# Bedeutende Vektoren

*Reinhard Oldenburg*

Vektoren können Punkte, Richtungen, Kräfte und Bewegungen beschreiben – und im Kontext der großen Sprachmodelle der künstlichen Intelligenz können Sie auch „Bedeutung“ transportieren (etwa, dass „three“ und „drei“ die gleiche Bedeutung haben). Das Folgende beschreibt einen open ended Zugang zu mathematischen Fragen rund um die Bedeutung von Embedding-Vektoren. Minimal kann man daraus eine Übungsaufgabe zur analytischen Geometrie machen, maximal ergibt sich ein Thema für Facharbeiten im Grenzgebiet zur Informatik.

### *Philosophie und Mathematik*

Dass künstliche Intelligenz unsere Gesellschaft tiefgreifend verändert, steht außer Frage und wird intensiv diskutiert. Weniger klar ist, welche Relevanz das für die Inhalte des Mathematikunterrichts hat. Es scheint aber offensichtlich, dass KI weiter relativiert, wie wichtig konkrete Fakten sind: Wer etwa die Formel für das Pyramidenvolumen vergessen hat, konnte sich schon lange durch Formelsammlungen helfen lassen, noch schneller durch Google und mit KI kommt, wenn gewünscht, auch eine Begründung und die Anwendung auf die konkrete Situation, die zur Frage Anlass gab. Faktenwissen wird deswegen nicht obsolet, hilft es doch, Halluzinationen und falsche Antwort der KI-Systeme zu erkennen, aber der Umfang kann gegebenenfalls reduziert werden, und wenn der Unterricht dank KI-Tutoren effektiver wird. Falls dadurch Zeit gespart wird, könnte man sie dazu verwenden, sich neuen Fragen zuzuwenden. Kandidaten dafür gibt es im Überfluss – es gibt eine Unzahl von mathematischen Fragen jenseits der üblichen Curricula, die Jugendlichen zugänglich sind. KI und andere Computerwerkzeuge relativieren aber die Relevanz von kleinteiligem Wissen und es scheint daher sinnvoll, sich den großen Fragen und intellektuell reizvollen Fragen zuzuwenden - etwa denen der Philosophie.

Philosophische Fragen der Mathematik werden im Unterricht fast nie angesprochen. Das ist schade, denn beispielsweise kann die Bedeutung von unendlich durchaus Interesse wecken und zum besseren Verständnis von Grenzwertprozessen beitragen. Große Sprachmodelle der künstlichen Intelligenz können nicht nur Raum schaffen für philosophische Reflektionen, sie können auch Anlass dafür bieten, wie hier anhand von „Bedeutungen“ diskutiert werden soll. Statt nur über menschliches Lernen nachzudenken und psychologische Erkenntnisse einzubeziehen, kann man anhand von großen Sprachmodellen das Lernen von Wortbedeutung mathematisch untersuchen und konkret experimentieren.

Eine klassische Frage der Philosophie ist, woher die Wörter ihre Bedeutung beziehen. Wieso bedeuten „drei“ und „blau“ was sie bedeuten, so dass wir über drei blaue Dinge mit anderen Menschen reden können und diese das gleiche darunter verstehen? Es sollte nicht verwundern, dass es darauf eine Unzahl von Antworten gibt, aber für eine erste Orientierung und um beispielsweise die Diskussion mit Schülerinnen und Schülern zu vereinfachen, bietet es sich an, zwei prototypische und einflussreiche Theorien gegenüberzustellen:

- Nach der Theorie der direkten Referenz, die u.a. von John Stuart Mill vertreten wurde, sind Wörter Referenzen auf gewisse kognitive Objekte, die Bedeutungen. Beispielsweise verweisen die Wörter fünf, five, cinque und die Symbole 5 und V alle auf das gleiche Bedeutungsobjekt, nämlich die Zahl 5. Für den Platonismus existieren diese mentalen Objekte unabhängig von uns Menschen, aber die Referenztheorie der Bedeutung ist auch möglich, wenn man davon ausgeht, dass jeder Mensch die Bedeutungsobjekte selbst im Gehirn konstruiert, sofern durch die Kommunikation mit anderen hinreichend sicher ist, dass

die Bedeutungsobjekte einander entsprechen, also isomorph sind. In dieser Theorie ist Bedeutung also eine Zuordnung von der Menge der Wörter in die Menge der Bedeutungen, und zwei Wörter sind synonym, wenn die Bilder unter dieser Abbildung übereinstimmen. Problematisch für diese Theorie sind aber Wörter mit unscharfer Bedeutung oder die Erklärung von Prozessen, bei denen sich die Bedeutung im Laufe der Zeit ändert.

- Nach der Sprachspieltheorie, die im Wesentlichen auf Ludwig Wittgenstein zurückgeht, liegt die Bedeutung von Wörtern in der Art, wie sie verwendet werden, also wie sie in gesprochener und geschriebener Sprache auftreten. Bedeutung erfordert also keine Referenz auf irgendetwas außerhalb der Sprache, sondern Bedeutungen kommen allein aus der Sprache und der Anordnung der Wörter in den Texten. Für diese Theorie ist es ganz selbstverständlich, dass sich die Bedeutung von Wörtern ändert, wenn sich der Sprachgebrauch ändert.

Beide Theorien haben didaktische Konsequenzen und deswegen kennen die Schülerinnen und Schüler didaktische Konsequenzen aus dem Fremdsprachenunterricht: Nach der Theorie der direkten Referenz liegt es nahe, die Bedeutung eines Wortes der Fremdsprache durch die Übersetzung zu erklären und ein bedeutungsgleiches deutsches Wort zu nennen. Nach der Sprachspieltheorie ist es dagegen sinnvoll, mehrere Beispiele der Wortverwendung in der Fremdsprache zu geben und die Lernenden daraus die Bedeutung erschließen zu lassen.

***Große Sprachmodelle***

Inwieweit kann die Analyse von großen Sprachmodellen helfen, diese Theorien der Bedeutung zu analysieren und am Ende eventuell sogar die philosophische Streitfrage zu entscheiden? Um das zu beantworten, benötigt man einige elementare Informationen über die Arbeit und das Trainieren von großen Sprachmodellen: Der Trainingsprozess anhand von großen Textmengen erfolgt mehrstufig, ebenso wie die Verarbeitung der Texte. Wesentliche Schritte sind dabei:

- Texte werden durch Folge von sog. Tokens dargestellt, das sind natürliche Zahlen, die ein kurzes Wort oder ein Wortteil darstellen. Einzelne Buchstaben sind willkürliche Symbole: Das „r“ in „rot“ hat nichts mit dem „r“ in „grün“ zu tun. Deswegen sind Buchstaben keine gute Einheit, um Texte inhaltlich zu analysieren. Wörter dagegen sind schon ziemlich große Brocken und es gibt Sprachen mit mehr als 100.000 Wörtern. Auch dürfte es sinnvoll sein, Wörter wie „Haustür“ in ihre Bestandteile aufzulösen. Genau das macht der Prozess des „Tokenizing“: Ein Text wird in eine endliche Folge von natürlichen Zahlen übersetzt. Mathematisch gesehen ist Tokenizing eine injektive Funktion von Texten nach Tupeln von natürlichen Zahlen und das Dekodieren ist die Umkehrfunktion dazu.
- Obwohl Tokenfolgen Text 1:1 abbilden, sind sie keine gute Grundlage für maschinelles Lernen, weil man schlecht sagen kann, ob sich zwei Tokens ähnlich sind. Sind sich 319 und 417 ähnlicher als 499 und 172? Man könnte den Abstand berechnen, aber Wortbestandteile haben viele Eigenschaften und eine natürliche Zahl enthält wenig Information. Deswegen ordnet man jedem Token einen Vektor eines hochdimensionalen Vektorraums $\mathbb{R}^n, n \gg 100$ zu. Aus der endlichen Folge natürlicher Zahlen (Tokens) wird dadurch eine Folge von Vektoren, die Embedding-Vektoren. Welche Vektoren das genau sind, wird in einem ersten Trainingsschritt gelernt und es gibt eine Vielzahl unterschiedlicher Embedding-Modelle. Solche Vektoren werden vom Modell verarbeitet und alle internen Berechnungen von großen Sprachmodellen arbeiten mit Folgen dieser Embedding-Vektoren. Erst am Ende des „Denkprozesses“ werden die Vektoren wieder in Tokens zurückverwandelt. Dabei muss die

Ähnlichkeit von Vektoren bewertet werden. Dazu gibt es viele verschiedene Maße (siehe auch Oldenburg, 2021), u.a. den Euklidischen Abstand, Skalarprodukte und die sogenannte Cosinus-Ähnlichkeit $\frac{u \cdot v}{|u| \cdot |v|}$. Geometrisch betrachtet gelten zwei Vektoren als ähnlich, wenn sie in eine ähnliche Richtung zeigen. Kasten 1 zeigt einen einfachen Weg auf, etwas mit diesen Vektoren zu rechnen.

- Positional encoding: zu den Vektoren wird ein weiterer Vektor addiert, der sich aus der Position eines Tokens in der Eingabefolge ergibt. Dadurch kann auch bei der Betrachtung eines einzelnen Tokens noch gesagt werden, wo es in der Eingabe steht.
- Das Training besteht im Wesentlichen darin, dass das System lernt, aus einem künstlich abgeschnittenen Trainingstext das nächste im echten Text folgende Token vorherzusagen. Die Erzeugung von Antworttexten der großen Sprachmodell funktioniert so, dass sukzessive Token für Token erzeugt wird. Das erste Token resultiert nur aus dem Eingabetext (evtl. inkl. weiteren Kontexts). Danach wird die bereits erzeugte Token-Folge an die neue Eingabe angehängt und daraus wiederum das nächste Token berechnet.
- Der Query-Key-Value-Transformer-Mechanismus ordnet jedem Embedding-Vektor der Eingabefolge durch lineare Projektionen drei Vektoren zu, den Query-, den Key- und den Value-Vektor. Für jeden Embedding-Vektor der Eingabefolge wird die Ähnlichkeit eines Query-Vektors mit den Key-Vektoren aller anderen Embedding-Vektoren bestimmt und diese bestimmen die Gewichte, mit der die Value-Vektoren in einer gewichteten Summe addiert werden.

In diesem Beitrag stehen die Embedding-Vektoren im Fokus.

***Wettkampf der Bedeutungstheorien***

Offensichtlich können große Sprachmodell erfolgreich Sprache lernen, zwischen Sprachen übersetzen und dabei auch die Bedeutung von Wörtern zumindest in gewissem Sinne verstehen. Kann man daraus Argumente für oder gegen die beiden genannten Bedeutungstheorien ableiten? Auf dem ersten Blick könnte man denken, dass die Sprachspieltheorie gestützt wird, weil die großen Sprachmodelle einfach nur „beobachten“, wie Sprache in den Trainingstexten verwendet wird. Zudem könnte man argumentieren, dass die Sprachmodelle keinerlei Referenz zum Beispiel auf physikalische Objekte haben können. Die Bedeutung von „Eiffelturm“ etwa kann sich für ein Sprachmodell immer nur aus den Texten, die beim Training verwendet wurden, erschließen, aber es hat keine Sensoren um das Metallbauwerk in der französischen Hauptstadt als physisches Objekt zu erfassen. Die Bedeutung von Eiffelturm kann für ein großes Sprachmodell keine Referenz auf dieses Objekt sein. Also 1:0 für die Sprachspieltheorie. Allerdings würde man vom Standpunkt der Sprachspieltheorie davon ausgehen, dass die Systeme am besten direkt mit solchen Folgen von Tokens lernen. Wie im technischen Überblick dargestellt, passiert das aber nicht, sondern den Token werden Embedding-Vektoren zugeordnet und für diese kann man Ähnlichkeiten bestimmen. Man kann also sagen, dass diese Embedding-Vektoren die Bedeutungen im Sinne der Referenztheorie sind. 1:1. Allerdings stellt man fest, dass die Frage der Bedeutungsgleichheit diffizil ist. Im Modell `all-MiniLM_L6-v2` beispielsweise, ist die Cosinus-Ähnlichkeit von „5“ und „five“ 0.91 nicht etwa 1, und die von „5“ und „3“ beträgt immerhin 0.72 – wohl weil beides Zahlen sind, nicht weil 3 und 5 als Zahlen sehr ähnlich sind. Die einfache Behauptung der Referenztheorie, man könne Bedeutungsgleichheit dadurch prüfen, ob die Bedeutungsobjekte gleich sind, greift also zu kurz, man muss notgedrungen zu Ähnlichkeitsmaßen greifen (welche das sind, erläutert Kasten 3). Immerhin in diesem abgemildert Sinne von Ähnlichkeit kann man also Bedeutungsgleichheit prüfen. Insofern: Beide Theorien haben einen 11-Meter verschossen, es bleibt beim 1:1.

Trotzdem liegt die Idee nahe, dass die Embedding-Vektoren Bedeutungen sind, auf die die Token referieren. Nun sind je nach Modell die Embedding-Vektoren Elemente eines $n$-dimensionalen Raums, wobei $n$ von einigen 100 bis einigen 1000 geht. Für Schülerinnen und Schüler, die meistens im $\mathbb{R}^3$ rechnen, mögen das gigantisch hohe Raumdimensionen sein, wenn man aber bedenkt, wie viele Bedeutungen und Bedeutungsnuancen es gibt, erscheint auch das wenig. Andererseits bieten Vektoren durch die reellen Einträge die Möglichkeiten, die Stärke von Bedeutungsanteilen zu messen und durch Linearkombinationen Bedeutungen zu kombinieren. Ein mittlerweile klassisches Beispiel stammt von Mikolov, Yih & Zweig (2013) und kann mit der App aus Kasten 1 nachvollzogen werden: Zieht man von „king" den „man" ab und addiert „woman", also formal also `q:=king-man+woman`, erhält man einen Vektor, der recht nahe bei „queen" liegt: In der App liefert etwa `sp(q,queen)` einen Wert von immerhin 0.85. Dies spricht dafür, dass die Embedding-Vektoren tatsächlich Bedeutung tragen – aber bevor der Treffer für die Referenztheorie gewertet wird, sollte er vom Schiedsrichter überprüft werden, denn es ist unwahrscheinlich, dass man beispielsweise sagen kann, dass bei Embedding-Vektoren z.B. die Komponente mit Index 417 die „Rotheit" (also die Eigenschaft, rot zu sein, die „Röte") misst. Die Basisvektoren dieses Vektorraum sind, wenn man so will, nicht direkt menschliche Bedeutungen, sondern es sind konstruktivistisch gesprochen, die LLM-Konstruktionen aus dem Lernprozess. Deswegen sind auch Ähnlichkeiten zwischen Embedding-Vektoren so relativ schwer zu beurteilen. Würde man erwarten, dass die Embedding-Vektoren von schwarz und weiß sehr verschieden sind, weil die Farben so gegensätzlich sind, oder dass sie sehr ähnlich, weil beides Farben sind? Oder sind Embeddings aus menschlicher Perspektive gänzlich unverständlich? Ein Experiment dazu kann Klarheit bringen: Für eine größere Zahl von kurzen Adjektiven, die verschiedene Bereiche abdecken, z.B. Farben und Wahrheitswerte, bestimmt man eine niederdimensionale Basis, aus denen sich die Embeddings bis auf einen möglichst kleinen Fehler linear kombinieren lassen. Diese Hauptkomponentenanalyse (PCA, Kasten 2) ist eine verbreitete Technik der Datenanalyse und Dimensionsreduktion. Das Ergebnis ist (Kasten 4), dass in den Embedding-Vektoren durchaus menschlich interpretierbar Dimensionen enthalten sind. Insofern ist die Sichtweise berechtigt, dass Embedding-Vektoren die Bedeutung kodieren. 2:1 für die Referenztheorie.

In der Struktur der so erfolgreich großen KI-Sprachmodelle nimmt die Transformer Architektur eine wesentliche Rolle ein. Darin gibt es einen Query-Key-Value-Mechanismus, der wie das Nachschlagen in einem intelligenten Register funktioniert: Jedes Wort erzeugt eine Anfrage (Query), die mit den Schlüsseln (Keys) aller anderen Wörter verglichen wird. Passt ein Schlüssel gut zur Anfrage, wird der dazugehörige Inhalt (Value) stärker berücksichtigt. So kann das Modell gezielt „auf" relevante Stellen im Text verweisen – ähnlich wie bei einer Referenz, die sagt: Schau genau dort nach, das ist hier wichtig. Das ist also so etwas wie eine Referenz im Kurzzeitgedächtnis (metaphorisch gesprochen). Auch das spricht also für die Bedeutung von Referenzen – wie genau ist allerdings weniger klar. Insofern: Lattentreffer.

Für Menschen ergibt sich die Bedeutung von Wörtern allerdings auch sehr stark durch Referenzen in die physikalische Welt: Viele Wörter bezeichnen physikalische Objekte oder Qualitäten von ihnen. Diese Erfahrungen sind großen Sprachmodellen gänzlich unzugänglich, sie sind, konstruktivistisch gesprochen, referentiell abgeschlossen. Insofern fehlt Ihnen ein wesentliches Fundament der menschlichen Bedeutungsgenese, denn ihre Referenzen sind Referenzen innerhalb der Sprache. 2:2. Aber: Der Treffer zählt nur, wenn diese Referenzen innerhalb der Sprache ausreichend sind, um Bedeutung voll zu erfassen. Es gibt allerdings seriöse Gegenpositionen, etwa Bender & Koller (2020). Ob das zur Aberkennung des Treffers führt, bleibt offen – so ist das in der Philosophie.

***Fazit***

Computer erlauben die Anwendung von Analyseverfahren, die geeignet sind, die Bedeutungskonstruktion von großen Sprachmodellen zu analysieren. Damit eröffnen sich dem Mathematikunterricht neue Dimensionen. Wichtig scheint die Bereitschaft, Verfahren auch als Blackbox zu benutzen, andererseits aber auch die Arbeitsweise der Verfahren zu erhellen. In diesem Beitrag wurde beides versucht und

**Literatur**


Bender, E. M., Koller, A. (2020). Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 5185–5198. Association for Computational Linguistics. https://aclanthology.org/2020.acl-main.463

Mikolov, T., Yih, W.T. , Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations. NAACL HLT. https://aclanthology.org/N13-1090.pdf

Oldenburg, R. (2011, 2026). *Mathematische Algorithmen im Unterricht*. 1. Aufl. Vieweg+Teubner, 2. Aufl. Springer.

Oldenburg, R. (2021). Big data – small school: oder: Likert-Skalen, Kaufempfehlungen, soziale Netzwerke, das Skalarprodukt und all das. In: H. Humenberger und B. Schuppar: *Neue Materialien für einen realitätsbezogenen Mathematikunterricht* 7, 137-142.

Schönbrodt, S., & Oldenburg, R. (2024). Mathematische Grundlagen als Schlüssel zu einem allgemeinbildenden Verständnis von KI: theoretische Perspektiven und praktische Unterrichtsideen. *Mathematik im Unterricht* 15, 17-36.


***Kasten 1: Embeddings***

Embeddingvektoren verschiedener Modelle lassen sich relativ leicht aus Programmiersprachen heraus berechnen, aber hinter „relativ leicht“ steckt dabei durchaus einiges an technischem Aufwand, den man in allgemeinbildenden Schulen nicht allen Schülerinnen und Schülern aufbürden kann. Deswegen gibt es auf der Seite https://myweb.rz.uni-augsburg.de/~oldenbre/EmbedVect.html die Möglichkeit, mit einigen Embedding-Modellen zu spielen. Der Nutzer muss als erstes ein Modell auswählen und die Menge der Wörter, für die Embedding-Vektoren bestimmt werden sollen, eingeben. Es ist bereits eine große Vielzahl kurzer englischer Wörter vorgegeben, aber man kann diese Liste beliebig verändern. Durch den Button „Embeddings berechnen“ Werden die Embedding-Vektoren über frei verfügbare Server abgerufen und als erste Analyse wird die Matrix ihrer Kosinus-Ähnlichkeiten ausgegeben. Diese kann man schon recht gut interpretieren: Im Modell `all-MiniLM_L6-v2` haben etwa „red“ und „blue“ eine Kosinus-Ähnlichkeit von 0.72, während „red“ und „true“ nur auf eine Ähnlichkeit von 0.17 kommen.

Darunter gibt es einen „Vektor-Interpreter“ mit dem man einfach Rechnungen durchführen kann, etwa Mittelwerte bilden `(man+woman)/2` oder Differenzen `white-black`. Die Ergebnisse kann man mit `:=` auch neuen Variablen zuordnen. `norm` berechnet die Norm und `sp` das Skalarprodukt.

***Kasten 2:Hauptkomponentenanalyse***

Die Hauptkomponentenanalyse ist ein wichtiges Verfahren der Datenanalyse. Im Folgenden wird die Grundidee beschrieben. Da in vielen Bundesländern zunehmend Python in der Schule unterrichtet wird, gibt es unter https://roldenburg.github.io/PCA-Demo/lab/index.html die Möglichkeit, den im folgenden beschriebenen Code auszuprobieren (dazu dort PCA.ipynb auswählen). Es handelt sich

dabei um ein im Browser laufendes Jupyter-Notebook. Das ist ein System, das es ermöglicht, ohne lokale Installation von Software Python-Programme im Browser auszuführen. Sie können den Code in den Zellen verändern und mit Shift-Enter ausführen.

Angenommen, man misst in einer Anzahl $n$ von Fällen zwei Werte, z.B. die Punkte in den Aufgaben zur Bruchaddition und zur Bruchmultiplikation, dann könnten die Daten in einem zweidimensionalen Streudiagramm etwa aussehen wie die in Abb. 1 dargestellten simulierte Daten. Offensichtlich gibt es in diesen Daten einen Zusammenhang zwischen den beiden gemessenen Größen. Es gibt verschiedene Methoden, diesen Zusammenhang quantitativ zu erfassen, etwa Korrelation oder Regressionsrechnung. Die Korrelation gibt allerdings nur ein Maß für die Enge des linearen Zusammenhangs an, die Regression erfordert, dass man eine der Variablen als abhängige und die andere als unabhängige deklariert. Wenn man Daten hat, über deren Struktur man noch nicht genauer informiert ist, verwendet man daher Methoden, die solche Annahmen nicht erfordern. Eine der Wichtigsten ist die Hauptkomponentenanalyse (principle component analysis, PCA), die hier vorgestellt werden soll. Im Fall von Daten aus Abb. 1 liefert die Hauptkomponentenanalyse zunächst die Hauptrichtung als Vektor, also einen Richtungsvektor für die sogenannte Hauptkomponente. Ein großer Vorteil dieses Verfahrens ist die Flexibilität: Es lässt sich unabhängig von der Dimension der Daten anwenden.

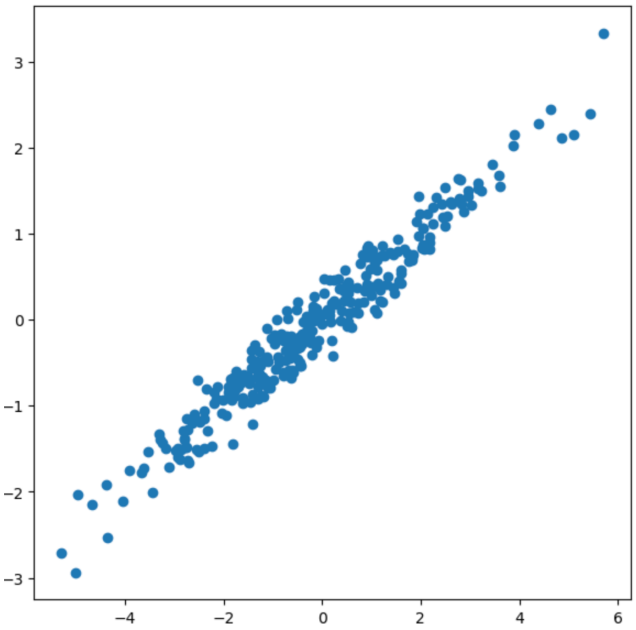


Abb. 1 Simulierte Datenpunkte aus dem $\mathbb{R}^2$, die eine klare Hauptkomponente als Richtung der Punktwolke zeigen

Wie berechnet man diese Hauptrichtung? Etwas Formalisierung ist nötig: Die Ausgangsdaten seien $n$ Vektoren $x_i \in \mathbb{R}^d, i = 1..n$ (im obigen Beispiel ist $d = 2$), die man auch zu einer $n \times d$ Datenmatrix $X$ zusammenfassen kann. Die weitere Analyse ist leichter, wenn die Daten zentriert sind, d.h. dass der Schwerpunkt der Punkte der Null-Vektor ist (wie in Abb. 1). Falls das noch nicht erfüllt ist, bildet man die Mittelwerte $m_j := \frac{1}{n}\sum_{i=1}^{n} X_{i,j}, j = 1..d$ und subtrahiert diese, d.h. man ersetzt $X$ durch die Matrix$\left(X_{i,j} - m_j\right)_{i,j}$.

Als nächstes soll ein Einheitsvektor $u \in \mathbb{R}^d, \|u\| = 1$ bestimmt werden, der die Hauptrichtung angibt. Eine gute Wahl für $u$ ist eine, bei der viele $x_i$ in eine ähnliche Richtung zeigen. Das lässt sich so formalisieren: $u(u \cdot x_i)$ ist ein Vektor in Richtung von $u$, er ist die Projektion von $x_i$ auf $u$, und entsprechend ist $d_i(u) := x_i - u(u \cdot x_i)$ ein Vektor, der den Anteil von $x_i$ darstellt, der nicht in

Richtung $u$ geht. Man sollte $u$ so wählen, dass diese Differenzen möglichst klein sind, d.h., die Zielfunktion $F(u) \coloneqq \frac{1}{n}\sum_{i=1}^{n}\left||d_i(u)|\right|^2$ ist unter der Nebenbedingung $\|u\| = 1$ zu minimieren ($F$ ist der mean square error). Der so gefundene Vektor ist die Hauptkomponente. Das Verfahren einer solchen Optimierung ist leicht zu verstehen: Es wird ausgehend von einem Startvektor sukzessive nach einem Vektor gesucht, der die Bedingung erfüllt und den Funktionswert möglichst klein macht. Die genaue Berechnungsmethode ist unwichtig und kann in einer black-box verbleiben. Eine direkte Umsetzung des eben beschrieben Verfahrens zeigt die Python-Funktion `pca1d` im Jupyter-Notebook `PCA.ipynb` (Link s.o.).

Was bedeuten die Komponenten des Vektors $u$? Die Minimierung erfolgt so, dass die quadrierten Fehler $x_i - u(u \cdot x_i)$ möglichst klein werden. Das bedeutet $x_i = (u \cdot x_i)u + \epsilon$ mit einem Fehler $\epsilon$ (Erwartungswert 0), der möglichst klein ist. Ein hoher Wert in Komponente $u_j$ bedeutet also, einen hohen Beitrag der Hauptkomponente zur Erklärung der Variable $j$ der Daten. Man nennt daher die $u_j$ auch Ladungen der Hauptkomponente auf die Variablen.

Bisher wurde eine Dimension extrahiert, dieser Vorgang wird nun iteriert: Stellen Sie sich Datenpunkte aus dem $\mathbb{R}^3$ vor, die bis auf einige zufällige Abweichungen auf einer Ebene liegen. In diesem Fall sollte man nicht einen Hauptkomponentenvektor, sondern zwei Hauptkomponenten-Vektoren berechnen. Im Allgemeinen gibt man sich eine Zahl $k$ der Komponenten vor und startet damit, den ersten Vektor $u_1$ gemäß obigem Rezept zu berechnen. Um die zweite Dimension zu finden, sucht man in den noch nicht durch den ersten Vektor aufgespannten Richtungen (man sagt dazu auch: in der noch nicht durch $u_1$ erklärten Varianz) nach einem zweiten, d.h. die neuen Daten sind $y_i \coloneqq x_i - u(u \cdot x_i)$. Mit diesen an Stelle der ursprünglichen $x_i$ führt man wieder eine eindimensionale PCA durch und liefert die nächste Hauptkomponente $u_2$. Das Verfahren wird iteriert bis man wie gewünscht $k$ Komponenten extrahiert hat. Ergebnis ist eine Liste von $k$ orthogonalen Einheitsvektoren $u_1, \dots, u_k \in \mathbb{R}^d$. Die obige Gleichung $x_i = (u \cdot x_i)u + \epsilon$ wird damit verallgemeinert zu $x_i = (u_1 \cdot x_i)u_1 + \cdots + (u_k \cdot x_i)u_k + \epsilon$, d.h. man stellt eine möglichst gute Linearkombination der Ausgangsdaten als Linearkombination $k$ gut gewählten Basisvektoren her.

Eine letzte Frage könnte sein, wie viele Hauptkomponenten man extrahieren sollte. Wenn man keine theoretische Begründung für eine bestimmte Zahl hat, kann man sich anschauen, wie die Größe des jeweils verbleibenden Fehlers $\epsilon$ kleiner wird. Die Programme in `PCA.ipynb` berechnen auch RMSE, den root square mean error. Damit kann man in einer Schleife $k$ sukzessive erhöhen und prüfen, ab wann der verbleibende Fehler klein ist.

***Kasten 3:Analyse von Ähnlichkeitsmaßen***

Aus dem Repertoire der Schulmathematik, bzw. schulnahen Mathematik gibt es vier Maße, mit denen man die Ähnlichkeit von Vektoren $a, b \in \mathbb{R}^n$ beurteilen kann:

- Der euklidische Abstand: $\left||a - b|\right| = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$
- Das Skalarprodukt: $a \cdot b = \sum_{i=1}^{n} a_i b_i$
- Der Winkel bzw der Cosinus des Winkels: $\cos(\alpha) = \frac{a \cdot b}{||a|| \cdot ||b||}$ („Kosinusähnlichkeit“)
- Die Korrelation: $\mathrm{cor}(a, b) = \frac{\sum_{i=1}^{n}(a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_{i=1}^{n}(a_i - \bar{a})^2 \sum_{i=1}^{n}(a_i - \bar{b})^2}}$, $\bar{a} \coloneqq \frac{1}{n}\sum_{i=1}^{n} a_i$

Es ist eine schöne Aufgabe für Lernende sich zu überlegen, welche Eigenschaften diese Ähnlichkeitsmaße haben, etwa, dass die Korrelation sich nicht ändert, wenn man einen Vektor mit einem Skalar multipliziert – es zählt also nur seine Richtung. Das gleiche gilt auch für den Cosinus des Winkels. Die meisten Embedding-Modelle liefern (zumindest ungefähr) zentrierte ($\bar{a} = 0$) und

normierte ($||a|| = 1$) Vektoren. Damit fallen Skalarprodukt, Kosinus-Ähnlichkeit und Korrelation zusammen. Der euklidische Abstand verhält sich aber anders, schon weil bei ihm kleine Werte für Ähnlichkeit sprechen, bei den anderen aber große. Außerdem liefert er nie negative Werte. Die Zahlenwerte sind also nicht direkt vergleichbar. Könnte es also sein, dass man die Ähnlichkeit von Vektoren zueinander anders einschätzt? Welches Maß sollte man nehmen? Unter https://developers.openai.com/api/docs/guides/embeddings kann man dazu nachlesen:

`We recommend cosine similarity. The choice of distance function typically doesn't matter much. OpenAI embeddings are normalized to length 1, which means that:`

- `Cosine similarity can be computed slightly faster using just a dot product`
- `Cosine similarity and Euclidean distance will result in the identical rankings`

Dass die letzte Aussage stimmt, ist eine schöne Übungsaufgabe: Angenommen, $a, b, c \in \mathbb{R}^n$ sind Einheitsvektoren und mit dem Abstand gemessen ist $a$ ähnlicher zu $b$ als zu $c$, also $||a - b|| < ||a - c||$. Könnte es sein, dass man mit dem Skalarprodukt $a$ für ähnlicher zu $c$ als zu $b$ hält? Quadrieren liefert: $\sum_{i=1}^{n}(a_i - b_i)^2 < \sum_{i=1}^{n}(a_i - c_i)^2$, also $\sum_{i=1}^{n} a_i^2 + b_i^2 - 2a_ib_i < \sum_{i=1}^{n} a_i^2 + c_i^2 - 2a_ic_i$. Führt man die Summen auf den einzelnen Summanden aus, sieht man, dass jeweils die ersten beiden Summen die Norm ergeben und weil die Vektoren alle normiert sind, hat man also $1 + 1 - 2\sum_{i=1}^{n} a_ib_i < 1 + 1 - 2\sum_{i=1}^{n} a_ic_i$, also ist $\sum_{i=1}^{n} a_ib_i > \sum_{i=1}^{n} a_ic_i$, d.h. auch mit dem Skalarprodukt findet man die gleiche Ähnlichkeitsbeziehung. Die Rechnung kann rückwärts gelesen werden, so dass man tatsächlich eine Äquivalenz nachgewiesen hat.

***Kasten 4: Analysen mit PCA und Clustering***

Die App aus Kasten 1 bestimmt die Embedding-Vektoren von vielen kurzen Wörtern. Die Wörter, die als abänderbarer Vorschlag gesetzt sind, gruppieren sich in verschiedene Gruppen, u.a. Farben, Größen, Geschmacksrichtungen. Spiegeln sich diese Gruppen menschlicher Bedeutung in den zugeordneten Vektoren? Um das zu analysieren, gibt es die Methode der Clusteranalyse: Man versucht eine endliche Menge von Vektoren in Teilmengen (Cluster) aufzuteilen, so dass die Vektoren in einem Cluster untereinander ähnlich sind, zwischen den Clustern aber unterschiedlich. Ein einfacher Clustering-Algorithmus wird in Oldenburg (2011, 2026) erklärt und eine online-erprobbare Implementation findet man auf https://roldenburg.github.io/PCA-Demo/lab/index.html in der Datei Cluster.ipynb. Die App aus Kasten 1 berechnet auch solche Clusterungen und man sieht leicht, dass sich problemlos z.B. Farben zu Farben ordnen und einen Cluster bilden. Um die gefundenen Cluster noch besser interpretieren zu können, werden die Datenvektoren aus den beiden ersten Hauptkomponenten linear kombiniert und dann in einem zweidimensionalen Koordinatensystem dargestellt, siehe Abb. 2. Damit kann man auch die Hauptkomponentenvektoren inhaltlich interpretieren: Die erste, waagrecht aufgetragene Dimension ist die Farbigkeit, die zweite, vertikal aufgetragene, dagegen könnte als Abstraktheit interpretiert werden: Weiter oben sind eher abstrakte Konzepte.


Abb. 2 Die Wörter in 7 Cluster eingeteilt (Farben) und gemäß den ersten beiden Hauptkomponenten aufgetragen.