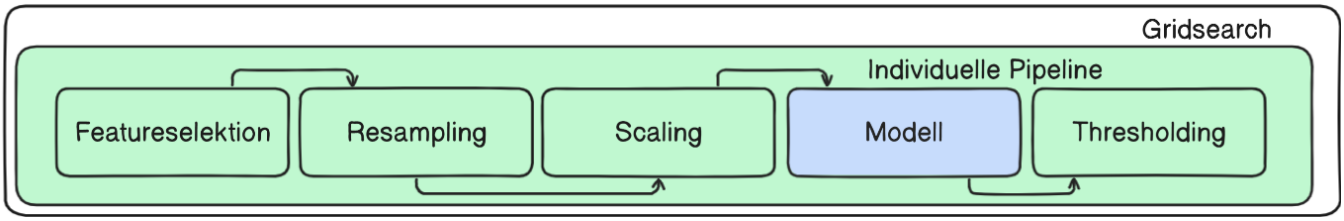
Inhalt

1. Einleitung

1.0 Gesamtvorgehen

Im Anschluss an die Datenanalyse haben wir bereits Methoden zur **Verbesserung der Daten** (z.B. Zusatzfeatures und Resampling) intensiv diskutiert. Bei der Einzelanwendung eines Machine Learning Algorithmus verfolgen wir außerdem das Ziel, durch **Modelloptimierung des Algorithmus** diesen zu verbessern. Im dritten Schritt können wir die Ergebnisse des Algorithmus weiter verbessern, indem wir über **Featureselektion und Resampling** Optimierungsmöglichkeiten einbauen, und die erzielten **Vorhersagen nachgelagert manipulieren**.

Ultimatives Ziel wäre es also, für einen Einzelalgorithmus eine Pipeline aufzubauen, die all diese Schritte vornimmt, und diese in einem GridSearch über alle Parameter zu optimieren:



Dabei muss jeder Teilschritt parameterabhängig sein (Featureselektion: Mitgabe möglicher Spaltenkombinationen; Resampling: Mitgabe möglicher Resamplingmethoden; Scaling: Mitgabe möglicher Scaler; Thresholding: Mitgabe möglicher Wahrscheinlichkeitsgrenzen für y=1), wir brauchen ein ausführliches Parametergrid, und eine vermutlich rechenintensive GridSearch.

Das gefundene Zielmodell kann dann mit der optimalen Parameterkombination als Pipelineobjekt an Folgevorhaben (z.B. Voting / diverse Ensemble-Methoden) übergeben werden.

Realisiertes Ziel: Im Rahmen dieses Projekts werden wir zunächst gute Startparameter für eine initiale Gridsearch suchen und diese durchführen. Dabei werden wir auch die Auswirkung von Featureselektion, Resampling und Thresholding punktuell testen. Schließlich werden wir die oben dargestellten Teilschritte weitestgehend in eine Gesamtpipeline einbauen (konkret: Featureselektion, Scaling, Modell und Thresholding). Auf das Resampling müssen wir dabei aufgrund von Aufwand und Kompatibilitätsgründen zwischen sklearn und imblearn verzichten.

1.1 Wahl des Kernels und relevanter Parameter

In unserem Fall verwenden wir offensichtlicherweise den Support Vector Classifier mit RBF Kernel:

- Dieser erlaubt uns das Aufbauen komplexerer Bereiche zur Klassentrennung.
- Gamma und C sind maßgeblich.

1.2 Voreinschätzung Eignung und Performance des Algorithmus

Wir halten den Algorithmus **grundsätzlich für gut geeignet**, um (zumindest teilweise) auf den vorliegenden Daten einigermaßen gute Ergebnisse zu zeigen, da er „zerstreute Daten“ und Verinselungen vergleichsweise gut abgrenzen kann.

Allerdings haben wir in der Datenanalyse eine sehr hohe Durchmischung der Klassen visualisiert. Bei der Anwendung des SVC wird es also schwierig werden, die richtigen Kombinationan aus gamma und C zu finden, um ausreichend große, aber nicht zu sehr durchmischte „Inseln“ zu bilden, und dabei nicht zu overfitten.

Wir rechnen außerdem damit, dass Verbesserungen des Recall mit einem starken Trade-Off bezüglich Precision einhergehen werden:

- Es wird kein Problem sein, auf der Trainingsmenge gut zu fitten (hohes Gamma und C). Dabei werden wir aber sehr wahrscheinlich zuerst overfitten.
- Wir müssen also vermutlich die Einschränkungen für die Kernelfunktion lockern (größere Inseln zulassen, verschwommene Inselgrenzen zulassen, Falschklassifikation auf Inseln zulassen).
- Dabei rechnen wir bei der vorliegenden Klassendurchmischung auf den Daten aber auch damit, dass auf gefitteten stroke-Bereichen häufig auch nicht-strokes liegen (und damit false negatives entstehen).

2. Vorgehensweise für erste Modellbewertung

Zunächst schildern wir die Vorgehensweise, mit der wir das initiale Modell aufsetzen und die ersten Parameter wählen. Dafür verwenden wir das Analyseprogramm, das in der Folge beschrieben wird.

2.1 Analyseprogramm

Das **Analyseprogramm in der python-Datei „Analyseprogramm_SVC.py“** hat folgende Struktur (schematisch erklärt / nur relevante Codeausschnitte):

1. Importe und Daten aus pickle laden
2. Block für Steuerung verschiedener Parameter

Hier stehen zentral alle Programmparameter.

```
' ###################### STEUERUNGSBLOCK ################################'
resampling = "none"           # none, smote, oder partial_smote
classweight_y = 4           # Gewichtung für strokes
gamma = 0.5                 # gamma für SVC
C = 0.005                     # C für SVC
scaling_method = "Standard"   # "Standard", "MinMax", oder leer (dann unskaliert)
show_rbf_plot = 0              # opb plot für brbf-bereiche gezeigt werden soll
show_threshold_curve = 0     # 1: thresholdingkurve anzeigen, 0 sonst
```

- C und gamma sind klar. Der parameter resampling kann für den Test von smote und undersampling verwendet werden (funktioniert auch und kann für Analysen verwendet werden, die wir hier aber nicht weiter vorgenommen haben) .
- Show_threshold_curve: Aktiviert, ob am Ende zusätzlich Threshold-Kurve berechnet wird (Erklärung in Kapitel gegen Ende).
- Direkt unter diesem Codeabschnitt folgt ein Block zur Steuerung der verwendeten Features. Dieser dient dazu, Originalspalten wegzulassen, und/oder in der Datenanalyse erzeugte Zusatzfeatures zu den Daten hinzuzunehmen. Wir haben aus Zeitgründen aber stets nur die Originalspalten verwendet, und keine Attributskombinationengetestet. Die Strukturen dafür wären aber da.

3. Block für Modellinstanz und Fitting

Hier wird das Modell gebildet. Es wird stets auf der Trainingsmenge gefittet, die Trainings- und Testdaten werden dann anhand dieses fits transformiert. Das machen wir nur für die initiale Modellbildung so, der spätere GridSearch wird dann ausschließlich auf den Trainingsdaten vorgenommen.

4. Block mit Reportingfunktion

In einem Guss gibt das Programm hier über eine Funktion einen umfangreichen Report zu den Ergebnissen auf Trainings- und Testdaten aus.

5. Block Plot

Hier wird ein zweidimensionaler Plot der auf der Trainingsmenge entstandenen rbf-Bereiche erstellt. Der Code ist inspiriert von sklearn (https://scikit-learn.org/1.5/auto_examples/svm/plot_rbf_parameters.html#sphx-glr-auto-examples-svm-plot-rbf-parameters-py) und auf unsere Bedürfnisse zugeschnitten.

Der Plot zeigt die Trainingsmenge (strokes = rote Punkte, nicht-strokes = blaue Punkte), eingeschränkt auf einen zweidimensionalen Teilbereich. Die erzeugten rbf-Bereiche sind dargestellt, sowie die strokes (ausschließlich diese!) aus den Testdaten (gelbe Punkte), um die Eignung des Fittings zu visualisieren und die Parameter gezielt ausprobieren zu können.

Hinweis: Der Plot skaliert die Daten nicht (im Gegensatz zum Report). Das ist für unsere Zwecke in Ordnung, da wir den Plot nur zur Sondierung der Parameter ergänzend zum Report dient.

6. Block Thresholding

Hier werden ganz am Ende die erzeugten Wahrscheinlichkeiten des Ergebnismodells untersucht, wie sich Änderungen der Wahrscheinlichkeitsgrenze auf den Recall auswirken würden. Mehr dazu später.

2.2 Zweck und Verwendung Analyseprogramm

Das Analyseprogramm wird verwendet, um „von Hand“ ein Gespür für den Algorithmus zu bekommen, und dann erste Parameterkombinationen zu finden, in deren Umgebung wir einen GridSearch starten können.

Außerdem ist sind der umfangreiche Report und der Plot äußerst hilfreich, um Veränderungen des Recalls / der Precision genauer zu verfolgen, wenn man die Parameter ändert. Erster Test mit Analyseprogramm

3.0 Skalierung

Wir verwenden für den Rest unserer Auswertungen den Standard-Scaler von sklearn zur Skalierung.

Wir wenden der Einfachheit halber den Scaler auf alle Spalten an (kontinuierliche, aber auch binäre und kategorische Spalten), auch wenn er eigentlich für binäre/kategoriale Spalten nicht grundsätzlich nötig ist. Bei letzteren sollte er keine störenden Auswirkungen haben.

Diese Entscheidung haben wir getroffen, um Übergewichtungen von Alter, BMI und Blutzucker bei der Distanzberechnung im Vergleich zu den anderen Spalten zu vermeiden.

3.1 Vorüberlegungen zur Wahl der Parameter Gamma und C

Da in unserem Fall die strokes auf dem gesamten Datenraum (Trainings- und Testdaten) sehr dünn verteilt sind und meistens von nicht-strokes umgeben, sollten wir Varianten testen.

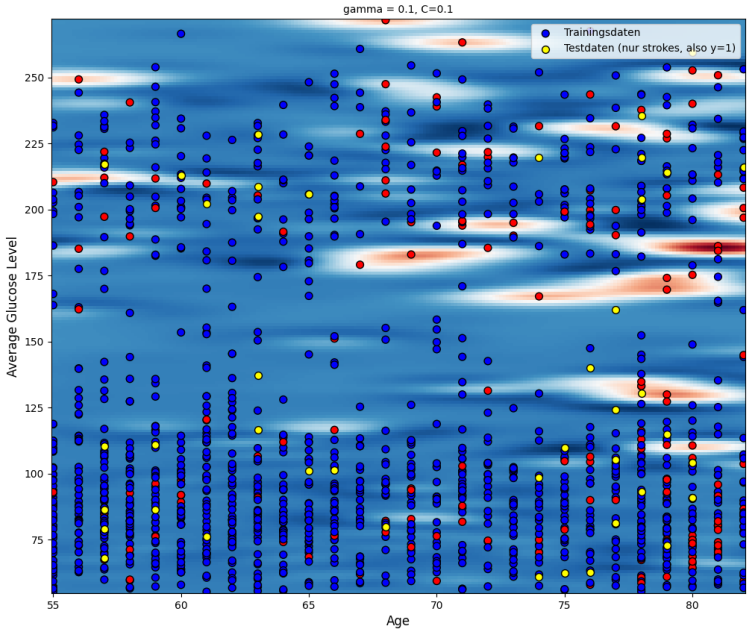
- Für **hohe gamma- und C-Werte** werden wir die **Trainingsdaten** sehr gut abbilden können:
    - Kleine, konkrete Inseln entstehen um die (meist einzelnen) Trainings-strokes herum.
    - **ABER**: Da wir aber auch bei den Testdaten davon ausgehen, dass dort die strokes oft einzeln im ganzen Datenraum verteilt sind, werden Test-strokes oft „ins Wasser fallen“ (also NICHT auf einer Insel der Trainings-strokes landen), was einen **schlechten Recall für Testdaten** ergibt (und starkes overfitting).
- Für **niedrige gamma- und C-Werte** entstehen beim rbf-kernel eher größere, flächige Bereiche, mit verschwommenen Grenzen. Hier haben wir **eher die Chance auf einen guten Recall auf der Testmenge,** also dass Test-strokes auf Inseln mit Trainings-strokes landen. Allerdings wird die Zuordnung hier **tendenziell fehleranfälliger für false positives** (nicht-strokes werden als strokes prognostiziert – sinkende Precision).

3.2 Parameteranalyse Gamma und C auf Trainings- und Testdaten

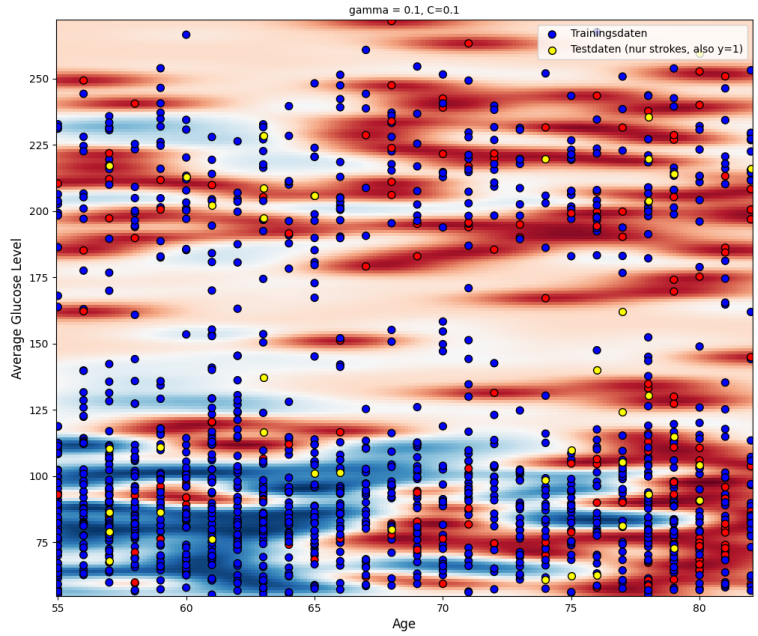
Wir testen jeweils Parameterkombinationen, um uns an ein gutes initiales Modell heranzutasten. Dabei hilft der Report, sowie die Visualisierung, um ein gutes Gespür und Verständnis für die Parameter zu bekommen.

3.2.1 Verschiedene Parameterkombinationen

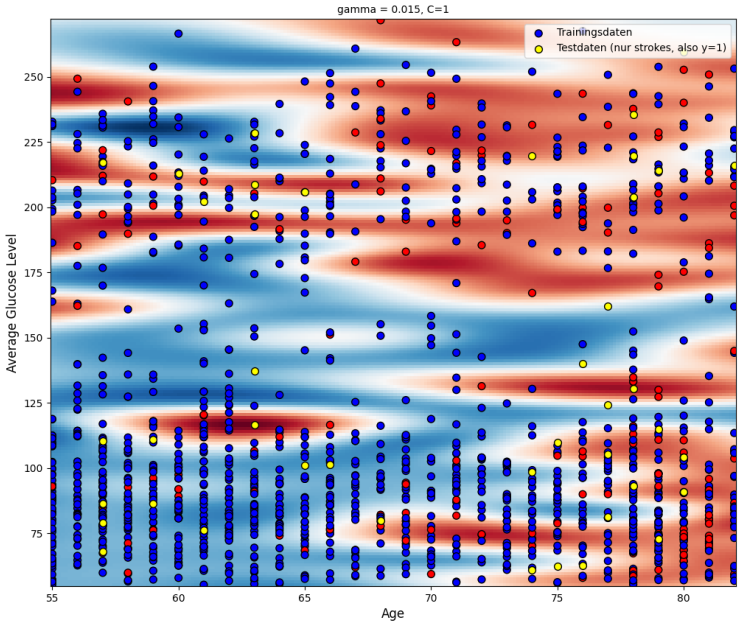
Wir zeigen hier kommentarlos verschiedene Parameterkombinationen (weitere Erläuterungen folgen später). Sie spiegeln das Herantasten an eine gute Startkombination und die Eingrenzung des Wertebereichs für die GridSearch wieder. Der Lerneffekt zur Auswirkung der Parameter war dabei enorm!



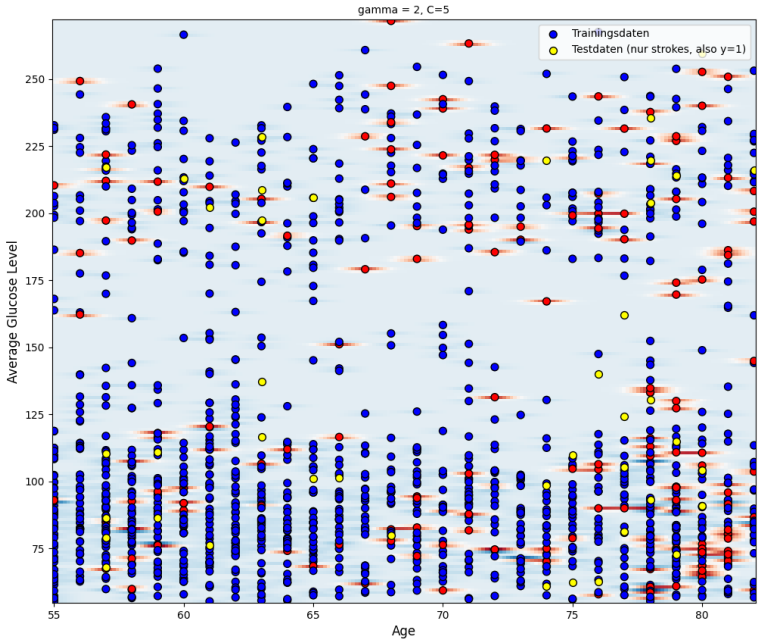

```
C:\Users\pprie\AppData\Local\Programs\Python\Python312\pyth
-------------------- Sampling --------------------------
resampling:  none
---------------------- Modellparameter -----------------
classweight y=1:     1
threshold probabilities:  0.3
gamma:  0.1
C:  0.1
--------------- Skalierung ----------
Standard
--------- Confusion Matrix und Scoring Report train -------
Confusion Matrix auf train:
 [[1253    0]
 [ 170    0]]
              precision    recall  f1-score   support

  stroke = 0       0.88      1.00      0.94      1253
  stroke = 1       0.00      0.00      0.00       170

    accuracy                           0.88      1423
   macro avg       0.44      0.50      0.47      1423
weighted avg       0.78      0.88      0.82      1423

--------- Confusion Matrix und Scoring Report test --------
Confusion Matrix auf test:
 [[314   0]
 [ 42   0]]
              precision    recall  f1-score   support

  stroke = 0       0.88      1.00      0.94       314
  stroke = 1       0.00      0.00      0.00        42

    accuracy                           0.88       356
   macro avg       0.44      0.50      0.47       356
weighted avg       0.78      0.88      0.83       356

<class 'pandas.core.frame.DataFrame'>
```




```
C:\Users\pprie\AppData\Local\Programs\Python\Python312\pyt
-------------------- Sampling --------------------------
resampling:  none
---------------------- Modellparameter -----------------
classweight y=1:     10
threshold probabilities:  0.3
gamma:  0.1
C:  0.1
--------------- Skalierung ----------
Standard
--------- Confusion Matrix und Scoring Report train -------
Confusion Matrix auf train:
 [[619 634]
 [ 25 145]]
              precision    recall  f1-score   support

  stroke = 0       0.96      0.49      0.65      1253
  stroke = 1       0.19      0.85      0.31       170

    accuracy                           0.54      1423
   macro avg       0.57      0.67      0.48      1423
weighted avg       0.87      0.54      0.61      1423

--------- Confusion Matrix und Scoring Report test --------
Confusion Matrix auf test:
 [[136 178]
 [ 13  29]]
              precision    recall  f1-score   support

  stroke = 0       0.91      0.43      0.59       314
  stroke = 1       0.14      0.69      0.23        42

    accuracy                           0.46       356
   macro avg       0.53      0.56      0.41       356
weighted avg       0.82      0.46      0.55       356

<class 'pandas.core.frame.DataFrame'>
```



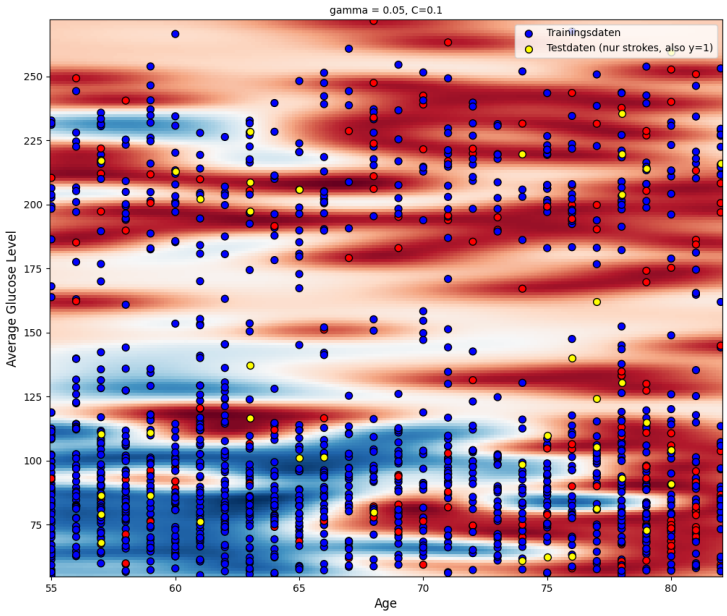

```
C:\Users\pprie\AppData\Local\Programs\Python\Python312\python.
-------------------- Sampling ----------------------------
resampling:  none
----------------------- Modellparameter ---------------------
classweight y=1:     9
threshold probabilities:  0.3
gamma:  0.015
C:  1
--------------- Skalierung ----------
Standard
--------- Confusion Matrix und Scoring Report train ----------
Confusion Matrix auf train:
 [[661 592]
 [ 30 140]]
              precision    recall  f1-score   support

  stroke = 0       0.96      0.53      0.68      1253
  stroke = 1       0.19      0.82      0.31       170

    accuracy                           0.56      1423
   macro avg       0.57      0.68      0.50      1423
weighted avg       0.87      0.56      0.64      1423

--------- Confusion Matrix und Scoring Report test ----------
Confusion Matrix auf test:
 [[148 166]
 [ 16  26]]
              precision    recall  f1-score   support

  stroke = 0       0.90      0.47      0.62       314
  stroke = 1       0.14      0.62      0.22        42

    accuracy                           0.49       356
   macro avg       0.52      0.55      0.42       356
weighted avg       0.81      0.49      0.57       356

<class 'pandas.core.frame.DataFrame'>
```




```
C:\Users\pprie\AppData\Local\Programs\Python\Python312\py
-------------------- Sampling ----------------------------
resampling:  none
----------------------- Modellparameter ---------------------
classweight y=1:     10
threshold probabilities:  0.3
gamma:  2
C:  5
--------------- Skalierung ----------
Standard
--------- Confusion Matrix und Scoring Report train -----
Confusion Matrix auf train:
 [[1205   48]
 [   0  170]]
              precision    recall  f1-score   support

  stroke = 0       1.00      0.96      0.98      1253
  stroke = 1       0.78      1.00      0.88       170

    accuracy                           0.97      1423
   macro avg       0.89      0.98      0.93      1423
weighted avg       0.97      0.97      0.97      1423

--------- Confusion Matrix und Scoring Report test ------
Confusion Matrix auf test:
 [[272  42]
 [ 34   8]]
              precision    recall  f1-score   support

  stroke = 0       0.89      0.87      0.88       314
  stroke = 1       0.16      0.19      0.17        42

    accuracy                           0.79       356
   macro avg       0.52      0.53      0.53       356
weighted avg       0.80      0.79      0.79       356

<class 'pandas.core.frame.DataFrame'>
```

3.2.4 Beste gefundene Parameterwahl

Gamma: winzig, C: niedrig, Gewichtung strokes: 10




```
C:\Users\pprie\AppData\Local\Programs\Python\Python312\python.exe "
---------- Sampling ------------------------------
resampling:  none
---------- Modellparameter ----------------------
classweight y=1:     10
threshold probabilities:  0.3
gamma:  0.05
C:  0.1
---------- Skalierung ---------
Standard
--------- Confusion Matrix und Scoring Report train ---------
Confusion Matrix auf train:
 [[598 655]
 [ 25 145]]
              precision    recall  f1-score   support

  stroke = 0       0.96      0.48      0.64      1253
  stroke = 1       0.18      0.85      0.30       170

    accuracy                           0.52      1423
   macro avg       0.57      0.67      0.47      1423
weighted avg       0.87      0.52      0.60      1423

--------- Confusion Matrix und Scoring Report test ---------
Confusion Matrix auf test:
 [[135 179]
 [ 13  29]]
              precision    recall  f1-score   support

  stroke = 0       0.91      0.43      0.58       314
  stroke = 1       0.14      0.69      0.23        42

    accuracy                           0.46       356
   macro avg       0.53      0.56      0.41       356
weighted avg       0.82      0.46      0.54       356

<class 'pandas.core.frame.DataFrame'>
```

Wir erreichen hier bereits einen **Recall von 0.69 auf den Testdaten**, bei **vergleichsweise geringem Overfitting** (Recall Training 0.85). **Allerdings ist die Precision bei 0.14**.

3.3 Zwischenfazit

Mit viel Ausprobieren und unseren Vorannahmen zu den Parametern haben wir bereits ohne GridSearch eine ganz gute Startkombination der Parameter finden können. Größere Inseln eignen sich in der Tat besser. Allerdings stellen wir auch die erwarteten Einbußen bei der Precision fest.

Für die Gridsearch im Folgenden haben wir herausgefunden:

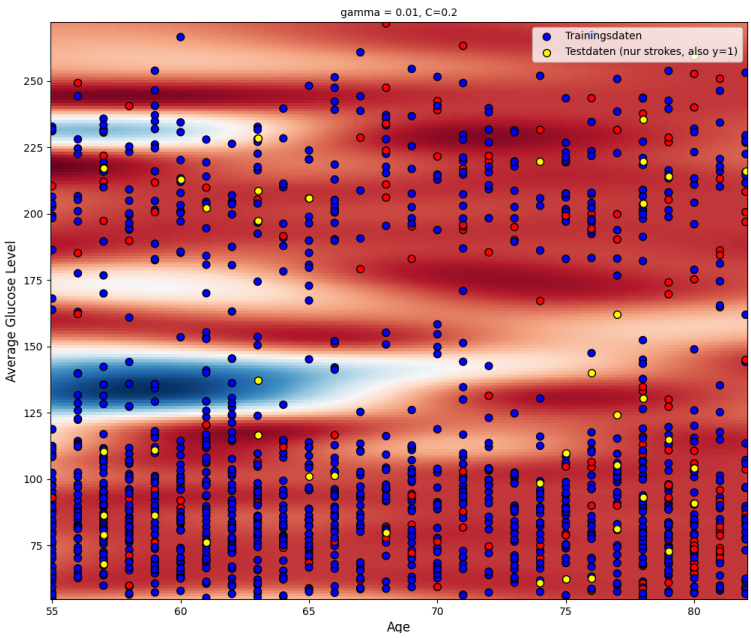
> **Wir spannen also für die Gridsearch ein Grid auf mit stroke-Gewichten im 10 herum, C zwischen 0 (bzw. einem sehr kleinem Wert) und 1, und gamma im kleinen bis sehr kleinen Bereich!**

3. Wichtige Ausführungen zum Recall-Tradeoff

Bevor wir uns die Ergebnisse der GridSearch anschauen, wollen wir nochmals genauer auf die Bedeutung von Recall und Precision jeweils in beiden Klassen y=0 (nicht-strokes) und y=1 (strokes) eingehen.

Da wir im Projekt ein Modell suchen, das strokes gut vorhersagt, suchen wir grundsätzlich ein Modell mit einem guten Recall für die Klasse y=1 (strokes). Hier gilt es jedoch genau hinzusehen:

- Betrachtet man **nicht zusätzlich auch die precision** der Klasse y=1 (und damit indirekt auch den Recall der Klasse y=0, so wählt man möglicherweise, ein Modell, das nur noch strokes vorhersagt, also auch Patienten als Risikopatienten einstuft, die kerngesund sind.
- Ein Extrembeispiel mit den entsprechenden Parametern ist unten abgebildet (Recall der Stroke-Klasse 1, Precision aber nur 12%, und alle nicht-strokes falsch als strokes vorhergesagt!)




```
C:\Users\pprie\AppData\Local\Programs\Python\Python312\p
-------------------- Sampling -------------------------
resampling:  none
---------------------- Modellparameter ----------------
classweight y=1:     100
threshold probabilities:  0.3
gamma:  0.01
C:  0.2
--------------- Skalierung ----------
Standard
--------- Confusion Matrix und Scoring Report train ----
Confusion Matrix auf train:
 [[   0 1253]
 [   0  170]]
              precision    recall  f1-score   support

  stroke = 0       0.00      0.00      0.00      1253
  stroke = 1       0.12      1.00      0.21       170

    accuracy                           0.12      1423
   macro avg       0.06      0.50      0.11      1423
weighted avg       0.01      0.12      0.03      1423

--------- Confusion Matrix und Scoring Report test -----
Confusion Matrix auf test:
 [[  0 314]
 [  0  42]]
              precision    recall  f1-score   support

  stroke = 0       0.00      0.00      0.00       314
  stroke = 1       0.12      1.00      0.21        42

    accuracy                           0.12       356
   macro avg       0.06      0.50      0.11       356
weighted avg       0.01      0.12      0.02       356

<class 'pandas.core.frame.DataFrame'>
```

Fazit:

- Wenn wir also ein Modell suchen, und bei der Suche zunächst den Recall der Klasse y=1 optimieren (erhöhen), sollten wir dieses Modell immer auch anhand der Precision bewerten!
- Idealerweise würden wir für jeden Algorithmus des Projekts einen Gridsearch machen, der Recall und Precision ausgibt, und für alle Algorithmen beides vergleichen. Wir nehmen das für unseren Algorithmus auf jeden Fall vor!

4. GridSearch und finales Modell

4.1 Ergebnisse GridSearch

Die GridSearch wurde im **Jupyternotebook „Gridsearch_SVC_original_data“** aufgesetzt. Dort wurde ein SVC mit einem StandardScaler zu einer Pipeline verbaut, und mit einem passend errichteten Parametergrid in die Gridsearch gegebem. Das Ergebnis (Modell, cv_re, und Parametergrid) wurde über pickle abgespeichert. Die Auswertung (Tabelle durchforsten, Plot) fand im **Jupyternotebook „Analyse_Gridsearch_SVC_areaplot“** statt.

Wir werden gleich sehen, dass sich unsere Mühen um gute Startparameter bezahlt machen: Wir konnten tatsächlich ein **sehr gutes Parameter-Grid** aufspannen! Wertebereiche und Schrittweite waren sehr gut gewählt:

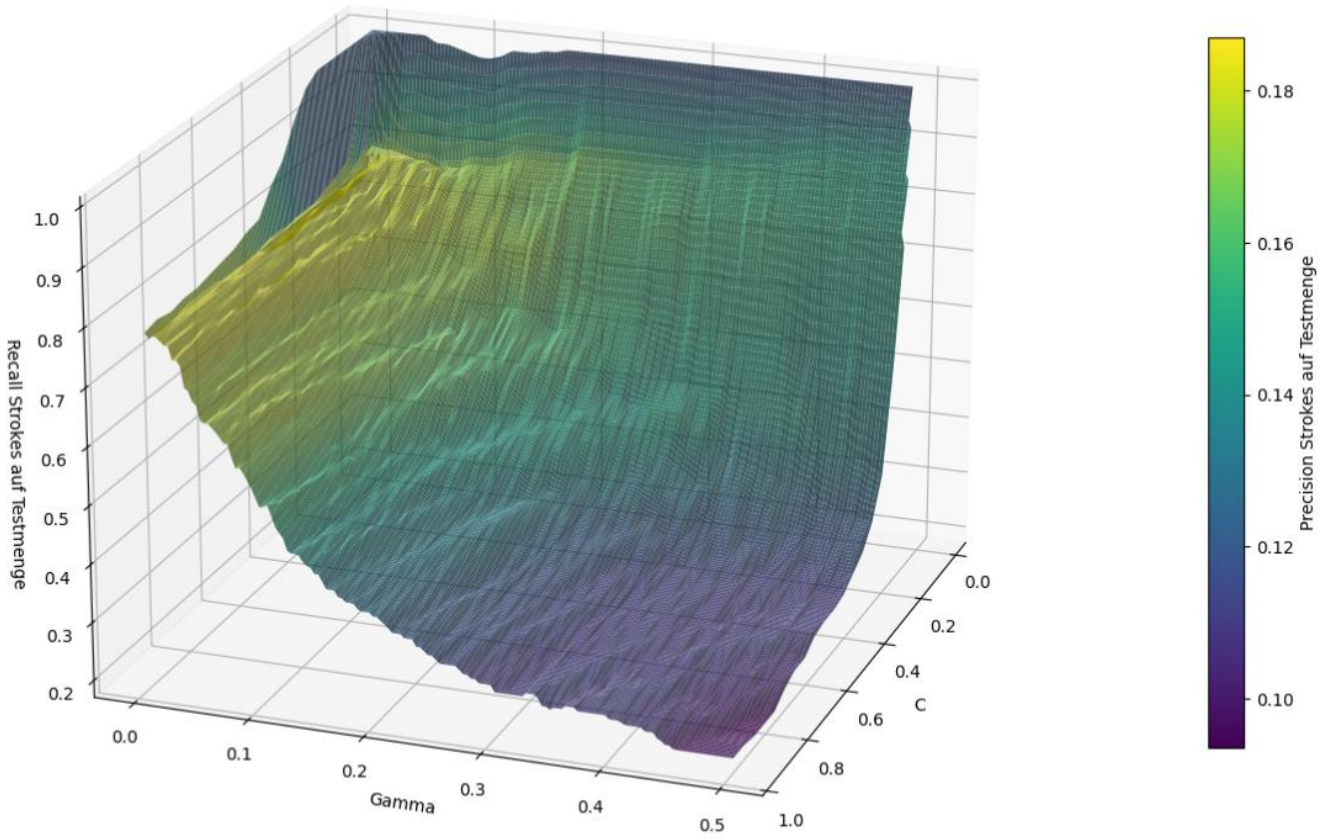
- Wir haben eine Liste erhalten, in der wir sortiert jeweils Recall und Precision ablesen können.
- Dabei haben wir genau die Kombinationen erwischt, bei denen stetig steigender Recall bei stetig fallender Precision nachvollziehbar ist.
- Damit können wir uns sehr gezielt die Variante aussuchen, bei der wir selbst den Tradoff zwischen Recall und Precision heraussuchen können!

Hier ein Blick in die obersten Zeilen der Tabelle:

```
ergebnis.sort_values(by=["mean_test_recall","mean_train_recall"], ascending=[False, False]).head(10)
```

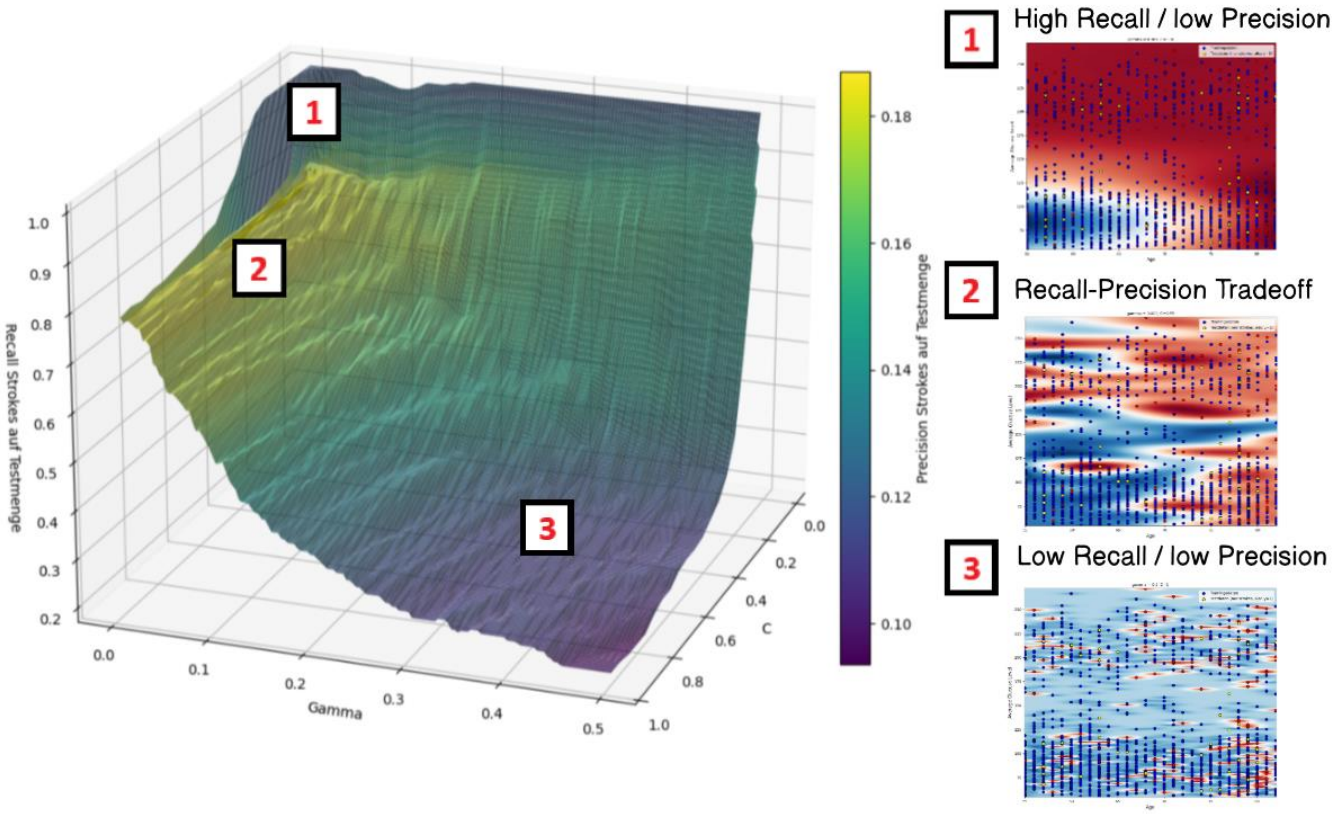
	mean_test_recall	mean_train_recall	mean_test_precision	mean_train_precision	param_svc__C	param_svc__gamma	param_svc__class_weight
980	0.894294	0.932899	0.135014	0.139517	0.10	0.010	{0: 1, 1: 11}
981	0.888342	0.929369	0.138764	0.142801	0.10	0.011	{0: 1, 1: 11}
982	0.882594	0.918773	0.141716	0.145438	0.10	0.012	{0: 1, 1: 11}
983	0.876847	0.916425	0.145100	0.149402	0.10	0.013	{0: 1, 1: 11}
984	0.864943	0.911722	0.147118	0.153361	0.10	0.014	{0: 1, 1: 11}
985	0.864943	0.907011	0.150387	0.156212	0.10	0.015	{0: 1, 1: 11}
2450	0.864943	0.904647	0.151109	0.155966	0.15	0.010	{0: 1, 1: 11}
986	0.864943	0.902307	0.153128	0.158377	0.10	0.016	{0: 1, 1: 11}
2451	0.864943	0.897604	0.155628	0.159805	0.15	0.011	{0: 1, 1: 11}
990	0.859401	0.887041	0.159200	0.165915	0.10	0.020	{0: 1, 1: 11}

Exemplarisch für die Gewichtsklasse 9 (der Plot mit Gewicht 10 oder 11 sieht ähnlich aus, höhere Gewichte verschieben jedoch die Fläche nochmals nach oben, teilweise jedoch mit Precision-Verlusten) die Kombinationen, mit gamma und C auf der x1- und x2-Achse, dem Test-Recall auf der x3-Achse, und der Test-Precision als:



Wir sehen nochmal: Es hat sich sehr gelohnt, viel Zeit in die initiale Parametersuche zu stecken, und das Parametergrid vorher gut abzustecken! Die gelben „Schlieren“ zeigen, dass man hier im Bruchteilbereich von Gamma bezüglich Precision teilweise einiges rausholen kann, ohne den Recall zu senken (im Gegenteil!).

Bevor wir uns das final ausgewählte Modell genauer ansehen, machen wir uns die Bereiche auf der Kurve nochmals klar:



1. High Recall / low Precision:
    a. Auf Trainings- und Testmenge maximaler Recall, minimale Precision
    b. Precision auf beiden Mengen 0
    Modelleignung: Alle Samples werden als Stroke identifiziert. Kein aussagekräftiges Modell:

```
Confusion Matrix auf test:
 [[  0 314]
 [  0  42]]
              precision    recall  f1-score   support

  stroke = 0       0.00      0.00      0.00       314
  stroke = 1       0.12      1.00      0.21        42
```

2. Recall-Precision Tradeoff:
    a. Recall und Precision bedingen sich gegenseitig (auf Trainings- und Testdaten)
    b. **Modelleignung**: Tradeoff-Entscheidung und Parameterwahl sind Ermessenssache des Anwenders/Auftraggebers/der Zielstellung

```
Confusion Matrix auf test:
 [[151 163]
 [ 16  26]]
              precision    recall  f1-score   support

  stroke = 0       0.90      0.48      0.63       314
  stroke = 1       0.14      0.62      0.23        42
```

3. Low Recall / low Precision:
    a. Resultat von Overfitting (Modell performt sehr gut auf Trainingsmenge, aber sehr schlecht auf Testmenge)
    b. **Modelleignung**: Keine gute Eignung für Testdaten da zu große Anpassung an Trainingsdaten

```
Confusion Matrix auf train:
 [[1035  218]
 [   9  161]]
              precision    recall  f1-score   support

  stroke = 0       0.99      0.83      0.90      1253
  stroke = 1       0.42      0.95      0.59       170
```

```
Confusion Matrix auf test:
 [[225  89]
 [ 30  12]]
              precision    recall  f1-score   support

  stroke = 0       0.88      0.72      0.79       314
  stroke = 1       0.12      0.29      0.17        42
```

4.2 Feinschliff finales Modell

Wir suchen uns anhand von Plot und Ergebnistabelle eine Parameterkombination aus dem Tradeoff-Bereich und versuchen diese final zu optimieren:

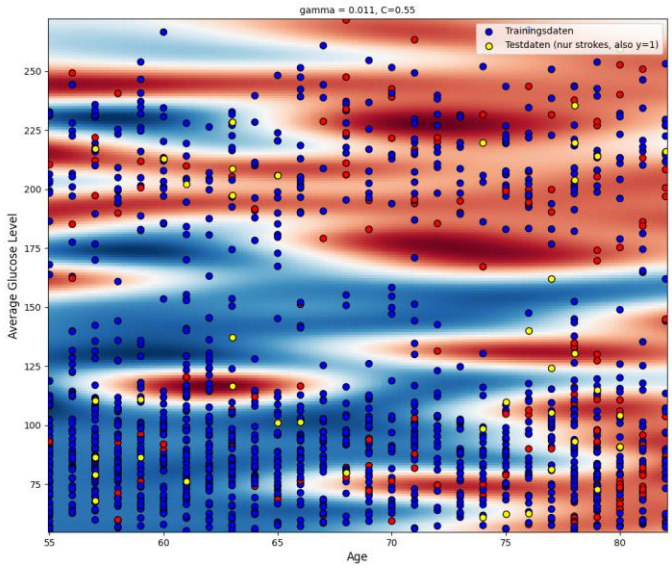
	mean_test_recall	mean_train_recall	mean_test_precision	mean_train_precision	param_svc__C	param_svc__gamma	param_svc__class_weight
13231	0.777094	0.824718	0.187031	0.197242	0.55	0.011	{0: 1, 1: 9}

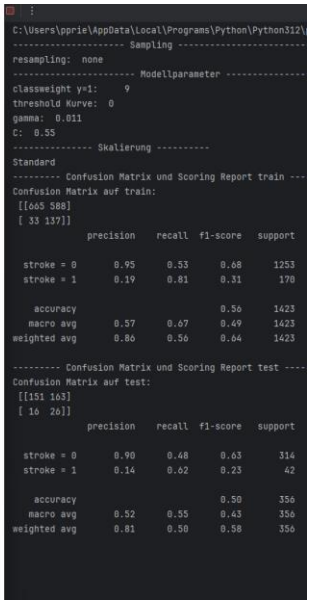
Beachte: Die Scores des Modells, die wir gleich sehen werden, sind auf den **echten** Testdaten, wohingegen die Ergebnisse der Gridsearch auf den **Testfolds der Trainingsdaten** generiert wurden! Wir rechnen also mit Abweichungen, die sich durch die nun „realen“ Testwerte ergeben. Hier nochmals in tabellarischer Form die **Ergebnisse auf den Testfolds der Trainingsdaten**:

Datengrundlage	- Training auf originären Attributen
C	0.55
Gamma	0.011
Gewichtung Strokes	9
Durchschnitt Recall (**Testfolds Trainingsdaten**)	0.78
Durchschnitt Precision (**Testfolds Trainingsdaten**)	0.19

4.2.1 Gewähltes Ausgangsmodell nach GridSearch

Hier rbf-Plot und Report des Modells mit obenstehenden Parametern:





```
C:\Users\pprie\AppData\Local\Programs\Python\Python312\
-------------------- Sampling --------------------
resampling:  none
---------------------- Modellparameter ----------------
classweight y=1:    9
threshold Kurve:  0
gamma:  0.011
C:  0.55
--------------- Skalierung ----------
Standard
--------- Confusion Matrix und Scoring Report train ---
Confusion Matrix auf train:
 [[665 588]
 [ 33 137]]
              precision    recall  f1-score   support

  stroke = 0       0.95      0.53      0.68      1253
  stroke = 1       0.19      0.81      0.31       170

    accuracy                           0.56      1423
   macro avg       0.57      0.67      0.49      1423
weighted avg       0.86      0.56      0.64      1423

--------- Confusion Matrix und Scoring Report test ----
Confusion Matrix auf test:
 [[151 163]
 [ 16  26]]
              precision    recall  f1-score   support

  stroke = 0       0.90      0.48      0.63       314
  stroke = 1       0.14      0.62      0.23        42

    accuracy                           0.50       356
   macro avg       0.52      0.55      0.43       356
weighted avg       0.81      0.50      0.58       356
```

4.2.2 Recall-Verbesserung durch höheres Klassengewicht

Wir erhöhen die Klassengewichtung der strokes nun von 9 auf 11 und vergleichen:

Gewichtung strokes: 9

```
Confusion Matrix auf test:
 [[151 163]
 [ 16  26]]
              precision    recall  f1-score   support

  stroke = 0       0.90      0.48      0.63       314
  stroke = 1       0.14      0.62      0.23        42
```

Gewichtung strokes: 12

```
Confusion Matrix auf test:
 [[117 197]
 [  7  35]]
              precision    recall  f1-score   support

  stroke = 0       0.94      0.37      0.53       314
  stroke = 1       0.15      0.83      0.26        42
```

Fazit: Bessere Stroke-Erkennung, prozentual minimale Verbesserung bzgl. Precision.

4.2.3 Recall-Verbesserung Featureselektion: Risikopatienten

Durch geeignete Featureselektion kann gegebenenfalls eine bessere Klassentrennung auf den Daten erzielt werden. Konkret ist unsere Idee: Wenn unter den Risikopatienten (Spalte „at_least_one_risk“) der Anteil an strokes deutlich höher ist, kann der Algorithmus über dieses Feature gut Teilmengen abtrennen! Wir verwenden das Modell also auf einer angepassten Datenmenge, in der wir lediglich die Spalten „age“ (= Alter) und „at_least_one_risk“ (=Risikopatienten) behalten. Dadurch ergibt sich bei Beibehalten der obigen Parameterwahl (C=0.55, gamma = 0.011, Gewicht strokes = 12):

Originale Datengrundlage:

```
Confusion Matrix auf test:
 [[117 197]
 [  7  35]]
              precision    recall  f1-score   support

  stroke = 0       0.94      0.37      0.53       314
  stroke = 1       0.15      0.83      0.26        42
```

Datengrundlage mit Risikoeinstufung:

```
Confusion Matrix auf test:
 [[ 43 271]
 [  3  39]]
              precision    recall  f1-score   support

  stroke = 0       0.93      0.14      0.24       314
  stroke = 1       0.13      0.93      0.22        42
```

Hier finden wir also nochmals deutlich mehr stroke-Tatienten (Recall steigt). Allerdings erhöht sich auch hier anteilig stärker die false-positive-Rate, was zu einer prozentual sinkenden Precision führt.

Wir leben hiermit und betrachten zuletzt, ob sich durch Setzen einer Entscheidungsgrenze für die Prognose stroke oder nicht nochmals der Recall mit verkraftbarem Precision-Verlust optimieren lässt.

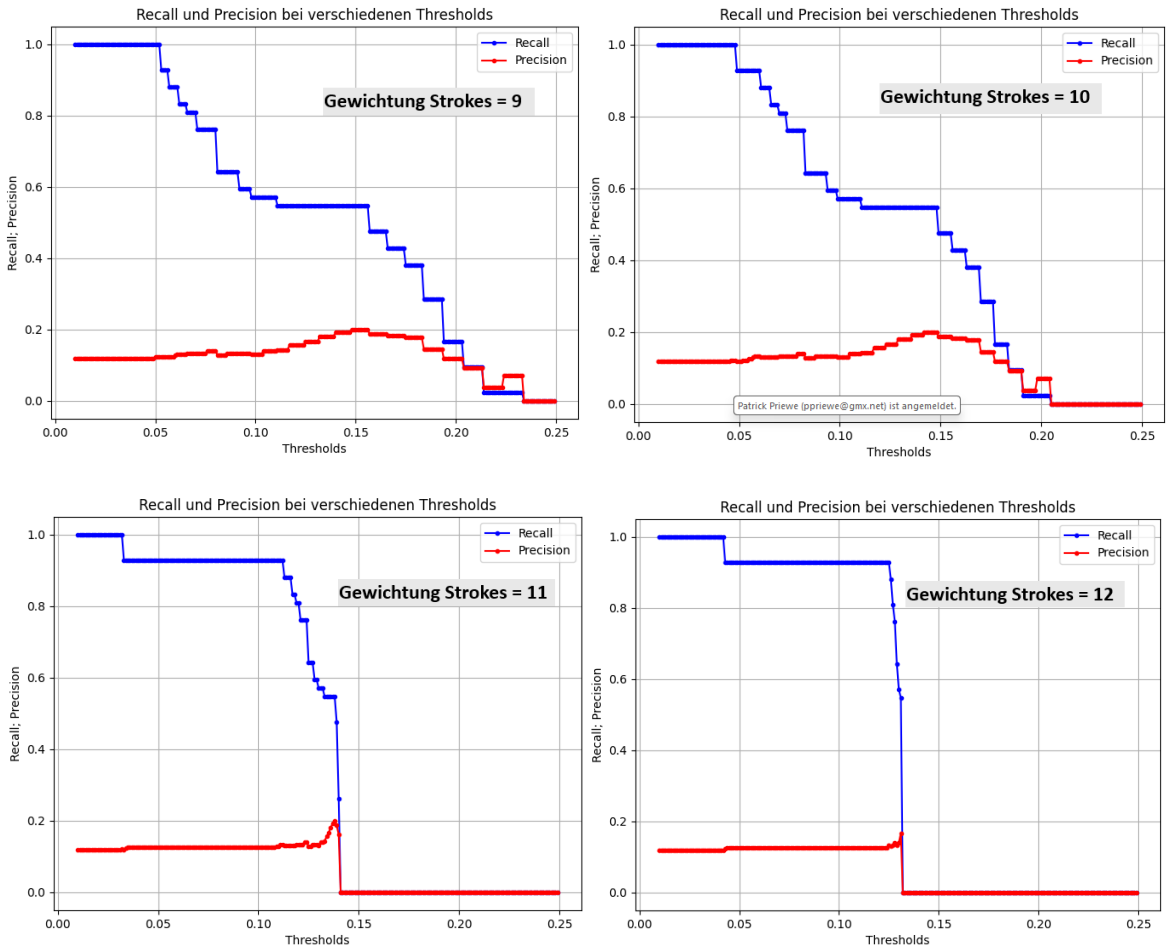
4.2.4 Thresholding

Für die vorhergesagten Klassenwahrscheinlichkeiten eines Modells können wir über eine Entscheidungsgrenze selbst eine neue stroke-Prognose machen: Wir legen einen Grenzwert fest, und alle Datenzeilen aus der Testmenge, deren vorhergesagte stroke-Wahrscheinlichkeit größer als unser Grenzwert ist, sollen der Klasse stroke zugeordnet werden.

Für diese Prognose wiederum können wir wieder Recall und Precision ausrechnen. So können wir also nachträglich die Prognosen unseres aus der Gridsearch gewonnenes Modells nochmals nachjustieren, und schauen, ob wir etwas verbessern.

Wenn im Analyseprogramm show_threshold_curve=1 gesetzt ist, machen wir das über eine Schleife, und am Ende des Analyseprogramms wird die entsprechende Kurve angezeigt.

Für unser **oben gewähltes und optimiertes Modell (C=0.55, gamma = 0.011, Gewicht strokes = 12) mit Verwendung des Features „at_least_one_risk“ (Risikopatienten)** betrachten wir die Precision-Recall-Kurve nun für die Gewichtsklassen 9, 10, 11, 12, und ziehen eine Schlussfolgerung, ob sich das Setzen einer Entscheidungsgrenze für uns noch lohnt:



Wir sehen hier, dass sich bei niedrigeren Gewichtungen der strokes (9 und 10) eine stärkere Abstufung des Tradeoffs zwischen Recall und Precision bildet (Treppenstruktur der Kurven im linken Bereich). Bei höheren Gewichten findet eine Plateaubildung statt. Die Wahl der Gewichtungen 9 und 10 würde insgesamt keine Verbesserungsmöglichkeiten bieten, da Recall-Erhöhungen auf über 0.9 auch dort letztlich ebenfalls stets zu einer Precision im Bereich unter 0.2 führen würde. **Mit unserem gewählten Modell und der Gewichtung 12 befinden wir uns in der Recall-Precision-Kurve also mit dem Recall auf dem blauen Plateau (Threshold-Bereich zwischen 0.05 und 0.1, Schaubild rechts unten im obigen Multiplot)**.

Wir erkennen also: **Das Setzen von Entscheidungsgrenzen liefert hier per se keine weiteren Verbesserungen**. Allerdings erzielen wir ein **besser steuerbares Abstufen** des Recalls für Entscheidungsträger, wenn wir geringere Gewichte erzeugen.

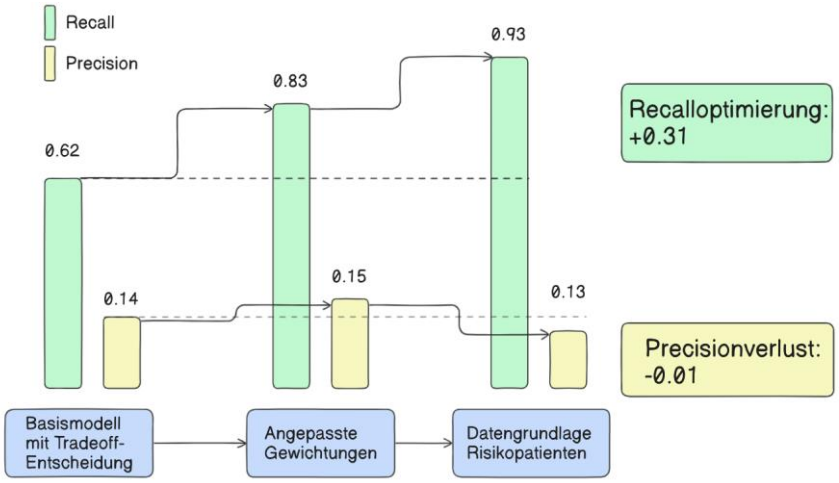
Wir befinden uns nun allerdings bereits im höherdimensionalen Parameterraum (C, gamma, Gewichtung, Entscheidungsgrenze). Final wäre also eine umfassende Pipeline, wie Sie im Kapitel zu den weiteren Ideen beschrieben wird, eine Möglichkeit zur globalen Optimierung des Gesamtmodells (mit Featureselektion, Resampling, Modelloptimierung und Thresholding).

4.2.5 Finales Modell – optimiert ohne Oversampling

Wir halten das finale Modell fest, wobei Maßnahmen zur Verbesserung ***fett kursiv*** dargestellt sind:

Datengrundlage	***- Training auf reduzierten Daten:*** ***- originäres Attribut „age“*** ***- erzeugtes Zusatzattribut „at_least_one_risk“ (Risikopatienten)***
C	0.55
Gamma	0.011
Gewichtung Strokes	**12**
Recall (**auf echten Testdaten**)	0.93
Precision (**auf echten Testdaten**)	0.13

Wir stellen hier abschließend die durch Gewichtung und Featureselektion erzielten Verbesserungen **auf der echten Testmenge (also nicht den Testfolds der Gridsearch!)** nochmals dar, und halten fest, dass bei in etwa gleichbleibender prozentualer Precision ein deutlicher Recall-Gewinn erzielt werden konnte:



4.3 Vergleich mit partiellem Resampling der Minderheitenklasse

4.3.1 Resamplingmethodik

Wir wollen nun den Algorithmus noch für Daten vergleichen, bei denen wir die Minderheitsklasse durch synthetische Daten künstlich erhöhen.

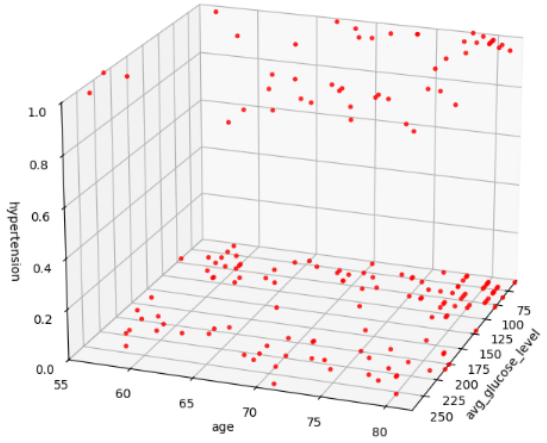
Wichtiger Hinweis: Wir nehmen dies hier auf den **gesamten Trainingsdaten** vor! Das bedeutet, dass die **Crossvalidierung** anhand von Trainings- und Testfolds (**beides auf der Trainingsmenge**) im Rahmen der Gridsearch auch auf den Testfolds synthetisierte Daten vorfindet. Für einen sauberen Test der Auswirkung von Resampling auf das Modell (bei nicht overgesampelten Testdaten) müsste man das Oversampling in einen Datentransformer einbauen, und diesen in die Pipeline einführen, die in die Gridsearch gegeben wird. Hierauf verzichten wir hier. Uns reicht, eine vergleichende Betrachtung des Modells auf einer Datenlage, bei der die Minderheitsklasse in den Gesamtdaten angereichert wird, vorzunehmen.

Wir nutzen die Methode SMOTE aus der Bibliothek imblearn für unbalancierte Klassen. Die Methode synthetisiert **Linearkombinationen zwischen n Nachbarn ausschließlich der Minderheitsklasse (die Mehrheitsklasse wird nicht berührt)**. Dabei ist n ein steuerbarer Parameter. Bei der Methode ist jedoch Vorsicht geboten, da keine Rücksicht auf die Zerstreuung von Daten oder Entfernungen in

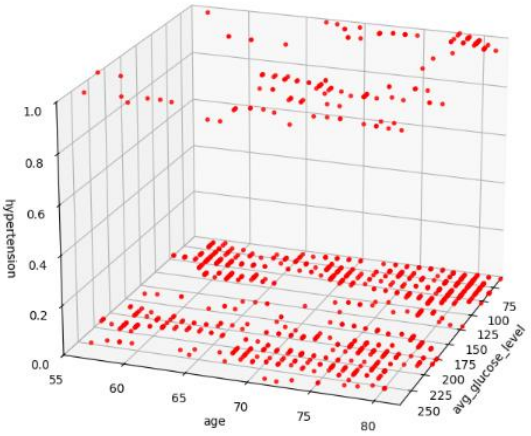
Nachbarschaften genommen wird. Bei ungünstigen Daten kann das dazu führen, dass künstlich synthetisierte Minderheitsklassenelemente in Bereichen erzeugt werden, in denen in den Trainingsdaten ausschließlich Mehrheitsklassenelemente liegen. Das kann später dazu führen, dass der Anteil an falsch positiven Resultaten (kein stroke liegt vor, es wird aber einer vorhergesagt) nach oben verzerrt wird.

Wir visualisieren als Beispiel die strokes des gesamten (auf die dreidimensionalen Daten Alter, Blutzuckerwert und Bluthochdruck reduzierten) Trainingsdatenraumes einmal ohne Resampling (SMOTE) und einmal mit Resampling:

Ohne Oversampling:

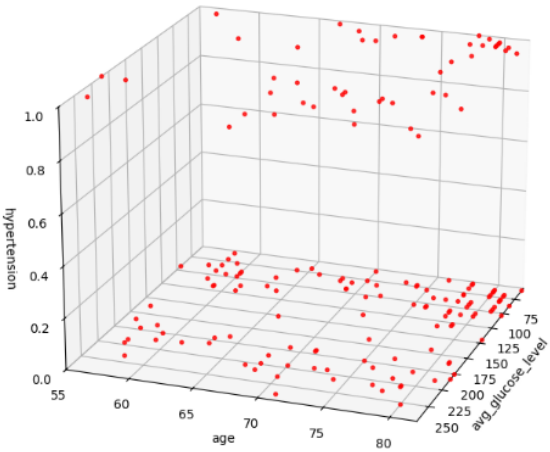


Mit großflächigem Oversampling:

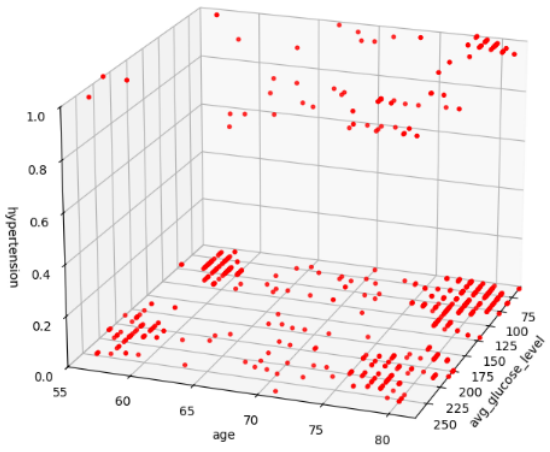


Dabei können wir erkennen, dass (wie gewünscht) die stroke-Dichte erhöht wurde, jedoch auch in Bereichen mit (vorher) geringer stroke-Dichte. Das wollen wir nicht, um resamplen daher nur in ausgewählten Bereichen, in denen auch vorher schon „Ballungszentren“ vorliegen. Das könnte wie folgt aussehen:

Ohne Oversampling:



Mit gezieltem Oversampling:

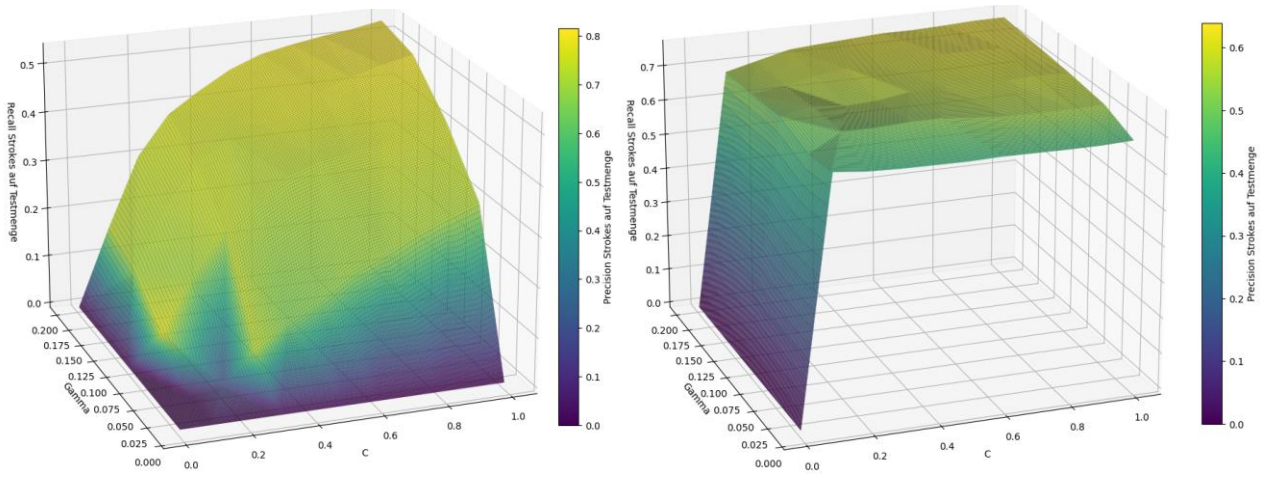


Mit diesem alternativen Datensatz gehen wir nun ebenfalls in eine (kurze) Analyse und untersuchen, wie unser Modell verbessert werden kann.

4.3.2 Tradeoff-Verhalten: Gezieltes Resampling, Originalattribute

Wir betrachten für eine grobe Gridsearch das Resultat der **Recall-Precision-Fläche** für die **stroke-Gewichtungen 1 und 2**. Dabei untersuchen wir einen **Algorithmuslauf über die Originalspalten (ohne Zusatzattribute)**:

Gewichtung strokes: 1 **Gewichtung strokes: 2**



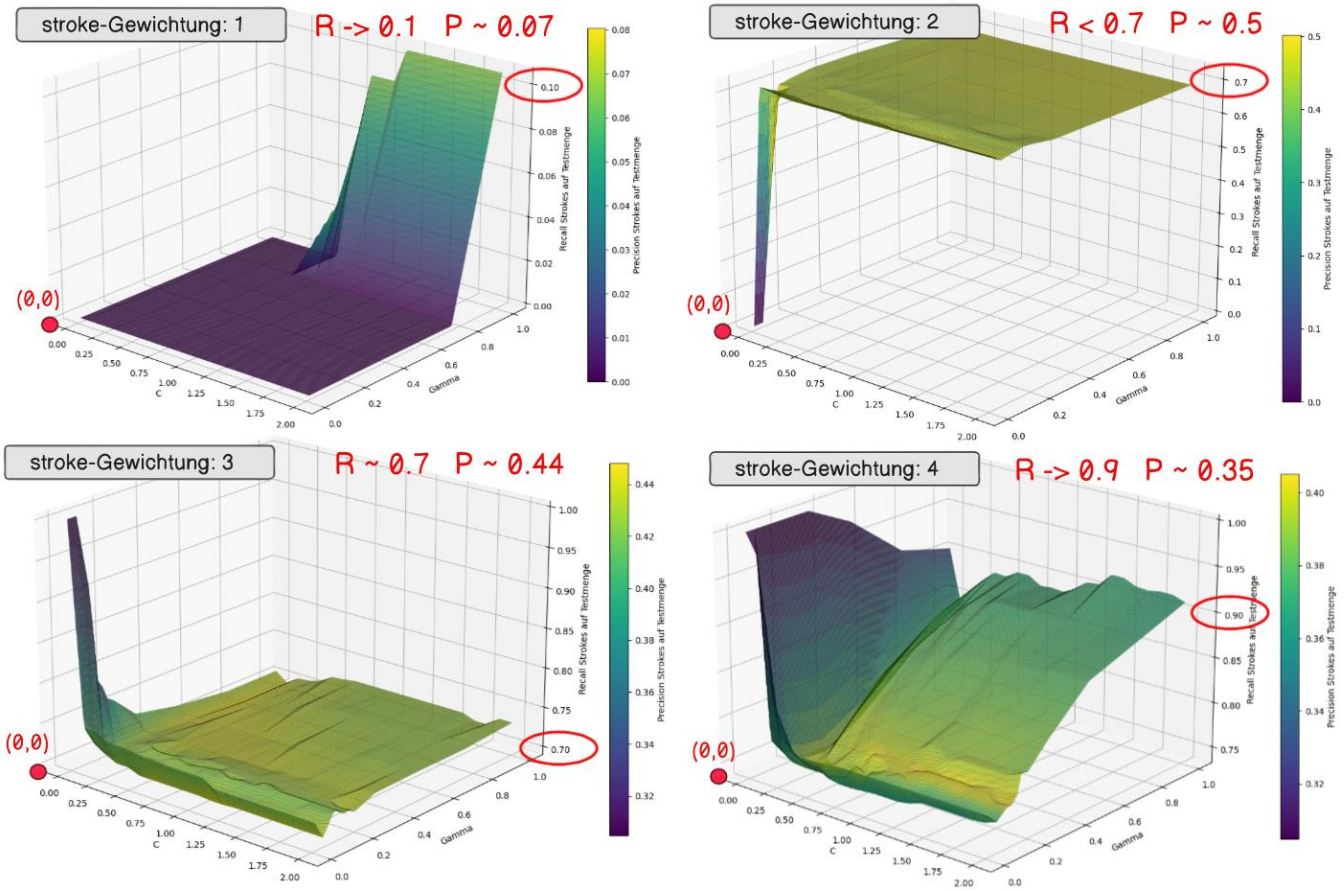
Dabei ist zu beachten, dass der Nullpunkt der x1- und x2- Achsen (Boden) hier jeweils vorne links ist.

- Es lässt sich beobachten, dass hier (im Gegensatz zur Datenlage ohne Resampling) größere gamma- und C-Werte (ebenso größere Klassengewichtung für die strokes) zu höherem Recall und besserer Precision führen (die ehemals konvexe Form wird in einen konkaven Charakter „umgestülpt“).
- Ebenso kann man an der Recall-Achse (senkrecht) ablesen, dass der der Recall an der gelben Spitze der Fläche (großes C, großes Gamma) wieder bei höherer Klassengewichtung steigt. Diesmal scheint der Tradeoff bezüglich Precision weniger drastisch zu sein.

4.3.3 Tradeoff-Verhalten: Gezieltes Resampling, Attribut: Risikopatienten

Zum Vergleich betrachten wir für eine feinere Gridsearch das Resultat der **Recall-Precision-Fläche** für die **stroke-Gewichtungen 1, 2, 3 und 4**. Dabei untersuchen wir einen **Algorithmuslauf nur über die Attribute Alter und Risikopatien („at_least_one_risk“)**.

Dabei wird klar, dass die **Auswahl der Attribute**, die dem Algorithmus „zugefüttert“ werden, in **Kombination mit der Klassengewichtung** der Strokes eine **enorme Rolle für die Tradeoff-Geometrie** spielt:



Wir können aus den gezeigten Geometrien und der Ergebnistabelle des Gridsearch also Entscheidungsvorlagen liefern (oder selbst entscheiden), wie wir hier wählen. Die Visualisierung zeigt drastisch, welchen Einfluss Klassengewichtung und Attributauswahl haben. Die starken Verformungen bei der Erhöhung der Klassengewichtung sind hier speziell bei der Attributszufütterung „Risikopatient“ an den Algorithmus zu beobachten.

4.3.4 Finales Modell – optimiert mit gezieltem Oversampling

Wir halten das finale Modell fest, und betrachten **die Ergebnisse auf den Testfolds der Trainingsdaten.** Maßnahmen, die wir hier (im Vergleich zum Originalmodell aus Kapitel 5.2) vorgenommen haben, sind ***fett kursiv*** dargestellt.:

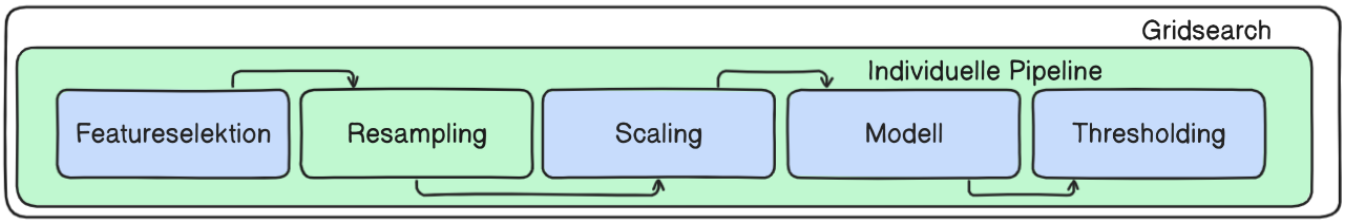
	mean_test_recall	mean_train_recall	mean_test_precision	mean_train_precision	param_svc__C	param_svc__gamma	param_svc__class_weight
41	0.972813	0.975650	0.311783	0.311784	0.005	0.500	{0: 1, 1: 4}

Datengrundlage	***- partielles Resampling zur Verdichtung ausgewählter „stroke-Inseln“*** ***- Training auf reduzierten Daten:*** ***- originäres Attribut „age“*** ***- erzeugtes Zusatzattribut „at_least_one_risk“ (Risikopatienten)***
C	0.005
Gamma	0.5
Gewichtung Strokes	4
Durchschnitt Recall (Testfolds Trainingsdaten)	0.97
Durchschnitt Precision (Testfolds Trainingsdaten)	0.31

Wir haben hier also durch das **Training ausschließlich auf dem Alter und der Risikopatienteneigenschaft** und das **Resampling ausgewählter Datenbereiche der Minderheitsklasse** auf den **Gesamtdaten** (einschließlich Testdaten) **erhebliche Verbesserungen** herbeigeführt! Das reicht uns als weiteres Gütekriterium für unser Modell.

5. Mehrschrittige Pipeline für globale Optimierung

Zur weiteren Verbesserung des Modells könnten wir die zu Beginn angesprochene Pipeline - weitestgehend - vollenden. Wir bauen dafür alle in der folgenden Abbildung blau dargestellten Teilschritte mit den entsprechenden optimierungsfähigen Parametern ein:



Transformer für Featureselektion: Hier verwenden wir den eigens erstellten Spaltentransformer „ColumnSelector“. Dieser akzeptiert als Eingabeparameter eine Liste an Spalten, die das Modell für Training und Prognose verwendet (Whitelist).

(Resampling): Auf den vollständigen Einbau eines entsprechenden Resampling-Transformers verzichten wir im Rahmen des Projekts. Ein eigenständiger Sampling-Transformer „PartialSmoteResampler“ wurde erstellt und erfolgreich alleinstehend getestet.

- Er akzeptiert als Eingabeparameter eine Indexmenge der Datenzeilen in X, was eine vorangestellte gezielte Ermittlung dichter Teilmengen erlaubt, die resampled werden sollen. Er gibt eine entsprechend erweiterte Menge X und Y zurück.
- Die Nutzung des Resampling-Transformers in einer sklearn-Pipeline ist jedoch aktuell nicht möglich (da eine solche von Transformern im Rahmen des fit_transform lediglich den Return von X erwartet, wir geben jedoch X und Y zurück). Möglich wäre, auf eine imblearn-Pipeline umzusteigen, und dort nochmal die Tauglichkeit (und Fähigkeit zur Einschränkung auf Teilmengen) der Resampling-Methoden aus imblearn zu prüfen.

Scaling: Bereits verfügbar, trivialer Einbau

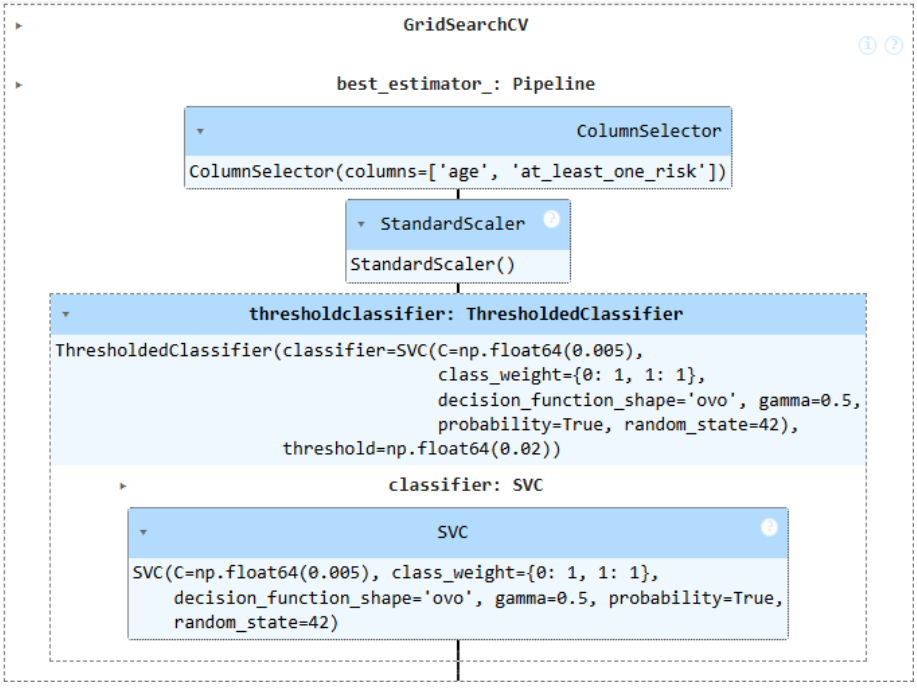
Thresholding: Eine entsprechende Wrapperklasse ThresholdClassifier wurde erstellt, die im Kern einen SVC verwendet und das Ergebnis anhand eines Threshold-Parameters entsprechend manipuliert (die vom SVC ausgegebene Wahrscheinlichkeit für einen stroke wird anhand der Entscheidungsgrenze zu einer Prognose 1 oder 0):

```python
class ThresholdedClassifier(BaseEstimator, ClassifierMixin):  2 usages
    def __init__(self, classifier = None, threshold=0.1):
        self.classifier = classifier
        self.threshold = threshold

    def fit(self, X, y):
        self.classifier.fit(X, y)
        return self

    def predict_proba(self, X):
        return self.classifier.predict_proba(X)

    def predict(self, X):
        y_proba = self.predict_proba(X)[:, 1]  # Wahrscheinlichkeiten für Klasse 1
        return (y_proba >= self.threshold).astype(int)
```

Damit können wir alle Verbesserungstechniken außer Resampling in die Pipeline holen und damit auch eine Gridsearch mit entsprechend erweitertem Parametergrid durchführen. Exemplarisch hier ein klein gefasstes Grid mit dem resultierenden optimiertem Modell:

```python
column_combinations = [["age", "at_least_one_risk"], ["age", "risk_sum"]]
y_weights = list(range(1, 3, 1))
list_weight_classes = [{0: 1, 1: value} for value in y_weights]
list_of_C_values = np.arange(0.005, 1.05, 1)
list_of_gamma_values = [0.5, 1]
thresholds = np.arange(0.02, 0.03, 0.005)
grid_search_name = "gridsearch_multistep_pipeline.pkl"

grid_parameters = {
            "columnselector__columns": column_combinations
            , "thresholdclassifier__threshold": thresholds
            , "thresholdclassifier__classifier__C": list_of_C_values
            , "thresholdclassifier__classifier__gamma": list_of_gamma_values
            , "thresholdclassifier__classifier__class_weight": list_weight_classes
                }
```

Resultat:



Auf die Durchführung einer Gridsearch mit breitflächigem und engmaschigem Grid müssen wir leider aufgrund mangelnder Rechenressourcen verzichten. Allerdings haben wir noch eine Abwandlung des Analyseprogramms explizit für unsere Pipelines erstellt.

6. Zusammenfassung und Fazit

Zusammenfassung:

- **Erprobungsphase der Startparameter:**
  - Durch deren intensive Vorsondierung für die Gridsearch mithilfe der Plots und des strukturierten Reports haben wir eine gute Ausgangslage für die Gridsearch erzeugt.
- **Einblick in das Tradeoffverhalten** :
  - Mithilfe einer GridSearch haben tiefe Einblicke in das Tradeoff-Verhalten zwischen Recall und Precision erhalten. Wir haben wir damit ein **erstes Modell gewählt**, das eine deutliche **Recallverbesserung** ermöglicht, und dabei trotzdem einen **Blick auf die Precision** gehabt.
- **Erhöhung der Klassengewichtung und Training Zusatzattributen:**
  - Über die anschließende nochmalige Erhöhung der Klassengewichtung für die Minderheitenklasse, sowie dem Training auf den Attributen Alter (originär) und Risikopatienteneigenschaft (Zusatzfeature) konnten wir den Recall nochmals ohne relevante Senkung der Precision erhöhen.
- **Vergleich des Modells an teilweise synthetischen Daten**
  - Mithilfe eines **gezielten Resampling-Ansatzes (der radikal andere Parameterkombinationen für den Modellkern erfordert)** sowie dem **Training auf den Attributen Alter und Risikopatienteneigenschaft** haben wir das Modell final auf eine Situation angewandt, in der die Minderheit stärker repräsentiert war als vorher. Dabei haben wir gesehen, dass das Modell auf den Testfolds der Trainingsmenge bei **sehr hohem Recall** zusätzlich eine **optimierte Precision** hat.
- **Erstellung mehrschrittige Pipeline**:
  - Wir haben eine mehrschrittige Pipeline erstellt, die zusätzlich zu den Modellparametern die Attributauswahl (Featureselektion) und die Entscheidungsgrenze (Thresholding) optimierbar im Rahmen einer Gridsearch macht. Ein optimierbares Resampling zu integrieren haben wir aus Aufwandsgründen hier weggelassen.

Fazit:

- Die Wahl des Recall und der Precision war absolut wesentlich.
- Das visuelle Auswerten des Tradeoff-Bereichs mag für die reale Praxis eventuell weniger relevant sein (da dort die reine Untersuchung einer Ergebnisliste der Gridsearch ausreicht). Sie trägt aber enorm zum Verständnis des Algorithmusverhaltens und der Auswirkung der Verbesserungsmaßnahmen bei, die erprobt wurden.
- Es lohnt sich, die Wirkung von Zusatzattributen und kleinster Parameteränderungen zu untersuchen. Ein „nacktes“ Modell kann, wie wir gesehen haben, deutlich optimiert werden.
- Eine größer angelegte vollständige Pipelineuntersuchung mit Featureselektion, Datentransformationen, feiner aber weiter Parameteranalyse, und ggf. Thresholding würde das Modell vermutlich nochmals optimieren
- Die wirksamste Methode zur Verbesserung wäre jedoch eindeutig, die Datenqualität zu erhöhen
- Im Sinne eines Übungsprojektes waren obige Untersuchungen extrem aufschlussreich und kompetenzbildend.