Programmierkurs Python II

Stefan Thater & Michaela Regneri
Universität des Saarlandes
FR 4.7 Allgemeine Linguistik (Computerlinguistik)

Übersicht

- Vektoren elementar
- Information Retrieval
- Semantische Ähnlichkeit
- Klassifikation (kNN)
- Cluster-Analyse (k-means)

Vektoren

- Vektoren entsprechen Listen reeller Zahlen:
  $\mathbf{v} = (v_1, v_2, \ldots, v_n)$
- Vektoren können addiert werden:
  $\mathbf{v} + \mathbf{u} = (v_1 + u_1, \ldots, v_n + u_n)$
- Vektoren können mit reellen Zahlen („Skalaren“) multipliziert werden:
  $a\mathbf{u} = (au_1, \ldots, au_n)$

Vektorräume

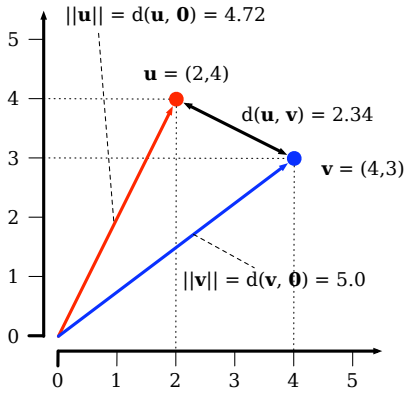
- Ein Vektorraum über $\mathbb{R}$ ist eine Menge von Vektoren, über denen Addition und Skalarmultiplikation definiert sind
- die Additionsoperation muss folgende Bedingungen erfüllen:
  - $(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$
  - $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$
  - $\mathbf{v} + \mathbf{0} = \mathbf{v}$
  - $\forall \mathbf{u} \exists \mathbf{v}\ \mathbf{u} + \mathbf{v} = \mathbf{0}$

Vektorräume

- Ein Vektorraum über $\mathbb{R}$ ist eine Menge von Vektoren mit Operationen Addition und Skalarmultiplikation.
- die Skalarmultiplikation muss folgende Bedingungen erfüllen:
  - $a(\mathbf{v} + \mathbf{w}) = a\mathbf{v} + a\mathbf{w}$
  - $(a + b)\mathbf{v} = a\mathbf{v} + b\mathbf{v}$
  - $a(b\mathbf{v}) = (ab)\mathbf{v}$
  - $1\mathbf{v} = \mathbf{v}$

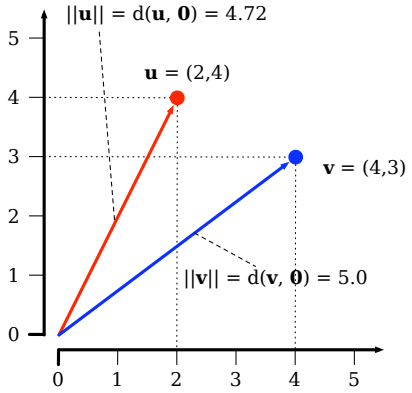
Geometrische Interpretation

- Vektoren beschreiben Punkte im Raum
- Euklidische Distanz:
  $d_2(\mathbf{u}, \mathbf{v}) = \sqrt{(u_1 - v_1)^2 + \ldots + (u_n - v_n)^2}$
- Distanz $d_2(\mathbf{u}, \mathbf{v})$ = Länge $||\mathbf{u} - \mathbf{v}||$ des Vektors $\mathbf{u} - \mathbf{v}$
  $||\mathbf{u}|| = d_2(\mathbf{u}, \mathbf{0})$



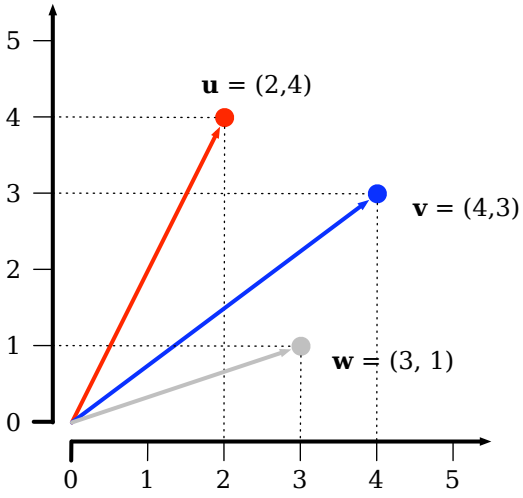
Geometrische Interpretation

- Vektoren beschreiben Punkte im Raum
- Skalarprodukt (dot product, inner product):
  $\mathbf{u} \bullet \mathbf{v} = u_1v_1 + \ldots + u_nv_n$
- Winkel φ zwischen Vektoren **u** und **v**
  $\cos \varphi = \mathbf{u} \bullet \mathbf{v} \,/\, ||\mathbf{u}|| \bullet ||\mathbf{v}||$
- φ als Distanzmaß
  (Länge der Vektoren wird nicht berücksichtigt)



Geometrische Interpretation

- Vektoren beschreiben Punkte im Raum
- Centroid von Vektoren $\mathbf{v_1}, \ldots, \mathbf{v_k}$
  $C = (\mathbf{v_1} + \ldots + \mathbf{v_k}) / k$



Vektoren in Python

```
class Vector:
    def __init__(self, data):
        self.data = [0] * data if isinstance(data, int) else data
    def __setitem__(self, idx, value):
        self.data[idx] = value
    def __getitem__(self, idx):
        return self.data[idx]
    def __add__(self, other):
        return Vector([x + y for (x, y) in zip(self.data, other.data)])
    def __mul__(self, scalar):
        return Vector([x * scalar for x in self.data])
    def __repr__(self):
        return 'Vector({0:})'.format(self.data)
    ...
```


Vektoren in Python

```
from math import sqrt, pow
class Vector:
    ...
    def dotproduct(self, other):
        return sum(x * y for (x, y) in zip(self.data, other.data))
    def norm(self):
        return sqrt(sum(pow(x, 2) for x in self.data))
    def cos(self, other):
        return self.dotproduct(other) / (self.norm() * other.norm())
    def distance(self, other):
        return sqrt(sum(pow(x - y, 2)
            for (x, y) in zip(self.data, other.data)))
    @staticmethod
    def centroid(vecs):
        return sum(vecs[1:], vecs[0]) * (1.0 / len(vecs))
```

Information Retrieval

- Typischerweise liefert ein Information Retrieval System viele Dokumente für eine Anfrage ⇒ beste Treffer zuerst
- Fasse Dokumente und Anfrage als Vektoren auf
  - Alle Wörter in Dokumentsammlung: $w_1, \ldots, w_n$
  - Vektor für Dokument d: $\mathbf{v_d} = (v_1, \ldots, v_n)$
    $= w_i$ kommt v_i mal in Dokument d vor (bzw. tf-idf o.ä.)
- Sortiere die Dokumente nach ihrer Ähnlichkeit bzw. Distanz zur Anfrage (Ähnlichkeit ~ Winkel zwischen Vektoren)

(Beispiel aus FSLT 2008/09, Pinkal)

Term–Dokument Matrix

	D_1	D_2	D_3	…
dolphin	3	2	0	…
human	1	0	3	…
language	0	0	5	…
like	1	1	0	…
mammal	1	1	0	…
technology	0	0	1	…
warm-blooded	1	1	0	…
whale	0	3	0	…

$\cos(D_1, D_2) = 0.62$

$\cos(D_1, D_3) = 0.14$

$\cos(D_2, D_3) = 0.0$

Term–Dokument Matrix

- Are dolphins mammals?

	D_1	D_2	D_3	Q
dolphin	3	2	0	1
human	1	0	3	0
language	0	0	5	0
like	1	1	0	0
mammal	1	1	0	1
technology	0	0	1	0
warm-blooded	1	1	0	0
whale	0	3	0	0

$\cos(D_1, Q) = 0.78$

$\cos(D_2, Q) = 0.53$

$\cos(D_3, Q) = 0.0$

Semantische Ähnlichkeit

- Distributionelle Hypothese: Wörter, die in ähnlichen Kontexten vorkommen, sind semantisch ähnlich
- eine Wort-Wort Matrix kodiert, wie häufig zwei Wörter im gleichen Kontext zusammen vorkommen.
- Kontexte können Dokumente, Abschnitte, Kontext-Fenster, … sein, können ggf. die syntaktische Struktur berücksichtigen, …

(Beispiel aus FSLT 2008/09, Pinkal)

Wort-Wort Matrix

	dolp.	hum.	lang.	like	mam.	tech.	wb	whale
dolphin	5	1	0	2	0	0	2	3
human	3	4	5	1	1	1	1	0
language	0	3	5	0	1	1	0	0
like	1	1	0	2	0	0	2	3
technology	0	3	5	0	1	1	0	0
warm-blooded	5	1	0	2	0	0	2	3
whale	2	0	0	1	0	0	1	3

Vektoren als Wörterbücher

- In viele Anwendungsfällen hat ein Vektor nur wenige Komponenten ≠ 0 → Darstellung mit Listen nicht optimal
- Alternativ können Vektoren als Wörterbücher repräsentiert werden, mit 0 als Defaultwert.

Vektoren als Wörterbücher

```
class Vector(dict):
   def __getitem__(self, key):
      return dict.get(self, key, 0)
   def __add__(self, other):
      return Vector((k, self[k] + other[k])
         for k in set(self.keys() + other.keys()))
   def __mul__(self, scalar):
      return Vector(((k, v * scalar)
         for (k, v) in self.items()))
   ...
```

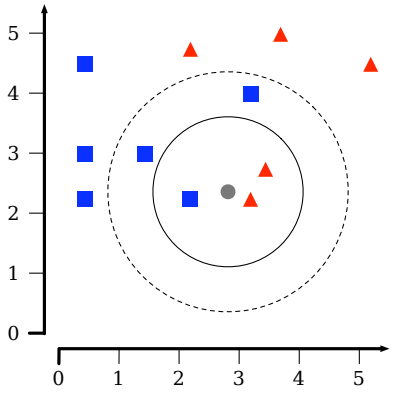

Vektoren als Wörterbücher

```
class Vector(dict):
   ...
   def dotproduct(self, other):
      return sum(self[k] * other[k]
         for k in self.iterkeys() if k in other)
   def norm(self):
      return sqrt(sum(v * v for v in self.values()))
   def cos(self, other):
      return self.dotproduct(other)/(self.norm()*other.norm())
   ...
```

(Beispiel aus Wikipedia)

k nearest neighbors (kNN)

- Der kNN Algorithmus ist ein Klassifikationsverfahren, das Objekten eine Klasse zuordnet, basierend auf den Klassen seiner k nächsten Nachbarn.
- Gegeben ist eine Menge von Objekten und deren Klassen („Trainingsdaten“)
- Berechne die Distanz zwischen dem zu klassifizierenden Objekt und allen Trainings-Objekten.
- Weise dem neuen Objekt die Klasse seiner k nächsten Nachbarn zu (Mehrheitsentscheidung).



19

WSD mit kNN

- kNN kann man auch für Word Sense Disambiguation benutzen
- Variante 1: Vektoren = (Trainings- u. Test-)Dokumente
  - wie Bayes von letzter Woche
  - klassifiziert ein _Dokument_ (nicht eine Wort-Instanz)
- Variante 2: Vektoren = Auftreten des Wortes im Kontext
  - Bestimme ein Kontextfenster von n Worten (oder dem Dokument)
  - pro Auftreten des Wortes in Frage ein Vektor im Modell
  - klassifiziere das (die) Auftreten des Wortes im Text (ggf. Mehrheitsentscheidung)

20

Clustering mit *k-means*

- Clusteranalyse: Ermittlung von Gruppen (Clustern) von Objekten bzw. Vektoren, deren beobachtbare Eigenschaften bestimmte Ähnlichkeiten aufweisen.
- k-means Algorithmus: Partitioniere eine Menge von Vektoren in k Cluster
- Zielsetzung: Minimiere den Abstand der Elemente eines Clusters zu seinem Zentrum.
- Residual sum of squares (RSS)
  - $RSS = \sum_{1 \leq k \leq K} \sum_{x \in \omega k} ||x - \mu(\omega_k)||^2$
  - ω_k = Cluster k, $\mu(\omega_k)$ = Centroid von ω_k

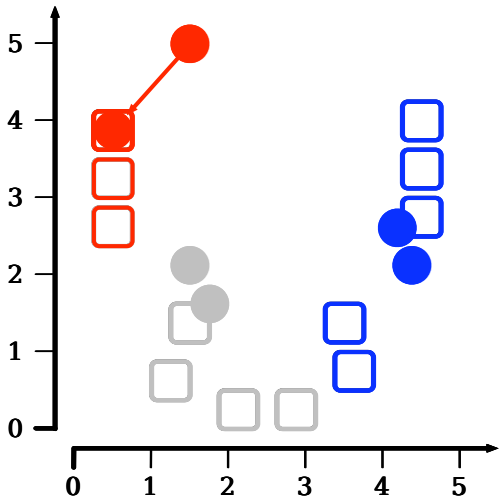
Der *k-means* Algorithmus

- Initialisierung: zufällige Auswahl von k Clusterzentren
- Iterative (Neu-) Berechnung der Clusterzentren bis Abbruchbedingung zutrifft:
  - Weise jedes Objekt (Vektor) dem ihm am nächsten liegenden Clusterzentrum zu
  - Berechne für jeden Cluster die Clusterzentren neu

(Beispiel aus Wikipedia)

Der *k-means* Algorithmus (Beispiel)

1. Zufällige Auswahl von drei Clusterzentren (●, ●, ●)
2. Zuordnung der Objekte zu Cluster mit dem nächstliegenden Zentrum.
3. Neuberechnung der Cluster-Zentren.
4. ...



(Manning, Raghavan, Schütze, 2008)

Der *k-means* Algorithmus

K-MEANS$(\{\vec{x}_1, \ldots, \vec{x}_N\}, K)$
1 $(\vec{s}_1, \vec{s}_2, \ldots, \vec{s}_K) \leftarrow$ SELECTRANDOMSEEDS$(\{\vec{x}_1, \ldots, \vec{x}_N\}, K)$
2 **for** $k \leftarrow 1$ **to** K
3 **do** $\vec{\mu}_k \leftarrow \vec{s}_k$
4 **while** stopping criterion has not been met
5 **do for** $k \leftarrow 1$ **to** K
6 **do** $\omega_k \leftarrow \{\}$
7 **for** $n \leftarrow 1$ **to** N
8 **do** $j \leftarrow \arg\min_{j'} |\vec{\mu}_{j'} - \vec{x}_n|$
9 $\omega_j \leftarrow \omega_j \cup \{\vec{x}_n\}$ *(reassignment of vectors)*
10 **for** $k \leftarrow 1$ **to** K
11 **do** $\vec{\mu}_k \leftarrow \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} \vec{x}$ *(recomputation of centroids)*
12 **return** $\{\vec{\mu}_1, \ldots, \vec{\mu}_K\}$

Mögliche Abbruchbedingungen

- Begrenzte Anzahl von Iterationsschritten
- Zuweisung von Objekten zu Clustern ändert sich zwischen zwei Iterationsschritten nicht.
- Die Centroide ändern sich zwischen zwei Iterationsschritten nicht.
- Abbruch, wenn RSS einen Schwellwert unterschreitet
- Abbruch, wenn zwischen den Iterationsschritten die Abnahme des RSS einen Schwellwert unterschreitet.

Eigenschaften

- k-means liefert für unterschiedliche initiale Clusterzentren ggf. unterschiedliche Ergebnisse
- ein Cluster kann in einem Schritt leer bleiben
- k-means findet nicht notwendigerweise die global optimale Lösung
- optimales Clustering zu finden gehört zur Komplexitäts-klasse NP