Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №2
По дисциплине: «Интеллектуальный анализ данных»
Тема: “Автоэнкодеры”

Выполнил:
Студент 4 курса
Группы ИИ-24
Мшар В.В.
Проверила:
Андренко К. В.

Брест 2025

Цель: научиться применять автоэнкодеры для осуществления визуализации данных и их анализа.

Общее задание

1. Используя выборку по варианту, осуществить проецирование данных на плоскость первых двух и трех главных компонент с использованием нейросетевой модели автоэнкодера (с двумя и тремя нейронами в среднем слое);

2. Выполнить визуализацию полученных главных компонент с использованием средств библиотеки matplotlib, обозначая экземпляры разных классов с использованием разных цветовых маркеров;

3. Реализовать метод t-SNE для визуализации данных (использовать также 2 и 3 компонента), построить соответствующую визуализацию;

4. Применить к данным метод PCA (2 и 3 компонента), реализованный в ЛР №1, сделать выводы;

5. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

№ варианта	Выборка	Класс
12	Mushroom	poisonous

Ход работы:

Код программы:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
import seaborn as sns
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import warnings

# Suppress warnings for cleaner output
warnings.filterwarnings('ignore')

settings = {
    'dataset_url': "https://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agaricus-lepiota.data",
```

```
    'column_names': [
        'class', 'cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',
        'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',
        'stalk-shape', 'stalk-root', 'stalk-surface-above-ring',
        'stalk-surface-below-ring', 'stalk-color-above-ring',
        'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number',
        'ring-type', 'spore-print-color', 'population', 'habitat'
    ],
    'target_class': 'poisonous',
    'autoencoder': {
        'epochs': 100,
        'batch_size': 64,
        'lr': 0.001,
        'encoder_layers': [64, 32], # Hidden layers for the encoder
    },
    'tsne': {
        'perplexities': [20, 30, 50],
        'random_state': 42
    },
    'visualization': {
        'colors': {'edible': '#2ecc71', 'poisonous': '#e74c3c'},
        'labels': {0: 'Edible', 1: 'Poisonous'},
        'output_file_2d': 'mushroom_analysis_2d.png',
        'output_file_3d': 'mushroom_analysis_3d.png'
    }
}

# Helper for formatted printing
def log_step(message):
    print(f"\n{'=' * 80}\n{message}\n{'=' * 80}")

def prepare_data(url, columns):
    """Loads, cleans, and preprocesses the Mushroom dataset."""
    log_step("[1] Loading and Preparing Mushroom Data")
    try:
        df = pd.read_csv(url, names=columns, na_values='?')
        print(f"✓ Data loaded successfully: {df.shape[0]} rows, {df.shape[1]} columns")
    except Exception as e:
        print(f"✗ Failed to load data from URL. Error: {e}")
        return None, None, None

    df = df.dropna()
    print(f"  - Missing values removed. Remaining rows: {df.shape[0]}")

    y = df['class'].map({'e': 0, 'p': 1})
    X = df.drop('class', axis=1)

    print(f"  - Class distribution:")
    print(f"    - Edible (0): {(y == 0).sum()} ({(y == 0).sum() / len(y) * 100:.1f}%)")
    print(f"    - Poisonous (1): {(y == 1).sum()} ({(y == 1).sum() / len(y) * 100:.1f}%)")
```

```python
    X_encoded = X.apply(LabelEncoder().fit_transform)

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_encoded)
    print(f"✓ Features encoded and scaled. Shape: {X_scaled.shape}")

    X_tensor = torch.FloatTensor(X_scaled)
    y_tensor = torch.LongTensor(y.values)

    return X_scaled, y.values, X_tensor

class ConfigurableAutoencoder(nn.Module):
    def __init__(self, input_dim, encoding_dim, hidden_layers):
        super().__init__()

        encoder_layers = []
        last_dim = input_dim
        for layer_size in hidden_layers:
            encoder_layers.extend([nn.Linear(last_dim, layer_size), nn.LeakyReLU(0.1)])
            last_dim = layer_size
        encoder_layers.append(nn.Linear(last_dim, encoding_dim))
        self.encoder = nn.Sequential(*encoder_layers)

        decoder_layers = []
        last_dim = encoding_dim
        for layer_size in reversed(hidden_layers):
            decoder_layers.extend([nn.Linear(last_dim, layer_size), nn.LeakyReLU(0.1)])
            last_dim = layer_size
        decoder_layers.append(nn.Linear(last_dim, input_dim))
        self.decoder = nn.Sequential(*decoder_layers)

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

    def encode(self, x):
        return self.encoder(x)

def run_autoencoder_training(data_tensor, input_dim, encoding_dim, config):
    """Initializes and trains an autoencoder."""
    log_step(f"[2] Training Autoencoder for {encoding_dim}D Representation")

    model = ConfigurableAutoencoder(
        input_dim=input_dim,
        encoding_dim=encoding_dim,
        hidden_layers=config['encoder_layers']
    )
```

```
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=config['lr'])
    dataloader = DataLoader(TensorDataset(data_tensor), batch_size=config['batch_size'],
shuffle=True)

    losses = []
    for epoch in range(config['epochs']):
        for batch in dataloader:
            x_batch = batch[0]
            reconstructed = model(x_batch)
            loss = criterion(reconstructed, x_batch)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        losses.append(loss.item())
        if (epoch + 1) % 20 == 0:
            print(f"  Epoch {epoch+1}/{config['epochs']}, MSE Loss: {loss.item():.6f}")

    with torch.no_grad():
        encoded_data = model.encode(data_tensor).numpy()

    print(f"✓ Training complete. Final Loss: {losses[-1]:.6f}")
    return encoded_data, losses

def apply_dim_reduction(X_scaled, tsne_config):
    """Applies PCA and t-SNE for 2D and 3D representations."""
    log_step("[3] Applying PCA and t-SNE")

    # PCA
    pca_2d = PCA(n_components=2).fit_transform(X_scaled)
    pca_3d = PCA(n_components=3).fit_transform(X_scaled)
    print("✓ PCA applied for 2D and 3D.")

    # t-SNE
    tsne_results = {}
    for perp in tsne_config['perplexities']:
        print(f"  - Running t-SNE with perplexity = {perp}...")
        tsne_2d = TSNE(n_components=2, perplexity=perp, init='pca',
random_state=tsne_config['random_state'])
        tsne_3d = TSNE(n_components=3, perplexity=perp, init='pca',
random_state=tsne_config['random_state'])
        tsne_results[perp] = {
            '2d': tsne_2d.fit_transform(X_scaled),
            '3d': tsne_3d.fit_transform(X_scaled)
        }
    print("✓ t-SNE applied for all perplexities.")

    return {'pca_2d': pca_2d, 'pca_3d': pca_3d, 'tsne': tsne_results}
```

```python
def plot_scatter(ax, data, y, title, labels, colors, dim=2, **kwargs):
    """Helper function to create a single scatter plot."""
    for class_val, label in labels.items():
        mask = y == class_val
        if dim == 2:
            ax.scatter(data[mask, 0], data[mask, 1], c=colors[label.lower()], label=label, alpha=0.7, s=20)
        elif dim == 3:
            ax.scatter(data[mask, 0], data[mask, 1], data[mask, 2], c=colors[label.lower()], label=label,
alpha=0.6, s=20)

    ax.set_title(title, fontsize=14, fontweight='bold')
    ax.set_xlabel('Component 1' if 'PCA' not in title else 'PC1')
    ax.set_ylabel('Component 2' if 'PCA' not in title else 'PC2')
    if dim == 3:
        ax.set_zlabel('Component 3' if 'PCA' not in title else 'PC3')
    ax.legend()
    ax.grid(True, linestyle='--', alpha=0.4)

def create_visualizations(results, vis_config):
    """Generates and saves 2D and 3D comparison plots."""
    log_step("[4] Generating Visualizations")
    sns.set_style("whitegrid")

    # ===== 2D Visualization =====
    fig_2d = plt.figure(figsize=(24, 14))
    gs = GridSpec(2, 4, figure=fig_2d)

    # Autoencoder 2D
    ax1 = fig_2d.add_subplot(gs[0, 0])
    plot_scatter(ax1, results['ae_2d'], results['y'], 'Autoencoder (2D)', vis_config['labels'],
vis_config['colors'])

    # t-SNE 2D
    for i, p in enumerate(settings['tsne']['perplexities']):
        ax = fig_2d.add_subplot(gs[0, i + 1])
        plot_scatter(ax, results['tsne'][p]['2d'], results['y'], f't-SNE (perplexity={p})', vis_config['labels'],
vis_config['colors'])

    # PCA 2D
    ax5 = fig_2d.add_subplot(gs[1, 0])
    plot_scatter(ax5, results['pca_2d'], results['y'], 'PCA (2D)', vis_config['labels'], vis_config['colors'])

    # Learning Curve
    ax6 = fig_2d.add_subplot(gs[1, 1])
    ax6.plot(results['losses_2d'], label='2D Autoencoder Loss', color='royalblue')
    ax6.plot(results['losses_3d'], label='3D Autoencoder Loss', color='darkorange', linestyle='--')
    ax6.set_title('Autoencoder Learning Curves', fontsize=14, fontweight='bold')
    ax6.set_xlabel('Epoch')
    ax6.set_ylabel('MSE Loss')
    ax6.legend()
```

```
    ax6.grid(True, linestyle='--', alpha=0.4)

    fig_2d.suptitle('Comparison of 2D Dimensionality Reduction Techniques', fontsize=20,
fontweight='bold')
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.savefig(vis_config['output_file_2d'], dpi=300)
    print(f"✓ 2D visualizations saved to: {vis_config['output_file_2d']}")

    # ===== 3D Visualization =====
    fig_3d = plt.figure(figsize=(20, 10))

    # Autoencoder 3D
    ax1 = fig_3d.add_subplot(1, 3, 1, projection='3d')
    plot_scatter(ax1, results['ae_3d'], results['y'], 'Autoencoder (3D)', vis_config['labels'],
vis_config['colors'], dim=3)

    # t-SNE 3D
    ax2 = fig_3d.add_subplot(1, 3, 2, projection='3d')
    plot_scatter(ax2, results['tsne'][30]['3d'], results['y'], 't-SNE 3D (perplexity=30)',
vis_config['labels'], vis_config['colors'], dim=3)

    # PCA 3D
    ax3 = fig_3d.add_subplot(1, 3, 3, projection='3d')
    plot_scatter(ax3, results['pca_3d'], results['y'], 'PCA (3D)', vis_config['labels'], vis_config['colors'],
dim=3)

    fig_3d.suptitle('Comparison of 3D Dimensionality Reduction Techniques', fontsize=20,
fontweight='bold')
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.savefig(vis_config['output_file_3d'], dpi=300)
    print(f"✓ 3D visualizations saved to: {vis_config['output_file_3d']}")

def print_summary(results):
    """Prints a summary of the findings."""
    log_step("ANALYSIS AND CONCLUSIONS")

    # Autoencoder Summary
    print("\n1. AUTOENCODER:")
    print("   - Successfully trained to create low-dimensional, non-linear representations.")
    print("   - Provides a strong separation between edible and poisonous classes in both 2D and 3D.")
    print(f"   - Final 2D Loss: {results['losses_2d'][-1]:.6f} | Final 3D Loss: {results['losses_3d'][-
1]:.6f}")

    # t-SNE Summary
    print("\n2. T-SNE (t-Distributed Stochastic Neighbor Embedding):")
    print("   - Excels at visualizing local neighborhood structures, forming very distinct clusters.")
    print("   - Perplexity values between 30-50 appear optimal for this dataset.")
    print("   - Caveat: The global arrangement and distances between clusters are not meaningful.")

    # PCA Summary
```

```
    pca_2d_var = np.sum(PCA(n_components=2).fit(results['X_scaled']).explained_variance_ratio_)
    pca_3d_var = np.sum(PCA(n_components=3).fit(results['X_scaled']).explained_variance_ratio_)
    print("\n3. PCA (Principal Component Analysis):")
    print(f"   - As a linear method, it captures a significant amount of variance: {pca_2d_var:.2%} in
2D and {pca_3d_var:.2%} in 3D.")
    print("   - Very fast and interpretable but shows more overlap between classes than non-linear
methods.")

    # Comparison
    print("\n4. COMPARATIVE INSIGHTS:")
    print("   - For Visualization: t-SNE is the clear winner, revealing the cleanest class separation.")
    print("   - For Feature Engineering: The Autoencoder provides powerful, learned features that can
be used in downstream models.")
    print("   - For Baseline/Speed: PCA offers a quick and easy-to-understand initial look at the data's
structure.")
    print("   - All methods confirm that the features in the dataset are highly effective for
distinguishing poisonous from edible mushrooms.")

    print("\n" + "=" * 80)
    plt.show()

def main():
    """Main function to run the entire pipeline."""
    X_scaled, y, X_tensor = prepare_data(settings['dataset_url'], settings['column_names'])

    if X_scaled is None:
        return

    input_dim = X_scaled.shape[1]

    ae_2d, losses_2d = run_autoencoder_training(X_tensor, input_dim, 2, settings['autoencoder'])
    ae_3d, losses_3d = run_autoencoder_training(X_tensor, input_dim, 3, settings['autoencoder'])

    other_reductions = apply_dim_reduction(X_scaled, settings['tsne'])

    all_results = {
        'X_scaled': X_scaled,
        'y': y,
        'ae_2d': ae_2d,
        'losses_2d': losses_2d,
        'ae_3d': ae_3d,
        'losses_3d': losses_3d,
        **other_reductions
    }

    create_visualizations(all_results, settings['visualization'])
    print_summary(all_results)

if __name__ == '__main__':
    main()
```

Вывод: Я научился применять метод PCA для осуществления визуализации данных.