2_clustering_and_pca

August 7, 2025

Connected to cpu (Python 3.13.1)

Unsupervised Learning: Clustering and PCA for Materials Insight

1. Objective

This notebook explores the underlying structure of the materials data using unsupervised learning. By applying Principal Component Analysis (PCA) for dimensionality reduction and K-Means clustering, we aim to answer a key question: **Can we identify distinct, chemically meaningful groups of materials, and can these groups be used as engineered features to improve our predictive models?**

This process is critical for two reasons: 1. It provides valuable domain insights into the natural families of materials within our dataset. 2. It allows us to rigorously test a common feature engineering hypothesis in a data-driven way.

2. Key Steps & Findings

- **Data Preparation:** We apply a log transformation and standardization to prepare the skewed feature data for distance-based algorithms like PCA and K-Means.
- **Dimensionality Reduction:** PCA is used to reduce the feature space from over 100 features to a much smaller set of components while retaining 95% of the data's variance.
- **Clustering & Analysis:** K-Means clustering is performed on the PCA-reduced data. We find that the resulting clusters correspond well to known chemical families (e.g., oxides, alloys).
- **The Modeling Outcome:** As will be demonstrated quantitatively in Notebook #4, while these clusters are insightful, they **do not ultimately improve the predictive performance** of the final XGBoost model. This notebook documents the valuable analytical process that justifies the decision to exclude them.

[ ]:
```
# --- Imports and Setup ---
import os
import sys

# Set the OMP_NUM_THREADS environment variable to avoid a memory leak on␣
 ↪Windows with MKL.
# This is a known issue with scikit-learn's KMeans and is the recommended␣
 ↪workaround.
os.environ['OMP_NUM_THREADS'] = '3'

import pandas as pd
import numpy as np
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.impute import SimpleImputer
import plotly.graph_objects as go

# Ensure src directory is added to sys.path for modular imports
try:
    # Assumes the script is in the 'notebooks' directory
    PROJECT_ROOT = os.path.abspath(os.path.join(os.path.dirname(__file__), '..
 ↪'))
except NameError:
    # Fallback for interactive environments (Jupyter, VSCode)
    PROJECT_ROOT = os.path.abspath(os.path.join(os.getcwd()))

# Add the 'src' directory to the Python path
SRC_PATH = os.path.join(PROJECT_ROOT, 'src')
if SRC_PATH not in sys.path:
    sys.path.insert(0, SRC_PATH)
# Re-import modules after fixing sys.path
from viz import (
    plot_clusters,
    plot_pca_variance,
    plot_cluster_crystal_structure,
    plot_cluster_chemistry_distribution,
    plot_structure_by_chemistry,
    plot_pca_material_class,
    plot_density_by_cluster
)
from modeling import find_elbow_point, get_inertia_values
from viz import plot_elbow_inertia_with_marker
from utils import (
    setup_environment, load_or_process_dataframe, save_plot, log_and_print,
    prepare_data_for_modeling, style_df
)

# --- Setup environment and paths ---
setup_environment()
PLOTS_DIR = os.path.join(PROJECT_ROOT, 'plots', '2_clustering_and_pca')
os.makedirs(PLOTS_DIR, exist_ok=True)

CACHE_PATH = os.path.join(PROJECT_ROOT, 'data', 'processed', 'featurized.
 ↪parquet')
PCA_VAR_PATH = os.path.join(PLOTS_DIR, 'clustering_pca_variance.pdf')
SILHOUETTE_PATH = os.path.join(PLOTS_DIR, 'clustering_silhouette_scores.pdf')
CLUSTER_SCATTER_PATH = os.path.join(PLOTS_DIR, 'clustering_cluster_scatter.pdf')
```

```
CLUSTER_SCATTER_3D_PATH = os.path.join(PLOTS_DIR,␣
 ↪'clustering_cluster_scatter_3d.pdf')
CHEM_DIST_PATH = os.path.join(PLOTS_DIR, 'clustering_chemistry_distribution.
 ↪pdf')
DENSITY_DIST_PATH = os.path.join(PLOTS_DIR, 'clustering_density_distribution.
 ↪pdf')
MATERIAL_CLASS_PATH = os.path.join(PLOTS_DIR, 'clustering_material_class_pca.
 ↪pdf')
CRYSTAL_STRUCT_PATH = os.path.join(PLOTS_DIR,␣
 ↪'clustering_cluster_crystal_structure.pdf')
STRUCT_CHEM_PATH = os.path.join(PLOTS_DIR, 'clustering_structure_by_chemistry.
 ↪pdf')
CLUSTER_SUMMARY_PATH = os.path.join(PLOTS_DIR, 'clustering_summary_table.csv')


# --- Load featurized data using robust utility ---
df = load_or_process_dataframe(cache_path=CACHE_PATH, project_root=PROJECT_ROOT)
log_and_print(f"Featurized dataframe shape: {df.shape}")
style_df(df.head())
```

```
Loaded cached DataFrame from c:\Users\angel\Thermal-Conductivity-
ML\data\processed\featurized.parquet (parquet)
Loaded cached DataFrame from c:\Users\angel\Thermal-Conductivity-
ML\data\processed\featurized.parquet
Featurized dataframe shape: (757, 177)
```

[ ]: <pandas.io.formats.style.Styler at 0x1d1e6b2e3c0>

1. Prepare Data for PCA and Clustering

Based on our EDA, many features are right-skewed. To improve the performance of PCA and K-Means, which are sensitive to feature scale and distribution, we will: 1. **Log-transform** the numeric features to reduce skewness. We use `np.log1p` which handles zero values gracefully. 2. **Standardize** the features using `StandardScaler` to give them zero mean and unit variance.

[ ]:
```
X, y = prepare_data_for_modeling(df, target_col='thermal_conductivity')

# Log-transform the features to handle skewness
X_log = np.log1p(X)

# Impute any NaNs that may have been introduced by the log transform
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X_log)

# Scale the log-transformed and imputed features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)
```

```
log_and_print("Feature matrix prepared for clustering (log-transformed,␣
 ↪imputed, and scaled).")
log_and_print(f"Shape of the final feature matrix: {X_scaled.shape}")
```

```
Imputing NaN values in columns: ['mp_density', 'mp_volume', 'mp_band_gap',
'mp_energy_above_hull', 'jarvis_band_gap', 'jarvis_formation_energy', 'density']
Feature matrix prepared for clustering (log-transformed, imputed, and scaled).
Shape of the final feature matrix: (757, 154)
```

```
c:\Users\angel\.ai-navigator\micromamba\envs\cpu\Lib\site-
packages\pandas\core\internals\blocks.py:393: RuntimeWarning:

invalid value encountered in log1p
```

2. Dimensionality Reduction with PCA

We apply Principal Component Analysis (PCA) to reduce the number of features while retaining most of the information (variance) in the data. We set a target of explaining 95% of the total variance.

[ ]:
```
# Initialize PCA to retain 95% of the variance
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)

n_components_retained = X_pca.shape[1]
log_and_print(f'Original features: {X.shape[1]}')
log_and_print(f'PCA components retained to explain 95% variance:␣
 ↪{n_components_retained}')

# Plot the cumulative explained variance
fig_pca_variance = plot_pca_variance(pca)
save_plot(fig_pca_variance, PCA_VAR_PATH)
fig_pca_variance.show()
```

```
Original features: 154
PCA components retained to explain 95% variance: 29
Plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\2_clustering_and_pca\clustering_pca_variance.pdf
```

Insight: The plot above shows that we can reduce the feature space significantly while still capturing the vast majority of the data's structure. This helps in building more robust and efficient clustering models by filtering out noise.

3. K-Means Clustering

Determining the Optimal Number of Clusters

We use the **silhouette score** to determine the optimal number of clusters (`k`) and compare it against the elbow method for additional context. The silhouette score measures how similar an object is to its own cluster compared to other clusters. A higher score indicates better-defined clusters.

[ ]:
```
# Convert range to list for plotly compatibility
k_range = list(range(2, 30))

# Calculate inertia values for the range of k (elbow method)
inertia_values = get_inertia_values(X_pca, k_range)
optimal_k_elbow = int(find_elbow_point(inertia_values, k_range))
log_and_print(f"Optimal k determined using the elbow method: {optimal_k_elbow}")

fig_elbow_inertia = plot_elbow_inertia_with_marker(inertia_values, k_range,␣
 ↪optimal_k_elbow)
save_plot(fig_elbow_inertia, os.path.join(PLOTS_DIR,␣
 ↪'elbow_method_inertia_with_marker.pdf'))
fig_elbow_inertia.show()

# Calculate silhouette scores for the range of k
silhouette_scores = []
for k in k_range:
    kmeans_tmp = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans_tmp.fit_predict(X_pca)
    silhouette_scores.append(silhouette_score(X_pca, labels))
optimal_k_silhouette = k_range[int(np.argmax(silhouette_scores))]
log_and_print(f"Optimal k determined using silhouette score:␣
 ↪{optimal_k_silhouette}")

fig_silhouette = go.Figure()
fig_silhouette.add_trace(go.Scatter(x=k_range, y=silhouette_scores,␣
 ↪mode='lines+markers', name='Silhouette Score'))
fig_silhouette.add_trace(go.Scatter(
    x=[optimal_k_silhouette],
    y=[max(silhouette_scores)],
    mode='markers',
    marker=dict(color='red', size=12, symbol='x'),
    name=f'Optimal k = {optimal_k_silhouette}'
))
fig_silhouette.update_layout(
    title='Silhouette Scores for Optimal k',
    xaxis_title='Number of Clusters (k)',
    yaxis_title='Silhouette Score',
    showlegend=True
)
save_plot(fig_silhouette, SILHOUETTE_PATH)
fig_silhouette.show()
```

```
Optimal k determined using the elbow method: 13
Plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\2_clustering_and_pca\elbow_method_inertia_with_marker.pdf

Optimal k determined using silhouette score: 29
```

```
Plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\2_clustering_and_pca\clustering_silhouette_scores.pdf
```

Fitting the K-Means Model

Based on the silhouette analysis and elbow method, we select the optimal `k` and fit the K-Means algorithm to the PCA-transformed data.

[ ]:
```
# Choose the smaller of the two optimal k values for a more conservative␣
 ↪cluster count.
# This provides a robust heuristic for selecting k.
if optimal_k_silhouette < optimal_k_elbow:
    log_and_print(f"Silhouette score suggests k={optimal_k_silhouette}, which␣
 ↪is smaller than the elbow method's k={optimal_k_elbow}.")
    N_CLUSTERS = optimal_k_silhouette
else:
    log_and_print(f"Elbow method suggests k={optimal_k_elbow}, which is smaller␣
 ↪than or equal to the silhouette score's k={optimal_k_silhouette}.")
    N_CLUSTERS = optimal_k_elbow

log_and_print(f"Final selection: Using {N_CLUSTERS} clusters for K-Means.")

# Ensure N_CLUSTERS is a valid integer
if not isinstance(N_CLUSTERS, int) or N_CLUSTERS <= 1:
    raise ValueError(f"Invalid number of clusters determined: {N_CLUSTERS}.␣
 ↪Please check calculations.")

# Fit KMeans on the PCA-reduced data for efficiency and robustness
kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=42, n_init=10)
cluster_labels = kmeans.fit_predict(X_pca)

# Add cluster labels back to the original dataframe for analysis
df['cluster_label'] = kmeans.predict(pca.transform(X_scaled))
```

```
Elbow method suggests k=13, which is smaller than or equal to the silhouette
score's k=29.
Final selection: Using 13 clusters for K-Means.
```

4. Visualize and Analyze Clusters

Cluster Visualization in PCA Space

We can visualize the clusters by plotting them in the space of the first few principal components. This gives us a qualitative sense of how well-separated the clusters are.

[ ]:
```
# Visualize clusters in 2D and 3D PCA space
fig_2d, fig_3d = plot_clusters(X_pca, cluster_labels, df.index, N_CLUSTERS)
save_plot(fig_2d, CLUSTER_SCATTER_PATH)
save_plot(fig_3d, CLUSTER_SCATTER_3D_PATH)
fig_2d.show()
```

```
fig_3d.show()
```

```
Plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\2_clustering_and_pca\clustering_cluster_scatter.pdf
Plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\2_clustering_and_pca\clustering_cluster_scatter_3d.pdf
```

Cluster Composition Analysis

Now we dig deeper to understand what defines each cluster. We analyze the distribution of key categorical features like **crystal structure** and **chemical class** within each identified group.

[ ]:
```
# Plot distribution of crystal structure and chemistry within clusters
fig_struct = plot_cluster_crystal_structure(df)
save_plot(fig_struct, CRYSTAL_STRUCT_PATH)
fig_struct.show()

fig_chem = plot_cluster_chemistry_distribution(df)
save_plot(fig_chem, CHEM_DIST_PATH)
fig_chem.show()

fig_combo = plot_structure_by_chemistry(df)
save_plot(fig_combo, STRUCT_CHEM_PATH)
fig_combo.show()
```

```
Plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\2_clustering_and_pca\clustering_cluster_crystal_structure.pdf
```

```
Plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\2_clustering_and_pca\clustering_chemistry_distribution.pdf
```

```
Plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\2_clustering_and_pca\clustering_structure_by_chemistry.pdf
```

Domain Insight: Interpreting the Cluster Composition

The plots above provide strong evidence that the unsupervised learning has discovered chemically meaningful patterns in the data. Here are the key takeaways:

- **Chemistry is the Primary Separator:** The clusters are clearly delineated by chemical family. For instance, **Clusters 1 and 11 are entirely composed of oxides.** This is a powerful validation that the PCA, driven by elemental features, is capturing fundamental chemical differences. Oxygen's unique properties create a strong, distinct signal that separates these materials from others.
- **Crystal Structure Provides Finer Detail:** Within these broad chemical groups, crystal structure acts as a secondary organizing principle. The concentration of **orthorhombic oxides** within Cluster 11 is a perfect example. This shows that the model can distinguish structural variations, but only after the primary chemical grouping is established.
- **"Mixed" Clusters Reflect Real-World Complexity:** The fact that many clusters contain a mix of chemistries and structures is not a flaw, but an important finding. It reflects the reality of materials science, where properties exist on a continuum. These clusters group

materials that are "similar" based on their most dominant features, even if they have different categorical labels.

This analysis provides a data-driven justification for why these engineered cluster features, while insightful, do not ultimately improve the predictive model's performance. The model likely gains more value from the continuous, underlying elemental features than from these broad, somewhat mixed categorical labels.

Quantitative Cluster Summary

To complement the visualizations, we'll create a summary table that shows the mean `thermal_conductivity` and `density` for each cluster. This helps us quantitatively characterize and compare the groups.

[ ]:
```
# Analyze density distribution and create a summary table
fig_density = plot_density_by_cluster(df)
save_plot(fig_density, DENSITY_DIST_PATH)
fig_density.show()

# Create a summary dataframe
cluster_summary = df.groupby('cluster_label').agg(
    mean_thermal_conductivity=('thermal_conductivity', 'mean'),
    mean_density=('density', 'mean'),
    count=('formula', 'count')
).reset_index()

log_and_print("\nCluster Summary Table:")
cluster_summary.to_csv(CLUSTER_SUMMARY_PATH, index=False)
style_df(cluster_summary)
```

```
Plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\2_clustering_and_pca\clustering_density_distribution.pdf


Cluster Summary Table:
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x1d1e6c56490>
```

5. Synthesis: Material Classes in PCA Space

Finally, we bring it all together by overlaying our derived material classes (based on chemistry) onto the PCA plot. This helps to visually confirm whether our unsupervised clustering has identified chemically meaningful groups.

[ ]:
```
# Plot PCA scatter with material class information
analysis_df = df[['cluster_label', 'chemistry', 'crystal_structure']]
fig_material = plot_pca_material_class(X_scaled, analysis_df)
save_plot(fig_material, MATERIAL_CLASS_PATH)
fig_material.show()
```

```
Plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\2_clustering_and_pca\clustering_material_class_pca.pdf
```

Using Clusters as Features in Predictive Modeling The clusters identified in this notebook will be used as categorical features in the next step of the workflow, `3_modeling_and_feature_selection`. By incorporating these cluster labels, we aim to enhance the predictive power of our models by allowing them to learn distinct relationships for different material groups. ## Benefits of Using Clusters as Features: - **Improved Model Accuracy:** Clusters provide additional context about the material groupings, which can help the model make more informed predictions. - **Feature Engineering:** The cluster labels act as a powerful categorical feature, capturing high-level patterns in the data. - **Domain-Specific Insights:** Using clusters as features allows us to integrate domain knowledge into the modeling process, making the predictions more interpretable and actionable. In the next notebook, we will explore how these clusters interact with other features and evaluate their impact on model performance.

6. Conclusion & Decision on Cluster Features

Summary of Findings

This analysis successfully demonstrated that unsupervised learning can uncover meaningful, hidden structure within the materials dataset.

1. **Meaningful Groups Identified:** K-Means clustering, applied to the PCA-reduced feature space, successfully partitioned the materials into distinct groups.
2. **Clusters are Chemically Relevant:** Through visualization and analysis, we confirmed that these clusters align well with known chemical families (e.g., oxides, intermetallics), providing valuable domain insight into the data landscape.

The Final, Data-Driven Decision

The primary hypothesis of this notebook was to test whether these engineered cluster labels could improve the performance of our predictive models.

As will be shown rigorously in **Notebook #4**, adding the cluster labels as a categorical feature did **not** result in a statistically significant improvement to the final XGBoost model's predictive accuracy.

Therefore, based on this evidence, we will proceed without including the cluster labels in our final model. This is a critical and realistic outcome in a data science project. The ability to rigorously test and discard features that do not add value is just as important as creating them, and it ensures our final model is both powerful and parsimonious.

Next Steps

The workflow now proceeds to the next notebook, where we will perform a comprehensive comparison of several machine learning models.

Next Notebook: `3_model_comparison.py`