FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

Support Vector Machine (SVM)

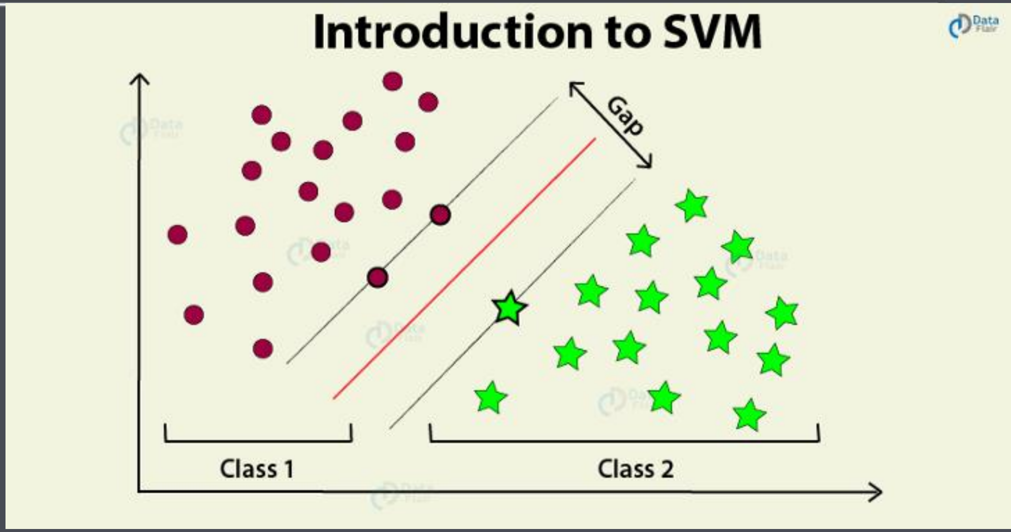
LIAD MAGEN

Support Vector Machine

The undisputed classification winner

Support Vector Machine (SVM)

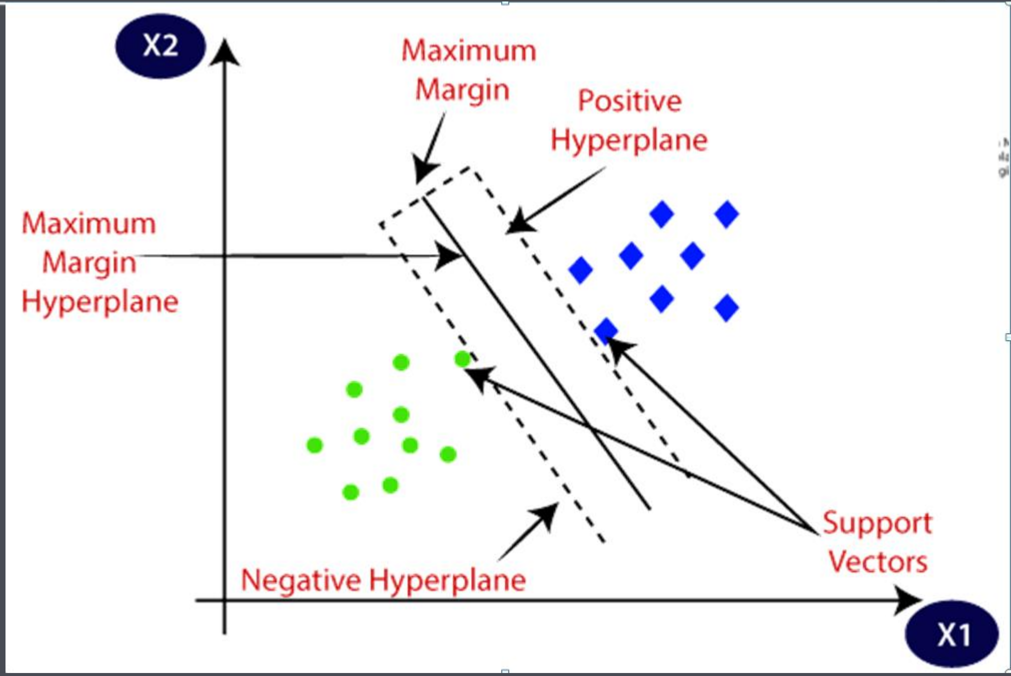
- Supervised Learning method
- Classification or Regression
- Separates the data linearly by finding the best *decision boundary*
- The Decision Boundary is called a **Hyperplane** in SVM
- Can be linear… and non-linear (!) (we get to this in a sec)



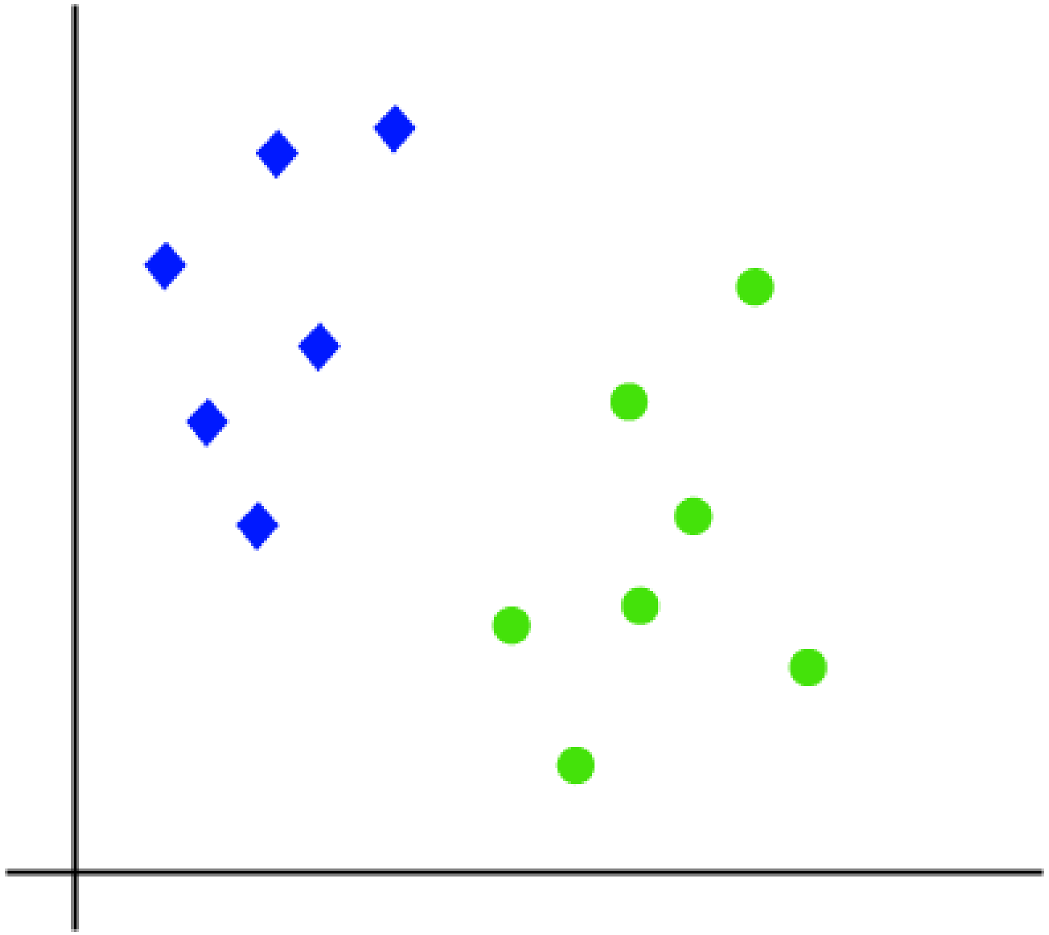
FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

Support Vectors and Margins

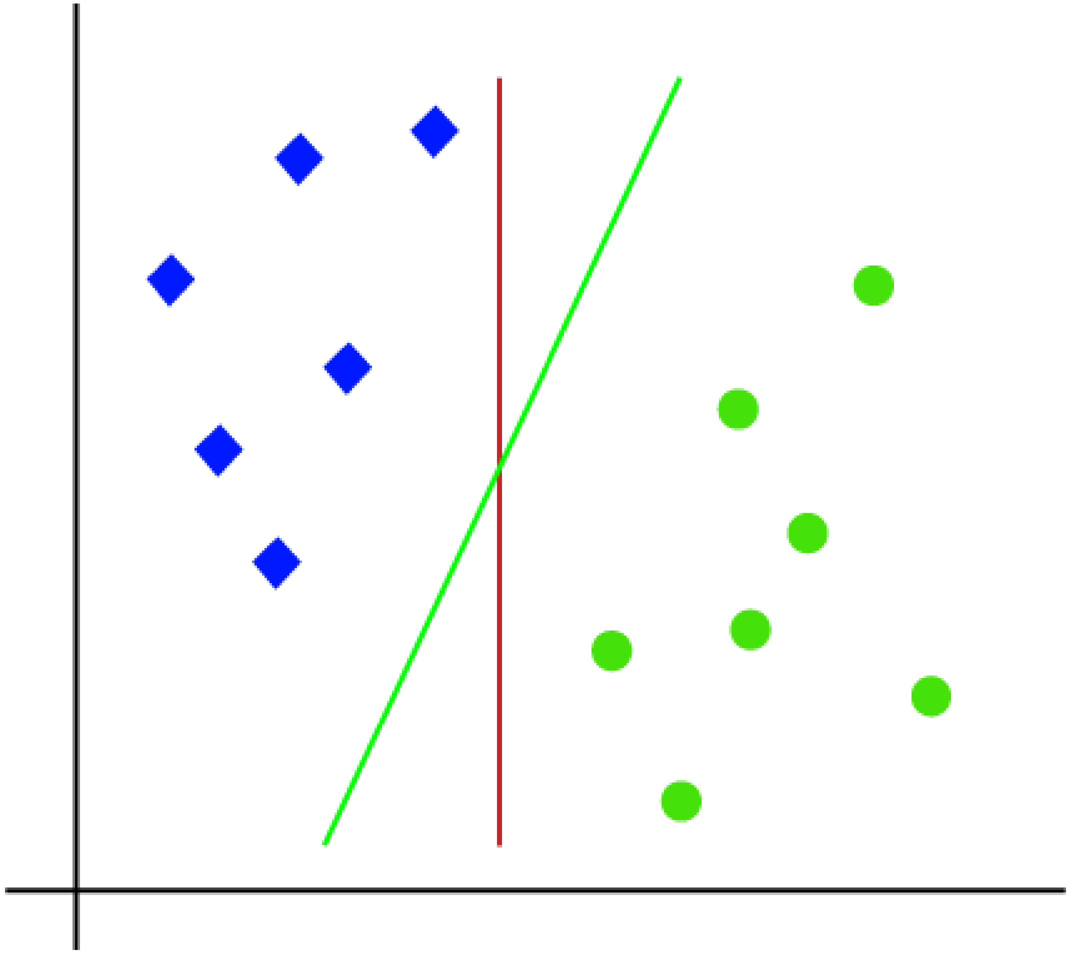
> The hyperplanes are in the $D - 1$ dimension:
  > 2 features – straight line (1D)
  > 3 features – 2D plane
> Hyperplanes have a **maximum margin** from the points
> SVM is looking to **maximize** this margin
> The nearest points (or **vectors** – array of points) to the margins are called the **support vectors**



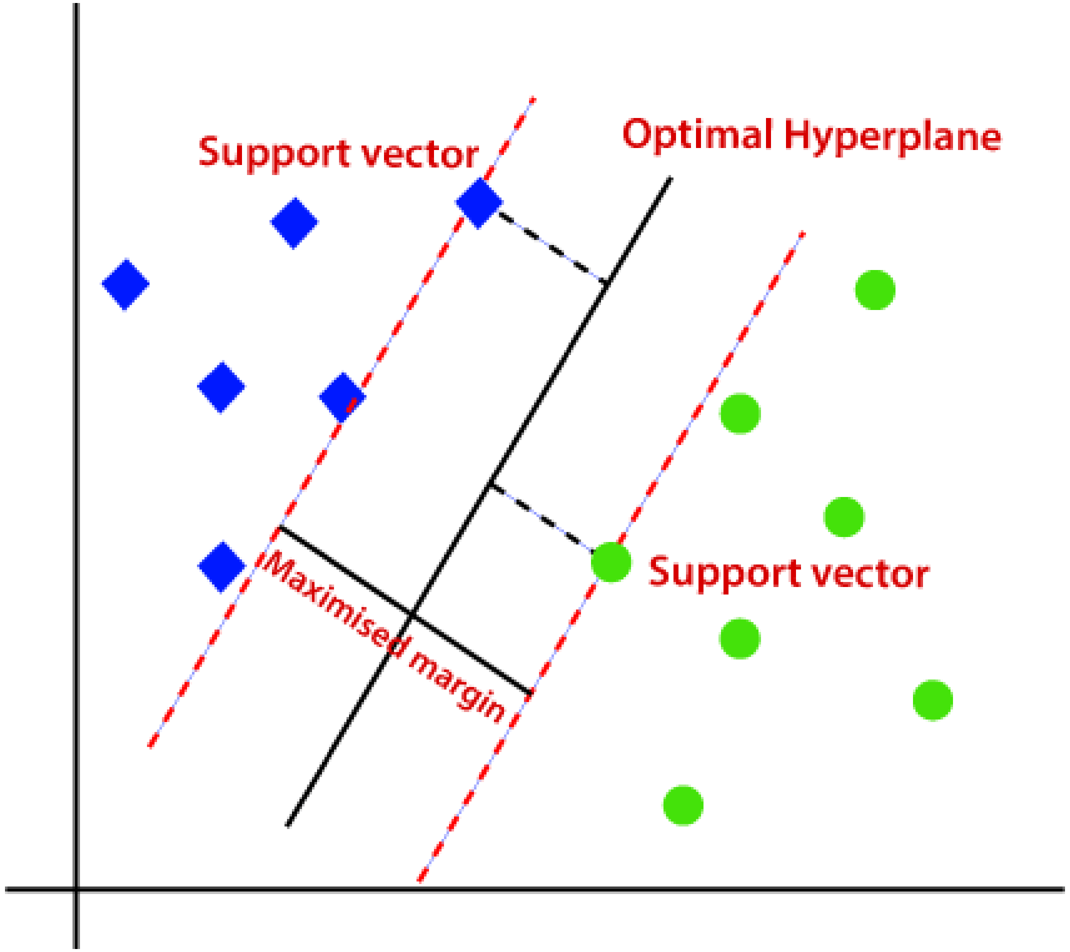
Example 1



Example 1



Example 1

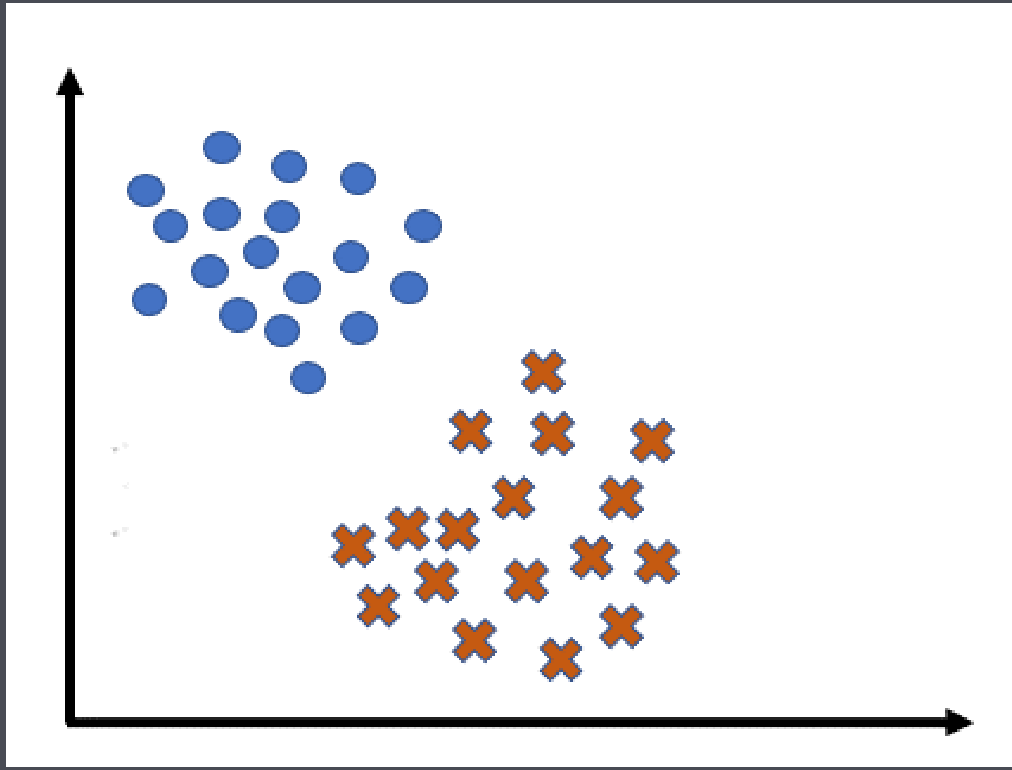

Support vector
Optimal Hyperplane
Maximised margin
Support vector

FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

SVM Margins

> Bigger is better!

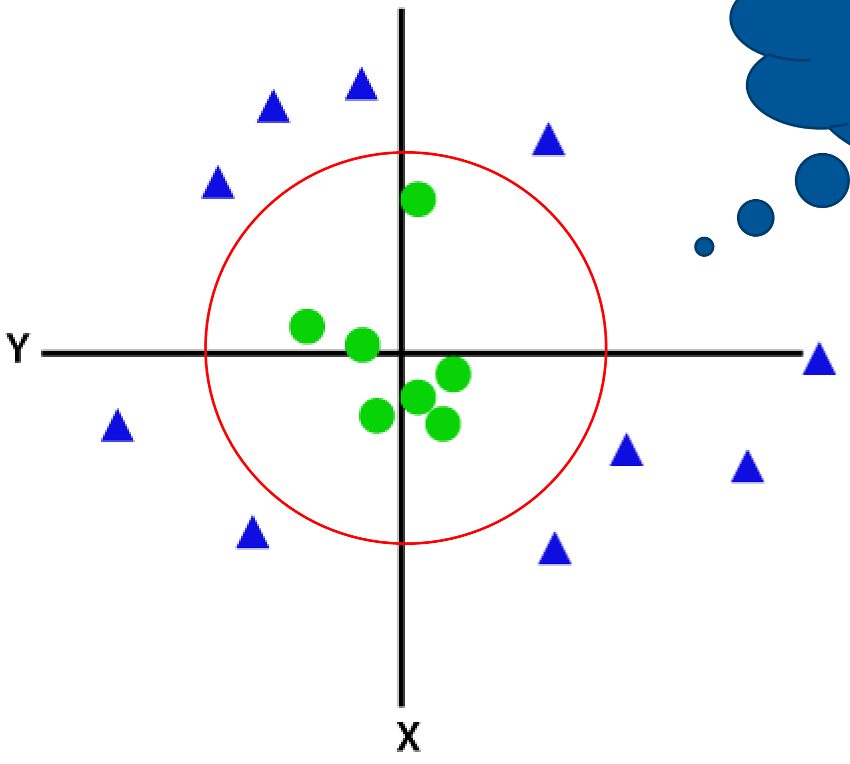
> Notice: the objective is different: Instead of minimizing distances, we maximize the margins



Big deal when the data is linear...

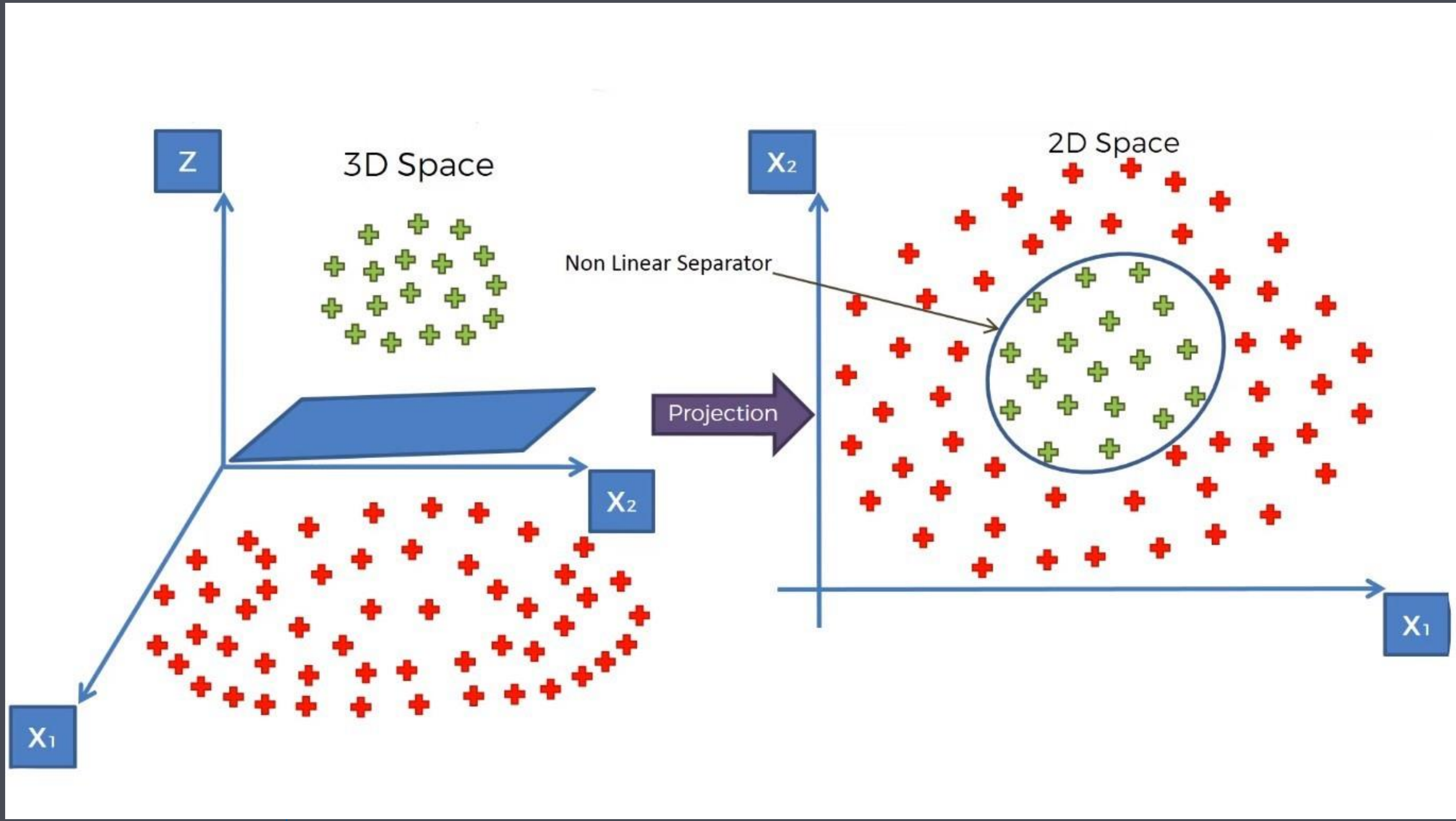
Linear regression does the same, so why do we need this?!

Non-Linearly Separable data



BTW, what is the formula of a circle?

We can add a dimension!

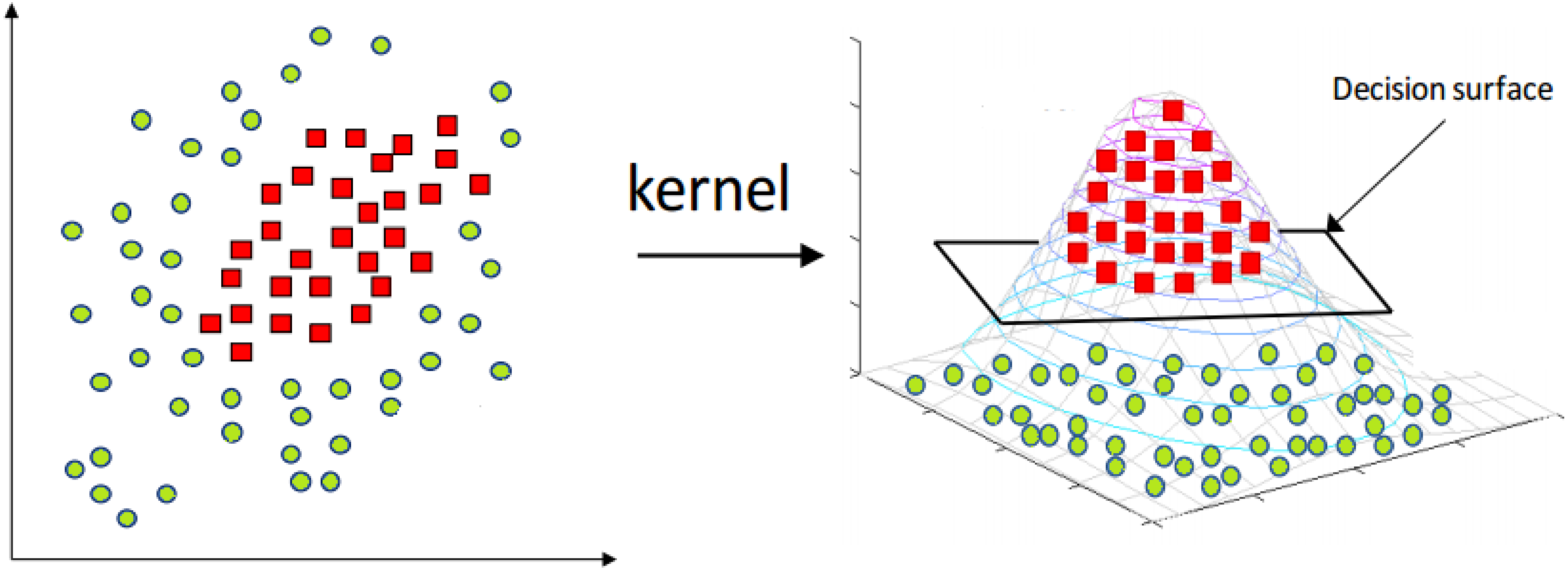


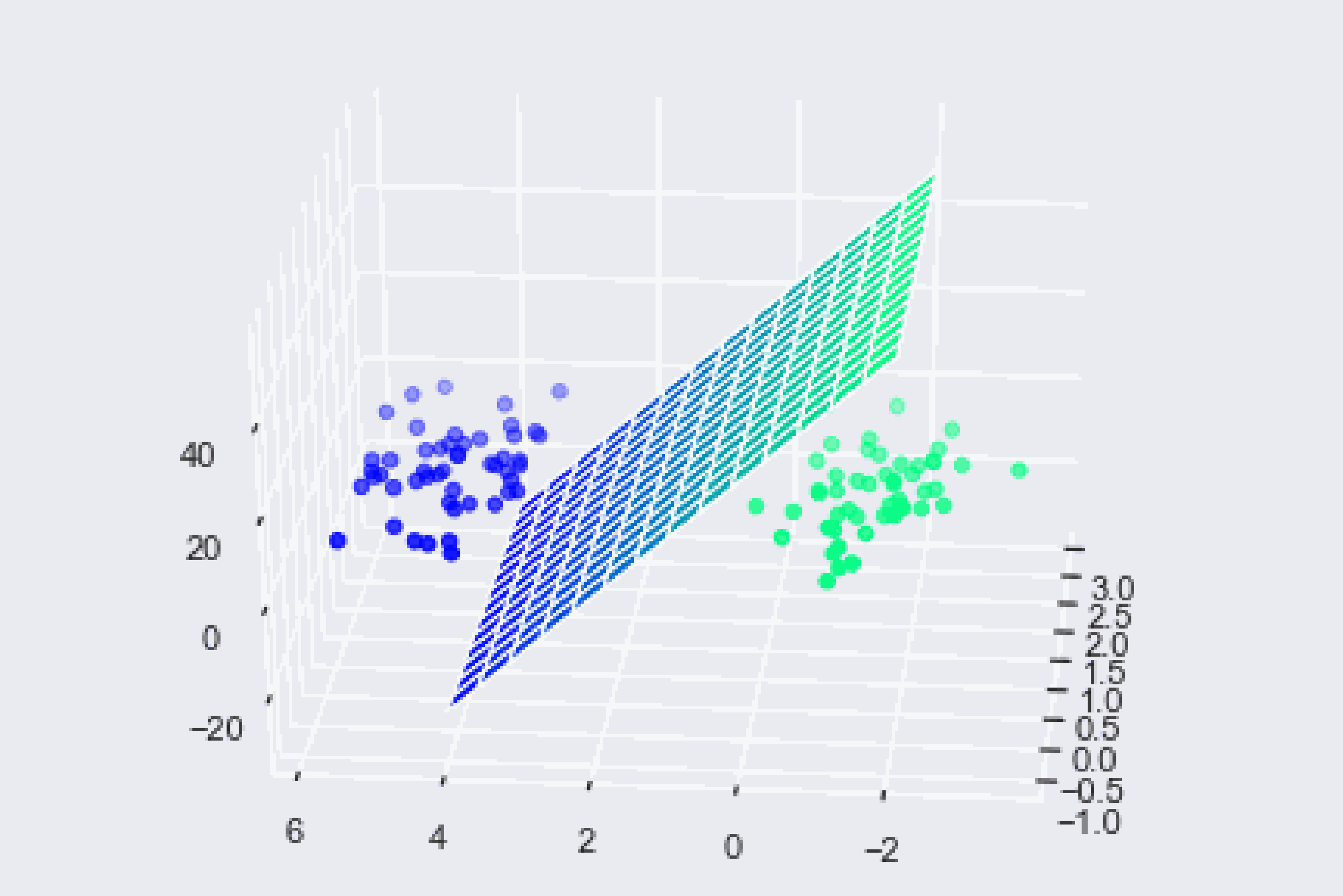
The new dimension – z can be created with some formula, e.g.:

$$z = x_1^2 + x_2^2$$

This formula is called a **kernel**

A kernel is a transformation function.





In 2D it finds a line.

In 3D it finds a hyperplane.

How can we possibly add a dimension?!

- Dimensions aka **z-space**
- Kernels
  - A **hyperparameter** of the classifier

Some Kernel Examples:

- Linear Kernel: $k(x, y) = sum(x, y)$
- Polynomial Kernel: $k(x, y) = (x, y + 1)^d$
- Gaussian Radial Basis Function (RBF): $k(x, y) = \exp(-\gamma * \| x - y \|^2)$
- Sigmoid Kernel: $k(x, y) = \tanh(\gamma x^T y + r)$

FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

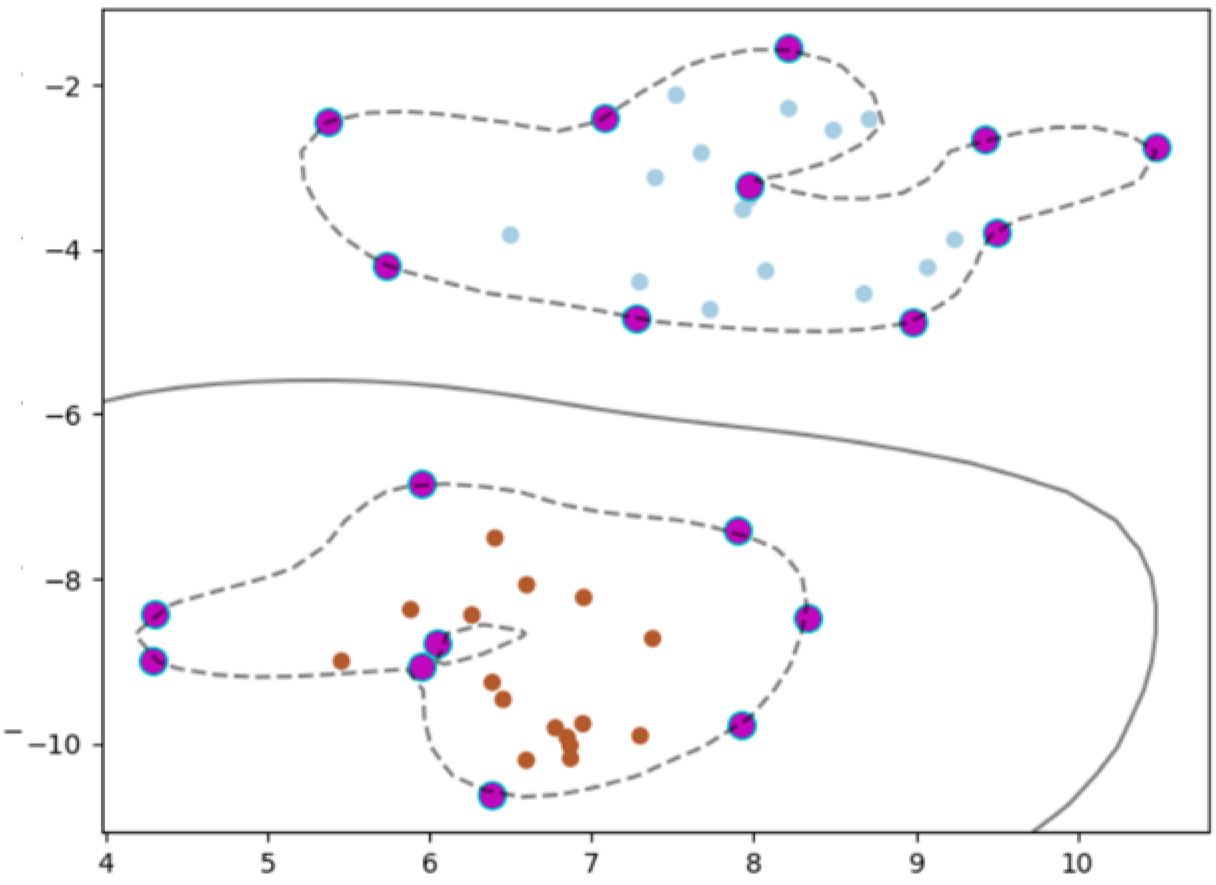
Python code example

```
from sklearn.svm import SVC  # Support Vector Classifier
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
```

Other kernel options are:
“poly”
“rbf”
“sigmoid”
…

Elaborated Code Example

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.datasets import make_blobs


# we create 40 separable points
X, y = make_blobs(n_samples=50, centers=2, random_state=6)

# fit the model, don't regularize for illustration purposes
clf = svm.SVC(kernel='rbf',    C=1000)
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

# plot the decision function
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
# plot support vectors
ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
           linewidth=1, facecolors='m', edgecolors='c')
plt.show()
```



SVM - Summary

- Up until NN – this was the most efficient classifier (!)
- It can solve many real-world problems
- Works well with high dimensionality
- Memory Efficient: It only uses the data points which are close to the decision boundary.
- Can give an *intuition* about the probability or *certainty* of data points (how close they are to the boundary vector)

- Bonus: It can be used in an *unsupervised* manner for anomaly classification (e.g., analyzing log-texts) – called "one-class SVM"