Machine Learning 2: Klassifikation

Ergänzungsfach Informatik, 2021/2022, pro@kswe.ch

19. November 2021

Lernziele:

- Sie erklären das Prinzip folgender Klassifikationsverfahren an einem Beispiel:
  - k-Nearest-Neightbor (inkl. mathematische Berechnung der nächsten Nachbarn)
  - Nearest Centroid Classifier (inkl. mathematische Berechnung des nächsten Centroid)
  - SVM
- Sie implementieren ein Klassifikationsverfahren mit einem Daten-Set in Python.

1 Was ist Klassifikation?

Bei der Klassifikation möchte man ein neues Sample einer Kategorie zuordnen. Beim Supervised Learning sind die Kategorien vorgegeben und werden Klassen genannt. Das Daten-Set besteht somit aus einer Menge von Features und einer Menge von Klassen. Jedes Sample besitzt neben den Features auch eine Klasse. Klassifikationsverfahren aus dem Machine Learning-Bereich versuchen nun für ein neues Sample (ohne Klasse) die zugehörige Klasse automatisch zu bestimmen.

Beispiele: Es gibt verschiedene Einsatzgebiete, einige Beispiele: Medizinische Bildgebung, Spracherkennung, Handschrifterkennung, Artbestimmung (Tiere und Pflanzen).

- Teile eine E-Mail in Spam oder nicht Spam ein (2 Klassen)
- Entscheide, ob ein Tumor gutartig oder bösartig ist (2 Klassen)
- Bestimme für eine Pflanze die Pflanzenart (meist mehr als 2 Klassen)

Wir schauen uns nun verschiedenen Klassifikationsverfahren an.

2 k-Nearest-Neighbor

Prinzip: Klassifiziere ein neues Sample durch das Betrachten der k nächsten Nachbarn.

Abstraktes Beispiel: Folgendes Daten-Set ist gegeben. Zu welcher Klasse soll das neue Sample $(3, 7)$ gehören?

Feature 1	Feature 2	Klasse
7	7	K1
7	4	K1
3	4	K2
1	4	K2

2.1 Sample als Vektor

Wir betrachten einen Sample als Vektor. Die Klasse für jeden Sample notieren wir separat. Dadurch können wir die "nächsten Nachbarn" einfacher berechnen.

Beispiel (Fortsetzung): Wir erhalten folgendes Daten-Set. Jeder Sample hat Dimension 2.

$$\vec{x_1} = \begin{bmatrix} 7 \\ 7 \end{bmatrix} \quad \vec{x_2} = \begin{bmatrix} 7 \\ 4 \end{bmatrix} \quad \vec{x_3} = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \quad \vec{x_4} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

2.2 Nächste Nachbarn

Man muss nun festlegen, wie man die nächsten Nachbarn für ein neues Sample $\vec{x^*}$ bestimmt. Meist verwendet man hier die euklidische Distanz zwischen zwei Vektoren:

$$d(\vec{x_1}, \vec{x_2}) = ||\vec{x_2} - \vec{x_1}||_2 = \sqrt{(x_{21} - x_{11})^2 + (x_{22} - x_{12})^2 + \cdots + (x_{2n} - x_{1n})^2}$$

$$\vec{x_1} = \begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{1n} \end{bmatrix} \qquad \vec{x_2} = \begin{bmatrix} x_{21} \\ x_{22} \\ \vdots \\ x_{2n} \end{bmatrix}$$

Man berechnet nun zwischen $\vec{x^*}$ und allen Vektoren des Sample-Sets die euklidische Distanz. Dann ermittelt man ein Ranking. Man bestimmt für die k kleinsten Distanzen die Klasse und gibt x^* diejenige Klasse, die am häufigsten vorkommt.

Beispiel (Fortsetzung): Wir berechnen nun die Distanzen und klassifizieren den Sample $\overrightarrow{x^*} = \begin{bmatrix} 3 \\ 7 \end{bmatrix}$ mit $k = 3$.

Sample	$d(\overrightarrow{x_i}, \overrightarrow{x^*}) = \|\|\overrightarrow{x^*} - \overrightarrow{x_i}\|\|_2$ für $i = 1, 2, 3, 4$	Ranking	Gehört das Sample zu den 3 nächsten Nachbarn?	Klasse des nächsten Nachbarn
$\overrightarrow{x_1} = \begin{bmatrix} 7 \\ 7 \end{bmatrix}$	$d(\overrightarrow{x_1}, \overrightarrow{x^*}) = \|\| \begin{bmatrix} 3 \\ 7 \end{bmatrix} - \begin{bmatrix} 7 \\ 7 \end{bmatrix} \|\|_2 = \|\| \begin{bmatrix} -4 \\ 0 \end{bmatrix} \|\|_2 = \sqrt{(-4)^2 + 0^2} = \sqrt{16} = 4$	3	Ja	K1
$\overrightarrow{x_2} = \begin{bmatrix} 7 \\ 4 \end{bmatrix}$	$d(\overrightarrow{x_1}, \overrightarrow{x^*}) = \|\| \begin{bmatrix} 3 \\ 7 \end{bmatrix} - \begin{bmatrix} 7 \\ 4 \end{bmatrix} \|\|_2 = \|\| \begin{bmatrix} -4 \\ 3 \end{bmatrix} \|\|_2 = \sqrt{(-4)^2 + 3^2} = \sqrt{25} = 5$	4	Nein	(wird nicht benutzt)
$\overrightarrow{x_3} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$	$d(\overrightarrow{x_1}, \overrightarrow{x^*}) = \|\| \begin{bmatrix} 3 \\ 7 \end{bmatrix} - \begin{bmatrix} 3 \\ 4 \end{bmatrix} \|\|_2 = \|\| \begin{bmatrix} 0 \\ 3 \end{bmatrix} \|\|_2 = \sqrt{0^2 + 3^2} = \sqrt{9} = 3$	1	Ja	K2
$\overrightarrow{x_4} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$	$d(\overrightarrow{x_1}, \overrightarrow{x^*}) = \|\| \begin{bmatrix} 3 \\ 7 \end{bmatrix} - \begin{bmatrix} 1 \\ 4 \end{bmatrix} \|\|_2 = \|\| \begin{bmatrix} 2 \\ 3 \end{bmatrix} \|\|_2 = \sqrt{2^2 + 3^2} = \sqrt{4+9} = \sqrt{13} \approx 3,6$	2	Nein	K2

$\overrightarrow{x^*}$ erhält die Klasse K2, da die 3 nächsten Nachbarn mehrheitlich K2 als Klasse besitzen.

3 KNN mit dem Iris Daten-Set

Wir werden nun das KNN-Verfahren auf das Iris-Daten-Set anwenden. Wir möchten eine noch nicht klassifizierte Iris-Pflanze einer der drei folgenden Klassen zuordnen:

- Iris versicolor (Verschiedenfarbige Schwertlilie)
- Iris virginica (Blaue Sumpfschwertlilie)
- Iris setosa (Borsten-Schwertlilie)

Für jede Iris-Pflanze werden folgende Eigenschaften (Features) bestimmt:

- Sepal Length (Kelchblattlänge)
- Sepal Width (Kelchblattbreite)
- Petal Length (Kronblattlänge)
- Petal Width (Kronblattbreite)

Alle Angaben sind in Zentimeter. Sie werden nun Schritt-für-Schritt in Python mit dem Daten-Set arbeiten und das KNN-Verfahren implementieren.

3.1 Vorbereitungen

1. Erstellen Sie in Ihrem Machine Learning Projekt einen neuen Ordner `k_nearest_neighbor`.
2. Laden Sie die CSV-Datei `iris.csv` aus Teams herunter und verschieben Sie die Datei in den Ordner `klassifikation`.
3. Erstellen Sie im Ordner `k_nearest_neighbor` eine neue Python-Datei `knn_iris.py`.
4. Installieren Sie die Python-Packages `pandas` und `matplotlib`.

3.2 CSV-Daten mit pandas einlesen

1. Importieren Sie das `pandas` Modul.

```
1 import pandas as pd
```

2. pandas arbeitet mit `DataFrame`s. Es eignet sich besonders gut, wenn man tabellarische Daten hat (wie z.B. eine Exceltabelle, eine Datenbanktabelle oder eine CSV-Datei[1]). Ein `DataFrame` ist eine Python-Klasse und kann man sich grafisch wie in Abbildung 1 vorstellen.

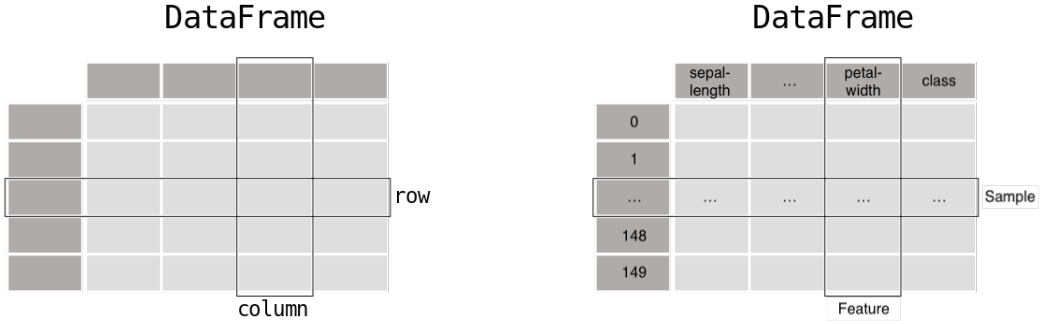


Abbildung 1: Pandas `DataFrame` besteht aus Zeilen und Spalten. Die rechte Grafik zeigt das `DataFrame` im Iris-Daten-Set-Kontext. Die Zahlen ganz rechts identifizieren jede Zeile eindeutig (Index).

[1]Eine CSV-Datei ist eigentlich nichts anders als eine Tabelle. Spalten werden einfach durch Kommas getrennt. Eine neue Zeile in der Datei entspricht einer Zeile der Tabelle.

Man kann nun mit pandas direkt die Daten aus einer CSV-Datei einlesen. Es wird dann automatisch ein `DataFrame`-Objekt erstellt. Man kann dann Attribute und Methoden verwenden. Lesen Sie nun das Iris-Daten-Set ein und vervollständigen Sie die rechte Grafik aus Abbildung 1 mit konkreten Werten. Notieren Sie mit Kommentaren die Methoden `head`, `tail` und `describe`.

```
import pandas as pd

column_names = ["sepal-length", "sepal-width", "petal-length",
        "petal-width", "class"]
# Lade die Daten aus der CSV-Datei iris.csv in ein DataFrame.
# Verwende die Spaltennamen aus column_names,
# da die CSV-Datei keine Spaltennamen besitzt.
iris_data_set = pd.read_csv("iris.csv", names=column_names,
        header=None)
print(iris_data_set.head())
print(iris_data_set.tail())
print(iris_data_set.describe())
```

Unter `pandas.pydata.org` finden Sie weitere Informationen zum Python-Package.

3.3 Daten-Set aufbereiten

1. Teilen Sie das Daten-Set auf. Eine Matrix beinhaltet nur die Features aller 150 Samples. Ein Vektor beinhaltet die Klassen der Samples.

```
# Die ersten vier Spalten bilden die Features.
# Wir speichern alle Zeilen in einer Matrix ab.
matrix_X = iris_data_set.iloc[0:150, 0:4].values
# Die letzte Spalte beinhaltet die Klassen. Dies ist ein Vektor.
vektor_y = iris_data_set.iloc[0:150, 4].values
```

2. Wir möchten das KNN-Verfahren mit $k = 5$ ausprobieren, das heisst, wir betrachten die 5 nächsten Nachbarn. Wie können wir nun überprüfen, wie gut das Verfahren funktioniert? Wir haben ja keine "neuen" (noch nicht gesehenen) Samples. Man behilft sich dabei mit einem "Trick". Wir teilen das Daten-Set in zwei Teile auf: **Trainingsdaten** und **Testdaten**. Es gilt:

   Wichtig!. *Folgende Regeln sollten eingehalten werden:*

   - *Das Daten-Set wird nach der* 80%-20%*-Regel aufgeteilt:* 80% *der Samples bilden die Trainingsdaten,* 20% *die Testdaten.*
   - *Man trainiert das Modell mit den Trainingsdaten (und* ***nur*** *mit den Trainingsdaten). Trainieren bedeutet das Erstellen des Modells.*
   - *Man testet das Modell mit den Testdaten (und* ***nur*** *mit den Testdaten). Testen bedeutet man bestimmt die Genauigkeit (Accuracy) des Modells.*

   Beim Testen macht man dann die Vorhersagen für die Samples aus den Testdaten. "Man tut zunächst so, als kenne man die tatsächlichen Klassen/Labels nicht". Dann vergleicht man das Resultat des Verfahrens mit den tatsächlichen Klassen/Labels und kann daraus eine Genauigkeit bestimmen.

   Fragen:

   (a) Wie erfolgt das Lernen bei KNN? Das Lernen erfolgt einfach dadurch, in dem alle Samples aus den Trainingsdaten abgespeichert werden.

   (b) Wie erfolgt das Lernen bei der linearen Regression? Man bestimmt die Gewichte durch das Lösen des Gleichungssystems.

Teilen Sie das Iris-Daten-Set nun auf. Sie können mit `sci-kit` das Aufteilen automatisch durchführen. Importieren Sie zunächst das passende Modul.

```
import sklearn.model_selection as modsel
```

Rufen Sie nun die Funktion auf, um die Trainings- und Testdaten zu erstellen.

```
# Teilt das Daten-Set in Trainingsdaten und Testdaten auf.
train_X, test_X, train_y, test_y = modsel.train_test_split(
        matrix_X, vektor_y, test_size=0.20)
```

3.4 Lernen

Die Lernphase bei KNN ist recht simpel. Eigentlich gibt es keine "echte" Lernphase in der wir ein Modell der Daten erlernen.

Erklärung 3.1 (Lernphase bei KNN)**.** *Das Lernen bei KNN besteht aus dem Speichern aller Samples der Trainingsdaten. Die Lernphase wird auch Trainingsphase genannt. Da KNN erst beim Vorhersagen "richtig" aktiv wird, gehört KNN zur Kategorie der "lazy learners".*

1. Bauen Sie nun die Lernphase in das Programm ein. Wir lernen mit den Trainingsdaten!

```
import sklearn.neighbors as neigh

# KNN mit k=5 (betrachte die 5 naechsten Nachbarn)
classifier = neigh.KNeighborsClassifier(n_neighbors=5)
classifier.fit(train_X, train_y)
```

3.5 Testen

Nach der Lernphase folgt die Testphase. In dieser Phase werden die Vorhersagen gemacht. Man verwendet die Testdaten und berechnet, wie gut das Modell die Vorhersagen bestimmen kann.

Erklärung 3.2 (KNN Testen)**.** *Bei KNN wird in der Testphase für jedes Sample aus den Testdaten eine Klasse bestimmt. Es wird eine Vorhersage gemacht (prediction).*

1. Bestimmen Sie nun für die Testdaten die Vorhersagen. Wir testen mit den Testdaten!

```
# Benutze die Testdaten fuer die Vorhersagen
pred_y = classifier.predict(test_X)
```

3.6 Modell bewerten

Wie gut sind die Vorhersagen? Wir vergleichen nun die Vorhersagen des Verfahrens mit den tatsächlichen (und bis jetzt zurückgehaltenen) Klassen/Labels. Daraus können wir verschiedene Bewertungen ableiten. Mit den Bewertungen können wir Aussagen darüber machen, wie gut das Verfahren "funktioniert". Ein Verfahren zur Messung der Qualität eines Verfahrens wird Metrik genannt. Wir können verschiedene Metriken berechnen und analysieren.

1. Importieren Sie das Modul zur Berechnung von Metriken.

```
import sklearn.metrics as met
```

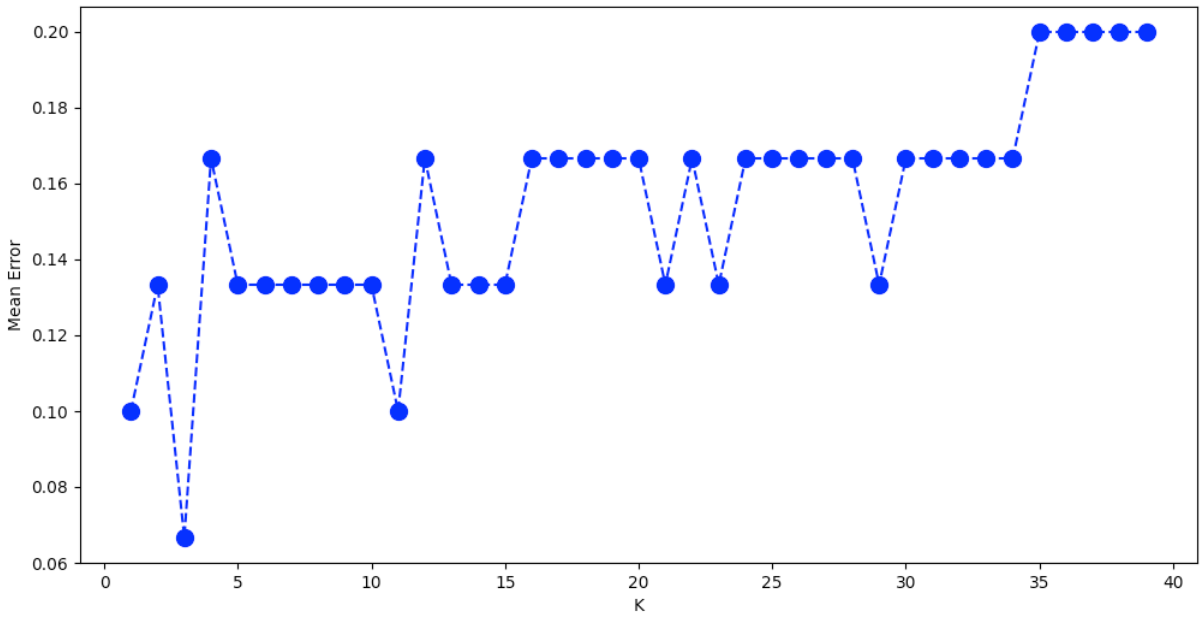
2. Ermitteln Sie die typischen Metriken für die Klassifizierung.

```
# Wahrheitsmatrix (Konfusionsmatrix) bestimmen
print(met.confusion_matrix(test_y, pred_y))
# Bestimme Precision, Recal, F1-Score, Support und Accuracy.
print(met.classification_report(test_y, pred_y))
```

3.7 Wie wählt man k?

Wir haben bisher den Wert für k bei KNN "einfach" auf 5 gesetzt. Ist dies eine optimale Wahl? Wie kann man dies bestimmen? Versuchen Sie folgenden Code nachzuvollziehen. Betrachten Sie auch den Plot.

```
1  errors = []
2  # Calculating error for K values between 1 and 40
3  for i in range(1, 40):
4      knn = neigh.KNeighborsClassifier(n_neighbors=i)
5      knn.fit(train_X, train_y)
6      pred_i = knn.predict(text_X)
7      error = []
8      for p in range(len(pred_i)):
9          if pred_i[p] == test_y[p]:
10             error.append(0)
11         else:
12             error.append(1)
13     errors.append(sum(error) / len(error))
14
15 plt.figure(figsize=(12, 6))
16 plt.plot(range(1, 40), errors, color="blue", linestyle="dashed",
17         marker="o", markerfacecolor="blue", markersize=10)
18 plt.xlabel("K")
19 plt.ylabel("Mean Error")
20 plt.show()
```



Der Code geht verschiedene Werte für k in einer for-Schleife durch. Pro k wird ein KNN Klassifizierer erstellt (`neigh.KNeighborsClassifier(n_neighbors=i)`). Dann werden mit den Testdaten die Vorhersagen für alle Samples gemacht (`test_X`). In `pred_i` sind die Vorhersagen gespeichert. Diese werden nun in einer for-Schleife mit den tatsächlichen Klassen verglichen. Stimmen die Klassen überein, dann wird in die Liste der Fehler (`error`) eine 0 hinzugefügt, sonst eine 1. Abschliessend wird der Durchschnitt ermittelt (Zeile 13) und in die Liste der durchschnittlichen Fehler (`errors`) hinzugefügt. Der Plot zeigt genau diese durchschnittlichen Fehler. Für jedes k von 1 bis 39 gibt es einen Wert (blauer Punkt im Plot). Man sieht im Plot, dass die Wahl von $k = 3$ vielleicht noch besser ist. Dies hängt jedoch auch von der Aufteilung von Trainingsdaten und Testdaten ab. Man sollte diesen Vorgang auch noch mehrmals wiederholen, um die Wahl von k zu verifizieren.

4 Zentrum pro Klasse bilden

Besuchen Sie den User Guide von scikit-learn (`scikit-learn.org`). Finden Sie ein Verfahren, welches die Klassifikation nach folgenden Punkten durchführt:

- "Zentrum" pro Klasse bilden.
- Ein neues Sample wird dem nächsten Zentrum zugeordnet.

Lösungsvorschlag: URL:
`https://scikit-learn.org/stable/modules/neighbors.html#nearest-centroid-classifier`
⇒ **Nearest Centroid Classifier**.

Beantworten Sie dann folgende Fragen. Suchen Sie ggf. via Google nach den Antworten.

1. Wie wird das "Zentrum" berechnet?

   Lösungsvorschlag: Man addiert alle Samples einer Klasse und dividiert durch die Anzahl der Samples in dieser Klasse. Dies gibt pro Klasse einen "Zentrumsvektor". Formaler:

$$\vec{z}_{klasse\_i} = \frac{\sum_1^n \vec{x_i}}{n}$$

2. Wie wird die Vorhersage durchgeführt? Wie berechnet man das nächste Zentrum?

   Lösungsvorschlag: Für ein neues Sample wird die Distanz zu allen Zentren berechnet, wie bei K-Nearest-Neighbor auch. Der neue Sample erhält dann diejenige Klasse, die das nächste Zentrum besitzt.

3. Wie wird das Verfahren noch genannt? In welchem Bereich wird es (gerne) eingesetzt?

   Lösungsvorschlag: Es wird auch Rocchio Algorithmus genannt und wird im Information Retrieval eingesetzt. Hier geht es darum, für eine Suchanfrage die relevanten Dokumente zu ermitteln.

Erstellen Sie ein Python-Programm, welches die Vorhersage für das Iris-Daten-Set durchführt. Sie können den KNN-Code wiederverwenden.
Lösungsvorschlag: Sie finden das Programm in der Python-Datei `nearest_centroid.py`

5 Konfusionsmatrix, Precision, Recall und Accuracy

Mit der Konfusionsmatrix können Sie Precision, Recall und Accuracy berechnen. Diese geben Aufschluss über die Qualität der Vorhersagen. Nehmen wir an es gibt 6 Samples und drei Klassen mit folgenden Zuordnungen und Vorhersagen.

	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Sample 6
Tatsächliche Klasse	ant	ant	cat	cat	ant	cat
Vorhergesagte Klasse	cat	ant	cat	cat	ant	bird

Die Konfusionsmatrix gibt das Ergebnis von korrekten und falschen Vorhersagen kompakt an.

		Tatsächliche Klassen		
		ant	bird	cat
Vorhergesagte Klassen	ant	2	0	0
	bird	0	0	1
	cat	1	0	2

Man kann Precision, Recall und Accuracy nun daraus berechnen. Für uns ist Accuracy (Richtigkeit) relevant: $\frac{\text{Summe der Diagonaleinträge}}{\text{Anzahl Samples}}$. Beispiel: $\frac{4}{6} \approx 0,67$. Die Vorhersage ist somit zu 67% korrekt.

6 Support Vector Machine (SVM)

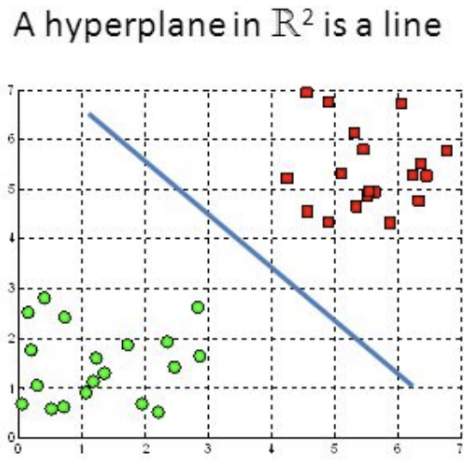
Prinzip: Klassifiziere ein neues Samples durch die Lage zur "Trennlinie". Je nachdem ob sich das Sample oberhalb oder unterhalb der Trennlinie befindet, wird die entsprechende Klasse vergeben.

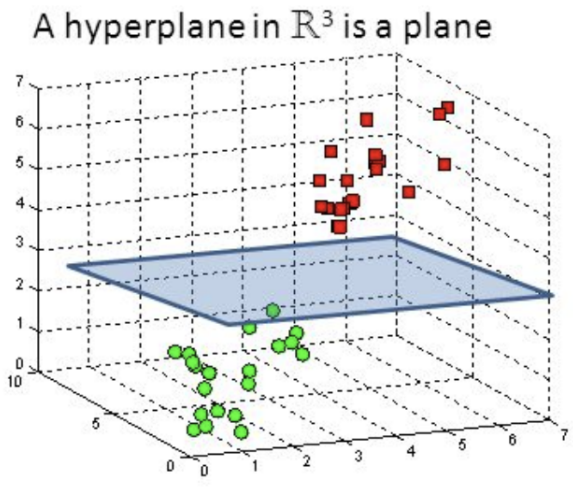
Beispiele: Abbildung 2 zeigt ein Beispiel mit zwei Klassen ("grün" und "blau") und 2-D Samples.

Abbildung 2: Das neue Sample (Kreis mit Fragezeichen) erhält die Klasse "grün", da es oberhalb der "Trennlinie" (gestrichelte Linie) liegt.

Die SVM ist keine Maschine im Sinne von Computer, sondern ein Klassifikationsverfahren, welches für ein Daten-Set die optimale "Trennlinie" ermittelt. "Machine" ist deshalb ein Verweis auf Machine Learning.

Erklärung 6.1 (Hyperebene)**.** *Der Fachbegriff für die "Trennlinie" lautet Hyperebene. Im 2-D entspricht die Hyperebene einer Geraden, im 3-D ist die Hyperebene eine "einfache" Ebene. Bei mehr als drei Dimensionen spricht man dann nur noch von einer Hyperebenen. Die Hyperebene soll die Samples in zwei Bereiche trennen.*





In der Lernphase ermittelt die SVM nun die optimale Hyperebene. Was ist die optimale Hyperebene?

Erklärung 6.2 (Optimale Hyperebene)**.** *Die optimale Hyperebene klassifiziert alle Samples korrekt und maximiert den Abstand zu den Samples. Der Abstand ist die Distanz zwischen Hyperebene und dem nächsten Punkt zur Hyperebene.*

Diese Wahl klingt plausibel, da dadurch neue Samples weniger häufig auf die "falsche" Seite der Hyperebene "fallen". Dadurch sollten weniger Fehler bei der Klassifizierung entstehen. Die optimale Hyperebene wird somit durch die nächsten Samples bestimmt. Die Samples stellen im mehrdimensionalen Raum einen Vektor dar. Deshalb werden die Samples, welche der Hyperebene am nächsten sind **Support-Vektoren** genannt. Diese Support-Vektoren (eng. support vectors) geben dem Verfahren den Namen. Manchmal liest man auch von "Stützvektoren". Für jede Klasse gibt es mindestens einen Support-Vektor. Abbildung 3 zeigt die Begriffe an einem Beispiel-Daten-Set (2-D).

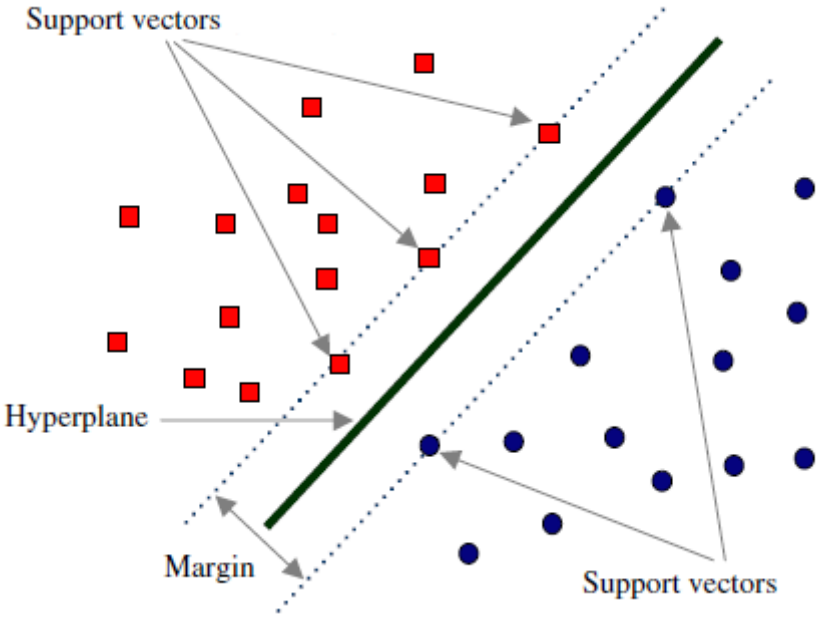


Abbildung 3: Die Hyperebene liegt in der Mitte zwischen den Support-Vektoren der beiden Klassen.

6.1 Hyperebene

Bei einem Daten-Set mit n-dimensionalen Samples kann man die Hyperebene H wie folgt angeben:

$$w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n + b = 0$$

oder in Vektorschreibweise:

$$\vec{w} \cdot \vec{x} + b = 0$$

Diese Schreibweise ist sehr ähnlich zur Hesseschen Normalform für Ebenen. Der Vektor $\vec{w}$ ist der Gewichtsvektor (eng. weight vector) und steht senkrecht zur Hyperebene. Die SVM bestimmt nun die optimale Hyperebene, in dem die Werte für den Vektor $\vec{w}$ und der Wert für b bestimmt wird.

6.2 Kernel-Trick

Nicht immer sind die Daten linear trennbar, das heisst durch eine Hyperebene in zwei Regionen aufteilbar. Abbildung 4 zeigt zwei Beispiele. Hyperebenen bilden einen linearen Klassifizierer. Es werden nur lineare Terme verwendet. Normalerweise funktioniert die SVM nur dann, wenn das Daten-Set linear trennbar ist. Man kann jedoch einen Trick anwenden, der als Kernel-Trick bekannt ist. Dazu werden die Samples in einen höheren Raum abgebildet. Die Abbildung in diesen Raum wird durch eine Kernelfunktion durchgeführt. Man erhält dabei neue, "künstliche" Features die durch die Anwendung der Kernelfunktion entstehen. Man erhofft sich dann, dass die Samples in einem höheren Raum linear trennbar sind. Abbildung 5 zeit die Idee und Abbildung 6 ein konkretes Beispiel.

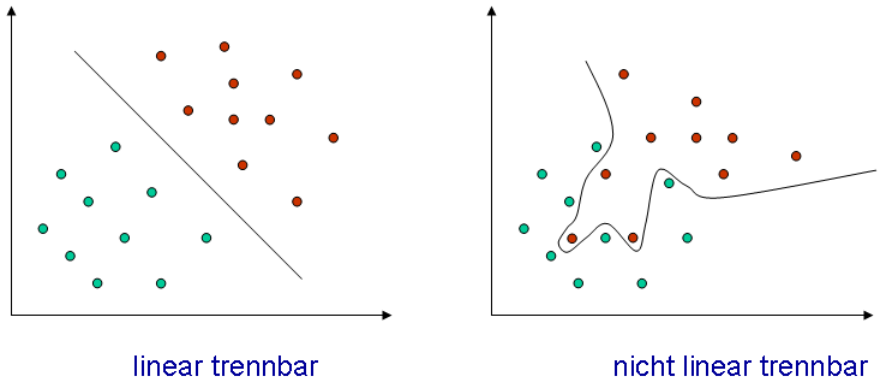


Abbildung 4: Das rechte Daten-Set ist nicht durch eine Gerade trennbar.

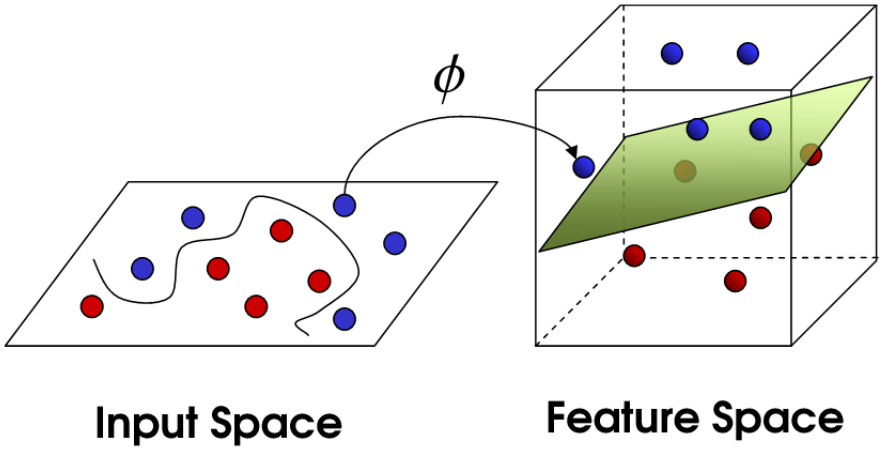


Abbildung 5: ϕ transformiert die Samples in einen höherdimensionalen Raum.

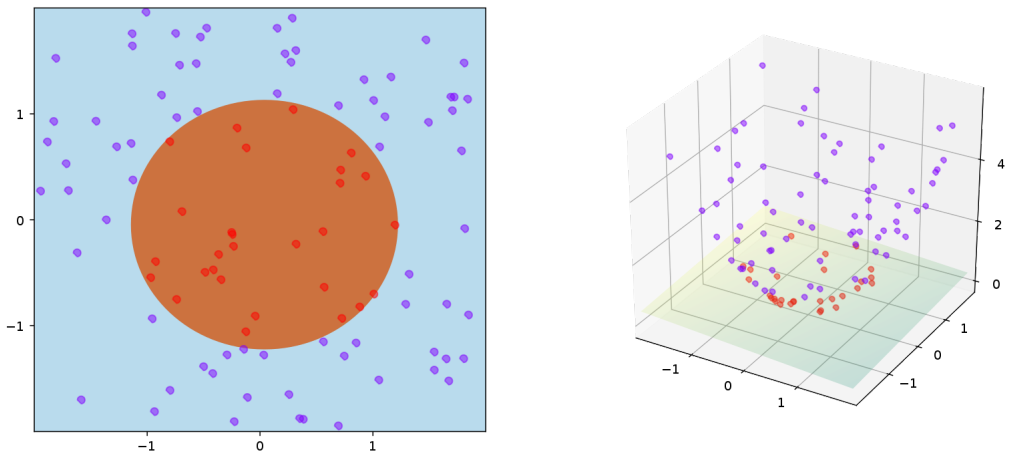


Abbildung 6: $\phi(a, b) = (a, b, a^2 + b^2)$ vom 2-D ins 3-D. Dann ist eine lineare Trennung möglich.

6.3 Aufgaben

6.3.1 Aufgabe 1

Es sind 12 Samples gegeben (siehe Abbildung 8). Es handelt sich um ein abstraktes Beispiel. Sie finden das Daten-Set via Teams in der Datei `abstrakt.csv`.

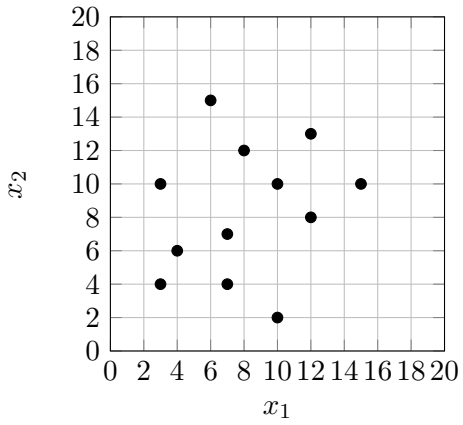
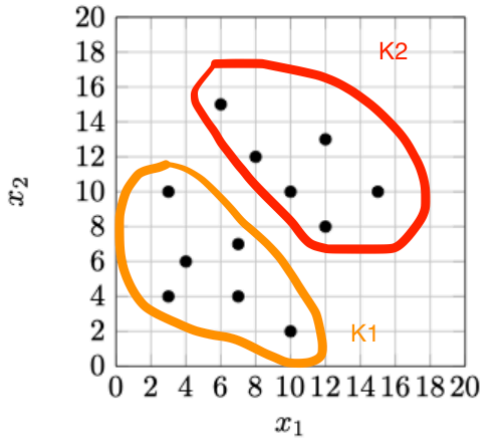


Abbildung 7: Daten-Set mit 12 Samples.

1. Kopieren Sie die Datei in den Ordner `svm` (PyCharm). Öffnen Sie die CSV-Datei und markieren Sie in Abbildung 8 für jeden Punkt die dazugehörige Klasse. Welche Dimension besitzen die Samples? Ist das Daten-Set linear trennbar?

   Lösungsvorschlag: Das Daten-Set ist linear trennbar, da die beiden Klassen durch eine Gerade getrennt werden können. Die Samples besitzen die Dimension 2, da es zwei "Features" gibt (x_1 und x_2).



2. Bestimmen Sie die Hyperebene mit `scikit`. Trainieren Sie eine SVM und lesen Sie die Werte für die Hyperebene aus. Stellen Sie die Hyperebene dann in Koordinatenform und Vektorform dar. Kombinieren Sie folgenden Code mit dem Code aus der "k-Nearest-Neighbor-Übung".

```
import sklearn.svm as svm

# Support Vector Classification
svclassifier = svm.SVC(kernel="linear")
svclassifier.fit(matrix_X, vektor_y)
print("w = ", svclassifier.coef_)
print("b = ", svclassifier.intercept_)
```

Lösungsvorschlag: Wir erhalten den Vektor $\vec{w} = \begin{bmatrix} 0,333 \\ 0,333 \end{bmatrix}$ und den "Achsenabschnitt" $b = -5,665$. Die Koordinatenform lautet allgemein $H : w_1 \cdot x_1 + w_2 \cdot x_2 + b = 0$. Wir können nun w_1, w_2 und b einsetzen und erhalten: $H : 0,333 \cdot x_1 + 0,333 \cdot x_2 - 5,665 = 0$. In Vektorform wäre dies $\vec{w} \circ \vec{x} + b = \begin{bmatrix} 0,3 \\ 0,3 \end{bmatrix} \circ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - 5,665 = 0$. Man muss hier mit drei Stellen nach dem Komma arbeiten, da sonst die Trennebene "falsch" liegt.

3. Zeichnen Sie die Hyperebene in Abbildung 8 ein. Tragen Sie auch den Vektor $\vec{w}$ ein. Markieren Sie die Support-Vektoren. Wie gross ist der Abstand? Liegt die Gerade in der "Mitte"?

   Lösungsvorschlag: Wir können die Trennebene (hier eine Gerade) ermitteln durch das Umformen der Koordinatenform: $0,333 \cdot x_1 + 0,333 \cdot x_2 - 5,665 = 0 \Leftrightarrow x_2 = -x_1 + 17,012$. Wir können dies nun einzeichnen.

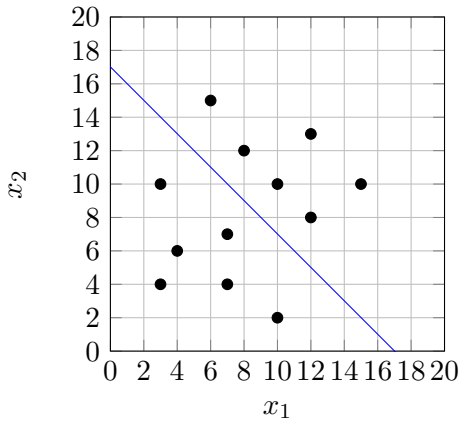
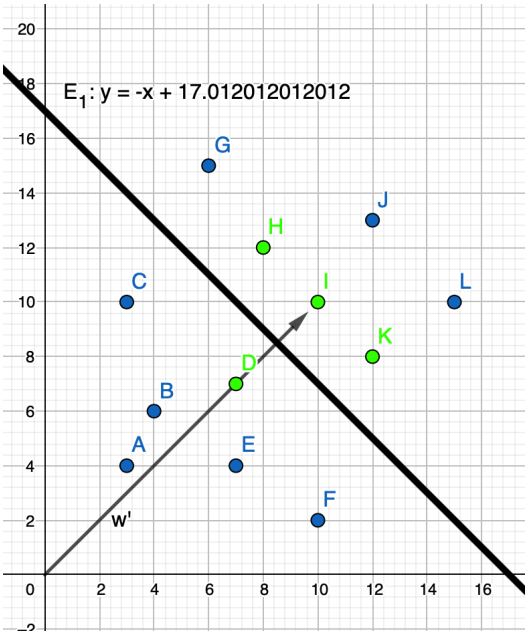


Abbildung 8: Daten-Set mit 12 Samples.

Der Vektor $\vec{w}$ ist senkrecht zur Ebenen. Die Supportvektoren sind in folgender Abbildung grün eingezeichnet. Der Vektor $\vec{w'}$ ist der gestreckte Vektor $\vec{w}$.



4. Klassifizieren Sie den Sample $\vec{x_1^*} = \begin{bmatrix} 8 \\ 8 \end{bmatrix}$ von Hand. Leiten Sie daraus die Klassifizierungsregel ab.

   Überprüfen Sie das Resultat in dem Sie das Programme erweitern und den Sample mit Python klassifizieren lassen.

   Lösungsvorschlag: Wir setzen den Sample $\vec{x_1^*} = \begin{bmatrix} 8 \\ 8 \end{bmatrix}$ in die Vektorform ein und erhalten:

$$\vec{w} \circ \vec{x^*} - 5,665 = \begin{bmatrix} 0,333 \\ 0,333 \end{bmatrix} \circ \begin{bmatrix} 8 \\ 8 \end{bmatrix} - 5,665 = 0,333 \cdot 8 + 0,333 \cdot 8 - 5,665 = -0,337$$

Für $\vec{x_1^*}$ vergeben wir die Klasse K1, da der Wert negativ ist.

Wir setzen noch $\vec{x_2^*} = \begin{bmatrix} 14 \\ 14 \end{bmatrix}$ in die Vektorform ein und erhalten:

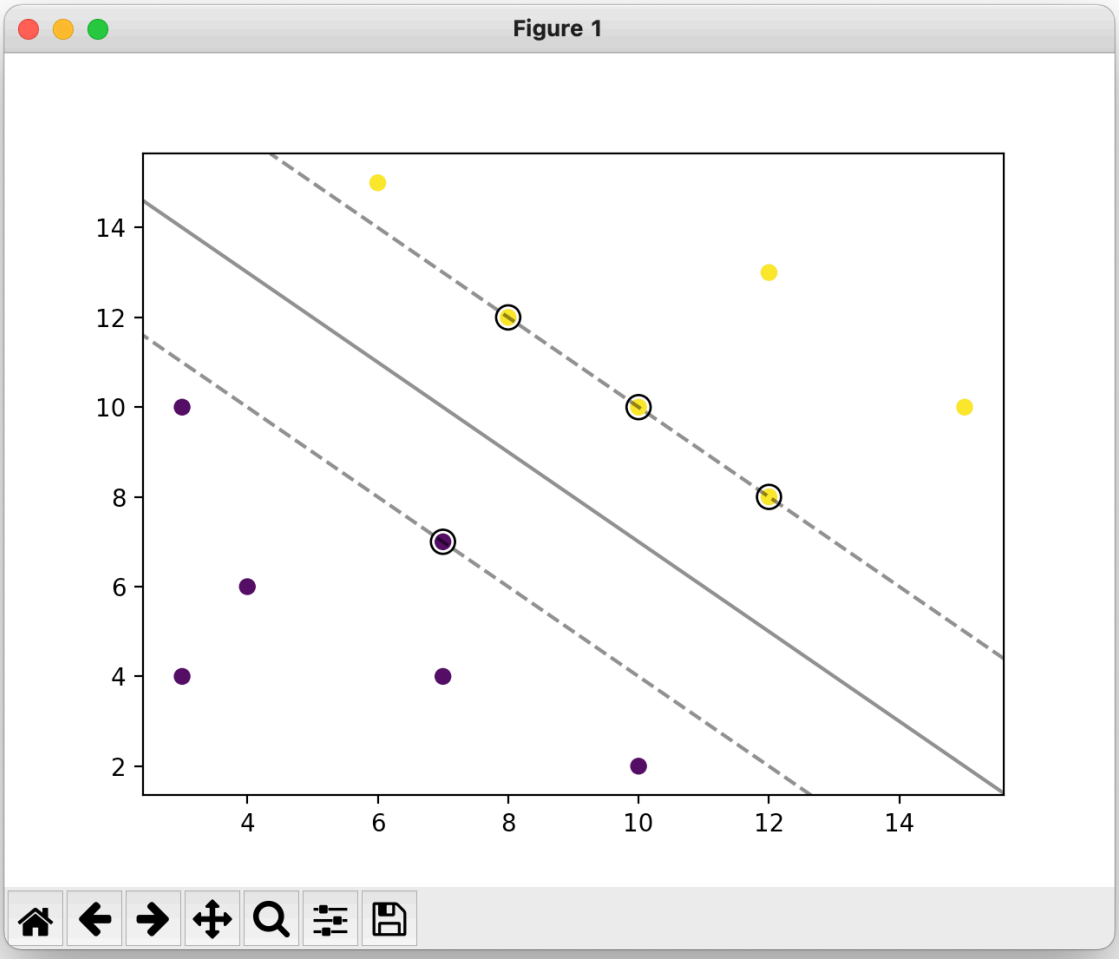
$$\vec{w} \circ \vec{x^*} - 5,665 = \begin{bmatrix} 0,333 \\ 0,333 \end{bmatrix} \circ \begin{bmatrix} 14 \\ 14 \end{bmatrix} - 5,665 = 0,333 \cdot 14 + 0,333 \cdot 14 - 5,665 = 3,659$$

Für $\vec{x_2^*}$ vergeben wir die Klasse K2, da der Wert positiv ist.

Die Entscheidungsregel lautet:

$$\vec{w} \circ \vec{x^*} - b < 0 \Rightarrow K_1$$
$$\vec{w} \circ \vec{x^*} - b \geq 0 \Rightarrow K_2$$

Die Python-Datei `svm_abstrakt_plot.py` stellt das Ergebnis grafisch ansprechend dar.



6.3.2 Aufgabe 2

1. Studieren Sie das "Banknoten"-Daten-Set unter `https://archive.ics.uci.edu/ml/datasets/banknote+authentication#`. Die Klasse 0 bedeutet "echt". Die Klasse 1 bedeutet, dass die Banknote gefälscht ist.

   Lösungsvorschlag: Es gibt 1372 Samples mit 5 Features und zwei Klassen.

   Data Set Information

   Data were extracted from images that were taken from genuine and forged banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have 400x 400 pixels. Due to the object lens and distance to the investigated object gray-scale pictures with a resolution of about 660 dpi were gained. Wavelet Transform tool were used to extract features from images.

   Attribute Information:

   (a) variance of Wavelet Transformed image (continuous)
   (b) skewness of Wavelet Transformed image (continuous)
   (c) curtosis of Wavelet Transformed image (continuous)
   (d) entropy of image (continuous)
   (e) class (integer)

2. Erstellen Sie ein Python-Programm, welches die Klassifizierung mit einer SVM vornimmt. Erstellen Sie Trainingsdaten und Testdaten. Sie finden die Daten in der Datei `banknoten.csv`.

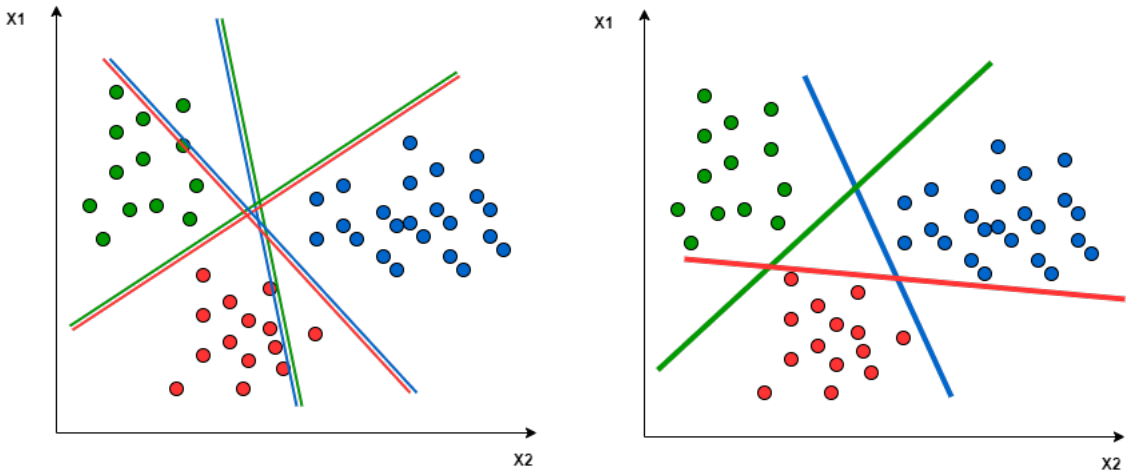
   Lösungsvorschlag: Sie finden die Lösung in der Python-Datei `svm_knn_banknoten.py`.

3. Vergleichen Sie KNN und SVM. Welches Verfahren schneidet beim besser ab?

   Lösungsvorschlag: Beide Verfahren funktionieren sehr gut. Es gibt praktisch keinen Unterschied. Bei mehrmaligem Ausführen werden (automatisch) verschiedene Testdaten erzeugt. Die SVM hat meist eine Accuracy von 0.99 oder 1.00. KNN erzeugt meist eine Accuracy von 1.00 bei $k = 5$.

4. Informieren Sie sich darüber, wie man die SVM bei mehr als zwei Klassen einsetzen kann. Wenden Sie dann die SVM auf das Iris-Daten-Set an.

   Lösungsvorschlag: Es gibt mehrere Möglichkeiten die Klassifizierung durchzuführen. Beliebt sind One vs One (OVO) und One vs All (OVA) Verfahren. Folgende Abbildungen sollen das Prinzip zeigen.



   Bei OVO wird jeweils zwischen allen Paaren von Klassen (dies sind bei n Klassen insgesamt $\frac{n\cdot(n-1)}{2}$ SVMs) eine SVM trainiert. Im Beispiel gibt es 3 Klassen und somit $\frac{3\cdot(2)}{2} = 3$ SVMs. In der linken Grafik sieht man jeweils zwei parallele Trennlinien. Dies zeigt dass die Trennlinie zwischen "Rot" und "Grün" aus Sicht der grünen Klasse identisch ist mit der Trennlinie aus Sicht

der roten Klasse. Deshalb ist in der Formel die Division durch 2 zu finden. Man muss also nur eine der beiden SVMs trainieren. Bei OVA wird pro Klasse eine SVM trainiert. Trainiert man Klasse c_1 werden alle anderen Samples zu einer Klasse gezählt. Dies ist in der rechten Grafik zu sehen. Mehr Details dazu gibt es z.B. unter `https://www.analyticsvidhya.com/blog/2021/05/multiclass-classification-using-svm/`. Laut scikit (`https://scikit-learn.org/stable/modules/svm.html#classification`) ist der OVR Ansatz in der Praxis vernünftig. Die LinearSVC kann dafür benutzt werden. Siehe `https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC`

In der Python-Datei `svm_iris_multiclass.py` befindet sich die Implementation. Darin wird auch ein Plot erstellt, bei dem die SVM mehrmals ausgeführt wird. Man könnte nun noch eine andere SVM mit verschiedenen Kernelfunktionen ausprobieren, ob diese besser funktionieren.