Pr1_Clustering_sol

October 30, 2020

1 Praktikum: Session 1

2 Clustern der Iris-Daten

Clustern von Daten ist eine typische *unsupervised* Aufgabe. Dabei sollen nicht-gelabelten Daten in mehrere Gruppen ("Cluster") unterteilt werden.

Wir wollen dies konkret anhand der klassischen Iris-Daten machen. *Wir verwenden hier nur die vierdimensionalen Daten, die Labels werden NICHT verwendet.*

Wir beginnen mit Imports für die Grafik:

[1]:
```
%matplotlib inline
import seaborn as sns; sns.set()
```

2.1 1. Clustern mit K-Means:

Video

Vorgehen entlang der 5 Schritte

2.1.1 1. Wähle Modellklasse

Wir wollen K-Means Clustering verwenden. Importieren Sie diese aus `sklearn.cluster`.

Hinweis: Wie genau die Modellklasse heißt, können Sie sich anzeigen lassen, indem Sie den Anfangsbuchstaben eingeben und dann kurz auf die Code-Vervollständigung warten.

[4]:
```
from sklearn.cluster import KMeans, k_means
```

[5]:
```
k_means?
```

2.1.2 2. Wähle Modellparameter

Informieren Sie sich, welche Parameter das Modell benötigt und wählen Sie diese passend. Geben Sie dem Modell den Namen `model_KM`.

[3]:
```
KMeans?
```

[4]:
```
model_KM = KMeans(n_clusters=3)
```

2.1.3 3. Bereite Daten vor

Laden Sie die Iris Daten. Verschaffen Sie sich einen Überblick, wie die Daten strukturiert sind.

Hinweise: - Die Iris Daten sind u.a. in der Bibliothek `seaborn` enthalten. Dort gibt es eine Funktion namens `load_dataset`. - Durch Aufrufen von `meine_daten.head()` werden die ersten fünf Zeilen des DataFrame `meine_daten` angezeigt.

[5]:
```
import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```

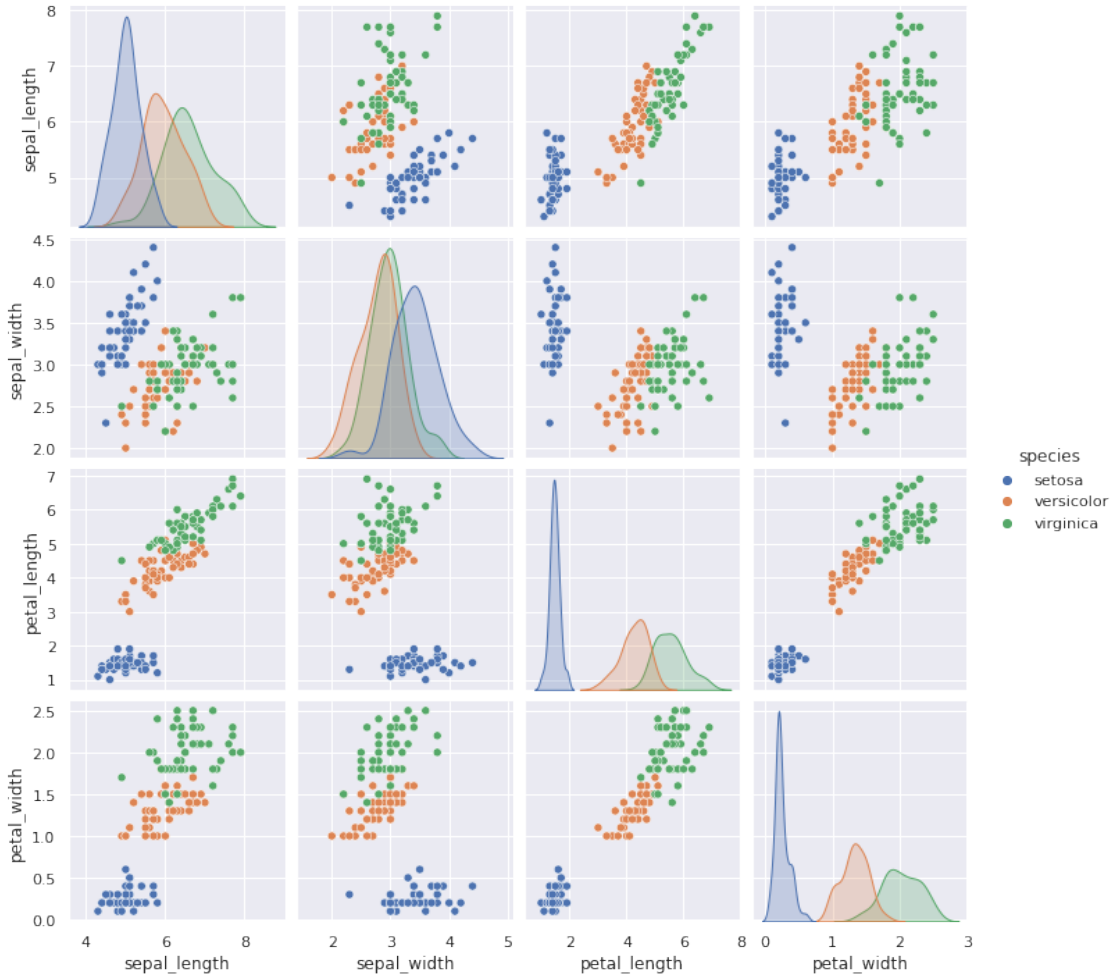
[5]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Plotten Sie die Daten, um einen Eindruck zu bekommen, ob ein Clustering vielversprechend ist. *Hinweis: Verwenden Sie eine* ***`plot`****-Funktion von Seaborn. Die automatische Code-Vervollständigung kann Ihnen zeigen, welche Funktionen es gibt, die* ***plot*** *im Namen haben.*

[6]:
```
sns.pairplot(iris, hue='species')
```

[6]:
```
<seaborn.axisgrid.PairGrid at 0x7f63445dab70>
```



Extrahieren Sie aus den eingelesenen Daten die Features-Matrix `X_iris` und den Labels-Vektor `y_iris`.

Hinweise: - Aus einem DataFrame `df` können Spalten entfernt werden durch `df.drop(['Namen der Spalten', 'die entfernt werden sollen'], axis=1)` - Auf eine einzelne Spalte eines DataFrame `df` kann zugegriffen werden durch `df['Name der Spalte']`

[7]:
```
import seaborn as sns
iris = sns.load_dataset('iris')
X_iris = iris.drop('species', axis=1)
y_iris = iris['species']
```

2.1.4 4. Passe das Modell an die Daten an

[8]:
```
model_KM.fit(X_iris)
```

[8]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

2.1.5 5. Wende das Modell an

Verwenden Sie das erzeugte Modell, um die Datensätze zu clustern.

Konkret heißt das, dass für jeden einzelnen Datensatz vorhergesagt werden soll, in welchen Cluster er gehört. Schauen Sie sich an, in welcher Form diese Information durch das Modell bereitgestellt wird.

Hinweis: Verwenden Sie die `predict` *Methode.*

[9]:
```
y_KM = model_KM.predict(X_iris)
```

[10]:
```
y_KM
```

[10]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 1, 1, 2, 1,
       1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 2, 2, 1, 1,
       1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2])
```

2.1.6 6. Betrachte Ergebnis

Fügen Sie zunächst die in Schritt 5. erzeugten Cluster-Daten dem Iris-Datensatz hinzu.

Hinweis: Einem DataFrame `df` *kann ein Array* `array` *als neue Spalte hinzugefügt werden durch* `df['Gewünschter Name der neuen Spalte'] = array`.

[11]:
```
iris['cluster_KM'] = y_KM
```

Verwenden Sie die Funktion `sns.lmplot`, um die Iris Daten entlang zweier von Ihnen gewählten Features darzustellen. Die Färbung soll gem. des Labels `species` gewählt werden. Die Zugehörigkeit zu den Clustern gem. dem K-Means Modell soll dadurch dargestellt werden, dass jeder Cluster seinen eigenen Subplot erhält.

Hinweis: Verwenden Sie `sns.lmplot("Name Feature 1", "Name Feature 2", data=iris, hue='nach was gefärbt werden soll', col='nach was die Subplots erstellt werden sollen', fit_reg=False)`

[13]:
```
sns.lmplot(x="petal_length", y="petal_width", data=iris, hue='species',␣
↪col='cluster_KM', fit_reg=False);
```

2.2 2. Clustern mit Gaussian Mixture Model

Video

Vorgehen entlang der 5 Schritte

2.2.1 1. Wähle Modellklasse

Nun soll zum Clustern ein sog. Gaussian Mixture Model verwendet werden.

Hinweis: Dieses befindet sich in ***`sklearn.mixture`****.*

[14]:
```
from sklearn.mixture import GaussianMixture
```

###2. Wähle Modellparameter

Informieren Sie sich, welche Parameter das Modell benötigt und wählen Sie diese passend. Geben Sie dem Modell den Namen `model_GMM`.

[15]:
```
model_GMM = GaussianMixture(n_components=3, covariance_type='full')
```

###3. Bereite Daten vor Die Iris Daten sind ja bereits geladen und somit noch immer verfügbar. Extrahieren Sie daraus die Features-Matrix `X_iris` und den Labels-Vektor `y_iris`.

Hinweis: Sie hatten in Teil 1 die Iris Daten verändert, indem Sie das Ergebnis des Clusterings hinzugefügt haben. Stellen Sie sicher, dass ***`X_iris`*** *und* ***`y_iris`*** *korrekt sind. Eine gute Möglichkeit wäre es, die Daten noch einmal zu betrachten (****`head()`****).*

[16]:
```
iris.head()
```

[16]:

	sepal_length	sepal_width	petal_length	petal_width	species	cluster_KM
0	5.1	3.5	1.4	0.2	setosa	0
1	4.9	3.0	1.4	0.2	setosa	0
2	4.7	3.2	1.3	0.2	setosa	0
3	4.6	3.1	1.5	0.2	setosa	0
4	5.0	3.6	1.4	0.2	setosa	0

[17]:
```
X_iris = iris.drop(['species', 'cluster_KM'], axis=1)
y_iris = iris['species']
```

###4. Passe das Modell an die Daten an

[18]:
```
model_GMM.fit(X_iris)
```

[18]:
```
GaussianMixture(covariance_type='full', init_params='kmeans', max_iter=100,
                means_init=None, n_components=3, n_init=1, precisions_init=None,
                random_state=None, reg_covar=1e-06, tol=0.001, verbose=0,
                verbose_interval=10, warm_start=False, weights_init=None)
```

###5. Wende das Modell an Verwenden Sie das neu erzeugte Modell, um die Datensätze zu clustern.

Konkret heißt das, dass für jeden einzelnen Datensatz vorhergesagt werden soll, in welchen Cluster er gehört. Schauen Sie sich an, in welcher Form diese Information durch das Modell bereitgestellt wird.

Hinweis: Verwenden Sie die `predict` *Methode.*

[19]:
```
y_GMM = model_GMM.predict(X_iris)
y_GMM
```

[19]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 0, 2, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

###6. Betrachte Ergebnis Fügen Sie zunächst die in Schritt 5 erzeugten neuen Clusterdaten dem Iris-Datensatz hinzu.

[20]:
```
iris['cluster_GMM'] = y_GMM
```

[21]:
```
iris.head()
```

[21]:

	sepal_length	sepal_width	petal_length	…	species	cluster_KM	cluster_GMM
0	5.1	3.5	1.4	…	setosa	0	1
1	4.9	3.0	1.4	…	setosa	0	1
2	4.7	3.2	1.3	…	setosa	0	1
3	4.6	3.1	1.5	…	setosa	0	1
4	5.0	3.6	1.4	…	setosa	0	1

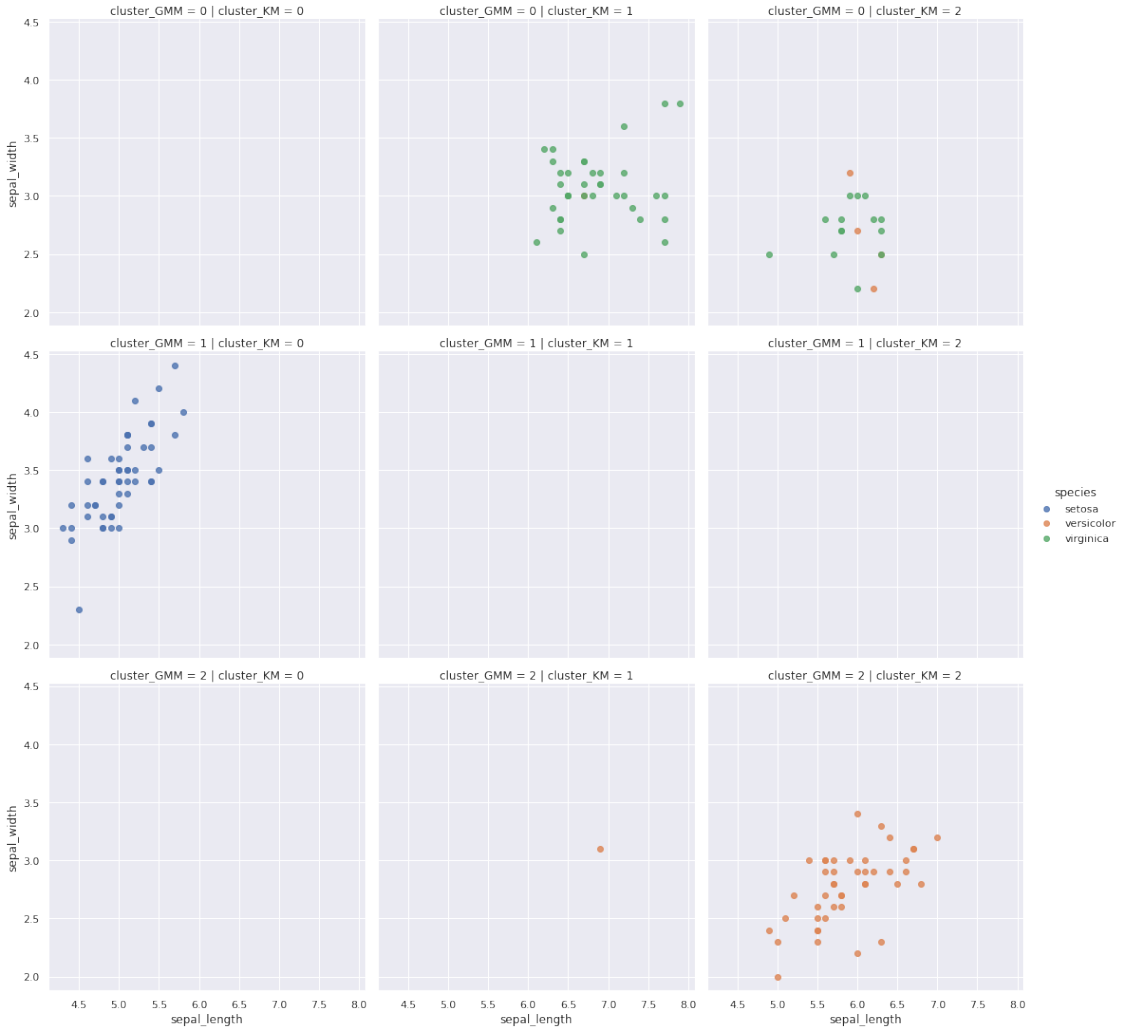
[5 rows x 7 columns]

Verwenden Sie die Funktion `sns.lmplot`, um die Iris Daten entlang zweier von Ihnen gewählten Features darzustellen. Die Färbung soll gem. des Labels `species` gewählt werden. Die Zuge-

hörigkeit zu den Clustern gem. dem K-Means Modell soll weiterhin dadurch dargestellt werden, dass jeder Cluster seinen eigenen Subplot innerhalb einer Zeile erhält. Die Zugehörigkeit zu den Clustern gem. dem Gaussian Mixture Modell soll dadurch dargestellt werden, dass jeder Cluster seinen eigenen Subplot innerhalb einer *Spalte* erhält.

Somit ergiben sich 9 Subplots angeordnet in einem 3x3 Schema. Dabei gilt: - eine Zeile enthält alle Punkte eines GMM-Clusters; - eine Spalte enthält alle Punkte eines KM-Clusters.

[22]:
```
sns.lmplot(x="sepal_length", y="sepal_width", data=iris, hue='species',␣
↪col='cluster_KM', row='cluster_GMM', fit_reg=False);
```



Interpretieren Sie die Grafik. Welches Clustering ist besser?

[23]:
```
iris.head()
```

[23]:
```
   sepal_length  sepal_width  petal_length  …  species cluster_KM  cluster_GMM
0           5.1          3.5           1.4  …   setosa          0            1
1           4.9          3.0           1.4  …   setosa          0            1
2           4.7          3.2           1.3  …   setosa          0            1
3           4.6          3.1           1.5  …   setosa          0            1
4           5.0          3.6           1.4  …   setosa          0            1

[5 rows x 7 columns]
```

[25]:
```
iris.loc[iris.cluster_KM==2].loc[iris.cluster_GMM==0]
```

[25]:
```
     sepal_length  sepal_width  …  cluster_KM  cluster_GMM
68            6.2          2.2  …           2            0
70            5.9          3.2  …           2            0
72            6.3          2.5  …           2            0
83            6.0          2.7  …           2            0
101           5.8          2.7  …           2            0
106           4.9          2.5  …           2            0
113           5.7          2.5  …           2            0
114           5.8          2.8  …           2            0
119           6.0          2.2  …           2            0
121           5.6          2.8  …           2            0
123           6.3          2.7  …           2            0
126           6.2          2.8  …           2            0
127           6.1          3.0  …           2            0
133           6.3          2.8  …           2            0
138           6.0          3.0  …           2            0
142           5.8          2.7  …           2            0
146           6.3          2.5  …           2            0
149           5.9          3.0  …           2            0

[18 rows x 7 columns]
```

2.3 3. Daten mit besonderer Struktur

Nun betrachten wir künstlich generierte Daten mit einer besonderen Struktur. Wir wollen vergleichen, wie die beiden Algorithmen darauf performen.

Führen Sie die beiden folgenden Zellen aus, um die Daten zu generieren. Diese liegen dann analog zu den Iris Daten als `DataFrame` Objekt mit Namen `df` und auch als `ndarray` mit Namen `X_aniso` vor.

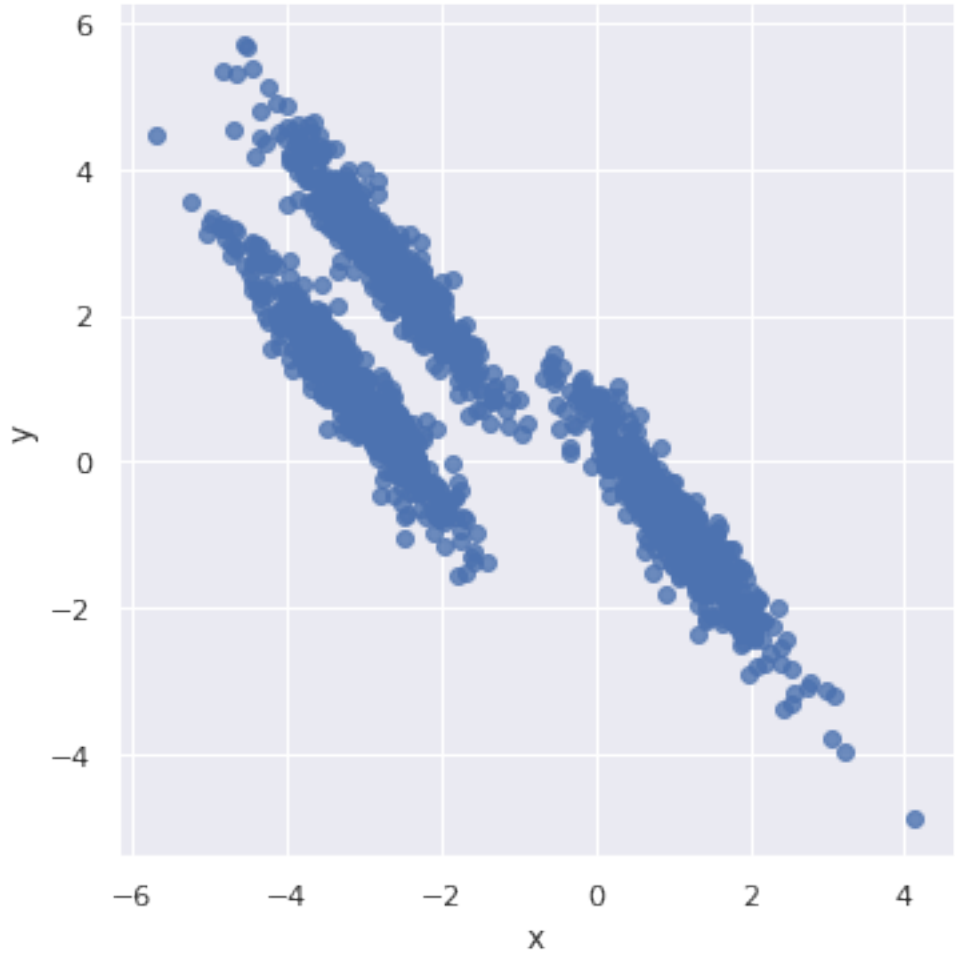
[26]:
```
import numpy as np
import pandas as pd

from sklearn import datasets
```

[27]:
```
n_samples = 1500
# Anisotropicly distributed data
random_state = 170
X,y = datasets.make_blobs(n_samples=n_samples, random_state=random_state)
transformation = [[0.6, -0.6], [-0.4, 0.8]]
X_aniso = np.dot(X, transformation)
df = pd.DataFrame({'x':X_aniso[:,0], 'y':X_aniso[:,1]})
```

Betrachten Sie die Daten und erzeugen Sie einen Plot. Welche Struktur fällt Ihnen auf, d.h. wie sollte ein Algorithmus diese Daten clustern?

[29]:
```
sns.lmplot(x='x',y='y', data=df, fit_reg=False)
```

[29]:
```
<seaborn.axisgrid.FacetGrid at 0x7f63407c0e80>
```



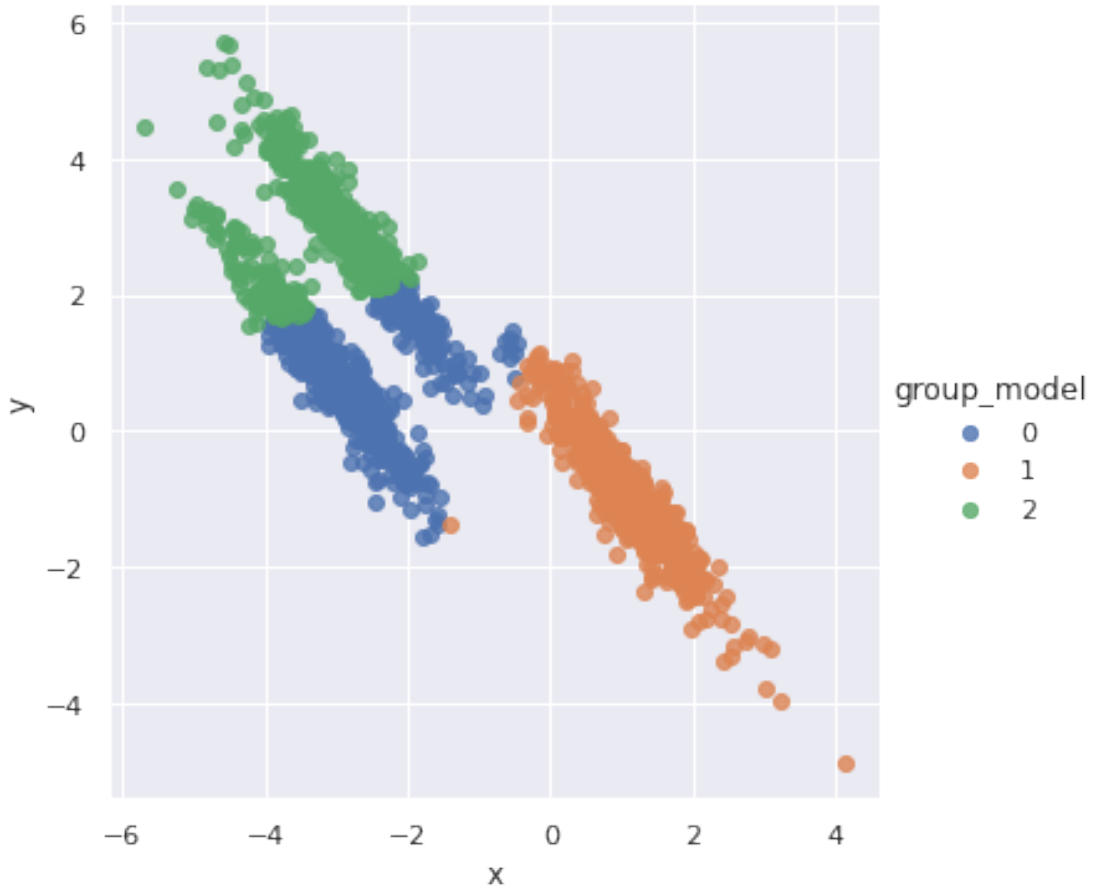
2.3.1 Clustern mit K-Means

Verwenden Sie den K-Means-Algorithmus um die Daten zu clustern. Visualisieren Sie das Ergebnis und bewerten Sie es basierend darauf.

[30]:
```
model = KMeans(n_clusters=3)
model.fit(X_aniso)
group_model = model.predict(X_aniso)
df['group_model'] = group_model
```

[32]:
```
sns.lmplot(x='x',y='y', data=df, hue='group_model', fit_reg=False)
```

[32]:
```
<seaborn.axisgrid.FacetGrid at 0x7f633ff778d0>
```

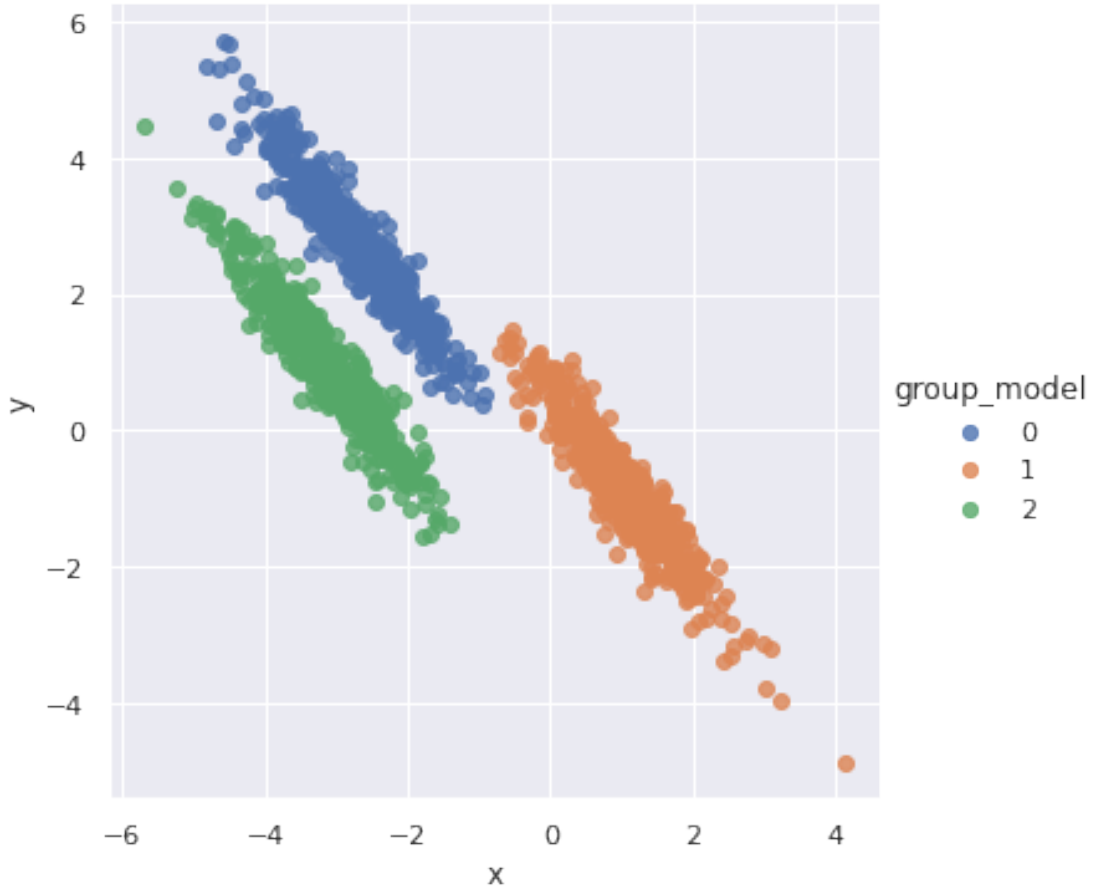


2.3.2 Clustern mit GMM

Verwenden Sie ein Gaussian Mixture Model um die Daten zu clustern. Visualisieren Sie das Ergebnis und bewerten Sie es basierend darauf.

[33]:
```
model = GaussianMixture(n_components=3, covariance_type='full')
model.fit(X_aniso)
group_model = model.predict(X_aniso)
df['group_model'] = group_model
sns.lmplot(x='x',y='y', data=df, hue='group_model', fit_reg=False)
```

[33]: <seaborn.axisgrid.FacetGrid at 0x7f633e2e11d0>



[ ]: