Представление многомерных данных

# UMAP

[UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction — umap 0.5 documentation](https://umap-learn.readthedocs.io/en/latest/index.html)

## Инициализация данных на латентном пространстве

[**UMAP API Guide — umap 0.5 documentation**](https://umap-learn.readthedocs.io/en/latest/api.html#module-umap.umap_)

**init: string (optional, default ‘spectral’)**

How to initialize the low dimensional embedding. Options are:

* ‘spectral’: use a spectral embedding of the fuzzy 1-skeleton
* ‘random’: assign initial embedding positions at random.
* **‘pca’: use the first n\_components from PCA applied to the**input data.
* **‘tswspectral’: use a spectral embedding of the fuzzy**1-skeleton, using a truncated singular value decomposition to “warm” up the eigensolver. This is intended as an alternative to the ‘spectral’ method, if that takes an excessively long time to complete initialization (or fails to complete).
* A numpy array of initial embedding positions.

## Итоговые диапазоны значений осей эмбеддингов

Какое по размеру будет итоговое латентное пространство? какие диапазоны у осей?

UMAP (Uniform Manifold Approximation and Projection) выполняет нелинейное уменьшение размерности, и диапазон значений в выходных измерениях (компонентах embedding[:,0], embedding[:,1], и т.д.) зависит от природы данных и параметров алгоритма.

### **Почему диапазон значений такой?**

1. **Случайная инициализация**:  
   * UMAP изначально размещает точки в выходном пространстве (в вашем случае, двумерном) случайным образом. Инициализация по умолчанию — "spectral", которая использует спектральную разложение графа данных. Это разложение может давать координаты с разными диапазонами.
   * Если вы используете другую инициализацию (например, "random"), точки изначально будут распределены случайно, а затем оптимизироваться через градиентный спуск.
2. **Оптимизация в пространстве потерь**:  
   * Во время оптимизации UMAP стремится сохранить топологию исходных данных, но не накладывает ограничений на абсолютные значения координат в низкоразмерном пространстве.
   * Координаты могут сдвигаться, масштабироваться и растягиваться так, чтобы минимизировать функцию потерь (KL-дивергенцию или другую метрику расхождения).
3. **Сжатие и растяжение из-за метрик**:  
   * Используемая метрика расстояния (например, евклидова) также влияет на то, как UMAP интерпретирует расстояния между точками.
   * Если данные исходно сильно вытянуты или кластеризованы, диапазоны значений в embedding могут быть асимметричными.
4. **Параметры UMAP**:  
   * Гиперпараметры UMAP, такие как n\_neighbors, min\_dist, metric, могут существенно влиять на диапазон значений embedding. Например:
     + Небольшое значение min\_dist (например, 0.1) приводит к тому, что кластеры уплотняются сильнее, и диапазоны координат могут быть уже.
     + Большое значение min\_dist делает кластеры более "разбросанными".

### **Как интерпретировать выходные значения?**

1. **Относительные координаты**:  
   * UMAP фокусируется на сохранении относительных расстояний между точками, а не их абсолютных координат. Поэтому диапазон значений координат зависит от данных и оптимизации, а не имеет фиксированных границ.
2. **Значения не нормализованы**:

Если вам важно нормализовать диапазоны значений (например, чтобы координаты лежали в диапазоне [0, 1] или [-1, 1]), вы можете сделать это самостоятельно после применения UMAP:  
 from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

embedding\_scaled = scaler.fit\_transform(embedding)

### **Итоги:**

* Диапазон значений в embedding формируется в результате оптимизации без привязки к фиксированным границам.
* Если нужно, диапазон можно нормализовать вручную.
* Убедитесь, что настройки параметров n\_neighbors, min\_dist и metric соответствуют вашей задаче, так как они сильно влияют на форму и распределение embedding.

## Обратное преобразование

Такой результат на датасете

data, labels = load\_diabetes(return\_X\_y=True, as\_frame=False)

Original Data:

[[ 0.038 0.051 0.062 0.022 -0.044 -0.035 -0.043 -0.003 0.02 -0.018]

[-0.002 -0.045 -0.051 -0.026 -0.008 -0.019 0.074 -0.039 -0.068 -0.092]

[ 0.085 0.051 0.044 -0.006 -0.046 -0.034 -0.032 -0.003 0.003 -0.026]

[-0.089 -0.045 -0.012 -0.037 0.012 0.025 -0.036 0.034 0.023 -0.009]

[ 0.005 -0.045 -0.036 0.022 0.004 0.016 0.008 -0.003 -0.032 -0.047]]

Reconstructed Data:

[[ 0.045 0.053 0.022 0.049 -0.035 -0.014 -0.047 0.009 -0.01 -0. ]

[ 0.016 -0.046 -0.057 -0.031 -0.011 -0.016 0.06 -0.045 -0.058 -0.014]

[ 0.047 0.052 0.021 0.041 -0.034 -0.015 -0.038 0.001 -0.008 -0.015]

[-0.051 -0.046 -0.009 -0.033 -0.001 0.011 -0.021 0.01 0.009 -0.028]

[ 0.04 -0.047 0.012 0.016 -0.025 -0.025 0.02 -0.034 -0.021 -0.015]]

Такой результат на датасете Чмиля

tr\_data = pd.read\_excel("/content/drive/MyDrive/Colab Notebooks/диплом/данные/Cylinder\_05\_lhs4.xlsx", sheet\_name = "train")

val\_data = pd.read\_excel("/content/drive/MyDrive/Colab Notebooks/диплом/данные/Cylinder\_05\_lhs4.xlsx", sheet\_name = "valid")

ts\_data = pd.read\_excel("/content/drive/MyDrive/Colab Notebooks/диплом/данные/Cylinder\_05\_lhs4.xlsx", sheet\_name = "test")

Original Data (first row):

[[ 2.87 8.662 1.151 1.718]

[ 3.924 10.164 0.668 1.317]

[ 4.417 11.776 2.468 1.738]

[ 2.694 14.571 1.673 1.49 ]

[ 3.647 18.138 1.95 1.37 ]]

Reconstructed Data (first row):

[[ 2.691 9.119 1.448 1.717]

[ 3.934 10.193 0.757 1.314]

[ 4.579 10.325 2.642 1.684]

[ 2.661 14.827 1.707 1.475]

[ 3.612 18.111 1.978 1.348]]

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Код

### umap.plot

Легче через матплотлиб, тк umap.plot.points параметры хз как настраивать

import matplotlib.pyplot as plt

plt.figure(figsize = (6, 7))

plt.scatter(embeddings\_my[:,0], embeddings\_my[:,1], c = y[val], s = 5)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

!pip install umap-learn[plot]

import umap

import umap.plot

umap0 = umap.UMAP(random\_state=42)

mapper = umap0.fit(x[val])

umap.plot.points(mapper, labels=y[val].flatten().numpy())

### Правильное обратное преобразование

В целом обратное преобразование, даёт не очень хорошие результаты для восстановленных данных, однако если снова преобразовать их в латентное пространство, то всё более менее нормально выглядит.

import numpy as np

from sklearn.datasets import load\_diabetes

from sklearn.preprocessing import StandardScaler

import umap

# Загрузка данных

data, labels = load\_diabetes(return\_X\_y=True, as\_frame=False)

# Нормализация данных для лучшей работы UMAP

scaler = StandardScaler()

data\_scaled = scaler.fit\_transform(data)

# Создание и настройка UMAP

reducer = umap.UMAP(n\_neighbors=15, min\_dist=0.1, n\_components=2, random\_state=42)

# Применение UMAP-преобразования

embedding = reducer.fit\_transform(data\_scaled)

plt.scatter(embedding[:,0],embedding[:,1], c= labels)

# Обратное преобразование (восстановление данных)

data\_reconstructed = reducer.inverse\_transform(embedding)

# Обратное масштабирование

original\_space\_reconstructed = scaler.inverse\_transform(data\_reconstructed)

print("Original Data (first row):", np.round(data[0],4))

print("Reconstructed Data (first row):", np.round(original\_space\_reconstructed[0], 4))

# Проверка ошибки восстановления

reconstruction\_error = np.mean(np.sqrt((data - original\_space\_reconstructed)\*\*2))

print(f"Mean Squared Reconstruction Error: {reconstruction\_error:.6f}")

original\_space\_reconstructed\_scale = scaler.transform(original\_space\_reconstructed)

embedding2 = reducer.transform(original\_space\_reconstructed\_scale)

plt.scatter(embedding2[:,0],embedding2[:,1], c = labels)

Plot