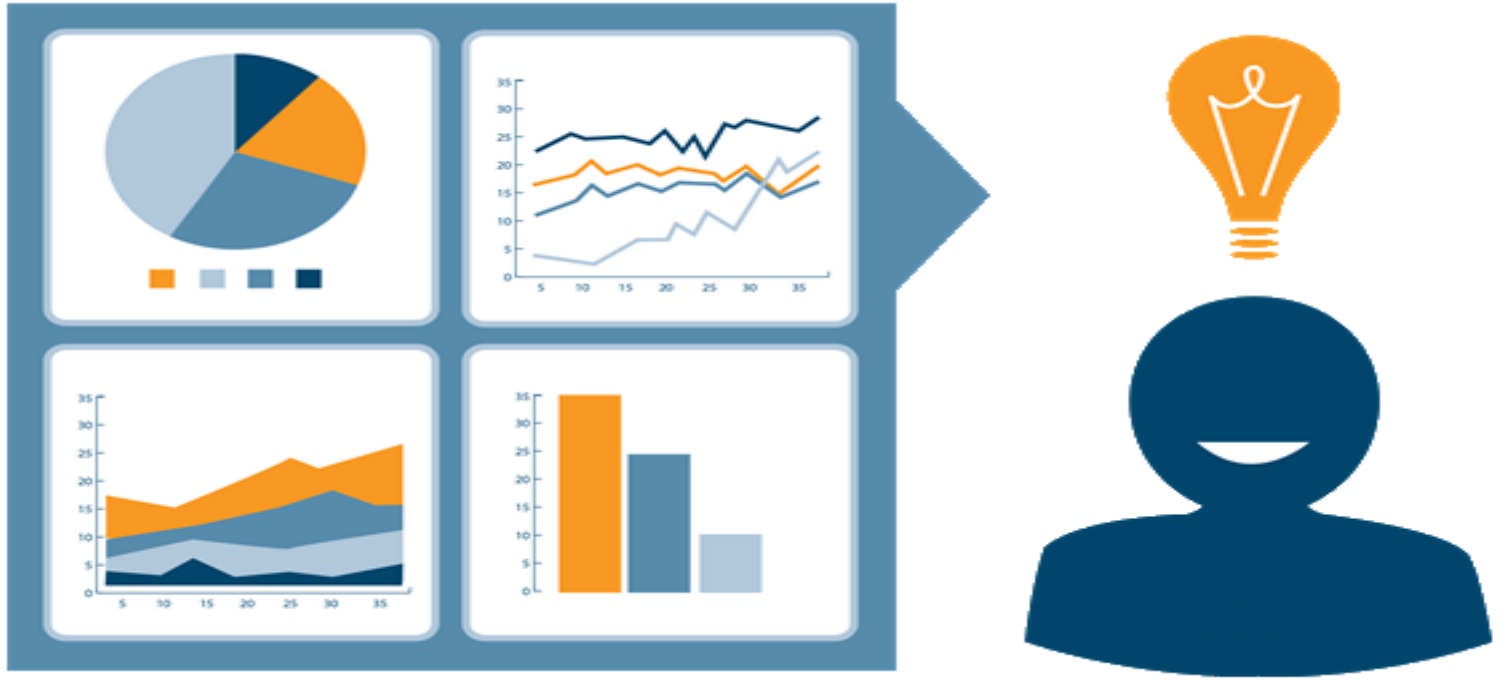
Introduction to Data Science
Prof. Dr. Bernhard Drabant
Duale Hochschule Baden-Württemberg Mannheim
Fakultät Wirtschaft

Methoden und Verfahren

Training, Validierung, Test, Gütebestimmung



Methoden und Verfahren – Training, Test, Güte

Ziel dieses Kapitels

- Generelle methodische Vorgehensweise bei der Wissensgewinnung
  - Auswahl der Lernalgorithmen
  - Training
  - Validierung und Test
  - Gütebestimmung des erlernten Wissens und der Modelle
  - anhand von Beispielen aus den Bereichen Regression und Klassifikation

Nicht Teil des Kapitels

- Detaillierte Darstellung von Lernalgorithmen und des Lernens
- Details der zugrundeliegenden Theorien und Konzepte

Herausforderungen der Wissensgewinnung (Lernen)

Wahl der Methoden und Verfahren (Lernalgorithmen)

- Overfitting
  - Lernalgorithmus / Modell ist für die vorliegenden Trainingsdaten zu komplex
    - Erkennung irreführender, nicht verallgemeinerbarer Muster
    - oft sehr hohe Genauigkeit auf Trainingsdaten und hohe Ungenauigkeit auf neuen Daten
  - Lösungen
    - Verwendung eines einfacheren Lernalgorithmus / Modells
    - Ggf. Restriktionen im Algorithmus festlegen → ***Regularisierung***
    - Verwendung einer kleineren Merkmalsmenge in den Trainingsdaten
    - Reduzierung von Rauschen
    - Verwendung einer größeren Trainingsdatenmenge

Herausforderungen der Wissensgewinnung (Lernen)

Wahl der Lernalgorithmen

- Underfitting
  - Lernalgorithmus / Modell ist für die vorliegenden Trainingsdaten zu einfach
    - Erkennt relevante Strukturen in den Daten nicht
    - oft große Ungenauigkeit auf Trainingsdaten und neuen Daten
  - Lösungen
    - Verwendung eines komplexeren Modells
    - Verwendung einer größeren oder besseren Merkmalsmenge in den Trainingsdaten
    - Ggf. Restriktionen im Modell verringern

Herausforderungen der Wissensgewinnung (Lernen)

Wahl der Lernalgorithmen

Konsequenz

- Wähle Lernalgorithmen / Modelle so, dass Over- und Underfitting im resultierenden Modell möglichst gering sind
- Modell soll möglichst genau **neue Daten** beschreiben
- Dazu erforderlich:
  - Test und Validierung → dazu später mehr
  - Abwägung!

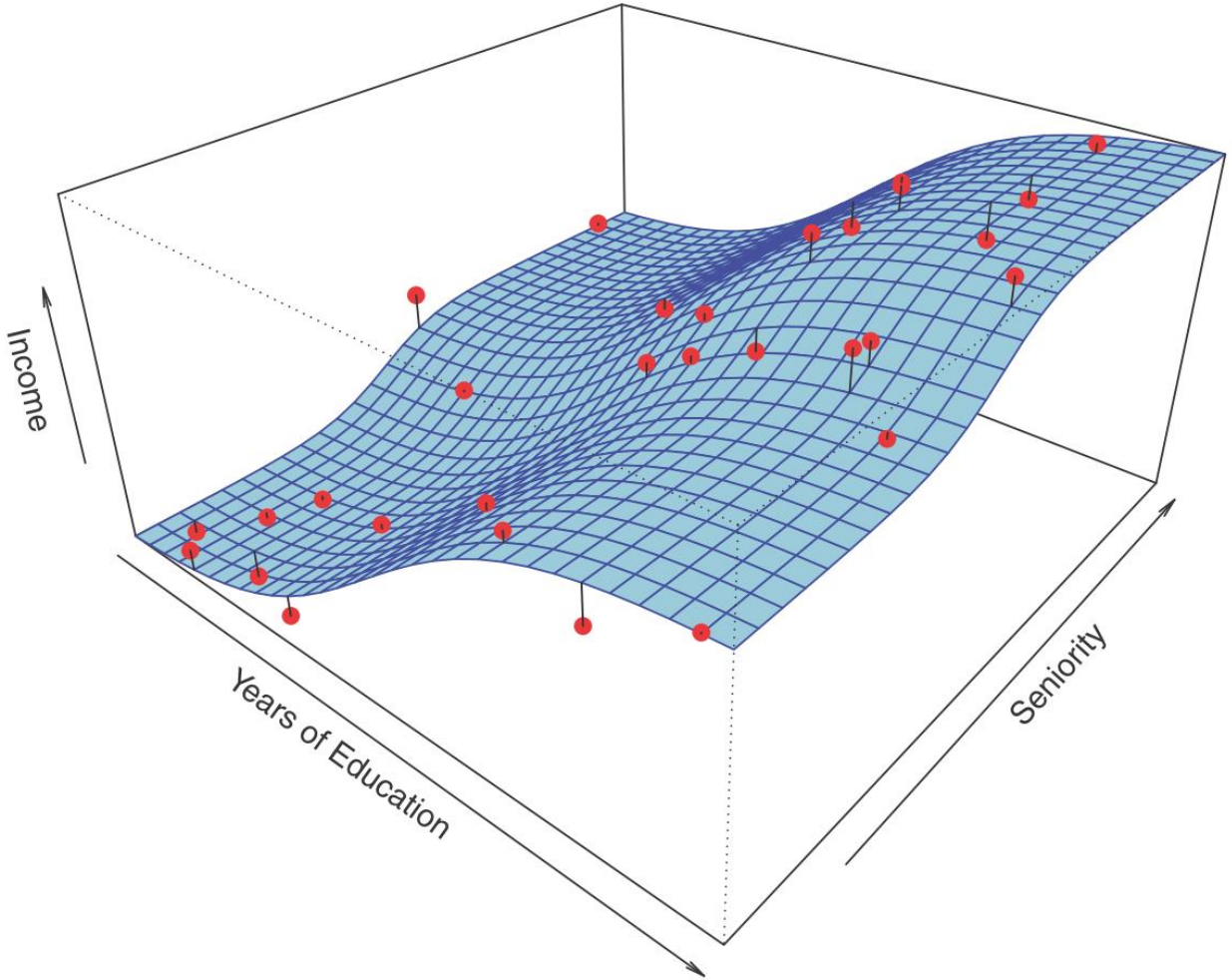
Beispiel-Szenario: Einkommensverhältnisse

- **Daten:**
  - **Input-Variablen:** Ausbildungsdauer (**`YoE`**), Berufserfahrung (**`Sen`**)
  - **Zielvariable:** Einkommen (**`Inc`**)
- **Annahme:**
  - Es gibt eine „exakte“ (unbekannte) Zuordnung $\varphi: (\mathbf{YoE}, \mathbf{Sen}) \rightarrow \mathbf{Inc}$

    $$\mathbf{Inc} = \varphi(\mathbf{YoE}, \mathbf{Sen}) + \varepsilon$$

    mit statistischem Fehler ε
  - Reale **Trainingsdaten** **`TR`** (**rote Punkte**)

    $$\mathbf{TR} = \{ (\mathbf{YoE}_k, \mathbf{Sen}_k, \mathbf{Inc}_k)_{k = 1,..,n} \}$$



Quelle: G. James, D. Witten, T. Hastie, R. Tibshirani. *An Introduction to Statistical Learning*. Springer (2015)

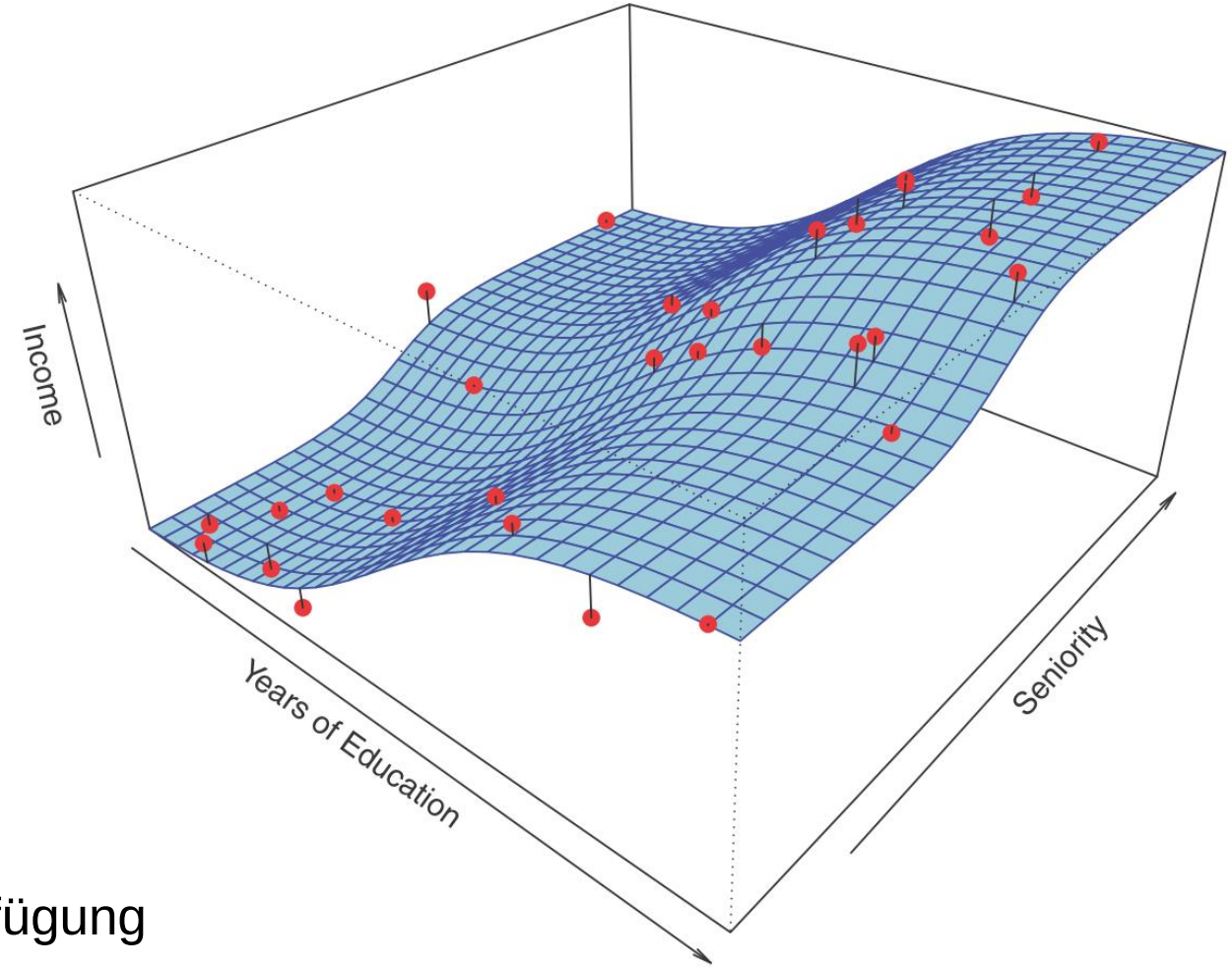
Beispiel-Szenario: Einkommensverhältnisse

- Ziel: Lerne ein geeignetes Modell

  $\hat{\varphi}: (\mathbf{YoE}, \mathbf{Sen}) \rightarrow \mathbf{Inc}$, das

  - φ approximiert: $\hat{\varphi} \cong \varphi$
  - auf **Trainingsdatensatz `TR`** mit Hilfe eines Lernalgorithmus erlernt wird

- Modell $\hat{\varphi}$ ist Ergebnis des Lernens
  - Verschiedene Lernalgorithmen stehen zur Verfügung



Quelle: G. James, D. Witten, T. Hastie, R. Tibshirani. *An Introduction to Statistical Learning*. Springer (2015)

Beispiel-Szenario: Einkommensverhältnisse

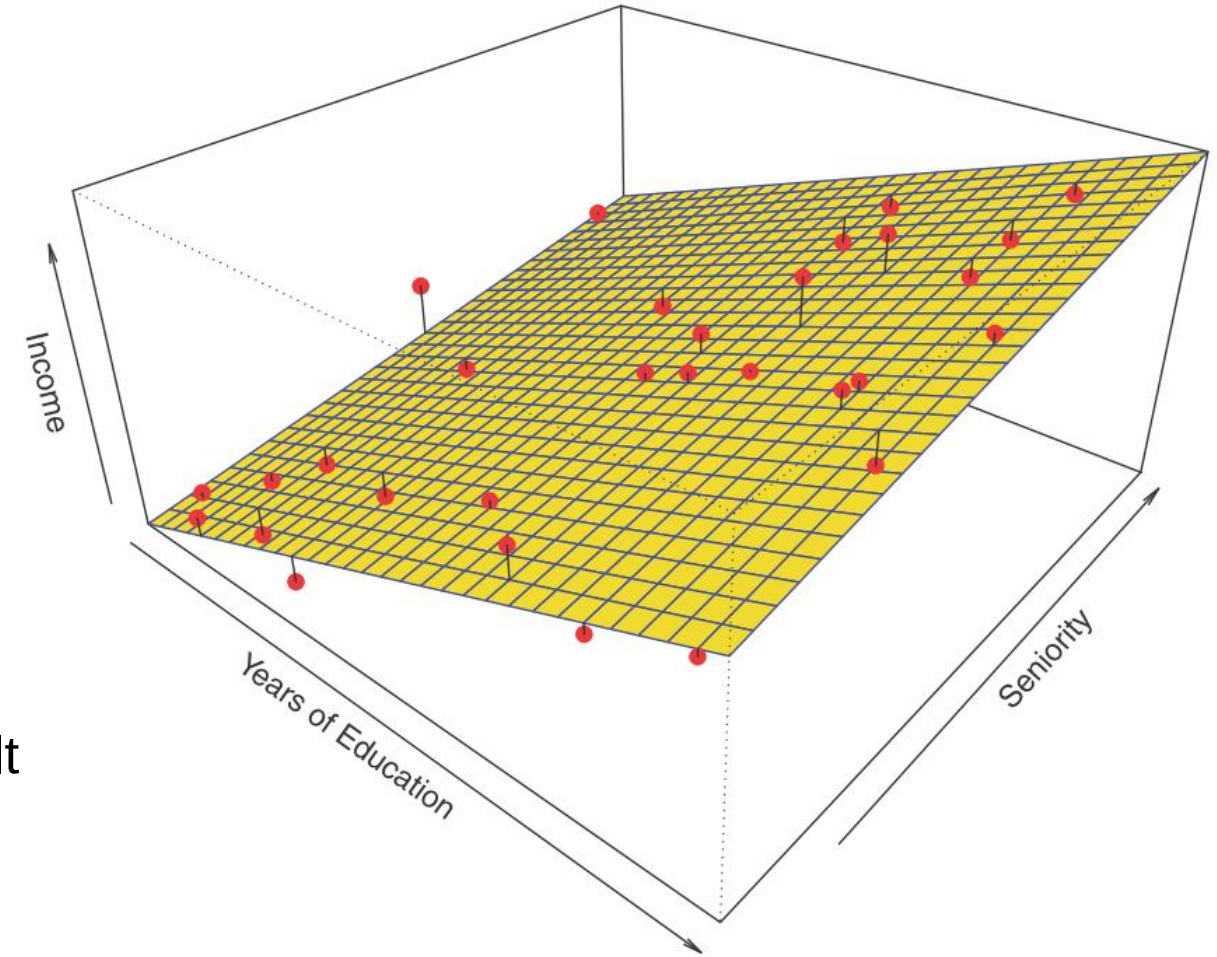
- **Annahme 1:** Modell $\varphi\hat{}$ ist **linear** in den Variablen

  $(\mathbf{YoE}, \mathbf{Sen})$

$$\varphi\hat{}(\mathbf{YoE}, \mathbf{Sen}) = a + b*\mathbf{YoE} + c*\mathbf{Sen}$$

- Ziel: Bestimme Parameter a, b, c aus Trainingsdatensatz **TR** so, dass möglichst genau gilt

$$\varphi\hat{}(\mathbf{YoE}, \mathbf{Sen}) \cong \varphi(\mathbf{YoE}, \mathbf{Sen}) + \varepsilon$$



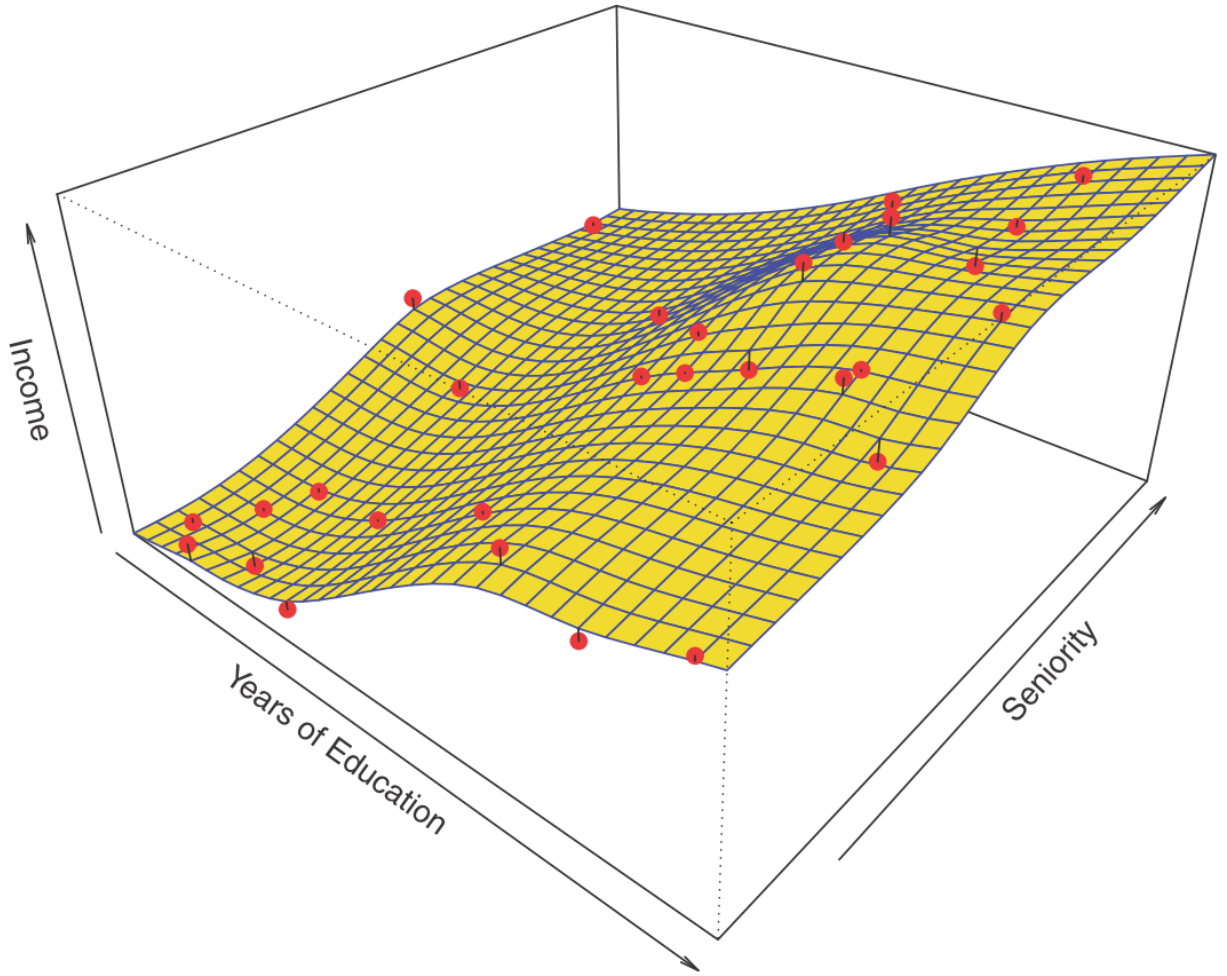
Beispiel-Szenario: Einkommensverhältnisse

- **Annahme 2:** Modell $\varphi\hat{}$ ist eine weiche **Spline-Funktion** in den Variablen $(\mathbf{YoE, Sen})$

$$\varphi\hat{}(\mathbf{YoE, Sen}) = \text{s-spline}(\mathbf{YoE, Sen})$$

- Ziel: Bestimme $\varphi\hat{}$ aus Trainingsdatensatz **TR** so, dass möglichst genau gilt

$$\varphi\hat{}(\mathbf{YoE, Sen}) \cong \varphi(\mathbf{YoE, Sen}) + \varepsilon$$



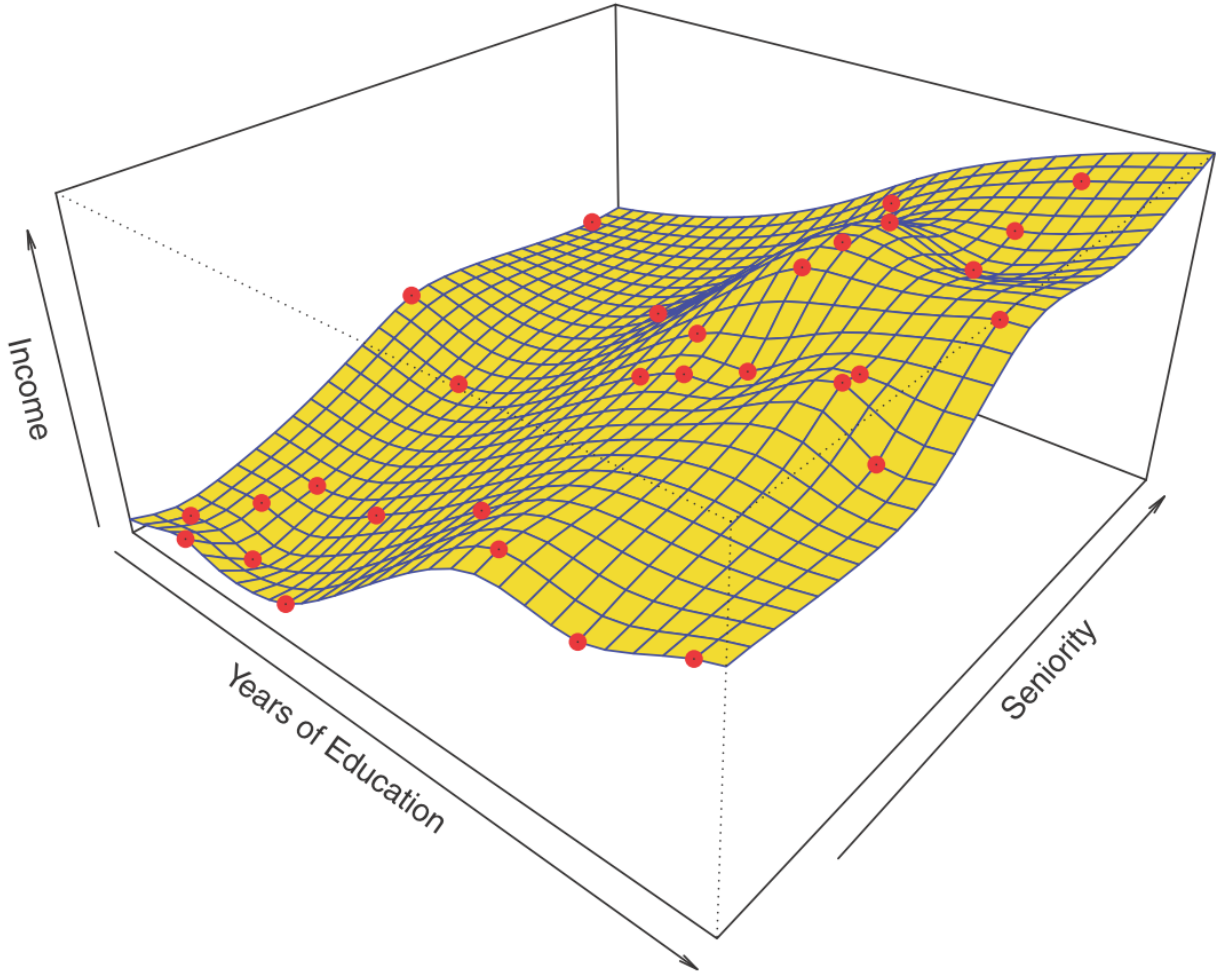
Beispiel-Szenario: Einkommensverhältnisse

- **Annahme 3:** Modell $\varphi\hat{}$ ist eine harte **Spline-Funktion** in den Variablen (**YoE, Sen**)

$\varphi\hat{}$(**YoE, Sen**) = h-spline(**YoE, Sen**)

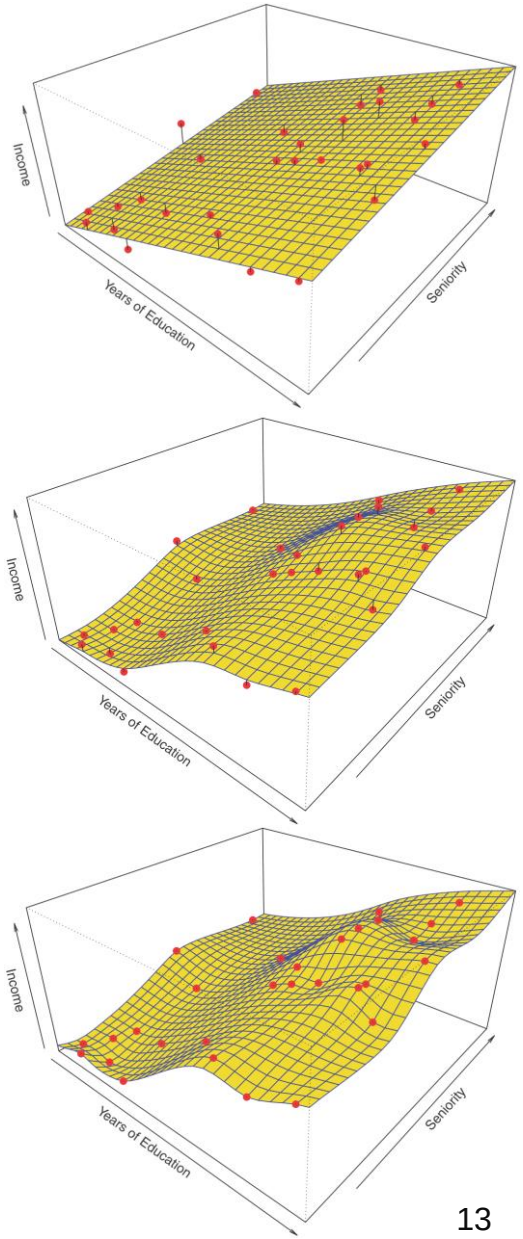
- Ziel: Bestimme $\varphi\hat{}$ aus Trainingsdatensatz **TR** so, dass möglichst genau gilt

$$\varphi\hat{}(\mathbf{YoE, Sen}) \cong \varphi(\mathbf{YoE, Sen}) + \varepsilon$$



Beispiel-Szenario: Einkommensverhältnisse

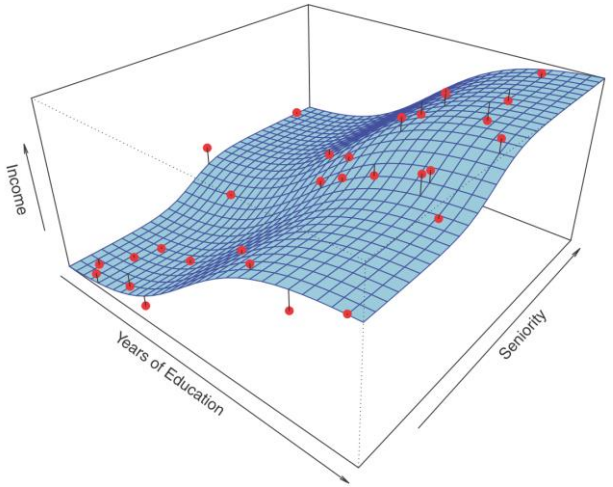
- **Diskussion:** Welches Modell ist einfacher / besser geeignet / …?
- Kriterien
  - Interpretierbarkeit
    - Beispiel: Wie hängt Einkommen von Ausbildungsdauer und Berufserfahrung ab?
  - Flexibilität / Anpassungsfähigkeit an Trainingsdaten
    - Freiheitsgrade
    - Komplexität des Modells
  - Realitätsnähe
    - Wie genau spiegelt das Modell die tatsächlichen (und neue) Daten wider?



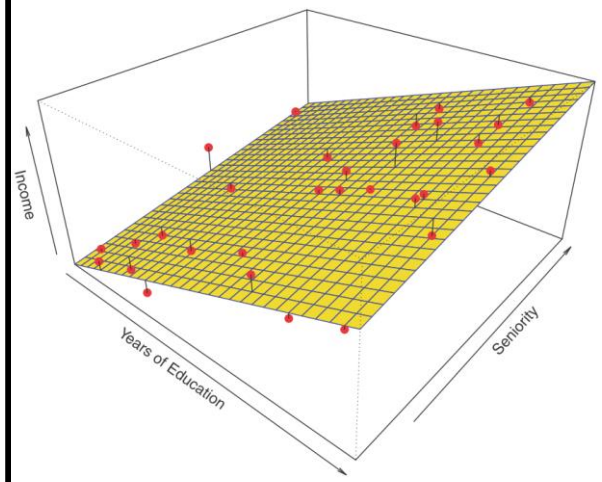
Beispiel-Szenario: Einkommensverhältnisse

Vergleich

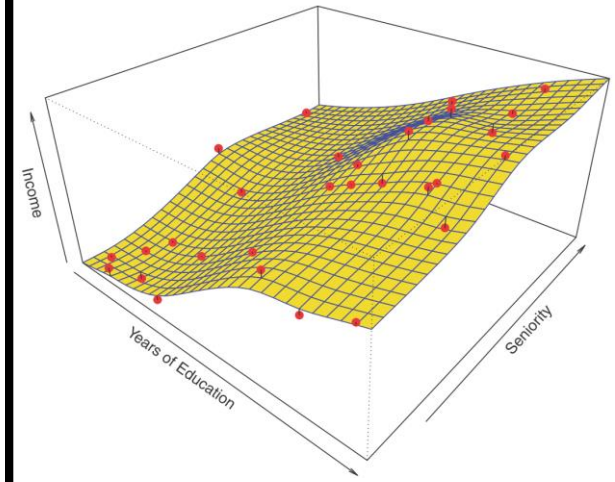
- „exakte“ Funktion φ
- erlernte Modelle φ^



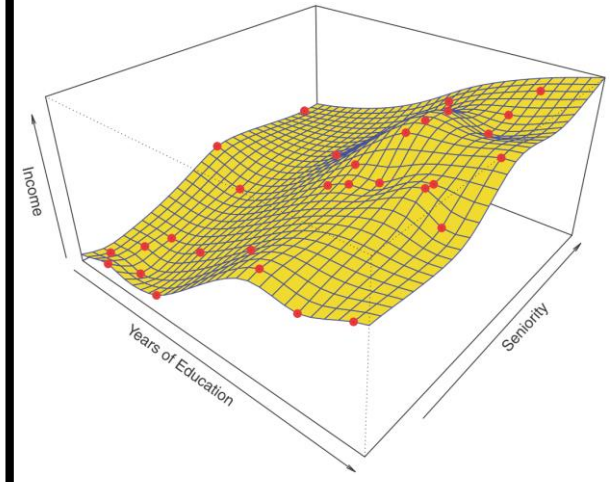
φ^(**YoE,Sen**)= a + b***YoE** + c***Sen**



φ^(**YoE,Sen**)= s-spline(**YoE,Sen**)



φ^(**YoE,Sen**)= h-spline(**YoE,Sen**)



Grad der Flexibilität / Komplexität

Validieren der Modelle

Warum Validierung?

- Bewertung der Güte des gewonnenen Wissens und der Modelle
- Wahl des zu verwendenden Modells anhand der Güte
- Ggf. Vorbereitung eines erneuten adaptierten Trainingslaufs
- Das heißt: Gütebestimmung hilft zu entscheiden
  - wann ein Modell geeignet ist
  - wie gut das Modell geeignet ist
  - wann Modellfindung abgeschlossen ist
  - wann ein Modell (später einmal) ungeeignet ist

Wann?

- Vor Anwendung auf neue Daten in Produktivumgebung!
- Warum nicht gleich Einsatz PU?

Validieren der Modelle

Welchen Daten werden für Validierung verwendet?

- **Vorgehensweise:**
  - Trainieren / Erlernen des Modells: **Trainingsdatensatz `TR`**
  - Eigentliche Aufgabe: Anwendung des Modells auf neuen Daten!
    - Also: Gütebestimmung / Validierung des Modells auf **Testdatensatz `TE`**

- **Testdatensatz `TE`:**
  - Eigentliche Validierung der Modelle
  - Testdaten werden **nicht** zum Lernen verwendet
  - Testdaten sind neue Daten aus Sicht des erlernten Modells

Validieren der Modelle

Auswahlkriterien für ein Modell / Lernalgorithmus

- Ohne Annahme über die Daten gibt es kein allgemein zu bevorzugendes Modell
  - No-Free-Lunch-Theorem (D. Wolpert, Neural Computation, Vol.8 (1996))

Validieren der Modelle

Auswahlkriterien für ein Modell / Lernalgorithmus

- Ausweg aus diesem Dilemma
  - Struktur der Daten berücksichtigen und Annahmen über Daten machen
  - Geeignete Modell-Typen auf dieser Grundlage vorauswählen
    - Beispiele für Modelle:
      - Numerische Funktionen
        - » lineare Funktionen bei einfachen Datenstrukturen
        - » Polynom- oder Spline-Funktionen bei komplexeren Datenstrukturen
      - Klassifikatoren
        - » binäre Klassifikatoren für Daten mit zwei Klassen
        - » höhere Klassifikatoren / neuronale Netze für Daten mit mehr Klassen
  - Genügend viele geeignete Modelle trainieren
  - Modell mit bester Güte auf den Testdaten auswählen

Aber:

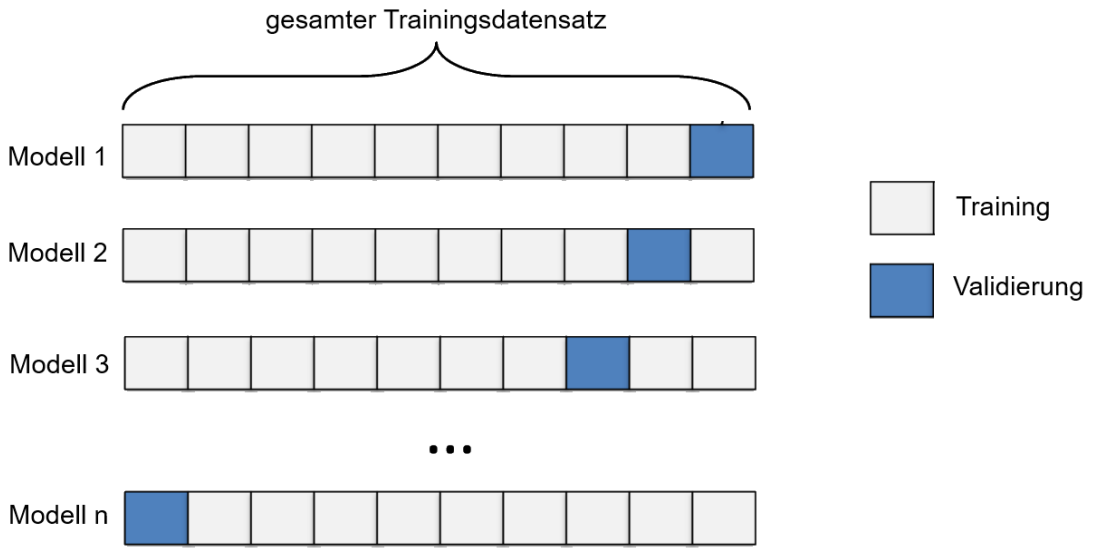
- Auswahl des “besten” Modells ist damit von den Testdaten abhängig!
- Lösung diese Problems: **Kreuzvalidierung**

Validieren der Modelle

Kreuzvalidierung

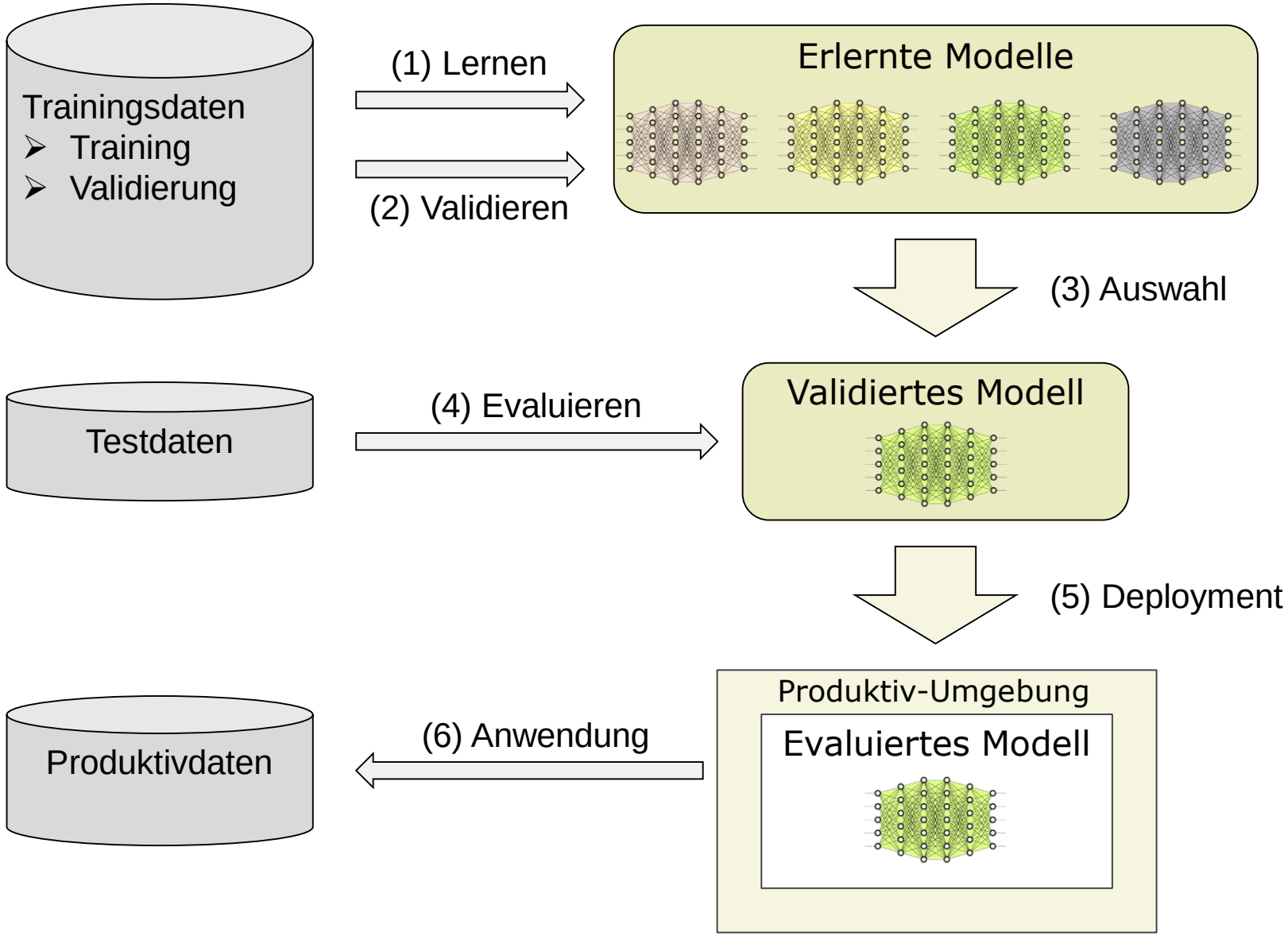
- n Modelle sollen trainiert werden
- Genügend großer ***gesamter Trainingsdatensatz*** wird in disjunkte Untermengen unterteilt
  - Eine dieser Untermengen wird für Training eines Modells verwendet: **Training Fold**
  - Andere Untermenge wird für Evaluierung des Modells verwendet: **Validation Fold**
  - Jedes Modell wird mit einem anderen Validation Fold (Validierungsdatensatz) evaluiert

n Training und Validation Folds:



- Modell mit der besten Güte wird ggf. mit gesamtem Trainingsdatensatz nach-trainiert und mit einem **neuen Testdatensatz** final evaluiert

Validieren der Modelle und Deployment



Validieren der Modelle

Immer noch offen: Wie wird die Güte der Modelle bestimmt?

- Mehrere mögliche Methoden der Gütebestimmung
  - abhängig vom Modelltyp und von der Datenlage

- Zuerst: Modelltypen in Form numerischer Funktionen $\varphi^{\wedge}$
  - Gebräuchliche Gütebestimmung bei Modelltypen dieser Form:
    - Bestimmung der **mittleren quadratischen Abweichung** (MQA)

- Dann: Binäre Klassifikatoren, n-äre Klassifikatoren, Klassifikatoren mit komplexen Klassen, …

Validieren der Modelle

Modelltyp: Numerische Funktionen φ^

- Eine gebräuchliche Gütebestimmung bei Modelltypen dieser Form:
  - Bestimmung der **mittleren quadratischen Abweichung** (MQA)

Validieren der Modelle

Mittlere quadratische Abweichung (MQA / MSE)

- Werte aus einem realem Datensatz **DS**

$$\mathbf{DS} = \{ (\mathbf{X_k}, \mathbf{Y_k})_{k = 1,..,n} \}$$

  mit realen Input-Werten $\mathbf{X_k}$ und realen Zielwerten $\mathbf{Y_k}$

- Modell $\varphi\hat{}$ berechnet Modell-Zielwerte aus den Input-Werten $\mathbf{X_k}$
- Mittlere Quadratische Abweichung zwischen Modell- und Real-Zielwerten

$$\mathbf{MQA}_{\varphi\hat{}}(\mathbf{DS}) = \frac{1}{n} \sum_{k=1}^{n} (\varphi\hat{} (\mathbf{X_k}) - \mathbf{Y_k})^2$$

- Warum wird quadratische Abweichung verwendet?

Validieren der Modelle

Mittlere quadratische Abweichung (MQA / MSE)

Beispiel: Einkommensverhältnisse

- **Variablen**
  - **Input-Variablen:** Ausbildungsdauer (**`YoE`**), Berufserfahrung (**`Sen`**)
  - **Zielvariablen:** Einkommen (**`Inc`**)
- Datensatz $\mathbf{DS} = \{ (\mathbf{YoE}_k, \mathbf{Sen}_k, \mathbf{Inc}_k)_{k = 1,..,n} \}$
- MQA für Modell $\varphi\hat{}$ auf Datensatz **`DS`**

$$\mathbf{MQA}_{\varphi\hat{}}(\mathbf{DS}) = \frac{1}{n} \sum_{k=1}^{n} (\varphi\hat{}(\mathbf{YoE}_k, \mathbf{Senk}) - \mathbf{Inc}_k)^2$$

Validieren der Modelle

Mittlere quadratische Abweichung (MQA / MSE)

Beispiel: Einkommensverhältnisse

- Betrachte zwei Datensätze
  - Trainingsdatensatz $\mathbf{TR} = \{ (\mathbf{YoE}_{\mathbf{TR},k}, \mathbf{Sen}_{\mathbf{TR},k}, \mathbf{Inc}_{\mathbf{TR},k})_{k = 1,..,n} \}$
  - Testdatensatz $\mathbf{TE} = \{ (\mathbf{YoE}_{\mathbf{TE},r}, \mathbf{Sen}_{\mathbf{TE},r}, \mathbf{Inc}_{\mathbf{TE},r})_{r = 1,..,m} \}$
- MQA für Modell $\varphi\hat{}$ auf den Datensätzen $\mathbf{TR}$ und $\mathbf{TE}$

$$\mathbf{MQA}_{\varphi\hat{}}(\mathbf{TR}) = \frac{1}{n} \sum_{k=1}^{n} (\varphi\hat{} (\mathbf{YoE}_{\mathbf{TR},k}, \mathbf{SenTR}_{,k}) - \mathbf{Inc}_{\mathbf{TR},k})^2$$

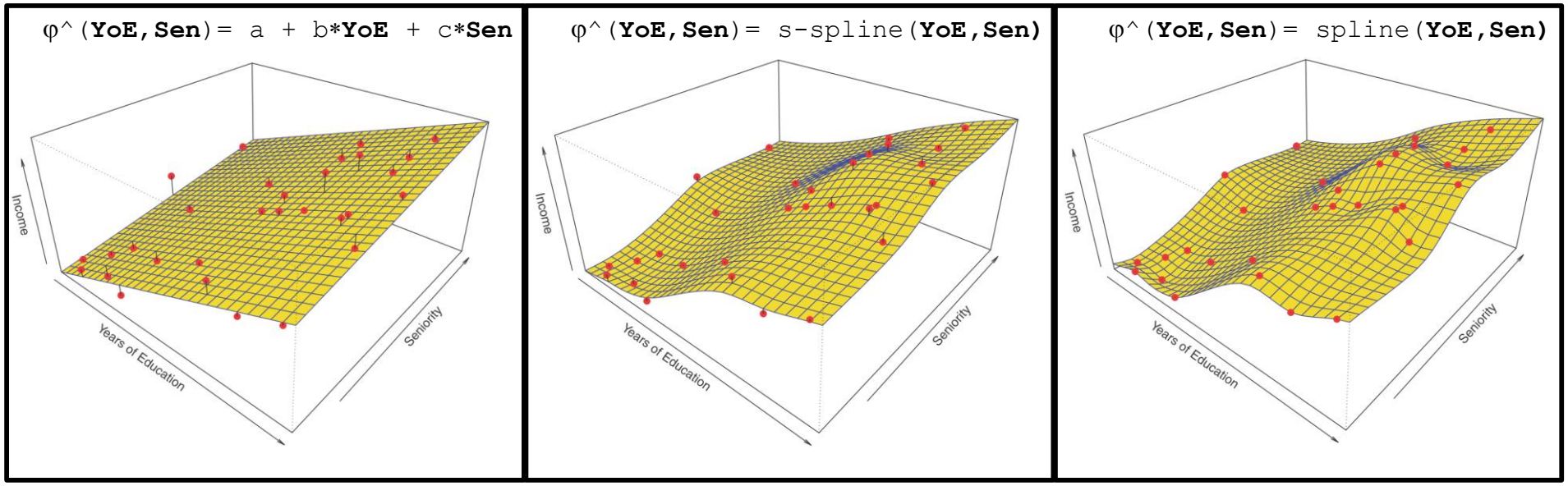
$$\mathbf{MQA}_{\varphi\hat{}}(\mathbf{TE}) = \frac{1}{m} \sum_{r=1}^{m} (\varphi\hat{} (\mathbf{YoE}_{\mathbf{TE},r}, \mathbf{SenTE}_{,r}) - \mathbf{Inc}_{\mathbf{TE},r})^2$$

Validieren der Modelle

Beispiel: Einkommensverhältnisse

Vergleich von $\mathbf{MQA}_{\hat{\varphi}}(\mathbf{TR})$ und $\mathbf{MQA}_{\hat{\varphi}}(\mathbf{TE})$ in den jeweiligen Modellen

- Wähle Modell mit bester Güte $\mathbf{MQA}_{\hat{\varphi}}(\mathbf{TE})$ auf den **Testdaten**!



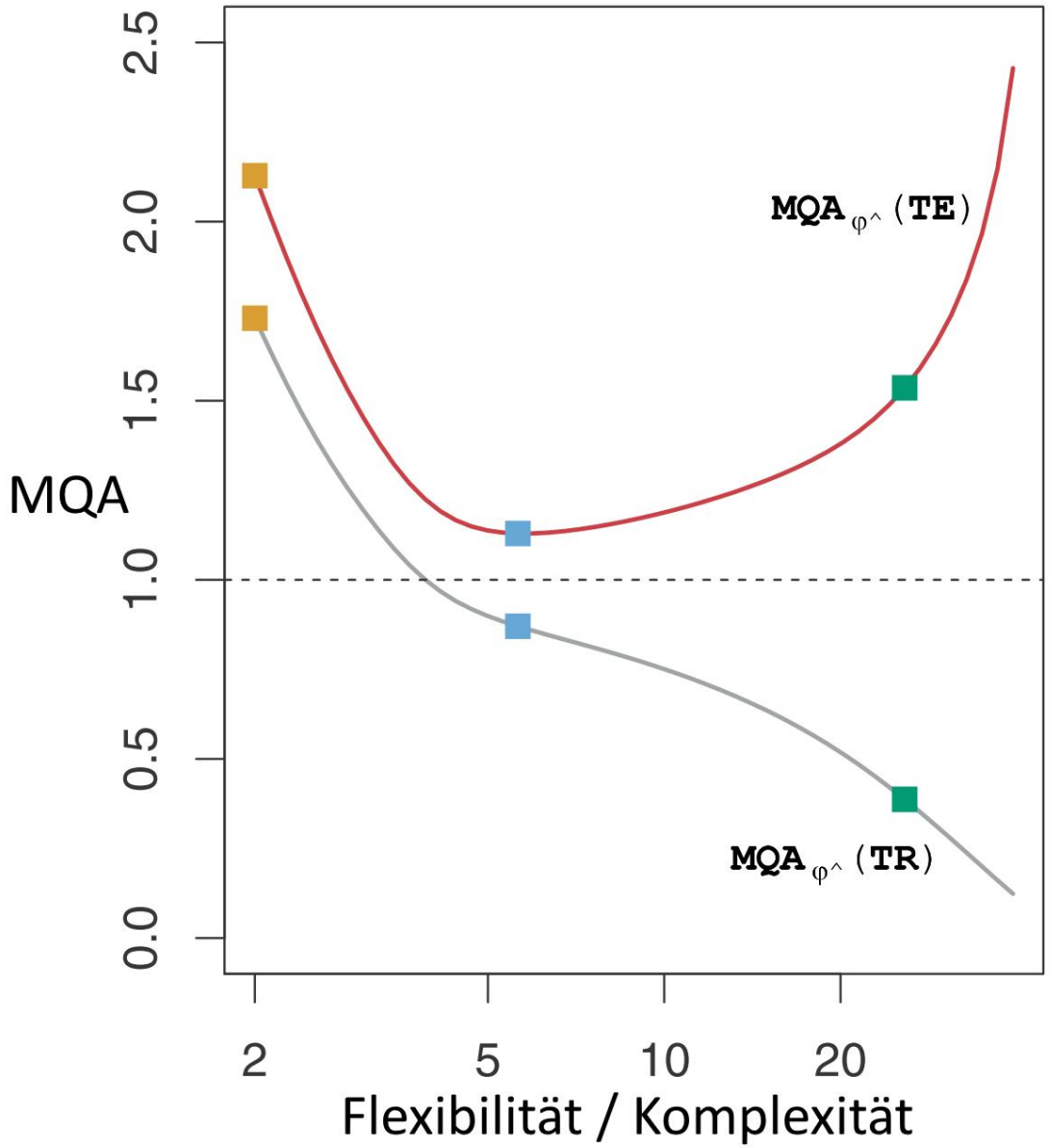
Validieren der Modelle

Bias-Variance Trade-Off (Verzerrung-Varianz-Konflikt)

- Generelles Problem bei Wissensgewinnung durch (statistische) Lernalgorithmen auf endlichen Trainingsdatenmengen
- Resultiert aus gleichzeitiger Minimierung zweier gegenläufig optimierender Fehlerquellen
  - Verzerrung und Varianz
- Verzerrung: Fehler aufgrund zu geringer Flexibilität / Komplexität des Modells
  - Underfitting
- Varianz: Fehler aufgrund zu hoher Flexibilität / Komplexität des Modells
  - Overfitting

Validieren der Modelle

Bias-Variance Trade-Off



Modelltyp: Binärer Klassifikator

- Gebräuchliche Gütemerkmale bei Modelltypen dieser Form:
  - Wahrheitsmatrix (oder Konfusionsmatrix)
  - Präzision (Relevanz) / negative Präzision
  - Trefferquote (Sensitivität) / negative Trefferquote
  - Ausfallrate / negative Ausfallrate
  - F-Maß (engl.: F-score)

Validieren der Modelle

Aber zuerst: Was ist ein binärer Klassifikator?

- **Annahme:** Die Daten (= alle Datenobjekte als Menge $\mathcal{O}$ betrachtet) können in zwei Klassen `false` und `true` unterteilt werden.
  - Jedem Datenobjekt $O \in \mathcal{O}$ kann somit ein (tatsächlicher) Klassenwert `false` oder `true` zugeordnet werden.

- Ein **binärer Klassifikator** K ist eine Abbildung von der Menge der Datenobjekte $\mathcal{O}$ in die Menge {`false` , `true`}

$$K: \begin{cases} \mathcal{O} \rightarrow \{\texttt{false}, \texttt{true}\} \\ O \rightarrow K(O) \end{cases}$$

- Ein binärer Klassifikator ist also ein **Modell für die Klassifikation** dieser Daten.

Validieren der Modelle

Sachverhalt:

- Einerseits: Datenobjekte O in der Menge $\mathcal{O}$ sind einem tatsächlichen Klassenwert w_O = `false` oder `true` zugeordnet
  - Bezeichnung: (O, w_O)
- Andererseits: Klassifikator K bildet ein Datenobjekt O in $\mathcal{O}$ auf eine Klasse K(O) ab

- Klassifikator K wurde auf Trainingsdaten $\mathbf{TR} \subset \mathcal{O}$ trainiert
- Klassifikator K wird nun auf Testdaten $\mathbf{TE} \subset \mathcal{O}$ evaluiert

- **Ziel:** Klassifikator K soll so genau wie möglich die tatsächlichen Klassenwerte bestimmen: $K(O) \cong w_O$

Validieren der Modelle

- Was bedeutet “so genau wie möglich“?
  - Evaluiere Klassifikator K durch Gütemaße

- Gütemaße für binäre Klassifikatoren:
  - Wahrheitsmatrix (Confusion Matrix)
  - Relevanz
  - Sensitivität
  - Ausfallrate
  - Genauigkeit
  - Spezifität
  - F-Maß (engl.: F-score)

Validieren der Modelle

		Prediction	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

Wahrheitsmatrix (Confusion Matrix)

- true positive: $\mathbf{TP} = \{O \in \mathbf{TE} \mid K(O) = \texttt{positive} \text{ und } w_o = \texttt{positive}\}$
  - Testdatenobjekte, die richtigerweise von K als `positive` bewertet werden
- true negative: $\mathbf{TN} = \{O \in \mathbf{TE} \mid K(O) = \texttt{negative} \text{ und } w_o = \texttt{negative}\}$
  - Testdatenobjekte, die richtigerweise von K als `negative` bewertet werden
- false positive: $\mathbf{FP} = \{O \in \mathbf{TE} \mid K(O) = \texttt{positive} \text{ und } w_o = \texttt{negative}\}$
  - Testdatenobjekte, die fälschlicherweise von K als `positive` bewertet werden
- false negative: $\mathbf{FN} = \{O \in \mathbf{TE} \mid K(O) = \texttt{negative} \text{ und } w_o = \texttt{positive}\}$
  - Testdatenobjekte, die fälschlicherweise von K als `negative` bewertet werden

Validieren der Modelle

Wahrheitsmatrix (Confusion Matrix)

Satz

- **TP**, **TN**, **FP**, **FN** sind zueinander disjunkte Untermengen von **TE**
- **TP** $\cup$ **TN** $\cup$ **FP** $\cup$ **FN** = **TE**
- **TP** $\cup$ **FP** = $\{O \in \mathbf{TE} \mid K(O) = \texttt{positive}\}$
- **TN** $\cup$ **FN** = $\{O \in \mathbf{TE} \mid K(O) = \texttt{negative}\}$
- **TP** $\cup$ **FN** = $\{O \in \mathbf{TE} \mid w_O = \texttt{positive}\}$
- **TN** $\cup$ **FP** = $\{O \in \mathbf{TE} \mid w_O = \texttt{negative}\}$
- **TP** $\cup$ **TN** = $\{O \in \mathbf{TE} \mid K(O) = w_O\}$
- **FP** $\cup$ **FN** = $\{O \in \mathbf{TE} \mid K(O) \neq w_O\}$

Definition (Wahrheitswerte)

tp = **|TP|** **tn** = **|TN|** **fp** = **|FP|** **fn** = **|FN|**

Beachte: Für eine Menge **M** bezeichnet **|M|** die Anzahl der Elemente in **M**

Folge

tp + tn + fp + fn = |TE|

Validieren der Modelle

Weitere Gütemaße (I)

- **Relevanz** (Präzision)

$$\frac{tp}{tp + fp}$$

Anteil der richtigerweise von K als `positive` bewerteten Objekte an der Gesamtheit aller von K als `positive` bewerteten Objekte

- **Sensitivität** (Trefferquote)

$$\frac{tp}{tp + fn}$$

Anteil der richtigerweise von K als `positive` bewerteten Objekte an der Gesamtheit aller `positive` klassifizierten Objekte

- **Ausfallrate** (Falsch-positiv-Rate)

$$\frac{fp}{tn + fp}$$

Anteil der fälschlicherweise von K als `positive` bewerteten Objekte an der Gesamtheit aller `negative` klassifizierten Objekte

Validieren der Modelle

Weitere Gütemaße (II)

- **Genauigkeit**

$$\frac{tp + tn}{tp + tn + fp + fn}$$

Anteil aller von K korrekt klassifizierten Objekte an der Gesamtheit aller Objekte

- **Spezifität** (negative Trefferquote)

$$\frac{tn}{tn + fp}$$

Anteil der richtigerweise von K als `negative` klassifizierten Objekte an der Gesamtheit aller `negative` klassifizierten Objekte

Validieren der Modelle

Weitere Gütemaße (III)

- **F-Score** (Harmonisches Mittel von Relevanz und Sensitivität)

$$\frac{2 \cdot \text{Relevanz} \cdot \text{Sensitivität}}{\text{Relevanz} + \text{Sensitivität}} = \frac{2\,\text{tp}}{2\,\text{tp} + \text{fp} + \text{fn}}$$

  - F-Score nimmt große Werte an, wenn Relevanz **und** Sensitivität groß genug sind
  - Nützliches Gütemaß für Daten, deren Klassen signifikant ungleich verteilt sind.
  - F-Score begünstigt Klassifikatoren mit ähnlicher (großer) Relevanz und Sensitivität. Das ist nicht immer zielführend!
    - Kinderschutzklassifikation für Videos: erwünscht ist hohe Relevanz $\frac{\text{tp}}{\text{tp}+\text{fp}}$
    - Erkennung Einkaufsdiebstahl: erwünscht ist hohe Sensitivität $\frac{\text{tp}}{\text{tp}+\text{fn}}$

Validieren der Modelle

Zusammenfasssung

- Mehrere unterschiedliche Gütemaße für binäre Klassifikatoren
- Auswahl von Datenzusammensetzung abhängig

- Im Folgenden exemplarisch: Klassifikationsszenario des Datensatzes MNIST

Klassifikation – MNIST

Beispiel-Szenario im Bereich Klassifikation

MNIST – annotierter Datensatz

- Datensatz mit 70.000 handschriftlichen Bildern der Ziffern 0, 1, … , 9
- Jedes Bild (Datenobjekt) ist mit dem Wert der dargestellten Ziffer annotiert
- Hello-World-Beispiel des Machine Learning
- Beispiel-Datenobjekt:

Lernalgorithmus: Stochastic Gradient Descent

- Beispiel eines schwellenwertbasierten Klassifikators

Klassifikation – MNIST

- Beschaffung des MNIST-Datensatzes

```
import sklearn.datasets as sklds

X,ystr = sklds.fetch_openml('mnist_784', version=1, return_X_y=True)

# Transformation von String-Array in int-Array für späteren Gebrauch
y = np.array([int(s) for s in ystr])
```

Klassifikation – MNIST

- Erzeugung eines Bildes aus einem Datenobjekt `X[n]`
  - Merkmalsvektor `X[n]` zu 28×28-Array umformatieren
  - Darstellung mit Funktion `imshow()` aus `matplotlib`

```
import matplotlib
import matplotlib.pyplot as plt

some_digit = X[36123]
some_digit_image = some_digit.reshape(28, 28)

plt.imshow(some_digit_image, cmap = matplotlib.cm.binary,
           interpolation="nearest")
plt.axis("off")
plt.show()
```

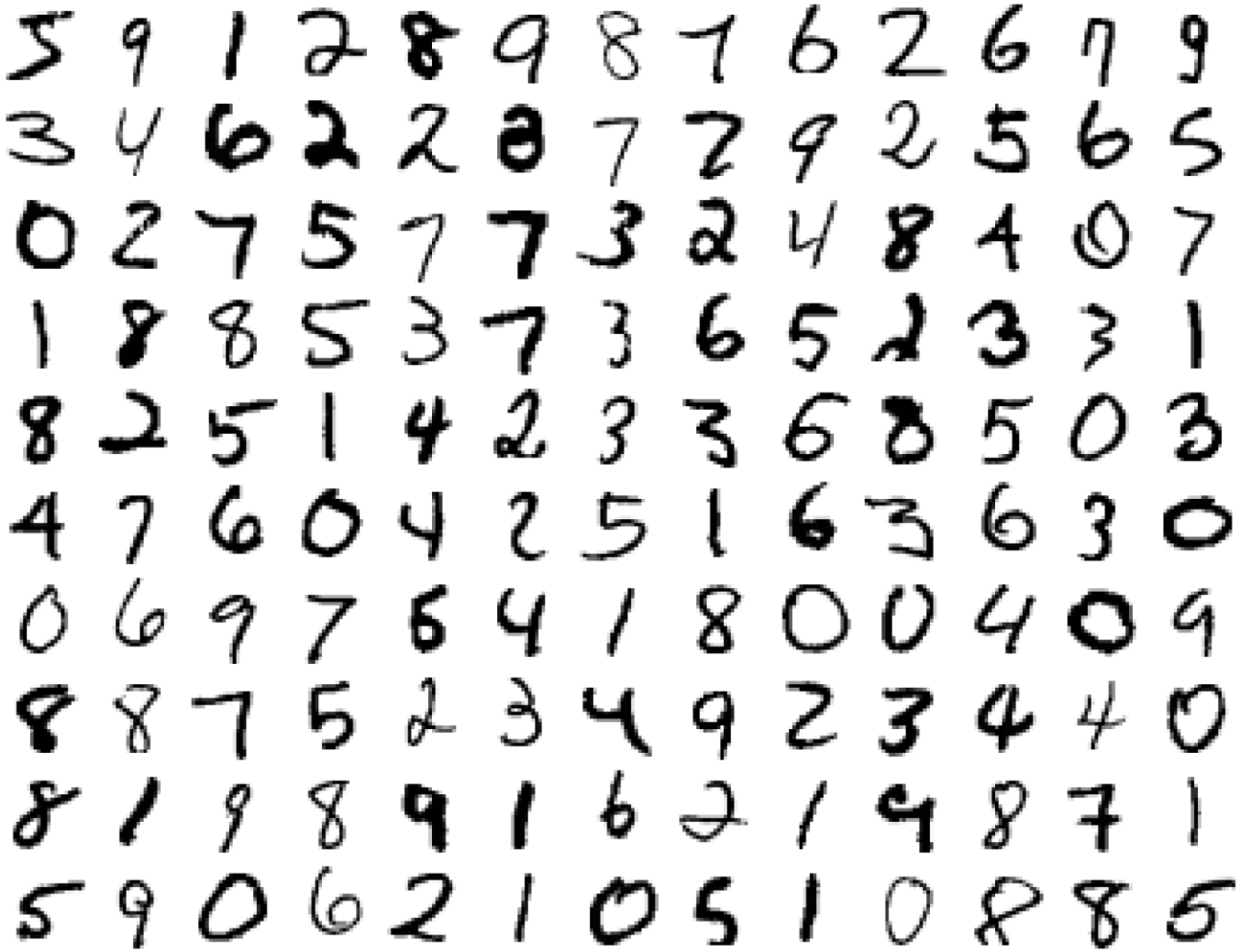
Klassifikation – MNIST

- Erzeugung eines Bildes aus einem Datenobjekt `X[n]`
  - Merkmalsvektor `X[n]` zu 28×28-Array umformatieren
  - Darstellung mit Funktion `imshow()` aus `matplotlib`

- Beispiel: (`X[36123]`, `y[36123] = 2`)

Klassifikation – MNIST

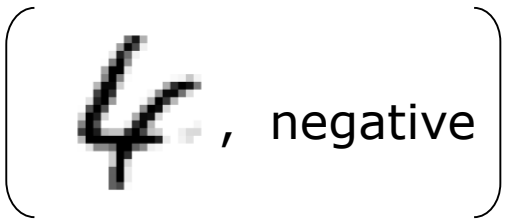
- Weitere handschriftliche Ziffern aus MNIST



Klassifikation – MNIST

MNIST – **binärer annotierte Trainings- und Testdatensätze**

- Jedes Bild (Datenobjekt) ist mit dem Wert `positive` oder `negative` annotiert
  - `positive`: Ziffer ist 5
  - `negative`: Ziffer ist nicht 5
- Beispiel-Datenobjekt:



- Die zwei Klassen sind ungleich verteilt: `#negative : #positive ≈ 9 : 1`
- Annotierte Datensätze müssen selbst erstellt werden
- Klassifikator für diesen binären Datensatz: Art „5er“-Detektor

Klassifikation – MNIST

Annotierte Datensätze (mit 10 bzw. 2 Klassen (binär))

- Erzeugung der Trainings- und Testdatensätze

```
shuffle_index = np.random.permutation(60000)

X_train = X[:60000]
y_train = y[:60000]

X_train = X_train[shuffle_index]   # 60.000 Objekte
y_train = y_train[shuffle_index]   # 60.000 Labels (zehn Klassen)
y_train_5 = (y_train == 5)         # 60.000 Labels (zwei Klassen)

X_test  = X[60000:]                # 10.000 Objekte
y_test  = y[60000:]                # 10.000 Labels (zehn Klassen)
y_test_5 = (y_test == 5)           # 10.000 Labels (zwei Klassen)
```

Klassifikation – MNIST

Erstellen und Training eines binären „5er“-Klassifikators

- Beispiel Lernalgorithmus: **Stochastisches Gradientenverfahren**
- Scikit-Learn: `SGDClassifier`
  - Erzeugung und Training eines `SGDClassifier`

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(max_iter=5, random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

- **Bisher: Nur Training auf einem Trainingsdatensatz**
  - Keine Kreuzvalidierung
  - Noch keine Gütebestimmung

Klassifikation – MNIST

- **Jetzt: Kreuzvalidierung (Training und Gütebestimmung)**
  - … und Bestimmung des Gütemaßes **Genauigkeit** („Accuracy“)

$$\frac{tp + tn}{tp + tn + fp + fn}$$

Anteil aller von K richtig klassifizierten Objekte aus Gesamtheit aller Objekte

- 3-faches Trainieren (3 Folds) und Kreuzvalidierung

```
from sklearn.model_selection import cross_val_score

cross_val_score(sgd_clf, X_train, y_train_5 ,cv=3 ,scoring="accuracy")
```

Klassifikation – MNIST

- **Jetzt: Kreuzvalidierung (Training und Gütebestimmung)**
  - … und Bestimmung des Gütemaßes **Genauigkeit** („Accuracy“)

$$\frac{tp + tn}{tp + tn + fp + fn}$$

Anteil aller von K richtig klassifizierten Objekte aus Gesamtheit aller Objekte

- 3-faches Trainieren (3 Folds) und Kreuzvalidierung
- Ergebnis:

```
[0.9605, 0.95595, 0.95375]
```

- Wie bewerten Sie dieses Ergebnis?

Klassifikation – MNIST

Erstellen und Training eines weiteren „5er“-Klassifikators

- Trivialer Klassifikator (Eigenbau): **`Never5Classifier`**
  - ordnet **jedem Bild** die Klasse `negative` (“nicht-5“) zu

```
from sklearn.base import BaseEstimator

class Never5Classifier(BaseEstimator):
        def fit(self, X, y=None):
                pass
        def predict(self, X):
                return np.zeros((len(X), 1), dtype=bool)
```

- **Aber:** (Noch) keine Kreuzvalidierung

Klassifikation – MNIST

Vergleich des Gütemaßes **Genauigkeit** für `sgd_clf` und `never_5_clf`

```
never_5_clf = Never5Classifier()

print(cross_val_score(sgd_clf, X_train, y_train_5,
                      cv=3, scoring="accuracy"))

print(cross_val_score(never_5_clf, X_train, y_train_5,
                      cv=3, scoring="accuracy"))
```

- Ergebnis:

```
sgd_clf:     [0.9532,  0.95125, 0.9625 ]
never_5_clf: [0.91125, 0.90855, 0.90915]
```

- Never5Classifier `never_5_clf` bewertet mindestens 90% aller Objekte korrekt.
- Warum hat `never_5_clf` eine so hohe Genauigkeit?

Klassifikation – MNIST

- Schlussfolgerung
  - Gütemaß **Genauigkeit** nicht präzise/aussagekräftig genug, insbesondere bei stark ungleich verteilten Klassenzuordnungen
- Besser:
  - Detailliertere Auswertung durch **Wahrheitsmatrix** (Confusion Matrix)
    - enthält detaillierte Information über binären Klassifikator, aber oft zu detailliert
    - in vielen Fällen die Kombination kompakterer Gütemaße ausreichend
      - **Relevanz (Precision)**
      - **Sensitivität (Recall)**

      (dazu später mehr)

Klassifikation – MNIST

Wahrheitsmatrix

- Zuerst 3-fache Kreuzvalidierung und Erzeugung der Vorhersagewerte

```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

- Dann Erzeugung der Wahrheitsmatrix
  - Vergleich der Vorhersagen mit den tatsächlichen Klassenwerten

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_train_5, y_train_pred)
```

- Ergebnis:

```
Wahrheitsmatrix:
[ tn = cm[0,0] | fp = cm[0,1] ]        [52972 | 1607]
[ fn = cm[1,0] | tp = cm[1,1] ]   =    [ 989  | 4432]
```

Klassifikation – MNIST

Gütemaß **Relevanz (Precision)**

```
from sklearn.metrics import precision_score

precision_score(y_train_5, y_train_pred)

# ... oder explizit:
precision = tp /(tp + fp)
```

- **Ergebnis**

```
precision:     0.7338963404537175
```

73% der vom Klassifikator als 5 erkannten Bilder sind tatsächlich ein 5-Bild

Klassifikation – MNIST

Gütemaß **Sensitivität (Recall)**

```
from sklearn.metrics import recall_score

recall_score(y_train_5, y_train_pred))

# ... oder explizit:
recall = tp /(tp + fn)
```

- **Ergebnis**

```
recall:     0.8175613355469471
```

Von allen 5-Bildern werden 81% als solche vom Klassifikator richtigerweise erkannt

Klassifikation – MNIST

Zusammenhang von Relevanz und Sensitivität

- Wechselbeziehung bei schwellenwertbasierten binären Klassifikatoren
  - auf der Grundlage von Wahrscheinlichkeiten oder Scores
  - allgemeiner Sachverhalt

- Anforderung an Relevanz und Sensitivität
  - bestimmt die Wahl des Schwellenwertes des Klassifikators

- Viele binäre Klassifikatoren sind schwellenwertbasiert
  - auch: Stochastic Gradient Descent Classifier `SGDClassifier` in Scikit-Learn

Klassifikation – MNIST

Zusammenhang von Relevanz und Sensitivität

Qualitative Darstellung am Beispiel des `SGDClassifier` und der Klassifikation der MNIST-Daten

- Für jeden Datenpunkt (Bild) des Datensatzes berechnet die Entscheidungsfunktion von `SGDClassifier` einen Score
- Liegt Score über dem von `SGDClassifier` vorgegebenen Schwellenwert, ordnet `SGDClassifier` dem Datenpunkt die Kategorie `positive` zu, ansonsten die Kategorie `negative`

- Jedem Datenpunkt ordnet `SGDClassifier` somit folgende Werte zu:
  - Score
  - tatsächliche Klasse `positive` oder `negative`

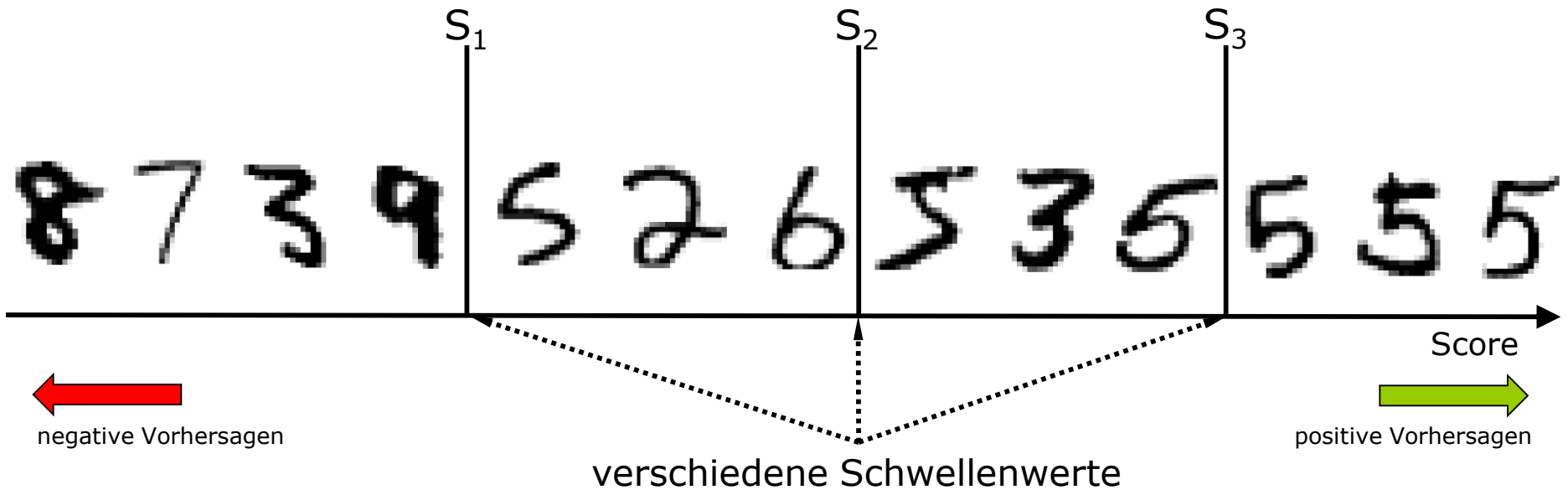
Folgerung:

- Aus Score und Klasse für jeden Datenpunkt und (beliebigem) Schwellenwert kann somit
  - der Vorhersagewert (`positive` oder `negative`) für jeden Datenpunkt
  - die Wahrheitsmatrix für den gesamten Datensatz (insbesondere Relevanz und Sensitivität)

  ermittelt werden

Klassifikation – MNIST [Beispiel “Is-5-Classifier“]

Relevanz $\frac{\text{tp}}{\text{tp+fp}}$:	$\frac{6}{9} = 0,\bar{6} \approx 67\%$	$\frac{5}{6} = 0{,}8\bar{3} \approx 83\%$	$\frac{3}{3} = 1 = 100\%$
Sensitivität $\frac{\text{tp}}{\text{tp+fn}}$:	$\frac{6}{6} = 1 = 100\%$	$\frac{5}{6} = 0{,}8\bar{3} \approx 83\%$	$\frac{3}{6} = 0{,}5 = 50\%$



Gegeben:
- Für jeden Datenpunkt im Datensatz: Klasse und Score
- Schwellenwerte S_1, S_2, S_3

Ermittle: Relevanz und Sensitivität (je Schwellenwert)

Klassifikation – MNIST

- Scores `y_scores` des MNIST-Trainingsdatensatzes `X_train` können mit Hilfe des `SGDClassifier` ermittelt werden (*):

```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method="decision_function")
```

- Außerdem liegen für den Trainingsdatensatz `X_train` die Klassen vor: `y_train_5`

- Damit können für einen beliebigen Schwellenwert die Relevanz und Sensitivität des Trainingsdatensatzes ermittelt werden.

(*) Das sollte in der Praxis auf den Test- oder Validierungsdaten erfolgen.

Klassifikation – MNIST

- Ermittlung von Relevanz und Sensitivität des Trainingsdatensatzes
  - mit Hilfe von Scikit-Learn-Funktion `precision_recall_curve`
  - zu verschiedenen Schwellenwerten
  - aus den Klassen `y_train_5` und Scores `y_scores` des Datensatzes
  - Ergebnis: Arrays `relevanzen, sensitivitaeten, thresholds`

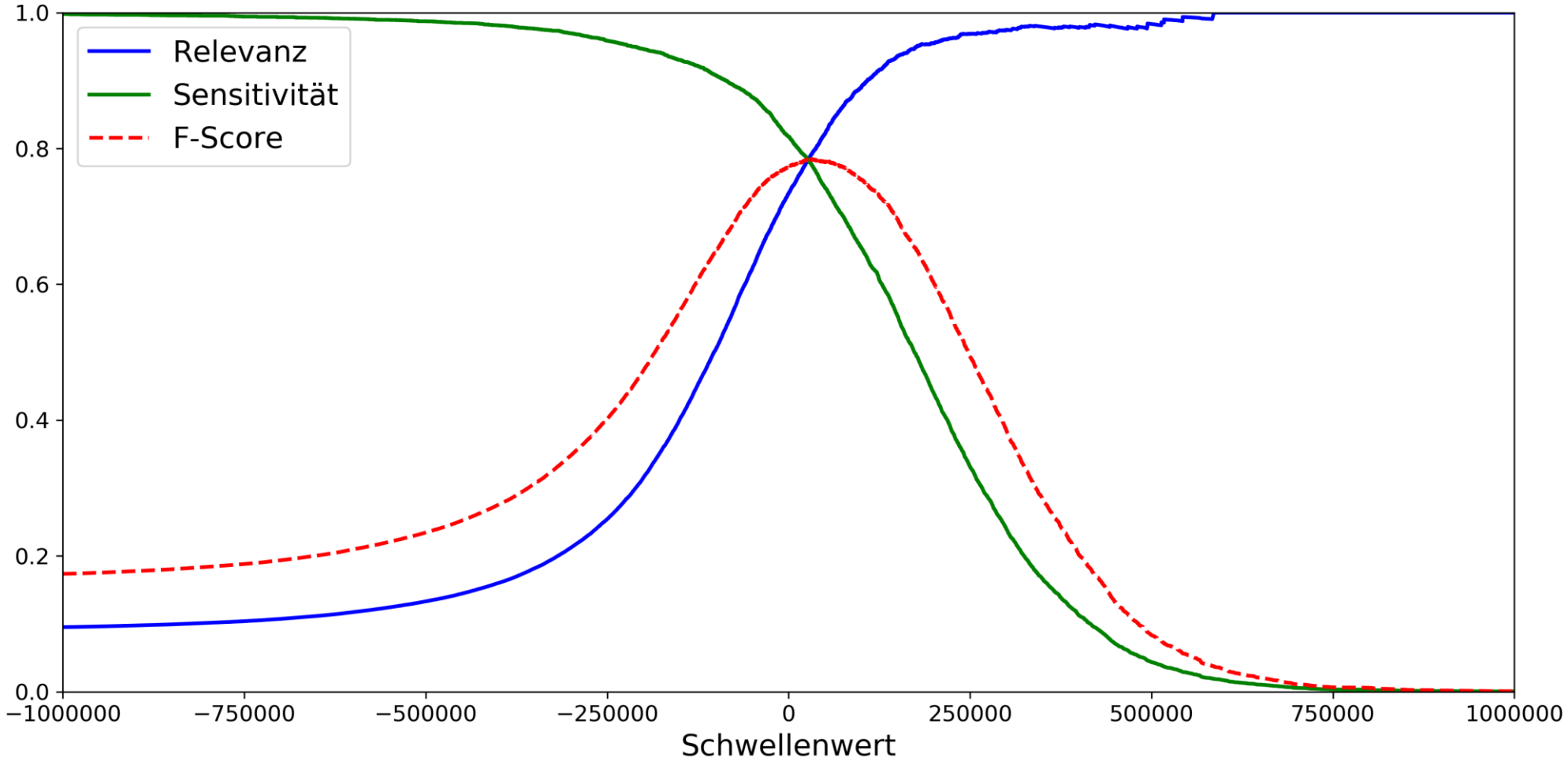
```
from sklearn.metrics import precision_recall_curve

relevanzen, sensitivitaeten, thresholds =
                    precision_recall_curve(y_train_5, y_scores)
```

- Kurze Diskussion dieses Codes!

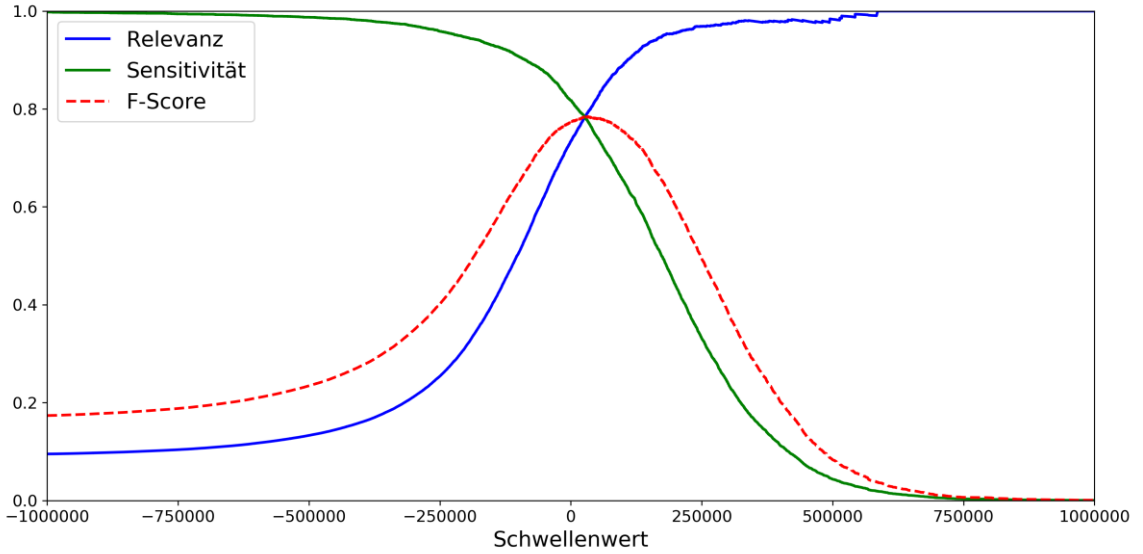
Klassifikation – MNIST

- Grafische Darstellung der Relevanzen und Sensitivitäten über den Schwellenwerten



Klassifikation – MNIST

- Anforderung (durch die Aufgabenstellung) an Relevanz und Sensitivität
  - bestimmt die Wahl des Klassifikators bzw. die Wahl des Schwellenwertes
- Recht einfach, einen Klassifikator mit beliebiger Relevanz oder mit beliebiger Sensitivität zu erstellen
  - aber nicht für Relevanz und Sensitivität gleichzeitig
- Häufiger Anwendungsfall: Relevanz und Sensitivität sollen gute Performance haben
  - Dann muss Kompromiss zwischen Relevanz und Sensitivität gefunden werden
    - z. B. durch Bestimmung des maximalen Wertes des F-Score

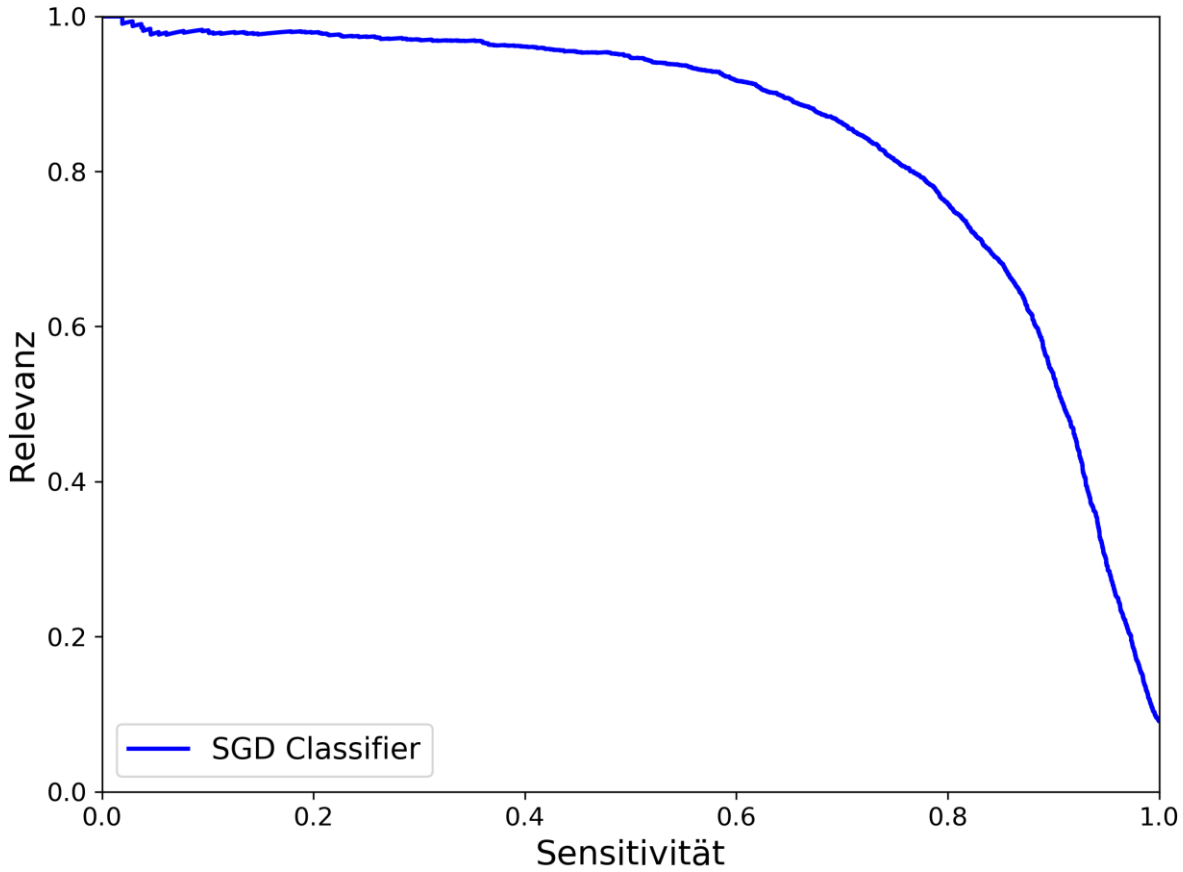


Wenn Sie also den Auftrag bekommen, einen Klassifikator mit einer Relevanz von 90% zu erstellen, müssten Sie z. B. auch fragen, welche Sensitivität der Klassifikator haben soll!

Klassifikation – MNIST

Weitere grafische Darstellung von Gütemaßen schwellenwertbasierter Klassifikatoren

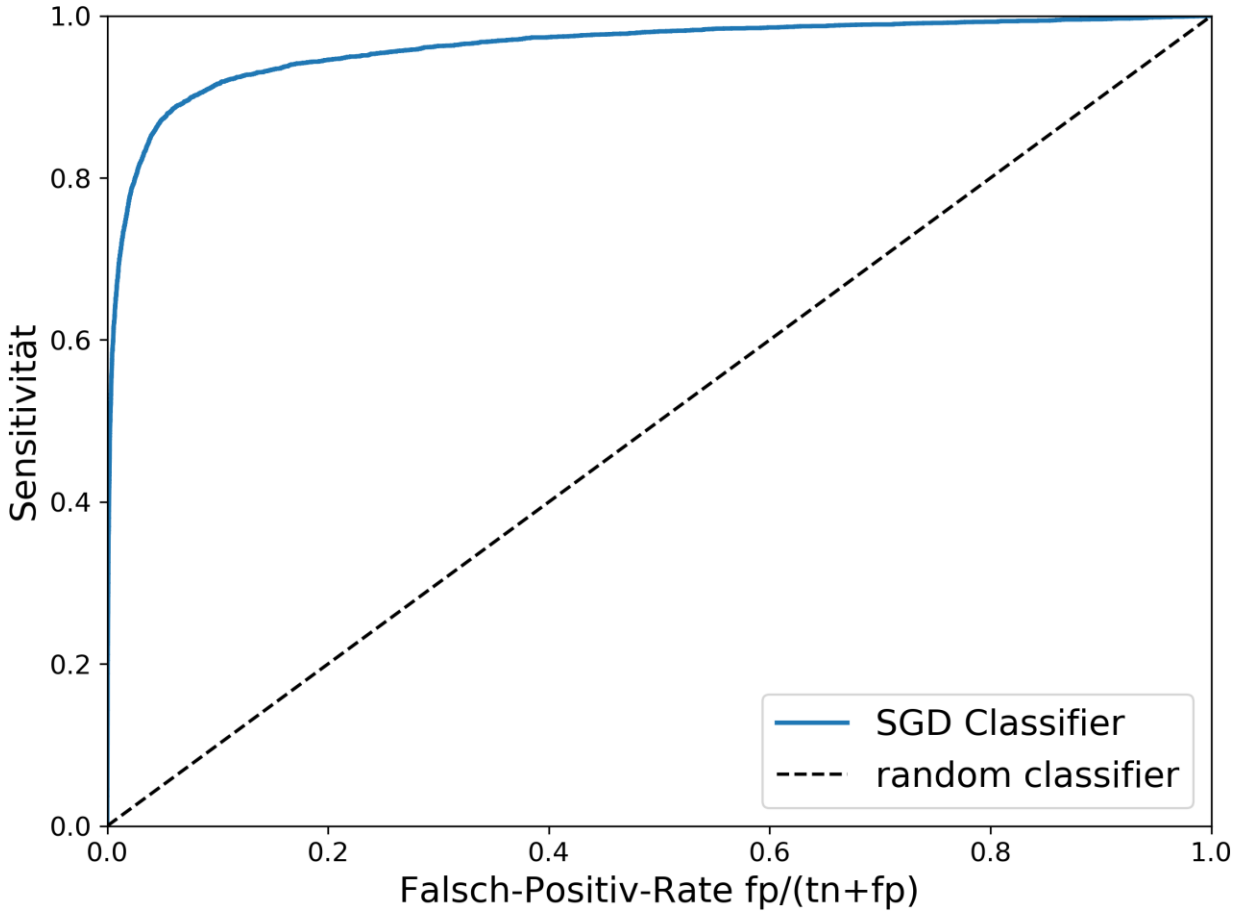
- Relevanz-über-Sensitivität-Kurve



Klassifikation – MNIST

Weitere grafische Darstellung von Gütemaßen schwellenwertbasierter Klassifikatoren

- **ROC Curve** (Sensitivität über Falsch-Positiv-Rate)

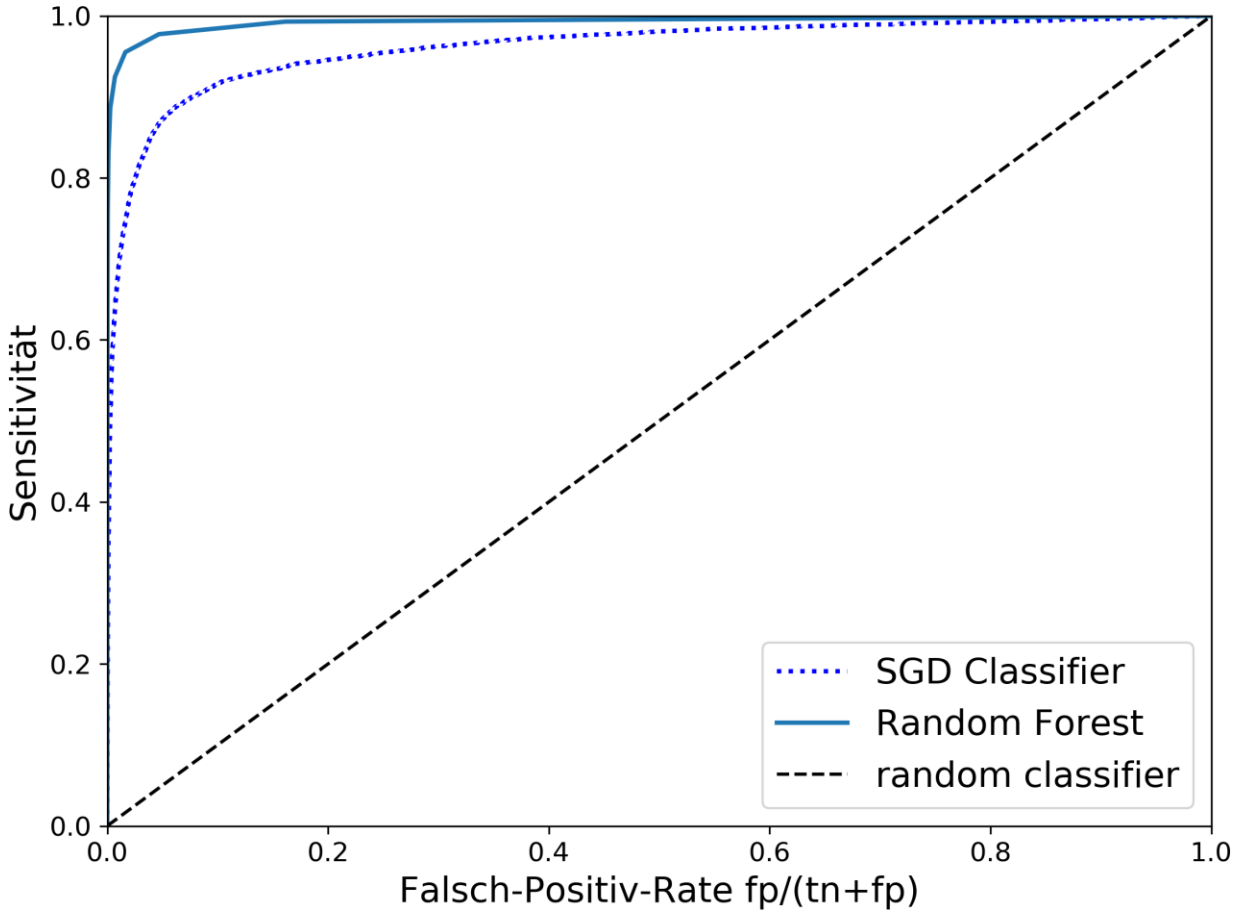


Fläche unter Kurve
- SGD: 0.9614189997126434
- RC: 0.5

Klassifikation – MNIST

Weitere grafische Darstellung von Gütemaßen schwellenwertbasierter Klassifikatoren

- Vergleich zweier ROC Curves



Fläche unter Kurve
- SGD: 0.9614189997126434
- RF: 0.9928250745111685
- RC: 0.5

Gütemaße von RF (!)
- Relevanz: 0.987038664
- Sensitivität: 0.828813871

Klassifikation – MNIST

- Daten, Code, Bilder und weiteres Hands-On für MNIST:
  - Jupyter Notebook `binary_classification_MNIST.ipynb`

Klassifikation – Gesichtserkennung und Statistik

Gesichtserkennung – Projekt der Bundesregierung

- Aussage der staatlichen Stellen:
  - „80% Trefferquote“ (= Genauigkeit?)
  - „0,1% Falsch-Alarm-Rate“ (false-positive-Rate)
- Zahlen und Annahmen
  - 12 Mio. Personen sollen täglich in Bahnhöfen gescanned werden
  - 600 davon sind als potenzielle Gefährder eingestuft
    - Annahme: Täglich befinden sich 100 von ihnen in Bahnhöfen
- Diskussion: Was kann man daraus schließen?
  - 80 von 100 Gefährder werden täglich erkannt
  - von 12 Mio. Personen werden täglich 0,1% oder 12.000 Personen fälschlicherweise als Gefährder erkannt, oder 350.000 Personen im Monat
  - Wahrscheinlichkeit, dass bei Alarm ein Gefährder erkannt wurde, beträgt etwa 80/12.000 oder 0,7%.
  - Also: 99,3% der Alarmeinschätzungen des Systems sind falsch

Fragen?