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-23

Копач А. 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.

Задание по вариантам

7	Mushroom	poisonous

Код программы:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import
LabelEncoder, StandardScaler
from sklearn.model_selection
import train_test_split
from sklearn.decomposition
import PCA
from sklearn.manifold import
TSNE
from sklearn.metrics import
accuracy_score
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import
layers

# Загрузка датасета
from ucimlrepo import
fetch_ucirepo
```

```
mushroom = fetch_ucirepo(id=73)
X = mushroom.data.features
y = mushroom.data.targets

print("Размерность данных:",
X.shape)
print("\nПервые 5 строк
данных:")
print(X.head())
print("\nЦелевая переменная:")
print(y.head())

# Предобработка данных
print("\n=== ПРЕДОБРАБОТКА
ДАННЫХ ===")

# Кодирование категориальных
переменных
label_encoders = {}
X_encoded = X.copy()

for column in
X_encoded.columns:
    le = LabelEncoder()
    X_encoded[column] =
le.fit_transform(X_encoded[colum
n].astype(str))
    label_encoders[column] = le

# Кодирование целевой
переменной
le_target = LabelEncoder()
y_encoded =
le_target.fit_transform(y.values.ra
vel())

print("Уникальные значения
целевой переменной:",
np.unique(y_encoded))
print("Размерность после
кодирования:", X_encoded.shape)

# Масштабирование данных
scaler = StandardScaler()
```

```
X_scaled =
scaler.fit_transform(X_encoded)

# Разделение на train/test
X_train, X_test, y_train, y_test =
train_test_split(
    X_scaled, y_encoded,
test_size=0.2, random_state=42,
stratify=y_encoded
)

print(f"Train size:
{X_train.shape}, Test size:
{X_test.shape}")

# 1. АВТОЭНКОДЕР С 2
НЕЙРОНАМИ
print("\n=== АВТОЭНКОДЕР С
2 НЕЙРОНАМИ ===")

# Архитектура автоэнкодера
input_dim = X_train.shape[1]
encoding_dim_2d = 2

# Создание модели с явным
определением входного слоя
input_layer =
layers.Input(shape=(input_dim,))
encoded = layers.Dense(64,
activation='relu')(input_layer)
encoded = layers.Dense(32,
activation='relu')(encoded)
encoded = layers.Dense(16,
activation='relu')(encoded)
bottleneck_2d =
layers.Dense(encoding_dim_2d,
activation='linear',
name='bottleneck')(encoded)

decoded = layers.Dense(16,
activation='relu')(bottleneck_2d)
decoded = layers.Dense(32,
activation='relu')(decoded)
decoded = layers.Dense(64,
```

```
activation='relu')(decoded)
decoded =
layers.Dense(input_dim,
activation='linear')(decoded)

autoencoder_2d =
keras.Model(input_layer, decoded)
encoder_2d =
keras.Model(input_layer,
bottleneck_2d)

autoencoder_2d.compile(optimizer
='adam', loss='mse')
autoencoder_2d.summary()

# Обучение автоэнкодера
history_2d = autoencoder_2d.fit(
    X_train, X_train,
    epochs=50,  # Уменьшим
количество эпох для скорости
    batch_size=32,
    validation_data=(X_test,
X_test),
    verbose=1,
    shuffle=True
)

# Получение кодированных
признаков
encoded_features_2d =
encoder_2d.predict(X_scaled)
print(f"Размерность
закодированных признаков:
{encoded_features_2d.shape}")

# 2. АВТОЭНКОДЕР С 3
НЕЙРОНАМИ
print("\n=== АВТОЭНКОДЕР С
3 НЕЙРОНАМИ ===")

encoding_dim_3d = 3

input_layer_3d =
layers.Input(shape=(input_dim,))
```

```
encoded_3d = layers.Dense(64,
activation='relu')(input_layer_3d)
encoded_3d = layers.Dense(32,
activation='relu')(encoded_3d)
encoded_3d = layers.Dense(16,
activation='relu')(encoded_3d)
bottleneck_3d =
layers.Dense(encoding_dim_3d,
activation='linear',
name='bottleneck_3d')(encoded_3
d)

decoded_3d = layers.Dense(16,
activation='relu')(bottleneck_3d)
decoded_3d = layers.Dense(32,
activation='relu')(decoded_3d)
decoded_3d = layers.Dense(64,
activation='relu')(decoded_3d)
decoded_3d =
layers.Dense(input_dim,
activation='linear')(decoded_3d)

autoencoder_3d =
keras.Model(input_layer_3d,
decoded_3d)
encoder_3d =
keras.Model(input_layer_3d,
bottleneck_3d)

autoencoder_3d.compile(optimizer
='adam', loss='mse')

history_3d = autoencoder_3d.fit(
    X_train, X_train,
    epochs=50,
    batch_size=32,
    validation_data=(X_test,
X_test),
    verbose=1,
    shuffle=True
)

encoded_features_3d =
encoder_3d.predict(X_scaled)
```

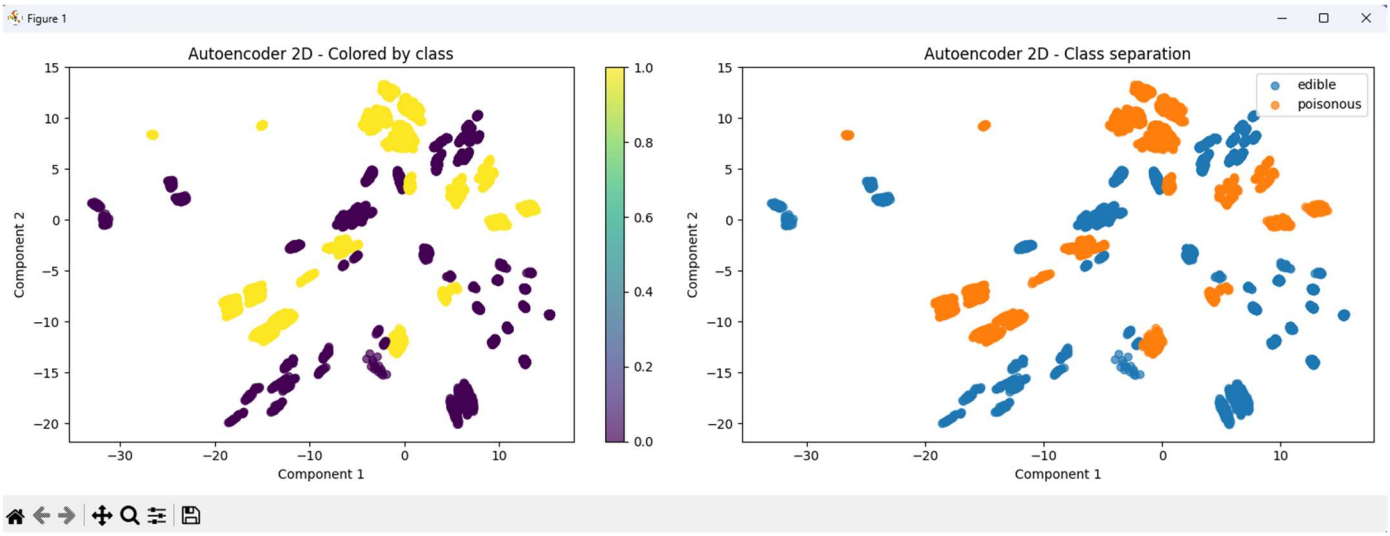
```
# ВИЗУАЛИЗАЦИЯ
РЕЗУЛЬТАТОВ
print("\n=== ВИЗУАЛИЗАЦИЯ
РЕЗУЛЬТАТОВ ===")



# Функция для визуализации
def plot_results_2d(features, title,
class_labels=['edible',
'poisonous']):
    plt.figure(figsize=(15, 5))

    plt.subplot(1, 2, 1)
    scatter = plt.scatter(features[:,
0], features[:, 1], c=y_encoded,
                          cmap='viridis',
alpha=0.7)
    plt.colorbar(scatter)
    plt.xlabel('Component 1')
    plt.ylabel('Component 2')
    plt.title(f'{title} - Colored by
class')

    plt.subplot(1, 2, 2)
    for class_label in
np.unique(y_encoded):
        mask = y_encoded ==
class_label
        plt.scatter(features[mask, 0],
features[mask, 1],

label=class_labels[class_label],
alpha=0.7)
    plt.xlabel('Component 1')
    plt.ylabel('Component 2')
    plt.title(f'{title} - Class
separation')
    plt.legend()

    plt.tight_layout()
    plt.show()
```

```
def plot_results_3d(features, title,
class_labels=['edible',
'poisonous']):
    fig = plt.figure(figsize=(15, 5))

    # Вид 1
    ax1 = fig.add_subplot(1, 2, 1,
projection='3d')
    scatter = ax1.scatter(features[:,
0], features[:, 1], features[:, 2],
                          c=y_encoded,
cmap='viridis', alpha=0.7)
    plt.colorbar(scatter)
    ax1.set_xlabel('Component 1')
    ax1.set_ylabel('Component 2')
    ax1.set_zlabel('Component 3')
    ax1.set_title(f'{title} - 3D
View')

    # Вид 2 (только 2 компоненты
для лучшей видимости)
    ax2 = fig.add_subplot(1, 2, 2)
    for class_label in
np.unique(y_encoded):
        mask = y_encoded ==
class_label
        ax2.scatter(features[mask, 0],
features[mask, 1],

label=class_labels[class_label],
alpha=0.7)
    ax2.set_xlabel('Component 1')
    ax2.set_ylabel('Component 2')
    ax2.set_title(f'{title} - 2D
Projection')
    ax2.legend()

    plt.tight_layout()
    plt.show()


# Визуализация автоэнкодера с 2
нейронами
print("Визуализация
```

автоэнкодера (2D):")

plot_results_2d(encoded_features_

2d, 'Autoencoder 2D')



Визуализация автоэнкодера с 3

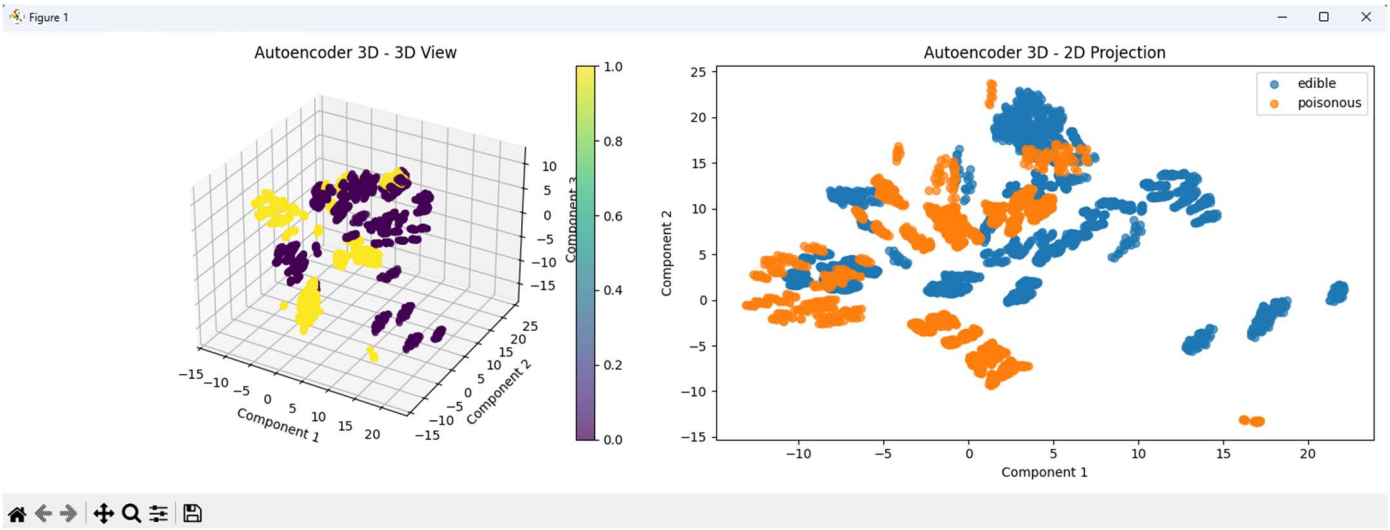
нейронами

print("Визуализация

автоэнкодера (3D):")

plot_results_3d(encoded_features_

3d, 'Autoencoder 3D')



3. МЕТОД t-SNE

print("\n=== МЕТОД t-SNE

===")

t-SNE с различными

значениями перплексивности

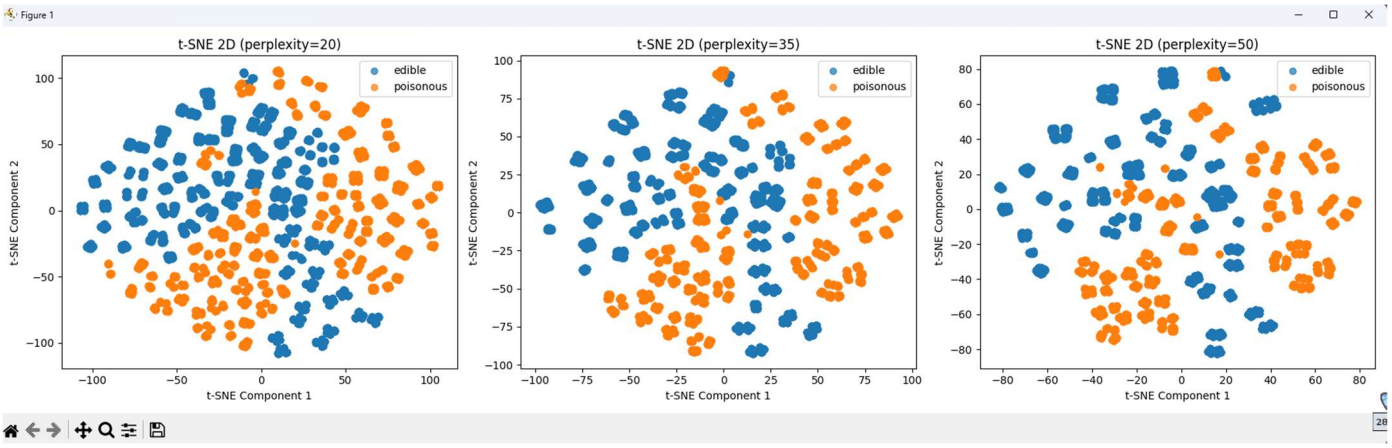
perplexities = [20, 35, 50]

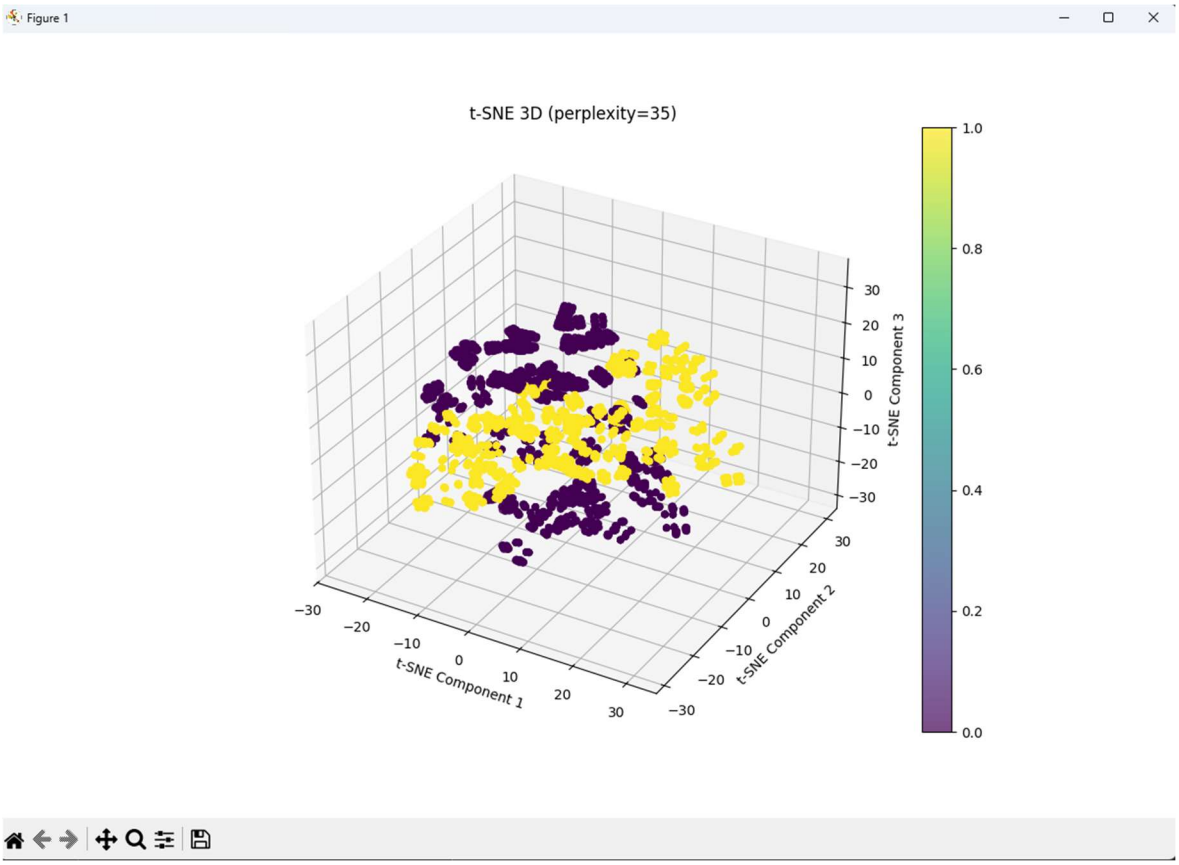
plt.figure(figsize=(18, 5))

for i, perplexity in

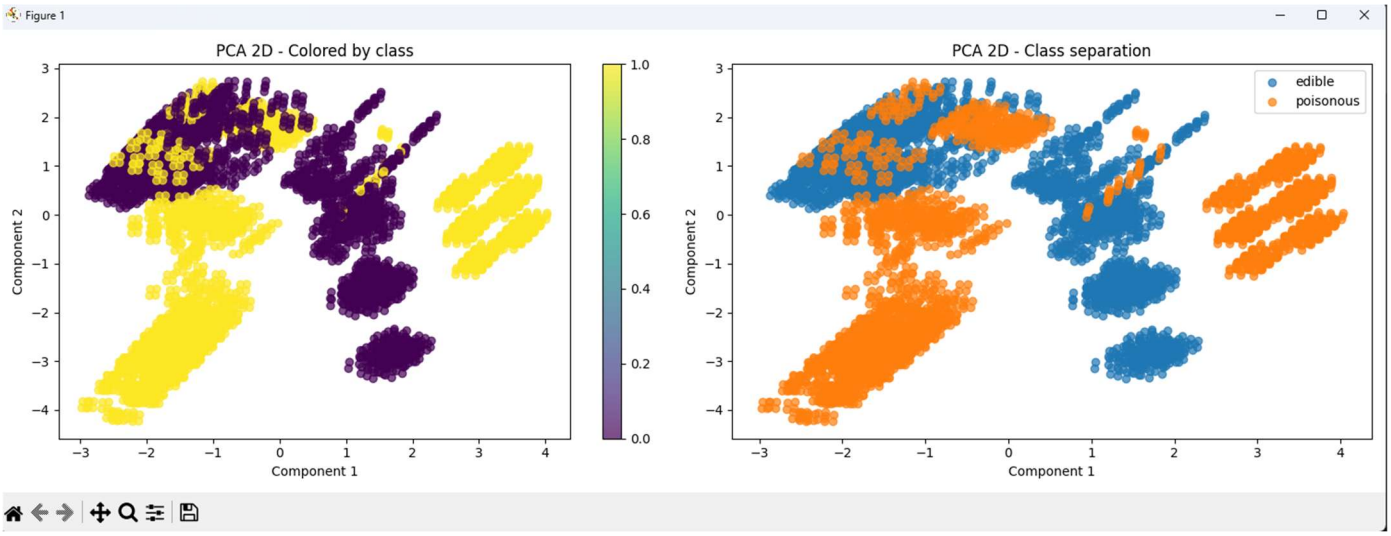
```
enumerate(perplexities, 1):
    tsne_2d =
TSNE(n_components=2,
perplexity=perplexity,
random_state=42, init='pca')
    X_tsne_2d =
tsne_2d.fit_transform(X_scaled)

    plt.subplot(1, 3, i)
    for class_label in
np.unique(y_encoded):
        mask = y_encoded ==
class_label
        plt.scatter(X_tsne_2d[mask,
0], X_tsne_2d[mask, 1],
                    label=['edible',
'poisonous'][class_label],
alpha=0.7)
    plt.xlabel('t-SNE Component 1')
    plt.ylabel('t-SNE Component 2')
    plt.title(f't-SNE 2D
(perplexity={perplexity})')
    plt.legend()

plt.tight_layout()
plt.show()

# t-SNE 3D с лучшим значением
перплексивности
best_perplexity = 35
print(f"t-SNE 3D с
perplexity={best_perplexity}:")
tsne_3d =
TSNE(n_components=3,
perplexity=best_perplexity,
random_state=42, init='pca')
X_tsne_3d =
tsne_3d.fit_transform(X_scaled)

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111,
projection='3d')
scatter = ax.scatter(X_tsne_3d[:,
0], X_tsne_3d[:, 1], X_tsne_3d[:,
```

```
2],
                c=y_encoded,
cmap='viridis', alpha=0.7)
plt.colorbar(scatter)
ax.set_xlabel('t-SNE Component
1')
ax.set_ylabel('t-SNE Component
2')
ax.set_zlabel('t-SNE Component
3')
ax.set_title(f't-SNE 3D
(perplexity={best_perplexity})')
plt.show()
```

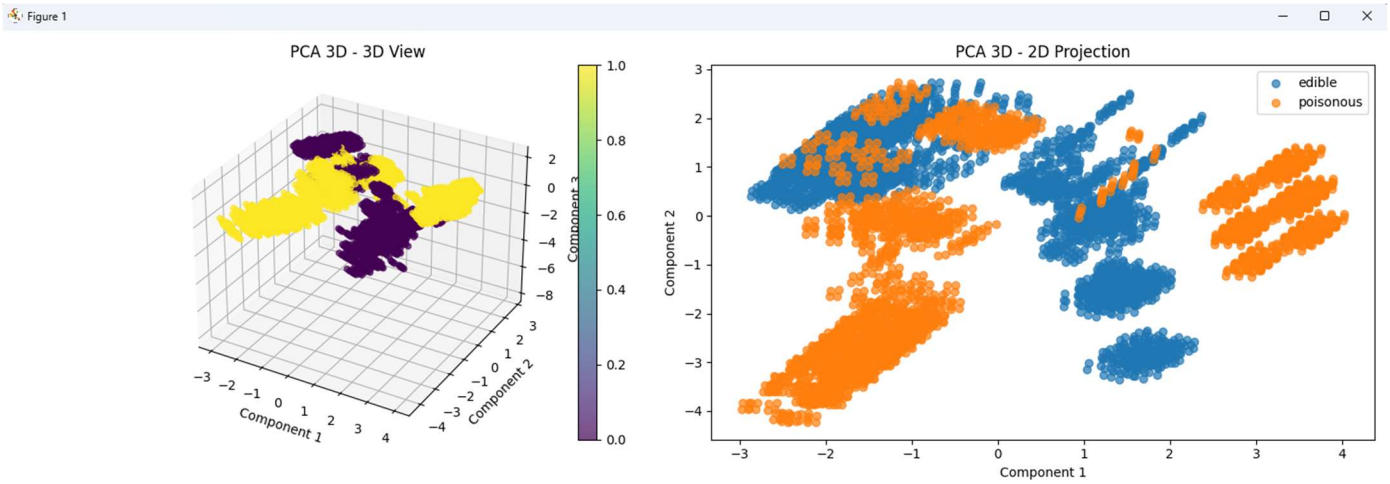




```
# 4. МЕТОД PCA
print("\n=== МЕТОД PCA ===")

# PCA с 2 компонентами
pca_2d = PCA(n_components=2)
X_pca_2d =
pca_2d.fit_transform(X_scaled)

print("Объясненная дисперсия
PCA (2 компоненты):",
pca_2d.explained_variance_ratio_
)
print("Суммарная объясненная
дисперсия:",
sum(pca_2d.explained_variance_r
atio_))
```



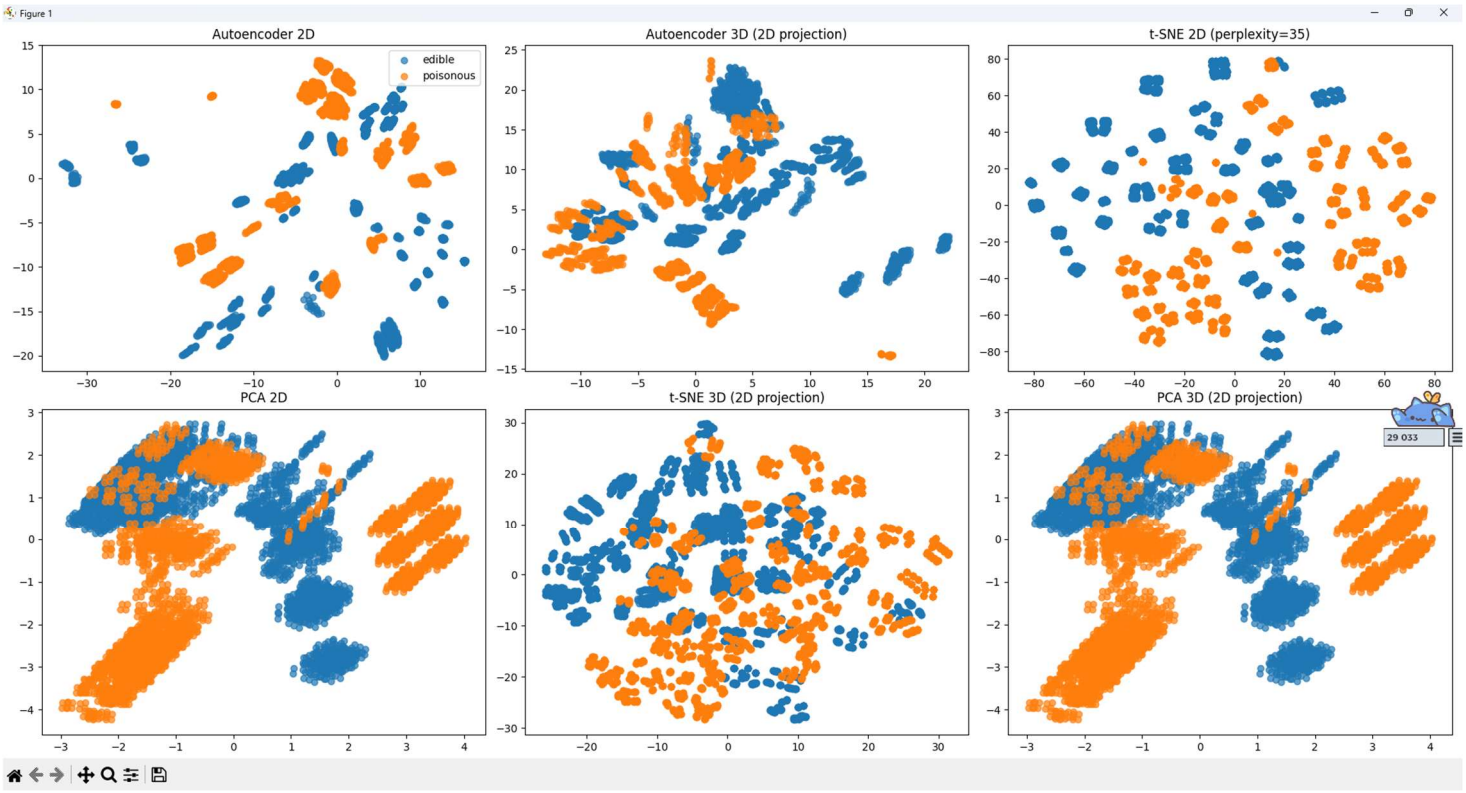

```
# PCA с 3 компонентами
pca_3d = PCA(n_components=3)
X_pca_3d =
pca_3d.fit_transform(X_scaled)

print("\nОбъясненная дисперсия
PCA (3 компоненты):",
pca_3d.explained_variance_ratio_
)
print("Суммарная объясненная
дисперсия:",
sum(pca_3d.explained_variance_r
atio_))
```

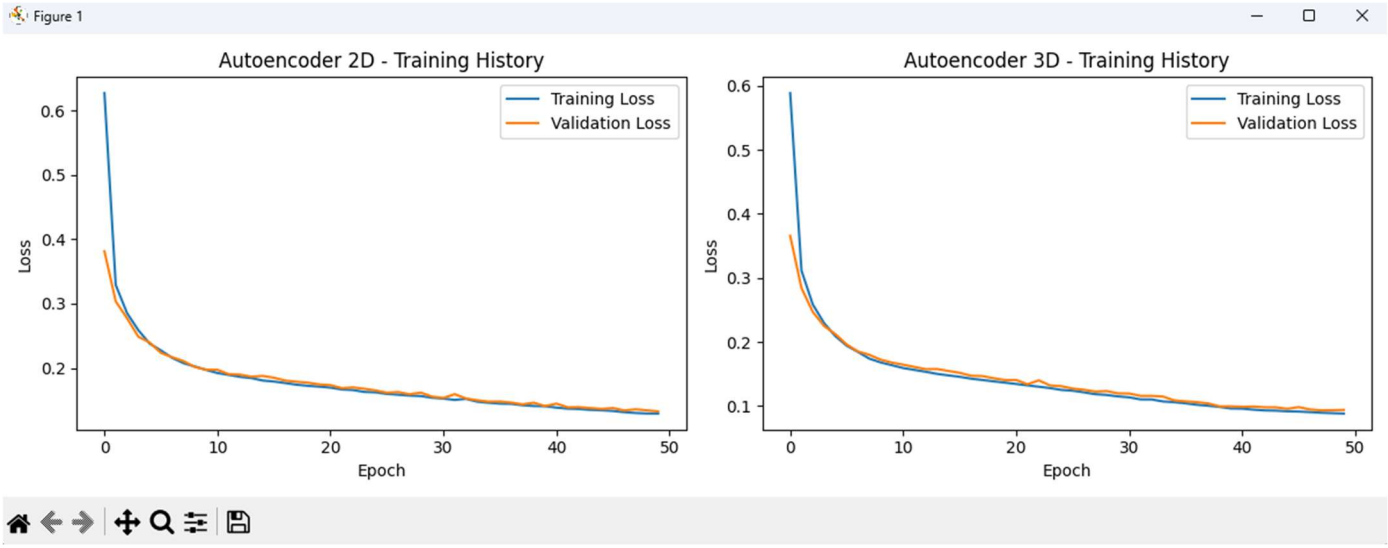



```
# Визуализация PCA
print("Визуализация PCA (2D):")
plot_results_2d(X_pca_2d, 'PCA
2D')

print("Визуализация PCA (3D):")
plot_results_3d(X_pca_3d, 'PCA
3D')

# СРАВНИТЕЛЬНЫЙ АНАЛИЗ
print("\n===
СРАВНИТЕЛЬНЫЙ АНАЛИЗ
===")

# Создание сравнительной
визуализации
fig, axes = plt.subplots(2, 3,
figsize=(20, 12))

methods = [
    (encoded_features_2d,
'Autoencoder 2D'),
    (encoded_features_3d[:, :2],
'Autoencoder 3D (2D projection)'),
    (X_tsne_2d, f't-SNE 2D
(perplexity={best_perplexity})'),
    (X_pca_2d, 'PCA 2D'),
    (X_tsne_3d[:, :2], 't-SNE 3D
(2D projection)'),
    (X_pca_3d[:, :2], 'PCA 3D (2D
projection)')
```

]

```
for i, (features, title) in
enumerate(methods):
    row = i // 3
    col = i % 3
    for class_label in
np.unique(y_encoded):
        mask = y_encoded ==
class_label
        axes[row,
col].scatter(features[mask, 0],
features[mask, 1],
                           label=['edible',
'poisonous'][class_label],
alpha=0.7)
    axes[row, col].set_title(title)
    if i == 0:
        axes[row, col].legend()

plt.tight_layout()
plt.show()
```






```
# ВЫВОДЫ
print("\n=== ВЫВОДЫ ===")
print("1. АВТОЭНКОДЕР:")
print("   - Способен извлекать
нелинейные зависимости в
данных")
print("   - Показывает хорошее
разделение классов в скрытом
пространстве")
print("   - Требует тщательной
настройки архитектуры и
параметров обучения")

print("\n2. t-SNE:")
print("   - Лучше всего
показывает локальные кластеры
и структуры")
print("   - Чувствителен к
параметру perplexity")
print("   - Визуализация более
интуитивно понятна для анализа
кластеров")

print("\n3. PCA:")
print("   - Сохраняет глобальную
структуру данных")
print("   - Объяснимая
дисперсия:",
f"{sum(pca_2d.explained_varianc
e_ratio_):.3f}")
print("   - Быстрый и стабильный
```

```
метод")

print("\n4. СРАВНЕНИЕ:")
print("   - PCA: лучше для
сохранения глобальной
структуры")
print("   - t-SNE: лучше для
визуализации локальных
кластеров")
print("   - Autoencoder:
универсальный метод, может
учитывать нелинейности")
print("   - Для данного датасета
все методы показывают хорошее
разделение классов")

# Дополнительно: графики
обучения
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history_2d.history['loss'],
label='Training Loss')
plt.plot(history_2d.history['val_los
s'], label='Validation Loss')
plt.title('Autoencoder 2D -
Training History')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_3d.history['loss'],
label='Training Loss')
plt.plot(history_3d.history['val_los
s'], label='Validation Loss')
plt.title('Autoencoder 3D -
Training History')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Результаты программы:

Размерность данных: (8124, 22)

Размерность закодированных признаков: (8124, 2)

Первые 5 строк данных:

	cap-shape	cap-surface	cap-color	...	spore-print-color	population	habitat
0	x	s	n	...	k	s	u
1	x	s	y	...	n	n	g
2	b	s	w	...	n	n	m
3	x	y	w	...	k	s	u
4	x	s	g	...	n	a	g

[5 rows x 22 columns]

Целевая переменная:

	poisonous
0	p
1	e
2	e
3	p
4	e

=== ПРЕДОБРАБОТКА ДАННЫХ ===

Уникальные значения целевой переменной: [0 1]

Размерность после кодирования: (8124, 22)

Train size: (6499, 22), Test size: (1625, 22)

=== АВТОЭНКОДЕР С 2 НЕЙРОНАМИ ===

Epoch 1/50

204/204 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.6269 - val_loss: 0.3812

Epoch 50/50

204/204 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.1292 - val_loss: 0.1326

254/254 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 547us/step

=== АВТОЭНКОДЕР С 3 НЕЙРОНАМИ ===

Epoch 1/50

204/204 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.5885 - val_loss: 0.3656

Epoch 50/50

204/204 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0881 - val_loss: 0.0939

254/254 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 539us/step

=== МЕТОД PCA ===

Объясненная дисперсия PCA (2 компоненты): [0.2037041 0.12412968]

Суммарная объясненная дисперсия: 0.3278337789616575

Объясненная дисперсия PCA (3 компоненты): [0.2037041 0.12412968 0.11071335]

Суммарная объясненная дисперсия: 0.43854712625072334

Визуализация PCA (2D):

Визуализация PCA (3D):

=== СРАВНИТЕЛЬНЫЙ АНАЛИЗ ===

=== ВЫВОДЫ ===

1. АВТОЭНКОДЕР:
   - Способен извлекать нелинейные зависимости в данных
   - Показывает хорошее разделение классов в скрытом пространстве
   - Требует тщательной настройки архитектуры и параметров обучения

2. t-SNE:
   - Лучше всего показывает локальные кластеры и структуры
   - Чувствителен к параметру perplexity
   - Визуализация более интуитивно понятна для анализа кластеров

3. PCA:
   - Сохраняет глобальную структуру данных
   - Объяснимая дисперсия: 0.328
   - Быстрый и стабильный метод

4. СРАВНЕНИЕ:
   - PCA: лучше для сохранения глобальной структуры
   - t-SNE: лучше для визуализации локальных кластеров
   - Autoencoder: универсальный метод, может учитывать нелинейности
   - Для данного датасета все методы показывают хорошее разделение классов

Вывод: научился применять автоэнкодеры для осуществления визуализации данных и их анализа