Python - Programmierung für die Computerbasierte Intelligenz

Tobias Häuser

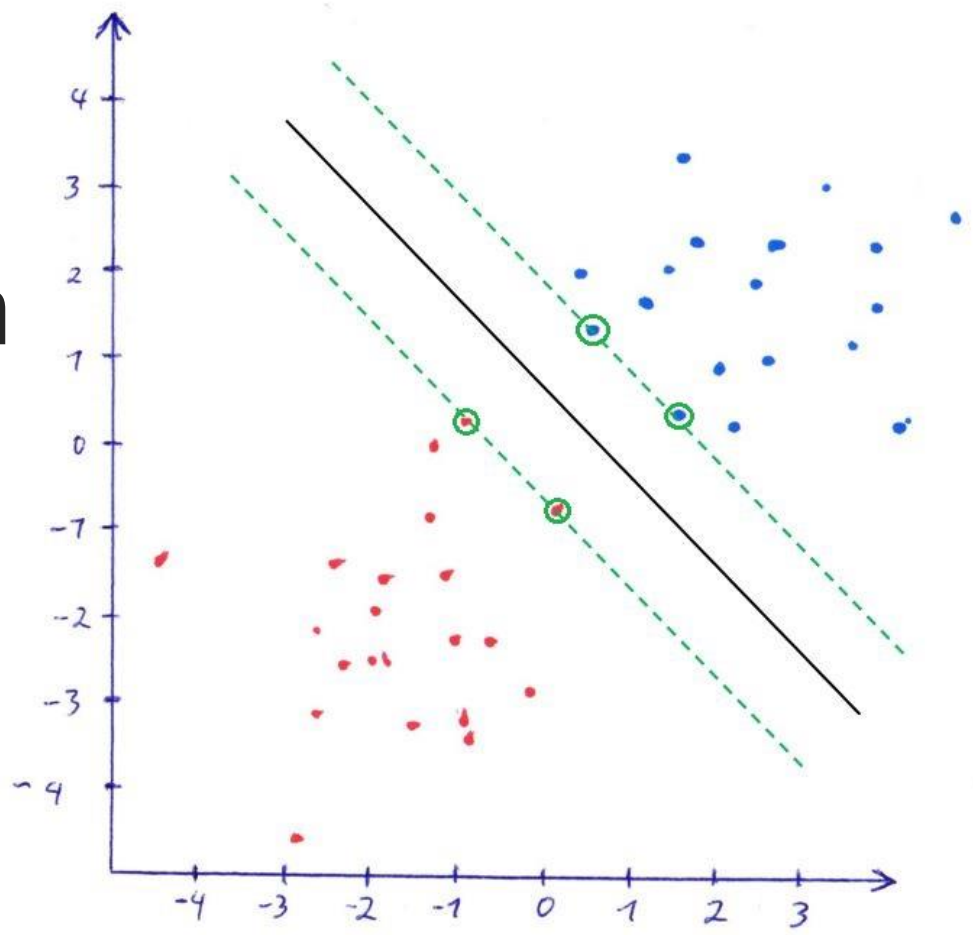
Vorlesung 6.

04.09.2018

1. Support Vector Machine (SVM) – ohne Kernel
    1. Einführung – large margin concept
    2. Einführung – soft margin concept
    3. Beispiel – sklearn iris_Dataset (Petalum)
2. SVM – mit Kernel
    1. Einführung
    2. Beispiel – sklearn iris_Dataset (Petalum)
3. Weitere Anpassungsmöglichkeiten
    1. Der C-Parameter
    2. Gamma
    3. Übersicht verschiedenen Parameter Kombinationen
    4. Beispiel – sklearn iris_Dataset (Petalum)

1.1. Einführung SVM – large margin concept

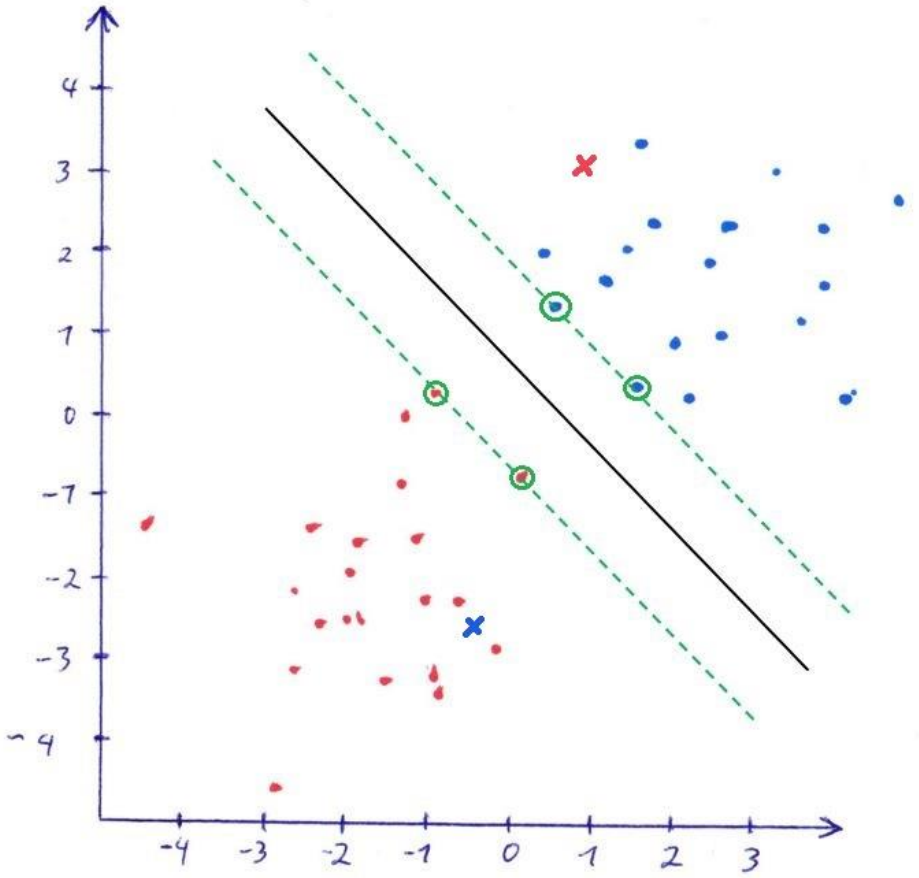
- Wichtige Gruppe von Klassifikationen
- Bsp.: im zweidimensionalen Raum, befinden sich 2 Klassen die sich linear trennen lassen
- Trennlinie wird so gewählt, dass maximale Separation erfolgt



- Grüner Kreise -> Stützvektoren
- Durchgezogene Linie -> Separationslinie
- Gestrichelte Linie -> Spanne (min/max)

1.2. Einführung SVM – soft margin concept

- Wird angewandt, wenn keine lineare Separation möglich ist oder die Spanne zw. den Klassen zu gering ist
  - Sieht Bild, Klassen nicht linear trennbar
- Idee dahinter
  - Man erlaubt Fehlklassifikationen
  - Einführung des C-Parameters

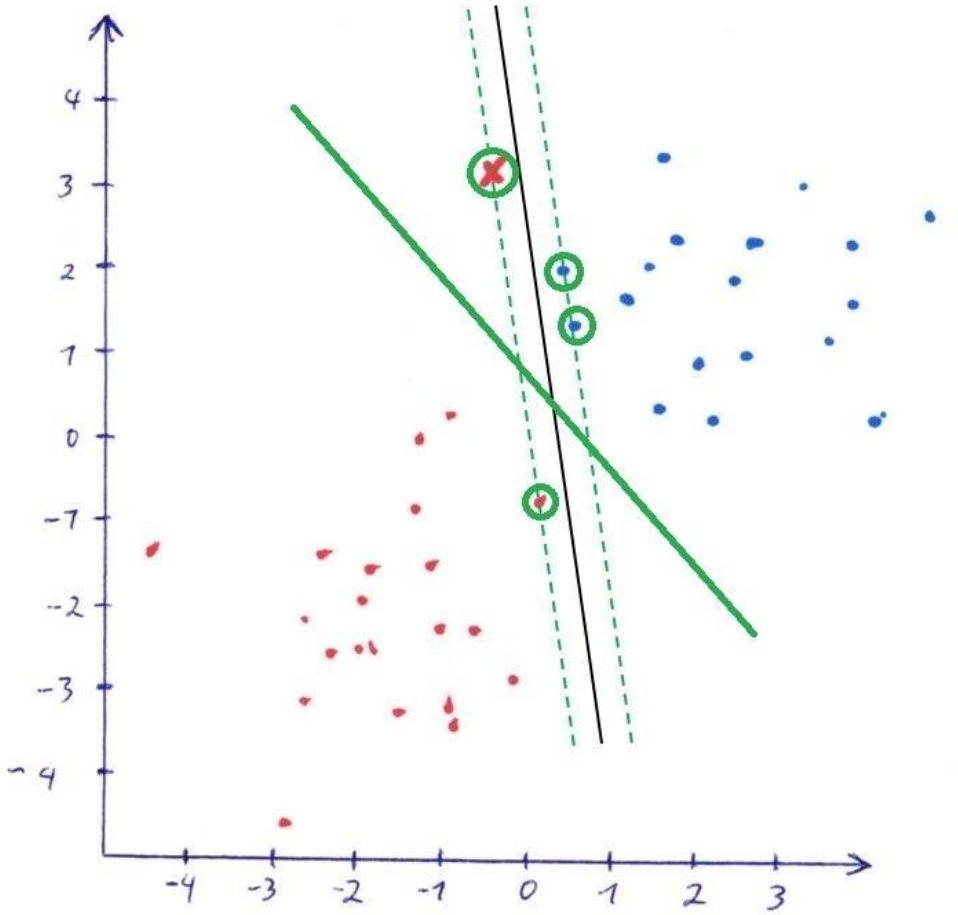


- Grüner Kreise -> Stützvektoren
- Durchgezogene Linie -> Separationslinie
- Gestrichelte Linie -> Spanne (min/max)

1.2. Einführung SVM – soft margin concept

- Ziel ist es, die Fehlklassifikationen so gering wie möglich zu halte

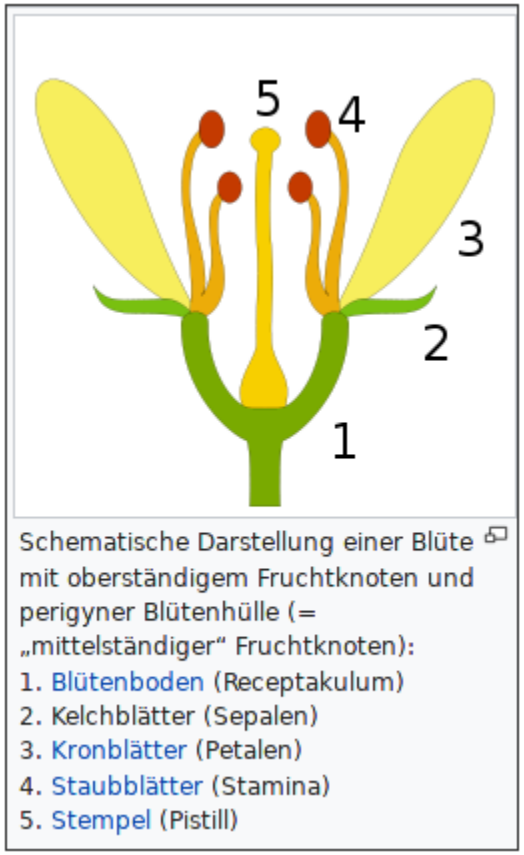
- Im Bild noch einmal, wenn die Spanne zu gering ist, erlaubt man Fehler und wählt die Grüne Linie als Separationslinie



- Grüner Kreise -> Stützvektoren
- Durchgezogene Linie -> Separationslinie
- Gestrichelte Linie -> Spanne (min/max)

1.3. Beispiel – sklearn iris_Dataset (Petalum)

- Beispiel am sklearn Datensatz, bezogen auf die Kronblätterlänge, -breite und dessen Klassifizierung
- In diesem Beispiel gibt es 3 Klasen von Iris Blättern (Setosa, Versicolor und Virginica)



Schematische Darstellung einer Blüte mit oberständigem Fruchtknoten und perigyner Blütenhülle (= „mittelständiger" Fruchtknoten):
1. Blütenboden (Receptakulum)
2. Kelchblätter (Sepalen)
3. Kronblätter (Petalen)
4. Staubblätter (Stamina)
5. Stempel (Pistill)

1.3. Beispiel – sklearn iris_Dataset (Petalum)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.svm import SVC

# Datensatz laden
dataset      = load_iris()
data         = np.array(dataset.data)
labels       = np.array(dataset.target)
```

1.3. Beispiel – sklearn iris_Dataset (Petalum)

```
def plot_data(data, labels, kernel_fn="linear", C=1.0, degree=3, gamma='auto'):
    # Nur Petalum Länge beachten | default = 0
    i = 1

    # SVM erstellen und mit Daten befuellen
    svm = SVC(kernel=kernel_fn, C=C, degree=degree, gamma=gamma)
    svm.fit(data[:, 2 * i:2 * i + 2], labels)

    # Minimale und Maximale Werte aus X nehmen und ein
    # Matrix bilden mit alle Koordinaten mit Abstand 0.01
    x_min, x_max = data[:, 0 + (2 * i)].min() - 1, data[:, 0 + (2 * i)].max() + 1
    y_min, y_max = data[:, 1 + (2 * i)].min() - 1, data[:, 1 + (2 * i)].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))
```

1.3. Beispiel – sklearn iris_Dataset (Petalum)

```
# Vorhersagen nehmen für alle Koordinaten
prediction = svm.predict(np.c_[xx.ravel(), yy.ravel()])

# Array auf Pixel Breite & Länge shapen
# Hintergrund färben für jede Punkt aus xx,yy
prediction = prediction.reshape(xx.shape)
plt.contourf(xx, yy, prediction, cmap=plt.get_cmap('summer'))

# Daten auch als Punkte plotten
plt.scatter(data[:, 0 + (2 * i)],
            data[:, 1 + (2 * i)],
            c=labels, cmap=plt.get_cmap('summer'),
            edgecolors='black')
```

1.3. Beispiel – sklearn iris_Dataset (Petalum)

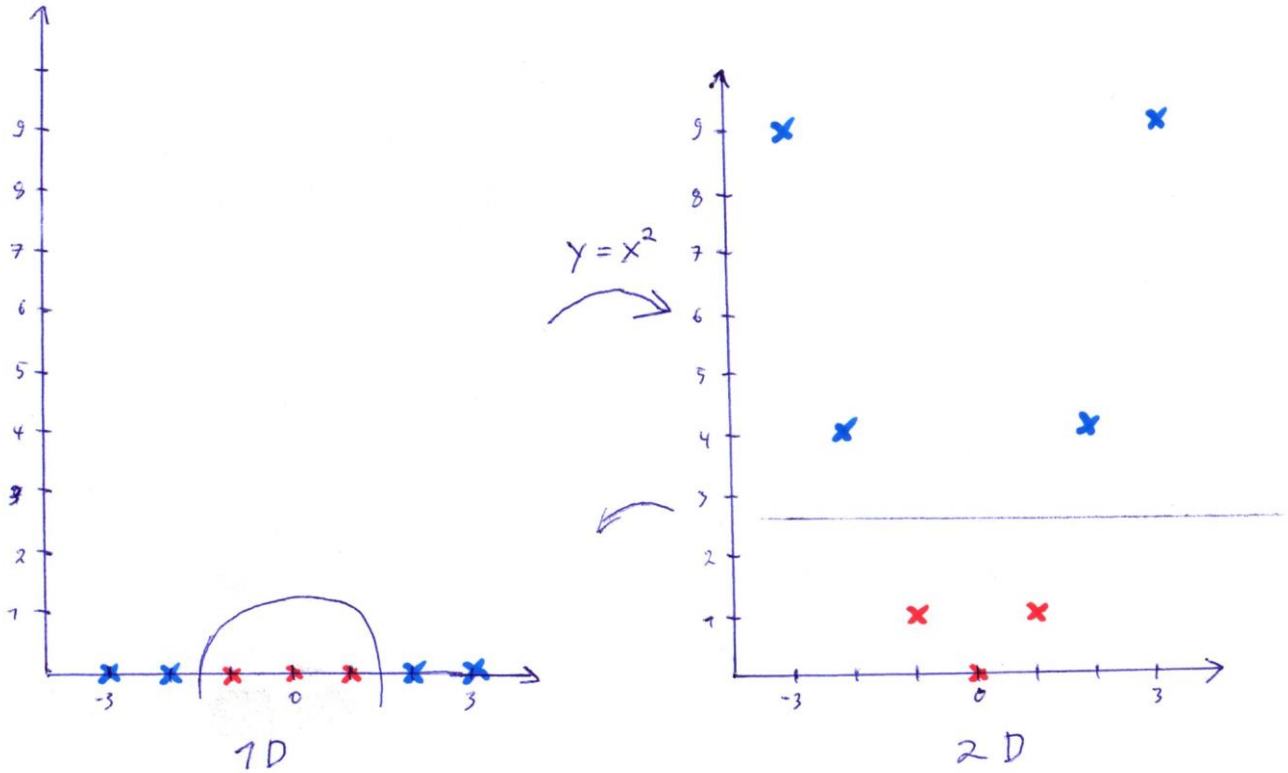
```
    # Beschrieftungen anpassen
    plt.ylabel('Petalum Plot')
    plt.title("Kernel Function: %s \n C: %.2f \n gamma: %s " %
                  (kernel_fn, C, str(gamma)))

    plt.show()

# SVM berechnen und Plotten
plot_data(data, labels)
```

2.1. Einführung SVM – mit Kernel

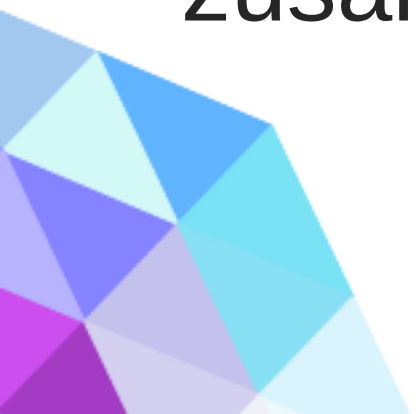
- Mit Kernel, eine nichtlineare Trennfunktion möglich
- Idee dahinter, Daten werden in höherdimensionalen Raum transferiert, wo sie dann linear getrennt werden können
  - z.B. 1D -> 2D
    oder 2D -> 3D



2.1. Einführung SVM – mit Kernel

- Bei der höher dimensionalen Transformierung, können die Daten dann linear durch eine (Hyper-)Ebene getrennt werden

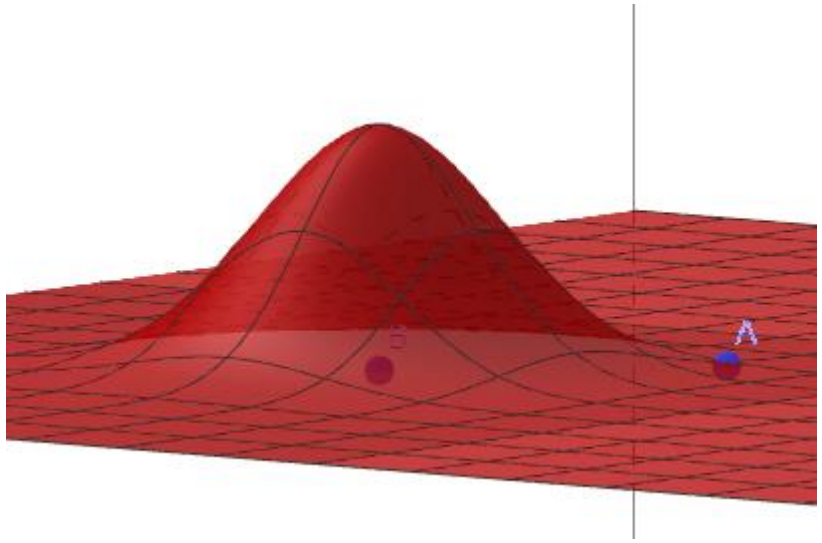
- Nach der Rücktransformierung, ist diese dann meist nicht mehr linear und auch nicht mehr zusammenhängend

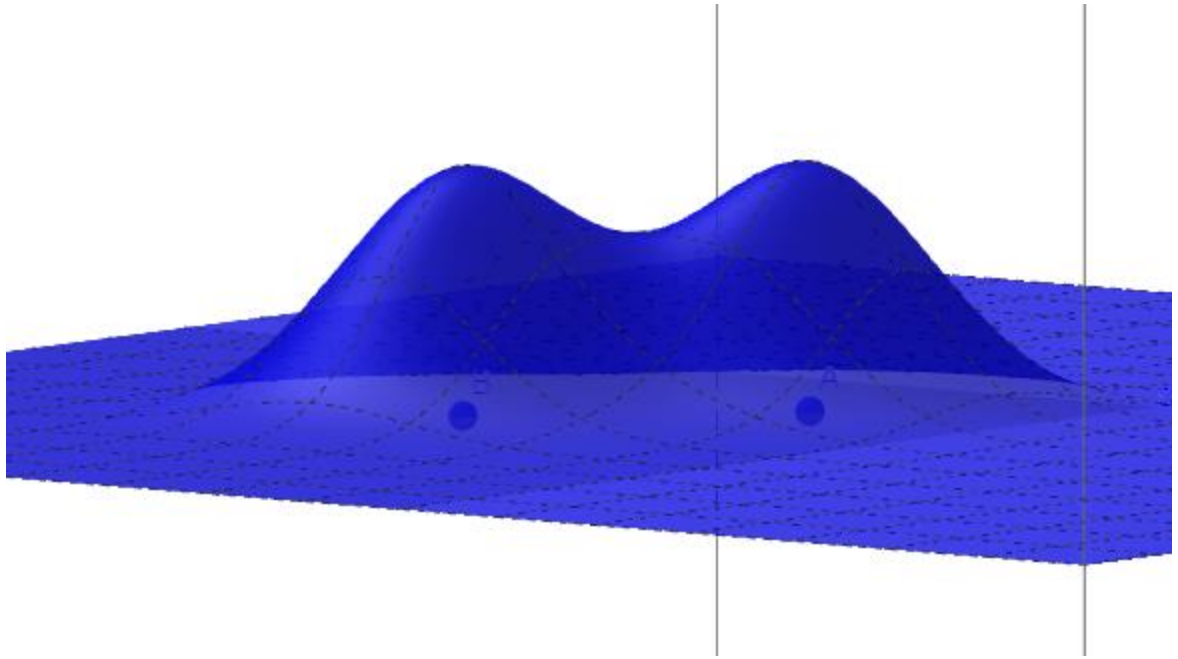


2.1. Einführung SVM – mit Kernel

- In sklearn Package sind mehere Kernel vorhanden
  - linear -> welcher nur bei linearen Trennfunktionen, ohne höher Dimensionierung, eingesetzt werden kann
  - rbf -> wir benutzt um geschlossene Bereiche z.B. Kreise oder Cluster zu erkennen
    - radial basis function
  - poly -> hinzufügen eines weiteren Merkmals, durch Wertkombinationen der vorhandenen Merkmale
    - Polynominale Kernel
    - z.B. [x,y,z] -> [x,y,z, x^3+y^2+z]

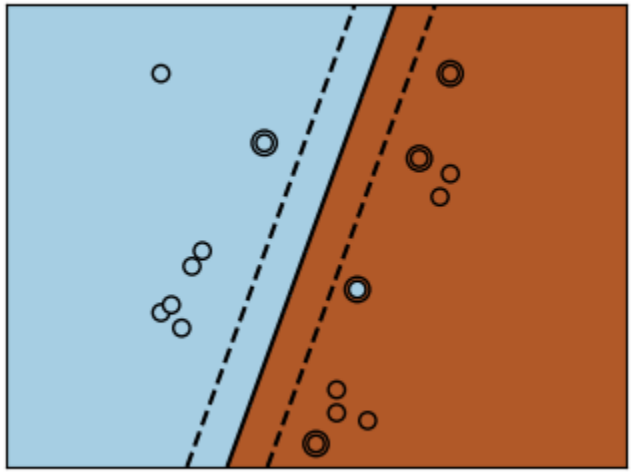
2.1. Einführung SVM – mit Kernel



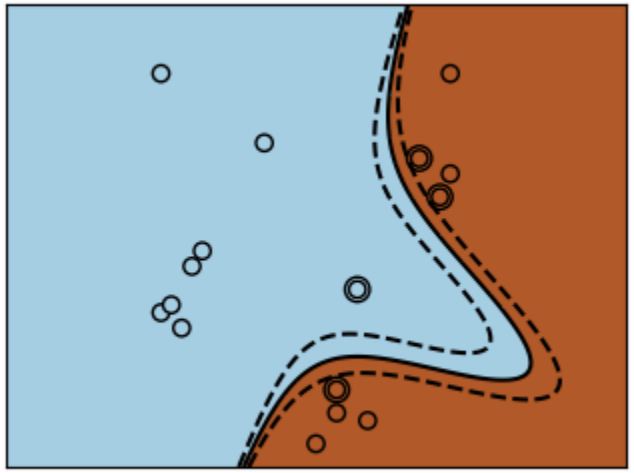


- Der rbf Kernel bildet Hügel, welche durch eine (Hyper-)Ebene linear getrennt werden können

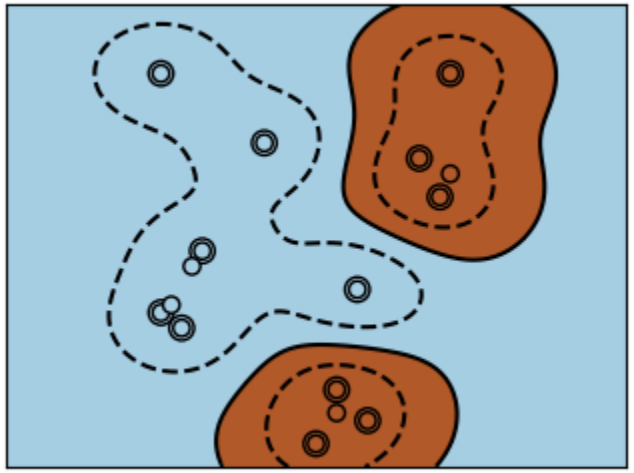
2.1. Einführung SVM – mit Kernel



linear



poly



rbf

http://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html#sphx-glr-auto-examples-svm-plot-svm-kernels-py

2.2. Beispiel – sklearn iris_Dataset (Petalum)

- Vorhandenes Beispiel abändern, beim aufrufen der Funktion

```
# hier Kernelfunktion anpassen - 'linear', 'poly' and 'rbf'
plot_data(data, labels, kernel_fn="linear")
plot_data(data, labels, kernel_fn="rbf")
plot_data(data, labels, kernel_fn="poly")
```

3.1. Anpassung – Der C-Parameter

- Auch oft als Strafparameter bezeichnet
- Je größere C gesetzt wird, desto strenger werden die falsch Klassifizierungen bestraft
  - Führt bei zu großen C zur Overfitting
- Je kleiner das C gesetzt wird, desto mehr Ausreiser werden falsch klassifiziert
  - Führt bei zu kleinem C zur Underfitting

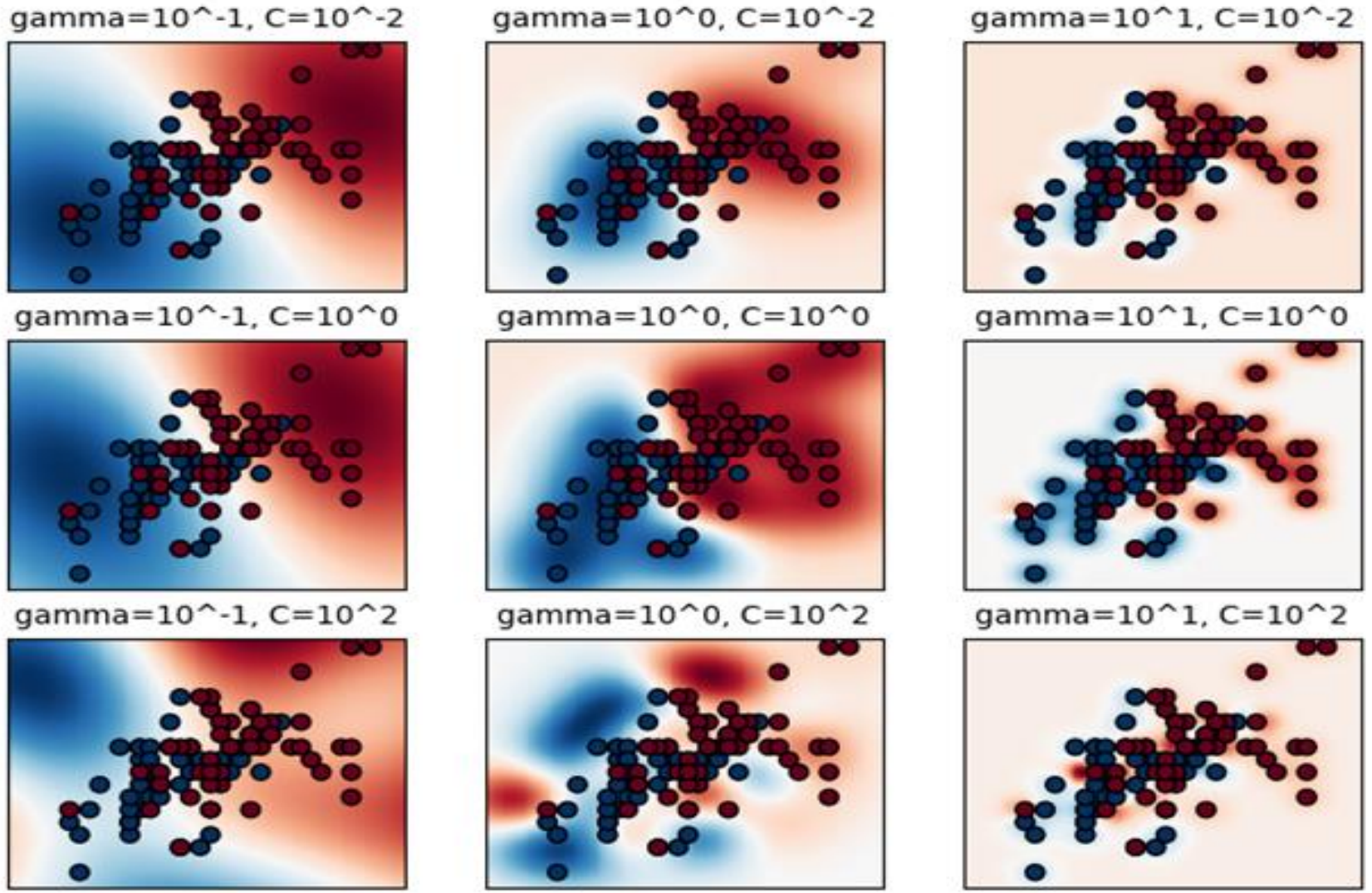
- Standartwert in Python ist auf 1 gesetzt

3.2. Anpassung – Gamma

- Wird beim rbf Kernel verwendet
- Je höher der Gamma-Wert desto höher der Hügel
  - Könnte Overfitting auftreten
- Je kleiner der Gamma-Wert desto kleiner der Hügel
  - Könnte Underfitting entstehen

- Standartwert ist 1/n-Merkmale

3.3. Übersicht verschiedenen Parameter Kombinationen



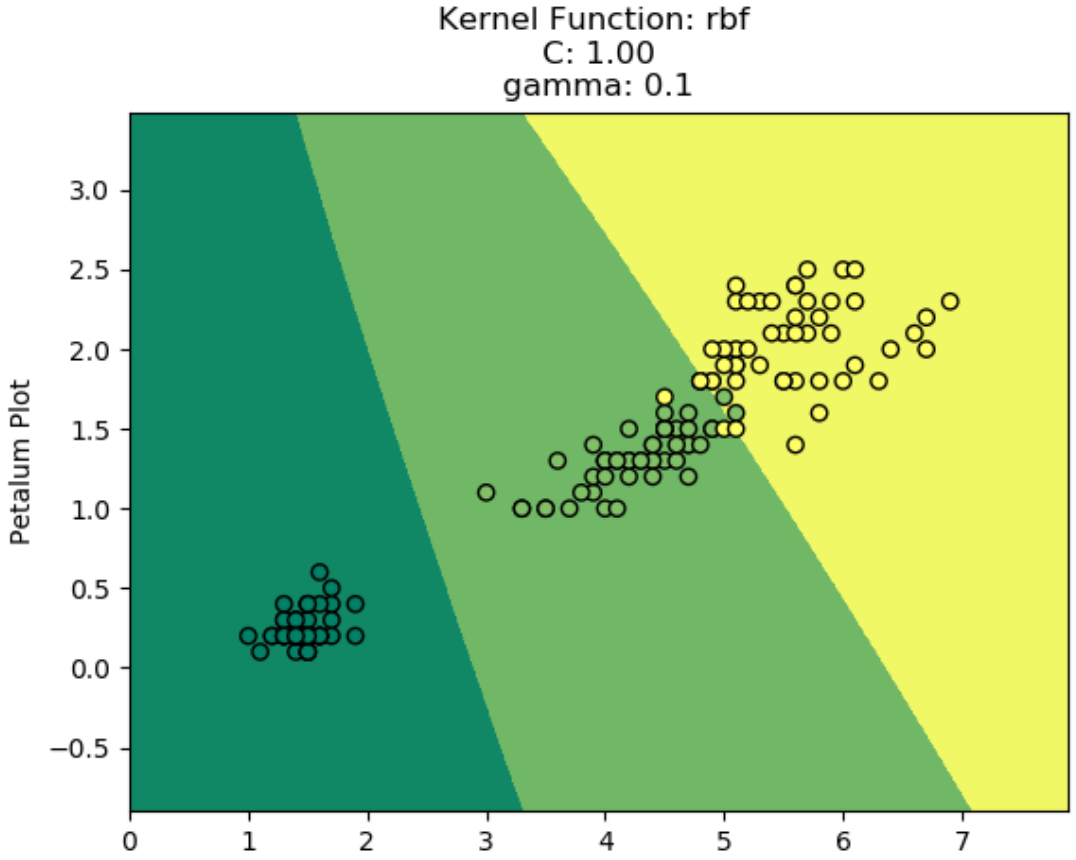
scikit-learn.org/stable/_images/sphx_glr_plot_rbf_parameters_001.png

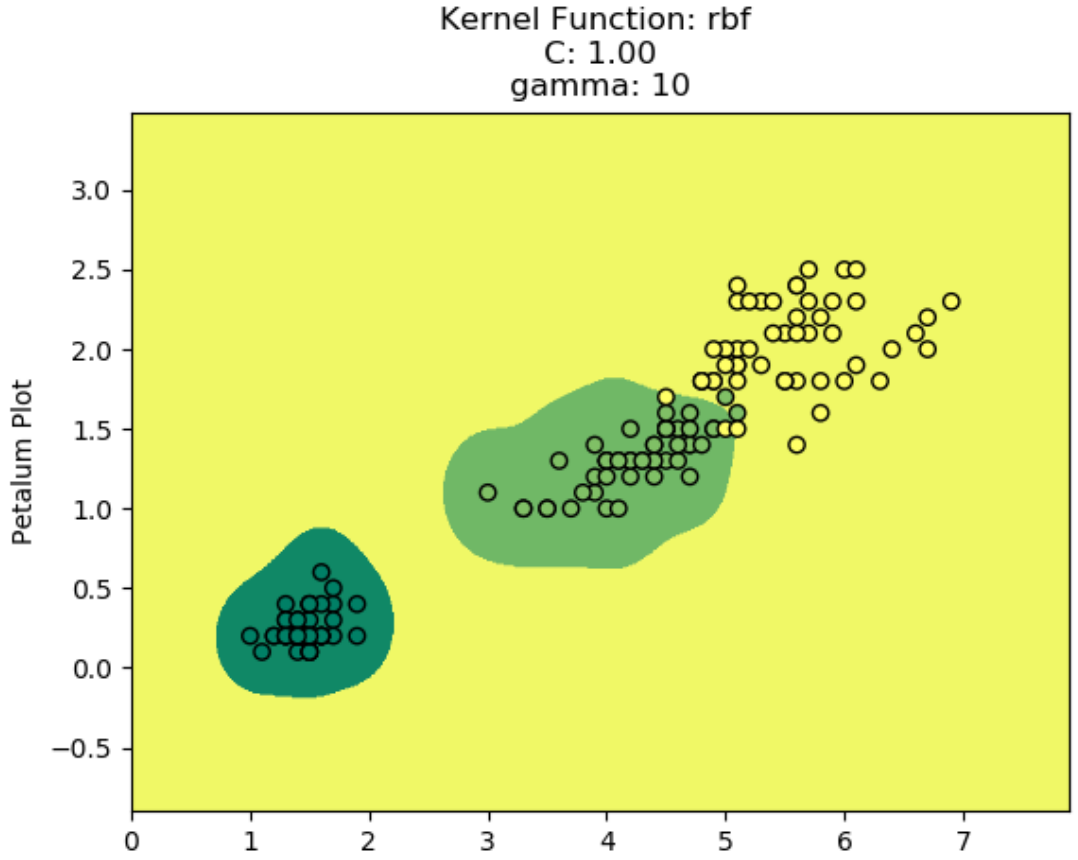
3.4. Beispiel – sklearn iris_Dataset (Petalum)

- Vorhandenes Beispiel abändern, beim aufrufen der Funktion

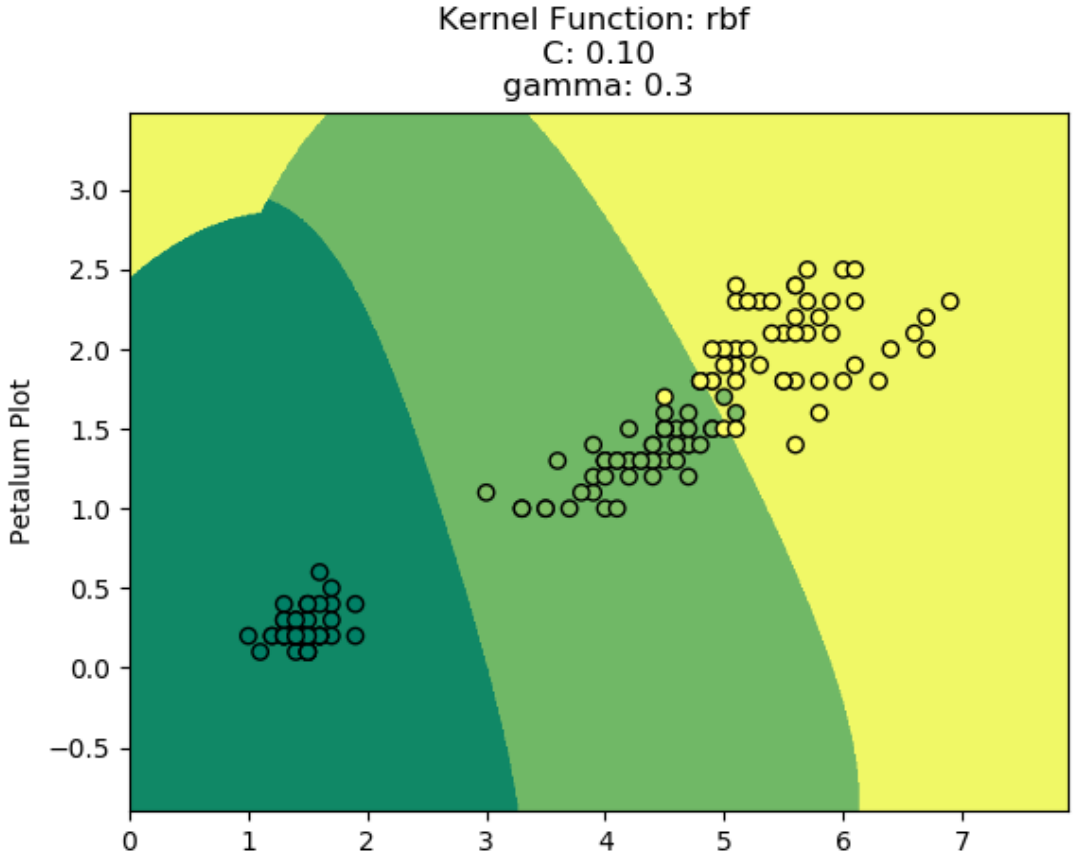
```
# verschiedene Parameter Kombinationen
plot_data(data, labels, kernel_fn="rbf", gamma=0.1, C=1)
plot_data(data, labels, kernel_fn="rbf", gamma=10, C=1)
plot_data(data, labels, kernel_fn="rbf", gamma=0.3, C=0.1)
plot_data(data, labels, kernel_fn="rbf", gamma=0.3, C=50)
```

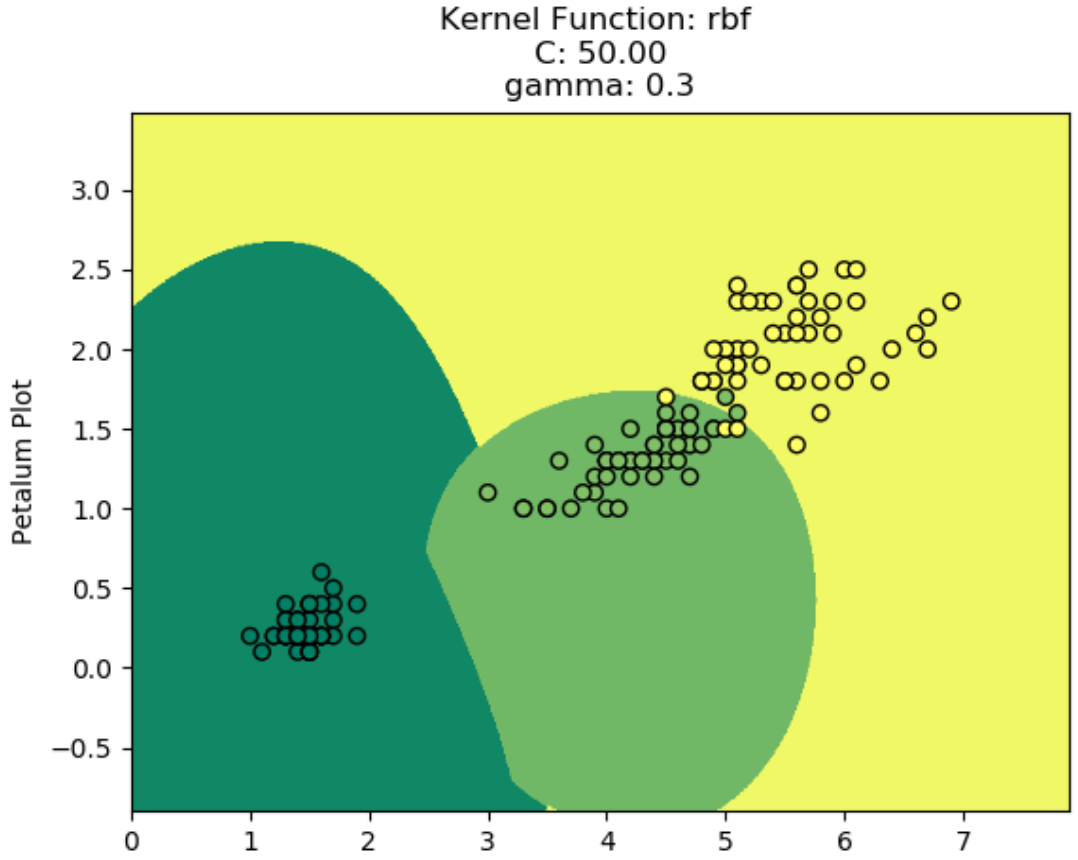
3.4. Beispiel – sklearn iris_Dataset (Petalum)





3.4. Beispiel – sklearn iris_Dataset (Petalum)





Literaturhinweise

- Folien aus der Vorlesung CI1
  - CI1_06_SVM_StÃ¼tzvektormethode
- https://blog.ancud.de/home/-/blogs/einfuhrung-in-machine-learning-mit-python-support-vector-machines
- http://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html#sphx-glr-auto-examples-svm-plot-svm-kernels-py