Introduktion til Logistisk Regression i Python

Af Henrik Sterner (henrik.sterner@gmail.com)

I det følgende introduceres logistisk regression og hvordan vi kan bruge det i Python.

Hvad er logistisk regression?

Logistisk regression er en metode til at forudsige sandsynligheden for at en given observation tilhører en given klasse. Eksempler er mange: * Er en given e-mail spam? * Er en given transaktion svindel? * Er en given patient syg? * Er en given kunde tilbøjelig til at købe et produkt? * Er en given kunde tilbøjelig til at opsige sit abonnement? * Har en given kunde en høj sandsynlighed for at købe et produkt?

Hvorfor logistisk regression?

Logistisk regression er en simpel og hurtig metode til at forudsige sandsynligheden for at en given observation tilhører en given klasse.
* Logistisk regression er hurtig at træne og kan derfor bruges til at træne modeller på store datasæt.
* Logistisk regression er hurtig at bruge til at forudsige sandsynligheder for nye observationer. * Modsat lineær regression er logistisk regression ikke begrænset til at forudsige værdier mellem 0 og 1. * Logistisk regression er en såkaldt supervised learning metode, dvs. at vi skal bruge træningsdata, hvor vi kender den sande værdi af y.

Hvordan virker logistisk regression?

Logistisk regression er en udvidelse af lineær regression. * Lineær regression:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$$

* Logistisk regression:

$$y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n)}},$$

hvor y er sandsynligheden for at en given observation tilhører en given klasse. Her er e Eulers tal, og $\beta_0, \beta_1, \beta_2, ..., \beta_n$ er parametre, som skal bestemmes ud fra træningsdata.

Eksempel: Overlever Titanic?

- I det følgende vil vi bruge logistisk regression til at forudsige om en given passager på Titanic overlever eller ej.
- Vi betragter først nogler af de variable, som vi vil bruge til at forudsige overlevelse:
- Køn: 0 = mand, 1 = kvinde
- Alder: Passagerens alder
- Klasse: 1 = 1. klasse, 2 = 2. klasse, 3 = 3. klasse
- Søskende/ægtefælle: Antal søskende/ægtefæller ombord
- Forældre/børn: Antal forældre/børn ombord
- Pris: Billetpris
- Havne: 0 = Southampton, 1 = Cherbourg, 2 = Queenstown
- Kajplads: 0 = kajplads ukendt, 1 = kajplads kendt

Eksempel: Overlever Titanic? Indlæs data

- Vi indlæser først data fra titanic.csv:

```
import numpy as np

data = np.loadtxt('titanic.csv', delimiter=',', skiprows=1)
```

Eksempel: Overlever Titanic? Første kig på data

- Vi kan nu kigge på de første 5 rækker af data. Her kolonnerne:
- 0: Overlever (0 = nej, 1 = ja)
- 1: Klasse
- 2: Alder
- 3: Søskende/ægtefælle
- 4: Forældre/børn
- 5: Pris
- 6: Havne

Eksempel: Overlever Titanic? Første kig på data

```
print(data[0:5,:])
```

```
[[ 0.      3.     22.      1.      0.      7.25    2.
 [ 1.      1.     38.      1.      0.     71.2833  0.
 [ 1.      3.     26.      0.      0.      7.925   2.
 [ 1.      1.     35.      1.      0.     53.1     2.
 [ 0.      3.     35.      0.      0.      8.05    2.
```

Det første tal i hver række er om passageren overlever (0 = nej, 1 = ja). De efterfølgende tal er de variable, som vi vil bruge til at forudsige overlevelse.

Eksempel: Overlever Titanic? Opdel data i trænings- og testdata

- Vi opdeler nu data i trænings- og testdata:

```
from sklearn.model_selection import train_test_split

X = data[:,1:]
y = data[:,0]
X_train, X_test, y_train, y_test = train_test_split(X, y, t
```

Eksempel: Overlever Titanic? Træn logistisk regression

- Vi træner nu en logistisk regression model:

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)
```

Eksempel: Overlever Titanic? Forudsætninger for logistisk regression

- Vi kan nu kigge på de estimerede parametre:

```
print(model.intercept_)
print(model.coef_)
```

```
[1.00000000e-06]
[[-0.9419627  -0.02945592 -0.26445592 -0.08145592  0.004544
  -0.08145592]]
```

Eksempel: Overlever Titanic? Estimeret sandsynlighed for overlevelse

Vi ser, at den estimerede sandsynlighed for at overleve er: -

$$y = \frac{1}{1 + e^{-(1.00000000e-06-0.9419627x_1-0.02945592x_2-0.26445592x_3-0.08145592x_4+}}$$

Her er x_1 køn, x_2 alder, x_3 klasse, x_4 søskende/ægtefælle, x_5 forældre/børn, x_6 pris og x_7 havne.

Vil man vide hvorledes logistisk regression virker - så kig med videre :-)

I de følgende slides vil vi se på hvordan logistisk regression virker både fra et matematisk og et programmeringsmæssigt perspektiv.

Det er ikke et krav for at kunne bruge den, men det er altid godt at vide hvordan tingene virker fra bunden :-).

Matematisk baggrund for logistisk regression

- Vi kan skrive logistisk regression som:
- $$y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n)}}$$
- Her er y sandsynligheden for at en given observation tilhører en given klasse.
- Her er e Eulers tal, og $\beta_0, \beta_1, \beta_2, ..., \beta_n$ er parametre, som skal bestemmes ud fra træningsdata.

Omskrivning af logistisk regression

Vi kan omskrive logistisk regression til:

$$\frac{y}{1-y} = e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n x_n}$$

Vi kan nu tage logaritmen på begge sider:

$$\log\left(\frac{y}{1-y}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

Vi ser, at venstresiden er logit-funktionen, som er defineret som:

$$\log\left(\frac{y}{1-y}\right)$$

Logit-funktionen

- Logit-funktionen er defineret som:
- $$\log\left(\frac{y}{1-y}\right)$$
- Her er y sandsynligheden for at en given observation tilhører en given klasse.
- Bruges til at omskrive lineær regression til logistisk regression.

Omskrivning af logistisk regression

Vi kan omskrive logistisk regression til:

$$\frac{y}{1-y} = e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n x_n}$$

Vi kan nu tage logaritmen på begge sider:

$$\log\left(\frac{y}{1-y}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

Vi ser, at venstresiden er logit-funktionen, som er defineret som:

$$\log\left(\frac{y}{1-y}\right)$$

Vi kan nu omskrive logit-funktionen til:

$$\frac{y}{1-y} = e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n x_n}$$

$$y = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n x_n}}{1+e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n x_n}}$$

Bestemmelse af parametre

- Vi skal nu bestemme parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$ ud fra træningsdata.
- Vi kan ikke bestemme parametrene ved at løse ligningen:
- $$y = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}}{1 + e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}}$$
- Det skyldes, at ligningen ikke kan løses analytisk.
- Vi skal derfor bruge en numerisk metode til at bestemme parametrene.

Numerisk bestemmelse af parametre

- En numerisk metode er en metode, hvor vi bruger en computer til at bestemme parametrene.
- Vi kan bruge en såkaldt gradient descent metode til at bestemme parametrene.

Gradient descent

- Gradient descent er en metode til at finde minimumspunktet for en funktion.
- Vi starter med at gætte på et punkt, hvor vi tror, at minimumspunktet er.
- Vi finder nu den afledede af funktionen i dette punkt.
- Vi flytter nu vores gæt i retningen af den afledede.
- Vi gentager nu processen, indtil vi er tæt nok på minimumspunktet.

Gradient descent og logistisk regression

- Vi kan bruge gradient descent til at bestemme parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$ i logistisk regression på følgende måde:
- Gæt på parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$.
- Beregn sandsynligheden for at en given observation tilhører en given klasse.
- Beregn den afledede af logistisk regression i forhold til parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$.
- Opdater parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$ i retningen af den afledede.
- Gentag processen, indtil vi er tæt nok på minimumspunktet.

Gradient descent: Gæt på parametre

Først gætter vi på parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$:

$$\beta_0 = 0, \beta_1 = 0, \beta_2 = 0, ..., \beta_n = 0$$

Vi beregner nu sandsynligheden for at en given observation tilhører en given klasse:

$$y = \frac{e^{0+0x_1+0x_2+...+0x_n}}{1 + e^{0+0x_1+0x_2+...+0x_n}}$$

Gradient descent: Beregn den afledede

Vi beregner nu den afledede af logistisk regression i forhold til parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$:

$$\frac{\partial y}{\partial \beta_0} = \frac{e^{0+0x_1+0x_2+...+0x_n}}{1+e^{0+0x_1+0x_2+...+0x_n}} - \frac{e^{0+0x_1+0x_2+...+0x_n} e^{0+0x_1+0x_2+...+0x_n}}{(1+e^{0+0x_1+0x_2+...+0x_n})^2}$$

$$\frac{\partial y}{\partial \beta_1} = \frac{e^{0+0x_1+0x_2+...+0x_n} x_1}{1+e^{0+0x_1+0x_2+...+0x_n}} - \frac{e^{0+0x_1+0x_2+...+0x_n} e^{0+0x_1+0x_2+...+0x_n} x_1}{(1+e^{0+0x_1+0x_2+...+0x_n})^2}$$

$$\frac{\partial y}{\partial \beta_2} = \frac{e^{0+0x_1+0x_2+...+0x_n} x_2}{1+e^{0+0x_1+0x_2+...+0x_n}} - \frac{e^{0+0x_1+0x_2+...+0x_n} e^{0+0x_1+0x_2+...+0x_n} x_2}{(1+e^{0+0x_1+0x_2+...+0x_n})^2}$$

...

$$\frac{\partial y}{\partial \beta_n} = \frac{e^{0+0x_1+0x_2+...+0x_n} x_n}{1+e^{0+0x_1+0x_2+...+0x_n}} - \frac{e^{0+0x_1+0x_2+...+0x_n} e^{0+0x_1+0x_2+...+0x_n} x_n}{(1+e^{0+0x_1+0x_2+...+0x_n})^2}$$

Gradient descent: Opdater parametre

Vi opdaterer nu parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$ i retningen af den afledede:

$$\beta_0 = \beta_0 - \alpha \frac{\partial y}{\partial \beta_0}$$

$$\beta_1 = \beta_1 - \alpha \frac{\partial y}{\partial \beta_1}$$

$$\beta_2 = \beta_2 - \alpha \frac{\partial y}{\partial \beta_2}$$

$$...$$

$$\beta_n = \beta_n - \alpha \frac{\partial y}{\partial \beta_n}$$

Her er α en konstant, som bestemmer hvor hurtigt vi bevæger os mod minimumspunktet. Denne konstant kaldes for læringsraten eller learning rate.

Gradient descent: Gentag processen

Vi gentager nu processen, indtil vi er tæt nok på minimumspunktet. Dvs. vi gentager processen, indtil den afledede er tæt på 0:

$$\frac{\partial y}{\partial \beta_0} \approx 0$$

$$\frac{\partial y}{\partial \beta_1} \approx 0$$

$$\frac{\partial y}{\partial \beta_2} \approx 0$$

$$\ldots$$

$$\frac{\partial y}{\partial \beta_n} \approx 0$$

Gradient descent: Partielt afledede skal beregnes mange gange

- Vi ønsker, at den afledede skal være tæt på 0.
- Det indikerer, at vi er tæt på minimumspunktet.
- Vi skal derfor beregne den partielt afledede mange gange.
- Det er meget tidskrævende.
- Vi kan bruge en såkaldt Newton-Raphson metode til at optimere gradient descent.

Newton-Raphson metoden

- Newton-Raphson metoden er en metode til at finde minimumspunktet for en funktion. Kort fortalt:
- Vi starter med at gætte på et punkt, hvor vi tror, at minimumspunktet er.
- Vi finder nu den afledede af funktionen i dette punkt.
- Vi finder nu den anden afledede af funktionen i dette punkt.
- Vi flytter nu vores gæt i retningen af den afledede divideret med den anden afledede.
- Vi gentager nu processen, indtil vi er tæt nok på minimumspunktet.
- Vi kan bruge Newton-Raphson metoden til at bestemme parametrene i logistisk regression.

Matematisk baggrund for Newton-Raphson metoden

- Vi kan bruge Newton-Raphson metoden til at bestemme parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$ i logistisk regression på følgende måde:
- Gæt på parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$.
- Beregn sandsynligheden for at en given observation tilhører en given klasse. Dvs.:

$$y = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}}{1 + e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}}$$

Newton-Raphson metoden: Den første afledede

Beregn den afledede af logistisk regression i forhold til parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$. Dvs.

$$\frac{\partial y}{\partial \beta_0} = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}}{1+e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}} - \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n} e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+}}{(1+e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n})^2}$$

...

$$\frac{\partial y}{\partial \beta_n} = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n} x_n}{1+e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}} - \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n} e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+}}{(1+e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n})^2}$$

Newton-Raphson metoden: Den anden afledede

Beregn den anden afledede af logistisk regression i forhold til parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$. Dvs.

$$\frac{\partial^2 y}{\partial \beta_0^2} = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n} e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}}{\left(1+e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}\right)^2}$$

$$-\frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n} e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n} e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}}{\left(1+e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}\right)^3}$$

$$...$$

$$\frac{\partial^2 y}{\partial \beta_n^2} = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n} e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n} x_n^2}{\left(1+e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}\right)^2}$$

$$-\frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n} e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n} e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n} x_n^2}{\left(1+e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}\right)^3}$$

Newton-Raphson metoden: Opdater parametre

Vi opdaterer nu parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$ i retningen af den afledede divideret med den anden afledede:

$$\beta_0 = \beta_0 - \frac{\frac{\partial y}{\partial \beta_0}}{\frac{\partial^2 y}{\partial \beta_0^2}}$$

$$\beta_1 = \beta_1 - \frac{\frac{\partial y}{\partial \beta_1}}{\frac{\partial^2 y}{\partial \beta_1^2}}$$

$$...$$

$$\beta_n = \beta_n - \frac{\frac{\partial y}{\partial \beta_n}}{\frac{\partial^2 y}{\partial \beta_n^2}}$$

Newton-Raphson metoden: Gentag processen

Vi gentager nu processen, indtil vi er tæt nok på minimumspunktet. Dvs. vi gentager processen, indtil den afledede er tæt på 0:

$$\frac{\partial y}{\partial \beta_0} \approx 0$$

$$\frac{\partial y}{\partial \beta_1} \approx 0$$

$$\frac{\partial y}{\partial \beta_2} \approx 0$$

$$\ldots$$

$$\frac{\partial y}{\partial \beta_n} \approx 0$$

Logistisk regression i Python fra bunden

- Vi kan nu implementere logistisk regression i Python fra bunden.
- Vi starter med at importere numpy:

```
import numpy as np
```

Logistisk regression i Python fra bunden

- Vi kan nu definere en funktion, som beregner sandsynligheden for at en given observation tilhører en given klasse:

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

Denne funktion kaldes for sigmoid-funktionen og er meget vigtig i logistisk regression:

$$\frac{1}{1 + e^{-x}}$$

- Her er x en vilkårlig værdi. - Sigmoid-funktionen er en special case af logit-funktionen, hvor y er en vilkårlig værdi. - Sigmoid-funktionen spiller også en central rolle i kunstige neurale netværk, som aktiveringsfunktion.

Logistisk regression i Python fra bunden

Efter at have defineret sigmoid-funktionen kan vi nu definere en funktion, som beregner sandsynligheden for at en given observation tilhører en given klasse:

```
def predict(X, beta):
    return sigmoid(np.dot(X, beta))
```

- Her er X en matrix med observationer og variable.
- beta er en vektor med parametre.

Logistisk regression i Python fra bunden

Vi kan nu definere en funktion, som beregner den afledede af logistisk regression i forhold til parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$:

```
def gradient(X, y, beta):
    return np.dot(X.T, y - predict(X, beta))
```

- Her er X en matrix med observationer og variable.
- y er en vektor med sande værdier.
- beta er en vektor med parametre.

Logistisk regression i Python fra bunden

Vi kan nu definere en funktion, som beregner den anden afledede af logistisk regression i forhold til parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$:

```
def hessian(X, beta):
    p = predict(X, beta)
    return np.dot(X.T, X * p * (1 - p))
```

- Her er X en matrix med observationer og variable.
- beta er en vektor med parametre. Vi får en matrix og ikke en vektor, da vi beregner den anden afledede i forhold til alle parametre samtidig. Matricen kaldes for Hessisk matricen opkaldt efter den tyske matematiker Ludwig Otto Hesse.

Logistisk regression i Python fra bunden

Vi kan nu definere en funktion, som bestemmer parametrene $\beta_0, \beta_1, \beta_2, ..., \beta_n$ i logistisk regression:

```
def fit(X, y, iterations=100):
    beta = np.zeros(X.shape[1])
    for i in range(iterations):
        beta -= np.linalg.solve(hessian(X, beta), gradient(
    return beta
```

- Her er X en matrix med observationer og variable.
- y er en vektor med sande værdier.
- iterations er antallet af gange, som vi skal gentage processen. Default er 100.
- beta er en vektor med parametre.

Afprøvning af logistisk regression i Python fra bunden

- Vi kan nu afprøve logistisk regression i Python fra bunden.
- Vi starter med at indlæse data:

```
import numpy as np

data = np.loadtxt('titanic.csv', delimiter=',', skiprows=1)
```

Afprøvning af logistisk regression i Python fra bunden

- Vi kan nu opdele data i trænings- og testdata:

```
from sklearn.model_selection import train_test_split

X = data[:,1:]
y = data[:,0]
X_train, X_test, y_train, y_test = train_test_split(X, y, t
```

Afprøvning af logistisk regression i Python fra bunden

- Vi kan nu træne en logistisk regression model:

```
beta = fit(X_train, y_train)
```

- Vi kan nu beregne sandsynligheden for at en given observation tilhører en given klasse:

```
y_pred = predict(X_test, beta)
```

Afprøvning af logistisk regression i Python fra bunden

- Vi kan nu beregne nøjagtigheden af modellen:

```
from sklearn.metrics import accuracy_score

print(accuracy_score(y_test, y_pred.round()))
```

```
0.8100558659217877
```

Her er nøjagtigheden 81%.