

SVM mit Kernel-Trick von \mathbb{R}^2 (2D) nach \mathbb{R}^3 (3D)

Ich möchte Jupyter Notebook (Python) schreiben für ein einfaches und anschauliches Support Vector Machine (SVM) Beispiel mit einem Kernel-Trick (2D nach 3D). Hier die Details: Gegeben sind die folgenden Datenpunkte: Klasse +1: A(1,1), B(-1,1) und Klasse -1: C(2,1), D(1,-2) und E(-2,1) in \mathbb{R}^2 . Mache eine Kernel Transformation von 2D nach 3D mit (x,y) wird nach (x, y, x^2+y^2) transformiert (x und y sind dabei die Koordinaten von 2D). Wie sieht die Hyperebene in 3D aus (plotten, animiert, mit Datenpunkten) und wie sieht die Rücktransformation in 2D aus (Plot mit Datenpunkten). Es kommt ein Kreis mit $(\text{Radius})^2 = 3,5$ raus.

Author/Datum: H.Völlinger, DHBW Stuttgart / 12.10.2023

Um ein einfaches und anschauliches Beispiel für eine Support Vector Machine (SVM) mit einem Kernel-Trick von 2D nach 3D in einem Jupyter Notebook zu erstellen, können Sie den folgenden Python-Code verwenden. Der Code verwendet die scikit-learn Bibliothek für die SVM und die matplotlib Bibliothek zum Plotten. Zunächst importieren Sie die erforderlichen Bibliotheken:

```

In [1]: import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import svm

# Datenpunkte definieren
X = np.array([[1, 1], [-1, 1], [2, 1], [1, -2], [-2, 1]])
y = np.array([+1, +1, -1, -1, -1])

# Transformation von 2D nach 3D
X_3D = np.array([[x, y, x ** 2 + y ** 2] for x, y in X])

# SVM-Modell erstellen
clf = svm.SVC(kernel='linear')
clf.fit(X_3D, y)

# Get the coefficients of the hyperplane
coefficients = clf.coef_[0]
bias = clf.intercept_[0]

# Berechnen der Entscheidungsebene in 3D
xx, yy = np.meshgrid(np.linspace(-3, 3, 100), np.linspace(-3, 3, 100))
zz = (-coefficients[0] * xx - coefficients[1] * yy - bias) / coefficients[2]

# Datenpunkte in 3D plotten
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_3D[y == 1][:, 0], X_3D[y == 1][:, 1], X_3D[y == 1][:, 2], color='blue', 1
label='Klasse +1', s=50)
ax.scatter(X_3D[y == -1][:, 0], X_3D[y == -1][:, 1], X_3D[y == -1][:, 2], color='red',
label='Klasse -1', s=50)
ax.plot_surface(xx, yy, zz, alpha=0.5, color='gray')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Hyperebene {(x,y,z) in R^3 | z=3.5} mit Datenpunkten A*, B*, C*, D* und E
*')

# Inverse Transformation von 3D nach 2D
X_2D = np.array([[x, y] for x, y, z in X_3D])

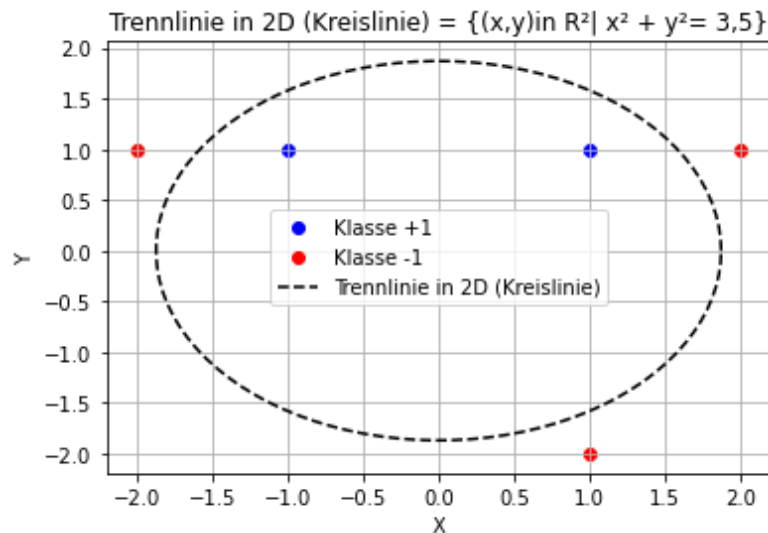
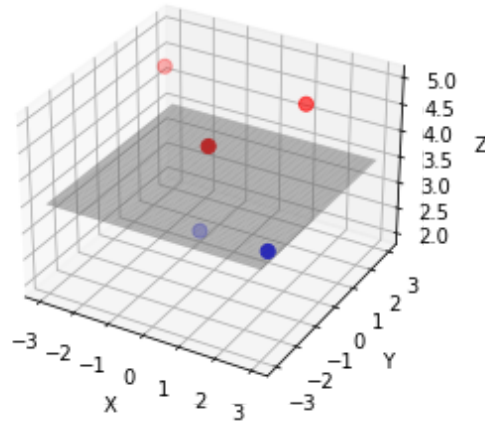
# Kreislinie in 2D erstellen
theta = np.linspace(0, 2 * np.pi, 100)
radius = np.sqrt(3.5)
x_circle = radius * np.cos(theta)
y_circle = radius * np.sin(theta)

# Datenpunkte in 2D plotten
plt.figure()
plt.scatter(X_2D[y == 1][:, 0], X_2D[y == 1][:, 1], color='blue', label='Klasse +1')
plt.scatter(X_2D[y == -1][:, 0], X_2D[y == -1][:, 1], color='red', label='Klasse -1')
plt.plot(x_circle, y_circle, 'k--', label='Trennlinie in 2D (Kreislinie)')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Trennlinie in 2D (Kreislinie) = {(x,y) in R^2 | x^2 + y^2 = 3,5}')
plt.legend()
plt.grid(True)

plt.show()

```

Hyperebene $\{(x,y,z) \in \mathbb{R}^3 \mid z=3.5\}$ mit Datenpunkten A*, B*, C*, D* und E*



Ja, das ist korrekt. Der Normalenvektor der SVM-Ebene in 3D ist $(0.0, 0.0, 1.0)$, was bedeutet, dass die Ebene senkrecht zur z-Achse steht. Die Gleichung der Ebene lautet $0x + 0y + 1z = 3.5$, was die optimale Trennebene für die gegebenen Punkte darstellt.

Dieser Code berechnet und zeigt die Punkte A', B', C', D' und E' in 3D und verwendet blaue bzw. rote Markierungen, um die Klassenzugehörigkeit anzuzeigen.

```
In [2]: # Print the transformed points in 3D
print("Transformed Points in 3D:")
# \033[94m und \033[0m ändern die Textfarbe auf Blau und setzt anschließend auf Standardfarbe zurück.
print("\033[94mKlasse +1: A*= Point 1, B*= Point 2\033[0m")
# \033[91m und \033[0m ändern die Textfarbe auf Rot und setzt anschließend auf Standardfarbe zurück.
print("\033[91mKlasse -1: C*= Point 3, D*= Point 4, E*= Point 5\033[0m")
for i, point in enumerate(X_3D):
    print(f"Point {i + 1}: {point}")
```

Transformed Points in 3D:

Klasse +1: A*= Point 1, B*= Point 2

Klasse -1: C*= Point 3, D*= Point 4, E*= Point 5

Point 1: [1 1 2]

Point 2: [-1 1 2]

Point 3: [2 1 5]

Point 4: [1 -2 5]

Point 5: [-2 1 5]

In []: