Machine Learning: Evaluation

Präsentation: Vorname, Nachname

Lizenz:

deep.TEACHING

www.deep-teaching.org

Fishers Iris-Daten

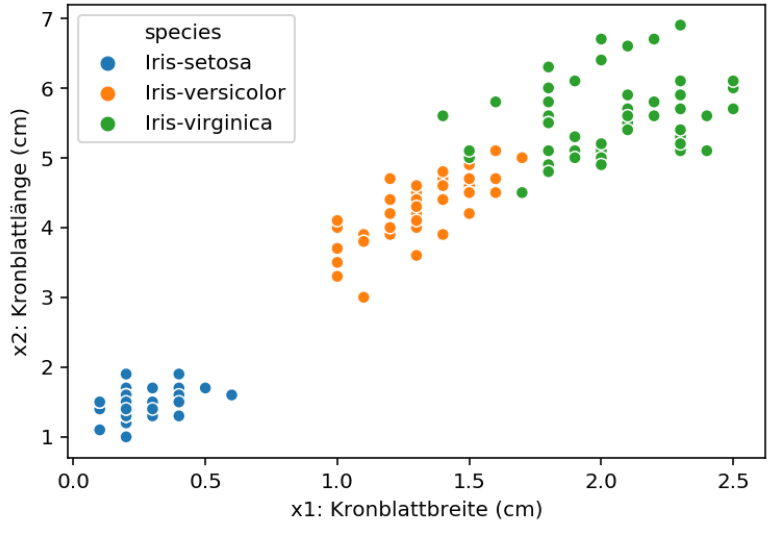
In [3]:
```
df = Iris(verbose=False).dataframe()
```

In [4]:
```
df['species'].value_counts() # Schwertlilien
```

Out[4]:
```
Iris-setosa        50
Iris-virginica     50
Iris-versicolor    50
Name: species, dtype: int64
```

Plot der Daten

In [5]:
```
sns.scatterplot(data=df, x='petal_width', y='petal_length', hue='species')
plt.xlabel('x1: Kronblattbreite (cm)'); plt.ylabel('x2: Kronblattlänge (cm)');
```



Klassifikationsproblem

Unterscheidung von **Iris-versicolor** und **Iris-virginica** anhand der der zwei Features Kronblattbreite (**x1**) und Kronblattlänge (**x2**).

In [6]:
```
df_reduced = df.query('species == "Iris-versicolor" | species == "Iris-virginic
a"')
df_reduced.head()
```

Out[6]:

	sepal_length	sepal_width	petal_length	petal_width	species
50	7.0	3.2	4.7	1.4	Iris-versicolor
51	6.4	3.2	4.5	1.5	Iris-versicolor
52	6.9	3.1	4.9	1.5	Iris-versicolor
53	5.5	2.3	4.0	1.3	Iris-versicolor
54	6.5	2.8	4.6	1.5	Iris-versicolor

In [7]:
```
df_reduced['species'].value_counts()
```

Out[7]:
```
Iris-virginica     50
Iris-versicolor    50
Name: species, dtype: int64
```

X- und Y-Arrays

In [8]:
```
X = df_reduced[['petal_width', 'petal_length']].values
Y = df_reduced['species'].replace({'Iris-versicolor': 0, 'Iris-virginica': 1}).v
alues
```

In [9]:
```
X[:5]  # x1, x2 pairs
```

Out[9]:
```
array([[1.4, 4.7],
       [1.5, 4.5],
       [1.5, 4.9],
       [1.3, 4. ],
       [1.5, 4.6]])
```

In [10]:
```
Y[:5]
```

Out[10]:
```
array([0, 0, 0, 0, 0])
```

In [11]:
```
X.shape, Y.shape
```

Out[11]:
```
((100, 2), (100,))
```

Skalierung

In [12]:
```
scaler = StandardScaler()  # import aus Scikit-Learn
X_scaled = scaler.fit_transform(X)  # x1 und x2 werden separat skaliert
```

In [13]:
```
X_scaled[:5]
```

Out[13]:
```
array([[-0.65303909, -0.25077906],
       [-0.41643072, -0.49425387],
       [-0.41643072, -0.00730424],
       [-0.88964745, -1.10294091],
       [-0.41643072, -0.37251647]])
```

In [14]:
```
X_scaled.shape
```

Out[14]:
```
(100, 2)
```

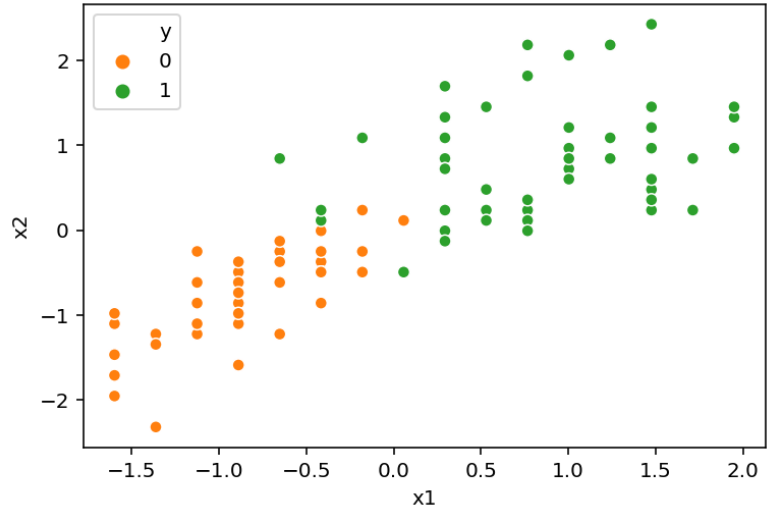
Plot der reduzierten und skalierten Daten

Wie gut kann eine Klassifikation anhand der Eingabe x_1, x_2 sein?

Wäre die Klassifikation mit nur einer Eingabevariable x_1 oder x_2 ähnlich gut?

In [16]:
```
df_scaled = pd.DataFrame(X_scaled, columns=['x1', 'x2'])
df_scaled['y'] = Y

sns.scatterplot(data=df_scaled, x='x1', y='x2', hue='y', palette=cp);
```

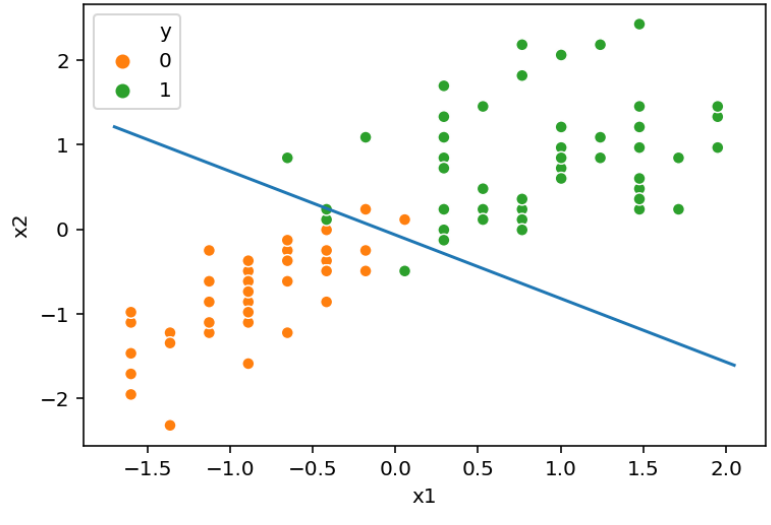


Logistische Regression zur Klassifizierung

In [25]:
```
alpha = 0.1
epochs = 1000
w1, w2, b = np.random.randn(3)
w1, w2, b, costs = sgd(X_scaled, Y, w1, w2, b, alpha, epochs)
w1, w2, b
```

Out[25]:
```
(2.687254583769273, 3.571507200616939, 0.23774639988349686)
```

In [26]:
```
plot_boundary(df_scaled, make_decision_boundary(w1, w2, b, 0.5))
```



Trennung von Trainings- und Testdaten

Das vorhandene Datenset wird in Trainings- (80%) und Testdaten (20%) unterteilt.

- Die Testdaten sollten eine **stratifizierte** Auswahl sein, sodass die Anteile der Klassen in Trainings- und Testdaten gleichen sind.

Das Gradientenabstiegsverfahren ermittelt die beste Hypothese (w_1, w_2, b) nur anhand der Trainingsdaten. Anschließend werden beide Datensets evaluiert.

In [27]:
```
X_train, X_test, Y_train, Y_test = train_test_split(
    X_scaled, Y, stratify=Y, test_size=0.2, random_state=42
)  # Import aus Scikit-Learn
```

In [28]:
```
train_classes = dict(zip(*np.unique(Y_train, return_counts=True)))
test_classes = dict(zip(*np.unique(Y_test, return_counts=True)))

train_classes, test_classes
```

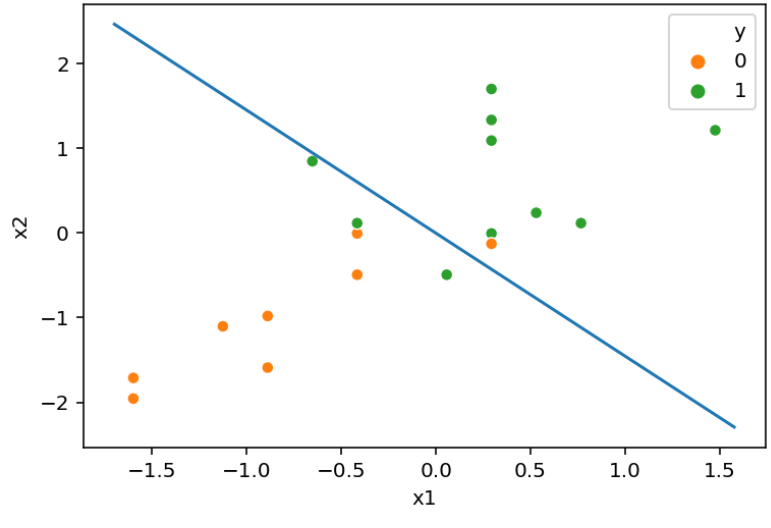
Out[28]:
```
({0: 40, 1: 40}, {0: 10, 1: 10})
```

SGD mit Trainingsdaten

In [29]:
```
alpha = 0.1
epochs = 1000
np.random.seed(42)
w1, w2, b = np.random.randn(3)
w1, w2, b, cost_per_epoch = sgd(X_train, Y_train, w1, w2, b, alpha, epochs)
w1, w2, b
```

Out[29]:
```
(3.6962765211562245, 2.548083316850051, 0.010892345474331 82)
```

In [30]:
```
df_test = pd.DataFrame(X_test, columns=['x1', 'x2'])
df_test['y'] = Y_test
plot_boundary(df_test, make_decision_boundary(w1, w2, b, 0.5))
```



Klassifikationsgenauigkeit

Die Klassifikationsgenauigkeit (accuracy) einer Hypothese für ein bestimmtes Datenset (X, Y) berechnet sich aus der Anzahl aller korrekten Klassifikationen T (True) geteilt durch die Anzahl aller korrekten Klassifikationen und aller falschen Klassifikationen F (False).

$$accuracy = \frac{T}{T + F}$$

In [32]:
```
classify = make_classify(w1, w2, b, 0.5)

C_test = [classify(x1, x2) for x1, x2 in X_test]
accuracy(C_test, Y_test)  # Implementierung in der Übung
```

Out[32]:
```
0.8
```

Korrektheit je Zielklasse

True Positive (TP)

Anzahl der Samples die **korrekt** (True) als Klasse **1** (Positive) bestimmt wurden.

False Positive (FP)

Anzahl der Samples die **falsch** (False) als Klasse **1** (Positive) bestimmt wurden.

True Negative (TN)

Anzahl der Samples die **korrekt** (True) als Klasse **0** (Negativ) bestimmt wurden.

False Negative (FN)

Anzahl der Samples die **falsch** (False) als Klasse **0** (Negativ) bestimmt wurden.

In [34]:
```
tp, fp, tn, fn = tp_fp_tn_fn(C_test, Y_test)
tp, fp, tn, fn
```

Out[34]:
```
(7, 1, 9, 3)
```

Precision und Recall

Precision ist der Anteil der korrekten positiv Klassifizierungen an allen positiv Klassifizierungen.

- Sind die als Klasse 1 bestimmten Samples tatsächlich in Klasse 1?

Recall ist der Anteil der korrekten positiv Klassifizierungen an allen positiven Samples.

- Wurden alle Samples der Klasse 1 vom Klassifizierer gefunden?

$$precision = \frac{TP}{TP + FP}$$

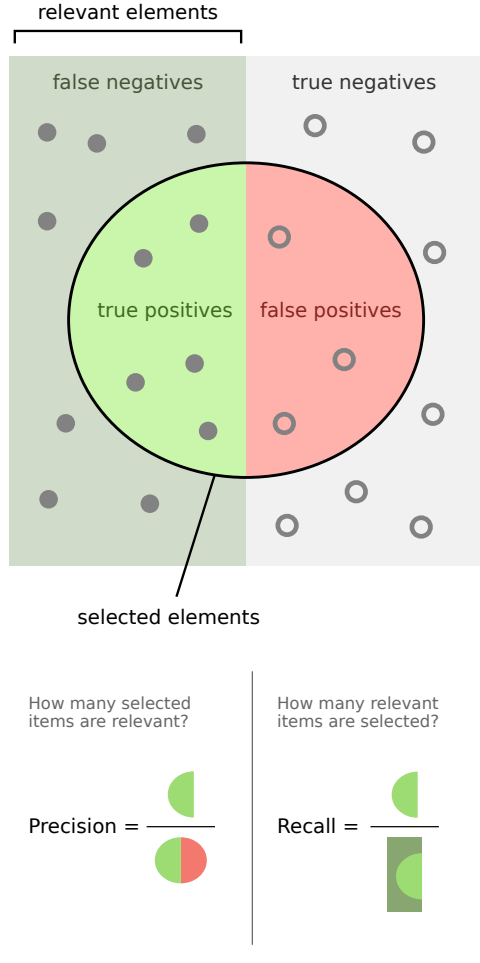
$$recall = \frac{TP}{TP + FN}$$

In [36]:
```
precision, recall = precision_recall(tp, fp, fn)  # Implementierung in der Übung
precision, recall
```

Out[36]:
```
(0.875, 0.7)
```

Grafik zu Precision und Recall

- Quelle: en.wikipedia.org/wiki/F1_score#/media/File:Precisionrecall.svg
- Lizenz: CC BY-SA 4.0 Walber (commons.wikimedia.org/wiki/User:Walber)



F-Score

F-Score kombiniert precision und recall.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

In der allgemeinen Formulierung kann der Einfluss von precision gewichtet werden.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

In [38]:
```
f_score(precision, recall, 1)  # Implementierung in der Übung
```

Out[38]:
```
0.7777777777777777
```

In [39]:
```
f_score(precision, recall, 0.5)
```

Out[39]:
```
0.8333333333333334
```

In [40]:
```
f_score(precision, recall, 2)
```

Out[40]:
```
0.7291666666666666
```

Auswertung

In [42]:
```
C_train = [classify(x1, x2) for x1, x2, in X_train]
evaluate(C_train, Y_train)
```

Out[42]:
```
{'TP': 39,
 'FP': 1,
 'TN': 39,
 'FN': 1,
 'Accuracy': 0.975,
 'Precision': 0.975,
 'Recall': 0.975,
 'F1-Score': 0.975}
```

In [43]:
```
C_test = [classify(x1, x2) for x1, x2, in X_test]
evaluate(C_test, Y_test)
```

Out[43]:
```
{'TP': 7,
 'FP': 1,
 'TN': 9,
 'FN': 3,
 'Accuracy': 0.8,
 'Precision': 0.875,
 'Recall': 0.7,
 'F1-Score': 0.7777777777777777}
```

Underfitting und Overfitting

Underfitting und Overfitting sind gleichermaßen unerwünscht.

Underfitting

Die Hypothese ist zu simpel ist, um das Problem abzubilden.

- Sowohl Training- als auch Test-Accuracy sind niedrig.

Overfitting

Die Hypothese ist so komplex, dass die Trainingsdaten "auswendig" gelernt werden, sodass die Lösung nicht generell für ungesehene Testdaten gültig ist.

- Die Training-Accuracy ist hoch, die Test-Accuracy ist niedrig.

Tensorflow Playground

https://playground.tensorflow.org (https://playground.tensorflow.org)