Unsupervised Learning Techniques for Clustering and Dimensionality Reduction in MNIST

Students Adam Vinestock, Idan Cohen

Introduction

In this report, we present our findings on the application of unsupervised learning techniques for clustering and dimensionality reduction on the MNIST database of handwritten digits. The report is organized such that in each assignment part we will explain our approach and findings

Part 0: Data Loading and Exploration

In this section, we load the MNIST dataset and explore its characteristics, such as the number of samples, image size and distribution of digits. We perform no prepossessing on the data for future computation on the raw pixel level values. We perform train-test split, to prepare for further experiments. It is apparent that the data is balanced.

Part 1: Clustering

In this section, we apply clustering methods to the MNIST dataset and analyze the results. We experiment with three different clustering methods: K-means, K-means++ and Fuzzy K-means. For each method, we cluster the data using the raw pixel level values. Our approach was to choose different clustering methods with emphasis on determining the number of clusters (DBSCAN for example was avoided). Each method should be inherently different:

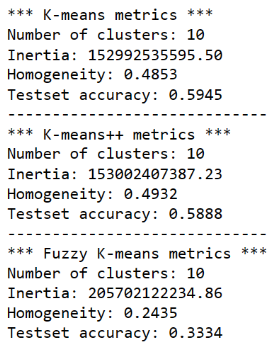
K-means being a popular centroid-based clustering algorithm that aims to minimize the within-cluster sum of squared distances.

K-means++ being an enhancement of K-means that improves the initialization step.

Fuzzy K-means allows relaxed membership assignment leading to more flexible and nuanced clustering, as data points can have partial membership to multiple clusters based on their proximity to different centroids.

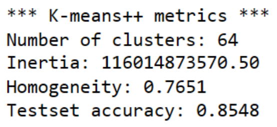
We evaluate the vanilla K-means clustering results using standard evaluation metrics, such as silhouette score and visualize the clusters by displaying a few images from each cluster. Based on the metrics it is apparent that there is no real structure in the clusters, although visually some clusters seem to show similarity in sample images.

We evaluate the test-set accuracy using each cluster method, all methods show above random performance (0.1) with fuzzy K-means being the most under performing method. Note that running the notebook may result in different metrics due to the stochastic methodologies. In parameter exploration We chose to focus on k = number of clusters. Although counter-intuitive to the nature of the task, some classes may have more than one way to sketch a digit (for example 7 with or without a cross line). This means that clustering a single class to multiple different clusters can improve performance. It is important to note that adding more clusters will always improve accuracy on the training data, that is why we seek the 'elbow effect' for choosing the optimal number of clusters. We then show the improvement of metrics and performance over the test set.



```
*** K-means metrics ***
Number of clusters: 10
Inertia: 152992535595.50
Homogeneity: 0.4853
Testset accuracy: 0.5945
-----------------------------
*** K-means++ metrics ***
Number of clusters: 10
Inertia: 153002407387.23
Homogeneity: 0.4932
Testset accuracy: 0.5888
-----------------------------
*** Fuzzy K-means metrics ***
Number of clusters: 10
Inertia: 205702122234.86
Homogeneity: 0.2435
Testset accuracy: 0.3334
```

Figure 1: Test-set performance



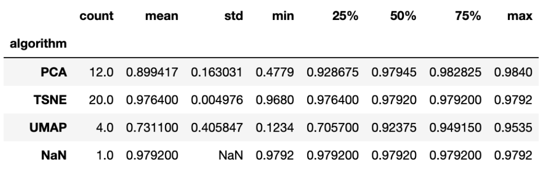
```
*** K-means++ metrics ***
Number of clusters: 64
Inertia: 116014873570.50
Homogeneity: 0.7651
Testset accuracy: 0.8548
```

Figure 2: Performance after adjusting number of clusters

Part 2: Dimensionality Reduction

We chose to apply Dimensionality Reduction using PCA, t-SNE (Barnes-Hut), and UMAP.
During this section, we will follow the experiments results and settings, as they were recorded in pandas data-frame, and will evaluate the performance using the classifier score. Although we have run many experiments with different parameters, this still needs to be examined while keeping in mind that we didn't implement cross-validation.

Our analysis will be accompanied by figures which will visualize features. In these figures, each axis represents a single principal component, and each class label is represented by a unique color.



	count	mean	std	min	25%	50%	75%	max
algorithm								
PCA	12.0	0.899417	0.163031	0.4779	0.928675	0.97945	0.982825	0.9840
TSNE	20.0	0.976400	0.004976	0.9680	0.976400	0.97920	0.979200	0.9792
UMAP	4.0	0.731100	0.405847	0.1234	0.705700	0.92375	0.949150	0.9535
NaN	1.0	0.979200	NaN	0.9792	0.979200	0.97920	0.979200	0.9792

Figure 3: Summarized classification performance on reduced space over all experiments. 'NaN' - baseline, classification on the original space.

0.1 PCA

Figure 4 shows the first 6 principal components extracted by the PCA algorithm. One can easily see how the separation between the classes is clearly visible in the first two PC, but becomes less distinct and more blurry in the 5th and 6th components.
As shown in figure 8, most of the information in the data is captured by the first 18 principal components, and a noticeable decrease in performance happens when removing the 18th component. Additionally, we also can see the low score obtained when run with PC=2 (0.4779), which shows that using only the first two PCs does not provide sufficient information to effectively separate the classes (unlike t-SNE and UMAP, which we will see later).

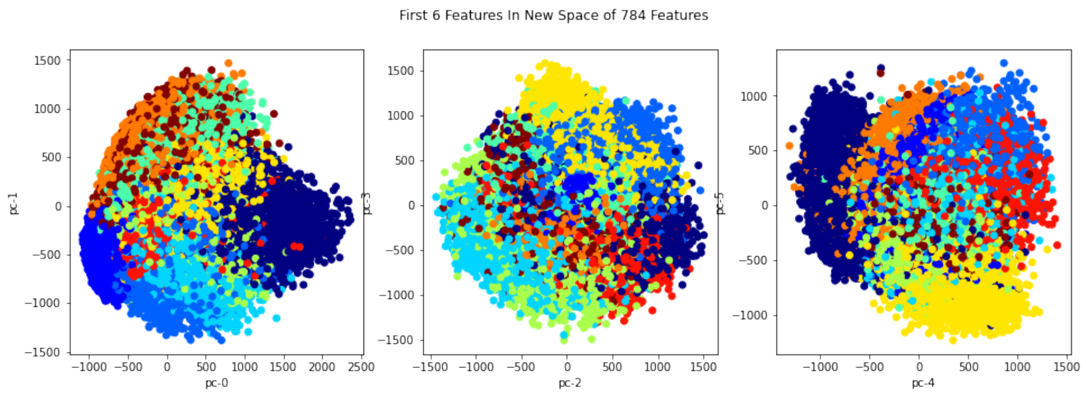
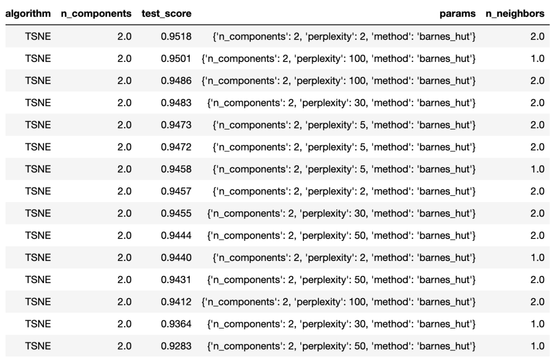


Figure 4: PCA visualization of first 6 PCs

0.2 t-SNE

While the PCA and UMAP allow us to learn a transformation from training data and use it for a new given sample, for the t-SNE solution we had to extend the algorithm in order to transform a new, previously unseen sample.

Therefore, inspired by concepts from Moodle discussion, we leveraged t-SNE such that we first apply t-SNE on the training set, then for new samples, such as those from the test-set, we applied the following steps:
1. Using the Nearest-Neighbors, find the $x_0, x_1, ..x_n$ nearest neighbors of x in the training set. Additionally, we label $s_i = s(x, x_i)$ as the similarity between the samples x, x_i, s.t, $\sum_{i=0}^{n} s_i = 1$.
2. Assume that $y_0, y_1, ..y_n$ are the reduced versions of $x_0, x_1, ..x_n$. Then, we use the following linear interpolation to define the transformed x in the new space: $s_0 * y_0 + s_1 * y_1 + ... + s_n *$



algorithm	n_components	test_score	params	n_neighbors
TSNE	2.0	0.9518	{'n_components': 2, 'perplexity': 2, 'method': 'barnes_hut'}	2.0
TSNE	2.0	0.9501	{'n_components': 2, 'perplexity': 100, 'method': 'barnes_hut'}	1.0
TSNE	2.0	0.9486	{'n_components': 2, 'perplexity': 100, 'method': 'barnes_hut'}	2.0
TSNE	2.0	0.9483	{'n_components': 2, 'perplexity': 30, 'method': 'barnes_hut'}	2.0
TSNE	2.0	0.9473	{'n_components': 2, 'perplexity': 5, 'method': 'barnes_hut'}	2.0
TSNE	2.0	0.9472	{'n_components': 2, 'perplexity': 5, 'method': 'barnes_hut'}	2.0
TSNE	2.0	0.9458	{'n_components': 2, 'perplexity': 5, 'method': 'barnes_hut'}	1.0
TSNE	2.0	0.9457	{'n_components': 2, 'perplexity': 2, 'method': 'barnes_hut'}	2.0
TSNE	2.0	0.9455	{'n_components': 2, 'perplexity': 30, 'method': 'barnes_hut'}	2.0
TSNE	2.0	0.9444	{'n_components': 2, 'perplexity': 50, 'method': 'barnes_hut'}	2.0
TSNE	2.0	0.9440	{'n_components': 2, 'perplexity': 2, 'method': 'barnes_hut'}	1.0
TSNE	2.0	0.9431	{'n_components': 2, 'perplexity': 50, 'method': 'barnes_hut'}	2.0
TSNE	2.0	0.9412	{'n_components': 2, 'perplexity': 100, 'method': 'barnes_hut'}	2.0
TSNE	2.0	0.9364	{'n_components': 2, 'perplexity': 30, 'method': 'barnes_hut'}	1.0
TSNE	2.0	0.9283	{'n_components': 2, 'perplexity': 50, 'method': 'barnes_hut'}	1.0

Figure 5: tSNE Barnes-Hut Experiments Results

y_n.

According to the experiments, the parameters that achieved the highest score were with $n\_components = 2$, $perplexity = 2$, and with $n\_neighbors = 2$ for Nearest-Neighbors. See Figure 6 for feature visualization. However, as seen in Figure 5 also other hyper-parameters achieved very close results, and these minor score differences could be due to chance, which would be eliminated if we conducted our tests using cross-validation.

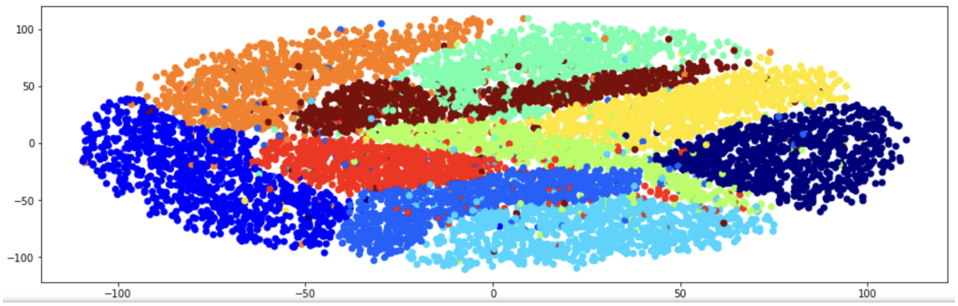


Figure 6: tSNE Barnes-Hut Features Visualization with n_components=2

0.3 UMAP

UMAP is known for its relatively fast algorithmic run-time, even for large datasets. Thus, our experiments have also shown that UMAP produces excellent results, as indicated by the test score, while running times were pretty fast. The best UMAP results were achieved with a score of 0.95, using the parameter $n\_neighbors = 15$ [1], in a running time of 90 seconds. See Figure 7 for feature visualization .

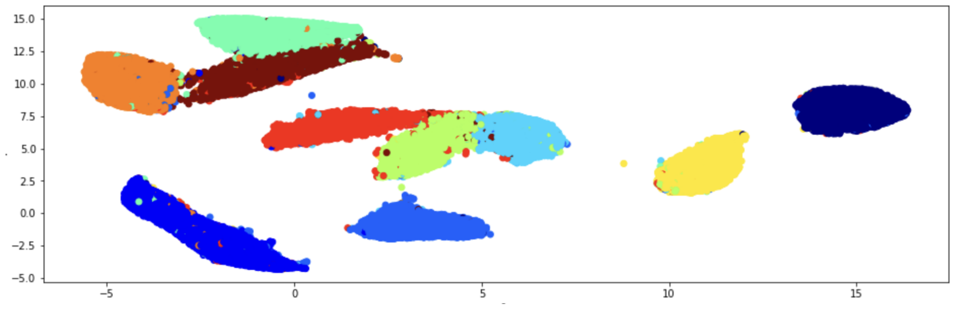


Figure 7: UMAP features Visualization with $n\_components = 2$

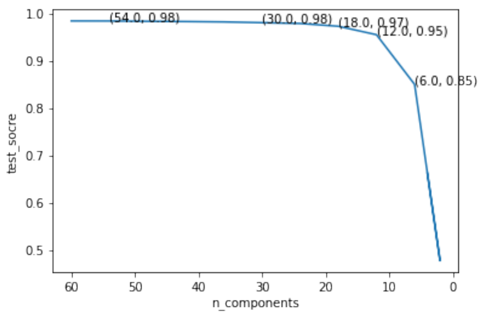


Figure 8: PCA classification score, using different n_components values

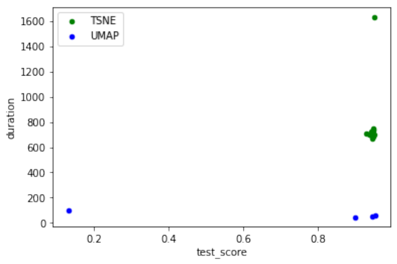


Figure 9: UMAP and tSNE, score and duration (seconds) plot

As UMAP and tSNE both reconstruct lower-dimensional space while preserving the global structure, we choose to examine them also in terms of running time. Figure 9 clearly shows how UMAP outperforms tSNE (Barnes-hut), with an

[1]Official documentation: "Controls how UMAP balances local versus global structure in the data. It does this by constraining the size of the local neighborhood UMAP will look at when attempting to learn the manifold structure of the data."

average completion time of 60 seconds compared to 769 seconds of tSNE[2].

Part 3: Classification of the Raw Data

In this section, we apply the KNN classification algorithm. We explored both SVM and logistic regression, but chose to show how a very simple classifier can achieve surprisingly high accuracy on test set classification. We experimented with different values of k, that represents the number of nearest neighbors used for classification. Although minimal in accuracy difference, we observed that k=3 yielded the best results with an accuracy of 0.9705. This could possibly be explained by:

1. Overlapping of classes, where the digits are not sparsely separated, such that using a smaller value of k allows for finer local decision boundaries.
2. A large number of samples, looking at smaller subsets will most likely find very similar samples, leading to accurate predictions. However, further analysis and experimentation may be needed to validate this hypothesis.

Part 4: Summary

Our findings demonstrate that a straightforward classification approach can effectively surpass unsupervised methods in a supervised classification task. Moreover, we conduct comprehensive experiments on various hyperparameters to carefully analyze three distinct dimensionality reduction algorithms. Our results reveal that we can successfully reduce the dimensionality of the data while preserving essential characteristics and information from the original space by selecting a specific set of hyperparameters for each algorithm. Moreover, as seen in a summarizing overview in Figure 3, we can see that all the algorithms, when optimal hyper-parameters were used, achieved more or less the same results.

[2]Recall that the tSNE running time includes also the Nearest-Neighbors running time