times2

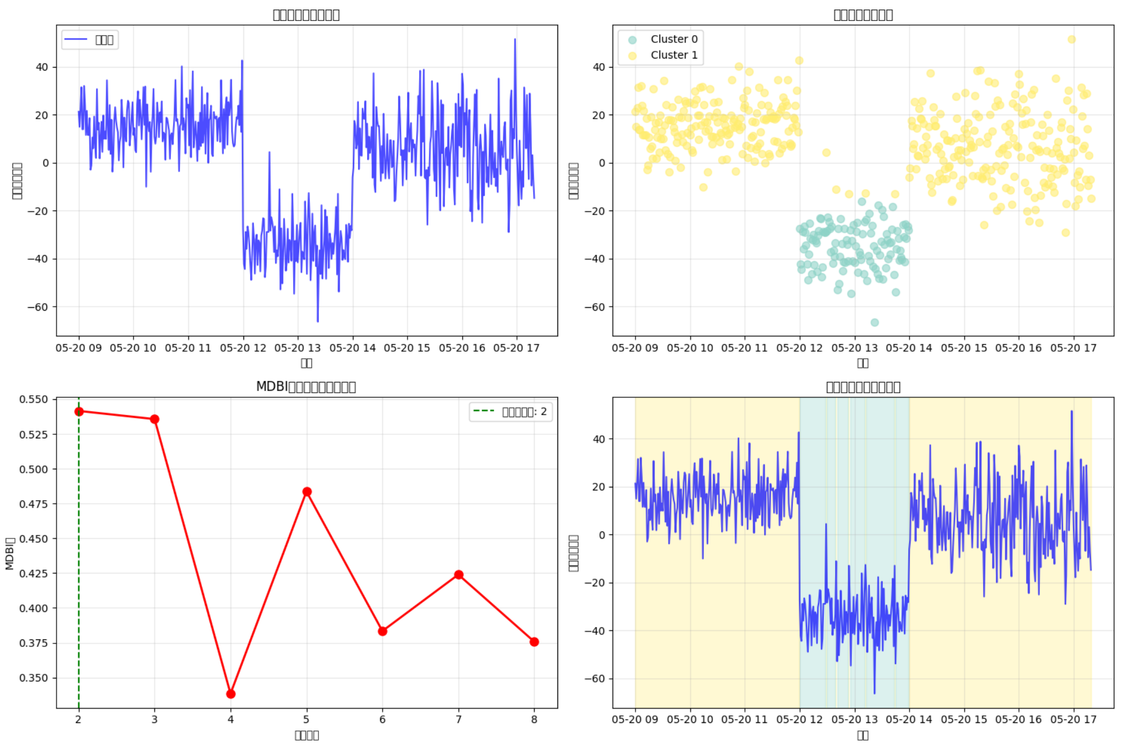
December 12, 2025

[5]:
```python
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from scipy.spatial.distance import mahalanobis
import matplotlib.pyplot as plt
from sklearn.metrics import davies_bouldin_score
import redis
import json
from datetime import datetime, timedelta
import warnings
warnings.filterwarnings('ignore')

class TimeSeriesClusteringPlatform:
    """
     G-G

    """

    def __init__(self, n_clusters_range=(2, 10)):
        """



         :
        n_clusters_range:
        """
        self.n_clusters_range = n_clusters_range
        self.optimal_n_clusters = None
        self.cluster_centers_ = None
        self.labels_ = None
        self.mdbis = []

    def load_data(self, data_path):
        """


            CSV
        """
        df = pd.read_csv(data_path)
        df['timestamp'] = pd.to_datetime(df['timestamp'])
        df.set_index('timestamp', inplace=True)
        return df

    def preprocess_data(self, df):
        """

         KICA   PCA
        """
        #
        df['hour'] = df.index.hour
        df['minute'] = df.index.minute
        df['weekday'] = df.index.weekday

        #  PCA
        pca = PCA(n_components=3)
        features = df[['hour', 'minute', 'weekday', 'visit_count']].values
        features_reduced = pca.fit_transform(features)

        return features_reduced

    def calculate_mdbi(self, X, labels):
        """
         MDBI  Modified Davies-Bouldin Index
        """
        try:
            #  Davies-Bouldin
            db_score = davies_bouldin_score(X, labels)
            #   MDBI   MDBI DBI
            mdbi = 1 / (1 + db_score) if db_score != 0 else 0
            return mdbi
        except:
            return 0

    def g_g_clustering(self, X, n_clusters, max_iter=100):
        """
        G-G

        """
        #
        gmm = GaussianMixture(n_components=n_clusters,
                              covariance_type='full',
                              max_iter=max_iter,
                              random_state=42)
        gmm.fit(X)
        #
        labels = gmm.predict(X)
        probabilities = gmm.predict_proba(X)

        #
        centers = gmm.means_

        return labels, probabilities, centers, gmm

    def find_optimal_clusters(self, X):
        """

        """
        best_mdbi = -np.inf
        best_n = 2

        for n in range(self.n_clusters_range[0], self.n_clusters_range[1] + 1):
            try:
                #  G-G
                labels, probabilities, centers, gmm = self.g_g_clustering(X, n)

                #  MDBI
                mdbi = self.calculate_mdbi(X, labels)
                self.mdbis.append((n, mdbi))

                print(f"  : {n}, MDBI : {mdbi:.4f}")

                #
                if mdbi > best_mdbi:
                    best_mdbi = mdbi
                    best_n = n
                    self.gmm = gmm
                    self.cluster_centers_ = centers

            except Exception as e:
                print(f"   {n}  : {e}")
                continue

        self.optimal_n_clusters = best_n
        print(f"\n   : {best_n},  MDBI : {best_mdbi:.4f}")
        return best_n

    def fuzzy_segmentation(self, X, timestamps):
        """

        """
        if not hasattr(self, 'gmm'):
            self.find_optimal_clusters(X)

        #
        probabilities = self.gmm.predict_proba(X)

        #
        segments = []
        current_segment = []
        current_cluster = None

        for i, (prob, ts) in enumerate(zip(probabilities, timestamps)):
            cluster = np.argmax(prob)

            if current_cluster is None:
                current_cluster = cluster
                current_segment.append((ts, cluster, prob[cluster]))
            elif cluster == current_cluster:
                current_segment.append((ts, cluster, prob[cluster]))
            else:
                #
                segments.append({
                    'start_time': current_segment[0][0],
                    'end_time': current_segment[-1][0],
                    'cluster': current_cluster,
                    'avg_probability': np.mean([p[2] for p in current_segment]),
                    'data_points': len(current_segment)
                })
                current_segment = [(ts, cluster, prob[cluster])]
                current_cluster = cluster

        #
        if current_segment:
            segments.append({
                'start_time': current_segment[0][0],
                'end_time': current_segment[-1][0],
                'cluster': current_cluster,
                'avg_probability': np.mean([p[2] for p in current_segment]),
                'data_points': len(current_segment)
            })

        return segments

    def plot_clustering_results(self, X, timestamps, segments):
        """

        """
        fig, axes = plt.subplots(2, 2, figsize=(15, 10))

        # 1.
        axes[0, 0].plot(timestamps, X[:, 0], 'b-', alpha=0.7, label='  ')
        axes[0, 0].set_title('     ')
        axes[0, 0].set_xlabel(' ')
        axes[0, 0].set_ylabel('   ')
        axes[0, 0].legend()
        axes[0, 0].grid(True, alpha=0.3)

        # 2.
        labels = self.gmm.predict(X)
        colors = plt.cm.Set3(np.linspace(0, 1, self.optimal_n_clusters))

        for cluster_id in range(self.optimal_n_clusters):
            mask = labels == cluster_id
            axes[0, 1].scatter(timestamps[mask], X[mask, 0],
                              c=[colors[cluster_id]],
                              label=f'Cluster {cluster_id}',
                              alpha=0.6, s=50)

        axes[0, 1].set_title('    ')
        axes[0, 1].set_xlabel(' ')
        axes[0, 1].set_ylabel('   ')
        axes[0, 1].legend()
        axes[0, 1].grid(True, alpha=0.3)

        # 3. MDBI
        clusters_n = [m[0] for m in self.mdbis]
        mdbi_values = [m[1] for m in self.mdbis]

        axes[1, 0].plot(clusters_n, mdbi_values, 'ro-', linewidth=2,␣
↪markersize=8)
        axes[1, 0].axvline(x=self.optimal_n_clusters, color='g', linestyle='--',
                          label=f'   : {self.optimal_n_clusters}')
        axes[1, 0].set_title('MDBI     ')
        axes[1, 0].set_xlabel('  ')
        axes[1, 0].set_ylabel('MDBI ')
        axes[1, 0].legend()
        axes[1, 0].grid(True, alpha=0.3)

        # 4.
        for seg in segments:
            color = colors[seg['cluster']]
            axes[1, 1].axvspan(seg['start_time'], seg['end_time'],
                              alpha=0.3, color=color,
                              label=f'Cluster {seg["cluster"]}' if␣
↪seg['cluster'] not in
                              [s['cluster'] for s in segments[:segments.
↪index(seg)]] else "")

        axes[1, 1].plot(timestamps, X[:, 0], 'b-', alpha=0.7)
        axes[1, 1].set_title('      ')
        axes[1, 1].set_xlabel(' ')
        axes[1, 1].set_ylabel('    ')
        axes[1, 1].grid(True, alpha=0.3)

        plt.tight_layout()
        plt.show()

        #
        print("\n    :")
        print("-" * 80)
        for i, seg in enumerate(segments):
            print(f"  {i+1}:")
            print(f"    : {seg['start_time']}")
            print(f"    : {seg['end_time']}")
            print(f"    : {seg['cluster']}")
            print(f"    : {seg['avg_probability']:.4f}")
            print(f"    : {seg['data_points']}")
            print("-" * 80)


class DistributedPlatform:
    """

    """

    def __init__(self):
        self.redis_client = None
        self.nodes = []
        self.load_history = []

    def init_redis(self, host='localhost', port=6379, db=0):
        """ Redis """
        try:
            self.redis_client = redis.Redis(host=host, port=port, db=db)
            print("Redis    ")
        except:
            print(" : Redis       ")
            self.redis_client = None
            self.cache = {}
    def get_from_cache(self, key):
        """    """
        if self.redis_client:
            try:
                data = self.redis_client.get(key)
                if data:
                    return json.loads(data)
            except:
                return None
        else:
            return self.cache.get(key)

    def set_to_cache(self, key, value, expire=3600):
        """    """
        if self.redis_client:
            try:
                self.redis_client.setex(key, expire, json.dumps(value))
            except:
                pass
        else:
            self.cache[key] = value

    def dynamic_scaling(self, current_load, segments, threshold_high=0.8,␣
↪threshold_low=0.3):
        """



        """
        recommendations = []

        for seg in segments:
            if seg['avg_probability'] > threshold_high:
                #
                recommendations.append({
                    'time_period': f"{seg['start_time']} - {seg['end_time']}",
                    'recommendation': '    ',
                    'current_load': seg['avg_probability'],
                    'suggested_nodes': min(5, int(seg['avg_probability'] * 10))
                })
            elif seg['avg_probability'] < threshold_low:
                #
                recommendations.append({
                    'time_period': f"{seg['start_time']} - {seg['end_time']}",
                    'recommendation': '    ',
                    'current_load': seg['avg_probability'],
                    'suggested_nodes': max(1, int(seg['avg_probability'] * 3))
                })
```

```python
            else:
                recommendations.append({
                    'time_period': f"{seg['start_time']} - {seg['end_time']}",
                    'recommendation': '    ',
                    'current_load': seg['avg_probability'],
                    'suggested_nodes': 3  #
                })

        return recommendations


#
def demo_time_series_clustering():
    """    """
    print("=" * 80)
    print("           -   ")
    print("=" * 80)

    # 1.
    np.random.seed(42)
    n_points = 500

    #
    timestamps = pd.date_range('2023-05-20 09:00', periods=n_points,␣
 ↪freq='1min')

    #
    visit_pattern = np.zeros(n_points)

    #    (9:00-12:00)
    morning_mask = (timestamps.hour >= 9) & (timestamps.hour < 12)
    visit_pattern[morning_mask] = np.random.normal(0.8, 0.1, morning_mask.sum())

    #    (12:00-14:00)
    noon_mask = (timestamps.hour >= 12) & (timestamps.hour < 14)
    visit_pattern[noon_mask] = np.random.normal(0.3, 0.1, noon_mask.sum())

    #    (14:00-18:00)
    afternoon_mask = (timestamps.hour >= 14) & (timestamps.hour < 18)
    visit_pattern[afternoon_mask] = np.random.normal(0.7, 0.15, afternoon_mask.
 ↪sum())

    #
    df = pd.DataFrame({
        'timestamp': timestamps,
        'visit_count': visit_pattern * 100  #
    })
    # 2.
    platform = TimeSeriesClusteringPlatform(n_clusters_range=(2, 8))

    # 3.
    print("\n1.    ...")
    features = platform.preprocess_data(df.set_index('timestamp'))

    # 4.
    print("\n2.      ...")
    optimal_n = platform.find_optimal_clusters(features)

    # 5.
    print("\n3.       ...")
    segments = platform.fuzzy_segmentation(features, timestamps)

    # 6.
    print("\n4.     ...")
    platform.plot_clustering_results(features, timestamps, segments)

    # 7.
    print("\n5.        :")
    print("-" * 80)

    distributed_platform = DistributedPlatform()
    distributed_platform.init_redis()

    recommendations = distributed_platform.dynamic_scaling(0.5, segments)

    for rec in recommendations:
        print(f"  : {rec['time_period']}")
        print(f"    : {rec['current_load']:.2%}")
        print(f"    : {rec['recommendation']}")
        print(f"     : {rec['suggested_nodes']}")
        print("-" * 40)

    return platform, segments, recommendations


#
if __name__ == "__main__":
    demo_time_series_clustering()
```

```
================================================================================
          -
================================================================================

1.    …
```

```
2.     …
  : 2, MDBI : 0.5415
  : 3, MDBI : 0.5357
  : 4, MDBI : 0.3384
  : 5, MDBI : 0.4836
  : 6, MDBI : 0.3834
  : 7, MDBI : 0.4239
  : 8, MDBI : 0.3757

   : 2,  MDBI : 0.5415

3.      …

4.     …
```



```
    :
--------------------------------------------------------------------------------
  1:
    : 2023-05-20 09:00:00
    : 2023-05-20 11:59:00
    : 1
    : 1.0000
    : 180
--------------------------------------------------------------------------------
  2:
    : 2023-05-20 12:00:00
    : 2023-05-20 12:28:00
    : 0
    : 0.9945
    : 29
--------------------------------------------------------------------------------
  3:
    : 2023-05-20 12:29:00
    : 2023-05-20 12:29:00
    : 1
    : 0.9999
    : 1
--------------------------------------------------------------------------------
  4:
    : 2023-05-20 12:30:00
    : 2023-05-20 12:39:00
    : 0
    : 0.9899
    : 10
--------------------------------------------------------------------------------
  5:
    : 2023-05-20 12:40:00
    : 2023-05-20 12:40:00
    : 1
    : 0.8156
    : 1
--------------------------------------------------------------------------------
  6:
    : 2023-05-20 12:41:00
    : 2023-05-20 12:53:00
    : 0
    : 0.9950
    : 13
--------------------------------------------------------------------------------
  7:
    : 2023-05-20 12:54:00
    : 2023-05-20 12:54:00
    : 1
    : 0.6677
    : 1
--------------------------------------------------------------------------------
  8:
    : 2023-05-20 12:55:00
    : 2023-05-20 13:11:00
    : 0
    : 0.9656
    : 17
--------------------------------------------------------------------------------
  9:
    : 2023-05-20 13:12:00
    : 2023-05-20 13:12:00
    : 1
    : 0.7841
    : 1
--------------------------------------------------------------------------------
  10:
    : 2023-05-20 13:13:00
    : 2023-05-20 13:43:00
    : 0
    : 0.9662
    : 31
--------------------------------------------------------------------------------
  11:
    : 2023-05-20 13:44:00
    : 2023-05-20 13:44:00
    : 1
    : 0.7355
    : 1
--------------------------------------------------------------------------------
  12:
    : 2023-05-20 13:45:00
    : 2023-05-20 13:59:00
    : 0
    : 0.9925
    : 15
--------------------------------------------------------------------------------
  13:
    : 2023-05-20 14:00:00
    : 2023-05-20 17:19:00
    : 1
    : 0.9998
    : 200
--------------------------------------------------------------------------------

5.       :
--------------------------------------------------------------------------------
Redis
  : 2023-05-20 09:00:00 - 2023-05-20 11:59:00
    : 100.00%
    :
     : 5
----------------------------------------
  : 2023-05-20 12:00:00 - 2023-05-20 12:28:00
```

```
    : 99.45%
    :
     : 5
----------------------------------------
  : 2023-05-20 12:29:00 - 2023-05-20 12:29:00
    : 99.99%
    :
     : 5
----------------------------------------
  : 2023-05-20 12:30:00 - 2023-05-20 12:39:00
    : 98.99%
    :
     : 5
----------------------------------------
  : 2023-05-20 12:40:00 - 2023-05-20 12:40:00
    : 81.56%
    :
     : 5
----------------------------------------
  : 2023-05-20 12:41:00 - 2023-05-20 12:53:00
    : 99.50%
    :
     : 5
----------------------------------------
  : 2023-05-20 12:54:00 - 2023-05-20 12:54:00
    : 66.77%
    :
     : 3
----------------------------------------
  : 2023-05-20 12:55:00 - 2023-05-20 13:11:00
    : 96.56%
    :
     : 5
----------------------------------------
  : 2023-05-20 13:12:00 - 2023-05-20 13:12:00
    : 78.41%
    :
     : 3
----------------------------------------
  : 2023-05-20 13:13:00 - 2023-05-20 13:43:00
    : 96.62%
    :
     : 5
----------------------------------------
  : 2023-05-20 13:44:00 - 2023-05-20 13:44:00
    : 73.55%
    :
     : 3
```

```
----------------------------------------
  : 2023-05-20 13:45:00 - 2023-05-20 13:59:00
     : 99.25%
     :
      : 5
----------------------------------------
  : 2023-05-20 14:00:00 - 2023-05-20 17:19:00
     : 99.98%
     :
      : 5
----------------------------------------
```

[ ]:

```
```