Digit classification with unsupervised learning model

Overview

The main objective of this project is to explor how good the performance can be using unsupervised learning models to classify the handwritten digits in MNIST dataest (https://www.kaggle.com/competitions/digit-recognizer (https://www.kaggle.com/competitions/digit-recognizer)).

This is the report of the project, which skipps steps for trying and tuning, only outlines the most important steps and results. The images/results are generated in other notebooks.

For more detailed steps, for readiblity reason I divided them into 4 notebooks, which can be found in the Github repository:

- **MNIST-1: Unsupervised-EDA**
  - Exploratory Data Analysis of the data, including statistics, plots, dimension reduction, etc.
- **MNIST-2:Unsupervised learning**
  - Expolor unsupervised learning models, anaylise the results and build a final model with test result generation.
- **MNIST-3: Data efficiency**
  - Assume we have limited amount of training data, and comparing the performance of supervised learning and unsupervised learning with training data for parameters selection.
- **MNIST-4: Validation of data efficiency**
  - To confirm what we concluded from the last notebook, we only use a labeled data set with 200 samples to predict the digits of the test data, if the result accuracy according to the conclusion.

References:

During the project I mainly refered to the following articles:

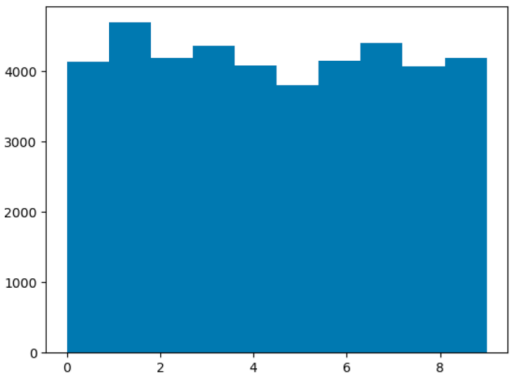
- https://www.digitalocean.com/community/tutorials/mnist-dataset-in-python (https://www.digitalocean.com/community/tutorials/mnist-dataset-in-python)
- https://www.kaggle.com/code/manabendrarout/unsupervised-learning-for-mnist-with-eda (https://www.kaggle.com/code/manabendrarout/unsupervised-learning-for-mnist-with-eda)

EDA

Data overview

- 28 x 28 = 784 pixels image
- Training data: 42k
- Testing data: 28k

Training data distribution

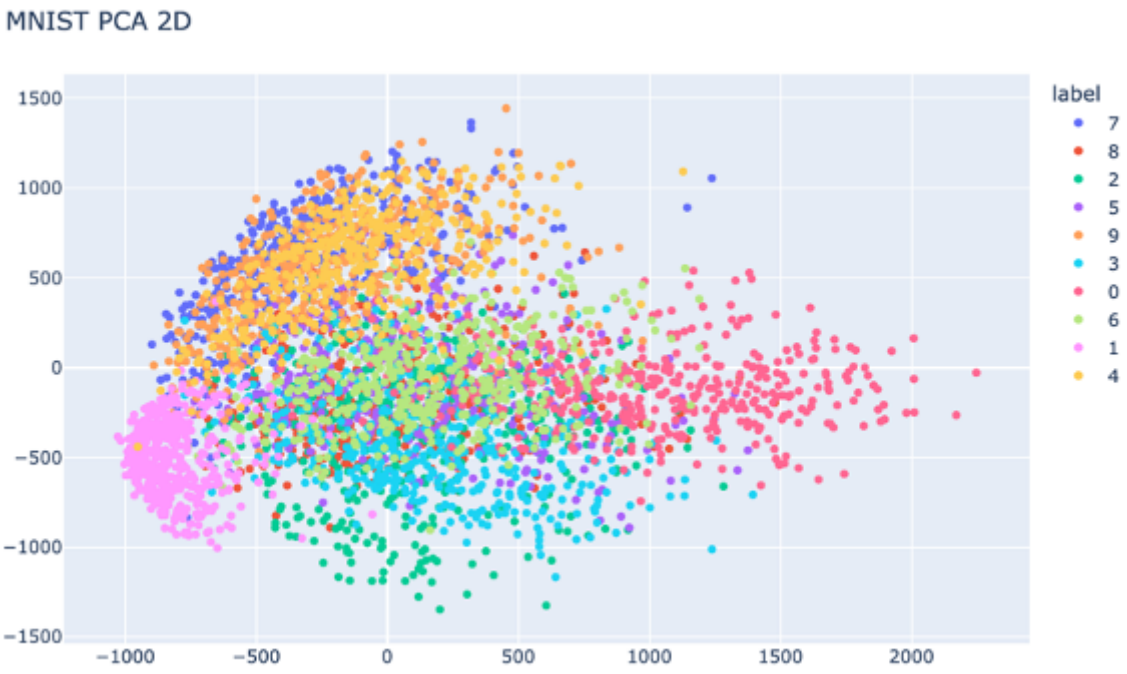


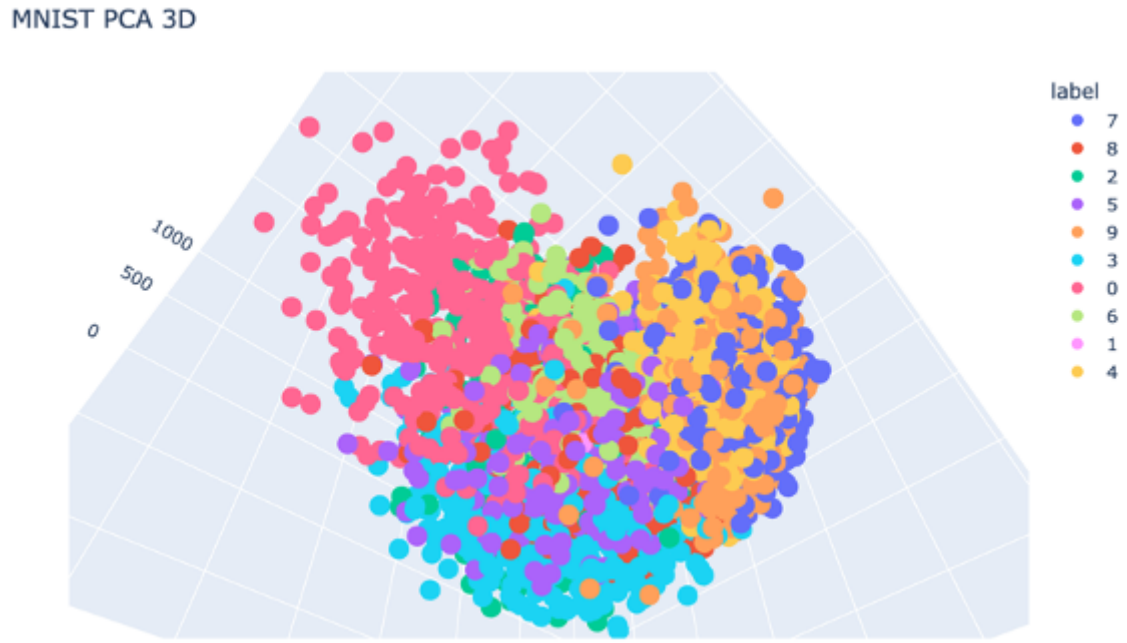
Digits images preview

Here I plotted the first 10 images from the training dataset

Dimension reduction - PCA

We try with the first 2 and 3 principal components, together with their true label in different color.





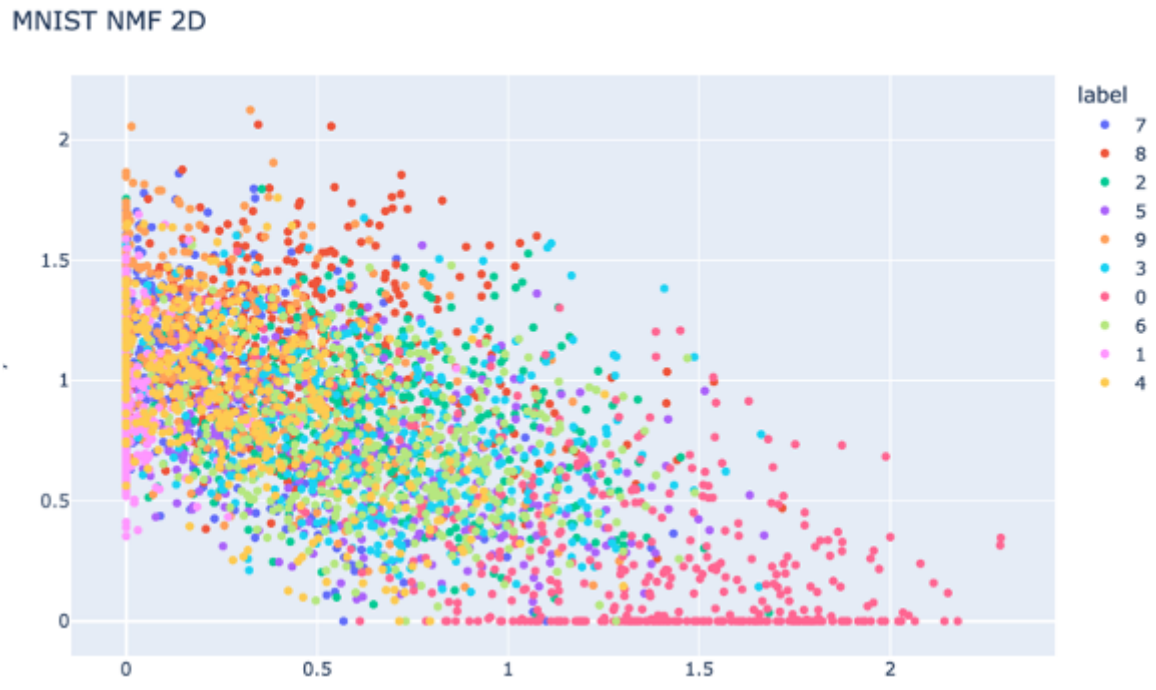
(***Note***: when run the notebook, you can explor the 3D plot interactivly, e.g. scale, rotate, etc.)

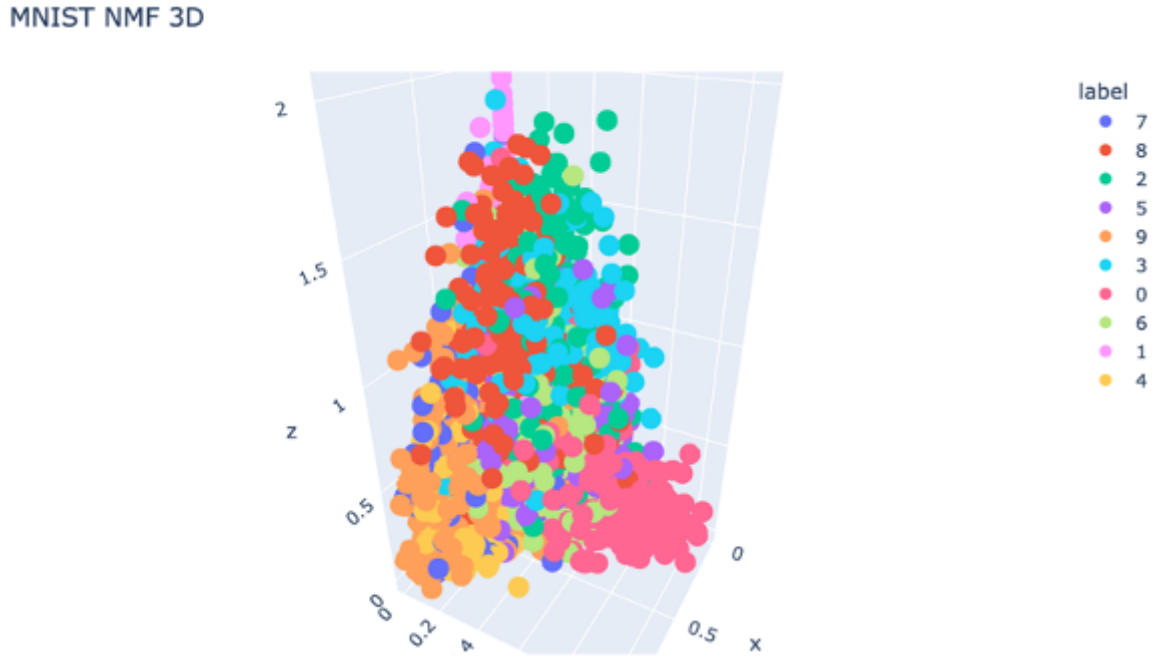
Observations:

- With both first 2 and 3 principal components, there are some separation, but they can not be well separated.
- '1' is relatively separated from others; '4', '9', '7' are mixed together, etc. This is quite intuitive, it depends if the original digits looks similar or not.

Dimension reduction - NMF

NMF is not mainly intened for dimension reduction, but it can dicompose the feature space to components, and we can see the different with PCS from plots.





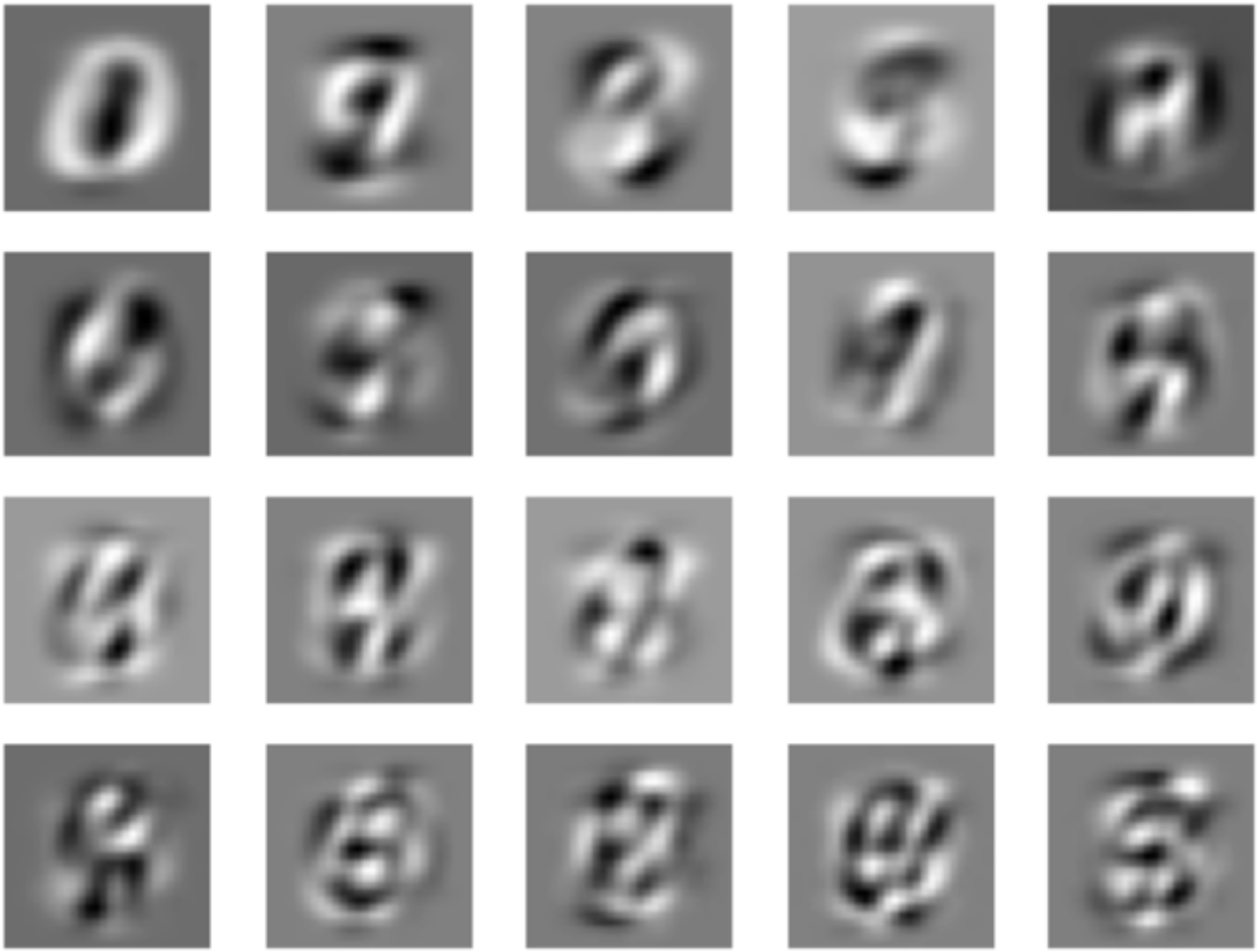
Observations:

- With 2 or 3 components, it seems the digits confuse even more than the PCA.
- I think it is because the factoration has a limitation of non-negative in comparing with PCA, so it not always catch the feature vectors of most variance.

Components plot

We want to take a look at how the components looks like in PCA and NMF. I plot 20 components of PCA and NMF respectively. You will see what the differences are between them.

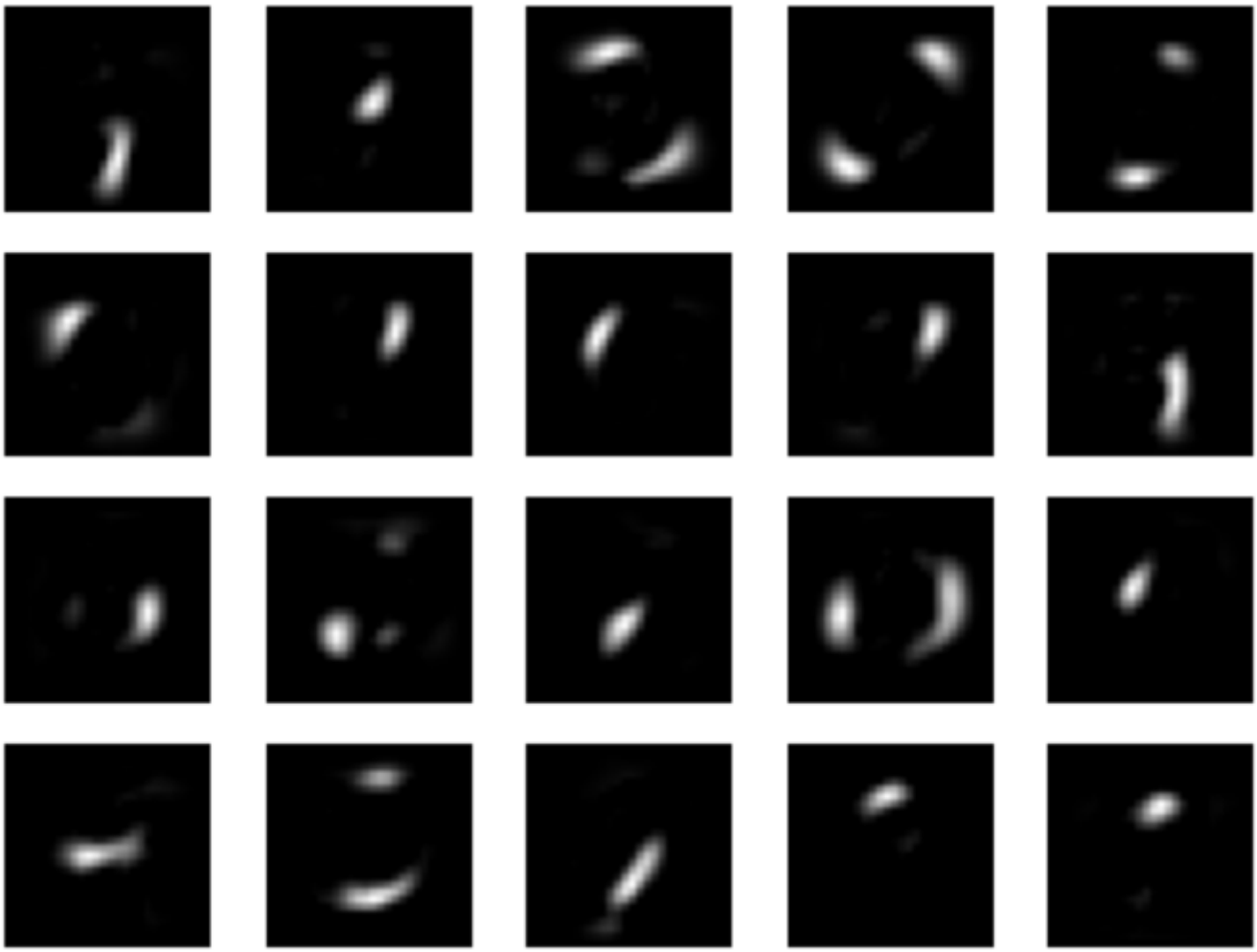
PCA - "Meta digits"



Observations:

1. For each of principal components, it contains a full layer of the whole picture (i.e. not a part), we may call them 'meta digit's. When it tryes to recover the image from pca compnents, it basically kind of layers all the components together.
2. There is no change when we change the number of components, because they always select the direction with the most variance.
3. The components in front extract the main structures of the digits, while the later ones are more fine tuned to extract details.

NMF – Digit parts

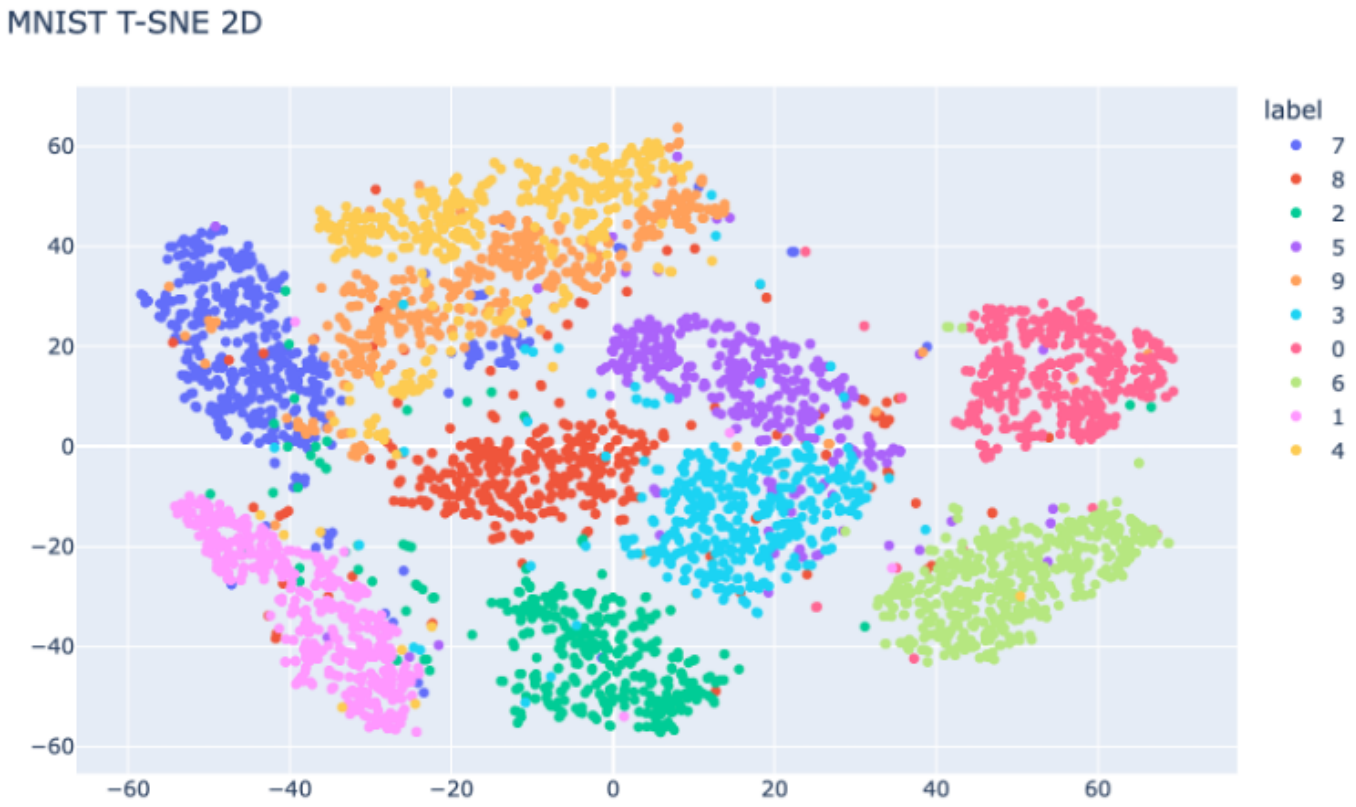


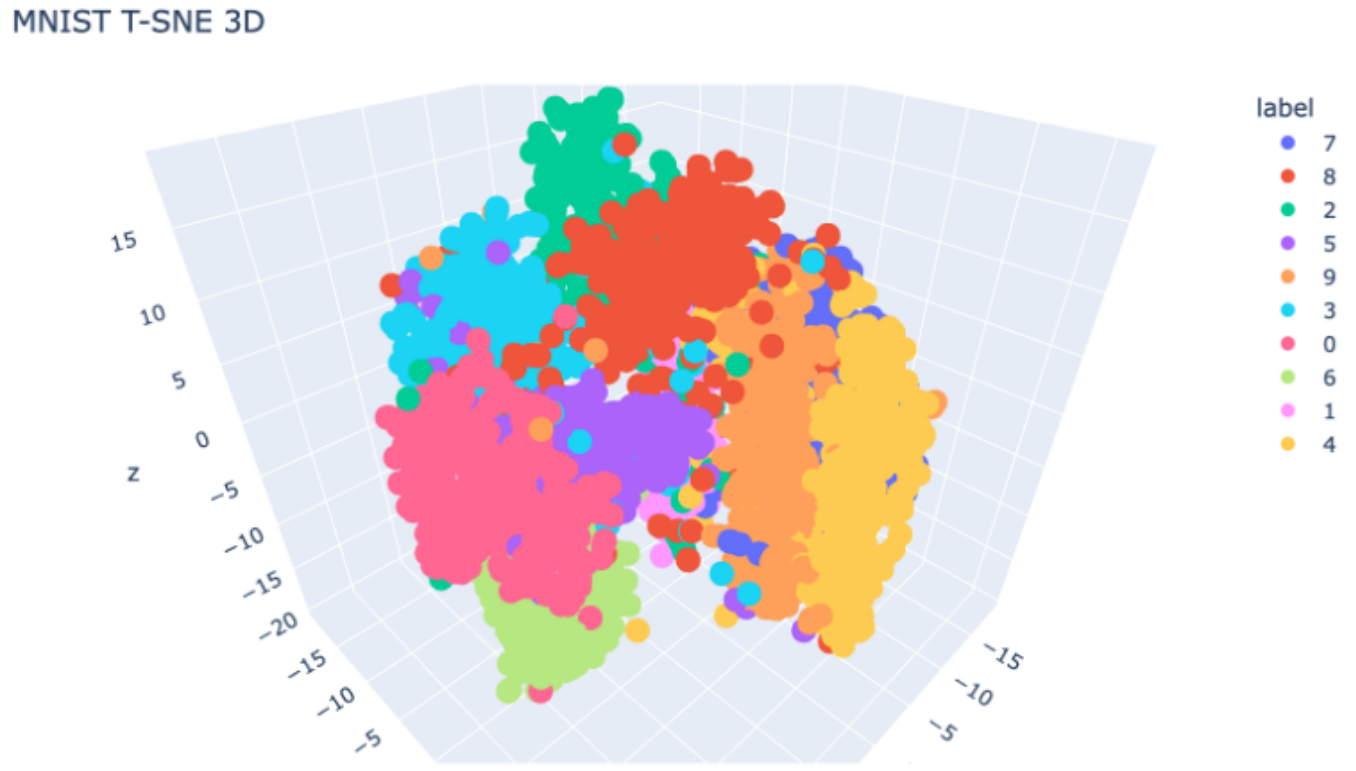
Observations:

1. The shape of each component changes when the number of components changes. This is because the NMF components decompose the images to different parts, and compose the parts from different components when recover the images.
2. There is no weights among the components, each component contains roughly similar variances
3. When there is less components, each part is big and complete. When the number of components increases, each part becomes smaller.

Dimension Reduction - T-SNE

T-SNE is the state-of-the-art method of dimension reduction. It aims to keep the points in the neighbourhood also in the reduced dimension, while it does not guarantee the relative distances between the poinds. So it is a good way to use it for dimension reduction for clustering and visualisation, but not suitable to recover the vector in the original dimension (Actually it is not possible, because the mapping back to the original dimension is not reserved)





Observations:

1. It is significantly better separated digis with both 2 and 3 dimensions from TNSE. Even though there are still some overlapping, but in general they are much separated.
2. It could be a good way to use the reduced dimensions data to do clustering, which we will explore later.

Unsupervised learning models

Now we want to try different models to see their performance, and select the best one from them.

I have tried different combinations in the second notebook, here is a summary of the result.

Learning Model	Accuracy
K-Means (10 clusters) with original features	0.58
PCA (50 components) and K-Means (10 clusters)	0.59
NMF (50 components) then K-Means (10 clusters)	0.30
T-SNE (3 components) and K-Means (10 clusters)	0.75
T-SNE (2 components) and K-Means (10 clusters)	0.83

As we can see, the T-SNE with 2 components and then perform K-Means with 10 clusters (will be tuned later) so far have the best accuracy. We will use this as a base for further improvement.

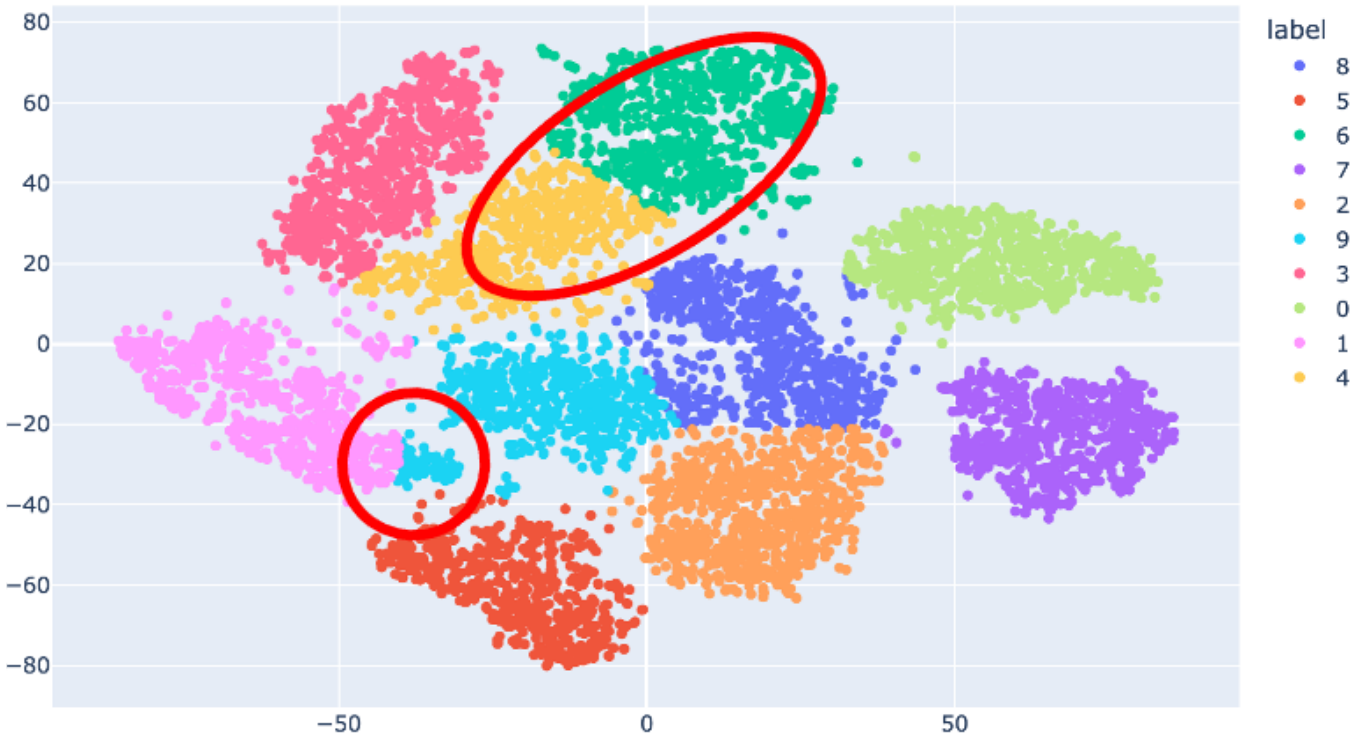
Analyze

Even though 0.83 is already a relative good result, we still want to understand where the errors are, so that we can improve it accordingly.

To do that, I plot the K-Means clusters and comparing it with the plot of the points with true label.

True clusters

K-Means clusters



Observations:

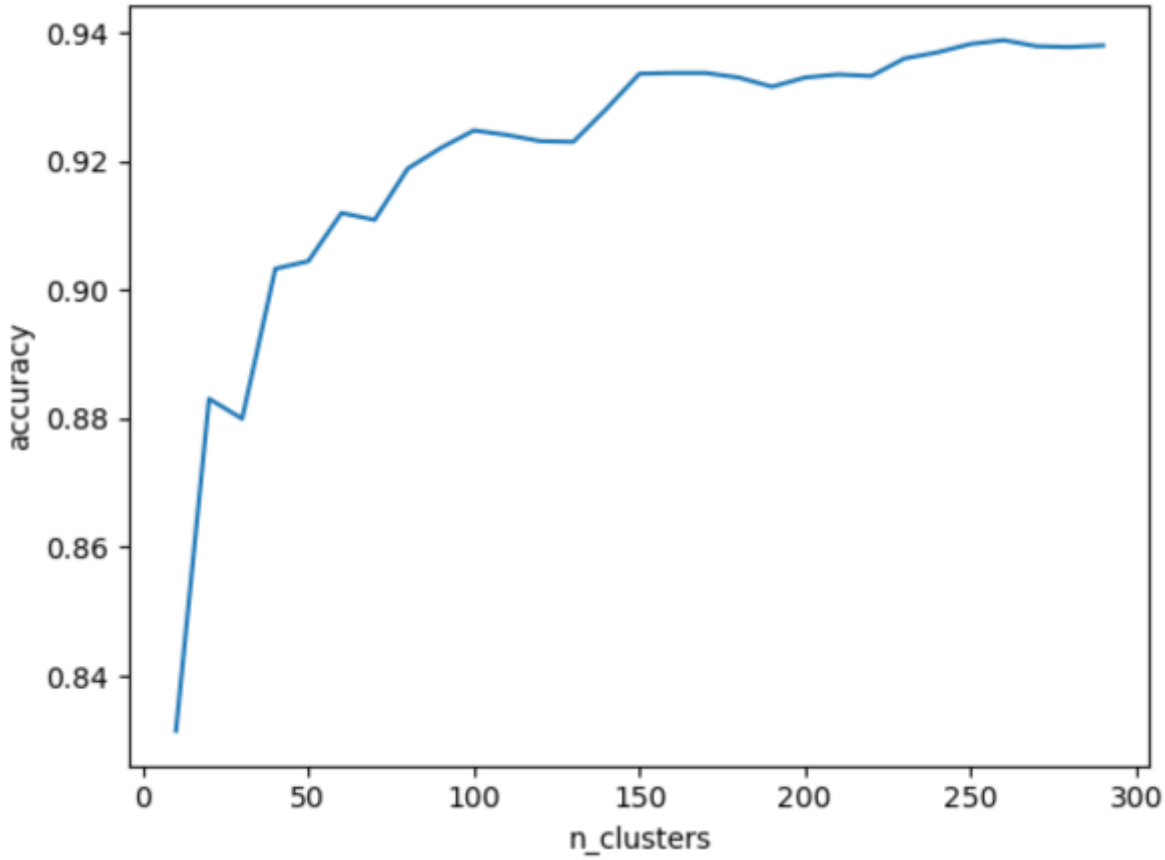
Comparing the 2 plots, We can say K-Means dis a relativly good job. How ever:

1. In top-middle part, the number 4 and 9 are overlapping each other.
2. In middle left part, the number 8 and 1 are separated well, but not separated by K-Means in the overlapping part.

Improvement 1 - increase n_clusters in K-Means

The idea is to try more clusters, that multiple cluster could be mapped to the same digit, it might be possible to keep the edges more clean.

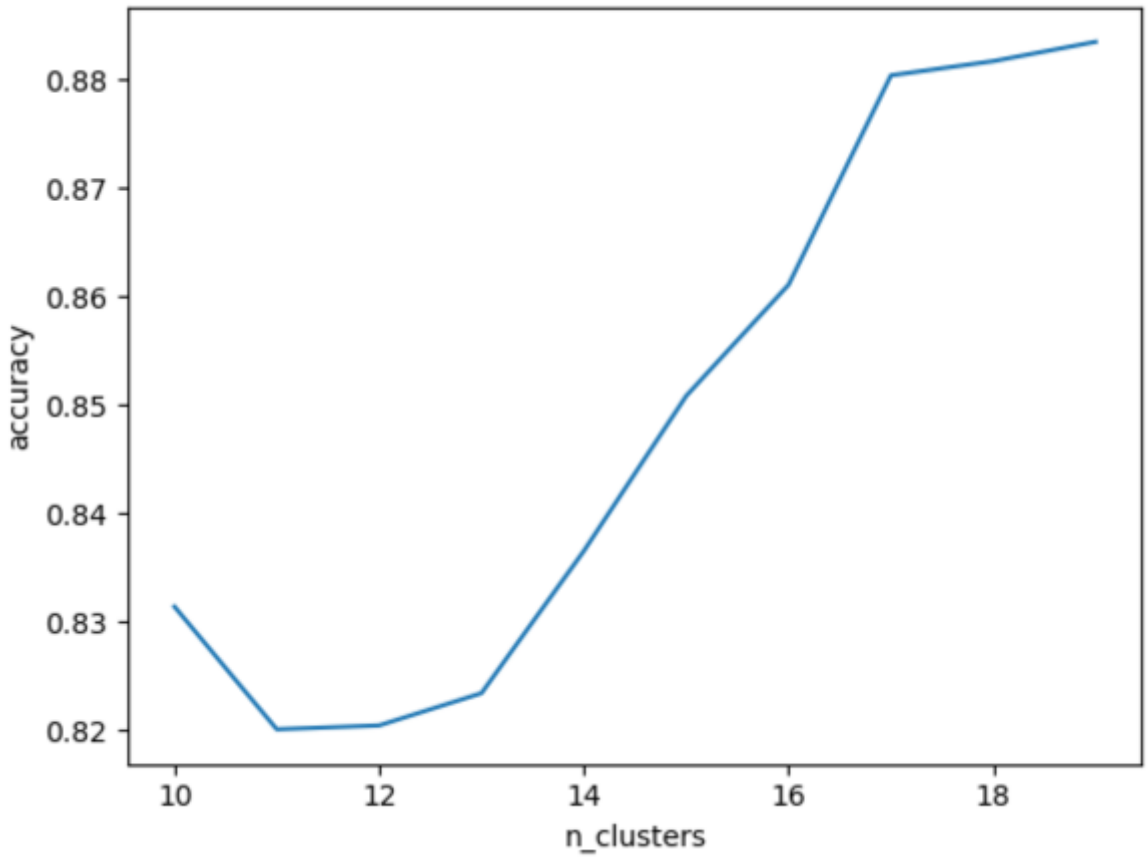
Here is a plot of the "accuracy(training)-n_cluster":



Observations

1. With increase of number of clsters, the accuracy is increasing.
2. This sounds promissing, BUT this is actually start to overfitting.
3. Even though K-Means is unsupervised learning, but the process of identifying the digits of each cluster is a kind of "labeling". When the number of clusters increase, we need to label the clusters. In a extream case, we create the same number of clusters as the number of samples, then it would be a perfect clustering, but it becomes a overfit of supervised learning.
4. So in practice, we should keep the culsters in resonablly low number (say, under 20).

Here is a plot with more fine steps under 20 clusters:



Observations

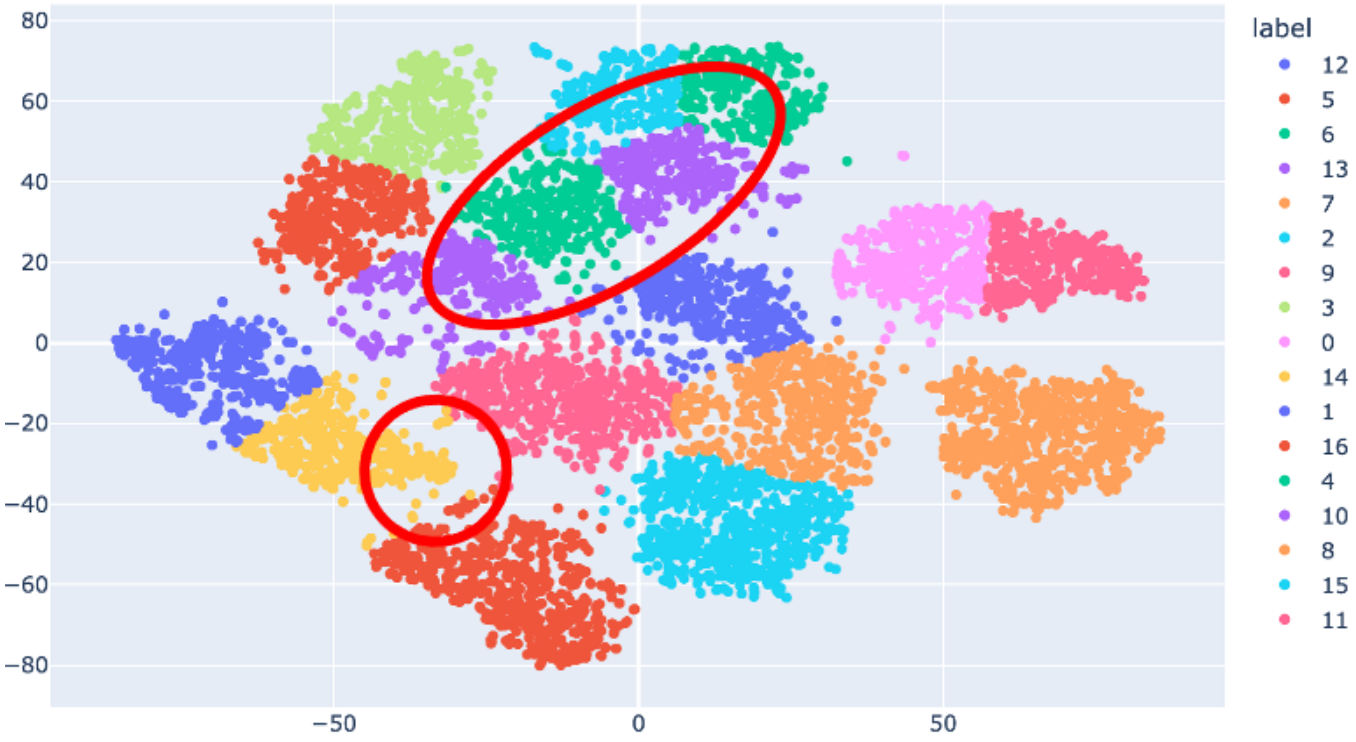
From the plot, I would say 17 is a reasonable number. with it we get 88% accuracy, but let's see how it works in test set later. We can also take a look at the plot, why with 17 clusters it works better

Result

Using n_clusters=17, we achieve an accuracy of 0.93. And we can plot the points again to see why it is improved.

True clusters

K-Means clusters:



Observations:

1. By Comparing the two plots, we can see some of the digits are are separated into 2.
2. The right end of the true cluster "1" and "8" are better clustered.
3. The mixed cluster of "4" and "9" are better separated.

Improvement 2 - Using GaussianMixture instead of K-Means

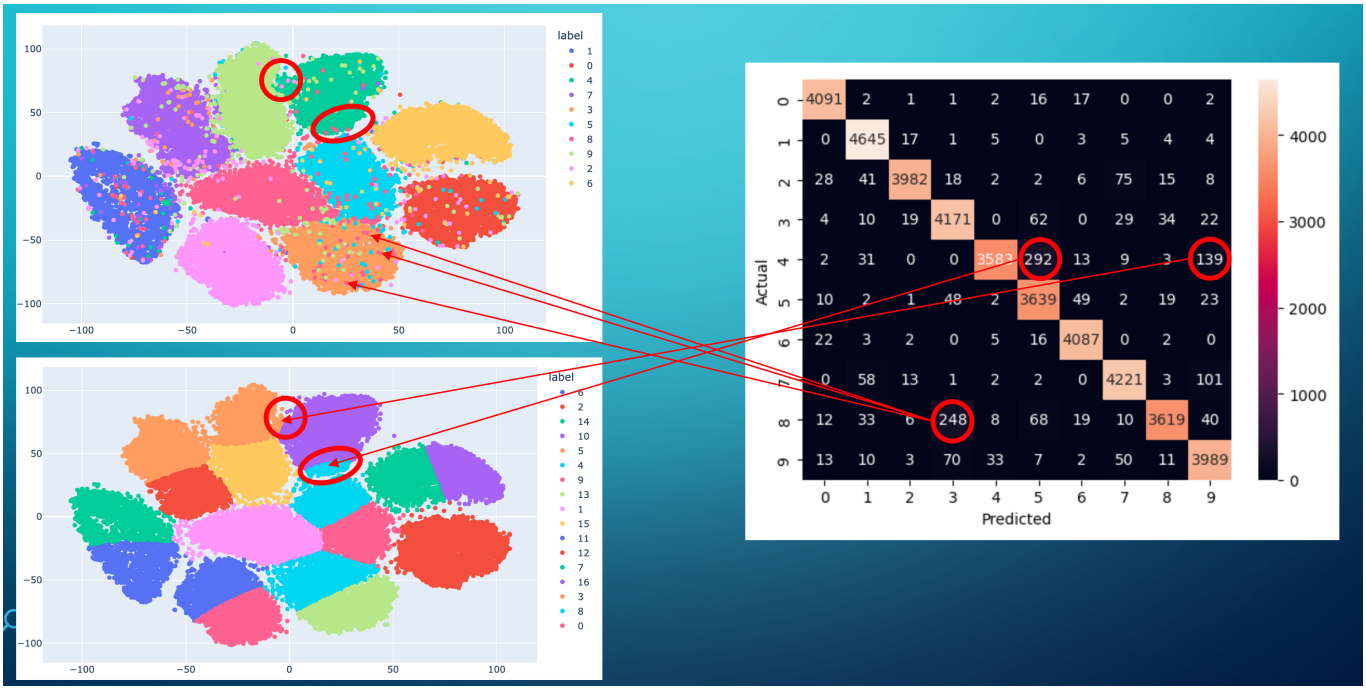
GaussianMixture by default based on the result of K-Means, and assume the culsters of the points are from Gaussian distribution centered in those K-Means centers. This has the benefit of making the edges more smooth in our case. I want to try if it can push the performance even better.

Result

With T-SNE (2 components) and GaussianMixture (17 clusters), the accuracy is improved again to 0.95. So we use that as my final model in this project.

Analyze

I ploted the true labels, GaussianMixture results and Confusion matrix, to compare them where are the rest of the errors are.



Observations:

From the confusion matrix, the main misclassifications are mainly 2 types of errors:

1. GaussianMixture does not cluster the data perfect. E.g. the missclassification of 4->9 and 4->5 belongs to this type.
2. Different digits are too close to each other in their feature space, using distance based methods can not differenciate them. This can be identified by the single dots in those clusters in the true label plot. E.g. the missclassification of 8->3, etc. belongs to this category.

Data efficiency

It sounds great that unsupervised learning can achieve an accuracy of 0.95 without much manual work.

But wait, we actually utilized the labeled data also in unsupervised learning for 1) mapping from clusters to true labels, 2) calculating accuracy for selecting hyper parameters. So even though the model itself does not require any labels, we still used them to make the life easier.

So next I want to explor how much labeled data we actually need for those tasks, and comparing it with supervised learning with same amount of data.

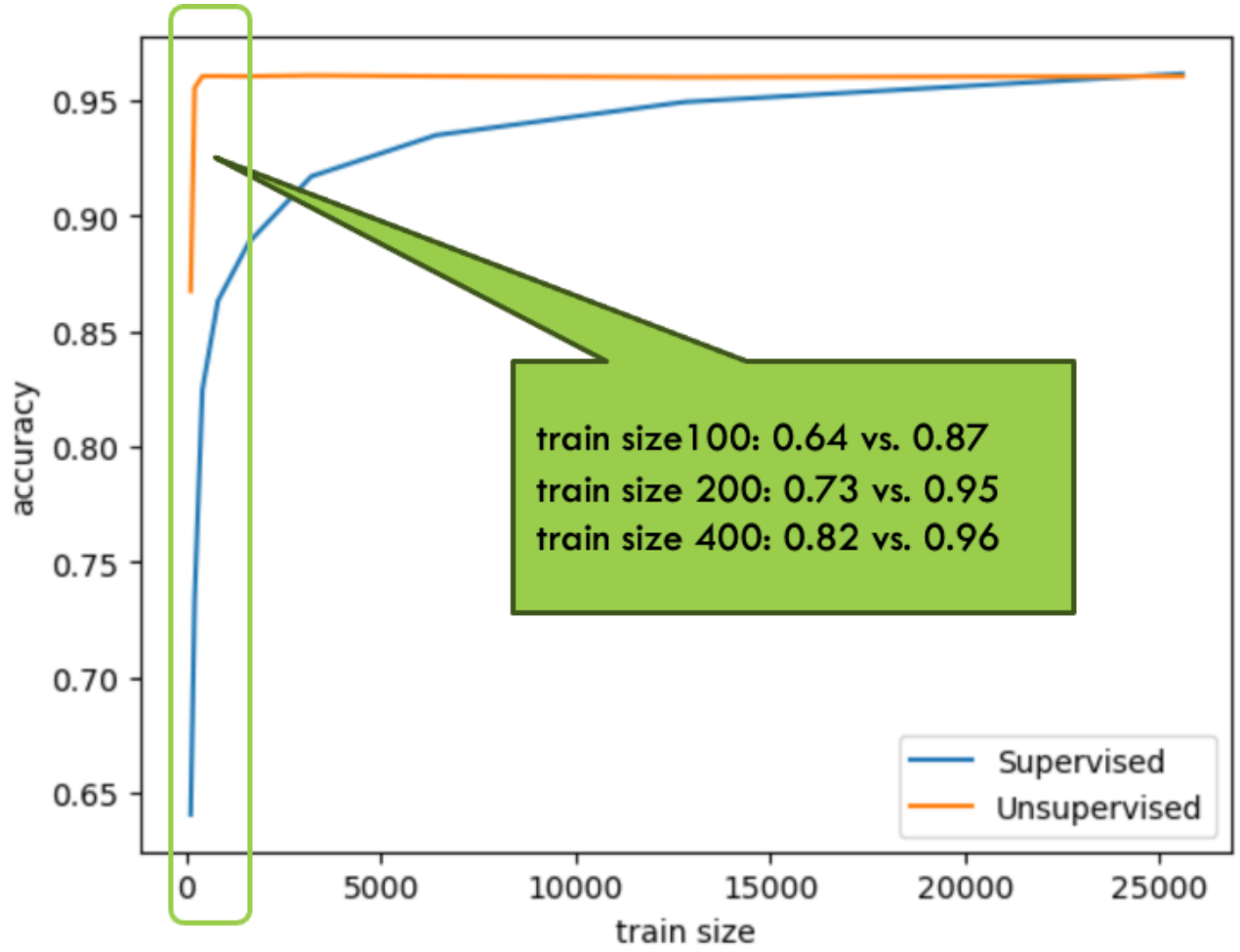
Method

We assume the scenario that we have 42k unlabeled images, and we are willing to spend time to manually label a limited number of randomly selected subset of the images. And then use those labled data to predict the label of the rest of the images.

We want to compare the performance with supervised learning and unsupervised learning in the way:

1. For supervised learning, we will use KNN and select the best k between 1 and 5
2. For unsupervised learning, we use T-SNE + GausianMixture, by selecting the n_clusters between 10 and 150. As long as the labels of the test set is not seen during training/prediction, we can use any value we want.

Result



Observations

1. As we can see, with 100 labeled data, unsupervised learning can achieve 0.87 accuracy (vs. 0.64 for supervised learning)
2. With 200 labled data, the unsupervised learning keeps about 0.95 (vs. 0.73 for supervised learning).
3. Supervised learning only over perform it starts around 25k labeled data.

Validate the conclusion

To validate if that is true in real case, I will do a experiementby:

1. Export a random set of 200 labeled data to a file in this notebook.
2. Will create a new Notebook to import only those 200 labeled data.
3. Use those data to perform both supervised and unsupervised learning, to predict the label of the test data
4. Submit those test result, and we would expect to see: about 0.95 accruacy for unsupervised learning and about 0.73 for supervised learning.

Result

Learning Model	Accuracy
Supervised (KNN with K=4)	0.737
Unsupervised (T-SNE with2 components and GassianMixture with17 clusters)	0.934

Observations

1. The submission of the result from knn has accuracy of 73.7% as expected.
2. For supervised learning, the performance does not dependent on the size of test data, but only on the size of training data.
3. The unspervised learning has a result of 0.93, which is a bit less than we'd expected.
4. I assume the reason is we have less test data (28k) than the data (42k) we used in drawing the conclusion.
5. For unsupervised learning, the size of unlabeled data is important too.

Discussion

Even though in this case with limited training data unsupervised learning have a supperisingly good performance, this can not be generalised to all the cases.

It works well in this case because:

1. The clusters are relatives well separated at the edges, if they would overlap with each other, the performance would not be that good
2. The shapes of the clusters are relatively regular, so that clustering algrithms works well
3. The number of categories are limit, so that a small number of samples can represent enough data in each category.
4. The number of samples in each category are well balanced, other wise the categories with smaller samples would not be well represented.

Without the above assumptions, we would need to use other methods to fix the issues, and the performance also might not be that good.