User Guide: Model

Using model.json for Multi-Resolution Clustering Scoring

Document type	Operational user guide
Model file	model.json
Use case	Deterministic cluster assignment
Supported resolutions	K = 2 to K = 16

This guide explains how to use the model.json file to assign cluster labels to new observations. It is intended for data scientists, developers, and operational teams integrating the model into an application or scoring pipeline.

1. Purpose

The model.json file contains all parameters required to execute a pre-trained clustering model without retraining it. It assigns each input observation to one or more clusters and can be deployed in any environment capable of reading JSON files.

The file is self-contained: it includes the expected input features, scaling parameters, cluster centroids, and metadata required for deterministic scoring.

2. File Structure

Section	Description
feature_names	List of input variables expected by the model.
scale_applied	Indicates whether standardization must be applied before scoring.
scaler.mean	Feature means used during training.
scaler.scale	Feature scaling values used during training.
centroids	Cluster centroids for each supported value of K.

The model supports multiple clustering resolutions from K = 2 to K = 16. Each value of K corresponds to a different level of segmentation detail.

3. Input Requirements

The input dataset must contain exactly the features listed in model["feature_names"]. For this model, the expected features are pixel_0_0, pixel_0_1, ..., pixel_7_7, representing 64 numerical variables.

Mandatory input conditions

- Feature names must match exactly.
- All required variables must be present.
- The feature order must follow feature_names.
- Values must be numeric.
- Missing values are not allowed unless handled before scoring.

4. Data Preprocessing

The model was trained using standardized input data. Before assigning clusters, apply the scaling parameters included in model.json:

```
X_scaled = (X - mean) / scale
```

The mean vector comes from model["scaler"]["mean"], and the scale vector comes from model["scaler"]["scale"]. Skipping this step will lead to incorrect cluster assignments.

5. Cluster Assignment Process

For a chosen value of K, the scoring process is as follows:

- Load the centroids associated with K.
- Standardize the input observation using the model scaler.
- Compute the Euclidean distance between the observation and each centroid.
- Select the closest centroid.
- Return the centroid index as the cluster label.

Cluster labels are zero-based: for a given K, labels range from 0 to K-1.

6. Output

The model may generate multiple cluster assignments for the same observation, one for each supported resolution: label_K2, label_K3, ..., label_K16.

This multi-resolution output lets the user choose between broad segmentation with smaller K values and more granular segmentation with larger K values.

7. Interpretation Guidelines

- Cluster labels are technical identifiers and do not have intrinsic business meaning.
- A cluster label from one K value must not be compared directly with the same label number from another K value.
- Higher K values produce finer segmentations.
- Lower K values produce broader groupings.

8. Python Example – Single Prediction

```
import json
import numpy as np

# Load model
with open("model.json", "r", encoding="utf-8") as f:
    model = json.load(f)

feature_names = model["feature_names"]
means = np.array(model["scaler"]["mean"], dtype=float)
scales = np.array(model["scaler"]["scale"], dtype=float)

# Example observation: all 64 features must be provided
row = {
    "pixel_0_0": 0,
    "pixel_0_1": 0,
    "pixel_0_2": 5,
    # ... complete all features through pixel_7_7
}

# Build feature vector in the exact expected order
x = np.array([row[col] for col in feature_names], dtype=float)
```

```
# Apply scaling
x_scaled = (x - means) / scales

# Select clustering resolution
K = 8
centroids = np.array(model["centroids"][str(K)], dtype=float)

# Compute Euclidean distances and assign cluster
distances = np.linalg.norm(centroids - x_scaled, axis=1)
cluster_label = int(np.argmin(distances))

print(f"Assigned cluster for K={K}: {cluster_label}")
```

9. Python Example – Batch Prediction with Pandas

```
import json
import numpy as np
import pandas as pd

# df must contain all required feature columns
with open("model.json", "r", encoding="utf-8") as f:
    model = json.load(f)

features = model["feature_names"]
means = np.array(model["scaler"]["mean"], dtype=float)
scales = np.array(model["scaler"]["scale"], dtype=float)

# Validate feature availability
missing = [col for col in features if col not in df.columns]
if missing:
    raise ValueError(f"Missing required features: {missing}")

# Extract and scale features
X = df[features].values.astype(float)
X_scaled = (X - means) / scales

# Predict labels for one selected K
K = 8
centroids = np.array(model["centroids"][str(K)], dtype=float)
distances = np.linalg.norm(X_scaled[:, None, :] - centroids[None, :, :], axis=2)
df[f"label_K{K}"] = np.argmin(distances, axis=1)

# Optional: predict all supported K values
for K in range(2, 17):
    if str(K) in model["centroids"]:
        centroids = np.array(model["centroids"][str(K)], dtype=float)
        distances = np.linalg.norm(X_scaled[:, None, :] - centroids[None, :, :], axis=2)
        df[f"label_K{K}"] = np.argmin(distances, axis=1)
```

10. Best Practices

Recommended

- Validate that all required features are present before scoring.
- Use the feature order provided by model["feature_names"].

- Apply the scaling parameters included in the model file.
- Store the model version or job ID alongside predictions.
- Keep both the input record and the assigned cluster label for auditability.

Avoid

- Do not rename or reorder features manually.
- Do not skip standardization.
- Do not compare cluster labels across different K values.
- Do not convert zero-based labels into one-based labels unless this is clearly documented downstream.

11. Audit and Compliance

The model.json file provides a transparent and reproducible scoring mechanism. It contains the feature definitions, scaling parameters, centroids, and deterministic assignment logic required to reproduce each cluster decision.

For operational traceability, store the applied K value, input feature vector, model file version or job identifier, and resulting label for each scored observation.