TSRI-9

July 23, 2024

1 Übung 9

Gruppenname: TSRI

- Christian Rene Thelen @cortex359
- Leonard Schiel @leo_paticumbum
- Marine Raimbault @Marine Raimbault
- Alexander Ivanets @sandrium

1.0.1 In dieser Übung ...

... werden Sie vertraut mit Clustervalidierungstechniken. Diese sind nützlich, um Parameter (wie beispielsweise die Anzahl der Cluster) einzustellen und die Qualität eines Clusterings zu bewerten.

1.0.2 9.1 Prediction Strength (Bestimmung der Clusteranzahl für K-Means)

Sie haben in der Vorlesung verschiedene Clustervalidierungsverfahren kennengelernt. Die *Prediction Strength* ist eine empfehlenswerte Validierungsmethode, die Sie im Rahmen dieser Übung implementieren und damit Ihr Verständnis vertiefen werden.

Hinweise

- Nutzen Sie die Implementierung des K-Means Algorithmus von Scikit-Learn.
- Für die Implementierung der *Prediction Strength* können Sie numpy, sklearn und scipy verwenden.

Ihre Aufgaben

(1) Schlagen Sie in den Vorlesungsfolien nach, wie die Prediction Strength definiert ist.

Der **Prediction Strength-Ansatz** geht davon aus, dass die Clusterqualität hoch ist, wenn Clusterzugehörigkeiten auf anderen Realisation der Daten zuverlässig vorhergesagt werden können. Dabei wird das Finden richtiger Clustering Parameter als *Model Selection Problem* (wie aus dem Supervised Learning) betrachtet. Dort werden beste Parameter über die Optimierung der Vorhersagen auf Out-of-Sample Daten (Maximierung der prediction strength bzw. Minimierung des Vorhersagefehlers E_{out}) im Rahmen einer Validierung gefunden.

Sei $\mathcal{C} = \{C_1, \dots, C_k\}$ ein Clustering von $\mathcal{D}_{\text{train}}$, deren Datenpunkte in den jeweiligen Regionen R_{C_i} des Merkmalsraum liegen, $\mathcal{A} = \{A_1, \dots, A_K\}$ ein Clustering von $\mathcal{D}_{\text{val}}$ und M eine $N_{\text{val}} \times N_{\text{val}}$-Matrix (sog. Ko-Mitgliedschaftsmatrix) mit

$$M_{ii'} := \begin{cases} 1 & \exists k \text{ mit } (i, i') \in R_{C_k} \\ 0 & \text{sonst} \end{cases}$$

ist. (Ein Matrixeintrag in M ist 1, wenn die jeweiligen zwei Datenpunkte aus dem Validierungsset zur selben Region eines Clusters k im Trainingset gehören.)

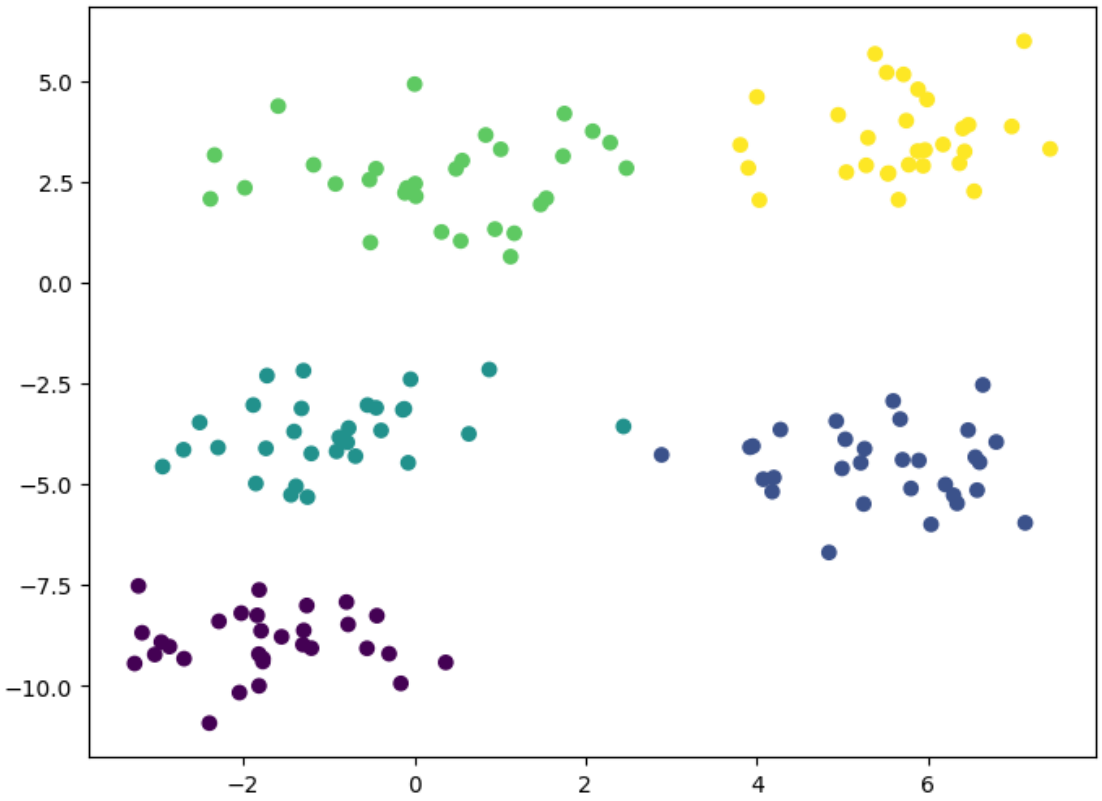
Dann ist die *prediction strength*

$$\mathrm{ps}(K) = \min_{j=1,\dots,K} \frac{1}{|A_j|(|A_j|-1)} \sum_{i,i' \in A_j,\ i \neq i'} M_{ii'}$$

(2) Ich habe Ihnen synthetische Daten zur Übung bereitgestellt. Bitte führen Sie die unten stehende Code-Zelle aus. Sie erzeugt 150 Datenpunkte mit je zwei Features (Merkmalen, Array $\mathbf{X}$), organisiert in 5 Clustern. Die "wahren" Clusterzugehörigkeiten sind im Vektor $\mathbf{y}$ kodiert.

- In der Praxis haben Sie selbstverständlich keine wahren Clusterzugehörigkeiten vorliegen. Sie wollen vielmehr im Rahmen Ihrer Clusteranalyse diese Clusterzugehörigkeiten erkennen. In dieser Übung nutzen wir $\mathbf{y}$ lediglich für die Visualisierung, die die folgende Codezelle erzeugt, und löschen danach diese Variable wieder.

[1]:
```
import numpy as np
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt, rcParams
rcParams['figure.figsize'] = (8, 6)

# generate data
X, y = make_blobs(n_samples=150, centers=5, cluster_std=1, random_state=40)

# visualize data
plt.scatter(x=X[:, 0], y=X[:, 1], c=y)

del y
```



(3) Teilen Sie die Daten **zufällig** in ein Trainings- und ein Validierungsset auf. Das Validierungsset soll dabei 20% der Gesamtdatenpunkte enthalten, das Trainingsset 80%.

- Ihre Datenvariablen heißen `X_train` und `X_valid`. (Scikit-Learn bezeichnet unser Validierungsset als Testset; lassen Sie sich dadurch nicht verwirren.)
- Nutzen Sie `random_state=40`, damit Sie Ihre Ergebnisse einfach mit denen Ihrer Kolleginnen und Kollegen vergleichen können.

[2]:
```
from sklearn import model_selection

X_train, X_valid = model_selection.train_test_split(X, test_size=0.2,␣
↪random_state=40)
```

(4) Für die nachfolgenden (Debugging-)Schritte ist es hilfreich, wenn Sie bereits jetzt mittels K-Means jeweils Cluster im Trainings- und Validierungsset bestimmen.

- Nutzen Sie bei den nachfolgenden Schritten jeweils `random_state=42`, um vergleichbare Ergebnisse wie Ihre Kolleginnen und Kollegen zu erhalten.

1. Ermitteln Sie mittels K-Means $K = 8$ Cluster im **Trainingsset** (setzen Sie den Parameter `n_init=10`, um die Warning zu deaktivieren). Speichern Sie die *Centroids* (auch *cluster_centers* genannt) der Cluster in der Variablen `train_centroids`.

2. Ermitteln Sie mittels eines **neuen** K-Means Modells ebenfalls $K = 8$ Cluster im **Validierungsset** (setzen Sie den Parameter `n_init=10`, um die Warning zu deaktivieren). Speichern Sie die Clusterzugehörigkeiten (*cluster labels*) der Daten im Validierungsset in der Variablen `valid_labels`.

[3]:
```
from sklearn.cluster import KMeans
kmeans_train = KMeans(n_clusters=8, random_state=42, n_init=10).fit(X_train)
kmeans_valid = KMeans(n_clusters=8, random_state=42, n_init=10).fit(X_valid)
train_centroids = kmeans_train.cluster_centers_
valid_labels = kmeans_valid.labels_
```

(5) Sie werden die Prediction Strength in mehreren Schritten implementieren. Implementieren Sie zunächst eine Funktion `get_comembership_matrix`, die die Co-Membership Matrix M (Ko-Mitgliedschaftsmatrix) aus den Daten bestimmt.

Hintergrund

Die Matrix M kodiert, welches Paar von Datenpunkten aus dem Validierungsset demselben Cluster im Trainingsset zugeordnet ist. *Damit enthält die Implementierung der Funktion `get_comembership_matrix` implizit unsere Annahmen darüber, wie ein Clustergebiet im Merkmalsraum aussehen sollte.* Verschiedene Clusterverfahren treffen unterschiedliche Annahmen. Daher unterscheidet sich die Implementierung von `get_comembership_matrix` je nach gewähltem Clusteringverfahren!

Wir wollen später Clusterings validieren, die wir mittels K-Means erzeugt haben. Bei K-Means werden Datenpunkte dem Cluster zugeordnet, dessen *Centroid* (Clusterzentrum) sie am nächsten liegen. Um zu bestimmen, ob zwei Datenpunkte aus dem Validierungsset demselben Cluster des Trainingssets angehören, müssen Sie also prüfen, ob Sie demselben Centroid aus dem Trainingsset am nächsten liegen.

Hinweise

- Die Funktion `get_comembership_matrix` nimmt die Daten `X_valid` sowie die `train_centroids` entgegen und gibt die Co-Membership Matrix `M` zurück.
- Bei der Implementierung können Ihnen Funktionen hilfreich sein, die Sie unterstützen bei (i) der Bestimmung euklidischer Distanzen, (ii) bei der Bestimmung des Arrayindexes mit dem kleinsten Eintrag sowie (iii) der Bestimmung aller Indizes von Arrayeinträgen, die eine Bedingung erfüllen.

[4]:
```
from scipy.spatial import distance

# Co-Membership Matirx für K-Means
def get_comembership_matrix(X_valid: np.ndarray, train_centroids: np.ndarray)␣
↪-> np.ndarray:
    # Distanzmatrix zwischen Validierungsdaten und den Centroids aus dem␣
↪Trainingsdatenset
    dist_matrix = distance.cdist(X_valid, train_centroids)

    # Clusterzugehörigkeit ergibt sich durch die kleinste Entfernung
    pred_labels = np.argmin(dist_matrix, axis=1)
```

```
    # und die Co-Membership-Matrix erhalten wir aus der paarweisen␣
↪Übereinstimmung der Clusterzugehörigkeiten
    M: np.ndarray = (pred_labels[:, np.newaxis] == pred_labels[np.newaxis, :]).
↪astype(int)

    return M # (N_val \times N_val)-Matrix
```

(6) Implementieren Sie nun die Funktion `get_prediction_strength`, die eine Co-Membership Matrix `M`, die Clusterzugehörigkeiten auf dem Validierungsset `valid_labels` sowie die Anzahl K von Clustern entgegennimmt und die *Prediction Strength* `ps` zurückliefert.

- Rufen Sie sich dazu noch einmal mithilfe der Vorlesungsfolien in Erinnerung, wie die Prediction Strength definiert ist.
- Bei der Implementierung kann Ihnen dies hier hilfreich sein.

$$\mathrm{ps}(K) = \min_{j=1,\dots,K} \frac{1}{|A_j|(|A_j|-1)} \sum_{i,i' \in A_j,\ i \neq i'} M_{ii'}$$

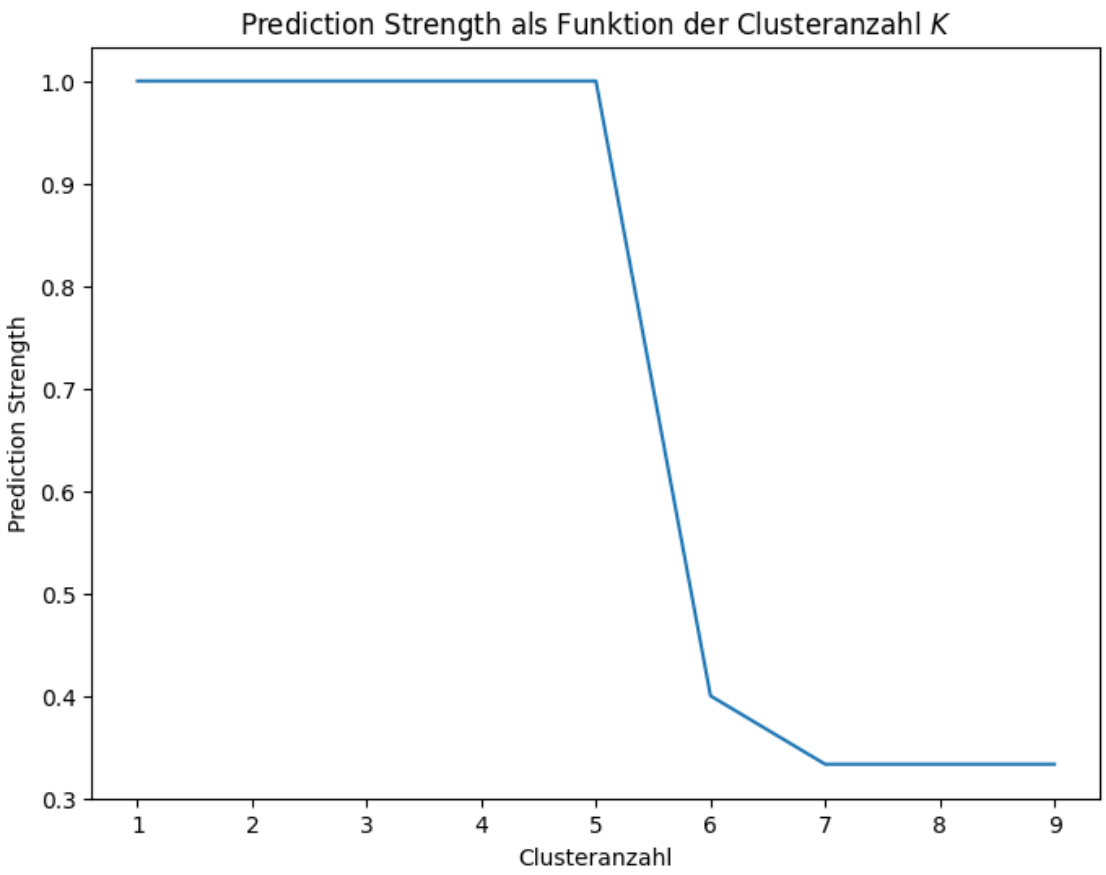
[5]:
```
def get_prediction_strength(M: np.ndarray, valid_labels: np.ndarray, K: int) ->␣
↪float:
    ps: float = np.inf

    for j in range(K):
        A_j = np.argwhere(valid_labels == j)[:, 0]

        # |A_j|
        N_A_j = A_j.size

        # Keine Division durch 0 oder negative Werte
        if N_A_j <= 1:
            continue

        # Submatrix für die aktuelle Clusterzuordnung
        M_j = M[A_j][:, A_j]

        # Vorhersagestärke für Cluster j:
        ps_j = (np.sum(M_j) - N_A_j) / (N_A_j * (N_A_j - 1))
        ps = min(ps, ps_j)

    return ps if ps != np.inf else 0.0


K = train_centroids.shape[0]
M = get_comembership_matrix(X_valid, train_centroids)
get_prediction_strength(M, valid_labels, K)
```

[5]:
```
np.float64(0.3333333333333333)
```

(7) Visualisieren Sie die *Prediction Strength* als Funktion der Clusteranzahl K für $K \in \{1, \dots 9\}$.

Hierzu müssen Sie für jeden Wert K ein Clustering auf den Trainingsdaten und auf den Validierungsdaten durchführen, die Co-Membership-Matrix M und anschließend die Prediction Strength bestimmen. Beschriften Sie die y-Achse mit "Prediction Strength" und die x-Achse mit "Clusteranzahl".

[6]:
```
def ps(X: np.ndarray, K: int) -> float:
    assert (K >= 1 and K <= X.size * 0.2),  "Anzahl der Cluster muss >= 1 sein␣
 ↪und darf die Anzahl der Datenpunkte (N_val) nicht übersteigen"

    # Train- / Valid-Split
    X_train, X_valid = model_selection.train_test_split(X, test_size=0.2,␣
 ↪random_state=40)

    # Erstelle Clustering
    kmeans_train = KMeans(n_clusters=K, random_state=42, n_init=10).fit(X_train)
    valid_labels = KMeans(n_clusters=K, random_state=42, n_init=10).
 ↪fit_predict(X_valid)
    train_centroids = kmeans_train.cluster_centers_

    # Berechne Co-Membership Matrix M
    M = get_comembership_matrix(X_valid, train_centroids)

    # Berechne Prediction Strength
    return get_prediction_strength(M, valid_labels, K)

k_range = range(1, 10)
plt.plot(k_range, [ps(X, K) for K in k_range])
plt.title("Prediction Strength als Funktion der Clusteranzahl $K$")
plt.xlabel("Clusteranzahl")
plt.ylabel("Prediction Strength")
plt.show()
```



(8) Lesen Sie die optimale Anzahl an Clustern anhand Ihrer Visualisierung aus Schritt (7) ab und geben Sie sie hier an. Die optimale Clusteranzahl ist die größte Zahl $K > 1$, für die die Prediction Strength noch den größten Wert annimmt.

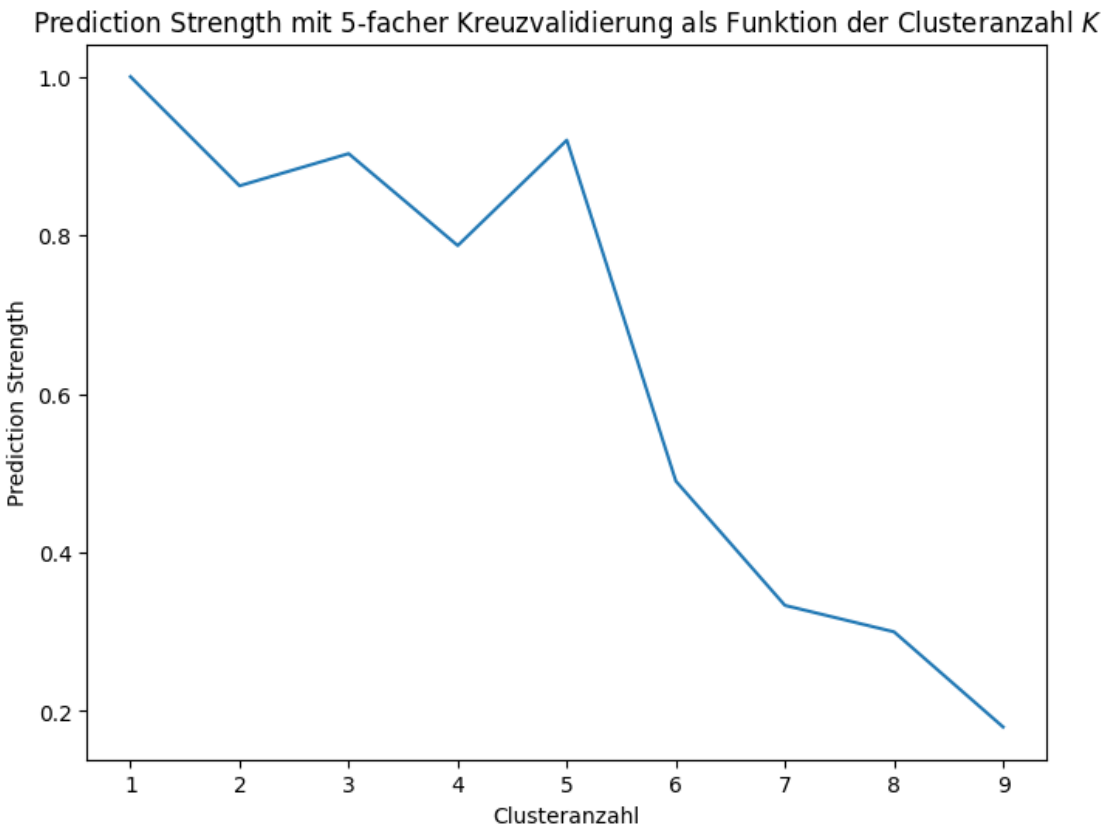
- Stimmt die von Ihnen ermittelte Clusteranzahl mit der "wahren" Anzahl der Cluster (vgl. Schritt (1)) in den Daten überein?

Die optimale Clusteranzahl ist bei $K = 5$, was deckungsgleich mit der erwarteten Clusteranzahl aus der Abbildung ist.

(9) [Optional] In der Praxis wird oft die Prediction Strength im Rahmen einer sogenannten Kreuzvalidierung ermittelt. Dabei werden alle Daten `X` in V disjunkte Mengen (sogenannte *Folds*) aufgespalten, wobei die Daten aus $(V-1)$ Folds als Trainingsset und dem verbliebenen Fold als Validierungsset angesehen wird. Durch Durchtauschen der Zuordnungen der Folds zu Trainings- und Validierungsset lassen sich so V mal die Prediction Strength bestimmen (jedes Fold dient einmal als Validierungsdatensatz). Die V Werte für die Prediction Strength werden anschließend arithmetisch gemittelt.

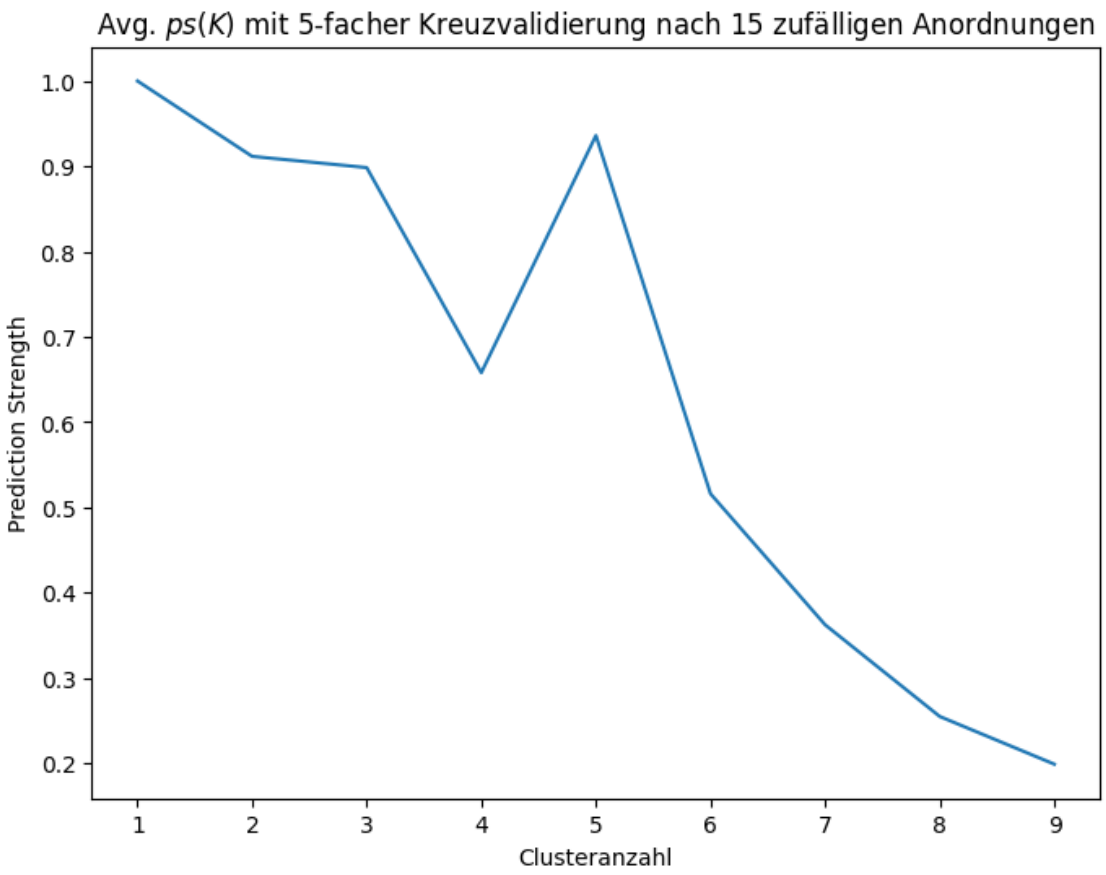
- Bestimmen Sie für $K \in \{1, \dots, 9\}$ mittels 5-facher Kreuzvalidierung die Prediction Strength und visualisieren Sie sie. Dabei kann Ihnen ggf. diese Funktion hilfreich sein.

[7]:
```
from sklearn.model_selection import KFold

def ps_KFold(X: np.ndarray, K: int, V: int, random_state=40) -> float:
    assert (K >= 1 and K <= X.size / 5),  "Anzahl der Cluster muss >= 1 sein␣
 ↪und darf die Anzahl der Datenpunkte (N_val) nicht übersteigen"

    kf = KFold(n_splits=V, shuffle=True, random_state=random_state)

    ps_sum: float = 0.0
    for train_index, test_index in kf.split(X):
        X_train, X_valid = X[train_index], X[test_index]

        # Erstelle Clustering
        kmeans_train = KMeans(n_clusters=K, random_state=42, n_init=10).
 ↪fit(X_train)
        valid_labels = KMeans(n_clusters=K, random_state=42, n_init=10).
 ↪fit_predict(X_valid)
        train_centroids = kmeans_train.cluster_centers_

        # Berechne Co-Membership Matrix M
        M = get_comembership_matrix(X_valid, train_centroids)

        # Berechne Prediction Strength
        ps_sum += get_prediction_strength(M, valid_labels, K)

    # Arithmetisches Mittel der Werte, die für die Prediction Strength␣
 ↪ermittelt wurden
    return ps_sum / V


k_range = range(1, 10)
plt.plot(k_range, [ps_KFold(X, K, 5, random_state=35) for K in k_range])
plt.title("Prediction Strength mit 5-facher Kreuzvalidierung als Funktion der␣
 ↪Clusteranzahl $K$")
plt.xlabel("Clusteranzahl")
plt.ylabel("Prediction Strength")
plt.show()
```



Frage: Der Prediction Strength Graph der Kreuzvalidierung variiert stark. Wäre es da nicht sinnvoll, diese mehrfach mit zufälligem Split durchzuführen, um bessere Ergebnisse zu erhalten?

[8]:
```
k_range = range(1, 10)
plt.plot(k_range, [np.array([ps_KFold(X, K, 5, random_state=i) for i in␣
 ↪range(15)]).mean() for K in k_range])
plt.title("Avg. $ps(K)$ mit 5-facher Kreuzvalidierung nach 15 zufälligen␣
 ↪Anordnungen")
plt.xlabel("Clusteranzahl")
plt.ylabel("Prediction Strength")
plt.show()
```


Avg. ps(K) mit 5-facher Kreuzvalidierung nach 15 zufälligen Anordnungen
Prediction Strength
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
1
2
3
4
5
6
7
8
9
Clusteranzahl