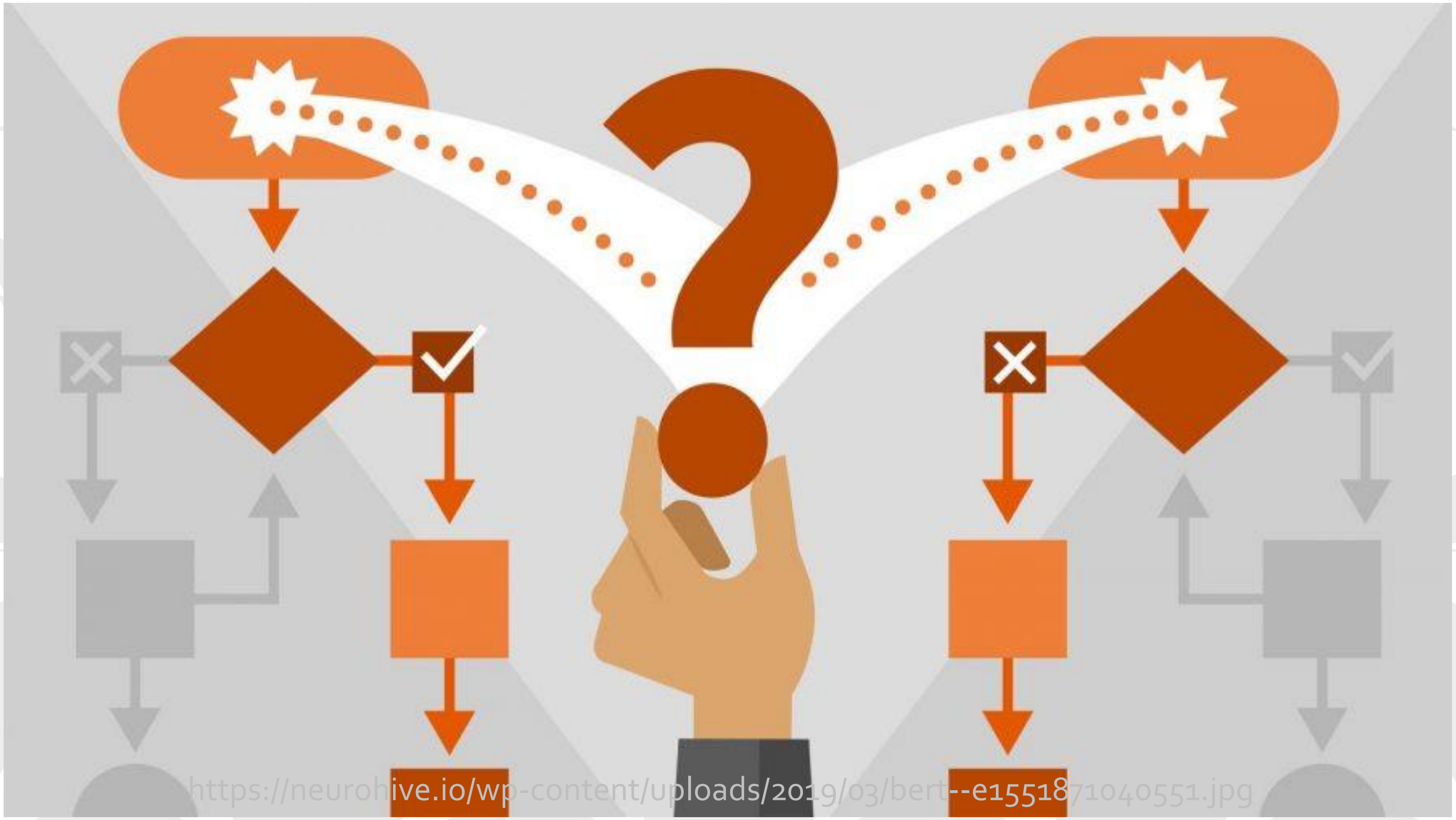
SUPPORT VECTOR MACHINES

Angewandte Informatik – Data Science – Iryna Trygub

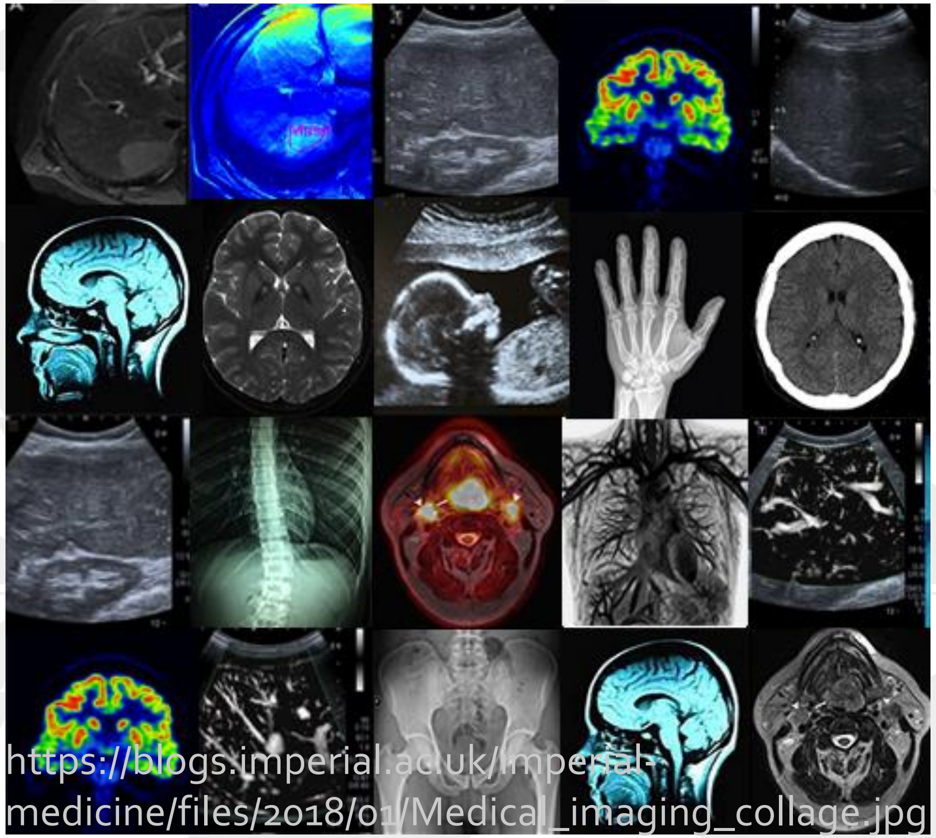
Klassifikationsverfahren

https://en.wikipedia.org/wiki/Sky_and_Water_I#/media/File:Sky_and_Water_I.jpg

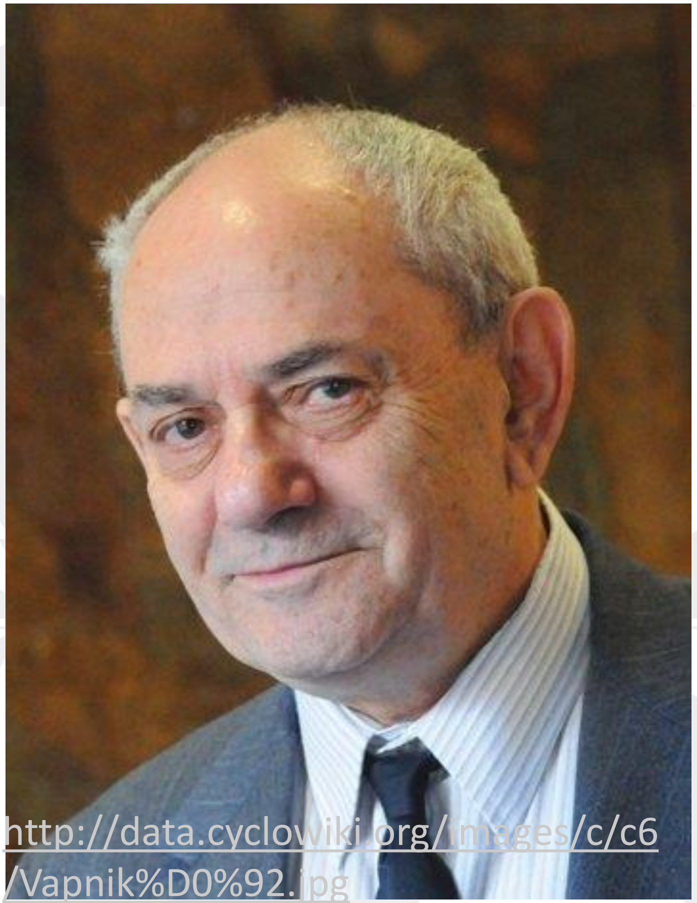

https://neurohive.io/wp-content/uploads/2019/03/bert--e1551871040551.jpg

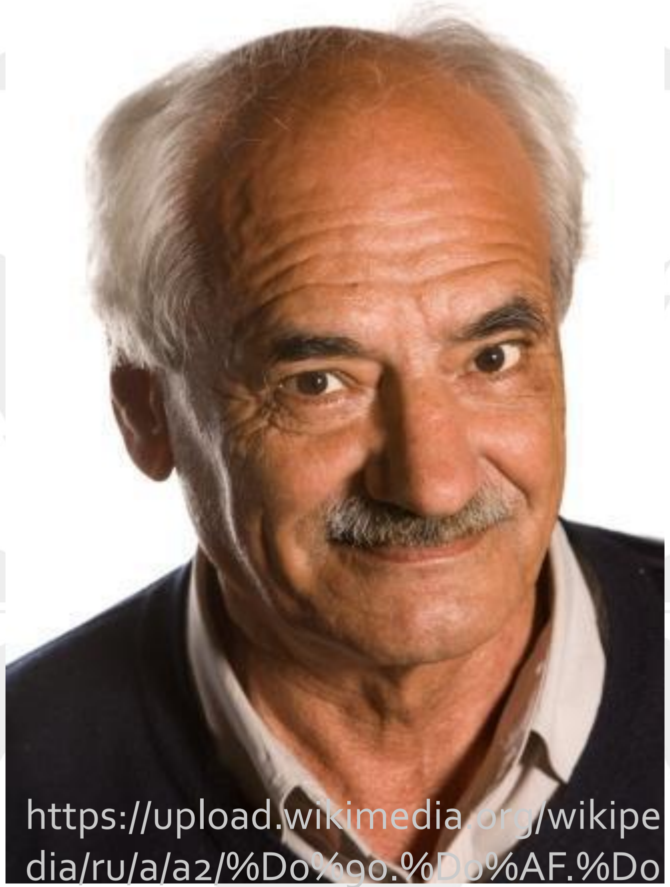
Objekte sind Vektoren im n-dimensionalen linearen Raum!

https://miro.medium.com/max/1400/1*UDi7KpyFX8gwV1k7aeMS-g.jpeg



https://blogs.imperial.ac.uk/imperial-medicine/files/2018/01/Medical_imaging_collage.jpg

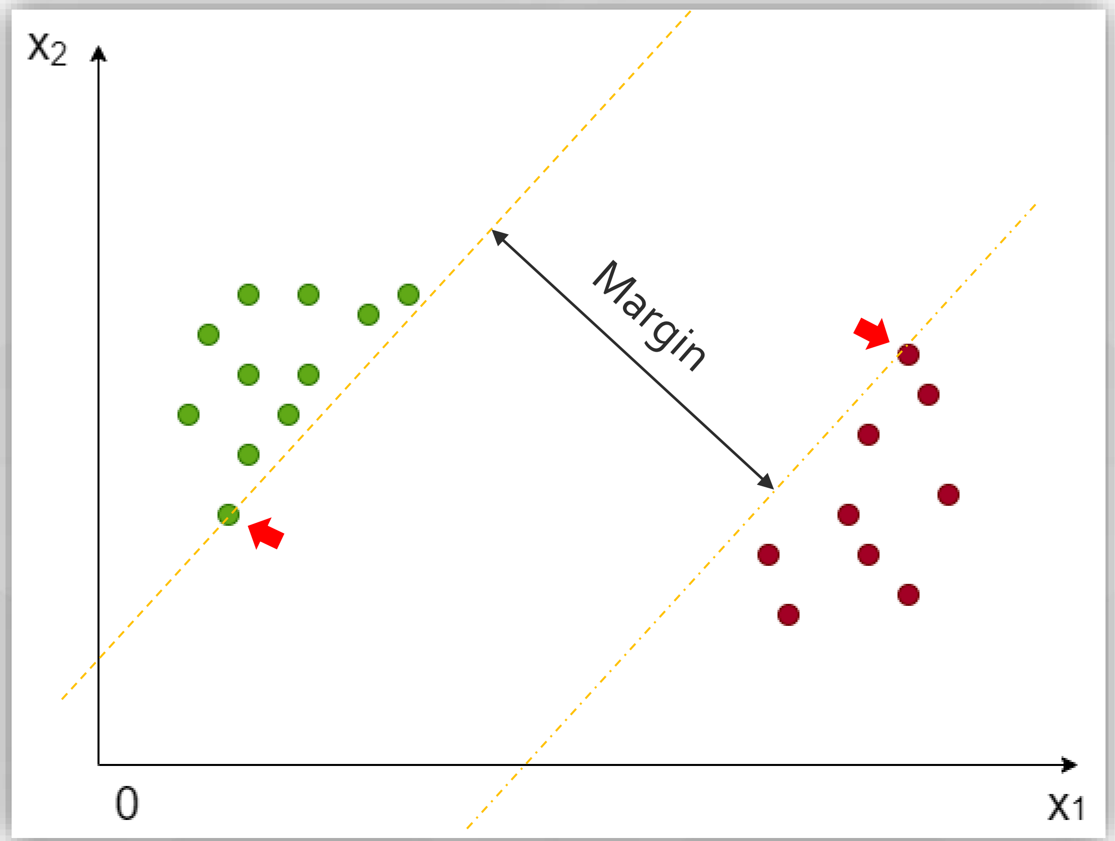

http://data.cyclowiki.org/images/c/c6/Vapnik%D0%92.jpg


https://upload.wikimedia.org/wikipedia/ru/a/a2/%D0%90.%D0%AF.%D0%A7%D0%B5%D1%80%D0%B2%D0%BE%D0%BD%D0%B5%D0%BD%D0%BA%D0%B8%D1%81.jpg

R^n

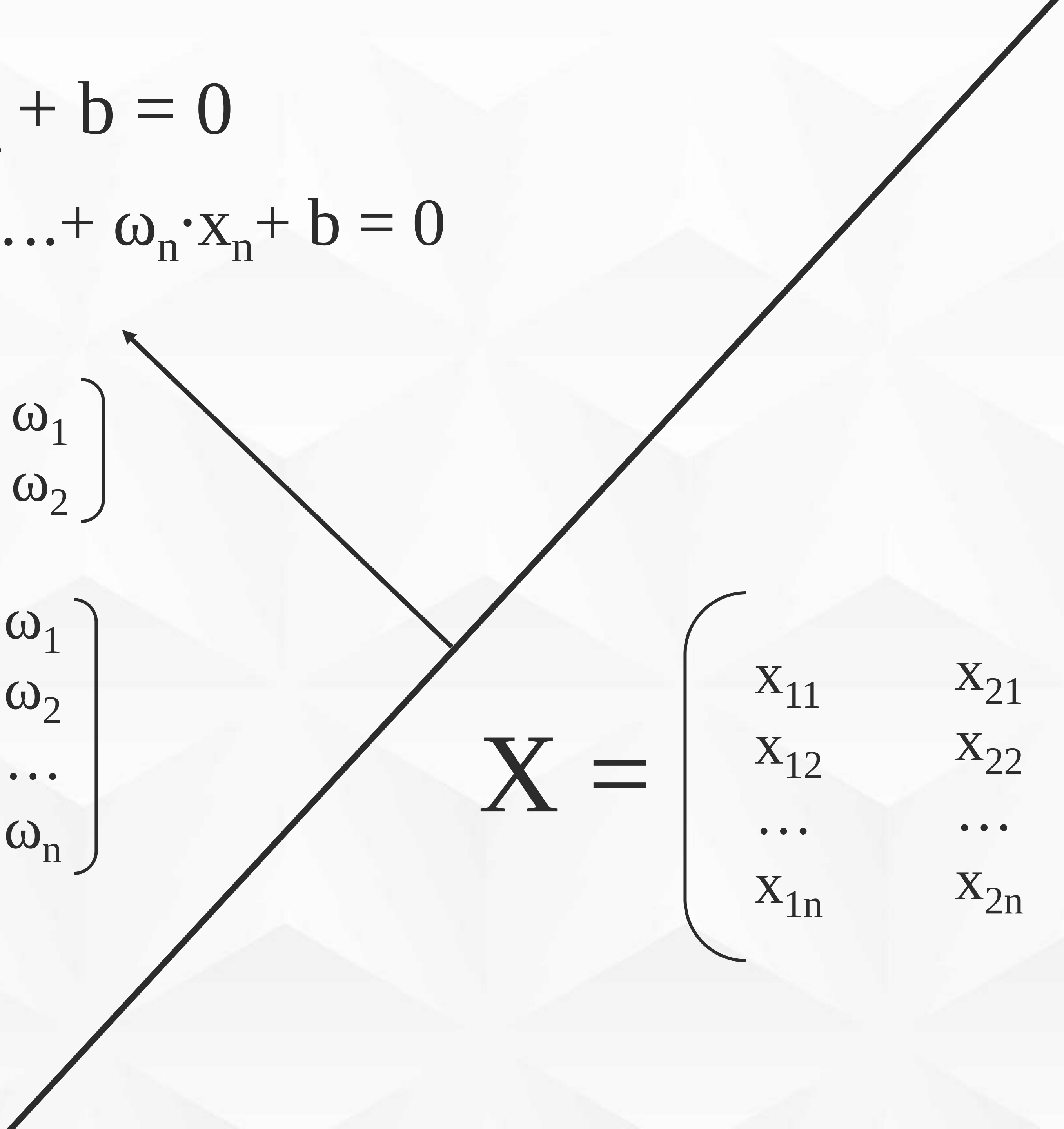
$(x_1, y_1), \ (x_2, y_2), \ \ldots, (x_m, y_m)$

$y_i \in Y, \quad Y = \{-1, 1\}$


X2
Margin
0
X1

$$\omega_1 \cdot x_1 + \omega_2 \cdot x_2 + b = 0$$

$$\omega_1 \cdot x_1 + \omega_2 \cdot x_2 + \ldots . + \omega_n \cdot x_n + b = 0$$



$$\begin{pmatrix} \omega_1 \\ \omega_2 \end{pmatrix}$$

$$\omega = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \ldots \\ \omega_n \end{pmatrix}$$

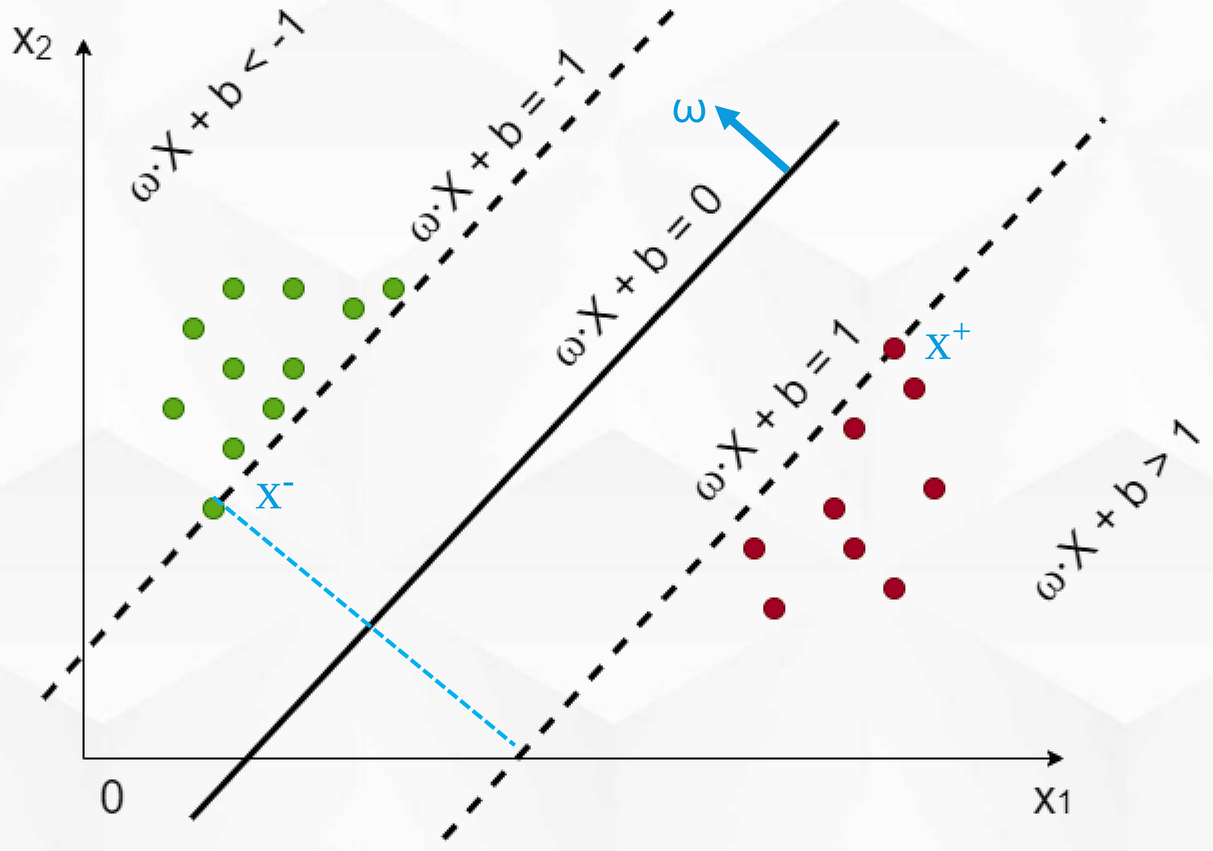
$$X = \begin{pmatrix} x_{11} & x_{21} & & x_{m1} \\ x_{12} & x_{22} & & x_{m2} \\ \ldots & \ldots & & \ldots \\ x_{1n} & x_{2n} & \ldots. & x_{mn} \end{pmatrix}$$

$$F(X) = \operatorname{sign}(\langle \omega \cdot X \rangle + b)$$

$$F(X) \leq -1, \quad y = -1$$

$$F(X) \geq 1, \quad y = 1$$

$$F(X) = 0$$



$$x^{+} = x^{-} + r \cdot \omega$$

$$\omega \cdot x^{-} + b = -1$$

$$\omega \cdot x^{+} + b = 1$$

$$\omega \cdot (x^{-} + r \cdot \omega) + b = 1$$

$$\|\omega\| = \sqrt{\omega \cdot \omega}$$

$$r \cdot \overbrace{\omega \cdot \omega}^{\|\omega\|^2} + \overbrace{\omega \cdot x^{-} + b}^{-1} = 1$$

$$r = \frac{2}{\|\omega\|^2}$$

$$r = \frac{2}{||\omega||^2}$$

Margin:

$$M = ||x^+ - x^-|| = ||r \cdot \omega|| = \frac{2 \cdot \omega}{||\omega||^2} = \frac{2}{||\omega||}$$

$$\begin{cases} ||\omega||^2 \rightarrow \min, \\ \underline{M(\ \omega, b) \geq 1} \end{cases}$$

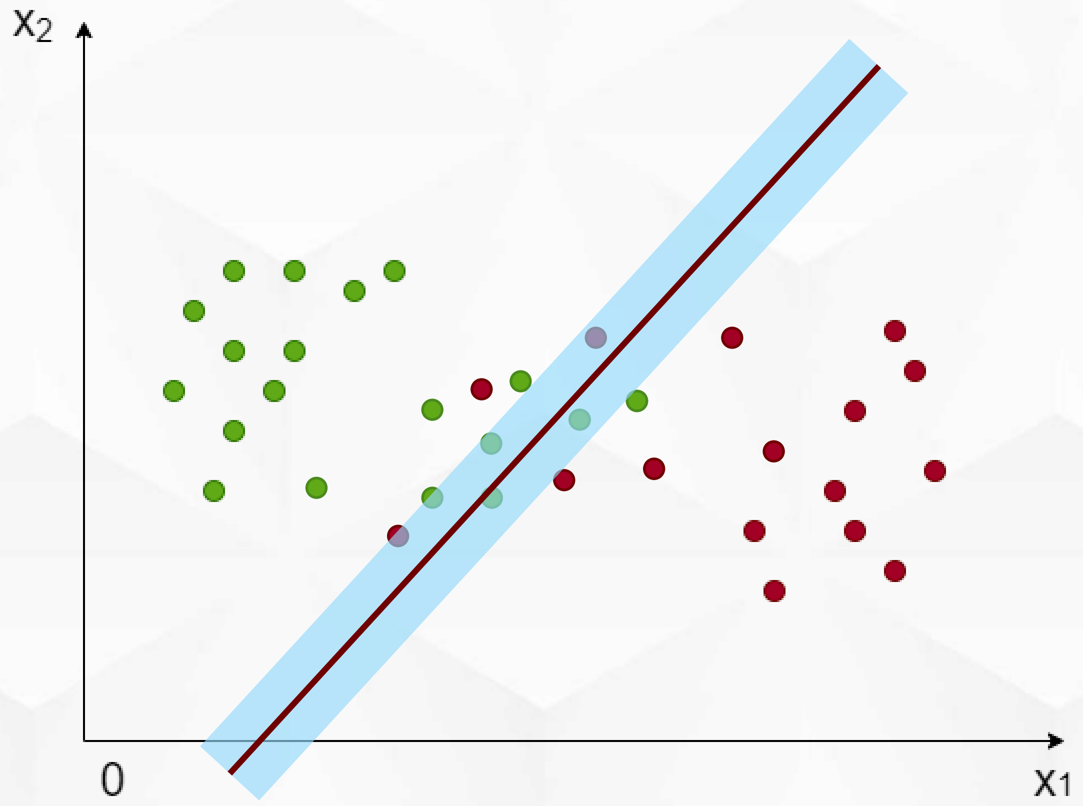
$$\begin{cases} ||\omega||^2 \rightarrow \min\limits_{\omega,\, b}, \\ M_i(\omega, b) \geq 1 , \; i = 1 \ldots n \end{cases}$$

Lagrange-Funktion:

$$L(\omega, b, \lambda) = \frac{1}{2||\omega||} - \sum_{i=1}^{n} \lambda_i \left(M(\omega, b) - 1\right)$$

$\lambda_i \geq 1$ - Lagrange-Multiplikatoren.

Nicht-linear separierbare Daten



$$\begin{cases} \|\omega\|^2 + \underline{C} \sum_{i=1}^{n} \boxed{\xi_i} \rightarrow \min\limits_{\omega,\, b,\, \xi} \\ M_i(\omega, b) \geq \underline{1 - \xi_i}, \; i = 1 \ldots n \\ \xi_i \geq 0, \; i = 1 \ldots n \end{cases}$$

Lagrange-Funktion:

$$L(\omega, b, \lambda, \eta) = \frac{1}{2\|\omega\|} - \sum_{i=1}^{n} \lambda_i \left(M(\omega, b) - 1\right) - \sum_{i=1}^{n} \xi_i \left(\lambda_i + \eta_i - C\right)$$

Karush-Kuhn-Tucker-Bedingungen

$$L(\omega, b, \lambda, \eta) = \frac{1}{2||\omega||} - \sum_{i=1}^{n} \lambda_i \left(M(\omega, b) - 1\right) - \sum_{i=1}^{n} \xi_i \left(\lambda_i + \eta_i - C\right)$$

$$\begin{cases} \dfrac{\partial L}{\partial \omega} = 0, \quad \dfrac{\partial L}{\partial b} = 0, \quad \dfrac{\partial L}{\partial \xi} = 0; \\ \xi_i \geq 0, \ \lambda_i \geq 0, \ \eta_i \geq 0, \ i = 1 \ldots n; \\ \lambda_i = 0 \ \text{oder} \ M_i(\omega, b) = 1 - \xi_i, \ i = 1 \ldots n; \\ \eta = 0 \ \text{oder} \ \xi_i = 0, \ i = 1 \ldots n \end{cases}$$

$$\frac{\partial L}{\partial \omega} = \omega - \sum_{i=1}^{n} \lambda_i y_i x_i = 0 \quad \Rightarrow \quad \omega = \sum_{i=1}^{n} \lambda_i y_i x_i;$$

$$\frac{\partial L}{\partial b} = - \sum_{i=1}^{n} \lambda_i y_i \quad \Rightarrow \quad \sum_{i=1}^{n} \lambda_i y_i = 0;$$

$$\frac{\partial L}{\partial \xi} = -\lambda_i - \eta_i + C = 0 \quad \Rightarrow \quad \lambda_i + \eta_i = C;$$

uninformative Objekte:
$\lambda_i = 0,\ \eta_i = C,\ \xi_i = 0,\ M \geq 1$

Support Vectors:
$0 < \lambda_i < C,\ 0 < \eta_i < C,\ \xi_i = 0,\ M = 1$

Support Vectors – Ausreißer:
$\lambda_i = C,\ \eta_i = 0,\ \xi_i > 0,\ M < 1$

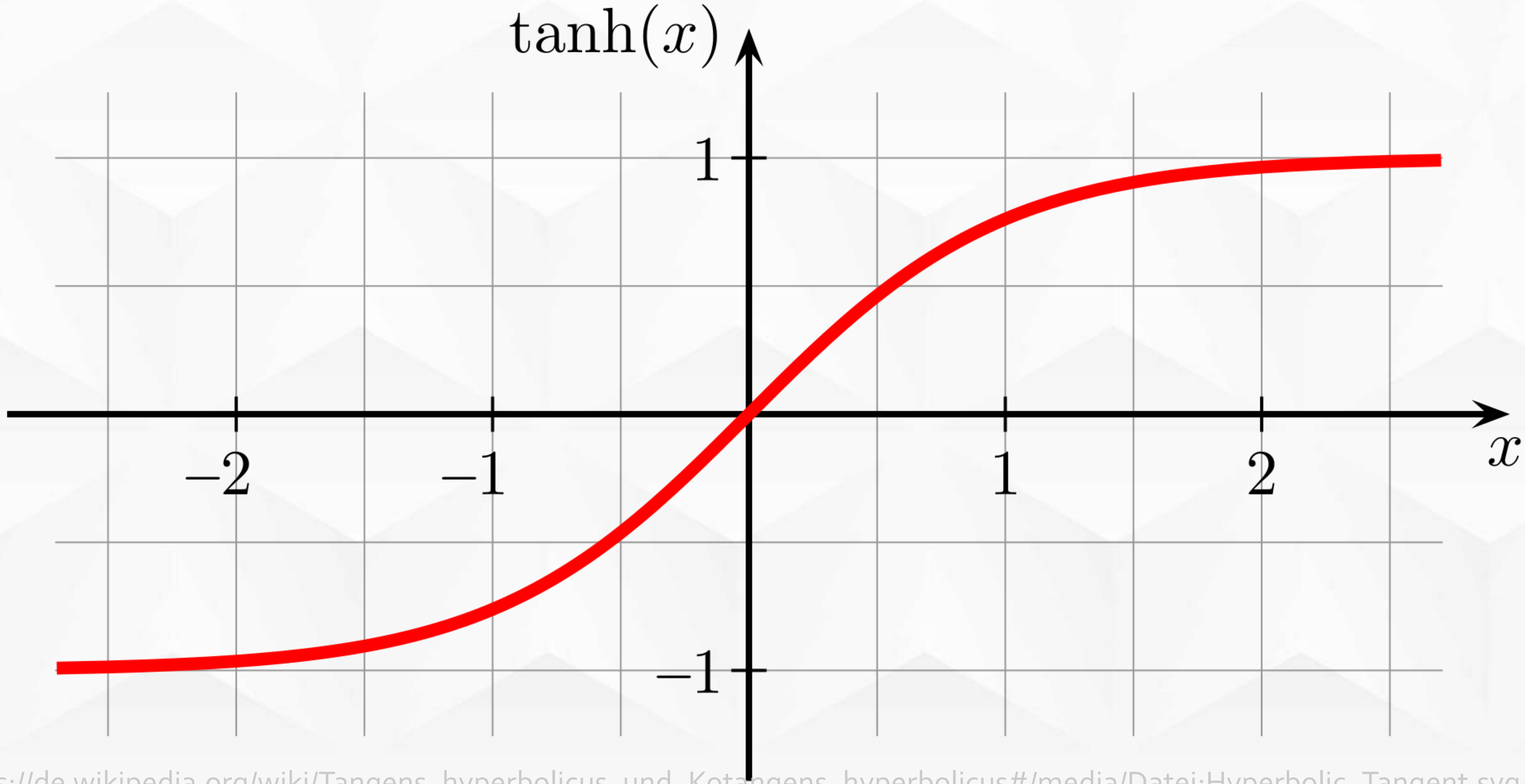
$$\begin{cases} -\sum_{i=1}^{n}\lambda_i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j\, y_i y_j \overset{K(x_i \cdot x_j)}{\boxed{(x_i \cdot x_j)}} \to \min\limits_{\lambda} \\ 0 \le \lambda_i \le C,\ \ i = 1\ldots n \\ \sum_{j=1}^{n}\lambda_i\, y_i = 0 \end{cases}$$

Kernel-Trick:

Die Anwendung der nichtlinearen Kernel-Funktion um die Dimensionalität des Raumes zu vergrößern und eine Hyperebene im neuen resultierenden Raum zu finden

Punktprodukt -> Kernel
Produkt des Kernels -> Kernel
Produkte von Funktionen -> Kernel
Summe der Kernel -> Kernel
(nur dann, wenn jeder der Terme mit einem positiven Koeffizienten multipliziert wird)

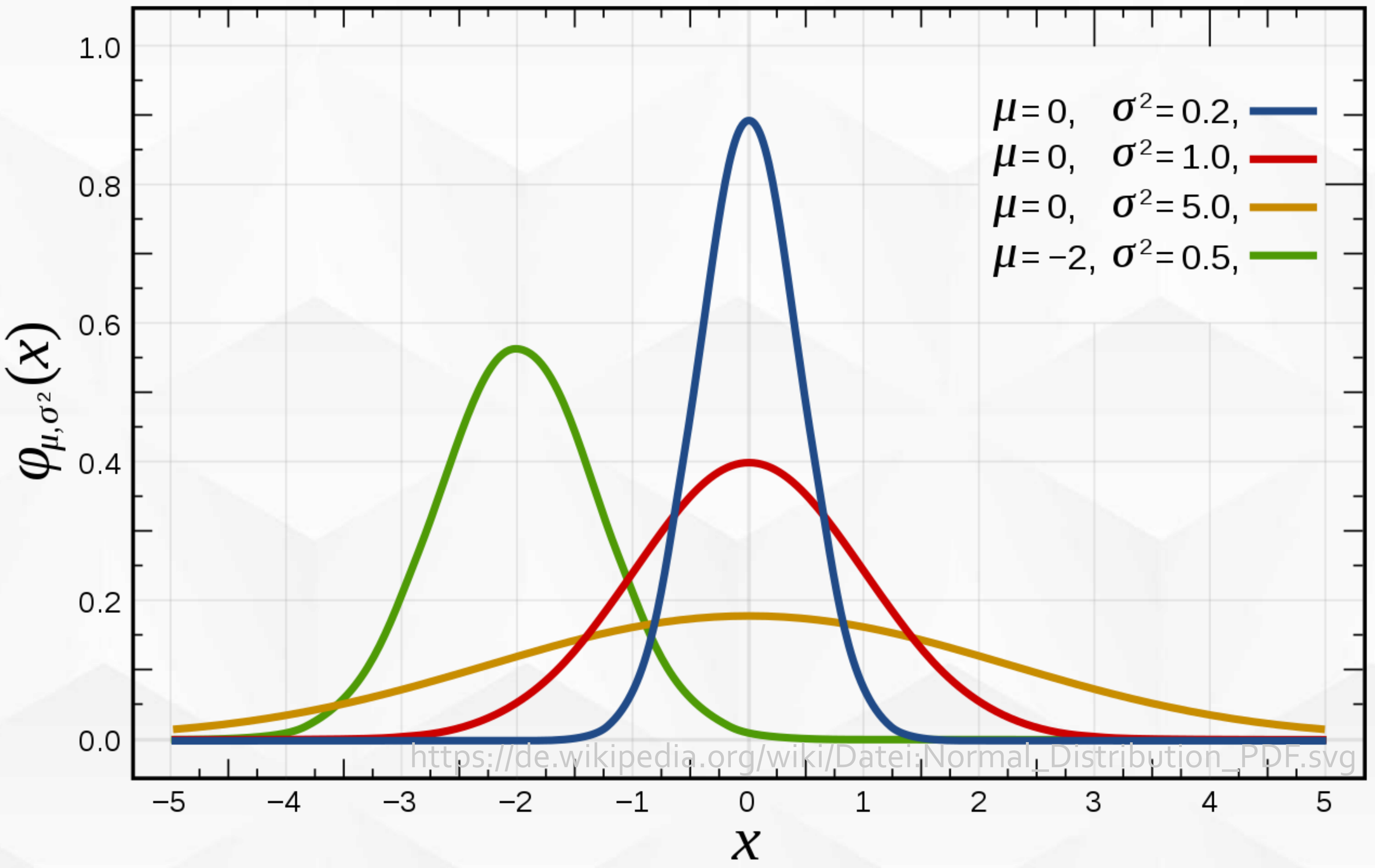
Tangens Hyperbolicus $k(x,y) = \tanh(\alpha x^T y + c)$



https://de.wikipedia.org/wiki/Tangens_hyperbolicus_und_Kotangens_hyperbolicus#/media/Datei:Hyperbolic_Tangent.svg

Gaussian Kernel

$$k(x,y) = \exp\left(-\frac{||x-y||^2}{2||\sigma||}\right)$$

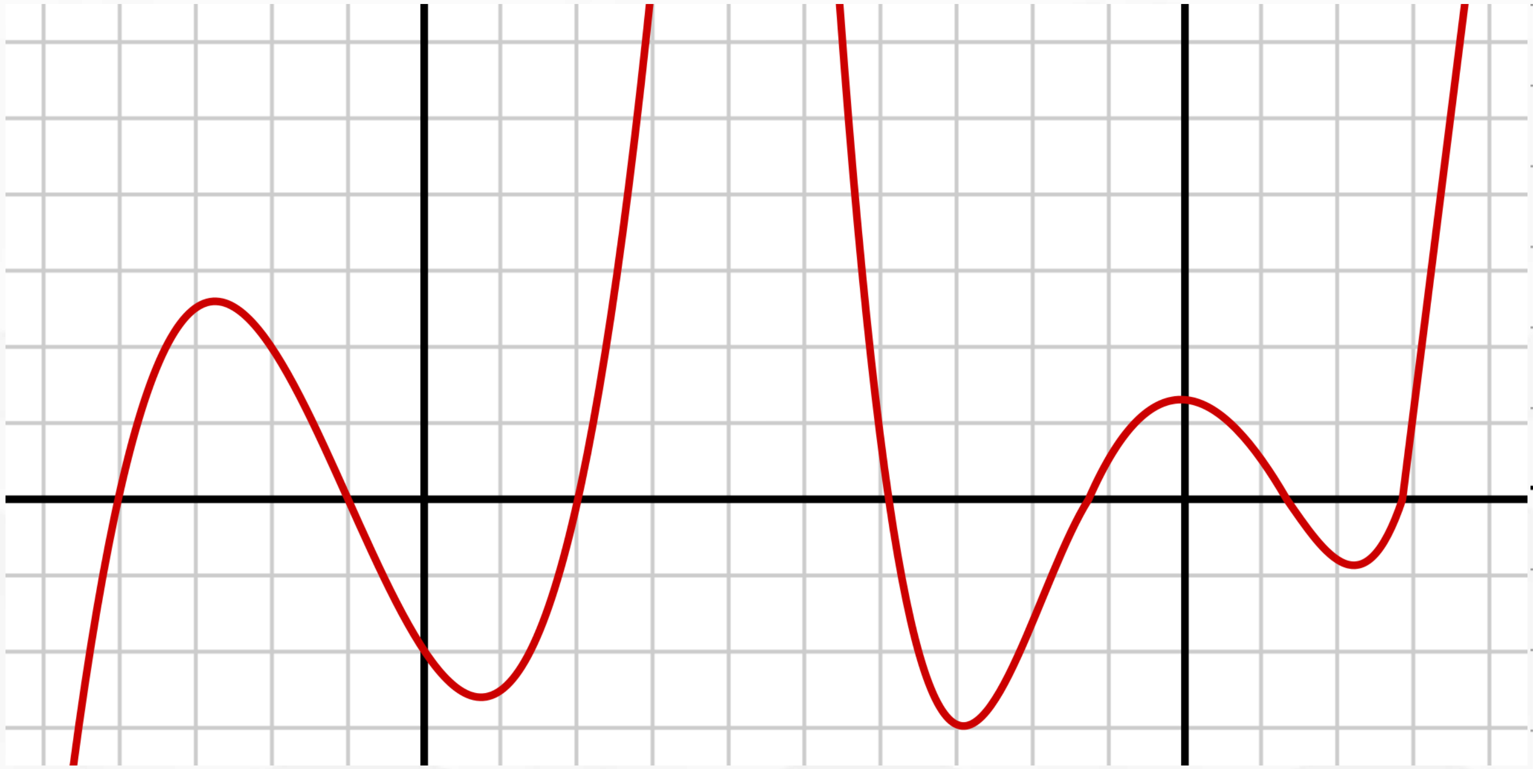


https://de.wikipedia.org/wiki/Datei:Normal_Distribution_PDF.svg

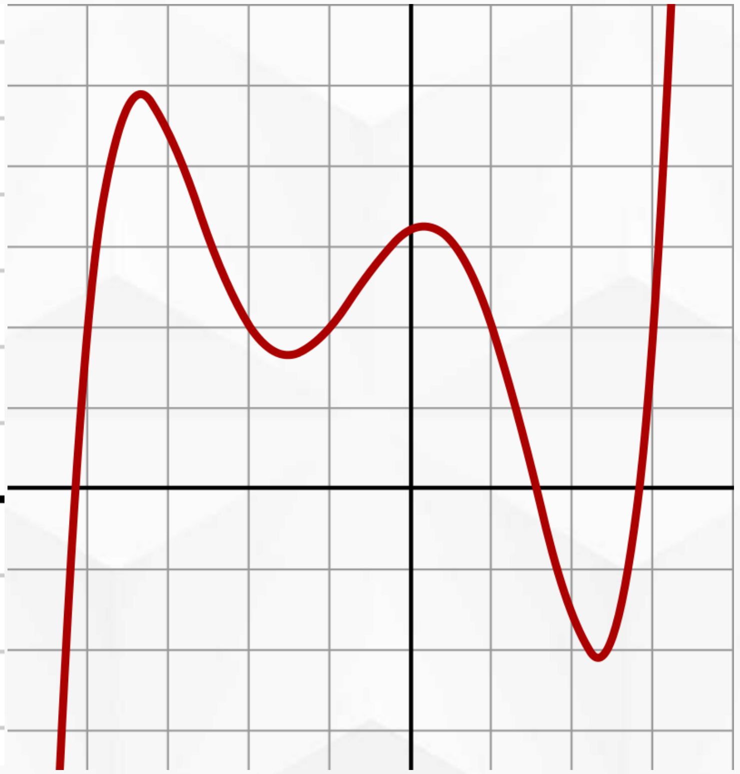
Polynomial Kernel

$$k(x,y) = (\alpha x^T y + c)^d$$

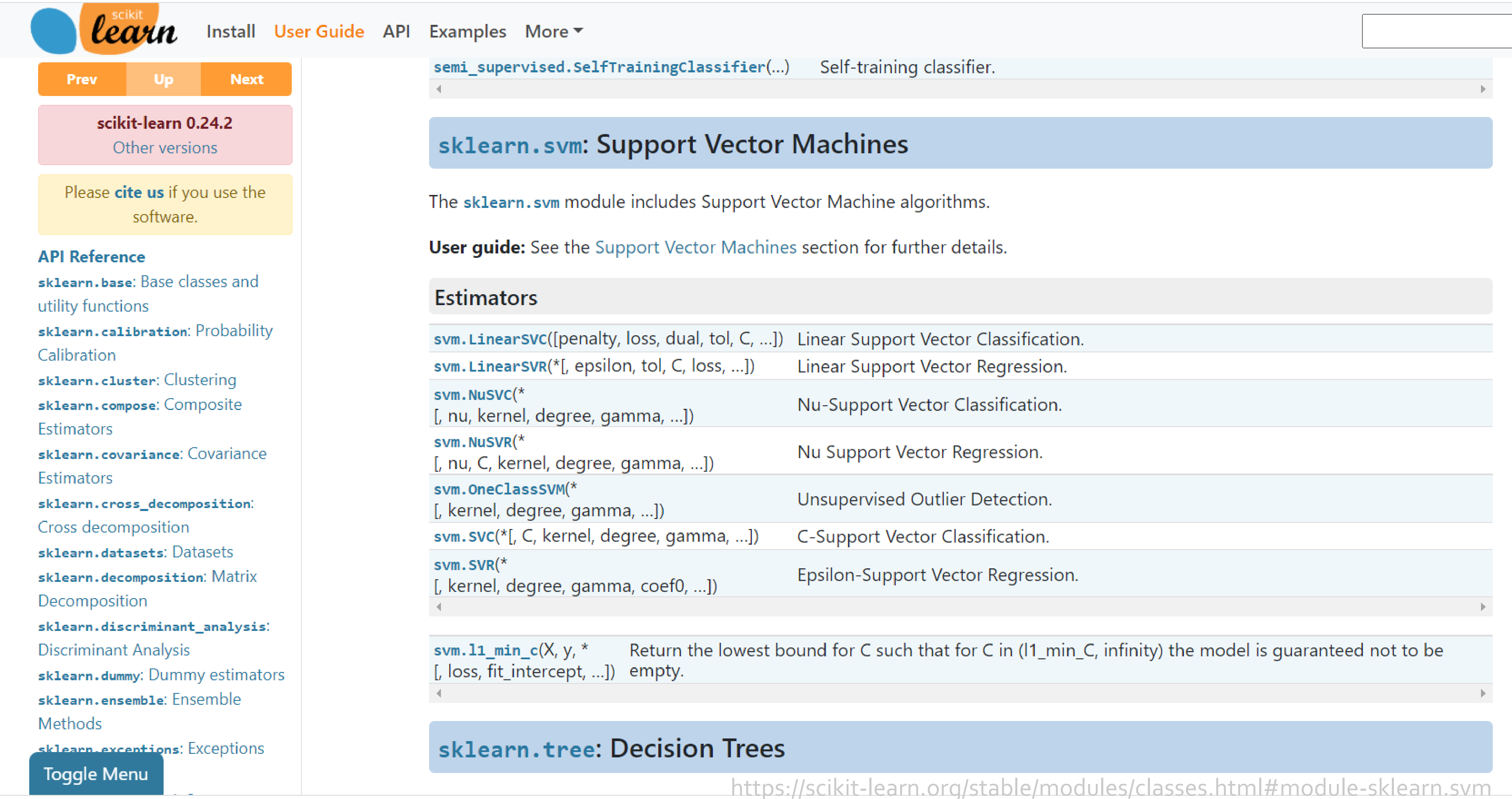
d = 3 d = 4



d = 5



https://en.wikipedia.org/wiki/Polynomial



scikit-learn 0.24.2

Other versions

Please **cite us** if you use the software.

API Reference

- `sklearn.base`: Base classes and utility functions
- `sklearn.calibration`: Probability Calibration
- `sklearn.cluster`: Clustering
- `sklearn.compose`: Composite Estimators
- `sklearn.covariance`: Covariance Estimators
- `sklearn.cross_decomposition`: Cross decomposition
- `sklearn.datasets`: Datasets
- `sklearn.decomposition`: Matrix Decomposition
- `sklearn.discriminant_analysis`: Discriminant Analysis
- `sklearn.dummy`: Dummy estimators
- `sklearn.ensemble`: Ensemble Methods
- `sklearn.exceptions`: Exceptions

`semi_supervised.SelfTrainingClassifier`(...)	Self-training classifier.

`sklearn.svm`: Support Vector Machines

The `sklearn.svm` module includes Support Vector Machine algorithms.

User guide: See the Support Vector Machines section for further details.

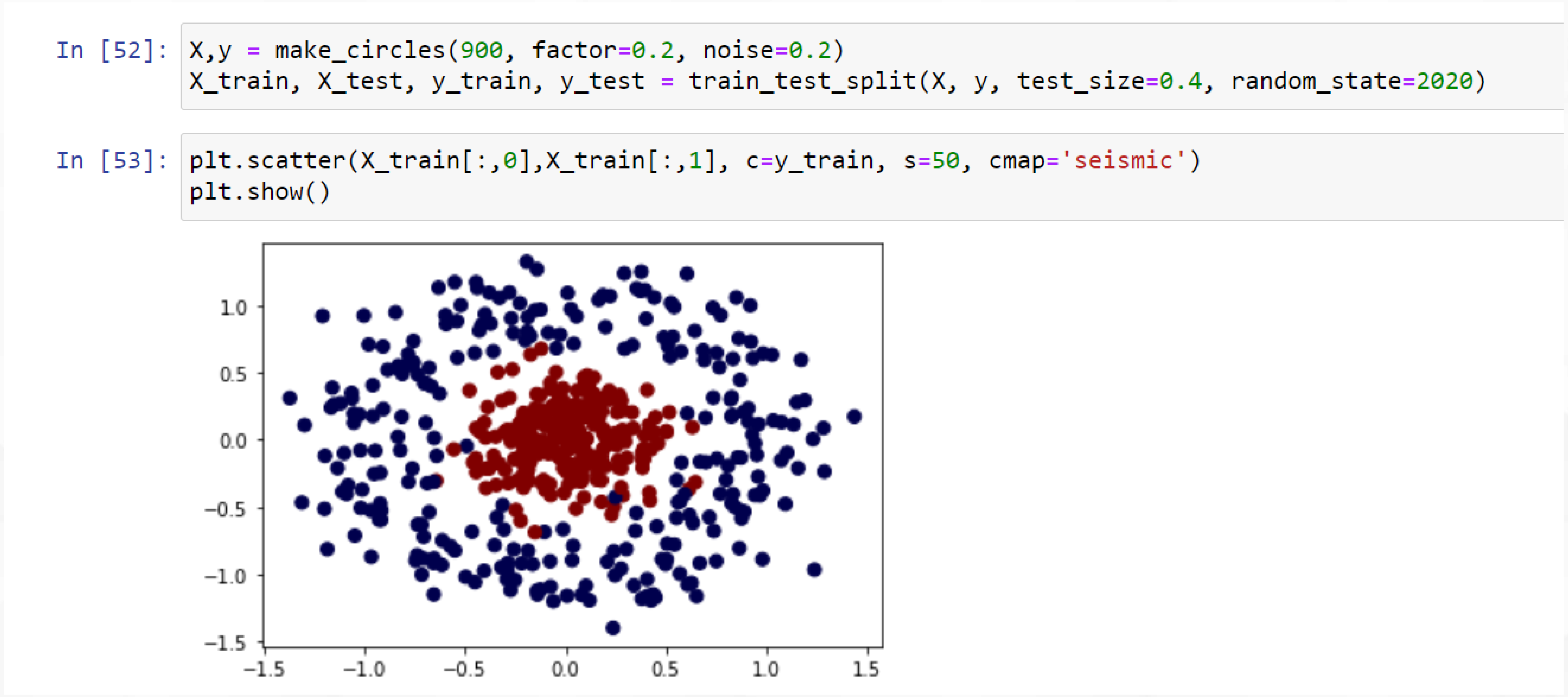
Estimators

`svm.LinearSVC`([penalty, loss, dual, tol, C, ...])	Linear Support Vector Classification.
`svm.LinearSVR`(*[, epsilon, tol, C, loss, ...])	Linear Support Vector Regression.
`svm.NuSVC`(*[, nu, kernel, degree, gamma, ...])	Nu-Support Vector Classification.
`svm.NuSVR`(*[, nu, C, kernel, degree, gamma, ...])	Nu Support Vector Regression.
`svm.OneClassSVM`(*[, kernel, degree, gamma, ...])	Unsupervised Outlier Detection.
`svm.SVC`(*[, C, kernel, degree, gamma, ...])	C-Support Vector Classification.
`svm.SVR`(*[, kernel, degree, gamma, coef0, ...])	Epsilon-Support Vector Regression.

`svm.l1_min_c`(X, y, *[, loss, fit_intercept, ...])	Return the lowest bound for C such that for C in (l1_min_C, infinity) the model is guaranteed not to be empty.

`sklearn.tree`: Decision Trees

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.svm

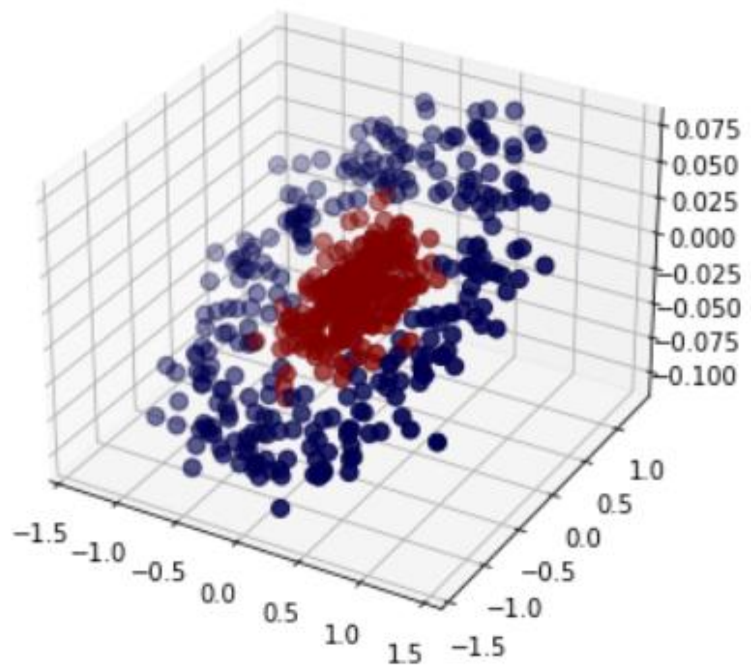


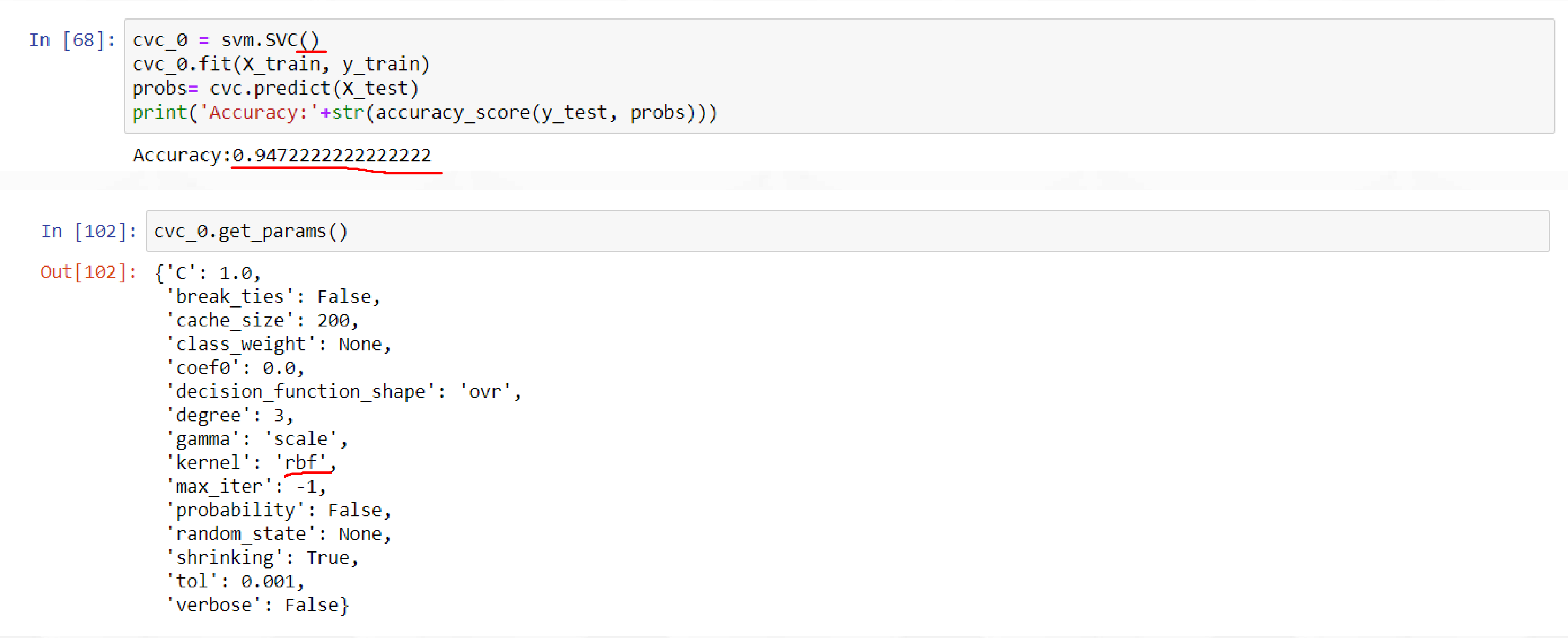
```
In [52]: X,y = make_circles(900, factor=0.2, noise=0.2)
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=2020)

In [53]: plt.scatter(X_train[:,0],X_train[:,1], c=y_train, s=50, cmap='seismic')
         plt.show()
```

```
In [69]: lsvc = LinearSVC( )
         lsvc.fit(X_train, y_train)
         probs= lsvc.predict(X_test)
         print('Accuracy:'+str(accuracy_score(y_test, probs)))

         Accuracy:0.35
```

```
In [85]: Zl = lsvc.decision_function(X_train)
```

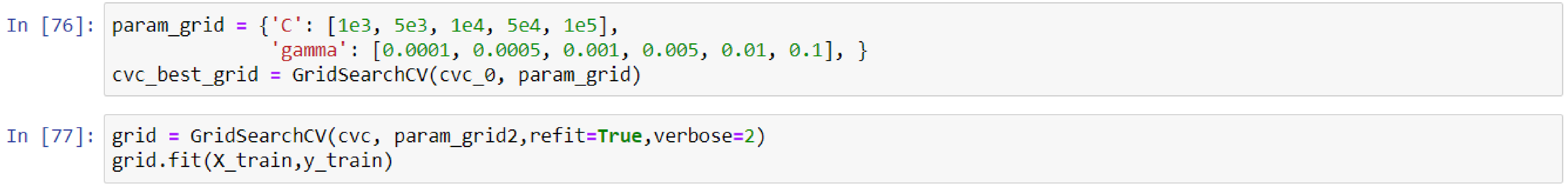
```
In [86]: fig = plt.figure()
         ax = Axes3D(fig)
         ax.scatter(X_train[:,0], X_train[:,1], Zl, c=y_train, s=50, cmap='seismic')

Out[86]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x25b66ce99a0>
```



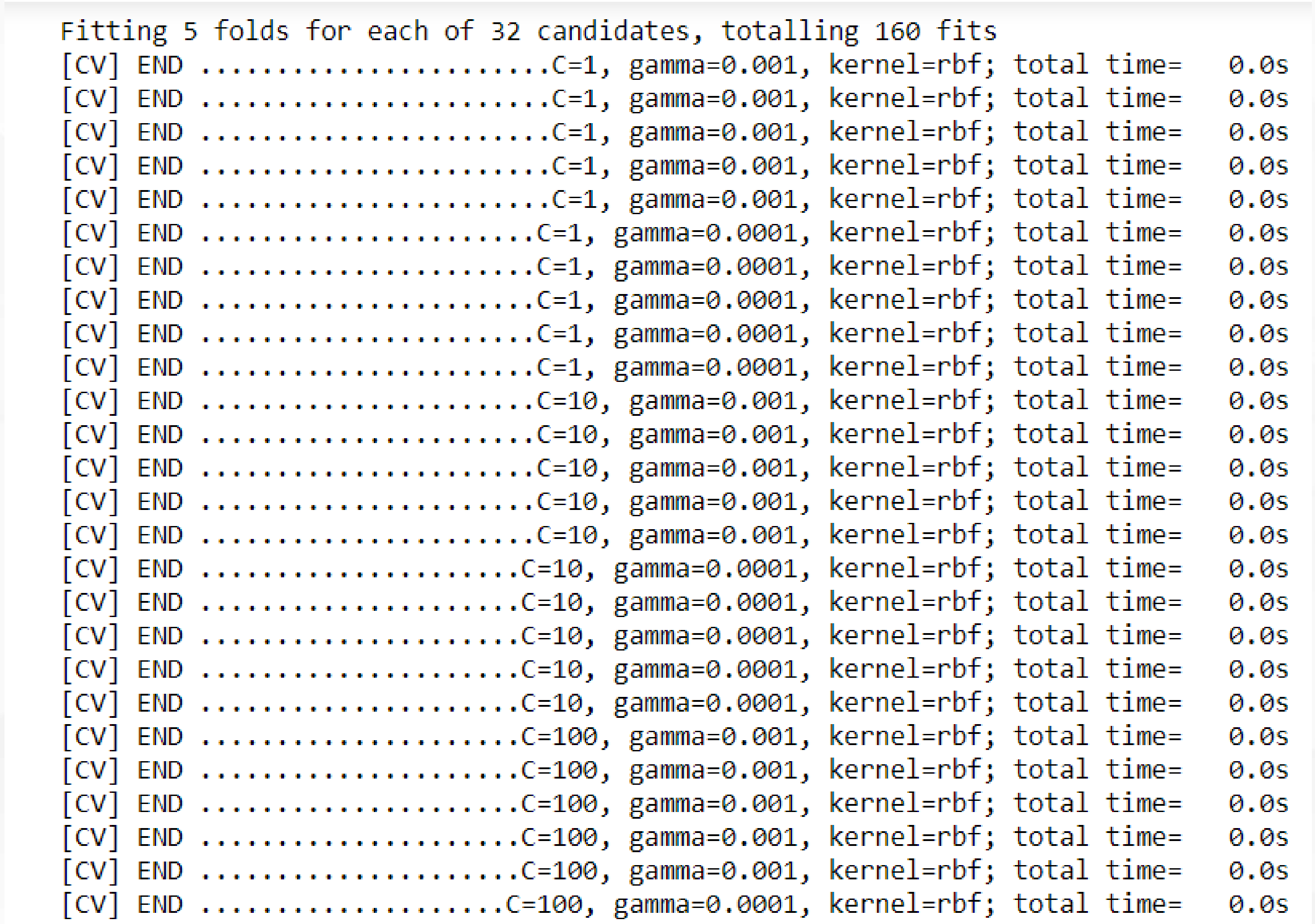


```
In [68]: cvc_0 = svm.SVC()
         cvc_0.fit(X_train, y_train)
         probs= cvc.predict(X_test)
         print('Accuracy:'+str(accuracy_score(y_test, probs)))

         Accuracy:0.9472222222222222
```

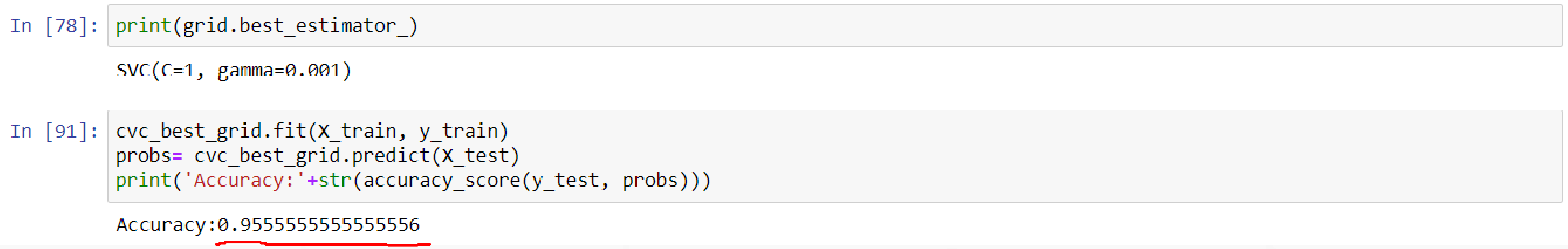
```
In [102]: cvc_0.get_params()

Out[102]: {'C': 1.0,
           'break_ties': False,
           'cache_size': 200,
           'class_weight': None,
           'coef0': 0.0,
           'decision_function_shape': 'ovr',
           'degree': 3,
           'gamma': 'scale',
           'kernel': 'rbf',
           'max_iter': -1,
           'probability': False,
           'random_state': None,
           'shrinking': True,
           'tol': 0.001,
           'verbose': False}
```



```
In [76]: param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
                       'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
         cvc_best_grid = GridSearchCV(cvc_0, param_grid)

In [77]: grid = GridSearchCV(cvc, param_grid2,refit=True,verbose=2)
         grid.fit(X_train,y_train)
```

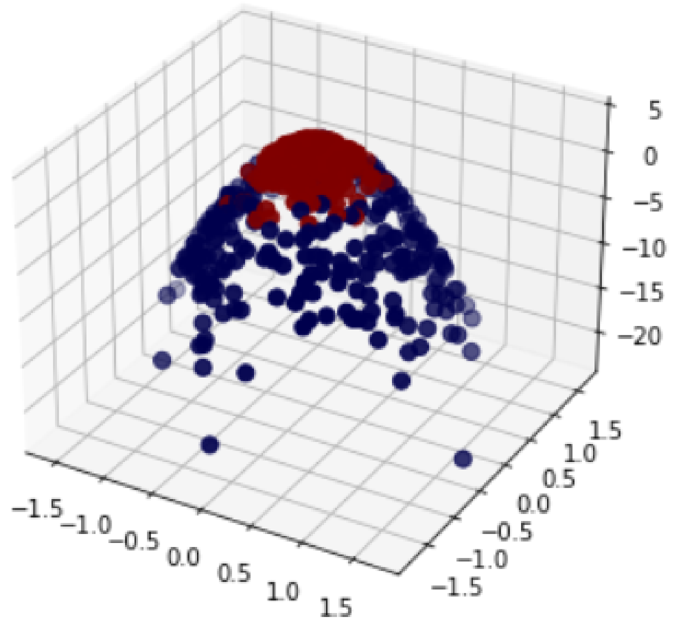


```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
[CV] END .......................C=1, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END .......................C=1, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END .......................C=1, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END .......................C=1, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END .......................C=1, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ......................C=1, gamma=0.0001, kernel=rbf; total time=   0.0s
[CV] END ......................C=1, gamma=0.0001, kernel=rbf; total time=   0.0s
[CV] END ......................C=1, gamma=0.0001, kernel=rbf; total time=   0.0s
[CV] END ......................C=1, gamma=0.0001, kernel=rbf; total time=   0.0s
[CV] END ......................C=1, gamma=0.0001, kernel=rbf; total time=   0.0s
[CV] END ......................C=10, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ......................C=10, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ......................C=10, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ......................C=10, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ......................C=10, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END .....................C=10, gamma=0.0001, kernel=rbf; total time=   0.0s
[CV] END .....................C=10, gamma=0.0001, kernel=rbf; total time=   0.0s
[CV] END .....................C=10, gamma=0.0001, kernel=rbf; total time=   0.0s
[CV] END .....................C=10, gamma=0.0001, kernel=rbf; total time=   0.0s
[CV] END .....................C=10, gamma=0.0001, kernel=rbf; total time=   0.0s
[CV] END .....................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END .....................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END .....................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END .....................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END .....................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END ....................C=100, gamma=0.0001, kernel=rbf; total time=   0.0s
```



```
In [78]: print(grid.best_estimator_)

         SVC(C=1, gamma=0.001)

In [91]: cvc_best_grid.fit(X_train, y_train)
         probs= cvc_best_grid.predict(X_test)
         print('Accuracy:'+str(accuracy_score(y_test, probs)))

         Accuracy:0.9555555555555556
```

```
In [91]: cvc_best_grid.fit(X_train, y_train)
         probs= cvc_best_grid.predict(X_test)
         print('Accuracy:'+str(accuracy_score(y_test, probs)))

         Accuracy:0.9555555555555556

In [100]: Zc = cvc_best_grid.decision_function(X_train)

In [101]: fig = plt.figure()
          ax = Axes3D(fig)
          ax.scatter(X_train[:,0], X_train[:,1], Zc, c=y_train, s=50, cmap='seismic')

Out[101]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x1c4f3b28940>
```



Fazit

In diesem Vortrag haben wir die mathematischen Grundlagen des SVM-Algorithmus skizziert. Eines der wichtigsten Merkmale dieses Algorithmus ist der Kernel-Trick, mit dem wir eine nichtlineare Funktion hinzufügen, die die Dimensionalität des Raums vergrößert und die Ergebnisse erheblich verbessert.

Quellen:

V. V. Vyugin: Mathematischehttps Grundlagen des maschinellen Lernens und der Prognosetheorie; s. 91-94; Moskau 2013
Anthony So, Thomas V. Joseph, Robert Thas John, Andrew Worsley, and Dr. Samuel Asare: The Data Science Workshop; s 371; Packt Publishing 2020
Chris Albon: Python Machine Learning Cookbook: Practical solutions from preprocessing to deep learning, 2018, O’Reilly
https://habr.com/ru/post/105220/
https://de.wikipedia.org/wiki/Lagrange-Multiplikator
https://www.coursera.org/lecture/vvedenie-mashinnoe-obuchenie/
mietod-opornykh-viektorov-obobshchieniie-dlia-nielinieinogho-sluchaia-EQzGo
https://de.wikipedia.org/wiki/Norm_(Mathematik)
https://www.youtube.com/watch?v=nF0rqbOwOAw&t=363s
https://www.youtube.com/watch?v=Adi67_94_gc
https://de.wikipedia.org/wiki/Karush-Kuhn-Tucker-Bedingungen
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
https://en.wikipedia.org/wiki/Polynomial#Polynomial_functions
https://en.wikipedia.org/wiki/Sigmoid_function
https://www.youtube.com/watch?v=adBmzj01CSs
https://www.youtube.com/watch?v=1aQLEzeGJC8
https://www.youtube.com/watch?v=jA9CpUSaSN4&t=2564s