Machine Learning am Beispiel der Vorhersage der Qualität von Weinsorten

1.1.1 Klärung der Aufgabenstellung

Das Ziel unserer Arbeit ist es, im Rahmen von überwachtem Machine Learning die Qualität von Weißweinen anhand ihrer chemischen Eigenschaften vorherzusagen und durch verschiedene Methoden aus dem Kurs, die Vorhersage soweit zu optimieren, dass der gewählte Score das beste Ergebnis am Ende liefert.

Arbeits-Hypothese vor Beginn der Untersuchung

Die Beurteilung von Weinen anhand einer Ordinalskala stellt eine Klassifikation dar, sodass wir auf alle Methoden der Klassifikation von überwachtem Lernen zurückgreifen können, um eine genaue Vorhersage für unbekannte Weine zu treffen.
Wir gehen davon aus, dass die Qualität eines Weißweins von seiner chemischen Komposition abhängt und dabei der gebundene Säuregehalt (fixed acidity) von z.B. L-(+)-Weinsäure, Zitronensäure oder Milchsäure für den guten Geschmack eine wichtige Rolle spielt. Die bei einer Wasserdampfdestillation flüchtig zu isolierenden niedermolekularen Säuren (volatile acidity), wie Essigsäure bis hin zur Buttersäure, sind hingegen organische Säuren, die aus unerwünschten aeroben Fermentationsprozessen von Bakterien- und nicht von der Hefe-stammen und sich eher negativ auf den Geschmack und daher auf die Qualität auswirken dürften. Ebenfalls sollte sich der Gehalt von Schwefeldioxid negativ auf die Qualität eines Weißweins auswirken, wobei auch hier zwischen gebundenem Schwefeldioxid und freiem Schwefeldioxid (free sulfur dioxide) unterschieden werden muss. Ein hoher Anteil an chemisch gebundenem Schwefeldioxid deutet auf einen hohen Gehalt an Carbonylverbindungen wie Acetaldehyd, Brenztraubensäure und 2-Ketoglutarsäure hin, die aus der Fermentation von Hefe stammen und sich sicherlich positiv auf die Qualität auswirken dürften. Der Restzuckergehalt dürfte keinen besonders großen Einfluss auf die Qualität haben, da dessen Konzentration bekanntlich zwischen “trockenen” und “lieblichen” Weißweinen stark variieren kann.
Für Eigenschaften, die sich positiv auf die Qualität auswirken, ist zu erwarten, dass sie einen positiven Korrelationskoeffizienten zur Zielvariablen Qualität aufweisen, wobei im besten Fall ein Wert von +1 erreicht wird. Im entgegengesetzten Fall wäre zu erwarten, dass die Korrelation negativ ist und den maximalen Wert von -1 erreicht. Zur besseren Beurteilung sollte der Rat eines Experten herangezogen werden.

Qualitätskriterien (Welche Sorte Score?, Wie hoch muss er am Ende sein?)
Nach Absprache mit der Auftraggeberin werden wir neben einem selbst geschriebenen Accuracy-Score, der in der Arbeit als Relaxed Accuracy bezeichnet wird, die Balanced Accuracy, Precision und den Recall für die zu testenden Algorithmen als Bewertungskriterien heranziehen. Es wird die Balanced Accuracy, und nicht die gewöhnliche Accuracy verwendet, da unsere Zielspalte unausgewogen ist (siehe Verteilung der Qualitätsvariabeln).

Welche Sorte von Machine Learning-Verfahren?
Es handelt sich um ein Klassifikationsproblem, wofür wir folgende geeignete ML-Methoden testen möchten:
-logistische Regression
-K- Nearest Neighbor (KNN)

-Support Vector Machine (SVM)
-Random Forest
- Multilaterales Perceptron(MLP)
Als Preprocessing-Methoden verwenden wir Klassenzusammenlegung und PCA.
Zur Hyperparameter-Optimierung verwenden wir das Grid-Search

1.1.2 Beschreibung der Daten

Die Datensätze zu Weißweinen und Rotweinen sind auf der Internetseite der Universität UC Irvine (https://archive.ics.uci.edu/dataset/186/wine+quality) zum Herunterladen frei verfügbar. Die beiden Datensätze beschreiben physikalische und chemische Eigenschaften von portugiesischen Weinen der Klasse "Vinho Verde" und liefern hierzu eine Qualitätsbeurteilung.Sie stammen ursprünglich aus einer Publikation (Cortez et al. 2009). Der Datensatz zu Weißweinen besteht aus 4898 Zeilen für einzelne Weißweinproben mit 12 Spalten, wobei 11 Spalten für die physikalischen und chemischen Eigenschaften stehen und die zwölfte Spalte die Qualität der Weine als Ordinalzahl zwischen 0 (sehr schlecht) und 10 (sehr gut) angibt.
Die physikalischen und chemischen Eigenschaften sind gebundener Säuregehalt (fixed_acidity), flüchtiger Säuregehalt (volatile acidity), Zitronensäuregehalt (citric acid), Restzuckergehalt (residual_sugar), Chloridgehalt (chloride), ungebundener Schwefeldioxidgehalt (free_sulfur_dioxide), Gesamtgehalt an Schwefeldioxid (total_sulfur_dioxide), Dichte (density), pH-Wert (pH), Sulfatgehalt (sulfphates) und Alkoholgehalt (alcohol).

Tabelle 1.: Der rohe Datensatz von 4898 Weißweinen.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
...	...	...	...	...	...	...	...	...	...	...	...	...
4893	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6
4894	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5
4895	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6
4896	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7
4897	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6

4898 rows × 12 columns

Tabelle 2.: Die statistischen Datenkennzahlen.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027	3.188267	0.489847	10.514267	5.877909
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151001	0.114126	1.230621	0.885639
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.090000	0.410000	9.500000	5.000000
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.180000	0.470000	10.400000	6.000000
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.280000	0.550000	11.400000	6.000000
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.820000	1.080000	14.200000	9.000000

Bei den Datensätzen fehlen keine Werte. Die in Tabelle 1. exemplarisch aufgelisteten Daten zeigen, dass nur Daten des Datentyps Float vorliegen und daher kein Encoder im Preprocessing verwendet werden muss, um Strings in Zahlenwerte zu überführen. Die voneinander abweichenden Größenordnungen der Daten in den Spalten zeigen auf, dass die Daten skaliert werden müssen.

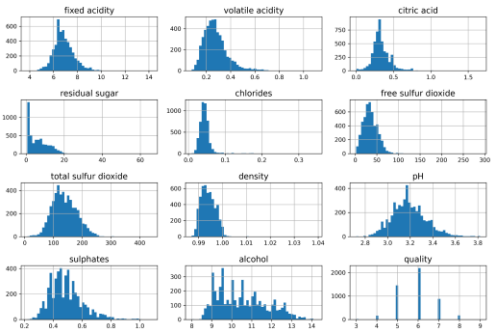

Abbildung: Verteilung der einzelnen Features. Es ist bei Residual Sugar ein großer Balken links zu erkennen; “fixed Acidity”, “volatile Acidity”, “citric acid”, “chlorides” , “free sulfur dioxide”, “total sulfur dioxide”, “density” , “pH-Wert und annähernd “sulphates” zeigen eine Normalverteilung- ebenso die Zielspalte “quality”.

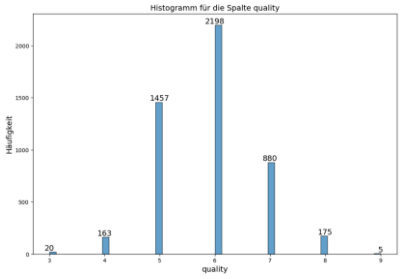

Abbildung: Detaillierte Darstellung der Zielvariabeln “Quality” für Weißwein. Es fällt auf, dass die Randklassen unterrepräsentiert sind:” 0” “1” und “10” fehlen ganz und in “1” und “9” kommen nur sehr wenige vor; hingegen sind “5”, “6” überrepräsentiert;

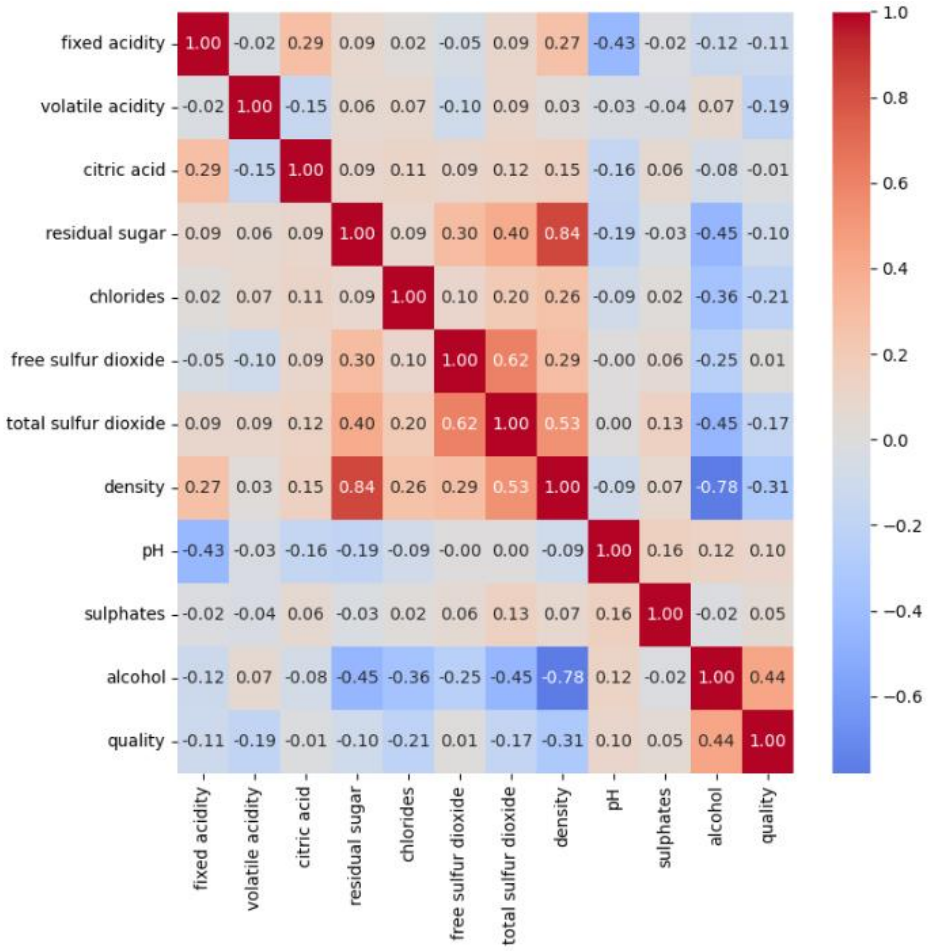

Abbildung 1: Die Korrelationsmatrix der einzelnen Eigenschaften und der Zielvariablen Quality.

Korrelationen zwischen einzelnen Spalten?
Deutlich ist an der Korrelationsmatrix in Abbildung 1. zu erkennen, dass die Dichte stark negativ, d.h. verringernde Dichte gegenüber der Dichte von Wasser, mit dem Alkohol und stark positiv, d.h. zunehmende Dichte gegenüber Wasser, mit dem Restzuckergehalt korreliert. Das zeigt, dass die Massenkonzentration beider Stoffe im Vergleich zu denen der anderen am höchsten ist und Alkohol und Restzucker nach Wasser am häufigsten im Wein vorkommen. Der pH-Wert korreliert negativ mit dem gebundenen Säuregehalt, da der pH-Wert als negativer dekadischer Logarithmus von der Protonenkonzentration berechnet wird, welche wiederum positiv mit der Säurekonzentration korreliert. Zitronensäure korreliert positiv mit dem gebundenen Säuregehalt, da diese, wie oben angeführt, eine gebundene Säure darstellt. Interessant ist, dass der Gesamtgehalt an Schwefeldioxid positiv mit dem Restzuckergehalt korreliert. Das liegt daran, dass Restzucker, vor allem D- Glucose und D-Fructose in ihrer offenkettigen Form als Carbonylverbindungen, Schwefeldioxid, bzw. im Wasser als Hydrogensulfit vorliegend, binden können und ein wasserlösliches Addukt bilden, dass reversibel unter Einwirkung von Protonen zurück reagieren kann.

In der Zielspalte Qualität ist deutlich zu erkennen, dass der Alkoholgehalt des Weines stark positiv mit dieser korreliert. Ebenso korreliert negativ die Dichte, da beide Größen, wie vorher angeführt, in entgegengesetzter Beziehung stehen. Die flüchtigen Säuren weisen eine

negative Korrelation zur Qualität auf, da sie ein Anzeichen von schlechter Qualität sind.

1.1.3 Beschreibung der Datenvorbereitung

Da die verwendeten Features allgemein normalverteilt waren, wurde der Standard Scaler zur Skalierung der Daten verwendet, was sich besser auf die Datenanalyse auswirkte.
Zunächst wurden die Features “Citric Acid” , “Residual_sugars” (mit hohem Balken auf der linken Seite), “free_sulfur_dioxide”, “Sulphates” und “pH-Wert” entfernt, weil sie nur sehr gering mit der Zielspalte “Quality” korrelieren. Dieser Versuch stellte sich jedoch als weniger erfolgreich dar, weshalb dieser hier nicht dokumentiert ist. In einem weiteren Versuch wurden nur die Features “Citric Acid” , “Residual_sugars” und “pH” entfernt, was sich als vielversprechend herausstellte. Für den endgültigen zweiten Durchgang mit nur noch acht von elf Eigenschaften wurde eine Klassenzusammenlegung der Zielspalte in drei Klassen (0-4 schlecht, 5-6 mittel, 7-10 gut) durchgeführt, was sich positiv auf die Scores auswirkte.

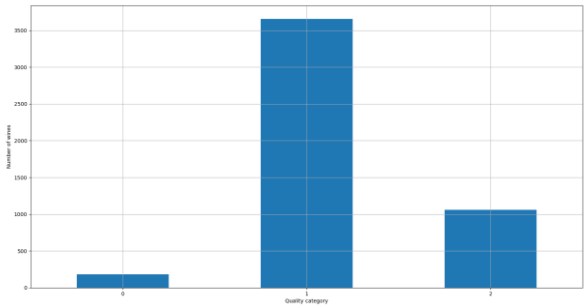

Abbildung: Histogramm zur Klassenzusammenlegung der Zielspalte in “schlecht”, “mittel” und “gut”.

1.1.4 Anwendung verschiedener Algorithmen

<u>K-Nearest Neighbor(KNN)</u>

Übergeordnete Kennzahlen (Durchschnitt) KNN:
Die Werte in dieser Tabelle repräsentieren die Leistung des Modells auf einem einzigen Split des Datensatzes, ohne Cross-Validation. Dies könnte zu einer stärkeren Abhängigkeit von der spezifischen Aufteilung der Daten führen und die Stabilität der Ergebnisse beeinträchtigen.

Model	Scaled	Grouped (3 classes)	Accuracy	Precision	Recall	F1-Score	Balanced Accuracy	Weighted Recall	Relaxed Accuracy
KNN	no	no	0.748	0.56	0.536	0.541	0.536	0.748	0.748

KNN	yes	no	0.787	0.615	0.575	0.59	0.575	0.787	0.787
KNN	no	yes	0.838	0.822	0.71	0.751	0.71	0.838	0.838
KNN	yes	yes	0.870	0.849	0.757	0.794	0.757	0.87	0.87

Das KNN Modell zeigt auf den Trainingsdaten eine moderate Leistung, aber auf den Testdaten gibt es eine erhebliche Abnahme der ausbalancierten Genauigkeit. Es scheint, als ob das Modell auf den Testdaten nicht gut generalisiert und Schwierigkeiten hat, bestimmte Klassen korrekt zu klassifizieren. Es könnte notwendig sein, das Modell zu verbessern, indem zum Beispiel Skaliert wird, wie im nächsten Beispiel.

Beim skalierten KNN Modell ist eine verbesserte Leistung des Modells ersichtlich. Die ausbalancierte Genauigkeit auf den Trainingsdaten und Testdaten ist gestiegen, und Precision, Recall sowie F1-Score haben sich verbessert. Die Konfusionsmatrix zeigt weniger Verwirrung zwischen den Klassen. Dennoch gibt es weiterhin Schwierigkeiten bei den Klassen 3 und 9. Es könnte notwendig sein, das Modell weiter zu optimieren, um diese spezifischen Herausforderungen anzugehen.

Beim unskaliert-gruppierten KNN Model zeigt der Klassifikationsbericht, dass das Modell für Klasse 2 (Label 2) gute Ergebnisse erzielt, während die Leistung für Klasse 3 (Label 3) niedriger ist, insbesondere im Hinblick auf Recall und F1-Score. Die Konfusionsmatrix zeigt, dass das Modell Schwierigkeiten hat, Klasse 3 korrekt zu klassifizieren.Die durchschnittliche Genauigkeit beträgt 71%, und die gewichtete Recall beträgt 83.8%, was darauf hindeutet, dass das Modell im Durchschnitt gut abschneidet, aber Verbesserungen in spezifischen Klassen möglich sind.

Beim skaliert-gruppierten KNN Model ist Verhältnis zu unskaliert-gruppierten KNN Model, haben sich die Präzision, der Recall und der F1-Score für alle Klassen (1, 2, 3) verbessert. Insbesondere für Klasse 3 (Label 3) ist eine deutliche Verbesserung beim Recall und F1-Score zu sehen. Durchschnittliche Precision: 84.9% (vorher: 82.2%) Durchschnittlicher Recall: 75.7% (vorher: 71%) Durchschnittlicher F1-Score: 79.4% (vorher: 75.1%) Ausgeglichene Genauigkeit: 75.7% (vorher: 71%) Gewichteter Recall: 87% (vorher: 83.8%) Relaxed Accuracy: 87% (vorher: 83.8%)

Support Vector Machine (SVM)

Zu einem ersten Überblick über die Leistungen der verschiedenen zur Auswahl stehenden Klassifikationsalgorithmen wurde ein gemeinsamer Durchlauf vorgenommen - mit und ohne Skalierung. Bei diesem Durchlauf wurde nur der StandardScaler betrachtet. (In einer früheren Projektphase wurde mit einer ganzen Reihe von Scalers experimentiert. Für

unsere Daten schienen der StandardScaler oder der MinMaxScaler besser abzuschneiden.)

Dabei wurden die Algorithmen nur mit deren Default-Werten in den Vergleich übergeben, abgesehen vom Parameter random_state, der (mit konstantem Wert 0) durchgängig mitgegeben wurde, wo immer anwendbar (im Falle von SVC ist der Parameter random_state zwar auch vorgesehen, wird aber in Kombination mit anderen Default-Werten ignoriert).

Nicht in dem gemeinsamen Durchlauf vertreten waren insbesondere:

- KNN (da separat untersucht).

- Logistische Regression (wäre auch separat zu untersuchen). Nach dem ersten Versuch ausgeschlossen, da mit dem Default-Wert für den Parameter max_iterint (default=100) eine Konvergenz nicht erreicht werden konnte:

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

(Anmerkung: Die Fehlermeldung wurde auch in dem Lauf mit dem StandardScaler produziert.)

Eine weitere Ausnahme hinsichtlich Default-Werten war das Modell Linear SVC (Modell analog SVC (SVM) mit kernel = 'linear', aber mit einer unterschiedlichen Implementierung). Hier wurde ein FutureWarning generiert, mit dem Hinweis, einen Parameter explizit zu setzen:

```
/Users/alexandrebouyer/anaconda3/lib/python3.11/site-packages/sklearn/svm/_classes.py:32: FutureWarning: The defaul
t value of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the wa
rning.
```

(mit dual='auto' ist er dann durchgelaufen).

Im Durchlauf ist für jeden Algorithmus eine Cross-Validierung enthalten (mit cv=5). Zuerst wurden die Trainingsscores ermittelt. Die im Durchlauf berechneten Scores wurden in einen Data Frame geschrieben:

	Model	Scaled	Accuracy	Precision	Recall	F1-Score	Balanced Accuracy	Weighted Recall	Relaxed Accuracy
0	Decision Tree	no	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1	Decision Tree	yes	1.000	1.000	1.000	1.000	1.000	1.000	1.000
2	Gaussian Naive Bayes	no	0.462	0.478	0.462	0.456	0.337	0.322	0.911
3	Gaussian Naive Bayes	yes	0.461	0.481	0.461	0.455	0.341	0.326	0.911
4	Random Forest	no	1.000	1.000	1.000	1.000	1.000	1.000	1.000
5	Random Forest	yes	1.000	1.000	1.000	1.000	1.000	1.000	1.000
6	SGD	no	0.438	0.283	0.438	0.337	0.179	0.420	0.937
7	SGD	yes	0.523	0.467	0.523	0.453	0.234	0.235	0.938
8	Linear SVC	no	0.538	0.512	0.538	0.468	0.198	0.401	0.946
9	Linear SVC	yes	0.537	0.508	0.537	0.471	0.199	0.405	0.947
10	SVM rbf	no	0.459	0.358	0.459	0.312	0.159	0.371	0.930
11	SVM rbf	yes	0.615	0.583	0.615	0.578	0.263	0.441	0.957

Neben fünf Standard-Scores aus sklearn wurden zwei selbstdefinierte Scores mit ausgegeben: Weighted Recall und Relaxed Accuracy - auf die Definition und Motivation für diese Scores wird noch später im Bericht eingegangen.

Precision, Recall und F1-Score wurden der Zeile weighted avg aus dem classification_report entnommen.

Die Übersicht oben deutet auf eine starke Überanpassung bei Decision Tree und Random Forest hin. Bei den anderen Algorithmen sind die meisten Scores meistens sehr schwach, mit Ausnahme der Metrik Relaxed Accuracy.

Im Durchlauf mit ausgegeben sind auch die Scores der Cross-Validierung und die Konfusionsmatrix für jeden Algorithmus, hier exemplarisch für SVM rbf:

```
Model: SVM rbf, Scaled: True

Cross-validation Scores: [0.565 0.585 0.562 0.57  0.557], Mean: 0.568

Classification Report:
                    3      4            5            6           7      8    9  \
precision    1.000000    0.0     0.676525     0.585259    0.633846    0.0  0.0
recall       0.090909    0.0     0.629948     0.821129    0.295552    0.0  0.0
f1-score     0.166667    0.0     0.652406     0.683415    0.403131    0.0  0.0
support     11.000000  112.0  1162.000000  1789.000000  697.000000  142.0  5.0

              accuracy    macro avg  weighted avg
precision     0.614599     0.413661      0.583446
recall        0.614599     0.262506      0.614599
f1-score      0.614599     0.272231      0.577729
support       0.614599  3918.000000   3918.000000

Confusion Matrix:
 [[  11    0    0    0    0    0    0]
 [   0  112    0    0    0    0    0]
 [   0    0 1162    0    0    0    0]
 [   0    0    0 1789    0    0    0]
 [   0    0    0    0  697    0    0]
 [   0    0    0    0    0  142    0]
 [   0    0    0    0    0    0    5]]

Accuracy: 0.615, Precision: 0.583, Recall: 0.615, F1-Score: 0.578,
Balanced Accuracy: 0.263, Weighted Recall: 0.441, Relaxed Accuracy: 0.957
```

Eine visuelle Inspektion der CV-Werte deutet darauf hin, dass sie robust sind (gilt für alle Algorithmen im Durchlauf). Auf eine Berechnung der Standardabweichung wurde zunächst verzichtet.

Auffallend sind dagegen alle Konfusionsmatrizen. Sie sind ausnahmslos diagonal! Die Besonderheiten, die bei der Ermittlung der Trainingsscores, hochgekommen sind, konnten trotz intensiver Diskussion mit dem Auftraggeber noch nicht abschließend geklärt werden.

Anschließend wurden die Testscores ermittelt, mit folgenden Ergebnissen:

	Model	Scaled	Accuracy	Precision	Recall	F1-Score	Balanced Accuracy	Weighted Recall	Relaxed Accuracy
0	Decision Tree	no	0.580	0.568	0.580	0.572	0.385	0.459	0.907
1	Decision Tree	yes	0.580	0.568	0.580	0.572	0.385	0.460	0.910
2	Gaussian Naive Bayes	no	0.443	0.445	0.443	0.433	0.349	0.347	0.903
3	Gaussian Naive Bayes	yes	0.442	0.444	0.442	0.431	0.350	0.347	0.902
4	Random Forest	no	0.649	0.668	0.649	0.629	0.397	0.464	0.948
5	Random Forest	yes	0.653	0.670	0.653	0.633	0.397	0.462	0.945
6	SGD	no	0.406	0.255	0.406	0.303	0.205	0.414	0.908
7	SGD	yes	0.477	0.429	0.477	0.404	0.246	0.216	0.923
8	Linear SVC	no	0.493	0.450	0.493	0.415	0.222	0.384	0.927
9	Linear SVC	yes	0.492	0.504	0.492	0.419	0.225	0.305	0.926
10	SVM rbf	no	0.418	0.305	0.418	0.268	0.169	0.507	0.907
11	SVM rbf	yes	0.540	0.504	0.540	0.495	0.264	0.488	0.942

Bei Decision Tree und Random Forest sind einige Werte stark zurückgegangen, was die Überanpassung im ersten Durchgang bestätigt. Bei den anderen Algorithmen war der Rückgang moderat, die meisten Scores sind aber noch schwach, mit Ausnahme von Relaxed Accuracy.

Schaut man sich die Ergebnisse aber im Detail an, werden andere Schwierigkeiten sichtbar (hier wieder für SVM):

```
Model: SVM rbf, Scaled: True

Cross-validation Scores: [0.565 0.585 0.562 0.57  0.557], Mean: 0.568

Classification Report:
                3     4           5           6           7     8  accuracy  \
precision  0.0   0.0    0.580420    0.512077    0.616438   0.0  0.539796
recall     0.0   0.0    0.562712    0.777506    0.245902   0.0  0.539796
f1-score   0.0   0.0    0.571429    0.617476    0.351562   0.0  0.539796
support    9.0  51.0  295.000000  409.000000  183.000000  33.0  0.539796

             macro avg  weighted avg
precision     0.284823      0.503542
recall        0.264353      0.539796
f1-score      0.256744      0.495362
support     980.000000    980.000000

Confusion Matrix:
 [[  0   0   6   3   0   0]
 [  0   0  33  17   1   0]
 [  0   0 166 129   0   0]
 [  0   0  74 318  17   0]
 [  0   0   7 131  45   0]
 [  0   0   0  23  10   0]]

Accuracy: 0.54, Precision: 0.504, Recall: 0.54, F1-Score: 0.495,
Balanced Accuracy: 0.264, Weighted Recall: 0.488, Relaxed Accuracy: 0.942
```

Es zeigt sich eine deutliche Verschiebung der Randklassen (3-4, 7-8) zu den mittleren (5-6), in manchen Algorithmen sogar innerhalb der mittleren Klassen. Der Linear SVC hat interessanterweise entgegen den genannten Verschiebungsrichtungen eine nicht existente Randklasse (9) erzeugt:

```
Model: Linear SVC, Scaled: True

Cross-validation Scores: [0.546 0.538 0.532 0.535 0.508], Mean: 0.532

Classification Report:
                3           4           5           6           7     8    9  \
precision  0.0   0.500000    0.556911    0.467967    0.615385   0.0  0.0
recall     0.0   0.019608    0.464407    0.821516    0.043716   0.0  0.0
f1-score   0.0   0.037736    0.506470    0.596273    0.081633   0.0  0.0
support    9.0  51.000000  295.000000  409.000000  183.000000  33.0  0.0

           accuracy   macro avg  weighted avg
precision  0.491837    0.305752      0.503880
recall     0.491837    0.192749      0.491837
f1-score   0.491837    0.174587      0.418518
support    0.491837  980.000000    980.000000

Confusion Matrix:
 [[  0   0   5   4   0   0   0]
 [  0   1  27  22   1   0   0]
 [  0   0 137 158   0   0   0]
 [  0   1  69 336   2   0   1]
 [  0   0   8 167   8   0   0]
 [  0   0   0  31   2   0   0]
 [  0   0   0   0   0   0   0]]

Accuracy: 0.492, Precision: 0.504, Recall: 0.492, F1-Score: 0.419,
Balanced Accuracy: 0.225, Weighted Recall: 0.305, Relaxed Accuracy: 0.926
```

```
/Users/alexandrebouyer/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:2394: UserWarning:
y_pred contains classes not in y_true
  warnings.warn("y_pred contains classes not in y_true")
```

Ein später, letzter Versuch, solche Probleme an dieser Stelle zu reduzieren, wurde mit Hilfe von sklearn.utils.class_weight.**compute_class_weight** unternommen. Die Dokumentation dazu ist sehr knapp gehalten. Die Berechnung des Arrays ist zwar gelungen, aber die konkrete Anwendbarkeit zeigte sich komplexer als ursprünglich erhofft und führte nicht zum Erfolg.

Unter den vorgestellten Algorithmen scheint der SVM insgesamt noch am besten geeignet für einen Versuch mit Hyperparameter-Tuning. Wie mit dem Auftraggeber vereinbart, soll fürs Grid-Search nun die Metrik Relaxed Accuracy verwendet werden, die sich robust gezeigt hat.

An dieser Stelle sei noch erwähnt, dass vor Erstellung dieser Metrik ein Grid-Search über mehrere Algorithmen und Parameter auf Basis der Accuracy-Metrik und für zwei verschiedene Scalers (StandardScaler und MinMaxScaler) testweise ausgeführt wurde. Der Vollständigkeitshalber wird hier ein Ausschnitt der damaligen Parameterliste und der Ergebnisse angezeigt (allerdings zu den Rotweindaten):

```
search_params = [ (DecisionTreeClassifier(random_state=random_state),
                   {'decisiontreeclassifier__criterion': ['gini', 'entropy', 'log_loss'],
                    'decisiontreeclassifier__max_depth': [2, 4, 8],
                    'decisiontreeclassifier__max_leaf_nodes': [2, 4, 8, 16],
                   },
                   MinMaxScaler()
                  ),
                  (DecisionTreeClassifier(random_state=random_state),
                   {'decisiontreeclassifier__criterion': ['gini', 'entropy', 'log_loss'],
                    'decisiontreeclassifier__max_depth': [2, 4, 8],
                    'decisiontreeclassifier__max_leaf_nodes': [2, 4, 8, 16],
                   },
                   StandardScaler()
                  ),
                  (KNeighborsClassifier(),
                   {'kneighborsclassifier__n_neighbors': [2, 4, 8, 16, 32, 64, 128],
                    'kneighborsclassifier__leaf_size': [2, 4, 8, 16]
                   },
                   MinMaxScaler()
                  ),
                  (KNeighborsClassifier(),
                   {'kneighborsclassifier__n_neighbors': [2, 4, 8, 16, 32, 64, 128],
                    'kneighborsclassifier__leaf_size': [2, 4, 8, 16]
                   },
                   StandardScaler()
                  ),
                  (LogisticRegression(max_iter=10000, random_state=random_state),
                   {'logisticregression__C': [1, 10, 100, 1000]},
                   MinMaxScaler()
                  ),
```

```
iterate_search_params = [search_best_parameters(estimator, grid_params, scaler) for estimator,
                         grid_params, scaler in search_params]
```

```
Best Hyperparameters: {'decisiontreeclassifier__criterion': 'gini', 'decisiontreeclassifier__max_depth': 8, 'decisi
ontreeclassifier__max_leaf_nodes': 16}
Test Accuracy with Best Model: 0.56
Best Hyperparameters: {'decisiontreeclassifier__criterion': 'gini', 'decisiontreeclassifier__max_depth': 8, 'decisi
ontreeclassifier__max_leaf_nodes': 16}
Test Accuracy with Best Model: 0.56
Best Hyperparameters: {'kneighborsclassifier__leaf_size': 2, 'kneighborsclassifier__n_neighbors': 16}
Test Accuracy with Best Model: 0.57
Best Hyperparameters: {'kneighborsclassifier__leaf_size': 2, 'kneighborsclassifier__n_neighbors': 64}
Test Accuracy with Best Model: 0.60
Best Hyperparameters: {'logisticregression__C': 10}
Test Accuracy with Best Model: 0.64
Best Hyperparameters: {'logisticregression__C': 100}
Test Accuracy with Best Model: 0.63
Best Hyperparameters: {'randomforestclassifier__criterion': 'entropy', 'randomforestclassifier__max_depth': 12, 'ra
ndomforestclassifier__max_features': 'sqrt'}
Test Accuracy with Best Model: 0.70
Best Hyperparameters: {'randomforestclassifier__criterion': 'entropy', 'randomforestclassifier__max_depth': 12, 'ra
ndomforestclassifier__max_features': 'sqrt'}
Test Accuracy with Best Model: 0.72
Best Hyperparameters: {'sgdclassifier__loss': 'log_loss'}
Test Accuracy with Best Model: 0.51
Best Hyperparameters: {'sgdclassifier__loss': 'log_loss'}
Test Accuracy with Best Model: 0.57
Best Hyperparameters: {'linearsvc__C': 10, 'linearsvc__tol': 0.0001}
Test Accuracy with Best Model: 0.63
Best Hyperparameters: {'linearsvc__C': 1, 'linearsvc__tol': 0.001}
Test Accuracy with Best Model: 0.63
```

die Hyperparameter für SVC wurden aufgrund der langen Laufzeit separat für jeden Kerneltyp gesucht:

```
search_params = [
                  (SVC(kernel='poly'),
                   {'svc__C': [1, 10, 100],
                    'svc__gamma': ['scale', 'auto', 1, 2],
                    'svc__degree': [3, 4, 5]
                   },
                   StandardScaler()
                  )
                ]

iterate_search_params_poly_mms = [search_best_parameters(estimator, grid_params, scaler) for estimator,
                                  grid_params, scaler in search_params]
```

```
Best Hyperparameters: {'svc__C': 10, 'svc__degree': 3, 'svc__gamma': 'scale'}
Test Accuracy with Best Model: 0.63
```

```
search_params = [
                  (SVC(kernel='poly'),
                   {'svc__C': [1, 10, 100],
                    'svc__gamma': ['scale', 'auto', 1, 2],
                    'svc__degree': [3, 4, 5]
                   },
                   StandardScaler()
                  )
                ]

iterate_search_params_poly_sts = [search_best_parameters(estimator, grid_params, scaler) for estimator,
                                  grid_params, scaler in search_params]
```

```
Best Hyperparameters: {'svc__C': 10, 'svc__degree': 3, 'svc__gamma': 'scale'}
Test Accuracy with Best Model: 0.63
```

Auch mit Boosting-Verfahren wurde versucht, bessere Ergebnisse zu erzielen:

```
search_params = [ (GradientBoostingClassifier(random_state=random_state),
                   {'gradientboostingclassifier__learning_rate': [0.1, 1, 10],
                    'gradientboostingclassifier__n_estimators': [100, 200, 400],
                    'gradientboostingclassifier__max_depth': [2, 4, 6, 8],
                   },
                   MinMaxScaler()
                  ),
                  (GradientBoostingClassifier(random_state=random_state),
                   {'gradientboostingclassifier__learning_rate': [0.1, 1, 10],
                    'gradientboostingclassifier__n_estimators': [100, 200, 400],
                    'gradientboostingclassifier__max_depth': [2, 4, 6, 8],
                   },
                   StandardScaler()
                  )
                ]

iterate_search_params_gb = [search_best_parameters(estimator, grid_params, scaler) for estimator,
                            grid_params, scaler in search_params]
```

```
Best Hyperparameters: {'gradientboostingclassifier__learning_rate': 0.1, 'gradientboostingclassifier__max_depth': 6
, 'gradientboostingclassifier__n_estimators': 100}
Test Accuracy with Best Model: 0.71
Best Hyperparameters: {'gradientboostingclassifier__learning_rate': 0.1, 'gradientboostingclassifier__max_depth': 6
, 'gradientboostingclassifier__n_estimators': 100}
Test Accuracy with Best Model: 0.71
```

```
dt  = DecisionTreeClassifier(max_depth=4, max_leaf_nodes=12, random_state=random_state)
gnb = GaussianNB()
kn  = KNeighborsClassifier(leaf_size=2, n_neighbors=64)
sgd = SGDClassifier(loss='log_loss', max_iter=10000)


search_params = [ (AdaBoostClassifier(estimator=dt, random_state=random_state),
                   {'adaboostclassifier__n_estimators': [100, 200, 400],
                    'adaboostclassifier__learning_rate': [0.1, 1, 10]
                   },
                   StandardScaler()
                  ),
                  (AdaBoostClassifier(estimator=gnb, random_state=random_state),
                   {'adaboostclassifier__n_estimators': [100, 200, 400],
                    'adaboostclassifier__learning_rate': [0.1, 1, 10]
                   },
                   MinMaxScaler(),
                  ),
                  (AdaBoostClassifier(estimator=gnb, random_state=random_state),
                   {'adaboostclassifier__n_estimators': [100, 200, 400],
                    'adaboostclassifier__learning_rate': [0.1, 1, 10]
                   },
                   StandardScaler()
                  ),
                 (AdaBoostClassifier(estimator=sgd, random_state=random_state),
                   {'adaboostclassifier__n_estimators': [100, 200, 400],
                    'adaboostclassifier__learning_rate': [0.1, 1, 10]
                   },
                   MinMaxScaler()
                  ),
```

```
Best Hyperparameters: {'adaboostclassifier__learning_rate': 1, 'adaboostclassifier__n_estimators': 400}
Test Accuracy with Best Model: 0.66
Best Hyperparameters: {'adaboostclassifier__learning_rate': 1, 'adaboostclassifier__n_estimators': 400}
Test Accuracy with Best Model: 0.60
Best Hyperparameters: {'adaboostclassifier__learning_rate': 1, 'adaboostclassifier__n_estimators': 400}
Test Accuracy with Best Model: 0.61
Best Hyperparameters: {'adaboostclassifier__learning_rate': 0.1, 'adaboostclassifier__n_estimators': 400}
Test Accuracy with Best Model: 0.62
Best Hyperparameters: {'adaboostclassifier__learning_rate': 0.1, 'adaboostclassifier__n_estimators': 100}
Test Accuracy with Best Model: 0.57
```

Es wurden in der Gruppe weitere Optimierungsverfahren wie PCA und Feature Auswahl besprochen und es gab erste Versuche dazu bereits am Anfang der Studie:

```python
def train_test_model(model_name, model, scaled, pca_n_components):

    # Split the data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)

    if scaled:
        # Initialize the StandardScaler
        scaler = StandardScaler()
        # Fit the scaler to the training data and transform both training and test data
        X_train = scaler.fit_transform(X_train)£
        X_test = scaler.transform(X_test)

    if pca_n_components:
        # Initialize PCA
        pca = PCA(pca_n_components)
        # Fit PCA on the scaled training data
        X_train = pca.fit_transform(X_train)
        # Transform both training and test data
        X_test = pca.transform(X_test)
```

	Model	Scaled	PCA Components	Accuracy	Precision	Recall	F1-Score
0	SVM rbf	no	none	0.559	0.590	0.559	0.528
1	SVM rbf	yes	none	0.694	0.665	0.694	0.676
2	SVM rbf	yes	3	0.600	0.564	0.600	0.579
3	SVM rbf	yes	6	0.659	0.632	0.659	0.639
4	Random Forest	no	none	0.712	0.681	0.713	0.696
5	Random Forest	no	3	0.612	0.583	0.613	0.596
6	Random Forest	no	6	0.703	0.669	0.703	0.684
7	Random Forest	yes	none	0.731	0.698	0.731	0.714
8	Random Forest	yes	3	0.669	0.640	0.669	0.654
9	Random Forest	yes	6	0.706	0.676	0.706	0.689

Zumindest zu dem Zeitpunkt und bezogen auf die Rotweindaten und die zwei oben dargestellten Algorithmen war eine Leistungsverbesserung bei Einsatz von PCA nicht unbedingt erkennbar.

Bei der Ermittlung von Features Importance wurden anfangs verschiedene Techniken ausprobiert. Einige Beispiele (weiterhin bezogen auf die Rotweindaten):

```
for feature in features:
    X = df[[feature]]
    clf = svc.fit(X, y)
    feature_score = clf.score(X, y)
    print(feature, '- score:', feature_score)
```

```
fixed acidity - score: 0.4590368980612883
volatile acidity - score: 0.48655409631019386
citric acid - score: 0.4590368980612883
residual sugar - score: 0.425891181988743
chlorides - score: 0.425891181988743
free sulfur dioxide - score: 0.425891181988743
total sulfur dioxide - score: 0.4884302689180738
density - score: 0.425891181988743
pH - score: 0.425891181988743
sulphates - score: 0.48467792370231394
alcohol - score: 0.557848655409631
```

alcohol > total sulfur dioxide > volatile acidity > sulphates > citric acid

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
```

```
X = df.iloc[:, 0:-1]

selector = SelectKBest(f_classif)
X_sel = selector.fit_transform(X, y)
```

```
selector.scores_
```

```
array([  6.28308116,  60.91399283,  19.69066447,   1.05337358,
         6.03563859,   4.7542331 ,  25.47850952,  13.3963569 ,
         4.3417643 ,  22.27337609, 115.85479747])
```

alcohol > volatile acidity > total sulfur dioxide > sulphates > citric acid > density

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
```

```
clf = RandomForestClassifier()
clf.fit(X, y)
clf.feature_importances_
```

```
array([0.07430174, 0.10459353, 0.07249162, 0.07181214, 0.07827887,
       0.06560783, 0.10565918, 0.09388781, 0.07724735, 0.11299726,
       0.14312266])
```

alcohol > sulphates > total sulfur dioxide > volatile acidity > density

```
from sklearn.feature_selection import RFE
#clf = RandomForestClassifier(1)
selector = RFE(clf, n_features_to_select=6)
selector = selector.fit(X, y)
```

```
selector.get_support()
```

```
array([ True,  True, False, False, False, False,  True,  True, False,
        True,  True])
```

fixed acidity, volatile acidity, total sulfur dioxide, density, sulphates, alcohol

Aus Zeitgründen wurde das Thema in der aktuellen Studie zunächst nicht weiter vertieft und zugunsten anderer Maßnahmen wie Klassenzusammenlegung oder eigener Metriken etwas zurück priorisiert.

Für die Metriken wird auf den Abschnitt Kommentar des Endergebnisses verwiesen.

Multilaterales Perceptron (MLP)

Tabelle: Durchschnittliche Scores für verschiedene MLPs mit einer, zwei, drei oder vier Hidden-Layers zur Vorhersage der Qualität von Weißweinen.

	Model	Scaled	Accuracy Training:	Precision Training	Recall Training	F1-Score Training	Balanced Accuracy Training	Relaxed Accuracy Training	Accuracy Test	Precision Test	Recall Test	F1-Score Test	Balanced Accuracy Test	Relaxed Accuracy Test
0	MLP(11) relu, adam	yes	0.573	0.556	0.573	0.550	0.253	0.960	0.533	0.525	0.533	0.502	0.281	0.938
1	MLP(22) relu, adam	yes	0.596	0.589	0.596	0.577	0.319	0.957	0.533	0.532	0.533	0.510	0.316	0.932
2	MLP(150,75,35) relu, adam	yes	0.970	0.971	0.970	0.969	0.948	0.995	0.602	0.601	0.602	0.594	0.407	0.924
3	MLP(300,150,75,35) relu, adam	no	0.558	0.559	0.558	0.536	0.282	0.953	0.485	0.438	0.485	0.454	0.252	0.932
4	MLP(400,150,75,35) relu, adam	yes	0.946	0.953	0.946	0.947	0.867	0.995	0.606	0.613	0.606	0.602	0.452	0.939
5	MLP(150,75) relu, adam	yes	0.993	0.993	0.993	0.993	0.994	0.999	0.610	0.600	0.610	0.601	0.405	0.937
6	MLP(150,75) relu, lbfgs	yes	1.000	1.000	1.000	1.000	1.000	1.000	0.595	0.587	0.595	0.590	0.420	0.920

Wie an der oberen Tabelle zu sehen, ist bei den Versuchen 2-6 ein Overfitting aufgetreten, wobei dieses sich stärker bei den skalierten Daten zeigt. Nur bei den MLPs mit einem Hidden-Layer zeigte sich, dass kein Overfitting auftrat (siehe 0. und 1.). Overfitting liegt

dann vor, wenn die Scores für die Trainingsmenge viel größer sind als für die Testmenge. Wir erhalten aber dennoch bessere Scores für die Testmengen bei den MLPs mit mehreren Hidden-Layers als bei den einfachen Perceptronen 0. und 1. .

Es ist zu sehen, dass die mit dem Standard- Scaler skalierten Daten bessere Scores ergeben, als die unskalierten Datensätze (vergleiche 3. und 4.) .

Tabelle: Genauere Aufschlüsselung der Klassifikationsscores nach Zielvariabeln.

MLP(11) relu, adam , Scaled: True Accuracy: 0,533, Balanced Accuracy: 0,281,Relaxed Accuracy: 0,938, Precision: 0,525, Recall: 0,533, F1-Score: 0,502, Weighted Recall: 0,309						
Zielklasse	**3**	**4**	**5**	**6**	**7**	**8**
Precision	0	0,428	0,581	0,523	0,518	0,5
Recall	0	0,059	0,559	0,724	0,311	0,03
F1-Score	0	0,103	0,57	0,6	0,389	0,057
Support	9	51	295	409	183	33

MLP(22) relu, adam , Scaled: True Accuracy: 0,533, Precision: 0,532, Recall: 0,533, F1-Score: 0,51, Balanced Accuracy: 0,316, Weighted Recall: 0,299, Relaxed Accuracy: 0,932						
Zielklasse	**3**	**4**	**5**	**6**	**7**	**8**
Precision	1,0	0,538	0,574	0,517	0,517	0,286
Recall	0,11	0,137	0,552	0,706	0,328	0,06
F1-Score	0,2	0,219	0,563	0,597	0,401	0,1
Anzahl	9	51	295	409	183	33

Model: MLP(150,75,35) relu, adam, Scaled: True Accuracy: 0.602, Precision: 0.601, Recall: 0.602, F1-Score: 0.594, Balanced Accuracy: 0.407, Weighted Recall: 0.476, Relaxed Accuracy: 0.924						
Zielklasse	**3**	**4**	**5**	**6**	**7**	**8**
Precision	0	0,567	0686	0,574	0,581	0,5
Recall	0	0,333	0,57	0,707	0,59	0,242
F1-Score	0	0,42	0,622	0,634	0,585	0,327
Anzahl	9	51	295	409	183	33

Die oberen Tabellen zeigen die Scores, aufgeschlüsselt nach der Zielvariabeln, für die ersten drei MLP aus der Tabelle zu den mittleren Scores. Die Precision gibt an, wie groß der Anteil der richtig Vorhergesagten in der Menge aller Vorhergesagten ist, die sich aus den richtig Vorhergesagten und Falsch-Positiven zusammensetzt.
Der Recall gibt an, wie groß der Anteil der richtig Vorhergesagten in der Menge aller Richtigen ist, die zu erwarten sind. Er gibt also Auskunft über die Falsch-Negativen. Die Konfusionsmatrix stellt dies anschaulich dar.
Repräsentativ sind für die ersten drei Modelle die Konfusionsmatrizen dargestellt. Es zeigt sich, dass vor allem zwischen den Klassen “5” und “6” Falsch-Negative (rechts oben von der Diagonalen aus) und Falsch-Positive (links unten von der Diagonalen) auftreten. Entlang der Diagonalen sind die richtig vorgesagten Klassifikationen aufgelistet. Viele andere Methoden zeigen die gleiche Präferenz für beide Klassen auf. Der Grund hierfür, ist dass die Klassen sehr unausgewogen sind und

Konfusionsmatrix für MLP(11) relu, adam , Scaled: True						
	Y_pred für “3“	**Y_pred für “4“**	**Y_pred für “5“**	**Y_pred für “6“**	**Y_pred für “7“**	**Y_pred für “8“**
Y_echt für “3“	0	1	5	3	0	0
Y_echt für “4“	0	3	31	16	1	0
Y echt für “5“	0	2	165	122	6	0
Y_echt für “6“	0	1	76	296	35	1
Y_echt für “7“	0	0	7	119	57	0
Y_echt für “8“	0	0	0	21	11	1

Konfusionsmatrix für MLP(22) relu, adam , Scaled: True						
	Y_pred “3“	**Y_pred “4“**	**Y_pred “5“**	**Y_pred “6“**	**Y_pred “7“**	**Y_pred “8“**
Y_echt für “3“	1	0	4	4	0	0
Y_echt für “4“	0	7	28	15	1	0
Y echt für “5“	0	4	163	120	8	0
Y_echt für “6“	0	2	79	289	36	3
Y_echt für “7“	0	0	10	111	60	2
Y_echt für “8“	0	0	0	20	11	2

Konfusionsmatrix für Model: MLP(150,75,35) relu, adam, Scaled: True						
	Y_pred “3“	**Y_pred “4“**	**Y_pred “5“**	**Y_pred “6“**	**Y_pred “7“**	**Y_pred “8“**
Y_echt für “3“	0	0	4	5	0	0
Y_echt für “4“	0	17	16	18	0	0
Y echt für	0	7	168	108	12	0

“5“						
Y_echt für “6“	0	6	50	289	57	7
Y_echt für “7“	0	0	6	68	108	1
Y_echt für “8“	0	0	1	15	9	8

1.1.4.1 Verfeinerung mit Kreuzvalidierung

Kreuzvalidierung K-Nearest Neighbor(KNN)

Die Anwendung von Cross-Validation ermöglicht eine robustere Bewertung des Modells, indem der Datensatz in mehrere Teile aufgeteilt wird, und das Modell auf verschiedenen Teilmengen trainiert und getestet wird. Dies führt zu einer besseren Schätzung der Modellleistung und hilft, Overfitting zu vermeiden.

Modell	**Die Accuracy- Scores von 5 Folds**	**Mittlerer Score**
KNN, no scale	[0.374, 0.215, 0.229, 0.335, 0.27]	0.279
KNN, scaled	[0.552, 0.554, 0.519 ,0.566 ,0.567]	0.34
KNN, no scale, grouped (to 3 classes)	[0.492, 0.457, 0.46, 0481 0.465]	0.471
KNN, scale, (to 3 classes)	[0.56. 0.494. 0.56, 0.50, 0.557,]	0.534

KNN, no scale: Der durchschnittliche Score über 5 Folds beträgt 0.279. Dies deutet darauf hin, dass das Modell im Durchschnitt eine geringe Genauigkeit aufweist.

KNN, scaled: Der durchschnittliche Score über 5 Folds beträgt 0.34. Im Vergleich zu "no scale" zeigt die Skalierung eine leichte Verbesserung der Leistung.

KNN, no scale, grouped: Der durchschnittliche Score über 5 Folds beträgt 0.471. Die Berücksichtigung der Gruppierung zu 3 Klassen verbessert die Genauigkeit im Vergleich zu "no group".

KNN, scale, grouped: Der durchschnittliche Score über 5 Folds beträgt 0.534. Die Kombination aus Skalierung und Gruppierung führt zu einem weiteren Anstieg der Genauigkeit.

Ergebnisse zum Classification report der Cross-Validation

Model	Scaled	Grouped (3 classes)	Accuracy	Precision	Recall	F1-Score	Balanced Accuracy	Weighted Recall	Relaxed Accuracy
KNN	no	no	0.463	0.256	0.232	0.237	0.232	0.463	0.463
KNN	yes	no	0.540	0.31	0.282	0.291	0.282	0.54	0.54
KNN	no	yes	0.838	0.822	0.71	0.751	0.71	0.838	0.838
KNN	yes	yes	0.870	0.849	0.757	0.794	0.757	0.87	0.87

KNN, no scale: Der unskalierte Ansatz zeigt niedrige Werte für Precision, Recall und F1-Score.

KNN, scaled: Die Skalierung verbessert Precision, Recall und F1-Score im Vergleich zu "no scale".

KNN, no scale, grouped: Dieser Ansatz zeigt eine deutliche Verbesserung der Genauigkeit, Precision, Recall und F1-Score. Die Gruppierung trägt dazu bei, die Leistung zu stabilisieren.

KNN, scale, grouped: Die beste Leistung wird mit Skalierung und Gruppierung erreicht. Höhere Werte für alle Metriken zeigen eine verbesserte Modellgenauigkeit.

Zusammenfassend lässt sich sagen, dass die Kombination aus Skalierung und Gruppierung zu den besten Ergebnissen führt, und die Gruppierung allein bereits einen positiven Einfluss hat.

Multilaterales Perceptron (MLP)

Tabelle : Ergebnis der 5-fold Kreuzvalidierung

Modell	**Die Accuracy- Scores von 5 Folds**	**Mittlerer Score**
MLP(11), relu, adam	[0.528 0.564 0.551 0.561 0.549]	0.551
MLP(22) relu, adam	[0.552 0.554 0.519 0.566 0.567]	0.552
MLP(150,75,35) relu, adam	[0.64 0.602 0.58 0.625 0.619]	0.613
MLP(300,150,75,35) relu, adam	[0.505 0.515 0.464 0.48 0.507]	0.494
MLP(400,150,75,35) relu, adam	[0.616 0.636 0.605 0.625 0.616]	0.62
MLP(150,75) relu, adam	[0.648 0.626 0.593 0.61 0.616]	0.619
MLP(150,75) relu, lbfgs	[0.597 0.61 0.603 0.605 0.639]	0.611

Die Durchschnittlichen Scores der Kreuzvalidierung zeigen, dass die MLPs nur eine geringe Genauigkeit(accuracy) bei der Vorhersage erzielten.

1.1.5 Gridsearch

Beschreibung und Einschätzung der Ergebnisse

KNN Gridsearch

Die Verwendung von GridSearch ermöglicht die systematische Durchsuchung verschiedener Hyperparameterkombinationen, um die optimalen Parameter für das Modell zu finden. Dies trägt dazu bei, die Modellleistung weiter zu verbessern, indem die besten Konfigurationen für das spezifische Problem identifiziert werden.

Model	Scaled	Grouped (3 classes)	Weights	Accuracy	Precision	Recall	F1-Score	Balanced Accuracy	Weighted Recall	Relaxed Accuracy
KNN	no	no	uniform	0.462	0.321	0.279	0.289	0.279	0.462	0.462
KNN	yes	no	uniform	0.537	0.413	0.347	0.367	0.347	0.537	0.537
KNN	no	no	distance	0.561	0.338	0.308	0.317	0.359	0.561	0.561
KNN	yes	no	distance	0.787	0.532	0.427	0.455	0.427	0.627	0.627
KNN	no	yes	uniform	0.654	0.503	0.458	0.469	0.458	0.654	0.654
KNN	yes	yes	uniform	0.721	0.596	0.528	0.547	0.528	0.721	0.721
KNN	no	yes	distance	0.719	0.595	0.561	0.575	0.561	0.719	0.719
KNN	yes	yes	distance	0.770	0.695	0.62	0.648	0.62	0.77	0.77

Skalierung scheint in einigen Fällen einen positiven Einfluss zu haben, während bestimmte Konfigurationen von Grid Search nicht signifikante Verbesserungen zeigen. Weitere Optimierungen könnten erforderlich sein.

Durch die Anwendung von GridSearch kann das Risiko von Overfitting verringert werden, da die Hyperparameter auf Basis einer systematischen Suche und Validierung ausgewählt werden. Das Modell könnte robuster und besser auf unbekannte Daten reagieren. Es bietet also eine solide Grundlage, für weitere Optimierung.

Insgesamt ist die Wahl zwischen den Modellen abhängig von den spezifischen Anforderungen des Anwendungsszenarios. Wenn die Stabilität der Ergebnisse und die Vermeidung von Overfitting entscheidend sind, könnte GridSearch trotz etwas niedrigerer Werte in bestimmten Metriken bevorzugt werden.

SVM

Das GriedSearch für den Algorithmus SVC mit Kernel rbf wurde mit folgenden Parametern ausgeführt:

```
search_params_svc_rbf = [ (SVC(kernel='rbf'),
                           {'svc__C': [1, 10, 100],
                            'svc__gamma': ['scale', 'auto', 1, 2],
                           },
                           MinMaxScaler()
                          ),
                          (SVC(kernel='rbf'),
                           {'svc__C': [1, 10, 100],
                            'svc__gamma': ['scale', 'auto', 1, 2],
                           },
                           StandardScaler()
                          )
                         ]

iterate_search_params_svc_rbf = [search_best_parameters(estimator, grid_params, scaler) for estimator,
                                 grid_params, scaler in search_params_svc_rbf]
```

Das Endergebnis wurde in einen Data Frame geschrieben und ausgegeben:

```
best_parameters = pd.DataFrame(iterate_search_params_svc_rbf)
best_parameters
```

	Model	Best Parameters	Scaler	Test Relaxed Accuracy
0	SVC	{'svc__C': 10, 'svc__gamma': 'scale'}	MinMaxScaler()	0.943
1	SVC	{'svc__C': 10, 'svc__gamma': 'scale'}	StandardScaler()	0.948

Eine detaillierte Sicht auf die CV bei den Läufen mit dem StandardScaler zeigt folgendes Bild:

	param_svc__gamma	params
0	scale	{'svc__C': 1, 'svc__gamma': 'scale'}
1	auto	{'svc__C': 1, 'svc__gamma': 'auto'}
2	1	{'svc__C': 1, 'svc__gamma': 1}
3	2	{'svc__C': 1, 'svc__gamma': 2}
4	scale	{'svc__C': 10, 'svc__gamma': 'scale'}
5	auto	{'svc__C': 10, 'svc__gamma': 'auto'}
6	1	{'svc__C': 10, 'svc__gamma': 1}
7	2	{'svc__C': 10, 'svc__gamma': 2}
8	scale	{'svc__C': 100, 'svc__gamma': 'scale'}
9	auto	{'svc__C': 100, 'svc__gamma': 'auto'}
10	1	{'svc__C': 100, 'svc__gamma': 1}
11	2	{'svc__C': 100, 'svc__gamma': 2}

	split0_test_score	split1_test_score	split2_test_score
0	0.952	0.949	0.954
1	0.939	0.941	0.940
2	0.948	0.944	0.949
3	0.952	0.949	0.953
4	0.954	0.949	0.957
5	0.939	0.939	0.941
6	0.954	0.952	0.954
7	0.954	0.949	0.952
8	0.954	0.939	0.938
9	0.946	0.946	0.952
10	0.957	0.948	0.952
11	0.958	0.949	0.950

	split3_test_score	split4_test_score	mean_test_score	std_test_score
0	0.948	0.967	0.9540	0.006841
1	0.944	0.944	0.9416	0.002059
2	0.943	0.953	0.9474	0.003611
3	0.945	0.959	0.9516	0.004630
4	0.955	0.962	0.9554	0.004224
5	0.946	0.949	0.9428	0.004020
6	0.950	0.963	0.9546	0.004454
7	0.954	0.968	0.9554	0.006560
8	0.939	0.972	0.9484	0.013215
9	0.945	0.955	0.9488	0.003970
10	0.950	0.960	0.9534	0.004454
11	0.945	0.962	0.9528	0.006242

Wir sehen, dass die Scores über die verschiedenen Folder ziemlich stabil sind, was auch aus den Standardabweichungen abzulesen ist. Die etwas größere Abweichung ist zudem bei einer der niedrigst bewerteten Parameterkombinationen aufgetreten:

```
    rank_test_score
0                 4
1                12
2                10
3                 7
4                 1
5                11
6                 3
7                 1
8                 9
9                 8
10                5
11                6
```

Wir können, in Anbetracht der besprochenen Qualitätseinschränkungen des untersuchten Datasets, zufrieden sein mit den vorläufig erzielten Ergebnissen (mehr dazu im Kommentar des Endergebnisses).

Multilaterales Perceptron MLP

Tabelle: Gridsearch für den MLPClassifier

Verwendete Daten	**Vordefinierte Parameter für das GridSearch**	**Bestes Ergebnis des Grid_Search**	**Ergebnis für einen Test-Durchlauf (gemittelte Scores):**
Originaldaten (11 Features und 0-10	param_grid= {'hidden_layer_sizes':	{'activation': 'relu', 'alpha': 0.05,	**Training:** Accuracy : 0.747

Klassen), Standard-skaliert	[(200,100),(22),(175,75,35)],'activation': ['tanh','relu'], 'learning_rate': ['constant', 'adaptive'], 'tol': [0.001,0.01], 'alpha': [0.0001,0.05],'max_iter': [1000,2000, 10000]}	'hidden_layer_sizes': (175, 75, 35), 'learning_rate': 'constant', 'max_iter': 10000, 'tol': 0.01}	Balanced Accuracy: 0.472 Genauigkeit : 0.747 Trefferquote: 0.747 **Test:** Accuracy : 0.742 Balanced Accuracy : 0.463 Genauigkeit: 0.742 Trefferquote: 0.742 **Kein Overfitting!**
Entfernen von “Residual_sugar”, “free_sulfur_dioxid” und pH” als Features, sowie Klassenzusammenle-gung in drei Klassen und standard skaliert	param_grid= {'hidden_layer_sizes': [(8,3),200,(100,50,20),(175,75,35)], 'activation': ['tanh','relu'], 'learning_rate': ['constant'], 'tol': [0.0001,0.001,0.01], 'solver': ['adam'],'alpha': [0.0001,0.05],'max_iter': [1000, 2000,10000] }	{'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (175, 75, 35), 'learning_rate': 'constant', 'max_iter': 1000, 'solver': 'adam', 'tol': 0.001}	**Training:** Accuracy : 0.798 Balanced Accuracy: 0.470 Genauigkeit: 0.798 Trefferquote: 0.798 **Testmenge:** Accuracy : 0.761 Balanced Accuracy: 0.444 Genauigkeit: 0.761 Trefferquote: 0.761 **Kein Overfitting!**

1.1.6 Benutzung einer Ensemble-Methode

Aus zeitlichen Gründen wurde keine Ensemble Methode angewandt und bietet Raum für weitere Optimierung.

AB: Es wurde mit Ensemble-Methoden experimentiert, wie in einem vorherigen Abschnitt (1.2.5) berichtet. Mit Random Forest standen wir in einer Überanpassungssituation bei den Trainingsdaten - bereits die Decision Trees zeigten dieses Verhalten.

Ferner wurden GradientBoosting und AdaBoost, aus Zeitgründen leider nur kurz, ausprobiert. Die Ergebnisse schienen vielversprechend (in den Bildern unten sind die Accuracy-Scores - mit MinMaxScaler und StandardScaler - für das Testset der Rotweine) :

```
Best Hyperparameters: {'gradientboostingclassifier__learning_rate': 0.1, 'gradientboostingclassifier__max_depth': 6
, 'gradientboostingclassifier__n_estimators': 100}
Test Accuracy with Best Model: 0.71
Best Hyperparameters: {'gradientboostingclassifier__learning_rate': 0.1, 'gradientboostingclassifier__max_depth': 6
, 'gradientboostingclassifier__n_estimators': 100}
Test Accuracy with Best Model: 0.71
```

```
Best Hyperparameters: {'adaboostclassifier__learning_rate': 1, 'adaboostclassifier__n_estimators': 400}
Test Accuracy with Best Model: 0.66
Best Hyperparameters: {'adaboostclassifier__learning_rate': 1, 'adaboostclassifier__n_estimators': 400}
Test Accuracy with Best Model: 0.60
Best Hyperparameters: {'adaboostclassifier__learning_rate': 1, 'adaboostclassifier__n_estimators': 400}
Test Accuracy with Best Model: 0.61
Best Hyperparameters: {'adaboostclassifier__learning_rate': 0.1, 'adaboostclassifier__n_estimators': 400}
Test Accuracy with Best Model: 0.62
Best Hyperparameters: {'adaboostclassifier__learning_rate': 0.1, 'adaboostclassifier__n_estimators': 100}
Test Accuracy with Best Model: 0.57
```

Die 'weak learner' in der obigen Anzeige für AdaBoost waren:

```
dt  = DecisionTreeClassifier(max_depth=4, max_leaf_nodes=12, random_state=random_state)
gnb = GaussianNB()
kn  = KNeighborsClassifier(leaf_size=2, n_neighbors=64)
sgd = SGDClassifier(loss='log_loss', max_iter=10000)
```

1.1.7 Kommentar des Endergebnisses

KNN Endergebnis

Bei KNN werden die optimalsten, das heißt einerseits solide, andererseits hohe Scores, im Rahmen von GridSearch und Distanzgewichtung, sowie Klassenzusammenlegung erzielt, mit einer Balanced Accuracy von 62%, sowie einer Accuracy von 77%. Die Werte in von GridSearch sind zwar niedriger als bei den gruppierten Modellen ohne Cross Validation oder GridSearch, allerdings deshalb, weil GridSearch dazu neigt, die besten Hyperparameter zu finden, um die Leistung des Modells zu verbessern. Auch wenn die Werte niedriger erscheinen, könnte das Modell besser auf andere Datensätze oder zukünftige Daten generalisieren.

Für eine weitere Verbesserung könnten die Hyperparameter (n_neighbors und weights) weiter optimiert werden. Weiterhin könnte Feature Engineering stattfinden, etwa durch neu abgeleitete Merkmale. Es könnten auch Klassen ausbalanciert werden, mittels Oversampling oder Undersampling, um seltene Klassen zu erkennen. Ensemble-Methoden wären eine weitere Möglichkeit, wie Random Forest oder Gradient Boosting, für verbesserte Vorhersagen. Eventuelle lassen sich noch Fehler in den Daten identifizieren, welche behoben werden können und so die Algorithmen verbessern. Eine andere Möglichkeit wäre auch eine weitere Anpassung der Verteilung, wie es beispielsweise mit der Klassenzusammenlegung gemacht worden ist. Zusätzlich könnte die Cross-Validierung ausgeweitet werden, indem man die Anzahl der Folds etwa noch weiter ausbaut. Schließlich kann durch die Kombination dieser genannten, potentiellen Verbesserungsmöglichkeiten, die Leistung der Vorhersagen verbessert werden.

Es zeigt, dass das Modell eine vernünftige Ausgewogenheit bei der Klassifizierung der Klassen erreicht hat. Zufriedenheit hängt jedoch von den spezifischen Anforderungen des Anwendungsfalls ab. Wenn dies für Ihr Anwendungsgebiet ausreichend genau ist, könnte man zufrieden sein.
Entscheidung für den Erfolg

Der Erfolg wurde wahrscheinlich durch mehrere Entscheidungen beeinflusst, darunter die Verwendung der skalierten Daten, die Anpassung an die Distanzgewichtung und die Berücksichtigung der Gruppierung bei der Aufteilung der Daten. Diese Entscheidungen haben dazu beigetragen, die Leistung des Modells zu verbessern.

Mögliche Gründe für besseren Erfolg könnten jedoch unzureichende Datenqualität, fehlende relevante Features oder die Notwendigkeit einer fortschrittlicheren Modellierung sein.

Das Vertrauen in das Ergebnis hängt von der sorgfältigen Evaluierung und Validierung des

Modells ab. Wenn wie in diesem Fall eine umfassende Validierung durchgeführt worden ist, einschließlich Kreuzvalidierung und Grid-Suche, und die Ergebnisse konsistent sind, können Sie tendenziell Vertrauen in das Modell haben. Es ist jedoch wichtig zu beachten, dass die Leistung auf zukünftigen Daten variieren kann.

Experimentieren Sie mit fortgeschritteneren Algorithmen wie Support Vector Machines (SVM), Random Forests oder Gradient Boosting.
Testen Sie fortgeschrittenere Techniken des Feature Engineerings, um die Informationen aus den Daten besser zu extrahieren.
Berücksichtigen Sie neuronale Netzwerke, insbesondere wenn der Datensatz groß und komplex ist.
Untersuchen Sie die Effekte verschiedener Preprocessing-Techniken und Datenreduktionsmethoden.
Es ist wichtig zu betonen, dass der Erfolg von maschinellem Lernen stark von der Natur der Daten und des spezifischen Anwendungsfalls abhängt. Daher ist es ratsam, verschiedene Ansätze zu explorieren und ihre Auswirkungen sorgfältig zu evaluieren.

MLP
Multilaterale Perceptrone sind, wie diese Arbeit ebenfalls zeigt, eine geeignete Methode, um die Qualität von Weißweinen für den verwendeten Datensatz vorherzusagen, wobei die Genauigkeit(Precision) und Trefferquote(Recall) im besten Ergebnis aus dem GridSearch auf 76,1% für beide bei der Testmenge beträgt. Es trat auch kein Overfitting auf. Für die Klassenzusammenlegung eignet sich leider nicht der Relaxed Accuracy Score von Alexandre zur Beurteilung.
Es ist jedoch nicht leicht, die künstlichen neuronalen Netzwerke für diesen Datensatz genau einzustellen. Jedoch hat es sich gezeigt, dass multilaterale NNs mit mehreren Hidden-Layers für diesen Datensatz geeigneter erscheinen. Sie haben den Vorteil, komplexe Zusammenhänge gut zu erkennen und mit dem Rauschen von Daten zurechtzukommen. Ein dreischichtiges MLP-Modell, das mit dem Package Tensorflow-Kiras erstellt wurde, konnte für den Weißwein-Datensatz eine Accuracy von 79% erreichen (Daten hierzu fehlen in der Arbeit).

Unsere Ergebnisse müssen aus der Perspektive einer Voranalyse angesehen werden, wie mit der Auftraggeberin abgestimmt. Unter Berücksichtigungen der Rahmenbedingungen, darunter: stark unbalancierte Klassen; insgesamt zu wenig Datenpunkte; sowie eine unterdimensionierte Gruppe in Anbetracht des knappen Zeitraums und der Breite der Thematik, haben wir eine sehr gute Grundlage geschaffen für die Weiterführung des Projekts und Verfeinerung des Endergebnisses.

Unter anderem sollen die neu definierten Scores weiter analysiert und optimiert werden:

Weighted Recall wurde durch Christopher angeregt. Die Idee ist vielversprechend, die

Gewichte müssen lediglich erst besser kalibriert werden (nach Trial and Error und in Abstimmung mit dem Auftraggeber).

Die Motivation für die Relaxed Accuracy war mehrfach begründet:

- zugrunde liegt u. a. die Annahme, dass die Verkoster, so professionell und kompetent sie sein mögen (und das betrifft die Weinhändler und Konsumenten genauso), nicht immer eindeutig zwischen einer Benotung 5 oder 6 (beispielsweise) unterscheiden können.

- Zudem erlaubt sie, mit Regressionsalgorithmen zu arbeiten, was jedenfalls versucht werden sollte.

Es ist bereits vorgesehen, wie mit der Auftraggeberin besprochen, Varianten dieser letzten Metrik mit Wahrscheinlichkeitsgewichtungen zu konzipieren und zu testen.

Wie in der Einleitung ausführlich erläutert, sind die gewählten physikalischen und chemischen Eigenschaften zur Qualitätsbestimmung durch Verkostung geeignet, erlauben aber in einer genaueren Betrachtung kein differenzierteres Bild der Qualität eines Weines. Wein zeichnet sich durch eine Vielzahl von chemischen Stoffen aus, die erst in ihrer Gesamtheit eine feinere Aussage zur Qualität eines Weines zulassen. In einem anderen Datensatz zu Weinen werden gesundheitsförderliche Komponenten wie Flavanoide und Polyphenole, Aminosäuren wie Prolin, und andere gebundene Säuren wie Apfelsäure oder gar Tannine (glykolisierte Gallussäureester) mit berücksichtigt. Es wäre außerdem empfehlenswert- in Anbetracht der unausgewogenen Klassen-, sich Gedanken über die Methode zur Qualitätsbeurteilung zu machen, was jedoch ein schwieriges Unterfangen darstellt.
Interessant wäre es auch, Rot- und Weißweine am Ende auf der Grundlage ihrer Feature-Importance (e.g. aus Random Forest oder sklearn feature_importances_) zu vergleichen und als Histogramm darzustellen.

1.1.8 Beschreibung von python-Funktionen oder Klassen, die sich gut für eine Wiederverwendbarkeit eignen

Algorithmus-Klassen, wie etwa KNN. Wenn eine Klasse zentral vorliegt, kann sie mehrmals instanziiert werden, etwa durch andere Hyperparameter die notwendig sind. Anstatt beispielsweise mehreren Code zu schreiben, der das selbe tut, nur mit anderen Parametern. Weiterhin reduziert es an sich viel Code, da zentral auf den Algorithmus zugegriffen wird.

Scaler-Klassen, wie etwa StandardScaler. Ein Scaler kann an vielen Stellen zum Einsatz kommen, etwa in verschiedenen Pipelines oder Programmiercode. Liegt eine zentrale Klasse vor, kann auch diese zentral genutzt werden.

Metriken-Klassen, zur Ausgabe diverser Metriken. Auch hierbei hilft es immens eine Klasse zu haben, weil auch hier nachfolgend viel Code erspart bleibt, da schließlich eine zentrale Metrik Klasse vorliegt, welche von allen notwendigen Codes oder Ergebnissen angesteuert werden kann.

Pipelines helfen auch immens, bereits vorhandene Klassen-Instanzen in verschiedenen

Szenarien, verschieden zusammenzusetzen, z.B. einmal KNN mit StandardScaler, einmal mit MinMax usw.

Es wurden diverse Funktionen, Pipelines und Bausteine geschrieben, die sich sehr gut in Klassen einbetten lassen und die beispielsweise beim GridSearch sowie den automatisierten Vergleich zwischen verschiedenen Algorithmen, Skalierungen und Anzahl PCA-Komponenten zur Anwendung kamen.