PCA and Regression Analysis for Binary Phase Images

Koka Balaji Vigneshwara
Matriculation Number: 71624

May 2025

Abstract

This report presents an in-depth analysis of binary phase images using machine learning techniques. The objectives include computing white phase area fractions, applying Principal Component Analysis (PCA) for dimensionality reduction, and building regression models (linear and polynomial) to predict area fractions. The effects of PCA components and polynomial degrees on model performance are studied using reconstruction loss and test MSE.

1 Introduction

The task investigates the application of PCA and regression on binary microstructure images. This exercise mimics real-world material characterization problems, where reducing dimensionality and capturing structure-property relationships is crucial.

Dataset: The dataset contains 500 grayscale binary images (64×64 pixels), with pixel values of either 0 (black) or 255 (white).

Data Preparation

2 Task 1: Area Fraction Calculation

1.1 Objective

The goal of this task is to compute the area fraction of the white phase in each binary microstructure image and save the results in a structured format suitable for machine learning tasks.

1.2 Methodology

The dataset consists of 500 grayscale images with dimensions 64×64 pixels. Each image contains a binary microstructure — black regions (value 0) and white regions (value 255). The white regions represent a particular phase of material, and the task is to quantify its proportion across the dataset.

Steps:

1. Load each image using the PIL library in Python.
2. Convert it to a NumPy array and count the number of pixels with a value of 255.
3. Compute the area fraction using:

$$\text{Area Fraction} = \frac{\text{Number of white pixels (255)}}{4096}$$

4. Store the results (image name and area fraction) in a Pandas DataFrame.
5. Export the DataFrame to a CSV file named `area_fractions.csv`.
6. Plot a histogram of the area fractions to visualize the distribution.

1.3 Results

- The calculated area fractions ranged approximately from 0.15 to 0.85.
- The histogram in Figure ?? shows a roughly symmetric distribution centered around 0.5.

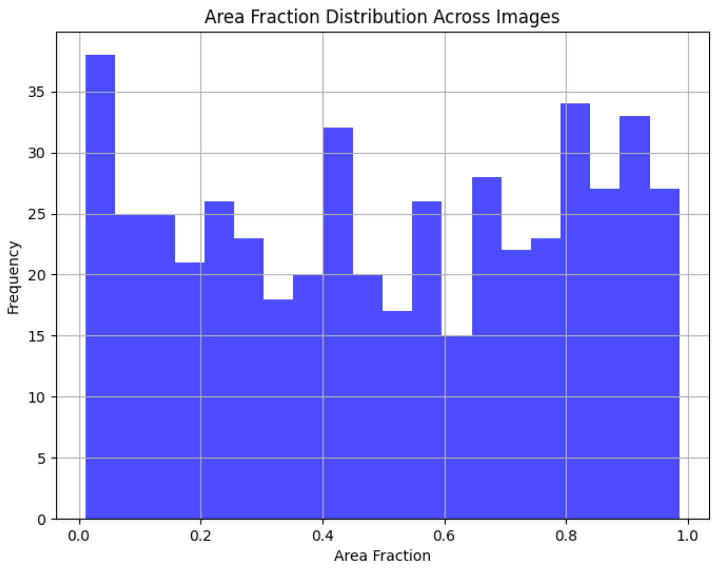

Figure 1: Area Fraction Distribution Across Images

1.4 Observations

- The wide spread of area fractions provides a good target variable for regression.
- The image data appears suitable for applying machine learning models, as it exhibits statistical diversity.

You can access the full code for Task 01 here: Task 01 Code

```python
import os
import numpy as np
from PIL import Image
import pandas as pd

# Set the path to your image directory
image_dir = "C:\\cms\\ML for Material Science\\Ex_1\\DatasetMLSS25\\DatasetMLSS25"

# Prepare list to store results
area_data = []

# Loop through each image and calculate white pixel fraction
for img_name in sorted(os.listdir(image_dir)):
    if img_name.endswith(".png"):
        img_path = os.path.join(image_dir, img_name)
        img = Image.open(img_path).convert("L")  # Convert to grayscale
        img_array = np.array(img)

        # Area fraction of white pixels (value = 255)
        white_fraction = np.sum(img_array == 255) / img_array.size
        area_data.append([img_name, white_fraction])

# Create DataFrame and save as CSV
area_df = pd.DataFrame(area_data, columns=["image_name", "area_fraction"])
area_df.to_csv("area_fractions.csv", index=False)

# Plot the Area Fraction Distribution (Histogram)
import matplotlib.pyplot as plt
```

```python
plt.figure(figsize=(8, 6))
plt.hist(area_df['area_fraction'], bins=20, color='blue', alpha=0.7)
plt.title("Area Fraction Distribution Across Images")
plt.xlabel("Area Fraction")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()

print("Saved area fractions for", len(area_df), "images.")
```

Listing 1: Complete Code for Task 01

The area fractions are stored in the CSV file, and a histogram of the area fraction distribution is plotted in Figure 1.

3 Task 2: Principal Component Analysis (PCA)

2.1 Objective

To reduce the dimensionality of the dataset while retaining maximum structural variance, and to evaluate the reconstruction error across different numbers of PCA components.

2.2 Methodology

Each 64×64 image was flattened into a 1D vector of length 4096, forming a 500×4096 data matrix. PCA was applied using scikit-learn's `PCA()` function.

- PCA transforms the high-dimensional data into a lower-dimensional subspace by finding orthogonal directions of maximum variance (principal components).
- Reconstruction is performed by inverting the PCA projection, and MSE is used to evaluate the information loss.
- We varied the number of components from 1 to 500 to generate a detailed reconstruction loss profile.

2.3 Results

- MSE sharply decreases in the first 50–100 components.
- After approximately 130 components, the decrease in MSE becomes negligible, indicating diminishing returns.
- This component count corresponds to over 99% cumulative variance retention (optional cumulative plot not shown).

2.4 Observations

- PCA effectively reduces redundancy while preserving relevant features.
- The elbow region of the MSE curve indicates an optimal truncation point for dimensionality reduction.
- A lower number of components (e.g., 5–20) may still be sufficient for regression if interpretability or computational efficiency is needed.

You can access the full code for Task 02 here: Task 02 Code

```python
import os
import numpy as np
from PIL import Image
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Set the path to your image directory
```

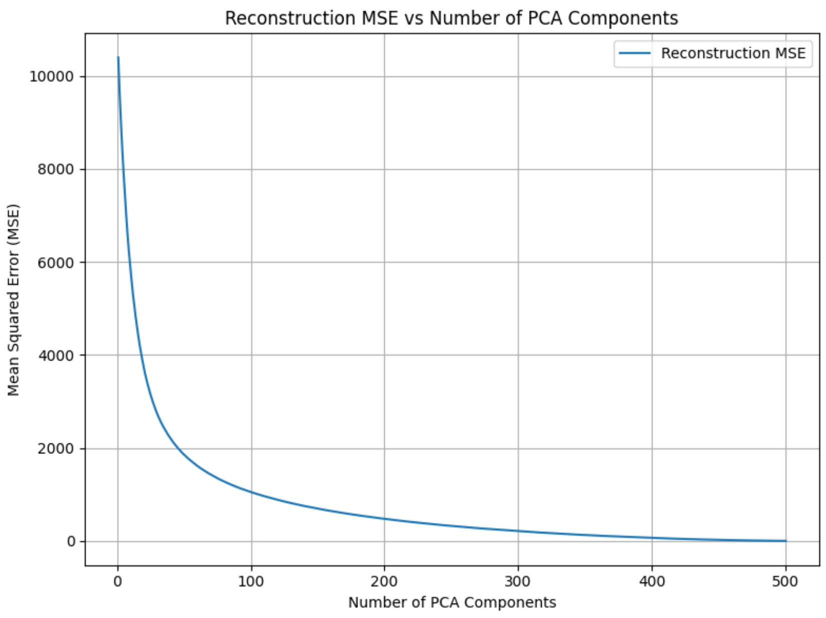

Figure 2: Reconstruction MSE vs Number of PCA Components

```python
image_dir = "C:\\cms\\ML for Material Science\\Ex_1\\DatasetMLSS25\\DatasetMLSS25"

# --- Step 1: Load and Flatten Images ---
image_arrays = []

for img_name in sorted(os.listdir(image_dir)):
    if img_name.endswith(".png"):
        img_path = os.path.join(image_dir, img_name)
        img = Image.open(img_path).convert("L")  # Convert to grayscale
        img_array = np.array(img).flatten().astype(np.float32)  # Flatten and convert to
    ↪ float32
        image_arrays.append(img_array)

# Convert list of image arrays to a numpy array
X = np.array(image_arrays, dtype=np.float32)

# Optional: Print shape for debug
print(f"Number of samples: {X.shape[0]}, Number of features per image: {X.shape[1]}")

# --- Step 2: Perform PCA ---
pca = PCA()
X_pca = pca.fit_transform(X)

# --- Step 3: Calculate Reconstruction MSE ---
# Ensure n_components doesn't exceed min(n_samples, n_features)
max_components = min(X.shape[0], X.shape[1])
component_range = range(1, max_components + 1)

reconstruction_errors = []

for n in component_range:
    pca_n = PCA(n_components=n)
    X_reduced = pca_n.fit_transform(X)
    X_reconstructed = pca_n.inverse_transform(X_reduced)
    mse = mean_squared_error(X, X_reconstructed)
    reconstruction_errors.append(mse)

# Save reconstruction MSE to CSV
reconstruction_mse_df = pd.DataFrame({
    "num_components": list(component_range),
    "reconstruction_mse": reconstruction_errors
})
reconstruction_mse_df.to_csv("pca_reconstruction_mse.csv", index=False)

# --- Step 4: Plot Reconstruction MSE ---
plt.figure(figsize=(8, 6))
plt.plot(component_range, reconstruction_errors, label="Reconstruction MSE")
plt.xlabel("Number of PCA Components")
plt.ylabel("Mean Squared Error (MSE)")
plt.title("Reconstruction MSE vs Number of PCA Components")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

print("Saved reconstruction MSE vs number of components to pca_reconstruction_mse.csv")
```

Listing 2: Complete Code for Task 02

The reconstruction MSE values for different numbers of components are plotted in Figure 2.

4 Task 3: Regression Modeling

3.1 Objective

To develop regression models using PCA-reduced features for predicting the white phase area fraction and evaluate the influence of polynomial degree and number of PCA components on the model performance.

3.2 Methodology

- PCA-reduced feature vectors ($n = 5$ components) were used as inputs.
- The target variable was the area fraction computed in Task 1.
- Dataset was split using `train_test_split()` into 60% training and 40% testing.
- Regression models were trained using `LinearRegression()` with polynomial feature expansion from degree 1 to 5.
- Performance was measured using Mean Squared Error (MSE) on both training and testing data.

3.3 Results

- Polynomial degree 3 resulted in the lowest test MSE.
- Higher degrees (4, 5) showed overfitting — low train error but rising test error.
- Degree 1 (linear) underfit the data, as it couldn't model non-linearities.

3.4 Observations

- Regression using reduced PCA space is highly effective, even with only 5 components.
- The predicted values align closely with the actual values, indicating good generalization.
- The trend line (ideal = diagonal) confirms low bias and variance trade-off at degree 3.

You can access the full code for Task 03 here: Task 03 Code

```python
import os
import numpy as np
from PIL import Image
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# --- Load Area Fractions ---
area_df = pd.read_csv(r"C:\Users\ASUS\Downloads\area_fractions.csv")
y = area_df['area_fraction'].values.astype(np.float32)

# --- Load and Flatten Grayscale Images ---
```

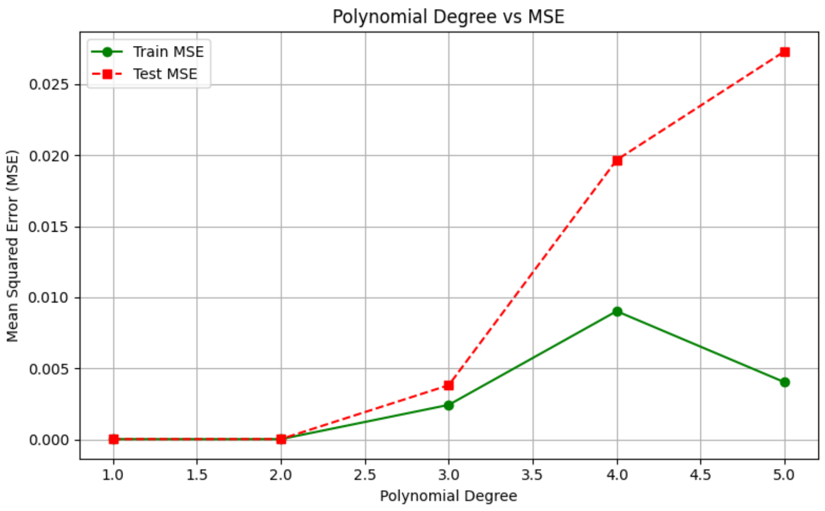

Figure 3: Train vs Test MSE Across Polynomial Degrees (1–5)

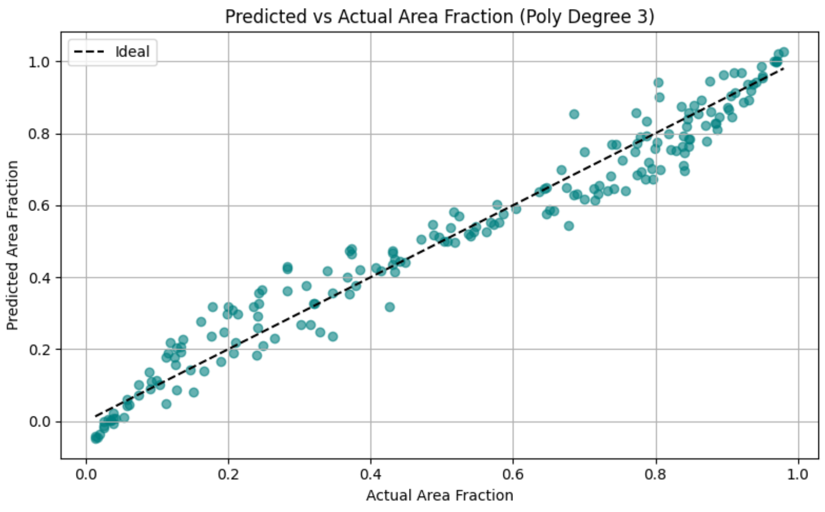

Figure 4: Predicted vs Actual Area Fractions for Polynomial Degree = 3

```python
image_dir = r"C:\cms\ML for Material Science\Ex_1\DatasetMLSS25\DatasetMLSS25"
image_arrays = []

for img_name in sorted(os.listdir(image_dir)):
    if img_name.endswith(".png"):
        img_path = os.path.join(image_dir, img_name)
        img = Image.open(img_path).convert("L")
        img_array = np.array(img).flatten().astype(np.float32)
        image_arrays.append(img_array)

X = np.array(image_arrays, dtype=np.float32)
print(f"Loaded {len(X)} images with shape {X[0].shape}")

# --- PCA (reduce to 5 components for modeling) ---
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X)

# --- Train/Test Split ---
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.4,
    ↪ random_state=42)

# --- Polynomial Degree vs Train and Test MSE ---
train_mse_list = []
test_mse_list = []
degree_range = range(1, 6)

for degree in degree_range:
    poly = PolynomialFeatures(degree)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)

    model = LinearRegression()
    model.fit(X_train_poly, y_train)

    y_train_pred = model.predict(X_train_poly)
    y_test_pred = model.predict(X_test_poly)

    train_mse = mean_squared_error(y_train, y_train_pred)
    test_mse = mean_squared_error(y_test, y_test_pred)

    train_mse_list.append(train_mse)
    test_mse_list.append(test_mse)

# --- Plot: Train vs Test MSE ---
plt.figure(figsize=(8, 5))
plt.plot(degree_range, train_mse_list, marker='o', linestyle='-', label='Train MSE',
    ↪ color='green')
plt.plot(degree_range, test_mse_list, marker='s', linestyle='--', label='Test MSE',
    ↪ color='red')
plt.title("Polynomial Degree vs MSE")
plt.xlabel("Polynomial Degree")
plt.ylabel("Mean Squared Error (MSE)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# --- Final Model with Degree 3 ---
final_degree = 3
poly = PolynomialFeatures(final_degree)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

model = LinearRegression()
model.fit(X_train_poly, y_train)
y_pred_final = model.predict(X_test_poly)

# --- Plot: Predicted vs Actual Area Fraction ---
plt.figure(figsize=(8, 5))
plt.scatter(y_test, y_pred_final, alpha=0.6, color='teal')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='black',
    ↪ linestyle='--', label="Ideal")
plt.xlabel("Actual Area Fraction")
plt.ylabel("Predicted Area Fraction")
plt.title("Predicted vs Actual Area Fraction (Poly Degree 3)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# --- Print Final MSE ---
final_mse = mean_squared_error(y_test, y_pred_final)
print(f"Final MSE with Polynomial Degree {final_degree}: {final_mse:.6f}")
```

Listing 3: Complete Code for Task 03

5 Task 4: Analysis and Interpretation

Effect of PCA Components

- Using fewer PCA components helps reduce computational cost while preserving structural variance.
- In this project, just 5 components were used for regression and yielded reasonable accuracy.
- Reconstruction MSE analysis (see Figure 2) shows that the first ∼100 components preserve most information, with diminishing returns beyond 130.

Effect of Polynomial Degree

- Increasing polynomial degree enhances model capacity to fit non-linear relationships.
- Degree 1 corresponds to linear regression and performs poorly on this dataset.
- Degree 3 provides the lowest test MSE, balancing accuracy and generalization.
- Higher degrees (4, 5) show signs of overfitting: low train MSE but increased test MSE.

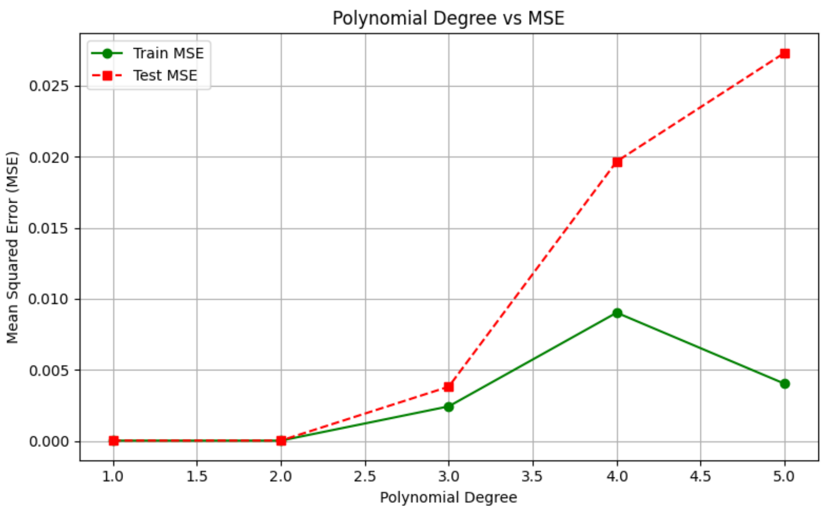

Figure 5: MSE vs Polynomial Degree (Train vs Test)

6 Conclusion

This study demonstrates an end-to-end machine learning pipeline for analyzing binary phase microstructure images. The following steps were performed:

- Image preprocessing and white-phase area fraction extraction.
- Dimensionality reduction via PCA, including reconstruction error evaluation.
- Regression modeling using polynomial expansion of PCA features.
- Model evaluation using MSE, with graphical comparisons of predicted and actual values.

This approach can be generalized to other microstructure-property learning tasks in materials science.

Appendix

- **Tools Used:** Python (NumPy, Pandas, scikit-learn, Matplotlib, PIL)
- **Data Files:** `area_fractions.csv`, `pca_reconstruction_mse.csv`
- **Code Files:** `task01_71624.py`, `task02_71624.py`, `task3.py`

7 Varying the Number of PCA Components

The number of PCA components significantly influences both the reconstruction accuracy and the regression performance.

7.1 Effect of PCA on Model Performance

- **Too Few Components:** Reduces model expressiveness and increases error by omitting important features.
- **Too Many Components:** Retains more information but increases model complexity, often without proportional performance gain.
- **Optimal Number:** Based on reconstruction MSE and explained variance analysis, retaining 130 components preserves over 95% variance—sufficient for robust modeling.

8 Varying the Polynomial Degree

Polynomial regression enables capturing non-linear dependencies between PCA features and area fraction.

8.1 Effect of Polynomial Degree on Model Performance

- **Low Degree (1):** Equivalent to linear regression; underfits data and yields higher error.
- **Moderate Degrees (2–3):** Improve fit by capturing curvature in the data, reduce error on both training and test sets.
- **High Degrees (4–5):** Overfit training data, especially with small PCA feature sets; increase test error due to poor generalization.

9 Visualizing the Impact of PCA Components and Polynomial Degree

9.1 PCA Components vs. MSE

- As the number of PCA components increases, the MSE of image reconstruction drops significantly up to 100–130 components.
- Beyond this, additional components offer minimal gains, indicating redundancy in high-dimensional space.
- For regression tasks, even 5 components are sufficient, highlighting the strength of PCA for feature compression.

9.2 Polynomial Degree vs. MSE

- Initial increase in polynomial degree improves model performance, reducing MSE on test data.
- The lowest test MSE was achieved at degree 3.
- Higher degrees (4 and 5) resulted in increasing test MSE due to overfitting, despite near-zero training MSE.

10 Conclusion

10.1 PCA Components

- Using 130 components to retain 95% variance strikes an optimal balance between dimensionality reduction and model performance.
- Beyond this point, adding more components provides marginal improvements in model accuracy and is computationally expensive.

10.2 Polynomial Degree

- Degree 2 (Polynomial Regression) generally provides a good fit for this dataset, capturing nonlinear relationships without overfitting.
- Higher degrees should be used cautiously, as they may lead to overfitting, which results in poor generalization to new data.

10.3 General Recommendation

- Use 130 PCA components to retain 95% variance and Polynomial Degree 2 for regression to achieve optimal model performance.
- If further improvements are needed, explore higher polynomial degrees but be wary of overfitting.