Explorative Datenanalyse: k-Means-Clustering

Grundbegriffe

Explorative Datenanalyse Das Ziel der explorativen Datenanalyse (auch: *unsupervisedun-supervised machine learning*) ist es, die Punkte in einem Datensatz in ähnliche Gruppen – *cluster* – einzuteilen.

Distanzfunktionen In der Regel basieren verschiedene Clusteringverfahren auf Distanzfunktionen. Um die verschiedenen Punkte eines Datensatzes in ähnliche Gruppen einzuteilen, benötigt man ein Maß, das die Ähnlichkeit zweier Punkte quantifiziert. Diese Kennzahlen werden mithilfe von Distanzfunktionen berechnet. Einige Distanzfunktionen beispielhaft hier:

- Hamming-Distanz: $\delta_H(x, y) = \text{count}_i(x_i \neq y_i)$,
- Euklidische Distanz: $\delta_E(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$,
- Manhatten-Distanz: $\delta_M(x, y) = \sum_i (|x_i - y_i|)$,
- Maximumsdistanz: $\delta_{Max}(x, y) = \max_i (|x_i - y_i|)$.

k-Means-Clustering

Algorithmus Eine sehr gängige Clusteringmethode (oder Clusteringalgorithmus) ist **k-Means**. Der Algorithmus verläuft wie folgt. Im ersten Schritt werden k zufällige Centroids (später: Clustermittelpunkte) gewählt. Es ist sinnvoll, die Wertebereiche der initialen Centroids auf die Wertebereiche der Daten in den jeweiligen Dimensionen zu beschränken. Dann werden alle Punkte dem jeweils nächste Centroid/Clustermittelpunkt zugeordnet; es werden also Cluster gebildet. Dann werden die neuen Clustermittelpunkte berechnet. Diese beiden Schritte werden so lange wiederholt, bis das Stopkriterium erreicht ist.

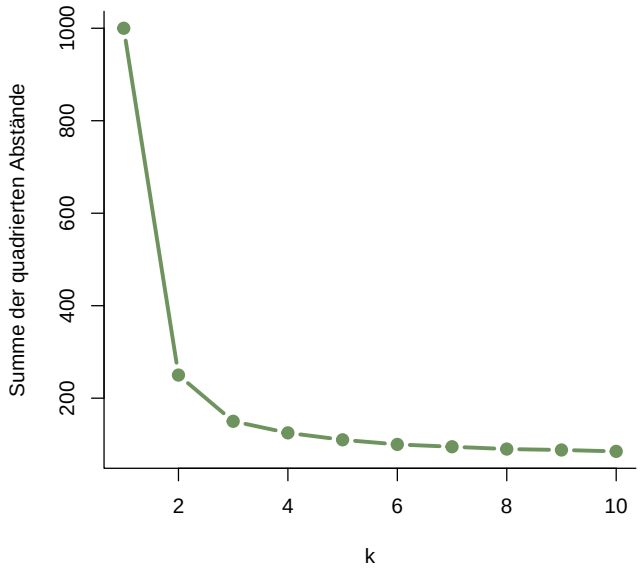
Generiere k zufällige Centroids
while Stopkriterium nicht erreicht **do**
    Ordne alle Punkte dem nächsten Cluster zu
    Berechne neue Clustermittelpunkte
end while

Stopkriterien Es sind verschiedene Kriterien denkbar, um k-Means zu beenden. Wenn nichts anderes gegeben ist, wird der Algorithmus beendet, sobald sich an der Zuteilung der Punkte zu den Clustern nichts mehr ändert. Sobald die Datensätze wachsen, bietet es sich aber auch an, aufzuhören, wenn sich bspw. weniger als 1 Prozent der Punkte noch ändern.

Nichtdeterminismus k-Means ist kein deterministischer Algorithmus. Das bedeutet, dass aus denselben Inputs (also dieselben Daten und derselbe Wert für k) bei mehrfacher Ausführung verschiedene Ergebnisse entstehen. Auf Basis der Distanzen zwischen Punkten und Centroids können Kennzahlen (wie die durchschnittliche Distanz zum Clustermittelpunkt) berechnet werden, die es uns erlauben, Ergebnisse von verschiedenen Durchläufen miteinander zu verlgeichen. Im Kontext einer solchen Kennzahl erreicht k-Means in der Regel nur lokale Optima. Es muss also kein Ergebnis dabei herauskommen, dass intuitiv oder sinnvoll ist. Sobald ein Datensatz mehr als zwei Dimensionen hat, wird der visuelle Vergleich schwierig. Eine Möglichkeit, damit umzugehen, ist es, mehrere k-Means-Durchläufe miteinander zu vergleichen – beispielsweise auf Basis der durchschnittlichen Abstände zwischen Punkten und den Clustermittelpunkten.

Laufzeit Ein Vorteil von k-Means ist die Laufzeit ($O\left(n^2\right)$). Diese ist (verglichen bspw. mit dem hierarchischem Clustering) deutlich besser, weshalb sich k-Means eher für große Datensätze eignet. Es entsteht ein *trade-off* zwischen Laufzeit und Determinismus.

Clusteranzahl In der Clusteranalyse ist die Frage nach der geeigneten Clusteranzahl zentral. Eine Herangehensweise dafür ist die sogenannte **Elbow-Methode**. Dabei wird k-Means für verschiedene Werte für k durchgeführt (zum Beispiel 1 bis 10) und eine Kennzahl wie die Summe der quadratischen Abstände zwischen Punkten und deren Clustermittelwerten berechnet. Dies liefert einen Graphen der folgenden Art.



Es wird dann diejenige Clusteranzahl gewählt, bei der absolute Änderungsrate stark abnimmt – in diesem Fall 2 oder 3. Die Intuition dahinter ist folgende: Eine höhere Clusteranzahl erschwert die Interpretation. Wir wollen im Normalfall eine möglichst geringe Clusteranzahl, gleichzeitig aber auch eine geringe Summe der quadrierten Abstände. Mehr als 2 oder 3 Cluster zu wählen bringt uns wenig für geringe Abstände, erschwert aber die Interpretation.

Konvergenz Um zu entscheiden, ob ein gegebenes Ergebnis einer Clusteranalyse von k-Means stammen kann, führt man eine weitere Iteration im Kopf durch und stellt folgende Fragen: Ändern sich die Positionen der Centroids? Ändert sich die Zuordnung einzelner Punkte zu den Clustern? Wenn eine der Fragen mit ja beantwortet wurde, würde k-Means noch nicht konvergieren.

k-Means Implementierung

```
#
# save as:
# Clustering.py
#

from Utils import loadData, plotClusters
from random import randint


def stop(cluster1, cluster2):
    """
    Simple stop criterium,
    compares whether cluster assignment changes between two clusters
    """
    return cluster1 == cluster2

def distance(p1, p2):
    """
    Euclidian distance function
    """
    dist = 0
    for i in range(len(p1)):
        dist += (p1[i]-p2[i])**2
    return dist**0.5

def randomCentroids(k, data):
    """
    generates k random centroids for data table
    """
    # find min and max value in each dimension
    mins = data[0].copy()
    maxs = data[0].copy()
    for row in data:
        for j in range(len(row)):
            if mins[j] > row[j]:
                mins[j] = row[j]
            elif maxs[j] < row[j]:
                maxs[j] = row[j]

    # return list of centroids with random values between min and max in each dimension
    return [[randint(mins[j], maxs[j]) for j in range(len(data[0]))] for i in range(k)]

def generateClusters(centroids, data):
    """
    assigns all points in data table to closest centroid in centroids,
    returns clusters as lists of data points
    """
    # go over all data points
```

```
    clusters = [[] for i in range(len(centroids))]
    for row in data:
        closestCluster = 0

        # find index of closest cluster
        for centroidIndex in range(len(centroids)):
            if distance(row, centroids[centroidIndex]) < \
               distance(row, centroids[closestCluster]):
                closestCluster = centroidIndex

        # assign point to closest cluster
        clusters[closestCluster].append(row)

    # return clusters
    return clusters

def updateCentroids(newClusters):
    """
    calculates centroids for clusters in newClusters list
    """
    # find dimensions in data & number of clusters
    dim = len(newClusters[0][0])
    nClusters = len(newClusters)

    # for each cluster, get mean value of points within cluster
    centroids = [[0 for i in range(dim)] for j in range(nClusters)]
    for clusterIndex in range(nClusters):

        # add value for each dimension in each point of cluster
        for pointIndex in range(len(newClusters[clusterIndex])):
            for valueIndex in range(dim):
                centroids[clusterIndex][valueIndex] += \
                        newClusters[clusterIndex][pointIndex][valueIndex]

        # divide by number of points in cluster to get average values
        for clusterValueIndex in range(dim):
            centroids[clusterIndex][clusterValueIndex] /= len(newClusters[clusterIndex])

    # return new clusters
    return centroids

def kmeans(k, filename):
    """
    performs k-Means algorithm on data in filename for k clusters,
    returns clusters as lists of data points
    """
    # load data and generate random centroids
    data = loadData(filename)
    centroids = randomCentroids(k, data)
```

```
    # while stop criterium is not met, generate new centroids,
    # and assign points to clusters
    oldClusters, newClusters = None, []
    while not stop(oldClusters, newClusters):

        # save old clusters for stop criterium, assign new clusters,
        # and generate new centroids
        oldClusters = newClusters
        newClusters = generateClusters(centroids, data)
        centroids = updateCentroids(newClusters)

    # return list of points for each cluster
    return newClusters

if __name__ == "__main__":
    """
    programme entry point
    - filename: name of csv file that contains data
    - k: number of clusters
    """
    # parameters
    filename = "data.csv"
    data = loadData(filename)
    k = 2

    # generate clusters and plot
    clusters = kmeans(k, filename)
    plotClusters(clusters, data)
```

```
#
# save as:
# Utils.py
#

import csv, re
from matplotlib import pyplot as plt
from Clustering import updateCentroids


def loadData(filename):
    data = []
    with open(filename) as csvfile:
        reader = csv.reader(csvfile)
        readFirstLine = False
        for row in reader:
            if not readFirstLine:
                if re.search("\D", row[0][0]):
```

```
                    readFirstLine = True
                    continue
            new_row = []
            for number in row:
                if re.search("\s", number[0]):
                    new_row.append(float(number[1:]))
                else:
                    new_row.append(float(number))
            data.append(new_row)
    return data

def printTable(data, nrow = 5):
    for i in range(nrow):
        for j in range(len(data[0])):
            print(data[i][j], "\t", end="")
        print()

def plotClusters(clusters, data):
    clusterAssignments = []
    for row in data:
        assignedCluster = 0
        for clusterIndex in range(len(clusters)):
            if row in clusters[clusterIndex]:
                assignedCluster = clusterIndex
                break
        clusterAssignments.append(assignedCluster)
    x = [row[0] for row in data]
    y = [row[1] for row in data]
    plt.scatter(x, y, c=clusterAssignments)
    centroids = updateCentroids(clusters)
    a = [row[0] for row in centroids]
    b = [row[1] for row in centroids]
    plt.scatter(a, b, marker = "x")
    plt.show()
```