# COMP 5313- ARTIFICIAL INTELLIGENCE

# ASSIGNMENT 2

# Data Mining via Dimensionality Reduction

## Methodology:

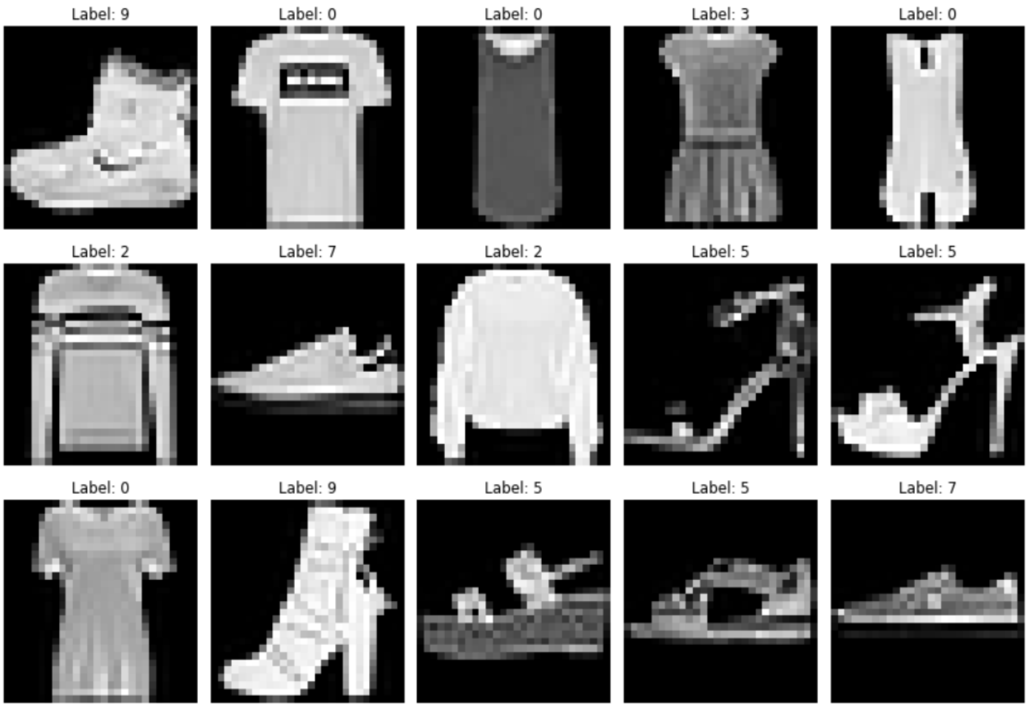
I created three .py file in order to do Data Mining via Dimensionality Reduction.

I used three methods that are PCA, t-SNE and UMAP to do dimensionality reduction.

## Dataset:

Here we used MNIST Fashion dataset. Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

This is the dataset.




## Dimensionality Reduction:

Dimensionality Reduction is a powerful technique that is widely used in data analytics, machine learning and data science to help visualize data, select good features, and to train models efficiently. We use dimensionality reduction to take higher-dimensional data and represent it in a lower dimension.

## PCA:

Principal component analysis, or PCA, is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

First, I flattened the image into one dimensional array then normalized and reshaped it before applying PCA.

```
# Creating DataFrame for train and test data
train = pd.DataFrame(x_train_pca)
train['label'] = y_train

test = pd.DataFrame(x_test_pca)
test['label'] = y_test

# Print shapes of the datasets
print("Train DataFrame shape:", train.shape)
print("Test DataFrame shape:", test.shape)
```

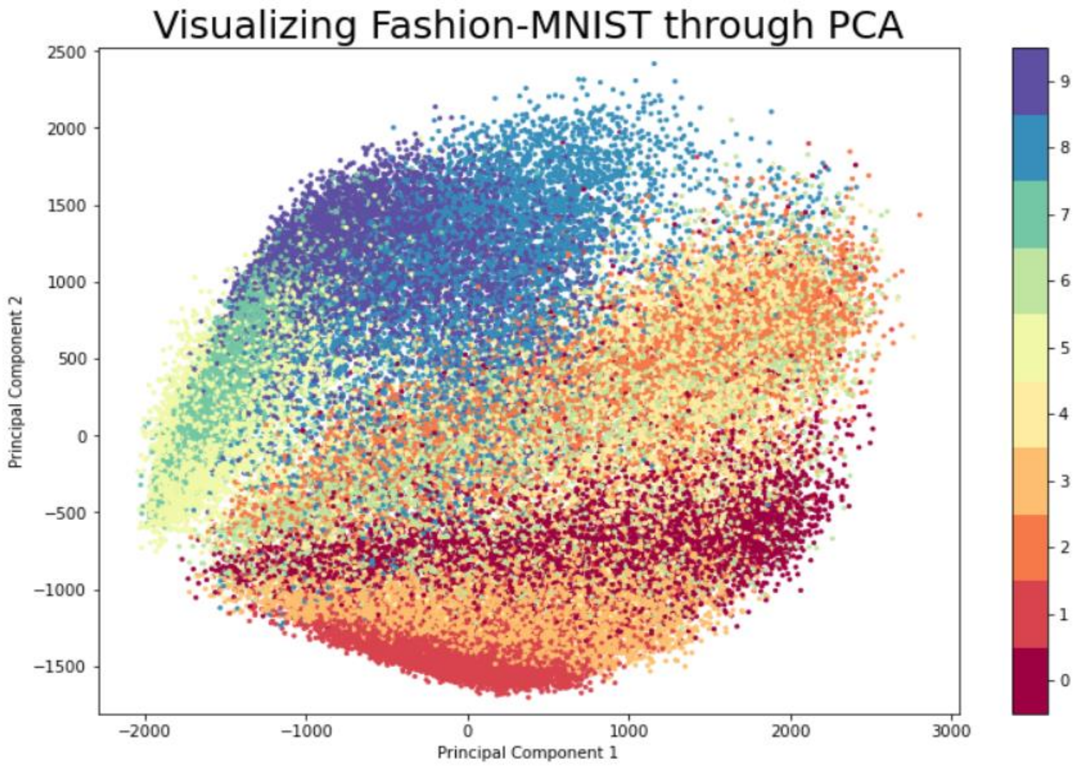
```
Train DataFrame shape: (60000, 51)
Test DataFrame shape: (10000, 51)
```

This is the dataframe shape after applying PCA.

```
# Displaying the first few rows of train DataFrame
print(train.head())
```

```
             0            1            2           3           4            5  \
0  -123.993791  1633.074396 -1211.041191  240.793118   -3.348351  -404.340455
1  1407.928853  -451.641336  -261.027034  366.436695  215.437558  1269.183187
2  -725.910795 -1101.838138   106.154242  210.031701 -105.123019   -53.417242
3    31.398664  -981.067672   202.580930  378.274376   16.283660   184.904390
4   804.119258 -1201.168720  -744.377121 -269.630116  404.982684  -150.401060

            6           7           8           9  ...         41  \
0  -91.505515  201.375258  -32.915772  -29.809373  ...  -86.447730
1 -148.350092 -224.292458 -115.631090 -229.845293  ...   81.263226
2   -2.085852   51.304638  -91.181274  -83.071415  ...  126.686722
3 -112.847785   15.280460 -344.278935   69.950293  ...  -80.660449
4  230.429128  141.440010   14.652716 -164.896822  ...   67.603761

           42          43         44          45          46         47  \
0   71.957822 -111.994070 -58.621685 -169.638475  124.868816 -29.768906
1   57.319452  113.628641  30.954408   95.326338   33.479146 -55.020765
2   56.953967  -72.160244 -64.945320   16.619006   45.028633 -14.221389
3 -110.426228 -178.700780   3.253866    7.188954  -26.138708 -58.071053
4   30.665631  124.556751 -16.643665  -23.944379    2.526679  -5.662108

          48         49  label
0  -83.062082  50.455423      9
1   53.708005 -57.874861      0
2   26.472528  28.104035      0
3   -6.223232 -61.465319      3
4  119.565354 -24.864397      0

[5 rows x 51 columns]
```

Dataframe of few rows after PCA.




Data Visualization after PCA.

Then I applied a simple SVM classifier to classify the MNIST fashion dataset.

```
# Evaluate the performance
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.86      0.84      1000
           1       0.99      0.96      0.98      1000
           2       0.78      0.79      0.79      1000
           3       0.87      0.89      0.88      1000
           4       0.79      0.81      0.80      1000
           5       0.97      0.94      0.95      1000
           6       0.69      0.65      0.67      1000
           7       0.92      0.94      0.93      1000
           8       0.97      0.97      0.97      1000
           9       0.95      0.95      0.95      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```


For, this I got an accuracy of 88%.

## t-SNE:

t-SNE (t-distributed Stochastic Neighbour Embedding) is an unsupervised non-linear dimensionality reduction technique for data exploration and visualizing high-dimensional data. Non-linear dimensionality reduction means that the algorithm allows us to separate data that cannot be separated by a straight line.

Similarly to PCA, I flattened the image into one dimensional array then normalized and reshaped it before applying t-SNE.

```
# Create DataFrame for train and test data
train = pd.DataFrame(x_train_tsne)
train['label'] = y_train

test = pd.DataFrame(x_test_tsne)
test['label'] = y_test

# Print shapes of the datasets
print("Train DataFrame shape:", train.shape)
print("Test DataFrame shape:", test.shape)
```

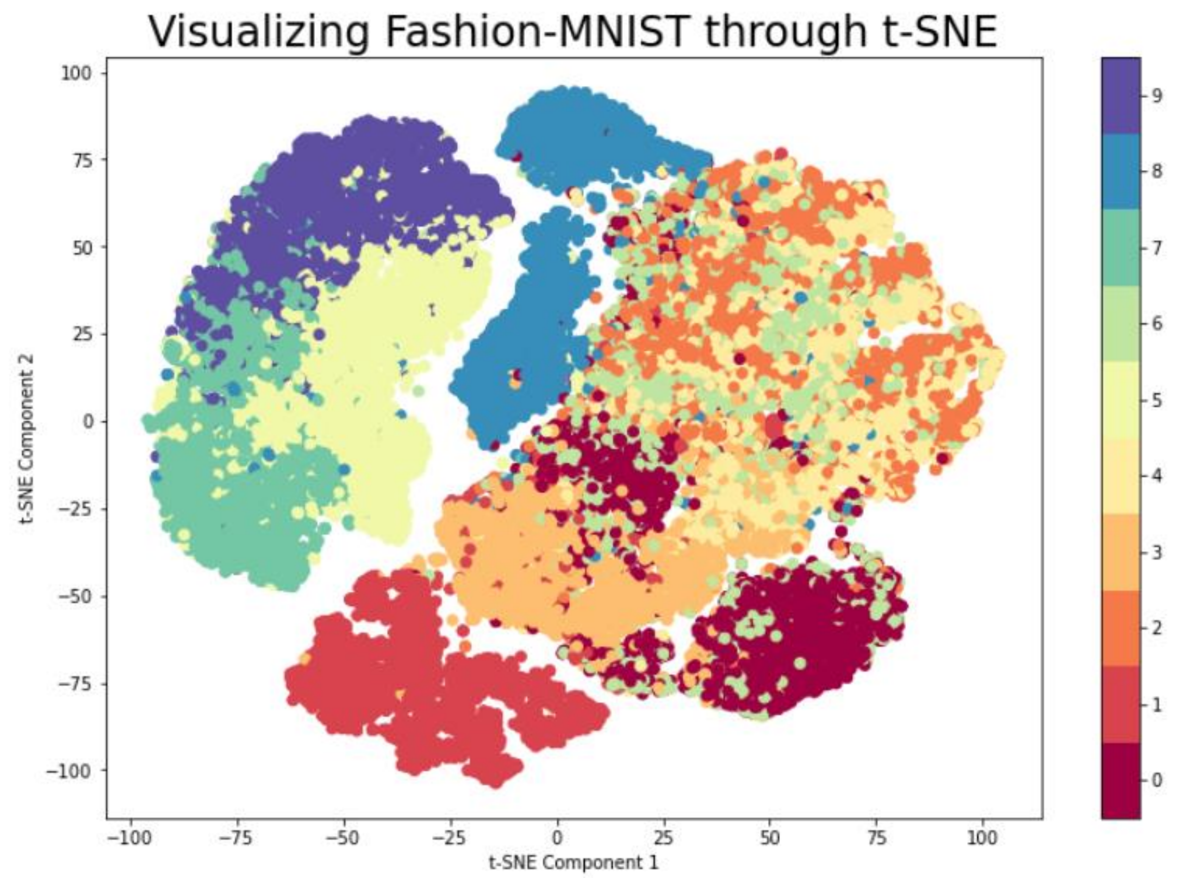
```
Train DataFrame shape: (60000, 3)
Test DataFrame shape: (10000, 3)
```

This is the dataframe shape after applying t-SNE.

```
# Display the first few rows of train DataFrame
print(train.head())
```

```
           0          1  label
0 -42.982185  73.936348      9
1  65.209343 -73.534035      0
2  -1.125311 -26.331484      0
3  10.212543 -31.360268      3
4   5.080834 -65.615913      0
```

Dataframe of few rows after t-SNE.




Data Visualization after t-SNE.

Then I applied a simple SVM classifier to classify the MNIST fashion dataset.

```python
# Evaluate the performance
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.49      0.49      1000
           1       1.00      0.59      0.74      1000
           2       0.41      0.44      0.42      1000
           3       0.41      0.69      0.52      1000
           4       0.48      0.44      0.46      1000
           5       0.50      0.81      0.62      1000
           6       0.33      0.27      0.29      1000
           7       0.86      0.68      0.76      1000
           8       0.80      0.84      0.82      1000
           9       0.92      0.46      0.61      1000

    accuracy                           0.57     10000
   macro avg       0.62      0.57      0.57     10000
weighted avg       0.62      0.57      0.57     10000
```


For, this I got an accuracy of 57%.

## UMAP:

UMAP is a new technique by McInnes et al. Uniform Manifold Approximation and Projection (UMAP) is a dimension reduction technique that can be used for visualisation similarly to t-SNE, but also for general non-linear dimension reduction. The algorithm is founded on three assumptions about the data

1. The data is uniformly distributed on Riemannian manifold;
2. The Riemannian metric is locally constant
3. The manifold is locally connected.

Here also, I flattened the image into one dimensional array then normalized and reshaped it before applying UMAP.

```
# Create DataFrame for train and test data
train = pd.DataFrame(x_train_umap)
train['label'] = y_train

test = pd.DataFrame(x_test_umap)
test['label'] = y_test

# Print shapes of the datasets
print("Train DataFrame shape:", train.shape)
print("Test DataFrame shape:", test.shape)
```

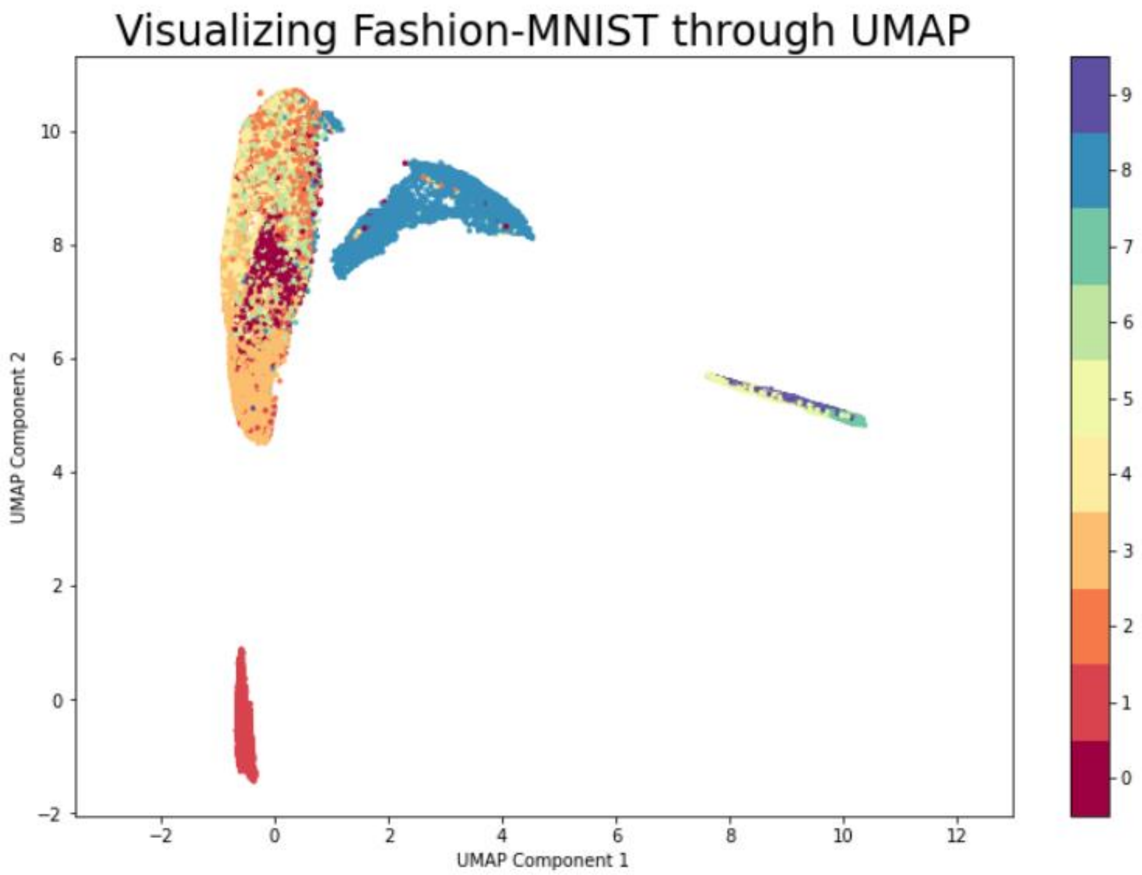
```
Train DataFrame shape: (60000, 51)
Test DataFrame shape: (10000, 51)
```

This is the dataframe shape after applying UMAP.

```
# Display the first few rows of train DataFrame
print(train.head())
```

```
           0         1         2         3         4         5         6  \
0   0.930203  4.629578  7.567747  4.337176  3.303705  4.401646  4.592235
1  10.143060  1.798088  4.262380  1.502978  6.882712  5.672943  3.115500
2  10.004306  3.413370  4.453558  3.040205  5.444238  3.964400  5.339095
3  10.217864  3.198436  4.393587  2.458318  5.913263  4.187828  5.002822
4  10.717884  4.024543  4.377642  1.729188  6.349250  3.768091  5.717562

          7         8         9  ...        41        42        43        44  \
0  4.509768  4.627343  3.981642  ...  3.928945  4.979936  3.985417  6.529401
1  5.206645  3.885250  4.095042  ...  3.880199  4.964187  3.980197  6.494970
2  5.331816  3.906265  4.716318  ...  3.907916  4.943336  3.979457  6.492531
3  5.237074  4.058956  4.491087  ...  3.911539  4.941634  3.982293  6.477735
4  5.038370  4.590066  4.229511  ...  3.911373  4.919974  3.985278  6.456738

         45        46        47        48        49  label
0  6.121536  3.751657  5.648276  6.366957  4.141622      9
1  6.115186  3.736057  5.620748  6.301548  4.133403      0
2  6.148813  3.759392  5.651603  6.372872  4.163128      0
3  6.150653  3.753586  5.656752  6.365251  4.161204      3
4  6.158533  3.755945  5.658025  6.366473  4.165531      0

[5 rows x 51 columns]
```

Dataframe of few rows after UMAP.




Data Visualization after UMAP.

Then I applied a simple SVM classifier to classify the MNIST fashion dataset.

```
# Evaluate the performance
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.87      0.76      1000
           1       0.99      0.94      0.97      1000
           2       0.56      0.62      0.59      1000
           3       0.82      0.83      0.82      1000
           4       0.55      0.59      0.57      1000
           5       0.93      0.82      0.87      1000
           6       0.49      0.32      0.39      1000
           7       0.85      0.88      0.86      1000
           8       0.96      0.89      0.93      1000
           9       0.86      0.94      0.90      1000

    accuracy                           0.77     10000
   macro avg       0.77      0.77      0.77     10000
weighted avg       0.77      0.77      0.77     10000
```


For, this I got an accuracy of 77%.

## Conclusion:

As you can see above, after using all the dimensionality reduction techniques. PCA got the higher accuracy of 88% while UMAP got 77% and t-SNE got 57% for this simple SVM model. Even though, PCA and t-SNE are widely used techniques their performance suffers with large datasets and using it correctly can be challenging. While UMAP offers a number of advantages over PCA and t-SNE.

## References:

[1] https://builtin.com/data-science/step-step-explanation-principal-component-analysis
[2] https://www.kaggle.com/code/parulpandey/part1-visualizing-kannada-mnist-with-pca?scriptVersionId=29322090
[3] https://www.datacamp.com/tutorial/introduction-t-sne
[4] https://www.kaggle.com/code/parulpandey/visualizing-kannada-mnist-with-t-sne
[5] https://umap-learn.readthedocs.io/en/latest/
[6] https://umap-learn.readthedocs.io/en/latest/auto_examples/plot_mnist_example.html
[7] https://pair-code.github.io/understanding-umap/