ISP AUFGABE 3

Gruppe 3, Team 5

06.06.2017

Dimitri Meier, Saeed Shanidar, Andreas Berks
dimitri.meier@haw-hamburg.de,
saeed.shanidar@haw-hamburg.de,
andreas.berks@haw-hamburg.de

Inhaltsverzeichnis

0. Allgemeine Definitionen

0.1. Konkatenation

Wir definieren eine Funktion $Konkatenation$ wie folgt:

$$Konkatenation : (\mathbb{R}^{m,1})^n \to \mathbb{R}^{m,n}, x \mapsto y$$

$$\forall i \in [1, m], j \in [1, n] : y_{i,j} = (x_i)_j$$

Beispiel:
Seien zwei Vektoren x_1 und x_2 gegeben:

$$x_1 = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \qquad x_2 = \begin{pmatrix} d \\ e \\ f \end{pmatrix}$$

Damit ergibt sich:

$$Konkatenation(\{x_1, x_2\}) = \begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix}$$

Implementation in Python:

```python
def concatenate(M1, M2):
    return np.concatenate([M1, M2], axis = horizontal_axis)
```

0.2. Bias

Wir definieren eine Funktion $Bias$ wie folgt:

$Bias : \mathbb{R}^{m,n} \rightarrow \mathbb{R}^{m,1}, x \mapsto y$

$\forall i \in [1, m] : y_{i,1} = 1$

Beispiel:
Sei eine Matrix X gegeben:

$$X = \begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix}$$

Damit ergibt sich:

$$Bias(X) = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Implementation in Python:

Wir definieren zuerst eine Hilfsfunktion, welche und die Anzahl der Zeilen und die Anzahl der Spalten der Matrix M liefert:

```
def get_dimensions(M):
    return np.shape(M)
```

Nun definieren wir eine Funktion zum Erzeugen des Bias:

```
def create_bias(M):
    m, n = get_dimensions(M)
    return np.ones((m,1))
```

Weiterhin definieren wir eine Funktion zum Hinzufügen eines Bias:

```
def add_bias(M):
    bias = create_bias(M)
    return concatenate(bias, M)
```

1. Lineare Regression

1.1. Training mittels abgeschlossener Lösung

1.1.1. Definition der Trainingsdaten

Gegeben sind die folgenden Trainingsdaten:

x_1	y
0.86	2.49
0.09	0.83
-0.85	-0.25
0.87	3.1
-0.44	0.87
-0.43	0.02
-1.1	-0.12
0.4	1.81
-0.96	-0.83
0.17	0.43

$$x_1 = \begin{pmatrix} 0.86 \\ 0.09 \\ -0.85 \\ 0.87 \\ -0.44 \\ -0.43 \\ -1.1 \\ 0.4 \\ -0.96 \\ 0.17 \end{pmatrix} \qquad y = \begin{pmatrix} 2.49 \\ 0.83 \\ -0.25 \\ 3.1 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{pmatrix}$$

Wir definieren nun eine Matrix X welche sämtliche Trainingsdaten beinhaltet.
Dazu konkatenieren wir sämtliche Spaten der Trainingsdaten:

$$Spalten = \{x_1\}$$

$$X = Konkatenation(Spalten)$$

1.1.2. Training des Modells

Im Folgenden definieren nun eine Methode $fit(X, y)$ zum Training des Modells anhand von gegebenen Trainingsdaten.

Zuerst fügen wir der Matrix X den Bias x_0 hinzu.
Die entstehende Matrix bezeichnen wir als X_{Bias}.

$$x_0 = Bias(X)$$

$$X_{Bias} = Konkatenation(\{x_0\} \cup Spalten)$$

Die entsprechenden Matrizen X und X_{Bias} stehen wie folgt aus:

$$X = \begin{pmatrix} 0.86 \\ 0.09 \\ -0.85 \\ 0.87 \\ -0.44 \\ -0.43 \\ -1.1 \\ 0.4 \\ -0.96 \\ 0.17 \end{pmatrix} \qquad X_{Bias} = \begin{pmatrix} 1.0 & 0.86 \\ 1.0 & 0.09 \\ 1.0 & -0.85 \\ 1.0 & 0.87 \\ 1.0 & -0.44 \\ 1.0 & -0.43 \\ 1.0 & -1.1 \\ 1.0 & 0.4 \\ 1.0 & -0.96 \\ 1.0 & 0.17 \end{pmatrix}$$

Nun bestimmen wir die Gewichte θ mittels abgeschlossener Lösung:

$$\theta = ({X_{Bias}}^T X_{Bias})^{-1} {X_{Bias}}^T y$$

$$\theta = \begin{pmatrix} 1.05881340999 \\ 1.61016841718 \end{pmatrix}$$

Implementation in Python:

Wir definieren zuerst Hilfsfunktionen, um Matrizen zu multiplizieren, zu transponieren und zu invertieren:

```python
def multiply(M1, M2):
    return np.dot(M1, M2)

def transpose(M):
    return np.transpose(M)

def inverse(M):
    return np.linalg.inv(M)
```

Nun definieren wir eine Funktion zum Bestimmen der Gewichte:

```python
def calcThetaClosedForm(X, y):
    XT = transpose(X)
    result = multiply(XT, X)
    result = inverse(result)
    result = multiply(result, XT)
    result = multiply(result, y)
    return result
```

Wir speichern uns die Werte von θ und nutzen diese in der Definition von $predict(X_{Test})$.

Damit ist die Definition der Methode $fit(X, y)$ zum Training des Modells anhand von gegebenen Trainingsdaten abgeschlossen.

1.1.3. Aufstellen der Hypothese

Weiterhin definieren wir eine Methode $predict(X_{Test})$ zur numerischen Vorhersage der Zielvariable.

Es gilt für X_{Test} :

$$X_{Test} = Konkatenation(\{x_{Test1}\})$$
$$Spalten_{Test} = \{x_{Test1}\}$$

Zuerst fügen wir der Matrix X_{Test} den Bias x_{0Test} hinzu.
Die entstehende Matrix bezeichnen wir als $X_{Test\,Bias}$.

$$x_{Test0} = Bias(X_{Test})$$
$$X_{Test\,Bias} = Konkatenation(\{X_{Test\,Bias}\} \cup Spalten_{Test})$$

Nun berechnen wir die Hypothese $h(X_{TestBias})$:

$$h(X_{TestBias}) = X_{TestBias}\theta$$

$$h(X_{TestBias})_{i,k} = \sum_{j=0}^{n} X_{TestBias\,i,j} \cdot \theta_{j,k}$$

Implementation in Python:

```python
def calcHypothesis(theta, X):
    thetaT = transpose(theta)
    result = multiply(X, thetaT)
    return result
```

Das Ergebnis der Hypothese $h(X_{TestBias})$ geben wir als Ergebnis zurück.

Damit ist die Definition der Methode $predict(X_{Test})$ zur numerischen Vorhersage der Zielvariable abgeschlossen.

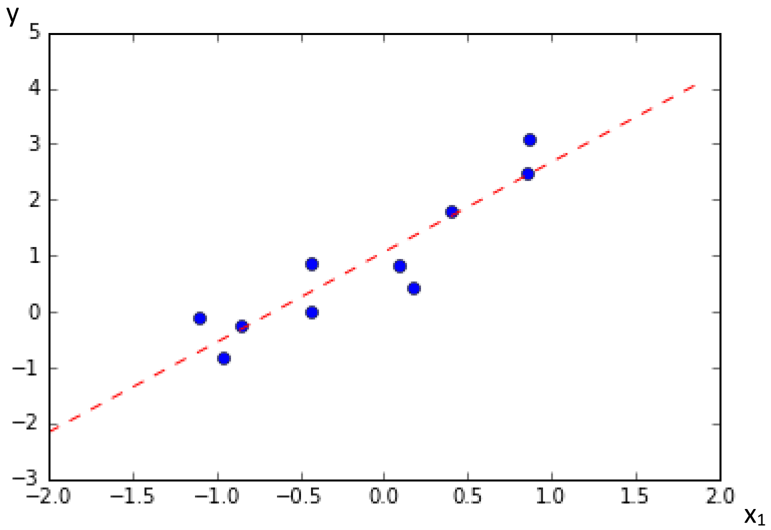
1.1.4. Ergebnis

Zur Erinnerung: Gegeben waren die folgenden Trainingsdaten:

x_1	y
0.86	2.49
0.09	0.83
-0.85	-0.25
0.87	3.1
-0.44	0.87
-0.43	0.02
-1.1	-0.12
0.4	1.81
-0.96	-0.83
0.17	0.43

$$x_1 = \begin{pmatrix} 0.86 \\ 0.09 \\ -0.85 \\ 0.87 \\ -0.44 \\ -0.43 \\ -1.1 \\ 0.4 \\ -0.96 \\ 0.17 \end{pmatrix} \qquad y = \begin{pmatrix} 2.49 \\ 0.83 \\ -0.25 \\ 3.1 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{pmatrix}$$

Das Ergebnis der Hypothese sieht wie folgt aus:

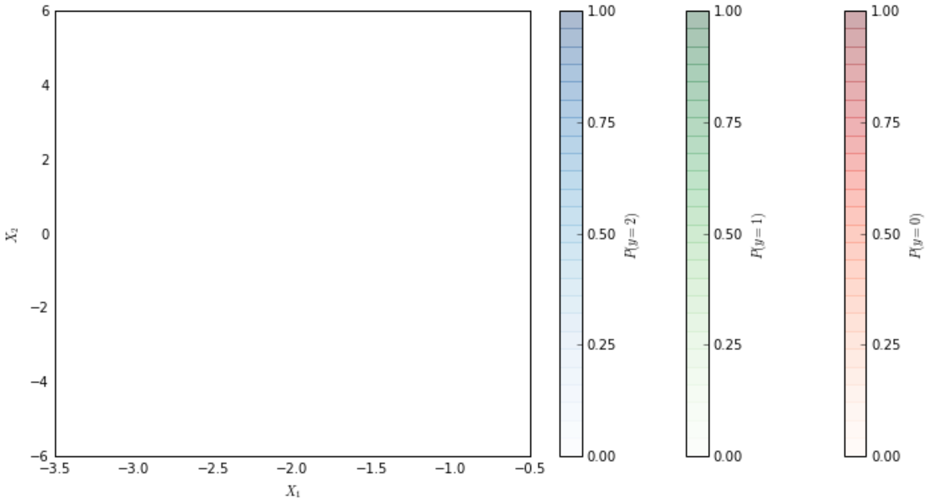


1.1.5. Beantwortung der Fragen

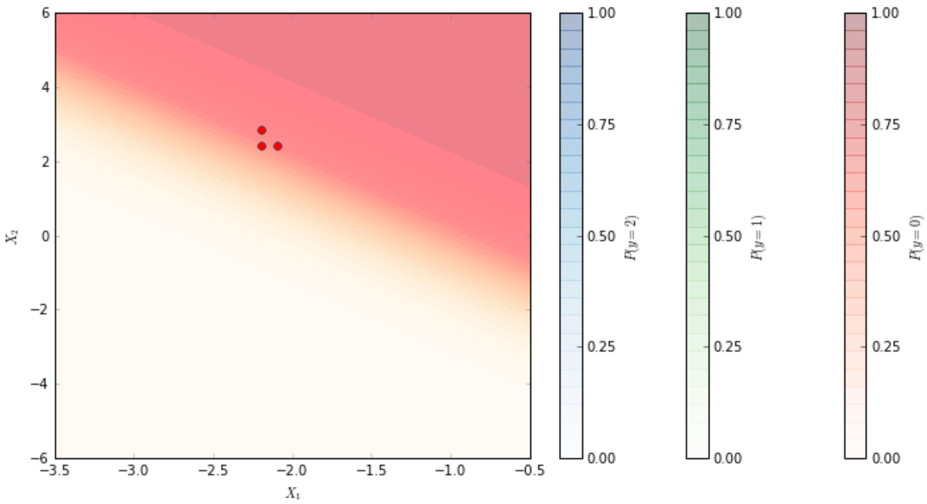
…

2.1.5. Beantwortung der Fragen

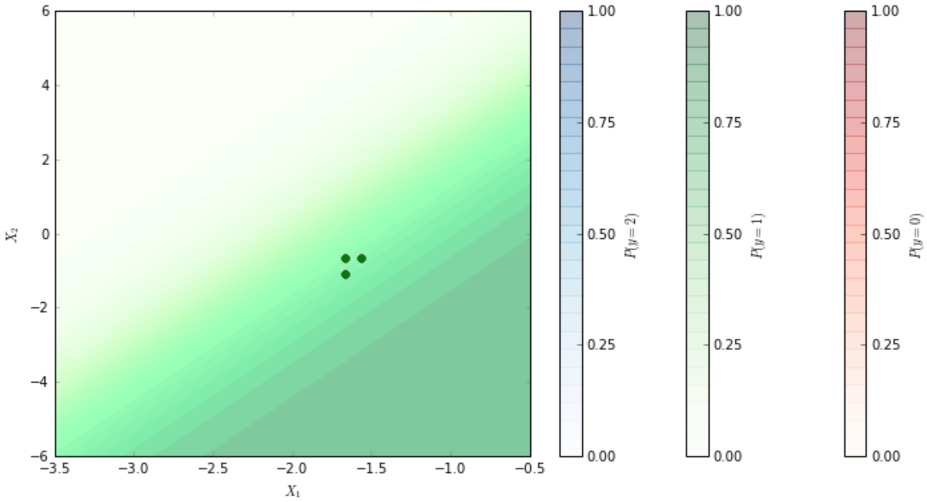
Darstellung keiner Klasse:



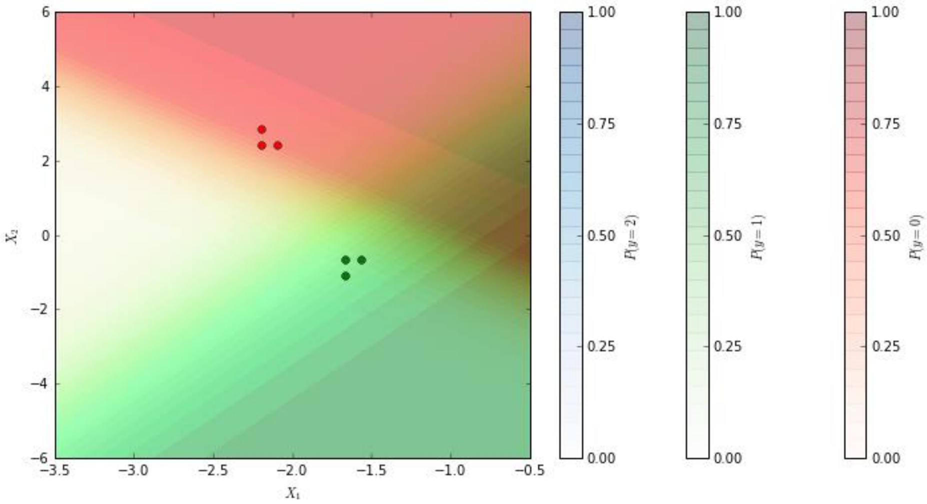
Darstellung der Klasse 0:



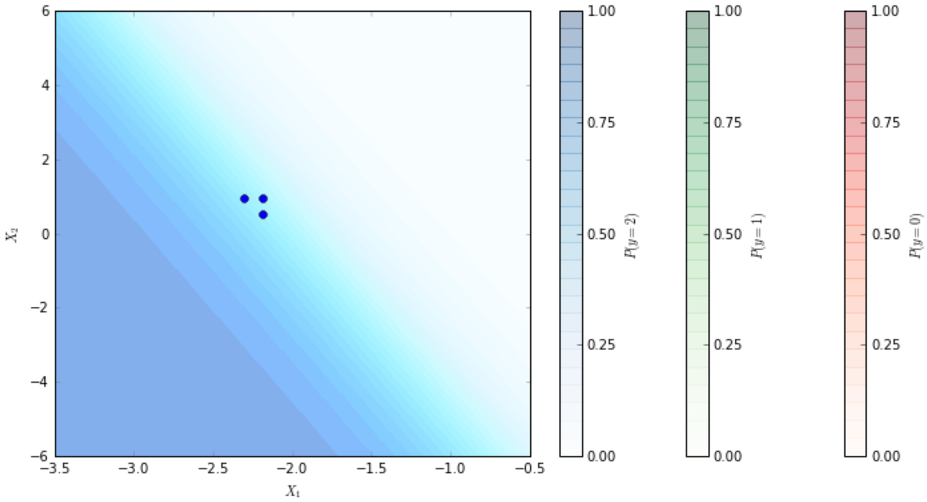
Darstellung der Klasse 1:



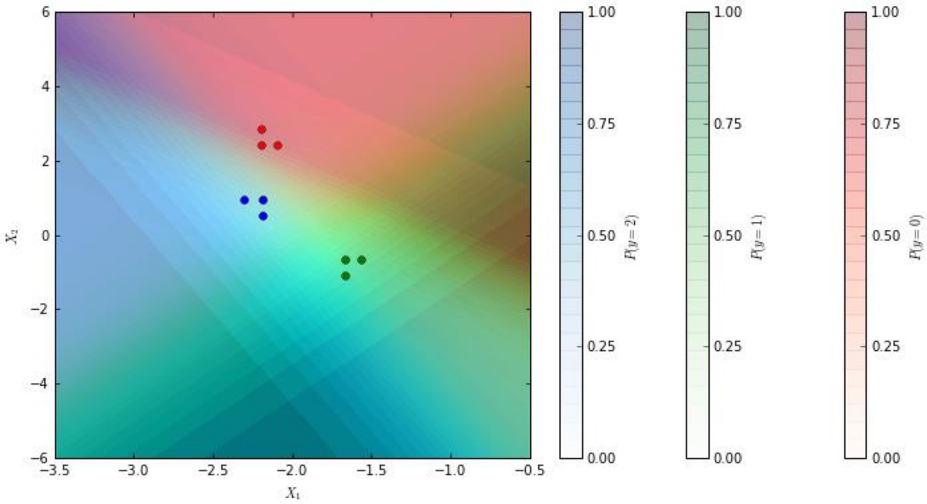
Darstellung der Klassen 0 und 1:



Darstellung der Klasse 3:



Darstellung der Klassen 1, 2 und 3:



...