Klassifikation von Pilzen – Essbar oder giftig?

Gegenstand dieser Projektarbeit ist der Vergleich verschiedener Klassifikationsalgorithmen bei der Erkennung von giftigen oder essbaren Pilzen. Hauptaugenmerk liegt hierbei auf der Präzision, mit der die ausgewählten Algorithmen die Pilze erkennen. Darüber hinaus soll auch ein Blick auf die Zeit geworfen werden, den die Modelle für Training und Evaluation benötigen.

Die Datenbasis stammt ursprünglich vom UCI Machine Learning Repository (Lichman, 2013). Sie umfasst eine Sammlung von 8124 Datensätzen, jeweils einer der beiden Klassen „essbar“ oder „giftig“ zugeordnet. Die Merkmale umfassen Angaben zu Aussehen, Geruch und Lebensraum der Pilze.

Die folgenden Abschnitte beschreiben das Vorgehen inklusive der eingesetzten Algorithmen und die Ergebnisse. Zum Abschluss wird ein Fazit gezogen.

1. Methoden

In diesem Abschnitt werden das allgemeine Vorgehen, die Datenbasis und die eingesetzten Algorithmen näher beschrieben.

1.1. Eingesetzte Bibliotheken

Eingesetzt wurden verschiedene Bibliotheken für Python 3. Das Lesen, Schreiben und Bearbeiten der Daten erfolgte mit Pandas 0.21.0. Die Modelle und Algorithmen wurden mit scikit-learn 0.19.0 trainiert und evaluiert. Auch ein Teil der Datenbereinigung und –Kodierung erfolgte mit dieser Bibliothek. Das Zeichnen der Diagramme erfolgte mit Matplotlib 2.0.2. Der Code zum Zeichner der Lernkurven wurde von http://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html#sphx-glr-auto-examples-model-selection-plot-learning-curve-py entnommen.

1.2. Datenbasis

Die Datenbasis enthält ausschließlich kategoriale Daten. Sie besteht aus 8124 Datensätzen, jeweils mit 23 Merkmalen, wovon ein Merkmal die Klassenzuordnung „class“ ist. „class“ hat zwei Ausprägungen, „e“ und „p“, die „essbar“ und „giftig“ entsprechen, es handelt sich hier um eine binäre Klassifikation. Da alle Daten kategorial sind, wurden sie in ganzzahlige Werte umkodiert. „e“ entspricht einer 0, „p“ einer 1.

Die Verteilung der Klassen ist relativ gleichmäßig mit circa 51,8% Auftreten der Klasse 0, während 48,2% der Datensätze mit einer 1 klassifiziert sind. Bei der weiteren Untersuchung der Datenbasis zeigt sich jedoch, dass 2480 Datensätze fehlende Werte in der Spalte „stalk-root“ haben. Bei der Behandlung der fehlenden Daten wurden zwei Ansätze angewandt:

1. Entfernen der Datensätze mit fehlenden Werten
2. Vorhersage der fehlenden Werte auf Basis der restlichen Werte

Durch das Entfernen der unvollständigen Datensätze schrumpft die Datenbasis auf nunmehr 5644 Datensätze. Auch die Verteilung der Klassen verschiebt sich dadurch, sodass 61,8% der Datensätze die Klasse 0 und 38,2% die Klasse 1 haben.

Aufgrund der Verschiebung der Verteilung der Klassen und der Verkleinerung der Datenbasis um circa 30% wurde auch der Ansatz der Vorhersage der fehlenden Werte getestet. Dies hat den Vorteil, dass alle Datensätze später zur Verfügung stehen, allerdings ist nicht gesichert, dass die Vorhersage auch korrekt ist. Dies lässt sich auch im Nachgang nicht verifizieren.

Zur Vorhersage der fehlenden Werte wurden die vollständigen von den unvollständigen Datensätzen getrennt und die Spalte der Klassen entfernt, um eine Beeinflussung zu verhindern, da dies später die Zielvariable sein wird. Das Merkmal „stalk-root“ wurde als Zielvariable festgelegt, da dessen fehlende Werte vorhergesagt werden sollen. Dann wurde ein K-nearest-neighbor Klassifizierer (KNN) auf den vollständigen Datensätzen trainiert und evaluiert. Die Evaluation fand auf 40% der Menge an vollständigen Datensätzen statt. Mit den optimierten Parametern wurden dann die fehlenden Werte in den unvollständigen Datensätzen vorhergesagt.

Ein KNN wurde gewählt, da er schnell und unkompliziert mit Merkmalen als Klassen arbeiten kann, die mehr als eine binäre Ausprägung haben, was auf das Merkmal „stalk-root“ zutrifft. Beim Training und der Evaluation auf den vollständigen Datensätzen erzielte das Modell eine Präzision von circa 0,994 bei der Vorhersage neuer Werte (Cross-Validation), weshalb man von einer zuverlässigen Vorhersage der fehlenden Werte ausgehen kann. Abbildung 1 zeigt die Lernkurve des Modells auf der Menge der vollständigen Datensätze. Der Score stellt den durchschnittlichen Wert von 100 Evaluationsläufen dar.

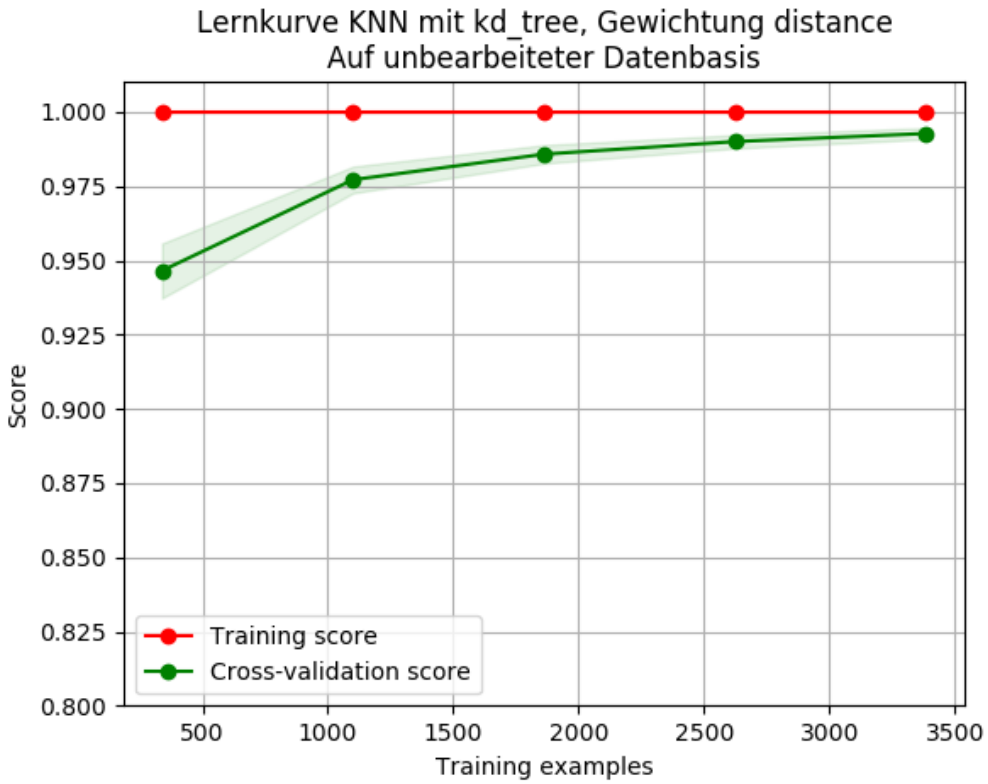

Abbildung 1. Präzision auf Trainings- und Testdaten eines KNN zur Vorhersage fehlender Werte.

Als Ergebnis stehen für die weitere Verarbeitung durch die Algorithmen zwei Datenbasen zur Verfügung, eine ohne unvollständige Datensätze und eine mit vorhergesagten Werten und allen Datensätzen. Die gewählten Modelle werden auf beiden Datenbasen trainiert und evaluiert und die Ergebnisse verglichen.

1.3. Modelle

Insgesamt wurden vier Klassifikationsalgorithmen auf beiden Datenbasen trainiert und evaluiert. Zusätzlich wurde jedes Modell jeweils mit verschiedenen Parametern evaluiert. Als Modelle wurden Logistische Regression, Support-Vector-Machines (SVM), KNN und Naive Bayes eingesetzt. Bei allen eingesetzten Algorithmen handelt es sich um überwachtes Lernen.

Für jedes Modell wurden die besten Parameter ermittelt, indem das Modell trainiert und auf 40% der Daten evaluiert wurde. Mit den ermittelten Parametern wurde das Modell dann auf 100 zufällig gewählten Trainings- und Testmengen evaluiert und eine Lernkurve mit der durchschnittlichen Präzision gezeichnet. Die Größe der Testmenge lag dabei immer bei 40% der Gesamtmenge. Die Zeit, die das jeweilige Modell für Training und Evaluation auf 100 Datensätzen benötigt, wurde gemessen.

Die Algorithmen sollen im folgenden Abschnitt näher beschrieben werden.

Logistische Regression

Das Prinzip der logistischen Regression basiert auf einer Hypothesenfunktion mit variablen Parametern, die jedem Datensatz einen Wert zwischen 1 und 0 zuordnet. Dieser Wert stellt die Wahrscheinlichkeit dar, dass ein bestimmter Datensatz zur Klasse 1 oder 0 gehört, abhängig von einem gewählten Schwellenwert (Manning, Raghavan, & Schütze, 2008; Ng, 2017).

Die optimalen Parameter für die Hypothesenfunktion werden durch die Minimierung einer Kostenfunktion ermittelt. Die Implementierung der logistischen Regression in Scikit-learn bringt zwei Kostenfunktionen mit (Pedregosa et al., 2011), bezeichnet mit „L1“ und „L2“ als Parameter des Modells. Zusätzlich lässt sich ein Regularisierungswert „C“ einstellen, der ein Overfitting des Modells verhindert.

Das Modell wurde mit beiden Kostenfunktionen getestet. Zusätzlich wurde jeweils aus acht Regularisierungswerten in der Spanne von 0.00001 und 100 der Wert ausgewählt, der bei der Evaluation des Modells die höchste Präzision erreichte. Das Modell wurde dann mit beiden Kostenfunktionen und jeweils dem besten Regularisierungswert auf beiden Datenbasen trainiert und eine Lernkurve errechnet.

SVM

Support-Vector-Machines klassifizieren Datensätze, indem sie die Datenbasis in zwei Mengen aufteilen. Dazu wird eine Ebene berechnet, die die größte Distanz zu beiden Datenmengen hat (Kotsiantis, 2007; Ng, 2017).

Zu diesem Zweck ist die Implementierung von SVMs in Scikit-learn mit fünf Kernel ausgestattet, die bis auf einen auf den Datenbasen getestet wurden. Der Kernel „Precomputed“ wurde nicht getestet, da er Datensätze verlangt, die genauso viele Merkmale haben, wie Datensätze vorhanden sind.

Auch SVMs haben einen Regularisierungsparameter „C“. Für jeden Kernel wurde jeweils der beste Regularisierungsparameter aus neun Werten zwischen 0.00001 und 1000 ermittelt. Mit diesem Parameter wurde dann für jeden Kernel eine Lernkurve gezeichnet.

KNN

Ein k-Nearest-Neighbor-Klassifizierer ordnet Datenpunkten eine Klasse auf Basis seiner k nächsten Nachbar zu. Hat beispielsweise ein Datenpunkt x überwiegend Datenpunkte der Klasse 1 in einer bestimmten Distanz, wird x die Klasse 1 zugewiesen (Kotsiantis, 2007; Ng, 2017).

Das Modell stellt in Scikit-learn mehrere Algorithmen zur Distanzberechnung sowie mehrere Gewichtungen für die Distanzen zur Verfügung. Jeder Algorithmus wurde jeweils mit jeder Distanzgewichtung auf den beiden Datenbasen getestet. Das beste Paar von Algorithmus und Gewichtung im Hinblick auf die erreichte Präzision wurde dann zur Berechnung der

Lernkurve herangezogen. Die übrigen Parameter des Modells wurden auf ihren Standardeinstellungen belassen, sodass für die Klassifizierung eines Datensatzes seine fünf nächsten Nachbarn betrachtet wurden.

Naive Bayes

Der Naive Bayes Klassifizierer ordnet Klassen auf Basis der Häufigkeit von Attributen in Klassen zu. Dabei ist die Grundannahme, dass jedes Attribut nur von der Klasse seines Datensatzes abhängt, die Attribute untereinander sind unabhängig. Das Modell zählt jede Ausprägung eines Attributs innerhalb einer Klasse und wählt für zu klassifizierende Datensätze dann die wahrscheinlichste Klasse (Kotsiantis, 2007; Ng, 2017).

Die Implementierung des Naive Bayes Klassifizierers in Scikit-learn stellt drei Algorithmen zur Verfügung, jeweils optimiert für eine bestimmte Verteilung der Datenpunkte. Jeder der Algorithmen wurde auf beide Datenbasen angewandt und evaluiert. Für jeden Algorithmus wurde eine Lernkurve berechnet.

2. Ergebnisse

In diesem Abschnitt sollen die Ergebnisse des Trainings der einzelnen Modelle auf den Datenbasen beschrieben und verglichen werden. Vorab lässt sich festhalten, dass alle Modelle gute bis sehr gute Präzision bei der Klassifizierung erreicht haben.

Logistische Regression

Die logistische Regression liefert hohe Präzisionswerte für beide Datenbasen und Kostenfunktionen. Die höchste Präzision von 1,0 wurde mit der Kostenfunktion „l1“ auf der Datenmenge mit entfernten unvollständigen Datensätzen erreicht. Die Kostenfunktion „l2“ erreicht jedoch einen beinahe gleich hohen Wert. Tabelle 1 enthält die Ergebnisse der logistischen Regression pro Datenbasis und Kostenfunktion.

Tabelle 1. Ergebnisse der logistischen Regression

Datenbasis	**Kostenfunktion: l1**	**Kostenfunktion: l2**
Ohne vorhergesagte fehlende Werte	Präzision: 1,0 Trainingszeit: 46,599 Sek.	Präzision: 0,999558 Trainingszeit: 13,664 Sek.
Mit vorhergesagten fehlenden Werten	Präzision: 0,957231 Trainingszeit: 158,416 Sek.	Präzision: 0,957231 Trainingszeit: 18,04 Sek.

Die Präzision beim Training auf der Datenmenge mit vorhergesagten Werten ist mit etwa 0,95 etwas niedriger, beide Kostenfunktionen lieferten die gleiche Präzision in der Evaluierung. Die größten Unterschiede lassen sich in der benötigten Zeit für Training und Evaluierung feststellen.

Eine logistische Regression mit der Kostenfunktion „l1“ dauert deutlich länger als mit „l2“. Zudem hat sich die benötigte Zeit auf der Datenbasis mit vorhergesagten Werten im Vergleich zur Datenbasis ohne vorhergesagte Werte bei der Verwendung von „l1“ mehr als verdreifacht. Im Gegensatz dazu dauert eine logistische Regression mit „l2“ auf der kleineren Datenbasis circa 14 Sekunden, auf der größeren sind es 18 Sekunden. Hier ist der Unterschied zwischen den Datenbasen geringer.

Abbildungen 2 und 3 zeigen die beiden Lernkurven für die logistische Regression auf der Datenbasis ohne vorhergesagte Werte, Abbildungen 4 und 5 zeigen die Lernkurven mit vorhergesagten Werten. Die rote Kurve stellt die Präzision auf der Trainingsmenge, die grüne Kurve die Präzision auf der Evaluations- oder Testmenge dar. Die y-Achse enthält die Präzisionswerte, die x-Achse die gesehenen Trainingsbeispiele. Aus allen Abbildungen geht hervor, dass weder ein Over- noch ein Underfitting stattfindet und die trainierten Modelle korrekt trainiert sind.

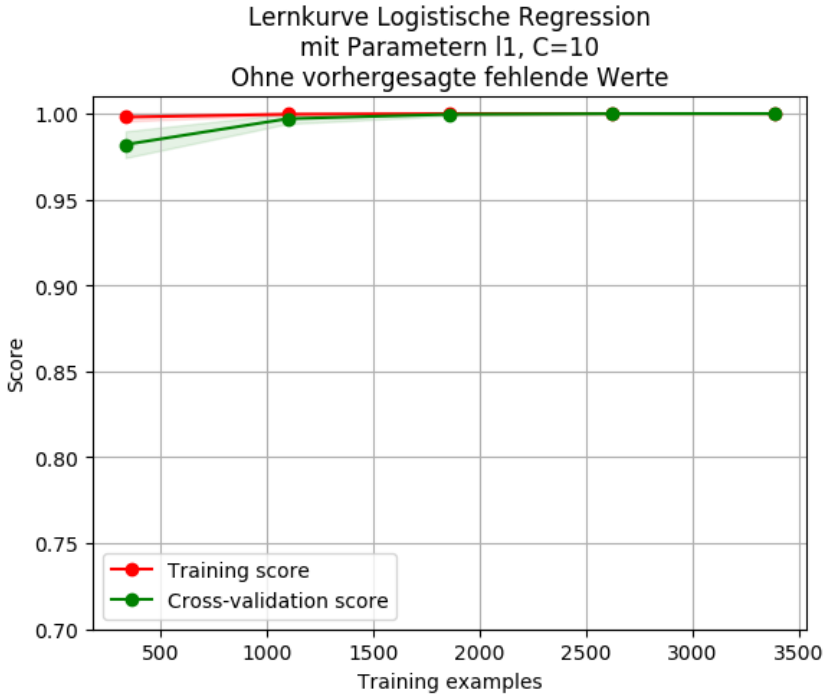

Abbildung 2. Lernkurve Logistische Regression mit Kostenfunktion l1 und ohne vorhergesagte Werte

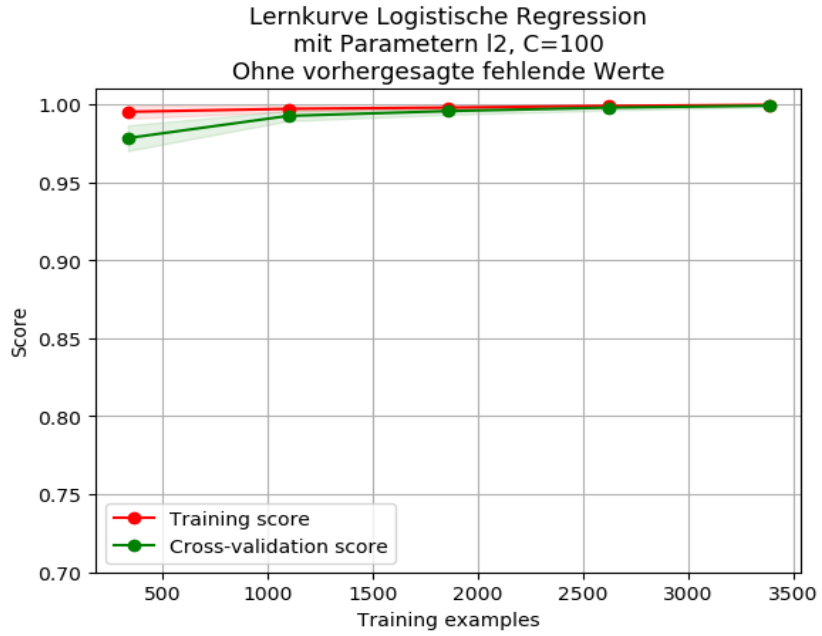

Abbildung 3. Lernkurve Logistische Regression mit Kostenfunktion l2 und ohne vorhergesagte Werte

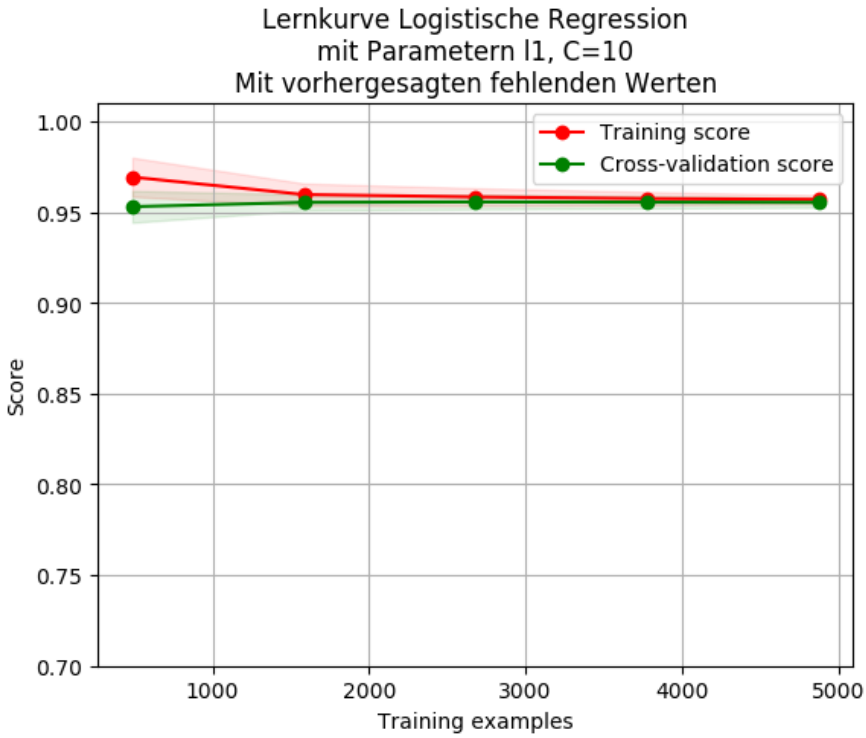

Abbildung 4. Lernkurve Logistische Regression mit Kostenfunktion l1 mit vorhergesagten Werten

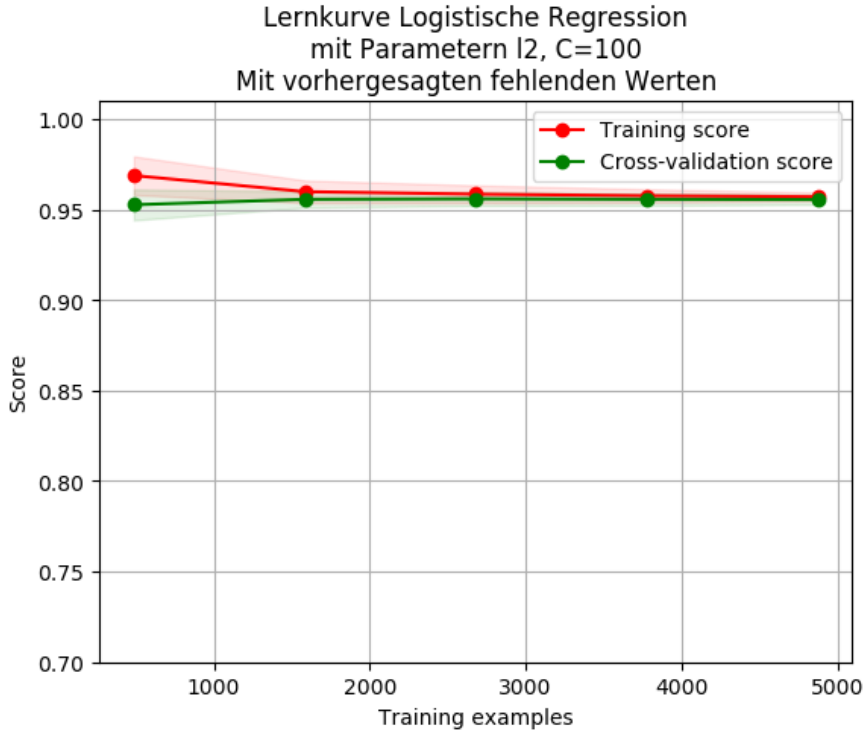

Abbildung 5. Lernkurve Logistische Regression mit Kostenfunktion l2 mit vorhergesagten Werten

SVM

SVMs lieferten ebenfalls größtenteils sehr gute Ergebnisse. Bis auf SVMs mit einem Sigmoid-Kernel liegen die Werte für die Präzision der Vorhersagen bei 1,0 oder leicht darunter. Der Sigmoid-Kernel liefert mit Präzisionen von circa 0,749 und 0,531 die schlechtesten Ergebnisse. Tabelle 1 zeigt die Ergebnisse pro Datenbasis und verwendeten Kernel.

Die meiste Zeit benötigte der lineare Kernel für Training und Evaluation mit ungefähr 46 beziehungsweise 1170 Sekunden. Die SVM mit einem polynomialen Kernel war am schnellsten mit 7 und 13 Sekunden, während bei beiden Datenbasen eine Präzision der Vorhersage von 1,0 erreicht wurde.

Insgesamt benötigen alle Kernel mehr Zeit für Training und Evaluation der Datenbasis mit vorhergesagten Werten, wobei der lineare Kernel ein Vielfaches der Zeit mehr benötigt als andere Kernel. Hier lässt sich festhalten, dass der polynomiale Kernel in Sachen Präzision und Zeit am besten geeignet für dieses Klassifikationsproblem ist. Der Sigmoid-Kernel hingegen ist aufgrund der im Vergleich geringeren Präzision nicht geeignet, zusätzlich benötigt er auch nach dem linearen Kernel am meisten Zeit.

Tabelle 2. Ergebnisse SVM

Datenbasis	**Kernel: linear**	**Kernel: poly**	**Kernel: rbf**	**Kernel: sigmoid**
Ohne vorhergesagte fehlende Werte	Präzision: 1,0 Trainingszeit: 46,308 Sek.	Präzision: 1,0 Trainingszeit: 7,132 Sek.	Präzision: 1,0 Trainingszeit: 13,092 Sek.	Präzision: 0,749 Trainingszeit: 42,211 Sek.
Mit vorhergesagten fehlenden Werten	Präzision: 0,969538 Trainingszeit: 1170,986 Sek.	Präzision: 1,0 Trainingszeit: 13,206 Sek.	Präzision: 1,0 Trainingszeit: 29, 248 Sek.	Präzision: 0, 530769 Trainingszeit: 177, 826 Sek.

Abbildungen 6 und 7 zeigen die Lernkurve für SVMs mit dem polynomialen Kernel auf beiden Datenbasen. Aus der Form der Kurven lässt sich erkennen, dass Training und Evaluation ohne Over- oder Underfitting stattfinden.

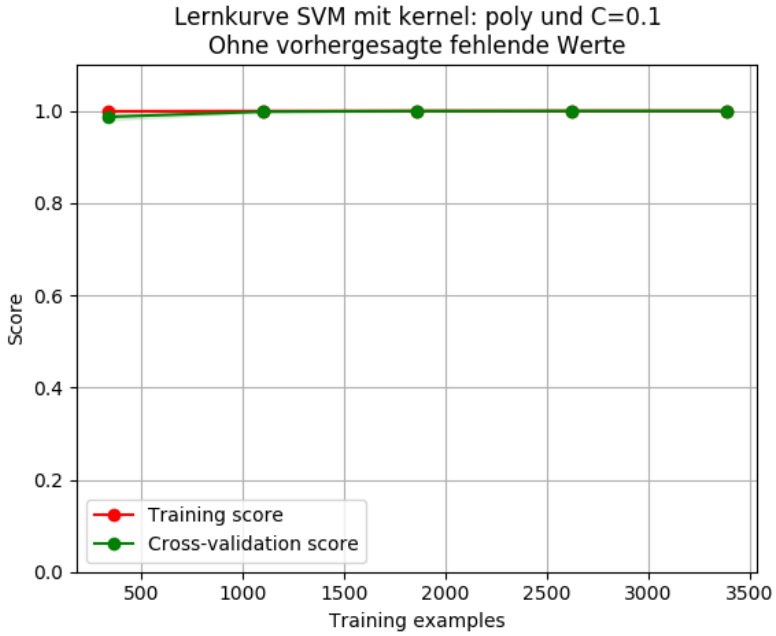

Abbildung 6. Lernkurve SVM mit Kernel poly ohne vorhergesagte Werte

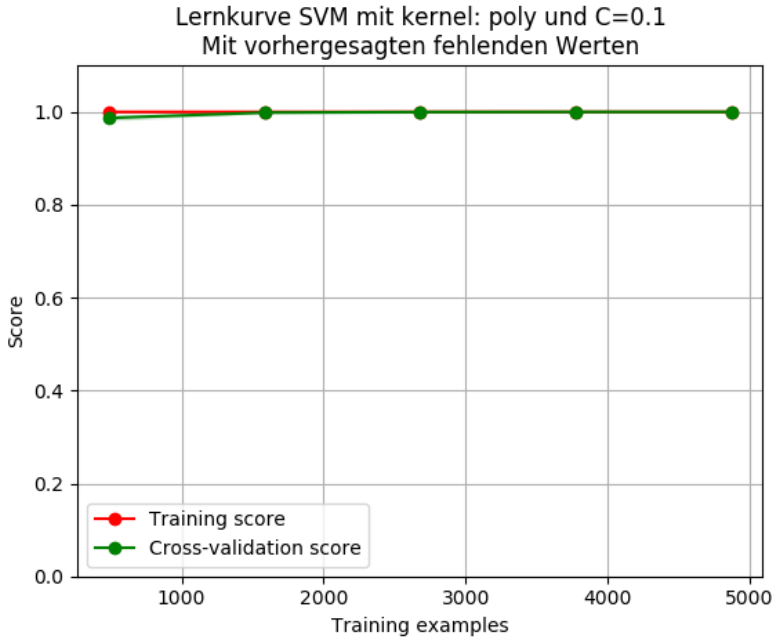

Abbildung 7. Lernkurve SVM mit Kernel poly mit vorhergesagten Werten

Abbildungen 8 und 9 zeigen die Lernkurven des Sigmoid-Kernels auf beiden Datenbasen. Auf die anderen Lernkurven wird aus Platzgründen an dieser Stelle verzichtet, da sie ohnehin beinahe identisch mit denen aus Abbildung 6 und 7 sind.

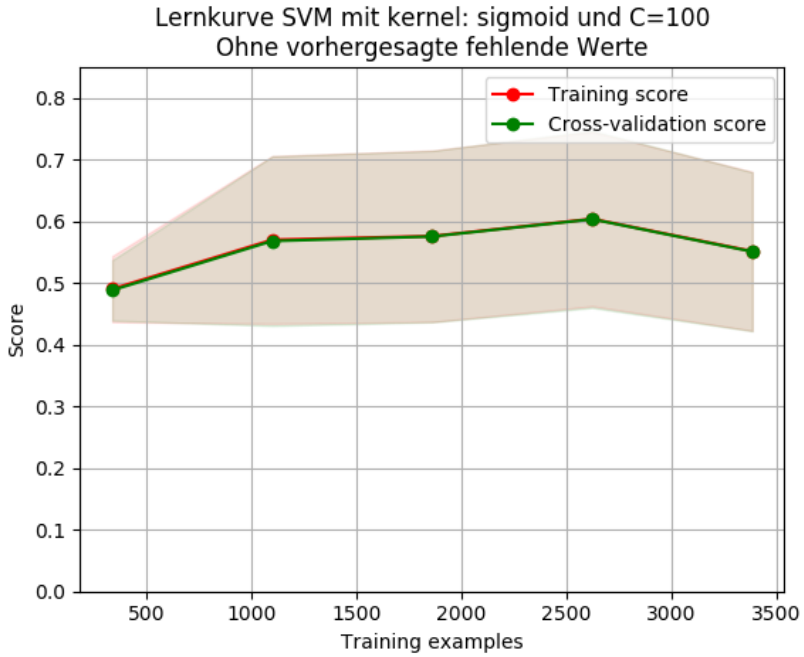

Abbildung 8. Lernkurve SVM mit Kernel sigmoid ohne vorhergesagte Werte

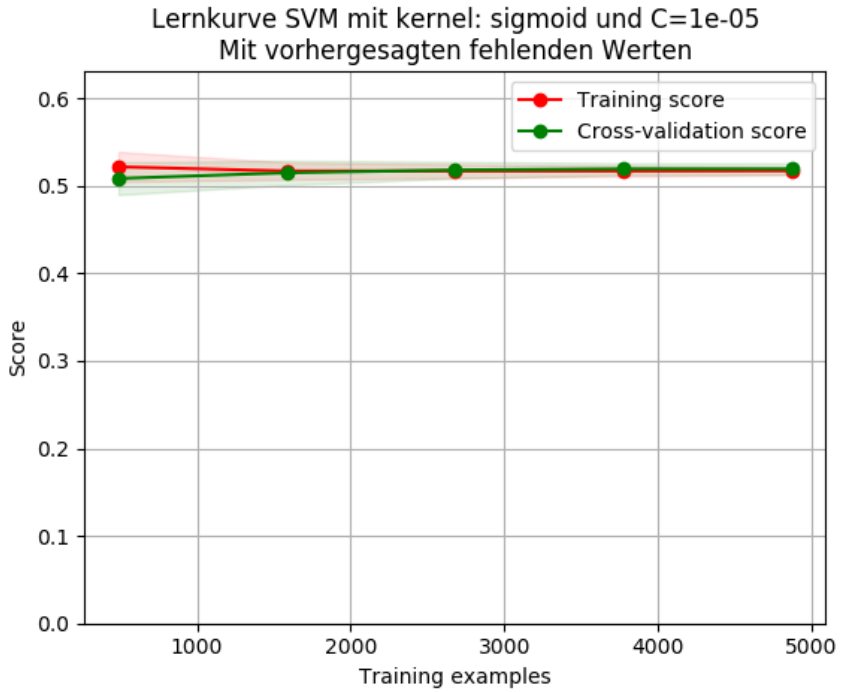

Abbildung 9. Lernkurve SVM mit Kernel sigmoid mit vorhergesagten Werten

KNN

Tabelle 3 zeigt die Ergebnisse des Trainings eines KNN mit verschiedenen Algorithmen und Distanzgewichtungen. Jede Spalte stellt ein Paar aus Algorithmus und Distanzgewichtung dar, jeweils für beide Datenbasen.

Insgesamt ist die Leistung von KNN bei der Klassifizierung der Daten sehr gut, es werden durchweg Präzisionen nahe 1,0 erreicht. Die erreichten Werte unterscheiden sich zwischen

den Algorithmus-Distanzgewichtungspaaren pro Datenbasis nur marginal, die Unterschiede sind aufgrund der Rundung in der Tabelle nicht sichtbar.

Der schnellste Algorithmus ist kd_tree, der langsamste brute. Da alle Algorithmen in etwa die gleiche Präzision liefern, kann man bei kd_tree von der höchsten Effizienz sprechen, zieht man Präzision und benötigte Zeit in Betracht. Abbildung 10 und 11 zeigen die Lernkurven des Algorithmus brute mit uniformer Distanzgewichtung, auf die restlichen Lernkurven wird aus Platzgründen verzichtet.

Tabelle 3. Ergebnisse KNN

Datenbasis	**ball_tree uniform**	**ball_tree distance**	**kd_tree uniform**	**kd_tree distance**	**brute uniform**	**brute distance**
Ohne vorhergesagte fehlende Werte	0,999557 29,848 Sek.	0,999557 29,115 Sek.	0,999557 16,281 Sek.	0,999557 16,076 Sek.	0,999557 39,275 Sek.	0,999557 38,268 Sek.
Mit vorhergesagten fehlenden Werten	0,999077 50,594 Sek.	0,999077 50,917 Sek.	0,999077 26,888 Sek.	0,999077 26,993 Sek.	0,999077 72,773 Sek.	0,999077 75,494 Sek.

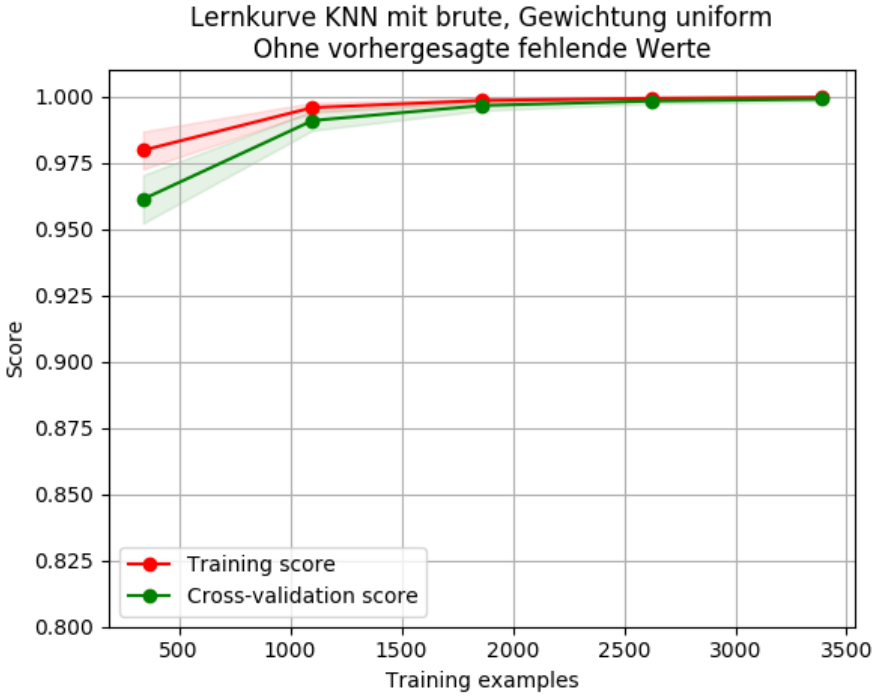

Abbildung 10. Lernkurve KNN ohne vorhergesagte Werte

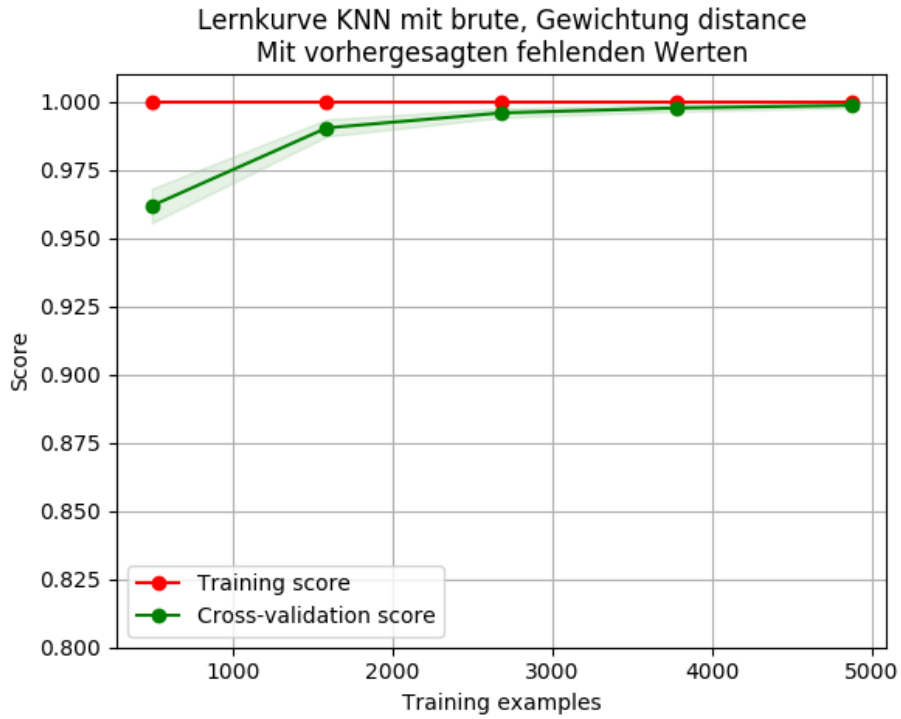

Abbildung 11. Lernkurve KNN mit vorhergesagten Werten

Naive Bayes

Die Ergebnisse des Trainings und der Evaluation der drei Naive Bayes Implementierungen sind in Tabelle 4 zu sehen. Der Gaussian Naive Bayes ist der schnellste auf beiden Datenbasen mit 3,4 und 3,6 Sekunden, gefolgt vom Multinomial Naive Bayes mit 7,3 und 8,2 Sekunden. Am meisten Zeit benötigt der Bernoulli Naive Bayes mit 8,2 beziehungsweise 9,8 Sekunden.

Auf den Daten ohne vorhergesagte fehlende Werte erreicht der Bernoulli Naive Bayes mit circa 0,91 die höchste Präzision, während auf den Daten mit vorhergesagten Werten der Gaussian Naive Bayes die beste Präzision mit ungefähr 0,91 zeigt. Die Leistung der Modelle auf den Daten mit vorhergesagten Werten ist insgesamt besser, hier liegt der niedrigste Präzisionswert bei 0,81, während er auf der anderen Datenbasis bei 0,69 liegt.

Tabelle 4. Ergebnisse Naive Bayes

Datenbasis	**Gaussian Naive Bayes**	**Multinomial Naive Bayes**	**Bernoulli Naive Bayes**
Ohne vorhergesagte fehlende Werte	Präzision: 0,6922055 Zeit: 3,408 Sek.	Präzision: 0,856510 Zeit: 7,326 Sek.	Präzision: 0,911426 Zeit: 8,210 Sek.
Mit vorhergesagten fehlenden Werten	Präzision: 0,916000 Zeit: 3,597Sek.	Präzision: 0,805230 Zeit: 8,203 Sek.	Präzision: 0,870462 Zeit: 9,830 Sek.

Abbildungen 12 bis 17 zeigen die Lernkurven der einzelnen Modelle des Naive Bayes, jeweils auf beiden Datenbasen. Für den Gaussian Naive Bayes zeigt sich, dass seine Präzision auf der einen Datenbasis mit zunehmenden Trainingsbeispielen abnimmt, während sie auf der anderen Datenbasis zunimmt. Bei den anderen Modellen bleibt sie weitestgehend konstant. Während des Trainings des Gaussian Naive Bayes zeigt sich auch eine hohe Standardabweichung, dargestellt durch die halbtransparenten Flächen in den Graphen der Lernkurven. Diese Standardabweichung nimmt mit mehr Trainingsbeispielen jedoch ab.

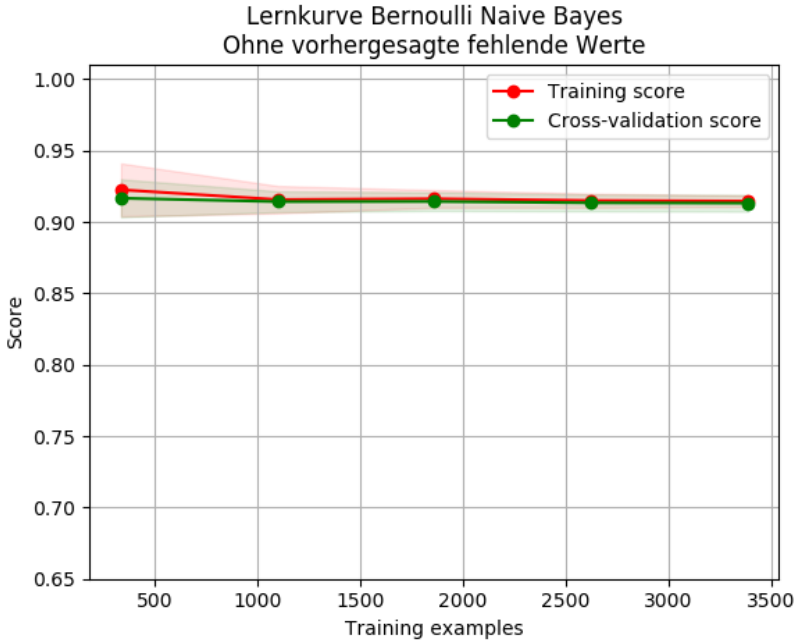

Abbildung 12. Lernkurve Bernoulli Naive Bayes ohne vorhergesagte Werte

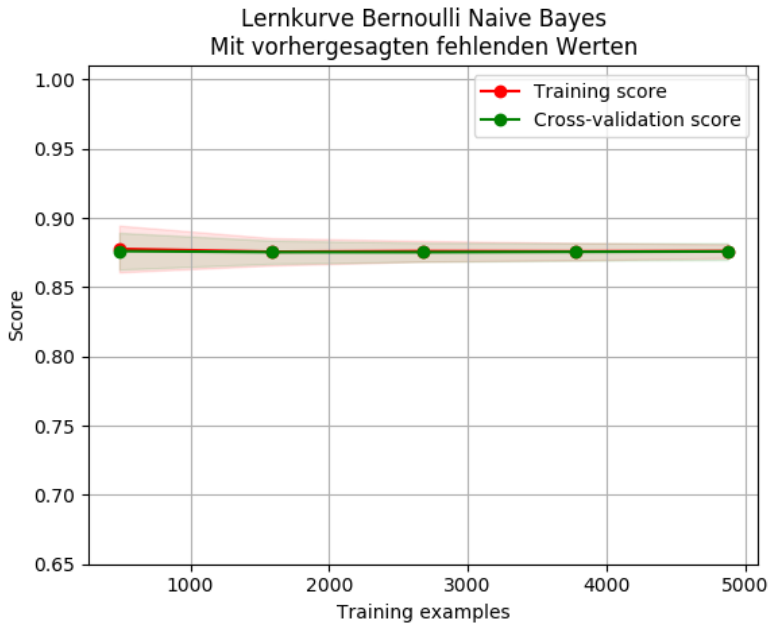

Abbildung 13. Lernkurve Benroulli Naive Bayes mit vorhergesagten Werten

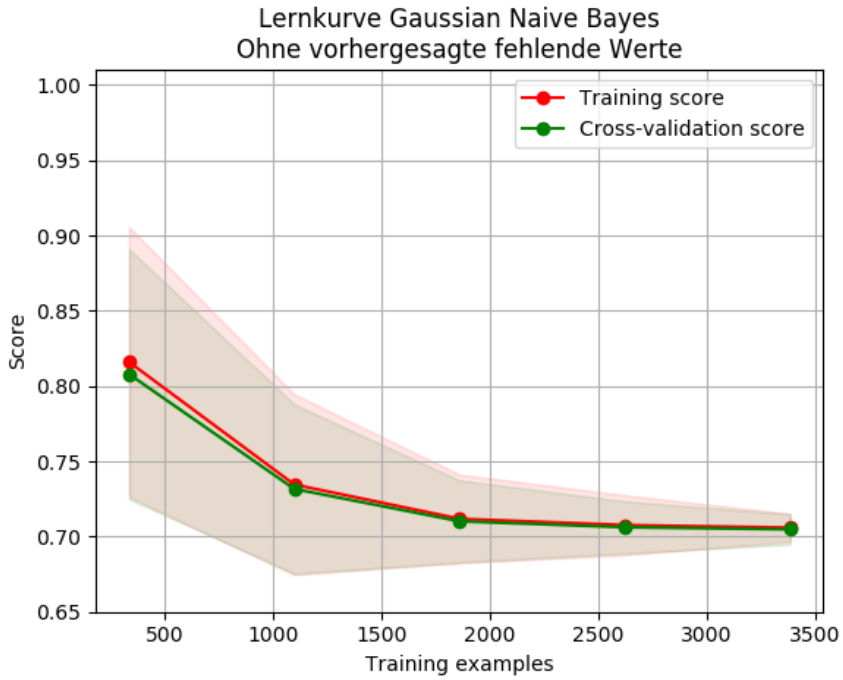

Abbildung 14. Lernkurve Gaussian Naive Bayes ohne vorhergesagte Werte

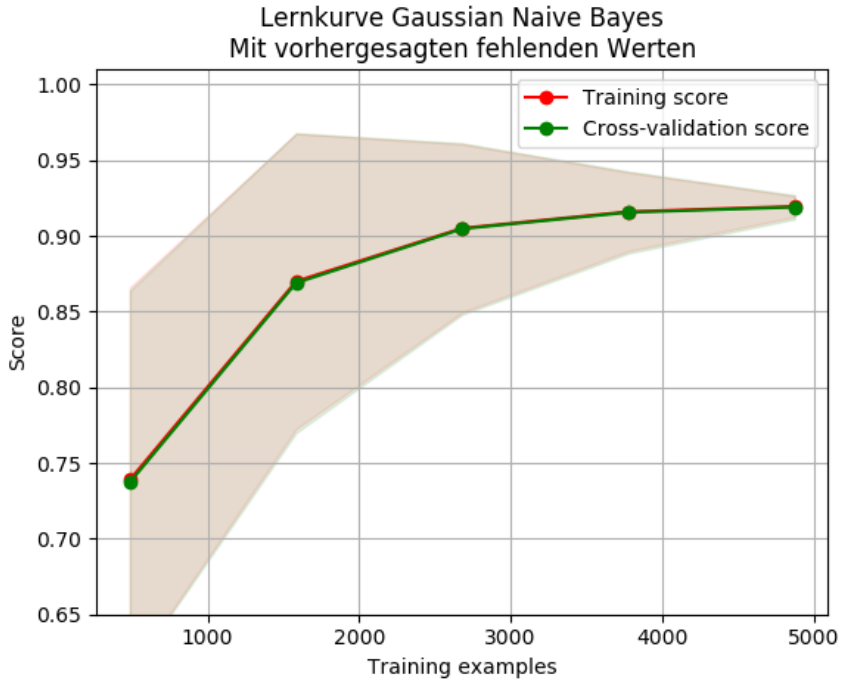

Abbildung 15. Lernkurve Gaussian Naive Bayes mit vorhergesagten Werten

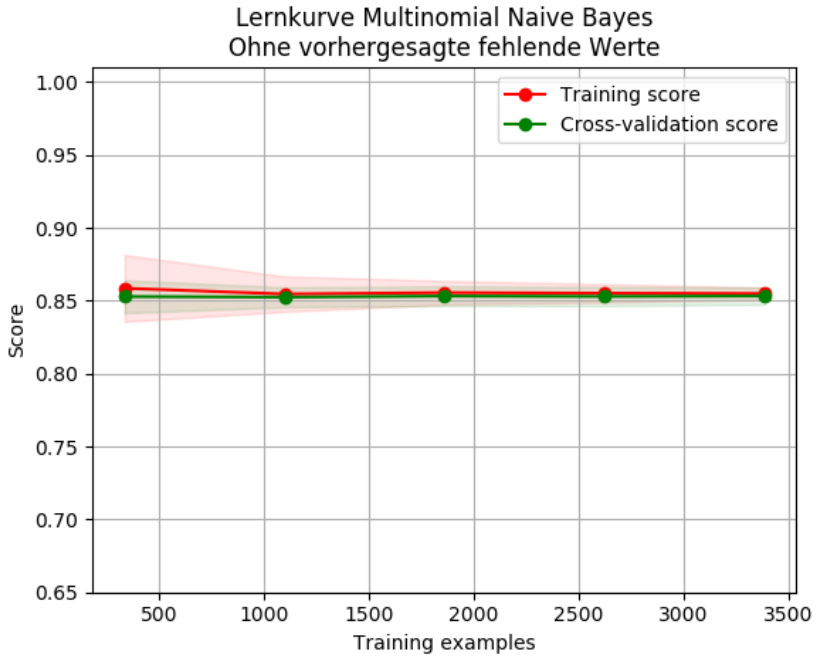

Abbildung 16. Lernkurve Multinomial Naive Bayes ohne vorhergesagte Werte

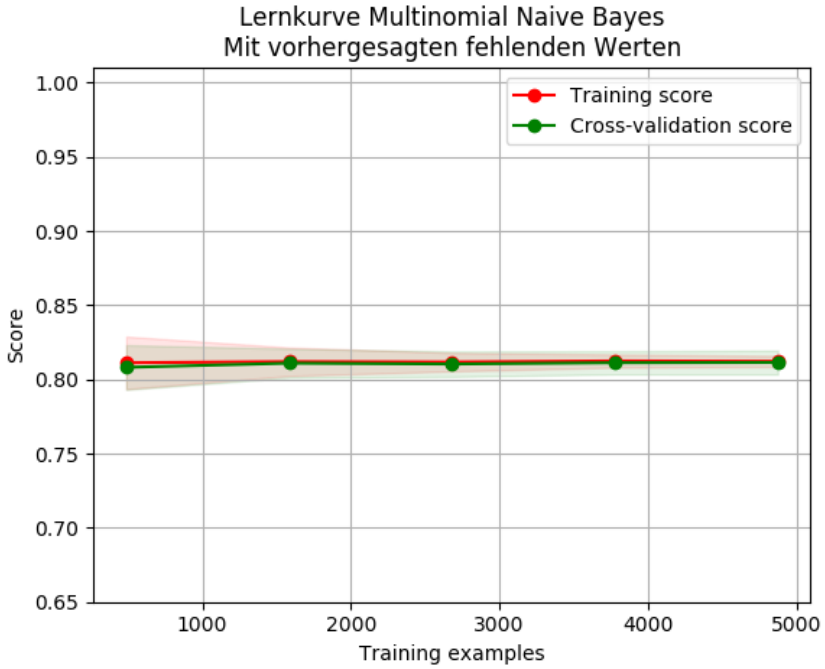

Abbildung 17. Lernkurve Multinomial Naive Bayes mit vorhergesagten Werten

Insgesamt lässt sich festhalten, dass alle Modelle eine hohe Vorhersagepräzision auf beiden Datenbasen erreichen, wobei die Parameter die richtigen Parameter verwendet werden müssen. Die Trainingszeit ist naturgemäß auf den Daten mit vorhergesagten Werten bei allen Modellen höher, ist aber bei einigen immer noch niedrig. Die höchsten Präzisionswerte von 1,0 werden mit SVMs erreicht, mit einem polynomialen Kernel ist außerdem die Trainingszeit gering. Die schnellsten Modelle sind Naive Bayes Modelle. Diese konnten

jedoch nicht die Präzision von SVMs erreichen, liefern aber trotzdem mit Werten von circa 0,9 gute Ergebnisse.

3. Fazit

Die Essbarkeit von Pilzen lässt sich anhand der Merkmale offenbar gut bestimmen. Alle eingesetzten Modelle erreichten hohe Präzisionswerte in der Klassifizierung der Pilze. Auch die Vorhersage der fehlenden Werte in den Datensätzen wirkte sich nicht negativ auf die Präzision aus, daher ist diese Methode in diesem konkreten Fall sehr nützlich, da man bei der ohnehin eher kleinen Datenmenge wertvolle Informationen behält. Signifikante Steigerungen in der benötigten Trainingszeit konnten für die Modelle mit relevanten Parametern auch nicht festgestellt werden.

Trotzdem bleibt bei vorhergesagten fehlenden Werten eine gewisse Unsicherheit, ob diese Werte auch der Realität entsprechen. Zudem sind nicht alle Merkmale in den Daten objektiv messbar, „odor“ (Geruch) basiert auf einer stark subjektiven Einschätzung. Für eine reale Anwendung einer Klassifikation von Pilzen sollte man diese Dinge berücksichtigen.

4. Quellenverzeichnis

Kotsiantis, S. B. (2007). Supervised machine learning: A review of classification techniques. *Informatica*, *31*, 249–268.

Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Manning, C.D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval.* Cambridge: Cambridge University Press.

Ng, A. (18. April 2017). *Machine Learning* [Online-Kurs]. Abgerufen von https://www.coursera.org/learn/machine-learning

Pedregosa, F., Varoquaux G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *JMLR 12,* 2825-2830.