Computer-Linguistische Anwendungen

CLA | B.Sc. | LMU

Perzeptron Algorithmus

Computerlinguistische Anwendungen

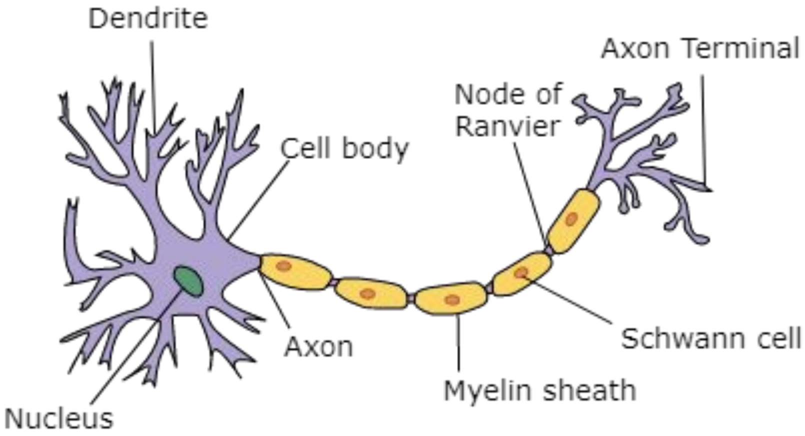
10”

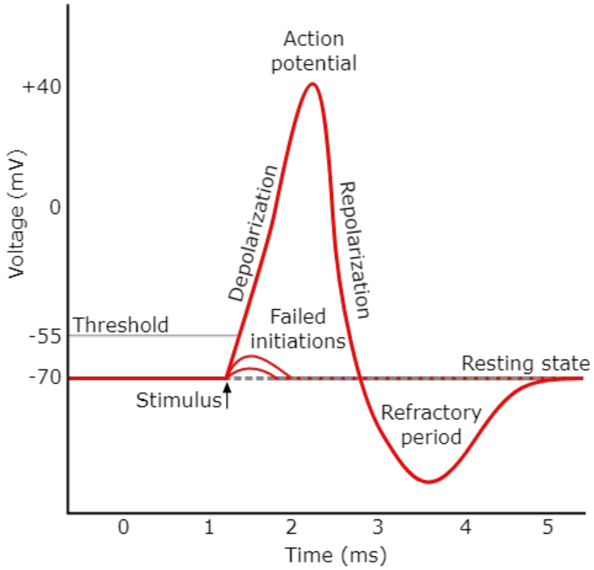
Perzeptron Algorithmus: Idee

- Der Perzeptron-Algorithmus fällt in die Gruppe der **diskriminativen** Modelle
- Optimiert die Qualität der Vorhersage.
- *Gegeben die Merkmale, was ist die korrekte Klasse?*
- P(Klasse | Merkmale) = ?

Perzeptron Algorithmus

Künstliches Neuron

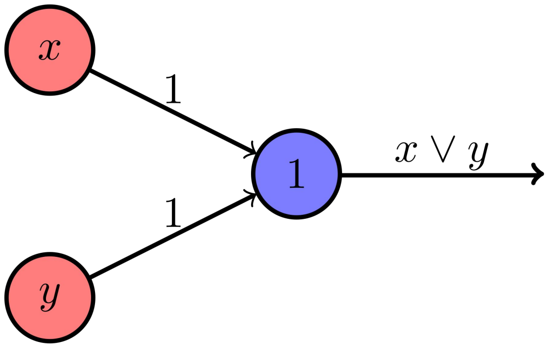




https://commons.wikimedia.org/wiki/File:Neuron.svg https://commons.wikimedia.org/wiki/File:Action_potential.svg

Perzeptron Algorithmus

Künstliches Neuron



Logisches ODER als künstliches Neuron
Das **Neuron** feuert nur, wenn die Summe der eingehenden **Input-Neuronen** genau 1 ist. Die Input-Neuronen können ebenfall nur feuern (1) oder nicht feuern (0).

Umwandlung eines Eingabevektors in einen Ausgabevektor durch trainierbare Gewichte.

https://de.wikipedia.org/wiki/Perzeptron#/media/Datei:Perceptron-or-task.svg

Perzeptron Algorithmus: Idee

- Für jedes mögliche Merkmal (z.B. Wort) wird ein Gewicht gelernt (Zahl, die positiv oder negativ sein kann)
- Vorhersage der Klasse für eine Instanz:
  - Für jedes Merkmal wird der Wert des Merkmals (z.B. Vorkommen im Dokument) mit dem jeweiligen Merkmals-Gewicht multipliziert, die Ergebnisse werden aufsummiert.
  - Ist der resultierende Wert größer (oder gleich) 0, wird die positive Klasse vorhergesagt (**True**), ansonsten die negative Klasse (**False**)
- Der Perzeptron-Algorithmus passt die Merkmals-Gewichte iterativ so an, dass Fehlklassifikation möglichst minimiert wird.

Wiederholung: Vektor Multiplikation (Skalarprodukt)

$$\boldsymbol{a} = \begin{bmatrix} 0,5 \\ 0 \\ 5 \\ 3 \\ 2 \end{bmatrix}; \qquad \boldsymbol{b} = \begin{bmatrix} 1 \\ 6 \\ 0 \\ 4 \\ 1,5 \end{bmatrix}$$

$$\boldsymbol{a}^T =$$

$$\boldsymbol{a}^T \boldsymbol{b} =$$

Wiederholung: Vektor Multiplikation (Skalarprodukt)

$$\boldsymbol{a} = \begin{bmatrix} 0,5 \\ 0 \\ 5 \\ 3 \\ 2 \end{bmatrix}; \qquad \boldsymbol{b} = \begin{bmatrix} 1 \\ 6 \\ 0 \\ 4 \\ 1,5 \end{bmatrix}$$

$$\boldsymbol{a}^T = \quad [0.5, \ \ 0, \ \ 5, \ \ 3, \ \ 2]$$

$$\boldsymbol{a}^T \boldsymbol{b} = \quad [0.5, \ \ 0, \ \ 5, \ \ 3, \ \ 2] \begin{bmatrix} 1 \\ 6 \\ 0 \\ 4 \\ 1,5 \end{bmatrix}$$

Wiederholung: Vektor Multiplikation (Skalarprodukt)

$$\boldsymbol{a} = \begin{bmatrix} 0,5 \\ 0 \\ 5 \\ 3 \\ 2 \end{bmatrix}; \qquad \boldsymbol{b} = \begin{bmatrix} 1 \\ 6 \\ 0 \\ 4 \\ 1,5 \end{bmatrix}$$

$$\boldsymbol{a}^T = \quad [0.5, \ 0, \ 5, \ 3, \ 2]$$

$$\boldsymbol{a}^T\boldsymbol{b} = [0.5, \ 0, \ 5, \ 3, \ 2] \begin{bmatrix} 1 \\ 6 \\ 0 \\ 4 \\ 1,5 \end{bmatrix} = 0.5 + 12 + 3 = 15.5$$

→ *dot product* : a ● b

Wiederholung: Vektor Multiplikation (Skalarprodukt)

- Jede Instanz *i* kann als Vektor $x^{(i)}$ dargestellt werden
- Die Anzahl der Komponenten des Vektors entspricht der Größe des Merkmalsraums (z.B. des Vokabulars)
- Bei einem Vokabular der Größe *n* ist $x^{(i)} \in \mathbf{R}^n$
- Jede Komponente des Vektors entspricht einem Wort

Beispiel Instanz ...

movie
waste
time

... als Vektor für das Vokabular

`[movie, entertain, highly, waste,`

$$x^{(i)} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Implementierung: Da die meisten Einträge 0 sind, wählen wir eine effiziente Darstellung mit *dicts* (Wort → wie oft es vorkommt)

Vektor Multiplikation mit Gewichtsvektor

Gewichtsvektor:

$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix}$$

Vorhersagewert (Score) für Instanz *i* :

$$x^{(i)T} w = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} = w_1 + w_4 + w_5$$

Implementierung: Auch für den Gewichtsvektor wählen wir eine Darstellung mit Dictionaries (Wort → Gewicht)

Entscheidungsregel:

$$x^{(i)T} w \geq 0 \Rightarrow \text{True}$$
$$x^{(i)T} w < 0 \Rightarrow \text{False}$$

Vorhersage

Entscheidungsregel:

$$x^{(i)T} w \geq 0 \Rightarrow \text{True}$$
$$x^{(i)T} w < 0 \Rightarrow \text{False}$$

Beispiel:

movie waste time	highly entertain movie

$$x^{(1)T} w =$$
$$w_{\text{movie}} + w_{\text{waste}} + w_{\text{time}}$$
$$x^{(2)T} w =$$
$$w_{\text{movie}} + w_{\text{entertain}} + w_{\text{highly}}$$

- Welche Gewichte würden zu einer Fehlklassifikation beider Instanzen führen (beliebiges Beispiel)?
- Wie müssten diese Gewichte korrigiert werden, damit die Instanzen richtig klassifiziert werden?

Vorhersage

Entscheidungsregel:

$$x^{(i)T}w \geq 0 \Rightarrow \text{True}$$
$$x^{(i)T}w < 0 \Rightarrow \text{False}$$

Beispiel:

$$x^{(1)T}w =$$
$$w_{\text{movie}} + w_{\text{waste}} + w_{\text{time}}$$
$$x^{(2)T}w =$$
$$w_{\text{movie}} + w_{\text{entertain}} + w_{\text{highly}}$$

Gewichte, die zu einer Fehlklassifikation beider Instanzen führen:

z.B. $w_{\text{movie}} = 1.0$; $w_{\text{waste}} = -2.0$; $w_{\text{time}} = 1.5$; $w_{\text{entertain}} = -2.0$; $w_{\text{highly}} = 0.5$

Korrektur der Gewichte:

- w_{waste} *und/oder* w_{time} *muss verkleinert werden*
- $w_{\text{entertain}}$ *und/oder* w_{highly} *muss vergrößert werden.*
- w_{movie}*: Unentschieden für beide Instanzen.*
- *alle anderen Gewichte neutral.*

Anpassung der Gewichte (Perzeptron-Update)

Idee:

- Falls richtige Vorhersage für Trainings-Instanz: mache nichts
- Sonst: Erhöhe/verringere Gewichte für jede Instanz so, dass der Score sich in die richtige Richtung verändert.
- Gewichte für Merkmale, die besonders häufig mit einer der beiden Klassen vorkommen, erhalten viele Anpassungen in die jeweilige Richtung.
- Die Gewichte für Merkmale, die in beiden Klassen ungefähr gleich häufig vorkommen (z.B. movie) werden mal in die eine und mal in die andere Richtung verändert

Perzeptron-Update für eine Instanz

Idee:

- Wenn das Label übereinstimmt, kein Update der Gewicht
- Ansonsten:
  - Wenn Vorhersage True und wahres Label False: error = 1
    (Verringern der Gewichte in Abhängigkeit des Merkmalswertes)
  - Wenn Vorhersage False und wahres Label True: error = -1
    (Erhöhen der Gewichte in Abhängigkeit des Merkmalswertes)
  - Update für Gewicht w_j (und Merkmalsvorkommen $x_j^{(i)}$)
    $w_j \leftarrow w_j - \text{error} * x_j^{(i)}$

Perzeptron-Algorithmus

Eingabe:

- Merkmalsvektoren (instances) $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots \mathbf{x}^{(n)}\}$

- Labels $\{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots \mathbf{y}^{(n)}\}$

Ausgabe:

- Merkmalsgewichtsvektoren *w*

```
procedure PerceptronWeights(instances, labels)
    w = 0
    repeat
        for i = 1 to n do
            prediction = (x^(i)T w ≥ 0)
            if prediction == True and y^(i) == False then
                error = 1
            else if prediction == False and y^(i) == True then
                error = −1
            else error = 0
            end if
            w = w − error · x
        end for
    until stopping criterion
    return w
end procedure
```

Hinweise

Mögliche Abbruchkriterien:

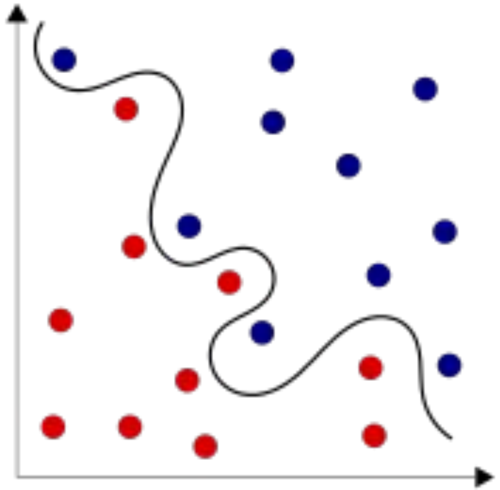
- Vorgegebene Anzahl an Iterationen erreicht
- Perfekte Klassifikation auf den Trainingsdaten
- Keine Fehlerreduktion auf den Entwicklungsdaten
  $\Rightarrow$ Man nimmt dann die Gewichte aus der vorherigen Iteration.

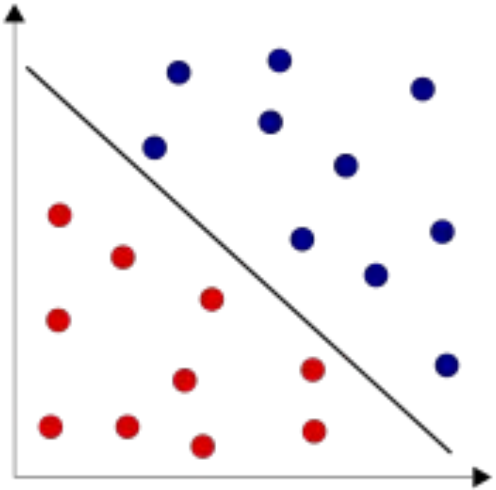
In unserem Fall waren die Merkmalswerte ganze Zahlen (Wortvorkommen), der Perzeptron-Algorithmus funktioniert aber auch mit nicht-ganzzahligen und negativen Werten.

Der Perzeptron-Algorithmus konvergiert zu der perfekten Klassifikation der Trainingsdaten, wenn diese linear trennbar sind.

- *linear trennbar*: es gibt eine Gewichtskombination mit perfekter Klassifikation
- nicht alle Trainingsdaten sind trennbar
- eine perfekte Klassifikation auf Trainingsdaten generalisiert oft schlecht

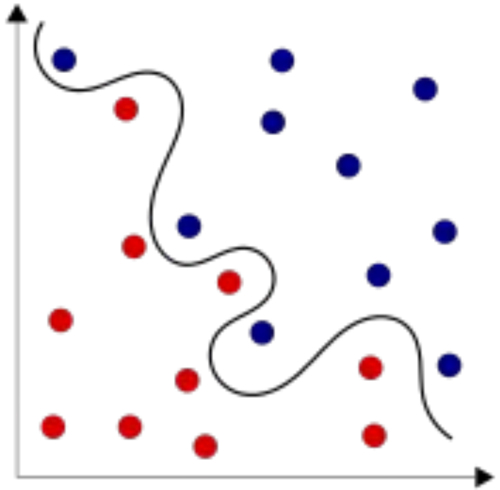
Hinweise: Lineare Trennbarkeit

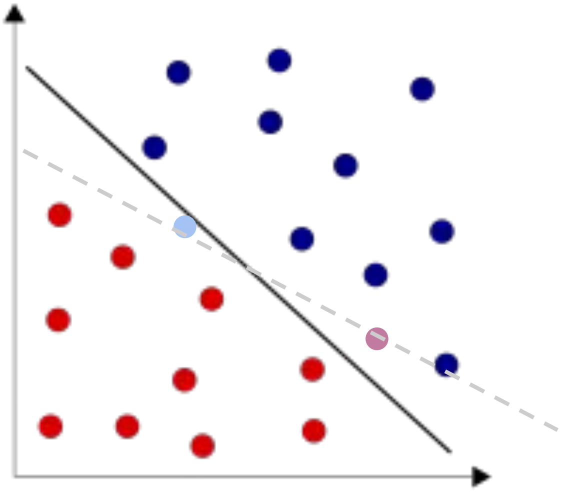




https://de.wikipedia.org/wiki/Lineare_Separierbarkeit

Hinweise: Lineare Trennbarkeit





https://de.wikipedia.org/wiki/Lineare_Separierbarkeit

Hyper-Parameter für den Perzeptron-Algorithmus

Hyper-Parameter: Parameter, die dem Algorithmus vorgegeben werden (=die dieser nicht automatisch lernt) ...

... werden auf den Entwicklungsdaten ausgewählt

Welche Hyper-Parameter für Perzeptron?

Hyper-Parameter für den Perzeptron-Algorithmus

Welche Merkmale, Anzahl der Merkmale (z.B. 1000 oder 10000 häufigste Wörter)

Anzahl der Trainings-Iterationen

- Vergleiche nach jeder Iteration die Accuracy auf den Entwicklungsdaten
- Wähle Gewichtswerte der Iteration mit bester Entwicklungs-Accuracy
- *"Early-stopping with patience n"*: Breche Training ab, falls $n + 1$ Iterationen in Folge keine Verbesserung auf den Entwicklungsdaten beobachtet wird

Step-Size: Ist der Datensatz linear trennbar, wird der Perzeptron-Algorithmus garantiert eine Lösung finden, unabhängig von der Schrittweite, die wir wählen. Wir können beispielsweise eine Schrittweite von 0,1 oder 0,01 oder 0,001 wählen, und der Algorithmus wird dennoch konvergieren und eine Entscheidungsgrenze finden, die die beiden Klassen trennt.