Automatisches Erkennen von handgeschriebenen Ziffern mit Hilfe von One-vs-all Logistischer Regression und Neuronalen Netzwerken

Nicole Lubrich

14. September 2013

Ziel: Ziel ist es zwei Klassifizierer, die handgeschriebene Ziffern zwischen 0 und 9 automatisch erkennen, mit Hilfe schon bekannter Daten zu trainieren und anschließend zu vergleichen.

Beschreibung des Datensatzes: Jede Ziffer ist durch ein Bild aus Grautönen mit 20×20 Pixel gegeben. Diese Ziffernbilder werden durch 400-dimensionale Zeilenvektoren repräsentiert, bei denen jeder Eintrag eine Dezimalzahl ist, welche die Intensität des Grautones eines Pixel angibt. Ein solcher Vektor, bei dem bekannt ist welche Ziffer er verkörpert, wird Beispiel genannt. Dass heißt zu einem Beispiel gehört eine Zahl zwischen 1 und 10 (dabei steht die 10 für die Ziffer 0), welche als Label bezeichnet wird. Es stehen 5000 Beispiele zur Verfügung.

Umsetzung: Mit Hilfe der Beispiele werden eine One-vs-all Logistische Regression und ein Neuronales Netzwerk trainiert und mit 10-facher Kreuzvalidierung getestet. Dafür wird Matlab beispielsweise Octave verwendet. Zunächst werden die Beispiele in eine Matrix X und die zugehörigen Labels in einen Vektor y geladen. Für die 10-fache Kreuzvalidierung müssen die Beispiele in zehn gleichgroße Mengen unterteilt werden. Dafür wird ein 5000-dimensionaler Spaltenvektor p erzeugt, der die Zahlen 1 bis 10 mit der gleichen Häufigkeit und in zufälliger Reihenfolge enthält. Ein Beispiel, das in Zeile i von X steht, wird dann der Teilmenge l zugeordnet, wenn $p(i) = l$ ist. Nun wird für die One-vs-all Logistische Regression als auch für das Neuronale Netzwerk in separaten Schleifen jede dieser Teilmengen als Testmenge und die verbleibenden Teilmengen als Trainingsmenge verwendet, um den Klassifizierer zu trainieren und zu testen. In jedem Durchlauf wird die Genauigkeit des Klassifizierers durch

$$100 * \frac{\text{Anzahl der richtig klassifizierten Beispiele}}{\text{Anzahl der Beispiele}}$$

mit Beispielen aus der Testmenge berechnet. Abschließend wird für jeden Klassifizierer der Durchschnitt der Genauigkeiten und der Durchschnitt der Trainingszeiten aus den 10 Durchläufen der Kreuzvalidierung angeben.

Liste der verwendeten Dateien

Daten.mat - Datensatz mit Beispielen von handgeschriebenen Ziffern
Main.m - Skript, das die Algorithmen und die Kreuzvalidierung ausführt
DatenVis.m - Funktion zur Visualisierung der Daten
sigmoid.m - Berechnet die Sigmoid-Funktion $g(t) = \frac{1}{1+e^{-t}}$

sigmoidGradient.m - Berechnet die Ableitung der Sigmoid-Funktion
fmincg.m - Funktion zur Bestimmung des Minimums einer übergebenen Funktion
lrFehler.m - Berechnet den Fehler der Logistischen Regression
LogReg.m - Berechnet die optimalen Parameter der Logistischen Regression
lrLabel.m - Klassifiziert ein Beispiel mit dem Neuronalen Netzwerk
zufInitialParameter.m - Erzeugt zufällige Start-Parameter für das Neuronale Netzwerk
nnFehler.m - Berechnet den Fehler des Neuronalen Netzwerkes
nnLabel.m - Klassifiziert ein Beispiel mit dem Neuronalen Netzwerk

Beschreibung der Klassifizierer

Im Folgenden sei m die Anzahl der Trainingsbeispiele, n die Anzahl der Features (Anzahl der Pixel) und K die Anzahl der Labels. Die $m \times n$-dimensionale Matrix X enthalte die Trainingsmenge, bei der die i-te Zeile dem Trainingsbeispiel $x^{(i)}$ oder $x^{(i)}$ entspricht. Für jedes Trainingsbeispiel stehe das zugehörige Label in dem m-dimensionalen Spaltenvektor y. Mit λ wird der Regularisierungsparameter bezeichnet, der Overfitting vermeiden soll.

One-vs-all Logistische Regression: Bei der One-vs-all Logistischen Regression wird für jedes Label separat eine Logistische Regression durchgeführt. Für ein Label k entspricht die Hypothese $h_{\theta^{(k)}}(x)$ der Wahrscheinlichkeit, dass Beispiel x das Label k hat. Dabei stehen in dem $(n+1)$-dimensionalen Spaltenvektor $\theta^{(k)}$ die zu optimierenden Parameter und es ist $h_{\theta^{(k)}}(x) = g((\theta^{(k)})^{\perp} x)$ mit $g(t) = \frac{1}{1+\mathrm{e}^{-t}}$. Die optimalen Parameter minimieren den Fehler der Logistischen Regression, welcher durch

$$E(\theta^{(k)}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_{\theta^{(k)}}(x^{(i)})) + (1-y^{(i)})\log(1-h_{\theta^{(k)}}(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}(\theta_j^{(k)})^2$$

gegeben ist. Hier ist $y^{(i)} = 1$, wenn $x^{(i)}$ das Label k hat und $y^{(i)} = 0$ ansonsten (one vs all). Zur numerischen Bestimmung des Minimums wird die von Edward Rasmussen implementierte Funktion **fmincg.m** verwendet. Da diese das Gradientenabstiegsverfahren verwendet muss der Gradient der Fehlerfunktion übergeben werden. Dieser ist gegeben durch

$$\begin{aligned}
\frac{\partial E}{\partial \theta_j^{(k)}} &= \frac{1}{m}\sum_{i=1}^{m}(h_{\theta^{(k)}}(x^{(i)}) - y^{(i)})x_j^{(i)}, && \text{für } j = 0,\\
\frac{\partial E}{\partial \theta_j^{(k)}} &= \frac{1}{m}\left[\sum_{i=1}^{m}(h_{\theta^{(k)}}(x^{(i)}) - y^{(i)})x_j^{(i)} + \lambda\theta_j^{(k)}\right], && \text{für } j \geq 1.
\end{aligned}$$

Die Berechnung des Fehlers und des Gradienten erfolgt vektorisiert. Nachdem für jedes Label k die optimalen Parameter $\theta^{(k)}$ bestimmt wurden, erfolgt die Klassifizierung für ein Beispiel x, in dem ihm ein Label k zugeordnet wird, für das $h_{\theta^{(k)}}(x)$ maximal wird.

Neuronales Netzwerk: Das hier verwendete Neuronale Netzwerk besteht aus vier Schichten, einer Input-Schicht mit n Neuronen, zwei inneren Schichten mit jeweils $S = 25$ Neuronen und einer Output-Schicht mit K Neuronen. Die Hypothese $h_\theta(x)$ entspricht dabei der Output-Schicht, ist also ein K-dimensionaler Spaltenvektor. Der k-te Eintrag von $h_\theta(x)$ wird als die Wahrscheinlichkeit aufgefasst, dass Beispiel x das Label k hat. Hier steht θ stellvertretend für

eine $S \times (n+1)$-dimensionale Matrix $\Theta^{(1)}$ und einer $K \times (S+1)$-dimensionale Matrix $\Theta^{(2)}$, welche die zu optimierenden Parameter des Neuronalen Netzwerkes enthalten. Die optimalen Parameter minimieren den Fehler des Neuronalen Netzwerkes, welcher gegeben ist durch

$$\begin{aligned} E(\theta) = &- \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[y_k^{(i)} \log(h_\theta(x^{(i)})_k) + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k) \right] \\ &+ \frac{\lambda}{2m} \sum_{s=1}^{S} \left[\sum_{j=1}^{n} (\Theta_{sj}^{(1)})^2 + \sum_{k=1}^{K} (\Theta_{ks}^{(2)})^2 \right]. \end{aligned}$$

Hier ist $y^{(i)}$ ein K-dimensionaler Spaltenvektor, bei dem $y_l^{(i)} = 1$ und $y_k^{(i)} = 0$ ist für $k \neq l$, wenn $x^{(i)}$ das Label l hat. Die Hypothese $h_\theta(x)$ ist eine Hintereinanderausführung von Multiplikation mit den Matrizen $\Theta^{(1)}$ und $\Theta^{(2)}$ sowie der Gewichtsfunktion $g(t) = \frac{1}{1+e^{-t}}$. Die Berechnung erfolgt schrittweise. Zunächst setzt man $a^{(1)} = \binom{1}{x^\perp}$, dass heißt man fügt dem transponierten Beispiel x als ersten Eintrag eine Eins hinzu. Dann berechnet man

$$z^{(2)} = g(\Theta^{(1)} a^{(1)})$$

und fügt wieder eine Eins hinzu: $a^{(2)} = \binom{1}{z^{(2)}}$. Schließlich erhält man

$$h_\theta(x) = g(\Theta^{(2)} a^{(2)}).$$

$$x \xrightarrow{+1} a^{(1)} \xrightarrow{g \circ \Theta^{(1)}} z^{(2)} \xrightarrow{+1} a^{(2)} \xrightarrow{g \circ \Theta^{(2)}} h_\theta(x)$$

Da der Fehler mit **fmincg.m** minimiert wird, muss der Gradient bestimmt werden. Dies geschieht mit Hilfe des Backpropagation Algorithmus. Zunächst wird für $l = 1, 2$ eine Null-Matrix $\Delta^{(l)}$ mit denselben Dimensionen wie $\Theta^{(l)}$ initialisiert. Dann werden für $i = 1, ..., m$ die folgenden vier Schritte durchgeführt:

1. Berechne $\delta^{(3)} = h_\theta(x^{(i)}) - y^{(i)}$, wobei, wenn $x^{(i)}$ das Label k hat, der k-te Eintrag von $y^{(i)}$ gleich 1 und alle anderen Einträge gleich 0 sind.

2. Berechne $\tilde{\delta}^{(2)} = (\Theta^{(2)})^\perp \delta^{(3)}$.

3. Entferne den erste Eintrag von $\tilde{\delta}^{(2)}$ und berechne $\delta^{(2)} = \tilde{\delta}^{(2)} .* g'(z^{(2)})$.

4. Setze $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^\perp$ für $l = 1, 2$.

Der Gradient ist dann gegeben durch

$$\begin{aligned} \frac{\partial E}{\partial \Theta_{ij}^{(l)}} &= \frac{1}{m} \Delta_{ij}^{(l)}, && \text{für } j = 0, \\ \frac{\partial E}{\partial \Theta_{ij}^{(l)}} &= \frac{1}{m} \left(\Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \right), && \text{für } j \geq 1 \end{aligned}$$

und für $l = 1, 2$. Die Berechnung des Fehlers und des Gradienten erfolgt vektorisiert. Nachdem die optimalen Parameter in $\Theta^{(1)}$ und $\Theta^{(2)}$ bestimmt wurden, wird ein Beispiel x klassifiziert, in dem ihm ein Label k zugeordnet wird, wenn der k-te Eintrag von $h_\theta(x)$ maximal ist.

Vergleich der beiden Klassifizierer: Das Neuronale Netzwerk erreicht mit knapp 92 eine etwas höhere durchschnittliche Genauigkeit als die One-vs-all Logistische Regression mit gut 90, hat dafür aber eine wesentlich längere durchschnittliche Trainingszeit von 1 Minuten und 1 Sekunden im Vergleich zu 1 Sekunden.