ДИСЦИПЛИНА **Интеллектуальные системы и технологии**

(полное наименование дисциплины без сокращений)

ИНСТИТУТ **информационных технологий**

КАФЕДРА **корпоративных информационных систем**

полное наименование кафедры

ВИД УЧЕБНОГО МАТЕРИАЛА **Лекция**

(в соответствии с пп.1-11)

ПРЕПОДАВАТЕЛЬ **Демидова Лилия Анатольевна**

(фамилия, имя, отчество)

СЕМЕСТР **1 семестр (осенний), 2024 – 2025 учебный год**

(семестр обучения, учебный год)

ЛЕКЦИЯ 3
АЛГОРИТМЫ НЕЛИНЕЙНОГО СНИЖЕНИЯ РАЗМЕРНОСТИ

Визуализация данных – представление набора данных в двух- или в трехмерном пространстве.

Для представления многомерных данных предварительно необходимо решить задачу снижения размерности данных.

Под снижением размерности данных понимается процесс преобразования в форму, наиболее удобную для анализа и интерпретации.

Обычно снижение размерности данных достигается за счёт уменьшения их объема, сокращения числа используемых признаков и разнообразия их значений.

Кроме визуализации снижение размерности данных может преследовать множество целей.

Цели

1. Сокращение вычислительных затрат при обработке данных.
2. Борьба с переобучением. Чем меньше число признаков, тем меньше требуется объектов для уверенного восстановления скрытых зависимостей в данных и тем выше качество восстановления подобных зависимостей.
3. Сжатие данных для более эффективного хранения информации.
4. Извлечение новых признаков. Новые признаки, полученные в результате преобразования, могут оказывать значимый вклад при последующем решении задач.

t-SNE

Пусть на обработку алгоритму снижения размерности поступает выборка $X = \{x_1, \dots, x_n\}$.

t-SNE (t-distributed stochastic neighbor embedding) – алгоритм нелинейного снижения размерности и визуализации многомерных переменных.

t-SNE может свернуть сотни измерений к меньшему числу, сохраняя при этом важные отношения между данными: чем ближе объекты располагаются в исходном пространстве, тем меньше расстояние между этими объектами в пространстве низкой размерности.

Описание алгоритма

1. t-SNE создаёт распределение вероятностей по парам объектов x_i и x_j высокой размерности таким образом, что близкие объекты будут иметь большую вероятность, а вероятность для непохожих объектов будет мала.

t-SNE преобразует многомерное евклидово расстояние между точками в условные вероятности, отражающие сходство точек.

$$p_{j|i} = \frac{exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)},$$

где x_i, x_j, x_k – точки в евклидовом пространстве, σ – отклонение (выбирается с помощью оценки перплексии так, чтобы точки в областях с большей плотностью имели меньшую дисперсию).

2. t-SNE определяет похожее распределение вероятностей по точкам в пространстве y_i и y_j малой размерности:

$$q_{j|i} = \frac{exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} exp(-\|y_i - y_k\|^2)},$$

где y_i, y_j, y_k – точки отображения.

Если точки y_i и y_j малой размерности корректно моделируют сходство между исходными точками высокой размерности x_i и x_j, то соответствующие условные вероятности будут эквивалентны $p_{j|i} \equiv q_{j|i}$.

3. t-SNE минимизирует расстояние Кульбака – Лейблера между двумя распределениями с учётом положения точек. При этом минимизируется сумма таких расстояний для всех точек отображения при помощи *классического градиентного спуска* и функции потерь:

$$C = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}},$$

$$\frac{\partial C}{\partial y_i} = 2\sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j),$$

где y_i, y_j – точки отображения, $p_{j|i}$ и $p_{i|j}$ – условная вероятность схожести точек большой размерности, $q_{j|i}$ и $q_{i|j}$ – условная вероятность схожести точек отображения.

Подробная информация о t-SNE:

van der Maaten L.J.P., Hinton G.E. Visualizing Data Using t-SNE // Journal of Machine Learning Research. – 2008. – Ноябрь (т. 9).

t-SNE в Python

https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

***class* `sklearn.manifold.TSNE`*(n_components=2, perplexity=30.0, early_exaggeration=12.0, learning_rate=200.0, n_iter=1000, n_iter_without_progress=300, min_grad_norm=1e-07, metric='euclidean', init='random', verbose=0, random_state=None, method='barnes_hut', angle=0.5, n_jobs=None*)**

Основными параметрами t-SNE-алгоритма являются:

- перплексия (по умолчанию – 30);
- параметр раннего преувеличения (по умолчанию – 12);
- скорость обучения (по умолчанию – 200);
- число итераций оптимизации (по умолчанию – 1000);
- максимальное число итераций без прогресса перед прерыванием оптимизации (по умолчанию – 300);
- пороговое значение нормы градиента (по умолчанию – 1E – 7);
- метрика (по умолчанию – евклидова);
- метод вычисления градиента (по умолчанию – 'barnes_hut');
- пороговый угол (по умолчанию – 0.5);

- число параллельных заданий для поиска соседей (по умолчанию – 1);
- размерность конечного метрического пространства (по умолчанию – 2).

Перплексию (perplexity) соотносят с числом ближайших соседей, которое используется в других алгоритмах машинного обучения.

Параметр раннего преувеличения (early exaggeration) управляет тем, насколько плотные кластеры из исходного метрического пространства находятся во встроенном метрическом пространстве и каково будет расстояние между ними. Большие значения этого показателя позволяют установить большее расстояние между кластерами во встроенном метрическом пространстве.

Число итераций определяет максимальное число итераций t-SNE-алгоритма.

Максимальное число итераций без прогресса перед прерыванием оптимизации используется после 250 начальных итераций с ранним преувеличением.

Если норма градиента ниже порогового значения, оптимизация будет остановлена.

Метрика определяет, каким образом будут вычисляться расстояния в исходном метрическом пространстве.

Метод вычисления градиента по умолчанию выбирается как метод, использующий приближение Барнса-Хата, работающее за время O (n·log(n)). Время работы точного метода вычисления градиента оценивается как $O(n^2)$, где n – число объектов (при этом точный метод не может масштабироваться в случае больших наборов данных (размером около 1 млн.).

Пороговый угол используется только, если метод вычисления градиента – 'barnes_hut'. Он позволяет обеспечить компромисс между скоростью и точностью этого метода и реализует оценку углового размера удаленного узла, измеренного от объекта. Если этот угловой размер меньше

порогового угла, он используется как итоговый узел всех объектов, содержащихся в нем. Пороговый угол, меньший чем 0.2, быстро увеличивает время вычислений, а пороговый угол, больший чем 0.8, быстро увеличивает погрешность вычислений.

При увеличении числа параллельных заданий для поиска соседей можно осуществить распараллеливание t-SNE-алгоритма.

Пример кода

```
import numpy as np
from sklearn.manifold import TSNE
X = np.array([[0, 0, 0], [0, 1, 1], [1, 0, 1], [1, 1, 1]])
X_embedded = TSNE(n_components=2).fit_transform(X)
X_embedded.shape
```

```
(4, 2)
```

UMAP

Uniform Approximation and Projection (UMAP, равномерная аппроксимация и проекция) – алгоритм машинного обучения, выполняющий нелинейное снижение размерности.

UMAP разработан Лилендом Макиннесом совместно с коллегами из Таттского института.

Цель – разработка алгоритм, похожего на t-SNE, но с более сильным математическим обоснованием.

При снижении размерности UMAP сначала выполняет построение взвешенного графа, соединяя ребрами только те объекты, которые являются ближайшими соседями.

Множество из ребер графа – это нечёткое множество с функцией принадлежности, она определяется как вероятность существования ребра между двумя вершинами.

Затем алгоритм создает граф в низкоразмерном пространстве и приближает его к исходному, минимизируя сумму дивергенций Кульбака-Лейблера для каждого ребра из множества.

Описание алгоритма

1. UMAP сначала выполняет построение взвешенного графа, соединяя ребрами только те объекты, которые являются ближайшими соседями.

UMAP рассчитывает расстояние ρ_i между объектами по заданной метрике и для каждого объекта x_i определяет список из его k ближайших соседей $Y = \{y_{1,} \dots, y_k\}$.

Метрика определяется формулой для вычисления расстояния между объектами.

Метрика:

$$\rho = d(x_i, y_j),$$

где $d(x_i, y_j)$ – правило вычисления расстояния между объектами ($i = \overline{1,n}$, $j = \overline{1,k}$).

Так как UMAP строит взвешенный ориентированный граф, то между вершинами могут существовать два ребра с разными весами. Вес ребра интерпретируется как вероятность существования данного ребра от объекта x_i к объекту x_j.

Исходя из этого, ребра между двумя вершинами объединяются в одно с весом, равным вероятности существования хотя бы одного ребра:

$$\omega(x_i, y_j) = \omega(x_i \to y_j) + \omega(x_i \to y_j) - \omega(x_i \to y_j)\omega(x_i \to y_j),$$

$$\omega(x_i \to y_j) = exp\left(-\frac{d(x_i, y_j) - \rho_i}{\sigma_i}\right),$$

где x_i, y_j – объекты в вершинах графа ($i = \overline{1,n}$, $j = \overline{1,k}$), ρ_i – расстояние до ближайшего соседа , σ_i – величина, нормирующая сумму весов.

Величина σ_i задается уравнением для каждого x_i:

$$\sum_{y \in Y} \exp\left(-\frac{d(x_i, y_j) - \rho_i}{\sigma_i}\right) = \log_2 k\,,$$

где k – число ближайших соседей.

2. UMAP создает новый граф в низкоразмерном пространстве и приближает множество его ребер к исходному.

Для этого UMAP минимизирует сумму дивергенций Кульбака-Лейблера для каждого ребра e из исходного и нового нечетких множеств:

$$\sum_{e \in E} \omega_h(e) \log \frac{\omega_h(e)}{\omega_l(e)} + (1-)) \log(\frac{1 - \omega_h(e)}{1 - \omega_l(e)}) \to \min_{\omega_l},$$

где $\omega_h(e)$ – функция принадлежности нечеткого множества из ребёр в высокоразмерном пространстве, $\omega_l(e)$ – функция принадлежности нечеткого множества из ребёр в низкоразмерном пространстве.

UMAP решает задачу минимизации с помощью ***стохастического градиентного спуска***. Полученное множество из ребер определяет новое расположение объектов и, соответственно, низкоразмерное отображение исходного пространства.

UMAP-алгоритм основан на трех предположениях о данных: данные равномерно распределены на римановом многообразии, представляющем собой вещественное дифференцируемое многообразие, в котором каждое касательное пространство снабжено скалярным произведением (метрическим тензором), меняющимся от точки к точке гладким образом; риманова метрика локально постоянна (или может быть аппроксимирована как таковая); римановo многообразие локально связно.

С учетом этих предположений моделируется многообразие с нечеткой топологической структурой. При этом искомое вложение определяется посредством поиска низкоразмерной проекции данных, которые имеют наиболее близкую эквивалентную нечеткую топологическую структуру.

Подробная информация об UMAP:

Leland McInnes, John Healy, James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction (англ.) // arXiv. – 2018. – 7 December.

UMAP в Python

umap

Основными параметрами UMAP-алгоритма являются:

- число ближайших соседей (по умолчанию – 15);
- минимальное расстояние, на котором могут находиться точки в новом метрическом пространстве (по умолчанию – 0.1);
- метрика (по умолчанию – евклидова);
- размерность конечного метрического пространства (по умолчанию – 2).

Варьируя число ближайших соседей, можно определить, что важнее сохранить в новом метрическом пространстве: глобальную или локальную структуру данных.

Малые значения этого параметра означают, что при оценке исходного метрического пространства UMAP-алгоритм ограничивается малой окрестностью вокруг каждого объекта, пытаясь сохранить локальную структуру данных (может быть, даже в ущерб общей картине). Большие значения *n* этого параметра означают, что при оценке исходного метрического пространства UMAP-алгоритм учитывает точки в большей окрестности, пытаясь сохранить глобальную структуру данных (может быть, упуская детали).

Варьируя величину минимального расстояния, можно определить, в каком виде должны быть представлены данные в новом метрическом пространстве. Малые значения этого параметра позволяют оценить, на какие

кластеры разделяются ваши данные. Большие значения этого параметра позволяют увидеть структуру данных, как единое целое.

Метрика определяет, каким образом будут вычисляться расстояния в исходном метрическом пространстве.

При определении параметра, отвечающего за размерность конечного метрического пространства, можно определить размерность уменьшенного пространства, в которое будут встраиваться исходные данные. В отличие от некоторых других алгоритмов визуализации, таких как, например, t-SNE-алгоритм, UMAP-алгоритм, хорошо масштабируется при встраивании, поэтому его можно использовать не только для визуализации в 2-х или 3-х мерном пространствах.

Считается, что UMAP-алгоритм похож на t-SNE, но имеет более сильное математическое обоснование.

ПРИМЕРЫ

Пример 1

Таблица 1. Характеристика наборов данных

Название	Число классов	Число признаков	Число объектов	Краткое описание
Digits	10	64	1797	Классификация рукописных цифр в картинке 8x8
Bank-additional	2	20	41188	Определить, будет ли клиент подписывать срочный депозит
Connect	3	42	67557	Определить, выиграет ли первый игрок в следующем ходе
Covertype	7	56	116203	Определение типа лесного покрова только по картографическим переменным
Localization Data for Person Activity	11	6	164860	Определение позы человека по датчикам

Каждый из этих наборов данных был визуализирован обоими алгоритмами. Также было измерено время в секундах, за которое алгоритм снижал размерность исходных данных (таблица 2). Кроме того, были рассмотрены возможности методов к параллельным вычислениям: UMAP в реализации на Python не имеет возможности к распараллеливанию вычислений в отличии от t-SNE. Снижение размерности при различном числе потоков было выполнено только для t-SNE.

Таблица 2. Время, затраченное на снижение размерности (в секундах)

	Digits	Bank-additional	Connect	Covertype	Localization Data for Person Activity
UMAP	5.890	44.969	95.906	4073.656	220.422
t-SNE	1 поток				
	5.844	227.000	562.875	11057.203	698.172
	2 потока				
	5.827	197.453	432.250	8416.781	693.641
	3 потока				
	5.782	186.172	405.313	7490.297	678.217
	4 потока				
	5.750	186.625	380.468	6938.961	686.235

Выводы

- при визуализации наборов данных с малым числом объектов время работы алгоритмов слабо отличается друг от друга;
- число признаков и число уникальных значений в них сильно влияет на время работы алгоритмов.

UMAP работает быстрее не зависимо от числа потоков. Следовательно, в ситуациях, когда скорость работы алгоритма необходимо минимизировать, лучше использовать UMAP.

Пример 2

Решение задачи кластеризации всегда предполагает выбор алгоритма кластеризации: от выбора алгоритма зависит качество полученного разбиения объектов на кластеры, оцениваемое с применением того или иного

показателя качества кластеризации, например, с применением индекса кластерного силуэта, который должен быть максимизирован.

При этом необходимо выбрать не только алгоритм кластеризации, попытавшись угадать структуру как самого набора данных в целом, так и каждого кластера в отдельности, но и предположить, какое число кластеров содержится в анализируемом наборе данных.

Каждый алгоритм кластеризации реализует конкретные жесткие математические принципы разбиения объектов на кластеры: даже при работе с одним алгоритмом кластеризации существенную роль могут сыграть используемые в нем метрика расстояния и метод группирования объектов в кластеры.

Например, для кластеров гиперэллиптической формы может оказаться целесообразным использование других метрик, чем в случае кластеров гиперсферической формы.

Для получения адекватных результатов кластеризации необходимо исследовать результаты, полученные с применением различных алгоритмов кластеризации при разном числе кластеров с реализацией в этих алгоритмах различных метрик расстояний и других настраиваемых параметров. Такой перебор алгоритмов кластеризации связан с существенными временными затратами.

Сокращение перечня исследуемых алгоритмов кластеризации позволило бы сократить временные затраты на получение приемлемого решения, однако ввиду высокой размерности анализируемых наборов данных их визуализация в исходном метрическом пространстве не представляется возможной. При этом очень остро при решении задачи кластеризации ощущается проблема плохой отделимости кластеров друг от друга: возможно, данные таковы, что попытка их кластеризации вообще не имеет смысла.

Пример 3

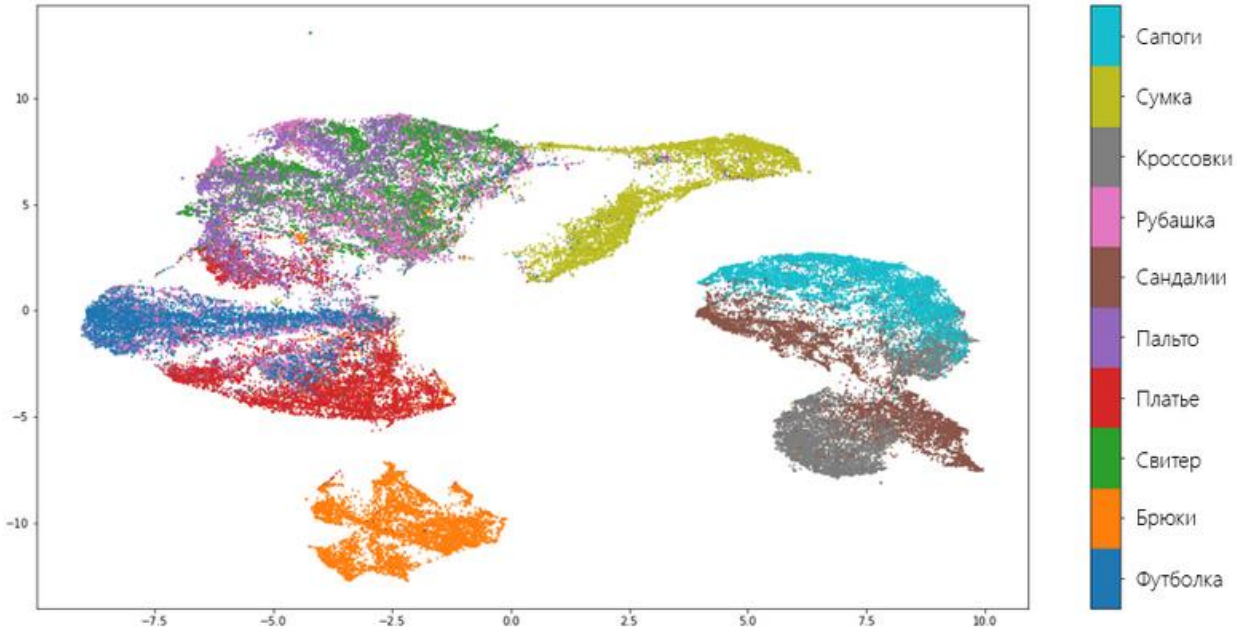
Датасет Fashion MNIST, который включает в себя 70000 черно-белых изображений различной одежды по 10 классам: футболки, брюки, свитеры, платья, кроссовки и т.д.

Каждая картинка имеет размер 28x28 пикселей (всего – 784 пикселя).

```
import pandas as pd
import umap

fmnist = pd.read_csv('fashion-mnist.csv') # считываем данные

embedding = umap.UMAP(n_neighbors=5).fit_transform(fmnist.drop('label', axis=1)) # преобразовываем
```



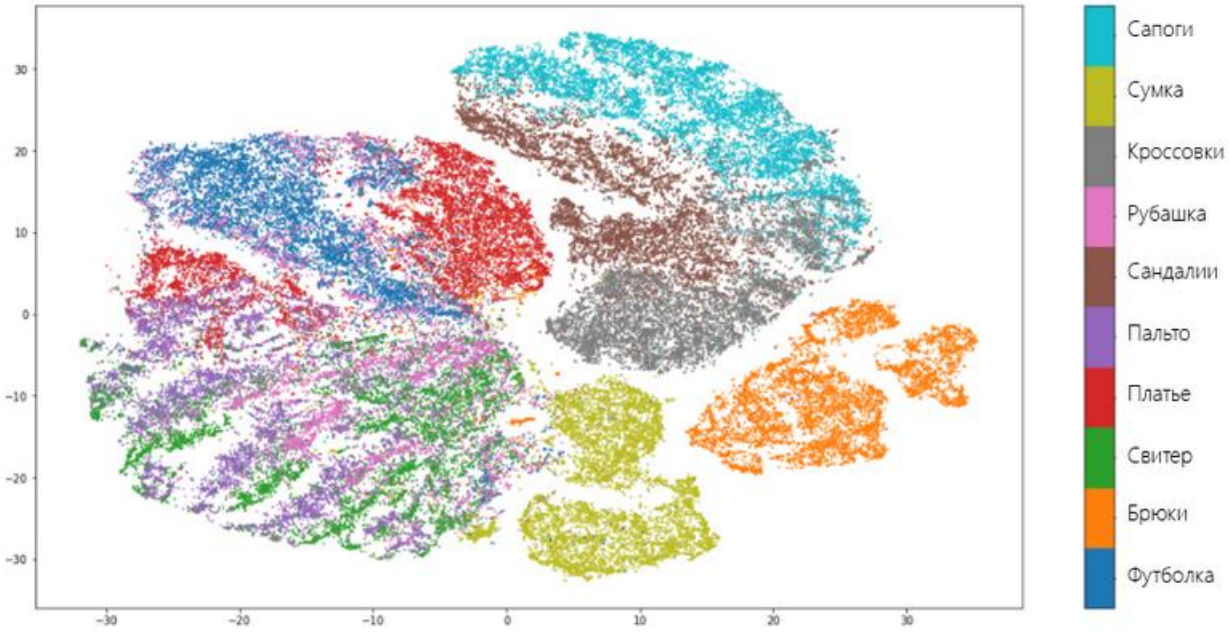
UMAP-алгоритм справился с задачей хорошо и «не растерял» большинство ценной информации, которая отличает один вид одежды от другого.

UMAP отделил обувь, одежду для туловища и брюки друг от друга, понимая, что это совершенно разные вещи.

Однако, решил, что рубашка и свитер – это одно и то же.

```
from MulticoreTSNE import MulticoreTSNE as TSNE

tsne = TSNE()
embedding_tsne = tsne.fit_transform(fmnist.drop('label', axis = 1))
```



t-SNE показывает схожие с UMAP результаты и допускает те же ошибки.

Однако, в отличии от UMAP, t-SNE не так очевидно объединяет виды одежды в отдельные группы: брюки, вещи для туловища и для ног находятся близко друг к другу.

Резюме: в целом можно сказать, что оба алгоритма одинаково хорошо справились с задачей.

Однако:

Имеется существенное превосходство UMAP над t-SNE в скорости обучения.

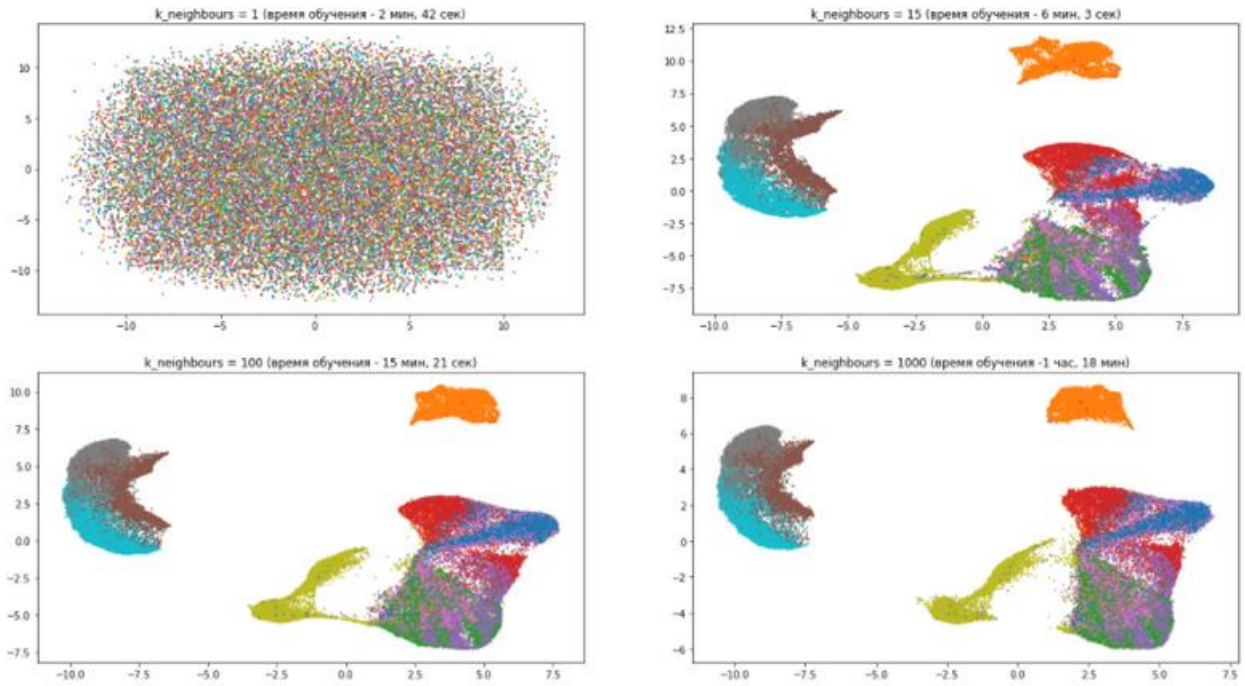
На сервере с 4 ядрами Intel Xeon E5- 2690v3, 2,6 Гц и 16 Гб оперативной памяти на наборе данных размера 70000x784 UMAP обучился за 4 минуты и 21 секунду, в то время как t-SNE потребовалось на это почти в 5 раз больше времени: 20 минут, 14 секунд.

Вывод: UMAP значительно более вычислительно эффективен, что дает ему огромное преимущество перед другими алгоритмами, в том числе и перед t-SNE.

Примеры с UMAP

Число соседей (*n_neighbors*).

Варьируя этот параметр, можно выбирать, что важнее сохранить в новом пространственном представлении данных: глобальную или локальную структуру данных. Маленькие значения параметра означают, что, пытаясь оценить пространство, в котором распределены данные, алгоритм ограничивается малой окрестностью вокруг каждой точки, то есть пытается уловить локальную структуру данных (возможно, в ущерб общей картине). Большие значения *n_neighbors* заставляют UMAP учитывать точки в большей окрестности, сохраняя глобальную структуру данных, но упуская детали.



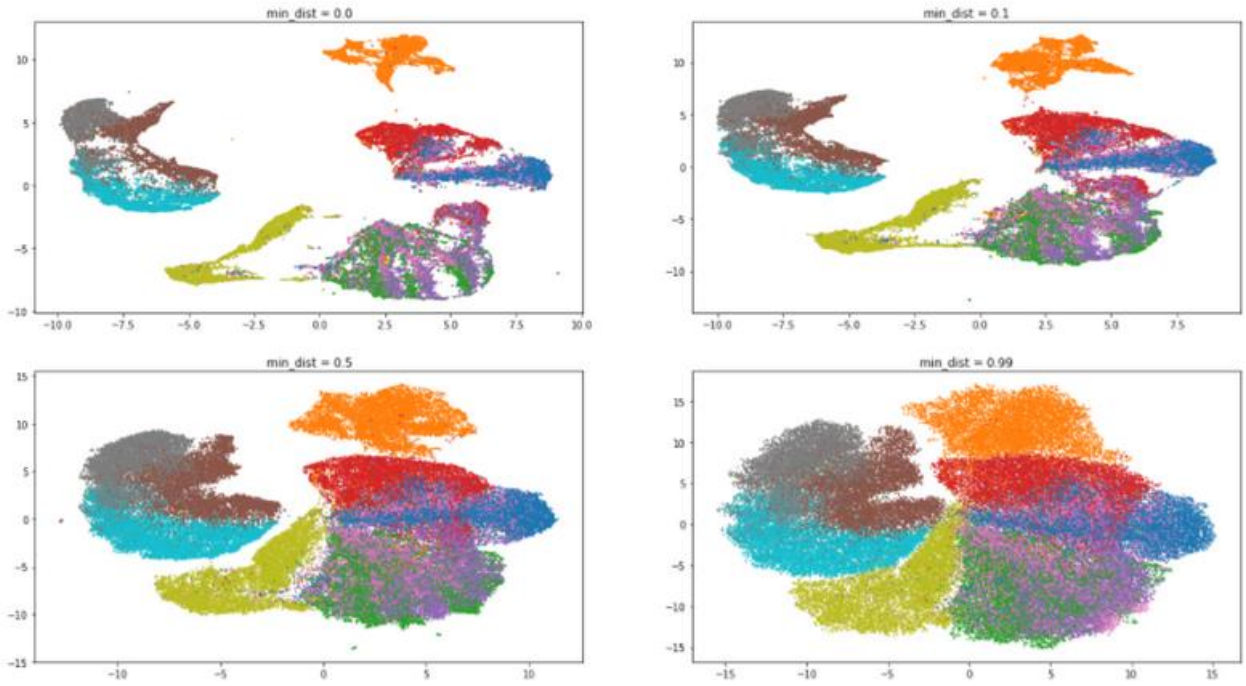
Выводы по рисунку

- С числом соседей, равным единице, UMAP работает плохо, так как слишком сильно фокусируется на деталях и не может уловить общую структуру данных.
- С ростом числа соседей UMAP все меньше внимания уделяет различиям между разными видами одежды, группируя похожие и смешивая их между собой. В то же время абсолютно разные предметы гардероба становятся все дальше друг от друга. Общая картина становится ясней, а детали размываются.
- Параметр *n_neighbors* существенно влияет на время обучения. поэтому нужно аккуратно подходить к его подбору.

Минимальное расстояние – *min_dist*.

Этот параметр определяет минимальное расстояние, на котором могут находиться точки в новом пространстве. Низкие значения стоит применять в случае, если нас интересует, на какие кластеры разделяются данные, а высокие – если важнее посмотреть на структуру данных, как на единое целое.

Оценим влияние параметра *min_dist* при значение *n_neighbors* =5.



Выводы по рисунку

Увеличение значения параметра приводит к меньшей степени кластеризации, данные собираются в кучу и различия между ними стираются.

При минимальном значении *min_dist*, равном 0, алгоритм пытается найти различия внутри кластеров и разделить их на еще более мелкие группы.

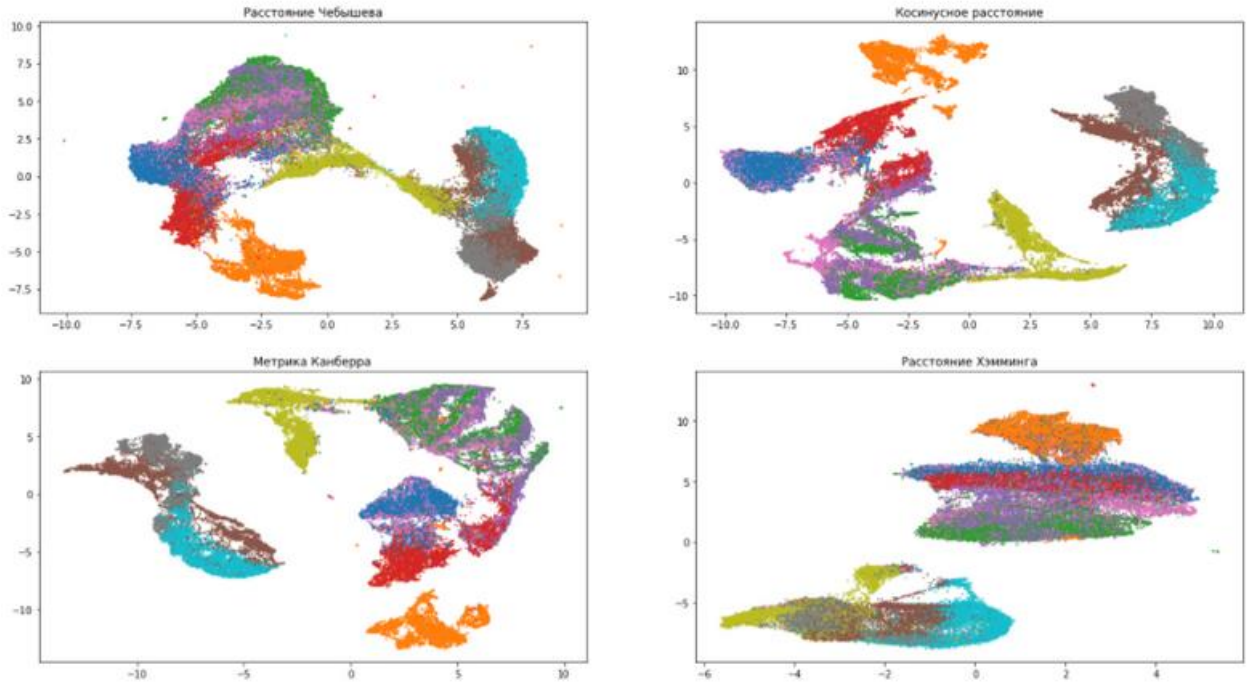
Метрика расстояния – *metric*

Параметр *metric* определяет, каким образом будут рассчитаны расстояния в пространстве исходных данных.

UMAP поддерживает всевозможные расстояния, от Минковского до Хэмминга.

Выбор метрики зависит от того, как нужно интерпретировать данные, а также от их типа.

Например, работая с текстовой информацией, предпочтительно использовать косинусное расстояние (*metric='cosine'*).



Выводы по рисунку

Выбор меры расстояния очень сильно влияет на итоговый результат, поэтому, к ее выбору нужно подходить ответственно. Расстояние Евклида является наиболее часто используемым вариантом (стоит по умолчанию в UMAP).

Размерность конечного пространства (*n_components*)

Параметр определяет размерность итогового пространства.

Если нужно визуализировать данные, то следует выбирать 2 или 3.

Если нужно использовать преобразованные вектора в качестве характеристик объектов для задач машинного обучения, то можно определить большую размерность пространства.

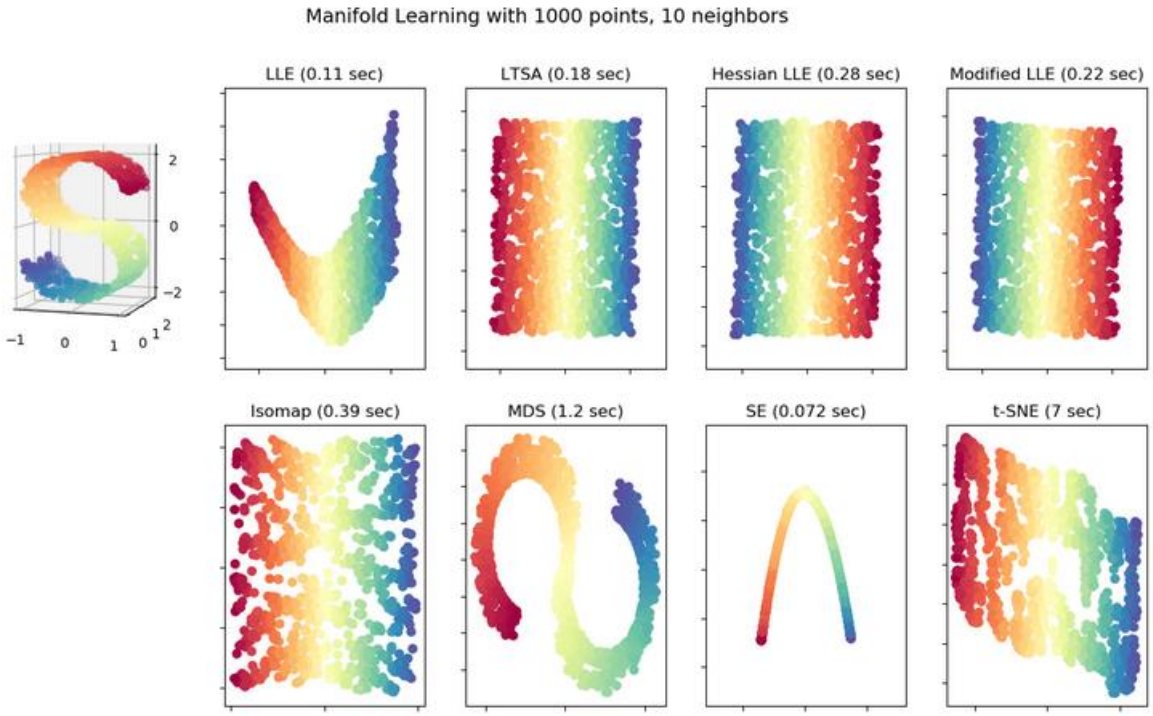
Пример 4

UMAP смог визуализировать набор данных Google News, состоящий из 3 миллионов векторов-слов за 200 минут, тогда как t-SNE потратил на визуализацию несколько дней.

Рисунок построен с помощью библиотеки для визуализации больших данных datashader.

Сравнение алгоритмов снижения размерности

https://scikit-learn.org/stable/auto_examples/manifold/plot_compare_methods.html#sphx-glr-auto-examples-manifold-plot-compare-methods-py

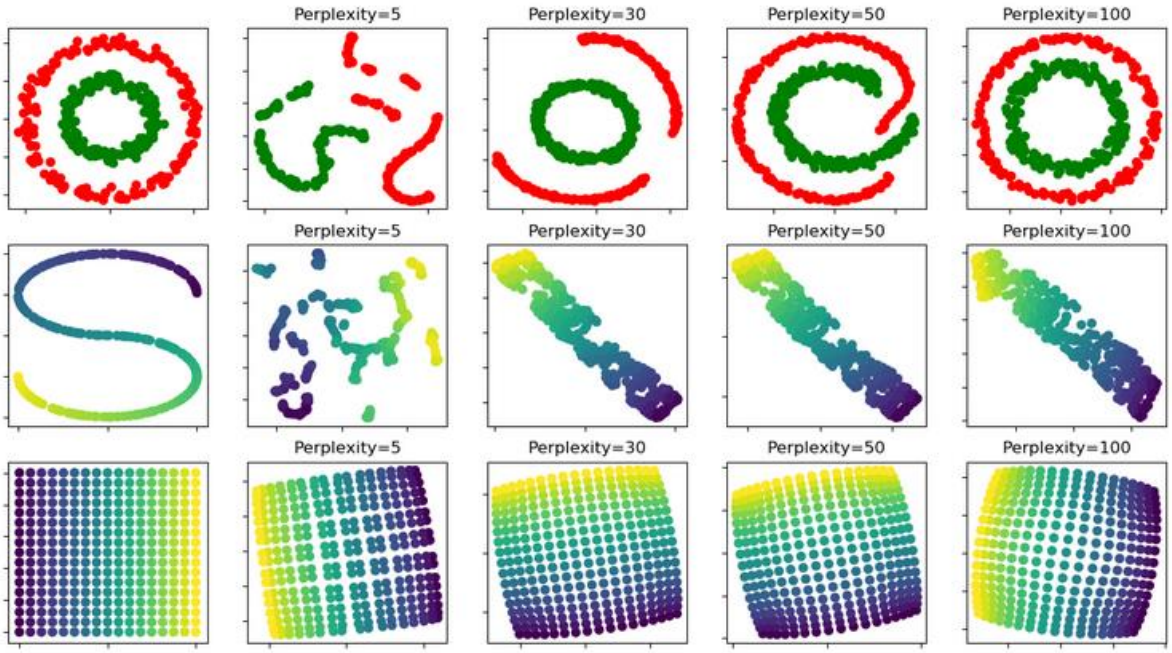


Время работы:

```
LLE: 0.11 sec
LTSA: 0.18 sec
Hessian LLE: 0.28 sec
Modified LLE: 0.22 sec
Isomap: 0.39 sec
MDS: 1.2 sec
SE: 0.072 sec
t-SNE: 7 sec
```

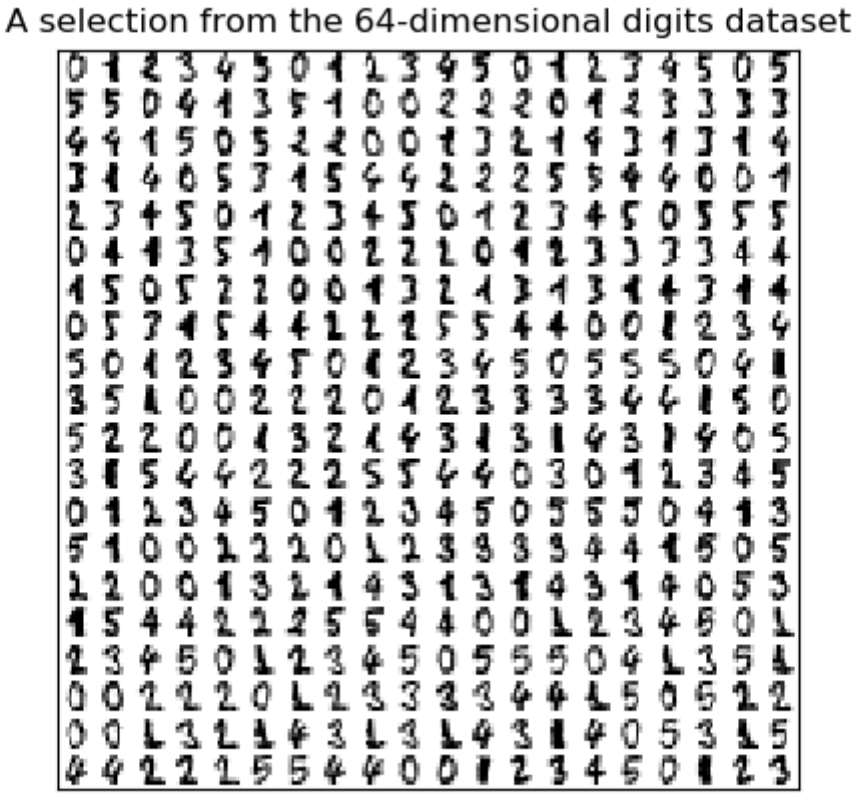
t-SNE: The effect of various perplexity values on the shape

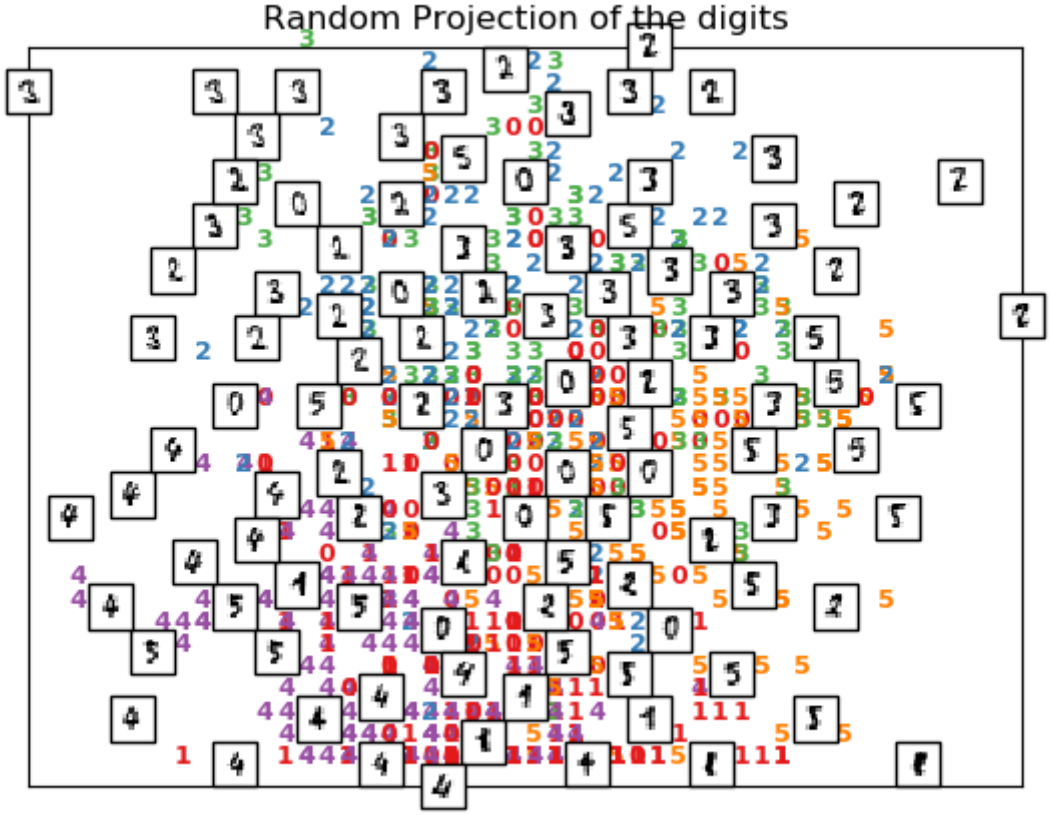
https://scikit-learn.org/stable/auto_examples/manifold/plot_t_sne_perplexity.html#sphx-glr-auto-examples-manifold-plot-t-sne-perplexity-py

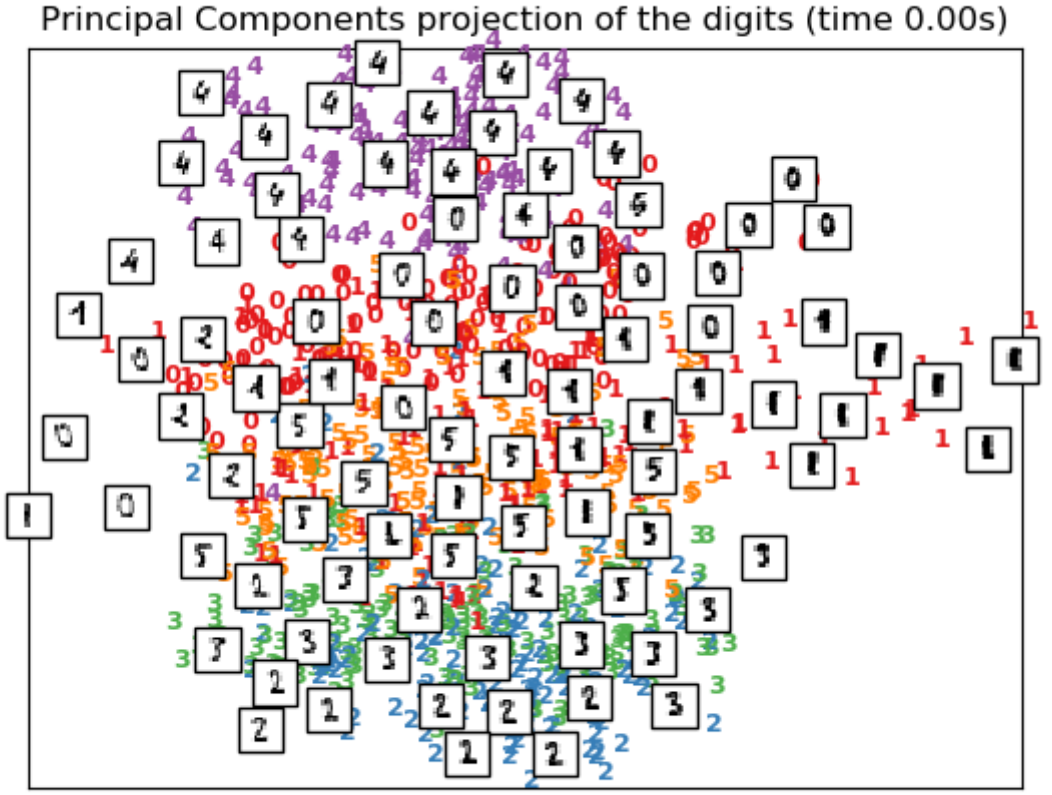


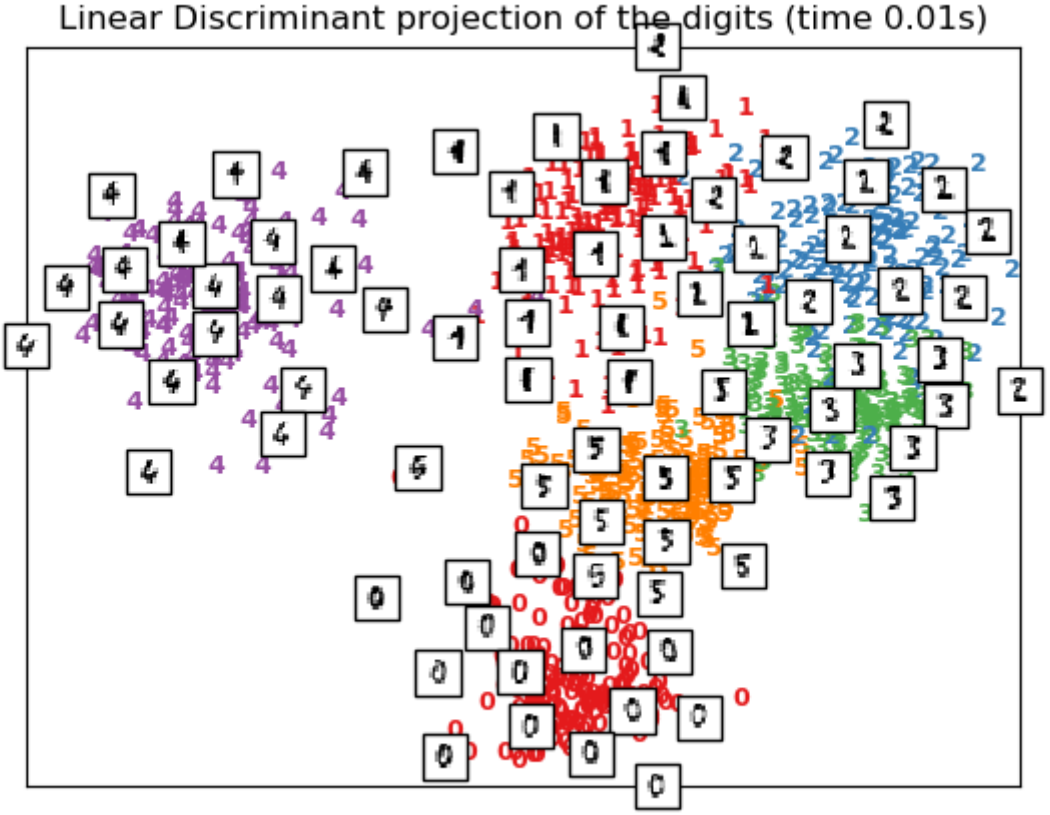
Manifold learning on handwritten digits: Locally Linear Embedding, Isomap…

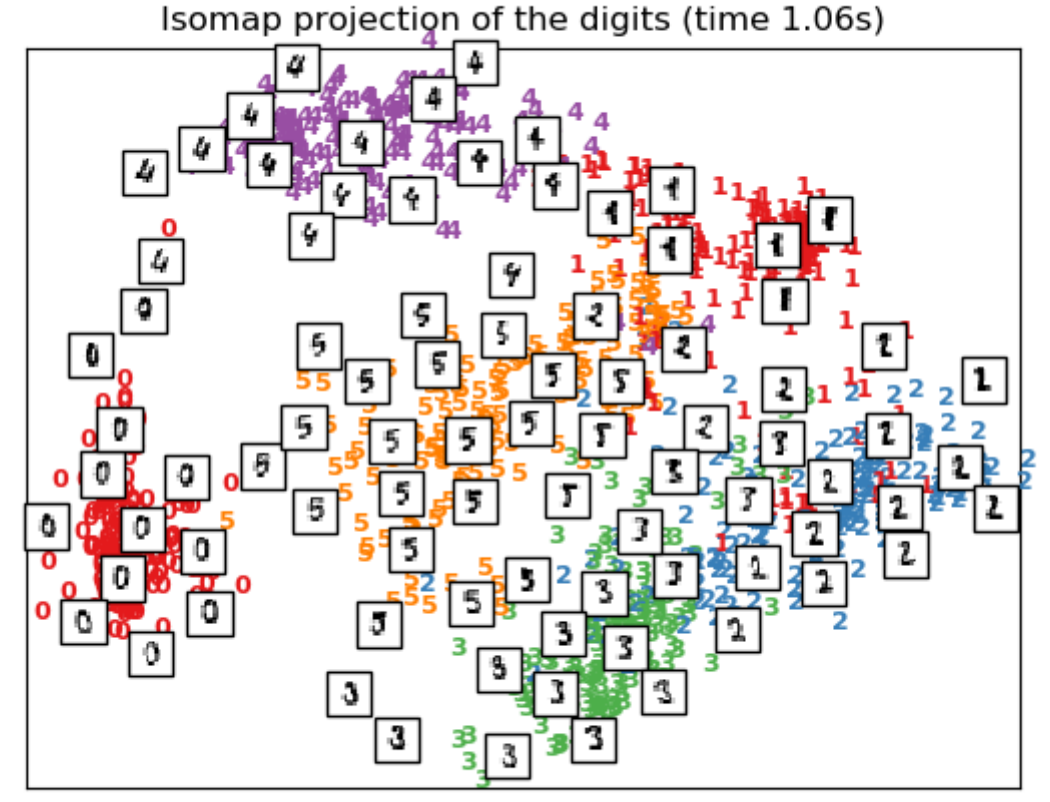
https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html#sphx-glr-auto-examples-manifold-plot-lle-digits-py

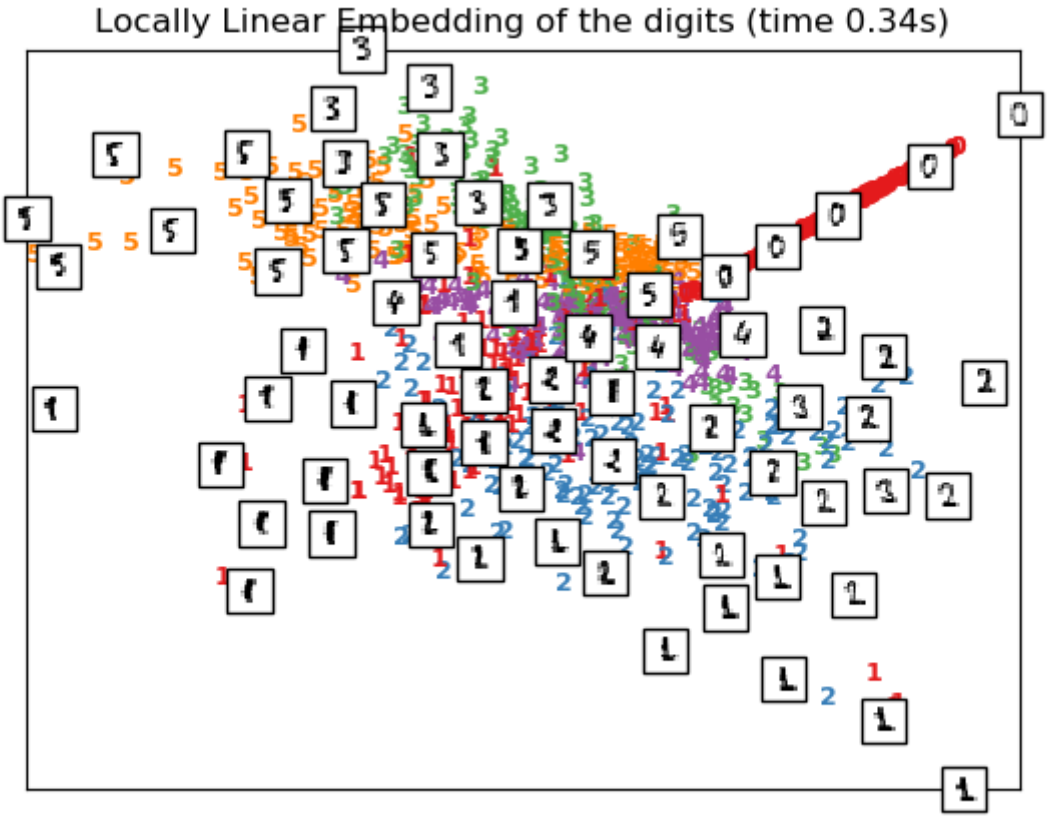


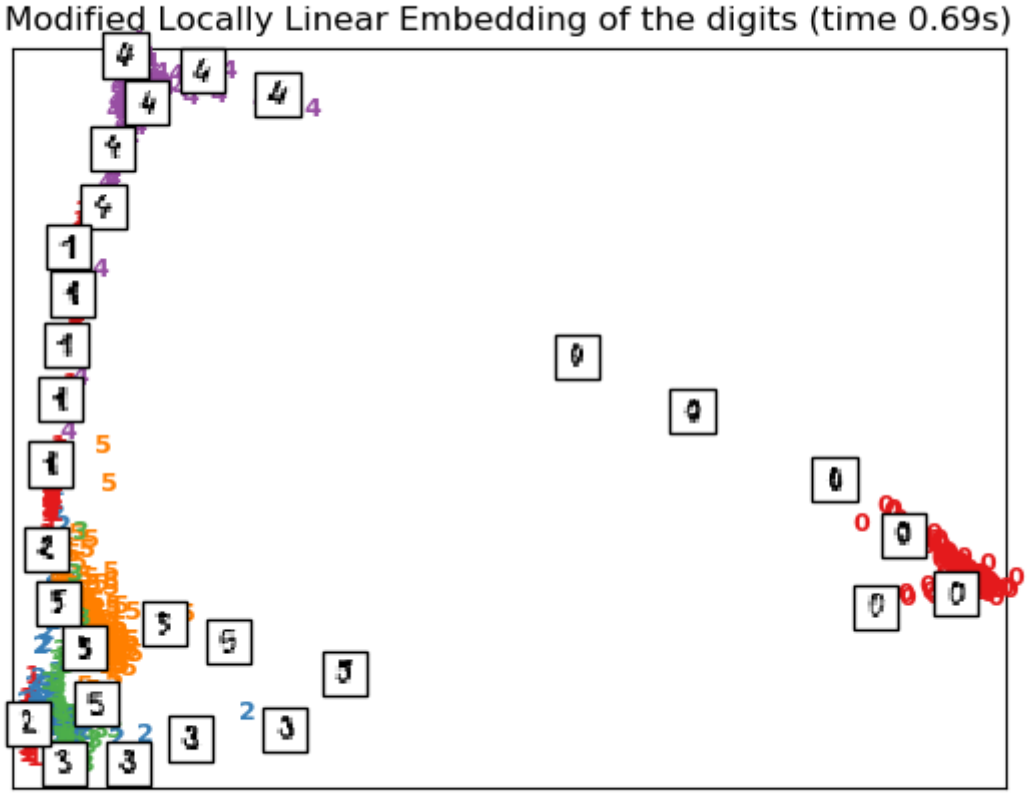

Random Projection of the digits

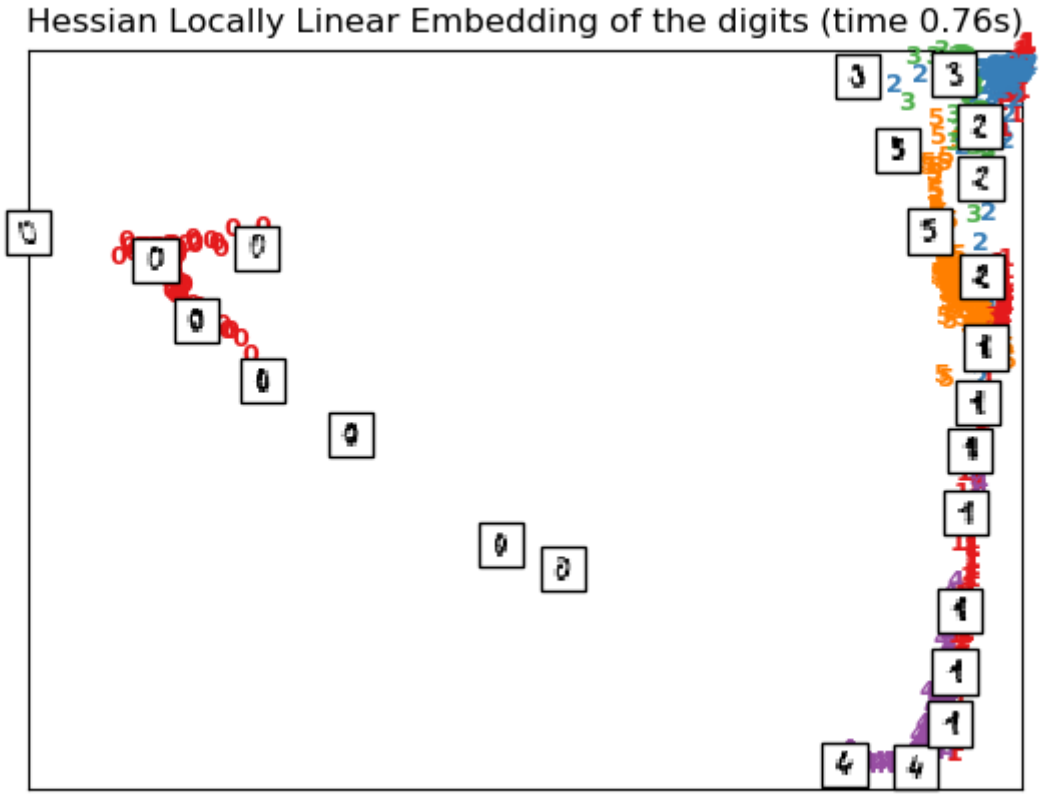

Principal Components projection of the digits (time 0.00s)

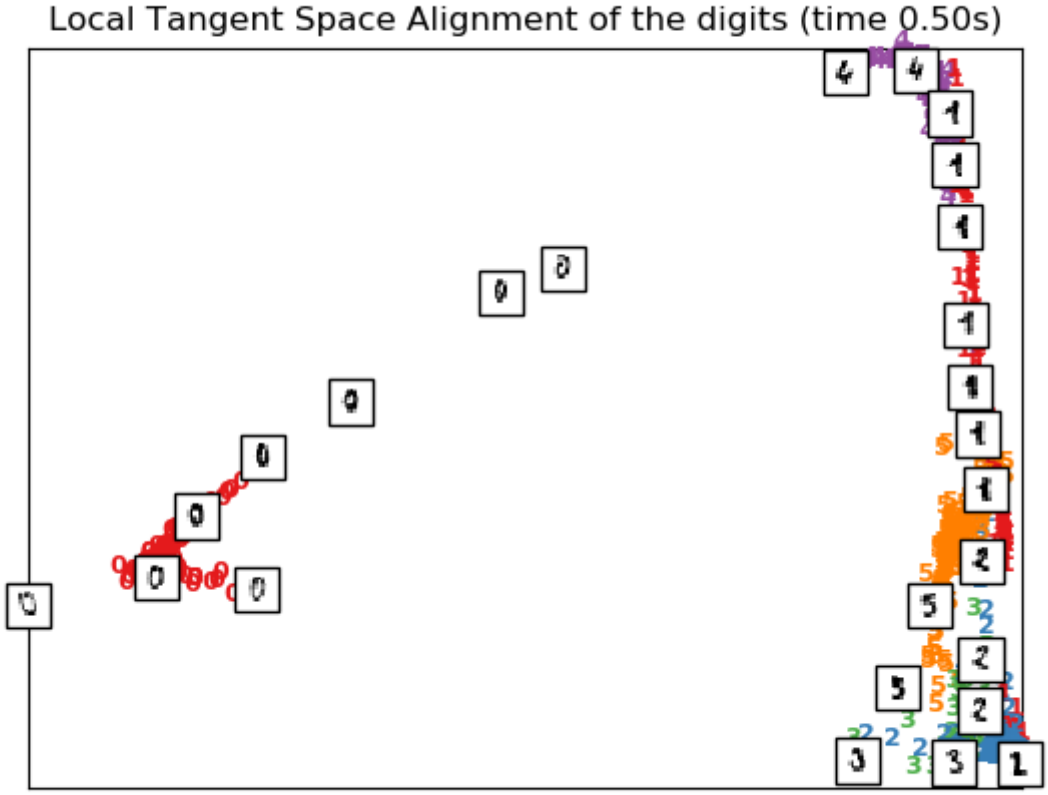

Linear Discriminant projection of the digits (time 0.01s)

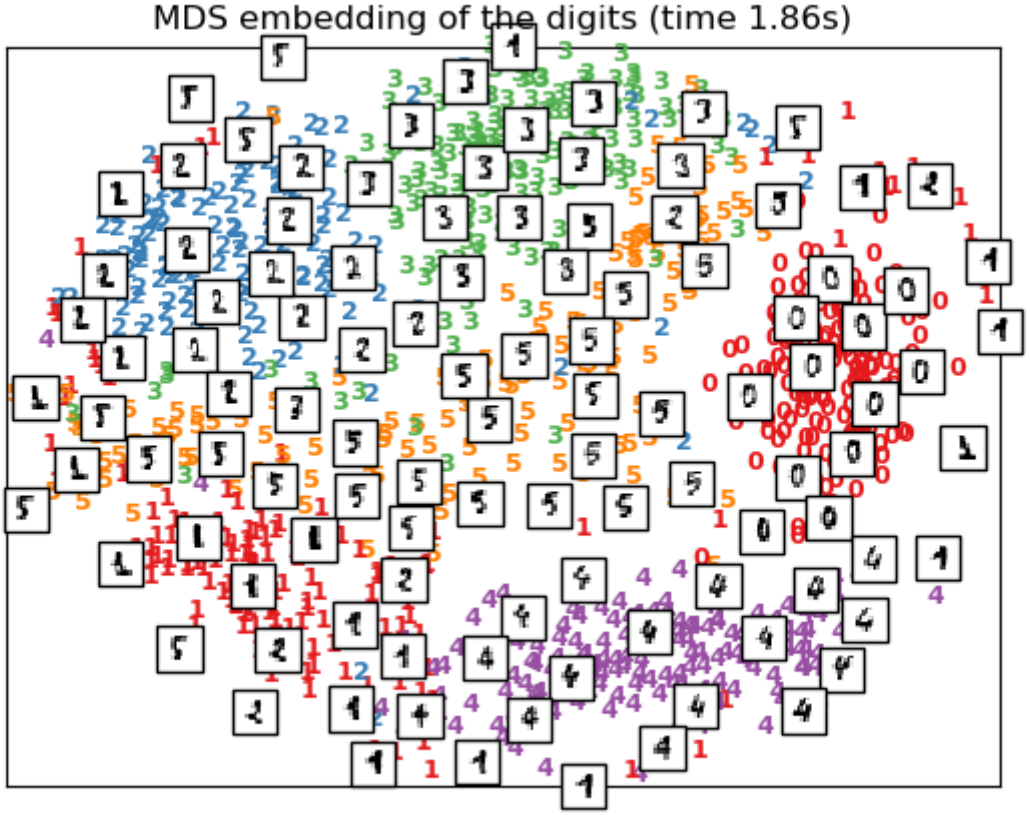

Isomap projection of the digits (time 1.06s)

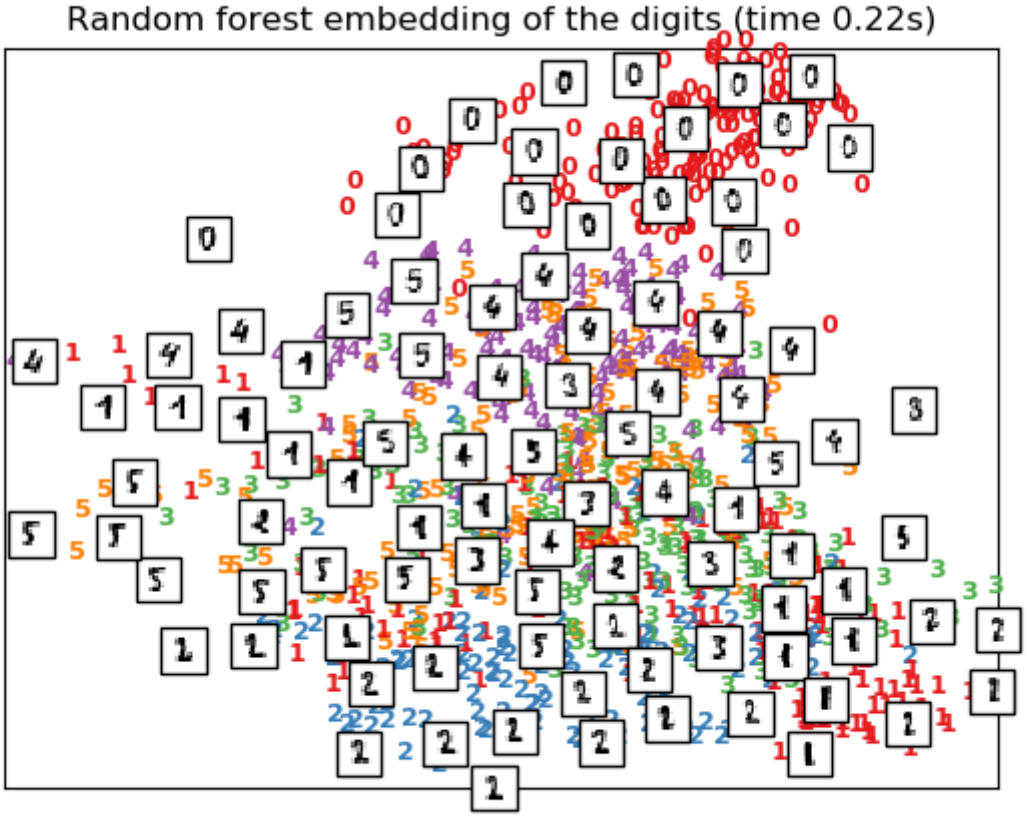

Locally Linear Embedding of the digits (time 0.34s)

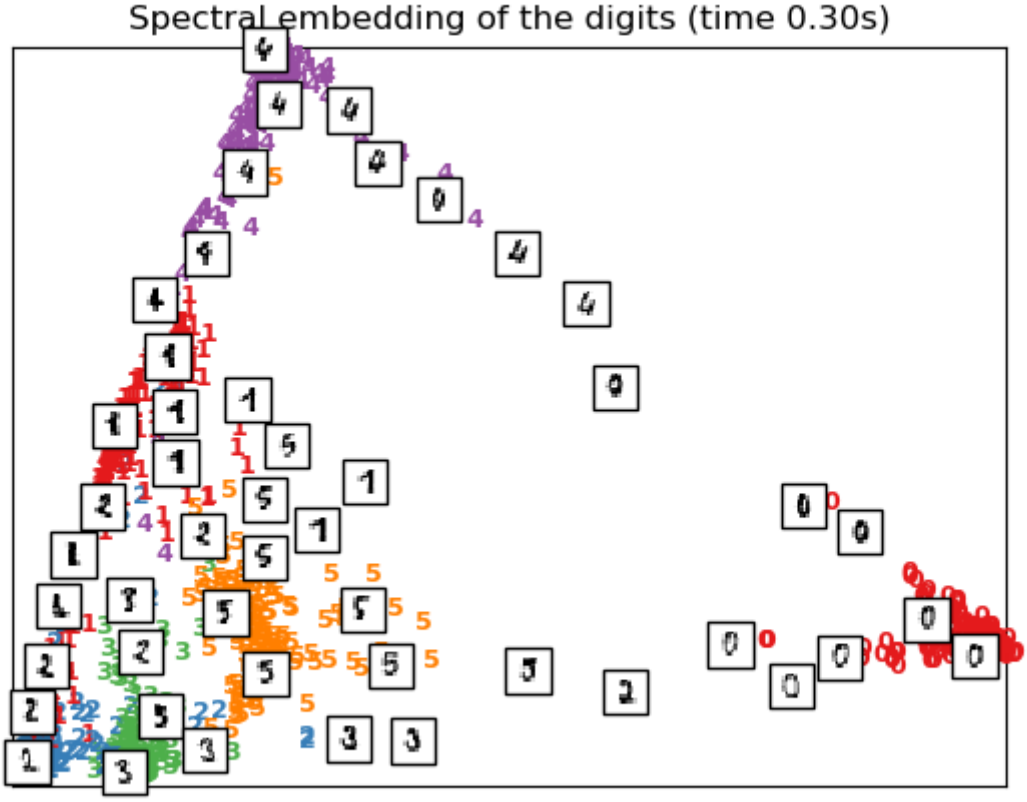

Modified Locally Linear Embedding of the digits (time 0.69s)

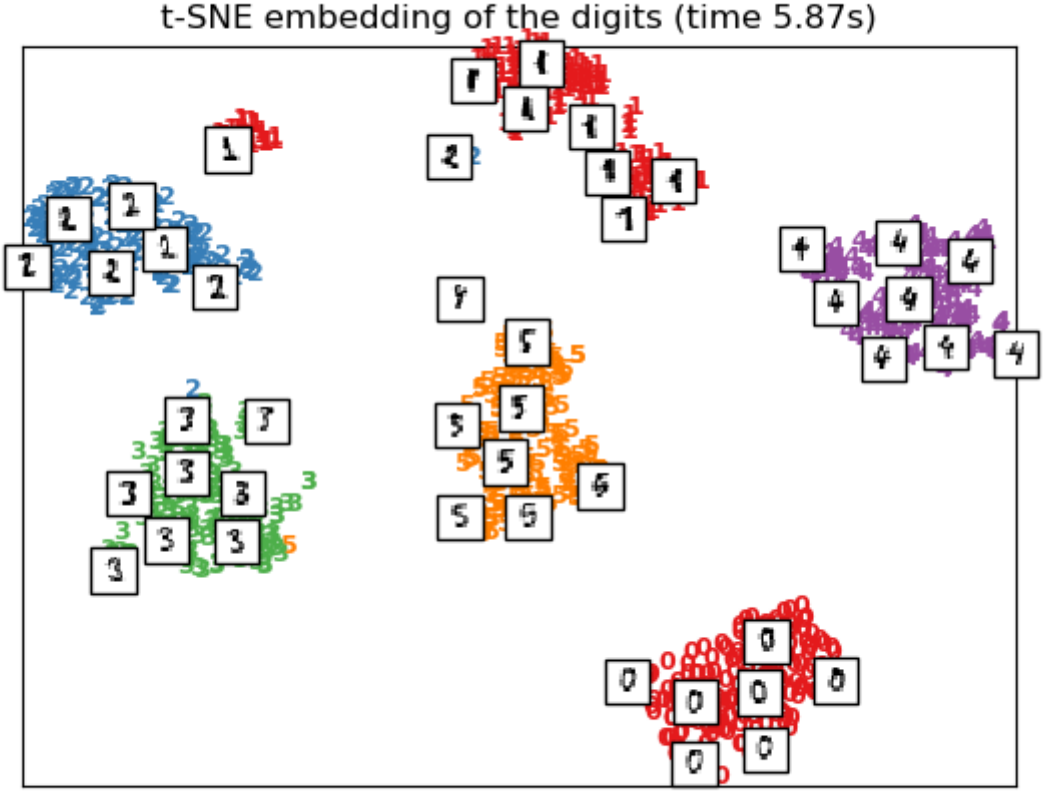

Hessian Locally Linear Embedding of the digits (time 0.76s)

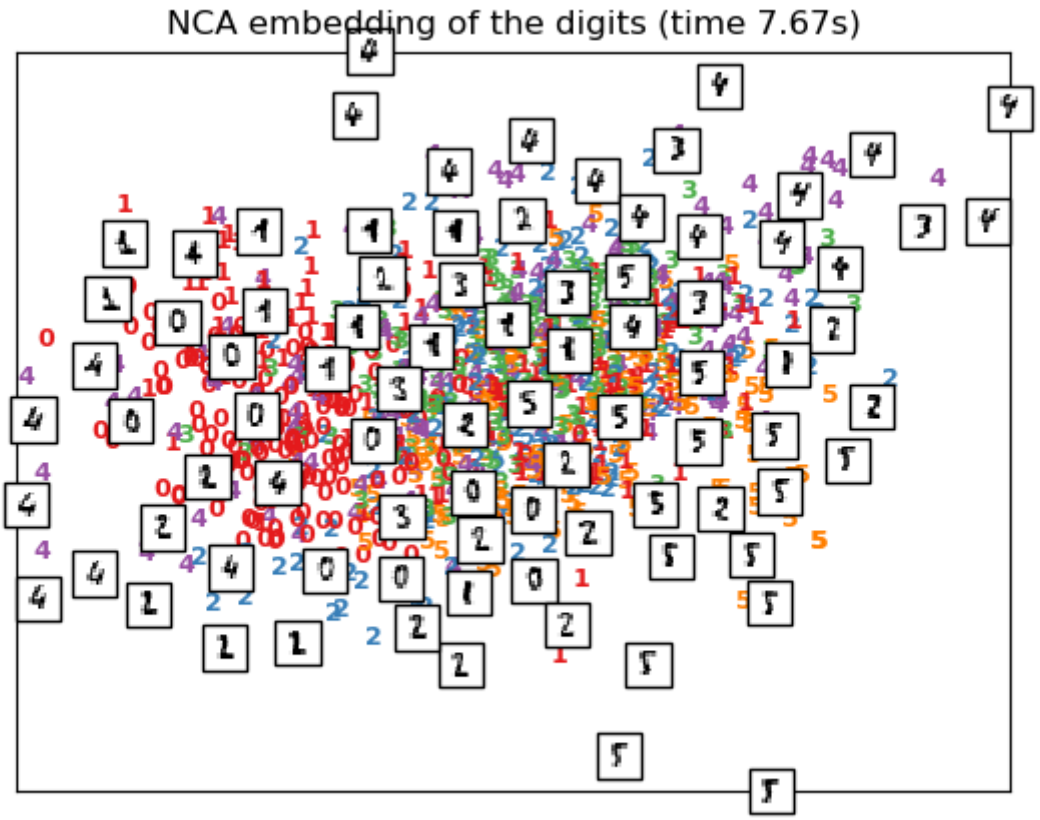

Local Tangent Space Alignment of the digits (time 0.50s)


MDS embedding of the digits (time 1.86s)


Random forest embedding of the digits (time 0.22s)


Spectral embedding of the digits (time 0.30s)


t-SNE embedding of the digits (time 5.87s)



t-SNE использовался для визуализации широкого ряда приложений, включая исследование компьютерной безопасности, музыкальный анализ, исследования по раку, биоинформатику и обработку биомедицинских сигналов. Алгоритм часто используется для визуализации высокоуровневых представлений, полученных из искусственной нейронной сети. UMAP имеет те же сферы применения.

Поскольку t-SNE отображения часто используются для показа кластеров, а на визуализацию кластеров может оказывать значительное влияние выбранная параметризация, постольку необходимо умение работать с параметрами алгоритма t-SNE. Для выбора параметров и проверки результатов могут оказаться необходимы интерактивные исследования. Было продемонстрировано, что алгоритм t-SNE часто способен обнаружить хорошо отделённые друг от друга кластеры, а при специальном выборе параметров аппроксимировать простой вид спектральной кластеризации.