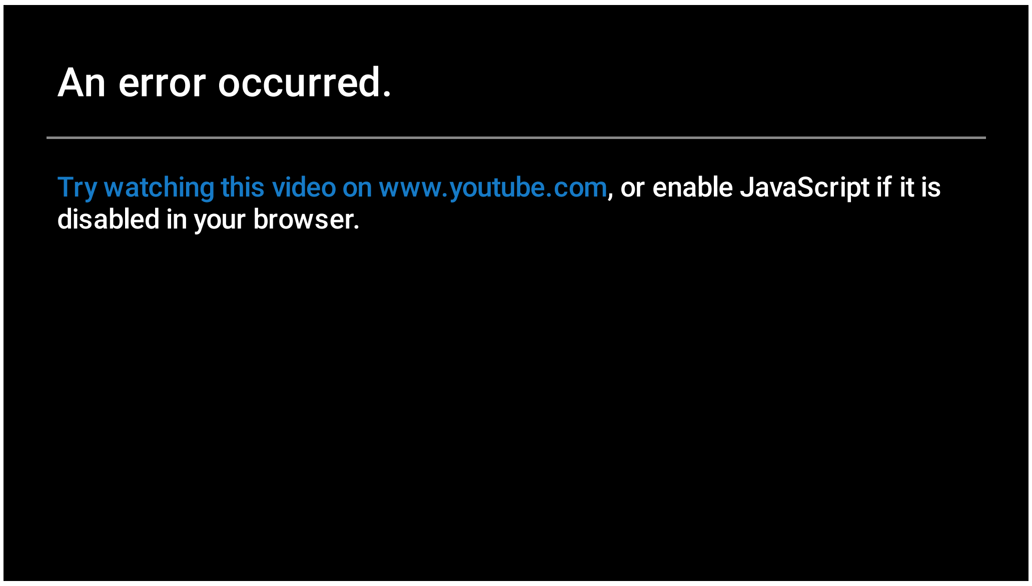
```python


# reconstruct matrix
from numpy import diag
from numpy import dot
from numpy.linalg import inv
from numpy import array
from numpy.linalg import eig
# define matrix
A = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(A)
# calculate eigenvectors and eigenvalues
values, vectors = eig(A)
# create matrix from eigenvectors
```

A video that shows good visualisation for Eigenvectors and Eigenvalues.

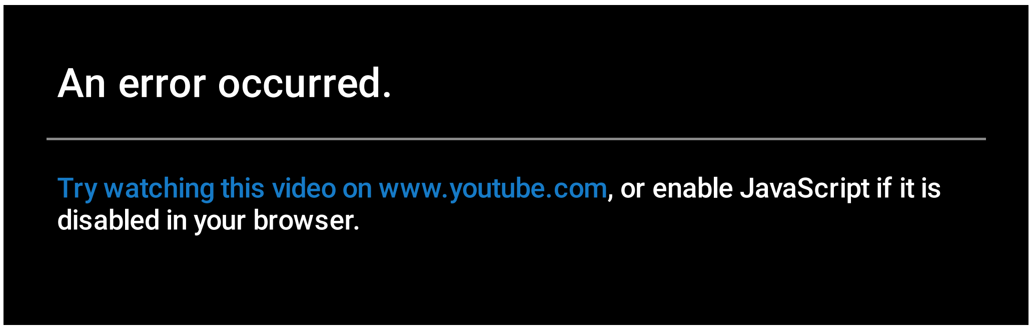

An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

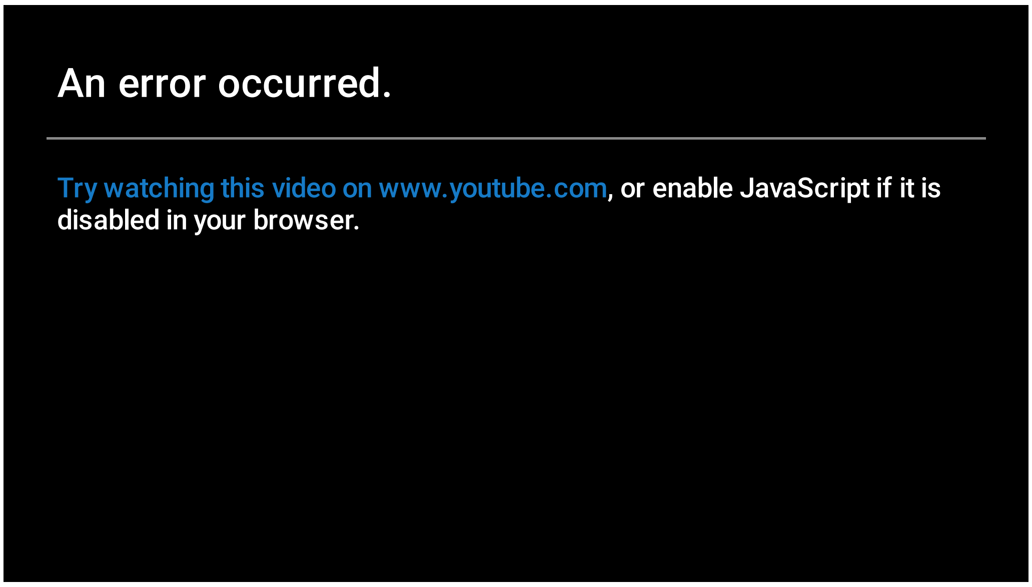
Further revision for linear algebra


An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

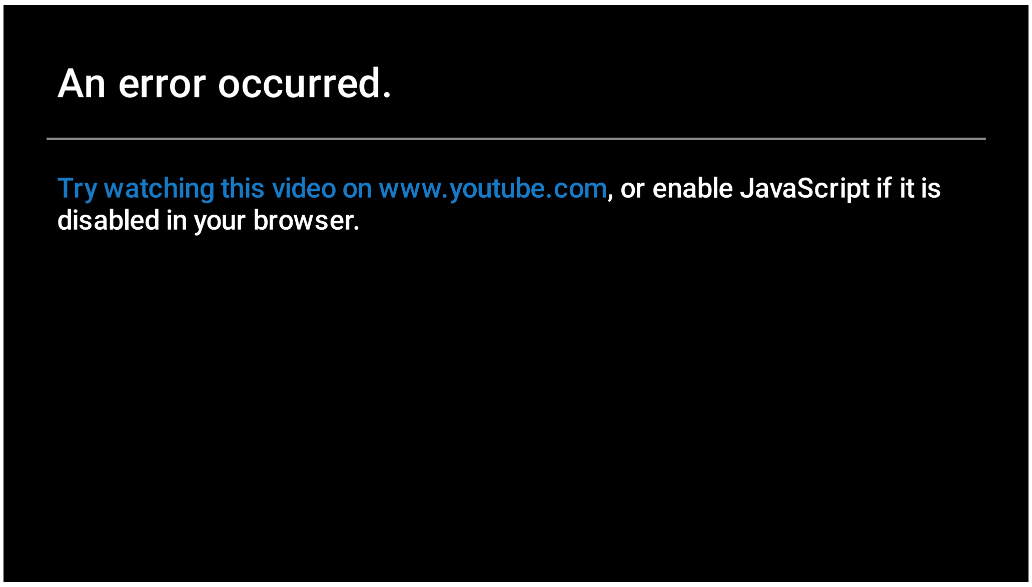

An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.



An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.



An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Unsupervised learning

In order to understand unsupervised learning, it is important to understand the differences between supervised and unsupervised learning.

In supervised learning, classes are known and need a “definition” in terms of the data.

Methods are known as: classification, discriminant analysis, class prediction, supervised pattern recognition.

In unsupervised learning, classes are initially unknown and need to be “discovered” with their definitions from the data.

Prominent methods are known as cluster analysis, class discovery, unsupervised pattern recognition.

Therefore, unsupervised learning methods, such as clustering, address the problem of assigning instances to classes given only observations about the instances, i.e., without being given class “labels” for instances by a “teacher”.

Why do we need unsupervised learning?

Most of the world’s data is unlabelled and manually labelling data is often very time consuming, expensive and sometimes not even possible! We also need to predetermine possible labels (classes) and the whole process is error-prone.

Examples

1. Dimensionality reduction with PCA
2. Clustering with K-Means Clustering

Dimensionality reduction

Often we need to deal with thousands and even millions of features in a data set. Each feature in a data set refers to one dimension. Recent technological advances allow us to capture many features at a very high frequency (number of instances). However, increasing the number of features (dimensions) in a data set is not always beneficial. It can significantly increase the amount of time required for training and importantly may result in overfitting. Géron, 2019 describes the phenomena as such:

✓ "High-dimensional datasets are at risk of being very sparse: most training instances are likely to be far away from each other. This also means that a new instance will likely be far away from any training instance, making predictions much less reliable than in lower dimensions, since they will be based on much larger extrapolations. In short, the more dimensions the training set has, the greater the risk of overfitting it.

✓ In theory, one solution to the curse of dimensionality could be to increase the size of the training set to reach a sufficient density of training instances. Unfortunately, in practice, the number of training instances required to reach a given density grows exponentially with the number of dimensions. With just 100 features (significantly fewer than in the MNIST problem), you would need more training instances than atoms in the observable universe in order for training instances to be within 0.1 of each other on average, assuming they were spread out uniformly across all dimensions." (Géron, 2019, p. 215)

Dimensionality reduction

One popular approach to address the curse of dimensionality is to select a representative subset of the available features to build a model. This may result in some information loss; however, it often avoids overfitting and significantly decreases the amount of time required for training.

Reduction in dimensionality allows us to easily visualise and explore datasets, which is particularly useful to explain or justify your conclusions to decision makers who are often not data scientists. Reduction in dimensionality also often results in simpler models.

Two popular approaches to address the curse of dimensionality by reducing dimensionality are:

- Projection
- Manifold Learning

Projection

Often training instances are not distributed equally across all dimensions, and it's possible to transform them from a high-dimensional space to a low-dimensional space such that the low-dimensional space representation keeps most of the properties available in the original dimensions.

For example, converting the following 3D data set to a new 2D subspace after projection retains most of the properties in the data set.

Figure: A 3D data set lying close to a 2D subspace. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

Figure: The new 2D data set after projection. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

However, the above approach may not prove useful for some data sets. For example, the following figure shows Swiss roll toy data set and its projection onto a plane, which doesn't retain properties from the original data set properties.

Figure: Swiss roll data set. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

Figure: Squashing by projecting onto a plane (left) versus unrolling the Swiss roll (right). Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

Below is another example.

Figure Source. Principal Component Analysis: Your Tutorial and Code: https://towardsdatascience.com/principal-component-analysis-your-tutorial-and-code-9719d3d3f376

Manifold learning

The following is an important quote from the section:

"Many dimensionality reduction algorithms work by modelling the manifold on which the training instances lie; this is called Manifold Learning. It relies on the manifold assumption, also called the manifold hypothesis, which holds that most real-world high-dimensional datasets lie close to a much lower-dimensional manifold. This assumption is very often empirically observed.

The manifold assumption is often accompanied by another implicit assumption: that the task at hand (e.g., classification or regression) will be simpler if expressed in the lower-dimensional space of the manifold. For example, in the top row of Figure 8-6 the Swiss roll is split into two classes: in the 3D space (on the left), the decision boundary would be fairly complex, but in the 2D unrolled manifold space (on the right), the decision boundary is a straight line.

However, this implicit assumption does not always hold. For example, in the bottom row of the following figure, the decision boundary is located at x1 = 5. This decision boundary looks very simple in the original 3D space (a vertical plane), but it looks more complex in the unrolled manifold (a collection of four independent line segments)." (Géron, 2019, pg. 218)

In short, reducing the dimensionality of your training set before training a model will usually speed up training, but it may not always lead to a better or simpler solution; it all depends on the data set.

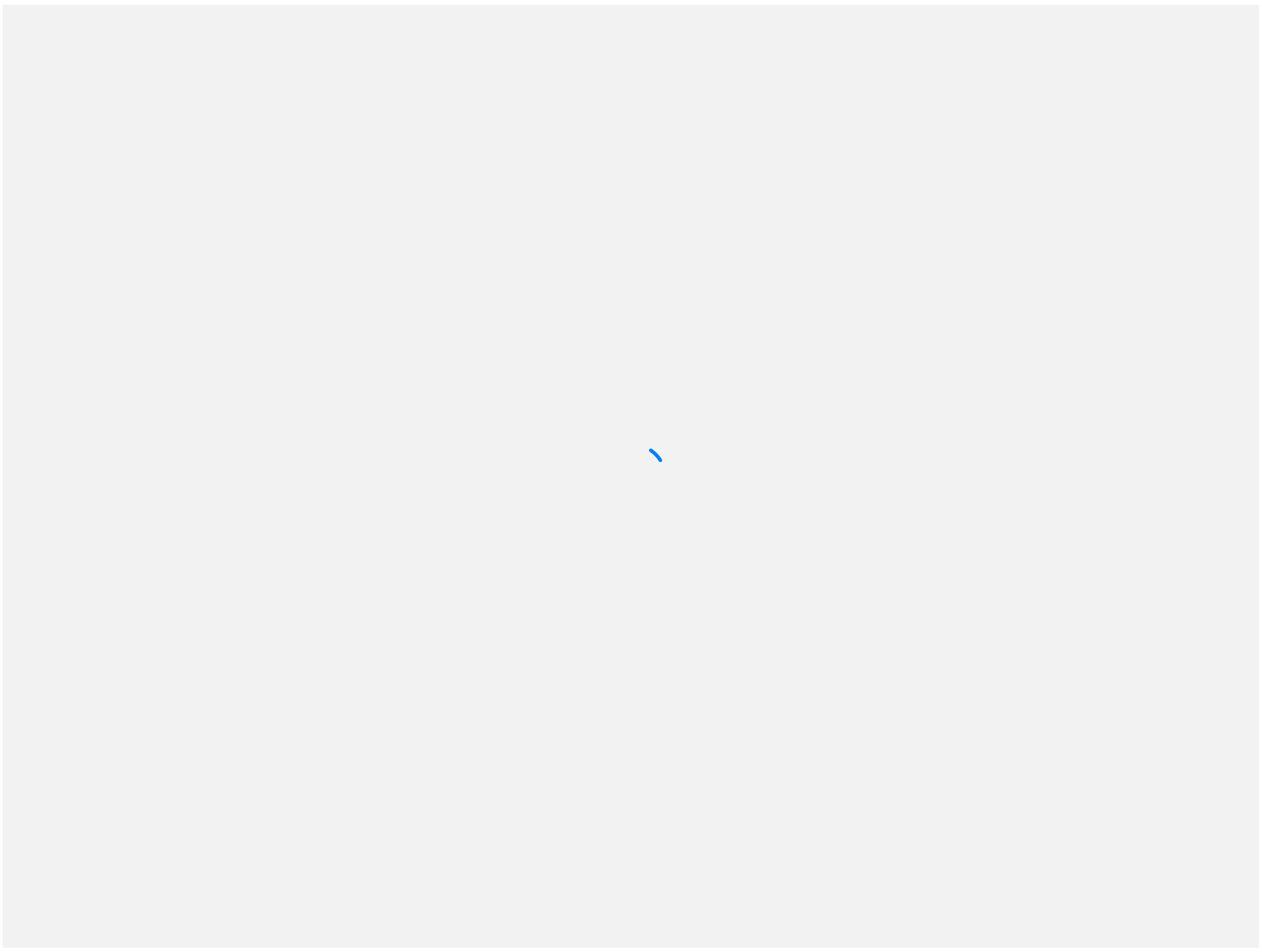


Figure: The decision boundary may not always be simpler with lower dimensions. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

Principal Component Analysis

Now that you know about the curse of dimensionality and the related issues, learn how to mitigate these issues using one of the common approaches known as Principal Component Analysis (PCA).

PCA is one of the most popular approaches used for reducing dimensions. The following are some of the important points describing PCA and its uses from Chapter 8 of Géron, 2019:

PCA identifies the hyperplane that lies closest to the data, and then it projects the data onto it.

- It selects a hyperplane that preserves the maximum amount of variance (the axis with the solid line), as it will most likely lose less information than the other projections. Another way to justify this choice is that it is the axis that minimises the mean squared distance between the original data set and its projection onto that axis. This is the rather simple idea behind PCA.
- PCA initially identifies the axis that accounts for the largest amount of variance in the training set. Later, PCA finds second axis, orthogonal to the first axis that accounts for the largest amount of remaining variance, similarly finds third axis orthogonal to both the previous axes, and so on. The *i*th axis is called the *i*th *principal component* (PC) of the data. In the figure below, the first axis is c1 and the second axis is c2.

Figure: Selecting the subspace to project on. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

Example

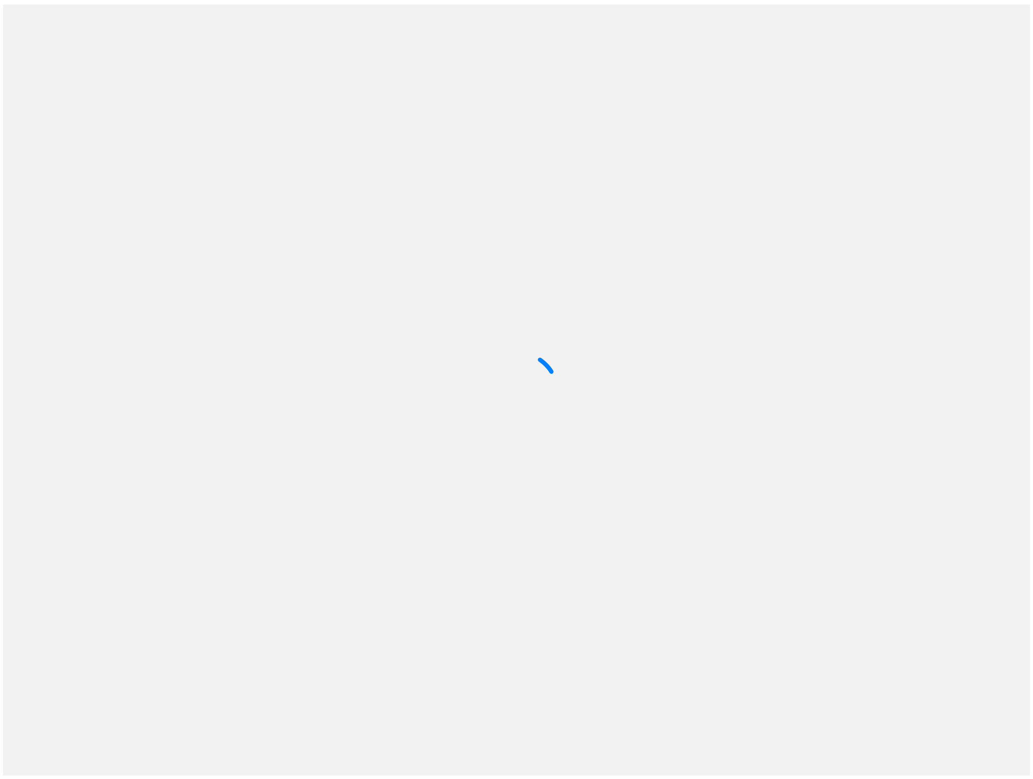


Figure Source.

Principle Component Analysis (PCA)

Below we see a plane that fits through the data rotated in 3D space.

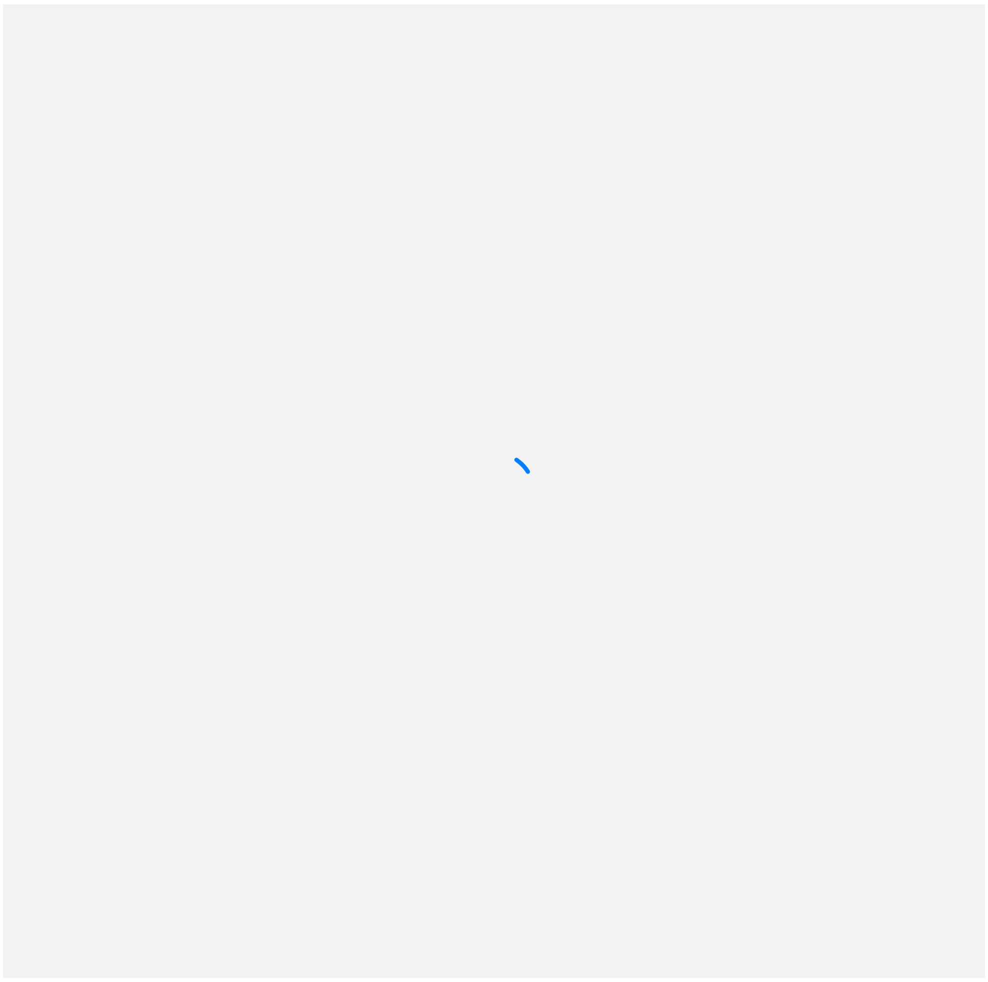


Figure Source. https://medium.com/@ashwin8april/dimensionality-reduction-and-visualization-using-pca-principal-component-analysis-8489b46c2ae0

PCA is implemented by rotation of the axis such that each chosen such that each successful axis captures or preserves as much variance as possible which can be visualised as follows.

Figure. We rotate our axes to find the best principal component(z_1 and z_2) such that the variance of xi_ projected onto z 1 is maximal. We drop z_2 later. So z_2 becomes our only needed direction. Source: https://medium.com/@ashwin8april/dimensionality-reduction-and-visualization-using-pca-principal-component-analysis-8489b46c2ae0

Figure Source. https://medium.com/@ashwin8april/dimensionality-reduction-and-visualization-using-pca-principal-component-analysis-8489b46c2ae0

We now see the basic PCA algorithm with key steps

Figure Source. PCA Lecture @CMU: https://www.cs.cmu.edu/~mgormley/courses/10701-f16/slides/lecture14-pca.pdf

Advantages of PCA

- PCA removes correlated features and hence after PCA is executed, the principal components are independent of one another and there is no correlation among them.
- In the case where there are too many features (variables or dimensions) in the dataset, overfitting mainly occurs by conventional learning algorithms (NNs). PCA helps in avoiding the overfitting issue by reducing the number of features.
- In some applications, PCA can be used for better visualisation of the data since it transforms a high dimensional data to low dimensional data (2 or 3 dimensions).

Limitations of PCA

- PCA cannot be used for sparse data, this is where SVD is more appropriate. We will cover SVD

next.
- PCA would not be appropriate when the features are completely uncorrelated. PCA would not preserve much of the variance in the original data and hence would be useless, and PCA works fine if a subset of features have correlation.
- PCA assumes that the principal components are orthogonal.
- New features obtained by PCA may not make any sense i.e. are not interpretable.
- Below see examples where PCA will struggle.

Figure Source: https://medium.com/@ashwin8april/dimensionality-reduction-and-visualization-using-pca-principal-component-analysis-8489b46c2ae0

We next see implementation in Python about PCA from scratch that shows details at each step.

Run PYTHON

```python
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target

print(X)


import numpy as np

# Compute the mean of the data
mean_vec = np.mean(X, axis=0)
```

Code Source. https://towardsdatascience.com/principal-component-analysis-your-tutorial-and-code-9719d3d3f376

Next we show the PCA being used with Iris dataset

Run PYTHON

```python
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target

print(X)


import numpy as np

# Compute the mean of the data
mean_vec = np.mean(X, axis=0)
```

Code - Figure Source: https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_iris.html#sphx-glr-auto-examples-decomposition-plot-pca-iris-py

Now we see how simple it is to use PCA with sklearn. The explained variance ratio gives a percentage of variance explained by each of the selected components. The singular values corresponding to each of the selected components.

Run PYTHON

```python
import numpy as np
from sklearn.decomposition import PCA
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
#print(X, 'data X')
pca = PCA(n_components=2) # If n_components is not set then all componen
pca.fit(X)
print(pca.explained_variance_ratio_, ' explained_variance_ratio_')
#Principal axes in feature space, representing the directions of maximum
#The components are sorted by explained_variance_.
print(pca.singular_values_, ' singular_values_')
#[6.30061... 0.54980...]
print(' now with SVD solver')
pca = PCA(n_components=2, svd_solver='full')
pca.fit(X)
```

Code Source: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Figure Source. PCA Lecture @CMU: https://www.cs.cmu.edu/~mgormley/courses/10701-f16/slides/lecture14-pca.pdf

Lets look at some code.

```python
from numpy import array
from numpy import diag
from numpy import zeros
from scipy.linalg import svd
# define a matrix
A = array([
    [1,2,3,4,5,6,7,8,9,10],
    [11,12,13,14,15,16,17,18,19,20],
    [21,22,23,24,25,26,27,28,29,30]])
print(A)
# Singular-value decomposition
U, s, VT = svd(A)
# create m x n Sigma matrix
Sigma = zeros((A.shape[0], A.shape[1]))
```

Code Source: https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning/

Next we see implementation in R

```r
library(MASS)

a <- matrix(c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1), 9, 4)

a.svd <- svd(a)
a.svd$d

ds <- diag(1/a.svd$d[1:3])
u <- a.svd$u
v <- a.svd$v
us <- as.matrix(u[, 1:3])
vs <- as.matrix(v[, 1:3])

```

Code Source: https://stats.idre.ucla.edu/r/codefragments/svd_demos/

R code for PCA

```r
library(ggfortify)
df <- iris[1:4]
pca_res <- prcomp(df, scale. = TRUE)

autoplot(pca_res)

autoplot(pca_res, data = iris, colour = 'Species')

autoplot(pca_res, data = iris, colour = 'Species', label = TRUE, label.s

autoplot(pca_res, data = iris, colour = 'Species', shape = FALSE, label.
```

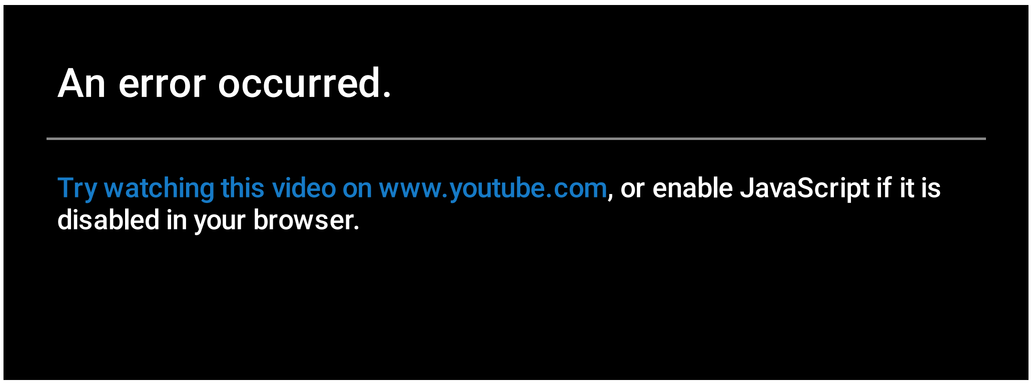
Code Source: https://cran.r-project.org/web/packages/ggfortify/vignettes/plot_pca.html

PCA vs SVD

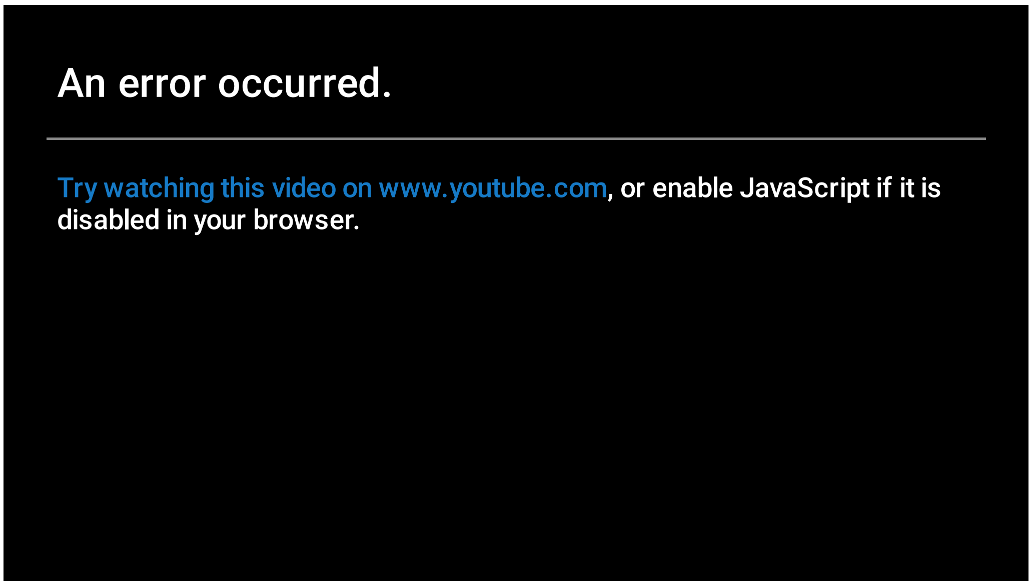
We compare the eigendecomposition $A = PDP^{-}1$ and SVD with A=UΣV∗ and note some key differences as follows.

- In PCA, the eigendecomposition matrix P vectors are not necessarily orthogonal (i.e. perpendicular), hence the change of basis isn't a simple rotation. Whereas the vectors in the matrices U and V in SVD are orthonormal, hence they do represent rotations.
- In the SVD, the nondiagonal matrices U and V are not necessarily the inverse of one another and usually not related. Whereas in the eigendecomposition the nondiagonal matrices P and P−1 are inverses of each other.
- The entries in the diagonal matrix Σ of SVD are all real and nonnegative. In the eigendecomposition, the entries of D can be any complex number; i.e. negative, positive, and imaginary.
- The SVD can apply to the rectangular or square matrix, whereas the eigendecomposition can only apply for square matrices.

Some videos on PCA


An error occurred.

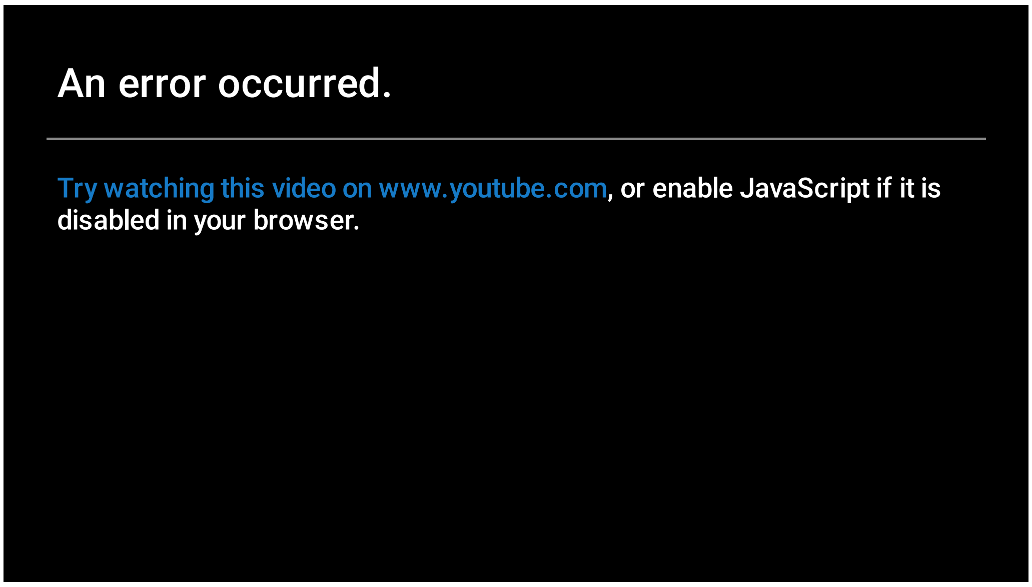
Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.



An error occurred.

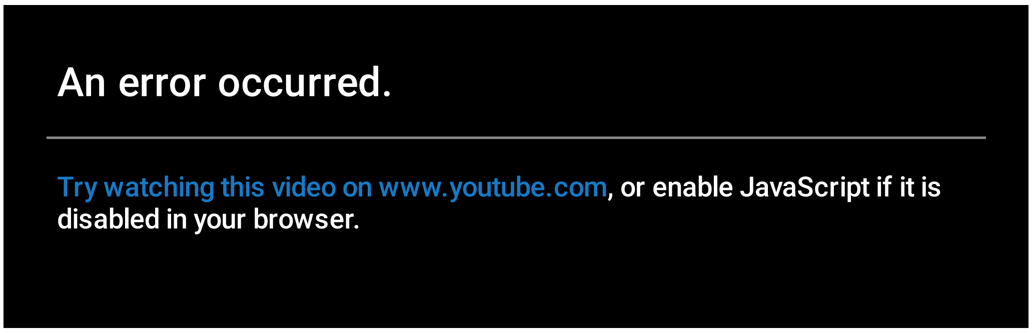
Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Some videos below on SVD



An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.



An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

[redacted]

More information

1. https://math.stackexchange.com/questions/320220/intuitively-what-is-the-difference-between-eigendecomposition-and-singular-valu
2. https://www.cc.gatech.edu/~dellaert/pubs/svd-note.pdf
3. http://hua-zhou.github.io/teaching/biostatm280-2017spring/slides/16-eigsvd/eigsvd.html
4. https://blog.umetrics.com/what-is-principal-component-analysis-pca-and-how-it-is-used

References

1. Pearson, K. (1901). "On Lines and Planes of Closest Fit to Systems of Points in Space". *Philosophical Magazine*. **2** (11): 559–572. doi:10.1080/14786440109462720.
2. Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, **24**, 417–441, and 498–520.
   Hotelling, H (1936). "Relations between two sets of variates". *Biometrika*. **28** (3/4): 321–377. doi:10.2307/2333955.
3. Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, *2*(1-3), 37-52. http://pzs.dstu.dp.ua/DataMining/pca/bibl/Principal%20components%20analysis.pdf
4. Jolliffe, I. T. (1986). Principal components in regression analysis. In *Principal component analysis* (pp. 129-155). Springer, New York, NY.
5. Ku, W., Storer, R. H., & Georgakis, C. (1995). Disturbance detection and isolation by dynamic principal component analysis. *Chemometrics and intelligent laboratory systems*, *30*(1), 179-196.
6. Wall, M. E., Rechtsteiner, A., & Rocha, L. M. (2003). Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis* (pp. 91-109). Springer, Boston, MA. https://arxiv.org/pdf/physics/0208101
7. Kim, K. I., Jung, K., & Kim, H. J. (2002). Face recognition using kernel principal component analysis. *IEEE signal processing letters*, *9*(2), 40-42. https://ieeexplore.ieee.org/document/991133
8. Abdi, H., & Williams, L. J. (2010). Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, *2*(4), 433-459.
9. Shlens, J. (2014). A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*. https://arxiv.org/abs/1404.1100

10. Lever, J., Krzywinski, M., & Altman, N. (2017). Points of significance: Principal component analysis. 641 https://www.nature.com/articles/nmeth.4346.pdf

PCA Continued

A standard matrix factorization technique is called *Singular Value Decomposition* (SVD) that can decompose the training set matrix **X** into the matrix multiplication of three matrices $\mathbf{U}\ \boldsymbol{\Sigma}\ \mathbf{V}^{\intercal}$, where **V** contains the unit vectors that define all the principal components that we are looking for, as shown below (Géron, 2019, p 215):

- The following Python code uses NumPy's `svd()` function to obtain all the principal components of the training set, then extracts the two unit vectors that define the first two PCs:

```python
import numpy as np
# Build 3D dataset:
np.random.seed(4)
m = 60
w1, w2 = 0.1, 0.3
noise = 0.1

angles = np.random.rand(m) * 3 * np.pi / 2 - 0.5
X = np.empty((m, 3))
X[:, 0] = np.cos(angles) + np.sin(angles)/2 + noise * np.random.randn(m)
X[:, 1] = np.sin(angles) * 0.7 + noise * np.random.randn(m) / 2
X[:, 2] = X[:, 0] * w1 + X[:, 1] * w2 + noise * np.random.randn(m)
# ----------------
print(X, ' X')
```

- Projecting Down to d Dimensions*:* Once you have identified all the principal components, you can reduce the dimensionality of the data set down to d dimensions by projecting it onto the hyperplane defined by the first d principal components. Selecting this hyperplane ensures that the projection will preserve as much variance as possible.
- To project the training set onto the hyperplane and obtain a reduced data set Xd-proj of dimensionality d, compute the matrix multiplication of the training set matrix X by the matrix Wd, defined as the matrix containing the first d columns of V, as shown below:

- The following Python code projects the training set onto the plane defined by the first two principal components:

PYTHON

```python
W2 = Vt.T[:, :2]
X2D = X_centered.dot(W2)
```

- Scikit-learn's PCA class uses SVD decomposition to implement PCA, as discussed above. The following code applies PCA to reduce the dimensionality of the data set down to two dimensions (note that it automatically takes care of centering the data).

Change n_components value and re-run the code.

Run PYTHON

```python
from sklearn.decomposition import PCA
import numpy as np
# Build 3D dataset:
np.random.seed(4)
m = 60
w1, w2 = 0.1, 0.3
noise = 0.1

angles = np.random.rand(m) * 3 * np.pi / 2 - 0.5
X = np.empty((m, 3))
X[:, 0] = np.cos(angles) + np.sin(angles)/2 + noise * np.random.randn(m)
X[:, 1] = np.sin(angles) * 0.7 + noise * np.random.randn(m) / 2
X[:, 2] = X[:, 0] * w1 + X[:, 1] * w2 + noise * np.random.randn(m)
# ----------------
```

- Explained Variance Ratio*:* This ratio indicates the proportion of the data set's variance that lies along each principal component. In the following example, 84.2% of the dataset's variance lies along the first PC, and 14.6% lies along the second PC. This leaves less than 1.2% for the third PC, so it is reasonable to assume that the third PC probably carries little information.

PYTHON

```python
>>> pca.explained_variance_ratio_
array([0.84248607, 0.14631839])
```

Choosing the right number of dimensions

Normally we choose the number of dimensions that could explain a large portion of the variance (say 95%).

The following code performs PCA without reducing dimensionality, then computes the minimum number of dimensions required to preserve 95% of the training set's variance.

Change 0.95 to 0.99 and re-run the code.

Run PYTHON

```python
from sklearn.decomposition import PCA
import numpy as np
# Build 3D dataset:
np.random.seed(4)
m = 60
w1, w2 = 0.1, 0.3
noise = 0.1

angles = np.random.rand(m) * 3 * np.pi / 2 - 0.5
X = np.empty((m, 3))
X[:, 0] = np.cos(angles) + np.sin(angles)/2 + noise * np.random.randn(m)
X[:, 1] = np.sin(angles) * 0.7 + noise * np.random.randn(m) / 2
X[:, 2] = X[:, 0] * w1 + X[:, 1] * w2 + noise * np.random.randn(m)
# ----------------
```

Alternatively, we can set n_components to be a float between 0.0 and 1.0, indicating the ratio of variance we wish to preserve:

Set n_components to be a float between 0.0 and 1.0 and re-run the code.

Run PYTHON

```python
from sklearn.decomposition import PCA
import numpy as np
# Build 3D dataset:
np.random.seed(4)
m = 60
w1, w2 = 0.1, 0.3
noise = 0.1

angles = np.random.rand(m) * 3 * np.pi / 2 - 0.5
X = np.empty((m, 3))
X[:, 0] = np.cos(angles) + np.sin(angles)/2 + noise * np.random.randn(m)
X[:, 1] = np.sin(angles) * 0.7 + noise * np.random.randn(m) / 2
X[:, 2] = X[:, 0] * w1 + X[:, 1] * w2 + noise * np.random.randn(m)
# ----------------
```

We can also plot the explained variance as a function of the number of dimensions, see below. Normally this results in an elbow shaped curve, indicating the explained variance stops growing fast after some value of dimensions. In the figure below, we can see that reducing the dimensionality to say 150 dimensions wouldn't lose too much explained variance.

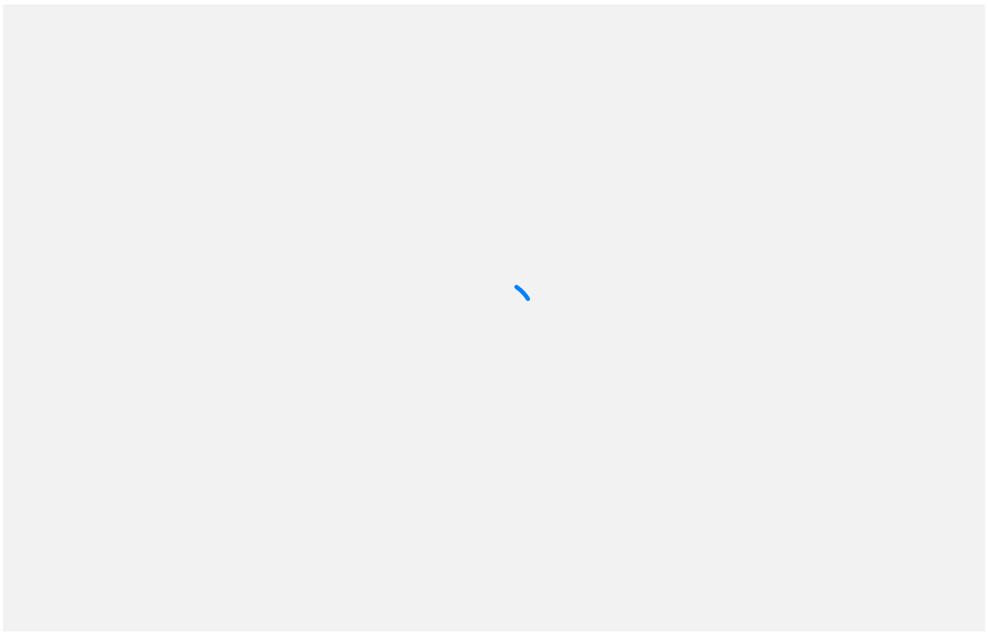


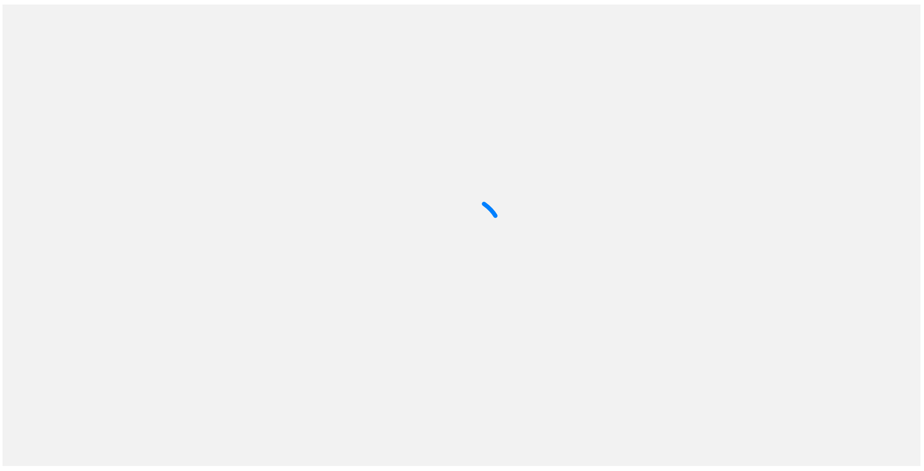
Figure: Explained variance as a function of the number of dimensions. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

The following code compresses the MNIST data set down to 154 dimensions.

PYTHON

```python
pca = PCA(n_components = 154)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)

```

The following figure shows a few digits from the original training set (on the left), and the corresponding digits after compression and decompression. We can see that the shapes of the digits are preserved, but there is a slight reduction in the image quality.



Face Recognition

Run PYTHON

```python
# Import matplotlib library
import matplotlib.pyplot as plt

# Import scikit-learn library
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import fetch_lfw_people
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.svm import SVC

import numpy as np

```

Code Source: https://www.geeksforgeeks.org/ml-face-recognition-using-pca-implementation/

Figure. Dataset with true labels. Source: https://www.geeksforgeeks.org/ml-face-recognition-using-pca-implementation/ https://scikit-learn.org/stable/auto_examples/applications/plot_face_recognition.html

Below are eigen-faces generated by PCA

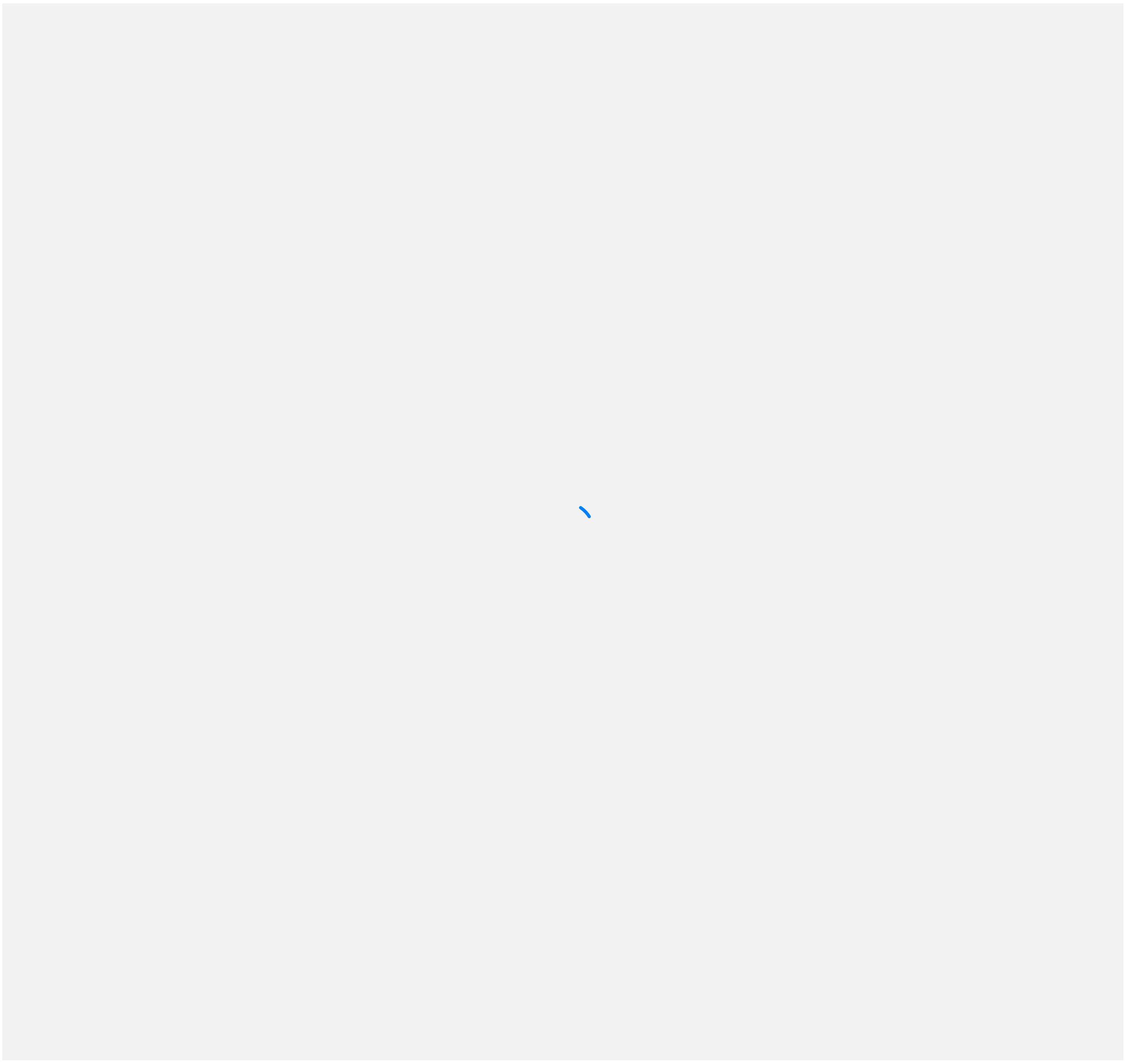


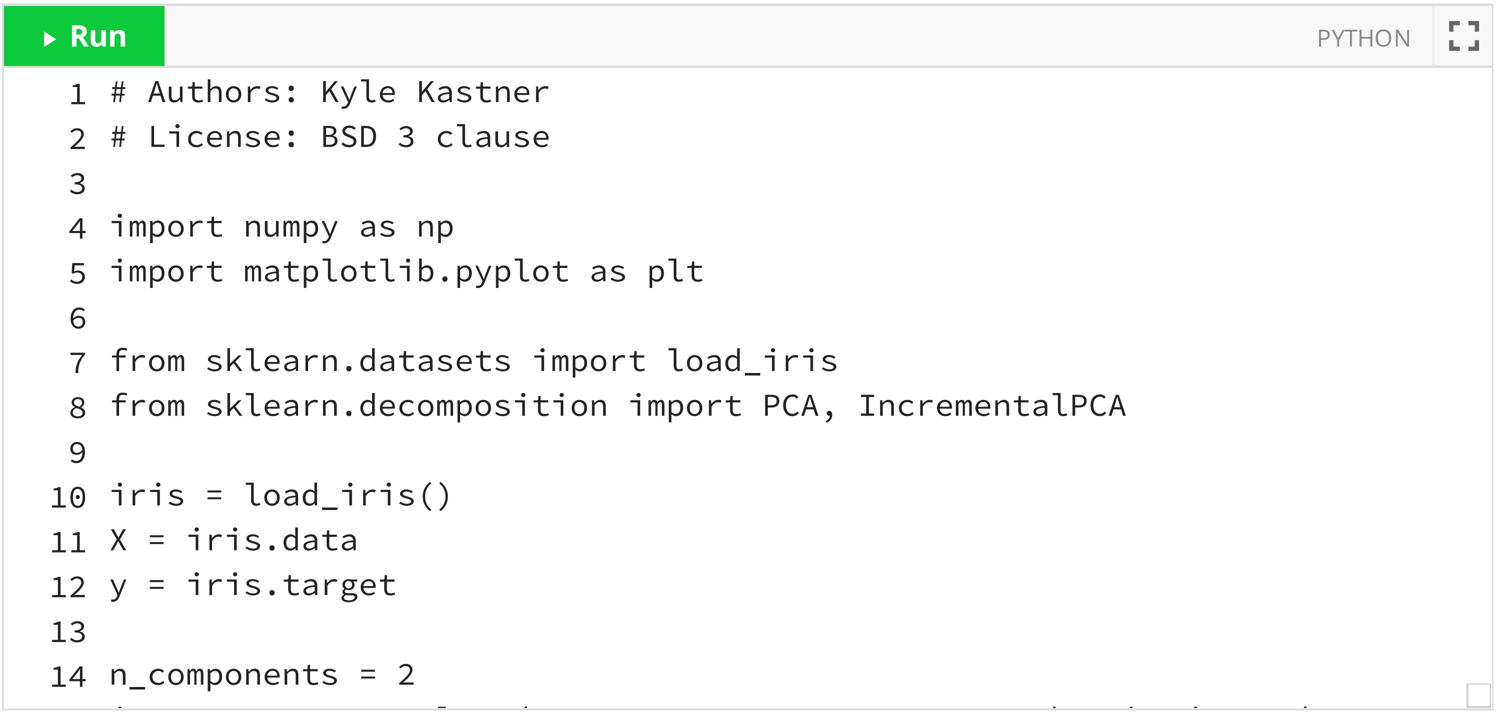
Figure. Eigenfaces generated by PCA. Source: https://www.geeksforgeeks.org/ml-face-recognition-using-pca-implementation/ https://scikit-learn.org/stable/auto_examples/applications/plot_face_recognition.html

Next use any classification algorithm (Support Vector Machine used) to train the data using the PCA coefficient generated in previous steps. In this case, the accuracy is *0.85* and predictions are shown below.

Figure. Predictions. Source: https://www.geeksforgeeks.org/ml-face-recognition-using-pca-implementation/ https://scikit-learn.org/stable/auto_examples/applications/plot_face_recognition.html

Incremental PCA

Incremental PCA (IPCA) algorithms allow us to split the training set into mini-batches and feed an IPCA algorithm one mini-batch at a time. This is useful for large training sets and for applying PCA online (i.e., on the fly as new instances arrive).



```python
# Authors: Kyle Kastner
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.decomposition import PCA, IncrementalPCA

iris = load_iris()
X = iris.data
y = iris.target

n_components = 2
```

Code Source. Scikit-learn I-PCA.

Other dimensionality reduction techniques

There are many other dimensionality reduction techniques available in scikit-learn. The textbook briefly explains some of the most popular techniques. They are not part of this course, however, you may want to read their descriptions.

- Kernel PC
- Locally Linear Embedding (LLE)
- Random Projections
- Multidimensional Scaling (MDS)
- Linear Discriminant Analysis (LDA)
- Isomap
- t-Distributed Stochastic Neighbor Embedding (t-SNE)

The following figure shows the results of three of the techniques listed above, MDS, Isomap, and t-SNE.

Figure: Reducing the Swiss roll to 2D using various techniques. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

Example in R

```r
#http://www.sthda.com/english/articles/31-principal-component-methods-in


library(factoextra)  # Ed to install
data(decathlon2) #Ed to install
decathlon2.active <- decathlon2[1:23, 1:10]
head(decathlon2.active[, 1:6])


```

Code Source: http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/118-principal-component-analysis-in-r-prcomp-vs-princomp/

Independent Component Analysis (ICA)

ICA is a method for separating a multivariate signal into additive subcomponents by assuming that the subcomponents are non-Gaussian signals and that they are statistically independent from each other. ICA is a special case of blind source separation. A common example of ICA application is the "cocktail party problem" of listening in on one person's speech in a noisy room.

In PCA, the basis you get is the one that best explains the variability of your data; i.e. the first vector of the PCA basis is the one that best explains the variability of your data.

In ICA the basis you get is the one in which each vector is an independent component of your data, you can think of your data as a mix of signals and then the ICA basis will have a vector for each independent signal.

ICA is not part of this course but its good to know how it relates to PCA.   As an example, imagine 3 instruments playing simultaneously and 3 microphones recording the mixed signals. ICA is used to recover the sources, and PCA fails at recovering as shown below.

Figure Source. https://scikit-learn.org/stable/auto_examples/decomposition/plot_ica_blind_source_separation.html#sphx-glr-auto-examples-decomposition-plot-ica-blind-source-separation-py

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

from sklearn.decomposition import FastICA, PCA

# ######################################################################
# Generate sample data
np.random.seed(0)
n_samples = 2000
time = np.linspace(0, 8, n_samples)

s1 = np.sin(2 * time)  # Signal 1 : sinusoidal signal
s2 = np.sign(np.sin(3 * time))  # Signal 2 : square signal
```

Code Source. https://scikit-learn.org/stable/auto_examples/decomposition/plot_ica_blind_source_separation.html#sphx-glr-auto-examples-decomposition-plot-ica-blind-source-separation-py

References

1. Comon, P. (1994). Independent component analysis, a new concept?. *Signal processing*, *36*(3), 287-314.https://hal.archives-ouvertes.fr/hal-00417283/document
2. Hyvärinen, A., & Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural networks*, *13*(4-5), 411-430.
3. Bach, F. R., & Jordan, M. I. (2002). Kernel independent component analysis. *Journal of machine learning research*, *3*(Jul), 1-48. http://www.jmlr.org/papers/volume3/bach02a/bach02a.pdf

Application in remote sensing

https://www.mdpi.com/2072-4292/12/8/1261

Exercise 1

Task 1

1. Try PCA for Abalone Dataset to reduce the data set's dimensionality, with an explained variance ratio of 95%.
2. Train the reduced dataset with NN-Adam/SGD and then compare with results given by original dataset where no dimensionality reduction is done.

Task 2

1. Load the MNIST data set and split it into a training set and a test set (take the first 60,000 instances for training, and the remaining 10,000 for testing).
2. Train a Random Forest classifier on the data set and time how long it takes, then evaluate the resulting model on the test set.
3. Next, use PCA to reduce the data set's dimensionality, with an explained variance ratio of 95%.
4. Train a new Random Forest classifier on the reduced data set and see how long it takes.
5. Was training much faster?
6. Next, evaluate the classifier on the test set.
7. How does it compare to the previous classifier?

References

1. MNIST dataset in OpenML: https://www.openml.org/d/554
2. Python: https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html#sphx-glr-auto-examples-neural-networks-plot-mnist-filters-py
3. R https://tensorflow.rstudio.com/guide/tfestimators/examples/mnist/

Clustering

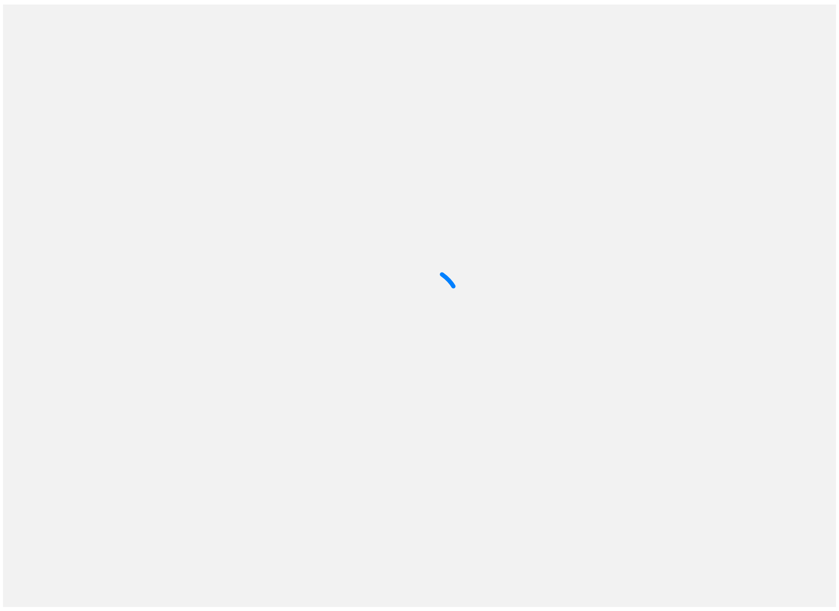
Clustering

Clustering is a task of finding groups of items that are similar. Importantly, clustering is an unsupervised task where a class (label) of any data instance is not known. The goal of clustering is to find a partition of N elements (instances) into homogeneous and well-separated clusters. Elements from same cluster should have high similarity, i.e, form a homogeneous cluster, while elements from different clusters should have low similarity, i.e., be well-separated.

We need to define **homogeneity** and **separation** measures. In practice, we need to use a distance measure appropriate to a given problem. Typically there are interactions between homogeneity and separation – usually, high homogeneity is linked with low separation, and vice versa, unless there is clear structure in the data.

The image below is an example of a bad cluster.

The clustering image below is an example of good clustering which satisfies both homogeneity and separation principles.



> "There is no universal definition of what a cluster is: it really depends on the context, and different algorithms will capture different kinds of clusters. Some algorithms look for instances centred around a particular point, called a centroid. Others look for continuous regions of densely packed instances: these clusters can take on any shape. Some algorithms are hierarchical, looking for clusters of clusters. And the list goes on." (Géron, 2019, p. 238)

Some of the possible applications of clustering:

- customer segmentation - based on their purchasing behaviours, activities and profiles.
- anomaly detection (outlier detection).
- image processing (image segmentation).
- semi-supervised learning - using few known labels to label other unlabelled instances.
- data analysis - identify clusters and examine each separately.

K-Means (Algorithm)

Let's understand and apply the K-Means algorithm, one of the most popular clustering algorithms. Here *k* refers to the number of clusters and *means* refers to *averaging* of the data in the process of finding the centroid. The overall framework of the algorithm is as below:

Steps:

1. Set value for k, the number of clusters (by prior knowledge or via search)
2. Initialise: choose points for centres (means) of k clusters (at random)
3. Procedure:
    1. assign each instance x_i to the closest of the k points
    2. re-assign the k points to be the means (μ_i) of each of the k clusters
    3. repeat 1 and 2 until convergence to a reasonably stable clustering

Below we see a visualisation of the process of some datasets and an application on the Iris dataset.

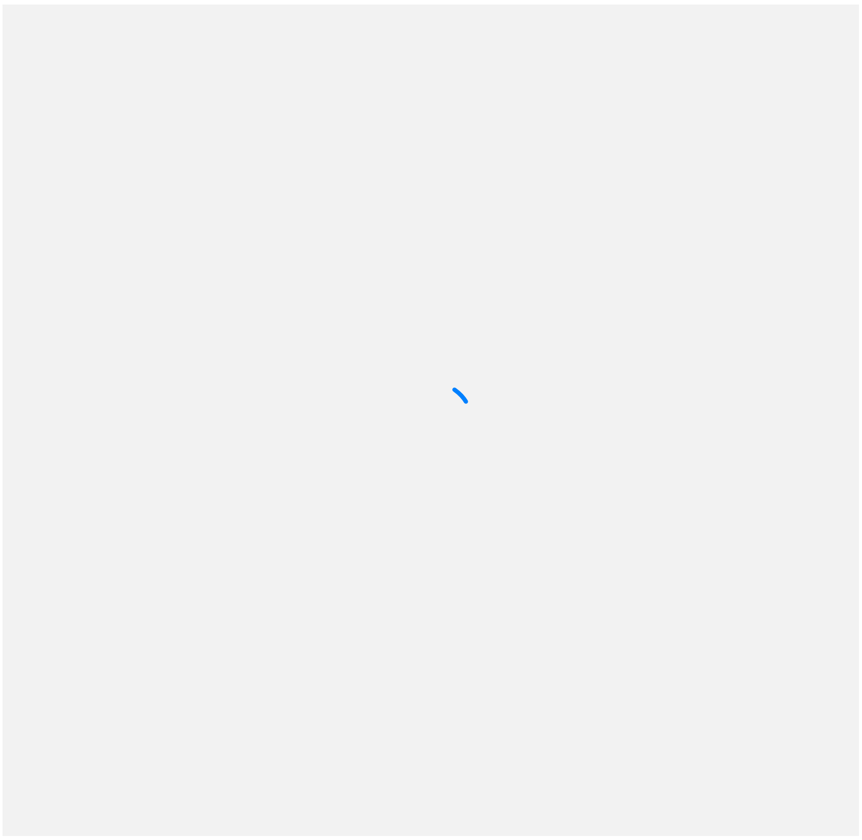


Figure Source. K-Means Clustering. Wikipedia: https://en.wikipedia.org/wiki/K-means_clustering

Figure Source. K-Means Clustering. Wikipedia: https://en.wikipedia.org/wiki/K-means_clustering

Given a set of observations ($x_1, x_2, ..., x_N$) of *d*-dimensional real vector, *k*-means clustering aims to partition the N observations into *k* (≤ *n*) sets $C = C_1, C_2, ..., C_k$ in order to minimize the within-cluster sum of squares.

The standard algorithm by Hartigan-Wong algorithm (1979) defines the total within-cluster variation as the sum of squared distances Euclidean distances between items and the corresponding centroid as given.

$$W(C_k) = \sum_{x_i \in C_k} (x_i - \mu_k)^2$$

where, x_i is a data point belonging to the cluster C_k and μ_k is the mean value of the points in cluster C_k.

We note that each observation (x_i) is assigned to a given cluster such that the sum of squares distance of the observation to their assigned cluster centres μ_k is minimized.

The total within-cluster sum of squares (WCSS) is given as follows

$$WCSS = \sum_{k=1}^{k} W(C_k) = \sum_{k=1}^{k} \sum_{x_i \in C_k} (x_i - \mu_k)^2$$

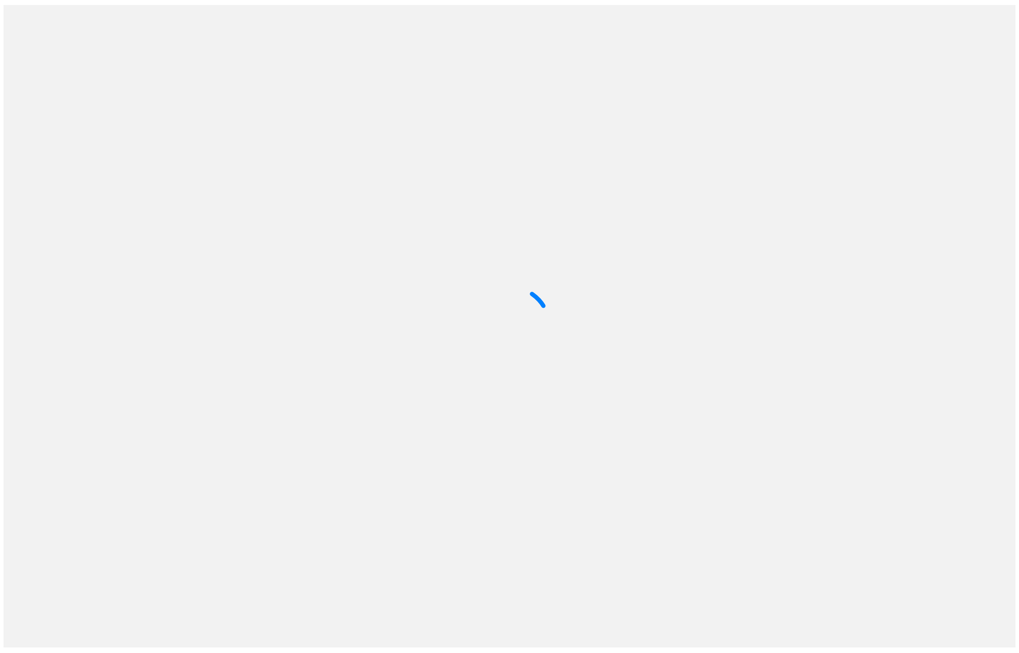
Below we see code from scratch in Python that implements the K-means clustering algorithm.

Run

PYTHON

```python
import matplotlib.pyplot as plt
from matplotlib import style
#style.use('ggplot')
import numpy as np


class K_Means:
    def __init__(self, k=2, tol=0.01, max_iter=300):
        self.k = k
        self.tol = tol
        self.max_iter = max_iter

        self.centroids = {}

```

The output would be



Code and Figure source: https://pythonprogramming.net/k-means-from-scratch-2-machine-learning-tutorial/?completed=/k-means-from-scratch-machine-learning-tutorial/

We review case when k is increased as shown below.

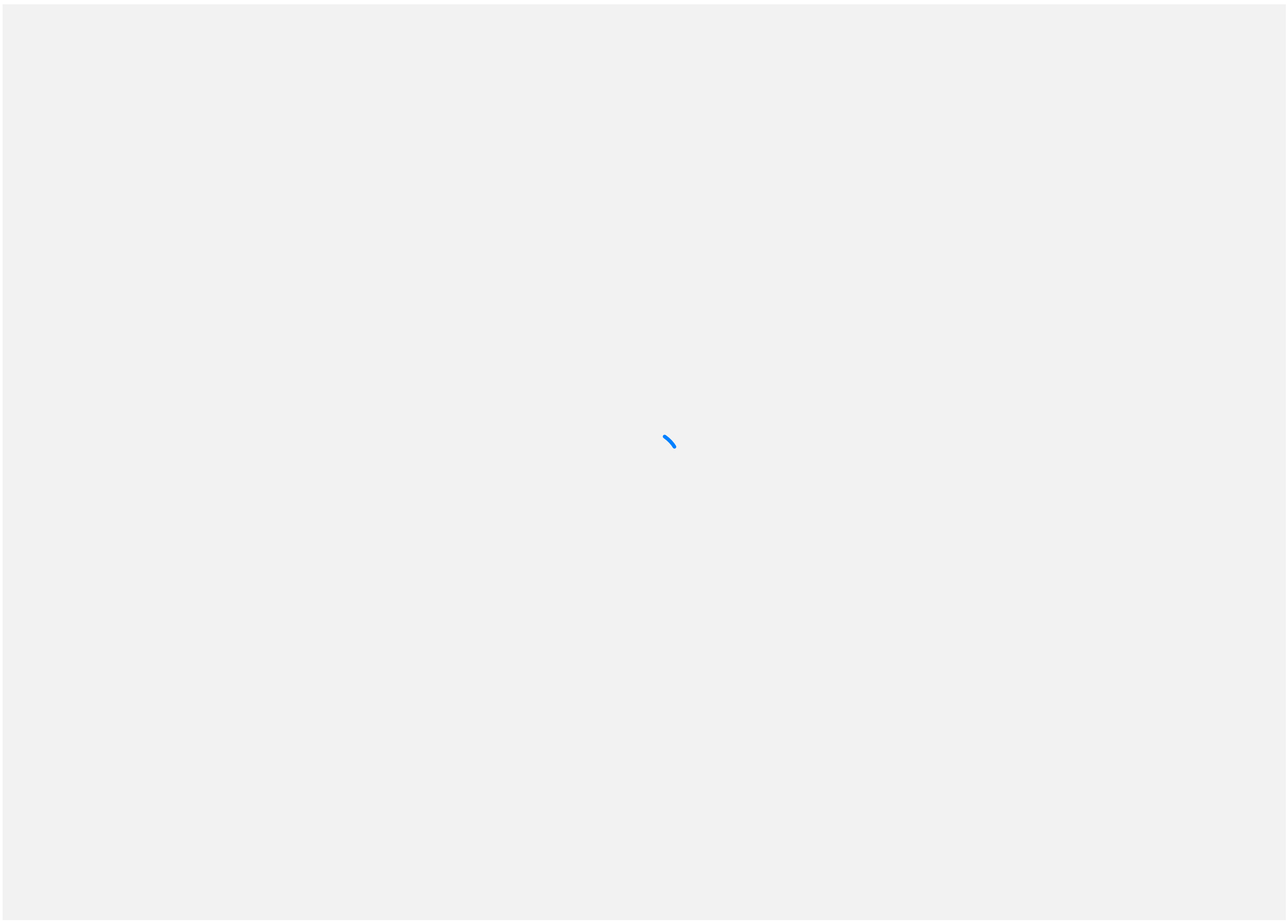


Figure Source. K-Means cluster analysis: https://uc-r.github.io/kmeans_clustering

Let's consider the following unlabelled data set, composed of five blobs of instances (from Chapter 9).

Figure: An unlabelled data set composed of five blobs of instances. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

As shown below, in three iterations the K-Means algorithm described above finalises five clusters.

Figure: The K-Means algorithm. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

The following code segment creates five clusters (k is set to 5) for the data set shown in the figure above. Each instance is assigned to one of the five clusters (labelled 0 to 4), and as shown below `KMeans` instance keeps a copy of the labels.

Run the following code.

```python
# Common imports
import numpy as np
import os
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
# Let's start by generating some blobs ---
blob_centers = np.array(
    [[ 0.2,  2.3],
     [-1.5 ,  2.3],
     [-2.8,  1.8],
     [-2.8,  2.8],
     [-2.8,  1.3]])
blob_std = np.array([0.4, 0.3, 0.1, 0.1, 0.1])
X, y = make_blobs(n_samples=2000, centers=blob_centers,
```

In the example above, it is easy to visualise the data set and decide on the number of the required clusters, i.e., the value of k. However, in many cases it is not easy to decide number of required clusters. We will discuss this point later.

The following figure shows decision boundaries of the clusters and each centroid is represented with an x.

Figure: K-Means decision boundaries. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

We can display the five centroids that the algorithm found as shown below.

```python
# Common imports
import numpy as np
import os
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
# Let's start by generating some blobs ---
blob_centers = np.array(
    [[ 0.2,  2.3],
     [-1.5 ,  2.3],
     [-2.8,  1.8],
     [-2.8,  2.8],
     [-2.8,  1.3]])
blob_std = np.array([0.4, 0.3, 0.1, 0.1, 0.1])
X, y = make_blobs(n_samples=2000, centers=blob_centers,
```

The following assigns new instances to the cluster whose centroid is closest:

Run the following code.

```python
# Common imports
import numpy as np
import os
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
# Let's start by generating some blobs ---
blob_centers = np.array(
    [[ 0.2,  2.3],
     [-1.5 ,  2.3],
     [-2.8,  1.8],
     [-2.8,  2.8],
     [-2.8,  1.3]])
blob_std = np.array([0.4, 0.3, 0.1, 0.1, 0.1])
X, y = make_blobs(n_samples=2000, centers=blob_centers,
```

Hard clustering vs soft clustering

Hard clustering assigns each instance to a single cluster.

Soft clustering assigns a score per cluster to each instance. This score may represent the distance between the instance and the respective centroid, or it could be a similarity score.

We can use the `transform()` method from the `KMeans` class to display the distance measures from each instance to every centroid.

```python
# Common imports
import numpy as np
import os
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
# Let's start by generating some blobs ---
blob_centers = np.array(
    [[ 0.2,  2.3],
     [-1.5 ,  2.3],
     [-2.8,  1.8],
     [-2.8,  2.8],
     [-2.8,  1.3]])
blob_std = np.array([0.4, 0.3, 0.1, 0.1, 0.1])
X, y = make_blobs(n_samples=2000, centers=blob_centers,
```

For a high-dimensional data set, often the above transformation is used to generate k-dimensional representative data set, a nonlinear dimensionality reduction technique.

Efficiency

In practice, K-Means algorithm is generally fast and linear with regards to the number of instances and the number of clusters. However, if there are no clustering structures in the data set, the time complexity could increase exponentially.

Suboptimal solutions

The selection of initial centroid plays an important role in deriving final clusters. As shown below, we could have ended up with suboptimal clusters if the initial randomly selected centroid were not appropriate for the data set.

Figure: Suboptimal solutions due to unlucky centroid initializations. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

Centroid initialisation methods

If we can approximately guess where the centroid values should be (say by visualising the data set or running other clustering algorithms), we should start clustering using these initial centroid values.

Alternatively we can run the algorithm multiple times with different random initialisations and keep the best solution. By default ,scikit-learn uses 10 random initialisations and keeps the best value. We can change the number of random initialisations by setting the corresponding parameter.

Internally the algorithm calculates each model's inertia, that is the mean squared distance between each instance and its closest centroid.

We can access a model's inertia value as shown below. The corresponding "score" value is negative considering we look for a higher score to select a suitable model.

```python
# Common imports
import numpy as np
import os
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
# Let's start by generating some blobs ---
blob_centers = np.array(
    [[ 0.2,  2.3],
     [-1.5 ,  2.3],
     [-2.8,  1.8],
     [-2.8,  2.8],
     [-2.8,  1.3]])
blob_std = np.array([0.4, 0.3, 0.1, 0.1, 0.1])
X, y = make_blobs(n_samples=2000, centers=blob_centers,
```

Mini-Batch K-Means

The Mini-Batch K-Means algorithm doesn't require the entire data set to be in memory at the same time. It uses mini-batches of the data set at a time and adjusts the centroids at each iteration. This is particularly useful when handling a large data set and also speeds up the performance by a factor of three or four.

```python
from sklearn.cluster import MiniBatchKMeans

minibatch_kmeans = MiniBatchKMeans(n_clusters=5)
minibatch_kmeans.fit(X)
```

For a large number of clusters, the Mini-Batch K-Means algorithm is much faster compared to the normal K-Means algorithm. However, as shown below, its inertia is slightly worse.

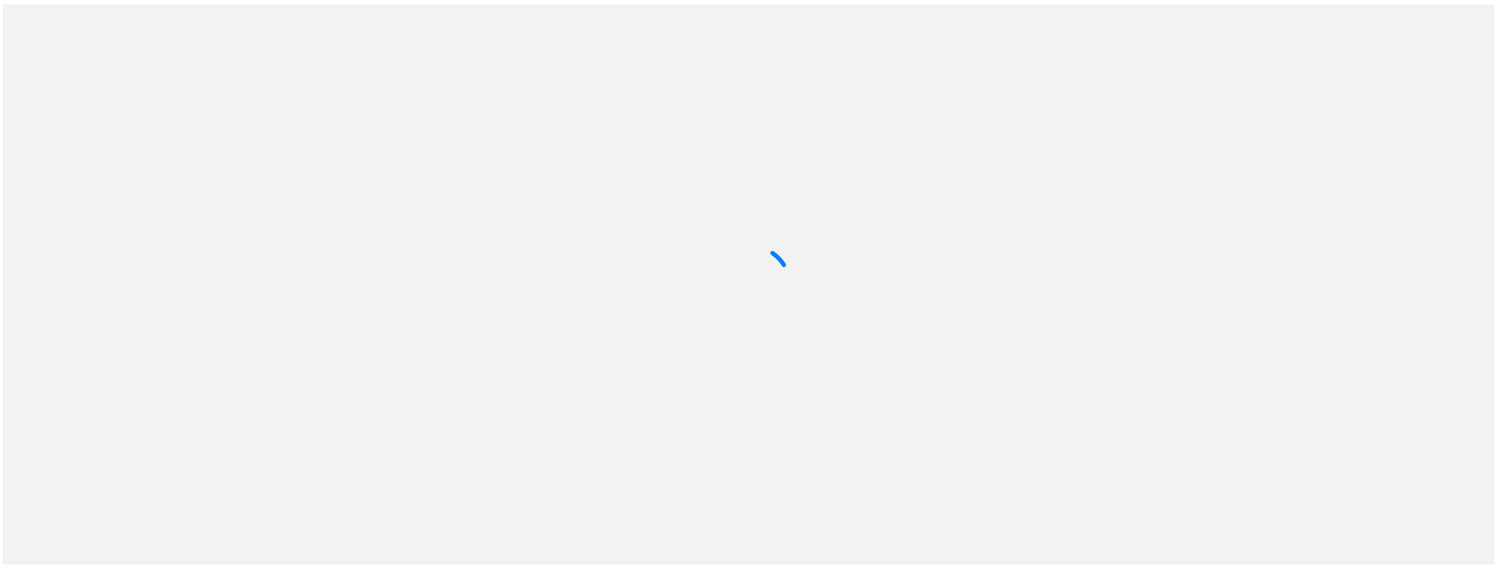


Figure: Mini-batch K-Means has a higher inertia than K-Means (left) but it is much faster (right), especially as k increases. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

Limits of K-Means

K-Means clustering does not perform very well when the clusters have varying sizes, different densities, or non spherical shapes. For example, in the following figure, both the solutions are unable to properly cluster elliptical shapes. We need to select a suitable clustering algorithm depending on the data set. As discussed in the textbook, Gaussian mixture models work better for such data sets.

It is advisable to scale the input features before we run K-Means algorithm to improve its performance.

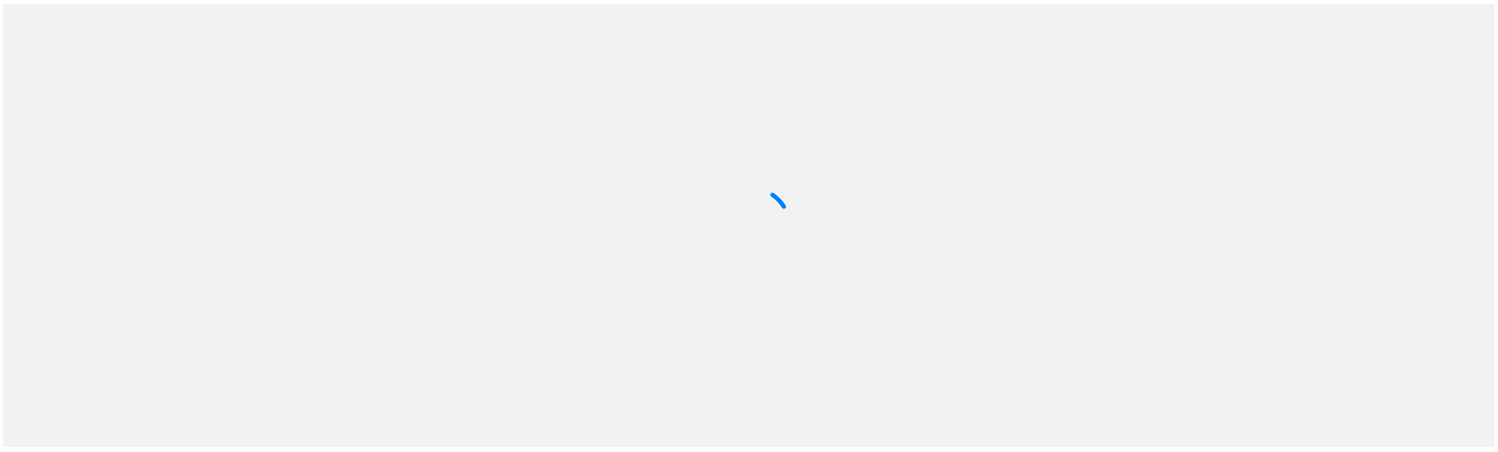


Figure: K-Means fails to cluster these ellipsoidal blobs properly. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

In R, we have an example below.

```r
#Source: https://uc-r.github.io/kmeans_clustering

library(tidyverse)  # data manipulation
library(cluster)    # clustering algorithms (Ed to install)
library(factoextra) # clustering algorithms & visualization (Ed to insta

df <- USArrests
#To remove any missing value that might be present in the data, type thi

df <- na.omit(df)
#As we don’t want the clustering algorithm to depend to an arbitrary
# variable unit, we start by scaling/standardizing the data using the R
# function scale:

```

References

1. MacQueen, J. Some methods for classification and analysis of multivariate observations. Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, 281--297, University of California Press, Berkeley, Calif., 1967. https://projecteuclid.org/euclid.bsmsp/1200512992
2. Alsabti, K., Ranka, S., & Singh, V. (1997). An efficient k-means clustering algorithm. https://surface.syr.edu/cgi/viewcontent.cgi?article=1042&context=eecs
3. Bradley, P. S., & Fayyad, U. M. (1998, July). Refining initial points for k-means clustering. In *ICML* (Vol. 98, pp. 91-99). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.8528&rep=rep1&type=pdf
4. Wagstaff, K., Cardie, C., Rogers, S., & Schrödl, S. (2001, June). Constrained k-means clustering with background knowledge. *ICML* (Vol. 1, pp. 577-584) https://web.cse.msu.edu/~cse802/notes/ConstrainedKmeans.pdf
5. Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., & Wu, A. Y. (2002). An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, *24*(7), 881-892. https://ieeexplore.ieee.org/iel5/34/21893/01017616.pdf
6. Fränti, P., & Sieranoja, S. (2019). How much can k-means be improved by using better initialization and repeats?. *Pattern Recognition*, *93*, 95-112. http://www.cs.joensuu.fi/pages/franti/sipu/pub/KM-Init.pdf

K-Means (Optimal Clusters)

Finding the optimal number of clusters

As shown in the figure below, often selecting a value of k is not easy.

Figure: Bad choices for the number of clusters: when k is too small, separate clusters get merged (left), and when k is too large, some clusters get chopped into multiple pieces (right). Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

In choosing k, we cannot only reply on the inertia value. This is because the inertial value normally increases with increase in k. The following figure plots the inertia as a function of the number of clusters k. The "elbow" at k=4 indicates that there is no significant decrease in inertia when we increase k, so there is no point in splitting clusters any further.

Figure: When plotting the inertia as a function of the number of clusters k, the curve often contains an inflexion point called the "elbow". Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

The following approach uses silhouette scores and offers more informative and precise representation for choosing k.

Silhouette diagrams

The following descriptions and definitions are supported by the information on pages 246 to 248 of the textbook.

An instance's silhouette score coefficient is defined as,

```
silhouette coefficient = (b-a)/max(a, b)
```

where,

- **a** is the mean distance to the other instances in the same cluster (i.e., the mean intra-cluster distance),
- **b** is the mean nearest-cluster distance (i.e., the mean distance to the instances of the next closest cluster, defined as the one that minimises b, excluding the instance's own cluster).

The silhouette coefficient can vary between –1 and +1.

- A coefficient close to +1 means that the instance is well inside its own cluster and far from other clusters.
- A coefficient close to 0 means that it is close to a cluster boundary.
- A coefficient close to –1 means that the instance may have been assigned to the wrong cluster.

The silhouette score is the mean silhouette coefficient over all the instances. To compute the silhouette score, you can use scikit-learn's `silhouette_score()` function, giving it all the instances in the data set and the labels they were assigned.

Run the following code to compute the silhouette score.

```python
# Common imports
import numpy as np
import os
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Let's start by generating some blobs ---
blob_centers = np.array(
    [[ 0.2,  2.3],
     [-1.5 ,  2.3],
     [-2.8,  1.8],
     [-2.8,  2.8],
     [-2.8,  1.3]])
```

The following figure compares the silhouette scores for different values of k.

Figure: Selecting the number of clusters k using the silhouette score. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

In the above figure we can see that k=4 is a very good option, and k=5 is also quite good, better than k=6 or k=7.

We can draw a silhouette diagram by plotting every instance's silhouette coefficient, sorted by the cluster they are assigned to and by the value of the coefficient.

Figure: Analysing the silhouette diagrams for various values of k. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

In analysing the diagrams we can see that:

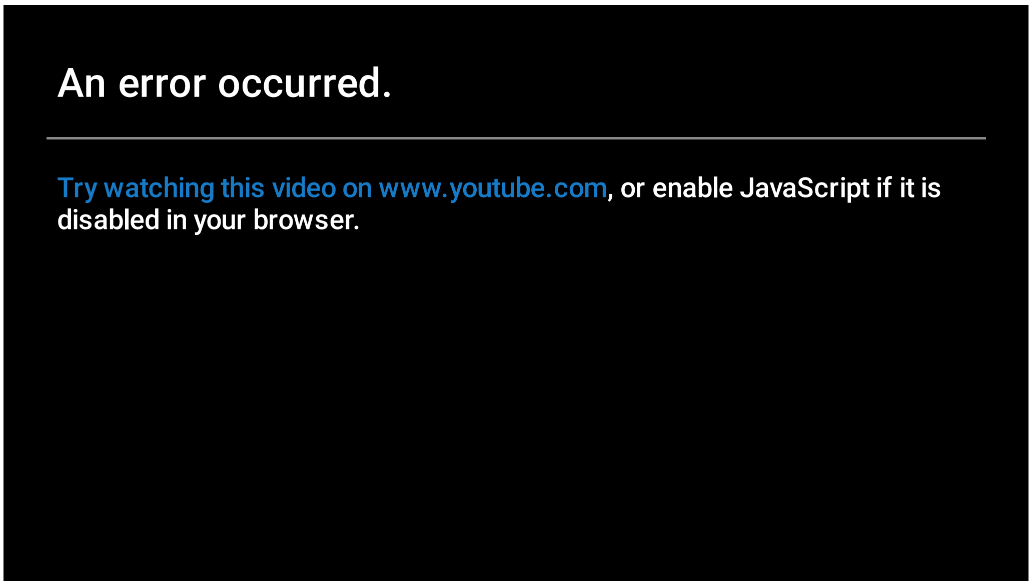
- Each diagram contains one knife shape per cluster.
- The shape's height indicates the number of instances the cluster contains.
- The shape's width represents the sorted silhouette coefficients of the instances in the cluster (wider is better).
- The dashed line indicates the mean silhouette coefficient for each k (number of clusters).
- A cluster is bad if most of the instances in that cluster have a lower coefficient than the dashed line.
- A cluster is good if most instances extend beyond the dashed line, to the right and closer to 1.0.

In the above diagram, both k=4 and k=5 are good clusters. However, when k=4, the third cluster from the top is very thick (very big). When k=5, all clusters of are reasonably equal sizes. So we may prefer k=5, instead of k=4.

Comparison with KNN

K-Means Clustering and k-Nearest Neighbors algorithm (KNN) are often confused with each other. The ‘K’ in K-Means Clustering has nothing to do with the ‘K’ in KNN algorithm. We note that K-Means Clustering is an unsupervised learning algorithm whereas KNN is a supervised learning algorithm used for classification.

The video below gives a summary of k-means clustering.



An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Hierarchical clustering

Agglomerative clustering is the most common type of hierarchical clustering used to group objects in clusters based on their similarity. It's also known as *AGNES* (*Agglomerative Nesting*) where the algorithm starts by treating each object as a singleton cluster. The pairs of clusters are successively merged until all clusters have been merged into one big cluster containing all objects. The result is a tree-based representation of the objects, named *dendrogram*.

The inverse of agglomerative clustering is *divisive clustering*, which is also known as DIANA (*Divise Analysis*) and it works in a "top-down" manner. It begins with the root, in which all objects are included in a single cluster.

Agglomerative clustering is good at identifying small clusters while divisive clustering is good at identifying large clusters. We will focus mainly on agglomerative hierarchical clustering.

- A hierarchy of clusters is built from the bottom up.
- At each iteration, agglomerative hierarchical clustering connects the nearest pair of clusters (starting with individual instances).
- It produces a flexible and informative cluster tree instead of forcing you to choose a particular cluster scale (size).

Below is a figure showing the process over a series of steps.

Figure Source. https://www.datanovia.com/en/lessons/agglomerative-hierarchical-clustering/

Figure: Dendrogram as the output of the Agglomerative Clustering. Adapted from "Breaking down the agglomerative clustering process" by C. Wijaya, 2019. (https://towardsdatascience.com/breaking-down-the-agglomerative-clustering-process-1c367f74c7c2)

Below you can see the dendrogram of hierarchical clustering for Iris dataset.

Figure Source. Hierarchical clustering. Wikipedia: https://en.wikipedia.org/wiki/Hierarchical_clustering

Lets look at a simple example where genes are clustered using the approach.

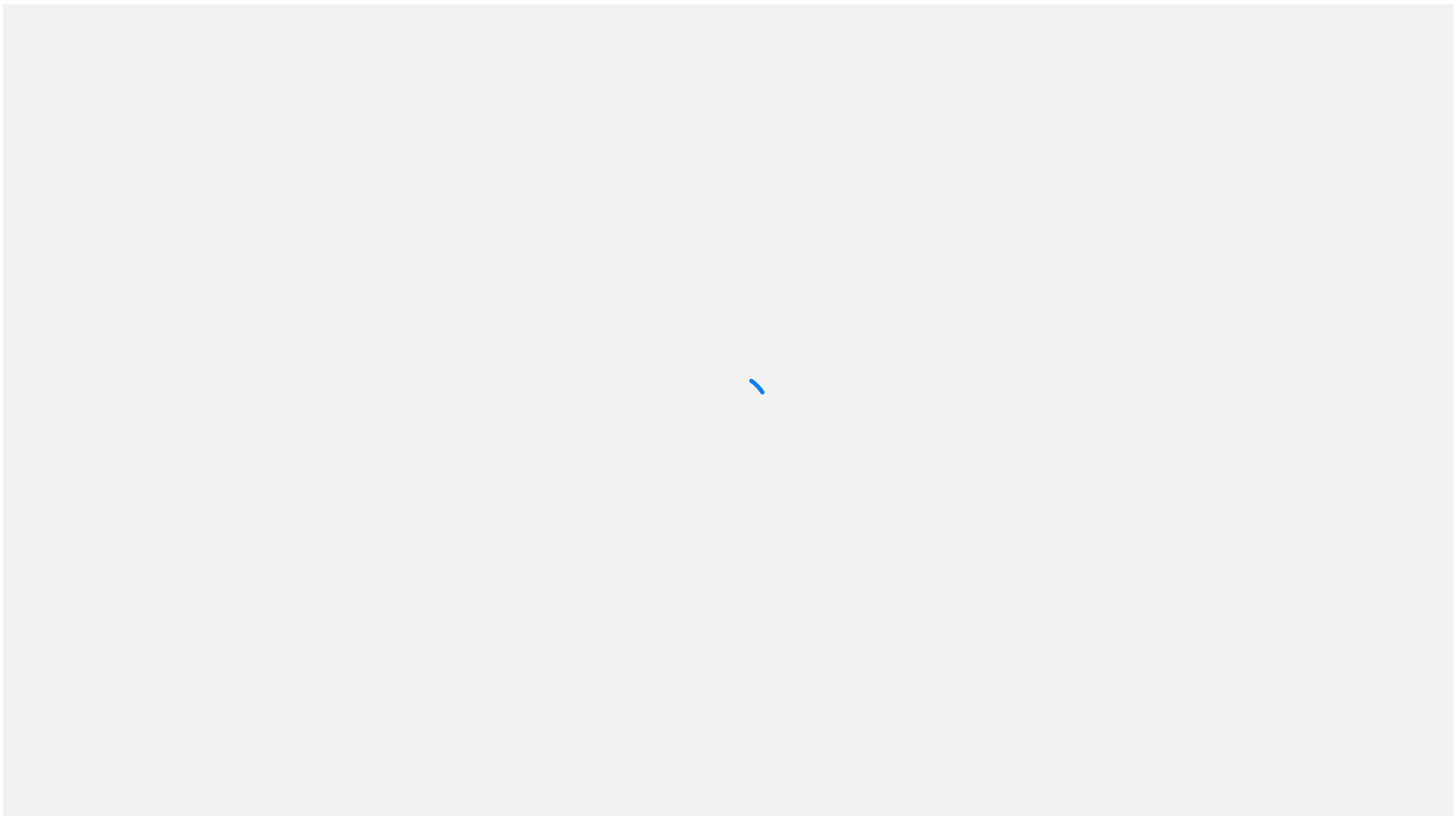


Figure Source. Hierarchical Clustering by StatQuest: https://www.youtube.com/watch?v=7xHsRkOdVwo&feature=emb_logo

Distance measures

Euclidean distance

$$d(a,b) = \sqrt{\sum_i (a_i - b_i)^2}$$

Squared Euclidean distance

$$d(a,b) = \sum_i (a_i - b_i)^2$$

Manhattan distance

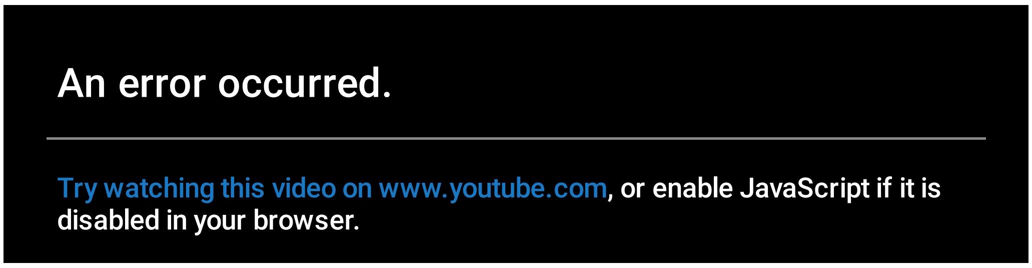
$$d(a,b) = \sum_i |a_i - b_i|$$

Figure Source. H. Clustering by StatQuest: https://www.youtube.com/watch?v=7xHsRkOdVwo&feature=emb_logo

R code

https://www.datanovia.com/en/lessons/agglomerative-hierarchical-clustering/

Video below summarising Hierarchical clustering.



An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

[redacted]

References

1. Johnson, S. C. (1967). Hierarchical clustering schemes. *Psychometrika*, *32*(3), 241-254.
2. Corpet, F. (1988). Multiple sequence alignment with hierarchical clustering. *Nucleic acids research*, *16*(22), 10881-10890. https://academic.oup.com/nar/article-pdf/16/22/10881/7047241/16-22-10881.pdf?casa_token=I1JVFaiT_SUAAAAA:zjHBR6kgawcRhHqMLYtY6T9lhjpAYNqGXtKP0U5f2CoovOYZzXT2pAuL9tCcFoN6bWVz5bISbBE8fTI
3. Olson, C. F. (1995). Parallel algorithms for hierarchical clustering. *Parallel computing*, *21*(8), 1313-1325.https://www2.eecs.berkeley.edu/Pubs/TechRpts/1994/CSD-94-786.pdf
4. Navarro, J. F., Frenk, C. S., & White, S. D. (1997). A universal density profile from hierarchical clustering. *The Astrophysical Journal*, *490*(2), 493.https://iopscience.iop.org/article/10.1086/304888/pdf
5. Heller, K. A., & Ghahramani, Z. (2005, August). Bayesian hierarchical clustering. In *Proceedings of the 22nd international conference on Machine learning* (pp. 297-304).https://dl.acm.org/doi/pdf/10.1145/1102351.1102389?casa_token=OaSVB-qRgiQAAAAA:lu7qKKaCX17Qg8sc_xsOjiKg92ziblhFiDy7kODqMFY-pz9ddUKfVea8TsqKosKkbpZppHw1vrDThkw

Density-based clustering

Density-based spatial clustering of applications with noise (DBSCAN) is a popular clustering algorithm that uses local density estimation to identify clusters of arbitrary shapes not possible with the K-Means algorithm. The following are some of the important points about DBSCAN algorithms drawn from the textbook.

- For each instance, the algorithm counts how many instances are located within a small distance ε (epsilon) from it. This region is called the instance's ε-neighbourhood.
- If an instance has at least min_samples instances in its ε-neighbourhood (including itself), then it is considered a core instance. In other words, core instances are those that are located in dense regions.
- All instances in the neighbourhood of a core instance belong to the same cluster. This neighbourhood may include other core instances; therefore, a long sequence of neighbouring core instances forms a single cluster.
- Any instance that is not a core instance and does not have one in its neighbourhood is considered an anomaly. (Géron, 2019, pp. 255-266)

This algorithm works well if all the clusters are dense enough and if they are well separated by low-density regions.

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (Vol. 96, No. 34, pp. 226-231). https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf?source=post_page

Figure Source. DBSCAN https://www.youtube.com/watch?v=6jl9KkmgDIw

The following code segment uses DBSCAN class for clustering the moons data set.

```python
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons
import numpy as np

X, y = make_moons(n_samples=1000, noise=0.05)
print(X)
dbscan = DBSCAN(eps=0.05, min_samples=5)
dbscan.fit(X)
print('dbscan.labels_[:10]: ', dbscan.labels_[:10])
print('np.unique(dbscan.labels_): ', np.unique(dbscan.labels_))


#eps, default=0.5
#The maximum distance between two samples for one to be considered
```

The clustering drawn from the code above is displayed in the left-hand side of the following figure. There are seven clusters and many anomalies are detected (denoted by the red crosses). However, if we increase `eps` to 0.2, we get better clustering as shown by the result on the right.

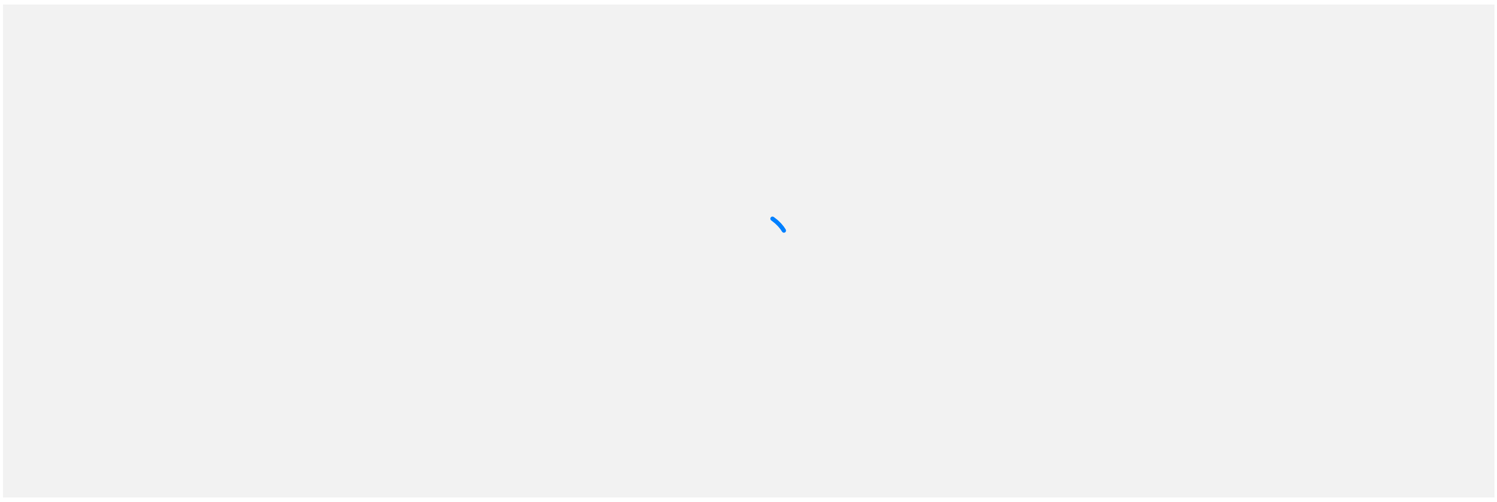


Figure: DBSCAN clustering using two different neighbourhood radiuses. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

One of the most useful features of DBSCAN algorithm is that it can identify clusters of any shapes, and it is also robust to outliers. It is simple to use with just two hyperparameters (eps and min_samples).

Below another example.

Run PYTHON

```python
import numpy as np

from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler


# #######################################################################
# Generate sample data
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=750, centers=centers, cluster_std=
                            random_state=0)

```

Code and Figure Source: https://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html#sphx-glr-auto-examples-cluster-plot-dbscan-py

References

1. Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (Vol. 96, No. 34, pp. 226-231). https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf?source=post_page

Other clustering methods

Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)

BIRCH is specifically developed to deal with very large data sets. It builds a tree structure capturing the required information such that a new instance can be quickly assigned to a cluster, without a need to store all the instances. The algorithm is faster than batch K-Means and requires less memory even while handling large data sets.

Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: an efficient data clustering method for very large databases. *ACM sigmod record*, *25*(2), 103-114. https://dl.acm.org/doi/pdf/10.1145/235968.233324

Below we see an example of BIRCH.

Run PYTHON

```

from sklearn.cluster import Birch
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(12)
p1 = np.random.randint(5,21,110)
p2 = np.random.randint(20,30,120)
p3 = np.random.randint(8,21,90)

data = np.array(np.concatenate([p1, p2, p3]))
x_range = range(len(data))
x = np.array(list(zip(x_range, data))).reshape(len(x_range), 2)

```

Code Source: https://www.datatechnotes.com/2019/09/clustering-example-with-birch-method-in.html

Spectral clustering

Spectral clustering transforms a clustering problem into a graph partitioning problem. The aim is to identify sub graphs (say, representing communities) based on the connections between nodes. As described by AlindGupta (n.d.), there are three main steps.

- Building the similarity graphs
- Projecting the data onto a lower Dimensional Space

- Clustering the Data

Below we can see the performance for spectral clustering for complicated problems.

Figure Source. Spectral Clustering: https://towardsdatascience.com/spectral-clustering-82d3cff3d3b7

The data points are treated as nodes of a graph and hence clustering is treated as a graph partitioning problem. The nodes are then mapped to a low-dimensional space that can be easily segregated to form clusters.

Please refer to AlindGupta (n.d.) for more detailed description and one Python example on spectral clustering in scikit-learn. https://www.geeksforgeeks.org/ml-spectral-clustering/

Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, *17*(4), 395-416. https://link.springer.com/content/pdf/10.1007/s11222-007-9033-z.pdf

Below is an overview of how to use it.

Run PYTHON

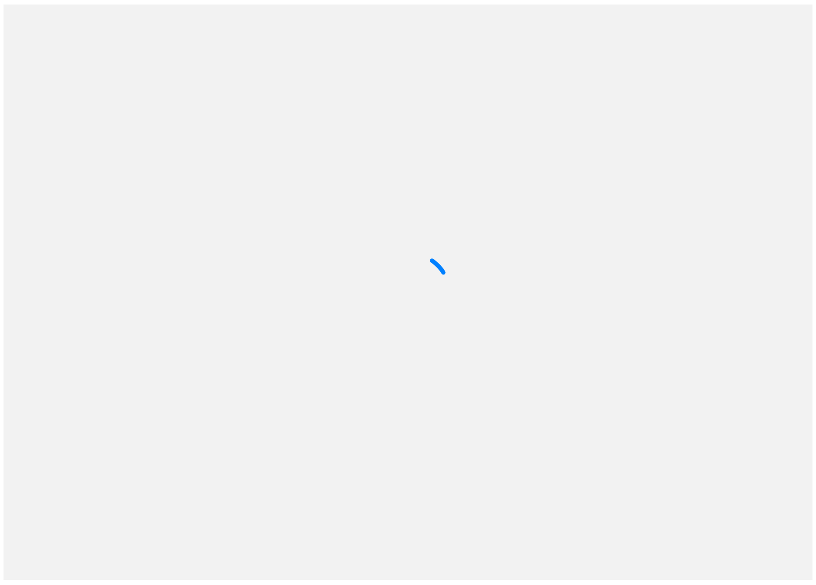
```python
from sklearn.cluster import SpectralClustering
import numpy as np
X = np.array([[1, 1], [2, 1], [1, 0],
              [4, 7], [3, 5], [3, 6]])
clustering = SpectralClustering(n_clusters=2,assign_labels="discretize",
print(clustering)

```

Code Source: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html

Gaussian mixture model (GMM)

GMM is a probabilistic model that assumes that the instances were generated from a mixture of several Gaussian distributions whose parameters are unknown. Such models are useful for clustering, anomaly detection and density estimation.

k-means model places a circle (and in higher dimensions, a hyper-sphere) at the center of each cluster. A radius is defined by the most distant point in the cluster. In contrast, Gaussian mixture models can handle even oblong or ellipsoidal form of clusters. The figure below shows the details where a circular cluster is used (top) and ellipsoidal clusters (bottom).



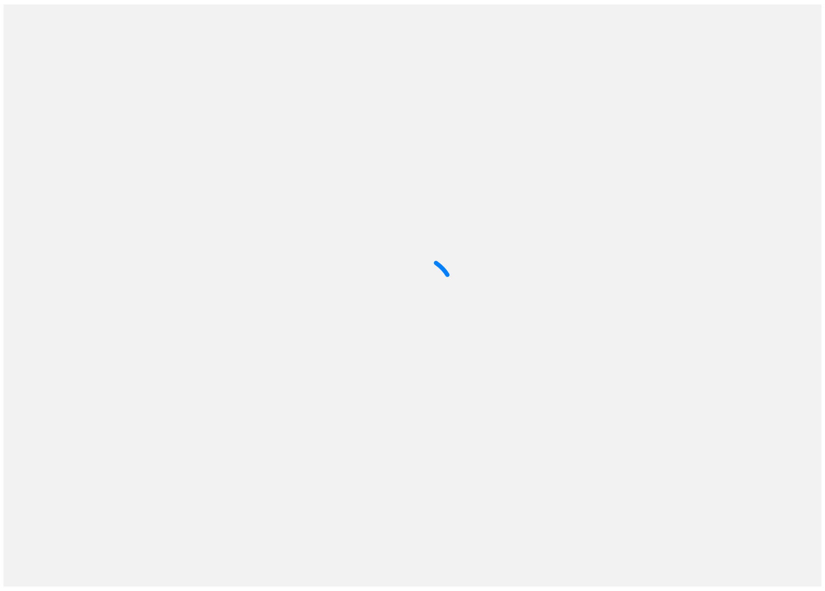


Figure Source: https://towardsdatascience.com/gaussian-mixture-models-d13a5e915c8e

Below we can see that there are three Gaussian functions, hence *K* = 3; where each Gaussian distribution explains the data contained in each of the three clusters available. The mixing coefficients for each cluster k are themselves probabilities π_k and must have a sum of 1 as shown below.

$$\sum_{k=1}^{K} \pi_k = 1$$

Figure Source. GMM Explained: https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95

Note that GMM for clustering is not covered in detail in the course. This is more for information purpose.

More information:

Biernacki, C., Celeux, G., & Govaert, G. (2000). Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE transactions on pattern analysis and machine intelligence*, *22*(7), 719-725.

More information: https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95 https://www.kaggle.com/vipulgandhi/gaussian-mixture-models-clustering-explained

Comparision

Below we see a comparison of different clustering algorithms. Note the performance of those covered in the lesson. The others are for information purpose.

Figure Source: https://scikit-learn.org/0.15/auto_examples/cluster/plot_cluster_comparison.html

Below is code that compares the different methods.

Run PYTHON

```python


import time

import numpy as np
import matplotlib.pyplot as plt

from sklearn import cluster, datasets
from sklearn.metrics import euclidean_distances
from sklearn.neighbors import kneighbors_graph
from sklearn.preprocessing import StandardScaler

np.random.seed(0)

```

Code Source: https://scikit-learn.org/0.15/auto_examples/cluster/plot_cluster_comparison.html

References

1. Biernacki, C., Celeux, G., & Govaert, G. (2000). Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE transactions on pattern analysis and machine intelligence*, *22*(7), 719-725.
2. Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, *17*(4), 395-416. https://link.springer.com/content/pdf/10.1007/s11222-007-9033-z.pdf
3. Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: an efficient data clustering method for very large databases. *ACM sigmod record*, *25*(2), 103-114. https://dl.acm.org/doi/pdf/10.1145/235968.233324

Anomaly detection (outlier detection)

Anomaly detection (also called outlier detection) is the task of detecting instances that deviate strongly from the norm. These instances are called anomalies, or outliers, while the normal instances are called inliers.

Anomaly detection is useful in a wide variety of applications, such as fraud detection, detecting defective products in manufacturing, or removing outliers from a data set before training another model (which can significantly improve the performance of the resulting model).

The clustering algorithms are often used to learn what “normal” data looks like, and then use that information to detect abnormal instances.

Any instance that has a low affinity to all the clusters is likely to be an anomaly. For example, if you have clustered the users of your website based on their behaviour, you can detect users with unusual behaviour, such as an unusual number of requests per second.

The following algorithms, available in scikit-learn, are dedicated to anomaly detection or novelty detection. They are not part of this course, however it may be useful for you to be aware of their functionality, so you can use them in the future if required.

- PCA
- Fast-MCD (minimum covariance determinant)
- Isolation Forest
- Local Outlier Factor (LOF)
- One-class SVM.

References

1. Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, *41*(3), 1-58.
2. Mahadevan, V., Li, W., Bhalodia, V., & Vasconcelos, N. (2010, June). Anomaly detection in crowded scenes. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 1975-1981). IEEE.
3. Lazarevic, A., Ertoz, L., Kumar, V., Ozgur, A., & Srivastava, J. (2003, May). A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the 2003 SIAM international conference on data mining* (pp. 25-36). Society for Industrial and Applied Mathematics. http://www.academia.edu/download/49113125/A_20comparative_20study_20of_20anomaly_20detection_20schemes_20in_20network_20intrusion_20detection.pdf
4. Eskin, E., Arnold, A., Prerau, M., Portnoy, L., & Stolfo, S. (2002). A geometric framework for

unsupervised anomaly detection. In *Applications of data mining in computer security* (pp. 77-101). Springer, Boston, MA. https://academiccommons.columbia.edu/doi/10.7916/D8HT2W48/download

Code examples in R

Run

R

```r
#source https://www.datanovia.com/en/lessons/k-means-clustering-in-r-alg
#data source: https://www.kaggle.com/deepakg/usarrests


data("USArrests")       # Loading the data set
df <- scale(USArrests) # Scaling the data

# View the firt 10 rows of the data
head(df, n = 10)

#The standard R function for k-means clustering is kmeans()
# [stats package], which simplified format is as follow:
#kmeans(x, centers, iter.max = 10, nstart = 1)

```

The above code employs R for k-means clustering for US Arrests data sets that looks at the different type of crime for each area, where the 4 clusters are identified as follows.

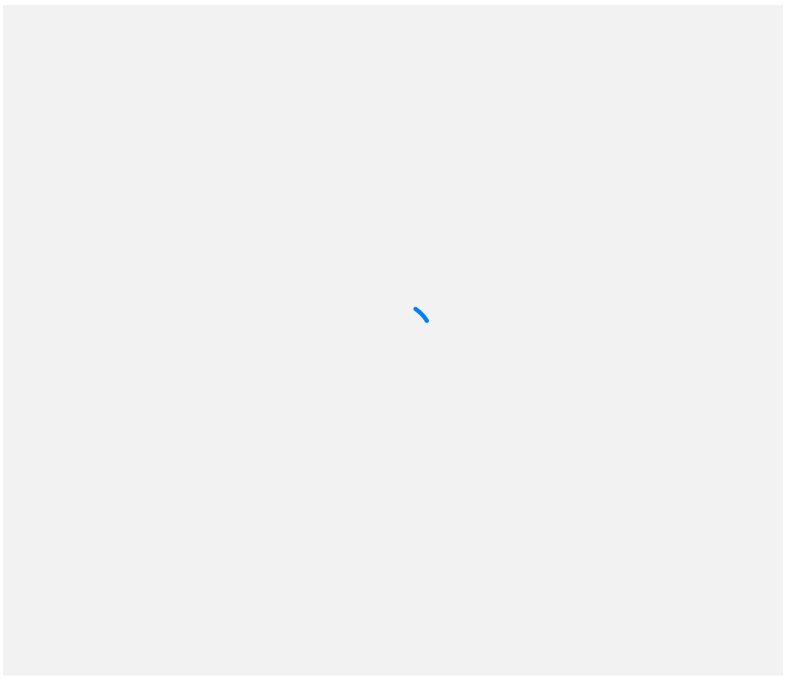


Figure: Clusters in data. Retrieved from https://www.datanovia.com/en/lessons/k-means-clustering-in-r-algorith-and-practical-examples/.

K-Means challenge 1

Use the Olivetti faces data set to complete the K-Means challenge.

The classic Olivetti faces data set contains 400 greyscale 64 × 64–pixel images of faces. Each image is flattened to a 1D vector of size 4,096. 40 different people were photographed (10 times each), and the usual task is to train a model that can predict which person is represented in each picture.

Load the data set using the `sklearn.datasets.fetch_olivetti_faces()` function, then split it into a training set, a validation set, and a test set (note that the data set is already scaled between 0 and 1).

Since the data set is quite small, you probably want to use stratified sampling to ensure that there are the same number of images per person in each set.

Next, cluster the images using K-Means, and ensure that you have a good number of clusters (using one of the techniques discussed in this chapter).

Visualise the clusters: do you see similar faces in each cluster?

Provide your answers in the file `kmeans_1.py`. The code to display faces is available in the file `plot_faces_lib.py`.

K-Means challenge 2

Use the Olivetti faces data set to complete the K-Means challenge 2.

Continuing with the Olivetti faces data set, train a classifier to predict which person is represented in each picture, and evaluate it on the validation set.

Next, use K-Means as a dimensionality reduction tool, and train a classifier on the reduced set.

Search for the number of clusters that allows the classifier to get the best performance.

- What performance can you reach?
- What if you append the features from the reduced set to the original features (again, searching for the best number of clusters)?

Provide your answers in the file `kmeans_2.py`. The code from the previous exercise is available in the files `kmeans_1.py` and `plot_faces_lib.py`.

Exercise 2

With either Python or R, Compare the performance of K-means, Hierarchical, and Density-based clustering for the features of the Iris, Parkinsons, and Abalone dataset.