

STAT 5244 – Unsupervised Learning

Homework 3

Name: Chuyang Su UNI: cs4570

1 Graphical Models.

1.1 Data Processing.

The log return transformation was applied to the daily closing prices. The daily log return r_t for a stock price P_t was calculated as:

$$r_t = \ln(P_t) - \ln(P_{t-1})$$

This dataset of log returns, spanning 1,228 trading days, was used for all subsequent graphical model fitting.

1.1.1 Descriptive Statistics.

The table below summarizes the descriptive statistics for the daily log returns.

Table 1: Descriptive Statistics of Daily Log Returns (Jan 2021 – Present)

| | AAPL | AMZN | BAC | CVX | GOOGL | JNJ | JPM | KO | META | MSFT | NVDA | PFE | PG | WMT | XOM |
|---------------------------|---------|---------|---------|---------|---------|---------|---------|---------|---------------|---------|--------------|---------|---------|---------|---------|
| Count | 1228.00 | 1228.00 | 1228.00 | 1228.00 | 1228.00 | 1228.00 | 1228.00 | 1228.00 | 1228.00 | 1228.00 | 1228.00 | 1228.00 | 1228.00 | 1228.00 | 1228.00 |
| Mean ($\times 10^{-3}$) | 0.63 | 0.27 | 0.53 | 0.64 | 1.02 | 0.33 | 0.81 | 0.38 | 0.65 | 0.66 | 2.13 | -0.12 | 0.18 | 0.68 | 1.01 |
| Std ($\times 10^{-2}$) | 1.76 | 2.23 | 1.72 | 1.60 | 1.96 | 1.05 | 1.53 | 1.00 | 2.78 | 1.63 | 3.29 | 1.59 | 1.09 | 1.32 | 1.71 |
| Min | -0.097 | -0.151 | -0.117 | -0.086 | -0.100 | -0.079 | -0.078 | -0.072 | -0.306 | -0.080 | -0.186 | -0.070 | -0.064 | -0.121 | -0.082 |
| Max | 0.143 | 0.127 | 0.081 | 0.085 | 0.097 | 0.060 | 0.109 | 0.046 | 0.209 | 0.097 | 0.218 | 0.103 | 0.042 | 0.091 | 0.062 |

The data clearly demonstrates the risk-return trade-off. The semiconductor stock NVDA shows the highest average daily return ($\sim 0.213\%$) but also the highest volatility (Standard Deviation: 3.29%) and largest maximum single-day return ($\sim 21.8\%$). Conversely, consumer staples stocks like KO (Coca-Cola) and PG (P&G) exhibit the lowest standard deviations ($\sim 1.0\%$), indicating high stability but lower returns. The largest single-day drop belongs to META (former FB) at -30.6% .

1.1.2 Time-Series Exploration.

The cumulative returns plot (Figure 1) illustrates the differential performance across sectors over the analysis period.

1.1.3 Correlation Analysis.

The correlation heatmap (Figure 2) reveals strong clustering of dependence among stocks within the same sector, which confirms the pervasive influence of systematic market risk.

Key Observations from the Heatmap:

- **Strong Correlation (0.6+):** High-tech stocks (AAPL, MSFT, AMZN, GOOGL, NVDA) are tightly coupled (e.g., MSFT–AMZN at 0.66, MSFT–GOOGL at 0.65). Financials (JPM–BAC at 0.82) and Energy stocks (CVX–XOM at 0.86) exhibit the highest correlations, reflecting their singular dependence on industry-specific factors (e.g., oil price, interest rates).
- **Weak/Low Correlation (0.0-0.3):** Healthcare stocks (JNJ, PFE) show low correlation with most other stocks (e.g., JNJ vs. Tech stocks often below 0.2), confirming their defensive, counter-cyclical nature.
- **Negative Correlation:** A notable weak negative correlation exists between the pharmaceutical stock JNJ and the high-growth technology stock NVDA (~ -0.09), suggesting an interesting divergence in their underlying risk drivers.

This preliminary analysis confirms the existence of strong, sector-specific dependencies, which the Graphical Lasso will aim to distill into a network of conditional dependencies.

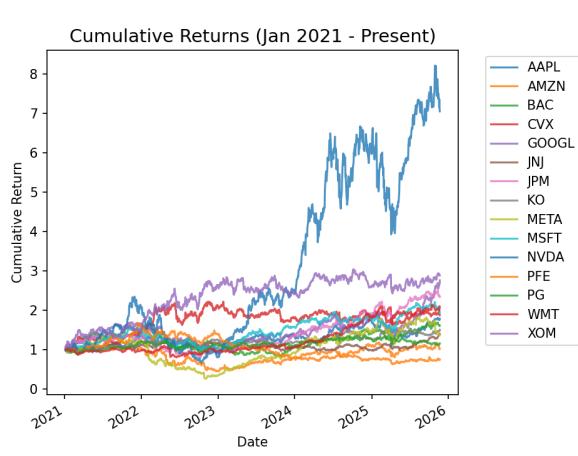


Figure 1: Cumulative Log Returns of Selected Stocks (Jan 2021 - Present)

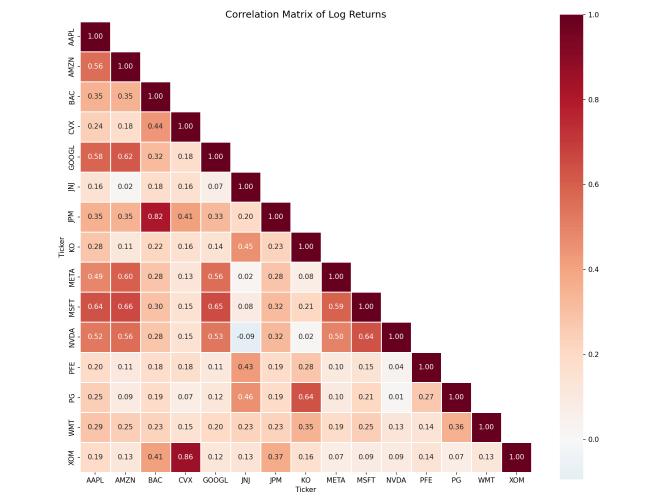


Figure 2: Correlation Heatmap of Daily Log Returns

1.2 Graphical Lasso.

Figure 3 shows the estimated precision matrices from both the Gaussian Graphical Lasso and the nonparanormal (rank-based) Graphical Lasso. The two heatmaps are almost identical, indicating that although individual stock returns are heavy-tailed, the dependence structure is well approximated by a Gaussian copula. Consequently, both methods recover essentially the same sparse conditional dependence network, suggesting that the underlying structure is stable, low-dimensional, and largely driven by sector-level factors.

The estimated graph highlights several strong conditional dependencies (e.g., AMZN–KO, JPM–GOOGL, WMT–BAC, NVDA–PFE) and a clear technology cluster consisting of AAPL, MSFT, GOOGL, AMZN, and META. NVIDIA does not join this cluster, likely due to its unusually strong and volatile performance during the sample period, which weakens

its partial correlations with the other technology stocks after conditioning on the full set of variables.

The regularization parameter α was selected via cross-validated Gaussian log-likelihood over a grid of 30 values spanning $\log_{10}(0.01)$ to $\log_{10}(0.8)$. This criterion is appropriate for unsupervised graphical models, as the validation likelihood measures generalization of the estimated precision matrix. The optimal values were

$$\alpha_{\text{Gaussian}} = 0.021287, \quad \alpha_{\text{Nonparanormal}} = 0.013528.$$

For brevity, only the tuning curve for the Gaussian estimator is shown in Figure 4.

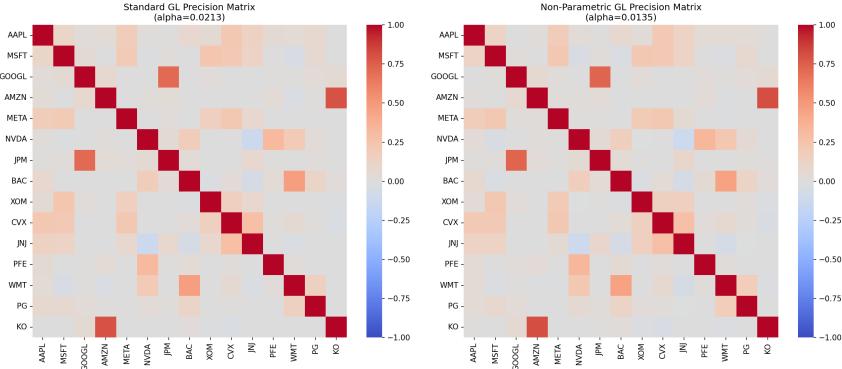


Figure 3: Gaphic Lasso Estimated Precision Matrices: Standard (Left) vs. Non-Parametric (Right)

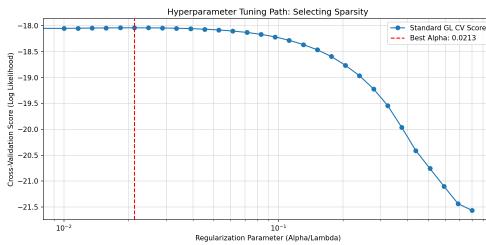


Figure 4: Cross-Validated Log-Likelihood Curve for Standard Graphical Lasso

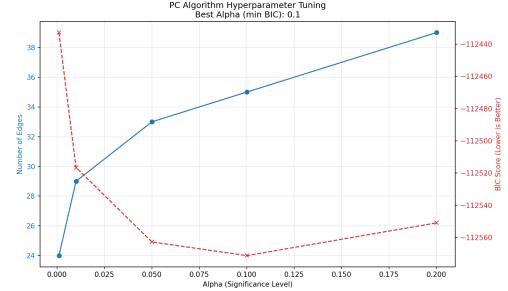


Figure 5: Best Alpha Selection for PC Algorithm via BIC Score

1.3 PC Algorithm.

To determine the optimal regularization level for the PC algorithm, we evaluated the Bayesian Information Criterion (BIC) across a range of significance thresholds

$$\alpha \in \{0.001, 0.01, 0.05, 0.1, 0.2\}.$$

The resulting BIC scores are shown in Figure 5. Although $\alpha = 0.05$ is often used as a conventional threshold, the BIC curve indicates that the model achieves its minimum score

at $\alpha = 0.1$, implying that this level of sparsity provides the best balance between model fit and complexity.

Based on this criterion, we select $\alpha = 0.1$ and construct the final directed graph using the PC algorithm. The resulting structure is displayed in Figure 6, which represents the learned conditional independence relations and the corresponding Markov equivalence class under this optimal parameter choice.

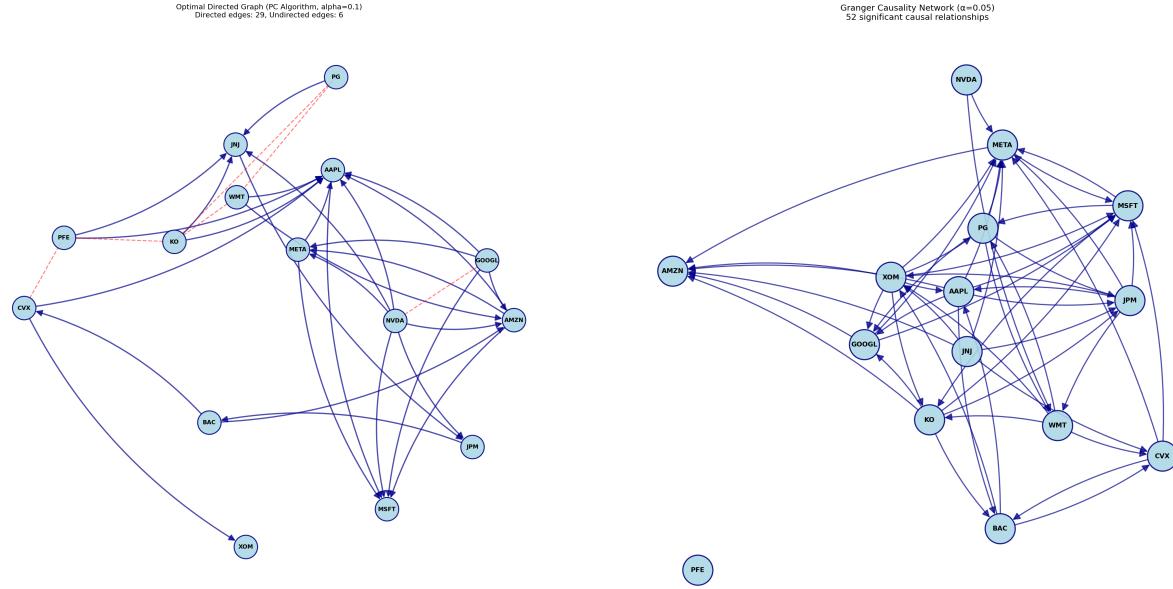


Figure 6: Best PC Algorithm Graph at $\alpha = 0.1$

Figure 7: Granger Causal Graphical Model at $\alpha = 0.05$, Max Lag = 5

1.4 Comparison and Interpretation.

The learned structure from Figure 6 displays a clear pattern: technology stocks exhibit substantially richer and more directional dependency relationships compared to the other sectors. For example, MSFT appears as a child of all technology names, including AAPL, GOOGL, NVDA, AMZN and META, indicating that its conditional distribution depends structurally on multiple peers within the same sector. In contrast, NVDA and GOOGL share an undirected edge but act as parents to all other surrounding nodes, placing them in more central positions within the technology cluster. These findings are consistent with the results from part (b), in which the precision matrices also implied a tightly connected technology block.

Other sectors exhibit markedly more isolated behavior. For instance, in the energy sector, XOM is connected only to CVX, and once CVX is conditioned on, XOM becomes conditionally independent of all other stocks in the universe. This highlights both the internal coherence of the energy sector and its relative independence from the remaining market, which are consistent with common sense.

An interesting contrast emerges when comparing the PC graph with the precision matrices from part (b). Pairs such as KO–AMZN and JPM–GOOGL exhibit strong partial

correlations under the Gaussian and nonparanormal graphical lasso, yet no direct edge appears between them in the PC graph. This difference reflects the distinct logics of the two models: graphical lasso identifies pairwise partial correlations, while the PC algorithm searches for a directed acyclic graph that satisfies a complete set of conditional independence relations. As a result, some associations are represented not by direct edges but by indirect paths—for example, KO connects to AMZN through AAPL or WMT. This is also consistent with economic intuition, as consumer staples and mega-cap technology firms often share indirect market linkages through broader macro or demand channels.

1.5 Granger Causal Graphical Model.

To evaluate the stability of the Granger causality model, we performed a hyperparameter sweep over significance levels $\alpha \in \{0.01, 0.05, 0.1\}$ and maximum lags $\{1, 3, 5, 7, 10\}$. The resulting edge counts and graph densities are summarized in Figure 8. Because interpretable causal networks should remain reasonably sparse, we selected $\alpha = 0.05$ and a maximum lag of 5.

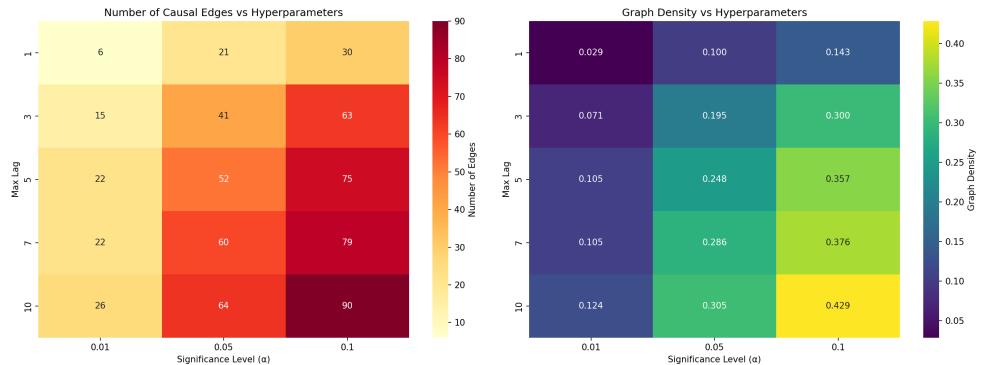


Figure 8: Granger Causality Model Tuning: Edge Counts (Left) and Graph Density (Right) Across Significance Levels and Maximum Lags

However, even under this moderate configuration, the resulting Granger graph still contains 52 significant edges (density = 0.248), which is far more dense than the sparse and interpretable structures obtained in parts (b) and (c). Figure 9 illustrates the adjacency matrix of significant Granger causal relationships, showing an unusually high concentration of edges across sectors. This density pattern suggests that the Granger framework is detecting a large number of spurious lead-lag relations rather than meaningful predictive structure.

The final inferred Granger network is shown in Figure 7. The resulting graph is visibly dense and lacks the sectoral organization and coherent clusters observed in the graphical lasso and PC algorithm results. Instead of revealing identifiable industry-level dependency patterns, the Granger graph displays widespread, cross-sector connections that contradict well-known market structure and are characteristic of noise-driven statistical artifacts in financial return data.

Together, these observations indicate that the Granger causality model does not fit the dataset well. Even after choosing a reasonably conservative hyperparameter setting, the

method overfits the volatility and idiosyncratic noise inherent in daily stock returns, producing a dense and unstable network that fails to capture the stable structural dependencies recovered by the graphical lasso and PC models.

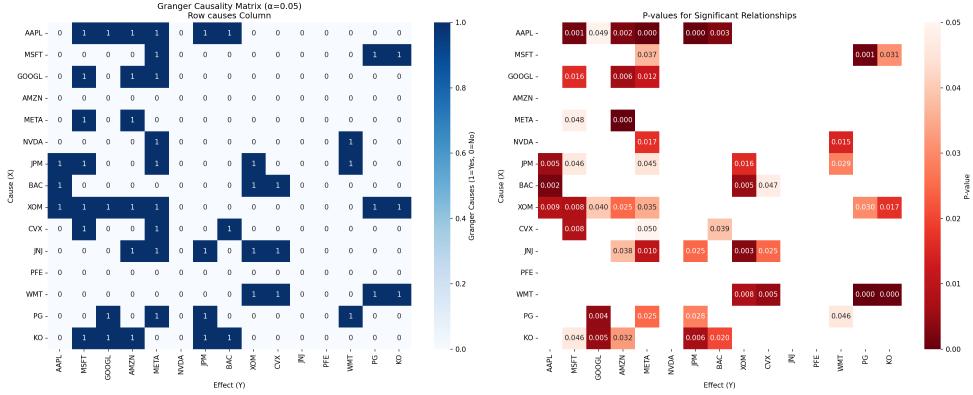


Figure 9: Granger Causality Adjacency Matrix at $\alpha = 0.05$, Max Lag = 5

2 Density estimation & Generative Models.

2.1 Kernel Density Estimation (KDE).

A grid search was performed on the bandwidth parameter h across three spaces (PCA-20/50 and original 64-dimensional space), with 5-fold cross-validation used to select the optimal value based on average log-likelihood, the result is shown in Figure 10. Since the CV score in the original space deteriorated significantly (-19606 vs -8081), confirming the curse of dimensionality, the PCA-20 space (optimal $h = 0.580$) was chosen for subsequent sampling. The pixel intensity distribution of generated samples closely matches the original data (mean 4.88 vs 4.95, standard deviation 6.02 vs 6.64), but spatial structure differs: generated samples exhibit only 23.91% sparsity compared to the original 48.93%, a 50% reduction, indicating KDE's tendency to produce denser pixel distributions. This results in some digits (e.g., 1, 0, 8) being clearly recognizable, though overall contrast remains insufficient(as shown in Figure 11).

2.2 Generative Adversarial Network (GAN).

Overall, the parameter combination `latent_dim=32, lr_g=0.0001, G=[128, 256]` proves most suitable: it achieves the lowest `std_diff` (indicating generated sample distributions closest in shape to the original data), while maintaining competitive `mean_diff`. Moreover, `latent=32` is more lightweight and less prone to overfitting, with diversity ≈ 1.7 indicating no severe mode collapse.

Other hyperparameters include α_D fixed at 0.0002, consistent with the DCGAN paper. The final choice employs $\alpha_G < \alpha_D$, allowing the Generator to learn more slowly, since my previous experiments with $\alpha_G = \alpha_D$ resulted in training collapse due to Generator dominance over the Discriminator.

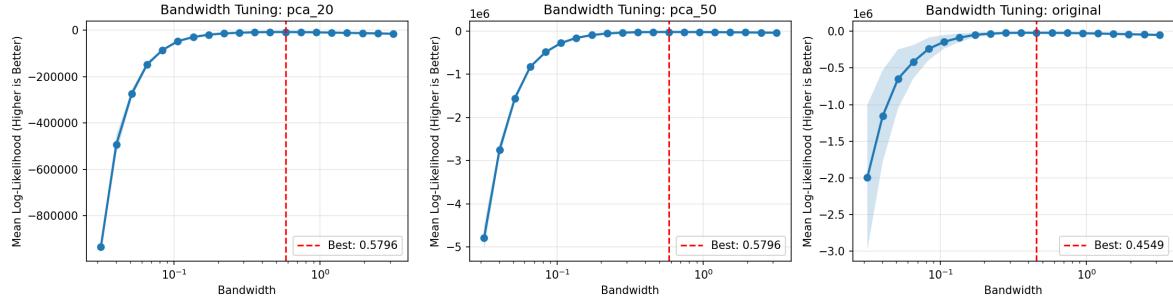


Figure 10: KDE Bandwidth Tuning via 5-Fold Cross-Validation in PCA-20, PCA-50, and Original Spaces



Figure 11: Generated Samples from KDE

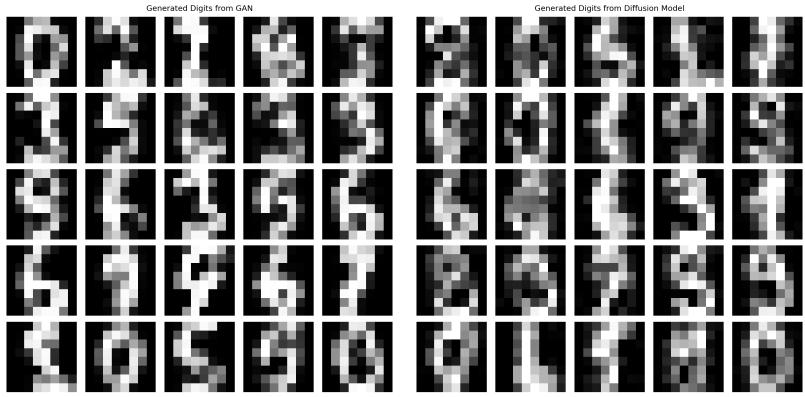


Figure 12: Generated Samples from GAN



Figure 13: Generated Samples from Diffusion Model

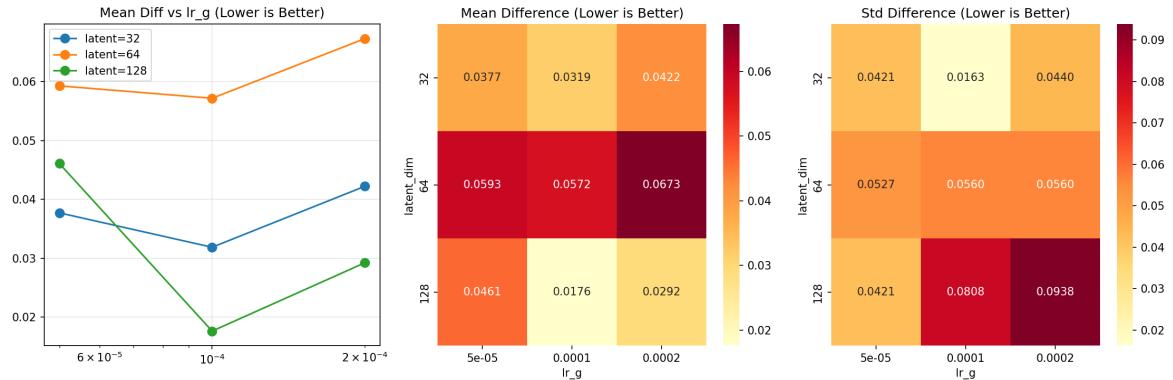


Figure 14: GAN Hyperparameter Tuning Results

Under optimal parameters, the model training process is illustrated in Figure 14. Compared to my previous experiments, both losses converge relatively well, stabilizing around 1.0–1.2, with Discriminator accuracy exceeding 0.5. However, a significant disparity exists in distinguishing real versus generated samples: accuracy remains above 0.8 when identifying fake samples, but falls below 0.6 for real samples. This reflects an inherent limitation of Vanilla GAN—the min-max game struggles to reach Nash equilibrium, explaining the

research motivation behind subsequent methods such as WGAN and Diffusion Models.

Generated sample examples are shown in Figure 12. Several digits (e.g., 0, 1, 2, 3, 5, 6, 9) exhibit high recognizability, with overall performance significantly superior to KDE.

2.3 Denoising Diffusion Model.

In this experiment, we optimized three hyperparameters: learning rate (lr), number of timesteps, and network architecture. Among all 18 combinations, we used heatmaps of mean_diff and std_diff to determine lr and timesteps, while final loss guided the selection of architecture. The results are shown in Figure 15.

From the final loss perspective, architecture [256, 512, 256] significantly outperforms [128, 256, 128] on average. Building upon this, the combination of timesteps = 500 and lr = 0.0005 achieves the lowest mean_diff (0.0054), with std_diff also within an acceptable range. Therefore, we select lr = 0.0005, timesteps = 500, and architecture = [256, 512, 256] as our optimal model configuration. All subsequent discussions are based on this setting.

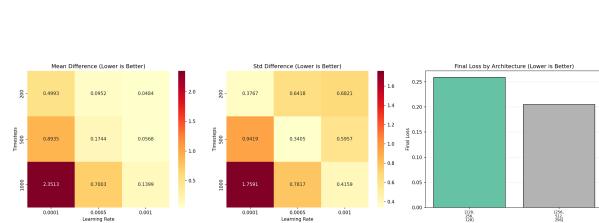


Figure 15: Diffusion Model Hyperparameter Tuning Results

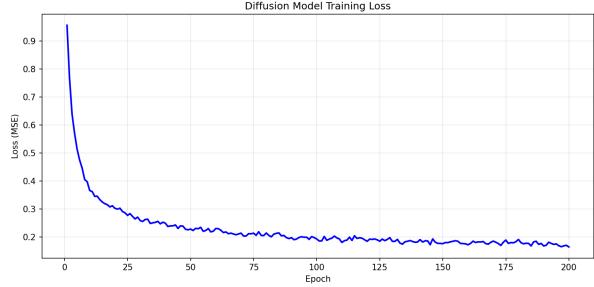


Figure 16: Training Trajectory of the Diffusion Model

The training trajectory of the diffusion model is shown in Figure 16, and representative generated samples are displayed in Figure 13.

Comparing the generated results from KDE, GAN, and diffusion, several clear patterns emerge. From a statistical perspective, the GAN produces samples with higher overall brightness and stronger contrast, and its sparsity level is closer to that of the original dataset. This suggests that the generator is able to capture certain global statistical features of the data distribution. However, in terms of structural clarity and recognizability, the diffusion model performs markedly better: its generated digits exhibit sharp contours and well-defined shapes, whereas the outputs of KDE and GAN appear significantly more blurred.

The weaker structural fidelity of the GAN can be attributed to the imbalance in the adversarial game. In our setting, the discriminator is insufficiently strong and fails to reliably distinguish real from generated digits. As a result, the generator can easily “fool” the discriminator without learning sharper or more realistic digit structures. This issue is further exacerbated by the high sensitivity of the vanilla GAN to hyperparameters, making it difficult to achieve a stable equilibrium between the generator and discriminator. These challenges highlight the fundamental limitations of the GAN approach in this task and explain the performance gap relative to the diffusion model.

A Appendix: Code Implementation.

For further details, please visit my GitHub repository:

<https://github.com/Schuyn/Unsupervised-Learning-Homework.git>

A.1 Problem 1: Graphical Models.

A.1.1 Main Script.

```

1 """
2 Author: Chuyang Su cs4570@columbia.edu
3 Date: 2025-11-23 17:41:45
4 LastEditTime: 2025-11-24 12:27:59
5 FilePath: /Unsupervised-Learning-Homework/Homework 3/Code/main.py
6 Description:
7     Main script to run data processing and graphical model fitting for
8         Homework 3.
9 """
10
11 from Prob1_utils import Prob1Analysis
12
13 def main():
14     # Problem 1a
15     p1=Prob1Analysis()
16     log_returns = p1.process_stock_data(verbose=False)
17
18     # Problem 1b
19     glasso_results = p1.fit_glasso_models(verbose=False)
20
21     # Problem 1c
22     pc_results = p1.fit_pc_model(verbose=False)
23
24     # Problem 1e
25     granger_results = p1.fit_granger_model(verbose=True)
26
27 if __name__ == "__main__":
28     main()

```

A.1.2 Utils Script.

```

1 """
2 Author: Chuyang Su cs4570@columbia.edu
3 Date: 2025-11-23 17:27:40
4 LastEditTime: 2025-11-26 11:22:20
5 FilePath: /Unsupervised-Learning-Homework/Homework 3/Code/prob1_utils.py
6 Description:
7     Utility functions and classes for Problem 1 analysis in Homework 3.
8 """
9 import os
10 import yfinance as yf
11 import pandas as pd
12 import numpy as np

```

```

13 import matplotlib.pyplot as plt
14 import seaborn as sns
15 from scipy.stats import norm
16 import networkx as nx
17 from sklearn.covariance import GraphicalLassoCV
18 from sklearn.preprocessing import StandardScaler
19 from sklearn.linear_model import LinearRegression, LassoCV
20 from statsmodels.tsa.stattools import grangercausalitytests
21 from causallearn.search.ConstraintBased.PC import pc
22
23 class Prob1Analysis:
24     def __init__(self,
25                  output_dir='Homework 3/Code/Data',
26                  figure_dir='Homework 3/Latex/Figures',
27                  start_date="2021-01-01"):
28         self.output_dir = output_dir
29         self.figure_dir = figure_dir
30         self.start_date = start_date
31
32         os.makedirs(self.output_dir, exist_ok=True)
33         os.makedirs(self.figure_dir, exist_ok=True)
34
35         self.raw_data_file = os.path.join(self.output_dir, 'raw_stock_data.csv')
36         self.log_returns_file = os.path.join(self.output_dir, 'log_returns.csv')
37
38         self.tickers = ["AAPL", "MSFT", "GOOGL", "AMZN", "META", "NVDA",
39                         "JPM", "BAC", "XOM", "CVX", "JNJ", "PFE", "WMT", "PG",
40                         "KO"]
41
42         self.sectors = {
43             'Tech': ['AAPL', 'MSFT', 'GOOGL', 'AMZN', 'META', 'NVDA'],
44             'Finance': ['JPM', 'BAC'],
45             'Energy': ['XOM', 'CVX'],
46             'Healthcare': ['JNJ', 'PFE'],
47             'Consumer': ['WMT', 'PG', 'KO']
48         }
49
50         self.log_returns = None
51         self.glasso_results = None
52
53     def process_stock_data(self, verbose=False):
54         # Download stock data and compute log returns
55         if os.path.exists(self.log_returns_file):
56             log_returns = pd.read_csv(self.log_returns_file, index_col=0,
57                                     parse_dates=True)
58
59         else:
60             print(f"Downloading data from yfinance...")
61             raw_data = yf.download(self.tickers, start=self.start_date, end=None)
62             raw_data.to_csv(self.raw_data_file)

```

```

62     # Extract closing prices and compute log returns
63     close_prices = raw_data['Close']
64     print(f"\nData shape: {close_prices.shape}")
65     print(f"Date range: {close_prices.index[0]} to {close_prices.index
66         [-1]}")
67
68     log_returns = np.log(close_prices / close_prices.shift(1)).dropna()
69     log_returns.to_csv(self.log_returns_file)
70     # print(f"\nMissing values per stock:\n{log_returns.isnull().sum()}")
71
72     self.log_returns = log_returns
73
74     # Visual exploration
75     if verbose:
76         self._plot_visual_exploration()
77
78     return log_returns
79
80 def _plot_visual_exploration(self):
81     # Summary statistics
82     summary_stats = self.log_returns.describe()
83     summary_stats.to_csv(os.path.join(self.output_dir, 'summary_statistics
84         .csv'))
85
86     # Plot correlation heatmap
87     plt.figure(figsize=(12, 10))
88     corr_matrix = self.log_returns.corr()
89     mask = np.triu(np.ones_like(corr_matrix, dtype=bool), k=1)
90     sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='RdBu_r', center=
91         0,
92         mask=mask, square=True, linewidths=0.5)
93     plt.title('Correlation Matrix of Log Returns', fontsize=14)
94     plt.tight_layout()
95     plt.savefig(os.path.join(self.figure_dir, '1a_correlation_heatmap.png'
96         ), dpi=150)
97     plt.show()
98
99     # Plot cumulative returns
100    plt.figure(figsize=(14, 8))
101    cumulative_returns = (1 + self.log_returns).cumprod()
102    cumulative_returns.plot(alpha=0.8)
103    plt.title('Cumulative Returns (Jan 2021 - Present)', fontsize=14)
104    plt.xlabel('Date')
105    plt.ylabel('Cumulative Return')
106    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
107    plt.tight_layout()
108    plt.savefig(os.path.join(self.figure_dir, '1a_cumulative_returns.png'
109        ), dpi=150)
110    plt.show()
111
112    # Relationships within different industries
113    fig, axes = plt.subplots(2, 3, figsize=(15, 10))

```

```

109     axes = axes.flatten()
110
111     for idx, (sector, stocks) in enumerate(self.sectors.items()):
112         sector_corr = self.log_returns[stocks].corr()
113         sns.heatmap(sector_corr, annot=True, fmt='.2f', cmap='RdBu_r',
114                     center=0,
115                     ax=axes[idx], square=True, vmin=-1, vmax=1)
116         axes[idx].set_title(f'{sector} Sector Correlation')
117
118     axes[-1].axis('off')
119     plt.suptitle('Within-Sector Correlation Analysis', fontsize=14)
120     plt.tight_layout()
121     plt.savefig(os.path.join(self.figure_dir, '1a_sector_correlations.png'
122                             ), dpi=150)
123     plt.show()
124
125 def fit_glasso_models(self, verbose=False):
126     if self.log_returns is None:
127         self.process_stock_data()
128
129     scaler = StandardScaler()
130     X_standardized = scaler.fit_transform(self.log_returns.values)
131
132     # Grid search best alpha values
133     alphas_grid = np.logspace(np.log10(0.01), np.log10(0.8), 30)
134
135     # Standard Graphical Lasso
136     if verbose:
137         print("Fitting Standard Graphical LassoCV...")
138     gl_cv = GraphicalLassoCV(alphas=alphas_grid, cv=5, n_jobs=-1, max_iter=
139                             3000, tol=1e-4)
140     gl_cv.fit(X_standardized)
141     # print(gl_cv.cv_results_)
142
143     # Non-parametric (Rank-based) Graphical Lasso
144     if verbose:
145         print("Fitting Non-parametric (Rank) Graphical LassoCV...")
146
147     # Transform data to ranks
148     n = self.log_returns.shape[0]
149     X_ranks = self.log_returns.rank() / (n + 1)
150     X_normal_scores = norm.ppf(X_ranks)
151     X_np_standardized = StandardScaler().fit_transform(X_normal_scores)
152
153     gl_np = GraphicalLassoCV(alphas=alphas_grid, cv=5, n_jobs=-1, max_iter=
154                             3000, tol=1e-4)
155     gl_np.fit(X_np_standardized)
156
157     # Compute partial correlations
158     prec = gl_cv.precision_
159     d = np.sqrt(np.diag(prec))
160     partial_corr = -prec / np.outer(d, d)
161     np.fill_diagonal(partial_corr, 1.0)

```

```

159     prec_np = gl_np.precision_
160     d_np = np.sqrt(np.diag(prec_np))
161     partial_corr_np = -prec_np / np.outer(d_np, d_np)
162     np.fill_diagonal(partial_corr_np, 1.0)
163
164     self.glasso_results = {
165         'gl_cv': gl_cv,
166         'gl_np': gl_np,
167         'precision_gl': prec,
168         'precision_np': prec_np,
169         'partial_corr_gl': partial_corr,
170         'partial_corr_np': partial_corr_np
171     }
172
173     # Verbose output and visualization
174     if verbose:
175         self._plot_glasso_results(gl_cv, gl_np, partial_corr, partial_corr_np
176                               )
177
178     return self.glasso_results
179
180 def _plot_glasso_results(self, gl_cv, gl_np, partial_corr, partial_corr_np
181                         ):
182     print("\n--- Graphical Lasso Results ---")
183     print(f"Standard GL Best Alpha: {gl_cv.alpha_:.6f}")
184     print(f"Non-Parametric GL Best Alpha: {gl_np.alpha_:.6f}")
185
186     # Plot CV curves
187     plt.figure(figsize=(10, 5))
188     cv_results = gl_cv.cv_results_
189     cv_means = cv_results['mean_test_score']
190     cv_alphas = cv_results['alphas']
191
192     plt.semilogx(cv_alphas, cv_means, 'o-', label='Standard GL CV Score')
193     plt.axvline(gl_cv.alpha_, linestyle='--', color='r', label=f'Best
194                 Alpha: {gl_cv.alpha_:.4f}')
195
196     plt.xlabel('Regularization Parameter (Alpha/Lambda)')
197     plt.ylabel('Cross-Validation Score (Log Likelihood)')
198     plt.title('Hyperparameter Tuning Path: Selecting Sparsity')
199     plt.legend()
200     plt.grid(True, which="both", ls="-", alpha=0.5)
201     plt.tight_layout()
202     plt.savefig(os.path.join(self.figure_dir, '1b_glasso_cv_curve.png'),
203                 dpi=150)
204     plt.show()
205
206     # Plot precision matrices
207     fig, axes = plt.subplots(1, 2, figsize=(16, 7))
208
209     # Standard
210     prec = gl_cv.precision_
211     d = np.sqrt(np.diag(prec))

```

```

209     partial_corr = -prec / np.outer(d, d)
210     np.fill_diagonal(partial_corr, 1.0)
211
212     sns.heatmap(partial_corr, ax=axes[0], cmap='coolwarm',
213                  xticklabels=self.tickers, yticklabels=self.tickers, vmin=-1,
214                  vmax=1)
215     axes[0].set_title(f'Standard GL Precision Matrix\n(alpha={gl_cv.alpha_:.4f})')
216
217     # Non-parametric
218     prec_np = gl_np.precision_
219     d_np = np.sqrt(np.diag(prec_np))
220     partial_corr_np = -prec_np / np.outer(d_np, d_np)
221     np.fill_diagonal(partial_corr_np, 1.0)
222
223     sns.heatmap(partial_corr_np, ax=axes[1], cmap='coolwarm',
224                  xticklabels=self.tickers, yticklabels=self.tickers, vmin=-1,
225                  vmax=1)
226     axes[1].set_title(f'Non-Parametric GL Precision Matrix\n(alpha={gl_np.alpha_:.4f})')
227
228     plt.tight_layout()
229     plt.savefig(os.path.join(self.figure_dir, '1b_glasso_precision_matrices.png'), dpi=150)
230     plt.show()
231
232     def calculate_bic_for_dag(self, data, G):
233         n_samples, n_features = data.shape
234         bic_total = 0
235
236         nodes = G.get_nodes()
237
238         for i, node in enumerate(nodes):
239             neighbors = G.get_adjacent_nodes(node)
240             neighbor_indices = [nodes.index(n) for n in neighbors]
241
242             y = data.iloc[:, i].values
243
244             if len(neighbor_indices) == 0:
245                 residuals = y - np.mean(y)
246             else:
247                 X = data.iloc[:, neighbor_indices].values
248                 model = LinearRegression()
249                 model.fit(X, y)
250                 residuals = y - model.predict(X)
251
252             variance = max(1e-9, np.var(residuals))
253
254             # Log-likelihood term for this node
255             log_likelihood = -0.5 * n_samples * (np.log(2 * np.pi * variance)
256                                                 + 1)

```

```

255     # Number of parameters = number of parents + 1 (intercept/variance
256     #                                     )
257     k = len(neighbor_indices) + 1
258
259     # BIC = k * ln(n) - 2 * ln(L)
260     bic_node = k * np.log(n_samples) - 2 * log_likelihood
261     bic_total += bic_node
262
263     return bic_total
264
265 def fit_pc_model(self, alphas=[0.001, 0.01, 0.05, 0.1, 0.2], verbose=False):
266     if self.log_returns is None:
267         self.process_stock_data()
268
269     data_np = self.log_returns.values
270     labels = self.log_returns.columns.tolist()
271
272     results = []
273     best_bic = float('inf')
274     best_graph = None
275     best_alpha = 0.05
276
277     if verbose:
278         print(f"Tuning PC Algorithm over alphas: {alphas}...")
279
280     for alpha in alphas:
281         cg = pc(data_np, alpha, "fisherz")
282         n_edges = cg.G.get_num_edges()
283         bic = self.calculate_bic_for_dag(self.log_returns, cg.G)
284
285         results.append({
286             'alpha': alpha,
287             'n_edges': n_edges,
288             'bic': bic,
289             'graph': cg.G
290         })
291
292     if verbose:
293         print(f" Alpha: {alpha:<6} | Edges: {n_edges:<4} | BIC: {bic:.2f}")
294
295     if bic < best_bic:
296         best_bic = bic
297         best_graph = cg.G
298         best_alpha = alpha
299
300     if verbose:
301         self._plot_pc_results(results, best_alpha, best_graph, labels)
302
303     return {
304         'best_graph': best_graph,
305         'best_alpha': best_alpha,
306         'results': results
307

```

```

306     }
307
308     def _plot_pc_results(self, results, best_alpha, best_graph, labels):
309         alphas_plot = [r['alpha'] for r in results]
310         edges_plot = [r['n_edges'] for r in results]
311         bics_plot = [r['bic'] for r in results]
312
313         fig, ax1 = plt.subplots(figsize=(10, 6))
314
315         color = 'tab:blue'
316         ax1.set_xlabel('Alpha (Significance Level)')
317         ax1.set_ylabel('Number of Edges', color=color)
318         ax1.plot(alphas_plot, edges_plot, marker='o', color=color, label='
319                         Sparsity (Edges)')
320         ax1.tick_params(axis='y', labelcolor=color)
321         ax1.grid(True, alpha=0.3)
322
323         ax2 = ax1.twinx()
324         color = 'tab:red'
325         ax2.set_ylabel('BIC Score (Lower is Better)', color=color)
326         ax2.plot(alphas_plot, bics_plot, marker='x', linestyle='--', color=
327                         color, label='BIC Score')
328         ax2.tick_params(axis='y', labelcolor=color)
329
330         plt.title(f'PC Algorithm Hyperparameter Tuning\nBest Alpha (min BIC):
331                         {best_alpha}')
332         fig.tight_layout()
333         plt.savefig(os.path.join(self.figure_dir, '1c_pc_tuning.png'), dpi=150
334 )
335         plt.show()
336
337         print(f"\nBest Alpha selected by BIC: {best_alpha}")
338         print(f"Number of edges in best graph: {best_graph.get_num_edges()}")
339
340         G_nx = nx.DiGraph()
341         nodes = best_graph.get_nodes()
342
343         for i in range(len(nodes)):
344             G_nx.add_node(i, label=labels[i])
345
346         graph_edges = best_graph.get_graph_edges()
347         directed_edges = []
348         undirected_edges = []
349
350         for edge in graph_edges:
351             node1 = edge.get_node1()
352             node2 = edge.get_node2()
353             idx1 = nodes.index(node1)
354             idx2 = nodes.index(node2)
355
356             endpoint1 = edge.get_endpoint1()
357             endpoint2 = edge.get_endpoint2()

```

```

355         ep1_name = endpoint1.name if hasattr(endpoint1, 'name') else str(
356                                         endpoint1)
357         ep2_name = endpoint2.name if hasattr(endpoint2, 'name') else str(
358                                         endpoint2)
359         if ep1_name == 'TAIL' and ep2_name == 'ARROW':
360             directed_edges.append((idx1, idx2))
361             G_nx.add_edge(idx1, idx2)
362         elif ep1_name == 'ARROW' and ep2_name == 'TAIL':
363             directed_edges.append((idx2, idx1))
364             G_nx.add_edge(idx2, idx1)
365         else:
366             undirected_edges.append((idx1, idx2))
367             G_nx.add_edge(idx1, idx2)
368             G_nx.add_edge(idx2, idx1)
369
370
371     plt.figure(figsize=(16, 16))
372     pos = nx.spring_layout(G_nx, k=2.5, iterations=100, seed=42)
373
374     nx.draw_networkx_nodes(G_nx, pos, node_size=2000, node_color='
375                               lightblue',
376                               alpha=0.9, edgecolors='navy', linewidths=2)
377     nx.draw_networkx_labels(G_nx, pos, {i: labels[i] for i in range(len(
378                               labels))},
379                               font_size=11, font_weight='bold')
380
381
382     if directed_edges:
383         nx.draw_networkx_edges(G_nx, pos, edgelist=directed_edges,
384                               edge_color='darkblue', arrows=True,
385                               arrowsize
386                               =25,
387                               arrowstyle='-'>', connectionstyle='arc3,rad=
388                               0.15'
389
390                               ,
391                               width=2.5, alpha=0.7, node_size=2000)
392
393
394     if undirected_edges:
395         nx.draw_networkx_edges(G_nx, pos, edgelist=undirected_edges,
396                               edge_color='red', arrows=False,
397                               width=2.0, alpha=0.5, style='dashed')
398
399
400     plt.title(f"Optimal Directed Graph (PC Algorithm, alpha={best_alpha})\
401               n"
402               f"Directed edges: {len(directed_edges)}, Undirected edges: {{
403                               len(undirected_edges)}}
404               ",
405               fontsize=14, pad=20)
406     plt.axis('off')
407     plt.tight_layout()
408     plt.savefig(os.path.join(self.figure_dir, '1c_pc_graph.png'), dpi=150,
409                 bbox_inches='tight')
410     plt.show()
411
412
413     def fit_granger_model(self, max_lag=5, significance_level=0.05,
414                           tune_params=True, verbose=False):

```

```

396     if self.log_returns is None:
397         self.process_stock_data()
398
399     # Hyperparameter tuning
400     if tune_params and verbose:
401         self._tune_granger_parameters(max_lag_range=[1, 3, 5, 7, 10],
402                                         alpha_range=[0.01, 0.05, 0.1])
403
404     n_features = len(self.tickers)
405     granger_matrix = np.zeros((n_features, n_features))
406     p_value_matrix = np.zeros((n_features, n_features))
407     optimal_lags = np.zeros((n_features, n_features), dtype=int)
408
409     if verbose:
410         print(f"\nRunning Granger Causality Tests (max_lag={max_lag},
411               alpha={significance_level})
412               ...")
413
414     for i, cause in enumerate(self.tickers):
415         for j, effect in enumerate(self.tickers):
416             if i == j:
417                 continue
418
419             data = self.log_returns[[effect, cause]].dropna()
420
421             try:
422                 test_result = grangercausalitytests(data, max_lag, verbose
423                                                     =False)
424
425                 min_p_value = 1.0
426                 best_lag = 0
427
428                 for lag in range(1, max_lag + 1):
429                     p_values = [test_result[lag][0][test][1]
430                                 for test in ['ssr_ftest', 'ssr_chi2test', 'lrtest',
431                                             ', 'params_ftest']]
432
433                 avg_p = np.mean(p_values)
434
435                 if avg_p < min_p_value:
436                     min_p_value = avg_p
437                     best_lag = lag
438
439                 if min_p_value < significance_level:
440                     granger_matrix[i, j] = 1
441                     p_value_matrix[i, j] = min_p_value
442                     optimal_lags[i, j] = best_lag
443
444             except Exception as e:
445                 if verbose:
446                     print(f"  Warning: Test failed for {cause} -> {effect}
447                           : {e}")

```

```

442         continue
443
444     granger_df = pd.DataFrame(granger_matrix, index=self.tickers, columns=
445                               self.tickers)
445     p_value_df = pd.DataFrame(p_value_matrix, index=self.tickers, columns=
446                               self.tickers)
446     lags_df = pd.DataFrame(optimal_lags, index=self.tickers, columns=self.
447                               tickers)
447
448     if verbose:
449         self._plot_granger_results(granger_df, p_value_df, lags_df,
450                                     significance_level)
450
451     return {
452         'granger_matrix': granger_df,
453         'p_values': p_value_df,
454         'optimal_lags': lags_df
455     }
456
457 def _tune_granger_parameters(self, max_lag_range, alpha_range):
458     print("=" * 70)
459     print("Hyperparameter Tuning for Granger Causality Test")
460     print("=" * 70)
461
462     results = []
463
464     for max_lag in max_lag_range:
465         for alpha in alpha_range:
466             n_features = len(self.tickers)
467             granger_matrix = np.zeros((n_features, n_features))
468
469             for i, cause in enumerate(self.tickers):
470                 for j, effect in enumerate(self.tickers):
471                     if i == j:
472                         continue
473
474                     data = self.log_returns[[effect, cause]].dropna()
475
476                     try:
477                         test_result = grangercausalitytests(data, max_lag,
478                                               verbose=
479                                               False)
480
481                         min_p_value = 1.0
482                         for lag in range(1, max_lag + 1):
483                             p_values = [test_result[lag][0][test][1]
484                                         for test in ['ssr_ftest', 'ssr_chi2
485                                         ,
486                                         ,
487                                         ,
488                                         ,
489                                         ,
490                                         ,
491                                         ,
492                                         ,
493                                         ,
494                                         ,
495                                         ,
496                                         ,
497                                         ,
498                                         ,
499                                         ,
500                                         ,
501                                         ,
502                                         ,
503                                         ,
504                                         ,
505                                         ,
506                                         ,
507                                         ,
508                                         ,
509                                         ,
510                                         ,
511                                         ,
512                                         ,
513                                         ,
514                                         ,
515                                         ,
516                                         ,
517                                         ,
518                                         ,
519                                         ,
520                                         ,
521                                         ,
522                                         ,
523                                         ,
524                                         ,
525                                         ,
526                                         ,
527                                         ,
528                                         ,
529                                         ,
530                                         ,
531                                         ,
532                                         ,
533                                         ,
534                                         ,
535                                         ,
536                                         ,
537                                         ,
538                                         ,
539                                         ,
540                                         ,
541                                         ,
542                                         ,
543                                         ,
544                                         ,
545                                         ,
546                                         ,
547                                         ,
548                                         ,
549                                         ,
550                                         ,
551                                         ,
552                                         ,
553                                         ,
554                                         ,
555                                         ,
556                                         ,
557                                         ,
558                                         ,
559                                         ,
560                                         ,
561                                         ,
562                                         ,
563                                         ,
564                                         ,
565                                         ,
566                                         ,
567                                         ,
568                                         ,
569                                         ,
570                                         ,
571                                         ,
572                                         ,
573                                         ,
574                                         ,
575                                         ,
576                                         ,
577                                         ,
578                                         ,
579                                         ,
580                                         ,
581                                         ,
582                                         ,
583                                         ,
584                                         ,
585                                         ,
586                                         ,
587                                         ,
588                                         ,
589                                         ,
590                                         ,
591                                         ,
592                                         ,
593                                         ,
594                                         ,
595                                         ,
596                                         ,
597                                         ,
598                                         ,
599                                         ,
599                                         ,
600                                         ,
601                                         ,
602                                         ,
603                                         ,
604                                         ,
605                                         ,
606                                         ,
607                                         ,
608                                         ,
609                                         ,
609                                         ,
610                                         ,
611                                         ,
612                                         ,
613                                         ,
614                                         ,
615                                         ,
616                                         ,
617                                         ,
618                                         ,
619                                         ,
619                                         ,
620                                         ,
621                                         ,
622                                         ,
623                                         ,
624                                         ,
625                                         ,
626                                         ,
627                                         ,
628                                         ,
629                                         ,
629                                         ,
630                                         ,
631                                         ,
632                                         ,
633                                         ,
634                                         ,
635                                         ,
636                                         ,
637                                         ,
638                                         ,
639                                         ,
639                                         ,
640                                         ,
641                                         ,
642                                         ,
643                                         ,
644                                         ,
645                                         ,
646                                         ,
647                                         ,
648                                         ,
649                                         ,
649                                         ,
650                                         ,
651                                         ,
652                                         ,
653                                         ,
654                                         ,
655                                         ,
656                                         ,
657                                         ,
658                                         ,
659                                         ,
659                                         ,
660                                         ,
661                                         ,
662                                         ,
663                                         ,
664                                         ,
665                                         ,
666                                         ,
667                                         ,
668                                         ,
669                                         ,
669                                         ,
670                                         ,
671                                         ,
672                                         ,
673                                         ,
674                                         ,
675                                         ,
676                                         ,
677                                         ,
678                                         ,
679                                         ,
679                                         ,
680                                         ,
681                                         ,
682                                         ,
683                                         ,
684                                         ,
685                                         ,
686                                         ,
687                                         ,
688                                         ,
689                                         ,
689                                         ,
690                                         ,
691                                         ,
692                                         ,
693                                         ,
694                                         ,
695                                         ,
696                                         ,
697                                         ,
698                                         ,
699                                         ,
699                                         ,
700                                         ,
701                                         ,
702                                         ,
703                                         ,
704                                         ,
705                                         ,
706                                         ,
707                                         ,
708                                         ,
709                                         ,
709                                         ,
710                                         ,
711                                         ,
712                                         ,
713                                         ,
714                                         ,
715                                         ,
716                                         ,
717                                         ,
718                                         ,
719                                         ,
719                                         ,
720                                         ,
721                                         ,
722                                         ,
723                                         ,
724                                         ,
725                                         ,
726                                         ,
727                                         ,
728                                         ,
729                                         ,
729                                         ,
730                                         ,
731                                         ,
732                                         ,
733                                         ,
734                                         ,
735                                         ,
736                                         ,
737                                         ,
738                                         ,
739                                         ,
739                                         ,
740                                         ,
741                                         ,
742                                         ,
743                                         ,
744                                         ,
745                                         ,
746                                         ,
747                                         ,
748                                         ,
749                                         ,
749                                         ,
750                                         ,
751                                         ,
752                                         ,
753                                         ,
754                                         ,
755                                         ,
756                                         ,
757                                         ,
758                                         ,
759                                         ,
759                                         ,
760                                         ,
761                                         ,
762                                         ,
763                                         ,
764                                         ,
765                                         ,
766                                         ,
767                                         ,
768                                         ,
769                                         ,
769                                         ,
770                                         ,
771                                         ,
772                                         ,
773                                         ,
774                                         ,
775                                         ,
776                                         ,
777                                         ,
778                                         ,
778                                         ,
779                                         ,
779                                         ,
780                                         ,
781                                         ,
782                                         ,
783                                         ,
784                                         ,
785                                         ,
786                                         ,
787                                         ,
788                                         ,
788                                         ,
789                                         ,
789                                         ,
790                                         ,
791                                         ,
792                                         ,
793                                         ,
794                                         ,
795                                         ,
796                                         ,
797                                         ,
798                                         ,
798                                         ,
799                                         ,
799                                         ,
800                                         ,
801                                         ,
802                                         ,
803                                         ,
804                                         ,
805                                         ,
806                                         ,
807                                         ,
808                                         ,
809                                         ,
809                                         ,
810                                         ,
811                                         ,
812                                         ,
813                                         ,
814                                         ,
815                                         ,
816                                         ,
817                                         ,
818                                         ,
819                                         ,
819                                         ,
820                                         ,
821                                         ,
822                                         ,
823                                         ,
824                                         ,
825                                         ,
826                                         ,
827                                         ,
828                                         ,
829                                         ,
829                                         ,
830                                         ,
831                                         ,
832                                         ,
833                                         ,
834                                         ,
835                                         ,
836                                         ,
837                                         ,
838                                         ,
839                                         ,
839                                         ,
840                                         ,
841                                         ,
842                                         ,
843                                         ,
844                                         ,
845                                         ,
846                                         ,
847                                         ,
848                                         ,
849                                         ,
849                                         ,
850                                         ,
851                                         ,
852                                         ,
853                                         ,
854                                         ,
855                                         ,
856                                         ,
857                                         ,
858                                         ,
859                                         ,
859                                         ,
860                                         ,
861                                         ,
862                                         ,
863                                         ,
864                                         ,
865                                         ,
866                                         ,
867                                         ,
868                                         ,
869                                         ,
869                                         ,
870                                         ,
871                                         ,
872                                         ,
873                                         ,
874                                         ,
875                                         ,
876                                         ,
877                                         ,
877                                         ,
878                                         ,
878                                         ,
879                                         ,
879                                         ,
880                                         ,
881                                         ,
882                                         ,
883                                         ,
884                                         ,
885                                         ,
886                                         ,
887                                         ,
888                                         ,
888                                         ,
889                                         ,
889                                         ,
890                                         ,
891                                         ,
892                                         ,
893                                         ,
894                                         ,
895                                         ,
896                                         ,
897                                         ,
897                                         ,
898                                         ,
898                                         ,
899                                         ,
899                                         ,
900                                         ,
901                                         ,
902                                         ,
903                                         ,
904                                         ,
905                                         ,
906                                         ,
907                                         ,
908                                         ,
909                                         ,
909                                         ,
910                                         ,
911                                         ,
912                                         ,
913                                         ,
914                                         ,
915                                         ,
916                                         ,
917                                         ,
918                                         ,
919                                         ,
919                                         ,
920                                         ,
921                                         ,
922                                         ,
923                                         ,
924                                         ,
925                                         ,
926                                         ,
927                                         ,
928                                         ,
929                                         ,
929                                         ,
930                                         ,
931                                         ,
932                                         ,
933                                         ,
934                                         ,
935                                         ,
936                                         ,
937                                         ,
938                                         ,
938                                         ,
939                                         ,
939                                         ,
940                                         ,
941                                         ,
942                                         ,
943                                         ,
944                                         ,
945                                         ,
945                                         ,
946                                         ,
946                                         ,
947                                         ,
947                                         ,
948                                         ,
948                                         ,
949                                         ,
949                                         ,
950                                         ,
951                                         ,
952                                         ,
953                                         ,
954                                         ,
955                                         ,
956                                         ,
957                                         ,
958                                         ,
959                                         ,
959                                         ,
960                                         ,
961                                         ,
962                                         ,
963                                         ,
964                                         ,
965                                         ,
966                                         ,
967                                         ,
968                                         ,
968                                         ,
969                                         ,
969                                         ,
970                                         ,
971                                         ,
972                                         ,
973                                         ,
974                                         ,
975                                         ,
975                                         ,
976                                         ,
976                                         ,
977                                         ,
977                                         ,
978                                         ,
978                                         ,
979                                         ,
979                                         ,
980                                         ,
981                                         ,
982                                         ,
983                                         ,
984                                         ,
985                                         ,
986                                         ,
987                                         ,
987                                         ,
988                                         ,
988                                         ,
989                                         ,
989                                         ,
990                                         ,
991                                         ,
992                                         ,
993                                         ,
994                                         ,
995                                         ,
995                                         ,
996                                         ,
996                                         ,
997                                         ,
997                                         ,
998                                         ,
998                                         ,
999                                         ,
999                                         ,
1000                                         ,
1001                                         ,
1002                                         ,
1003                                         ,
1004                                         ,
1005                                         ,
1006                                         ,
1007                                         ,
1008                                         ,
1008                                         ,
1009                                         ,
1009                                         ,
1010                                         ,
1011                                         ,
1012                                         ,
1013                                         ,
1014                                         ,
1015                                         ,
1016                                         ,
1017                                         ,
1018                                         ,
1018                                         ,
1019                                         ,
1019                                         ,
1020                                         ,
1021                                         ,
1022                                         ,
1023                                         ,
1024                                         ,
1025                                         ,
1026                                         ,
1027                                         ,
1028                                         ,
1028                                         ,
1029                                         ,
1029                                         ,
1030                                         ,
1031                                         ,
1032                                         ,
1033                                         ,
1034                                         ,
1035                                         ,
1036                                         ,
1037                                         ,
1038                                         ,
1038                                         ,
1039                                         ,
1039                                         ,
1040                                         ,
1041                                         ,
1042                                         ,
1043                                         ,
1044                                         ,
1045                                         ,
1046                                         ,
1047                                         ,
1048                                         ,
1048                                         ,
1049                                         ,
1049                                         ,
1050                                         ,
1051                                         ,
1052                                         ,
1053                                         ,
1054                                         ,
1055                                         ,
1056                                         ,
1057                                         ,
1058                                         ,
1058                                         ,
1059                                         ,
1059                                         ,
1060                                         ,
1061                                         ,
1062                                         ,
1063                                         ,
1064                                         ,
1065                                         ,
1066                                         ,
1067                                         ,
1068                                         ,
1068                                         ,
1069                                         ,
1069                                         ,
1070                                         ,
1071                                         ,
1072                                         ,
1073                                         ,
1074                                         ,
1075                                         ,
1076                                         ,
1076                                         ,
1077                                         ,
1077                                         ,
1078                                         ,
1078                                         ,
1079                                         ,
1079                                         ,
1080                                         ,
1081                                         ,
1082                                         ,
1083                                         ,
1084                                         ,
1085                                         ,
1086                                         ,
1087                                         ,
1087                                         ,
1088                                         ,
1088                                         ,
1089                                         ,
1089                                         ,
1090                                         ,
1091                                         ,
1092                                         ,
1093                                         ,
1094                                         ,
1095                                         ,
1095                                         ,
1096                                         ,
1096                                         ,
1097                                         ,
1097                                         ,
1098                                         ,
1098                                         ,
1099                                         ,
1099                                         ,
1100                                         ,
1101                                         ,
1102                                         ,
1103                                         ,
1104                                         ,
1105                                         ,
1106                                         ,
1107                                         ,
1108                                         ,
1108                                         ,
1109                                         ,
1109                                         ,
1110                                         ,
1111                                         ,
1112                                         ,
1113                                         ,
1114                                         ,
1115                                         ,
1116                                         ,
1117                                         ,
1118                                         ,
1118                                         ,
1119                                         ,
1119                                         ,
1120                                         ,
1121                                         ,
1122                                         ,
1123                                         ,
1124                                         ,
1125                                         ,
1126                                         ,
1127                                         ,
1127                                         ,
1128                                         ,
1128                                         ,
1129                                         ,
1129                                         ,
1130                                         ,
1131                                         ,
1132                                         ,
1133                                         ,
1134                                         ,
1135                                         ,
1136                                         ,
1137                                         ,
1137                                         ,
1138                                         ,
1138                                         ,
1139                                         ,
1139                                         ,
1140                                         ,
1141                                         ,
1142                                         ,
1143                                         ,
1144                                         ,
1145                                         ,
1146                                         ,
1147                                         ,
1147                                         ,
1148                                         ,
1148                                         ,
1149                                         ,
1149                                         ,
1150                                         ,
1151                                         ,
1152                                         ,
1153                                         ,
1154                                         ,
1155                                         ,
1156                                         ,
1157                                         ,
1157                                         ,
1158                                         ,
1158                                         ,
1159                                         ,
1159                                         ,
1160                                         ,
1161                                         ,
1162                                         ,
1163                                         ,
1164                                         ,
1165                                         ,
1166                                         ,
1167                                         ,
1167                                         ,
1168                                         ,
1168                                         ,
1169                                         ,
1169                                         ,
1170                                         ,
1171                                         ,
1172                                         ,
1173                                         ,
1174                                         ,
1175                                         ,
1176                                         ,
1176                                         ,
1177                                         ,
1177                                         ,
1178                                         ,
1178                                         ,
1179                                         ,
1179                                         ,
1180                                         ,
1181                                         ,
1182                                         ,
1183                                         ,
1184                                         ,
1185                                         ,
1186                                         ,
1187                                         ,
1187                                         ,
1188                                         ,
1188                                         ,
1189                                         ,
1189                                         ,
1190                                         ,
1191                                         ,
1192                                         ,
1193                                         ,
1194                                         ,
1195                                         ,
1195                                         ,
1196                                         ,
1196                                         ,
1197                                         ,
1197                                         ,
1198                                         ,
1198                                         ,
1199                                         ,
1199                                         ,
1200                                         ,
1201                                         ,
1202                                         ,
1203                                         ,
1204                                         ,
1205                                         ,
1206                                         ,
1207                                         ,
1208                                         ,
1208                                         ,
1209                                         ,
1209                                         ,
1210                                         ,
1211                                         ,
1212                                         ,
1213                                         ,
1214                                         ,
1215                                         ,
1216                                         ,
1217                                         ,
1217                                         ,
1218                                         ,
1218                                         ,
1219                                         ,
1219                                         ,
1220                                         ,
1221                                         ,
1222                                         ,
1223                                         ,
1224                                         ,
1225                                         ,
1226                                         ,
1226                                         ,
1227                                         ,
1227                                         ,
1228                                         ,
1228                                         ,
1229                                         ,
1229                                         ,
1230                                         ,
1231                                         ,
1232                                         ,
1233                                         ,
1234                                         ,
1235                                         ,
1236                                         ,
1237                                         ,
1237                                         ,
1238                                         ,
1238                                         ,
1239                                         ,
1239                                         ,
1240                                         ,
1241                                         ,
1242                                         ,
1243                                         ,
1244                                         ,
1245                                         ,
1246                                         ,
1246                                         ,
1247                                         ,
1247                                         ,
1248                                         ,
1248                                         ,
1249                                         ,
1249                                         ,
1250                                         ,
1251                                         ,
1252                                         ,
1253                                         ,
1254                                         ,
1255                                         ,
1256                                         ,
1257                                         ,
1257                                         ,
1258                                         ,
1258                                         ,
1259                                         ,
1259                                         ,
1260                                         ,
1261                                         ,
1262                                         ,
1263                                         ,
1264                                         ,
1265                                         ,
1266                                         ,
1267                                         ,
1267                                         ,
1268                                         ,
1268                                         ,
1269                                         ,
1269                                         ,
1270                                         ,
1271                                         ,
1272                                         ,
1273                                         ,
1274                                         ,
1275                                         ,
1276                                         ,
1276                                         ,
1277                                         ,
1277                                         ,
1278                                         ,
1278                                         ,
1279                                         ,
1279                                         ,
1280                                         ,
1281                                         ,
1282                                         ,
1283                                         ,
1284                                         ,
1285                                         ,
1286                                         ,
1287                                         ,
1287                                         ,
1288                                         ,
1288                                         ,
1289                                         ,
1289                                         ,
1290                                         ,
1291                                         ,
1292                                         ,
1293                                         ,
1294                                         ,
1295                                         ,
1295                                         ,
1296                                         ,
1296                                         ,
1297                                         ,
1297                                         ,
1298                                         ,
1298                                         ,
1299                                         ,
1299                                         ,
1300                                         ,
1301                                         ,
1302                                         ,
1303                                         ,
1304                                         ,
1305                                         ,
1306                                         ,
1307                                         ,
1308                                         ,
1308                                         ,
1309                                         ,
1309                                         ,
1310                                         ,
1311                                         ,
1312                                         ,
1313                                         ,
1314                                         ,
1315                                         ,
1316                                         ,
1316                                         ,
1317                                         ,
1317                                         ,
1318                                         ,
1318                                         ,
1319                                         ,
1319                                         ,
1320                                         ,
1321                                         ,
1322                                         ,
1323                                         ,
1324                                         ,
1325                                         ,
1326                                         ,
1326                                         ,
1327                                         ,
1327                                         ,
1328                                         ,
1328                                         ,
1329                                         ,
1329                                         ,
1330                                         ,
1331                                         ,
1332                                         ,
1333                                         ,
1334                                         ,
1335                                         ,
1336                                         ,
1337                                         ,
1337                                         ,
1338                                         ,
1338                                         ,
1339                                         ,
1339                                         ,
1340                                         ,
1341                                         ,
1342                                         ,
1343                                         ,
1344                                         ,
1345                                         ,
1346                                         ,
1346                                         ,
1347                                         ,
1347                                         ,
1348                                         ,
1348                                         ,
1349                                         ,
1349                                         ,
1350                                         ,
1351                                         ,
1352                                         ,
1353                                         ,
1354                                         ,
1355                                         ,
1356                                         ,
1357                                         ,
1357                                         ,
1358                                         ,
1358                                         ,
1359                                         ,
1359                                         ,
1360                                         ,
1361                                         ,
1362                                         ,
1363                                         ,
1364                                         ,
1365                                         ,
1366                                         ,
1366                                         ,
1367                                         ,
1367                                         ,
1368                                         ,
1368                                         ,
1369                                         ,
1369                                         ,
1370                                         ,
1371                                         ,
1372                                         ,
1373                                         ,
1374                                         ,
1375                                         ,
1376                                         ,
1376                                         ,
1377                                         ,
1377                                         ,
1378                                         ,
1378                                         ,
1379                                         ,
1379                                         ,
1380                                         ,
1381                                         ,
1382                                         ,
1383                                         ,
1384                                         ,
1385                                         ,
1386                                         ,
1386                                         ,
1387                                         ,
1387                                         ,
1388                                         ,
1388                                         ,
1389                                         ,
1389                                         ,
1390                                         ,
1391                                         ,
1392                                         ,
1393                                         ,
1394                                         ,
1395                                         ,
1395                                         ,
1396                                         ,
1396                                         ,
1397                                         ,
1397                                         ,
1398                                         ,
1398                                         ,
1399                                         ,
1399                                         ,
1400                                         ,
1401                                         ,
1402                                         ,
1403                                         ,
1404                                         ,
1405                                         ,
1406                                         ,
1406                                         ,
1407                                         ,
1407                                         ,
1408                                         ,
1408                                         ,
1409                                         ,
1409                                         ,
1410                                         ,
1411                                         ,
1412                                         ,
1413                                         ,
1414                                         ,
1415                                         ,
1416                                         ,
1416                                         ,
1417                                         ,
1417                                         ,
1418                                         ,
1418                                         ,
1419                                         ,
1419                                         ,
1420                                         ,
1421                                         ,
1422                                         ,
1423                                         ,
1424                                         ,
1425                                         ,
1425                                         ,
1426                                         ,
1426                                         ,
1427                                         ,
1427                                         ,
1428                                         ,
1428                                         ,
1429                                         ,
1429                                         ,
1430                                         ,
1431                                         ,
1432                                         ,
1433                                         ,
1434                                         ,
1435                                         ,
1436                                         ,
1436                                         ,
1437                                         ,
1437                                         ,
1438                                         ,
1438                                         ,
1439                                         ,
1439                                         ,
1440                                         ,
1441                                         ,
1442                                         ,
1443                                         ,
1444                                         ,
1444                                         ,
1445                                         ,
1445                                         ,
1446                                         ,
1446                                         ,
1447                                         ,
1447                                         ,
1448                                         ,
1448                                         ,
1449                                         ,
1449                                         ,
1450                                         ,
1451                                         ,
1452                                         ,
1452                                         ,
1453                                         ,
1453                                         ,
1454                                         ,
1454                                         ,
1455                                         ,
1455                                         ,
1456                                         ,
1456                                         ,
1457                                         ,
1457                                         ,
1458                                         ,
1458                                         ,
1459                                         ,
1459                                         ,
1460                                         ,
1461                                         ,
1462                                         ,
1463                                         ,
1464                                         ,
1465                                         ,
1466                                         ,
1466                                         ,
1467                                         ,
1467                                         ,
1468                                         ,
1468                                         ,
1469                                         ,
1469                                         ,
1470                                         ,
1471                                         ,
1472                                         ,
1473                                         ,
1474                                         ,
1475                                         ,
1476                                         ,
1476                                         ,
1477                                         ,
1477                                         ,
1478                                         ,
1478                                         ,
1479                                         ,
1479                                         ,
1480                                         ,
1481                                         ,
1482                                         ,
1483                                         ,
1484                                         ,
1485                                         ,
1486                                         ,
1486                                         ,
1487                                         ,
1487                                         ,
1488                                         ,
1488                                         ,
1489                                         ,
1489                                         ,
1490                                         ,
1491                                         ,
1492                                         ,
1493                                         ,
1494                                         ,
1495                                         ,
1495                                         ,
1496                                         ,
1496                                         ,
1497                                         ,
1497                                         ,
1498                                         ,
1498                                         ,
1499                                         ,
1499                                         ,
1500                                         ,
1501                                         ,
1502                                         ,
1503                                         ,
1504                                         ,
1505                                         ,
1506                                         ,
1506                                         ,
1507                                         ,
1507                                         ,
1508                                         ,
1508                                         ,
1509                                         ,
1509                                         ,
1510                                         ,
1511                                         ,
1512                                         ,
1513                                         ,
1514                                         ,
1515                                         ,
1516                                         ,
1516                                         ,
1517                                         ,
1517                                         ,
1518                                         ,
1518                                         ,
1519                                         ,
1519                                         ,
1520                                         ,
1521                                         ,
1522                                         ,
1523                                         ,
1524                                         ,
1525                                         ,
1525                                         ,
1526                                         ,
1526                                         ,
1527                                         ,
1527                                         ,
1528                                         ,
1528                                         ,
1529                                         ,
1529                                         ,
1530                                         ,
1531                                         ,
1532                                         ,
1533                                         ,
1534                                         ,
1535                                         ,
1536                                         ,
1536                                         ,
1537                                         ,
1537                                         ,
1538                                         ,
1538                                         ,
1539                                         ,
1539                                         ,
1540                                         ,
1541                                         ,
1542                                         ,
1543                                         ,
1544                                         ,
1545                                         ,
1545                                         ,
1546                                         ,
1546                                         ,
1547                                         ,
1547                                         ,
1548                                         ,
1548                                         ,
1549                                         ,
1549                                         ,
1550                                         ,
1551                                         ,
1552                                         ,
1553                                         ,
1554                                         ,
1555                                         ,
1556                                         ,
1556                                         ,
1557                                         ,
1557                                         ,
1558                                         ,
1558                                         ,
1559                                         ,
1559                                         ,
1560                                         ,
1561                                         ,
1562                                         ,
1563                                         ,
1564                                         ,
1565                                         ,
1565                                         ,
1566                                         ,
1566                                         ,
1567                                         ,
1567                                         ,
1568                                         ,
1568                                         ,
1569                                         ,
1569                                         ,
1570                                         ,
1571                                         ,
1572                                         ,
1573                                         ,
1574                                         ,
1575                                         ,
1575                                         ,
1576                                         ,
1576                                         ,
1577                                         ,
1577                                         ,
1578                                         ,
1578                                         ,
1579                                         ,
1579                                         ,
1580                                         ,
1581                                         ,
1582                                         ,
1583                                         ,
1584                                         ,
1585                                         ,
1585                                         ,
1586                                         ,
1586                                         ,
1587                                         ,
1587                                         ,
1588                                         ,
1588                                         ,
1589                                         ,
1589                                         ,
1590                                         ,
1591                                         ,
1592                                         ,
1593                                         ,
1594                                         ,
1595                                         ,
1595                                         ,
1596                                         ,
1596                                         ,
1597                                         ,
1597                                         ,
1598                                         ,
1598                                         ,
1599                                         ,
1599                                         ,
1600                                         ,
1601                                         ,
1602                                         ,
1603                                         ,
1604                                         ,
1605                                         ,
1606                                         ,
1606                                         ,
1607                                         ,
1607                                         ,
1608                                         ,
1608                                         ,
1609                                         ,
1609                                         ,
1610                                         ,
1611                                         ,
1612                                         ,
1613                                         ,
1614                                         ,
1615                                         ,
1615                                         ,
1616                                         ,
1616                                         ,
1617                                         ,
1617                                         ,
1618                                         ,
1618                                         ,
1619                                         ,
1619                                         ,
1620                                         ,
1621                                         ,
1622                                         ,
1623                                         ,
1624                                         ,
1625                                         ,
1625                                         ,
1626                                         ,
1626                                         ,
1627                                         ,
1627                                         ,
1628                                         ,
1628                                         ,
1629                                         ,
1629                                         ,
1630                                         ,
1631                                         ,
1632                                         ,
1633                                         ,
1634                                         ,
1635                                         ,
1635                                         ,
1636                                         ,
1636                                         ,
1637                                         ,
1637                                         ,
1638                                         ,
1638                                         ,
1639                                         ,
1639                                         ,
1640                                         ,
1641                                         ,
1642                                         ,
1643                                         ,
1644                                         ,
1645                                         ,
1645                                         ,
1646                                         ,
1646                                         ,
1647                                         ,
1647                                         ,
1648                                         ,
1648                                         ,
1649                                         ,
1649                                         ,
1650                                         ,
1651                                         ,
1652                                         ,
1653                                         ,
1654                                         ,
1655                                         ,
1655                                         ,
1656                                         ,
1656                                         ,
1657                                         ,
1657                                         ,
1658                                         ,
1658                                         ,
1659                                         ,
1659                                         ,
1660                                         ,
1661                                         ,
1662                                         ,
1663                                         ,
1664                                         ,
1665                                         ,
1665                                         ,
1666                                         ,
1666                                         ,
1667                                         ,
1667                                         ,
1668                                         ,
1668                                         ,
1669                                         ,
1669                                         ,
1670                                         ,
1671                                         ,
1672                                         ,
1673                                         ,
1674                                         ,
1675                                         ,
1675                                         ,
1676                                         ,
1676                                         ,
1677                                         ,
1677                                         ,
1678                                         ,
1678                                         ,
1679                                         ,
1679                                         ,
1680                                         ,
1681                                         ,
1682                                         ,
1683                                         ,
1684                                         ,
1685                                         ,
1685                                         ,
1686                                         ,
1686                                         ,
1687                                         ,
1687                                         ,
1688                                         ,
1688                                         ,
1689                                         ,
1689                                         ,
1690                                         ,
1691                                         ,
1692                                         ,
1693                                         ,
1694                                         ,
1695                                         ,
1695                                         ,
1696                                         ,
1696                                         ,
1697                                         ,
1697                                         ,
1698                                         ,
1698                                         ,
1699                                         ,
1699                                         ,
1700                                         ,
1701                                         ,
1702                                         ,
1703                                         ,
1704                                         ,
1705                                         ,
1705                                         ,
1706                                         ,
1706                                         ,
1707                                         ,
1707                                         ,
1708                                         ,
1708                                         ,
1709                                         ,
1709                                         ,
1710                                         ,
1711                                         ,
1712                                         ,
1713                                         ,
1714                                         ,
1715                                         ,
1715                                         ,
1716                                         ,
1716                                         ,
1717                                         ,
1717                                         ,
1718                                         ,
1718                                         ,
1719                                         ,
1719                                         ,
1720                                         ,
1721                                         ,
1722                                         ,
1723                                         ,
1724                                         ,
1725                                         ,
1725                                         ,
1726                                         ,
1726                                         ,
1727                                         ,
1727                                         ,
1728                                         ,
1728                                         ,
1729                                         ,
1729                                         ,
1730                                         ,
1731                                         ,
1732                                         ,
1733                                         ,
1734                                         ,
1735                                         ,
1735                                         ,
1736                                         ,
1736                                         ,
1737                                         ,
1737                                         ,
1738                                         ,
1738                                         ,
1739                                         ,
1739                                         ,
1740                                         ,
1741                                         ,
1742                                         ,
1743                                         ,
1744                                         ,
1745                                         ,
1745                                         ,
1746                                         ,
1746                                         ,
1747                                         ,
1747                                         ,
1748                                         ,
1748                                         ,
1749                                         ,
1749                                         ,
1750                                         ,
1751                                         ,
1752                                         ,
1753                                         ,
1754                                         ,
1755                                         ,
1755                                         ,
1756                                         ,
1756                                         ,
1757                                         ,
1757                                         ,
1758                                         ,
1758                                         ,
1759                                         ,
1759                                         ,
1760                                         ,
1761                                         ,
1762                                         ,
1763                                         ,
1764                                         ,
1765                                         ,
1765                                         ,
1766                                         ,
1766                                         ,
1767                                         ,
1767                                         ,
1768                                         ,
1768                                         ,
1769                                         ,
1769                                         ,
1770                                         ,
1771                                         ,
1772                                         ,
1773                                         ,
1774                                         ,
1775                                         ,
1775                                         ,
1776                                         ,
1776                                         ,
1777                                         ,
1777                                         ,
1778                                         ,
1778                                         ,
1779                                         ,
1779                                         ,
1780                                         ,
1781                                         ,
1782                                         ,
1783                                         ,
1784                                         ,
1785                                         ,
1785                                         ,
1786                                         ,
1786                                         ,
1787                                         ,
1787                                         ,
1788                                         ,
1788                                         ,
1789                                         ,
1789                                         ,
1790                                         ,
1791                                         ,
1792                                         ,
1793                                         ,
1794                                         ,
1795                                         ,
1795                                         ,
1796                                         ,
1796                                         ,
1797                                         ,
1797                                         ,
1798                                         ,
1798                                         ,
1799                                         ,
1799                                         ,
1800                                         ,
1801                                         ,
1802                                         ,
1803                                         ,
1804                                         ,
1805                                         ,
1805                                         ,
1806                                         ,
1806                                         ,
1807                                         ,
1807                                         ,
1808                                         ,
1808                                         ,
1809                                         ,
1809                                         ,
1810                                         ,
1811                                         ,
1812                                         ,
1813                                         ,
1814                                         ,
1815                                         ,
1815                                         ,
1816                                         ,
1816                                         ,
1817                                         ,
1817                                         ,
1818                                         ,
1818                                         ,
1819                                         ,
1819                                         ,
1820                                         ,
1821                                         ,
1822                                         ,
1823                                         ,
1824                                         ,
1825                                         ,
1825                                         ,
1826                                         ,
1826                                         ,
1827                                         ,
1827                                         ,
1828                                         ,
1828                                         ,
1829                                         ,
1829                                         ,
1830                                         ,
1831                                         ,
1832                                         ,
1833                                         ,
1834                                         ,
1835                                         ,
1835                                         ,
1836                                         ,
1836                                         ,
1837                                         ,
1837                                         ,
1838                                         ,
1838                                         ,
1839                                         ,
1839                                         ,
1840                                         ,
1841                                         ,
1842                                         ,
1843                                         ,
1844                                         ,
1845                                         ,
1845                                         ,
1846                                         ,
1846                                         ,
1847                                         ,
1847                                         ,
1848                                         ,
1848                                         ,
1849                                         ,
1849                                         ,
1850                                         ,
1851                                         ,
1852                                         ,
1853                                         ,
1854                                         ,
1855                                         ,
1855                                         ,
1856                                         ,
1856                                         ,
1857                                         ,
1857                                         ,
1858                                         ,
1858                                         ,
1859                                         ,
1859                                         ,
1860                                         ,
1861                                         ,
1862                                         ,
1863                                         ,
1864                                         ,
1865                                         ,
1865                                         ,
1866                                         ,
1866                                         ,
1867                                         ,
1867                                         ,
1868                                         ,
1868                                         ,
1869                                         ,
1869                                         ,
1870                                         ,
1871                                         ,
1872                                         ,
1873                                         ,
1874                                         ,
1875                                         ,
1875                                         ,
1876                                         ,
1876                                         ,
1877                                         ,
1877                                         ,
1878                                         ,
1878                                         ,
1879                                         ,
1879                                         ,
1880                                         ,
1881                                         ,
1882                                         ,
1883                                         ,
1884                                         ,
1885                                         ,
1885                                         ,
1886                                         ,
1886                                         ,
1887                                         ,
1887                                         ,
1888                                         ,
1888                                         ,
1889                                         ,
1889                                         ,
1890                                         ,
1891                                         ,
1892                                         ,
1893                                         ,
1894                                         ,
1895                                         ,
1895                                         ,
1896                                         ,
1896                                         ,
1897                                         ,
1897                                         ,
1898                                         ,
1898                                         ,
1899                                         ,
1899                                         ,
1900                                         ,
1901                                         ,
1902                                         ,
1903                                         ,
1904                                         ,
1905                                         ,
1905                                         ,
1906                                         ,
1906                                         ,
1907                                         ,
1907                                         ,
1908                                         ,
1908                                         ,
1909                                         ,
1909                                         ,
1910                                         ,
1911                                         ,
1912                                         ,
1913                                         ,
1914                                         ,
1915                                         ,
1915                                         ,
1916                                         ,
1916                                         ,
1917                                         ,
1917                                         ,
1918                                         ,
1918                                         ,
1919                                         ,
1919                                         ,
1920                                         ,
1921                                         ,
1922                                         ,
1923                                         ,
1924                                         ,
1925                                         ,
1925                                         ,
1926                                         ,
1926                                         ,
1927                                         ,
1927                                         ,
1928                                         ,
1928                                         ,
19
```

```

        ,
        '
        params_f
        '
        ]
    ]

483     avg_p = np.mean(p_values)
484     if avg_p < min_p_value:
485         min_p_value = avg_p
486
487     if min_p_value < alpha:
488         granger_matrix[i, j] = 1
489
490     except:
491         continue
492
493     n_edges = int(granger_matrix.sum())
494     density = n_edges / (n_features * (n_features - 1))
495
496     results.append({
497         'max_lag': max_lag,
498         'alpha': alpha,
499         'n_edges': n_edges,
500         'density': density
501     })
502
503     results_df = pd.DataFrame(results)
504     print("\nParameter Tuning Results:")
505     print(results_df.to_string(index=False))
506
507     # Visualization
508     pivot_edges = results_df.pivot(index='max_lag', columns='alpha',
509                                     values='n_edges')
510     pivot_density = results_df.pivot(index='max_lag', columns='alpha',
511                                     values='density')
512
513     _, axes = plt.subplots(1, 2, figsize=(16, 6))
514
515     sns.heatmap(pivot_edges, annot=True, fmt='0f', cmap='YlOrRd',
516                  ax=axes[0], cbar_kws={'label': 'Number of Edges'})
517     axes[0].set_title('Number of Causal Edges vs Hyperparameters')
518     axes[0].set_xlabel('Significance Level ()')
519     axes[0].set_ylabel('Max Lag')
520
521     sns.heatmap(pivot_density, annot=True, fmt='.3f', cmap='viridis',
522                  ax=axes[1], cbar_kws={'label': 'Graph Density'})
523     axes[1].set_title('Graph Density vs Hyperparameters')
524     axes[1].set_xlabel('Significance Level ()')
525     axes[1].set_ylabel('Max Lag')

526     plt.tight_layout()

```

```

526     plt.savefig(os.path.join(self.figure_dir, '1e_granger_tuning.png'),
527                  dpi=150)
528     plt.show()
529
530     print("\n" + "=" * 70)
531
532     def _plot_granger_results(self, granger_df, p_value_df, lags_df,
533                               significance_level):
534         n_edges = int(granger_df.sum().sum())
535
536         print("\n--- Granger Causality Test Results ---")
537         print(f"Significance level: {significance_level}")
538         print(f"Number of significant Granger causal relationships: {n_edges}")
539
540         print(f"Graph density: {n_edges / (len(self.tickers) * (len(self.
541                                         tickers) - 1)):.3f}")
542
543         # Plot 1: Granger causality adjacency matrix
544         _, axes = plt.subplots(1, 2, figsize=(20, 8))
545
546         sns.heatmap(granger_df, annot=True, fmt='.'0f', cmap='Blues',
547                     xticklabels=self.tickers, yticklabels=self.tickers,
548                     ax=axes[0], cbar_kws={'label': 'Granger Causes (1=Yes, 0=No
549 )'})
550         axes[0].set_title(f'Granger Causality Matrix ({significance_level})\
551                         nRow causes Column')
552         axes[0].set_xlabel('Effect (Y)')
553         axes[0].set_ylabel('Cause (X)')
554
555         p_value_masked = p_value_df.copy()
556         p_value_masked[granger_df == 0] = np.nan
557
558         sns.heatmap(p_value_masked, annot=True, fmt='.'3f', cmap='Reds_r',
559                     xticklabels=self.tickers, yticklabels=self.tickers,
560                     ax=axes[1], cbar_kws={'label': 'P-value'}, vmin=0, vmax=
561                         significance_level)
562         axes[1].set_title(f'P-values for Significant Relationships')
563         axes[1].set_xlabel('Effect (Y)')
564         axes[1].set_ylabel('Cause (X)')
565
566         plt.tight_layout()
567         plt.savefig(os.path.join(self.figure_dir, '1e_granger_matrix.png'),
568                     dpi=150)
569         plt.show()
570
571         # Plot 2: Network graph
572         G = nx.DiGraph()
573         for ticker in self.tickers:
574             G.add_node(ticker)
575
576         edge_list = []
577         for i, cause in enumerate(self.tickers):
578             for j, effect in enumerate(self.tickers):
579                 if granger_df.iloc[i, j] == 1:
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
999

```

```

572         edge_list.append((cause, effect))
573         G.add_edge(cause, effect,
574                     weight=1 - p_value_df.iloc[i, j],
575                     lag=int(lags_df.iloc[i, j]))
576
577     plt.figure(figsize=(14, 14))
578     pos = nx.spring_layout(G, k=2, iterations=100, seed=42)
579
580     nx.draw_networkx_nodes(G, pos, node_size=2500, node_color='lightblue',
581                           alpha=0.9, edgecolors='navy', linewidths=2)
582     nx.draw_networkx_labels(G, pos, font_size=11, font_weight='bold')
583
584     if edge_list:
585         nx.draw_networkx_edges(G, pos, edgelist=edge_list,
586                               edge_color='darkblue', arrows=True,
587                               arrowsize
588                               =25,
589                               arrowstyle='->', connectionstyle='arc3,rad=
590                               0.1',
591                               width=2, alpha=0.7, node_size=2500)
592
593     plt.title(f"Granger Causality Network ({significance_level})\n"
594               f"{n_edges} significant causal relationships",
595               fontsize=14, pad=20)
596     plt.axis('off')
597     plt.tight_layout()
598     plt.savefig(os.path.join(self.figure_dir, '1e_granger_network.png'),
599                 dpi=150, bbox_inches='tight')
599     plt.show()
600
601     # Summary statistics
602     print("\nTop 10 strongest Granger causal relationships:")
603     relationships = []
604     for i, cause in enumerate(self.tickers):
605         for j, effect in enumerate(self.tickers):
606             if granger_df.iloc[i, j] == 1:
607                 relationships.append({
608                     'Cause': cause,
609                     'Effect': effect,
610                     'P-value': p_value_df.iloc[i, j],
611                     'Lag': int(lags_df.iloc[i, j])
612                 })
613
614     if relationships:
615         relationships_df = pd.DataFrame(relationships).sort_values('P-
616                                     value')
616         print(relationships_df.head(10).to_string(index=False))

```

A.2 Problem 2a: KDE

A.2.1 Main Script.

```
1 """
2 Author: Chuyang Su cs4570@columbia.edu
3 Date: 2025-11-24 19:58:17
4 LastEditTime: 2025-11-25 09:37:20
5 FilePath: /Unsupervised-Learning-Homework/Homework 3/Code/Prob2_main.py
6 Description:
7     Main script to run data processing and clustering model fitting for
8         Homework 3.
9     Optimized flow:
10    1. process_data() -> X, y
11    2. fit_kde_models() -> trained models
12    3. generate_samples() -> generated data
13    4. evaluate_samples() -> metrics
14    5. visualize_generated_samples() + compare_distributions()
15
16
17
18 def main():
19     # Initialize analysis
20     p2 = Prob2Analysis(
21         output_dir='Homework 3/Code/Data',
22         figure_dir='Homework 3/Latex/Figures'
23     )
24     X, y = p2.process_data(verbose=False)
25
26     kde_results = p2.fit_kde_models(
27         spaces=['pca_20', 'pca_50', 'original'],
28         bandwidth_range=None, # Auto-generate
29         kernel='gaussian',
30         cv_folds=5,
31         verbose=False
32     )
33
34     # Generate from PCA-20 space (fastest and often best quality)
35     generated_pca20 = p2.generate_samples(
36         space='pca_20',
37         n_samples=100,
38         verbose=True
39     )
40
41     metrics = p2.evaluate_samples(verbose=True)
42
43     # Visualization and comparison
44     print("\nVisualizing generated samples...")
45     p2.visualize_generated_samples(n_display=25)
46
47     print("\nComparing original vs generated distributions...")
48     p2.compare_distributions()
49
50
51 if __name__ == "__main__":
52     main()
```

A.2.2 Utils Script.

```

1 '''
2 Author: Chuyang Su cs4570@columbia.edu
3 Date: 2025-11-24 19:58:10
4 LastEditTime: 2025-11-25 09:43:24
5 FilePath: /Unsupervised-Learning-Homework/Homework 3/Code/Prob2a_utils.py
6 Description:
7     Utility functions and classes for Problem 2 of Homework 3.
8     Kernel Density Estimation (KDE) for digit generation.
9     - process_data: Load and preprocess sklearn digits dataset
10    - fit_kde_models: Fit KDE in multiple spaces with hyperparameter tuning
11    - generate_samples: Generate new samples from trained KDE
12    - evaluate_samples: Evaluate quality of generated samples
13 '''
14 import os
15 import numpy as np
16 import pandas as pd
17 import matplotlib.pyplot as plt
18 import seaborn as sns
19 from sklearn.datasets import load_digits
20 from sklearn.preprocessing import StandardScaler
21 from sklearn.decomposition import PCA
22 from sklearn.neighbors import KernelDensity
23 from sklearn.model_selection import KFold
24 from sklearn.metrics import silhouette_score
25 from scipy.spatial.distance import cdist
26
27
28 class Prob2Analysis:
29     def __init__(self,
30                  output_dir='Homework 3/Code/Data',
31                  figure_dir='Homework 3/Latex/Figures'):
32         self.output_dir = output_dir
33         self.figure_dir = figure_dir
34
35         os.makedirs(self.output_dir, exist_ok=True)
36         os.makedirs(self.figure_dir, exist_ok=True)
37
38         self.data = None
39         self.targets = None
40         self.kde_results = None
41         self.generated_samples = None
42
43     def process_data(self, verbose=False):
44         digits = load_digits()
45         X = digits.data
46         y = digits.target
47
48         self.data = X
49         self.targets = y
50
51     if verbose:
52         print(f"\nDataset shape: {X.shape}")

```

```

53     print(f"Number of classes: {len(np.unique(y))}")
54     print(f"Pixel value range: [{X.min():.1f}, {X.max():.1f}]")
55     print(f"Mean pixel value: {X.mean():.2f} ± {X.std():.2f}")
56
57     return X, y
58
59 def fit_kde_models(self,
60                     spaces=['pca_20', 'pca_50', 'original'],
61                     bandwidth_range=None,
62                     kernel='gaussian',
63                     cv_folds=5,
64                     verbose=False):
65     if self.data is None:
66         self.process_data()
67
68     if bandwidth_range is None:
69         bandwidth_range = np.logspace(-1.5, 0.5, 20)
70
71     results = {}
72
73     if verbose:
74         print("Hyperparameter Tuning: Bandwidth Selection")
75
76     for space in spaces:
77         # Prepare data in target space
78         if space == 'original':
79             X_space = StandardScaler().fit_transform(self.data)
80             n_features = self.data.shape[1]
81
82         else:
83             n_comp = int(space.split('_')[1])
84             pca = PCA(n_components=n_comp)
85             X_space = pca.fit_transform(self.data)
86             X_space = StandardScaler().fit_transform(X_space)
87             n_features = n_comp
88
89         # Hyperparameter tuning
90         kde = KernelDensity(kernel=kernel, algorithm='ball_tree')
91         best_bandwidth = None
92         best_score = -np.inf
93         cv_scores_dict = {'bandwidth': [], 'mean_score': [], 'std_score': []}
94
95         kf = KFold(n_splits=cv_folds, shuffle=True, random_state=25)
96
97         for bw in bandwidth_range:
98             fold_scores = []
99             for train_idx, test_idx in kf.split(X_space):
100                 X_train, X_test = X_space[train_idx], X_space[test_idx]
101                 kde = KernelDensity(bandwidth=bw)
102                 kde.fit(X_train)
103                 score = kde.score(X_test)
104                 fold_scores.append(score)
105

```

```

106         mean_score = np.mean(fold_scores)
107         std_score = np.std(fold_scores)
108
109         cv_scores_dict['bandwidth'].append(bw)
110         cv_scores_dict['mean_score'].append(mean_score)
111         cv_scores_dict['std_score'].append(std_score)
112
113     if mean_score > best_score:
114         best_score = mean_score
115         best_bandwidth = bw
116
117     if verbose:
118         print(f"  Best bandwidth: {best_bandwidth:.6f}")
119         print(f"  Cross-validation score: {best_score:.4f}")
120
121     # Train final model
122     kde_final = KernelDensity(
123         kernel=kernel,
124         bandwidth=best_bandwidth,
125         algorithm='ball_tree'
126     )
127     kde_final.fit(X_space)
128
129     results[space] = {
130         'model': kde_final,
131         'bandwidth': best_bandwidth,
132         'cv_score': best_score,
133         'X_train': X_space,
134         'cv_results': cv_scores_dict,
135         'n_features': n_features,
136         'pca': PCA(n_components=int(space.split('_')[1]))
137             if space.startswith('pca_') else None
138     }
139
140     self.kde_results = results
141
142     if verbose:
143         self._plot_kde_tuning_results(results)
144
145     return results
146
147 def _plot_kde_tuning_results(self, results):
148     """Visualize bandwidth tuning for each space."""
149     n_spaces = len(results)
150     fig, axes = plt.subplots(1, n_spaces, figsize=(5*n_spaces, 4))
151
152     if n_spaces == 1:
153         axes = [axes]
154
155     for idx, (space, result) in enumerate(results.items()):
156         cv_results = result['cv_results']
157         bandwidths = cv_results['bandwidth']
158         mean_scores = cv_results['mean_score']
159         std_scores = cv_results['std_score']

```

```

160
161     axes[idx].semilogx(bandwidths, mean_scores, 'o-', linewidth=2,
162                           markersize=6)
163     axes[idx].fill_between(bandwidths,
164                            np.array(mean_scores) - np.array(std_scores),
165                            np.array(mean_scores) + np.array(std_scores),
166                            alpha=0.2)
167     axes[idx].axvline(result['bandwidth'], linestyle='--', color='r',
168                         label=f"Best: {result['bandwidth']:.4f}")
169     axes[idx].set_xlabel('Bandwidth')
170     axes[idx].set_ylabel('Mean Log-Likelihood (Higher is Better)')
171     axes[idx].set_title(f'Bandwidth Tuning: {space}')
172     axes[idx].legend()
173     axes[idx].grid(True, alpha=0.3)
174
175     plt.tight_layout()
176     plt.savefig(os.path.join(self.figure_dir, '2a_kde_tuning.png'), dpi=
177                 150)
178
179
180     def generate_samples(self, space='pca_20', n_samples=100, verbose=False):
181         result = self.kde_results[space]
182         model = result['model']
183
184         # Generate samples in latent space
185         X_generated = model.sample(n_samples, random_state=42)
186
187         # Transform back to original space
188         if space == 'original':
189             # Need to inverse standardization
190             # For original space, we need the scaler that was used
191             scaler = StandardScaler()
192             scaler.fit(self.data)
193             X_original = scaler.inverse_transform(X_generated)
194
195         else:
196             # Inverse PCA transformation
197             pca = result['pca']
198             pca.fit(self.data)
199             X_pca_space = X_generated
200
201             # Inverse standardization in PCA space
202             scaler = StandardScaler()
203             scaler.fit(pca.transform(self.data))
204             X_pca_unscaled = scaler.inverse_transform(X_pca_space)
205
206             # Inverse PCA
207             X_original = pca.inverse_transform(X_pca_unscaled)
208
209             # Clip to valid range [0, 16]
210             X_original = np.clip(X_original, 0, 16)

```

```

210     self.generated_samples = {
211         'samples': X_original,
212         'space': space,
213         'n_samples': n_samples
214     }
215
216     if verbose:
217         print(f"\nGenerated {n_samples} samples from space: {space}")
218         print(f"Generated samples shape: {X_original.shape}")
219         print(f"Generated samples range: [{X_original.min():.2f}, {
220                           X_original.max():.2f}]")
221
222     return X_original
223
224 def evaluate_samples(self, generated_samples=None, verbose=False):
225     if generated_samples is None:
226         generated_samples = self.generated_samples['samples']
227
228     metrics = {}
229
230     # 1. Sample statistics comparison
231     metrics['original_mean'] = self.data.mean()
232     metrics['original_std'] = self.data.std()
233     metrics['generated_mean'] = generated_samples.mean()
234     metrics['generated_std'] = generated_samples.std()
235
236     # 2. Sparsity check (% of zero pixels)
237     metrics['original_sparsity'] = (self.data == 0).mean()
238     metrics['generated_sparsity'] = (generated_samples == 0).mean()
239
240     # 3. Quality check: Can we classify generated samples?
241     try:
242         from sklearn.neighbors import KNeighborsClassifier
243         from sklearn.model_selection import cross_val_score
244
245         knn = KNeighborsClassifier(n_neighbors=5)
246         scores = cross_val_score(
247             knn, self.data, self.targets, cv=5, scoring='accuracy'
248         )
249         metrics['knn_accuracy_original'] = scores.mean()
250
251     except Exception as e:
252         metrics['knn_accuracy_original'] = None
253
254     if verbose:
255         self._print_evaluation_metrics(metrics)
256
257     return metrics
258
259 def _print_evaluation_metrics(self, metrics):
260     print(f"Mean (Original): {metrics['original_mean']:.4f}")
261     print(f"Mean (Generated): {metrics['generated_mean']:.4f}")
262     print(f"Std (Original): {metrics['original_std']:.4f}")
263     print(f"Std (Generated): {metrics['generated_std']:.4f}")

```

```

263     print(f"Sparsity (Original): {metrics['original_sparsity']:.4f}")
264     print(f"Sparsity (Generated): {metrics['generated_sparsity']:.4f}")
265     if metrics['knn_accuracy_original'] is not None:
266         print(f"KNN Accuracy (Original): {metrics['knn_accuracy_original']
267             '']:.4f}")
268
269     def visualize_generated_samples(self, generated_samples=None, n_display=25
270                                     ):
271         if generated_samples is None:
272             generated_samples = self.generated_samples['samples']
273             space = self.generated_samples['space']
274         else:
275             space = 'provided'
276
277         grid_size = int(np.sqrt(n_display))
278         fig, axes = plt.subplots(grid_size, grid_size, figsize=(10, 10))
279         axes = axes.flatten()
280
281         for i in range(min(n_display, len(generated_samples))):
282             axes[i].imshow(generated_samples[i].reshape(8, 8), cmap='gray')
283             axes[i].axis('off')
284
285         plt.suptitle(f'Generated Digits from KDE ({space})', fontsize=14)
286         plt.tight_layout()
287         plt.savefig(os.path.join(self.figure_dir, f'2a_generated_samples_{space}.png'), dpi=150)
288         plt.show()
289
290     def compare_distributions(self, generated_samples=None):
291         if generated_samples is None:
292             generated_samples = self.generated_samples['samples']
293
294         fig, axes = plt.subplots(1, 2, figsize=(14, 5))
295
296         # Pixel intensity distribution
297         axes[0].hist(self.data.flatten(), bins=50, alpha=0.6, label='Original',
298                     ,
299                     color='blue', density=True)
300         axes[0].hist(generated_samples.flatten(), bins=50, alpha=0.6, label='Generated',
301                     ,
302                     color='red', density=True)
303         axes[0].set_xlabel('Pixel Intensity')
304         axes[0].set_ylabel('Density')
305         axes[0].set_title('Pixel Intensity Distribution')
306         axes[0].legend()
307         axes[0].grid(alpha=0.3)
308
309         # Sparsity comparison
310         sparsity_orig = (self.data == 0).sum(axis=1).mean()
311         sparsity_gen = (generated_samples == 0).sum(axis=1).mean()
312
313         categories = ['Original', 'Generated']
314         sparsities = [sparsity_orig, sparsity_gen]

```

```

311     axes[1].bar(categories, sparsities, color=['blue', 'red'], alpha=0.7,
312                     edgecolor='black')
313     axes[1].set_ylabel('Average # of Zero Pixels')
314     axes[1].set_title('Sparsity Comparison')
315     axes[1].grid(axis='y', alpha=0.3)
316
317     plt.tight_layout()
318     plt.savefig(os.path.join(self.figure_dir, '2a_distribution_comparison.
319                               png'), dpi=150)
320     plt.show()

```

A.3 Problem 2b: GAN

A.3.1 Main Script.

```

1 """
2 Author: Chuyang Su cs4570@columbia.edu
3 Date: 2025-11-25 10:49:06
4 LastEditTime: 2025-11-25 13:28:54
5 FilePath: /Unsupervised-Learning-Homework/Homework 3/Code/Prob2b_main.py
6 Description:
7     Main script to run GAN model fitting for Homework 3 Problem 2b.
8     Optimized flow:
9     1. process_data() -> X, y
10    2. tune_hyperparameters() (optional) -> best config
11    3. fit_gan() -> trained GAN
12    4. generate_samples() -> generated data
13    5. evaluate_samples() -> metrics
14    6. visualize_generated_samples() + compare_distributions()
15 """
16 from Prob2b_utils import Prob2bAnalysis
17
18
19 def main():
20     # Initialize analysis
21     p2b = Prob2bAnalysis(
22         output_dir='Homework 3/Code/Data',
23         figure_dir='Homework 3/Latex/Figures'
24     )
25
26     # Load and preprocess data
27     X, y = p2b.process_data(verbose=False)
28
29     # Hyperparameter tuning
30     tuning_results = p2b.tune_hyperparameters(
31         latent_dims=[32, 64, 128],
32         hidden_configs=[
33             ([128, 256], [256, 128]),
34             ([256, 512], [512, 256]),
35         ],
36         n_epochs=200,
37         verbose=False

```

```

38     )
39 # Best parameters from tuning: latent_dim=32, lr_g=0.0001, G=[128, 256]
40
41 gan_results = p2b.fit_gan(
42     latent_dim=32,
43     g_hidden_dims=[128, 256],
44     d_hidden_dims=[256, 128],
45     lr_g=0.0001,
46     lr_d=0.0002,
47     batch_size=64,
48     n_epochs=300,
49     n_critic=1,
50     beta1=0.5,
51     label_smoothing=0.1,
52     verbose=False
53 )
54
55 # Generate samples
56 generated_samples = p2b.generate_samples(
57     n_samples=100,
58     verbose=False
59 )
60
61 # Evaluate generated samples
62 metrics = p2b.evaluate_samples(verbose=True)
63
64 # Visualization
65 print("\nVisualizing generated samples...")
66 p2b.visualize_generated_samples(n_display=25)
67
68 print("\nComparing original vs generated distributions...")
69 p2b.compare_distributions()
70
71 print("\nVisualizing latent space interpolation...")
72 p2b.visualize_latent_interpolation(n_steps=10)
73
74 print("\nComparing real vs generated samples side-by-side...")
75 p2b.compare_with_real_samples(n_display=10)
76
77
78 if __name__ == "__main__":
79     main()

```

A.3.2 Utils Script.

```

1 """
2 Author: Chuyang Su cs4570@columbia.edu
3 Date: 2025-11-25 10:49:19
4 LastEditTime: 2025-11-25 12:44:23
5 FilePath: /Unsupervised-Learning-Homework/Homework 3/Code/Prob2b_utils.py
6 Description:
7     Utility functions and classes for Problem 2b of Homework 3.

```

```
8 Generative Adversarial Network (GAN) for digit generation.
9 - process_data: Load and preprocess sklearn digits dataset
10 - build_generator: Build generator network
11 - build_discriminator: Build discriminator network
12 - fit_gan: Train GAN with hyperparameter tuning
13 - generate_samples: Generate new samples from trained GAN
14 - evaluate_samples: Evaluate quality of generated samples
15 '''
16 import os
17 import numpy as np
18 import pandas as pd
19 import matplotlib.pyplot as plt
20 import seaborn as sns
21 from sklearn.datasets import load_digits
22 from sklearn.preprocessing import MinMaxScaler, StandardScaler
23 from sklearn.decomposition import PCA
24 from sklearn.neighbors import KNeighborsClassifier
25 from sklearn.model_selection import cross_val_score
26 import torch
27 import torch.nn as nn
28 import torch.optim as optim
29 from torch.utils.data import DataLoader, TensorDataset
30
31 class Generator(nn.Module):
32     def __init__(self, latent_dim=64, hidden_dims=[128, 256], output_dim=64,
33                  activation='relu', use_batchnorm=True):
34         super(Generator, self).__init__()
35
36         self.latent_dim = latent_dim
37
38         # Build layers
39         layers = []
40         in_dim = latent_dim
41
42         for h_dim in hidden_dims:
43             layers.append(nn.Linear(in_dim, h_dim))
44             if use_batchnorm:
45                 layers.append(nn.BatchNorm1d(h_dim))
46             if activation == 'relu':
47                 layers.append(nn.ReLU())
48             elif activation == 'leaky_relu':
49                 layers.append(nn.LeakyReLU(0.2))
50             elif activation == 'tanh':
51                 layers.append(nn.Tanh())
52             in_dim = h_dim
53
54         # Output layer with Sigmoid for [0, 1] range
55         layers.append(nn.Linear(in_dim, output_dim))
56         layers.append(nn.Sigmoid())
57
58         self.model = nn.Sequential(*layers)
59
60     def forward(self, z):
61         return self.model(z)
```

```
62
63
64 class Discriminator(nn.Module):
65     def __init__(self, input_dim=64, hidden_dims=[256, 128],
66                  activation='leaky_relu', dropout_rate=0.3):
67         super(Discriminator, self).__init__()
68
69         layers = []
70         in_dim = input_dim
71
72         for h_dim in hidden_dims:
73             layers.append(nn.Linear(in_dim, h_dim))
74             if activation == 'leaky_relu':
75                 layers.append(nn.LeakyReLU(0.2))
76             elif activation == 'relu':
77                 layers.append(nn.ReLU())
78             if dropout_rate > 0:
79                 layers.append(nn.Dropout(dropout_rate))
80             in_dim = h_dim
81
82         # Output layer with Sigmoid for probability
83         layers.append(nn.Linear(in_dim, 1))
84         layers.append(nn.Sigmoid())
85
86         self.model = nn.Sequential(*layers)
87
88     def forward(self, x):
89         return self.model(x)
90
91
92 class Prob2bAnalysis:
93     def __init__(self,
94                  output_dir='Homework 3/Code/Data',
95                  figure_dir='Homework 3/Latex/Figures',
96                  device=None,
97                  seed=25):
98         self.output_dir = output_dir
99         self.figure_dir = figure_dir
100
101     os.makedirs(self.output_dir, exist_ok=True)
102     os.makedirs(self.figure_dir, exist_ok=True)
103
104     # Set device
105     if device is None:
106         self.device = torch.device('cuda' if torch.cuda.is_available()
107                                   else 'cpu')
108         print(f"Using device: {self.device}")
109     else:
110         self.device = device
111
112     self.data = None
113     self.targets = None
114     self.scaler = None
115     self.gan_results = None
```

```

115     self.generated_samples = None
116     self.training_history = None
117
118     self.seed = seed
119
120     def _set_seed(self):
121         np.random.seed(self.seed)
122         torch.manual_seed(self.seed)
123         if torch.cuda.is_available():
124             torch.cuda.manual_seed_all(self.seed)
125
126     def process_data(self, verbose=False):
127         # Different form 2a process_data, so cannot share code
128         digits = load_digits()
129         X = digits.data
130         y = digits.target
131
132         # Scale to [0, 1] for GAN training
133         self.scaler = MinMaxScaler(feature_range=(0, 1))
134         X_scaled = self.scaler.fit_transform(X)
135
136         self.data = X
137         self.data_scaled = X_scaled
138         self.targets = y
139
140         if verbose:
141             print(f"\nDataset shape: {X.shape}")
142             print(f"Number of classes: {len(np.unique(y))}")
143             print(f"Original pixel range: [{X.min():.1f}, {X.max():.1f}]")
144             print(f"Scaled pixel range: [{X_scaled.min():.3f}, {X_scaled.max():.3f}]")
145             print(f"Device: {self.device}")
146
147         return X, y
148
149     def fit_gan(self,
150                 latent_dim=64,
151                 g_hidden_dims=[128, 256],
152                 d_hidden_dims=[256, 128],
153                 lr_g=0.0002,
154                 lr_d=0.0002,
155                 batch_size=64,
156                 n_epochs=300,
157                 n_critic=1,
158                 beta1=0.5,
159                 beta2=0.999,
160                 label_smoothing=0.1,
161                 verbose=False):
162         self._set_seed()
163         if self.data is None:
164             self.process_data()
165
166         output_dim = self.data.shape[1] # 64 for 8x8 images
167

```

```

168     # Build networks
169     generator = Generator(
170         latent_dim=latent_dim,
171         hidden_dims=g_hidden_dims,
172         output_dim=output_dim,
173         activation='relu',
174         use_batchnorm=True
175     ).to(self.device)
176
177     discriminator = Discriminator(
178         input_dim=output_dim,
179         hidden_dims=d_hidden_dims,
180         activation='leaky_relu',
181         dropout_rate=0.3
182     ).to(self.device)
183
184     # Optimizers
185     optimizer_g = optim.Adam(generator.parameters(), lr=lr_g, betas=(beta1
186                                     , beta2))
186     optimizer_d = optim.Adam(discriminator.parameters(), lr=lr_d, betas=(
187                                     beta1, beta2))
188
189     # Loss function
190     criterion = nn.BCELoss()
191
192     # Prepare data
193     X_tensor = torch.FloatTensor(self.data_scaled).to(self.device)
194     dataset = TensorDataset(X_tensor)
195     dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True,
196                             drop_last=True)
197
198     # Training history
199     history = {
200         'epoch': [],
201         'd_loss': [],
202         'g_loss': [],
203         'd_real_acc': [],
204         'd_fake_acc': []
205     }
206
207     if verbose:
208         print(f"\nTraining GAN...")
209         print(f"  Latent dim: {latent_dim}")
210         print(f"  Generator: {g_hidden_dims}")
211         print(f"  Discriminator: {d_hidden_dims}")
212         print(f"  Epochs: {n_epochs}, Batch size: {batch_size}")
213         print("-" * 60)
214
215     # Training loop
216     for epoch in range(n_epochs):
217         d_losses = []
218         g_losses = []
219         d_real_accs = []
220         d_fake_accs = []

```

```

219
220     for batch_data in dataloader:
221         real_data = batch_data[0]
222         current_batch_size = real_data.size(0)
223
224         # Labels with smoothing
225         real_labels = torch.ones(current_batch_size, 1).to(self.device
226                                 ) * (1 -
227                                       label_smoothing)
228         fake_labels = torch.zeros(current_batch_size, 1).to(self.
229                                   device)
230
231         # -----
232         # Train Discriminator
233         # -----
234         for _ in range(n_critic):
235             optimizer_d.zero_grad()
236
237             # Real data
238             d_real_output = discriminator(real_data)
239             d_real_loss = criterion(d_real_output, real_labels)
240
241             # Fake data
242             z = torch.randn(current_batch_size, latent_dim).to(self.
243                                     device)
244             fake_data = generator(z)
245             d_fake_output = discriminator(fake_data.detach())
246             d_fake_loss = criterion(d_fake_output, fake_labels)
247
248             # Total discriminator loss
249             d_loss = d_real_loss + d_fake_loss
250             d_loss.backward()
251             optimizer_d.step()
252
253             d_losses.append(d_loss.item())
254             d_real_accs.append((d_real_output > 0.5).float().mean().item()
255                                 )
256             d_fake_accs.append((d_fake_output < 0.5).float().mean().item()
257                                 )
258
259             # -----
260             # Train Generator
261             # -----
262             optimizer_g.zero_grad()
263
264             z = torch.randn(current_batch_size, latent_dim).to(self.device
265                           )
266             fake_data = generator(z)
267             g_output = discriminator(fake_data)
268
269             # Generator wants discriminator to think fake is real
270             g_loss = criterion(g_output, torch.ones(current_batch_size, 1
271                               .to(self.device)))
272             g_loss.backward()

```

```

265         optimizer_g.step()
266
267         g_losses.append(g_loss.item())
268
269     # Record epoch metrics
270     history['epoch'].append(epoch + 1)
271     history['d_loss'].append(np.mean(d_losses))
272     history['g_loss'].append(np.mean(g_losses))
273     history['d_real_acc'].append(np.mean(d_real_accs))
274     history['d_fake_acc'].append(np.mean(d_fake_accs))
275
276     # Print progress
277     if verbose and (epoch + 1) % 50 == 0:
278         print(f"Epoch [{epoch+1}:{4d}]/[{n_epochs}] | "
279               f"D Loss: {history['d_loss'][-1]:.4f} | "
280               f"G Loss: {history['g_loss'][-1]:.4f} | "
281               f"D(real): {history['d_real_acc'][-1]:.3f} | "
282               f"D(fake): {history['d_fake_acc'][-1]:.3f}")
283
284     # Store results
285     self.gan_results = {
286         'generator': generator,
287         'discriminator': discriminator,
288         'latent_dim': latent_dim,
289         'history': history,
290         'config': {
291             'latent_dim': latent_dim,
292             'g_hidden_dims': g_hidden_dims,
293             'd_hidden_dims': d_hidden_dims,
294             'lr_g': lr_g,
295             'lr_d': lr_d,
296             'batch_size': batch_size,
297             'n_epochs': n_epochs
298         }
299     }
300
301     self.training_history = history
302
303     if verbose:
304         print("Training completed!")
305         self._plot_training_history(history)
306
307     return self.gan_results
308
309 def tune_hyperparameters(self,
310                         latent_dims=[32, 64, 128],
311                         lr_g_values=[0.00005, 0.0001, 0.0002],
312                         lr_d=0.0002,
313                         hidden_configs=[
314                             ([128, 256], [256, 128]),
315                             ([256, 512], [512, 256]),
316                             ([64, 128, 256], [256, 128, 64])
317                         ],
318                         n_epochs=200,

```

```

319                     n_eval_samples=500,
320                     verbose=False):
321             if self.data is None:
322                 self.process_data()
323
324             results = []
325
326             for latent_dim in latent_dims:
327                 for lr_g in lr_g_values:
328                     for g_hidden, d_hidden in hidden_configs:
329                         # Train model
330                         self.fit_gan(
331                             latent_dim=latent_dim,
332                             g_hidden_dims=g_hidden,
333                             d_hidden_dims=d_hidden,
334                             lr_g=lr_g,
335                             lr_d=lr_d,
336                             n_epochs=n_epochs,
337                             verbose=False
338                         )
339
340             # Generate and evaluate samples
341             samples = self.generate_samples(n_samples=n_eval_samples,
342                                             verbose=False)
342             metrics = self.evaluate_samples(samples, verbose=False)
343
344             # Store results
345             result = {
346                 'latent_dim': latent_dim,
347                 'lr_g': lr_g,
348                 'g_hidden': str(g_hidden),
349                 'd_hidden': str(d_hidden),
350                 'mean_diff': abs(metrics['generated_mean'] - metrics['
351                                         original_mean']
352                                         ),
351                 'std_diff': abs(metrics['generated_std'] - metrics['
351                                         original_std'])
353                                         ,
352                 'diversity_ratio': metrics.get('diversity_ratio', 0)
353             }
354             results.append(result)
355
356             if verbose:
357                 print(f"  Mean Diff: {result['mean_diff']:.4f} | "
358                       f"Std Diff: {result['std_diff']:.4f} | "
359                       f"Diversity: {result['diversity_ratio']:.3f}")
360
361             results_df = pd.DataFrame(results)
362
363             if verbose:
364                 print("Tuning Results Summary:")
365                 print(results_df.to_string(index=False))
366                 self._plot_tuning_results(results_df)
367

```

```

368     return results_df
369
370 def _plot_tuning_results(self, results_df):
371     fig, axes = plt.subplots(1, 3, figsize=(15, 5))
372
373     # Mean diff by lr_g, grouped by latent_dim
374     for latent in results_df['latent_dim'].unique():
375         subset = results_df[results_df['latent_dim'] == latent]
376         avg_by_lr = subset.groupby('lr_g')['mean_diff'].mean()
377         axes[0].plot(avg_by_lr.index, avg_by_lr.values, 'o-', label=f'latent={latent}',
378                      markersize=8)
379     axes[0].set_title('Mean Diff vs lr_g (Lower is Better)')
380     axes[0].set_xscale('log')
381     axes[0].legend()
382     axes[0].grid(True, alpha=0.3)
383
384     # Mean difference
385     pivot = results_df.pivot_table(
386         values='mean_diff',
387         index='latent_dim',
388         columns='lr_g',
389         aggfunc='mean'
390     )
391     sns.heatmap(pivot, annot=True, fmt='.4f', cmap='YlOrRd', ax=axes[1])
392     axes[1].set_title('Mean Difference (Lower is Better)')
393
394     # Std difference
395     pivot_std = results_df.pivot_table(
396         values='std_diff',
397         index='latent_dim',
398         columns='lr_g',
399         aggfunc='mean'
400     )
401     sns.heatmap(pivot_std, annot=True, fmt='.4f', cmap='YlOrRd', ax=axes[2])
402     axes[2].set_title('Std Difference (Lower is Better)')
403
404     plt.tight_layout()
405     plt.savefig(os.path.join(self.figure_dir, '2b_gan_tuning.png'), dpi=150)
406     plt.show()
407
408 def _plot_training_history(self, history):
409     fig, axes = plt.subplots(1, 2, figsize=(14, 5))
410
411     epochs = history['epoch']
412
413     # Losses
414     axes[0].plot(epochs, history['d_loss'], label='Discriminator Loss',
415                  alpha=0.8)
416     axes[0].plot(epochs, history['g_loss'], label='Generator Loss', alpha=
417                  0.8)
418     axes[0].set_xlabel('Epoch')

```

```

416     axes[0].set_ylabel('Loss')
417     axes[0].set_title('GAN Training Losses')
418     axes[0].legend()
419     axes[0].grid(True, alpha=0.3)
420
421     # Discriminator accuracy
422     axes[1].plot(epochs, history['d_real_acc'], label='D(real) accuracy',
423                   alpha=0.8)
423     axes[1].plot(epochs, history['d_fake_acc'], label='D(fake) accuracy',
424                   alpha=0.8)
424     axes[1].axhline(y=0.5, color='r', linestyle='--', alpha=0.5, label='Random guess')
425     axes[1].set_xlabel('Epoch')
426     axes[1].set_ylabel('Accuracy')
427     axes[1].set_title('Discriminator Performance')
428     axes[1].legend()
429     axes[1].grid(True, alpha=0.3)
430     axes[1].set_ylim([0, 1])
431
432     plt.tight_layout()
433     plt.savefig(os.path.join(self.figure_dir, '2b_gan_training.png'), dpi=
434                 150)
435     plt.show()
436
437 def generate_samples(self, n_samples=100, verbose=False):
438     generator = self.gan_results['generator']
439     latent_dim = self.gan_results['latent_dim']
440
441     generator.eval()
442     with torch.no_grad():
443         z = torch.randn(n_samples, latent_dim).to(self.device)
444         generated_scaled = generator(z).cpu().numpy()
445
446         # Inverse transform to original scale
447         generated = self.scaler.inverse_transform(generated_scaled)
448
449         # Clip to valid range [0, 16]
450         generated = np.clip(generated, 0, 16)
451
452         self.generated_samples = {
453             'samples': generated,
454             'samples_scaled': generated_scaled,
455             'n_samples': n_samples
456         }
457
458     if verbose:
459         print(f"\nGenerated {n_samples} samples from GAN")
460         print(f"Generated samples shape: {generated.shape}")
461         print(f"Generated samples range: [{generated.min():.2f}, {generated.max():.2f}]")
462
463     return generated
464
465 def evaluate_samples(self, generated_samples=None, verbose=False):

```

```

465     if generated_samples is None:
466         generated_samples = self.generated_samples['samples']
467
468     metrics = {}
469
470     # 1. Sample statistics comparison
471     metrics['original_mean'] = self.data.mean()
472     metrics['original_std'] = self.data.std()
473     metrics['generated_mean'] = generated_samples.mean()
474     metrics['generated_std'] = generated_samples.std()
475
476     # 2. Sparsity check (% of near-zero pixels)
477     threshold = 0.5
478     metrics['original_sparsity'] = (self.data < threshold).mean()
479     metrics['generated_sparsity'] = (generated_samples < threshold).mean()
480
481     # 3. Coverage: fraction of original data modes covered
482     # Using PCA + clustering approximation
483     try:
484         pca = PCA(n_components=10)
485         orig_pca = pca.fit_transform(self.data)
486         gen_pca = pca.transform(generated_samples)
487
488         # Simple coverage: check if generated samples are near original
489         # samples
490         from scipy.spatial.distance import cdist
491         distances = cdist(gen_pca, orig_pca, metric='euclidean')
492         min_distances = distances.min(axis=1)
493         coverage_threshold = np.percentile(
494             cdist(orig_pca, orig_pca).flatten(), 50
495         )
496         metrics['coverage'] = (min_distances < coverage_threshold).mean()
497     except Exception as e:
498         metrics['coverage'] = None
499
500     # 4. Mode collapse check: diversity of generated samples
501     try:
502         gen_pca = PCA(n_components=10).fit_transform(generated_samples)
503         pairwise_dist = cdist(gen_pca, gen_pca, metric='euclidean')
504         np.fill_diagonal(pairwise_dist, np.inf)
505         metrics['avg_nn_distance'] = pairwise_dist.min(axis=1).mean()
506
507         orig_pca = PCA(n_components=10).fit_transform(self.data)
508         orig_pairwise = cdist(orig_pca, orig_pca, metric='euclidean')
509         np.fill_diagonal(orig_pairwise, np.inf)
510         metrics['orig_avg_nn_distance'] = orig_pairwise.min(axis=1).mean()
511
512         # Diversity ratio (higher is better, 1.0 means same diversity as
513         # original)
514         metrics['diversity_ratio'] = metrics['avg_nn_distance'] / metrics[
515             'orig_avg_nn_distance']
516     except Exception as e:
517         metrics['diversity_ratio'] = None

```

```

516     if verbose:
517         self._print_evaluation_metrics(metrics)
518
519     return metrics
520
521 def _print_evaluation_metrics(self, metrics):
522     """Print evaluation metrics."""
523     print("\n--- GAN Evaluation Metrics ---")
524     print(f"Mean (Original): {metrics['original_mean']:.4f}")
525     print(f"Mean (Generated): {metrics['generated_mean']:.4f}")
526     print(f"Std (Original): {metrics['original_std']:.4f}")
527     print(f"Std (Generated): {metrics['generated_std']:.4f}")
528     print(f"Sparsity (Original): {metrics['original_sparsity']:.4f}")
529     print(f"Sparsity (Generated): {metrics['generated_sparsity']:.4f}")
530     if metrics.get('coverage') is not None:
531         print(f"Coverage: {metrics['coverage']:.4f}")
532     if metrics.get('diversity_ratio') is not None:
533         print(f"Diversity Ratio: {metrics['diversity_ratio']:.4f}")
534
535 def visualize_generated_samples(self, generated_samples=None, n_display=25):
536     """Visualize grid of generated samples."""
537     if generated_samples is None:
538         generated_samples = self.generated_samples['samples']
539
540     grid_size = int(np.sqrt(n_display))
541     fig, axes = plt.subplots(grid_size, grid_size, figsize=(10, 10))
542     axes = axes.flatten()
543
544     for i in range(min(n_display, len(generated_samples))):
545         axes[i].imshow(generated_samples[i].reshape(8, 8), cmap='gray')
546         axes[i].axis('off')
547
548     plt.suptitle('Generated Digits from GAN', fontsize=14)
549     plt.tight_layout()
550     plt.savefig(os.path.join(self.figure_dir, '2b_generated_samples_gan.
551                                         png'), dpi=150)
552     plt.show()
553
554 def compare_distributions(self, generated_samples=None):
555     if generated_samples is None:
556         generated_samples = self.generated_samples['samples']
557
558     fig, axes = plt.subplots(1, 3, figsize=(18, 5))
559
560     # Pixel intensity distribution
561     axes[0].hist(self.data.flatten(), bins=50, alpha=0.6, label='Original',
562                  color='blue', density=True)
563     axes[0].hist(generated_samples.flatten(), bins=50, alpha=0.6, label='Generated',
564                  color='red', density=True)
565     axes[0].set_xlabel('Pixel Intensity')
566     axes[0].set_ylabel('Density')

```

```

566     axes[0].set_title('Pixel Intensity Distribution')
567     axes[0].legend()
568     axes[0].grid(alpha=0.3)
569
570     # Sparsity comparison
571     sparsity_orig = (self.data < 0.5).sum(axis=1).mean()
572     sparsity_gen = (generated_samples < 0.5).sum(axis=1).mean()
573
574     categories = ['Original', 'Generated']
575     sparsities = [sparsity_orig, sparsity_gen]
576     axes[1].bar(categories, sparsities, color=['blue', 'red'], alpha=0.7,
577                  edgecolor='black')
577     axes[1].set_ylabel('Average # of Near-Zero Pixels')
578     axes[1].set_title('Sparsity Comparison')
579     axes[1].grid(axis='y', alpha=0.3)
580
581     # PCA projection comparison
582     pca = PCA(n_components=2)
583     orig_pca = pca.fit_transform(self.data)
584     gen_pca = pca.transform(generated_samples)
585
586     axes[2].scatter(orig_pca[:, 0], orig_pca[:, 1], alpha=0.3, label='
587                     Original', s=10)
587     axes[2].scatter(gen_pca[:, 0], gen_pca[:, 1], alpha=0.5, label='
588                     Generated', s=10)
589     axes[2].set_xlabel('PC1')
590     axes[2].set_ylabel('PC2')
591     axes[2].set_title('PCA Projection Comparison')
592     axes[2].legend()
593     axes[2].grid(alpha=0.3)
594
595     plt.tight_layout()
596     plt.savefig(os.path.join(self.figure_dir, '
597                     2b_distribution_comparison_gan.
598                     png'), dpi=150)
599     plt.show()
600
601     def visualize_latent_interpolation(self, n_steps=10):
602         generator = self.gan_results['generator']
603         latent_dim = self.gan_results['latent_dim']
604
605         generator.eval()
606
607         # Generate two random latent vectors
608         z1 = torch.randn(1, latent_dim).to(self.device)
609         z2 = torch.randn(1, latent_dim).to(self.device)
610
611         # Interpolate
612         interpolations = []
613         for alpha in np.linspace(0, 1, n_steps):
614             z = (1 - alpha) * z1 + alpha * z2
615             with torch.no_grad():
616                 img = generator(z).cpu().numpy()
617             img = self.scaler.inverse_transform(img)

```

```

615         img = np.clip(img, 0, 16)
616         interpolations.append(img.reshape(8, 8))
617
618     # Plot
619     fig, axes = plt.subplots(1, n_steps, figsize=(2 * n_steps, 2))
620     for i, img in enumerate(interpolations):
621         axes[i].imshow(img, cmap='gray')
622         axes[i].axis('off')
623         if i == 0:
624             axes[i].set_title('Start')
625         elif i == n_steps - 1:
626             axes[i].set_title('End')
627
628     plt.suptitle('Latent Space Interpolation', fontsize=14)
629     plt.tight_layout()
630     plt.savefig(os.path.join(self.figure_dir, '2b_latent_interpolation.png',
631                           ''), dpi=150)
632     plt.show()
633
634 def compare_with_real_samples(self, n_display=10):
635     generated = self.generated_samples['samples'][:n_display]
636     real_indices = np.random.choice(len(self.data), n_display, replace=False)
637     real = self.data[real_indices]
638
639     fig, axes = plt.subplots(2, n_display, figsize=(2 * n_display, 4))
640
641     for i in range(n_display):
642         axes[0, i].imshow(real[i].reshape(8, 8), cmap='gray')
643         axes[0, i].axis('off')
644         if i == 0:
645             axes[0, i].set_ylabel('Real', fontsize=12)
646
647         axes[1, i].imshow(generated[i].reshape(8, 8), cmap='gray')
648         axes[1, i].axis('off')
649         if i == 0:
650             axes[1, i].set_ylabel('Generated', fontsize=12)
651
652     plt.suptitle('Real vs Generated Samples', fontsize=14)
653     plt.tight_layout()
654     plt.savefig(os.path.join(self.figure_dir, '2b_real_vs_generated.png'),
655                 dpi=150)
656     plt.show()

```

A.4 Problem 2c: Diffusion Model.

A.4.1 Main Script.

```

1 ...
2 Author: Chuyang Su cs4570@columbia.edu
3 Date: 2025-11-25 15:08:40
4 LastEditTime: 2025-11-25 20:54:04

```

```
5 FilePath: /Unsupervised-Learning-Homework/Homework 3/Code/Prob2c_main.py
6 Description:
7     Main script to run Diffusion Model fitting for Homework 3 Problem 2c.
8     Optimized flow:
9     1. process_data() -> X, y
10    2. tune_hyperparameters() (optional) -> best config
11    3. fit_diffusion() -> trained model
12    4. generate_samples() -> generated data
13    5. evaluate_samples() -> metrics
14    6. visualize_generated_samples() + compare_distributions()
15 '''
16 from Prob2c_utils import Prob2cAnalysis
17
18
19 def main():
20     # Initialize analysis (with seed for reproducibility)
21     p2c = Prob2cAnalysis(
22         output_dir='Homework 3/Code/Data',
23         figure_dir='Homework 3/Latex/Figures',
24         seed=25
25     )
26
27     # Load and preprocess data
28     X, y = p2c.process_data(verbose=False)
29
30     # Hyperparameter tuning
31     # tuning_results = p2c.tune_hyperparameters(
32     #     num_timesteps_values=[200, 500, 1000],
33     #     lr_values=[1e-4, 5e-4, 1e-3],
34     #     hidden_configs=[
35     #         [128, 256, 128],
36     #         [256, 512, 256],
37     #     ],
38     #     n_epochs=150,
39     #     verbose=False
40     # ) # Best results: lr=0.0005, timesteps=500, architecture=[256, 512, 256]
41
42     # Train Diffusion Model with selected hyperparameters
43     diffusion_results = p2c.fit_diffusion(
44         num_timesteps=500,
45         hidden_dims=[256, 512, 256],
46         time_emb_dim=64,
47         lr=5e-4,
48         batch_size=128,
49         n_epochs=200,
50         beta_start=1e-4,
51         beta_end=0.02,
52         verbose=False
53     )
54
55     # Generate samples
56     generated_samples = p2c.generate_samples(
57         n_samples=100,
58         verbose=True
```

```

59 )
60
61 # Evaluate generated samples
62 metrics = p2c.evaluate_samples(verbose=True)
63
64 # Visualization
65 print("\nVisualizing generated samples...")
66 p2c.visualize_generated_samples(n_display=25)
67
68 print("\nComparing original vs generated distributions...")
69 p2c.compare_distributions()
70
71 print("\nVisualizing reverse diffusion process...")
72 p2c.visualize_diffusion_process(n_steps=10)
73
74 print("\nComparing real vs generated samples side-by-side...")
75 p2c.compare_with_real_samples(n_display=10)
76
77
78 if __name__ == "__main__":
79     main()

```

A.4.2 Utils Script.

```

1 """
2 Author: Chuyang Su cs4570@columbia.edu
3 Date: 2025-11-25 15:08:49
4 LastEditTime: 2025-11-25 20:56:24
5 FilePath: /Unsupervised-Learning-Homework/Homework 3/Code/Prob2c_utils.py
6 Description:
7     Utility functions and classes for Problem 2c of Homework 3.
8     Denoising Diffusion Probabilistic Model (DDPM) for digit generation.
9     - process_data: Load and preprocess sklearn digits dataset
10    - build_denoiser: Build denoising network
11    - fit_diffusion: Train diffusion model with hyperparameter tuning
12    - generate_samples: Generate new samples via reverse diffusion
13    - evaluate_samples: Evaluate quality of generated samples
14 """
15 import os
16 import numpy as np
17 import pandas as pd
18 import matplotlib.pyplot as plt
19 import seaborn as sns
20 from sklearn.datasets import load_digits
21 from sklearn.preprocessing import MinMaxScaler
22 from sklearn.decomposition import PCA
23 from scipy.spatial.distance import cdist
24 import torch
25 import torch.nn as nn
26 import torch.optim as optim
27 from torch.utils.data import DataLoader, TensorDataset
28 from tqdm import tqdm

```

```

29
30
31 class SinusoidalPositionEmbeddings(nn.Module):
32     def __init__(self, dim):
33         super().__init__()
34         self.dim = dim
35
36     def forward(self, time):
37         device = time.device
38         half_dim = self.dim // 2
39         embeddings = np.log(10000) / (half_dim - 1)
40         embeddings = torch.exp(torch.arange(half_dim, device=device) * -
41                               embeddings)
42         embeddings = time[:, None] * embeddings[None, :]
43         embeddings = torch.cat((embeddings.sin(), embeddings.cos()), dim=-1) #
44                                         Use both sin and cos to
45                                         capture more information
46
47         return embeddings
48
49
50
51 class DenoisingMLP(nn.Module):
52     def __init__(self, input_dim=64, hidden_dims=[256, 512, 256], time_emb_dim =
53                  =64):
54         super().__init__()
55
56         self.time_mlp = nn.Sequential(
57             SinusoidalPositionEmbeddings(time_emb_dim),
58             nn.Linear(time_emb_dim, time_emb_dim * 2),
59             nn.GELU(), # Instead of ReLU, GELU(Gaussian Error Linear Unit) as
59                         a smooth version often
59                         performs better
60             nn.Linear(time_emb_dim * 2, time_emb_dim)
61         )
62
63
64         # Input layer
65         self.input_layer = nn.Linear(input_dim, hidden_dims[0])
66
67         # Time embedding projection for each layer
68         self.time_projections = nn.ModuleList([
69             nn.Linear(time_emb_dim, h_dim) for h_dim in hidden_dims
70         ])
71
72
73         # Hidden layers
74         self.hidden_layers = nn.ModuleList()
75         for i in range(len(hidden_dims) - 1):
76             self.hidden_layers.append(
77                 nn.Sequential(
78                     nn.Linear(hidden_dims[i], hidden_dims[i + 1]),
79                     nn.GroupNorm(8, hidden_dims[i + 1]),
80                     nn.GELU()
81                 )
82             )
83
84
85         # Output layer
86
87
88

```



```

127     def q_sample(self, x_0, t, noise=None):
128         """Forward diffusion:  $q(x_t | x_0)$ """
129         if noise is None:
130             noise = torch.randn_like(x_0)
131
132         sqrt_alphas_cumprod_t = self.sqrt_alphas_cumprod[t][:, None]
133         sqrt_one_minus_alphas_cumprod_t = self.sqrt_one_minus_alphas_cumprod[t
134                                         ][:, None]
135
136         return sqrt_alphas_cumprod_t * x_0 + sqrt_one_minus_alphas_cumprod_t *
137             noise
138
139     def p_sample(self, model, x_t, t):
140         """Reverse diffusion:  $p(x_{t-1} | x_t)$ """
141         betas_t = self.betas[t][:, None]
142         sqrt_one_minus_alphas_cumprod_t = self.sqrt_one_minus_alphas_cumprod[t
143                                         ][:, None]
144         sqrt_recip_alphas_t = self.sqrt_recip_alphas[t][:, None]
145
146         # Predict noise
147         predicted_noise = model(x_t, t.float())
148
149         # Compute mean
150         model_mean = sqrt_recip_alphas_t * (
151             x_t - betas_t * predicted_noise / sqrt_one_minus_alphas_cumprod_t
152         )
153
154         # Add noise (except for t=0)
155         if (t > 0).any():
156             posterior_variance_t = self.posterior_variance[t][:, None]
157             noise = torch.randn_like(x_t)
158             # Only add noise where t > 0
159             mask = (t > 0).float()[:, None]
160             return model_mean + mask * torch.sqrt(posterior_variance_t) *
161                 noise
162         else:
163             return model_mean
164
165     class Prob2cAnalysis:
166         def __init__(self,
167                      output_dir='Homework 3/Code/Data',
168                      figure_dir='Homework 3/Latex/Figures',
169                      device=None,
170                      seed=25):
171             self.output_dir = output_dir
172             self.figure_dir = figure_dir
173             self.seed = seed
174
175             os.makedirs(self.output_dir, exist_ok=True)
176             os.makedirs(self.figure_dir, exist_ok=True)
177
178         # Set device
179         if device is None:

```

```
177         self.device = torch.device('cuda' if torch.cuda.is_available()
178                                     else 'cpu')
179
180     else:
181         self.device = device
182
183     self.data = None
184     self.targets = None
185     self.scaler = None
186     self.diffusion_results = None
187     self.generated_samples = None
188     self.training_history = None
189
190     def __set_seed(self):
191         np.random.seed(self.seed)
192         torch.manual_seed(self.seed)
193         if torch.cuda.is_available():
194             torch.cuda.manual_seed_all(self.seed)
195
196     def process_data(self, verbose=False):
197         digits = load_digits()
198         X = digits.data
199         y = digits.target
200
201         # Scale to [0, 1] then to [-1, 1] for diffusion
202         self.scaler = MinMaxScaler(feature_range=(-1, 1))
203         X_scaled = self.scaler.fit_transform(X)
204
205         self.data = X
206         self.data_scaled = X_scaled
207         self.targets = y
208
209         if verbose:
210             print(f"\nDataset shape: {X.shape}")
211             print(f"Number of classes: {len(np.unique(y))}")
212             print(f"Original pixel range: [{X.min():.1f}, {X.max():.1f}]")
213             print(f"Scaled pixel range: [{X_scaled.min():.3f}, {X_scaled.max():.3f}]")
214             print(f"Device: {self.device}")
215
216     return X, y
217
218     def fit_diffusion(self,
219                     num_timesteps=500,
220                     hidden_dims=[256, 512, 256],
221                     time_emb_dim=64,
222                     lr=1e-3,
223                     batch_size=128,
224                     n_epochs=200,
225                     beta_start=1e-4,
226                     beta_end=0.02,
227                     verbose=False):
228         self.__set_seed()
229
230         if self.data is None:
```

```
229         self.process_data()
230
231     input_dim = self.data.shape[1] # 64 for 8x8 images
232
233     # Build model
234     model = DenoisingMLP(
235         input_dim=input_dim,
236         hidden_dims=hidden_dims,
237         time_emb_dim=time_emb_dim
238     ).to(self.device)
239
240     # Build scheduler
241     scheduler = DiffusionScheduler(
242         num_timesteps=num_timesteps,
243         beta_start=beta_start,
244         beta_end=beta_end,
245         device=self.device
246     )
247
248     # Optimizer
249     optimizer = optim.Adam(model.parameters(), lr=lr)
250
251     # Loss function
252     criterion = nn.MSELoss()
253
254     # Prepare data
255     X_tensor = torch.FloatTensor(self.data_scaled).to(self.device)
256     dataset = TensorDataset(X_tensor)
257     dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True,
258                             drop_last=True)
259
260     # Training history
261     history = {
262         'epoch': [],
263         'loss': []
264     }
265
266     if verbose:
267         print(f"\nTraining Diffusion Model...")
268         print(f"    Timesteps: {num_timesteps}")
269         print(f"    Hidden dims: {hidden_dims}")
270         print(f"    Learning rate: {lr}")
271         print(f"    Epochs: {n_epochs}, Batch size: {batch_size}")
272
273     # Training loop
274     model.train()
275     for epoch in range(n_epochs):
276         epoch_losses = []
277
278         for batch_data in dataloader:
279             x_0 = batch_data[0]
280             current_batch_size = x_0.size(0)
281
282             # Sample random timesteps
```

```
282         t = torch.randint(0, num_timesteps, (current_batch_size,),  
283                             device=self.device)  
  
284         # Sample noise  
285         noise = torch.randn_like(x_0)  
  
286         # Forward diffusion  
287         x_t = scheduler.q_sample(x_0, t, noise)  
  
288         # Predict noise  
289         optimizer.zero_grad()  
290         predicted_noise = model(x_t, t.float())  
  
291         # Loss  
292         loss = criterion(predicted_noise, noise)  
293         loss.backward()  
294         optimizer.step()  
  
295         epoch_losses.append(loss.item())  
  
296         avg_loss = np.mean(epoch_losses)  
297         history['epoch'].append(epoch + 1)  
298         history['loss'].append(avg_loss)  
  
299  
300  
301     if verbose and (epoch + 1) % 20 == 0:  
302         print(f"Epoch [{epoch+1:4d}/{n_epochs}] | Loss: {avg_loss:.6f}")  
303  
304  
305     # Store results  
306     self.diffusion_results = {  
307         'model': model,  
308         'scheduler': scheduler,  
309         'history': history,  
310         'config': {  
311             'num_timesteps': num_timesteps,  
312             'hidden_dims': hidden_dims,  
313             'time_emb_dim': time_emb_dim,  
314             'lr': lr,  
315             'batch_size': batch_size,  
316             'n_epochs': n_epochs  
317         }  
318     }  
319  
320     self.training_history = history  
  
321  
322     if verbose:  
323         self._plot_training_history(history)  
  
324  
325     return self.diffusion_results  
  
326  
327  
328 def tune_hyperparameters(self,  
329                         num_timesteps_values=[200, 500, 1000],  
330                         lr_values=[1e-4, 5e-4, 1e-3],  
331                         hidden_configs=[
```

```

334                 [128, 256, 128],
335                 [256, 512, 256],
336                 [256, 512, 512, 256]
337             ],
338             n_epochs=150,
339             n_eval_samples=500,
340             verbose=False):
341     if self.data is None:
342         self.process_data()
343
344     results = []
345     total_configs = len(num_timesteps_values) * len(lr_values) * len(
346                             hidden_configs)
347     current_config = 0
348
349     if verbose:
350         print("Hyperparameter Tuning for Diffusion Model")
351         print(f"Total configurations: {total_configs}")
352
353     for num_timesteps in num_timesteps_values:
354         for lr in lr_values:
355             for hidden_dims in hidden_configs:
356                 current_config += 1
357                 if verbose:
358                     print(f"\n[{current_config}/{total_configs}] "
359                         f"T={num_timesteps}, lr={lr}, hidden={"
360                                         hidden_dims
361                                         }")
362
363                 # Train model
364                 self.fit_diffusion(
365                     num_timesteps=num_timesteps,
366                     hidden_dims=hidden_dims,
367                     lr=lr,
368                     n_epochs=n_epochs,
369                     verbose=False
370                 )
371
372                 # Generate and evaluate samples
373                 samples = self.generate_samples(n_samples=n_eval_samples,
374                                                 verbose=False)
375                 metrics = self.evaluate_samples(samples, verbose=False)
376
377                 # Store results
378                 result = {
379                     'num_timesteps': num_timesteps,
380                     'lr': lr,
381                     'hidden_dims': str(hidden_dims),
382                     'final_loss': self.training_history['loss'][-1],
383                     'mean_diff': abs(metrics['generated_mean'] - metrics['
384                                     original_mean']),
385                     'std_diff': abs(metrics['generated_std'] - metrics['
386                                     original_std'])
387                 }
388

```

```

381             'diversity_ratio': metrics.get('diversity_ratio', 0)
382         ]
383     results.append(result)
384
385     if verbose:
386         print(f"  Loss: {result['final_loss']:.6f} | "
387               f"Mean Diff: {result['mean_diff']:.4f} | "
388               f"Std Diff: {result['std_diff']:.4f}")
389
390     results_df = pd.DataFrame(results)
391
392     if verbose:
393         print("Tuning Results Summary:")
394         print(results_df.to_string(index=False))
395
396     # Find best configuration
397     best_idx = results_df['mean_diff'].idxmin()
398     best_config = results_df.iloc[best_idx]
399     print("Best Configuration (by Mean Diff):")
400     print(f"  Timesteps: {best_config['num_timesteps']} ")
401     print(f"  Learning Rate: {best_config['lr']} ")
402     print(f"  Hidden Dims: {best_config['hidden_dims']} ")
403     print(f"  Mean Diff: {best_config['mean_diff']:.4f} ")
404     print(f"  Std Diff: {best_config['std_diff']:.4f} ")
405
406     self._plot_tuning_results(results_df)
407
408     return results_df
409
410 def _plot_tuning_results(self, results_df):
411     fig, axes = plt.subplots(1, 3, figsize=(18, 5))
412
413     # Mean difference heatmap (timesteps x lr)
414     pivot_mean = results_df.pivot_table(
415         values='mean_diff',
416         index='num_timesteps',
417         columns='lr',
418         aggfunc='mean'
419     )
420     sns.heatmap(pivot_mean, annot=True, fmt='.4f', cmap='YlOrRd', ax=axes[0])
421     axes[0].set_title('Mean Difference (Lower is Better)')
422     axes[0].set_xlabel('Learning Rate')
423     axes[0].set_ylabel('Timesteps')
424
425     # Std difference heatmap
426     pivot_std = results_df.pivot_table(
427         values='std_diff',
428         index='num_timesteps',
429         columns='lr',
430         aggfunc='mean'
431     )

```

```

432     sns.heatmap(pivot_std, annot=True, fmt='.4f', cmap='YlOrRd', ax=axes[1])
433         ])
434     axes[1].set_title('Std Difference (Lower is Better)')
435     axes[1].set_xlabel('Learning Rate')
436     axes[1].set_ylabel('Timesteps')

437     # Final loss by architecture
438     arch_loss = results_df.groupby('hidden_dims')['final_loss'].mean()
439     colors = plt.cm.Set2(np.linspace(0, 1, len(arch_loss)))
440     axes[2].bar(range(len(arch_loss)), arch_loss.values,
441                 color=colors, edgecolor='black')
442     axes[2].set_xticks(range(len(arch_loss)))
443     axes[2].set_xticklabels([s.replace(',', ', ', '\n') for s in arch_loss.
444                             index],
445                             fontsize=8, rotation=0)
446     axes[2].set_ylabel('Final Loss')
447     axes[2].set_title('Final Loss by Architecture (Lower is Better)')
448     axes[2].grid(axis='y', alpha=0.3)

449     plt.tight_layout()
450     plt.savefig(os.path.join(self.figure_dir, '2c_diffusion_tuning.png'),
451                 dpi=150)
452     plt.show()

453 def _plot_training_history(self, history):
454     fig, ax = plt.subplots(figsize=(10, 5))

455     epochs = history['epoch']
456     losses = history['loss']

457     ax.plot(epochs, losses, 'b-', linewidth=2)
458     ax.set_xlabel('Epoch')
459     ax.set_ylabel('Loss (MSE)')
460     ax.set_title('Diffusion Model Training Loss')
461     ax.grid(True, alpha=0.3)

462     plt.tight_layout()
463     plt.savefig(os.path.join(self.figure_dir, '2c_diffusion_training.png'))
464                 , dpi=150)
465     plt.show()

466     def generate_samples(self, n_samples=100, verbose=False):
467         model = self.diffusion_results['model']
468         scheduler = self.diffusion_results['scheduler']
469         num_timesteps = self.diffusion_results['config']['num_timesteps']
470         input_dim = self.data.shape[1]

471         model.eval()

472         with torch.no_grad():
473             # Start from pure noise
474             x = torch.randn(n_samples, input_dim, device=self.device)

475             # Reverse diffusion

```

```

482         for t in reversed(range(num_timesteps)):
483             t_batch = torch.full((n_samples,), t, device=self.device,
484                                 dtype=torch.long)
485             x = scheduler.p_sample(model, x, t_batch)
486
487             generated_scaled = x.cpu().numpy()
488
489             # Inverse transform to original scale
490             generated = self.scaler.inverse_transform(generated_scaled)
491
492             # Clip to valid range [0, 16]
493             generated = np.clip(generated, 0, 16)
494
495             self.generated_samples = {
496                 'samples': generated,
497                 'samples_scaled': generated_scaled,
498                 'n_samples': n_samples
499             }
500
501             if verbose:
502                 print(f"\nGenerated {n_samples} samples from Diffusion Model")
503                 print(f"Generated samples shape: {generated.shape}")
504                 print(f"Generated samples range: [{generated.min():.2f}, {generated.max():.2f}]")
505
506             return generated
507
508     def evaluate_samples(self, generated_samples=None, verbose=False):
509         """Evaluate quality of generated samples."""
510         if generated_samples is None:
511             generated_samples = self.generated_samples['samples']
512
513         metrics = {}
514
515         # Sample statistics comparison
516         metrics['original_mean'] = self.data.mean()
517         metrics['original_std'] = self.data.std()
518         metrics['generated_mean'] = generated_samples.mean()
519         metrics['generated_std'] = generated_samples.std()
520
521         # Sparsity check
522         threshold = 0.5
523         metrics['original_sparsity'] = (self.data < threshold).mean()
524         metrics['generated_sparsity'] = (generated_samples < threshold).mean()
525
526         # Diversity check
527         try:
528             gen_pca = PCA(n_components=10).fit_transform(generated_samples)
529             pairwise_dist = cdist(gen_pca, gen_pca, metric='euclidean')
530             np.fill_diagonal(pairwise_dist, np.inf)
531             metrics['avg_nn_distance'] = pairwise_dist.min(axis=1).mean()
532
533             orig_pca = PCA(n_components=10).fit_transform(self.data)
534             orig_pairwise = cdist(orig_pca, orig_pca, metric='euclidean')

```

```

534         np.fill_diagonal(orig_pairwise, np.inf)
535         metrics['orig_avg_nn_distance'] = orig_pairwise.min(axis=1).mean()
536
537         metrics['diversity_ratio'] = metrics['avg_nn_distance'] / metrics[
538                                         'orig_avg_nn_distance']
539     except Exception as e:
540         metrics['diversity_ratio'] = None
541
542     if verbose:
543         self._print_evaluation_metrics(metrics)
544
545     return metrics
546
547 def _print_evaluation_metrics(self, metrics):
548     """Print evaluation metrics."""
549     print("\n--- Diffusion Model Evaluation Metrics ---")
550     print(f"Mean (Original): {metrics['original_mean']:.4f}")
551     print(f"Mean (Generated): {metrics['generated_mean']:.4f}")
552     print(f"Std (Original): {metrics['original_std']:.4f}")
553     print(f"Std (Generated): {metrics['generated_std']:.4f}")
554     print(f"Sparsity (Original): {metrics['original_sparsity']:.4f}")
555     print(f"Sparsity (Generated): {metrics['generated_sparsity']:.4f}")
556     if metrics.get('diversity_ratio') is not None:
557         print(f"Diversity Ratio: {metrics['diversity_ratio']:.4f}")
558
559 def visualize_generated_samples(self, generated_samples=None, n_display=25):
560     if generated_samples is None:
561         generated_samples = self.generated_samples['samples']
562
563     grid_size = int(np.sqrt(n_display))
564     fig, axes = plt.subplots(grid_size, grid_size, figsize=(10, 10))
565     axes = axes.flatten()
566
567     for i in range(min(n_display, len(generated_samples))):
568         axes[i].imshow(generated_samples[i].reshape(8, 8), cmap='gray')
569         axes[i].axis('off')
570
571     plt.suptitle('Generated Digits from Diffusion Model', fontsize=14)
572     plt.tight_layout()
573     plt.savefig(os.path.join(self.figure_dir,
574                             '2c_generated_samples_diffusion.' +
575                             'png'), dpi=150)
576     plt.show()
577
578 def compare_distributions(self, generated_samples=None):
579     if generated_samples is None:
580         generated_samples = self.generated_samples['samples']
581
582     fig, axes = plt.subplots(1, 3, figsize=(18, 5))
583
584     # Pixel intensity distribution
585     axes[0].hist(self.data.flatten(), bins=50, alpha=0.6, label='Original',
586

```

```

583             color='blue', density=True)
584     axes[0].hist(generated_samples.flatten(), bins=50, alpha=0.6, label='
585                 Generated',
586                 color='red', density=True)
587     axes[0].set_xlabel('Pixel Intensity')
588     axes[0].set_ylabel('Density')
589     axes[0].set_title('Pixel Intensity Distribution')
590     axes[0].legend()
591     axes[0].grid(alpha=0.3)

592     # Sparsity comparison
593     sparsity_orig = (self.data < 0.5).sum(axis=1).mean()
594     sparsity_gen = (generated_samples < 0.5).sum(axis=1).mean()

595     categories = ['Original', 'Generated']
596     sparsities = [sparsity_orig, sparsity_gen]
597     axes[1].bar(categories, sparsities, color=['blue', 'red'], alpha=0.7,
598                  edgecolor='black')
599     axes[1].set_ylabel('Average # of Near-Zero Pixels')
600     axes[1].set_title('Sparsity Comparison')
601     axes[1].grid(axis='y', alpha=0.3)

602     # PCA projection comparison
603     pca = PCA(n_components=2)
604     orig_pca = pca.fit_transform(self.data)
605     gen_pca = pca.transform(generated_samples)

606     axes[2].scatter(orig_pca[:, 0], orig_pca[:, 1], alpha=0.3, label='
607                     Original', s=10)
608     axes[2].scatter(gen_pca[:, 0], gen_pca[:, 1], alpha=0.5, label='
609                     Generated', s=10)
610     axes[2].set_xlabel('PC1')
611     axes[2].set_ylabel('PC2')
612     axes[2].set_title('PCA Projection Comparison')
613     axes[2].legend()
614     axes[2].grid(alpha=0.3)

615     plt.tight_layout()
616     plt.savefig(os.path.join(self.figure_dir, ' '
617                             '2c_distribution_comparison_diffusion
618                             .png'), dpi=150)
619     plt.show()

620     def visualize_diffusion_process(self, n_steps=10):
621         if self.diffusion_results is None:
622             raise ValueError("Diffusion model not trained. Call fit_diffusion
623                             () first.")

624         model = self.diffusion_results['model']
625         scheduler = self.diffusion_results['scheduler']
626         num_timesteps = self.diffusion_results['config']['num_timesteps']
627         input_dim = self.data.shape[1]

628         model.eval()

```

```

630
631     # Choose timesteps to visualize
632     vis_timesteps = np.linspace(num_timesteps - 1, 0, n_steps, dtype=int)
633
634     with torch.no_grad():
635         # Start from pure noise
636         x = torch.randn(1, input_dim, device=self.device)
637
638         samples_at_timesteps = []
639
640         # Reverse diffusion
641         for t in reversed(range(num_timesteps)):
642             t_batch = torch.full((1,), t, device=self.device, dtype=torch.
643                                 long)
643             x = scheduler.p_sample(model, x, t_batch)
644
645             if t in vis_timesteps:
646                 sample = x.cpu().numpy()
647                 sample = self.scaler.inverse_transform(sample)
648                 sample = np.clip(sample, 0, 16)
649                 samples_at_timesteps.append((t, sample.reshape(8, 8)))
650
651     # Sort by timestep (descending)
652     samples_at_timesteps.sort(key=lambda x: x[0], reverse=True)
653
654     # Plot
655     fig, axes = plt.subplots(1, n_steps, figsize=(2 * n_steps, 2.5))
656     for i, (t, img) in enumerate(samples_at_timesteps):
657         axes[i].imshow(img, cmap='gray')
658         axes[i].axis('off')
659         axes[i].set_title(f't={t}', fontsize=10)
660
661     plt.suptitle('Reverse Diffusion Process (Noise → Image)', fontsize=14)
662     plt.tight_layout()
663     plt.savefig(os.path.join(self.figure_dir, '2c_diffusion_process.png'),
664                 dpi=150)
664     plt.show()
665
666 def compare_with_real_samples(self, n_display=10):
667     if self.generated_samples is None:
668         self.generate_samples(n_samples=n_display)
669
670     generated = self.generated_samples['samples'][:n_display]
671     real_indices = np.random.choice(len(self.data), n_display, replace=
672                                     False)
672     real = self.data[real_indices]
673
674     fig, axes = plt.subplots(2, n_display, figsize=(2 * n_display, 4))
675
676     for i in range(n_display):
677         axes[0, i].imshow(real[i].reshape(8, 8), cmap='gray')
678         axes[0, i].axis('off')
679         if i == 0:
680             axes[0, i].set_ylabel('Real', fontsize=12)

```

```
681
682     axes[1, i].imshow(generated[i].reshape(8, 8), cmap='gray')
683     axes[1, i].axis('off')
684     if i == 0:
685         axes[1, i].set_ylabel('Generated', fontsize=12)
686
687     plt.suptitle('Real vs Generated Samples (Diffusion)', fontsize=14)
688     plt.tight_layout()
689     plt.savefig(os.path.join(self.figure_dir, '2c_real_vs_generated_diffusion.
690                                         png'), dpi=150)
plt.show()
```