Fachhochschule
Dortmund
University of Applied Sciences and Arts

Angewandtes Programmierprojekt

„Anwendung und Erarbeitung eines Random Forest Klassifikators in Python am Beispiel des Framingham Heart Study Datensatzes“

Laura Garcia Petershof
Master Biomedizinische Informationstechnik
Matrikelnummer: 7216187

Modul: PA 11051 – Angewandtes Programmierprojekt
Prüfer: Prof. Dr. Burkhard Igel und Dr. Georg Pietrek
Abgabe: 05.02.2023

Inhalt

1. Einleitung

Gemäß dem *Robert-Koch-Institut* (RKI) sind Herz-Kreislauf-Erkrankungen für über 40 % der Sterbefälle in Deutschland verantwortlich, womit diese mit zu den häufigsten Todesursachen zählen. Gleichzeitig führen Herz-Kreislauf-Erkrankungen zu einer Minderung der Lebensqualität und erzeugen hohe Kosten für die gesetzlichen Krankenkassen [1]. Aufgrund dessen ist es von hoher Bedeutung, das Risiko eines Patienten für eine Herz-Kreislauf-Erkrankung möglichst frühzeitig zu identifizieren, sodass dieser durch Prävention vorgebeugt werden kann. Somit können die Lebensqualität der Patienten erhalten und die Krankheitskosten minimiert werden.

Durch die fortschreitende Entwicklung des maschinellen Lernens und der künstlichen Intelligenz, erhält diese insbesondere im Bereich medizinischer Anwendungen ein immer größer werdendes Interesse. Anwendungsbereiche des maschinellen Lernens in der Medizin liegen beispielsweise in der Steuerung von Robotern oder in der Vorhersage und Diagnostik von Erkrankungen eines Patienten.

Im Rahmen des Mastermoduls *Angewandtes Programmierprojekt* wurde mit Hilfe eines öffentlich zur Verfügung gestellten Datensatzes ein Random Forest Klassifikator in Python trainiert. Das trainierte Random Forest Modell soll in diesem Fall eine Vorhersage über das Risiko einer möglichen Herzerkrankung in den nächsten 10 Jahren treffen.

Für die Modellentwicklung lag der Datensatz der Framingham Heart Studie zur Verfügung. Diese Studie wird bereits in der dritten Generation in Framingham seit 1948 durchgeführt und dient zur Identifikation möglicher Ursachen und Risiken von koronaren Herzerkrankungen. Die Studie zählt zu einer der bedeutsamsten Studien in den USA. Mittlerweile gehören die Ergebnisse dieser Studie zum medizinischen Standard [2].

Der Datensatz ist öffentlich verfügbar und steht unter folgender Adresse zum Download bereit: https://www.kaggle.com/datasets/aasheesh200/framingham-heart-study-dataset

In der folgenden schriftlichen Ausarbeitung zum Projekt wird zunächst auf die verwendete Machine-Learning Methode eingegangen. In diesem Fall wurde ein Random Forest verwendet, da dieser sich sehr gut für Klassifizierungsprobleme eignet und gegenüber anderen Methoden einige Vorteile besitzt. Zum Beispiel weist die Methode eine kurze Trainingszeit auf, kann große Datenmengen und Merkmale präzise verarbeiten und jede Entscheidung lässt sich aufgrund der Baumstruktur im Nachhinein nachvollziehen [3]. Im Anschluss wird der Code und seine objektorientierte Struktur thematisiert. Beispielsweise werden die einzelnen Prozessschritte und die verwendeten Methoden der sklearn-Bibliothek dargestellt. Zusätzlich wird ein Blick in die verwendete Resampling-Methode und auf das Thema Unittest geworfen.

Ein wesentlicher Teil des Moduls war das Erlernen und Erarbeiten der agilen Methodik SCRUM. Für diese war neben der Teilnahme an einem Workshop, die Aufgabe einen persönlichen Sprint und Backlogplan zu erstellen. Dieser befindet sich in Abschnitt 4 dieser Abgabe.

2. Random Forest

Der vorliegenden Datensatz *Framingham Heart Study* umfasst 4240 Patienten-Daten mit jeweils 15 Eigenschaften und der Angabe einer Zielvariable. Die Zielvariable beschreibt, ob der Patient ein Risiko für eine Herzerkrankung in den nächsten 10 Jahren besitzt. Die 15 Eigenschaften umfassen unter anderem demographische Eigenschaften, wie Geschlecht und Alter, das Rauchverhalten der Patienten und medizinische Indikatoren, wie der Blutdruck, einzunehmende Medikamente und die Herzrate.

Vor dem Hintergrund, dass der Datensatz eine Zielvariable aufweist, kann angenommen werden, dass es sich hierbei um eine überwachte (supervised) maschinelle Lernmethode handelt. Aufgrund dessen wurde im Code ein Random Forest Klassifikator, der zur Gruppe der überwachten maschinellen Lernmethoden zugehört, implementiert.

Ein Random Forest setzt sich aus *n* - verschiedenen Entscheidungsbäumen zusammen und wird daher häufig auch als Ensemble-Methode bezeichnet. Dies liegt daran, dass die diversen Entscheidungsbäume als Ensemble/Gruppe bei der Vorhersage eines Ereignisses zusammenarbeiten. Es werden dementsprechend mehrere grundlegende Machine-Learning Methoden verwendet, die folglich eine gemeinsame Entscheidung treffen [4].

Im Falle des Random Forest trifft jeder einzelne Entscheidungsbaum in Abhängigkeit seiner ausgewählten Merkmale eine individuelle Entscheidung. Die abschließende Vorhersage wird nach dem Mehrheitsprinzip beschlossen. Das bedeutet, dass das Ergebnis ausgegeben wird, welches von den meisten Entscheidungsbäumen klassifiziert wurde [4].

Der Vorteil gegenüber eines einzelnen Entscheidungsbaum ist es, dass meistens mehrere unterschiedliche Bäume, die diversen Merkmale für ihre Entscheidung in Betracht ziehen, eine größere Stichprobe abdecken und folglich deren aggregierte Vorhersage präziser ist als die eines einzelnen Entscheidungsbaums [4].

Für die Leistung des Random Forest ist es von hoher Bedeutung, dass die einzelnen Entscheidungsbäume untereinander nicht korreliert sind. Hierfür wird das sogenannte Bagging verwendet, welches sich aus der Methode des Bootstrapings und der Aggregation zusammensetzt. Als Bootstrap wird hierbei definiert, dass der Datensatz in verschiedene Teile zerlegt und der Random Forest im Anschluss nur auf diese Stichprobe trainiert wird. Diese Methodik gewährleistet, dass die Bäume nicht korreliert sind. Die Aggregation fasst anschließend die Ergebnisse aller Entscheidungsbäume zusammen [4][5].

Der Random Forest Algorithmus umfasst drei wesentliche Schritte [6]:

1. Es werden *n* - Bootstrap-Beispiele ausgewählt.
2. Von den **M** Features werden an jedem Knoten des Baums, an denen eine Entscheidung getroffen werden muss, **m << M** Merkmale zufällig aus dem Datensatz gewählt. Diese Merkmale dienen als Kriterium für die nächste Verästelung des Entscheidungsbaums.
3. Der Baum wird ausgebaut. Ein Zurückschneiden des Baums findet nicht statt.

Anschließend werden die Ergebnisse der *n* - Entscheidungsbäume ausgewertet und die Klasse, welche am häufigsten vorhergesagt wurde, wird als Ergebnis des Random Forest angenommen (siehe oben) [6].

3. Code

Im folgenden Kapitel soll auf die Struktur des Codes sowie auf die verschiedenen Methoden, die zum Trainieren des Random Forest Algorithmus verwendet wurden, eingegangen werden. In Kapitel 3.3 wurde eine Methode erarbeitet, mit der die Genauigkeit des Algorithmus und die daraus resultierende Vorhersage der Wahrscheinlichkeit einer Herzerkrankung verbessert werden kann. Zum Schluss wurde in Kapitel 3.4 die Bedeutung von Unittest und die Implementierung im Code aufgezeigt.

3.1. Struktur

Der Programmcode umfasst neben dem Hauptprogramm *main*, die objektorientierten Klassen *Daten, FeatureSelection, Resampling, TestTrain* und *Random Forest*. Diese Klassen werden zum Trainieren des Random Forest Klassifikator verwendet und greifen nacheinander aufeinander zu (siehe Abbildung 1).

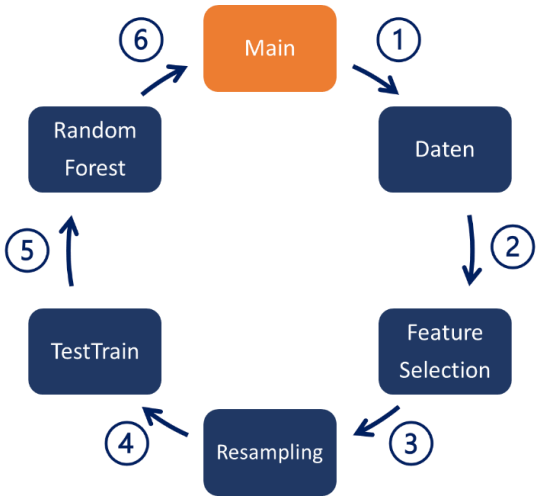

Abbildung 1: Code-Struktur (eigene Darstellung).

Das Programm wird aus der Main-Umgebung heraus gestartet. Diese zieht sich zunächst das trainierte Random Forest Modell, bevor der Anwender aufgefordert wird, eine Eingabe zu machen.

Hierfür werden als erstes die Daten aus der CSV-Tabelle der *Framingham Heart Studie* eingelesen und die zehn besten Features und deren zugehörigen Daten aus dem Datensatz extrahiert. Die zehn besten Features umfassen hierbei die Merkmale, die laut der Chi-Quadrat-Funktion den höchsten Informationsgehalt liefern. Der angepasste Datensatz wird im nächsten Schritt in eine *feature*- und eine *target*-Komponente getrennt. Im darauffolgenden Schritt werden die Komponenten des Datensatzes in eine Trainings- und eine Testgruppe geteilt. Hierfür bekommt die Testgruppe 30 Prozent der Daten. Somit wird der Random Forest Klassifikator folglich mit 70 Prozent der Daten trainiert. Darauf wird mittels der SMOTE-Methode die Minderheitsgruppe erweitert, sodass es in den Daten gleich viele Einträge für Patienten mit und ohne ein Risiko für eine Herzerkrankung gibt. Diese Methode muss durchgeführt werden, um die Genauigkeit des Modells zu erhöhen, da der Datensatz ein Ungleichgewicht aufweist und der Random Forest mit ausgeglichenen Daten am besten performt.

Im Anschluss wird die Funktion Question() im *main*-Programm ausgeführt. Die Funktion besitzt die Aufgabe in der Python Console, die Informationen zum Gesundheitszustand des Patienten abzufragen. Die jeweiligen Eingaben werden in einem Daten-Frame verknüpft und an das Modell des Random Forest Klassifikators übergeben, welcher eine Vorhersage trifft. Zum Schluss wird mit einer if-Abfrage überprüft, ob der Patient ein Risiko für eine mögliche Herzerkrankung besitzt.

In der Abfrage der Question()-Funktion werden folgende Parameter (Features), die zum Trainieren des Random Forest Algorithmus verwendet wurden, thematisiert:

- Alter
- Geschlecht
- Anzahl täglich gerauchter Zigaretten
- Systolischer Blutdruck
- Diastolischer Blutdruck
- Cholesterinwerte
- Bluthochdruck
- Diabetes
- Glucosewert
- Blutdruckmedikamente

Die Tabelle gibt eine kurze Übersicht über die sechs Klassen, deren Aufgabe und ihre entsprechenden Rückgabeelemente:

Nr.	Klasse	Aufgabe	Rückgabeelement
1	Main	• Erstellt eine Abfrage für den Anwender. • Führt mit den eingegebenen Daten eine Vorhersage durch. • Gibt die Vorhersage aus.	Angabe, ob der Patient ein Risiko für eine mögliche Herzerkrankung besitzt.
2	Daten	• Lädt den Datensatz ein. • Entfernt das Feature „Education“ und alle Einträge mit „na“.	Gibt den Datensatz in einem Daten-Frame an die Klasse Feature-Selection weiter.
3	Feature Selection	• Selektiert die 10 besten Features, entsprechend der Chi-Quadrat-Funktion, aus dem Datensatz. • Erstellt einen neuen Datensatz, der nur die Daten der 10 besten Features besitzt.	Gibt den Datensatz mit den Daten der zehn besten Features weiter.
4	Resampling	• Führt ein Upsampling der Minderheitsgruppe mit der SMOTE-Methode durch.	Gibt die upgesampelten Daten für die Feature- und Target-Variable zurück.
5	TestTrain	• Trennt den Datensatz in die Features und die Target-Variabel. • Teilt des Datensatz in einen Trainings- und Testdatensatz.	Gibt einen Testdatensatz mit 30 % und einen Trainingsdatensatz mit 70 % der Daten zum Trainieren des Random Forest zurück.
6	Random Forest	• Trainiert den Random Forest Klassifikator. • Berechnet die Genauigkeit des Modells.	Die Klasse Random Forest erzeugt einen trainierten Random Forest Klassifikator und wird an die main-Funktion zurückgegeben. Mit dem Modell kann im Anschluss eine Vorhersage getroffen werden.

3.2. Verwendete Methoden zur Erstellung des Random Forest

Für die Erstellung des Programm-Codes und des Random Forest Klassifikators wurde zu einem Großenteil die *sklearn*-Bibliothek in Python verwendet. Mit der *sklearn*-Bibliothek lassen sich effektiv gängige Machine-Learning und KI-Modelle realisieren. Die folgende Tabelle fasst die im Code verwendeten Methoden mit den ausgewählten Eingabeparametern der *sklearn*-Bibliothek zusammen.

Bibliothek	Methode	Beschreibung
sklearn. feature_selection Quelle: [7]	SelectKBest	Wählt k-Features aus dem Datensatz aus, die den höchsten Wert entsprechend der verwendeten Score-Funktion haben. **Eingabe Parameter:** - *Score Function*: Weist den Features entsprechend der Chi-Quadrat Funktion Werte zu. - *k*: Anzahl an Features, die ausgewählt werden sollen.
sklearn. feature_selection Quelle: [7]	SelectKBest.fit	Wendet die Score-Funktion auf den Datensatz an und wählt konform die 10 besten Features aus.
sklearn. model_selection Quelle: [8]	train_test_split	Die Daten werden in einen zufälligen Trainings- und Testdatensatz geteilt. **Eingabe Parameter:** - *array*: Vektor/Datensatz, der in Trainings- und Testdaten geteilt werden soll. - *test_size*: Gibt den Anteil der Daten an, die den Testdatensatz bilden. - *random_state*: Gibt an, wie stark die Daten vor der Teilung durchmischt werden.
Sklearn. ensamble Quelle: [9]	RandomForestClassifier	Der Randam Forest Classifier erstellt einen Klassifikator mit n zufälligen Entscheidungsbäumen. **Übergabe Parameter:** - *n_estimators*: Anzahl der Entscheidungsbäume. - *random_state*: Steuert zum einen die Zufälligkeit, mit der das Bootstrapping für die Bildung der Entscheidungsbäume durchgeführt wird und zum anderen die Stichprobe der Merkmale, die an jedem Entscheidungsknoten für die nächste Teilung des Baumes in Betracht gezogen werden. - *max_depth*: Entspricht der maximalen Tiefe des Entscheidungsbaums. - *max_features*: Gibt die maximale Anzahl an Merkmalen an, die bei der Teilung des Entscheidungsbaums berücksichtigt werden.
Sklearn. ensamble Quelle: [9]	RandomForestClassifier. fit	Trainiert aus den gegebenen Trainingsdaten ein Random Forest Modell. Übergabe Parameter: - *X*: feature , *Y*: target values

3.3. Resample-Methoden

Für die Vorhersage des Risikos für eine Herzerkrankung in den nächsten 10 Jahren gibt der Datensatz zwei Zielvariablen an. Hierbei steht die Klasse 0 für Patienten, die kein Risiko besitzen und die Klasse 1 für Patienten, die ein Risiko aufweisen.

Im Laufe der Entwicklung des Random Forest Modells wurde ersichtlich, dass der gegebene Datensatz ein starkes Ungleichgewicht zwischen den zwei Klassen aufweist. So besitzt der Datensatz für die Klasse 0 insgesamt 3179 Einträge und die Klasse 1 lediglich 572 Einträge. Wird mit diesem Datensatz ein Modell trainiert, kann dies zu einer unpräzisen Vorhersage der Minderheitsgruppe Klasse 1 führen. Das Modell wird folglich hauptsächlich auf die Mehrheitsgruppe Klasse 0 trainiert. Das Ziel des Machine Learnings ist jedoch die Entwicklung zuverlässiger Vorhersagemodelle. Insbesondere vor dem Hintergrund eines Machine Learning Modells, welches zur Vorhersage des Risikos einer Herzerkrankung dienen soll, ist es von Interesse, dass mit dem Modell eine zuverlässige und genaue Vorhersage getroffen werden kann, damit folglich der Patient eine richtige Diagnose erhält.

Da der verwendete Datensatz eine Unausgewogenheit von 5,5:1 besitzt, ist der Datensatz ohne Nachbearbeitung nicht für ein Random Forest Modell geeignet. Denn ohne eine Modifizierung der Daten erhält man mit den oben gewählten Einstellungen der *sklearn*-Methoden nur eine Genauigkeit von 84 %. Grundsätzlich ist dies keine unzureichende Genauigkeit. Werden jedoch die Merkmale Precision, Recall, F1-Score und die Confusion Matrix (siehe Tabelle 1 und Abbildung 2) betrachtet, ist zu erkennen, dass die Genauigkeit in der Vorhersage der Minderheitsgruppe Klasse 1 signifikant schlechter ist als die der Mehrheitsgruppe Klasse 0. Hierbei wurde eine Mehrheit der Klasse 1 als Klasse 0 vorhergesagt, wodurch ein großer Anteil der Vorhersage als *False Negative* klassifiziert wurde. Resultierend würden in diesem Fall einige Patienten eine fehlerhafte Diagnose bzw. keine Diagnose erhalten.

Tabelle 1: Classification Report ohne SMOTE-Resampling.

Class	Precision	Recall	F1-Score
0	0.86	0.96	0.91
1	0.29	0.08	0.13

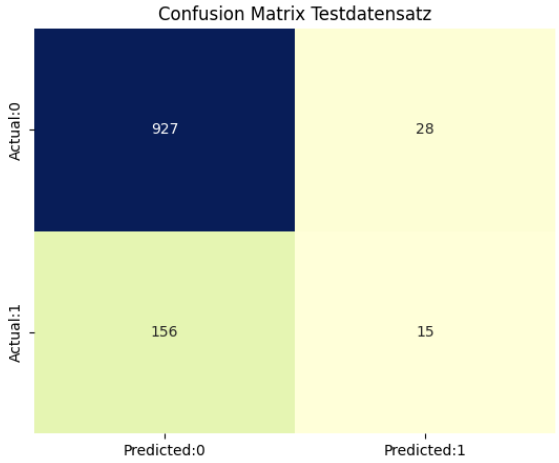

Abbildung 2: Confusion Matrix für das Random Forest Modell ohne SMOTE-Resampling (eigene Darstellung).

In der Regel werden bei unausgewogenen Datensätze sogenannte Resample-Methoden, die entweder die Anzahl der Einträge der Mehrheitsgruppe reduziert oder die Anzahl der Einträge der Minderheitsgruppe erhöht, verwendet. Aufgrund der geringen Anzahl an Daten der

Minderheitsgruppe Klasse 1, bietet sich in diesem Fall keine Undersampling Methode an. Diese würde ansonsten dazu führen, dass der Datensatz nur noch ungefähr 500 Einträge besitzt und somit sehr klein für das Trainieren eines Machine Learning Algorithmus ist. Aufgrund dessen bietet es sich an, eine Upsampling-Methode auf den Datensatz anzuwenden. Hierbei wird die Minderheit an Einträgen an die Mehrheitsgruppe angeglichen. Beispielsweise werden bei einer einfachen Upsampling-Methode zufällig Einträge aus der Minderheitsklasse kopiert und wieder in den Datensatz eingefügt. Dies führt jedoch dazu, dass lediglich die bereits bekannten Einträge für das Trainieren des Modells verwendet werden, dieses somit keine neuen Informationen erhält und das Modell folglich weniger präzise auf fremde Einträge zur Vorhersage reagieren kann. Gleichzeitig steigt die Gefahr, dass es zum Overfitting des Modells kommt [10].

Für die Entwicklung des Modells wurde schließlich die sogenannte *synthetic minority oversampling technique* (SMOTE) verwendet [10]. Bei der SMOTE-Methode werden Beispiele betrachtet, deren Merkmale nah aneinander liegen. Zwischen die ausgewählten Beispiele wird schließlich eine Linie gelegt, entlang dieser Linie neue Merkmalwerte in einem Muster gezogen und ihnen ein entsprechendes Klassenergebnis zugewiesen. Somit werden Einträge aus dem ursprünglichen Datensatz nicht nur einfach kopiert, sondern neue aus den bestehenden Einträgen gebildet. Die Methode führt demnach eine Anpassung der Minderheitsgruppe an die Mehrheitsgruppe durch, ohne dass Einträge verdoppelt werden und dadurch die Gefahr des Overfitting entsteht [11].

Resultierend wird mit dieser Upsampling-Methode eine Genauigkeit von 95 % erreicht und ist demnach deutlich präziser als die Genauigkeit des Modells ohne Anpassung des Datensatzes. Des Weiteren ist zu erkennen, dass die Werte der Minderheitsgruppe Klasse 1 für die Precision, den Recall und den F1-Score deutlich gestiegen sind (siehe Tabelle 2). Beispielsweise hat sich der Recall-Wert für die Klasse 1 ungefähr verzehnfacht und auch die Precision und der F1-Score nähern sich der 100 % an. In der Confusion Matrix (siehe Abbildung 3) ist abzulesen, dass die Anzahl der *False Negative* Ergebnisse deutlich gesunken ist. Jedoch ist die Anzahl der *False Positive* Ergebnisse leicht angestiegen. Nichtsdestotrotz wird die Präzession des Random Forest Klassifikator durch die SMOTE-Methode deutlich verbessert.

Tabelle 2: Classification Report mit SMOTE-Resampling.

Class	Precision	Recall	F1-Score
0	0.98	0.96	0.97
1	0.77	0.88	0.82

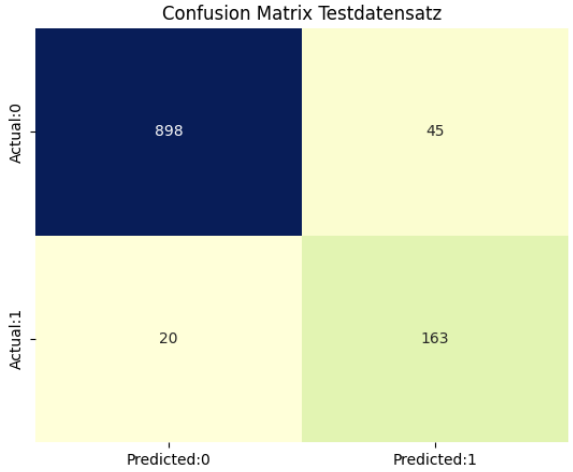

Abbildung 3: Confusion Matrix für das Random Forest Modell mit SMOTE-Resampling (eigene Darstellung).

3.4. Unittest

In der Softwareentwicklung spielt das Thema Testen von Software eine sehr große Rolle. Werden Fehler bereits am Anfang des Prozesses entdeckt, können hohe Fehlerkosten vermieden werden. Aufgrund dessen empfiehlt es sich bereits während der Entwicklungsphase der Software vereinzelte Komponententest, auch Unittest genannt, einzubauen. Bei einem Unittest werden im Code einzelne Tests eingebaut, die kleine Bausteine im Code, exemplarische Funktionen und Klassen, testen [12].

In Zusammenhang mit dem implementierten Code wurde ein Unittest zum Testen der Resample Methode SMOTE geschrieben. Die Implementierung eines Unittest in PyCharm wird direkt über den Tasten-Kürzel Alt+Einfg und der Auswahl, dass ein Test eingefügt werden soll, erstellt. Die offizielle Unittest-Dokumentation von Python liefert einige Asset-Methoden mit denen beispielsweise verglichen werden kann, ob zwei Variablen identisch sind.

Für die SMOTE-Methode wurde der Unittest so geschrieben, dass er zunächst überprüft, ob nach dem Resampling die Anzahl der Einträge der Klasse 1 ungleich zu der Anzahl der Einträge vor dem Resampling ist. Im Anschluss wird getestet, ob die Anzahl der Einträge der Klasse 1 nach der Anwendung der Resampling-Methode gleich der Anzahl der Einträge der Mehrheitsgruppe Klasse 0 ist. Läuft der Unittest ohne Fehlercode durch, ist die Funktionsweise der SMOTE-Methode korrekt.

```
1   import ...
8
9
10  class TestResample(TestCase):
11      def test_upsampling_smote(self):
12          # Zugriff auf die Daten
13          data = Daten()
14          data = data.data
15          new_data = FeatureSelection()
16          new_data = new_data.new_data_frame(data)
17
18          k = TestTrain()
19          [X, y] = k.define_target_features(new_data)
20          # Anzahl der Elemente ungleich 0 ohne Resampling
21          self.n_y_1 = np.count_nonzero(y)
22
23          r = Resample()
24          #[X, y] = r.upsampling_smote(X, y)
25          # Anzahl der Elemente ungleich 0 mit SMOTE-Resampling
26          self.n_y_2 = np.count_nonzero(y)
27          # Anzahl der Elemente gleich 0 mit SMOTE-Resampling
28          self.n_y_3 = len(y) - self.n_y_2
29
30          # Anzahl der Elemente ungleich 0 vor dem Resampling sind ungleich der Elemente
31          # ungleich 0 nach dem Resampling.
32          self.assertNotEqual(self.n_y_1, self.n_y_2, 'Die Anzahl der Elemente ungleich Null'
33                                                      ' haben sich nach dem SMOTE-Resampling '
34                                                      'nicht verändert.')
35          # Anzahl der Elemente ungleich 0 nach dem Resampling sind gleich der Anzahl der Elemente
36          # gleich 0 nach dem Resampling. Das SMOTE-Resampling hat funktioniert.
37          self.assertEqual(self.n_y_2, self.n_y_3, 'Die Anzahl der Elemente der Klasse "1" ist'
38                                                   ' nicht gleich der Elemente der Klasse "0"')
```

Abbildung 4: Unittest der SMOTE-Methode (eigene Darstellung).

4. Scrum – Sprint-Backlog

KW	SPRINT-BACKLOG	DONE
40 - 44	**Themenfindung** 1. Recherche nach möglichen Projekten für das angewandte Programmierprojekt. 2. Suchen nach passenden Datensätzen für ein Machine Learning Projekt. 3. Vorbereitung einer Präsentation zur Vorstellung des Projekts.	☒ 1. Recherche ☒ 2. Auswahl eines Datensatzes ☒ 3. Erstellung einer Präsentation
45 - 47	**Einarbeitung in Python / OOP / Scrum** 1. Installation von PyCharm Professional. 2. Einarbeitung in die Programmierspreche Python. 3. Einarbeitung und erlernen der objektorientierten Programmiersprache. 4. Scrum Workshop 5. Schreiben eines eigenen Sprint-Plans	☒ 1. Installation PyCharm ☒ 2. Einarbeitung Python ☒ 3. Einarbeitung OOP ☒ 4. Scrum Workshop ☒ 5. Persönlicher Sprint-Plan
48 - 51	**Programmierung** 1. Schreiben von Objektklassen zur Erstellung eines Random Forests 2. Fehlerbehebung	☒ 1. Schreiben der Objektklassen ☒ 2. Fehlerbehebung
52	Winterpause	Winterpause
1 - 3	**Validierung** 1. Erprobung diverser Resample-Methoden zur Erhöhung der Genauigkeit bei einem unausgewogenen Datensatzes. 2. Erstellung von 1-2 Unit-Tests	☒ 1. Wahl einer Resample-Methode ☒ 2. Erstellung eines Unit-Tests
4 - 6	**Dokumentation** 1. Schriftliche Ausarbeitung 2. Erstellung einer Präsentation 3. Code fertigstellen, auskommentieren und auf Git-Hub hochladen.	☒ 1. Schriftliche Ausarbeitung ☒ 2. Erstellung einer Präsentation ☒ 3. Code Fertigstellung

Hinweis: In der Regel sollten Sprints immer über einen identisch langen Zeitplan geplant und durchgeführt werden. Aufgrund der Gegebenheiten war dies jedoch nicht möglich, wodurch die Sprints zwischen 2-4 Wochen in Anspruch genommen haben.

5. Literaturverzeichnis

[1] Robert-Koch-Institut (o. D.): Herz-Kreislauf-Erkrankungen, RKI - Themenschwerpunkt Herz-Kreislauf-Erkrankungen, [online] https://www.rki.de/DE/Content/Gesundheitsmonitoring/Themen/Chronische_Erkrankungen/HKK/HKK_node.html [abgerufen am 27.10.2022].

[2] Wikipedia-Autoren (2004): Framingham-Herz-Studie, [online] https://de.wikipedia.org/wiki/Framingham-Herz-Studie [abgerufen am 27.10.2022].

[3] Litzel, Nico/Stefan Luber (2020): Was ist Random Forest?, BigData-Insider, [online] https://www.bigdata-insider.de/was-ist-random-forest-a-913937/ [abgerufen am 01.02.2023].

[4] Was ist ein Random Forest? (2022): Data Basecamp, [online] https://databasecamp.de/ki/random-forest [abgerufen am 01.02.2023].

[5] Choudhary, Divya (2022): Bootstrapping and OOB samples in Random Forests - Analytics Vidhya, Medium, [online] https://medium.com/analytics-vidhya/bootstrapping-and-oob-samples-in-random-forests-6e083b6bc341 [abgerufen am 01.02.2023].

[6] Wikipedia-Autoren (2009): Random Forest, [online] https://de.wikipedia.org/wiki/Random_Forest [abgerufen am 01.02.2023].

[7] Sklearn.feature_selection.SelectKBest (o. D.): scikit-learn, [online] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html [abgerufen am 03.02.2023].

[8] Sklearn.model_selection.train_test_split (o. D.): scikit-learn, [online] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html [abgerufen am 03.02.2023].

[9] Sklearn.ensemble.RandomForestClassifier (o. D.): scikit-learn, [online] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html [abgerufen am 03.02.2023].

[10] Brownlee, Jason (2021): SMOTE for Imbalanced Classification with Python, Machine Learning Mastery, [online] https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/ [abgerufen am 03.02.2023].

[11] Huh, Kyoun (2021): Surviving in a Random Forest with Imbalanced Datasets, Medium, [online] https://medium.com/sfu-cspmp/surviving-in-a-random-forest-with-imbalanced-datasets-b98b963d52eb [abgerufen am 03.02.2023].

[12] Redaktion ComputerWeekly.de (2020): Unit-Test, ComputerWeekly.de, [online] https://www.computerweekly.com/de/definition/Unit-Test [abgerufen am 03.02.2023].