Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

2024 FS CAS PML - Supervised Learning
4 Validierung (und mehr)
4.4 Performance Metriken

Werner Dähler 2024

4 Validierung und mehr - AGENDA

41. Sampling und Resampling
42. Validierungstechniken
43. Grid Search und Random Search
44. Performance Metriken
    411. Klassifikation
    412. Regression
45. Unbalancierte Daten

4.4 Validierung und mehr - Performance Metriken

Rekapitulation:

- Hauptunterschiede zwischen Klassifikation und Regression
  - Skalierung des Targets (kategorial oder metrisch)
  - Trainingsmethoden (Klassifikator oder Regressor)
  - Performance Metrik (accuracy oder r2)
- daher Gliederung in diesem Kapitel
  4.1.1 Performance Metriken Klassifikation
  4.1.2 Performance Metriken Regression

- unterschiedliche Konzepte bei der Darstellung von Voraussagen von Klassifikationsmodellen
  - nicht numerische Klassen
  - numerische Klassen (binär dargestellt)
  - Wahrscheinlichkeitswerte für einzelne Klassen
- und somit entsprechend nach unterschiedlichen Typen von Performance Metriken

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.1 Vorbereitung

- für die folgenden Darstellungen werden einige Vorbereitungen vorgenommen
  - laden des in FE aufbereiteten Bankkunden Dataset
  - Features - Target - Split
  - Train - Test - Split
  - trainieren eines Modells mit RandomForestClassifier auf dem Trainingsset mit Standard Parametrisierung
  - erstellen von drei unterschiedlichen Formen von Predictions für das Testset
    - **y_pred_c**: nicht numerisch dargestellte Klassen ("no", yes") mit der Methode .predict()
    - **y_pred_n**: binär numerische Umcodierung von y_pred_c, "no" -> 0, "yes" -> 1
    - **y_pred_p**: Wahrscheinlichkeitswerte für die beiden Klassen mit der Methode `.predict_proba()`

  (vgl. [ipynb])

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.1 Vorbereitung

- die Methode .predict_proba() gibt tatsächlich einen `numpy.ndarray` zurück

```
print(y_pred_p[:5,])
```

```
[[0.08 0.92]
 [0.22 0.78]
 [0.36 0.64]
 [0.99 0.01]
 [0.99 0.01]]
```

- mit einer Zeile für jede Prediction und einer Spalte für jede Klasse
- die Zuordnung der Spalten zu den jeweiligen Klassen erfolgt lexikographisch anhand der Klassenbezeichnungen, d.h. erste Spalte für "no", zweite für "yes"
- für die untenstehende Darstellung wurde daher y_pred_p noch wie folgt gesplittet
  - y_pred_p_0: entspricht "no"
  - y_pred_p_1: entspricht "yes"

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.1 Vorbereitung

- Darstellung der im Folgenden zu verwendenden Predictions

	y_pred_c	y_pred_n	y_pred_p_0	y_pred_p_1
0	yes	1	0.08	0.92
1	yes	1	0.22	0.78
2	yes	1	0.36	0.64
3	no	0	0.99	0.01
4	no	0	0.99	0.01
:				

- ausserdem wird für späteren Gebrauch auch vom Target der Testdaten (y_test) eine numerische Darstellung erstellt (y_test_n)

	y_test	y_test_n
2809	yes	1
4052	yes	1
658	yes	1
786	no	0
6675	no	0
:		

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.2 Metriken für nicht numerische Klassen

- bisher wurde ausschliesslich mit accuracy_score gearbeitet
- Vorteile
  - intuitiv verständlich
  - kann mit nicht numerischen Targetwerten (Labels) umgehen
  - auch für Multiklass Fragestellungen geeignet
    (wie das untenstehende Beispiel demonstriert)
- Nachteil
  - Typ 1 und Typ 2 Fehler werden nicht unterschieden (mehr dazu später)

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.2 Metriken für nicht numerische Klassen

- Beispiel einer Multiklass Klassifikation (hier basierend auf dem iris Dataset), vgl. [ipynb], die korrekten Zuordnungen sind hier farblich hervorgehoben

species	setosa	versicolor	virginica
row_0			
setosa	17	0	0
versicolor	0	16	0
virginica	0	1	16

- der Anteil, der korrekt zugeordneten ist somit der Quotient der korrekten gegenüber allen Beobachtungen und kann wie folgt ermittelt werden
  - score Methode des Learners
  - `sklearn.metrics.accuracy_score` (welcher bereits in Kap. 2.1 eingeführt wurde) Konvention bei Aufruf der Performance Metriken bei scikit-learn:
    - erster Parameter: actual (tatsächlicher Wert, hier `y_test`),
    - zweiter Parameter: prediction

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.2 Metriken für nicht numerische Klassen

- eine weitere oft verwendete Metrik: Classification Error (CE), hat in sklearn.metrics keine eigene Methode, kann aber einfach aus accuracy_score errechnet werden:
-

$$classification\_error = 1 - accuracy\_score$$

- weitere Multiklass Metriken von sklearn.metrics:
  - balanced_accuracy_score
    - geeignet für unbalanciertes Datasets
    - ist definiert als Mittelwert des recall_scores (vgl. Kap. 4.4.1.3) für jede Klasse gegenüber alle anderen
    - bester Wert: 1, schlechtester Wert: 0
  - cohen_kappa_score:
    - ein statistisches Mass für Interrater Zuverlässigkeit (vgl. Wikipedia)
    - bester Wert: +1, schlechtester Wert: -1

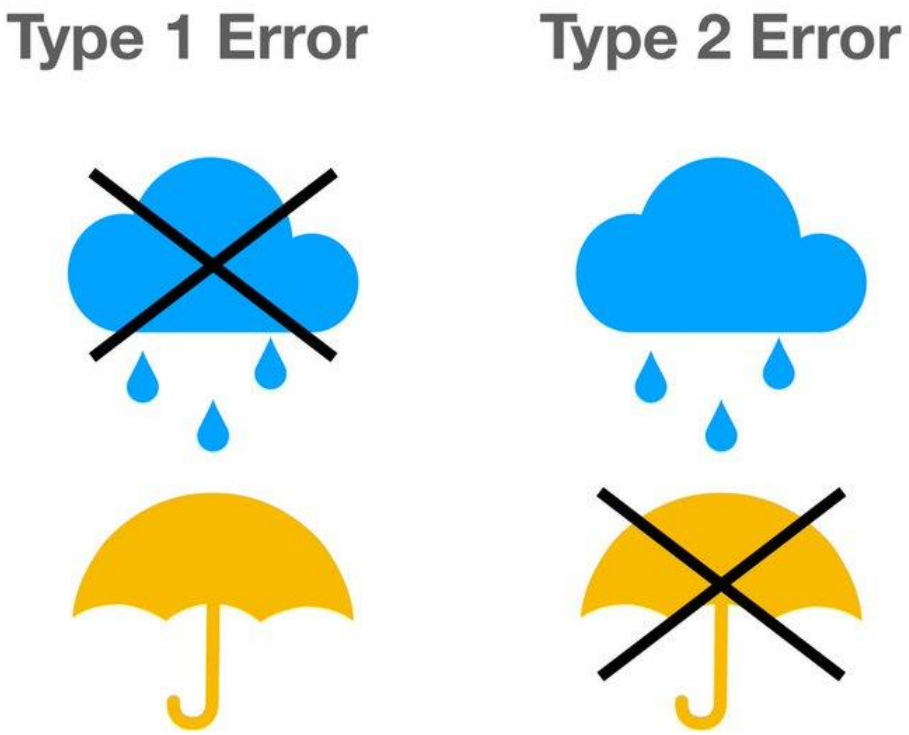
4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

- Multiklass-Metriken können selbstverständlich auch auf binäre Klassifikationen (zwei Klassen Probleme) angewendet werden - was bei gewissen Fragestellungen allerdings etwas problematisch sein kann

- Typ 1 und Typ 2 Fehler werden bei Accuracy **nicht** unterschieden
  - Typ 1 Fehler (auch: α-Fehler)
  - Typ 2 Fehler (auch: β-Fehler)

(in der nebenstehenden Darstellung symbolisieren die Regenschirme fpr die Vorhersagen, die Regenwolken dagegen für die Sachverhalte)



4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

- Metriken, welche derartige Sachverhalte berücksichtigen sollen, benötigen aber als Kennzeichnung der Klassen zwingend numerische Werte (positiv=1, negativ=0) mit Sicht auf die Voraussage (vgl. Vorbereitung am Anfang dieses Kapitels)

- die Confusion Matrix zwischen y_pred_n und y_test_n zeigt sich dann wie folgt:

col_0	0	1
row_0		
0	1465	141
1	270	1411

- zur Erinnerung: 0 steht für negativ, 1 für positiv (je nach Fragestellung)
- da in der Literatur eine solche Matrix immer zuerst das positive Ergebnis darstellt und dann das negative, müssen in der obigen Darstellung die Reihenfolgen der Zeilen und Spalten angpasst werden (das ist aber nur für die folgenden Darstellungen von Bedeutung, vgl. [ipynb])

col_0	1	0
row_0		
1	1411	270
0	141	1465

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

- eine weitere Übereinkunft (die aber nicht überall konsequent eingehalten wird):
  - prediction (Vorhersage des Modells, **Befund**) wird auf den **Zeilen** der Confusion Matrix dargestellt
  - actual (wahrer Wert, **Sachverhalt**) wird auf den **Spalten** dargestellt
- eine Übersicht der daraus resultierenden Begriffe:

		Tatsächlich (actual, reference)	
		Sachverhalt	kein Sachverhalt
Voraussage (prediction)	Befund	richtig positiv (TP, A)	falsch positiv (FP, B)
	kein Befund	falsch negativ (FN, C)	richtig negativ (TN, D)

- die folgenden Formulierungen beziehen sich auf die Abkürzungen TP, FP, FN, TN oder auch auf die Buchstaben A, B, C, D (in der Statistik Fachliteratur)
- ein konkretes Anwendungsbeispiel auf Wikipedia: HIV in der BRD

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

- für die folgenden Diskussionen werden die Einzelwerte wie folgt aus der Wahrheitstabelle extrahiert (vgl. [ipynb]):

```
from sklearn.metrics import confusion_matrix
tn, fp, fn, tp = confusion_matrix(y_test_n, y_pred_n).ravel()
print('tp =', tp)
print('fn =', fn)
print('fp =', fp)
print('tn =', tn)
```

```
tp = 1411
fn = 141
fp = 270
tn = 1465
```

- diese Werte werden für die weiteren Diskussionen verwendet

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

precision - positive predictive value - PPV

- Formulierung (gemäss obiger Nomenklatur): $PPV = \frac{TP}{TP+FP}$
- der Anteil der tatsächlich positiven Werte an der Gesamtheit der positiven Voraussagen
- ausserdem gilt folgende Beziehung: $PPV = 1 - FPR$ (siehe unten)

		act	
		y	n
pred	y	TP	FP
	n	FN	TN

```
ppv = tp / (tp + fp)
print(ppt)
from sklearn.metrics import precision_score
print(precision_score(y_test_n, y_pred_n))
```

```
0.8393813206424747
0.8393813206424747
```

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

recall, hit rate, sensitivity, true positive rate (TPR)

- Formulierung: $TPR = \frac{TP}{TP+FN}$
- der Anteil der korrekt als positiv vorausgesagten Ergebnisse an der Gesamtheit der tatsächlich positiven Sachverhalte
- ausserdem gilt folgende Beziehung: $TPR = 1 - FNR$

		act	
		y	n
pred	y	TP	FP
	n	FN	TN

```
tpr = tp / (tp + fn)
print(tpr)
from sklearn.metrics import recall_score
print(recall_score(y_test_n, y_pred_n))
```

```
0.9091494845360825
0.9091494845360825
```

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

F1

- Formulierung: $F1 = 2 \cdot \frac{precision * recall}{precision + recall} = 2 \cdot \frac{PPV * TPR}{PPV + TPR}$
- kombiniert precision und recall mittels des gewichteten harmonischen Mittels
- ein Kompromiss zwischen precision und recall

		act	
		y	n
pred	y	TP	FP
	n	FN	TN

```
f1 = 2 * (ppv * tpr) / (ppv + tpr)
print(F1)
from sklearn.metrics import f1_score
print(f1_score(y_test_n, y_pred_n))
```

```
0.8728734921125888
0.8728734921125888
```

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

classification_report

- eine Zusammenstellung der oben genannten Metriken in einer kombinierten Ausgabe (vgl. [ipynb])

```
from sklearn.metrics import classification_report
print(classification_report(y_test_n, y_pred_n))
```

```
              precision    recall  f1-score   support

           0       0.91      0.84      0.88      1735
           1       0.84      0.91      0.87      1552

    accuracy                           0.87      3287
   macro avg       0.88      0.88      0.87      3287
weighted avg       0.88      0.87      0.88      3287
```

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

classification_report

- neben precision, recall und f1-score wird auch support ausgegeben, d.h. die Mächtigkeit der jeweiligen Klasse
- wird verwendet, um gewichtete Mittelwerte der jeweiligen Scores zu berechnen
- die einzelnen Zeilen berechnen die Scores mit der Voraussetzung, dass die entsprechende Klasse (Spalte links) als positiv aufgefasst wird, die andere (oder alle anderen zusammen) als negativ
- das bedeutet auch, dass der classification_report auf Multiklass Fragestellungen angewendet werden kann
- und daher müssen die Targets hierfür nicht zwingend binär numerisch dargestellt werden, die notwendigen Transformationen werden intern durchgeführt

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

classification_report

- zur Zusammenfassung (im unteren Teil)
  - accuracy: bekannte Multiklass Metrik
    (wird etwas unschön unter der Spaltenüberschrift "f1-score" angegeben)
  - macro_avg: ungewichteter Mittelwert der Scores pro Klasse
  - weighted_avg: gewichteter Mittelwert der Scores pro Klasse
- die Funktion kann auch auf nicht numerisch codierte Targets angewendet werden (vgl. [ipynb])

- damit precision, recall und f1 auch auf Multiklass Modelle angewendet werden können, müssen die einzelnen binären Scores gemittelt werden, was über einen entsprechenden Parameter beim Aufruf erreicht werden kann (vgl. [ipynb])

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

classification_report

- Beispiel eines Classification Report einer Multiklass Fragestellung (Code: vgl. [ipynb])

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	17
versicolor	1.00	0.94	0.97	17
virginica	0.94	1.00	0.97	16
accuracy			0.98	50
macro avg	0.98	0.98	0.98	50
weighted avg	0.98	0.98	0.98	50

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

Sensitivität und Spezifität

- in Medizin (und anderswo in der deutschsprachigen Welt) werden oft auch die folgenden Begriffe verwendet
  - Sensitivität, $TPR = \frac{TP}{TP+FN}$ ist somit dasselbe wie Recall

		act	
		y	n
pred	y	TP	FP
	n	FN	TN

```
tpr = tp / (tp + fn)
print(tpr)
from sklearn.metrics import recall_score
print(recall_score(y_test_n, y_pred_n))
```

```
0.9091494845360825
0.9091494845360825
```

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

Sensitivität und Spezifität

- Spezifität, $TNR = \frac{TN}{TN+FP}$
  wird auch als correct rejection rate bezeichnet
  (steht so in sklearn.metrics als Funktion nicht zur Verfügung,
  wird aber hier festgehalten, da weiter unten wieder verwendet)

		act	
		y	n
pred	y	TP	FP
	n	FN	TN

```
tnr = tn / (tn + fp)
print(tnr)
```

```
0.8443804034582133
```

- ausserdem wird für späteren Gebrauch auch noch "False positive rate" hinterlegt, welche wie folgt definiert ist: $FPR = 1 - TNR$

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

ROC AUC - Teil 1 (binär)

- in Anlehnung an Sensitivität und Spezifität liefert ROC / AUC ein Mass, welches einen Kompromiss zwischen diesen beiden Massen liefert
- wenn eine Sensitivität von 1 (100%) gefordert ist, nimmt man eine kleine Spezifität in Kauf
- wenn umgekehrt eine Spezifität von 1 gefordert ist, wird eine kleine Sensitivität in Kauf genommen

- roc / roc_auc können sowohl auf binäre (0, 1) wie auf kontinuierliche Voraussagewerte ($0 \leq p \leq 1$) angewendet werden
  - hier wird roc_auc für binäre Voraussagen besprochen
  - roc_auc für kontinuierliche Voraussagen dann im Kap. 4.4.1.4

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

ROC AUC - Teil 1 (binär)

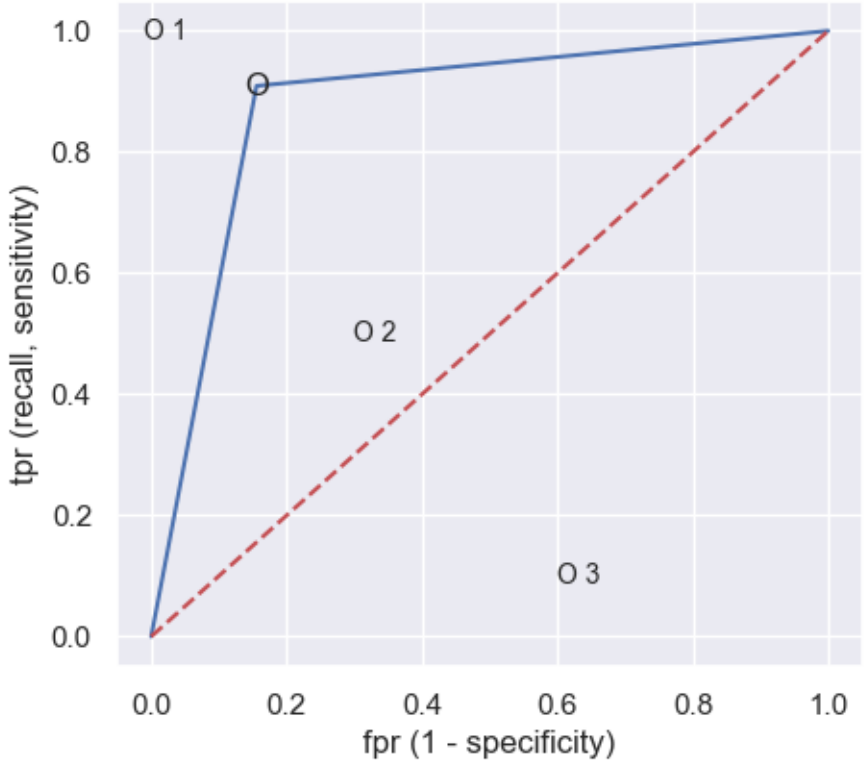
- Begriffe:
  - ROC: Receiver Operation Characteristics
  - AUC: Area under Curve
- roc_auc berechnet die Fläche unter der Kurve (resp. dem Polygon), wenn tpr (y) und fpr (x) einander gegenübergestellt werden

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.3 Metriken für binär numerische Klassen

ROC AUC - Teil 1 (binär)

- idealer Klassifikator weist folgende Werte auf: tpr = 1, fpr = 0
  - der Punkt erscheint im Diagramm in der linken oberen Ecke (1)
  - die Fläche unter der "Kurve" (eigentlich Polygon) beträgt genau 1
- ein schlechter Klassifikator (z.B. zufällig generiert)
  - der Punkt erscheint in der Nähe der Diagonalen von unten links nach oben rechts (2)
  - näher bei 0.5 als bei 1
- komplett untauglich:
  - der Punkt erscheint deutlich unterhalb der Diagonalen (3)
  - die Fläche ist nahe bei 0
  - fehlerhafte Annahmen (oder sonst ein Problem)



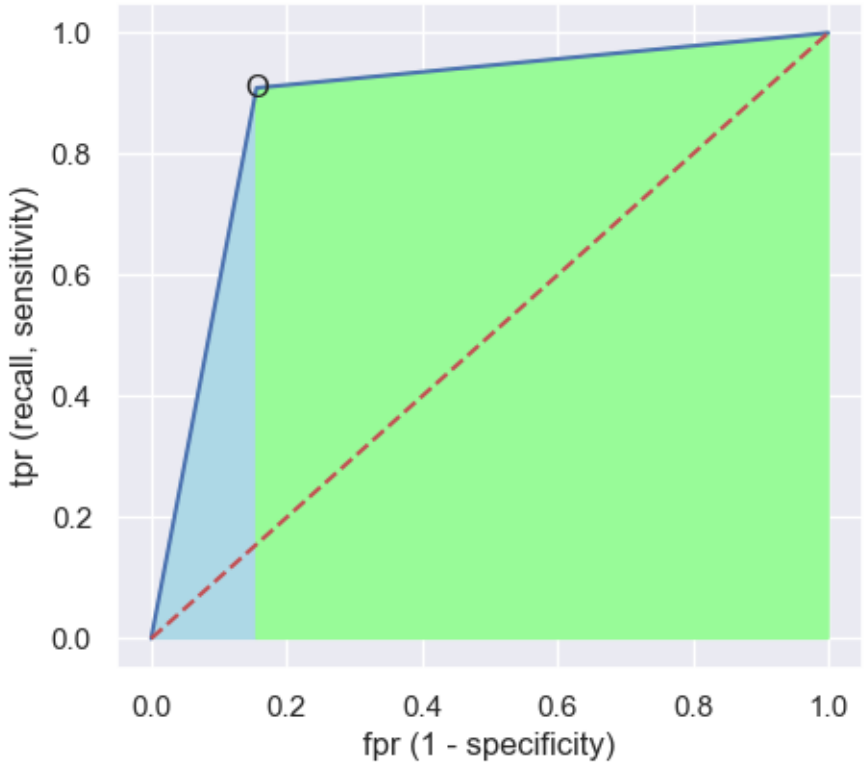
4.4.1.3 Metriken für binär numerische Klassen

ROC AUC - Teil 1 (binär)

- zur Berechnung der Fläche unter der Kurve
- die Fläche besteht aus
  - einem Dreieck (blau)
  - einem Trapez (grün)
- und kann nach geometrischen Überlegungen wie folgt berechnet werden

$$AUC = \frac{TPR \cdot FPR}{2} + \frac{(1 - FPR) \cdot (TPR + 1)}{2}$$

$$= \frac{TPR - FPR + 1}{2}$$



4.4.1.3 Metriken für binär numerische Klassen

ROC AUC - Teil 1 (binär)

- ermittelt nach der obigen Formel:

```
print((tpr - fpr + 1) / 2)
```

```
0.876764943997147 9
```

- in `sklearn.metrics` stehen die Funktionen `roc_curve()` und `auc()` zur Verfügung, welche in Kombination direkt auf binär numerisch codierte Targets angewendet werden können

```
from sklearn.metrics import roc_curve, auc
fpr_arr, tpr_arr, thresholds = roc_curve(y_test_n, y_pred_n)
print(auc(fpr_arr, tpr_arr))
```

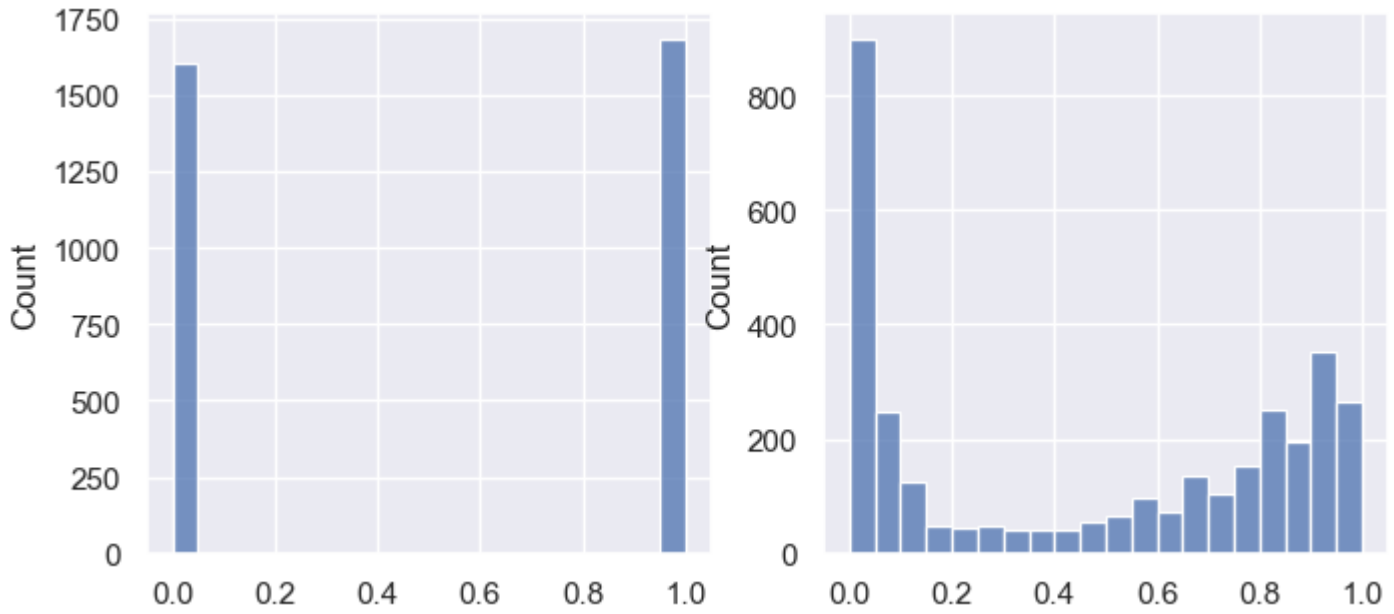
```
0.876764943997147 9
```

(Details zu deren Funktionsweise in Kap. 4.4.1.4)

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.4 Metriken für Wahrscheinlichkeitswerte von Klassen

- wie erwähnt, können neben nicht numerischen und numerischen Klassenbezeichnungen auch Wahrscheinlichkeitswerte für alle vorhandenen Klassen vorhergesagt werden, mittels `.predict_proba()` aller Klassifikatoren-Klassen
- während `.predict()` einen eindimensionalen Vektor zurückgibt, liefert `.predict_proba()` eine Matrix, nämlich je einen Vektor (Matrix-Spalte) mit den Wahrscheinlichkeitswerten für jede Klasse
- nebenstehend zum Vergleich die Häufigkeitsverteilungen von
  - y_pred_n (links)
  - y_pred_p[:,1] (rechts)



4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.4 Metriken für Wahrscheinlichkeitswerte von Klassen

ROC AUC - Teil 2 (kontinuierlich)

- tatsächlich wird bei den meisten Klassifikatoren bei Aufruf der Methode `.predict()` zuerst die Wahrscheinlichkeit für alle Klassen berechnet, und dann diejenige Klasse mit der höchsten Wahrscheinlichkeit zurückgegeben (Ausnahme z.B. SVC)
- bei Zweiklassen-Fragestellung ist der Schwellenwert (threshold) für diese Entscheidung auf 0.5 festgelegt
- wird zur Beurteilung der Performance die Wahrscheinlichkeit berücksichtigt, erlaubt dies eine feinere Differenzierung, da Werte nahe 0 und 1 stärker gewichtet werden als Werte in der Nähe von 0.5, was eher einem zufälligen Ergebnis entspricht
- die Wahrscheinlichkeitswerte erlauben es ausserdem, nachträglich möglicherweise bessere Schwellenwerte als 0.5 zu ermitteln
- zur Erinnerung: nebenstehend noch einmal eine Gegenüberstellung von binären und kontinuierlichen wahrscheinlichkeitsbasierte Voraussagen

	y_pred_n	y_pred_p_1
0	1	0.92
1	1	0.78
2	1	0.64
3	0	0.01
4	0	0.01

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.4 Metriken für Wahrscheinlichkeitswerte von Klassen

ROC AUC - Teil 2 (kontinuierlich)

- eine nützliche Funktion dazu ist `roc_curve()` aus `sklearn.metrics`, welche wie folgt eingesetzt werden kann:

```
from sklearn.metrics import roc_curve
fpr_arr, tpr_arr, thresholds = roc_curve(y_test_n, y_pred_p[:, 1])
```

- entgegengenommene Parameter
  - array der numerischen wahren Targetwerte [0,1]
  - array der Wahrscheinlichkeiten für True aus den Voraussagen
- Ergebnisse (in dieser Reihenfolge) als gleich lange Arrays
  - fpr_arr: false positive rate für jeden untersuchten Schwellenwert
  - tpr_arr: true positive rate für jeden untersuchten Schwellenwert
  - thresholds: Schwellenwerte von 1 bis 0 in Schritten von 0.01

  (um Randprobleme zu vermeiden, wird an erster Stelle auch noch ein Schwellenwert von 2 sowie fpr=0 und tpr=0 hinzugefügt)

4.4.1.4 Metriken für Wahrscheinlichkeitswerte von Klassen

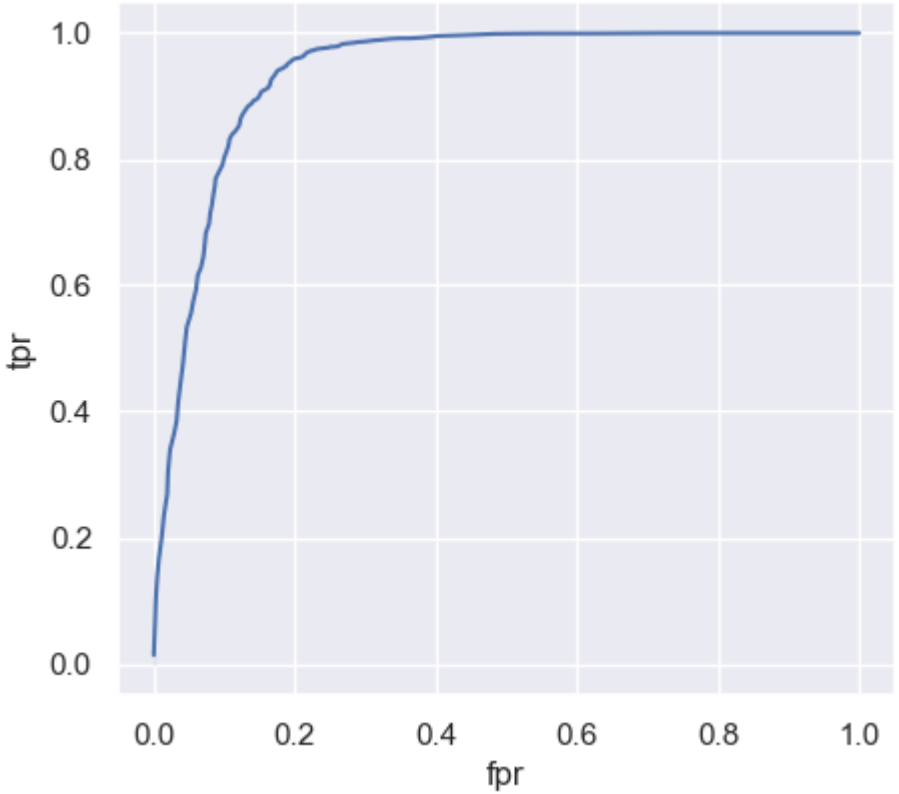
ROC AUC - Teil 2 (kontinuierlich)

- Ausschnitt der Ergebnisse:

	thresholds	tpr_arr	fpr_arr
0	2.00	0.000000	0.000000
1	1.00	0.027706	0.000000
2	0.99	0.064433	0.001729
3	0.98	0.101160	0.002882
4	0.97	0.134665	0.004611

- eine Darstellung der so ermittelten fpr und tpr Werte ist die sogenannte roc-Kurve

```
plt.figure(figsize=(5,4.5))
sns.lineplot(x=fpr_arr, y=tpr_arr)
plt.xlabel('fpr')
plt.ylabel('tpr');
```



4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.4 Metriken für Wahrscheinlichkeitswerte von Klassen

ROC AUC - Teil 2 (kontinuierlich)

- mit der Funktion auc() aus sklearn.metrics (vgl. 4.4.1.3) kann daraus auch gleich die Fläche unterhalb der Kurve berechnet werden, wenn als Argumente die beiden Arrays fpr_arr und tpr_arr übergeben werden

```
from sklearn.metrics import auc
print('auc :', auc(fpr_arr, tpr_arr))
```

```
auc : 0.9366306188537984
```

- dazu wird die sogenannte Trapezregel eingesetzt
  - für jedes Paar von benachbarten Schwellenwerten wird die Fläche des von ihnen begrenzten Trapezes berechnet und die Flächen summiert (n bezeichnet die Anzahl Schwellenwerte)

$$AUC = \sum_{i=1}^{n-1} \frac{TPR_i + TPR_{i+1}}{2(FPR_{i+1} - FPR_i)}$$

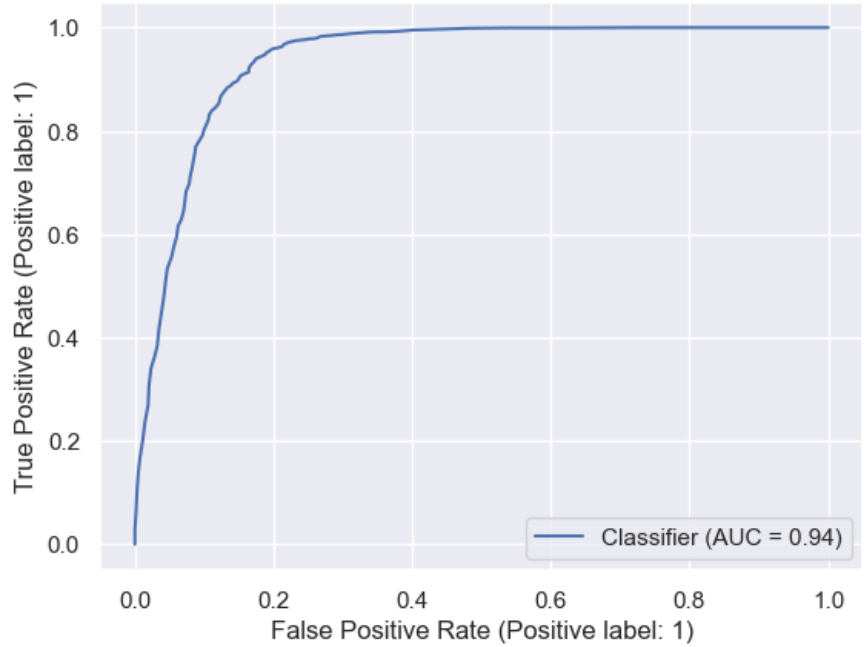
  - rechnerische Kontrolle: vgl. [ipynb]

4.4.1.4 Metriken für Wahrscheinlichkeitswerte von Klassen

ROC AUC - Teil 2 (kontinuierlich)

- eine Funktion von `sklearn.metrics`, welche das Ganze mit einem einzigen Aufruf erledigt: `RocCurveDisplay()`

```
from sklearn.metrics import RocCurveDisplay
RocCurveDisplay.from_predictions(
    y_test_n, y_pred_p[:,1]);
```



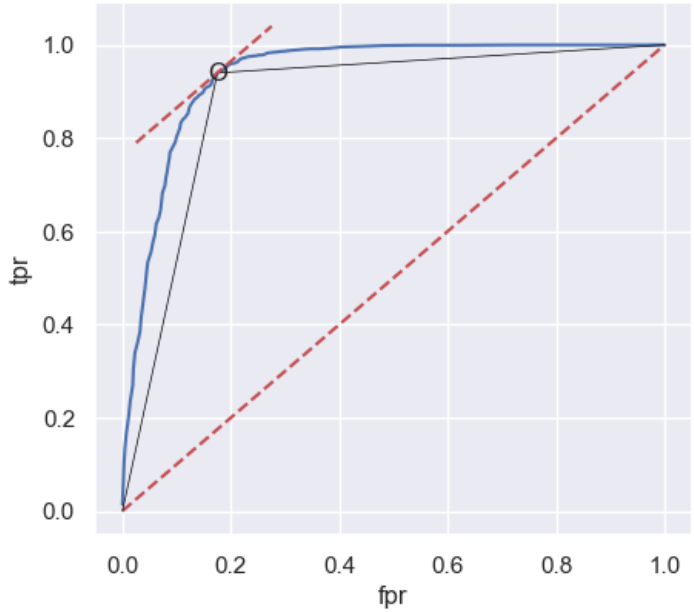
4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.4 Metriken für Wahrscheinlichkeitswerte von Klassen

ROC AUC - Teil 2 (kontinuierlich)

- zum ermitteln des Schwellenwertes für den besten AUC
  - dazu werden wiederum die AUC Werte aus Kap. 4.4.1.3 betrachtet
  - die grösste Polygonfläche liefert der Punkt auf der ROC Kurve, welcher am weitesten von der Diagonalen entfernt liegt: er führt zum Dreieck mit der grössten Fläche (oberhalb der Diagonalen)

- aus geometrischen Überlegungen befindet sich der Punkt [fpr,tpr] mit dem grössten AUC Wert an der Stelle der Kurve, an welcher diese von einer Tangenten, parallel zur Diagonalen, berührt wird



4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.4 Metriken für Wahrscheinlichkeitswerte von Klassen

ROC AUC - Teil 2 (kontinuierlich)

- da AUC direkt aus FPR und TPR errechnet werden kann, wird ein entsprechender Array aus den Ausgangsarrays erstellt

```
auc_arr = (tpr_arr - fpr_arr + 1) / 2
```

- und darauf gefiltert

```
print('best auc :', max(auc_arr))
print('best threshold :', thresholds[np.where(auc_arr == max(auc_arr))][0])
```

```
best auc : 0.8824305906295493
best threshold : 0.44
```

- 0.44 ist offenbar etwas besser als der Default 0.5

4.4.1 Validierung und mehr - Performance Metriken - Klassifikation

4.4.1.4 Metriken für Wahrscheinlichkeitswerte von Klassen

Log Loss

- diese Metrik arbeitet ebenfalls mit Wahrscheinlichkeitswerten für die vorhergesagten Klassen
- Formulierung für Binäre Fragestellungen

$$LogLoss = -\big(y \cdot log(p) + (1-y) \cdot log(1-p)\big)$$

- Wertebereich
  - bester Wert: 0
  - schlechtester Wert: >> 1
- Log Loss ist auch auf Multiklass Fragestellungen anwendbar (vgl. [ipynb])
- ist beliebt bei kaggle competitions
- Aufruf:

```
from sklearn.metrics import log_loss
print(log_loss(y_test_n, y_pred_p))
```

```
0.304115865220962
```

4 Validierung und mehr

Workshop 15

Gruppen zu 2 bis 4, Zeit: 30'

- `.predict_proba()` gibt bei allen Klassifikatoren die Wahrscheinlichkeit für die Zugehörigkeit zu den einzelnen Klassen zurück, `predict()` dagegen die wahrscheinlichste Klasse selber
- für Zwei-Klassen Fragestellungen bedeutet dies, dass bei einer Wahrscheinlichkeit (proba) `> 0.5` für die erste Klasse diese zurückgegeben wird, andernfalls die zweite Klasse, 0.5 ist somit ein scheinbar willkürlicher Schwellenwert
- untersuchen Sie die Auswirkung anderer Schwellenwerte auf die Accuracy mit RandomForestClassifier auf den aufbereiteten Bankkunden-Daten

4 Validierung und mehr

Workshop 15

- vorgeschlagenes Vorgehen
  - trainieren eines RandomForestClassifier mit den vorbereiteten Bankkundendaten (Trainingsdaten)
  - bestimmen der Wahrscheinlichkeit für jede Beobachtung der entsprechenden Testdaten zur Klasse 'no'
  - erstellen Sie einens Range der zu untersuchenden Schwellenwerte, z.B. mit np.arange()
  - in einem Loop über alle Werte dieses Ranges
    - y_pred für den jeweiligen Schwellenwert berechnen (wiederum als ['no', 'yes'])
    - accuracy_score() der jeweiligen Prediction (und sammeln in einer Liste)
  - ausgeben des besten Score-Wertes und des zugehörigen Schwellenwertes in der Konsole
  - visualisieren der Ergebnisse auch als Lineplot

4.4.2 Validierung und mehr - Performance Metriken - Regression

4.4.2.1 Vorbereitung

- im Gegensatz zur Klassifikation sind die Aussagen von Performance Metriken bei Regressionsmodellen viel eingeschränkter
- eine davon wurden im Rahmen von Regression (Kap. 3) bereits vorgestellt
  - r2 (r2_score), der Standard Scorer der trainierten Modelle
- die Performance Metriken für Regression können nicht absolut interpretiert werden, wie z.B."accuracy" bei der Klassifikation

- es handelt sich dabei viel eher um Indikatoren, um die Qualität unterschiedlicher Learner und allenfalls Tuning Parameterwerte untereinander vergleichen zu können,
  aber nicht für genau interpretierbare Aussagen wie bei Klassifikation

4.4.2 Validierung und mehr - Performance Metriken - Regression

4.4.2.1 Vorbereitung

- für die weitere Diskussion in diesem Kapitel werden folgende Vorbereitungen getroffen (vgl. [ipynb])
  - laden von "melb_data_prep.csv"
  - Features - Target - Split
  - Train - Test - Split
  - Trainieren eines Modells exemplarisch mit DecisionTreeRegressor auf den Trainingsdaten
  - erstellen einer Prediction auf den Testdaten

- ein Ausschnitt von y_test und y_pred nach diesen Vorbereitungen:

	y_test	y_pred
18178	3400000.0	901000.0
16248	3625000.0	2100000.0
7472	565000.0	580000.0
11719	2525000.0	1680000.0
5228	1350000.0	1200000.0
:		

4.4.2 Validierung und mehr - Performance Metriken - Regression

4.4.2.2 Einige Scorer im Überblick

mean_squared_error (mse)

- Mittelwert der quadrierten Fehler

$$mse = \frac{1}{n}\sum(y_i - \hat{y}_i)^2$$

  - wobei (hier und im Folgenden)
    - $\hat{y}_i$: Voraussagewert für Instanz i
    - y_i: wahrer Wert (Gold Standard) des Targets für Instanz i
    - n: Anzahl Instanzen

  (vgl. auch RSS aus Kap. 3.2.1)

- Anwendung:

```
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))
```

```
166638679688.0558
```

- bester Wert: 0, schlechtester Wert: $\rightarrow \infty$

4.4.2 Validierung und mehr - Performance Metriken - Regression

4.4.2.2 Einige Scorer im Überblick

mean_squared_error (mse)

- Eigenschaften:
  - wegen der Quadratfunktion in der Formel werden grosse Fehler bestraft
  - schlecht interpretierbar im Massstab des Targets

rmse (Rooth Mean Squared Error)

- Quadratwurzel von mse
- ist als solches nicht implementiert in `sklearn.metrics`, kann aber ebenfalls mit `mean_squared_error` ermittelt werden, indem der Parameter `squared=False` gesetzt wird
- ist vergleichbar mit der Standardabweichung
- dadurch können z.B. Vertrauensintervalle abgeschätzt werden,
  z.B. 95% conf int = +/- 1.96 * rmse

4.4.2 Validierung und mehr - Performance Metriken - Regression

4.4.2.2 Einige Scorer im Überblick

mean_absolute_error (mae)

- mittlerer absoluter Fehler (Betrag)

$$mae = \frac{1}{n}\sum |y_i - \hat{y}_i|$$

- (Nomenklatur und Parametrisierung wie bei mse)
- Anwendung:

```
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_test, y_pred))
```

```
242732.12227477843
```

- bester Wert: 0, schlechtester Wert: $\rightarrow \infty$
- bestraft grosse Fehler nicht
- erzeugt einen Fehler in der Grössenordnung des Targets, was nützlich für Interpretation sein kann

4.4.2 Validierung und mehr - Performance Metriken - Regression

4.4.2.2 Einige Scorer im Überblick

mean_absolute_percentage_error (mape)

- mittlerer relativer absoluter Fehler (Betrag)

$$mape = \frac{1}{n}\sum\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

- (Nomenklatur und Parametrisierung wie bei mse)
- Anwendung:

```
from sklearn.metrics import mean_absolute_percentage_error
print(mean_absolute_percentage_error(y_test, y_pred))
```

```
0.22264410757249675
```

- bester Wert: 0, schlechtester Wert: $\rightarrow \infty$
- wegen der Division durch die wahren Werte kann es für Werte von oder nahe 0 problematisch sein, da dies zu Fehler oder extrem grossen Werten führen kann
- ob dies gut genug ist, muss mit dem Fach abgesprochen werden

4.4.2 Validierung und mehr - Performance Metriken - Regression

4.4.2.2 Einige Scorer im Überblick

mean_squared_log_error (msle)

- mittlerer quadrierter logarithmierter Fehler

$$msle = \frac{1}{n}\sum(\log(1 + y_i) - \log(1 + \hat{y}_i))^2$$

- die Addition von 1 vor dem jeweiligen Logarithmieren ist notwendig, um zu verhindern, dass bei Einzeldifferenzen < 1 negative Werte resultieren (vgl. Feature Engineering, Kap. 1.3.2.2)
- Anwendung

```
from sklearn.metrics import mean_squared_log_error
print(mean_squared_log_error(y_test, y_pred))
```

```
0.0842922380924371
```

- bester Wert: 0, schlechtester Wert: $\rightarrow \infty$

4.4.2 Validierung und mehr - Performance Metriken - Regression

4.4.2.2 Einige Scorer im Überblick

median_absolute_error (medae)

- Median der absoluten Fehler

$$medae = (|y_1 - \hat{y}_1|, \ldots, |y_n - \hat{y}_n|)$$

- Anwendung

```
from sklearn.metrics import median_absolute_error
print(median_absolute_error(y_test, y_pred))
```

```
139500.0
```

- bester Wert: 0, schlechtester Wert: >> 1

4.4.2 Validierung und mehr - Performance Metriken - Regression

4.4.2.2 Einige Scorer im Überblick

r2_score (r2)

- quadrierter Korrelationskoeffizient
- entspricht dem **multiplen** quadrierten Korrelationskoeffizienten (Bestimmtheitsmass), wie von Statistikprogrammen bei Regressionen angezeigt wird (vgl.Output in Kap. 3.2.1.4 oder lm() bei R)
- Details zur Berechnung finden sich z.B hier

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

- Anwendung:

```
from sklearn.metrics import r2_score
print(r2_score(y_test, y_pred))
```

```
0.568854390696127
```

- bester Wert: 1, kann aber auch negativ sein bei unbrauchbaren Modellen
- (wenn nur ein Feature in die Regression einfliesst, entspricht dieser Wert genau dem quadrierten Korrelationskoeffizienten zwischen y und X)

4.4.2 Validierung und mehr - Performance Metriken - Regression

Schlussbemerkung zu Performance Metriken

- diverse ML Projekte können im Verlauf von Regression auf eine Klassifikations-Fragestellung modifiziert werden
- Grund: Schwäche der Performance Metriken der Regression in Bezug auf deren absolute Interpretierbarkeit
- Beispiel: Voraussage der Präzision von Messgeräten aufgrund von Parametern des Herstellungsprozesses
- wenn anstelle deren absolute Abweichung von einem Zielwert das Überschreiten einer zum Voraus definierten Schwelle als Target dient, sind die Ergebnisse für die Auftraggeberin einfacher und besser nachvollziehbar zu verstehen
- derartige Überlegungen sollten bereits in der Phase Project Understanding berücksichtigt werden

- dann noch dies: eine spannende Library für allerhand Visualisierungen von Klassifikationen, Regressionen, Clustering und vielem mehr: Yellowbrick, probieren Sie sie aus!