Bericht zur

„Praxisphase“

vom 01.10.2020 bis 30.04.2021

Praxisphasen-Firma:
Commerzbank AG Frankfurt,
Konzernbereich: Big Data & Advanced Analytics (BDAA)
Neue Mainzer Str. 66
60311 Frankfurt

Betreuer:
Dr. Alexander Fischer
BDAA Cost Analytics

Student:
Rainer Gogel
Matrikel-Nummer: 1272442

B.A. Informatik
University of Applied Sciences, Frankfurt
Sommersemester 2021

Mittwoch, 6. September 2023

Inhaltsverzeichnis

1. Executive Summary

Dies ist die Dokumentation zu meiner Praxisphase im Rahmen meines Bachelor-Studiums der Informatik an der University of Applied Sciences in Frankfurt. Meine Praxisphase habe ich vom 01.Oktober 2020 bis zum 30.April 2021 in einer Abteilung der Commerzbank AG absolviert, die mit Hilfe der künstlichen Intelligenz versucht Kreditbetrugsfälle zu erkennen und zu vermeiden. Meine Aufgabe bestand darin mich in das Themengebiet „Maschinelles Lernen" einzuarbeiten und dabei Python-Machine-Learning-Programme zu schreiben. Im Folgenden sollen meine Tätigkeiten während dieser Praxisphase beschrieben werden.

2. Einleitung

Von März bis April 2020 hatte ich die Möglichkeit ein Praktikum bei der Commerzbank AG im Konzernbereich „Big Data & Advanced Analytics" zu absolvieren. Das anschließende Angebot einer Werkstudententätigkeit gab mir zudem auch die Möglichkeit mein Praxissemester dort zu verbringen. Da ich bereits während meines Studiums der Volkswirtschaftslehre und auch während meiner Berufstätigkeit als Vermögensverwalter mit Datenanalysen zu tun hatte und mich für dieses Gebiet auch weiterhin sehr interessiere, nutzte ich die Gelegenheit und wechselte für das Praxissemester in die Abteilung „Predictive Analytics", die sich auch mit Methoden der Datenwissenschaft („Data Science") beschäftigt.

Die Abteilung nutzt für ihre Datenanalysen und das maschinelle Lernen („Machine Learning" oder „ML") bisher hauptsächlich die Programmiersprachen R und SAS. Da langfristig angedacht ist auf die Programmiersprache Python und Python-Bibliotheken umzustellen, war es meine Aufgabe mich während der Praxisphase in das Thema ML einzuarbeiten und dabei eine Python-Code-Basis zu schaffen, die später mitunter für die Umstellung auf Python verwendet werden kann.

Ein Anwendungsfall dieser Abteilung ist die Datenanalyse und -modellierung zur Vermeidung von Kreditbetrugsfällen. Über den genauen Analyseprozess oder die dabei verwendeten ML-Algorithmen und Modellvariablen, kann ich in diesem Bericht aber keine konkreten Angaben machen, da Bankdaten im Allgemeinen und Betrugsdaten im Besonderen sowie der Untersuchungsprozess der Commerzbank AG streng vertraulich bleiben müssen und ich diesen auch nur am Rande kennengelernt habe.

In diesem Bericht beschreibe ich daher an Beispielen, was ich über das maschinelle Lernen gelernt und wie ich dies mit Hilfe von Python und Python-Bibliotheken in Programmcode umgesetzt habe. Da dies auch die gesetzten Aufgabenschwerpunkte in meinem Praxissemester waren, entspricht die Beschreibung des von mir Gelernten auch tatsächlich einem Tätigkeitsbericht.

Einige Betrugsdaten wurden mir in CSV-Format zur Verfügung gestellt, da der ursprüngliche Plan diese direkt aus dem Hadoop-Umfeld[1] zu extrahieren aufgrund von Systemumstellungen zu der entsprechenden Zeit nicht umsetzbar war. Für die

[1] https://hadoop.apache.org/

Einarbeitung und die Programmierung habe ich aber hauptsächlich öffentlich zugängliche Beispieldatensätze verwendet. Die in Python geschriebenen Programme wurden dann anhand der mir zur Verfügung gestellten (mitunter sehr großen) Betrugsdatensätze getestet und daraus auch weitere Analyseergebnisse produziert.

Für die Codierung verwendete ich „Intellij Idea"[2] als Entwicklungsumgebung („IDE"), zum Testen der Programme „Jupyter Notebooks"[3] und für das maschinelle Lernen die Python Bibliothek „Scikit-Learn"[4].

3. Beschreibung des Unternehmens

A. Commerzbank AG

Die Commerzbank AG ist, nach Bilanzsumme gerechnet, hinter der Deutschen Bank, der DZ-Bank und der KFW-Bank die viertgrößte Bank in Deutschland.[5] Sie hat hier rund 30.000 Firmenkunden sowie rund 11 Millionen Privat- und Unternehmerkunden. Die Commerzbank wickelt rund 30 Prozent des deutschen Außenhandels ab und ist im Firmenkundengeschäft international in knapp 40 Ländern vertreten.[6]

B. Konzernbereich BDAA

Die Organisation der Commerzbank unterteilt sich in fünf Unternehmenssparten. Einer dieser Sparten ist das „Risk Management", zu dem auch der Konzernbereich „Big Data & Advanced Analytics" („BDAA") gehört, der wiederum in weitere sieben sogenannte „Cluster" unterteilt ist. Zum Cluster „Cost Analytics" gehört auch die Abteilung „Predictive Analytics". Die BDAA wurde 2017 aufgebaut und beschäftigt mittlerweile rund 450 Mitarbeiter in fünf verschiedenen Ländern.
Die Anwendungsfälle, für die der Konzernbereich Dienstleistungen anbietet, reichen von der Bereitstellung älterer Kontoauszüge über die automatisierte Bonitätseinstufung von Kunden bis hin zur Entwicklung von Software-Applikationen.

Eines der Kerngeschäfte der Commerzbank AG ist die Kreditvergabe an Firmenkunden. Dort kommt es in sehr seltenen Fällen zum Kreditbetrug. Ein Kreditbetrug definiert sich als Erschleichung von Krediten durch die Angabe falscher Informationen im Kreditantrag durch den Antragsteller und zielt darauf ab, die erhaltenen Geldmittel nicht oder nur unvollständig zurückzuzahlen. Dadurch entsteht der Commerzbank nicht nur ein wirtschaftlicher Schaden, sondern zieht auch zeitaufwendige Untersuchungsprozesse nach sich. Einer der Anwendungsfälle des Konzernbereichs BDAA umfasst daher auch die Analyse von Firmenkundendaten mit Hilfe des ML. Die Modellierung ermöglicht dabei zunächst eine Vorauswahl von Verdachtsfällen. Die eigentliche forensische Analyse und Entscheidung über das Vorliegen eines Kreditbetrugs erfolgt dann durch speziell ausgebildete Investigatoren, ggfs. unter Einbeziehung weiterer Einheiten der Bank.

In der Abteilung „Predictive Analytics" des BDAA-Clusters „Cost Analytics" ist Dr. Alexander Fischer als Mathematiker und Datenwissenschaftler für die Entwicklung und Pflege solcher Modelle mitverantwortlich. Dort habe ich mein Praxissemester absolviert und wurde von ihm auch fachlich betreut.

[2] https://www.jetbrains.com/de-de/idea/
[3] https://jupyter.org/
[4] https://scikit-learn.org/stable/index.html
[5] (Wikipedia: Banken in Deutschland, 2021)
[6] (Commerzbank.de, 2021)

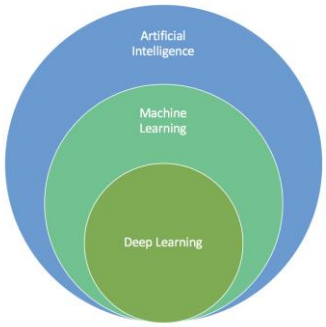
4. Beschreibung des Fachgebiets und der Aufgaben

Da ich vor der Praxisphase weder in der Programmiersprache Python noch auf dem Gebiet des ML Erfahrung hatte, habe ich mich zunächst einmal in die Programmiersprache Python und das Themengebiet ML eingearbeitet und dazu auch verschiedene Bücher [7, 8, 9, 10, 11, 12, 13, 14, 15] und Onlinequellen [16, 17, 18] herangezogen.

A. Maschinelles Lernen

Maschinelles Lernen ist dem Bereich der „Künstlichen Intelligenz" („KI") zuzuordnen und steht für den Versuch aus Daten Wissen zu generieren oder zu extrahieren:

Abbildung 1: KI, Machine Learning und Deep Learning



Dabei werden mit Hilfe von Algorithmen statistische Modelle entwickelt und angewendet, um Muster und Gesetzmäßigkeiten aus Beispieldaten zu erkennen oder „zu lernen". Die aus den Daten gewonnenen Erkenntnisse lassen sich nach der Lernphase verallgemeinern und für neue Problemlösungen oder für die Analyse von bisher unbekannten Daten verwenden.[19]

Beim ML wird zwischen überwachtem („Supervised Learning"), nicht-überwachtem („Unsupervised Learning") und halb-überwachtem („Semi-Supervised Learning") Lernen unterschieden.

I. Supervised Learning

Beim überwachten Lernen besteht das Ziel darin, mit Hilfe statistischer Methoden eine Abbildungsfunktion zu entwickeln, die das unbekannte Verhältnis von bekannten Eingabedaten (X-Vektoren oder X-Matrix unabhängiger Variablen) zu bekannten Ausgabedaten (Y-Vektor oder abhängige Variable) am besten beschreibt. Die Abbildungsfunktion kann dann auf neue bekannte Eingabedaten angewendet werden, um damit Ausgabedaten zu prognostizieren.
Beim Supervised Learning unterscheidet man dabei zwei Arten von Problemstellungen: Die Klassifizierung und die Regression.
Falls die Ausgabedaten kategorischer Natur sind, liegt ein Klassifizierungsproblem vor. Falls die Ausgabedaten reale, oft stetige Zahlenwerte sind, handelt es sich um ein Regressionsproblem.

[7] (Brownlee, 2020)
[8] (Burkov, 2019)
[9] (Hastie, et al., 2009)
[10] (McKinney, 2018)
[11] (VanderPlas, 2017)
[12] (Géron, 2019)
[13] (Albon, 2018)
[14] (Barry, 2017)
[15] (Rossum & team, 2018)
[16] (Brownlee, 2021)
[17] (Ng, 2020)
[18] (Scikit-Learn, 2021)
[19] (www.bigdata-insider.de, 2021)

II. Unsupervised Learning

Im Gegensatz zum überwachten Lernen erfordert das nicht-überwachte Lernen keine Ausgabedaten (kein „Label"), sondern nur Eingabedaten/unabhängige Variablen. Ziel dabei ist es, die informativste Struktur zu präsentieren, die die Eingabedaten am besten beschreibt. Das nicht-überwachte Lernen unterteilt sich in verschiedene Unterarten:

Beim „Clustering" wird versucht Gruppierungen in den Eingabedaten zu erkennen. Dabei soll die Frage beantwortet werden, welche unabhängigen Variablen in welchem Ausmaß die Abgrenzung einzelner Gruppen bestimmen.

Bei der „Anomaly Detection" soll die Frage beantwortet werden, welche Eingabedaten so weit außerhalb des Bereichs der restlichen Eingabedaten liegen, daß man diese als „untypisch" oder „abnormal" bezeichnen muss. Ein häufiger Anwendungsfall dafür ist die Betrugserkennung.

Daneben bestehen noch weitere nicht-überwachte Lernmethoden, die zum Ziel haben, die Dimension hochdimensionaler Datensätze (=Datensätze mit sehr vielen unabhängigen Variablen) zu reduzieren, um damit auch Visualisierungen in einem niedrig-dimensionalen Raum zu ermöglichen, die für das menschliche Auge begreifbar sind.

III. Semi-Supervised Learning

Beim halbüberwachten Lernen handelt es sich, wie der Name bereits preisgibt, um eine Mischung der beiden oben besprochenen Methoden. Dabei werden die beiden Methoden in einem Zwei-Schritt-Verfahren kombiniert.
Zunächst wird, wie beim nicht-überwachten Lernen versucht, für Eingabedaten, zu denen keine (oder nur vereinzelt) korrespondierende Ausgabedaten vorliegen, Gruppierungen zu finden. Anhand der Gruppenzugehörigkeit können für alle Eingabedaten nun die korrespondierenden Ausgabedaten („Label"), etwa mit Durchschnittsverfahren, prognostiziert werden. Im nächsten Schritt, der dem überwachten Lernen entspricht, wird versucht eine stabile Abbildungsfunktion zu finden, die das Verhältnis zwischen Ein- und Ausgabedaten am besten beschreibt.

Ein Anwendungsfall für das halbüberwachte Lernen ergibt sich, wenn für Eingabedaten nur vereinzelt Ausgabedaten („Labels") vorliegen, man aber unbedingt die gesamten Daten etwa für ein Klassifikationsproblem verwenden möchte. Um alle Eingabedaten mit entsprechenden Ausgabedaten „aufzufüllen" oder „zu ergänzen", wird zunächst über die Cluster-Methode den Eingabedaten entsprechende Ausgabedaten „zugewiesen". Im Anschluss daran wird nun mit Methoden des überwachten Lernens, ein Abbildungsverhältnis von Eingabedaten und (nun vollständigen) Ausgabedaten berechnet.

IV. Deep Learning

Deep Learning ist ein Teil des ML, das die Funktionsweise menschlicher Gehirn-Nervenzellen („Neuronen") bei der Verarbeitung von Daten nachahmt. Das Modell verfügt über sogenannte Eingangs- und Ausgangsneuronen, die sich über Zwischenneuronen miteinander verknüpfen. Das Adjektiv "tief" bezieht sich hierbei auf die Verbindung mehrerer Schichten von Neuronen, die als neuronale Netze bezeichnet werden.

Da die verwendeten Algorithmen bei der Betrugsprävention primär dem klassischen ML und dort überwiegend dem Bereich des überwachten Lernens zugeordnet werden können, habe ich mich während meiner Praxisphase auch nur mit dieser Art des ML beschäftigt. Die Methoden des nicht-überwachten Lernens und neuronale Netze sind daher nicht Teil der Darstellung in den folgenden Kapiteln.

B. Einarbeitung in das Thema

I. Übungsdaten

Zum Einstieg in die Datenwissenschaft bieten verschiedene Plattformen[20, 21] auch Übungsdaten an. Zwei der bekanntesten Übungsdatensätze sind der Schwertlilien-Datensatz („Iris")[22] aus dem Bereich der Botanik und der Wisonsin-Brustkrebs-Datensatz („Wisconsin Breast Cancer" oder „WBC")[23] aus dem Bereich der Medizin.
Beim Iris-Datensatz werden drei verschiedene Arten von Schwertlilien („Setosa", „Versicolor" und „Virginica") durch vier Eigenschafts- oder Variablen-Vektoren charakterisiert: Kelchblattlänge und -Breite („Sepal Length" und „Sepal Width") sowie Blütenblattlänge und -Breite („Petal Length" und „Petal Width").
Beim WBC-Datensatz werden gutartige („benign") und bösartige („malign") Brusttumore und Größenwerte für die dabei vorgefundenen neun physiologischen Eigenschaften angegeben: Klumpendicke („ClumpThickness"), Gleichmäßigkeit der Zellgröße („Uniformity Of Cell Size"), Gleichmäßigkeit der Zellform („Uniformity Of Cell Shape"), Randadhäsion („MarginalAdhesion"), Größe einzelner Epithelzellen („Single Epithelial Cell Size"), Nackte Kerne („Bare Nuclei"), Mildes Chromatin („Bland Chromatin"), Normale Nukleolen („Normal Nucleoli") und Mitose („Mitoses").
Die unabhängigen Variablen-Vektoren, die später für das ML verwendet werden, werden nach der Datenbereinigung und Auswahlphase (siehe unten) auch Feature-Vektoren genannt.

Ziel für einen Datenwissenschaftler ist es nun aus den Übungsdaten ein Muster oder eine Gesetzmäßigkeit zu erkennen, die zur Klassifizierung noch unbekannter Daten herangezogen werden kann.
Konkret soll beim Iris-Datensatz die Frage beantwortet werden, um welche Art („abhängige Variable") es sich bei einer noch nicht klassifizierten Schwertlilie wohl handeln dürfte, wenn diese bestimmte Blattlängen und -breiten („unabhängige Variablen") aufweist.
Beim WBC-Datensatz geht es konkret um die Frage, ob der noch nicht näher diagnostizierte Tumor wohl gut- oder bösartig ist („abhängige Variable"), wenn für die neun oben beschriebenen physiologischen Eigenschaften („unabhängige Variablen") bestimmte Werte gemessen wurden.
Dabei kann keine sichere Prognose abgegeben werden, sondern nur eine auf Basis der bisherigen Daten beruhende Wahrscheinlichkeit, dass die zu klassifizierende Variable zu einer bestimmten Klasse gehört. Die Treffsicherheit der Prognose hängt dabei sowohl von der Stabilität und Qualität der im ML verwendeten Modelle ab, als auch von der Stabilität und Stärke der Kovarianz von abhängigen und unabhängigen Variablen.
Anhand dieser beiden Datensätze habe ich begonnen, die für das ML typischen und notwendigen Schritte zu durchlaufen und mich in die Thematik einzuarbeiten.

[20] (Kaggle, 2021)
[21] (Irvine, 2021)
[22] (Irvine, 1995)
[23] (Irvine, 1991)

II. Extract-Transform-Load (ETL)

Die ML-Modellierung beginnt mit der Datenaufbereitung. Doch zuvor sind drei Prozessschritte notwendig, die im englischsprachigen Raum oft auch mit den Anfangsbuchstaben „ETL" zum Ausdruck gebracht werden:

a. Extract:
   Mitunter müssen Daten aus verschiedenen Datenquellen extrahiert und aggregiert werden. Dies trifft für die Daten in der Kreditbetrugsmodellierung zu, da hier beispielweise sowohl Konzerndaten als auch Einzelunternehmensdaten eingehen, die sich in unterschiedlichen Datenbanken befinden. Für die Beispieldatensätze hier ist aber keine besondere Datenextraktion notwendig, da diese aggregiert aus den bekannten Online-Quellen[24] in CSV-Format einfach heruntergeladen werden können.
b. Transform:
   Bei den Datenarten unterscheidet man zwischen strukturierten und unstrukturierten Daten. Strukturierte Daten passen üblicherweise in ein Tabellenformat (wie zum Beispiel Excel, CSV, etc.) und sind dadurch gekennzeichnet, dass für jeden Datenpunkt („Zeile") einzigartige Attribute in Form von numerischen Zahlenwerten in „Spalten" ausgewiesen werden. Im Gegensatz dazu haben unstrukturierte Daten keine identifizierbare Tabellenstruktur und bestehen in der Regel aus Bildern, Objekten, Text, Tonaufnahmen, Emails und anderen Datentypen. Letztere müssen in eine strukturierte Form überführt werden, die von den ML-Algorithmen verarbeitet werden können. Dies trifft mitunter auch für die Daten in der Kreditbetrugsmodellierung zu, aber nicht für die Beispielsdatensätze, da diese bereits in strukturierter Tabellenform als Textdatei („CSV-Format") vorliegen. Auf die Transformation von unstrukturierten Daten in strukturierte Daten soll daher hier auch nicht weiter eingegangen werden.
c. Load:
   Dieser Prozess umfasst die Speicherung der (umgewandelten) strukturierten Daten in einer Datenbank, aus der diese für den weiteren ML-Prozess dann mit Hilfe von Datenbankabfragen (wie beispielsweise SQL-Befehlen) abgerufen und in den Arbeitsspeicher geladen werden können. Dieser Prozess wird ebenfalls in der Kreditbetrugsmodellierung angewandt, aber nicht bei den Übungsdatensätzen im CSV-Format, da diese lokal auf einem Laufwerk abgelegt werden.
d. Python und Python-Bibliotheken:
   Um die Übungsdatensätze in den Arbeitsspeicher zu laden, müssen diese je nach Programmiersprache und ML-Programm in einer bestimmten Form vorliegen. Die Python ML-Bibliothek Scikit-Learn („Sklearn") akzeptiert verschiedene Datentypen wie beispielsweise Numpy[25]-Arrays, Scipy[26]-sparse-Matrizen oder Pandas[27] Dataframes. Pandas wurde für Matrizen-Datenstrukturen optimiert und liefert verschiedene Funktionen zur Datenverarbeitung und -analyse an.
   So kann mit der Pandas-Funktion „read_csv" beispielsweise der WBC-Datensatz in CSV-Format gelesen und als Pandas Dataframe in den Arbeitsspeicher geladen werden:

[24] (Irvine, 2021)
[25] (Numpy, 2021)
[26] (Scipy, 2021)
[27] (Pandas, 2021)

Abbildung 2: Pandas read_csv()

```
# Import Library
import pandas as pd

# Load data from CSV
wiscons = pd.read_csv('../../UebungsDaten/WisconsBreastCancer.csv')
```

Sklearn selbst stellt aber auch bestimmte Datensätze zur Verfügung, ohne daß dazu erst eine CSV-Datei ausgelesen werden muss. So kann der Iris-Datensatz mit der Funktion „load_iris" direkt aus der Sklearn-Bibliothek in ein Numpy-Array geladen werden:

Abbildung 3: Sklearns load_iris()

```
from sklearn import datasets

# Get data
iris = datasets.load_iris()
```

III. Datenaufbereitung – Teil 1

Rohdaten können typischerweise nicht sofort im ML verwendet, sondern müssen zuerst aufbereitet werden. Einer Anaconda-Studie[28] zufolge verwenden Datenwissenschaftler rund 45% ihrer Zeit für die Datenaufbereitung (inklusive der ETL-Phase). Die Gründe für die Aufbereitungsnotwendigkeit können vielfältiger Natur sein:

- ML-Algorithmen erfordern numerische Datentypen, also Zahlen
- Bestimmte ML-Algorithmen stellen bestimmte Anforderungen an die Zahlen
- Statistisches Rauschen und Fehler müssen mitunter zuerst korrigiert werden
- Komplexe nichtlineare Zusammenhänge müssen mitunter im Vorfeld identifiziert werden

Die Datenaufbereitung ist auch deshalb so wichtig, weil qualitativ schlechte Eingangsdaten auch qualitativ schlechte Modell-Ergebnisse produzieren (Prinzip: „Garbage in – Garbage Out"). In den folgenden Abschnitten soll auf diese Punkte näher eingegangen werden.

Bei der Datenaufbereitung geht es darum, systematische Probleme und Fehler im Datensatz zu beheben. Die Gründe für fehlerhafte Daten können vielfältig sein: Tippfehler bei der Datenerhebung, Beschädigung der Daten bei der Übertragung oder beim Speichern, irrtümliche Duplizierung, nicht fachgerechte Handhabung, usw. Fachliche Expertise ist für die Fehlererkennung und -bereinigung hilfreich, aber nicht immer zwingend notwendig.
Einen schnellen Blick auf die WBC-Daten erhält man im Jupyter-Notebook durch den Pandas-Befehl „info":

[28] (Anaconda, 2020)

Abbildung 4: Pandas DataFrame info()

```
# Get general description and data types
wiscons.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   SampleCodeNumber             699 non-null    int64
 1   ClumpThickness               699 non-null    int64
 2   UniformityOfCellSize         699 non-null    int64
 3   UniformityOfCellShape        699 non-null    int64
 4   MarginalAdhesion             699 non-null    int64
 5   SingleEpithelialCellSize     699 non-null    int64
 6   BareNuclei                   699 non-null    object
 7   BlandChromatin               699 non-null    int64
 8   NormalNucleoli               699 non-null    int64
 9   Mitoses                      699 non-null    int64
 10  Class                        699 non-null    object
dtypes: int64(9), object(2)
```

Aus Abb. 4 kann man erkennen, dass die unabhängige Variable „BareNuclei" und die Target-Variable „Class" keine numerischen Datentypen zu sein scheinen, sondern als „object" ausgewiesen werden. Bei näherer Betrachtung stellt man fest, dass die Variable „BareNuclei" nur String-Datentypen enthält:

Abbildung 5: BareNuclei-Datentypen

```
set([type(val) for val in wiscons['BareNuclei']])
{str}
```

Diese Strings enthalten 683 Zahlenwerte und 16 Nichtzahlenwerte (siehe Abb.6). Die String-Zahlenwerte müssen durch wirkliche Zahlenwerte ersetzt werden. Die Nichtzahlenwerte bestehen aus Fragezeichen, die korrigiert und ebenfalls ersetzt werden müssen, da die ML-Algorithmen nur Zahlenwerte akzeptieren:

Abbildung 6: BareNuclei-Numerics und Non-Numerics

```
# Check if the strings are numeric and whether there are any non-numeric strings
non_numerics_in_bn = [val for val in wiscons['BareNuclei'] if not val.isnumeric()]
numerics_in_bn = [val for val in wiscons['BareNuclei'] if val.isnumeric()]
print("Number of non-numerics in 'BareNuclei':", len(non_numerics_in_bn))
print("Number of numerics in 'BareNuclei':", len(numerics_in_bn))
Number of non-numerics in 'BareNuclei': 16
Number of numerics in 'BareNuclei': 683

# What are the non-numerics in 'BareNuclei'
print(non_numerics_in_bn)
['?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?']
```

Das Ersetzen der String-Zahlenwerte durch wirkliche Zahlenwerte geschieht mit der Pandas-Funktion „to_numeric". Dabei werden auch die Fragezeichen durch Numpy-Nullwerte (Numpy NANs) als „Stellvertreter" ersetzt, damit wir später leichter eine Ersetzen-Strategie für diese Fragezeichen/Stellvertreter umsetzen können. Nun könnte man alle Zeilen, die Fragezeichen enthalten, einfach löschen. Dadurch würden aber auch die Informationen aus den anderen Variablen verloren gehen. Besser ist es daher, die Fragezeichen/Stellvertreter durch sinnvolle Werte zu ersetzen. Warum wir das Ersetzen der Fragezeichen nicht bereits an dieser Stelle diskutieren und durchführen, erläutere ich im Abschnitt „Datenaufbereitung – Teil 2". Nach Ausführung der Funktion können wir sehen, dass nun alle Werte der Variable „BareNuclei" vom Datentyp „float64" sind:

Abbildung 7: BareNuclei nach to_numeric()

```
wiscons['BareNuclei'] = pd.to_numeric(wiscons['BareNuclei'], errors='coerce',
downcast='integer')
wiscons.info()
```

```
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   SampleCodeNumber     699 non-null    int64
 1   ClumpThickness       699 non-null    int64
 2   UniformityOfCellSize 699 non-null    int64
 3   UniformityOfCellShape 699 non-null   int64
 4   MarginalAdhesion     699 non-null    int64
 5   SingleEpithelialCellSize 699 non-null int64
 6   BareNuclei           683 non-null    float64
 7   BlandChromatin       699 non-null    int64
 8   NormalNucleoli       699 non-null    int64
 9   Mitoses              699 non-null    int64
 10  Class                699 non-null    object
dtypes: float64(1), int64(9), object(1)
```

Im nächsten Schritt überprüfen wir, ob es Duplikate gibt, die es theoretisch nicht geben sollte, da jede Patientin und ihr zugeordneter Schlüssel („Sample Code Number") nur jeweils einmal vorkommen müsste:

Abbildung 8: Duplikate

```
duplicates = wiscons.duplicated(keep=False)
wiscons[duplicates]
```

	SampleCodeNumber
684	466906
683	466906
561	1321942
560	1321942
338	704097
314	704097
272	320675
267	320675
258	1198641
254	1116116
253	1100524
208	1218860
207	1218860
168	1198641
62	1116116
42	1100524

Aus Abb.8 ist zu erkennen, dass es in der Tat acht Duplikate gibt, bei denen alle Zeilenwerte identisch sind. Allerdings ist über die Pandas-Funktion „nunique" auch zu sehen, dass es nur 645 eindeutige Schlüssel („Sample Code Number") gibt. Es existieren also 54 Datenpunkte, bei denen ein Schlüssel mindestens zweimal vorkommt:

Abbildung 9: Anzahl einzigartiger Werte

```
wiscons.nunique()
```

```
SampleCodeNumber        645
ClumpThickness           10
UniformityOfCellSize     10
UniformityOfCellShape    10
MarginalAdhesion         10
SingleEpithelialCellSize 10
BareNuclei               10
BlandChromatin           10
NormalNucleoli           10
Mitoses                   9
Class                     2
```

```
wiscons.drop_duplicates(inplace=True, ignore_index=True)
```

Nun stellt sich die Frage, ob eine Schlüsselnummer bei diesen medizinischen Daten auch mehrmals vorkommen kann. Dies könnte beispielsweise dann der Fall sein, wenn etwa ein und dieselbe Patientin innerhalb eines längeren Zeitraums mehrfach untersucht wurde.
Die Frage kann an dieser Stelle nicht eindeutig beantwortet werden, da mir das dazu notwendige Fach- oder „Domain"-Wissen fehlt.

Es wird hier aber angenommen, dass es sich nur bei vollständigen Duplikaten um wirkliche Fehler handelt, die über die Funktion „drop_duplicates" an dieser Stelle auch entfernt werden können (Abb. 9), da durch das Löschen die noch verbleibenden Daten nicht verändert werden. Die Duplikate der Schlüsselnummer belasse ich aber im Datensatz, da ich unterstelle, dass eine Schlüsselnummer auch mehrfach vorkommen darf.
Im nächsten Schritt überprüfe ich, ob es unabhängige Variablen gibt, bei denen alle Werte identisch sind. Solche Variablen haben keine Varianz und können daher die abhängige Variable auch in keiner Weise erklären. Wie aus Abb. 9 zu sehen, ist dies aber nicht der Fall. Alle unabhängigen Variablen haben mindestens neun verschiedene Werte.
Die abhängige Variable („target") kann zwei verschiedene Kategorien annehmen: benign („gutartig") oder malignant („bösartig"). Auch diese Werte müssen in Zahlen umgewandelt werden. Dazu kann die Sklearn-Funktion „LabelBinarizer" verwendet werden:

Abbildung 10: LabelBinarizer()

```
# Binarize target columns so that categorical data will convert to numerical (that have no hierarchy)
y = wiscons['Class']
lb = LabelBinarizer()
y = lb.fit_transform(y)
```

IV. Test- und Trainingsdaten

Wie oben bereits erläutert, wird beim ML versucht, Muster und Gesetzmäßigkeiten aus Beispieldaten zu erkennen oder „zu lernen". Die aus den Daten gewonnenen Erkenntnisse lassen sich nach der Lernphase für die Analyse und Prognose von bisher unbekannten Daten verwenden.
Da in der Regel nur ein Datensatz zur Verfügung steht, der sowohl für das Training als auch für das Testen der Erkenntnisse an „unbekannten" Daten dienen soll, muss der Datensatz zunächst in einen Test- und einen Trainingsdatensatz aufgespalten werden. Falls diese Trennung nicht vorgenommen würde und man die Trainingsdaten auch zum Testen nähme, gäbe es keine „unbekannten" Daten mehr. Die Trainingsdaten würden dann bereits die Informationen enthalten, die man ja gerade versucht zu prognostizieren. In der Datenwissenschaft spricht man hierbei von einem Datenleck („Data Leakage"), weil Informationen von einem Lerndatensatz in einen Prognosedatensatz „auslaufen". Dies macht Modelle unbrauchbar und muss vermieden werden.
In Sklearn stehen verschiedene Funktionen zur Aufspaltung des Datensatzes zur Verfügung. Eine einfache Trennung von Test- und Trainingsdatensatz kann mit der Funktion „train_test_split" durchgeführt werden. Der Parameter „train_size" bestimmt die jeweilige Größe, ausgedrückt in Prozent. Im folgenden Beispiel wird ein 70:30-Split vorgenommen. Zuvor müssen die Übungsdatensätze aber in abhängige („y" oder „target") und unabhängige („X" oder „Feature-Vektor") Variablen getrennt werden (hier am Beispiel des Iris-Datensatz):

Abbildung 11: Pandas DataFrame

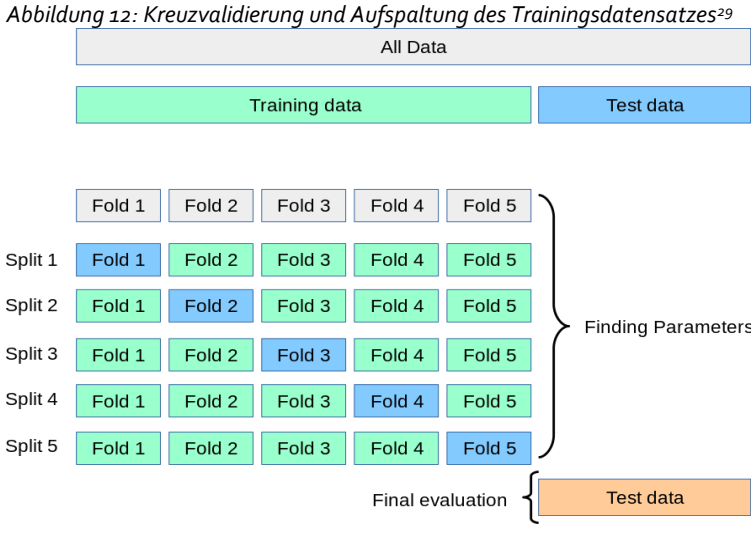
```
# Convert to pandas DataFrame
feature_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
X = pd.DataFrame(iris["data"], columns=feature_names)
y = pd.DataFrame((iris["target"]), columns=['species'])
# Split into train_set and test_set
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.70)
```

Nun stehen zum Modelltraining ein Target-Vektor („y_train") und eine Feature-Matrix („X_train") in zwei Pandas DataFrames zur Verfügung. Der Testdatensatz („y_test" und „X_test") wird „beiseite" gelegt und erst wieder zum Testen des fertigen Modells herangezogen.

V. Kreuzvalidierung

Generell möchte man bereits in der Trainingsphase die Güte verschiedener Modelle überprüfen und vergleichen. Der Testdatensatz sollte dazu aber nicht herangezogen werden, da dieser nur für eine finale Evaluierung des letztendlich ausgewählten Modells dient. Damit der Trainingsdatensatz sowohl für das Training als auch zur Evaluierung der Güte eines Modells herangezogen werden kann, muss auch dieser erneut aufgespalten werden. Die Validierung von Modellen innerhalb des Trainingsdatensatzes ist in der Datenwissenschaft auch als „Kreuzvalidierung" („Cross Validation" oder „CV") bekannt. In Abb. 11 wird ein Trainingsdatensatz beispielsweise fünfmal und damit jeweils im Verhältnis von 4:1 aufgespalten. Die entlang der Zeilen aufgespaltenen Datenpunkte sind auch als „Fold" bekannt. Bei fünf Modelltrainings-Durchläufen werden vier der fünf Folds aggregiert jeweils für das Training verwendet und ein Fold für die Validierung. So kann an den Ergebnissen der fünf Validierungs-Folds und deren Varianz die Güte und Stabilität eines Modells abgelesen werden.

Abbildung 12: Kreuzvalidierung und Aufspaltung des Trainingsdatensatzes[29]



VI. Datenaufbereitung – Teil 2

a. Imputation:

Nun können wir zu der Frage zurückkommen, warum wir die vorgefundenen Fragezeichen bzw. die als Stellvertreter stehenden Numpy NAN-Werte für die Variable „BareNuclei" in den WBC-Rohdaten nicht sofort ersetzt haben. Für das Ersetzen („Imputation") solcher Fehlerwerte stehen in Sklearn verschiedene Funktionen und Strategien zur Verfügung. Mit dem „SimpleImputer"

[29] (Scikit-Learn, 2021)

beispielsweise können die Fehlerwerte entweder durch den „Mean", „Median", den am häufigsten vorkommenden („most_frequent") oder einen beliebig gewählten Wert („constant") ersetzt werden. Nach einer eingehenden Datenanalyse kann hier die beste Strategie gewählt werden. Falls wir die Fehlerwerte aber an dieser Stelle ersetzten, würden die Informationen des gesamten Trainingsdatensatzes auf die einzelnen Folds in der Kreuzvalidierung übertragen. Somit läge wieder ein Datenleck („Data Leakage") vor, weil Informationen von einem Lerndatensatz in einen Prognosedatensatz „auslaufen". Das Ersetzen der Fragezeichen im BareNuclei-Feature darf deshalb nur innerhalb der Kreuzvalidierung bzw. innerhalb der jeweiligen Folds geschehen.

Hilfreich dafür ist die Sklearn-Funktion „Pipeline", die genau dies ermöglicht. In der folgenden Abbildung wird so beispielsweise eine Pipeline erstellt, die später innerhalb der einzelnen Folds die NAN-Werte im Feature-Vektor „BareNuclei" durch deren Median ersetzt.

Abbildung 13: Pipeline()

```
impute_only_list = X_train['BareNuclei']
impute_function = SimpleImputer(strategy='median', missing_values=np.nan)

preprocess_transformer = ColumnTransformer(transformers=[('impute', impute_function,
impute_only_list)], remainder='passthrough')

pipeline = Pipeline(steps=[('preprocess_transformer', preprocess_transformer)])
```

b. <u>Skalierung und Winsorisierung:</u>

Im nächsten Schritt erfolgt die Analyse des Trainingsdatensatzes. Einige ML-Algorithmen erfordern, dass die unabhängigen Variablen untereinander keine hohen Korrelationen aufweisen, einzeln betrachtet keine extremen Ausreißer („Outliers") haben oder standardnormalverteilt sind. Ein Blick auf den WBC-Trainingsdatensatz mit der Pandas-Funktion „skew" zeigt:

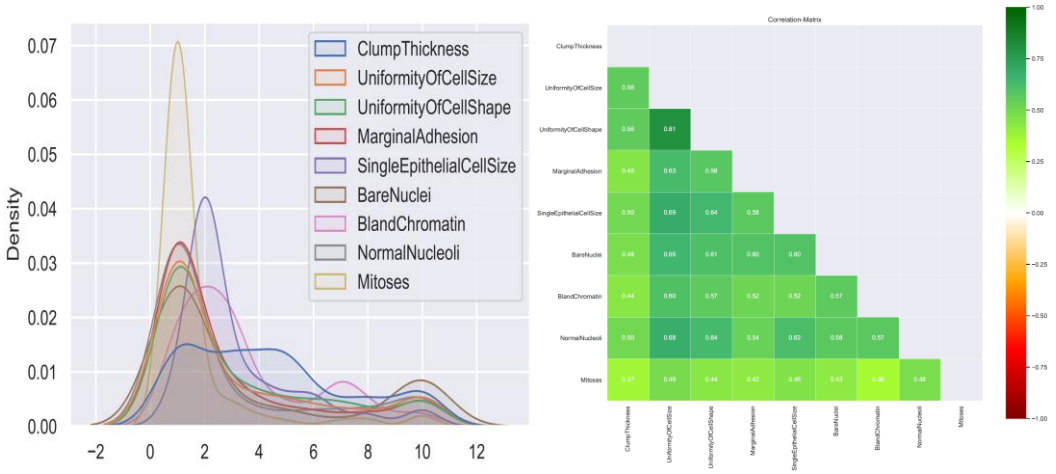
Abbildung 14: Skew()

```
X_train.skew()

ClumpThickness          0.593860
UniformityOfCellSize    1.270976
UniformityOfCellShape   1.202291
MarginalAdhesion        1.561012
SingleEpithelialCellSize  1.727318
BareNuclei              1.040087
BlandChromatin          1.071758
NormalNucleoli          1.405776
Mitoses                 3.431733
```

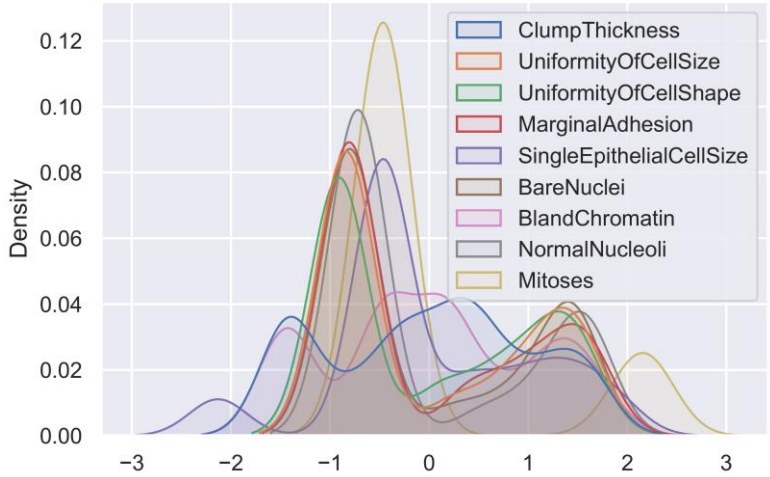
Die Skew-Werte für die unabhängigen Variablen sind alle positiv und bis auf „ClumpThickness" auch größer als eins. Sie sind damit stark rechtschief verteilt. Ein Blick auf das Verteilungs-Diagramm und die Korrelationsmatrix deckt zudem die Existenz einiger Ausreißer und positive Korrelationen zwischen den unabhängigen Variablen auf:

Abbildung 15: Verteilungen und Korrelationen vor der Transformation



Falls als ML-Modell die logistische Regression („LogisticRegression") gewählt wird, sollten die Feature-Vektoren standardnormalverteilt sein, unter anderem auch um später eine Interpretation der Feature-Wichtigkeit vornehmen zu können. Beim Vorliegen extremer Korrelationen zwischen einzelnen Feature-Paaren sollte zudem darüber nachgedacht werden, ein Element des jeweiligen Feature-Paars zu entfernen oder beide Elemente durch mathematische Operationen zu einem neuen Feature zu verbinden, das die beiden ursprünglichen Features ersetzt. Dadurch erhöht sich die Stabilität der Koeffizienten und damit auch deren Interpretierbarkeit. Die Korrelationen stellen hier aus meiner Sicht aber kein großes Problem dar, da der Höchstwert von 0,81 („Pearson-Korrelationskoeffizient") nicht als „extrem" anzusehen ist.
In Sklearn stehen verschiedene Transformatoren zur Verfügung. Unter anderem auch der „PowerTransformer", der die Feature-Werte so umwandelt, dass sie einen Mittelwert von null und eine Standardabweichung von eins ausweisen, ohne dabei die Informationen aus den Outliers komplett zu verlieren:

Abbildung 16: Verteilungen nach der Transformation



Möchte man hingegen die Outliers aus dem Feature-Vektor vollständig entfernen, kann dies auch durch eine sogenannte „Winsorisierung" umgesetzt werden. Dort werden die Extremwerte am oberen und unteren Rand - festgelegt beispielsweise durch ein bestimmtes Quantil - einfach „abgeschnitten" und durch den jeweiligen Rand-Wert ersetzt. Der daraus womöglich entstandene Informationsverlust kann durch das Hinzufügen eines neuen Binär-Features (Wert eins bei Vorliegen eines Outliers) verhindert werden.

Sowohl die Power-Transformation als auch die Winsorisierung sollten aber wieder innerhalb der einzelnen Folds in der Kreuzvalidierung stattfinden, um auch hier ein „Data Leakage" zu verhindern.

VII. Feature-Selection und Feature-Engineering

Das Ersetzen zweier hoch korrelierter Features durch eine Kombination beider Features oder das Entfernen von Fehlern aus dem Rohdatensatz ist in der Datenwissenschaft als „Feature-Engineering" bekannt. Dazu gehören alle Schritte, die ich in den vorigen Abschnitten „Datenaufbereitung-Teil 1 und 2" beschrieben habe. Ziel dabei ist es immer, die unabhängigen Variablen so in Features zu verwandeln, dass sie von den ML-Modellen auch wirklich verwendet werden können.

Daneben existiert noch das Konzept der Feature-Selection, bei dem es darum geht, die Anzahl der Features zu reduzieren. Auch im Bereich der Datenwissenschaft ist von „Occam's razor"[30] auszugehen, nach dem die einfachere Lösung, sprich weniger Features, oft die bessere ist. Für die Feature-Auswahl stehen verschiedene Methoden zur Verfügung. Sie lassen sich generell in drei Klassen unterteilen: Wrapper, Filter und Intrinsic.

Bei Filter-Methoden wird jedes einzelne Feature - beispielsweise anhand eines Korrelationsmaßes oder einer t-Statistik - daran gemessen, wie stark es individuell mit der Target-Variable „verbunden" ist. Für das ML-Modell werden dann die Features ausgewählt, die bei den gewählten Statistiken die höchsten Werte aufweisen.

Bei Wrapper-Methoden werden mehrere zufällige Untergruppen aus der Gesamtzahl der Features gebildet und diese dann jeweils für die Modellierung verwendet. Ausgewählt wird die Untergruppe an Features, für die das Modell die besten Ergebnisse produziert. In der Datenwissenschafts-Praxis ist die Anzahl der Features und die Anzahl der Datenpunkte oft sehr hoch. Das Produzieren von zufälligen Feature-Untergruppen und deren Modellierung ist daher auch sehr rechenintensiv.

Bei der Intrinsic-Methode verwendet ein ML-Modell zunächst alle verfügbaren Features. Im Anschluss daran werden die Features eliminiert, deren Feature-Wichtigkeiten („Feature Importances") für das gewählte Modell die geringsten Werte aufweisen.

Für jeden der drei Methoden bietet Sklearn eigene Funktionen und Methodiken an. Aufgrund der Komplexität des Themas werde ich hier aber nicht weiter darauf eingehen.

Zum jetzigen Zeitpunkt haben wir also eine Pipeline erstellt, in der später innerhalb der Kreuzvalidierung die Fragezeichen der ‚BareNuclei'-Variable ersetzt und die Werte aller unabhängigen Variablen mit einer Powertransformation standardisiert werden:

[30] (Occam's, 1287–1347)

Abbildung 17: Pipeline noch ohne Schätzer

```
impute_list = X_train['BareNuclei']
impute_function = SimpleImputer(strategy='median', missing_values=np.nan)
power_transform_list = X_train.columns.to_list()
power_transform_function = PowerTransformer(standardize=True)
preprocess_transformer = ColumnTransformer(
                                    transformers=
                                    [('impute', impute_function, impute_list),
                                    ('power_transform', power_transform_function,
                                    power_transform_list)], remainder='passthrough')
pipeline = Pipeline(steps=[('preprocess_transformer', preprocess_transformer)])
```

Der Pipeline fehlt nur noch ein ML-Algorithmus oder „Schätzer", damit wir mit der eigentlichen Modellierung beginnen können.

VIII. Modellierung

Für die (binäre) Klassifikation im Bereich des „Supervised Learning" stehen eine Vielzahl verschiedener Schätzer zur Verfügung. Zwei der am weitest verbreiteten sind die logistische Regression („LogisticRegression" oder „Logit") und der „Random Forest Classifier" oder „RFC".

Das Logit-Modell ist trotz seines Namens ein lineares Modell für die Klassifizierung diskreter abhängiger Variablen mit Hilfe von Regressionsverfahren. Dabei wird nicht nur versucht den Unterschied von Prognose („y_pred") und den jeweilig gemessenen Werten („y") zu minimieren, sondern je nach Art des Modells über einen Komplexitätsparameter („α") auch die Größe der Regressionskoeffizienten („Penalty"). Man spricht hierbei von „Regularisierung", um die Stabilität des Modells zu erhöhen. Die Regularisierung wird beim maschinellen Lernen standardmäßig angewendet, jedoch nicht in der Statistik.

Beim RFC handelt es sich um eine „Ensemble"-Methode, da der Algorithmus die Vorhersagen mehrerer Entscheidungsbäume („Decision Trees") kombiniert, indem er aus den einzelnen Baum-Prognosen einen Durchschnitt berechnet. Das RFC ist ein nicht-lineares Modell, da auch die Entscheidungsbäume selbst nicht linearer Natur sind. Eine Standardisierung, wie sie beim Logit-Modell erforderlich ist, ist hier nicht notwendig.

Wie oben ausgeführt, können wir beim Logit-Schätzer mit Komplexitätsparametern die „Einstellungen" des Modells verändern. Neben dem Regularisierungsparameter „α", der in Sklearn mit „C" (= 1 / α) eingestellt werden kann, gibt es für das Logit-Modell noch viele weitere Parameter zur Modelladjustierung, die im Allgemeinen auch als „Hyperparameter" bekannt sind. Hyperparameter gibt es aber nicht nur für das Logit-Modell, sondern auch für die meisten anderen Schätzer.

Damit wir diese Hyperparameter auch in der Modellierung spezifizieren können, bietet Sklearn verschiedene Möglichkeiten an, unter anderem die Funktion „GridSearchCV". Diesen verbinden wir nun mit unserer Pipeline zu folgendem Code:

Abbildung 18: Kompletter ML-Code Logit

```
1)  unfitted_estimator = LogisticRegression()
2)  impute_list = ['BareNuclei']
3)  impute_function = SimpleImputer(strategy='median', missing_values=np.nan)
4)  power_transform_list = X_train.columns.to_list()
5)  power_transform_function = PowerTransformer(standardize=True)
6)  preprocess_transformer = ColumnTransformer(transformers=
                              [('impute', impute_function, impute_list),
                               ('power_transform', power_transform_function,
                                 power_transform_list)],
                              remainder='passthrough')
7)  pipeline = Pipeline(steps=[('preprocess_transformer', preprocess_transformer),
    ('estimator', unfitted_estimator)])
8)  hyper_params = {'C': [1.0],'max_iter': [100]}
9)  score = 'roc_auc'
10) cv_grid = GridSearchCV(estimator=pipeline,
                  param_grid=hyper_params,
                  scoring=score,
                  cv=10)
11) cv_grid.fit(X_train, y_train)
```

Zur Erklärung:

1) Im ersten Schritt definieren wir unseren Sklearn-Schätzer, in diesem Fall die logistische Regression
2) In der „impute_list" geben wir die Liste der Features (hier: nur „BareNuclei") an, bei denen …
3) Numpy NAN-Werte (vorherige Fragezeichen) durch den Median des jeweiligen Features über die SimpleImputer-Funktion ersetzt werden sollen
4) In der „power_transform_list" geben wir die Liste der Features an (hier: alle Features), die …
5) durch die PowerTransformer-Funktion standardisiert werden sollen
6) In einem ColumnTransformer fügen wir die beiden Transformer (SimpleImputer und PowerTransformer) zuammen und …
7) übergeben diese zusammen mit dem vorher definierten Logit-Schätzer an eine Pipeline
8) In „hyper_params" definieren wir zwei Hyperparameter für den Logit-Schätzer
9) In „score" benennen wir das Gütemaß für die Messung der Modellergebnisse (dazu unten mehr)
10) In der GridSearchCV-Funktion fügen wir alle vorherigen Schritte zusammen: Die Funktion „GridSearchCV" erhält als aggregierten Schätzer die vorher definierte Pipeline (hier: „pipeline"), die Anzahl der Aufspaltungen des Trainingsdatensatzes für die Kreuzvalidierung (hier zehn: cv=10) und das Maß zur Berechnung der Modellqualität (hier: score = „roc_auc").
11) Mit der GridSearchCV-Methode „fit" beginnt nun das eigentliche Modell-Training: Der Algorithmus spaltet zunächst den Trainingsdatensatz („X_train") in zehn Folds auf, legt einen dieser zehn Folds (10% von X_train und y_train) „beiseite" und **verwendet nur die restlichen 90%** (90% von X_train und y_train) für folgende Schritte:

Der Algorithmus ersetzt in diesen 90% zunächst die Numpy-NAN-Werte des Features „BareNuclei" mit dessen Median, standardisiert in diesen 90% sämtliche Features mit der Funktion „PowerTransformer", setzt die Hyperparameter des Logit-Schätzers auf die angegebenen Werte und regressiert nun mit dem Logit-Logarithmus die transformierten Feature-Werte (90% von X_train) gegen die Target-Werte (90% von y_train). Anschließend berechnet er die Güte des errechneten Modells anhand des als Parameter übergebenen Gütemaß (score='roc_auc').
Dazu wird eine Prognose auf Basis des „beiseite" gelegten Fold (10% von „X_train") errechnet und anschließend diese Prognose („y_pred" für diese 10% von „X_train") mit den tatsächlichen Werten (10% von „y_train") verglichen. Das gesamte Prozedere wird in diesem Fall zehn Mal wiederholt und die Trainingsdaten dabei jeweils unterschiedlich aufgespalten.

Möchte man anstatt des Logit-Schätzers beispielsweise den RandomForest-Schätzer verwenden, ändert sich der Code nur im Hinblick auf die Angabe des Schätzers selbst (Zeile 1 in Abb.17) und dessen Hyperparameter (Zeile 8 in Abb.17).

IX. Modellergebnisse

Das Gütemaß von Modellen leitet sich von bestimmten Kennzahlen ab, die auf Basis der Frage berechnet werden, ob die Prognose für einen einzelnen Datenpunkt richtig oder falsch war. Falsch-Positiv („FP") sind Prognosen, die negative Ereignisse fälschlicherweise als positiv kategorisieren. Richtig-Positiv („TP") sind Prognosen, die positive Ereignisse richtigerweise als positiv kategorisieren. Falsch-Negativ („FN") sind Prognosen, die positive Ereignisse fälschlicherweise als negativ kategorisieren. Richtig-Negativ („TN") sind Prognosen, die negative Ereignisse richtigerweise als negativ kategorisieren. Aus der Anzahl der prognostizierten Datenpunkte für diese vier Klassen lassen sich nun bestimmte Gütemaße berechnen:

Abbildung 19: Formeln zur Prognosegüte

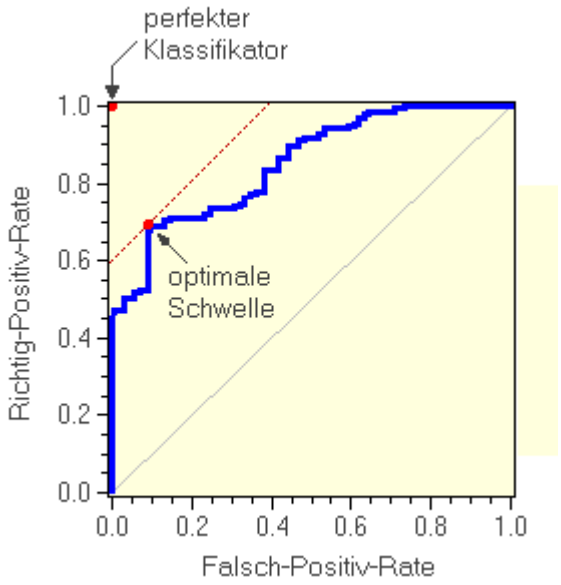
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$TP - Rate = \frac{TP}{TP + FN}$$

$$FP - Rate = \frac{FP}{FP + TN}$$

In der Regel kann kein Modell alle Datenpunkte vollständig korrekt prognostizieren. Beim Versuch, die Anzahl der Fehlprognosen - etwa durch geeignete Veränderung der Parameter - zu minimieren, stellt man fest, dass man zwar meist eine der oben aufgeführten Kennzahlen verbessern kann, aber oft nur auf Kosten der Verschlechterung der anderen Kennzahlen („Trade-Off"). „roc_auc" steht in unserer Modellierung für die „Fläche unterhalb der ROC-Kurve" (AUC = "area under the ROC curve"). Die ROC-Kurve (ROC = „Receiver Operating Characteristics") ist ein Diagramm, in dem dieser Trade-Off zwischen der Verbesserung der Richtig-Positiv-Rate (siehe oben: „TP-Rate") gegen die Falsch-Positiv-Rate (siehe oben: „FP-Rate") aufgezeigt wird:

Abbildung 20: ROC_AUC-Score (Quelle: http://www.statistics4u.info)



Der „roc_auc"-score beziffert in Abb. 19 die Fläche innerhalb des Quadrats, die rechts/unterhalb der blauen Linie liegt. Je kleiner die Fläche links/oberhalb der blauen Linie ist, umso besser ist die Modellgüte zu beurteilen, die mit einem gegebenen Datensatz erreicht werden kann.

Bei Datensätzen, bei denen die Werte der zu erklärenden Variable zahlenmäßig in etwa gleichverteilt sind („balanced datasets"), kann die Güte eines Modells auch anhand des „Accuracy"-score (siehe oben: Accuracy) aufgezeigt werden. Bei unausgewogenen Datensätzen („unbalanced datasets") hingegen verlieren hohe Werte des Accuracy-score an Aussagekraft. Hier sollten dann Gütemaße wie etwa der „roc_auc"-score verwendet werden.

Die Kreuzvalidierungs-Ergebnisse unseres Modelllaufs können mit der Funktion „cv_results_" abgefragt werden:

Abbildung 21: Modellergebnisse Logit

```
'splito_test_score': array([0.96415441]),
'split1_test_score': array([0.92095588]),
'split2_test_score': array([0.98713235]),
'split3_test_score': array([0.98242188]),
'split4_test_score': array([0.9921875]),
'split5_test_score': array([0.96484375]),
'split6_test_score': array([0.98925781]),
'split7_test_score': array([0.9921875]),
'split8_test_score': array([0.89941406]),
'split9_test_score': array([0.9921875]),
'mean_test_score': array([0.96847426]),
'std_test_score': array([0.03117807]),
```

Der „roc_auc"-Score unseres Modells in Abb. 20 liegt auf einer möglichen Skala von 0,5 bis 1,0 im Durchschnitt über alle zehn Folds hinweg bei 0,9685 und schwankt von minimal 0,8994 bis maximal 0,99219. Werte über 0,90 deuten allgemein auf eine hohe Güte des Modells hin.

Wir ziehen das beste Modell der Kreuzvalidierung nun zur finalen Evaluierung heran und verwenden dazu den Testdatensatz, den wir ganz am Anfang genau für diesen Zweck beiseitegelegt haben:

Abbildung 22: Validierung

```
y_pred = cv_grid.best_estimator_.predict(X_test)
roc_auc_score(y_test, y_pred)
```

```
0.9449624060150376
```

Das Ergebnis von 0,9450 liegt nur geringfügig unterhalb des durchschnittlichen Kreuzvalidierungsergebnisses. Damit können wir zunächst von einem stabilen Modell ausgehen (da ich aus Komplexitätsgründen an dieser Stelle nicht auf die weiteren Überlegungen eingehen möchte).

X. Modellverbesserung

Zur Modellverbesserung stehen grundsätzlich drei Methoden zur Verfügung:

- Verwendung eines anderen Schätzers
- Veränderung der Hyperparameter für einen gegebenen Schätzer
- Veränderung der Features

Andere Schätzer produzieren meist auch andere Werte für die Gütemaße. So können die Gütemaße verschiedener Schätzer miteinander verglichen und der Schätzer mit dem besten Gütemaß ausgewählt werden. Bei einem gegebenen Schätzer können die einzelnen Hyperparameter verändert und die Hyperparameter ausgewählt werden, die das beste Gütemaß liefern. Die Anzahl der Features kann mit den Methoden des „Feature-Selection" und die Features selbst mit den Methoden des Feature-Engineering verändert werden (siehe Abschnitt „Feature-Selection und Feature-Engineering"). Auch hier können die verschiedenen Modell-Inputs anhand des Gütemaßes evaluiert werden.

Bei der Verbesserung des Modellergebnisses sollte man allerdings nicht nur auf die Höhe des „roc_auc"- oder „accuracy"-scores achten, sondern auch darauf, ob diese Gütemaße über verschiedene Folds und Datensätze hinweg auch stabil sind. Ist dies nicht der Fall, ist davon auszugehen, dass die Prognosekraft des Modells für weitere, unbekannte Daten eher schwach ausfällt.

Auch hier gibt es meist einen Trade-Off zwischen der Prognosegenauigkeit („High Variance, low Bias") für einzelne Folds oder Modelldurchläufe und der Modellstabilität („High Bias, low Variance"). Ein instabiles Modell („High Variance, low Bias") lässt sich häufig daran erkennen, dass die Kreuzvalidierungsergebnisse auffällig hoch ausfallen, stark schwanken und/oder das durchschnittliche Kreuzvalidierungsergebnis stark vom Ergebnis der finalen Evaluierung abweicht.

C. ML-Programme in Python

Im Zuge der Programmierung von ML-Programmen mit Python, habe ich während meines Praxissemesters viele Funktionen erstellt. Ein kleiner Auszug davon soll in diesem Kapitel dargestellt und näher beschrieben werden. Dabei erhebe ich aufgrund der Neuheit des Themas für mich keinen Anspruch auf die vollständige Befolgung aller Konventionen der Python-Community („Pythonic Thinking").

I. Recursive Feature Selection (RFE)

Wie im Kapitel „Feature-Selection und Feature-Engineering" dargestellt, kann die Modellkomplexität durch die Verringerung der Feature-Anzahl reduziert werden, wodurch sich allerdings auch oft das Gütemaß des Modells verringert. Um eine optimale Feature-Anzahl/Modell-Gütemaß-Kombination auszuwählen, müssen die Gütemaße bei einer unterschiedlichen Anzahl von Features miteinander verglichen werden.

In Sklearn steht für die rekursive Feature-Eliminierung die Funktion „RFE" zur Verfügung. Dabei wird unter anderem die gewünschte Anzahl der im finalen Modell enthaltenen Features als Parameter übergeben. Die Funktion eliminiert, zunächst ausgehend von allen Features, bis zur Erreichung der gewünschten Feature-Anzahl nach und nach die Features, deren Feature-Wichtigkeit („Feature Importance") beim jeweiligen Modelllauf am geringsten ist. Bei linearen Modellen wie der logistischen Regression, beruht die Feature-Wichtigkeit auf der (absoluten) Stärke der errechneten Feature-Koeffizienten, nachdem deren Werte zuvor (zwingend) standardisiert wurden. Nach Ausführung der RFE-Funktion erhält man die gewünschte Anzahl von Features, die nach der Eliminierung der restlichen Features „übriggeblieben" sind, mithin also für das gegebene Modell die höchsten Feature-Wichtigkeiten ausweisen. Für dieses Modell mit den übrig gebliebenen Features kann dann ein Gütemaß errechnet werden.

Um Gütemaße mit einer unterschiedlichen Anzahl von Features miteinander zu vergleichen, müsste dieser Prozess mehrfach „händisch" wiederholt werden. Um dies zu rationalisieren, habe ich die Funktion „plot_rfe_to_model_score" geschrieben, die genau diese Wiederholungen für eine verschiedene Anzahl von Features durchführt und die jeweiligen Gütemaße in einem Schaubild ausweist:

Abbildung 23: Funktion plot_rfe_to_model_score()

```
def plot_rfe_to_model_score(X_df, y_df, unfitted_estimator, est_params: dict, cv, scoring: str = 'roc_auc', lowest_n_feat: int = 0,
                            highest_n_feat: int = 2000, n_feat_step_low_to_high: int = 1):
    estimator_name = unfitted_estimator.__class__.__name__
    if estimator_name == 'Pipeline':
        estimator = unfitted_estimator.named_steps.get('estimator')
    else:
        estimator = unfitted_estimator
    selector = RFE(estimator=estimator, step=n_feat_step_low_to_high)
    selector_name = selector.__class__.__name__
    selector_parameters = selector_name + '__n_features_to_select'
    selector_name_in_grid_search = 'param_' + selector_parameters
    num_features = X_df.shape[1]
    selector_parameters_grid = {selector_parameters: [i for i in range(lowest_n_feat, highest_n_feat, n_feat_step_low_to_high)]}
    estimator_parameters_grid = {'estimator__' + k: v for k, v in est_params.items()}
    parameter_grid = {}
    parameter_grid.update(selector_parameters_grid)
    parameter_grid.update(estimator_parameters_grid)
    pipeline, estimator = est_is_preprocessing_pipe_then_insert_other_pipe(unfitted_estimator, selector_name, selector)
    grid_search = GridSearchCV(pipeline, parameter_grid, scoring=scoring, cv=cv, return_train_score=False)
    grid_search.fit(X_df, y_df)
    best_est_in_grid = grid_search.best_estimator_
    best_params_in_grid = grid_search.best_params_
    best_score = grid_search.best_score_
    rfe_results = best_est_in_grid.named_steps[selector_name]
    index_of_selected_features = rfe_results.get_support(indices=True)
    feature_ranks = rfe_results.ranking_[index_of_selected_features]
    cv_results_df = pd.DataFrame(data=grid_search.cv_results_)
    mean_test_scores = cv_results_df.mean_test_score
    best_mean_test_score_index = mean_test_scores[mean_test_scores == best_score].index[0]
    n_features_to_select_list = cv_results_df[selector_name_in_grid_search].to_list()
    max_num_bars = min(best_mean_test_score_index + 11, len(n_features_to_select_list))
    x_axis_vals = range(len(n_features_to_select_list[0:max_num_bars]))
    y_axis_vals = mean_test_scores[0:max_num_bars]
    y_std_dev_vals = cv_results_df['std_test_score'][0:max_num_bars]
    x_labels = n_features_to_select_list[0:max_num_bars]
    bar_colors = ['r' if (x == mean_test_scores.iloc[best_mean_test_score_index]) else 'b' for x in mean_test_scores[0:max_num_bars]]
    fig_width = max(int(max_num_bars / 5), 15)
    fig_height = max(int(len(index_of_selected_features) / 5), 5)
    prtms_fig, (prtms_ax1, prtms_ax2) = plt.subplots(1, 2, figsize=(fig_width, fig_height))
    prtms_ax1.barh([i for i in range(len(feature_ranks))], feature_ranks, color='r', tick_label=X_df.columns[index_of_selected_features],
                   label=selector_name + '-rank')
    chart_title_1 = 'RFE-selected features from {}-Model'.format(estimator.__class__.__name__)
    prtms_ax1.set_title(chart_title_1, fontsize=14)
    prtms_ax1.legend(loc='best')
    prtms_ax2.set_ylim(ymin=0.5)
    prtms_ax2.bar(x_axis_vals, y_axis_vals, tick_label=x_labels, color=bar_colors, label=scoring + '-score')
    prtms_ax2.errorbar(x_axis_vals, y_axis_vals, yerr=y_std_dev_vals, fmt='o', color='black', elinewidth=1,
                       capthick=3, errorevery=1, alpha=1, ms=4, capsize=5, label='StandDev.')
    chart_title = 'RFE: Best test mean {}-Score: 0.967, Best number of features: {}'.format(scoring, n_features_to_select_list[
                                                                                            best_mean_test_score_index])
    for patch, score, stddev in zip(prtms_ax2.patches, y_axis_vals, y_std_dev_vals):
        prtms_ax2.text(patch.get_x() + patch.get_width() / 2, 0.52, '{:.3f} ({:.2f})'.format(score-0.023, stddev),
                       ha='center', va='bottom', rotation=90, color='white', size=10)
    prtms_ax2.set_title(chart_title, fontsize=14)
    prtms_ax2.legend(loc='best')
    return cv_results_df
```

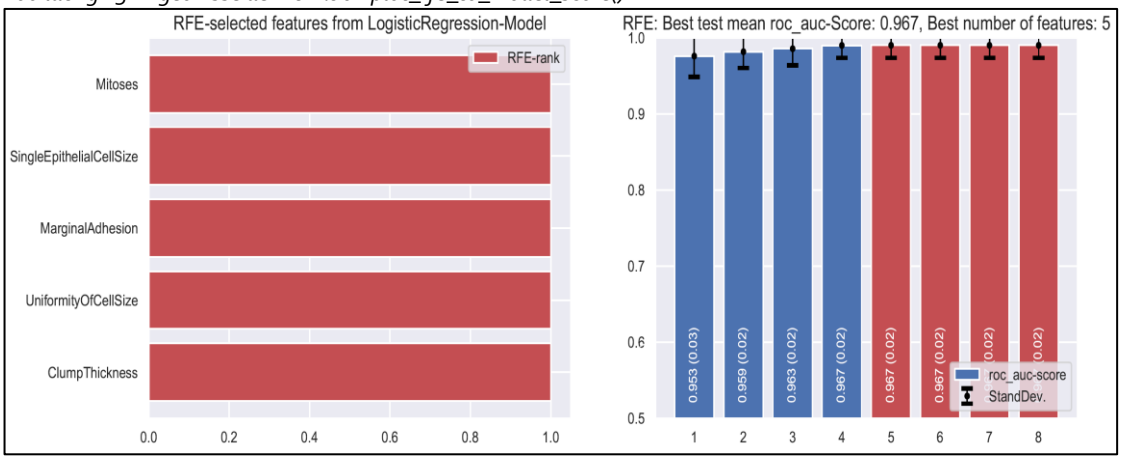
In unserem WBC-Übungsdatensatz gibt es nur neun Features, eine rekursive Feature-Eliminierung ist hier aufgrund der geringen Anzahl der Features nicht zwingend notwendig. Nicht unüblich in der Praxis sind hingegen hunderte oder tausende Features. Dort ist es sinnvoll, die Komplexität des Modells durch Feature-Eliminierung zu verringern (siehe Kapitel „Feature-Selection und Feature-Engineering"). Wir führen die rekursive Feature-Eliminierung an unserem WBC-Datensatz daher nur zur Demonstrationszwecken durch:

Abbildung 24: Ausführung der Funktion plot_rfe_to_model_score()

```
plot_rfe_to_model_score(X_train, y_train, pipeline, est_params=hyper_params, cv=10,
                        scoring= 'roc_auc', lowest_n_feat = 0, highest_n_feat = 9,
                        n_feat_step_low_to_high = 1)
```

Die Funktion „plot_rfe_to_model_score" liefert nach Eingabe der oben gezeigten Parameter folgende Ergebnisse:

Abbildung 25: Ergebnisse der Funktion plot_rfe_to_model_score()



Die höchste Modellgüte auf Basis einer rekursiven Feature-Eliminierung liefert das Modell mit den fünf Features „Mitoses", „SingleEphitelialCellSize", „MarginalAdhesion", „UniformityOfCellSize" und „ClumpThickness" mit einem Kreuzvalidierungs-roc_auc-score von 0,967. Da dieses Gütemaß nur knapp unterhalb des Gütemaßes des Modells mit allen Features (0,968) liegt, wäre eine Reduzierung von neun auf fünf Features überlegenswert. Auf diese Fragestellung möchte ich an dieser Stelle aufgrund der Komplexität aber nicht weiter eingehen.

II. Datenvisualisierungen

Häufig ist es wünschenswert, die Trainingsdaten auch optisch zu inspizieren, um sich mit den Daten besser vertraut zu machen oder um sie im Vorfeld zu analysieren. Insbesondere gilt dies bei einer großen Datenmenge und einer hohen Anzahl von Features. Dazu stellen einige Python-Bibliotheken, wie „Matplotlib", Funktionalitäten zur Verfügung. Nichtsdestotrotz müssen dort immer die einzelnen Aspekte der Visualisierung manuell angepasst werden. Da die Daten oft im immergleichen Datenformat vorliegen, bietet es sich für diese Visualisierungen an, eigene Funktionen zu erstellen. An dieser Stelle möchte ich deshalb auch beispielhaft einige Visualisierungs-Funktionen herausgreifen.

Die Funktion „plot_kde" erstellt aus den Daten eines Pandas DataFrame ein so genanntes KDE-Schaubild, wobei „KDE" für „Kernel Density Estimate" steht:

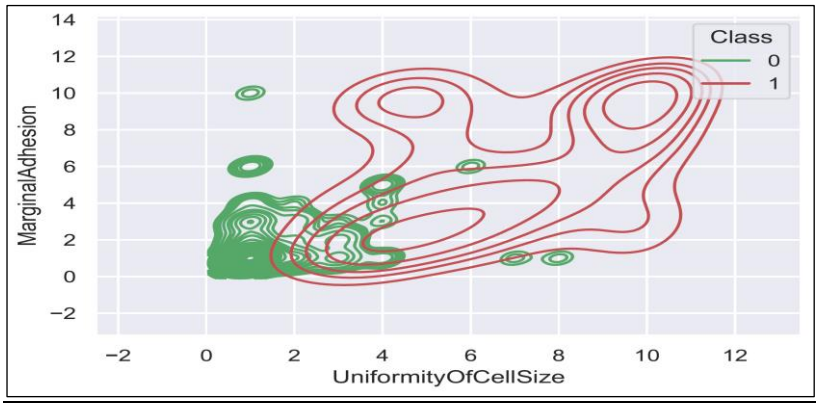
Abbildung 26: plot_kde()

```
def plot_kde(X_df, X_col1_num: int, X_col2_num: int, y_df, save_image: bool = False):
    pk_fig, pk_ax = plt.subplots()
    pk_Xy = pd.merge(X_df, y_df, right_index=True, left_index=True)
    pk_target = pk_Xy.iloc[:, -1]
    x_axis = pk_Xy.iloc[:, X_col1_num]
    y_axis = pk_Xy.iloc[:, X_col2_num]
    pk_palette = {1: 'r', 0: 'g'}
    sns.kdeplot(x=x_axis, y=y_axis, hue=pk_target, levels=30, fill=False, ax=pk_ax, alpha=1.0, shade=False,
                palette=pk_palette)
    if save_image:
        name_of_image = make_name_of_image(path_to_image_folder='../Images')
        pk_fig.savefig(name_of_image, format='pdf', bbox_inches='tight')
```

Mit Aufruf der Funktion und Übergabe des Trainingsdatensatzes und der entsprechenden Spaltennummern für die gewünschten Features, werden diese sofort in einem KDE-Schaubild angezeigt:

Abbildung 27: Ergebnis plot_kde()

```
plot_kde(X_train, X_col1_num=1, X_col2_num=3, y_df=y_train)
```



So kann man beispielsweise aus Abb. 26 erkennen, dass es verschiedene Häufigkeitszentren der Datenpunkte („Kernel Density") gibt und mit zunehmender Größe beider Variablen eine ansteigende Tendenz für eine bösartige Prognose/Brustkrebs besteht.

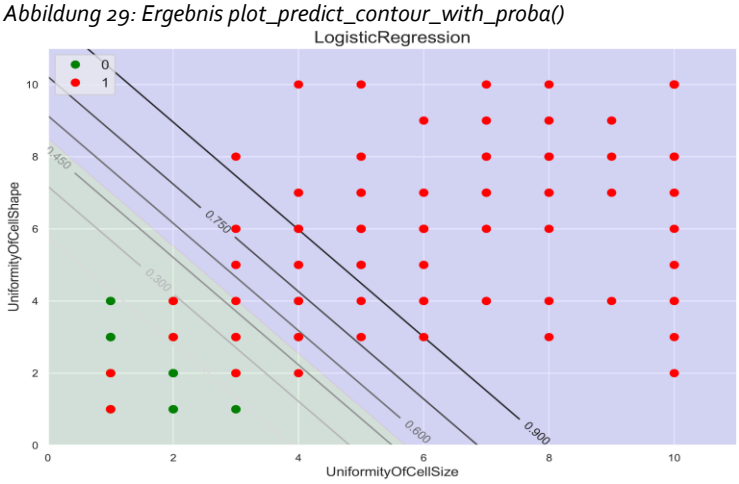
Mit der Funktion „plot_predict_contour_with_proba" lassen sich zwei Feature-Werte nach einer Modellierung in einem Punkt-Diagramm darstellen:

Abbildung 28: Funktion plot_predictcontour_with_proba()

```
def plot_predict_contour_with_proba(X_df, X_col1_num, X_col2_num, y_df, unfitted_estimator, hyper_params: dict = None,
                                    save_image: bool = False):
    df_col1 = X_df.iloc[:, X_col1_num]
    df_col2 = X_df.iloc[:, X_col2_num]
    min_x_first = df_col1.min() - 1
    min_x_second = df_col2.min() - 1
    max_x_first = df_col1.max() + 1
    max_x_second = df_col2.max() + 1
    x_first, x_second = np.meshgrid(np.linspace(min_x_first, max_x_first, 500).reshape(-1, 1),
                                    np.linspace(min_x_second, max_x_second, 500).reshape(-1, 1), )
    X_new = pd.DataFrame(np.c_[x_first.ravel(), x_second.ravel()], columns=[df_col1.name, df_col2.name])
    df_col1_col2 = pd.merge(df_col1, df_col2, right_index=True, left_index=True)
    if hyper_params is not None:
        unfitted_estimator.set_params(**hyper_params)
    estimator = unfitted_estimator.fit(df_col1_col2, y_df)
    if hasattr(estimator, "predict_proba"):
        y_proba = estimator.predict_proba(X_new)
        zz1 = y_proba[:, 1].reshape(x_first.shape)
    y_predict = estimator.predict(X_new)
    zz = y_predict.reshape(x_first.shape)
    pppc_fig, pppc_ax = plt.subplots(figsize=(12, 8))
    y_unique = len(y_df.iloc[:, 0].unique())
    colors = ['green', 'red', 'blue']
    cmap = mpl.colors.ListedColormap(colors)
    for ii in range(0, y_unique):
        pppc_ax.plot(df_col1[y_df.iloc[:, 0] == ii], df_col2[y_df.iloc[:, 0] == ii], marker='o', mfc=colors[ii],
                     mec=colors[ii],
                     ms=8, linestyle='', label=ii)
    pppc_ax.contourf(x_first, x_second, zz, cmap=cmap, alpha=0.1)
    if hasattr(estimator, "predict_proba"):
        contour = pppc_ax.contour(x_first, x_second, zz1, cmap='binary', alpha=1.0)
        pppc_ax.clabel(contour, inline=1, fontsize=12)
    pppc_ax.set_xlabel(df_col1.name, fontsize=14)
    pppc_ax.set_ylabel(df_col2.name, fontsize=14)
    pppc_ax.set_title(estimator.__class__.__name__, fontsize=18)
    pppc_ax.legend(loc="best", fontsize=14)
    pppc_ax.axis([min_x_first, max_x_first, min_x_second, max_x_second])
    if save_image:
        name_of_image = make_name_of_image(path_to_image_folder='../Images')
        pppc_fig.savefig(name_of_image, format='pdf', bbox_inches='tight')
    return estimator
```

Dazu wird der Funktion sowohl der Schätzer, als auch die Hyperparameter übergeben. Nach Aufruf der Funktion werden die Wahrscheinlichkeitsgrenzen des Modells für die beiden gewählten Features aufgezeigt:

Abbildung 29: Ergebnis plot_predict_contour_with_proba()



In Abb.28 sind die Wahrscheinlichkeitsgrenzen durch Linien gekennzeichnet. Ein Punkt im lila-gekennzeichneten Bereich hat eine höhere Wahrscheinlichkeit einer positiven, ein Punkt im grün-gekennzeichneten eine höhere Wahrscheinlichkeit einer negativen Diagnose. Passt man die Hyperparameter nun an und führt die Funktion erneut aus, lässt sich an der Verschiebung der Wahrscheinlichkeitsgrenzen der Einfluss der Hyperparameter auf das Modell ablesen.

Die Wahrscheinlichkeiten lassen sich auch für ein einzelnes Feature mit der Funktion „plot_predict_proba" darstellen:

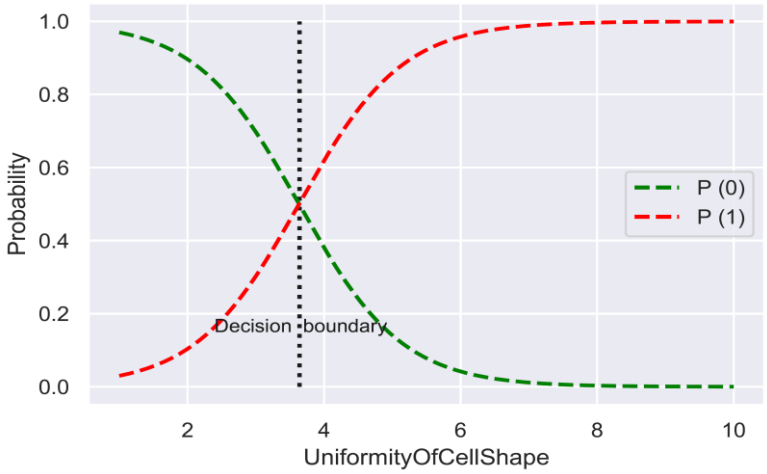
Abbildung 30: Funktion plot_predict_proba()

```
def plot_predict_proba(X_df, X_col_num: int, y_df, unfitted_estimator, hyper_parameters=None):
    df_col = X_df.iloc[:, X_col_num]
    ppp_fig, ppp_ax = plt.subplots()
    min = df_col.min()
    print('min is:', min)
    max = df_col.max()
    print('max is:', max)
    X_new = np.linspace(min, max, 1000).reshape(-1, 1)
    if hyper_parameters is not None:
        unfitted_estimator.set_params(**hyper_parameters)
    unfitted_estimator.fit(df_col.to_frame(), y_df.squeeze())
    y_predict_proba = unfitted_estimator.predict_proba(X_new)
    num_targets = len(y_df.iloc[:, 0].unique())
    if num_targets == 2:
        decision_boundary = X_new[y_predict_proba[:, 1] >= 0.5][0]
        ppp_ax.plot([decision_boundary, decision_boundary], [0, 1], "k:", linewidth=2)
        ppp_ax.text(decision_boundary + 0.02, 0.15, "Decision  boundary", fontsize=10, color="k", ha="center")

    ppp_ax.set_xlabel(df_col.name)
    ppp_ax.set_ylabel('Probability')
    colors = ['green', 'red', 'blue']  # Must be at least of equal size as number of unique target values
    for target in range(0, num_targets):
        ppp_ax.plot(X_new, y_predict_proba[:, target], color=colors[target], linestyle='--', linewidth=2,
                label='P ({})'.format(target))
    ppp_ax.legend()
```

Nach Ausführung der Funktion für ein gewähltes Feature lässt sich die Wahrscheinlichkeitsgrenze für den Wert, ab der das Target-Feature eine höhere Positiv-Wahrscheinlichkeit prognostiziert, ablesen:

Abbildung 31: Ergebnis plot_predict_proba()



Im Beispiel der Abb. 30 ist dies ab einem Wert von ca. 3,7 für das Feature „UniformityOfCellShape" der Fall.

Insgesamt habe ich während meiner Praxisphase rund 50 Funktionen und Algorithmen in Python geschrieben. Diese reichen bezüglich des Anwendungsgebiets von der ETL-Phase, über die Datenanalyse und -aufbereitung, das Feature-Engineering und die Feature-Auswahl bis hin zur Modellierung und Modell-Evaluierung.

5. Bewertung der Tätigkeiten

Mit der Praxisphase verband ich den Wunsch, mein Wissen über die Methoden der Datenanalyse, die ich bereits während meines Studiums der Volkswirtschaftslehre und meiner Berufstätigkeit als Vermögensverwalter kennengelernt hatte, zu vertiefen. Die Entwicklung der Computertechnologien und der Anstieg der Rechenkapazität moderner Computer hat die Möglichkeiten der Datenanalyse in den letzten Jahren drastisch erweitert. Maschinelles Lernen ermöglicht heute Einsichten in Daten, die noch vor wenigen Jahren nahezu undenkbar gewesen wären. Da ich im Berufsfeld Wertpapieranalyse „groß geworden" bin, in der die Auswertung von Daten ein wesentlicher Erfolgsfaktor ist, verfolge ich dieses Thema mit großem Interesse. Die Commerzbank AG hat mir während des Praxissemesters die Möglichkeit gegeben mich in das für mich neue Themengebiet des ML einzuarbeiten und ist mir dabei stets hilfreich zur Seite gestanden. Ich habe die wesentlichen Methoden des ML im Bereich des Supervised Learnings und teilweise auch des Unsupervised Learnings kennengelernt. Die Commerzbank bietet mir darüber hinaus auch noch die Möglichkeit meine Bachelor-Arbeit, die ich ebenfalls im Bereich ML schreiben möchte, zu begleiten. Abschließend kann ich daher sagen, dass meine Erwartungen, die ich vor dem Praxissemester hatte, voll erfüllt wurden.

6. Schlussbetrachtung

Ich möchte mich an dieser Stelle bei meinem Betreuer bei der Commerzbank AG, Herrn Dr. Alexander Fischer, ganz herzlich bedanken. Dr. Fischer hat einen bedeutenden Teil seiner Zeit, und wohl auch Energie, darauf aufgewendet mir das notwendige Wissen zu vermitteln. Er hat dies stets mit großer Hingabe, Ausdauer, Geduld und viel Verständnis getan. Dafür möchte ich ihm ganz herzlich danken.

7. Anhang

A. Abbildungsverzeichnis

Abkürzungsverzeichnis

BDAA ..... *Big Data & Advanced Analytics: Ein Commerzbank-Konzernbereich*
Fold ..... *Teil des Datensatzes nach Aufspaltung entlang der vorhandenen Datenpunkte (Zeilen)*
IDE ..... *Integrated Development Environment - Entwicklungsumgebung*
Iris ..... *Iris: Schwertlilien-Datensatz*
Logit ..... *Logistische Regression: Schätzer*
ML ..... *Machine Learning*
RFC ..... *Random Forest Classifier: Schätzer*
Schätzer ..... *ML-Modell oder ML-Algorithmus*
Sklearn ..... *Scikit-Learn: Eine Python ML Bibliothek*
WBC ..... *Wisconsin Breast Cancer: Brustkrebs-Datensatz*
y ..... *Tatsächlich gemessener Wert für die abhängige Variable*
y_pred ..... *Prognosewert für die abhängige Variable*

B. Quellenverzeichnis

Albon, C., 2018. *Machine Learning with Python Cookbook.* First Edition Hrsg. s.l.:O'Reilly.

Anaconda, 2020. [Online]
Available at: https://www.anaconda.com/state-of-data-science-2020

Barry, P., 2017. *Head First Python.* Second Edition Hrsg. s.l.:O'Reilly.

Brownlee, J., 2020. *Data Preparation for Machine Learning - Data Cleaning, Feature Selection and Data Transforms in Python.* v1.1 Hrsg. s.l.:Jason Brownlee.

Brownlee, J., 2021. *machinelearningmastery.com.* [Online]
Available at: https://machinelearningmastery.com/start-here/

Burkov, A., 2019. *The Hundred-Page Machine Learning Book.* s.l.:Andriy Burkov.

Commerzbank.de, 2021. *www.commerzbank.de.* [Online]
Available at: https://www.commerzbank.de/de/hauptnavigation/konzern/konzern.html

Géron, A., 2019. *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow.* Second Edition Hrsg. s.l.:O'Reilly.

Hastie, T., Tibshirani, R. & Friedman, J., 2009. *The Elements of Statistical Learning.* Second Edition Hrsg. s.l.:Springer.

Irvine, U. C., 1991. *Wisconsin Breast Cancer Data.* [Online]
Available at: http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/

Irvine, U. C., 1995. *Iris.* [Online]
Available at: https://archive.ics.uci.edu/ml/machine-learning-databases/iris/

Irvine, U. C., 2021. *Machine Learning Repository.* [Online]
Available at: https://archive.ics.uci.edu/ml/index.php

Kaggle, 2021. *Kaggle: Datasets.* [Online]
Available at: https://www.kaggle.com/datasets

McKinney, W., 2018. *Python for Data Analysis.* Second Edition Hrsg. s.l.:O'Reilly.

Ng, A., 2020. *Coursera: machine-learning.* [Online]
Available at: https://www.coursera.org/learn/machine-learning

Numpy, 2021. *Numpy.* [Online]
Available at: https://numpy.org/

Occam's, R., 1287–1347. *Wikipedia Occam's razor.* [Online]
Available at: https://en.wikipedia.org/wiki/Occam%27s_razor

Pandas, 2021. *Pandas.* [Online]
Available at: https://pandas.pydata.org/

Rossum, G. v. & team, P. d., 2018. *Python Tutorial.* Release 3.7.0 Hrsg. s.l.:Python Software Foundation.

Scikit-Learn, 2021. *https://scikit-learn.org/stable/.* [Online]
Available at: https://scikit-learn.org/stable/

Scipy, 2021. *Scipy.* [Online]
Available at: https://www.scipy.org/

towardsdatascience.com, 2021. *https://towardsdatascience.com.* [Online]
Available at: https://towardsdatascience.com/ai-machine-learning-deep-learning-explained-simply-7b553da5b960

VanderPlas, J., 2017. *Python Data Science Handbook.* First Edition Hrsg. s.l.:O'Reilly.

Wikipedia: Banken in Deutschland, 2021. *Wikipedia.* [Online]
Available at:
https://de.wikipedia.org/wiki/Liste_der_gr%C3%B6%C3%9Ften_Banken_in_Deutschland

www.bigdata-insider.de, 2021. *bigdata-insider.* [Online]
Available at: https://www.bigdata-insider.de/was-ist-machine-learning-a-592092/#:~:text=Machine%20Learning%20ist%20ein%20Teilbereich,k%C3%BCnstliches%20Wissen%20aus%20Erfahrungen%20generiert.