# Unsupervised Deep Embedding for Clustering Analysis

## Bachelorseminar Data Mining

Lukas Mahr

Ludwig-Maximilians-Universität München





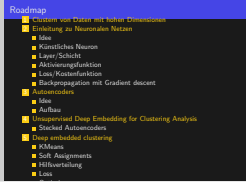
Unsupervised Deep Embedding for Clustering Analysis
Bachelorseminar Data Mining
Lukas Mahr
Ludwig-Maximilians-Universität München

# Roadmap






└Roadmap

Roadmap
1. Clustern von Daten mit hohen Dimensionen
2. Einleitung zu Neuronalen Netzen
   - Idee
   - Künstliches Neuron
   - Layer/Schicht
   - Aktivierungsfunktion
   - Loss/Kostenfunktion
   - Backpropagation mit Gradient descent
3. Autoencoders
   - Idee
   - Aufbau
4. Unsupervised Deep Embedding for Clustering Analysis
   - Stecked Autoencoders
5. Deep embedded clustering
   - KMeans
   - Soft Assignments
   - Hilfsverteilung
   - Loss

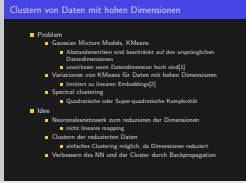
# Clustern von Daten mit hohen Dimensionen

- Problem
  - Gaussian Mixture Models, KMeans
    - Abstandsmetriken sind beschränkt auf den ursprünglichen Datendimensionen
    - unwirksam wenn Datendimension hoch sind[1]
  - Variationen von KMeans für Daten mit hohen Dimensionen
    - limitiert zu linearen Embeddings[2]
  - Spectral clustering
    - Quadratische oder Super-quadratische Komplexität
- Idee
  - Neuronalesnetzwerk zum reduzieren der Dimensionen
    - nicht lineares mapping
  - Clustern der reduzierten Daten
    - einfaches Clustering möglich, da Dimensionen reduziert
  - Verbessern des NN und der Cluster durch Backpropagation





Clustern von Daten mit hohen Dimensionen




viele Daten Punkte viele Distanzen zu berechnen schwierig zu visualisieren ohne die Dimensionen zu reduzieren Komplexität von Kmeans die exponentiell ansteigt

# Neuronale Netze

# Einleitung zu Neuronalen Netzen

- Nicht-lineare statistische Modelle zur Informationsverarbeitung
- Informationsverarbeitung umfasst hierbei unter anderem
  - Klassifikation
  - Prognosenerstellung
- Units der Neuronalen Netze angelehnt an Neuronen
  - Inputs zusammenfassen
  - Mit Schwellenwert vergleichen bzw. aktivieren
- Verbindungen zwischen Units angelehnt an Synapsen
  - Gewichtung mit verstärkender oder schwächender Wirkung





Einleitung zu Neuronalen Netzen
Nicht-lineare statistische Modelle zur Informationsverarbeitung
Informationsverarbeitung umfasst hierbei unter anderem
Klassifikation
Prognosenerstellung
Units der Neuronalen Netze angelehnt an Neuronen
Inputs zusammenfassen
Mit Schwellenwert vergleichen bzw. aktivieren
Verbindungen zwischen Units angelehnt an Synapsen
Gewichtung mit verstärkender oder schwächender Wirkung

wofür braucht man neuronale Netze
- Klassifizierung von Daten
- Prognose eines bestimmten Wertes
- Neuronen an Neuronen in unserem Gehirn angelehnt
- Verbindungen zwischen Künstlichen Neuronen an Synapsen angelehnt
- supervised learning = überwachtes lernen
– man besitzt Daten mit Beschriftungen (labels)
- unsperviesed learning nicht überwachtes lernen
– unbeschriftete Daten, also es sind keine labels vorhanden

# Einleitung zu Neuronalen Netzen

## Künstlichen Neurons

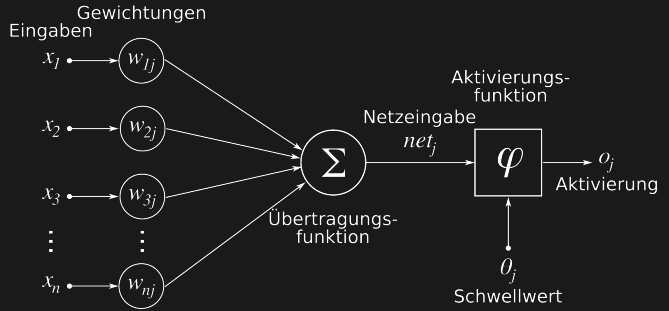
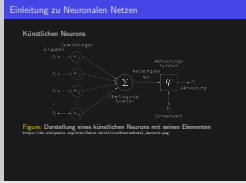



Figure: Darstellung eines künstlichen Neurons mit seinen Elementen
https://de.wikipedia.org/wiki/Datei:ArtificialNeuronModel_deutsch.png







$x_1, ..., x_n$ sind die input variablen, jede der Eingabe variablen besitzt ein Gewicht, $w_{1j}, ..., w_{nj}$. Diese werden Multipliziert und davon dann die summe berechnet. Hier die Übertragungsfunktion. Dazu wird ein Bias, in dem Fall der Schwellenwert gerechnet. Als letztes gibt es noch die Aktivierungsfunktion die meistens einen Wert zwischen 0 und 1 zurückgibt. Das ist dann der input für das nächste Neuron.

# Einleitung zu Neuronalen Netzen

## Layer/Schichten

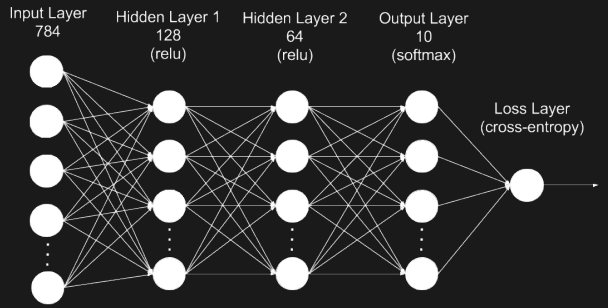
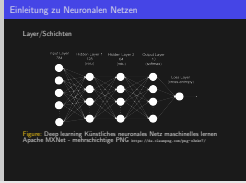



Figure: Deep learning Künstliches neuronales Netz maschinelles lernen Apache MXNet - mehrschichtige PNG https://de.cleanpng.com/png-x3zkr7/








Layer/Schicht sind mehrere Neuronen die mit allen Neuronen des nächsten Layer/Schicht verbunden sind. Alle Neuronen in einem Layer haben die gleiche Aktivierungsfunktion. Hidden Layer haben meistens die Aktivierungsfunktion rectified linear, da diese recht einfach und schnell zu berchnen ist. Das outputlayer hat meistens eine etwas kompliziertere Funktion wie softmax oder sigmoid. Abhängig von der Aufgabe des Netzwerkes. Letztes Layer hier direkt mit dem Loss

# Einleitung zu Neuronalen Netzen

## Aktivierungsfunktionen

Figure: Rectifier-Aktivierungsfunktion
https://de.wikipedia.org/wiki/Datei:Activation_rectified_linear.svg

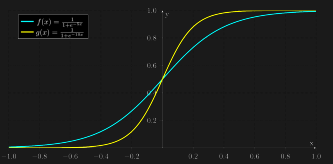
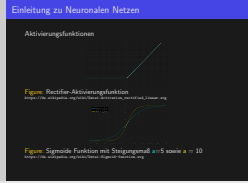



Figure: Sigmoide Funktion mit Steigungsmaß a=5 sowie a = 10
https://de.wikipedia.org/wiki/Datei:Sigmoid-function.svg








alles negativ ist wird bei relu zu 0 während bei sigmoid, abhängig von der Steigung Werte zwischen -1 und 1 möglich sind

# Einleitung zu Neuronalen Netzen

Loss/Kostenfunktion

**Mean Squared Error**

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

**Mean absolute error**

$$MAE = \frac{\sum_{i=1}^{n}|\hat{y}_i - y_i}{n}$$

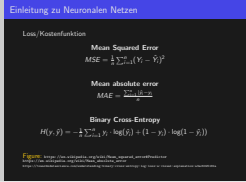
**Binary Cross-Entropy**

$$H(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n} y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i))$$

Figure: https://en.wikipedia.org/wiki/Mean_squared_error#Predictor
https://en.wikipedia.org/wiki/Mean_absolute_error
https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a








Man berechnet immer den unterschied zwischen den wahren labeln und den predicteden labeln um zu erkenne wie weit diese auseinander liegen. Es wird immer versucht den Loss zu minimieren. Also ein Minimum der Kostenfunktion zu finden. Die Parameter der Funktion, welche angepasst werden müssen sind alle weights und biases der einzelnen Neuronen und den Layern.

## Backpropagation mit Gradient descent

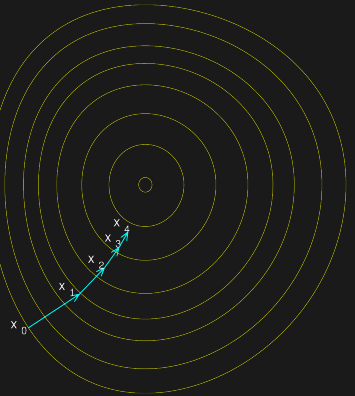
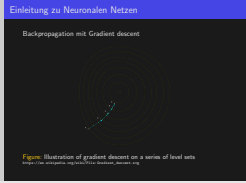



Figure: Illustration of gradient descent on a series of level sets
https://en.wikipedia.org/wiki/File:Gradient_descent.svg








Über Backpropagation wird hier mit z.B Gradient Descent die Loass funktion minimiert. Der Gradient der Loss/ Kostenfunktion wird für alle wigths and biases gleichzeitig berechnet. Man kann sich das vorstellen, wie eine Kugel die man einen in einer Hügellandschaft rollen lässt ein kleinen schritten und zwischen den schritten immer nach der Steigung des Abhanges schaut und dabei versucht die Kugel in das tiefste Tal zu bekommen.

# Autoencoder





Autoencoders
Autoencoder

Ein Autoencoder ist ein feedforward Neural network, was versucht den input zu Kopieren. Das hört sich im ersten Moment nutzlos an, hat aber doch ein paar Anwendungsmöglichkeiten. Dazu gehört z.B Denosing oder Reduktion des Features Spaces

# Autoencoders

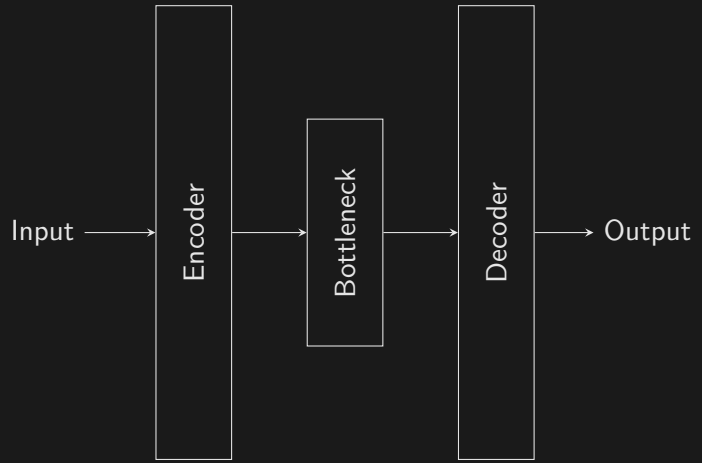
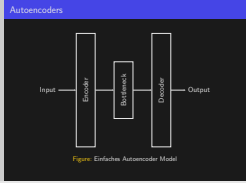



Figure: Einfaches Autoencoder Model








- Aufbau, Das Netzwerk besteht im Grunde aus 2 Teilen.
- Encoder und Decoder
- Encoder Transformiert die Input Daten in eine kleinere gewünschte Dimension
- Decoder Transformiert die Daten aus der Kleinen Dimension zurück in die Orginal Dimensionen.
- Hoffnung das der Encoder die Daten auf die Wichtigsten Features reduziert.

# Autoencoders

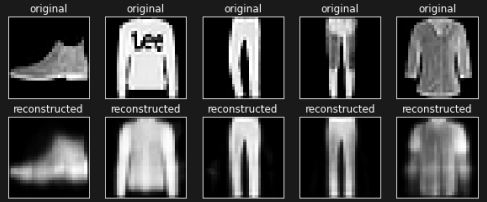



Figure: Orginal und Decoded Bilder

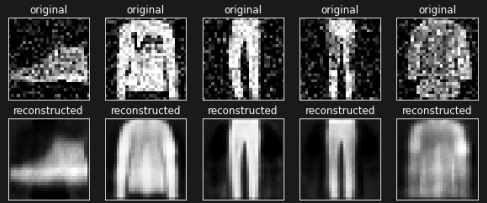
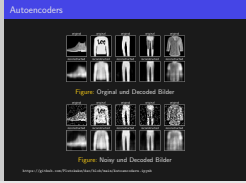



Figure: Noisy und Decoded Bilder

https://github.com/Plutokekz/dec/blob/main/Autoencoders.ipynb








- 784 input zu 10 zu 784 (28*28)
- Bottleneck size = 10 oder feature redutktion auf 10
- Random noise

# DEC

## Unsupervised Deep Embedding for Clustering Analysis





Unsupervised Deep Embedding for Clustering Analysis

DEC
Unsupervised Deep Embedding for Clustering Analysis

DEC has two phases: (1) parameter initialization with a deep autoencode parameter optimization (i.e., clustering), where we iterate between computing an auxiliary target distribution and minimizing the Kullback–Leibler (KL) divergence to it.
- Besteht aus 2 teilen
– embedding mapping in den kleineren raum Z
— Stacked Autoencoders
– clustering
— Clustern der embddedten Daten — Embeddings verbessern so wie die Cluster anpassen

# Stacked Autoencoders

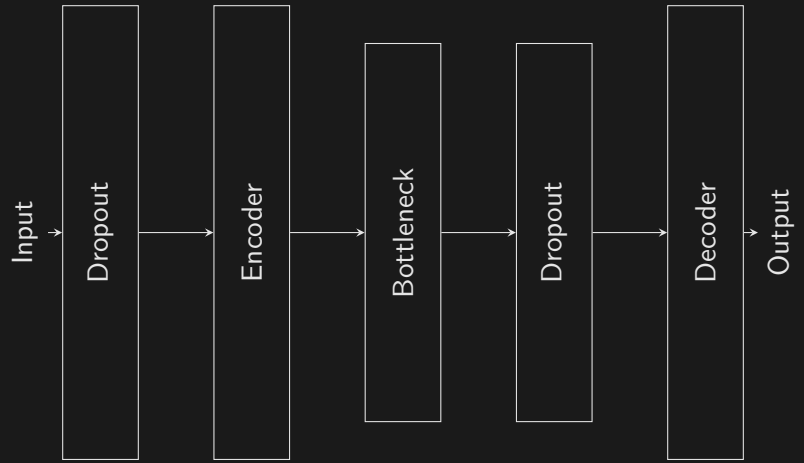
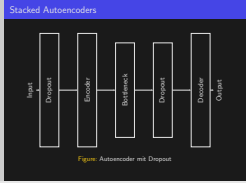



Figure: Autoencoder mit Dropout








Stacked Autoincoders ist genau das was der name Sagt.
mehrere Autoencoder hintereinander.
man started mit einem Autoencoder, den man Trainniert.
dann entfernt man den decoder Teil
und setzt an diese stelle den nächsten Autoencoder, wird wieder trainniert bis die gewünschte anzahl der stacks erreicht ist.
Besonderheit man trainiert die folgenden Autoencoder mit dem Output des vorigen encoders.

# Stacked Autoencoders

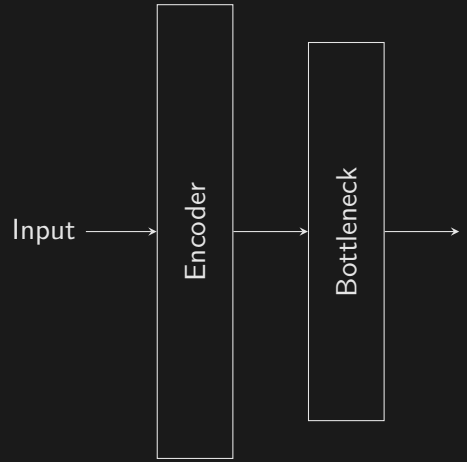
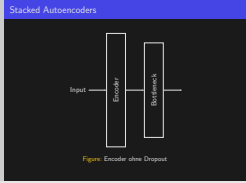



Figure: Encoder ohne Dropout








Encoder teil ohne noise/dropout und ohne dem decoder

# Stacked Autoencoders

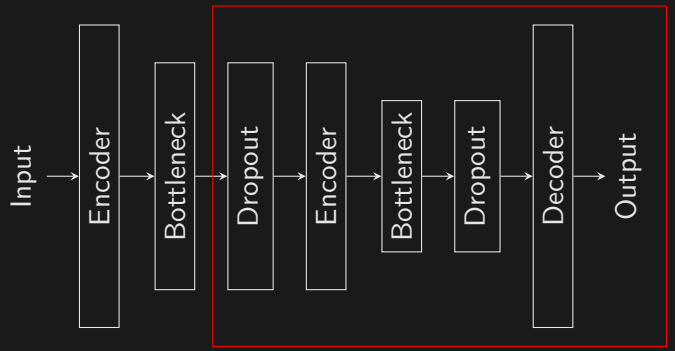
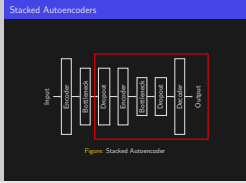



Figure: Stacked Autoencoder








Aktivierungsfunktion relu außer im letzten Decoder und Encoder Layer sigmoid
für die zero-mean imageas
Zero-mean images -¿ durchschnitt pro pixel über alle bilder im Datensatzt das bild wird dann von jedem bild abgezogen und damit liegt der durschnitt bei den Bilder bei null deswegen zero mean images.
loss = least-square

nach dem Training des Autoencoders wird der Encoder als nicht lineares mapping für die daten genutzt die man Clustern möchte

# Deep embedded clustering





Deep embedded clustering
Deep embedded clustering

# Deep embedded clustering

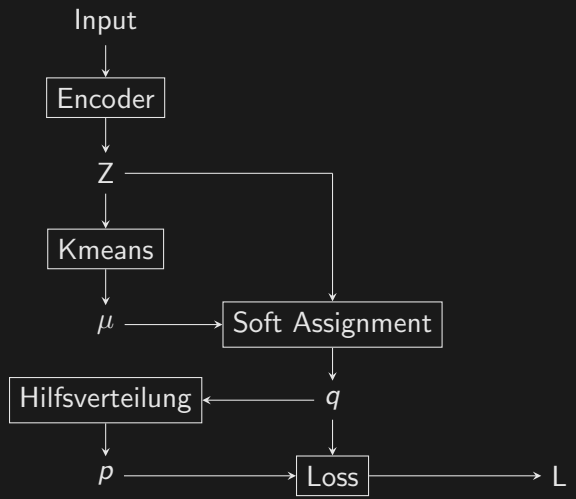
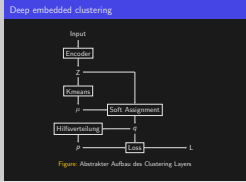



Figure: Abstrakter Aufbau des Clustering Layers



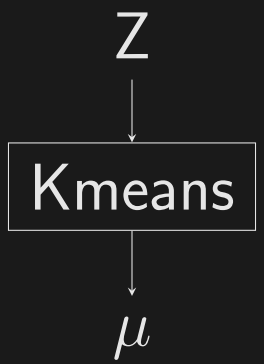


Deep embedded clustering




- input daten durch sae werden auf kleinen feature raum Z abgebildet
- clustern der Daten Z es entstehen die Cluster Schwerpunkte $\mu$
- soft assignments zwischen $\mu$ und z, wird zu q
- Hilfsverteilung aus q, wird zu p
- Loss zwischen q und p

# KMeans



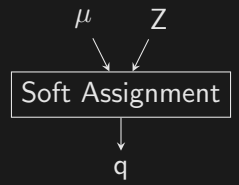





KMeans
Z
Kmeans
$\mu$

Clustern der Daten aus dem SAE
- Cluster Schwerpunkte für Soft Assignment
- eigentliche Cluster die man haben will

# Soft Assignments




$$q_{ij} = \frac{(1+||z_i-\mu_j||^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1+||z_i-\mu'_j||^2/\alpha)^{-\frac{\alpha+1}{2}}}$$





Soft Assignments
μ Z
Soft Assignment
q
$q_{ij} = \frac{(1+||z_i-\mu_j||^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1+||z_i-\mu'_j||^2/\alpha)^{-\frac{\alpha+1}{2}}}$

Student t-Verteilung um die Ähnlichkeit zwischen $z_i$ und $\mu_j$ zu messen
Freiheitsgrad ist 1[3]
$q_{ij}$ ist die Wahrscheinlichkeit die $z_i$ dem Cluster $q_j$ zuzuordnen

# Hilfsverteilung


$$p_{ij} = \frac{q_{ij}^2 / f_j}{\sum_{j'} q_{ij'}^2 / f_{j'}}$$

$$f_j = \sum_j q_{ij}$$





Hilfsverteilung
q
Hilfsverteilung
p
$p_{ij} = \frac{q_{ij}^2 / f_j}{\sum_{j'} q_{ij'}^2 / f_{j'}}$
$f_j = \sum_j q_{ij}$

Eigenschaften der Hilfsverteilung
- Prognosen Verstärken
- mehr Gewicht auf genaue Datenpunkte
- loss Verteilung von jedem Cluster Schwerpunkt normalisieren um zu verhindern das der versteckte Merkmals Bereich nicht verzehrt wird.
— man will das letzte Encoder Layer schützen.
$f_j$ Soft-Cluster-Frequenzen
Erklärung wie die Eigenschaft erreicht werden im Fazit.

# Loss

## Kullback-Leibler-Divergenz


$$L = KL(P||Q) \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$





Loss
Kullback-Leibler-Divergenz
q p
Loss
L
$L = KL(P||Q) \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$

Kullback-Leibler-Divergenz oder Kullback-Leibler-Abstand
- maß für Unterschiedlichkeit zweier Wahrscheinlichkeitsverteilungen
- hier als loss funktionen
- unterschied zwischen q und p
- der unterschied wird versucht zu minimieren.

# Optimierung

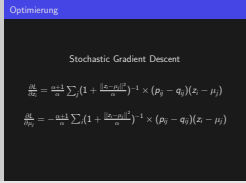
## Stochastic Gradient Descent

$$\frac{\partial L}{\partial z_i} = \frac{\alpha+1}{\alpha} \sum_j (1 + \frac{||z_i - \mu_j||^2}{\alpha})^{-1} \times (p_{ij} - q_{ij})(z_i - \mu_j)$$

$$\frac{\partial L}{\partial \mu_j} = -\frac{\alpha+1}{\alpha} \sum_i (1 + \frac{||z_i - \mu_j||^2}{\alpha})^{-1} \times (p_{ij} - q_{ij})(z_i - \mu_j)$$








Gradient von L wird berechnet in Abhängigkeit von $q_i$ und $\mu_j$
$\frac{\partial L}{\partial z_i}$ wird dann für backpropagation genutzt.
wird Optimiert bis eine bestimmter Prozentsatz an Cluster punkten nicht mehr das Cluster wechselt, zwischen zwei fortlaufenden Iterationen.

# Vorherige Arbeiten

andere Clustering algorithmen ? andere Dimensions-Reduktions-algorithmen





Vorherige Arbeiten
andere Clustering algorithmen ? andere Dimensions-Reduktions-algorithmen

# Von wem ist das Paper

Junyuan Xie
University of Washington

Ross Girshick
Facebook AI Research (FAIR)

Ali Farhadi
University of Washington





Von wem ist das Paper

Junyuan Xie
University of Washington
Ross Girshick
Facebook AI Research (FAIR)
Ali Farhadi
University of Washington

# Referenzen


- Steinbach, Michael, Ertöz, Levent, and Kumar, Vipin. The challenges of clustering high dimensional data. In *New Directions in Statistical Physics,* pp. 273–309. Springer, 2004.
- Ye, Jieping, Zhao, Zheng, and Wu, Mingrui. Discriminative k-means for clustering. In *NIPS*, 2008.
- van der Maaten, Laurens. Learning a parametric embedding by preserving local structure. In *International Conference on Artificial Intelligence and Statistics*, 2009.






Referenzen
Steinbach, Michael, Ertöz, Levent, and Kumar, Vipin. The challenges of clustering high dimensional data. In *New Directions in Statistical Physics*. pp. 273–309. Springer, 2004.
Ye, Jieping, Zhao, Zheng, and Wu, Mingrui. Discriminative k-means for clustering. In *NIPS*, 2008.
van der Maaten, Laurens. Learning a parametric embedding by preserving local structure. In *International Conference on Artificial Intelligence and Statistics*, 2009.