سپیده دربان مقامی

تمرین درس یادگیری ماشین : پیاده سازی یک شبکه خودکدگذار با هدف خوشه بندی

مقدمه :

در یادگیری ماشین به طور کلی، سه فرآیند یادگیری ماشین متفاوت داریم:

1. یادگیری نظارت شده فرآیند آموزش یک مدل یادگیری ماشین بر روی مجموعه داده برچسب‌گذاری شده است. مجموعه داده ای که در آن متغیر هدف شناخته شده است. در این تکنیک، مدل به دنبال یافتن روابط بین متغیر مستقل و وابسته است. نمونه هایی از یادگیری تحت نظارت طبقه بندی، رگرسیون و پیش بینی هستند.

2. یادگیری بدون نظارت فرآیندی برای آموزش یک مدل یادگیری ماشین بر روی یک مجموعه داده است که در آن متغیر هدف شناخته شده نیست. در این تکنیک هدف مدل یافتن مرتبطترین الگوها در داده‌ها یا بخش‌های داده است. نمونه هایی از یادگیری بدون نظارت عبارتند از: خوشه بندی، تقسیم بندی، کاهش ابعاد و ...

3. یادگیری نیمه نظارت شده ترکیبی از فرآیندهای یادگیری تحت نظارت و بدون نظارت است که در آن از داده های بدون برچسب برای آموزش یک مدل نیز استفاده می شود. در این رویکرد، از ویژگی های یادگیری بدون نظارت برای یادگیری بهترین نمایش ممکن از داده ها و ویژگی های یادگیری نظارت شده برای یادگیری روابط در نمایش ها استفاده می شود که سپس برای پیش بینی استفاده می شود.

در اینجا، از رمزگذارهای خودکار برای یادگیری نمایش داده ها استفاده می کنیم، سپس یک طبقه بندی خطی ساده برای طبقه بندی مجموعه داده به کلاس های مربوطه آموزش داده می شود(دلیل استفاده از طبقه بند به دلیل وجود کلاس در مجموعه داده است)، در انتها از الگوریتم DBSCAN برای خوشه بندی استفاده میکنیم که در خروجی مشاهده میشود خروجی آن همانند الگوریتم طبقه بند خوشه بندی شده است.

1. توضیح درمورد داده انتخاب شده :

مجموعه داده انتخابی شامل تراکنش‌های انجام شده با کارت‌های اعتباری در ماه سپتامبر 2013 توسط دارندگان کارت اروپایی است. این مجموعه داده تراکنش‌های دو روز را نشان می‌دهد که در آن 492 تراکنش کلاهبرداری(Fraud) در بین 484,807 تراکنش وجود دارد.

تنها متغیرهای ورودی عددی در این مجموعه داده وجود دارند که نتیجه یک تبدیل PCA هستند که به دلیل محرمانه بودن ویژگی‌ها و حفظ حریم خصوصی نام ستون ها مشخص نشده است.

ویژگی‌های V1 تا V28 اجزاء اصلی‌اند که با استفاده از PCA به دست آمده‌اند، تنها ویژگی‌هایی که با PCA تبدیل نشده‌اند 'Time' و 'Amount' هستند.

ویژگی 'Time' : ثانیه‌های گذشته بین هر تراکنش و اولین تراکنش در مجموعه داده است.

ویژگی 'Amount' : مقدار تراکنش است، این ویژگی می‌تواند به عنوان یک ویژگی مستقل برای یادگیری حساس به هزینه مورد استفاده قرار گیرد.

ویژگی 'Class' دارای مقدار 1 در صورت وقوع کلاهبرداری و 0 در غیر این صورت دارد.

2. آماده سازی مجموعه داده :

ابتدا، تمام کتابخانه های مورد نیاز را بارگیری می کنیم و مجموعه داده را با استفاده از pandas dataframe بارگذاری می کنیم.

```
from keras.layers import Input, Dense
from keras.models import Model, Sequential
from keras import regularizers
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
from sklearn.manifold import TSNE
from sklearn import preprocessing
import matplotlib.pyplot as plt
import pandas as pd
```

```
import numpy as np
import seaborn as sns

sns.set(style="whitegrid")

# Set random seed for reproducibility
np.random.seed(203)

# Load the credit card fraud dataset from a CSV file
data = pd.read_csv("creditcard.csv")

# Convert the "Time" feature to represent hours of the day (24-hour format)
data["Time"] = data["Time"].apply(lambda x : x / 3600 % 24)

# Display the first few rows of the dataset
data.head()
```

output :

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.000000	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.000000	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	0.000278	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	0.000278	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	0.000556	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

5 rows × 31 columns

```
# Count the occurrences of each class ('0' for non-fraud, '1' for fraud)
vc = data['Class'].value_counts().to_frame().reset_index()

# Calculate the percentage of each class with respect to the total number of transactions
vc['percent'] = vc["Class"].apply(lambda x : round(100*float(x) / len(data), 2))

# Rename the columns for clarity
vc = vc.rename(columns = {"index" : "Target", "Class" : "Count"})

# Display the resulting table
vc
```

output :

	Target	Count	percent
0	0	284315	99.83
1	1	492	0.17

یکی از بزرگترین چالش ها این است که هدف به شدت نامتعادل است زیرا تنها 0.17 درصد موارد تراکنش های کلاهبرداری هستند. اما مزیت Autoencoder این است که میتواند این داده های نامتعادل را مدیریت کند.

حالا در ادامه فقط 2000 ردیف تراکنش های بدون کلاهبرداری(non fraud) را در نظر میگیریم.

```
# Select a subset of non-fraud transactions (Class 0) containing 2000 samples
```

```
non_fraud = data[data['Class'] == 0].sample(2000)

# Extract all fraud transactions (Class 1)
fraud = data[data['Class'] == 1]

# Combine the non-fraud and fraud subsets and shuffle the order
df = non_fraud.append(fraud).sample(frac=1).reset_index(drop=True)

# Extract feature matrix X by dropping the 'Class' column
X = df.drop(['Class'], axis = 1).values

# Extract the target variable Y containing class labels (0 for non-fraud, 1 for fraud)
Y = df["Class"].values
```

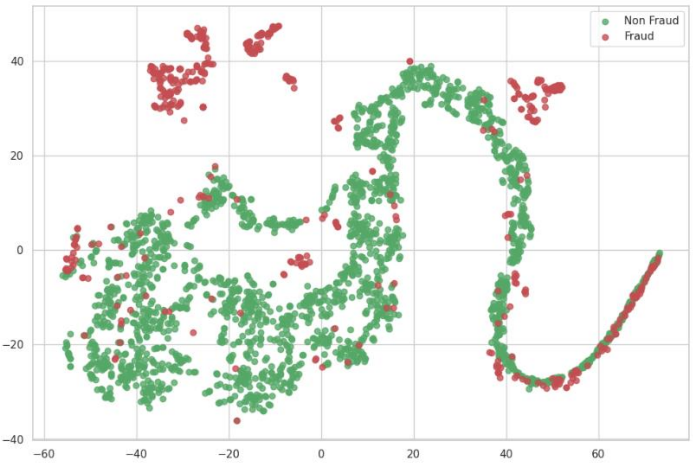
3. نمایش تراکنش های کلاهبرداری و غیر کلاهبرداری

T-SNE (t-Distributed Stochastic Neighbor Embedding) یک تکنیک تجزیه مجموعه داده است که ابعاد داده را کاهش می دهد و تنها n مؤلفه برتر را با حداکثر اطلاعات تولید می کند. این تکنیک برای تجزیه و تحلیل داده های پیچیده و چند بعدی مناسب است و معمولاً برای تجزیه و تحلیل داده های تصویری یا داده های پراکنده مورد استفاده قرار می گیرد.

در اینجا T-SNE برای تجزیه و تفسیر داده های مربوط به تراکنش ها استفاده می شود. هر نقطه در نمودار نشان دهنده یک تراکنش است، و تراکنش های غیر کلاهبرداری با رنگ سبز و تراکنش های کلاهبرداری با رنگ قرمز نشان داده می شوند.

```
# Define a function for plotting 2D t-SNE visualization
def tsne_plot(x1, y1, name="graph.png"):
    # Initialize t-SNE with 2 output dimensions and a fixed random state for reproducibility
    tsne = TSNE(n_components=2, random_state=0)
    # Transform the input features to 2D using t-SNE
    X_t = tsne.fit_transform(x1)

    # Create a figure for the plot with a specified size
    plt.figure(figsize=(12, 8))
    # Scatter plot for non-fraud transactions (Class 0) in green
    plt.scatter(X_t[np.where(y1 == 0), 0], X_t[np.where(y1 == 0), 1], marker='o', color='g', linewidth=1, alpha=0.8,
label='Non Fraud')
    # Scatter plot for fraudulent transactions (Class 1) in red
    plt.scatter(X_t[np.where(y1 == 1), 0], X_t[np.where(y1 == 1), 1], marker='o', color='r', linewidth=1, alpha=0.8,
label='Fraud')

    # Add a legend to the plot
    plt.legend(loc='best')
    # Save the plot as an image file with the specified name
    plt.savefig(name)
    # Display the plot
    plt.show()

#Call the tsne_plot function with data X ,labels Y, and save the result as"original.png"
tsne_plot(X, Y, "original.png")
```

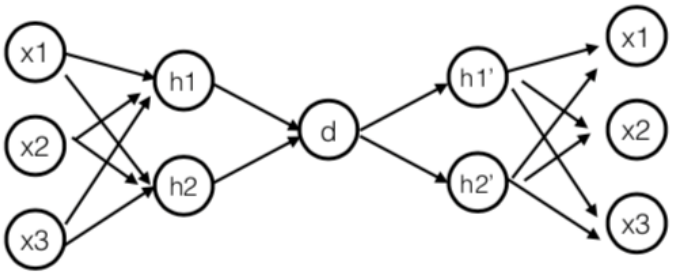
output :



در نمودار بالا می‌توان مشاهده کرد که بسیاری از تراکنش‌های غیر کلاهبرداری هستند که بسیار نزدیک به تراکنش‌های کلاهبرداری هستند، بنابراین طبقه‌بندی دقیق آن دشوار است.

4. پیاده سازی Autoencoder

توضیح درمورد Autoencoder : Autoencode معماری‌های ویژه‌ای از شبکه‌های عصبی هستند که خروجی آن‌ها همانند ورودی است. این شبکه‌ها به صورت بدون نظارت آموزش داده می‌شوند تا نمایندگی‌های بسیار پایین سطح داده ورودی را یاد بگیرند. این ویژگی‌های پایین سطح سپس به شکلی مجدد انحراف داده می‌شوند تا داده واقعی را نمایش دهند. Autoencoder یک کار رگرسیونی است که در آن از شبکه خواسته می شود تا ورودی خود را پیش بینی کند. این شبکه‌ها یک گلوگاه محکم از چند نورون در وسط دارند که آنها را مجبور می‌کند تا نمایش‌های مؤثری ایجاد کنند که ورودی را به یک کد کم‌بعد فشرده می‌کند که می‌تواند توسط رمزگشا برای بازتولید ورودی اصلی استفاده شود.



در این مدل رمزگذار خودکار که در آن فقط موارد غیر کلاهبرداری را یادمیگیرد تا از همین مدل برای تولید بازنمایی نمونه های کلاهبرداری استفاده شود و که باید با موارد غیر کلاهبرداری متفاوت باشند.

```
from keras.layers import Input, Dense
from keras.models import Model
from keras import regularizers

# Assuming X is input data and already defined

## input layer
input_layer = Input(shape=(X.shape[1],))

## encoding part
encoded = Dense(100, activation='tanh', activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoded = Dense(70, activation='relu')(encoded)
encoded = Dense(50, activation='relu')(encoded)
```

```
## decoding part
decoded = Dense(50, activation='relu')(encoded)
decoded = Dense(70, activation='tanh')(encoded)
decoded = Dense(100, activation='tanh')(decoded)

## output layer
output_layer = Dense(X.shape[1], activation='relu')(decoded)

# Create the autoencoder model
autoencoder = Model(input_layer, output_layer)

# Compile the model
autoencoder.compile(optimizer='adam', loss='mean_squared_error')

# Summary of the model to check the architecture
autoencoder.summary()
```

output :

```
Model: "model_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 30)]              0

 dense_25 (Dense)            (None, 100)               3100

 dense_26 (Dense)            (None, 70)                7070

 dense_27 (Dense)            (None, 50)                3550

 dense_29 (Dense)            (None, 70)                3570

 dense_30 (Dense)            (None, 100)               7100

 dense_31 (Dense)            (None, 30)                3030

=================================================================
Total params: 27420 (107.11 KB)
Trainable params: 27420 (107.11 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
# Define the autoencoder model using the Keras library
autoencoder = Model(input_layer, output_layer)
# Compile the autoencoder model with the specified optimizer and loss function
autoencoder.compile(optimizer="adadelta", loss="mse")

# Extract features (x) by removing the "Class" column from the dataset
x = data.drop(["Class"], axis=1)
# Extract class labels (y) from the "Class" column
y = data["Class"].values

# Scale the features using Min-Max scaler
x_scale = preprocessing.MinMaxScaler().fit_transform(x.values)
# Separate scaled data into two subsets: x_nfraud for non-fraud transactions (Class 0) and x_fraud for fraud
transactions (Class 1)
```

```
x_nfraud, x_fraud = x_scale[y == 0], x_scale[y == 1]
```

در این مدل برای یادگیری نمایش خوب به نمونه های زیادی از داده ها نیاز نداریم و فقط از 2000 ردیف موارد غیر کلاهبرداری برای آموزش رمزگذار خودکار استفاده شده.

توضیح: انتخاب نمونه های کوچک از مجموعه داده اصلی بر اساس این است که ویژگی های یک کلاس (غیر کلاهبرداری) با دیگری (کلاهبرداری) متفاوت است. برای تمایز این ویژگی‌ها، باید تنها یک دسته از داده‌ها را به رمزگذارهای خودکار نشان دهیم. این به این دلیل است که رمزگذار خودکار سعی می کند فقط یک کلاس را یاد بگیرد و کلاس دیگر را به طور خودکار متمایز کند.

در کد بالا قبل از آموزش، ابتدا مقیاس بندی را انجام دادم.

```
# Train the autoencoder model on a subset of non-fraud transactions
autoencoder.fit(x_nfraud[0:2000], x_nfraud[0:2000],
                batch_size = 256, epochs = 10,
                shuffle = True, validation_split = 0.20);
```

output :

```
Epoch 1/10
7/7 [==============================] - 0s 34ms/step - loss: 0.2296 - val_loss: 0.2293
Epoch 2/10
7/7 [==============================] - 0s 25ms/step - loss: 0.2291 - val_loss: 0.2287
Epoch 3/10
7/7 [==============================] - 0s 9ms/step - loss: 0.2285 - val_loss: 0.2282
Epoch 4/10
7/7 [==============================] - 0s 11ms/step - loss: 0.2280 - val_loss: 0.2276
Epoch 5/10
7/7 [==============================] - 0s 11ms/step - loss: 0.2275 - val_loss: 0.2271
Epoch 6/10
7/7 [==============================] - 0s 9ms/step - loss: 0.2269 - val_loss: 0.2266
Epoch 7/10
7/7 [==============================] - 0s 8ms/step - loss: 0.2264 - val_loss: 0.2260
Epoch 8/10
7/7 [==============================] - 0s 11ms/step - loss: 0.2259 - val_loss: 0.2255
Epoch 9/10
7/7 [==============================] - 0s 12ms/step - loss: 0.2253 - val_loss: 0.2250
Epoch 10/10
7/7 [==============================] - 0s 8ms/step - loss: 0.2248 - val_loss: 0.2244
```

5. بدست آوردن بازنمایی های پنهان

حالا که مدل آموزش دیده، می‌خواهیم بدونیم چه جوری اطلاعات از داده‌ها رو یاد گرفته و می‌خواهیم به این اطلاعات یا نمایندگی پنهان دسترسی پیدا کنیم. می‌توان توسط وزن‌های مدل آموزش دیده شده دسترسی پیدا کنیم.

برای این کار، یک شبکه دیگه می‌سازیم که تا لایه سوم از مدل قبلی (جایی که این اطلاعات پنهان هستند) وزن‌هاش رو اضافه می‌کنیم. این شبکه جدید به عنوان یک "مدل کدگذار" عمل می‌کنه و می‌تونه از اون اطلاعات پنهان برای داده‌های جدید هم استفاده کنه.

این مرحله به منظور ایجاد یک مدل جدید به نام `hidden_representation` صورت می‌گیرد. این مدل جدید از مدل Autoencoder تشکیل شده است و به منظور استخراج لایه‌های مخفی (hidden layers) از مدل اصلی استفاده می‌شود. این مرحله برای دسترسی به نتایج میانی ویژگی‌های مهمی که توسط لایه‌های مخفی مدل Autoencoder استخراج می‌شوند، انجام می‌شود.

استفاده از این مدل `hidden_representation` به تحلیل و بررسی ویژگی‌های مهم و نهان در داده‌ها کمک می‌کند.

```
# Create a new Sequential model for the hidden representation
hidden_representation = Sequential()
# Add the input layer of the Autoencoder model to the Sequential model
hidden_representation.add(autoencoder.layers[0])
# Add the first hidden layer of the Autoencoder model to the Sequential model
hidden_representation.add(autoencoder.layers[1])
# Add the second hidden layer of the Autoencoder model to the Sequential model
hidden_representation.add(autoencoder.layers[2])
# Add the output layer of the Autoencoder model to the Sequential model
```

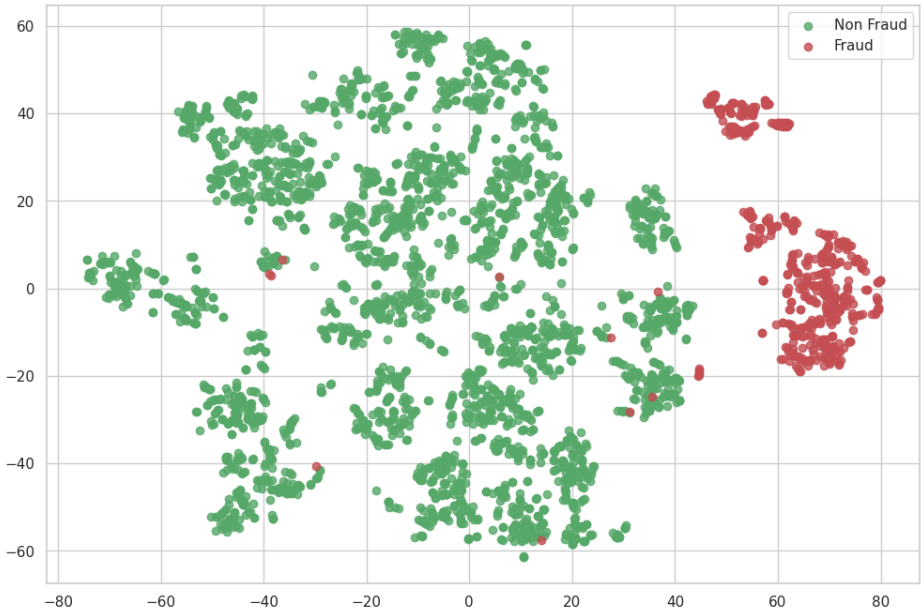
```
hidden_representation.add(autoencoder.layers[3])
```

```
# Generate hidden representations for 3000 samples of non-fraud transactions using the hidden_representation model
norm_hid_rep = hidden_representation.predict(x_nfraud[:3000])
# Generate hidden representations for all fraud transactions using the hidden_representation model
fraud_hid_rep = hidden_representation.predict(x_fraud)
```

6. نمایش بازنمایی های نهفته داده های کلاهبرداری و غیر کلاهبرداری

```
# Combine the hidden representations of non-fraud and fraud transactions
rep_x = np.append(norm_hid_rep, fraud_hid_rep, axis = 0)

# Create labels for non-fraud transactions (0) and fraud transactions (1)
y_n = np.zeros(norm_hid_rep.shape[0])
y_f = np.ones(fraud_hid_rep.shape[0])

# Combine the labels for both classes
rep_y = np.append(y_n, y_f)

# Visualize the latent representations using t-SNE and save the plot as "latent_representation.png"
tsne_plot(rep_x, rep_y, "latent_representation.png")
```

output :



حالا می‌توانیم ببینیم که تراکنش‌های کلاهبرداری و غیر کلاهبرداری کاملاً قابل مشاهده هستند و به صورت خطی قابل تفکیک هستند و برای طبقه بندی به هیچ مدل پیچیده ای نیاز نداریم، حتی از مدل های ساده تر نیز می توان برای پیش بینی استفاده کرد.

7. classifier خطی ساده:

با توجه به اینکه داده ها دارای لیبل هستن میتوان از طبقه بند برای دسته بندی استفاده کنیم.

```
# Split the latent representations and labels into training and validation sets
train_x, val_x, train_y, val_y = train_test_split(rep_x, rep_y, test_size=0.25)
```

```
# Create a Logistic Regression model using the lbfgs solver and train it on the training set
clf = LogisticRegression(solver="lbfgs").fit(train_x, train_y)
# Predict the labels for the validation set using the trained Logistic Regression model
pred_y = clf.predict(val_x)

# Print the Classification Report showing precision, recall, and other metrics
print ("")
print ("Classification Report: ")
print (classification_report(val_y, pred_y))

# Print the Accuracy Score of the Logistic Regression model on the validation set
print ("")
print ("Accuracy Score: ", accuracy_score(val_y, pred_y))
```

output :

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

         0.0       0.98      1.00      0.99       753
         1.0       1.00      0.84      0.91       120

    accuracy                           0.98       873
   macro avg       0.99      0.92      0.95       873
weighted avg       0.98      0.98      0.98       873


Accuracy Score:  0.9782359679266895
```

7. الگوریتم خوشه‌بندی DBSCAN :

DBSCAN مخفف "Density-Based Spatial Clustering of Applications with Noise" است و یک الگوریتم خوشه‌بندی مبتنی بر چگالی است. این الگوریتم به ازای هر نقطه در فضا، چگالی نقاط اطراف آن را محاسبه کرده و نقاطی که دارای چگالی کافی برای تشکیل یک خوشه هستند، به عنوان یک خوشه شناخته می‌شوند. همچنین، نقاطی که نمی‌توانند به هیچ خوشه‌ای تعلق یابند به عنوان نویز در نظر گرفته می‌شوند.

پارامترهای مهم در DBSCAN عبارتند از:

- eps: شعاع چگالی که مشخص می‌کند نقاط در این فاصله یا کمتر از هم به عنوان همسایگی در نظر گرفته می‌شوند.

- min_samples: تعداد حداقل نقاط مورد نیاز برای تشکیل یک خوشه.

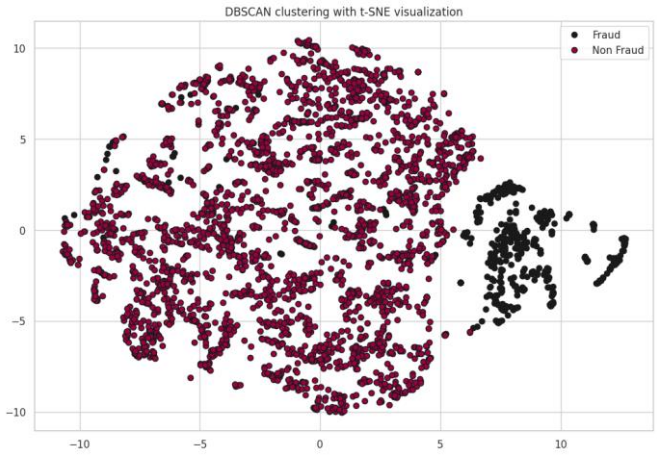
DBSCAN برای تشخیص خوشه‌های با اشکال و اندازه‌های متفاوت مناسب است و مقاوم به نویز نیز می‌باشد.

برای ارزیابی خوشه‌بندی، می‌توان از معیار Silhouette Score استفاده کنیم. این معیار بر اساس میزان جداپذیری و همبستگی داخلی خوشه‌ها ارزیابی می‌کند.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from sklearn.cluster import DBSCAN
from sklearn.model_selection import train_test_split
from sklearn.metrics import silhouette_score
# Assuming rep_x is complete dataset
# Splitting the dataset for the purpose of demonstration, even though we'll only use rep_x for DBSCAN and t-SNE
train_x, val_x, train_y, val_y = train_test_split(rep_x, rep_y, test_size=0.25)
```

```
# Combining train_x and val_x for clustering (assuming rep_x is already full dataset)
X = np.vstack((train_x, val_x))
Y = np.hstack((train_y, val_y))
# Perform DBSCAN clustering
dbscan = DBSCAN(eps=0.1, min_samples=300).fit(X)
labels = dbscan.labels_

# Run t-SNE to reduce dimensionality for visualization
tsne = TSNE(n_components=2, perplexity=30, n_iter=300)
X_tsne = tsne.fit_transform(X)

# Plotting
plt.figure(figsize=(12, 8))
unique_labels = np.unique(labels)
for label in unique_labels:
    if label == -1:
        # Black used for noise.
        col = 'k'
    else:
        col = plt.cm.Spectral(float(label) / len(unique_labels))
    class_member_mask = (labels == label)
    xy = X_tsne[class_member_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=col, markeredgecolor='k', markersize=6, label='Non Fraud' if label !=
-1 else 'Fraud')

plt.title('DBSCAN clustering with t-SNE visualization')
plt.legend(loc='best')
plt.show()

y_predict = labels * -1
# Print the Accuracy Score
print ("")
print ("Accuracy Score: ", accuracy_score(Y, y_predict))
```

output :

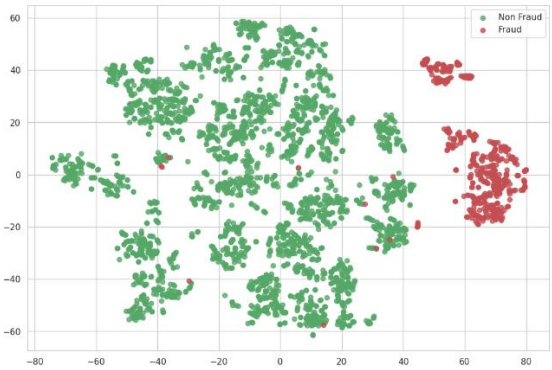


Accuracy Score: 0.9842497136311569

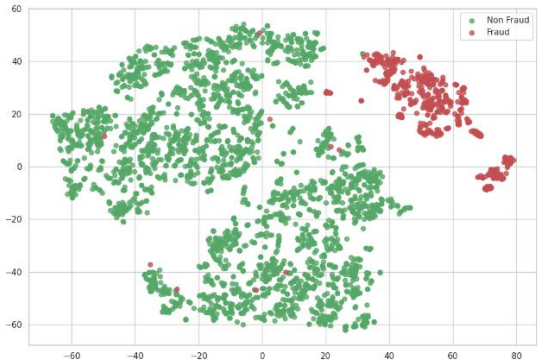
تغییر لایه های AutoEncoder به :

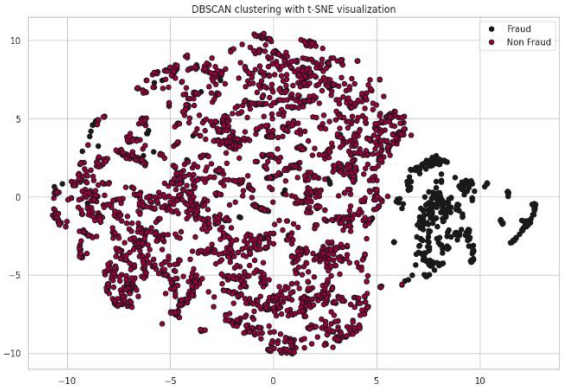
Layer with 100 - 70 - 50 / 70 - 100

```
## encoding part
encoded = Dense(100, activation='tanh', activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoded = Dense(70, activation='relu')(encoded)
encoded = Dense(50, activation='relu')(encoded)

## decoding part
# decoded = Dense(50, activation='relu')(encoded)
decoded = Dense(70, activation='tanh')(encoded)
decoded = Dense(100, activation='tanh')(decoded)
```

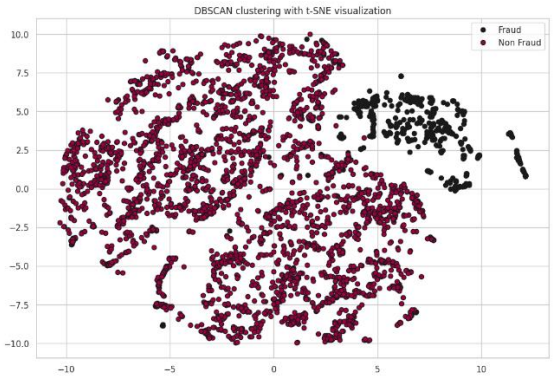


=>





=>

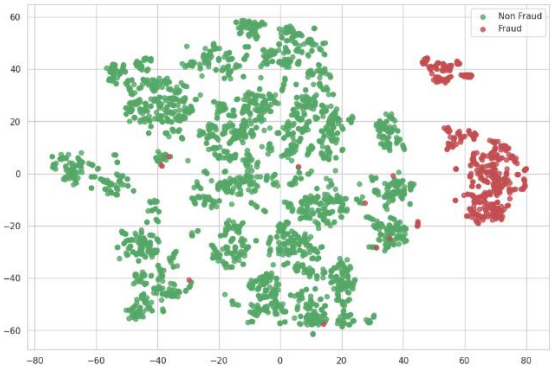



```
Accuracy Score:  0.9842497136311569
```

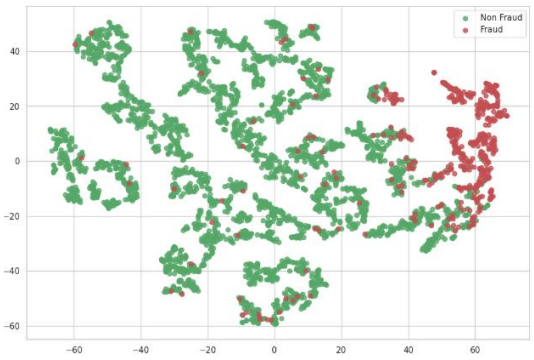
```
Accuracy Score:  0.9916953035509737
```

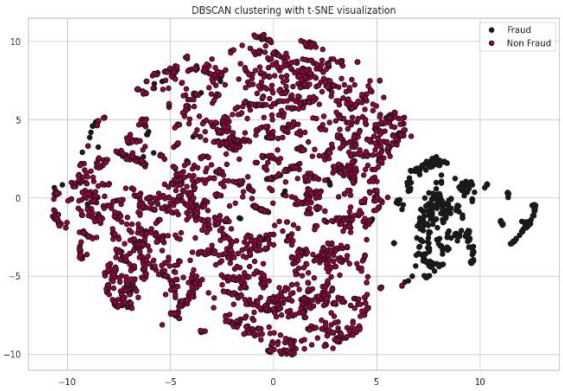
Layer with 30 - 8 - 2 / 2 - 8 – 30

```
# encoding part
encoded = Dense(30, activation='tanh', activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoded = Dense(8, activation='relu')(encoded)
encoded = Dense(2, activation='relu')(encoded)

# decoding part
decoded = Dense(2, activation='relu')(encoded)
decoded = Dense(8, activation='tanh')(encoded)
decoded = Dense(30, activation='tanh')(decoded)
```

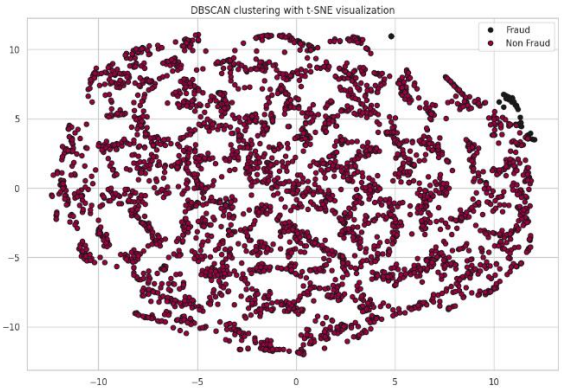


=>





=>



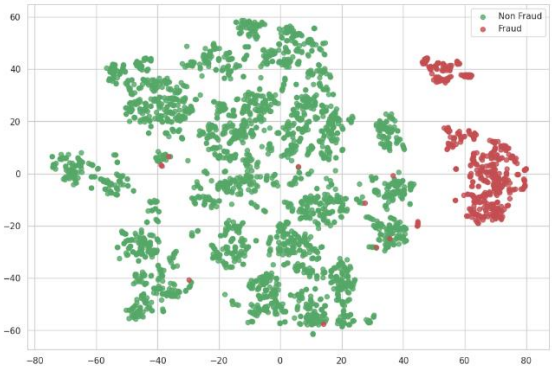

```
Accuracy Score:  0.9842497136311569
```

```
Accuracy Score:  0.8748568155784651
```

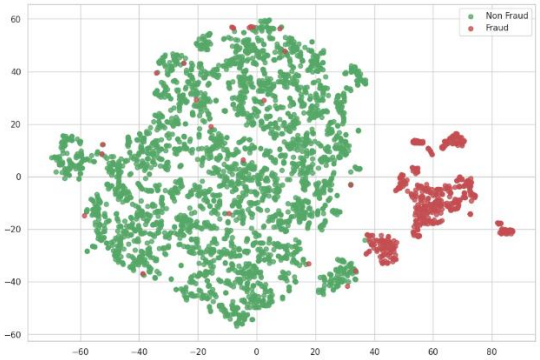
Layer with 30 - 10 / 10 – 30

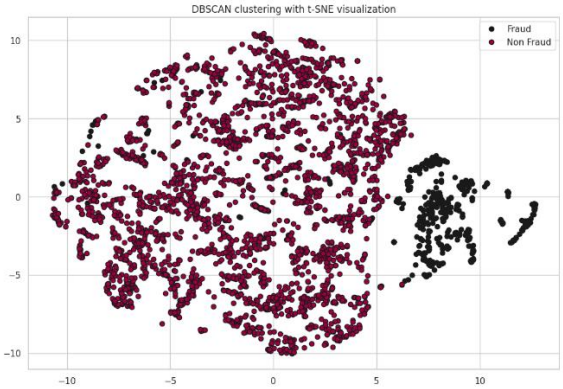
```
# encoding part
encoded = Dense(30, activation='tanh', activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoded = Dense(10, activation='relu')(encoded)

# decoding part
decoded = Dense(10, activation='tanh')(encoded)
decoded = Dense(30, activation='tanh')(decoded)
```

```
# Create a new Sequential model for the hidden representation
hidden_representation = Sequential()
# Add the input layer of the Autoencoder model to the Sequential model
hidden_representation.add(autoencoder.layers[0])
# Add the first hidden layer of the Autoencoder model to the Sequential model
hidden_representation.add(autoencoder.layers[1])
# Add the second hidden layer of the Autoencoder model to the Sequential model
hidden_representation.add(autoencoder.layers[2])
# Add the output layer of the Autoencoder model to the Sequential model
hidden_representation.add(autoencoder.layers[3])
```

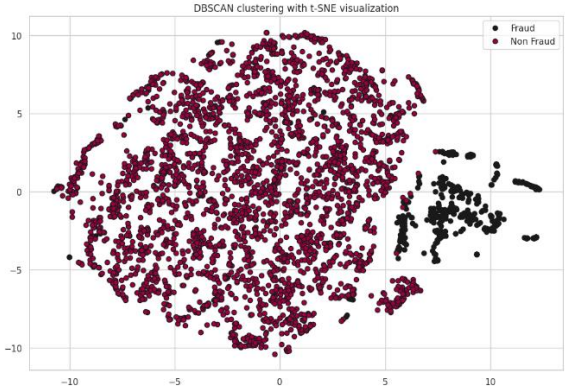


=>





=>




```
Accuracy Score:  0.9842497136311569
```

```
Accuracy Score:  0.9891179839633448
```

دلیل بزرگ کردن ابعاد :

در واقع، این تبدیل معمولاً با استفاده از وزن‌ها و توابع فعال‌سازی اعمال می‌شود. به طور ساده‌تر، هر نورون در این لایه به یک ترکیب خطی از ویژگی‌های ورودی (هر کدام از 30 ویژگی) با وزن‌های مخصوص خود، توسط تابع فعال‌سازی تبدیل می‌شود.

اثرات مختلف زیاد کردن تعداد نورون‌ها در لایهٔ مخفی:

1.استخراج ویژگی پیچیده تر: افزایش تعداد نورون‌ها می‌تواند به شبکه امکان استخراج ویژگی‌های پیچیده‌تر از داده‌ها را بدهد. این امکان می‌تواند در حل مسائل پیچیده با ساختارهای داده‌ای پیچیده مفید باشد.

2. رفع مشکل کمبود داده: در مواقعی که تعداد نمونه‌های آموزشی کم است، افزایش ابعاد بردار نهان ممکن است به مدل کمک کند که ویژگی‌های خود را از داده‌های موجود بهتر استخراج کند.

تابع TSNE :

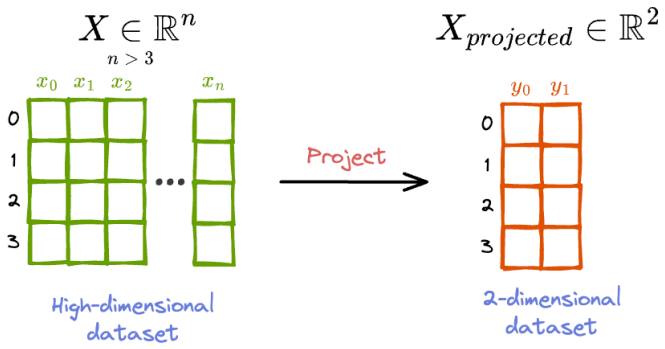
t-SNE (t-distributed Stochastic Neighbor Embedding) یک ابزار برای نمایش داده‌ها با ابعاد بالا است. تابع هزینهt-SNE ، که مقدار مینیممی ندارد و ناصاف (non-convex) است، یعنی با شروع‌های مختلف ممکن است نتایج متفاوتی به دست آید.

t-SNE یک چیز به نام کاهش ابعاد غیرخطی است. این به این معناست که این الگوریتم به ما امکان می‌دهد داده‌ها را جدا کنیم که نمی‌توانیم آنها را با هیچ خطی ساده جدا کنیم.

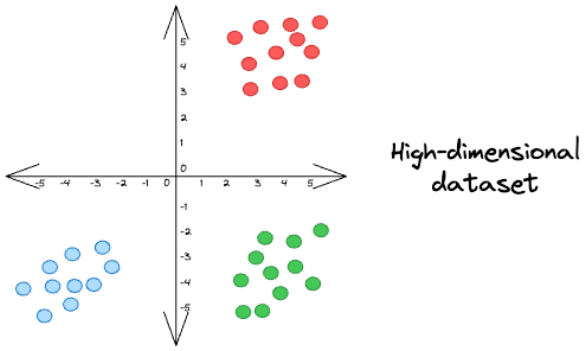
PCA از ماتریس کوواریانس کلی برای کاهش ابعاد استفاده می‌کند ومی‌توان آن ماتریس را به یک مجموعه داده جدید با همان نتیجه اعمال کنید. این کار مفید است زمانی که می‌خواهیم تعداد ویژگی‌های خود را کاهش دهیم و ماتریس از داده‌های آموزشی ایجاد شده را مجدداً استفاده کنیم. اما t-SNE اصولاً برای درک داده‌های با بُعد بالا استفاده می‌شود.

در t-SNE، محور عمودی و افقی در نمودار نشان‌دهنده ابعاد کاهش یافته فضای نهان (low-dimensional space) هستند. این مختصات جدید توسط t-SNE بر اساس فواصل نسبی بین نقاط در فضای اصلی (با ابعاد بالا) محاسبه می‌شوند.

به طوری که نمایش با ابعاد پایین‌تر تا حد امکان ساختار محلی و جهانی را در مجموعه داده اصلی حفظ کند.



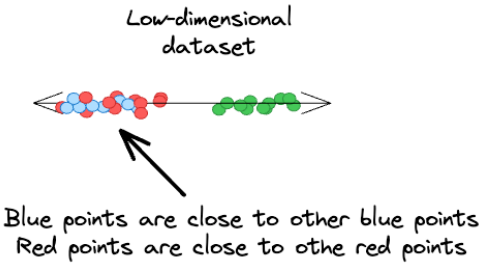
منظور ما از ساختار محلی و جهانی چیست؟



ساختار محلی، همانطور که از نام آن پیداست، به چیدمان نقاط داده ای اطلاق می شود که در فضای با ابعاد بالا به یکدیگر نزدیک هستند.

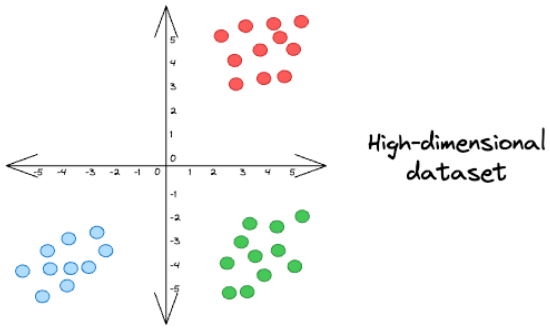
بنابراین، حفظ ساختار محلی به این معنی است که: نقاط قرمز باید به نقاط قرمز دیگر نزدیکتر باشند. نقاط آبی باید به سایر نقاط آبی نزدیکتر باشند. نقاط سبز باید به سایر نقاط سبز نزدیکتر باشند.

اما اگر بخواهیم صرفاً بر حفظ ساختار محلی تمرکز کنیم، ممکن است به وضعیتی منجر شود که نقاط آبی در واقع نزدیک‌تر به یکدیگر بمانند، اما با نقاط قرمز همپوشانی دارند، همانطور که در زیر نشان داده شده است:

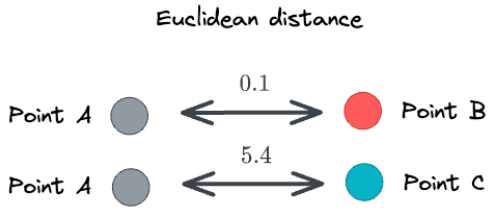


این خوب نیست. در عوض، ما همچنین می‌خواهیم پیش‌بینی‌های کم‌بعدی ساختار جهانی را به تصویر بکشند. بنابراین، حفظ ساختار جهانی به این معنی است که: خوشه قرمز به خوبی از خوشه دیگر جدا شده است. خوشه آبی به خوبی از خوشه دیگر جدا شده است. خوشه سبز به خوبی از خوشه دیگر جدا شده است.

بطور خلاصه : حفظ ساختار محلی به معنای حفظ روابط بین نقاط داده نزدیک در هر خوشه است. حفظ ساختار جهانی مستلزم حفظ روندها و روابط گسترده تر است که در همه خوشه ها اعمال می شود.



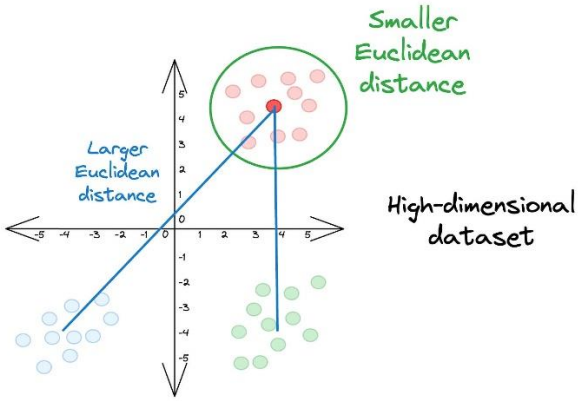
فاصله اقلیدسی معیار خوبی برای دانستن اینکه آیا دو نقطه به هم نزدیک هستند یا خیر است.



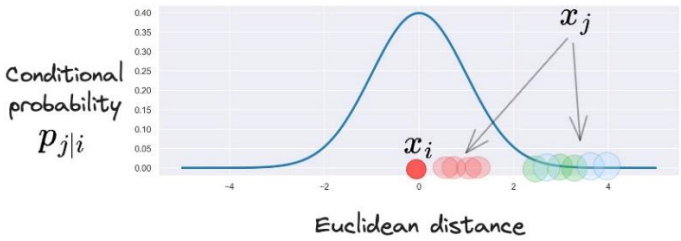
به عنوان مثال، در شکل بالا، به راحتی می توان فهمید که نقطه A و نقطه B به یکدیگر نزدیک هستند اما نقطه C نسبتاً از نقطه A فاصله بیشتری دارد.

بنابراین، اولین مرحله از الگوریتم SNE تبدیل این فواصل اقلیدسی با ابعاد بالا بین نقاط داده به احتمالات شرطی است که نشان دهنده شباهت ها هستند.

در اینجا، نقاطی که به نقطه مشخص شده نزدیکتر هستند، فواصل اقلیدسی کمتری خواهند داشت، در حالی که سایر نقاط دورتر، فواصل اقلیدسی بزرگتری خواهند داشت.



بنابراین، برای هر نقطه داده (i)، الگوریتم SNE ابتدا این فواصل اقلیدسی با ابعاد بالا را به احتمالات شرطی pj|i تبدیل می‌کند. فرض می شود که این احتمال شرطی متناسب با چگالی احتمال یک گاوسی با مرکز xi باشد.



از توزیع گاوسی فوق با مرکزیت xi مشهود است که: برای نقاط نزدیک به xi، pj|i نسبتاً زیاد خواهد بود. برای نقاط دور از xi، pj|i کوچک خواهد بود.

بنابراین، به طور خلاصه، برای یک نقطه داده xi، ما فواصل اقلیدسی آن را به تمام نقاط دیگر xj به احتمالات شرطی pj|i تبدیل می کنیم.

بر اساس آنچه تاکنون بحث کردیم، احتمالات مشروط pj|i ممکن است با استفاده از تابع چگالی احتمال گاوسی به صورت زیر محاسبه شود:

$$p_{j|i} = \frac{p_{j|i}}{\sum_{k \neq i} p_{k|i}}$$

$$= \frac{exp(-\frac{||x_i - x_j||^2}{2\sigma_i^2})}{\sum_{k \neq i} exp(-\frac{||x_i - x_k||^2}{2\sigma_i^2})}$$

بنابراین، برای هر نقطه داده xi ∈ Rn، همتای آن yi ∈ R2 را تعریف می کنیم

- به طور خلاصه :

محاسبه‌ی شباهت بین نقاط: برای هر جفت نقطه در داده‌ها، یک مقیاس شباهت محاسبه می‌شود.

کاهش بعد: از الگوریتمی برای کاهش بعد داده‌ها به دو بعد استفاده می‌شود.

محاسبه‌ی امتیاز:t-SNE : برای هر نقطه در فضای دو بعدی، یک امتیاز t-SNE محاسبه می‌شود.

نمایش نقاط: هر نقطه در فضای دو بعدی با توجه به امتیاز t-SNE خود در نمودار نمایش داده می‌شود.

از آنجایی که t-SNE یک الگوریتم احتمالی است. نتایج t-SNE می‌توانند با هر بار اجرا کمی متفاوت باشند.

محور افقی و عمودی موقعیت در بعد کاهش یافته هست.