**Lernen eines Naive-Bayes-Klassifikators für Restaurant-Bewertungen in TripAdvisor**

|  |  |
| --- | --- |
| **Name** | Jannik Bikowski, Till Emme, Dominik Ramsauer |
| **E-Mail** | [Jannik.Bikowski@stud.uni-regensburg.de](mailto:Jannik.Bikowski@stud.uni-regensburg.de), [Till.Emme@stud.uni-regensburg.de](mailto:Till.Emme@stud.uni-regensburg.de),  [Dominik.Ramsauer@stud.uni-regensburg.de](mailto:Dominik.Ramsauer@stud.uni-regensburg.de) |
| **Fachsemester** | 5, 5, 5 |
| **Matrikelnummer** | 1768542, 1750770, 1737690 |
| **Thema:** | „Lernen eines Naive-Bayes-Klassifikators für Restaurant-Bewertungen in TripAdvisor“ |
| **Kurs** | Informationssysteme (LV-Nr. 36670b) |
| **Semester** | SS 2017 |
| **Kursleiter** | Prof. Dr. Bernd Ludwig |

Inhaltsverzeichnis

[1. Ziele der Arbeit 1](#_Toc492392070)

[2. Theoretischer Hintergrund 2](#_Toc492392071)

[3. Implementierung 3](#_Toc492392072)

[3.1 Datenakquise 3](#_Toc492392073)

[3.2 Algorithmus 3](#_Toc492392074)

[4. Optimierung und Vergleich mit Weka 5](#_Toc492392075)

[4.1 Optimierungen 5](#_Toc492392076)

[4.2 Performanz 6](#_Toc492392077)

[4.3 Vergleich mit Weka 6](#_Toc492392078)

[5. Fazit 7](#_Toc492392079)

Hisejfsiejfosikefjalsejkfalsßeifjalskfjlkajsdddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd

# Ziele der Arbeit

Das Ziel der Arbeit ist die Implementierung eines Naive-Bayes-Klassifikators, der anhand vorher gesammelter Daten von Restaurant-Bewertungen die Gesamtbewertung eines Restaurants vorhersagt. Hierfür wurden Bewertungen von 150 Restaurants der Touristikwebsite „TripAdvisor“ gesammelt. Es wurden dafür für jedes Restaurant die Teilbewertungen für Service, Küche, Qualität und Einrichtung, sowie die Gesamtbewertung als Datengrundlage benutzt.

In dieser Arbeit soll zunächst der theoretische Hintergrund beleuchtet werden. Im Anschluss wird näher auf die praktische Implementierung des Klassifikators eingegangen. Der implementierte Algorithmus soll dann mithilfe von Optimierungsversuchen verbessert und die Klassifizierungsrate mit dem Machine-Learning-Tool „Weka“ verglichen werden.

# Theoretischer Hintergrund

In der modernen Gesellschaft spielen Informationssysteme eine immer größere Rolle. Besonders in der angewandten Informatik kommen diese zum Einsatz, zum Beispiel in der automatischen Sprachverarbeitung aber auch in der Bioinformatik, der Krebsforschung oder Psychologie. Ein wesentlicher Bestandteil solcher Informationssysteme ist die Lösung von Klassifizierungsproblemen. Um Informationen zu verarbeiten müssen Daten in festgelegte Kategorien eingeordnet werden. Zur Lösung der genannten Probleme können zahlreiche Ansätze verfolgt werden, z.B. Maximum-Entropie-Modell oder Naives-Bayes-Klassifikation. Letzterer wird im Folgenden genauer erläutert.

Der Naive-Bayes-Klassifikator versucht jedes einzelne Objekt derjenigen Klasse zuzuordnen, zu der es am wahrscheinlichsten gehört. Für ein gegebenes Objekt *o* soll für jede Klasse *c* die bedingte Wahrscheinlichkeit der Klasse *P(c|o)* berechnet werden. Diejenige Klasse mit der höchsten Wahrscheinlichkeit wird die vom Klassifikator vorhergesagte Klasse.

Daraus ergibt sich folgende Formel:

C:\Users\Dominik-PC\AppData\Local\Microsoft\Windows\INetCache\Content.Word\1.png

Mit dem Bayes’schen Theorem folgt:

C:\Users\Dominik-PC\AppData\Local\Microsoft\Windows\INetCache\Content.Word\2.png

*P(o)* kann weggelassen werden, weil es für alle Klassen gleich ist.

C:\Users\Dominik-PC\AppData\Local\Microsoft\Windows\INetCache\Content.Word\3.png

1. Modellannahme: Die gewählten Merkmale sind angemessen, um das Objekt zu repräsentieren. *x1, …, xn* bilden die Merkmale des Objekts.

C:\Users\Dominik-PC\AppData\Local\Microsoft\Windows\INetCache\Content.Word\4.png

2. Modellannahme: Die gewählten Merkmale sind voneinander unabhängig.

C:\Users\Dominik-PC\AppData\Local\Microsoft\Windows\INetCache\Content.Word\5.png

# Implementierung

Der Klassifikator wurde vollständig in Python 3 implementiert und das Programm läuft ohne sichtbare GUI. Zum Ausführen des Programms muss die reviews.csv Datei im selben Ordner liegen wie NaiveBayesClassifier.py.

## 3.1 Datenakquise

Vor der Implementierung des Naive-Bayes Algorithmus mussten zunächst Daten akquiriert werden. Dazu wurden die Bewertungen für Service, Küche, Qualität und Einrichtung, sowie die jeweilige Gesamtbewertung von 150 Restaurants im Raum Regensburg, Landshut und München auf der Seite TripAdvisor gesammelt und in einer Komma-separierten CSV-Datei gespeichert. Auch der Name des Restaurants wurde zur Strukturierung der Daten in die CSV-Datei integriert. Bei der Datenakquise wurde darauf geachtet, jede mögliche Bewertung der 5-stufigen Bewertungsskala in den Datensatz mit einzubeziehen.

Zur Integration der Daten in das Programm wurde eine readData-Funktion implementiert, die die Datensätze in der CSV-Datei einliest und zur Weiterverarbeitung strukturiert. Das Attribut Name wird ignoriert, da es keinen Einfluss auf die Klassifikation besitzt.

## 3.2 Algorithmus

Der Naive-Bayes-Algorithmus wird in einer eigenen Klasse „NaiveBayesClassifier“ implementiert. Dazu werden zunächst wichtige Klassenvariablen deklariert, in denen später die Wahrscheinlichkeiten der einzelnen Klassen, die durch die Gesamtbewertungen repräsentiert werden, sowie die bedingten Wahrscheinlichkeiten der Features, repräsentiert durch die Teilbewertungen, gespeichert werden.

Durch Aufruf der train-Funktion, die als Übergabeparameter die strukturierten Daten der readData-Funktion erwartet, werden sowohl die Klassenwahrscheinlichkeiten als auch die Featurewahrscheinlichkeiten in zwei separaten Methoden berechnet. Das Ermitteln der Klassenwahrscheinlichkeiten erfolgt durch Auszählen der absoluten Auftretenswahrscheinlichkeiten der einzelnen Klassen und anschließendem Teilen dieser Wahrscheinlichkeiten durch die Anzahl der Datensätze. Die Bestimmung der bedingten Wahrscheinlichkeiten der Features erfolgt ähnlich: Zunächst wird für jedes Feature die absolute Häufigkeit innerhalb einer Klasse bestimmt und dieser Wert anschließend durch die Gesamtanzahl der Teilbewertungen der Klasse geteilt. Dieser Vorgang wird für alle Features jeder Klasse wiederholt, damit bereits vor dem Klassifizieren alle für den Klassifikationsvorgang benötigten Wahrscheinlichkeiten vorhanden sind und nur noch abgerufen werden müssen. Anhand dieser Werte kann der Klassifizierer nun Vorhersagen über unbekannte Daten treffen.

Zur Klassifizierung kann nun die classify-Methode genutzt werden, die eine Liste aus den 4 Teilbewertungen eines Restaurants erwartet. Die Vorhersage einer Klasse erfolgt nun durch Berechnung der Wahrscheinlichkeit, dass das unbekannte Restaurant zu einer bestimmten Klasse gehört, für jede Klasse. Dazu werden für jede Klasse die bedingten Wahrscheinlichkeiten P(Teilbewertung|Klasse) nachgeschlagen und multipliziert. Anschließend wird die Wahrscheinlichkeit für die Klasse selbst mit diesem Wert multipliziert. Diejenige Klasse, für die der berechnete Wert am größten ist, ist nun die wahrscheinlichste Klasse des unbekannten Restaurants und bildet somit die Vorhersage des Klassifikators.

Desweiteren kann mithilfe der classifyAll-Methode eine größere Menge an Datensätzen klassifiziert werden, indem für jeden einzelnen Datensatz der Menge die classify-Methode aufgerufen wird. Da die Methode zur Berechnung der Klassifikationsrate des Klassifizierers dienen soll, müssen die Gesamtbewertungen der übergebenen Restaurants bereits bekannt sein. Die Accuracy wird nun berechnet, indem der Klassifizierer für jedes einzelne Restaurant eine Vorhersage trifft und diese anschließend mit der wahren Gesamtbewertung verglichen wird. Der Anteil der korrekt klassifizierten Restaurants an der Gesamtmenge aller klassifizierten Restaurants spiegelt dann die Genauigkeit des Klassifikators wieder.

# 4. Optimierung und Vergleich mit Weka

## 4.1 Optimierungen

Zur Verbesserung der Arbeitsweise des Programms wurden einige Änderungen sowohl am Programm selbst als auch am Algorithmus vorgenommen. Da die bedingten Wahrscheinlichkeiten der Features bei wachsender Größe der Daten sehr klein werden können, wurde folgende logarithmische Umformung vorgenommen, um das Rechnen mit kleinen Zahlen zu verhindern:

https://www.zahlen-kern.de/editor/equations/eyrn.png

Entsprechend dieser Formel wurden die classify- und die calculateFeatureProbabilities-Methoden angepasst. Da alle Ausgangswahrscheinlichkeiten der Features und Klassen zwischen 0 und 1 liegen, werden diese Werte durch die Logarithmierung negativ.

Neben dieser Umformung wurde das sogenannte Add-One Smoothing angewandt, da sonst die Wahrscheinlichkeit ungesehener Teilbewertungen 0 wäre, was den Algorithmus unbrauchbar machen würde, da das Produkt zur Berechnung der Wahrscheinlichkeit einer Klasse ebenso 0 werden würde. Auch die logarithmische Umformung würde nicht mehr funktionieren, da log(0) nicht definiert ist. Ohne Smoothing berechnet man die bedingten Featurewahrscheinlichkeiten folgendermaßen:

https://www.zahlen-kern.de/editor/equations/eysg.png

Beim Add-One Smoothing wird nun im Zähler und Nenner 1 sowie die Gesamtanzahl unterschiedlicher Teilbewertungen im Nenner addiert. Somit ergibt sich folgende Umformung:

https://www.zahlen-kern.de/editor/equations/eysh.png

Durch die Addition von 1 wird also suggeriert, dass jede Teilbewertung mindestens einmal gesehen wurde. Indem die Gesamtanzahl unterschiedlicher Teilbewertungen im Nenner addiert wird, wird der Effekt der ungesehenen Bewertungen ausgeglichen, um die Wahrscheinlichkeit wieder anzugleichen.

Zuletzt wurde eine Methode crossFoldValidation zur besseren Bestimmung der Genauigkeit des Klassifizierers implementiert. Mit dieser Methode kann eine k-Kreuzvalidierung durchgeführt werden, wobei der Parameter k an die Funktion übergeben wird. Dann wird der Datensatz in k Teilmengen aufgeteilt und anschließend wird in k Schritten die Genauigkeit des Klassifizierers getestet, in dem eine der k Mengen als Testmenge und die übrigen Mengen als Trainingsset fungieren. Abschließend wird die durschnittliche Accuracy des Klassifikators in den k-Durchläufen zurückgegeben.

## 4.2 Performanz

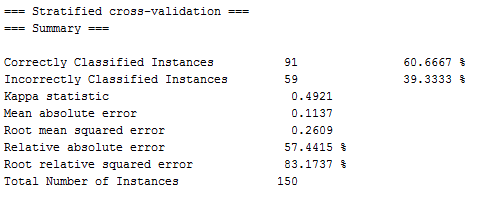
Die Klassifikationsrate des Klassifikators bei Aufteilen der Daten in Trainings- und Testset im Verhältnis 80:20 liegt bei Durchführung einer Cross Fold Validation bei 66%. Es wäre denkbar, dass durch bessere Parameterabschätzung (beispielsweise besseres Smoothing-Verfahren) sowie durch Vergrößerung der Datensammlung eine gesteigerte Accuracy erreicht werden könnte.

## 4.3 Vergleich mit Weka

Beim Vergleich der Accuracy des implementierten Algorithmus mit dem Naive-Bayes des Weka Explorers zeigt sich, dass fast dieselbe Klassifikationsrate von beiden Programmen erreicht wird. Während der Naive-Bayes von Weka bei Cross Fold Validation eine Klassifikationsrate von 60,6% erreicht, schafft unsere Implementierung eine Klassifikationsrate von 66%, was jedoch möglicherweise an einer anderen Aufteilung des Datensatzes liegen könnte. Daher liegt es nahe, dass der Naive-Bayes von Weka in einer sehr ähnlichen Art und Weise und möglicherweise mit dem selben Smoothing-Verfahren implementiert wurde.

Vergleich bei Cross Fold Validation:

Weka:



Unsere Implementierung:



# 5. Fazit

Der Naive-Bayes-Algorithmus ist in der Programmiersprache Python schnell und einfach zu implementieren. Und mit einigen Optimierungen kann der hier implementierte Ansatz des Naive-Bayes mit den Klassifikationsraten des Weka Algorithmus mithalten. Damit eignet er sich auch für weitere Anwendungsbereiche, abseits von den hier getesteten Restaurantbewertungen. Weiterhin ist anzumerken, dass die Genauigkeit des Klassifikators sinkt, je mehr die Datensätze streuen, das Selbe trifft jedoch auch bei der Implementierung von Weka zu. Dies könnte womöglich durch eine noch größere Datenmenge kompensiert werden. Um die Genauigkeit weiter zu erhöhen könnte außerdem das Smoothing Verfahren noch verbessert werden. Insgesamt erfüllt unsere Implementierung aber ihren Zweck, und zwar der Klassifikation von Restaurantbewertungen auf TripAdvisor.