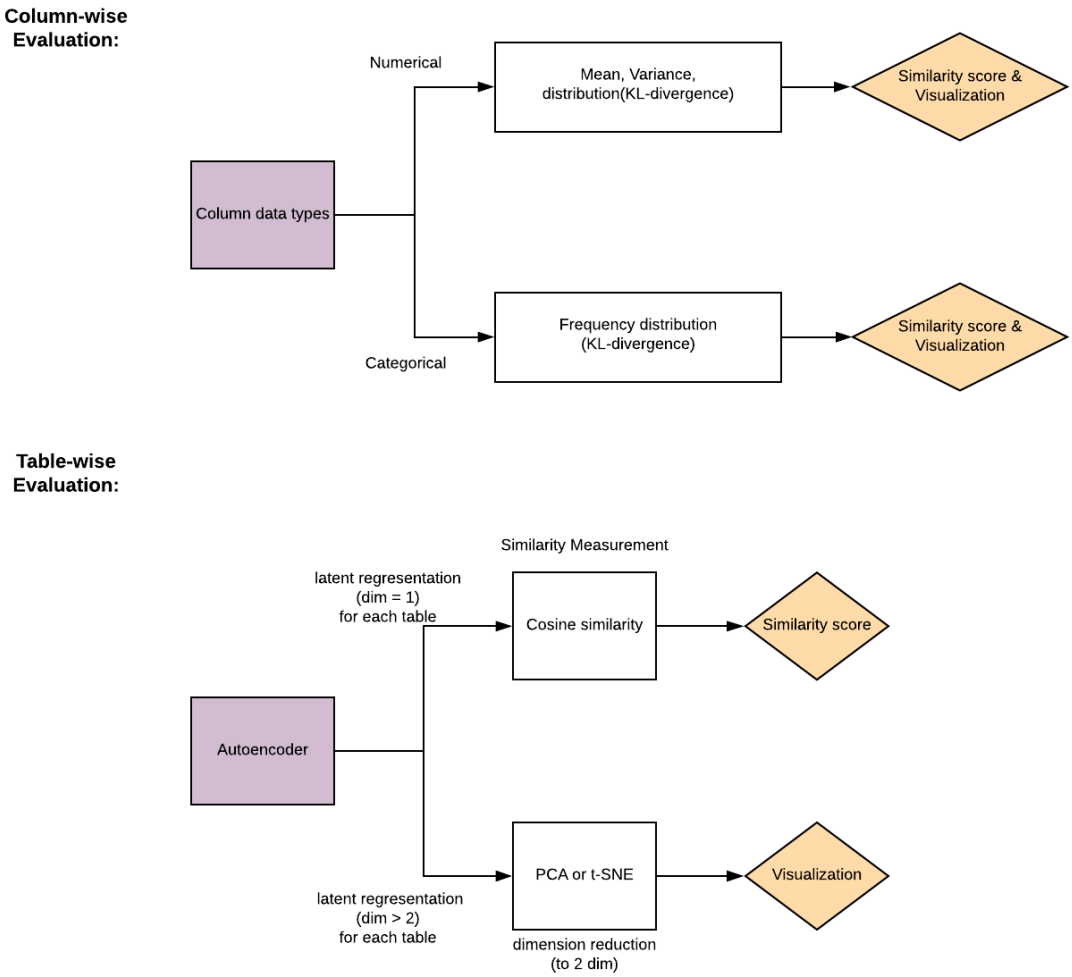
Statistical Similarity Methodology

To calculate the similarity between two tables, our methodology transfers the problem into calculating how different the synthetic data generated by GAN algorithm is from the original data. The smaller the difference between two data sets, the more similar they are with each other. We will check the similarity between the synthetic data and original data in two perspectives - the corresponding columns and the relationships between all columns in each tables. The diagram of the metrics is shown below:



Dataset Explanation

Original datasets:

Data set	# of rows	# of columns	Column types	Column names
POC dataset	382 rows	7 columns	1 continuous ("los"), 6 categorical	'subject_id', 'insurance', 'ethnicity', 'gender', 'expire_flag', 'los', 'hospital_expire_flag'
MIMIC 3 data - Use case 1: Length of Stay Prediction	116354 rows	27 columns	1 continuous ("los"), 26 categorical	'final_id', 'ADMISSION_TYPE', 'INSURANCE', 'ETHNICITY', 'HOSPITAL_EXPIRE_FLAG', 'GENDER', 'EXPIRE_FLAG', 'LOS', 'certain conditions originating in the perinatal period', 'complications of pregnancy, childbirth, and the puerperium', 'congenital anomalies', 'diseases of the blood and blood-forming organs', 'diseases of the circulatory system', 'diseases of the digestive system', 'diseases of the genitourinary system', 'diseases of the musculoskeletal system and connective tissue', 'diseases of the nervous system', 'diseases of the respiratory system', 'diseases of the sense organs', 'diseases of the skin and subcutaneous tissue', 'endocrine, nutritional and metabolic diseases, and immunity disorders', 'external causes of injury and supplemental classification', 'infectious and parasitic diseases', 'injury and poisoning', 'mental disorders', 'neoplasms', 'symptoms, signs, and ill-defined conditions'
MIMIC 3 data - Use case 2: Mortality Prediction	58976 rows	19 columns	12 continuous, 7 categorical	SUBJECT_ID', 'HADM_ID', 'ADMISSION_TYPE', 'MARITAL_STATUS', 'ETHNICITY', 'HOSPITAL_EXPIRE_FLAG', 'GENDER', 'NUMCALLOUT', 'NUMCPTEVENTS', 'NUMDIAGNOSIS', 'NUMOUTEVENTS', 'NUMRX', 'NUMPROCEVENTS', 'NUMMICROLABEVENTS', 'NUMPROC', 'NUMTRANSFERS', 'NUMINPUTEVENTS', 'NUMLABEVENTS', 'NUMNOTEVENTS'

Similarity Results with GAN Algorithms Overview:

Dataset Type	GAN algorithms	Table Similarity
POC Data (382 rows originally)	t-GAN (400 rows)	
	ct-GAN (382 rows)	
	t-GAN (10,000 rows)	
	ct-GAN	
MIMIC 3 data - Use case 1: Length of Stay Prediction (116,354 rows originally)	t-GAN (116,400 rows)	
	ct-GAN (rows)	
	t-GAN (rows)	
	ct-GAN	
MIMIC 3 data - Use case 2: Mortality Prediction (rows originally)	t-GAN (rows)	
	ct-GAN (rows)	
	t-GAN (rows)	
	ct-GAN	

1. Column-wise Similarity Evaluation

Distribution Similarity:

Quantifying the difference between two distributions can be referred to calculating the statistical distance between them. However this can be challenging as it can be difficult to interpret the measure. Instead, we can use a more common measurement, *Kullback-Leibler Divergence* (KL-divergence), to measure the distribution similarity between the pair-columns in both original table and synthetic table. KL-divergence quantifies how much information is lost when we compare the synthetic column distribution with original one. More specifically, the KL-divergence is just a slight modification of our formula for entropy and essentially, in our case, is the expectation of the log difference between the probability of data in the original column distribution(p) with the synthetic column distribution(q). The formula is below:

For continuous probability distribution:

$$KL(P||Q) = \int p(x) log(\frac{p(x)}{q(x)}) dx$$

For discrete probability distribution:

$$KL(P||Q) = \sum p_i(x) log(\frac{p_i(x)}{q_i(x)})$$

KL divergence is not symmetric, which means that how different two distributions compared to each other is usually not the same. In this case, KL divergence would be used to calculate how different the synthetic data generated is from the original data. The intuition for KL divergence score is that when the probability for an event from p is large, but the probability for the same event in q is small, then it would result in a large divergence.

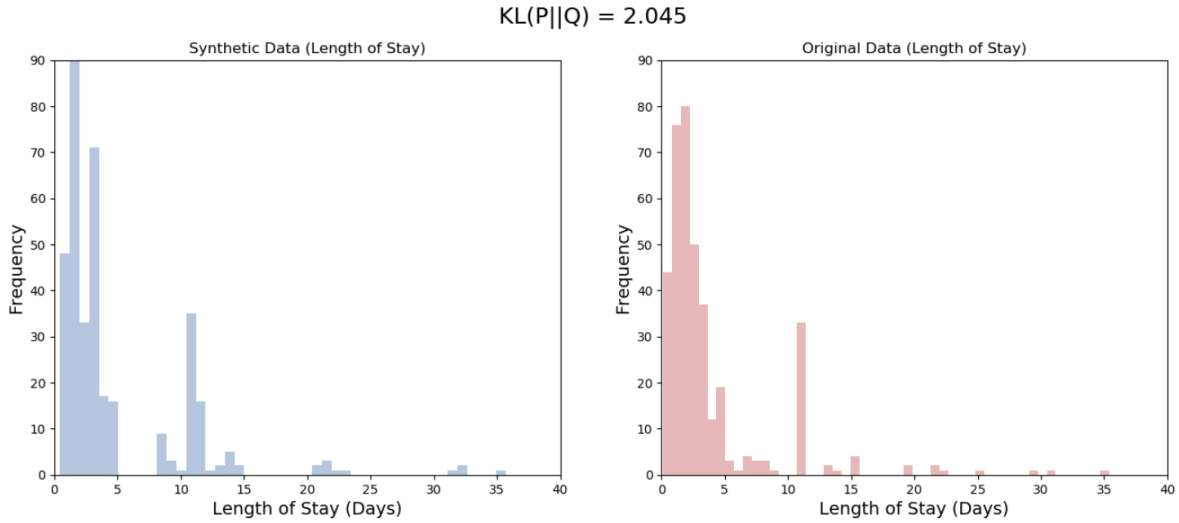
KL divergence can be used to measure continuous or discrete probability distributions, where in the latter case the integral of the events is calculated in stead of the sum of the probabilities of the discrete events.

Limitation:

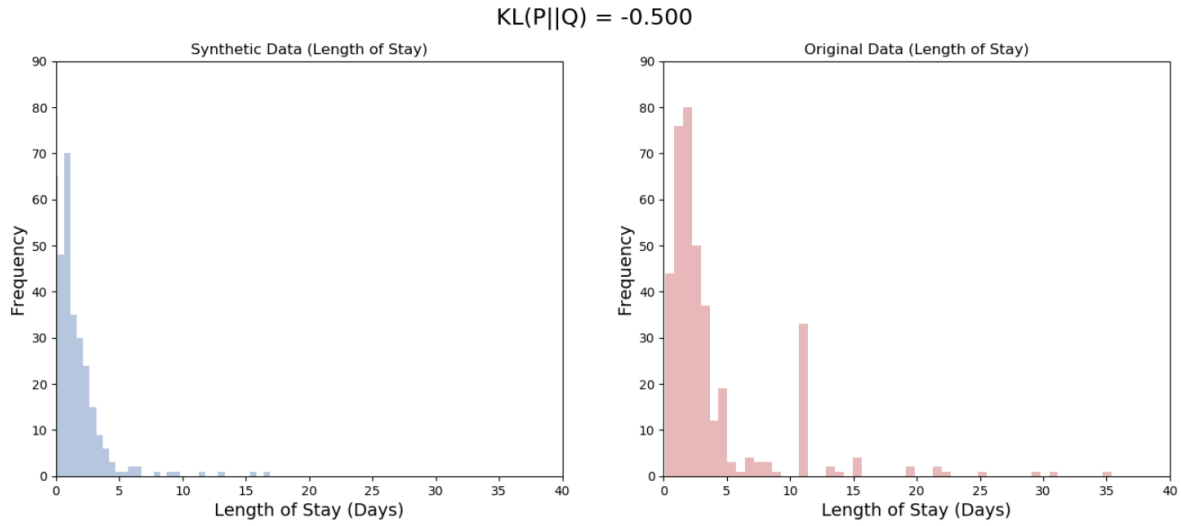
Even though KL divergence is a good measurement for common cases, it is restricted to be used on two variables that have the same length. But obtaining a synthetic table that has a different number of rows than the original one happens, especially on tGAN.

Demo:

KL-divergence score and distributions of 'Length of Stay' pair-columns on dataset generated by tGAN:



KL-divergence score and distributions of 'Length of Stay' pair-columns on dataset generated by ctGAN:



2. Table-wise Similarity Evaluation

2.1 Supervised Similarity Measure

Instead of comparing manually-combined feature data, we can reduce the feature data to representations to calculate or visualize the similarity of two tables.

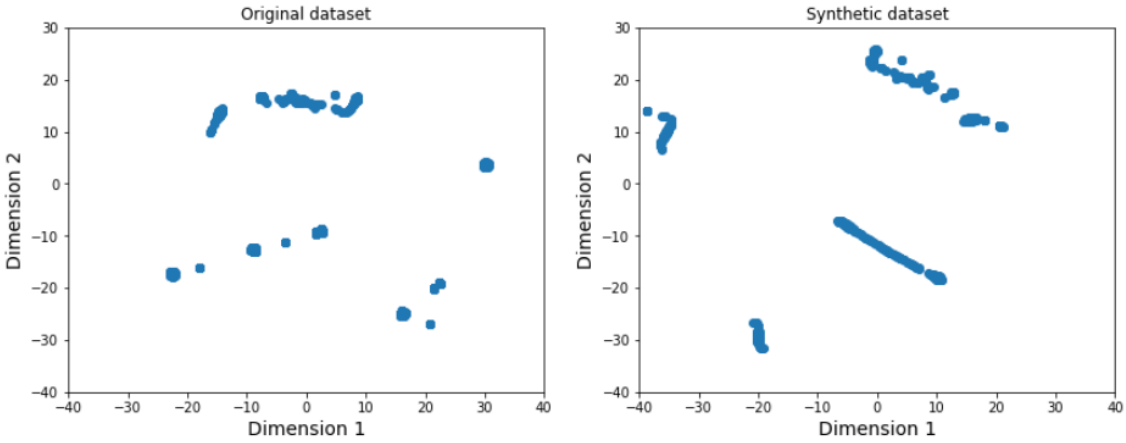
2.1.1 PCA and t-SNE

For quick visualization, we can use PCA or t-SNE to generate a lower dimensional data and then visualize them to compare.

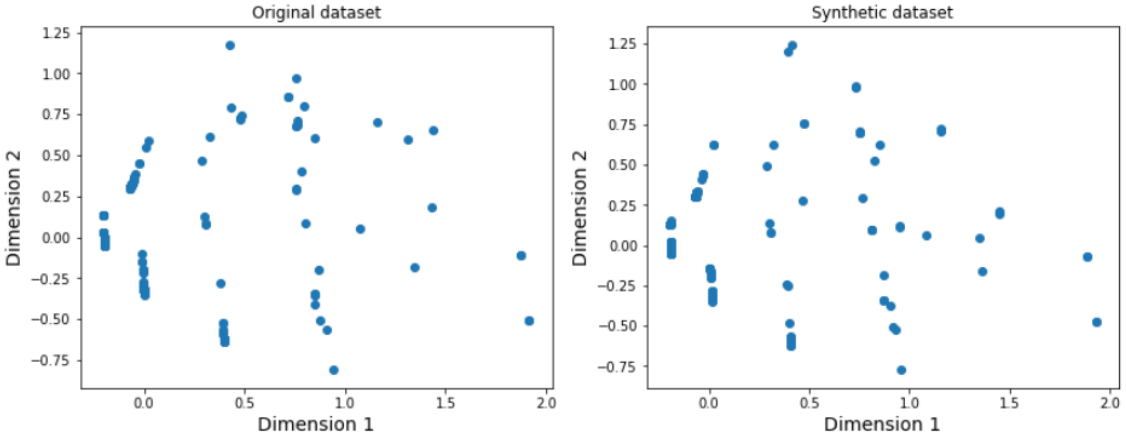
We propose using *Auto-encoder* to first compress the original data to a certain dimensional latent representation. We will define the number of dimensions the latent representation based on whether we want to get a similarity score or visualization result of both tables. To get a similarity score, we will extract 1-dimensional latent representation for both tables and then apply similarity measurements, such as *Cosine Similarity* and *Euclidean distance*, to output the similarity score. To generate a visualization for eyeballing comparison of both tables, we will set a number for the dimension of the latent representation in auto-encoder that is higher than 2, and then use *PCA* or *t-SNE* to reduce the dimensions to only 2 for scatter plotting. Using *Autoencoder* to compress a high-dimensional dataset to a lower dimension makes using *t-SNE* for visualization feasible, since it requires low-dimensional input in terms of computation. In order to make sure both synthetic table and original table are transformed exactly in the same process, we will train the auto-encoder using the original dataset and then predict on the synthetic dataset using the model built. The same logic is applied for *PCA* and *t-SNE* transformation, where we use the same transformation function for both datasets.

Demo:

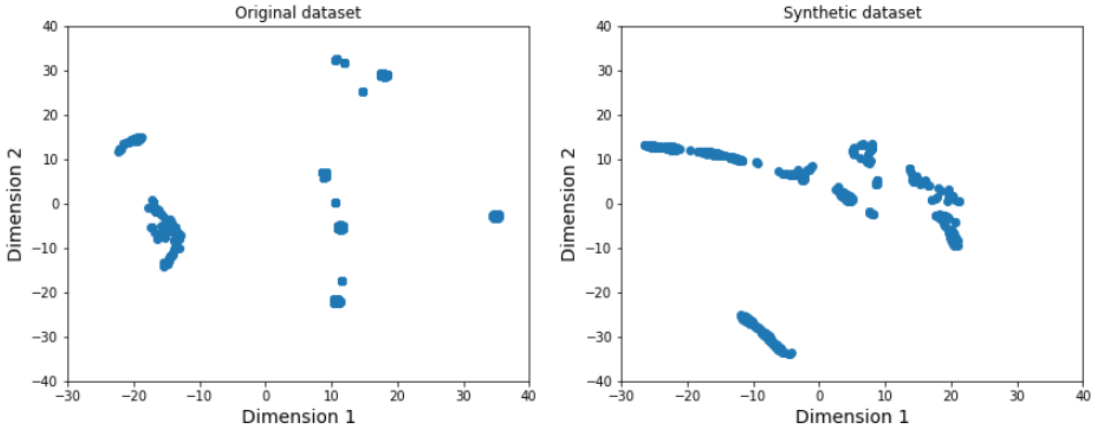
t-SNE visual result using 5-dim latent representation from autoencoder on dataset generated by ctGAN (400 rows):



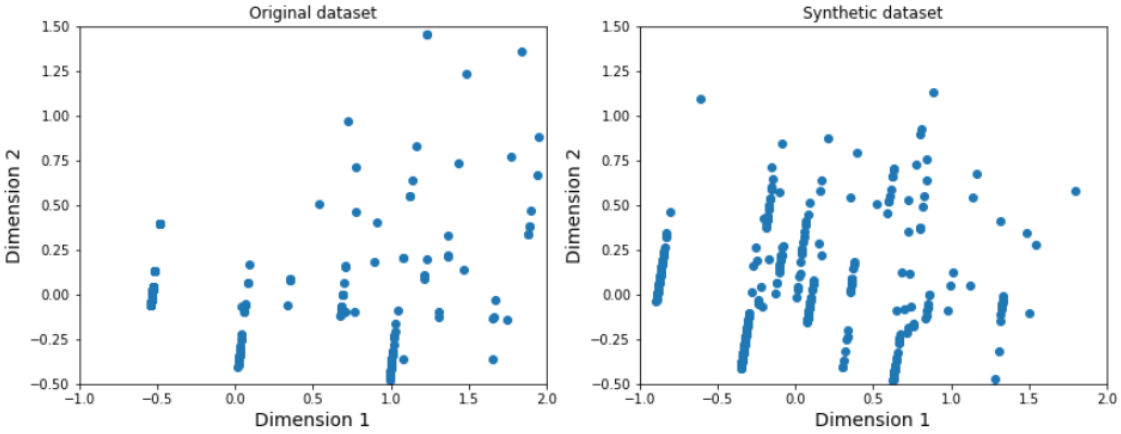
PCA visual result using 5-dim latent representation from autoencoder on dataset generated by tGAN (400 rows):



t-SNE visual result using 5-dim latent representation from autoencoder on dataset generated by ctGAN (382 rows):



PCA visual result using 5-dim latent representation from autoencoder on dataset generated by ctGAN (382 rows):



2.1.2 Auto-encoder

Autoencoders are similar to dimensionality reduction techniques like Principal Component Analysis (PCA). They project the data from a higher dimension to a lower dimension using linear transformation and try to preserve the important features of the data while removing the non-essential parts. However, the major difference between autoencoders and PCA lies in the transformation part: as you already read, PCA uses linear transformation whereas autoencoders use non-linear transformations.

We can also reduce the feature data to representations called **embeddings**, and then compare the embeddings. Embeddings are generated by training a supervised **deep neural network** (**DNN**) on the feature data itself. The embeddings map the feature data to a vector in an embedding space. Typically, the embedding space has fewer dimensions than the feature data in a way that captures some latent structure of the feature data set.

Because Neural Networks can slowly approximate any function that maps inputs to outputs through an iterative optimization process called **training**. If we set the output to be the same as the input, we can call this neural network an **autoencoder**, because it encodes a more dense representation of the input data.

An autoencoder can be used to generate the embedding. An **autoencoder** is a type of artificial neural network used to learn efficient data coding in an unsupervised manner. The aim of an **autoencoder** is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal “noise”. There are many types of autoencoders we could make, a more recent generative model is the "variational autoencoder" which learns a latent variable of its input data.

Step1: Train an Autoencoder

Autoencoder can be broken in to three parts

- Encoder: this part of the network compresses or downsamples the input into a fewer number of bits. The space represented by these fewer number of bits is often called the *latent-space* or *bottleneck*. The bottleneck is also called the "maximum point of compression" since at this point the input is compressed the maximum. These compressed bits that represent the original input are together called an “encoding” of the input.
- Decoder: this part of the network tries to reconstruct the input using only the encoding of the input. When the decoder is able to reconstruct the input exactly as it was fed to the encoder, you can say that the encoder is able to produce the best encodings for the input with which the decoder is able to reconstruct well!

Train an autoencoder on our dataset by following these steps:

1. Ensure the hidden layers of the autoencoder are smaller than the input and output layers.
2. Calculate the loss for each output as described in Supervised Similarity Measure.
3. Create the loss function by summing the losses for each output. Ensure you weight the loss equally for every feature. For example, because color data is processed into RGB, weight each of the RGB outputs by 1/3rd.
4. Train the DNN.

Step2: Extracting Embeddings from DNN

After training your DNN, extract the embedding for an example from the DNN. Extract the embedding by using the feature data of the example as input, and read the outputs of the final hidden layer. These outputs form the embedding vector. Remember, the vectors for similar houses should be closer together than vectors for dissimilar houses.

Step3: Measuring Similarity from Embeddings

Next we would talk about how to quantify the similarity for pairs of examples by using their embedding vectors.

We now have embeddings for the pair of tables. A similarity measure takes these embeddings and returns a number measuring their similarity. Remember that embeddings are simply vectors of numbers. To find the similarity between two vectors A=[a1,a2,...,an] and B=[b1,b2,...,bn], we have three similarity measures to choose from, as listed in the table below.

Measure	Meaning	Formula	Relationship to increasing similarity
Euclidean distance	Distance between ends of vectors	$\sqrt{(a_1-b_1)^2+(a_2-b_2)^2+\ldots+(a_N-b_N)^2}$	Decreases
Cosine	Cosine of angle θ between vectors	$\frac{a^T b}{\lvert a\rvert \cdot \lvert b\rvert}$	Increases
Dot Product	Cosine multiplied by lengths of both vectors	$a_1b_1+a_2b_2+\ldots+a_nb_n=\lvert a\rvert\lvert b\rvert cos(\theta)$	Increases. Also increases with length of vectors.

The **Euclidean distance** between two points in either the plane or 3-dimensional space measures the length of a segment connecting the two points. It is the most obvious way of representing distance between two points.

Cosine similarity is the **cosine** of the angle between two n-dimensional vectors in an n-dimensional space. It is the dot product of the two vectors divided by the product of the two vectors' lengths (or magnitudes). Therefore, it is a similarity measurement determining how similar two vectors are irrespective of their size.

To compare these two measurements in an informal but rather intuitive way, Euclidean distance and cosine similarity can be considered capturing the *magnitude* and *direction* respectively. Direction is the "sentiment" of the vector, while the magnitude is how strong it is towards that direction. When measuring the similarity between two vectors, it would be great if we can consider both.

Cosine similarity score using 5-dim latent representation from autoencoder using on dataset generated by ctGAN: 0.963.

```python
# Compute Cosine Similarity
from scipy.spatial import distance
cosine_similarity = 1 - distance.cosine(encoded_traindata, encoded_testdata)
print("cosine similarity: ",cosine_similarity)
# Compute euclidean distance
# euclidean_distance = distance.euclidean(encoded_traindata, encoded_testdata)
# print(euclidean_distance)
# Compute dot product
# dot_product = np.dot(encoded_traindata, encoded_testdata)
# print(dot_product)
```

```
cosine similarity:  0.9630141258239746
```

Similarity Measurement Selection:

In contrast to the cosine, the dot product is proportional to the vector length. This is important because examples that appear very frequently in the training set. If you want to capture popularity, then choose dot product. However, the risk is that popular examples may skew the similarity metric. To balance this skew, you can raise the length to an exponent to calculate the dot product.

Limitations:

An autoencoder is the simplest choice to generate embeddings. However, an autoencoder isn't the optimal choice when certain features could be more important than others in determining similarity.

2.2 Clustering Similarity Metric

gower distance + clustering + mutual information calculation

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mutual_info_score.html