(Binäre) Logistische Regression

Katja Markert

Institut für Computerlinguistik
Universität Heidelberg
markert@cl.uni-heidelberg.de

January 9, 2023

Heute

1. Erinnerung: Überwachtes maschinelles Lernen
2. (Binäre) Logistische Regression: **Diskriminatives** Klassifikationsverfahren des überwachten maschinellen Lernens, beruht auf **Skalarprodukt** von Merkmalsvektor mit Gewichtsvektor zur Klassifikation.
3. Gewichtsvektoren zeigen, welche Merkmale besonders wichtig sind, und für welche Klasse diese sprechen
4. Logistische Regression: wichtig in vielen Bereichen der Naturwissenschaften, Basis von neuronalen Netzen, Basis von Word2vec

Übersicht

Übersicht

Was ist eine Klassifikationsaufgabe?

Klassifikation

Gegeben eine Instanz oder eine Menge von Instanzen, weise der Instanz ein Konzept/Klasse/Kategorie aus einer fixen, diskreten Menge von Konzepten zu. Tue dies auf der Basis von Merkmalen (features/explanatory variables).

Topic, $\geq$2 Kategorien: Hotels, restaurants or movies?

A good budget hotel' .... Price includes breakfast with really nice food. Rooms are modern and of a reasonable size. The centre of Leeds is about a 15 min walk at the most. Hotel has bar area.

Sentiment binär: positiv=1 oder positiv=0

A good budget hotel' .... Price includes breakfast with really nice food. Rooms are modern and of a reasonable size. The centre of Leeds is about a 15 min walk at the most. Hotel has bar area.

Überwachtes Lernen (binär, probabilistisch)

Gegeben eine binäre Klasse y.

- **Trainingsset** von m Instanzen $x^{(j)}$ $(j = 1, \ldots m)$ auf dem man die Klassen per Instanz kennt, also Paare $(x^{(1)}, y^{(1)}), \ldots (x^{(m)}, y^{(m)})$
- Jede Instanz $x^{(j)}$ ist **repräsentiert** als ein Vektor von n Merkmalen $(x_1^{(j)}, x_2^{(j)}, \ldots x_n^{(j)})$, manchmal auch geschrieben als $f_1, \ldots f_n$
- Training: Eine **objective function** für das Training: **minimiert** normalerweise den Fehler auf den Trainingsbeispielen
- Training: Ein Algorithmus, um die *objective function* zu **optimieren**
- Testing: Eine **Klassifikationsfunktion**, die einer Testinstanz x mit Repräsentation $(x_1, x_2, \ldots x_n)$ eine geschätzte Klasse $\tilde{y}$ mithilfe von Berechnung $p(y = 0|x)$ sowie $p(y = 1|x)$ zuweist.

Überwachtes Lernen (binär, probabilistisch)

Gegeben eine binäre Klasse y.

- **Trainingsset** von m Instanzen $x^{(j)}$ $(j = 1, \ldots m)$ auf dem man die Klassen per Instanz kennt, also Paare $(x^{(1)}, y^{(1)}), \ldots (x^{(m)}, y^{(m)})$
- Jede Instanz $x^{(j)}$ ist **repräsentiert** als ein Vektor von n Merkmalen $(x_1^{(j)}, x_2^{(j)}, \ldots x_n^{(j)})$, manchmal auch geschrieben als $f_1, \ldots f_n$
- Training: Eine **objective function** für das Training: **minimiert** normalerweise den Fehler auf den Trainingsbeispielen
- Training: Ein Algorithmus, um die *objective function* zu **optimieren**
- Testing: Eine **Klassifikationsfunktion**, die einer Testinstanz x mit Repräsentation $(x_1, x_2, \ldots x_n)$ eine geschätzte Klasse $\tilde{y}$ mithilfe von Berechnung $p(y = 0|x)$ sowie $p(y = 1|x)$ zuweist.

Überwachtes Lernen (binär, probabilistisch)

Gegeben eine binäre Klasse y.

- **Trainingsset** von m Instanzen $x^{(j)}$ $(j = 1, \ldots m)$ auf dem man die Klassen per Instanz kennt, also Paare $(x^{(1)}, y^{(1)}), \ldots (x^{(m)}, y^{(m)})$
- Jede Instanz $x^{(j)}$ ist **repräsentiert** als ein Vektor von n Merkmalen $(x_1^{(j)}, x_2^{(j)}, \ldots x_n^{(j)})$, manchmal auch geschrieben als $f_1, \ldots f_n$
- Training: Eine **objective function** für das Training: **minimiert** normalerweise den Fehler auf den Trainingsbeispielen
- Training: Ein Algorithmus, um die *objective function* zu **optimieren**
- Testing: Eine **Klassifikationsfunktion**, die einer Testinstanz x mit Repräsentation $(x_1, x_2, \ldots x_n)$ eine geschätzte Klasse $\tilde{y}$ mithilfe von Berechnung $p(y = 0|x)$ sowie $p(y = 1|x)$ zuweist.

Naive Bayes vs. Logistische Regression

NB generativ:

$$\begin{aligned}
\tilde{y} &= \underset{l \in \{0,1\}}{\operatorname{argmax}} p(y = l|x) && (1)\\
&= \underset{l \in \{0,1\}}{\operatorname{argmax}} \frac{p(x|y = l)p(y = l)}{p(x)} && (2)\\
&= \underset{l \in \{0,1\}}{\operatorname{argmax}} p(x|y = l)p(y = l) && (3)
\end{aligned}$$

Naive Bayes vs. Logistische Regression

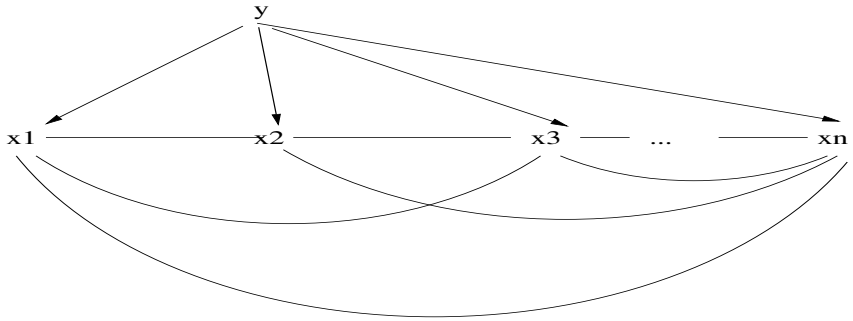
NB generativ:

$$\tilde{y} = \underset{l \in \{0,1\}}{\operatorname{argmax}} p(y = l|x) \quad (1)$$

$$= \underset{l \in \{0,1\}}{\operatorname{argmax}} \frac{p(x|y = l)p(y = l)}{p(x)} \quad (2)$$

$$= \underset{l \in \{0,1\}}{\operatorname{argmax}} p(x|y = l)p(y = l) \quad (3)$$

Full Bayesian Model mit $x = (x_1, x_2, \ldots x_n)$



Naive Bayes vs. Logistische Regression

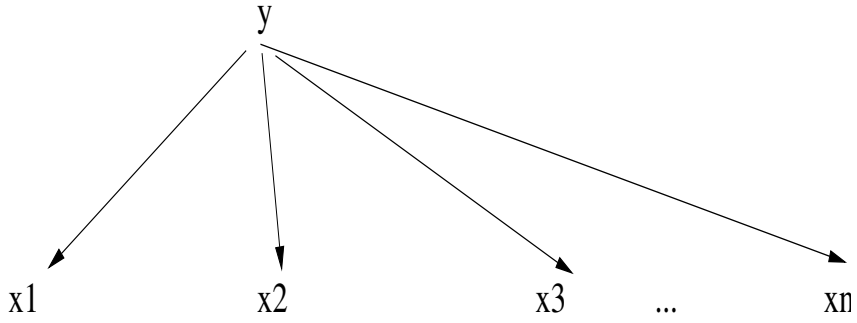
NB generativ:

$$\tilde{y} = \underset{l \in \{0,1\}}{\operatorname{argmax}} p(y = l|x) \quad (1)$$

$$= \underset{l \in \{0,1\}}{\operatorname{argmax}} \frac{p(x|y = l)p(y = l)}{p(x)} \quad (2)$$

$$= \underset{l \in \{0,1\}}{\operatorname{argmax}} p(x|y = l)p(y = l) \quad (3)$$

Naive Bayes mit $x = (x_1, x_2, \ldots x_n)$



Naive Bayes vs. Logistische Regression

NB generativ:

$$\begin{aligned}
\tilde{y} &= \operatorname*{argmax}_{l \in \{0,1\}} p(y = l|x) && (1)\\
&= \operatorname*{argmax}_{l \in \{0,1\}} \frac{p(x|y = l)p(y = l)}{p(x)} && (2)\\
&= \operatorname*{argmax}_{l \in \{0,1\}} p(x|y = l)p(y = l) && (3)
\end{aligned}$$

Logistische Regression: diskriminativ

$$\tilde{y} = \operatorname*{argmax}_{l \in \{0,1\}} P(y = l|x) \quad (4)$$

Übersicht

(binäre) Logistische Regression

- Überwachtes Klassifikationsverfahren, das Wahrscheinlichkeit ausgibt
- Wiederum jede Instanz mit Merkmalsvektor repräsentiert
- Diskriminativ: Schätze $p(y = l|x)$ direkt ($l \in \{0, 1\}$)
- Training: Lerne **Gewichte** für jedes Merkmal durch Minimierung einer **objective function = cross-entropy loss**
- Training: Optimierungsalgorithmus ableitungsbasiert = **gradient descent**
- Training: Geschieht normalerweise dann iterativ durch Anwendung der Klassifikation auf einzelne Trainingsbeispiele und danach Anpassung der Gewichte = **stochastic gradient descent**
- Testing: Berechne $p(y = 1|x)$ sowie $p(y = 0|x)$ mithilfe eines Skalarprodukts der gelernten Gewichte und Merkmale von x sowie der **sigmoid** Klassifikationsfunktion

(binäre) Logistische Regression

- Überwachtes Klassifikationsverfahren, das Wahrscheinlichkeit ausgibt
- Wiederum jede Instanz mit Merkmalsvektor repräsentiert
- Diskriminativ: Schätze $p(y = l|x)$ direkt ($l \in \{0, 1\}$)
- Training: Lerne **Gewichte** für jedes Merkmal durch Minimierung einer **objective function = cross-entropy loss**
- Training: Optimierungsalgorithmus ableitungsbasiert = **gradient descent**
- Training: Geschieht normalerweise dann iterativ durch Anwendung der Klassifikation auf einzelne Trainingsbeispiele und danach Anpassung der Gewichte = **stochastic gradient descent**
- Testing: Berechne $p(y = 1|x)$ sowie $p(y = 0|x)$ mithilfe eines Skalarprodukts der gelernten Gewichte und Merkmale von x sowie der **sigmoid** Klassifikationsfunktion

(binäre) Logistische Regression

- Überwachtes Klassifikationsverfahren, das Wahrscheinlichkeit ausgibt
- Wiederum jede Instanz mit Merkmalsvektor repräsentiert
- Diskriminativ: Schätze $p(y = l|x)$ direkt ($l \in \{0, 1\}$)
- Training: Lerne **Gewichte** für jedes Merkmal durch Minimierung einer **objective function = cross-entropy loss**
- Training: Optimierungsalgorithmus ableitungsbasiert = **gradient descent**
- Training: Geschieht normalerweise dann iterativ durch Anwendung der Klassifikation auf einzelne Trainingsbeispiele und danach Anpassung der Gewichte = **stochastic gradient descent**
- Testing: Berechne $p(y = 1|x)$ sowie $p(y = 0|x)$ mithilfe eines Skalarprodukts der gelernten Gewichte und Merkmale von x sowie der **sigmoid** Klassifikationsfunktion

Merkmale und Gewichte: Beispiele mit erfundenen Gewichten

Beispiel Sentimentklassifikation.

Positive Dokumente $y = 1$, sonst $y = 0$.

Mit erfundenen Gewichten:

Merkmal	mögliche Werte	Gewichte
enthält *awesome*	$\{0, 1\}$	+10
enthält *mediocre*	$\{0, 1\}$	-1
#worte	$\{1, 2, \ldots \infty\}$	-2

Merkmale und Gewichte: Beispiele mit erfundenen Gewichten

- Ein Gewicht per Merkmal $\longrightarrow$ Gewichtsvektor aus reellen Zahllen $\vec{w} = (w_1, w_2, \ldots w_n) \in \mathbb{R}^n$ oder $\vec{\theta} = (\theta_1, \ldots \theta_n)$
- Gewicht positiv: spricht für $y = 1$
- Gewicht negativ: spricht für $y = 0$
- Absolutbetrag des Gewichts: je höher desto stärker mit jeweiligem $y = 0/1$ assoziiert

Übersicht

Klassifikationsidee = lineare Gewichtskombination

Gegeben sei Testinstanz x mit Merkmalsvektor $\vec{x} = (x_1, \ldots x_n)$ sowie ein aus dem Training gelernter Gewichtsvektor $\vec{w} = (w_1, \ldots w_n)$ (und ein aus dem Training gelernter bias $b \in \mathbb{R}$)

Berechne Skalarprodukt aus Instanzvektor und Gewichtsvektor

$$z = (\sum_{i=1}^{n} w_i x_i) + b = \langle \vec{w}, \vec{x} \rangle + b = \vec{w} \cdot \vec{x} + b$$

Wenn hoch, $\tilde{y} = 1$, sonst $\tilde{y} = 0$

Man will aber eine Wahrscheinlichkeit: Sigmoidfunktion

z kann Werte zwischen $-\infty$ und $+\infty$ annehmen, keine Wahrscheinlichkeit

Benutze die **Sigmoidfunktion** (**logistische Funktion**) als Klassifikationsfunktion $\sigma(z)$:

$$p(y=1|\vec{x};\vec{w};b) := \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$$

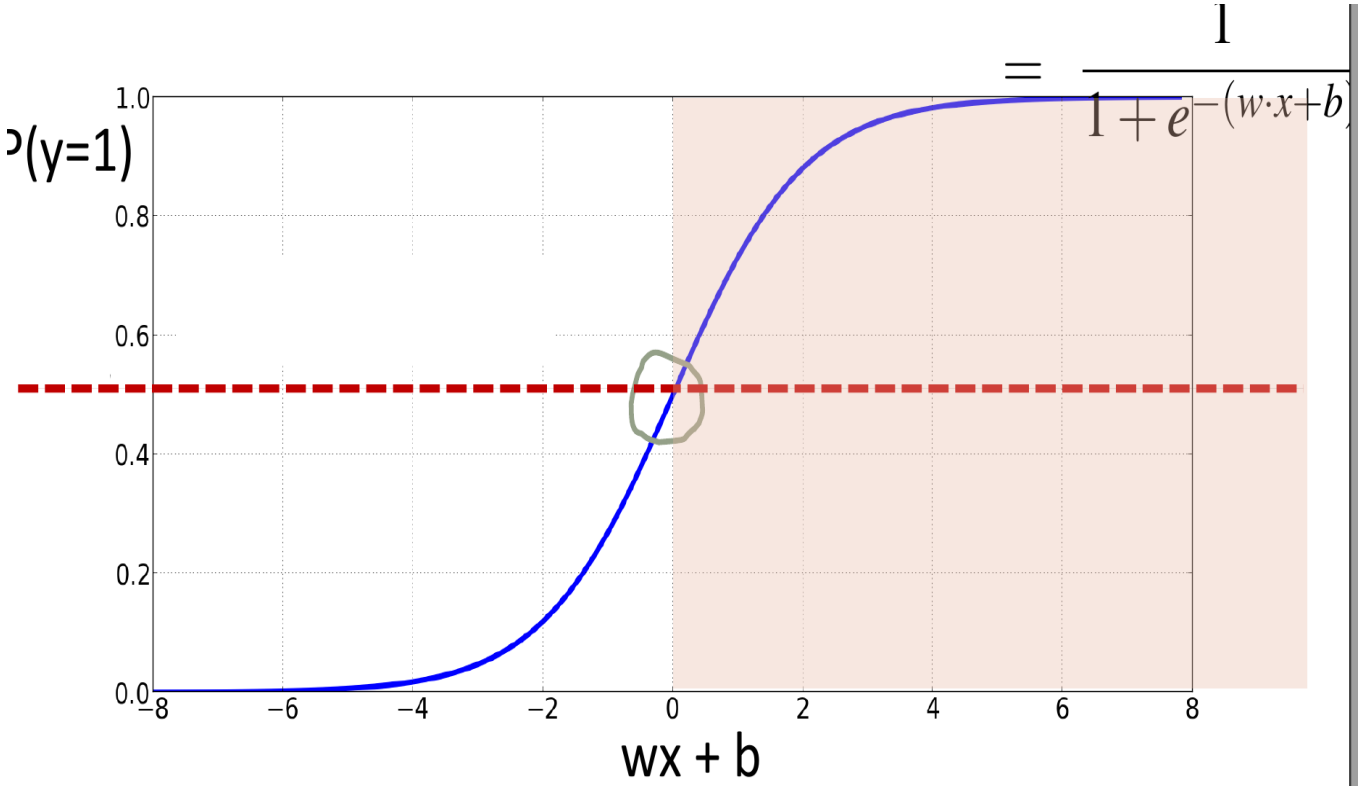
und dann

$$\begin{aligned} p(y=0|\vec{x};\vec{w};b) := 1-\sigma(z) &= 1-\frac{1}{1+e^{-z}} = 1-\frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}} \\ &= \frac{e^{-(\vec{w}\cdot\vec{x}+b)}}{1+e^{-(\vec{w}\cdot\vec{x}+b)}} \end{aligned}$$

Und entscheide
$\tilde{y}=1$, wenn $p(y=1|\vec{x};\vec{w};b) > 0.5$; $\tilde{y}=0$ sonst

0.5 = **decision boundary**

Bild der Sigmoidfunktion



- Bild aus Jurafsky und Martin, Ed 3, Figure 5.1
- Zwischen 0 und 1
- Punktsymmetrisch bei (0, 0.5). Es gilt: $1 - \sigma(z) = \sigma(-z)$

Beispiel A als Testinstanz in Sentimentklassifikation I

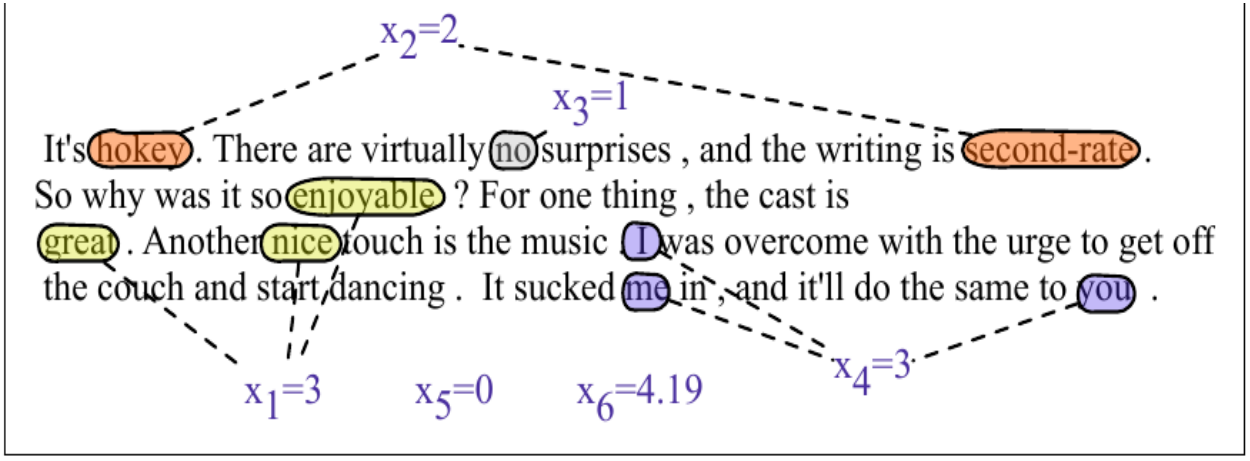


Bild aus Jurafsky und Martin, Ed 3, Figure 5.2

Merkmalsvektor (und vorgegebene Gewichte aus dem Training):

ID	Beschr.	Wert im Bsp.	Gewicht
x_1	#positive Wörter	3	2.5
x_2	#negative Wörter	2	-5.0
x_3	*no* im Dok?	1	-1.2
x_4	# 1st and 2nd pronouns	3	0.5
x_5	*!* im Dok?	0	2.0
x_6	# $\ln(wc)$	4.19	0.7

sowie $b = 0.1$

Beispiel A als Testinstanz in Sentimentklassifikation I

ID	Beschr.	Wert im Bsp.	Gewicht
x_1	#positive Wörter	3	2.5
x_2	#negative Wörter	2	-5.0
x_3	*no* im Dok?	1	-1.2
x_4	# 1st and 2nd pronouns	3	0.5
x_5	*!* im Dok?	0	2.0
x_6	# ln(wc)	4.19	0.7

$$
\begin{aligned}
p(y=1|\vec{x};\vec{w};b) &= \sigma(\vec{w}\cdot\vec{x}+b) \\
&= \sigma((2.5\cdot 3+(-5.0)\cdot 2+(-1.2)\cdot 1+0.5\cdot 3 \\
&+ 2.0\cdot 0+0.7\cdot 4.19)+0.1) \\
&= \sigma(0.833) \\
&= 0.70
\end{aligned}
$$

Und damit

$$p(y=0|\vec{x};\vec{w};b) = 1-p(y=1|\vec{x};\vec{w};b) = 0.3$$

Featureskalierung

Sei μ_i Mittelwert eines Features x_i im Training und σ_i seine Standardabweichung.

$$\mu_i = \frac{1}{m} \sum_{j=1}^{m} x_i^{(j)}$$

$$\sigma_i = \sqrt{\frac{1}{m} \sum_{j=1}^{m} (x_i^{(j)} - \mu_i)}$$

Standardisierung zu z-score: $x_i' = \frac{x_i - \mu_i}{\sigma_i}$

oder

Normalisierung zwischen 0 und 1: $x_i' = \frac{x_i - min(x_i)}{max(x_i) - min(x_i)}$

Viele Testinstanzen

Man nehme an, man hat k Testinstanzen mit jeweils n Merkmalen.

- Stelle die k Testinstanzen in einer Matrix $X \in \mathbb{R}^{k \times n}$ dar, wobei in jeder Matrixreihe eine Testinstanz steht. Stelle Gewichtsvektor $\vec{w} \in \mathbb{R}^n$ als Spaltenvektor da. Stelle b als (Spalten)Vektor im $\mathbb{R}^k$ dar.

$$\begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \cdots & \cdots & \cdots & \cdots \\ x_1^{(k)} & x_2^{(k)} & \cdots & x_n^{(k)} \end{pmatrix} \qquad \vec{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} b \\ b \\ \vdots \\ b \end{pmatrix}$$

- Dann gilt

$$\vec{y} = X \cdot \vec{w} + \vec{b}$$

ist ein (Spalten) Vektor $\vec{y} \in \mathbb{R}^k$, der genau die Vorhersagen per Testbeispiel enthält.

Zwischenfazit

- Logistische Regression ist überwachtes maschinelles Lernen mit reellwertigen Merkmalsrepräsentationen
- Merkmale werden von Menschen entworfen, gibt aber auch **feature templates** und automatisch extrahierte
- klassifiziert Instanzen durch ein Skalarprodukt von Gewichtsvektor und Merkmalsvektor, durch **Sigmoid**-Funktion zu einer Wahrscheinlichkeit umgewandelt
- Entscheidung fällt anhand einer **decision threshold**
- Gewichte werden im Training gelernt
- Gut interpretierbar
- Braucht keine Unabhängigkeitsannahmen

Zwischenfazit

- Logistische Regression ist überwachtes maschinelles Lernen mit reellwertigen Merkmalsrepräsentationen
- Merkmale werden von Menschen entworfen, gibt aber auch **feature templates** und automatisch extrahierte
- klassifiziert Instanzen durch ein Skalarprodukt von Gewichtsvektor und Merkmalsvektor, durch **Sigmoid**-Funktion zu einer Wahrscheinlichkeit umgewandelt
- Entscheidung fällt anhand einer **decision threshold**
- Gewichte werden im Training gelernt
- Gut interpretierbar
- Braucht keine Unabhängigkeitsannahmen

Zwischenfazit

- Logistische Regression ist überwachtes maschinelles Lernen mit reellwertigen Merkmalsrepräsentationen
- Merkmale werden von Menschen entworfen, gibt aber auch **feature templates** und automatisch extrahierte
- klassifiziert Instanzen durch ein Skalarprodukt von Gewichtsvektor und Merkmalsvektor, durch **Sigmoid**-Funktion zu einer Wahrscheinlichkeit umgewandelt
- Entscheidung fällt anhand einer **decision threshold**
- Gewichte werden im Training gelernt
- Gut interpretierbar
- Braucht keine Unabhängigkeitsannahmen

Übersicht

Training Grundidee

- Training = Bestimmung der Gewichte $\vec{w}$ und des bias b auf einem Datensatz $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots (x^{(m)}, y^{(m)})$, auf dem wir die Lösungen kennen
- Die Gewichte sollen so bestimmt sein, dass jedes $\tilde{y}^{(j)}$ möglichst nahe an $y^{(j)}$ dran ist für alle Trainingsinstanzen $x^{(j)}$
- Was heisst nahe? $\rightarrow$ Kostenfunktion (loss function, cost function) $L(\tilde{y}, y) \rightarrow$ **Cross-entropy loss**
- Wie kommt man nahe ran für alle Trainingsdaten? Iterative Bestimmung der Gewichte mittels **stochastic gradient descent**

Training Grundidee

- Training = Bestimmung der Gewichte $\vec{w}$ und des bias b auf einem Datensatz $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots (x^{(m)}, y^{(m)})$, auf dem wir die Lösungen kennen
- Die Gewichte sollen so bestimmt sein, dass jedes $\tilde{y}^{(j)}$ möglichst nahe an $y^{(j)}$ dran ist für alle Trainingsinstanzen $x^{(j)}$
- Was heisst nahe? $\rightarrow$ Kostenfunktion (loss function, cost function) $L(\tilde{y}, y) \rightarrow$ **Cross-entropy loss**
- Wie kommt man nahe ran für alle Trainingsdaten? Iterative Bestimmung der Gewichte mittels **stochastic gradient descent**

Training Grundidee

- Training = Bestimmung der Gewichte $\vec{w}$ und des bias b auf einem Datensatz $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots (x^{(m)}, y^{(m)})$, auf dem wir die Lösungen kennen
- Die Gewichte sollen so bestimmt sein, dass jedes $\tilde{y}^{(j)}$ möglichst nahe an $y^{(j)}$ dran ist für alle Trainingsinstanzen $x^{(j)}$
- Was heisst nahe? $\rightarrow$ Kostenfunktion (loss function, cost function) $L(\tilde{y}, y) \rightarrow$ **Cross-entropy loss**
- Wie kommt man nahe ran für alle Trainingsdaten? Iterative Bestimmung der Gewichte mittels **stochastic gradient descent**

Kostenfunktion

Kostenfunktion $L(\tilde{y}, y)$

Funktion, die für eine Instanz bestimmt, wie weit $\tilde{y}$ und y auseinander sind

Optional:
Warum nicht einfach

$$|\tilde{y} - y| = |\sigma(wx + b) - y|$$

Kostenfunktion Intuition: Sei so nahe am wahren y dran wie möglich

- Es gilt: $y \in \{0, 1\}$.
- Es gilt auch $\tilde{y} \in \{0, 1\}$
- Zudem haben wir aber auch für $\tilde{y}$ eine Wahrscheinlichkeit berechnet, mittels $\sigma(wx + b)$; oft wird auch diese mit $\tilde{y} \in [0, 1]$ bezeichnet
- Für y=1, sollte $\tilde{y}$ so hoch wie möglich sein
- Für y=0, sollte $\tilde{y}$ so niedrig wie möglich sein bzw. $1 - \tilde{y}$ hoch wie möglich

Cross Entropy Loss

Wir wollen die Kreuzentropie *minimieren*:

$$\begin{aligned} L_{CE}(\tilde{y}, y) &= -[y \log \tilde{y} + (1-y) \log(1-\tilde{y})] \\ &= -[y \log \sigma(wx+b) + (1-y) \log(1-\sigma(wx+b))] \end{aligned}$$

Fragen

$$L_{CE}(\tilde{y}, y) = -[y \log \sigma(wx + b) + (1 - y) \log(1 - \sigma(wx + b))]$$

1. Was passiert bei $y = 1$ bzw $y = 0$?
2. Gibt es negative Kosten?
3. Was ist der kleinste erreichbare Cost? Wann wird dieser jeweils erreicht?
4. Was ist die Obergrenze?
5. Wie stark wird jeweils bestraft?

Beispiel A

Wir hatten berechnet $\tilde{y} = 1$ mit $\sigma(wx + b) = 0.7$.
Nehmen wir an, richtig wäre $y = 1$:

$$\begin{aligned} L_{CE}(\tilde{y}, y) &= -[y \log \sigma(wx + b) + (1 - y) \log(1 - \sigma(wx + b))] \\ &= -[1 \cdot \log 0.7 + (1 - 1) \cdot \log(1 - 0.7] \\ &= -\log(0.7) \\ &= 0.37 \end{aligned}$$

Nehmen wir an, richtig wäre $y = 0$:

$$\begin{aligned} L_{CE}(\tilde{y}, y) &= -[y \log \sigma(wx + b) + (1 - y) \log(1 - \sigma(wx + b)] \\ &= -[0 \cdot \log 0.7 + (1 - 0) \cdot \log(1 - 0.7] \\ &= -\log(0.3) \\ &= 1.17 \end{aligned}$$

Beispiel A

Wir hatten berechnet $\tilde{y} = 1$ mit $\sigma(wx + b) = 0.7$.
Nehmen wir an, richtig wäre $y = 1$:

$$\begin{aligned} L_{CE}(\tilde{y}, y) &= -[y \log \sigma(wx + b) + (1 - y) \log(1 - \sigma(wx + b))] \\ &= -[1 \cdot \log 0.7 + (1 - 1) \cdot \log(1 - 0.7] \\ &= -\log(0.7) \\ &= 0.37 \end{aligned}$$

Nehmen wir an, richtig wäre $y = 0$:

$$\begin{aligned} L_{CE}(\tilde{y}, y) &= -[y \log \sigma(wx + b) + (1 - y) \log(1 - \sigma(wx + b)] \\ &= -[0 \cdot \log 0.7 + (1 - 0) \cdot \log(1 - 0.7] \\ &= -\log(0.3) \\ &= 1.17 \end{aligned}$$

Optional: Cross-entropy (Kreuzentropie) hat was mit Entropie zu tun?

Entropie: Anzahl der Bits, die wir brauchen um Zufallsvariable Y zu enkodieren

$$H(Y) = H(p) = -\sum_y p(y) \log p(y) = \sum_y p(y) \log \frac{1}{p(y)}$$

Kreuzentropie: Anzahl der Bits um Y zu enkodieren, wenn wir ein suboptimales $q(y)$ anstatt $p(y)$ benutzen

$$H_K(p, q) := -\sum_y p(y) \log q(y) = \sum_y p(y) \log \frac{1}{q(y)}$$

Kreuzentropie ist immer größer als Entropie also $H_K(p, q) \geq H(p)$ (Ausnahme: $p = q$)

Beweis im Appendix

Optional: Cross entropy (Kreuzentropie) hat was mit Entropie zu tun?

Kullback-Leibler (KL) Divergenz: Unterschied zwischen

cross entropy und entropy

$$KL(p||q) = \sum_y p(y) log \frac{1}{q(y)} - \sum_y p(y) log \frac{1}{p(y)} = \sum_y p(y) log \frac{p(y)}{q(y)}$$

$\Rightarrow$ Anzahl der *extra* bits wenn man $q(y)$ anstatt $p(y)$ verwendet

Optional: Cross entropy (Kreuzentropie) hat was mit Entropie zu tun?

Kullback-Leibler (KL) Divergenz: Unterschied zwischen

cross entropy und entropy

$$KL(p||q) = \sum_y p(y) log \frac{1}{q(y)} - \sum_y p(y) log \frac{1}{p(y)} = \sum_y p(y) log \frac{p(y)}{q(y)}$$

$\Rightarrow$ Anzahl der *extra* bits wenn man $q(y)$ anstatt $p(y)$ verwendet

Optional: Cross-Entropy

$$H_K(p,q) := -\sum_y p(y) \log q(y)$$

In unserem Fall hatten wir nur zwei outcomes $y = 1$ oder $y = 0$, und unsere $q(y)$ Annäherungen sind unsere $\tilde{y} = \sigma(wx + b)$
Damit

$$H_K(p,q) = -[y \log \tilde{y} - (1-y) \log(1-\tilde{y})]$$

Gewichtsberechnung: Unser Ziel

Ziel

- Minimiere Kostenfunktion über das ganze Trainingsset der Grösse m
- Bzw: Durchschnittlicher loss pro Trainingsinstanz soll minimal sein

$$\widetilde{(w, b)} = \underset{w,b}{\operatorname{argmin}} \frac{1}{m} \sum_{j=1}^{m} L_{CE}(\tilde{y}^{(j)}, y^{(j)})$$

Wie minimiere ich Funktionen?

Lokale Minima

Critical points in 2D (one input value):

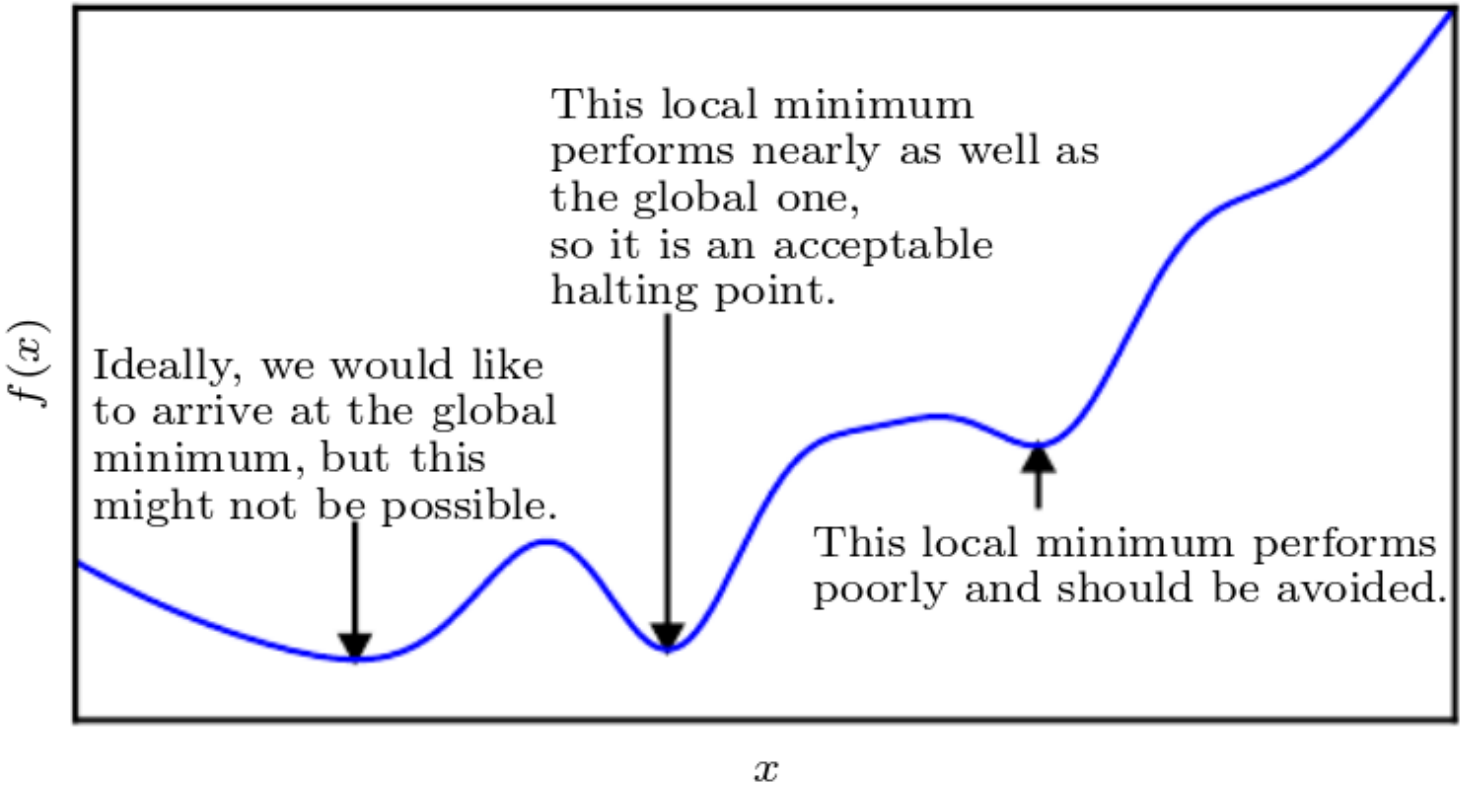


Bild von Goodfellow, Bengio und Courville unter
`https://www.deeplearningbook.org/slides/04_numerical.pdf` , Figure 4.3

Konvexe Loss Function (optional)

- Unsere loss funktion ist konvex in $(\vec{w}, b) \rightarrow$ nur ein Minimum (Beweis im Appendix)
- Gradient descent wird dieses Minimum immer finden, egal wo wir starten

GD: Formalisierung mithilfe einzelnen Skalars w

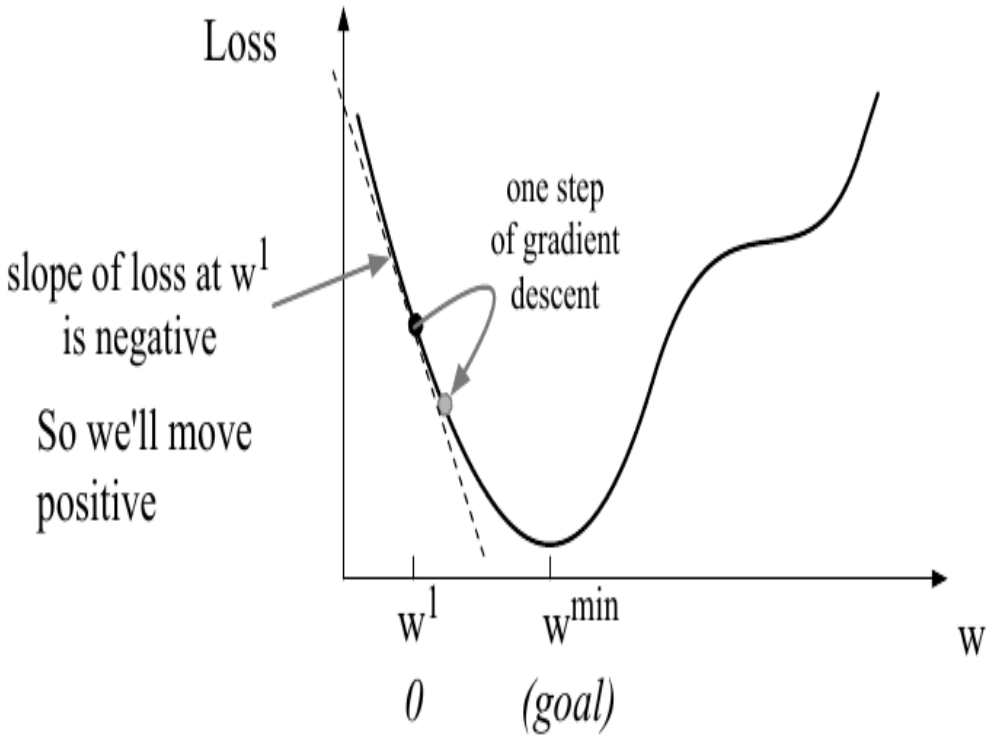


Bild aus Jurafsky und Martin, Ed 3, Figure 5.4

1. Ableitung negativ: bewege Dich nach rechts, sonst nach links
2. Wieweit sollten wir uns bewegen: Gewichtet mit **learning rate** η

$$w^{(t+1)} = w^{(t)} - \eta \cdot \frac{d}{dw} L_{CE}$$

3. Je höher die learning rate, desto schneller bewegen wir uns

Verallgemeinerung auf n Dimensionen

Unser $(\vec{w}, b)$ ist ein Vektor mit mehr als einer Dimension. Hat $n+1$ Dimensionen.

Für zwei Dimensionen:

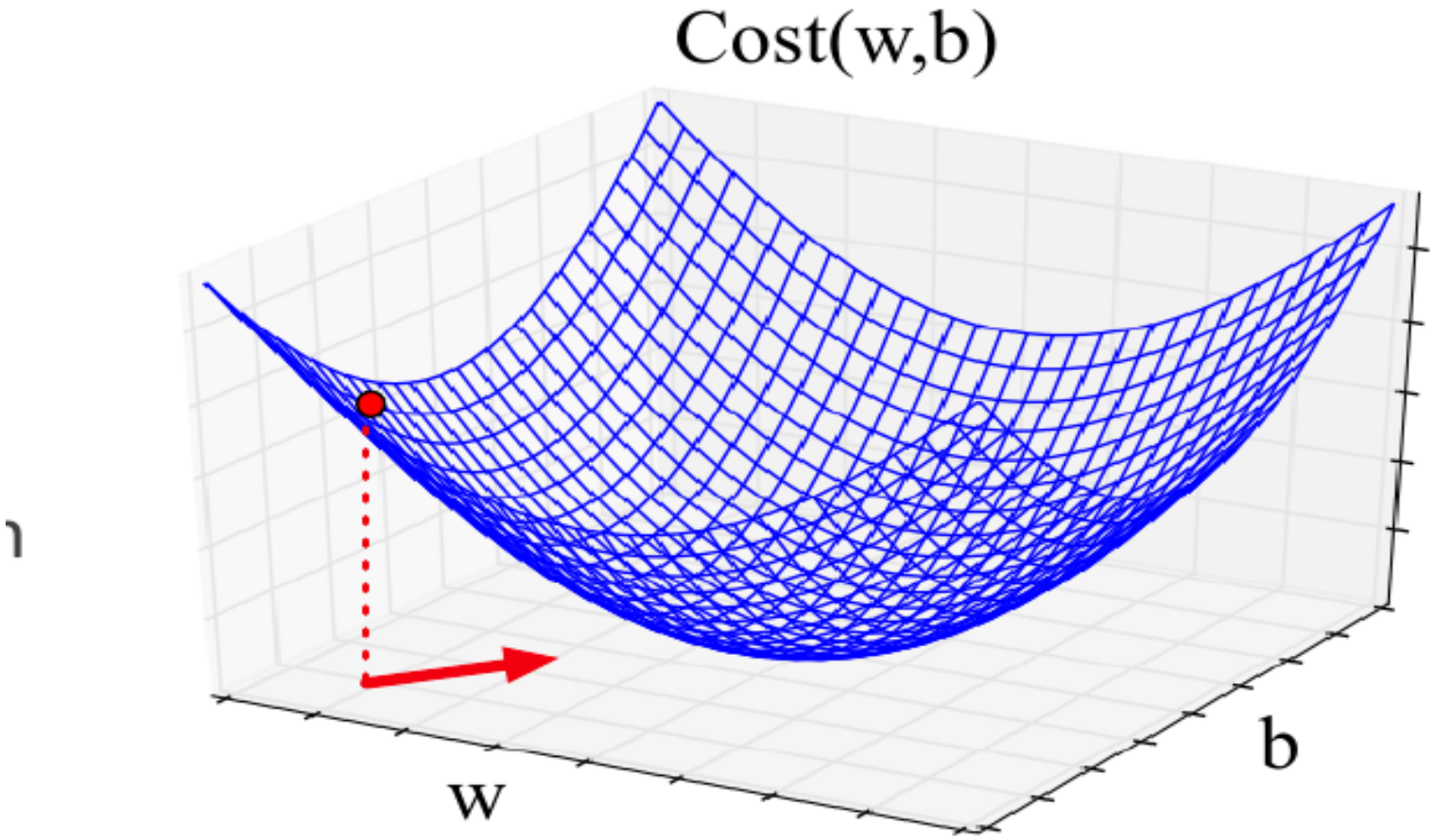


Bild aus Jurafsky und Martin, Ed 3. Figure 5.5

Gradient

.

Wieviel würde eine Änderung in w_i oder b ($i \in \{1, \ldots, n\}$) die gesamte Loss Function beeinflussen?

Gradient ist ein $n+1$-dimensionaler Vektor, der genau dies ausdrückt mithilfe partieller Ableitungen nach allen Parametern

$$\nabla_{(\vec{w},b)}(L_{CE}(\tilde{y},y)) = \begin{pmatrix} \frac{\partial L_{CE}(\tilde{y},y)}{\partial w_1} \\ \frac{\partial L_{CE}(\tilde{y},y)}{\partial w_2} \\ \ldots \\ \frac{\partial L_{CE}(\tilde{y},y)}{\partial w_n} \\ \frac{\partial L_{CE}(\tilde{y},y)}{\partial b} \end{pmatrix} \tag{5}$$

Und damit mit Schreibweise $\theta = (w, b)$:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_{(\theta}(L_{CE}(\tilde{y},y))$$

Wie sehen die Ableitungen aus?

Wir erinnern uns:

$$L_{CE}(\tilde{y}, y) = -[y \log \sigma(wx + b) + (1 - y) \log(1 - \sigma(wx + b))]$$

Die partiellen Ableitungen sind (siehe Kapitel 5.8 in Jurafsky und Martin für Herleitung bzw Appendix)

$$\frac{\partial L_{CE}(\tilde{y}, y)}{\partial w_i} = [\sigma(wx + b) - y]x_i = (\tilde{y} - y)x_i$$

sowie

$$\frac{\partial L_{CE}(\tilde{y}, y)}{\partial b} = \sigma(wx + b) - y = \tilde{y} - y$$

Intuition

Wir berücksichtigen, wieweit wir vom wirklichen Wert der Antwortsvariable weg sind $(\tilde{y} - y)$ sowie den Wert von x_i

Problem mit Gradient descent

1. Gleichzeitige Optimierung über das ganze Trainingsset meist zu teuer
2. Lösung Stochastic Gradient Descent: Nehme nur eine einzige oder eine kleine Menge an Trainingsinstanzen, berechne $\tilde{y}$ und die entsprechenden Fehler und update dann die Gewichte

Der Algorithmus für Stochastic gradient descent: Aus Jurafsky und Martin, Ed 3

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

where: L is the loss function
f is a function parameterized by θ
x is the set of training inputs $x^{(1)}$, $x^{(2)}$,..., $x^{(m)}$
y is the set of training outputs (labels) $y^{(1)}$, $y^{(2)}$,..., $y^{(m)}$

$\theta \leftarrow 0$

repeat til done # see caption

For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

1. Optional (for reporting):
   Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$
   Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$
2. $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$
3. $\theta \leftarrow \theta - \eta\, g$

How are we doing on this tuple?
What is our estimated output $\hat{y}$?
How far off is $\hat{y}^{(i)}$ from the true output $y^{(i)}$?
How should we move θ to maximize loss?
Go the other way instead

return θ

Bei uns: $f = \sigma(wx + b)$

Hyperparameter und Konvergenz

- **Learning rate** η st ein Hyperparameter
- Normalerweise vom Menschen gesetzt
- Zu klein: Lernen geht sehr langsam
- Oft am Anfang höher und wird dann immer kleiner
- Wann höre ich auf: keine Fortschritte mehr oder Gradient sehr klein

Beispiel Training Sentimentklassifikation: Gradient I

$y = 1$ sei positiv. Trainingsset:

id	x_1 (#pos words)	x_2 (#neg words)	y
1	**3**	**1**	**0**
2	5	1	1
3	2	5	0

Starte mit $w_1^{(0)} = w_2^{(0)} = b^{(0)} = 0$ sowie $\eta = 0.1$

Gradient (Instanz 1):

$$\nabla_{(\vec{w},b)}(L_{CE}(\tilde{y},y)) = \begin{pmatrix} \frac{\partial L_{CE}(\tilde{y},y)}{\partial w_1} \\ \frac{\partial L_{CE}(\tilde{y},y)}{\partial w_2} \\ \frac{\partial L_{CE}(\tilde{y},y)}{\partial b} \end{pmatrix} = \begin{pmatrix} (\sigma(wx+b)-y)x_1 \\ (\sigma(wx+b)-y)x_2 \\ \sigma(wx+b)-y \end{pmatrix}$$

$$= \begin{pmatrix} (\sigma(0)-0)\cdot 3 \\ (\sigma(0)-0)\cdot 1 \\ \sigma(0)-0 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 0.5 \\ 0.5 \end{pmatrix}$$

Beispiel Gewichtsupdate I

id	x_1 (#pos words)	x_2 (#neg words)	y
1	**3**	**1**	**0**
2	5	1	1
3	2	5	0

$w_1^{(0)} = w_2^{(0)} = b^{(0)} = 0$ sowie $\eta = 0.1$

$$\nabla_{(\vec{w},b)}(L_{CE}(\tilde{y}, y)) = \begin{pmatrix} 1.5 \\ 0.5 \\ 0.5 \end{pmatrix}$$

Neue Gewichte:

$$\begin{pmatrix} w_1^{(1)} \\ w_2^{(1)} \\ b^{(1)} \end{pmatrix} = \begin{pmatrix} w_1^{(0)} - \eta \cdot 1.5 \\ w_2^{(0)} - \eta \cdot 0.5 \\ b^{(0)} - \eta \cdot 0.5 \end{pmatrix} = \begin{pmatrix} -0.15 \\ -0.05 \\ -0.05 \end{pmatrix}$$

Beispiel Gradient II

id	x_1 (#pos words)	x_2 (#neg words)	y
1	3	1	0
2	**5**	**1**	**1**
3	2	5	0

$w_1^{(1)} = -0.15; w_2^{(1)} = -0.05; b^{(1)} = -0.05$ sowie $\eta = 0.1$

Gradient (Instanz 2):

$$\nabla_{(\vec{w},b)}(L_{CE}(\tilde{y},y)) = \begin{pmatrix} (\sigma(wx+b)-y)x_1 \\ (\sigma(wx+b)-y)x_2 \\ \sigma(wx+b)-y \end{pmatrix}$$

$$= \begin{pmatrix} (\sigma(-0.85)-1)\cdot 5 \\ (\sigma(-0.85)-1)\cdot 1 \\ \sigma(-0.85)-1 \end{pmatrix} = \begin{pmatrix} (0.3-1)\cdot 5 \\ (0.3-1)\cdot 1 \\ 0.3-1 \end{pmatrix} = \begin{pmatrix} -3.5 \\ -0.7 \\ -0.7 \end{pmatrix}$$

Beispiel Gewichtsupdate II

id	x_1 (#pos words)	x_2 (#neg words)	y
1	3	1	0
2	**5**	**1**	**1**
3	2	5	0

$w_1^{(1)} = -0.15; w_2^{(1)} = -0.05; b^{(1)} = -0.05$ sowie $\eta = 0.1$

$$\nabla_{(\vec{w},b)}(L_{CE}(\tilde{y}, y)) = \begin{pmatrix} -3.5 \\ -0.7 \\ -0.7 \end{pmatrix}$$

Neue Gewichte:

$$\begin{pmatrix} w_1^{(2)} \\ w_2^{(2)} \\ b^{(2)} \end{pmatrix} = \begin{pmatrix} w_1^{(1)} - \eta \cdot (-3.5) \\ w_2^{(1)} - \eta \cdot (-0.7) \\ b^{(1)} - \eta \cdot (-0.7) \end{pmatrix} = \begin{pmatrix} 0.20 \\ 0.02 \\ 0.02 \end{pmatrix}$$

Beispiel Gradient III

id	x_1 (#pos words)	x_2 (#neg words)	y
1	3	1	0
2	5	1	1
3	**2**	**5**	**0**

$w_1^{(2)} = 0.20; w_2^{(2)} = 0.02; b^{(1)} = 0.02$ sowie $\eta = 0.1$

Gradient (Instanz 3):

$$\nabla_{(\vec{w},b)}(L_{CE}(\tilde{y},y)) = \begin{pmatrix} (\sigma(wx+b)-y)x_1 \\ (\sigma(wx+b)-y)x_2 \\ \sigma(wx+b)-y \end{pmatrix}$$

$$= \begin{pmatrix} (\sigma(0.52)-0)\cdot 2 \\ (\sigma(0.52)-0)\cdot 5 \\ \sigma(0.52)-0 \end{pmatrix} = \begin{pmatrix} (0.63-0)\cdot 2 \\ (0.63-0)\cdot 5 \\ 0.63-0 \end{pmatrix} = \begin{pmatrix} 1.26 \\ 3.15 \\ 0.63 \end{pmatrix}$$

Beispiel Gewichtsupdate III

id	x_1 (#pos words)	x_2 (#neg words)	y
1	3	1	0
2	5	1	1
3	**2**	**5**	**0**

$w_1^{(2)} = 0.20; w_2^{(2)} = 0.02; b^{(2)} = 0.02$ sowie $\eta = 0.1$

$$\nabla_{(\vec{w},b)}(L_{CE}(\tilde{y}, y)) = \begin{pmatrix} 1.26 \\ 3.15 \\ 0.63 \end{pmatrix}$$

Neue Gewichte:

$$\begin{pmatrix} w_1^{(3)} \\ w_2^{(3)} \\ b^{(3)} \end{pmatrix} = \begin{pmatrix} w_1^{(2)} - \eta \cdot 1.26 \\ w_2^{(2)} - \eta \cdot 3.15 \\ b^{(2)} - \eta \cdot 0.63 \end{pmatrix} = \begin{pmatrix} 0.074 \\ -0.295 \\ -0.043 \end{pmatrix}$$

Eigenschaften und Zwischenfazit: Training

- "Forward Pass": Berechne Output $\tilde{y} = \sigma(wx + b)$ für ein Trainingsbeispiel
- Vergleiche $\tilde{y}$ mit dem eigentlich richtigem y
- Berechne Fehler $L_{CE}(\tilde{y}, y)$ sowie $\nabla_{(\vec{w},b)}(L_{CE}(\tilde{y}, y))$
- Passe Gewichte und Bias an mittels

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_{(\theta}(L_{CE}(\tilde{y}, y))$$

- In unserem Fall wird dies einfach zu

$$w_i^{(t+1)} = w_i^{(t)} - \eta[\sigma(wx + b) - y)x_i] = w_i^{(t)} - \eta[(\tilde{y} - y)x_i]$$

sowie

$$b^{(t+1)} = b^{(t)} - \eta[\sigma(wx + b) - y] = b^{(y)} - \eta[\tilde{y} - y]$$

Eigenschaften: Training

- Learning Rate Hyperparameter
- Computation time per update wird mit mehr Trainingsbeispielen nicht größer
- Initialisierung der Gewichtsvektoren?
- Stark korrelierte Features sind (im Gegensatz zu NB) kein Problem: jedes Feature bekommt Teil des Gewichtes

Übersicht

Ausblick: Mini-batch training

- Per-Beispiel Training: geht recht ruckartig hin und her + ineffizient
- Meist: Trainiere nicht per Beispiel, sondern per Gruppe von m Beispielen wie 512 Beispiele
- Heisst **Mini-batch** Training
- Anpassung der loss function sowie des Gradienten (siehe Kapitel 5.4.4 in Jurafsky and Marting, 3rd edition)

Optional: Mini-batch training: Anpassung der Loss Function und Gradienten

Ausweitung der Loss Function von einem Beispiel auf m Trainingsbeispiele, die wir als voneinander unabhängig sehen:

$$Cost(\tilde{y}, y) = \frac{1}{m} \sum_{i=1}^{m} L_{CE}(\widetilde{y^{(i)}}, y^{(i)})$$

Wir gehen also über das Mini-Batch, berechnen den Loss pro Beispiel und nehmen dann den average loss in diesem Minibatch

Mini-batch Gradient ist Durchschnitt der einzelnen Gradienten:

$$\frac{\partial Cost(\tilde{y}, y)}{\partial(w_j)} = \frac{1}{m} \sum_{i=1}^{m} [\sigma(w \cdot x^{(i)} + b) - y^{(i)}] x_j^{(i)}$$

Optional: Mini-batch training: Anpassung der Loss Function und Gradienten

Ausweitung der Loss Function von einem Beispiel auf m Trainingsbeispiele, die wir als voneinander unabhängig sehen:

$$Cost(\tilde{y}, y) = \frac{1}{m} \sum_{i=1}^{m} L_{CE}(\widetilde{y^{(i)}}, y^{(i)})$$

Wir gehen also über das Mini-Batch, berechnen den Loss pro Beispiel und nehmen dann den average loss in diesem Minibatch

Mini-batch Gradient ist Durchschnitt der einzelnen Gradienten:

$$\frac{\partial Cost(\tilde{y}, y)}{\partial(w_j)} = \frac{1}{m} \sum_{i=1}^{m} [\sigma(w \cdot x^{(i)} + b) - y^{(i)}] x_j^{(i)}$$

Übersicht

Regularisierung

Overfitting

Ein Modell passt sich zu genau an die Trainingsdaten an, und modelliert auch Rauschen. Generalisiert daher nicht gut auf neue Daten.

In Logistischer Regression, wie bisher gesehen:

- Nehmen wir an Merkmale Wort *word* mit Werten $\{0, 1\}$
- Ein Wort kommt zufällig genau einmal mit Klasse $y = 1$ vor und sonst nie
- Dieses Wort wird sehr hohes Gewicht erhalten

Regularisierung

- Idee: Vermeide Gewichtsvektoren θ mit sehr hohen Gewichten
- Minimiere zusätzlich noch ein positives $R(\theta)$, das die Größe des Gewichtsvektors ausdrückt
- Kombiniere diese Minimisierung gewichtet mit der Minimisierung der Loss function in einer regularisierten Zielfunktion ($\alpha > 0$)

Anstatt

$$\widetilde{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(\tilde{y}^{(i)}, y^{(i)})$$

benutzen wir also

$$\widetilde{\theta} = \underset{\theta}{\operatorname{argmin}} [\frac{1}{m} \sum_{i=1}^{m} L_{CE}(\tilde{y}^{(i)}, y^{(i)}) + \alpha R(\theta)]$$

bzw

$$\widetilde{\theta} = \underset{\theta}{\operatorname{argmax}} [\sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}) - \alpha R(\theta)]$$

L2-Regularisierung

L2-Regularisierung: ridge regression

$$\widetilde{\theta} = \underset{\theta}{\operatorname{argmax}}[\sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}) - \alpha \sum_{j=1}^{n} \theta_j^2]$$

- schöne Ableitung $2 \cdot \theta$, stetig
- präferiert Gewichtsvektoren mit vielen kleinen Gewichten

L2-Regularisierung

L2-Regularisierung: ridge regression

$$\widetilde{\theta} = \underset{\theta}{\operatorname{argmax}}[\sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}) - \alpha \sum_{j=1}^{n} \theta_j^2]$$

- schöne Ableitung $2 \cdot \theta$, stetig
- präferiert Gewichtsvektoren mit vielen kleinen Gewichten

Optional: L1-Regularisierung

L1-Regularisierung: lasso regression

$$\widetilde{\theta} = \underset{\theta}{\operatorname{argmax}}[\sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}) - \alpha \sum_{j=1}^{n} |\theta_j|]$$

- Ableitung von $|\theta|$ leider am Nullpunkt nicht stetig
- Nicht leicht zu optimieren
- Präferiert sparse Vektoren mit vielleicht einigen größeren Gewichten, aber vielen Nullgewichten

Optional: L1-Regularisierung

L1-Regularisierung: lasso regression

$$\widetilde{\theta} = \underset{\theta}{\operatorname{argmax}}[\sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}) - \alpha \sum_{j=1}^{n} |\theta_j|]$$

- Ableitung von $|\theta|$ leider am Nullpunkt nicht stetig
- Nicht leicht zu optimieren
- Präferiert sparse Vektoren mit vielleicht einigen größeren Gewichten, aber vielen Nullgewichten

Übersicht

Ausweitung auf mehr als 2 Klassen

Genannt auch **Multinomiale logistische Regression** oder **Softmax Regression** oder **Maximum-entropy classifier**

Nehmen wir an y kann nun K Werte annehmen.

Wir benötigen jetzt verschiedene Gewichtsvektoren und biases für jede der K Klassen

Beispiel: 3-wertige Sentiment classification (positiv, negativ, neutral)

Merkmal	Def	$w_{5,pos}$	$w_{5,neg}$	$w_{5,neut}$
$f_5(x)$	Ausrufezeichen kommt vor	3.5	3.1	-5.2

Multinomiale Regression Testing und Klassifikationsfunktion

Berechne wieder Skalarprodukt aus Instanzvektor und Gewichtsvektor pro Klasse k

$$z_k = (\sum_{i=1}^{n} w_{(i,k)} x_i) + b_k = \vec{w}_k \cdot \vec{x} + b_k$$

Dies gibt einen Wert zwischen $-\infty$ und ∞ pro Klasse k, und somit einen Vektor $z = (z_1, \ldots z_k) \in \mathbb{R}^n$
Klassifikationsfunktion, die diesen in einen Wahrscheinlichkeitsvektor umwandelt, ist nun **die Softmax Funktion**:

$$softmax(z_k) = \frac{e^{z_k}}{\sum_{r=1}^{K} e^{z_r}} = \frac{e^{w_k x + b_k}}{\sum_{r=1}^{K} e^{w_k x + b_k}} \quad fuer\ alle\ 1 \leq k \leq K$$

Multinomiale Regression: Softmax

$$softmax(z_k) = \frac{e^{z_k}}{\sum_{r=1}^{K} e^{z_r}} = \frac{e^{w_k x + b_k}}{\sum_{r=1}^{K} e^{w_k x + b_k}} \quad fuer\ alle\ 1 \leq k \leq K$$

- Wird positiv
- Normalisiert
- **max**: Die Wahrscheinlichkeit des größten z_k wird amplifiziert
- **soft**: kleinere z_k bekommen immer noch eine Wahrscheinlichkeit

Der Softmax ergibt also wieder einen Vektor, einen Wahrscheinlichkeitsvektor
Beispiel:

$$z = (0.6, 1.1, -1.5, 1.2, 3.2, -1.1)$$

ergibt mit softmax

$$(0.055, 0.090, 0.006, 0.099, 0.74, 0.010)$$

Multinomiale Regression: Negative Loglikelihood Loss

Loss Function generalisiert binäre Kreuzentropie:
Erinnerung:

$$L_{CE}(\tilde{y}, y) = -\log p(y|x) = -[y \log \tilde{y} + (1-y) \log(1-\tilde{y})]$$

wird zu

$$L_{CE}(\tilde{y}, y) = -\sum_{k=1}^{K} y_k \log \tilde{y}_k = -\sum_{k=1}^{K} y_k \log \tilde{p}(y=k|x)$$

Da alle y_k ausserhalb der korrekten Klasse 0 sind, wird dies zu

$$L_{CE}(\tilde{y}, y) = -\log \tilde{y}_k = -\log \frac{e^{w_k x + b_k}}{\sum_{l=1}^{K} e^{w_l x + b_l}}$$

wobei k korrekte Klasse

Multinomiale Regression: Negative Loglikelihood Loss

Loss Function generalisiert binäre Kreuzentropie:
Erinnerung:

$$L_{CE}(\tilde{y}, y) = -\log p(y|x) = -[y \log \tilde{y} + (1-y) \log(1-\tilde{y})]$$

wird zu

$$L_{CE}(\tilde{y}, y) = -\sum_{k=1}^{K} y_k \log \tilde{y}_k = -\sum_{k=1}^{K} y_k \log \tilde{p}(y=k|x)$$

Da alle y_k ausserhalb der korrekten Klasse 0 sind, wird dies zu

$$L_{CE}(\tilde{y}, y) = -\log \tilde{y}_k = -\log \frac{e^{w_k x + b_k}}{\sum_{l=1}^{K} e^{w_l x + b_l}}$$

wobei k korrekte Klasse

Multinomiale Regression: Training und Gradient

Es stellt sich heraus

$$\frac{\partial L_{CE}}{\partial w_{k,i}} = (\tilde{p}(y = k|x) - \mathbb{I}\{y = k\})x_i$$

mit k korrekter Klasse, i Index des Gewichtsvektors. $\mathbb{I}$ Indikatorfunktion/charakteristische Funktion.

Zusammenfassung

- Logistische Regression ist ein **probabilistischer überwachter Klassifizierer**, dessen Testentscheidungen auf Skalarprodukt von Merkmalsvektoren mit Gewichtsvektor sowie Sigmoidfunktion beruhen
- Training der Gewichte: Minimiere **cross-entropy loss** durch **stochastic gradient descent**
- Sollte regularisiert werden
- Kann auf mehr als 2 Klassen ausgeweitet werden
- Jurafsky und Martin, 3rd edition, Kapitel 5
- Optional für ECL: Minibatchtraining, Multinomiale logistische Regression, L1-Regularisierung (siehe Statistik, Embeddingsvorlesungen)

Übersicht

Beweis: Cross-Entropy always bigger than entropy

$$
\begin{aligned}
0 &= \log 1 = \log \sum_x q(x) \qquad q \text{ Wverteilung} \\
&\geq \log \sum_{x,p(x)\neq 0} q(x) = \log \sum_{x,p(x)\neq 0} p(x)\frac{q(x)}{p(x)} \\
&\geq \sum_{x,p(x)\neq 0} p(x) \log \frac{q(x)}{p(x)} \qquad \log \text{ konkav} \\
&= \sum_{x,p(x)\neq 0} p(x)[\log \frac{1}{p(x)} - \log \frac{1}{q(x)}] \\
&= \sum_{x,p(x)\neq 0} p(x) \log \frac{1}{p(x)} - \sum_{x,p(x)\neq 0} p(x) \log \frac{1}{q(x)}
\end{aligned}
$$

Und damit

$$
\sum_{x,p(x)\neq 0} p(x) \log \frac{1}{q(x)} \geq \sum_{x,p(x)\neq 0} p(x) \log \frac{1}{p(x)}
$$

was zu zeigen war (die Teile mit p(x) =0 addieren nichts und verändern nichts)

Appendix: Herleitung der Ableitung I

Herleitung mit einem einzelnen w im eindimensonalem Raum für den ersten Teil der Cross-Entropy:

$f(w) := -y \log \sigma(wx+b) = -y \log \frac{1}{1+e^{-wx-b}} = y \log(1+e^{-wx-b})$

$$
\begin{aligned}
f'(w) &= y \frac{1}{1+e^{-wx-b}} e^{-wx-b}(-x) \\
&= -xy \frac{e^{-wx-b}}{1+e^{-wx-b}} \\
&= xy(-1+\frac{1}{1+e^{-wx-b}})
\end{aligned}
$$

Herleitung der Ableitung II

Herleitung mit einem einzelnen w im eindimensonalem Raum für den zweiten Teil der Cross-Entropy:

$g(w) := -(1-y)\log(1-\sigma(wx+b)) = -(1-y)\log(1-\frac{1}{1+e^{-wx-b}}) = -(1-y)\log(\frac{e^{-wx-b}}{1+e^{-wx-b}}) = (1-y)\log\frac{1+e^{-wx-b}}{e^{-wx-b}}$

$$\begin{aligned} g'(w) &= (1-y)\frac{e^{-wx-b}}{1+e^{-wx-b}}(-1)(\frac{1}{e^{-wx-b}})^2 e^{-wx-b}(-x) \\ &= x(1-y)\frac{1}{1+e^{-wx-b}} \end{aligned}$$

Herleitung der Ableitung III

Und damit die Ableitung der Cross-entropy die Summe der beiden
$f'(w) + g'(w) =$

$$
\begin{aligned}
xy(-1 + \frac{1}{1+e^{-wx-b}}) + x(1-y)\frac{1}{1+e^{-wx-b}} &= \\
-xy + xy\frac{1}{1+e^{-wx-b}} + x\frac{1}{1+e^{-wx-b}} - xy\frac{1}{1+e^{-wx-b}} &= \\
(\frac{1}{1+e^{-wx-b}} - y)x &= \\
(\sigma(wx+b) - y)x &
\end{aligned}
$$

Beweis der Konvexität der Cross-Entropy

Zu zeigen, dass die Ableitung monoton steigend ist d.h. die zweite Ableitung ist nicht-negativ

Erste Ableitung war $h(w) := (\sigma(wx+b)-y)x = \frac{1}{1+e^{-wx-b}}x - yx$
Zweite Ableitung

$$h'(w) = x(-1)(\frac{1}{1+e^{-wx-b}})^2 e^{-wx-b}(-x) = x^2(\frac{1}{(1+e^{-wx-b})})^2 e^{-wx-b}$$

Damit ist die zweite Ableitung nicht-negativ und damit die Kreuzentropie konvex.