Hunter Worssam
September 29th, 2025
Data Science Modeling & Analytics
Research Discussion #2

Module 5: Leveraging Unsupervised Learning Models for Clustering and Dimensionality Reduction from TensorFlow Hub and Kaggle.
Students should critically explore unsupervised learning methods specifically related to clustering and dimensionality reduction by using pre-trained models or techniques available from TensorFlow Hub and Kaggle. Investigate TensorFlow Hub's pre-trained auto encoder architectures for dimensionality reduction tasks, and examine clustering algorithms such as DBSCAN and K-Means demonstrated in Kaggle notebooks. In your discussion, emphasize the comparative performance of these approaches, appropriate use-cases, and the computational resources required. Highlight how such unsupervised methods significantly enhance data interpretation and the discovery of inherent patterns within complex datasets. Provide real-world or simulated dataset examples to support your analysis.

Background

Unsupervised learning helps uncover structure in unlabeled data. In this discussion I compare two families of approaches: dimensionality reduction with auto encoders (AEs) from TensorFlow Hub (followed by clustering on the reduced dimensions); and direct clustering on raw and PCA-reduced features using K-Means and DBSCAN as commonly demonstrated in Kaggle notebooks. Auto encoders can learn non-linear patterns that linear methods such as PCA can miss, often yielding tighter clusters downstream. K-Means is fast and scalable but assumes roughly spherical, equal-variance clusters which may not hold up across unique datasets. DBSCAN is density-based and can recover arbitrary shapes and detect noise, but is sensitive to the hyper parameters `eps & min_samples,` and can be memory-heavy at scale. I also touch on compute trade-offs: auto encoders typically benefit from a GPU; K-Means is a lighter-weight, CPU friendly approach; and DBSCAN can become costly in high dimensions depending on parameters.

Datasets Used:

- MNIST Images: to test whether non-linear latent spaces improve cluster separation over raw pixels.

- Kaggle "Mall Customers": a small, widely used dataset for comparing K-Means and DBSCAN on consumer segments.

Models Used:

TensorFlow Hub encoders (for dimensionality reduction):

- VTAB WAE-MMD encoder (latent features via Wasserstein auto encoder).
- VTAB VAE encoder (latent features via variational auto encoder).

Baselines / Clusterers:

- PCA & K-Means via scikit-learn.
- DBSCAN for density clusters/outlier detection via scikit-learn..

Questions to Answer:

1. Do AE-based features improve cluster quality vs. PCA or raw features?
2. When does K-Means outperform DBSCAN, and vice versa?
3. What are the compute and memory costs of each approach?
4. How sensitive are results to hyper-parameters (latent size, `k`, `eps`, `min_samples`)?
5. What use-cases best fit each method?

Experimental Design:

Datasets:
MNIST (Kaggle train.csv): 42k rows × 784 pixels with labels
Mall Customers: Gender, Age, Annual Income (k$), Spending Score (1–100) (no labels).

Preprocessing:
For the MNIST dataset, each 28x28 image was taken and each of the 784 pixels were standardized from ranges [0, 255] to [0, 1] to support auto encoding. Additionally, the pixels in range [0, 1] were standardized using the standard scaler to mean = 0 and variance = 1. This was a requirement for PCA and so for consistency the final results were run using the standard-scaled pixels.

For the mall customers dataset, the gender column was one hot encoded to a binary gender_is_male column. The three numeric columns (age, annual income, spending score) were scaled using the standard scaler.

Dimensionality reduction
In an effort to test the impact of linear and non-linear dimensionality reduction techniques on clustering accuracy, principal component analysis (linear) and auto encoders (non-linear) were deployed on both MNIST and mall customer datasets.

For the MNIST dataset, PCA was used to create four feature sets varying in sizes of 16, 32, 64 and 100 features total. For the auto encoder approach, a dense auto encoder was used to create latent feature sizes of size 16 and 32. Including the raw 784-feature dataset, this created seven unique data sources to test clustering on.

For the customer dataset, the starting feature count was much smaller at only 4 unique features. Because of this, the scope of the PCA step was much smaller, resulting in datasets containing 2 and 3 features. For the auto encoder approach, the choice was made to distill 2 latent features given the auto encoder's ability to distill potent non-linear patterns from the data. Including the raw 200 x 4 feature dataset, this yielded four unique data sources to cluster on.

Clustering

To recap, we now have seven unique MNIST datasets featuring a wide variety of dimensionality reduction techniques and depths. These range from just 16 unique features up to 784 in the raw dataset. For the customer dataset, we have four unique feature sets, ranging from 2 - 4 features in total. The goal with the clustering step was to evaluate both label matching capability (for MNIST) and clustering characteristics (for Customers) using each of these unique feature sets in a kMeans approach and in a DBSCAN approach.

For KMeans on the MNIST dataset, the target k was fixed at 10 since we knew going in that ten unique digits were present in the dataset. The customers dataset was a bit more interesting here since we did not have predefined labels in our data. As a result, the silhouette method was leveraged to deploy the elbow method for an optimal k selection. As a result, the chosen k was either 4 or 6 depending on the feature set used in the model. Time to fit, silhouette score, Davies-Bouldin and Calinski-Harabasz scores were reported for each feature set in the MNIST and Customer datasets. High silhouette scores, low D-B values and high C-H scores indicate optimal clustering among a set of k's.

For the DBSCAN approach, each of the unique feature sets called for their own hyper parameter tuning. The min_samples parameter was fixed at 5 for both the MNIST and Customer datasets, which means five samples had to be within the neighborhood of a point for it to be considered a core point in a cluster.

The eps parameter on the other hand was optimized for each feature set because it determines the maximum distance between two samples for on to be considered as "in the neighborhood" of the other. This tuning was done by taking a subset of the training data (12k rows of MNIST, full Customer dataset), fitting a kNN to the subset, and calculating the distribution of distances between each point and it's 5th nearest neighbor. With this distribution of distances in hand, three "eps" values were chosen at random between the 60th and 90th percentile of the distribution, and these three eps values were screened in a DBSCAN grid search. The value that yielded the highest silhouette score was chosen to be the EPS value for that given feature set.

The final eps value chosen differed greatly depending on the dimensionality reduction technique used on the feature set. The MNIST feature sets derived from PCA often required much larger eps values (2.9 - 9.3 for 16-feature PCA and 100-feature PCA respectively). This is intuitive, because as we expand our dimensionality we are

inherently introducing sparsity, meaning our eps value must increase to achieve the same 5-nearest neighbor clustering.

The auto encoded feature sets on the other hand are far more compact due to the non-linear embedding process it uses. As a result, the optimal eps values chosen for the 16-feature AE and 32-feature AE approaches were 0.90 and 0.28 respectively. The difference in optimal DBSCAN eps values between the PCA and AE approaches really highlights just how much more compact the feature spaces are when using the AE method.

Metrics and runtime

I evaluated clustering quality with three internal metrics that don't need labels: Silhouette (higher is better, meaning points are closer to their own cluster than to others), Davies–Bouldin (lower is better, penalizes overlapping/loose clusters), and Calinski–Harabasz (higher is better, rewards well-separated, compact clusters). For MNIST, where labels exist, I also reported two external metrics, NMI and ARI, which measure how well the discovered clusters align with the true digit classes (higher is better for both).

To reflect practical performance, I recorded the wall-clock fit time for every run. Because MNIST is large, Silhouette was computed on a random subset of the data when needed to keep computations fast, preserving the relative comparisons while significantly reducing runtime.

Results

MNIST (42k images × 784 pixels)

- Label accuracy:
  • K-Means on PCA-16-features gave the best external scores: *NMI ≈ 0.416, ARI ≈ 0.291* with okay silhouette (~0.105).
  • K-Means on raw data was similar externally (NMI ≈ 0.430, ARI ≈ 0.317) but with very low silhouette (~0.008).

- Cluster compactness:
  • AE-32-features with K-Means showed much higher compactness (*sil ≈ 0.344, DB ≈ 0.829, very large CH*) but lower NMI/ARI (~0.17/0.12), i.e., well-separated clusters that don't map neatly to digits.
  • DBSCAN on reduced spaces had the highest silhouettes (e.g., PCA-32 sil ≈ 0.485, PCA-16 sil ≈ 0.428) but near-zero NMI/ARI, indicating density islands not aligned with labels.

- Compute requirement:
  • DBSCAN on raw data was very slow (≈ 103 s).
  • DBSCAN on PCA/AE spaces dropped to 0.2–30 s (AE-32 DBSCAN ≈ 0.23 s).
  • K-Means across all spaces completed in a few seconds.

- Latent size sensitivity:
  • AE-32 was better than AE-16 on cluster quality (higher silhouette), but external scores remained low for both.
  • For PCA, smaller MNIST subspaces (≈16D) tended to match labels better than 64–100D.

Full MNIST Results Table, featuring 7 unique feature sets on kMeans and DBSCAN Clustering. Higher Score = better ability to predict class label

[66]:

	Feature_Space	Method	silhouette	NMI	ARI	davies_bouldin	calinski_harabasz	Fit_Time_s	Score
0	MNIST-PCA16-std	KMeans(k=10)	0.105	0.416	0.291	2.001	2300.1	2.532	4.0
1	MNIST-raw-std	KMeans(k=10)	0.008	0.430	0.317	3.310	868.4	11.995	4.0
2	MNIST-PCA32-std	KMeans(k=10)	-0.005	0.381	0.244	2.916	1068.0	3.118	5.7
3	MNIST-raw-std	DBSCAN(eps=27.4674, ms=5)	0.445	0.012	0.000	3.198	67.9	103.271	6.0
4	MNIST-AE32-std	KMeans(k=10)	0.344	0.169	0.124	0.829	35895.9	3.042	6.3
5	MNIST-PCA64-std	KMeans(k=10)	0.018	0.310	0.143	3.571	518.3	4.148	6.3
6	MNIST-AE16-std	KMeans(k=10)	0.212	0.173	0.090	1.229	28006.1	2.852	6.7
7	MNIST-PCA100-std	KMeans(k=10)	-0.009	0.315	0.173	3.576	349.9	4.451	6.7
8	MNIST-PCA100-std	DBSCAN(eps=9.2670, ms=5)	0.371	0.007	0.000	3.206	64.4	44.457	8.0
9	MNIST-PCA32-std	DBSCAN(eps=4.8487, ms=5)	0.485	0.005	0.000	2.342	158.5	16.218	8.3
10	MNIST-PCA64-std	DBSCAN(eps=7.3711, ms=5)	0.376	0.006	0.000	3.482	122.4	29.290	8.7
11	MNIST-PCA16-std	DBSCAN(eps=2.8653, ms=5)	0.428	0.003	0.000	2.377	242.0	8.186	9.7
12	MNIST-AE32-std	DBSCAN(eps=0.2848, ms=5)	-0.201	0.009	0.000	1.735	76.9	0.227	11.0
13	MNIST-AE16-std	DBSCAN(eps=0.9041, ms=5)	-0.023	0.002	0.000	1.899	59.5	1.545	13.7

<u>Customers (200 rows x 4 features)</u>

- Best overall segmentation:
  • AE-2-features + K-Means(k=4) led with *sil ≈ 0.50*, low DB ≈ 0.65, high CH, suggesting clean, compact segments.
  • PCA-2/3-features + K-Means also strong (sil ≈ 0.43).

- DBSCAN behavior:
  • Competitive silhouettes on AE-2-features (≈ 0.45) and PCA-2-features (≈ 0.42), but DB index sometimes worse (AE-DBSCAN DB score ≈ 4.06), suggesting less compact groups.

- Compute requirement:
  • All runs (K-Means/DBSCAN) completed in milliseconds on this small dataset.

Full Customers Results Table, featuring 4 unique feature sets on kMeans and DBSCAN Clustering. Higher Score = better clustering characteristics

[68]:

	Feature_Space	Method	silhouette	davies_bouldin	calinski_harabasz	Fit_Time_s	Score
0	CUST-AE-std	KMeans(k=4)	0.500	0.650	273.2	0.004	1.3
1	CUST-PCA2-std	KMeans(k=4)	0.429	0.772	174.3	0.003	3.3
2	CUST-PCA3-std	KMeans(k=6)	0.431	0.812	122.1	0.004	3.7
3	CUST-PCA2-std	DBSCAN(eps=0.5973, ms=5)	0.419	0.696	9.1	0.012	4.7
4	CUST-AE-std	DBSCAN(eps=0.4991, ms=5)	0.449	4.063	63.6	0.014	5.0
5	CUST-raw-std	KMeans(k=6)	0.356	1.005	99.7	0.006	5.3
6	CUST-raw-std	DBSCAN(eps=1.1732, ms=5)	0.304	0.513	3.4	0.014	5.3
7	CUST-PCA3-std	DBSCAN(eps=0.6846, ms=5)	0.274	3.299	7.5	0.014	7.3

Discussion

Do AE-based features improve cluster quality vs. PCA or raw features?

- MNIST: AE latents (especially AE-32-features) improved clustering metrics and dramatically accelerated DBSCAN hyper parameter tuning and training time, but did not improve label alignment via NMI/ARI score. PCA-16-features + K-Means best matched labels despite modest silhouette. Perhaps a tradeoff exists here between having perfectly tight clusters and actually accommodating the variance inherent to the different image classes.
- Customers: AE + K-Means gave the cleanest, most actionable segments; PCA-2-features and 3-features were strong, simpler alternatives.

When does K-Means outperform DBSCAN, and vice versa?

- K-Means wins when classes are roughly centroid-separable or when you need stable, interpretable segments (MNIST label alignment; Customer centroids).
- DBSCAN excels at density islands/outliers and arbitrary shapes; on MNIST it produced the highest silhouettes in reduced spaces, but its clusters didn't correspond to digit labels. Personally, I'd lean on DBSCAN more when trying to simply identify unlabeled patterns, and less for classification tasks.

Compute and memory costs.

- K-Means is fast and memory-light (≈ O(k·n·iterations)).
- DBSCAN is very sensitive to dimensionality; on raw MNIST it was slow (≈ 103 s). After PCA/AE reduction it became practical (sub-second to tens of seconds).
- AE training/encoding adds upfront cost but pays back in faster clustering, especially for DBSCAN.

Hyper-parameter sensitivity.

- Latent size: On MNIST, AE-32 > AE-16 for clustering scores, but was still poor at matching labels. Unsupervised AEs don't guarantee class separation.

- k (K-Means): MNIST fixed to k=10; on Customers, k ≈ 4 & 6 maximized silhouette (best was k=4 on AE).
- DBSCAN (`eps`, `min_samples`): Performance hinged on eps; using a kNN-derived ε band + 3 trials stabilized results and drastically cut runtime. `min_samples=5` worked well; larger values sometimes fragmented clusters.

Best-fit use-cases.

- PCA-16 + K-Means (MNIST): when you want clusters that track labels reasonably well.
- AE-32 + K-Means (MNIST): when you want compact, fast clusters for exploration or investigation, not for classification.
- DBSCAN on PCA/AE (MNIST): when the goal is structure discovery & outlier detection, not label matching.
- AE (or PCA-2-features) + K-Means (Customers): for marketing-style segmentation with clean, interpretable clusters.

Best Practices

- Always compare raw vs. PCA vs. AE features before clustering.
- Ensure data is scaled, preferably using the StandardScaler, prior to clustering to ensure alignment across all dimensionality reduction conditions.
- For DBSCAN, leverage k-mean clustering to help choose eps hyper parameter grid values. This can be done by performing kMeans, calculating average distance for each point to your min_samples value, and then trying a couple eps values around that average.
- Record compute time alongside clustering metrics for a fair comparison.

References

685.701 Model Building PDF (Available in “Model Building” section of Course Information).

685.701 Module05_LectureNotes PDF (Available in “Module 5” section).

TensorFlow Core: *Intro to Autoencoders: https://www.tensorflow.org/tutorials/generative/autoencoder*

TensorFlow Hub: VTAB WAE-MMD and VAE encoders: https://www.tensorflow.org/tutorials/generative/cvae

scikit-learn docs: K-Means complexity & MiniBatchKMeans; DBSCAN memory notes:

- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html
- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
- https://scikit-learn.org/stable/modules/generated/sklearn.cluster/MiniBatchKMeans

Kaggle notebooks demonstrating K-Means vs DBSCAN on “Mall Customers”: https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python