Projekt: Pilzbestimmung

2.1. Alle Teilnehmer und ihre Arbeitsschwerpunkte

Gunther Adelt

Matthias Junker

Nicolas Potocki

Gabriela Rotari

Thorsten Tiede

Alle Aufgaben wurden zusammen bearbeitet.

2.2. Klärung der Aufgabenstellung

Ziel der Untersuchung
Unterscheidung zwischen essbaren und giftigen Pilzen. Eine Identifizierung von Pilzen als 100 % essbar. Es sollen Fehler zweiter Art ausgeschlossen werden, d. h. es darf kein giftiger Pilz als essbar, jedoch essbare Pilze dürfen als giftig klassifiziert werden.

Arbeits-Hypothese vor Beginn der Untersuchung
Random-Forest ist der beste Klassifizierer für dieses Projekt, laut der durchgeführten Studie „Mushroom data creation, curation, and simulation to support classification tasks“, Scientific Reports volume 11, Article number: 8134 (2021), Dennis Wagner, Dominik Heider & Georges Hattab

Projekthypothese:
Wir können den Score der ohne Random-Forest erreicht wurde mit ensemble learning ohne Random-Forest verbessern (Score > 0.8)

Qualitätskriterien (Welche Sorte Score? Wie hoch muss er am Ende sein?)

Precision-Score möglichst 1,0 bzw. in dem Ergebnis sollen keine giftigen Pilze als essbar identifiziert werden.

Welche Sorte von Machine Learning-Verfahren?

- Logistic Regression → *sklearn.linear_model.LogisticRegression*
- Naive Bayes → *sklearn.naive_bayes.GaussianNB*
- KNN → *sklearn.neighbors.KNeighborsClassifier*
- SVC → *sklearn.svm.SVC*
- MLP → *sklearn.neural_network.MLPClassifier*

2.3. Beschreibung der Daten

Datenquelle:

Philipps Universität Marburg
Der untersuchte Datensatz basiert auf einem Stammdatensatz mit 173 Zeilen (173 unterschiedliche Pilzarten) die durch Simulation auf einen Datensatz mit 61.069 hypothetischen Pilzen hochgerechnet wurde.

Wie leicht ist die Datenbeschaffung?
Daten lagen in csv-Format vor. Da es in den generierten Daten mehrere Spalten gab, in denen sehr viele Werte fehlten, haben die Ersteller der Studie für die Spalten, in denen mehr als 50 % der Werte fehlten, diese Spalten komplett aus dem Datensatz entfernt (Fehlende Daten in Prozent: „stem-root“ ca. 84 %, „stem-surface“ ca. 62 %, „veil-type“ ca. 95 %, „veil-color“ ca. 88 % und „spore-print-color“ ca. 90 %). Die fehlenden Werte der anderen Spalten wurden in der Studie spaltenweise durch Ersetzung des häufigsten Vorkommen aufgefüllt: *Simple Imputer(strategy = 'most frequent').*

(Fehlende Daten in Prozent: „cap-surface“ ca. 23 %, „gill-attachment“ ca. 16 %, „gill-spacing“ ca. 41 %, „ring type“ ca. 4 %)

Größe des Datensatzes (Zeilen, Spalten, Gigabyte ...)

Größe mit fehlenden Daten:
61.069 Zeilen, Zielspalte + 20 Spalten (3 metrische Spalten, 17 nominale Spalten), 13,4 MB

Größe mit angeglichenen Daten:
61.069 Zeilen, Zielspalte + 20 Spalten (3 metrische Spalten, 12 nominale Spalten)

Maximum, Minimum, Mean …, (bei den wichtigsten Spalten)
keine Angaben, da hier keine wichtigen Spalten identifiziert werden konnten

gibt es fehlende Werte? Passen die Größenordnungen zusammen?

s. o.

Datentypen: String, Zahl, Datum?

s. o., (3 Spalten mit Zahlenwerten, 17 Spalten mit nominaler Einteilung)

Korrelationen zwischen einzelnen Spalten?

In den drei metrischen Spalten ist keine Korrelation vorhanden

2.4 Beschreibung der Datenvorbereitung

Welche Methoden wurden bezüglich Imputing und Skalierung angewandt?

Imputing:

Da eine Vorauswahl und Entfernung von Spalten mit fehlenden Werten > 50 % schon vorgenommen wurden (s. o.), sind alle anderen Spalten mit fehlenden Werten mittels Imputer und Strategie „most frequent“ ersetzt worden.
Allerdings gab es für die Spalte „gill-attachment“ eine spezielle Transformierung. Dort wurden alle fehlenden Werte mit einem von uns selbst eingefügtem zusätzlichen/neuem Attribut „m“ (missing) ersetzt. Diese Strategie erschien uns nach Sichtung der Daten von Vorteil, um die Daten durch den alternativen „most frequent“-Wert nicht zu verfälschen.

Skalierung:
Bei den drei metrischen Spalten wurde ein Min-Max-Scaler verwendet.

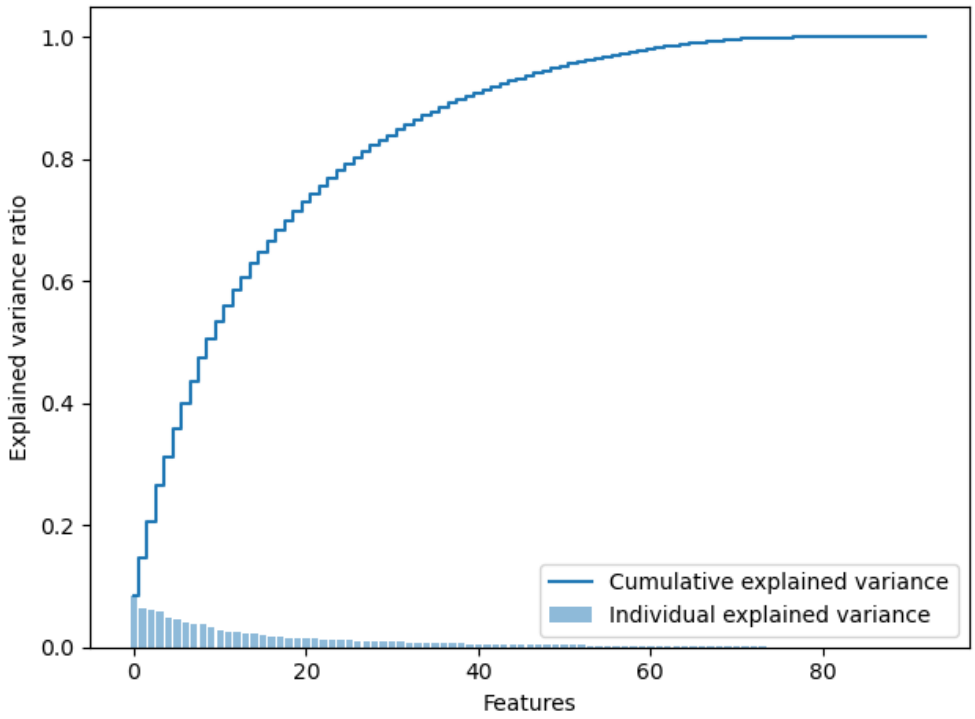
Wurden Features weggelassen? (intuitiv oder mit sklearn.FeatureSelection)?
s.o.

Wurden aus den Daten zusätzliche Spalten errechnet?

Nein, da wir fast nur Spalten mit nominalen Merkmalen haben, konnten wir daraus keine weiteren errechnen. Alle nominalen Spalten wurden mittels One-Hot-Encoding in maschinenlesbare Form gebracht. Die drei metrischen Spalten wurden auch nicht weiter in Betracht gezogen, da nach eigenen Erfahrungen die metrische Einordnung von Pilzen bei der Analyse eine untergeordnete Rolle spielt. Wir denken, es hat damit zu tun, dass die Größe eines Pilzes oft mit dem Alter zu tun hat. Ein nicht erfahrener Pilzsucher würde, wenn er das Alter (junger oder ausgewachsener Pilz) nicht eindeutig bestimmen kann, falsche Rückschlüsse auf die Spezies und damit auf essbar/giftig ziehen. Bei Wanderungen mit erfahrenen Pilzsuchern wird oftmals ausdrücklich darauf hingewiesen, dass z. B. im Falle der Unterscheidung vom Parasol-Pilz (Macrolepiota procera) zu anderen verwandten (giftigen) Schirmlingen, die kleiner sind, man sich nie auf den Schirmdurchmesser verlassen kann und andere Unterscheidungskriterien viel wichtiger sind.

Wurde eine PCA durchgeführt? Wenn ja, mit welcher Methode? Welcher Prozentsatz der Erklärungskraft wurde beibehalten?

Es wurde eine PCA mit dem MLP-Klassifizierer durchgeführt. Mit dem zuvor durchgeführten GridSearch wurden geeignete Parameter auf eine kleinere Menge Parameter begrenzt und mit dieser neuen Menge ein neues Gridsearch mit vorheriger PCA durchgeführt. Mit zwei verschiedenen PCA-Instanziierungen wurde jeweils ein identisches Gridsearch-durchlaufen. Das Kriterium für die PCA war eine Halbierung sowie eine Viertelung der Featurespalten-Anzahl, und nicht in erster Linie der erreichbare Informationsgehalt. Von 93 ursprünglichen Features wurde mit der PCA auf 47 bzw. 23 Features reduziert.



Folgend aufgelistet die Einstellungen und Ergebnisse der Durchläufe mit
a) 23 Features,
b) 47 Features und
c) allen Features (93)

pca = PCA(n_components=23)
mlp = MLPClassifier(hidden_layer_sizes=(50,10), random_state=42, max_iter=100000, tol=0.00001)
Test Score: 0.9948132099008601
test number: 15231, true: 8464, false: 6767
test number: 15231, _11: 8464, _10: 55, _01: 24, _00: 6688

pca = PCA(n_components=47)
mlp = MLPClassifier(hidden_layer_sizes=(50,10), random_state=42, max_iter=100000, tol=0.00001)
Test Score: 0.9987525441533714
test number: 15231, true: 8486, false: 6745
test number: 15231, _11: 8486, _10: 17, _01: 2, _00: 6726

ohne PCA
mlp = MLPClassifier(hidden_layer_sizes=(50,10), random_state=42, max_iter=100000, tol=0.00001)
Test Score: 0.9998686888582496
test number: 15231, true: 8488, false: 6743
test number: 15231, _11: 8488, _10: 2, _01: 0, _00: 6741

Durch die PCA wird zwar der Test-Score nicht wesentlich beeinflusst, jedoch sind mehr Predicts im Vergleich zur Testmenge falsch positiv bzw. falsch negativ klassifiziert, was bei der hohen Priorisierung auf eine richtige Vorhersage inaktzeptabel ist. Im Detail kann die PCA potenziell die Feature-Anzahl verringern, sie ist allerdings generell für die hier notwendige Präzision und Richtigkeit bei der Klassifizierung sehr vorsichtig einzusetzen. Im finalen Scoring haben wir daher auf eine PCA verzichtet.

2.5 Anwendung verschiedener Algorithmen

2.5.1 Begründung für die Wahl der Algorithmen

Die Autoren der Studie, welche die Daten erhoben haben, haben vier Klassifizierer angewandt, darunter Logistische Regression und einen Naive-Bayes.
Deshalb wurden diese beiden Algorithmen als Vergleich verwendet.
Ein erster Versuch mit den anderen beiden verwendeten Klassifizierern (Random Forest und LDA) führte zu keinem zufrieden stellenden Ergebnis bei langen Laufzeiten und wurden daher nicht weiterverfolgt.

Weiteren haben wir einen Support Vektor Klassifizierer, einen K-Nearest Neighbor Klassifizierer sowie ein Mehrschichtiges Perceptron (MLP) verwendet.
Diese Algorithmen wurden gewählt, da sie als State-of-the Art Klassifizierer gelten.
Wir wollten zeigen, dass diese Methoden bessere Ergebnisse liefern als die von den Autoren eingesetzten.

Berechnung des Scores auf Trainingsmenge und Testmenge

Klassifizierer	Trainings-Score	Test-Score	Cross-Val Score (mean)	Größe Testmenge (absolut / % Gesamt)	Gesamt falsch vorher gesagt	% falsch vorher gesagt	Falsch-Negative (Fn)	Fn Anteil in %	Falsch-Positive (Fp)	Fp Anteil in %
Support Vector Machine	0.9986	0.9982	0.9981	21324	38	0.18	0	0	38	0.18
Logistic Regression	0.6512	0.6510	0.6497	21324	7442	34.9	7402	34.71	40	0.19
Knearest Neighbors	1.0	0.9998	0.9998	21324	4	0.02	0	0	4	0.02
Gaussian Naive Bayes	0.7294	0.7265	0.7262	21324	5830	27.34	4292	20.13	1538	7.22
Multi-Layer Perceptron	1.0	0.9999	0.9996	21324	1	0.01	0	0	1	0.01
Ensemble (Hard-Voting)	1.0	0.9999	0.9996	21324	1	0.01	0	0	1	0.01
Ensemble (Soft-Voting)	1.0	0.9999	0.9996	21324	1	0.01	0	0	1	0.01
Ensemble (Ohne MLP, Hard-Voting)	0.8054	0.8062	0.8029	21324	4132	19.38	4132	19.38	0	0
Ensemble (Ohne MLP, Soft-Voting)	0.9999	0.9998	0.9997	21324	4	0.02	3	0.01	1	0.01

Anmerkungen ob, Overfitting vorliegt

Aufgrund der Tatsache, dass sich weder die Cross-Validierungs-Scores von den Trainings-Scores noch die Test-Scores von den Cross-Validierungs-Scores unterscheiden, gehen wir davon aus, dass kein Overfitting vorliegt.

2.5.1. Verfeinerung

Kreuzvalidierung mit sklearn.model_selection.cross_validate

Eine Kreuzvalidierung wird im „GridsearchCV“ durchgeführt, den wir für alle Algorithmen durchgeführt haben.
Es wurde im abschließenden Scoring der besten Algorithmen aus der Grid-Search nochmals eine Kreuzvalidierung mit scoring durchgeführt. Diese Scores sind in obiger Tabelle zu finden.

2.6 Gridsearch

Beschreibung und Einschätzung der Ergebnisse

Die einzeln durchgeführten Grid-Searches mit verschiedenen Parameter-Kombinationen brachten v.a. beim Multi-Layer Perceptron und Support Vector Machines deutliche Verbesserungen der Scores mit Default-Werten.
Beispielsweise konnte für die Support Vector Machine der Score von 0,8322 mit den Parametern {C=1.0, gamma=‘scale', kernel='linear', tol=0.001} verbessert werden, auf 0,998 mit den Parametern {C=100.0, gamma='scale', kernel='rbf', tol=0.001}.
Dies lässt sich mit der Verschärfung des Regularisierungsparameters und dem Wechsel des Kernels von einem Linearen auf einen Kreis-basierten Kernel (rbf), der bei diesen hoch-dimensionalen Daten bessere Performance zeigt.
Die Ergebnisse sind allgemein als sehr gut einzuschätzen, da ein fast-perfekter Score ohne zu erkennendes Overfitting erzielt werden konnte.

2.7 Benutzung einer Ensemble-Methode

Beschreibung der Ergebnisse

Die Ensemble-Methode aus allen fünf Klassifizierern ergab keine Verbesserung gegenüber den einzelnen Klassifizierern.
Sowohl das Hard-Scoring, als auch das Soft-Scoring erzielten dieselben Scores wie das Multi-Layer Perceptron.
Weiterhin wurde ein Ensemble aus vier Klassifizierern erstellt, welches den MLP Klassifizierer nicht enthält.
Mit Hard-Voting erreicht das Ensemble ein Test-Score von 0,8, mit Soft-Voting von 0,99.
Dies zeigt, dass die Soft-Voting Methode die Nachteile der einzelnen Klassifizierer deutlich besser ausgleichen kann, als die Hard-Voting Methode. Dies spricht dafür, dass die Klassifizierer durchaus zu unterschiedlichen Ergebnissen kommen, bzw. die als sicher eingestuften Vorhersagen (individuell mit hoher Wahrscheinlichkeit) mitunter sich gut ergänzen. Eine detaillierte Analyse dieses Phänomens wäre wünschenswert, kann aus Zeitgründen im Rahmen dieses Projekts jedoch nicht erfolgen.

2.8 Kommentar des Endergebnisses

Kann man mit dem Score zufrieden sein?

Alle Klassifizierer erreichen Scores von über 0,99, außer der Naive Bayes und Logistische Regression mit einem Score von 0,72, bzw. 0,64. Mit den besten Klassifizierer-Parametern werden teilweise sogar Test-Scores von über 0,999 erreicht. In dieser Hinsicht sind alle Score-Ergebnisse mehr als zufriedenstellend.
Die Validierung von y-Test und y-Predict und einem Train-Test-Split von 0,35 ergibt mit den besten Klassifizierern eine falsch-positiv Rate von 1/15.000 in Bezug auf das Label „giftig“ und vice versa. Der MLP-Klassifizierer dominiert mit seinem fast optimalen Verhalten im Ensemble mit anderen Klassifizierern den Ensemble-Score. Ein Ensemble ohne den MLP-Klassifizierer reicht in Bezug auf Score und falsch-positiv-Rate an ihn im Soft Voting jedoch heran.

Welche Entscheidung brachte den Erfolg? Welcher Faktor verhinderte den Erfolg?

Letztendlich stellte sich durch eine Schnellanalyse des Datensatzes mit einigen gängigen Klassifizierern schnell heraus, dass ein hoher Score im Bereich von 0,99 erreichbar ist. Dies wird wahrscheinlich an den inhärenten Eigenschaften des Datensatzes liegen, der aus simulierten, hypothetischen Einträgen von Pilzen besteht, die wiederum auf der Basis von automatisch gecrawlten Daten aus einem digitalisierten Textbuch basieren. Das durch die Eigenschaften des Datensatzes allgemein ein hoher Score erreicht wurde ist an dieser Stelle eine Hypothese, die nicht im Detail untersucht wurde.

Kann man dem Ergebnis trauen? (Könnte man es auf zukünftigen Daten reproduzieren?)

Wünschenswert wäre eine Analyse des UCI-Pilzdatensatzes von 1987 gewesen. Dieser könnte als Testmenge dienen. So könnte untersucht werden, ob die Klassifizierer reproduzierbar mit neuen, unbekannten Datensätzen ähnlich gut funktionieren. Dies würde jedoch ein vorheriges Matching der Features erfordern, und dies konnte aus Zeitgründen nicht umgesetzt werden.

Mit welcher Methode sollte in Zukunft, wenn man mehr Zeit hat, noch ausprobieren?

Da das Multi-Layer Perceptron bereits alle bis auf einen Pilz korrekt klassifiziert, kann eine Verbesserung kaum mehr erzielt werden.

Denkbar wäre noch der Versuch des „Boostings“, um die wenigen falsch klassifizierten Daten spezifisch durch weitere Klassifizierer beurteilen zu lassen.

Eine weitere Grid-Search auf dem MLP ist eine weitere Möglichkeit, die verbleibenden falschen Vorhersagen zu korrigieren.

Außerdem wäre es interessant mit mehr Zeit einen Random-Forest auszuprobieren, um zu versuchen die Ergebnisse der Autoren der Studie zu replizieren.

2.9 [Beschreibung von Python-Funktionen oder Klassen, die sich gut für eine Wiederverwendbarkeit eignen]

Wir haben ein Modul „util“ erstellt, in dem drei Funktionen zusammengefasst werden:
1. remove_na_columns um die Spalten aus den Daten zu löschen, die mehr als 50% fehlende Daten enthalten
2. do_grid_search, um für einen Klassifizierer und das Parameter-Grid, die übergeben werden, die Grid-Search auszuführen
3. prep() zur Erzeugung der Preprocessing Piepline mittels ColumnTransformer, Pipeline, sowie SimpleImputer (2 Mal), OneHotEncoder und MinMaxScaler.

Dieses Modul „util“ haben wir dann in den zwei Hauptprogrammen „grid_search_main“ und „eval_best_classifier“ eingesetzt um die Daten einheitlich vorzubereiten und in ersterer die GridSearch durchzuführen.
Das Hauptprogramm „grid_search_main“ liest die Daten ein, bereitet die Daten vor und verwendet einen Klassifizierer für eine Grid-Search. Diesen Klassifizerer haben wir jeweils durch einen der Ausprobierten ersetzt, und haben das Hauptprogramm für verschiedene Parameter im parameter-grid ausgeführt.

Das Hauptprogramm „eval_best_classifier“ haben wir benutzt, um die besten Parameter-Kombinationen der einzelnen Klassifizierer einzeln und im Ensemble zu evaluieren und die Falsch-positiv und Falsch-negativ Rate zu berechnen.