

BDA, Praktikumsbericht 2

Gruppe mi6xc: Alexander Kniesz, Maximilian Neudert, Oskar Rudolf

Aufgabe 1

a)

Zuerst sollen wir ein Clustering auf der 10k-Stichprobe machen. Die KMeans Methode von `pyspark.ml` erwartet dazu ein DataFrame mit genau einer Spalte oder ein DataFrame mit einer Spalte `features`. Um dies zu ermöglichen transformieren wir die `bin_x` Spalten mit einem `VectorAssembler` zu einer feature Spalte, in der die vorherhigen Spalten die Dimensionen der neuen Vektoren sind. Dabei ist zu beachten, dass `VectorAssembler` je nach Speicherauslastung automatisch sparse oder voll wählt und in unserem Fall werden es sparse Vektoren, Das heißt, dass wir sehr viele 0 Werte haben. Anschließend erstellen wir ein KMeans object `KMeans().setK(2).setSeed(1)` durch Angabe der gewünschten Clusteranzahl ($k=2$) und des Start-Seeds und fitten damit dann das Modell anhand des neuen DataFrames.

```
%pyspark
# aufgabe 1
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans
from pyspark.ml.clustering import KMeansModel
from pyspark.ml.evaluation import ClusteringEvaluator

df = spark.sql("SELECT * FROM mi6xc_profilevectors")

assy = VectorAssembler(
    inputCols=["bin_0", "bin_1", "bin_2", "bin_3", "bin_4", "bin_5", "bin_6", "bin_7", "bin_8"],
    outputCol="features")

df_vec = assy.transform(df).select("features")

df_vec.show(5)

kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(df_vec)
```

```
+-----+
|          features|
+-----+
|[0.0,0.0,0.01,0.0...|
|(9,[4,5,6,7],[0.0...|
|(9,[5,6,7],[0.02,...|
|[0.0,0.01,0.02,0....|
|(9,[4,5,6,7],[0.0...|
+-----+
only showing top 5 rows
```

b)

Als quadratischen Fehler erhalten wir für die Wahl an Centroiden:

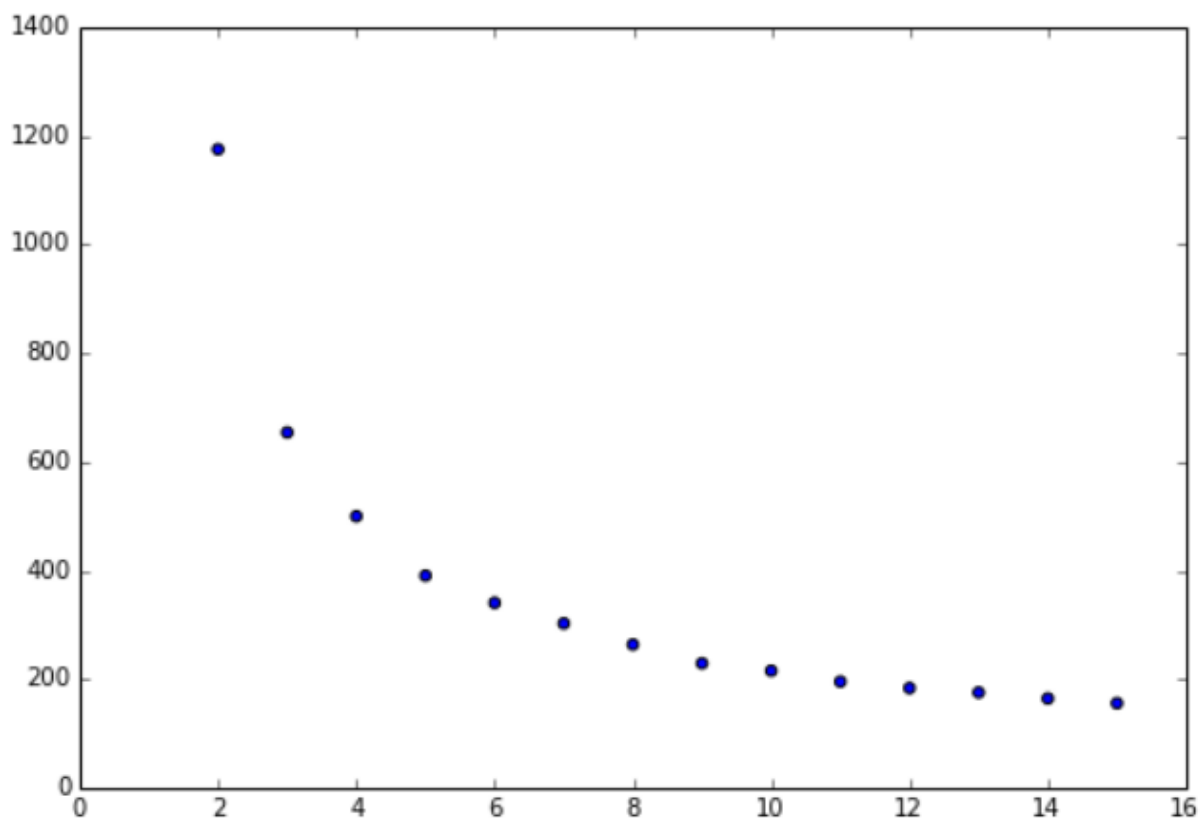
```
%pyspark
# Evaluate clustering.
cost = model.computeCost(df_vec)

# Quadratischer Fehler
print(cost)
```

1175.5134576277374

c)

Plotten wir die quadratischen Fehler im Bereich 2 bis 16 (Anzahl an Centroiden), so erhalten wir folgenden Plot:



Bei dem dabei entstandenen Plot wenden wir die Elbow-Methode an, bei der wir abschätzen, ab welcher Anzahl von Clustern sich die Steigung kaum noch ändert. Wir entscheiden uns für **5** Cluster, da der Unterschied von 4 zu 5 nach unserer Meinung noch nennenswert ist, zusätzliche Cluster ab diesem Punkt aber kaum noch Mehrwert bringen:

FINISHED   

```
%pyspark

kmeans_final = KMeans().setK(5).setSeed(1)
model_final = kmeans_final.fit(df_vec)

centers = model_final.clusterCenters()

print("Cluster Centers: ")
for center in centers:
    print(center)

# quadratischer Fehler
cost = model_final.computeCost(df_vec)
print("Error: " + str(cost))

# Save model
model_final.write().overwrite().save("hdfs://141.100.62.85:9000/user/istosrudo/gruppe_mi6x_k_means_5_model")
```

Cluster Centers:

```
[1.26396425e-03 2.11937440e-03 4.82604532e-03 1.24130227e-02
 3.98691350e-02 2.04101500e-01 6.33562081e-01 9.91318225e-02
 9.57548675e-06]
[3.90025575e-04 7.03324808e-04 1.68797954e-03 4.11125320e-03
 9.92966752e-03 2.75255754e-02 1.59283887e-01 7.75147059e-01
 1.74552430e-02]
[8.98804048e-03 3.05059798e-02 9.27046918e-02 2.37442502e-01
 3.75722171e-01 2.09337626e-01 4.00459982e-02 3.69825207e-03
 2.75988960e-05]
[1.71490281e-03 3.62850972e-03 1.05874730e-02 3.91317495e-02
 1.71943844e-01 4.92496760e-01 2.62574514e-01 1.55507559e-02
 7.77537797e-05]
[0.00091005 0.0016728 0.00395581 0.01051026 0.02789058 0.09693319
 0.42960021 0.42037875 0.00516044]
Error: 390.7951234005651
```

d)

Anwendung unseres Cut-Off-Kriteriums $k=5$ auf den k-means-Algorithmus und abspeichern des resultierenden Modells:

```
%pyspark

kmeans_final = KMeans().setK(5).setSeed(1)
model_final = kmeans_final.fit(df_vec)

centers = model_final.clusterCenters()

# quadratischer Fehler
cost = model_final.computeCost(df_vec)
print("Error: " + str(cost))

# Save model
model_final.write().overwrite().save("hdfs://141.100.62.85:9000/user/istosrudo/gruppe_mi6x_k_means_5_model")

Error: 390.7951234005651
```

Took 4 sec. Last updated by istosrudo at May 14 2019, 3:30:08 PM.

e) - f)

Um mit den Daten in Tableau arbeiten zu können erstellen wir eine neue Tabelle mit den Profilvektoren einerseits und den aus dem Modell berechneten Predictions andererseits. Die so erstellte Tabelle zeigt uns nun für jeden Track eine zugehörige Clusternummer:

FII

```
%pyspark
from pyspark.sql.functions import concat, col, lit
from pyspark.sql.functions import monotonically_increasing_id
import numpy as np

predictions = model_final.transform(df_vec).select("prediction")

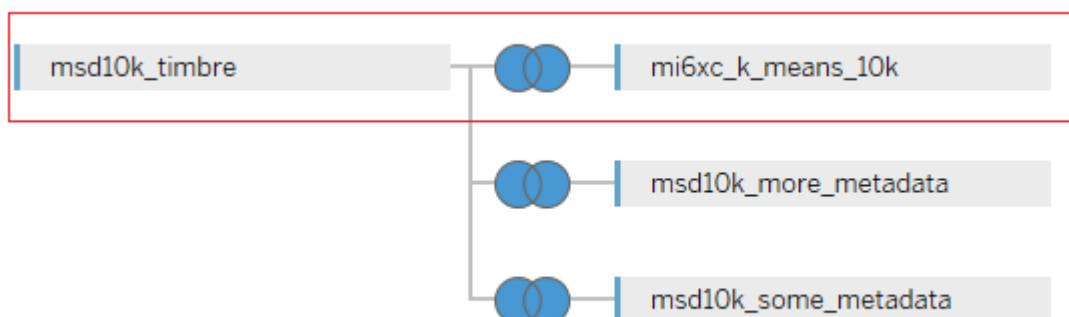
left = df.withColumn("index",monotonically_increasing_id())
right = predictions.withColumn("index",monotonically_increasing_id())

result_df = left.join(right, on="index")
result_df.write.mode("overwrite").saveAsTable("mi6xc_k_means_10k")

result.show()
```

index	track_id	bin_0	bin_1	bin_2	bin_3	bin_4	bin_5	bin_6	bin_7	bin_8	prediction
0	TRAZKDD12903CC8328	0.0	0.0	0.01	0.01	0.07	0.21	0.51	0.19	0.0	0
1	TRALYIQ12903CEC83A	0.0	0.0	0.0	0.0	0.03	0.32	0.6	0.04	0.0	0
2	TRADTBA128F92CA48F	0.0	0.0	0.0	0.0	0.0	0.02	0.22	0.76	0.0	1
3	TRAEDJW128F422B4CB	0.0	0.01	0.02	0.07	0.24	0.56	0.11	0.0	0.0	3
4	TRATPUS128F4271260	0.0	0.0	0.0	0.0	0.03	0.29	0.65	0.02	0.0	0
5	TRABIOI12903CD8B9B	0.0	0.0	0.0	0.0	0.01	0.02	0.12	0.84	0.0	1
6	TRAZUKY128F426C8BC	0.0	0.0	0.01	0.0	0.01	0.03	0.15	0.8	0.0	1
7	TRBHBEB128F92CC0C6	0.0	0.0	0.0	0.01	0.15	0.54	0.29	0.0	0.0	3
8	TRASOQC12903CFFED7	0.0	0.0	0.0	0.0	0.0	0.01	0.04	0.94	0.0	1
9	TRAVHPV128F933E986	0.0	0.0	0.0	0.01	0.01	0.06	0.39	0.53	0.0	4
10	TRALNOD128F4264AEB	0.0	0.0	0.0	0.01	0.04	0.38	0.56	0.0	0.0	0
11	TRAOVFR128F93275B4	0.01	0.0	0.02	0.04	0.16	0.32	0.42	0.03	0.0	3
12	TRAXDLB128F423F8F8	0.0	0.0	0.01	0.0	0.01	0.12	0.76	0.09	0.0	0
13	TRAJLOY128F92E4EAF	0.01	0.01	0.01	0.02	0.16	0.41	0.13	0.23	0.0	3
14	TRAP00P128F42142F6	0.0	0.02	0.16	0.38	0.14	0.18	0.12	0.0	0.0	2

Anschließend haben wir die Tabelle mit den restlichen Daten verbunden:



g)

Um sich unter den Clustern etwas vorstellen zu können, kann man diese einmal als Verlauf plotten und eine gewichtete Summe über die Bins berechnen:

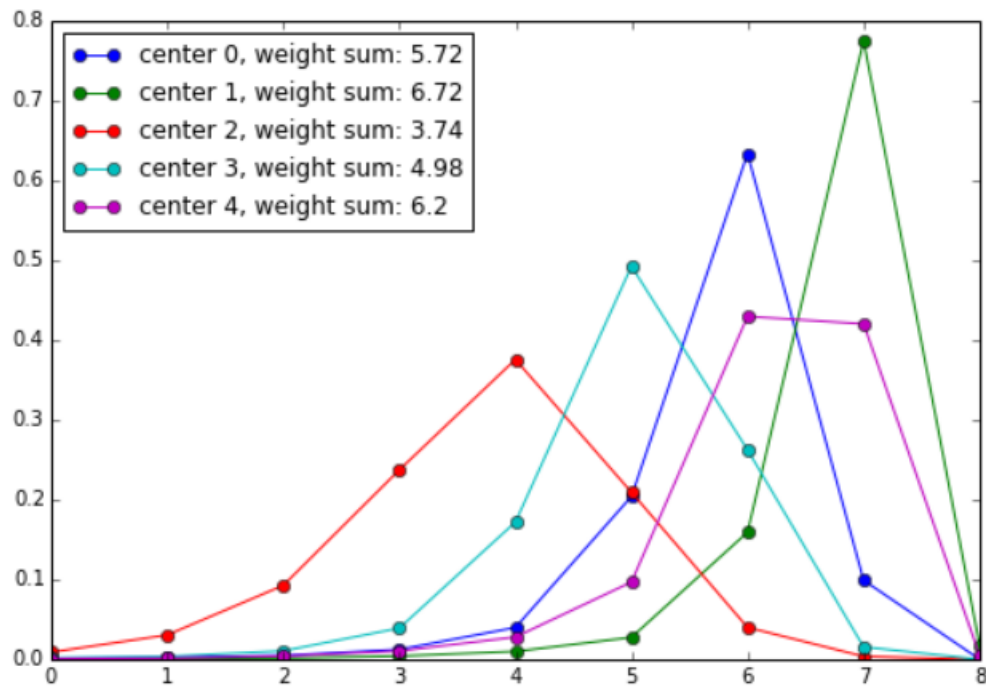
```
%pyspark
from matplotlib import pyplot as plt

for c, center in enumerate(centers):
    k = 0
    for i in range(0, len(center)):
        k += i * center[i]

    # plot models
    plt.plot(range(0, len(center)), center, 'o-', label='center {}, weight sum: {}'.format(c, round(k, 2)))
    plt.legend(loc='upper left')

plt.show()
```

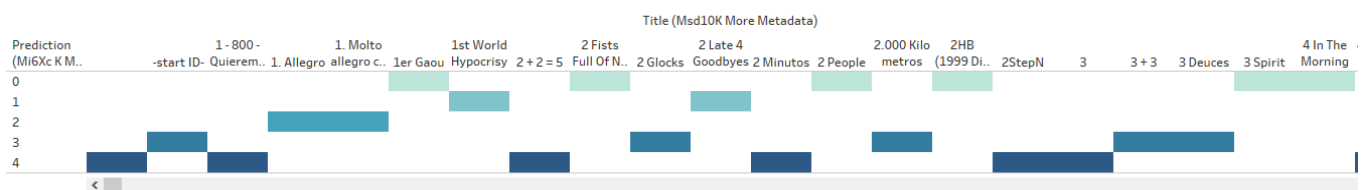
FINISHED ▶ ✖ 📖



Als Resultat sieht man eine Art Loudness Profil pro Cluster.

In Tableau kann man nun diverse Visualisierungen bilden. Ein paar Beispiele:

Song_To_cluster



In der Grafik sieht man die Songs mit und deren zugeordneter Cluster.

Hörprobe

Als Hörproben haben wir verschiedene Songs von einer ausgewählten Künstlerin (Britney Spears) angehört, die nach unserer Analyse in verschiedenen Clustern liegen und daher unterschiedliche Lautstärkeprofile haben sollten.

Aufgabe 2

Wir übertragen nun die Arbeit aus Aufgabe 1 a)-c). Wichtig ist dabei nochmal mittels Elbow Methode zu prüfen, ob eventuell ein andere Anzahl an Clustern notwendig ist.

```
%pyspark
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans
from pyspark.ml.clustering import KMeansModel
from pyspark.ml.evaluation import ClusteringEvaluator
from matplotlib import pyplot as plot

#spark.sql("show tables like 'profilevectorpertrack_1m']").show()

df = spark.sql("SELECT * FROM profilevectorpertrack_1m")

assy = VectorAssembler(
    inputCols=["bin1", "bin2", "bin3", "bin4", "bin5", "bin6", "bin7", "bin8", "bin9", "bin10"],
    outputCol="features")

df_vec = assy.transform(df).select("features")
df_vec.show(10)

kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(df_vec)

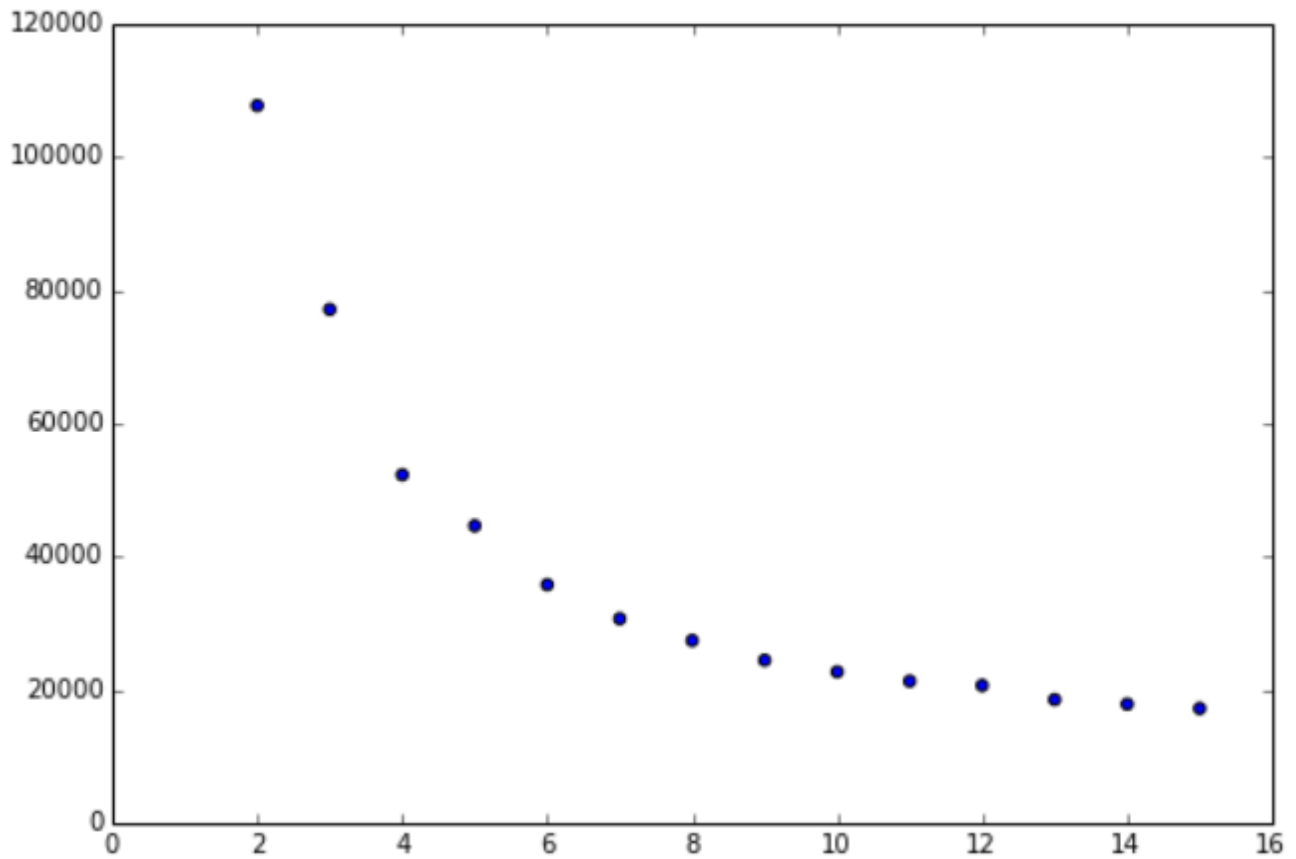
costvector = []

for i in range(2,16):
    kmeans_loop = KMeans().setK(i).setSeed(1)
    model_loop = kmeans_loop.fit(df_vec)

    centers = model_loop.clusterCenters()
    print("k = " + str(i))
    print("Cluster Centers: ")
    for center in centers:
        print(center)

    # Evaluate clustering.
    costvector.append(model_loop.computeCost(df_vec))

plot.scatter(x=range(2,16) , y=costvector)
```



Man sieht einen kleinen Unterschied und hier würde es sich anbieten 6 Cluster zu verwenden.

Die genauen Arbeitsschritte sind im Notebook unter [Zeppelin Notebook_GruppeMi6xc](#) dokumentiert.

Unterschiede und Gemeinsamkeiten zwischen 10k und 1M: