**TSNE**

**T-distributed Stochastic Neighbor Embedding (t-SNE)** is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions.

**What is Dimensionality Reduction?**   
Dimensionality Reduction is the technique of representing n-dimensions data(multidimensional data with many features) in 2 or 3 dimensions.  
An example of dimensionality reduction can be discussed as a classification problem i.e. student will play football or not that relies on both temperature and humidity can be collapsed into just one underlying feature, since both of the features are correlated to a high degree. Hence, we can reduce the number of features in such problems. A 3-D classification problem can be hard to visualize, whereas a 2-D one can be mapped to simple 2-dimensional space and a 1-D problem to a simple line.

**How does t-SNE works?**   
t-SNE a non-linear dimensionality reduction algorithm finds patterns in the data based on the similarity of data points with features, the similarity of points is calculated as the conditional probability that a point A would choose point B as its neighbour.   
It then tries to minimize the difference between these conditional probabilities (or similarities) in higher-dimensional and lower-dimensional space for a perfect representation of data points in lower-dimensional space.

**Space and Time Complexity:**   
The algorithm computes pairwise conditional probabilities and tries to minimize the sum of the difference of the probabilities in higher and lower dimensions. This involves a lot of calculations and computations. So the algorithm takes a lot of time and space to compute. t-SNE has a quadratic time and space complexity in the number of data points.

**Code: Python code implementing T-SNE on MNIST dataset**

Python3

*# Importing Necessary Modules.*

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.manifold** **import** TSNE

**from** **sklearn.preprocessing** **import** StandardScaler

**Code #1: Reading data**

Python3

*# Reading the data using pandas*

df = pd.read\_csv('mnist\_train.csv')

*# print first five rows of df*

print(df.head(4))

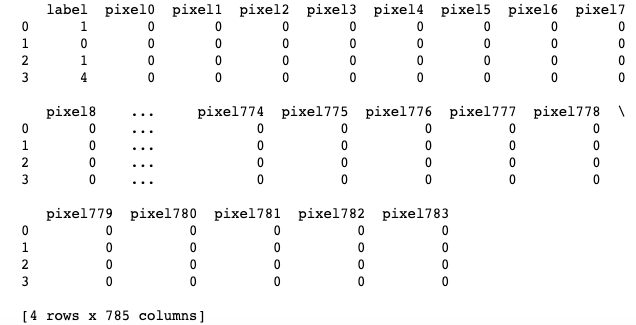
*# save the labels into a variable l.*

l = df['label']

*# Drop the label feature and store the pixel data in d.*

d = df.drop("label", axis = 1)

**Output:**



**Code #2:** Data-preprocessing

Python3

*# Data-preprocessing: Standardizing the data*

**from** **sklearn.preprocessing** **import** StandardScaler

standardized\_data = StandardScaler().fit\_transform(data)

print(standardized\_data.shape)

Python3

*# TSNE*

*# Picking the top 1000 points as TSNE*

*# takes a lot of time for 15K points*

data\_1000 = standardized\_data[0:1000, :]

labels\_1000 = labels[0:1000]

model = TSNE(n\_components = 2, random\_state = 0)

*# configuring the parameters*

*# the number of components = 2*

*# default perplexity = 30*

*# default learning rate = 200*

*# default Maximum number of iterations*

*# for the optimization = 1000*

tsne\_data = model.fit\_transform(data\_1000)

*# creating a new data frame which*

*# help us in plotting the result data*

tsne\_data = np.vstack((tsne\_data.T, labels\_1000)).T

tsne\_df = pd.DataFrame(data = tsne\_data,

columns =("Dim\_1", "Dim\_2", "label"))

*# Plotting the result of tsne*

sn.FacetGrid(tsne\_df, hue ="label", size = 6).map(

plt.scatter, 'Dim\_1', 'Dim\_2').add\_legend()

plt.show()

**Output:**

