Methods 3: Multilevel Statistical Modeling and Machine Learning

Week 9: *Dimensionality Reduction, Principled Component Analysis (PCA)*

November 23, 2021

by: Lau Møller Andersen

These slides are distributed according to the CC BY 4.0 licence:
https://creativecommons.org/licenses/by/4.0/

Did you learn?

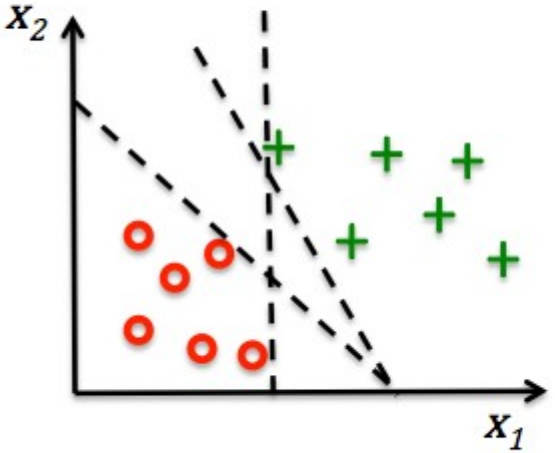
Logistic regression (machine learning)

1) Understanding of how logistic regression can be adapted to a classification framework

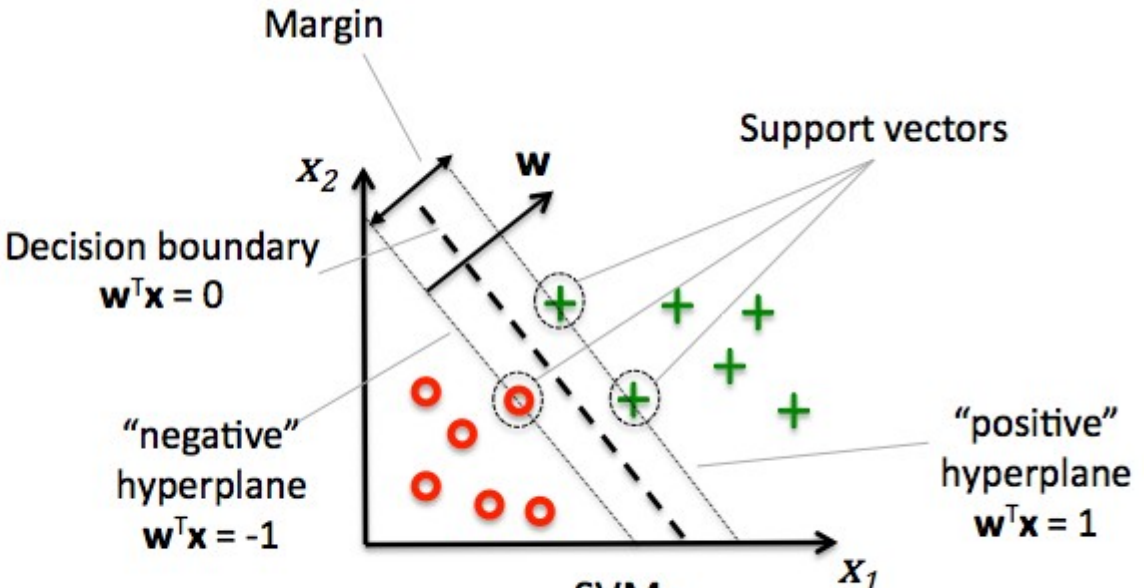
2) Understanding the idea of a Support Vector Machine

3) Getting acquainted with how Support Vector Machines can solve non-linear problems

SUPPORT VECTOR MACHINES
Recapitulation



Which hyperplane?



SVM:
Maximize the margin

(p. 69: Raschka, 2015)

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

Creating the higher dimensions can be computationally expensive

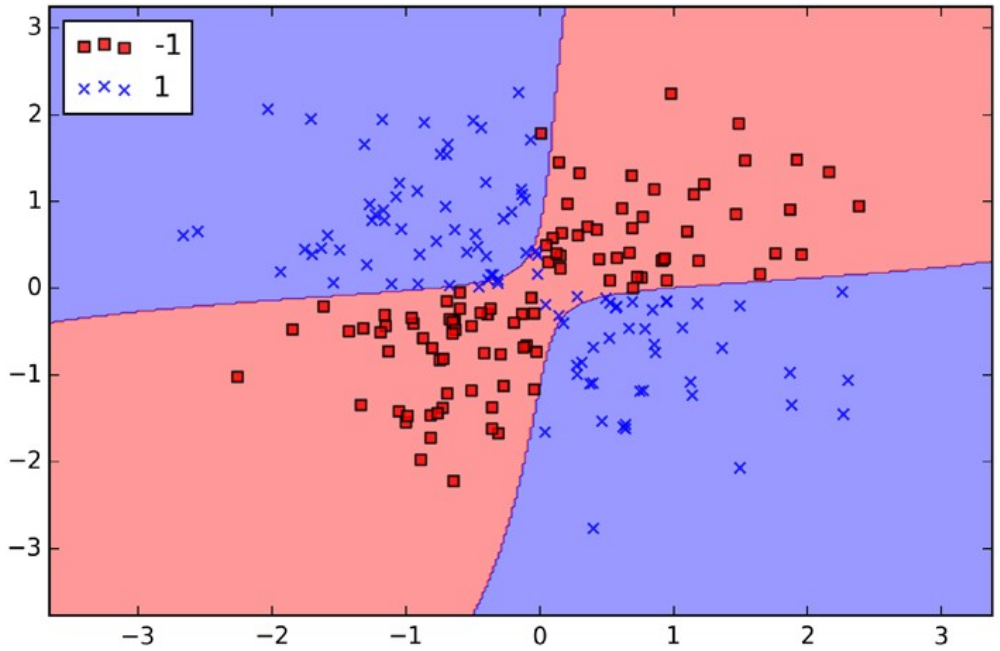
Kernel (*k*) function

$$k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \phi(\boldsymbol{x}^{(i)})^T \phi(\boldsymbol{x}^{(j)})$$

$$k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = e^{(-\gamma \left\| \boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)} \right\|^2)}$$

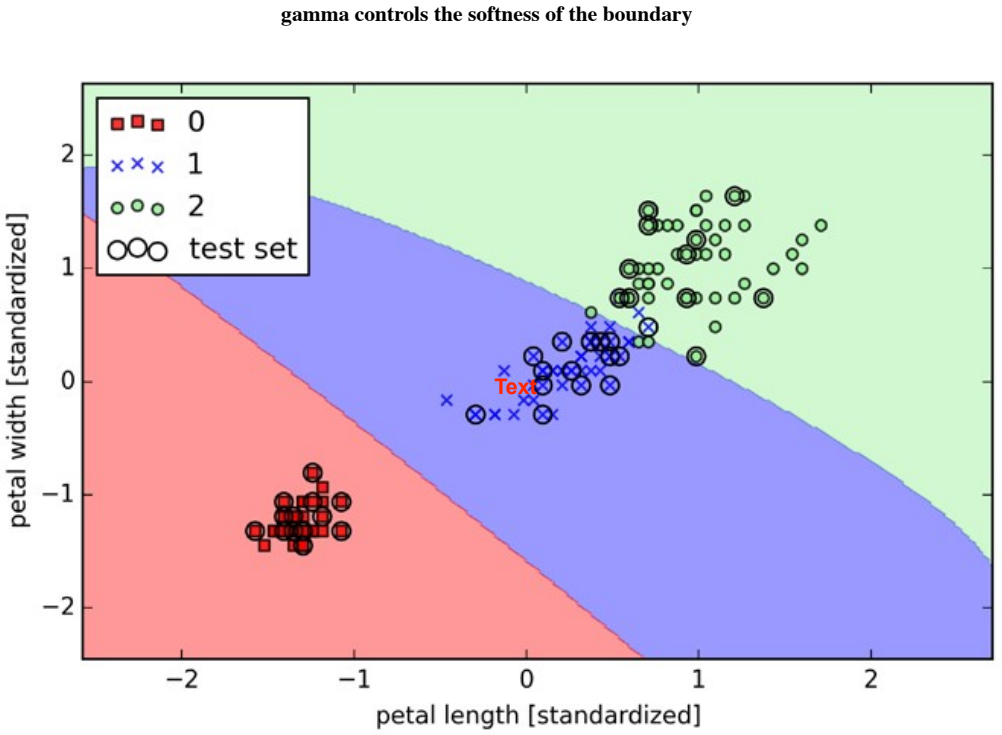
$$\left(\gamma = \frac{1}{2\sigma^2}, \text{ also called the precision} \right)$$

Non-linear decision boundaries



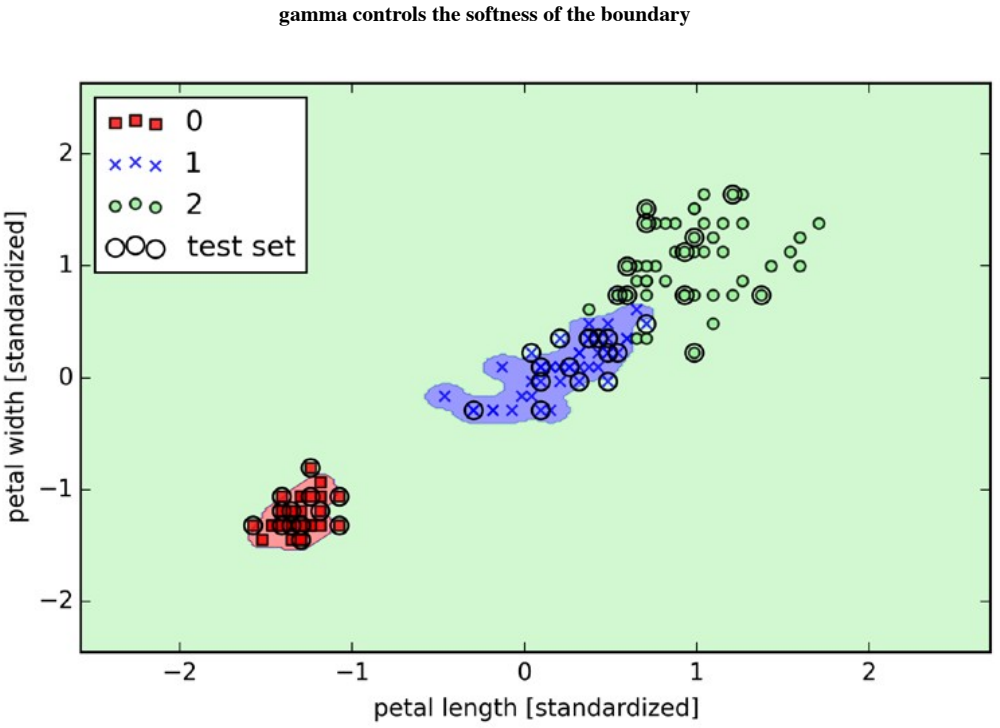
(p. 78: Raschka, 2015)

Low γ - soft boundary



(p. 79: Raschka, 2015)

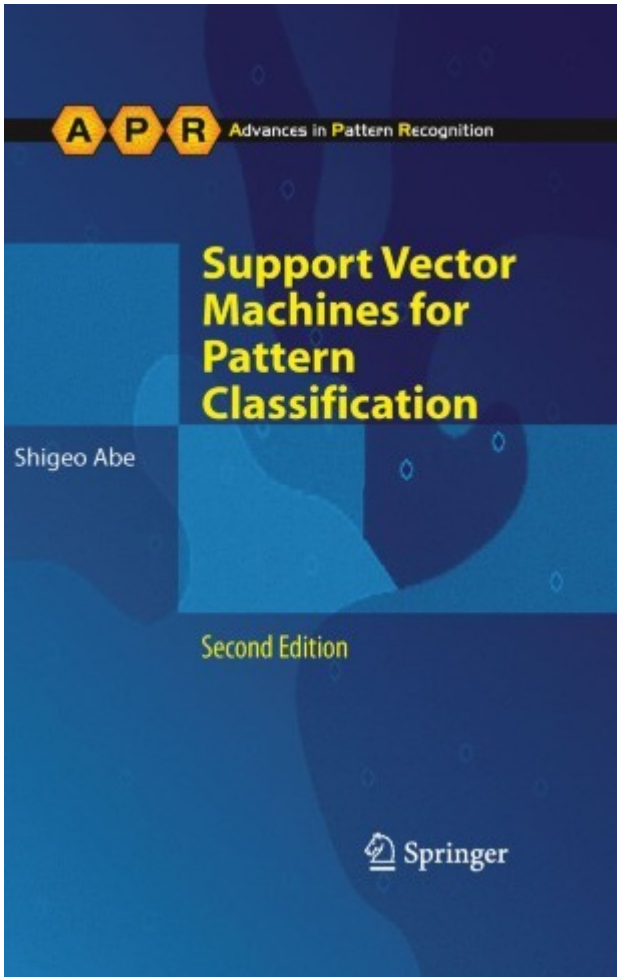
High γ - tight boundary



(p. 80: Raschka, 2015)

Live coding

RECAPITULATION_SUPPORT_VECTOR_MACHINE.ipynb



Available online on The Royal Library

Learning goals

Dimensionality reduction

1) Learning how we can extract the features that explain the most variance

2) Understanding how that can improve classification

3) Get acquainted with the concept of a eigenvector

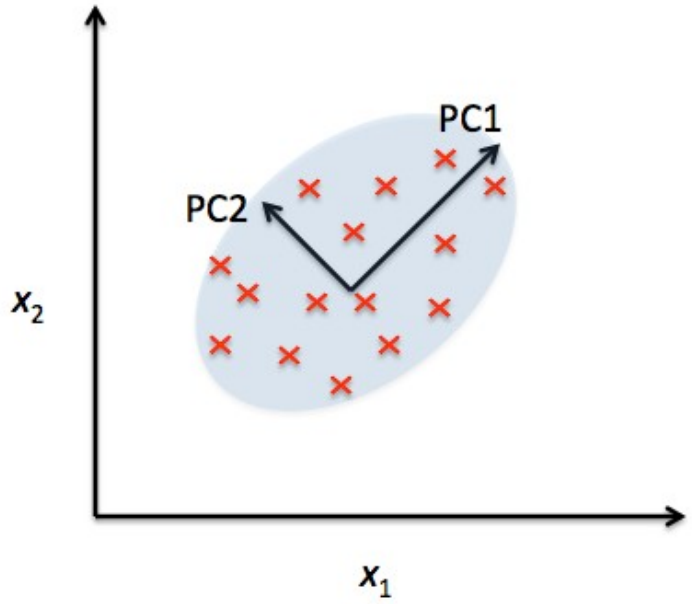
The curse of dimensionality

```
import numpy as np
import matplotlib.pyplot as plt
from os.path import join

path = '/home/lau/Skrivebord/class_subject'
data = np.load(join(path, 'megmag_data.npy'))

print('Shape: ' + str(data.shape))
print('n measurements: ' + str(np.prod(data.shape)))
print('n observations: ' + str(data.shape[0]))
print('n features: ' + str(np.prod(data.shape[1:])))
```

```
Shape: (682, 102, 251)
n measurements: 17460564
n observations: 682
n features: 25602
```

Principled components

FINDING THE DIRECTIONS OF MOST VARIANCE

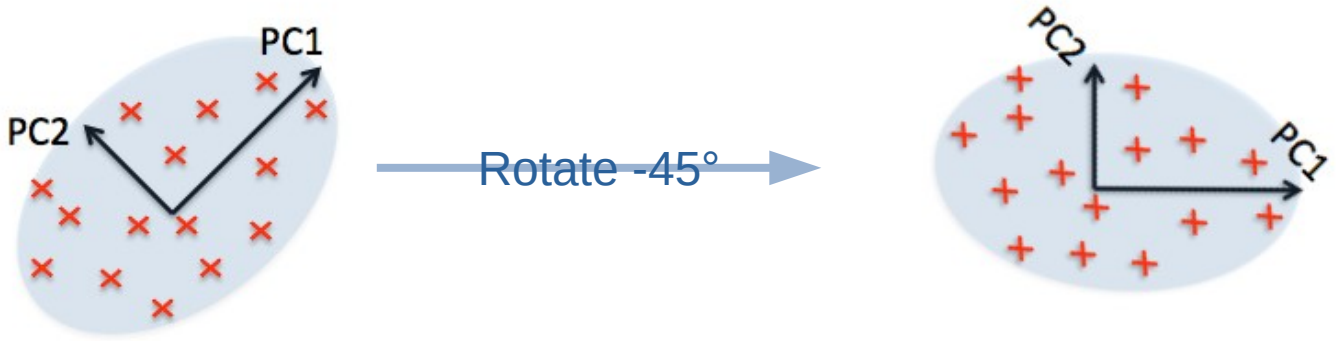


(p. 128: Raschka, 2015)

Principled components

FINDING THE DIRECTIONS OF MOST VARIANCE

why are they orthogonal (retvinklet)?
- we wanna find vectors that are orthogonal in order for them to share variance. :)
we waant ecah of the pca's to explain new parts of the variance



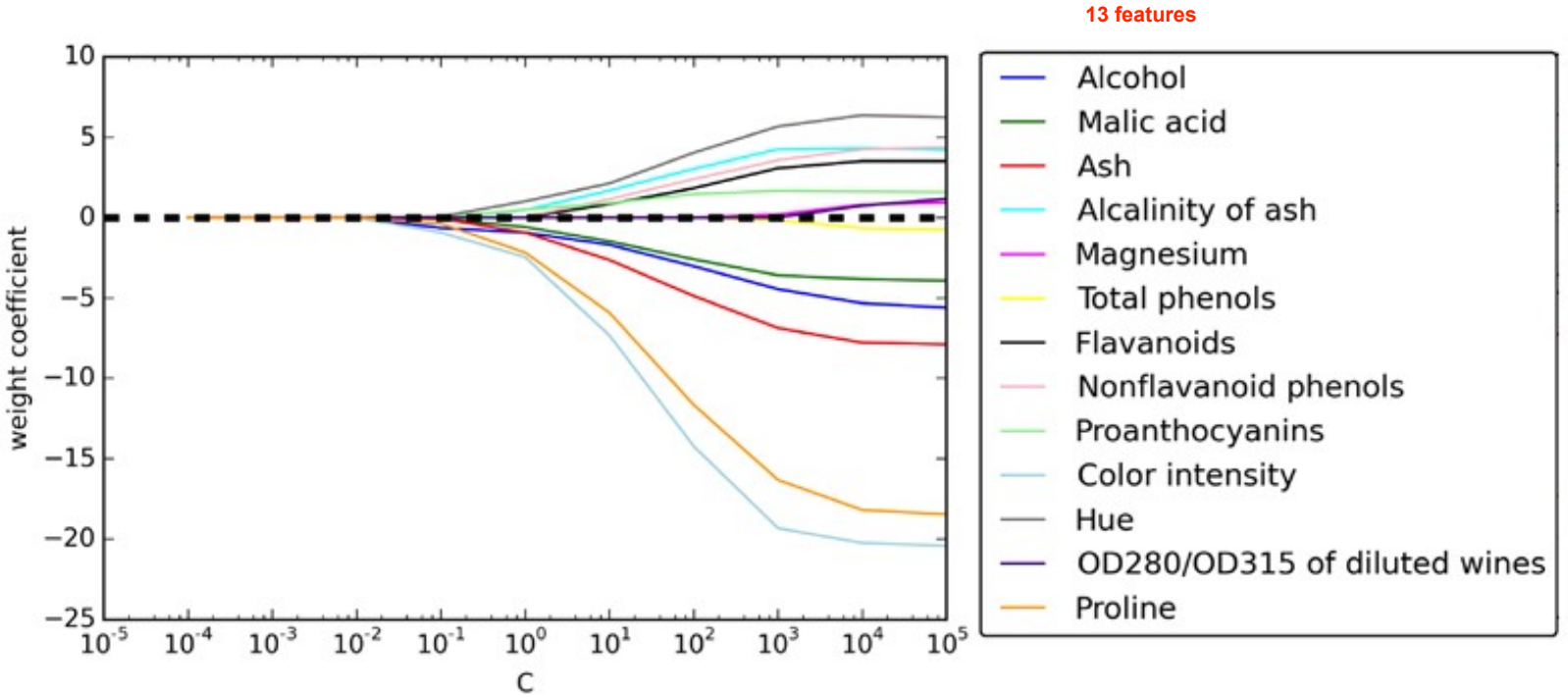
PC1 has the most variance

we are gonna find vectors (PC! and PC2)) that explains the most variance

(p. 128: Raschka, 2015)

This can be **generalized** to as many dimensions as you like

A dataset (wine)



(p. 118: Raschka, 2015)

```
## import wine data
import pandas as pd
url = 'https://archive.ics.uci.edu/ml/' + \
        'machine-learning-databases/wine/wine.data'
df_wine = pd.read_csv(url, header=None)
```

```
X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values
print(X.shape)
print(np.unique(y))
```

```
(178, 13)
[1 2 3]
```

X: 13 different kinds of features

y= targets (type of wine)

AIM: for $\boldsymbol{x}$: find $\boldsymbol{W}$ such that:

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_d], \boldsymbol{x} \in \mathbb{R}^{\boldsymbol{d}}$$

$$\downarrow \boldsymbol{xW}, \boldsymbol{W} \in \mathbb{R}^{d \times k}$$

$$\boldsymbol{z} = [z_1, z_2, \ldots, z_k], \boldsymbol{z} \in \mathbb{R}^{\boldsymbol{k}}$$

The z is then our new data after dimension reduction

k = the number of dimenssions we wanna end up with (reduced amount of dimensions)

d = 13

Approach

PRINCIPLED COMPONENT ANALYSIS

1) Standardize the *d*-dimensional dataset
2) Construct the covariance matrix
3) Decompose the covariance matrix into its eigenvectors and eigenvalues
4) Select *k* eigenvectors that correspond to the *k* largest eigenvalues where *k* is the dimensionality of the new feature subspace ($k \leq d$)

we cant have more dimensions in the new data set after dimension reduction:)

5) Construct a projection matrix ***W*** from the "top" *k* eigenvectors
6) Transform the *d*-dimensional input dataset ***X*** using the projection matrix ***W*** to obtain the new *k*-dimensional feature subspace

(p. 129: Raschka, 2015)

Standardize the dataset (1)

FINDING THE DIRECTIONS OF MOST VARIANCE

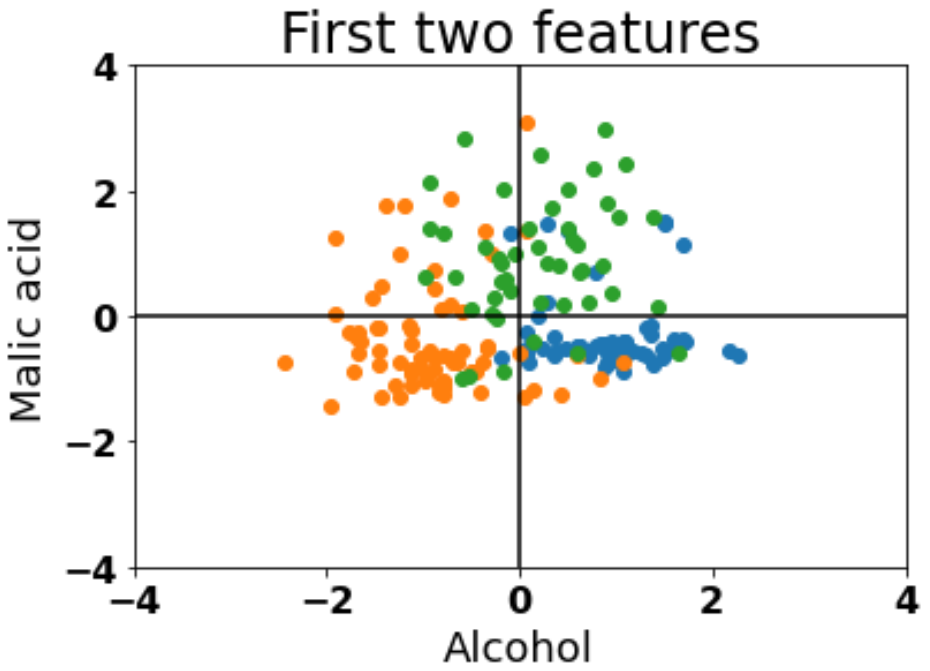
we cant compare mouse height to human height, so we scale :))
we cant compare mouse weight to human voice pitch, so we scale (standardize)

```
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.3, random_state=0)

sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.fit_transform(X_test)
X_std = sc.fit_transform(X)
```

Question: why do we need to standardize?

Compare the variance of a measurement made in millimetres to one made in kilometres!

Data



Construct the covariance matrix (2)

FINDING THE DIRECTIONS OF MOST VARIANCE

$$\sigma_{jk} = \frac{1}{n} \sum_{i=1}^{n} \left(x_j^{(i)} - \mu_j \right) \left(x_k^{(i)} - \mu_k \right)$$

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_2^2 & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_3^2 \end{bmatrix}$$

Question: what will be the size of the covariance matrix in the *Wine* dataset?

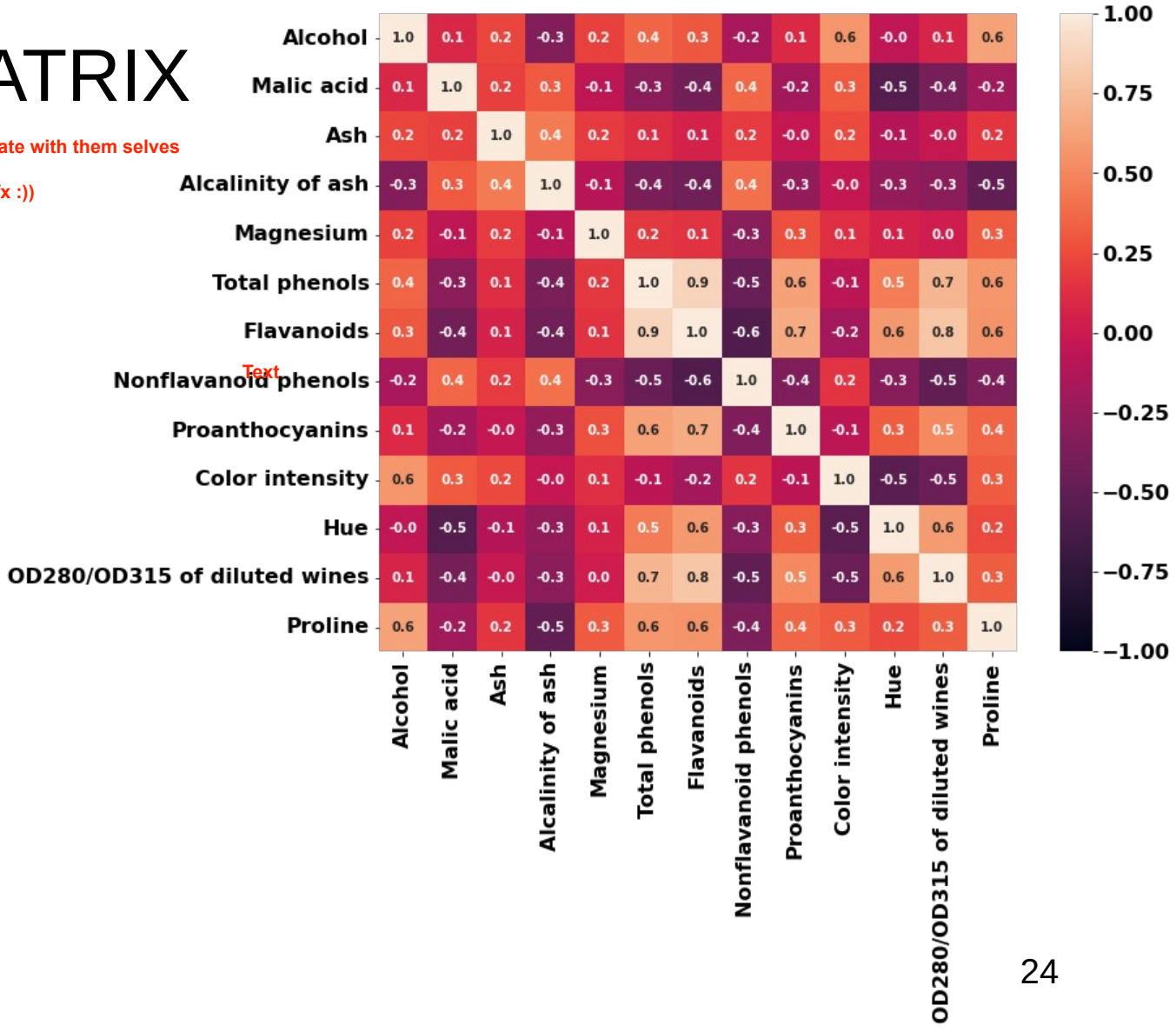
13*13 = our features*features in data

see next slide

COVARIANCE MATRIX

its covariance, they are all 1 diagnoally, because the 13 features covariate with them selves

some of the features will explain the same variance = these being 1 fx :))



Text

Find the eigenvectors and eigenvalues (3)

FINDING THE DIRECTIONS OF MOST VARIANCE

multiplying the covariance matrix with eigenvector

$$\Sigma \boldsymbol{v} = \lambda \boldsymbol{v}$$

$\boldsymbol{v}$: eigenvector
λ: eigenvalue
Σ: covariance matrix

Link to Chris Mathys's lecture from Methods II

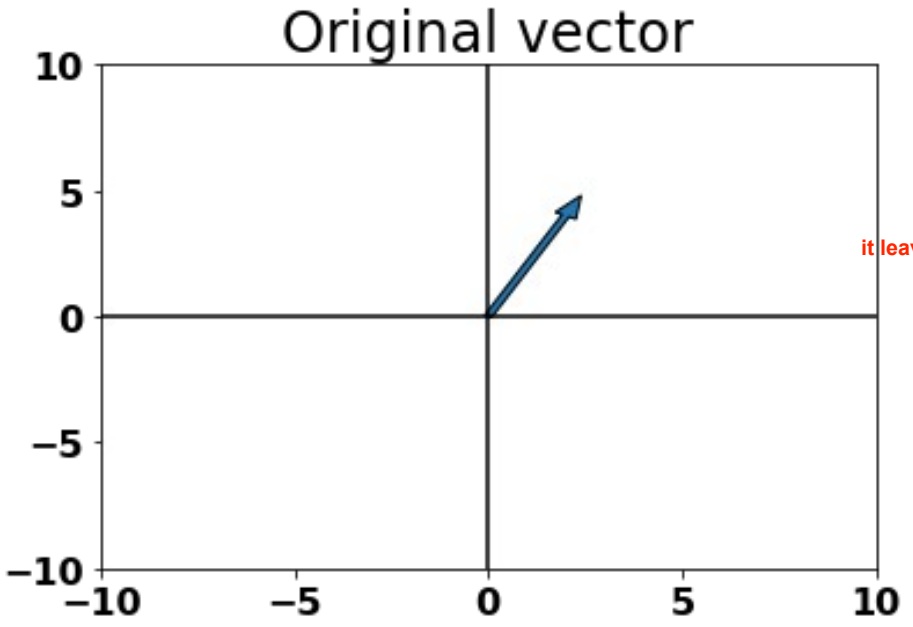
Link to 3blue1brown's video of the same matter

Remember: matrix multiplication of a vector can be seen as a **transformation** of the vector

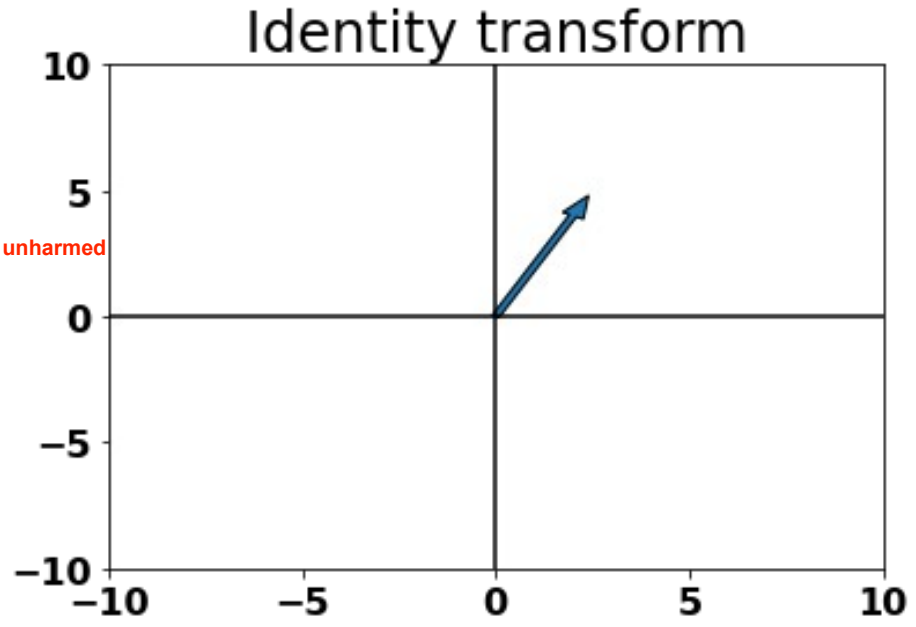
$$T(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$$

$\boldsymbol{A}$: transformation matrix
$\boldsymbol{x}$: column vector
$\boldsymbol{T}$: transformed column vector

Identity Transformation



it leaves it unharmed

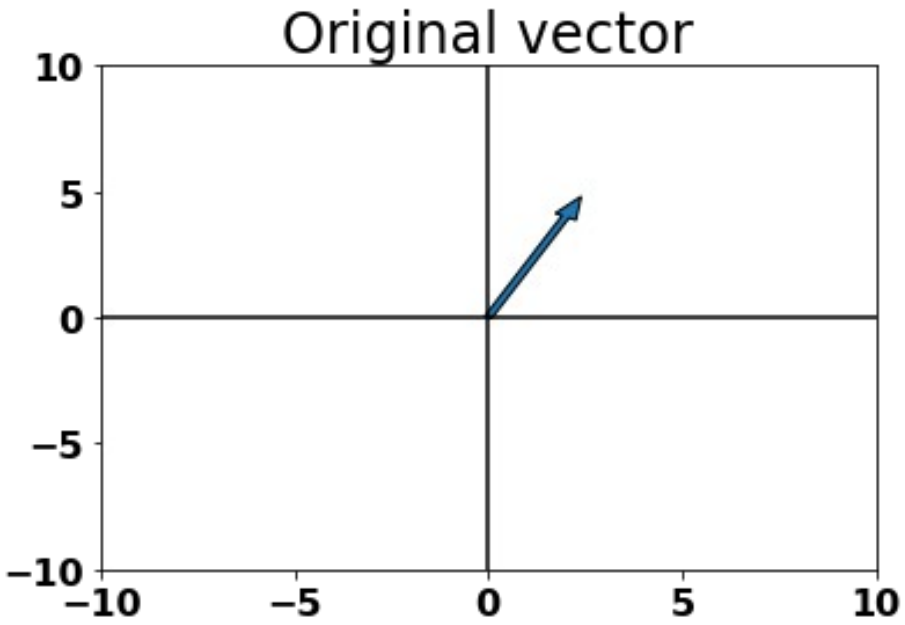


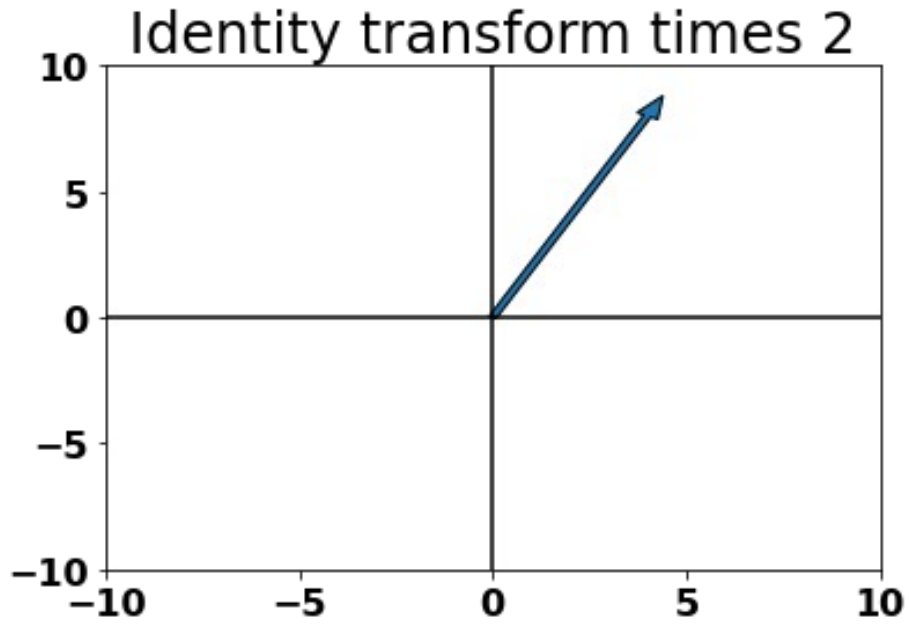

```
## plot vector (2, 4)
v = (2, 4)
plot_vector(v, 'Original vector')
```

```
A = np.identity(2)
print('A = \n' + str(A))
plot_vector(A @ v, 'Identity transform')
```

```
A =
[[1. 0.]
 [0. 1.]]
```

Scaling Transformation

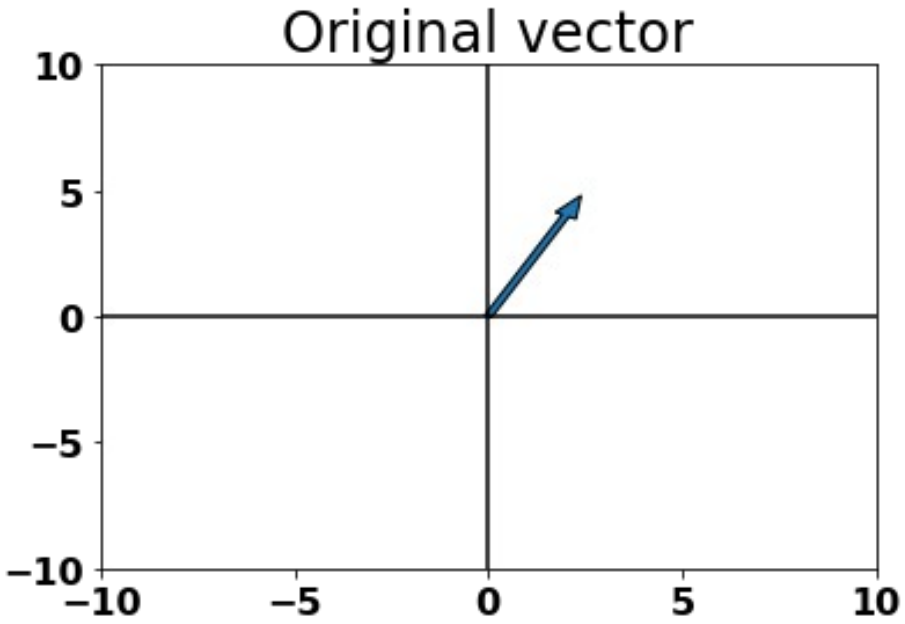


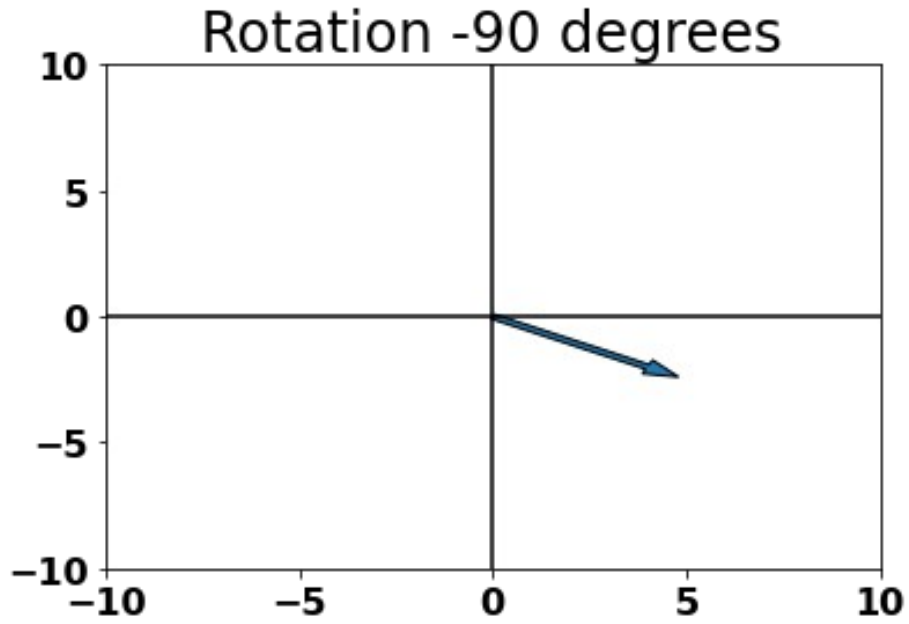



```
## transform with identity matrix times 2
A = 2 * np.identity(2)
print('A = \n' + str(A))
plot_vector(A @ v, 'Identity transform times 2')
```

```
A =
[[2. 0.]
 [0. 2.]]
```

Rotation Transformation

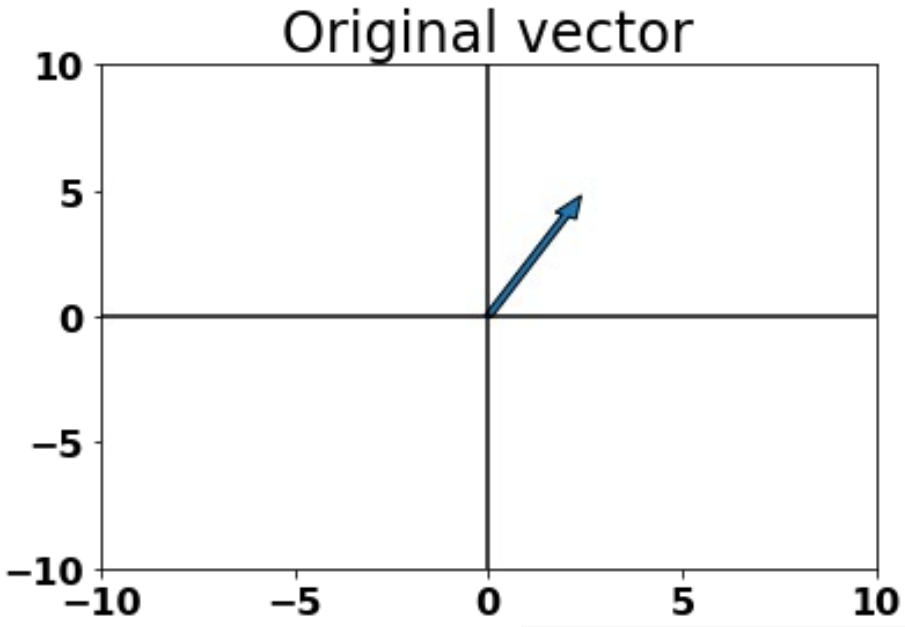


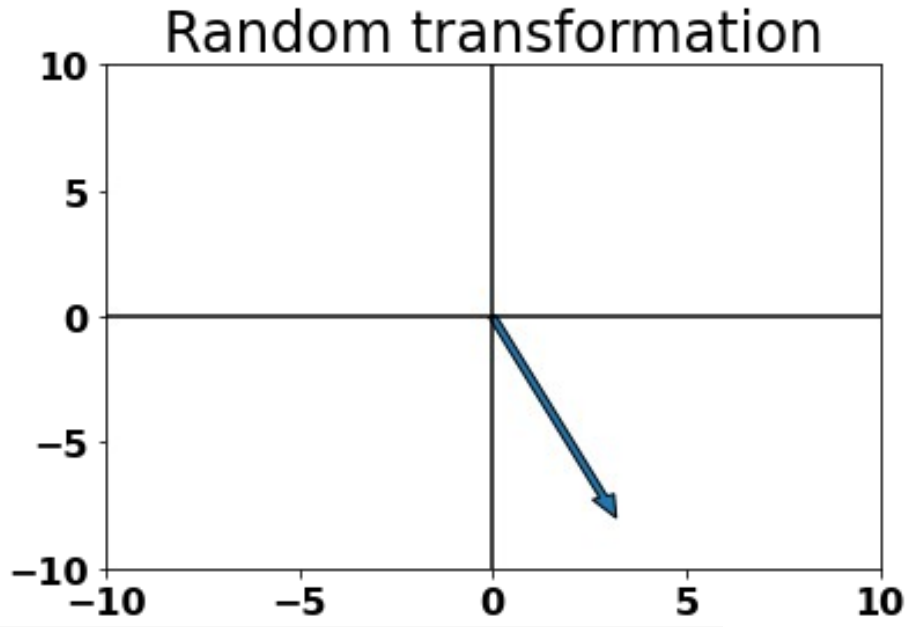



```
A = np.array([[0, 1], [-1, 0]])
print('A = \n' + str(A))
plot_vector(A @ v, 'Rotation -90 degrees')
```

```
A =
[[ 0  1]
 [-1  0]]
```

Random Transformation






```
np.random.seed(1)
A = np.reshape(np.random.uniform(low=-2, high=2, size=4), newshape=(2, 2))
print('A = \n' + str(A))
plot_vector(A @ v, 'Random transformation')
```

```
A =
[[-0.33191198  0.88129797]
 [-1.9995425  -0.79066971]]
```

In [142]:
```
## eigenvalues and eigenvectors
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
print(eigen_vals.shape)
print(eigen_vecs.shape)
```

```
(13,)
(13, 13)
```

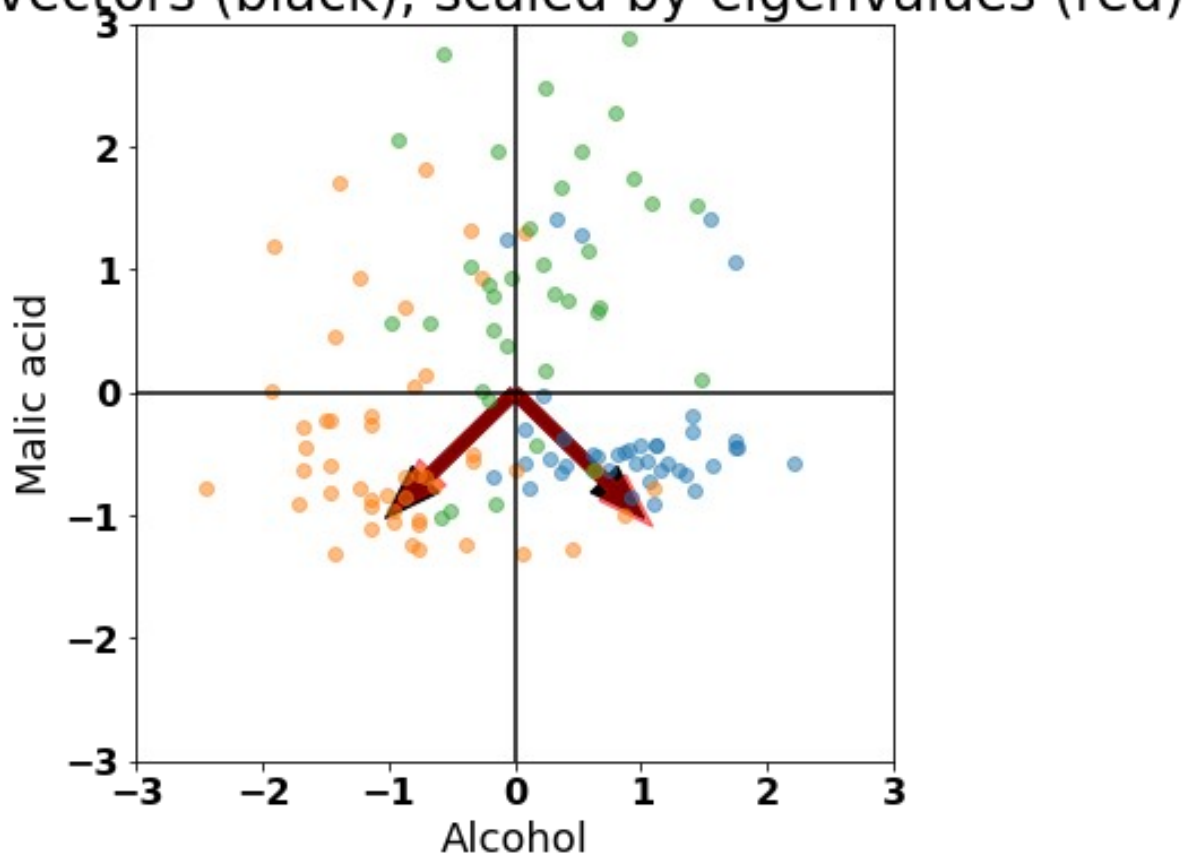
we will have as many eigen values as dimension :))

In [17]:
```
# checking whether the equation holds
evec_0 = eigen_vecs[:, 0]
eval_0 = eigen_vals[0]

mat_trans = cov_mat @ evec_0 ## A times x
scalar_trans = eval_0 * evec_0 # lambda times x
print(mat_trans)
print(scalar_trans)

print(np.isclose(mat_trans, scalar_trans)) ## instead of using "==", due to rounding error
```

```
[ 0.7176924  -1.18513985 -0.14644842 -1.24846827  0.5909797   1.90479358
  2.07074217 -1.49875647  1.49568723 -0.48283127  1.46928422  1.80140436
  1.43147537]
[ 0.7176924  -1.18513985 -0.14644842 -1.24846827  0.5909797   1.90479358
  2.07074217 -1.49875647  1.49568723 -0.48283127  1.46928422  1.80140436
  1.43147537]
[ True  True  True  True  True  True  True  True  True  True  True  True
  True]
```

$$\Sigma \boldsymbol{v} = \lambda \boldsymbol{v}$$

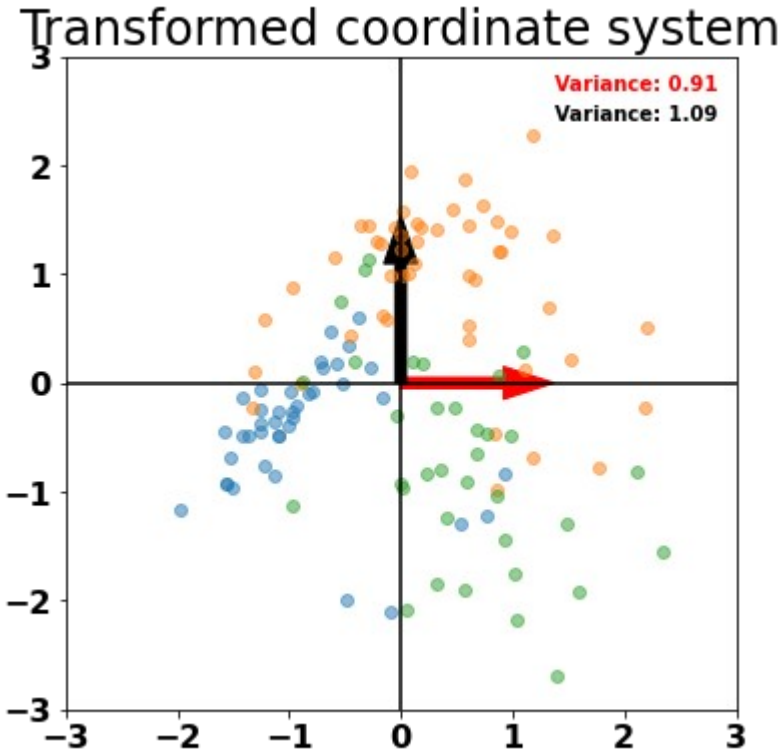
```
Eigenvalues:
[0.92015307 1.09610709]

Eigenvectors:
 [[-0.70710678 -0.70710678]
 [ 0.70710678 -0.70710678]]

Covariance matrix:
[[1.00813008 0.08797701]
 [0.08797701 1.00813008]]
```



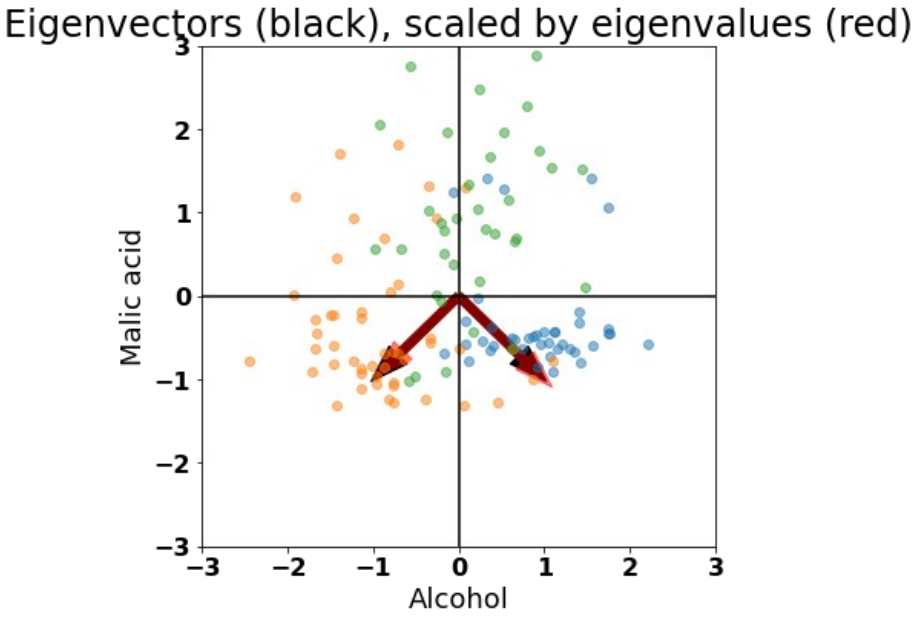
Applying a transformation

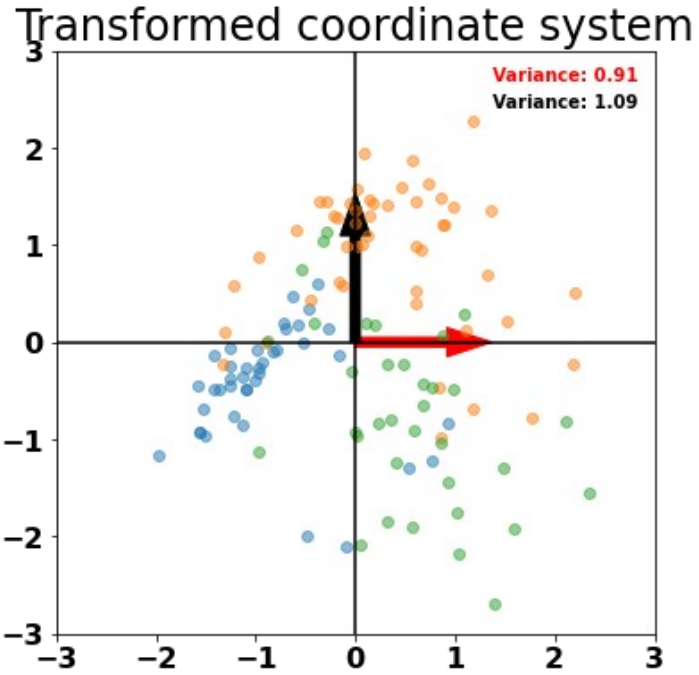
how to rotate something in 2d :)

```
radian = np.arctan(reduced_eigen_vecs[1, 1] / reduced_eigen_vecs[1, 0])


A = np.array([
                [np.cos(radian), np.sin(radian)],
                [-np.sin(radian), np.cos(radian)]
            ])
```



For comparison





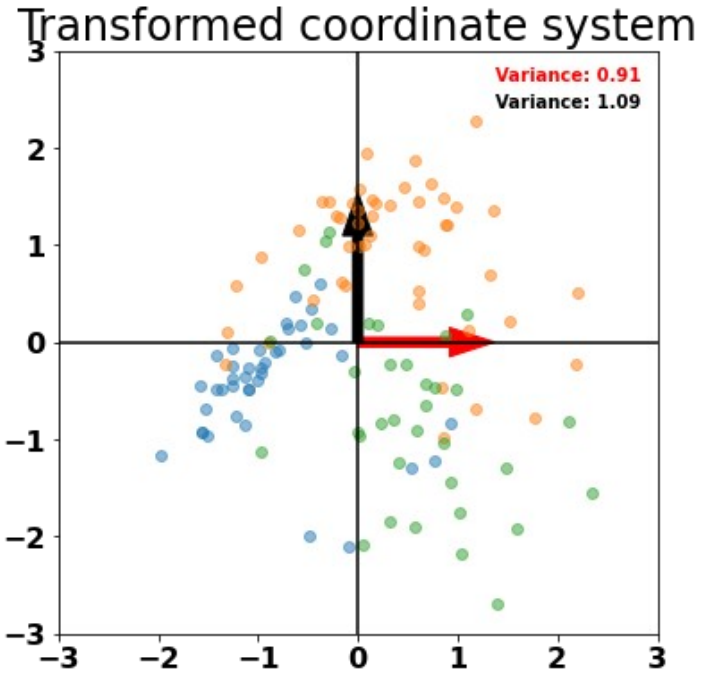
In the case where $k=d$...

4) Select k eigenvectors that correspond to the k largest eigenvalues where k is the dimensionality of the new feature subspace ($k \leq d$)

5) Construct a projection matrix $\boldsymbol{W}$ from the “top” k eigenvectors

6) Transform the d-dimensional input dataset $\boldsymbol{x}$ using the projection matrix $\boldsymbol{W}$ to obtain the new k-dimensional feature subspace

... applying steps 4, 5 & 6 result in:



this is **not** feature reduction, however...

“reducing” to a new subspace (*k*=*d*) (4)

Eigenvalues for the **full** covariance matrix

```
## going back to the full feature matrix
print(eigen_vals)
```

```
[4.8923083  2.46635032 1.42809973 1.01233462 0.84906459 0.60181514
 0.52251546 0.08414846 0.33051429 0.29595018 0.16831254 0.21432212
 0.2399553 ]
```

Variance explained ratio

$$\frac{\lambda_j}{\sum_{j=1}^{d} \lambda_j}$$

$$\lambda_1 = max(\lambda_j)$$
$$\lambda_7 = min(\lambda_j)$$

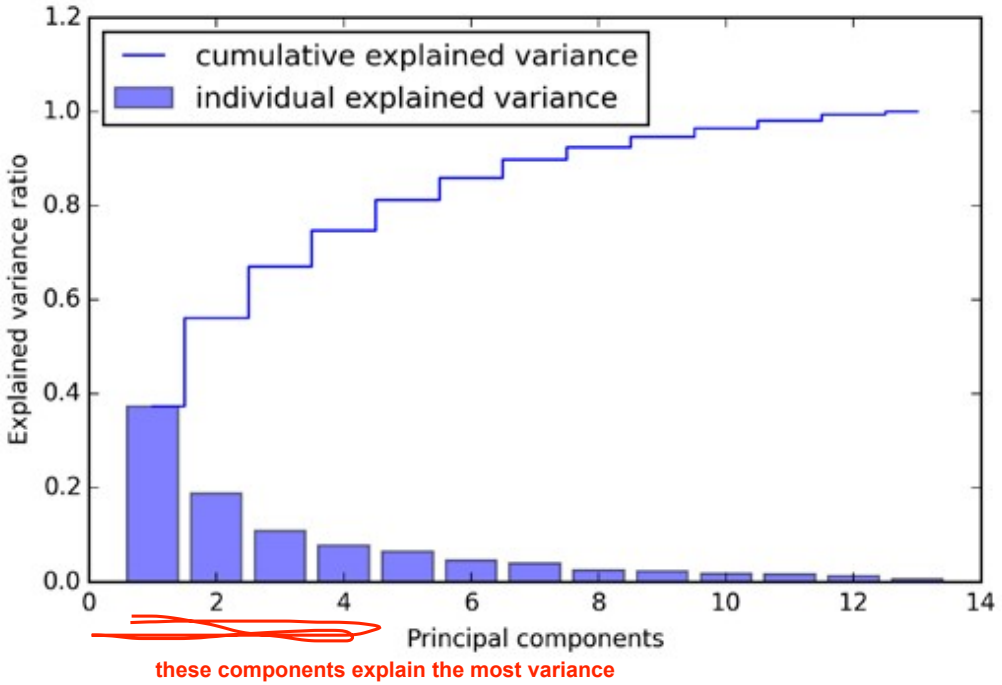
$$\frac{\lambda_1}{\sum_{j=1}^{d} \lambda_j} = 37\%$$

1 gamma

$$\frac{\lambda_7}{\sum_{j=1}^{d} \lambda_j} = 0.64\%$$

7 gamma

Sorted explained variance



(p. 132: Raschka, 2015)

Setting $k = 2$

Construct a projection matrix ***W*** from the "top" *k* eigenvectors (5)

eigenvalues in an eigen vector made into a W matrix

```
print('Weight matrix:\n', W)
```

```
Weight matrix:
 [[ 0.14669811  0.50417079]
 [-0.24224554  0.24216889]
 [-0.02993442  0.28698484]
 [-0.25519002 -0.06468718]
 [ 0.12079772  0.22995385]
 [ 0.38934455  0.09363991]
 [ 0.42326486  0.01088622]
 [-0.30634956  0.01870216]
 [ 0.30572219  0.03040352]
 [-0.09869191  0.54527081]
 [ 0.30032535 -0.27924322]
 [ 0.36821154 -0.174365  ]
 [ 0.29259713  0.36315461]]
```

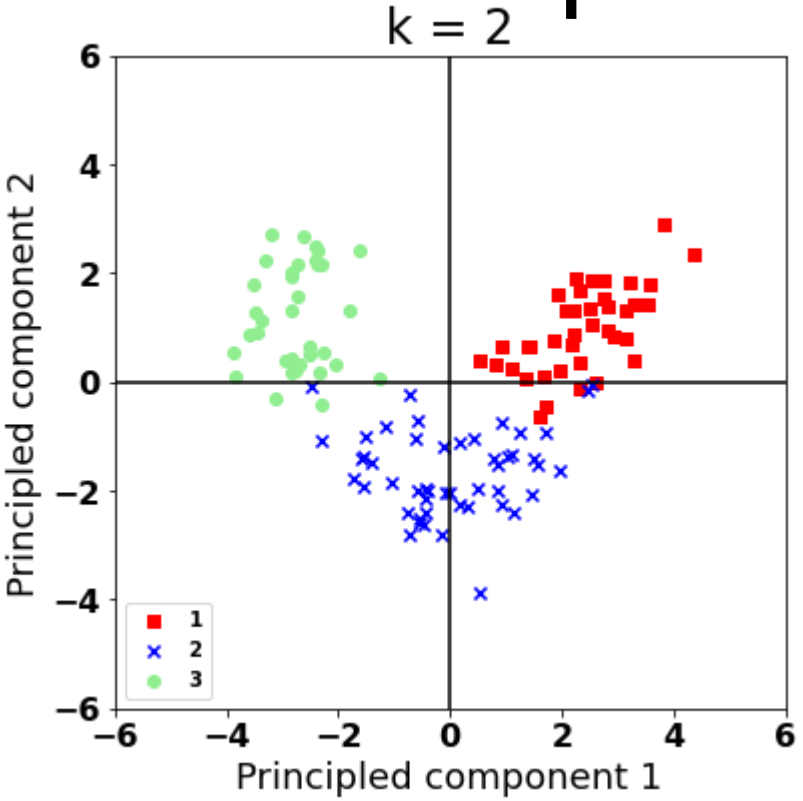
Transform the *d*-dimensional input dataset ***X*** using the projection matrix ***W*** to obtain the new *k*-dimensional feature subspace (6)

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_d], \boldsymbol{x} \in \mathbb{R}^{\boldsymbol{d}}$$
$$\downarrow \boldsymbol{xW}, \boldsymbol{W} \in \mathbb{R}^{d \times k}$$
$$\boldsymbol{z} = [z_1, z_2, \ldots, z_k], \boldsymbol{z} \in \mathbb{R}^{\boldsymbol{k}}$$

$$\boldsymbol{Z} = \boldsymbol{XW}$$

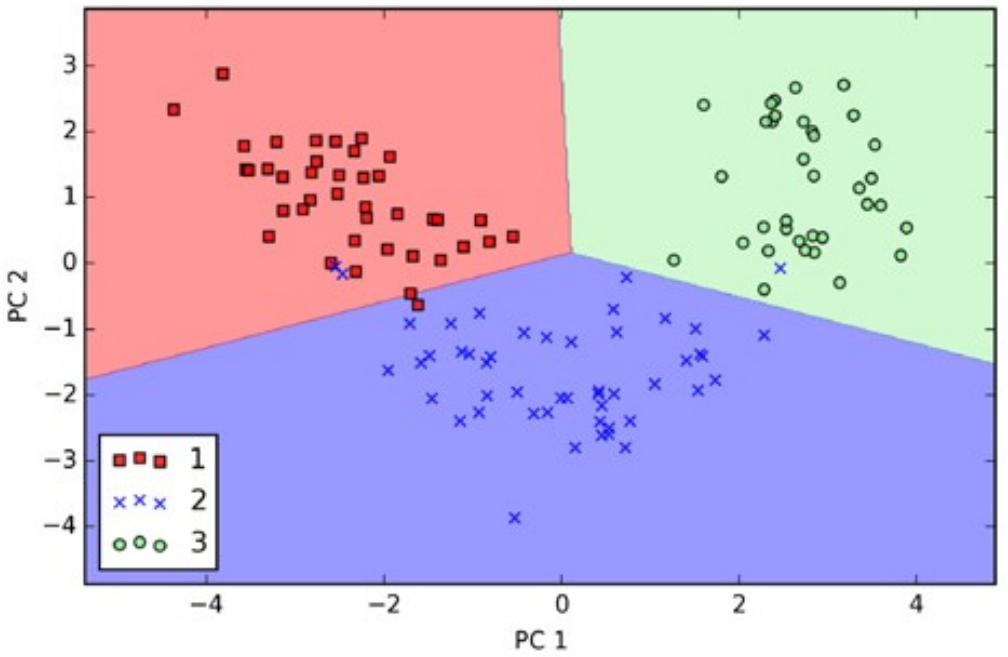
Reduced space

NOW its easier to calssify



We can now do logistic regression in this space

Logistic regression on reduced space (k = 2)

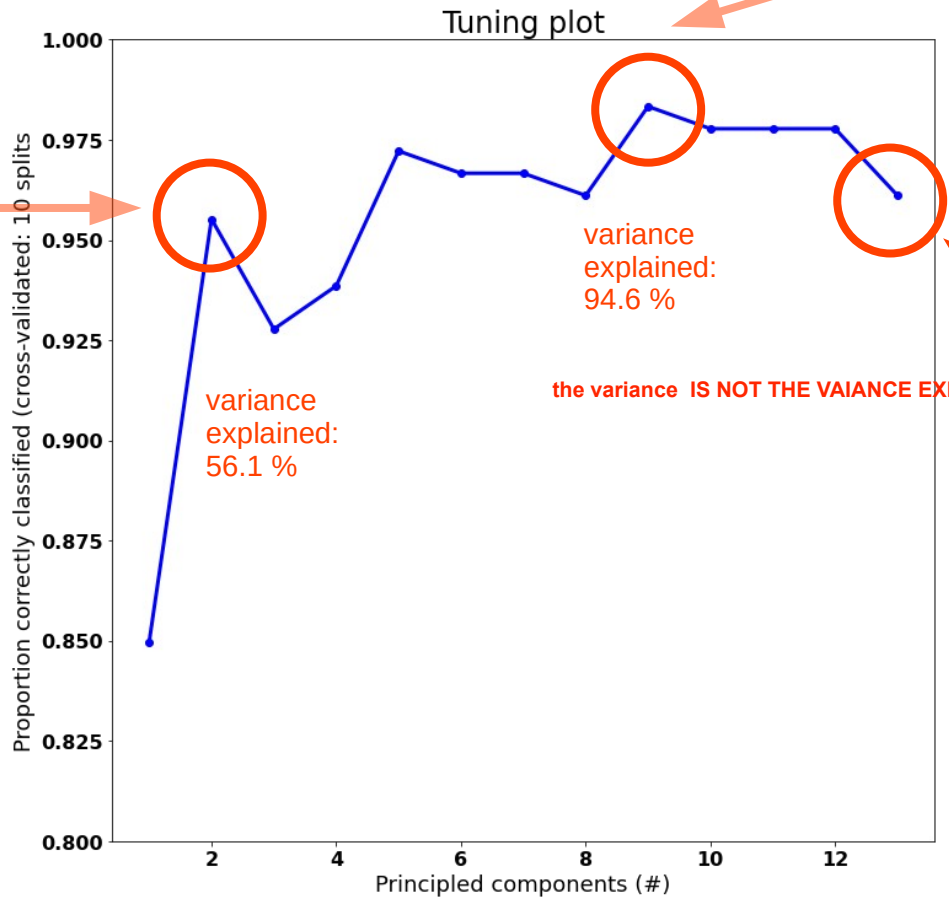


(p. 136: Raschka, 2015)

how do we choose k?
we want a k that corresponds to the real world

peak
$k = 9$

$k = 2$



variance explained: 56.1 %

variance explained: 94.6 %

What does this analysis correspond to?

the variance IS NOT THE VAIANCE EXPLAINED BY THE MODEL, but variance ratio

1 features… 2 3, 4 until 13 (features in wine df)

Did you learn?

Dimensionality reduction

1) Learning how we can extract the features that explain the most variance **we learn nothing about WHICH features**

2) Understanding how that can improve classification

3) Get acquainted with the concept of a eigenvector

(OPTIONAL)

Live coding

WEEK_09.ipynb

(OPTIONAL)

Live coding

NUMPY.ipynb

References

- Abe, S., 2010. Support Vector Machines for Pattern Classification. Springer, London.
- Raschka, S., 2015. Python Machine Learning. Packt Publishing Ltd.