Woche 06: Programmierung – Logistische Regression mit Scikit-learn

Skript

Erarbeitet von
Ludmila Himmelspach

Lernziele

- Ein logistisches Regressionsmodell in Python erstellen und es für einen gegebenen Datensatz trainieren können
- Ein trainiertes Regressionsmodell nutzen können, um für Beobachtungen der Testmenge Werte für die Zielgröße vorherzusagen

Inhalt

Einstieg

Eine logistische Regression modelliert die Beziehung zwischen den Werten „erklärender" Merkmale bzw. Features und den Werten eines kategorialen Zielmerkmals.

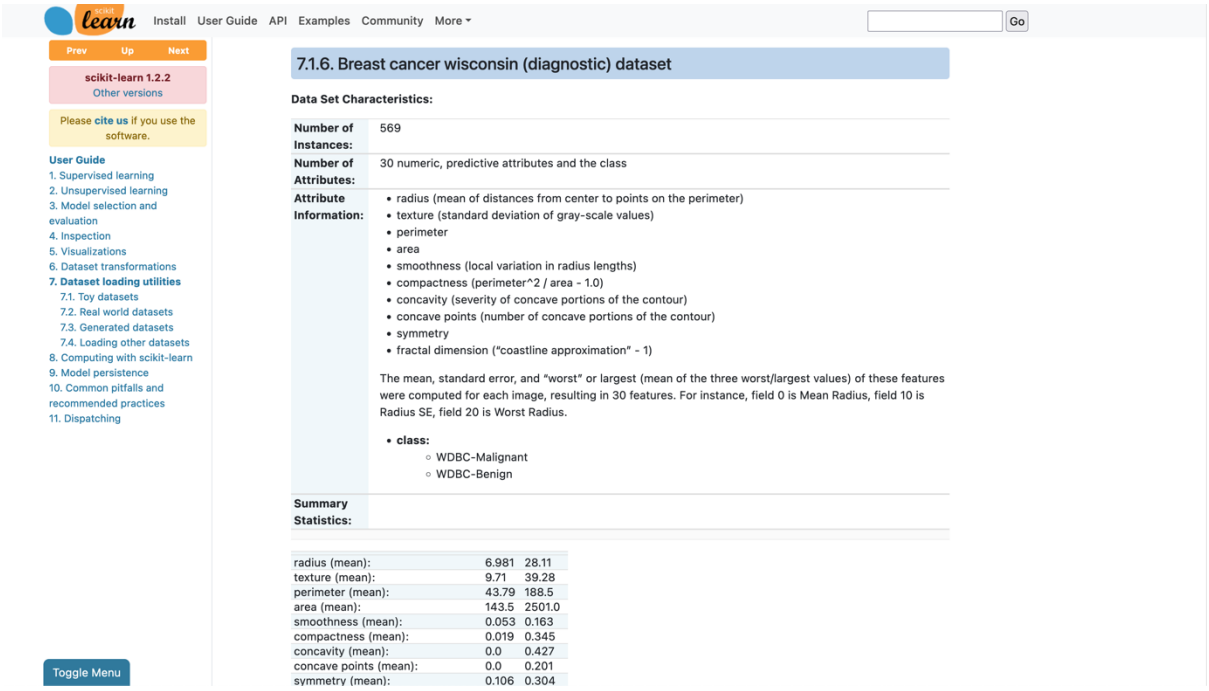
Projekt „KI für alle"
HeiCAD – Heine Center for Artificial Intelligence and Data Science

Vorbereitung des Datensatzes

Um die Funktionsweise der logistischen Regression besser zeigen zu können, arbeiten wir in diesem Video mit dem *Breast Cancer Wisconsin (Diagnostic)* Datensatz. Die Merkmale dieses Datensatzes beschreiben einige Eigenschaften von Zellkernen, die aus der Brustmasse entnommen wurden. Die Eigenschaften der Zellkerne wurden anhand von digitalisierten Mikroskopbildern extrahiert. Die Werte des Zielmerkmals geben an, ob die Zelle aus einem gutartigen Tumor (*benign*) oder aus einem bösartigen Tumor (*malignant*) entnommen wurde. Dabei steht die Merkmalsausprägung 1 für einen gutartigen und 0 für einen bösartigen Tumor.

Quelle [1]

Falls du dich für Details zu diesem Datensatz und für die Beschreibung aller Merkmale interessierst, kannst du diese im User Guide zum *Scikit-learn* finden.



Einblendung User Guide

Quelle [2]

Zuerst aber importieren wir alle Module, die wir in unserem Programm brauchen werden.

```
# Importiere die nötigen Module
from sklearn import linear_model, datasets, model_selection
import plotly.express as px
```

```
import plotly.offline as pyo
import plotly.graph_objects as go
```

Wir laden den *Breast Cancer Wisconsin (Diagnostic)* Datensatz in unser Programm mit der Funktion *load_breast_cancer()*.

Quelle [3]

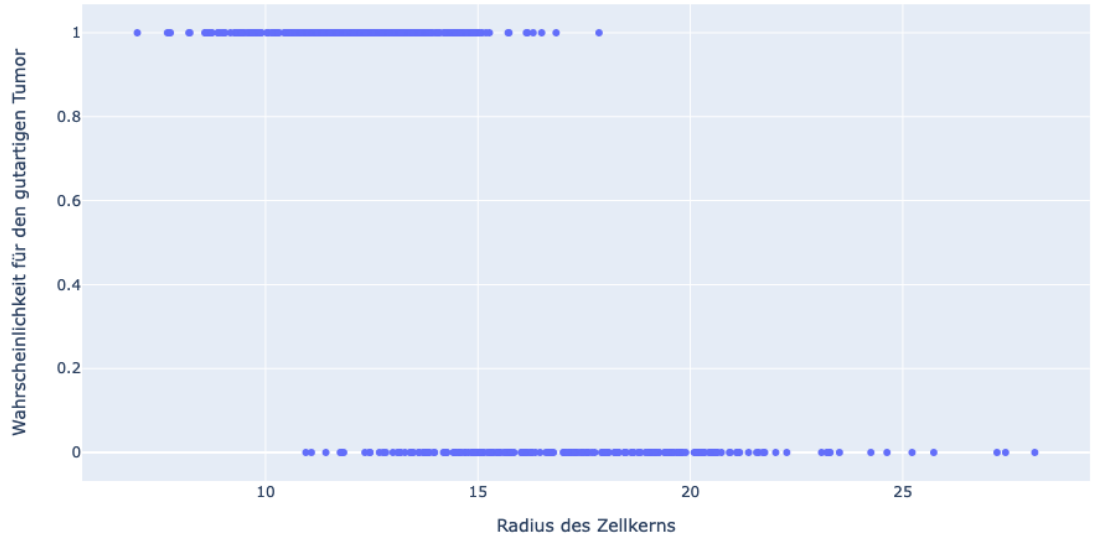
```
# Lade den Datensatz "Breast Cancer Wisconsin (Diagnostic)"
X_breast_cancer, y_breast_cancer = \
    datasets.load_breast_cancer(return_X_y=True)
```

Beim *Breast Cancer Wisconsin (Diagnostic)* Datensatz besteht unsere Aufgabe darin, ein logistisches Regressionsmodell zu entwickeln, das anhand der Merkmale einer Zelle prognostizieren kann, ob diese aus einem gutartigen oder bösartigen Tumor entnommen wurde. Dabei beschränken wir uns auf die Betrachtung des ersten Merkmals, das dem Radius des Zellkerns bzw. der mittleren Entfernung vom Zentrum zu den Punkten auf dem Umkreis entspricht. Wir visualisieren den Radius in Abhängigkeit vom Zielmerkmal.

```
# Wähle nur die erste Spalte des Arrays X_breast_cancer
X_breast_cancer = X_breast_cancer[:, 0]

pyo.init_notebook_mode()
fig1 = px.scatter(x=X_breast_cancer, y=y_breast_cancer)
fig1.update_layout(xaxis_title='Radius des Zellkerns',
                   yaxis_title='Wahrscheinlichkeit für '
                               'den gutartigen Tumor')
fig1.show()
```

In der Abbildung sehen wir, dass es zu einer Anhäufung der „gutartigen“ Beobachtungen bei einem kleineren Radius des Zellkerns und zu einer Anhäufung der „bösartigen“ Beobachtungen bei einem größeren Radius des Zellkerns kommt.



Nun erweitern wir das Datenarray um eine Dimension und teilen dieses in eine Trainings- und eine Testmenge auf.

```
# Erweitere das Datenarray um eine Dimension
X_breast_cancer = X_breast_cancer[:, None]
print(X_breast_cancer.shape)

# Zerlege den Datensatz in eine Training- und eine Testmenge
X_train_bc, X_test_bc, y_train_bc, y_test_bc =\
    model_selection.train_test_split(X_breast_cancer,
                                     y_breast_cancer,
                                     test_size=0.1,
                                     random_state = 45)
```

Logistisches Regressionsmodell

Das logistische Regressionsmodell wird mit der Funktion *LogisticRegression()* des *Scikit-learn*-Untermoduls *Linear Models* erstellt.

Quelle [4]

Während des Trainings versucht der Algorithmus für die Beobachtungen der Trainingsmenge eine optimale Regressionskurve zu berechnen. Er startet mit einer zufälligen Regressionskurve und versucht diese nach und nach an die gegebenen Beobachtungen anzupassen. Dabei wiederholt der Algorithmus immer wieder die gleichen Schritte. Jede solche Wiederholung nennt man auch Iteration. Für manche Daten dauert der Optimierungsprozess der Regressionskurve sehr lange. Wenn man Zeit sparen möchte und sich dafür aber mit einer weniger perfekten Regressionskurve zufrieden gibt, kann man bereits beim Erstellen des Regressionsmodells mit dem Parameter *max_iter* die Anzahl der Iterationen beschränken, die der Optimierungsalgorithmus durchlaufen soll. Wenn der Algorithmus die maximale Anzahl der Iterationen erreicht, aber die optimale Regressionskurve noch nicht gefunden hat, wird neben dem Modell eine entsprechende Warnung ausgegeben. Falls du an der Berechnung einer optimalen Regressionskurve interessiert bist, solltest du die maximale Anzahl der Iterationen mit einer großen Zahl angeben, weil dieser Wert mit 100 voreingestellt ist. Wir weisen dem Parameter *max_iter* einen Wert von 1000 zu. Das Trainieren des Modells erfolgt wieder mit der Funktion *fit()*.

```
# Erstelle ein logistisches Regressionsmodell
log_reg = linear_model.LogisticRegression(max_iter = 1000)

# Trainiere das Modell mit der Trainingsmenge
log_reg.fit(X_train_bc, y_train_bc)
```

Die Werte dafür, wie wahrscheinlich Beobachtungen der Testmenge zu den beiden Ausprägungen des Zielmerkmals „gutartig“ oder „bösartig“ gehören, werden mit der Funktion *predict_proba()* vorhergesagt.

```
# Erstelle die Vorhersagen für die gesamte Testmenge
y_pred_bc = log_reg.predict_proba(X_test_bc)
```

In der Ausgabe dieser Funktion erhältst du ein zweidimensionales Array, in dem für jede Beobachtung der Testmenge die Schätzungen für alle Ausprägungen des Zielmerkmals, geordnet nach der Bezeichnung der Ausprägungen, stehen. In unserem Fall steht der erste Wert dafür, wie wahrscheinlich die Beobachtung zu der Ausprägung null des Zielmerkmals – in unserem Fall zu „bösartig“ – gehört. Der zweite Wert gibt entsprechend die Wahrscheinlichkeit der Zugehörigkeit der Beobachtung zu der Ausprägung eins – in unserem Fall zu „gutartig“ – an.

Um sehen zu können, wie gut unser Regressionsmodell die Wahrscheinlichkeiten der Zugehörigkeit der neuen Beobachtungen zu allen Ausprägungen des Zielmerkmals schätzen kann, geben wir exemplarisch für die erste Beobachtung der Testmenge den Radius des Zellkerns, die tatsächliche und die durch unser Modell geschätzten Wahrscheinlichkeiten aus, dass die entsprechende Zelle aus einem bösartigen bzw. einem gutartigen Tumor entnommen wurde.

```
# Gebe den Wert des beschreibenden Merkmals für
# die erste Beobachtung der Testmenge aus
print("Der Radius des Zellkerns der ersten "
      "Beobachtung aus der Testmenge:", X_test_bc[0,0])

# Gebe den tatsächlichen Wert des Zielmerkmals für
# die erste Beobachtung der Testmenge aus
print("Die tatsächliche Zugehörigkeit zu einem bösartigen "
      "bzw. gutartigen Tumor der ersten Beobachtung "
      "aus der Testmenge:", y_test_bc[0])

# Gebe die geschätzten Wahrscheinlichkeiten der Zugehörigkeit
# der ersten Beobachtung aus der Testmenge zu allen
# Ausprägungen des Zielmerkmals aus
print("Die geschätzten Wahrscheinlichkeiten der Zugehörigkeit "
      "der ersten Beobachtung aus der Testmenge zu einem bösartigen "
      "bzw. gutartigen Tumor:", y_pred_bc[0])
```

In der Ausgabe kannst du sehen, dass der geschätzte Wert (0,98) sehr nah an dem tatsächlichen Wert (1) liegt. Das spricht erstmal für eine sehr gute Fähigkeit unseres logistischen Regressionsmodells, die Zugehörigkeit der Zellen zu den gutartigen bzw. bösartigen Tumoren zu schätzen. Aber wenn wir die Regressionskurve andeutungsweise mithilfe der Zugehörigkeitswerten zu der Ausprägung „gutartig“ visualisieren, sehen wir, dass die Wahrscheinlichkeiten einiger Beobachtungen im Bereich zwischen 0,4 und 0,6 liegen und somit eine große Abweichung zum tatsächlichen Wert haben.

```
fig2 = px.scatter(x=X_test_bc[:,0], y=y_pred_bc[:,1],
                  color_discrete_sequence=['red'])
fig3 = go.Figure(data=fig1.data + fig2.data)
fig3.update_layout(xaxis_title='Radius des Zellkerns',
                   yaxis_title='Wahrscheinlichkeit für '
                               'den gutartigen Tumor')
fig3.show()
```

Take-Home Message

In diesem Video hast du gelernt, wie du in Python ein logistisches Regressionsmodell erstellen und es für einen gegebenen Datensatz trainieren kannst. Außerdem weißt du jetzt, wie du mit Hilfe deines Modells Werte für das Zielmerkmal für neue Beobachtungen schätzen kannst.

Quellen

Quelle [1] Street, W. N., Wolberg, W. H., & Mangasarian, O. L. (1993). Nuclear feature extraction for breast tumor diagnosis. In Proc. SPIE 1905, *Biomedical Image Processing and Biomedical Visualization* (pp. 861–870). https://doi.org/10.1117/12.148698

Quelle [2] Scikit-learn (2022). Breast cancer Wisconsin (diagnostic) dataset. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/datasets/toy_dataset.html#breast-cancer-dataset

Quelle [3] Scikit-learn (2022). Datasets. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html

Quelle [4] Scikit-learn (2022). Linear Models. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model

Weiterführendes Material

Nguyen, C. N., & Zeigermann, O. (2021). *Machine Learning -- kurz & gut: Eine Einführung mit Python, Pandas und Scikit-Learn*. O'Reilly

Disclaimer

Transkript zu dem Video „Woche 06: Programmierung – Logistische Regression mit Scikit-learn", Ludmila Himmelspach.