Fully Connected Self-Supervised Deep Learners in Data-Driven Mass Spectrometry Proteomics

Journal:	*Analytical Chemistry*
Manuscript ID	Draft
Manuscript Type:	Technical Note
Date Submitted by the Author:	n/a
Complete List of Authors:	Nouri Nigjeh, Benjamin; University of Washington,

Fully Connected Self-Supervised Deep Learners in Data-Driven Mass Spectrometry Proteomics

Benjamin Nouri Nigjeh [1, *]

[1, *] Department of Medicine, University of Washington at Seattle, WA 92014

Address all correspondence to nigjeh@uw.edu

Abstract:

Self-supervised mass spectrometry proteomics can facilitate automation, and automized data-driven data mining from big mass spectrometry data generated by high throughput technologies. The data-driven proteomic technology presented in this study includes an unsupervised algorithm for data compression of highly dimensional mass spectrometry data by fully connected variational autoencoders regularized by gaussian distribution. Data compression follows by clustering of the sample clusters generated at the latent hidden space layer by a density-based algorithm. The labels generated from the unsupervised algorithm will be used to train downstream supervised algorithms such as fully connected neural network classifiers. Here, the impact of data quantity and quality based on the number of informative features and signal to noise ratio was assessed by a synthetic dataset. Several clustering algorithms (DBSCAN, k-means, agglomerative clustering, spectral clustering, and affinity propagation) were benchmarked in clustering the compressed synthetic datasets. Finally, the performance of the density-based algorithm and its key hyperparameter, i.e., epsilon, was optimized according to cluster homogeneity and completeness monitored from 4 common proteomic sample types: whole tissue lysates, excised gel bands, ion-exchange fractions, and immunoprecipitated samples. It is suggested that the unsupervised algorithm requires tune samples to generate the optimized key hyperparameter according to the maximum harmonic mean between homogeneity and completeness from the prelabelled tune samples.

Keywords: Mass Spectrometry, Proteomics, Machine Learning, Neural Networks, Data Compression

Introduction:

Advent of high-frequency and high-resolution mass spectrometry instruments has enabled generation of big proteomic datasets from discovery-based quantitative technologies such as 10-plex TMTs [Pandit et. al., 2022], super SILAC [Pozniak et. al., 2016], and DIA mass spectrometry [Nouri Nigjeh et. al., 2017]. Yet, data mining from these discovery-based proteomic datasets is challenging due to highly dimensional nature of these datasets, high level of background biological noise, and difficulty to accurately label the proteomic samples due to the complexity of their origin (cell types, tissue types, population effect, etc.). In addition, variations occurring throughout sample collection, storage and processing makes the domain expertise labeling of the proteomic samples a daunting task.

Machine learning algorithms, including neural networks, are gaining attention in deciphering the complexities of these proteomic datasets [Jesse G. Meyer, 2021]. These machine learning algorithms are efficient in extracting data from highly dimensional datasets and are self-correcting when learning from noisy biological datasets. Neural networks, in addition, offer learning from nonlinear relationships, and non-normal distributions, when dataset is sparse and contains missing data [LeCun et. al., 2015] [Francis Chollet, 2021]. However, supervised training of these algorithms relies heavily on accurate labeling of proteomic samples by domain expertise to explore the unique biological features and patterns.

In this study, an unsupervised algorithm is utilized to generate labels to train the downstream supervised algorithms. This unsupervised algorithm consists of a fully connected variational autoencoder with a density-based algorithm to generate sample clusters in the hidden latent space layer and annotate labels for the proteomic samples for the downstream supervised algorithms.

Experimental procedures:

Hardware and Software: Deep learner algorithms were trained on an NVIDIA GeForce RTX 3050 GPU system with an Intel Core i5 CPU and 8 GB of RAM. The operating system was Windows 11. The computations on NVIDIA GPU were parallelized with the installation of CUDA/cuDNN. Coding was done on Python version 3.9.7 running on PyCharm IDE version 2022.1. Sequential and functional neural network structures were built using keras API, a high-level frontend module integrated on tensorflow library version 2.8. Data visualizations were done on GraphPad Prism version 5, and matplotlib API.

Proteomic samples: The studied proteomic samples were excised from mouse liver tissues. The proteomic samples are categorized as (a) whole mouse liver tissue lysates (n= 23) (b) SCX-fractionated tissue lysates (eluted after 25, 30, 50, 60, 120-, 200-, 20-, and 500-mM ammonium acetate) (n= 15) (c) immunoprecipitation of a membrane kinase protein (n= 13) and (d) gel separated samples at MW between 60 and 70 kDa (n= 2). The protein content of these samples (after elution by 10 mM acetic acid for the immune precipitated beads) were precipitated with acetone and washed twice and digested on the pellet following a protocol presented in an earlier publication [Nouri Nigjeh et. al., 2014]. The gel samples were reduced, alkylated, and digested inside the gel bands [Schevchenko et. al., 2006].

LC/MS instrumentation: Analyses were performed on a Q Exactive™ Hybrid Quadrupole Orbitrap mass spectrometer connected to an Easy-nLC 1000 system. The analytical column was a C18 EASY-Spray column, 25 cm × 75 mm ID, filled with 2 mm particles (100 Å pore size), connected in series with a C18 cartridge trapping column, 5 mm × 300 mm ID, filled with 5 mm particles (100 Å pore size). The tryptic peptides were resolved with a flow rate of 300 nL/min and a 150-min gradient. Solvent A was 100% water containing 0.1% formic acid. Solvent B content (100% acetonitrile containing 0.1% formic acid) was increased from 2 to 44% within 140 min. The peptides were then analyzed under data-dependent acquisition mode with a survey scan between 375 and 1700 m/z, with a resolution of 70,000 at 200 m/z, and AGC target of 1e6 (maximum injection time was set at 60 ms). Following the survey scan, top 10 product ions were selected for fragmentation, under normalized collision energy (NCE) of 27, with a resolution of 17,500 at 200 m/z, and AGC target of 5e4 (maximum injection time was set at 64 ms).

Results and Discussion:

Data-driven data mining from proteomic datasets, generated from discovery-based mass spectrometry experiments, can be divided into two major steps, first to generate structured tabular proteomic datasets, and then to perform data mining using several machine learning algorithms. **Scheme 1** shows the data-driven pipeline workflow from the self-supervised mass spectrometry proteomics including a series of unsupervised and supervised algorithms. The raw mass spectrometry data was first converted to open mzML format using msconvert executive file from proteowizard and then searched against mouse uniport database using comet search engine [Eng et. al., 2013]. These two functions are implemented in a python loop (**code S1**). Database search was limited to the tryptic peptides without miss-cleave sites and no dynamic modification, while cysteine residues were constantly alkylated (+57). The doubly charged tryptic peptides were matched with sequence length's more than 7 amino acids. The identified peptides were weighted based on the peptide prophet algorithm implemented on Trans Proteome Pipeline (version 6.1.0) with cut-off value of 0.95 and then imported to Skyline (version 22.2). The protein quantification was performed based on the AUC sum of the peptides and the protein levels were normalized against its maximum values to confine the protein expression level between 0 and 1 (**code S1**). The proteomic dataset used for data mining contained top 1296 most abundant proteins for training the neural network algorithms. Data-driven data mining includes an unsupervised neural network to reduce the dimensionality of the proteomic dataset to 2 dimensions in the latent layer space of a variational autoencoder, and a suitable clustering algorithm to generate labels. The labeled samples will be used to train a supervised neural network such as fully connected neural network classifiers as illustrated in **Scheme 1**.

Data compression and dimensionality reduction of the proteomic dataset was performed with a fully connected variational autoencoder which is regularized by the assumption that data distribution in the latent space layer follows a gaussian distribution [Diedrik Kingma and Max Welling, 2014]. The model regularization is a key factor toward generalization from the training dataset and avoiding memorization and overfitting [Franco et. al., 2021]. The architecture of decoder and encoder of the fully connected variational autoencoder is summarized in **code S2**. Data visualization for each sample is performed by plotting two-dimensional Kullback-Leibler divergence factors (z_0 and z_1) from the normal distribution [Francis Chollet, 2021]. The performance of the algorithm was evaluated by the measurement of the Euclidean distance between cluster centers of synthetic random noise datasets with increasing number of samples and increasing number of informative features (with constant signal to noise ratio of 6). **Figure 1** shows the Euclidean cluster center distances for synthetic datasets with 500, 1000, 2000, and 10,000 sample sizes, and informative features between 1 to 6. By increasing the number of samples in the training dataset, the number of informative features to separate the groups decreases. This indicates the efficiency of the algorithm in training from the noisy datasets when trained by increasing quantity of the samples. The unsupervised algorithm shows a tradeoff between quality and quantity of the training samples, while they are self-correcting when trained from a large quantity of noisy datasets. For a fixed number of features, the increasing signal to noise ratio, i.e., quality of the informative features shows a similar trend (**Figure S1**). Therefore, the tradeoff is between the number of samples used for training, the number of informative features and quality of the informative features (signal to noise ratio).

Clustering of groups generated after data compression in the latent space layer were benchmarked with a synthetic noise dataset with 4 different set of informative features with signal to noise ratio of 6. **Figure S2** shows the clustering performed by Density-Based Spatial Clustering of Applications with Noise (DBSCAN), k-means, agglomerative clustering, spectral clustering, and affinity propagation [Aurelien

Geron, 2017] in compare with the original labels. The pre-knowledge of cluster numbers is required for these three clustering algorithms: k-means, agglomerative clustering, and spectral clustering. The affinity propagation didn't converge. The technical details of these benchmarked algorithms are presented in **code S2**. To estimate the cluster number, k-means silhouette score was plotted against presumed cluster numbers to determine the number of clusters from the maximum silhouette score. Yet, this approach heavily depends on the shape of the clusters formed. **Figure S3** compares the performance of k-means silhouette score for synthetic noise dataset with 4 informative features versus 3, 4, and 5 blub-shape clusters generated by Scikit learn make classification. Evidently, the k-means and maximum silhouette score algorithm works better for the blub-shape clusters than strips generated from the synthetic noise datasets with limited number of informative features. In contrast, DBSCAN algorithm doesn't need to receive the number of clusters and it performs well regardless of the shape of the clusters (either circular blub shapes or stripe shapes). However, the main hyperparameter for this algorithm, i.e., epsilon, which indicates the maximum distance between two samples for one to be considered as in the neighborhood of the other needs to be optimized. This key hyperparameter depends heavily on the performance of the unsupervised algorithms and nature of the proteomic dataset.

The DBSCAN key hyperparameter was optimized based on the correlation between number of assigned clusters, homogeneity and completeness of data generated by increasing number of epsilon value for the given dataset. Homogeneity is a parameter to indicate if clusters contain only data points which are members of a single class, while completeness is a parameter to indicate if all the data points that are members of a given class are elements of the same cluster. The V-measure is the harmonic mean between homogeneity and completeness. **Figure S4** shows the performance of DBSCAN algorithm with increasing epsilon value from the synthetic noise dataset. This shows that though the density-based algorithm is performing well for a verity of cluster shapes, the choice of right epsilon value is crucial for the successful performance of the algorithm. The outlier samples with less than a minimum number of samples within the epsilon are assigned as the noise. The optimal epsilon value for these synthetic noise datasets, according to the maximum value of v-measure score is 0.2. This epsilon value also corresponds to the presence of 4 sample clusters in the dataset.

The optimal unsupervised algorithm, in **code s2**, was used to perform data compression from 4 group of common mass spectrometry data. **Figure 2** shows the dimensionality reduction, and data compression for these 4 mass spectrometry groups. Clearly, these 4 groups of samples (whole tissue lysates, SCX-fractionated, immune precipitated and excised gel bands) are distinguishable after data compression. The gel samples were removed from clustering as it only contains two samples. Epsilon, the key hyperparameter optimized for three groups of immune precipitated samples, whole tissue, and SCX-fractionated samples. Hyperparameter optimization was done for the combined dimensionality reduction and cluster formation, as shown in **Figure 3**. The algorithm shows the pre-assumed number of clusters at epsilon 1 correlating with the maximum v-measure score, and minimal number of unassigned samples. This emphasizes the importance of assignment of right epsilon value dependent on the cluster shapes to generate the correct cluster numbers. Since the optimum epsilon value is dependent on the shape of the data clusters, this necessitates the presence of pre-labeled tune samples to estimate the optimal epsilon value with maximizing the v-measure score according to the prelabeled tune samples. Therefore, this study suggests that a series of tune samples should be added to the training dataset with pre-known labels and the epsilon value corresponding to the maximum v-measure to be used for clustering the experiment samples.

The labels generated by data compression with variational autoencoder and DBSCAN algorithm will be used for training supervised algorithms such as fully connected neural network classifiers. The tandem use of the supervised algorithm provides the benefits for highly sensitive classification of sample groups. The

variational autoencoder when trained with a big number of data will be useful for data inference, and generation of augmented data from complex biological matrices [Gronbach et. al. 2020]. The validity of linear arithmetic transformation in the latent space of variational autoencoders can reveal key biological features distinguishing different biological groups [Gomari et. al, 2022].

Conclusions:

Accurate labeling of proteomic samples is a daunting task due to the complexities originated from sample heterogeneity, storage, and processing. Yet, accurate labeling is necessary for training supervised machine learning algorithms. Here, we proposed a dimensionality reduction, and data compression algorithm combined with a density-based clustering algorithm to generate the labels for training the downstream supervised algorithms. The key hyperparameter to generate clusters, the epsilon value, needs to be obtained by the addition of pre-labeled tune samples to the dataset, and estimated from the maximized v-measure score. This approach can lead to self-supervised mass spectrometry proteomics.

References:

Aurélien Géron. Hands-On Machine Learning with Scikit-Learn and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'REILY. 2017. 3rd Edition. ISBN:1491962291

Francois Chollet. Deep Learning with Python. MANNING. 2021. 2nd Edition. ISBN:1617296864

Diederik P. Kingma, and Max Welling. Auto-Encoding Variational Bayes. Computing Research Repository, abs. 2014. 1312(6114).

http://arxiv.org/abs/1312.6114

Eng et. al. Comet: an open-source MS/MS sequence database search tool. Proteomics. 2013, 13(1): 22-4.

doi.org/10.1002/pmic.201200439

Franco et. al. Performance Comparison of Deep Learning Autoencoders for Cancer Subtype Detection Using Multi-Omics Data. Cancers. 2021, 13 (2013).

doi.org/10.3390/cancers13092013

Gomari et. al. Variational autoencoders learn transferrable representations of metabolomics data. Communications Biology. 2022, 5(645).

doi.org/10.1038/s42003-022-03579-3

Grønbech et. al. scVAE: variational auto-encoders for single-cell gene expression data. Bioinformatics. 2020. 36(16): 4415–4422

doi.org/10.1093/bioinformatics/btaa293

Jesse G. Meyer. Deep learning neural network tools for proteomics. Cell Reports Methods. 2021, 1(2): 100003

doi.org/10.1016/j.crmeth.2021.100003

LeCun et. al. Deep learning. Nature. 2015, 521(7553): 436-44.

doi.org/10.1038/nature14539

Nouri Nigjeh et. al. Quantitative proteomics based on optimized data-independent acquisition in plasma analysis. Journal of Proteome Research. 2017, 16 (2): 665–676.

doi.org/10.1021/acs.jproteome.6b00727

Nouri Nigjeh et. al. Highly Multiplexed and Reproducible Ion-Current-Based Strategy for Large-Scale Quantitative Proteomics and the Application to Protein Expression Dynamics Induced by Methylprednisolone in 60 Rats. Analytical Chemistry. 2014, 86(16): 8149–8157.

doi.org/10.1021/ac501380s

Pandit et. al. Proteome Analysis of Pancreatic Tumors Implicates Extracellular Matrix in Patient Outcome. Cancer Research Communications. 2022, 2(6): 434-446.

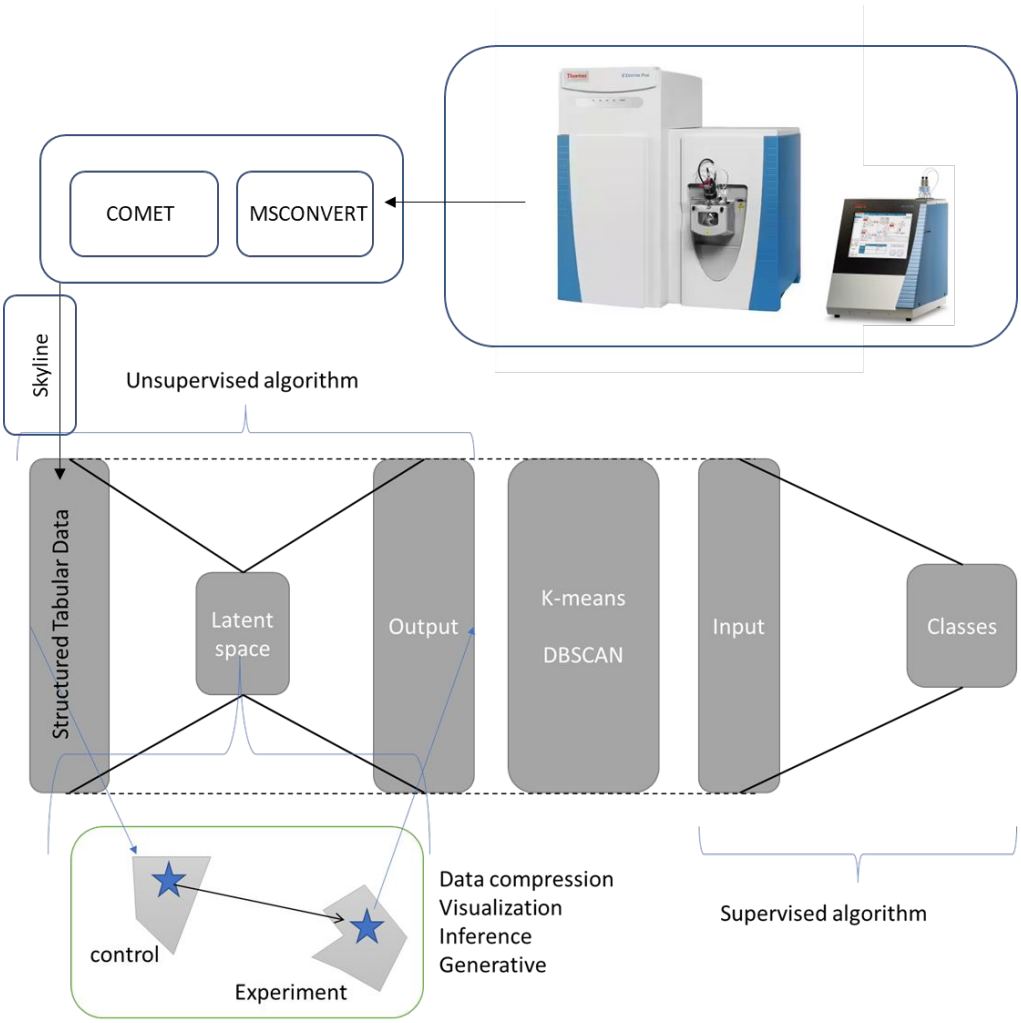
doi.org/10.1158/2767-9764.CRC-21-0100

Pozniak et. al. System-wide Clinical Proteomics of Breast Cancer Reveals Global Remodeling of Tissue Homeostasis. Cell Systems. 2016, 2(3): 172-184.

doi.org/10.1016/j.cels.2016.02.001

Schevchenko et. al. In-gel digestion for mass spectrometric characterization of proteins and proteomes. Nature Protocols. 2006, 1: 2856–2860.

doi.org/10.1038/nprot.2006.468



Scheme 1. Workflow for the data-driven mass spectrometry proteomics from generation of the structured tabular proteomic datasets to data compression and dimensionality reduction and generation of clusters in the hidden latent space layer of a fully connected variational autoencoder by using DBSCAN algorithm followed by downstream supervised algorithm.

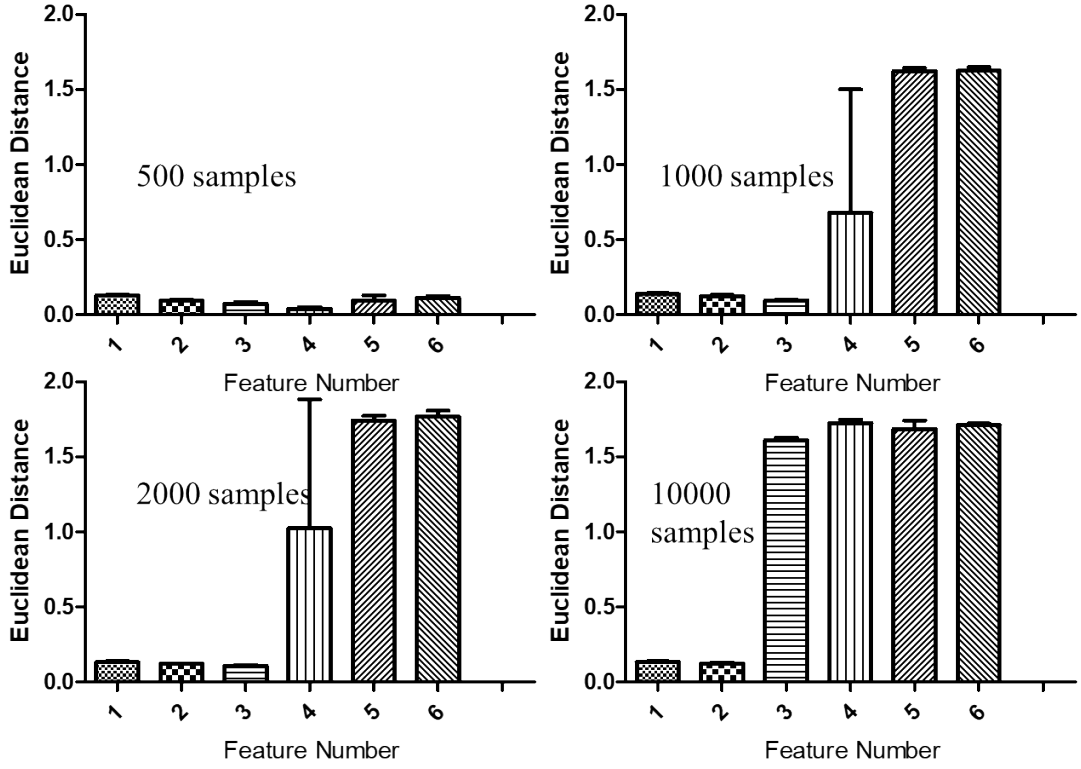


Figure 1. The Euclidean distance between cluster centers of experiment versus control groups assigned from the compressed synthetic noise datasets with increasing number of samples and informative features defined by signal to noise ratio of 6.

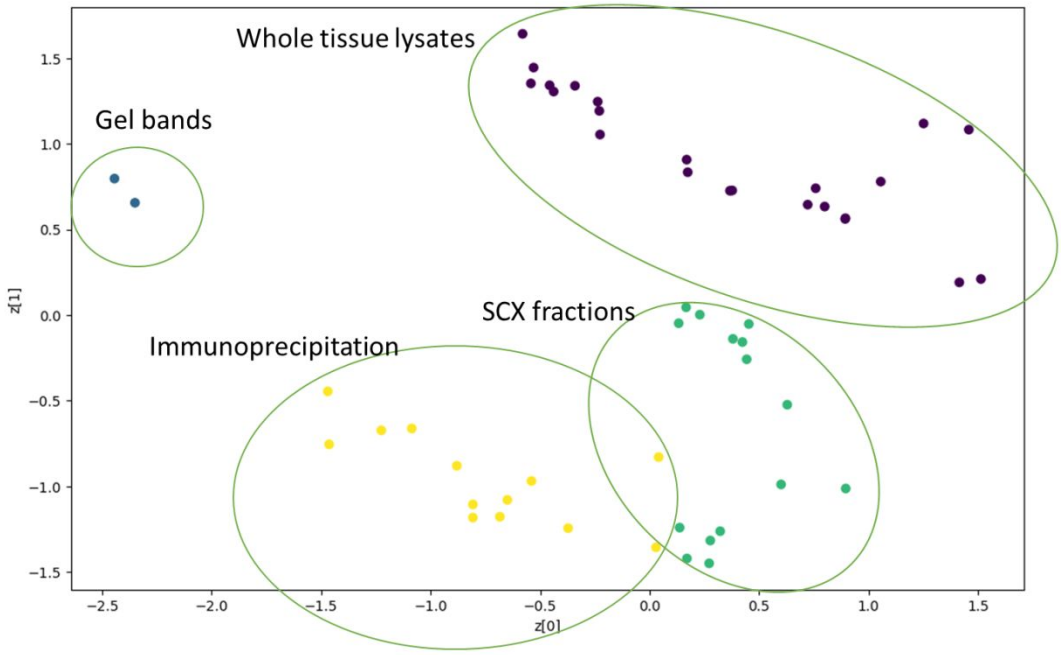


Figure 2. Data compression by fully connected variational autoencoder from 4 groups of common mass spectrometry data (whole tissue lysates, SCX-fractions, immunoprecipitated samples, and gel bands) generated in the hidden latent space layer.

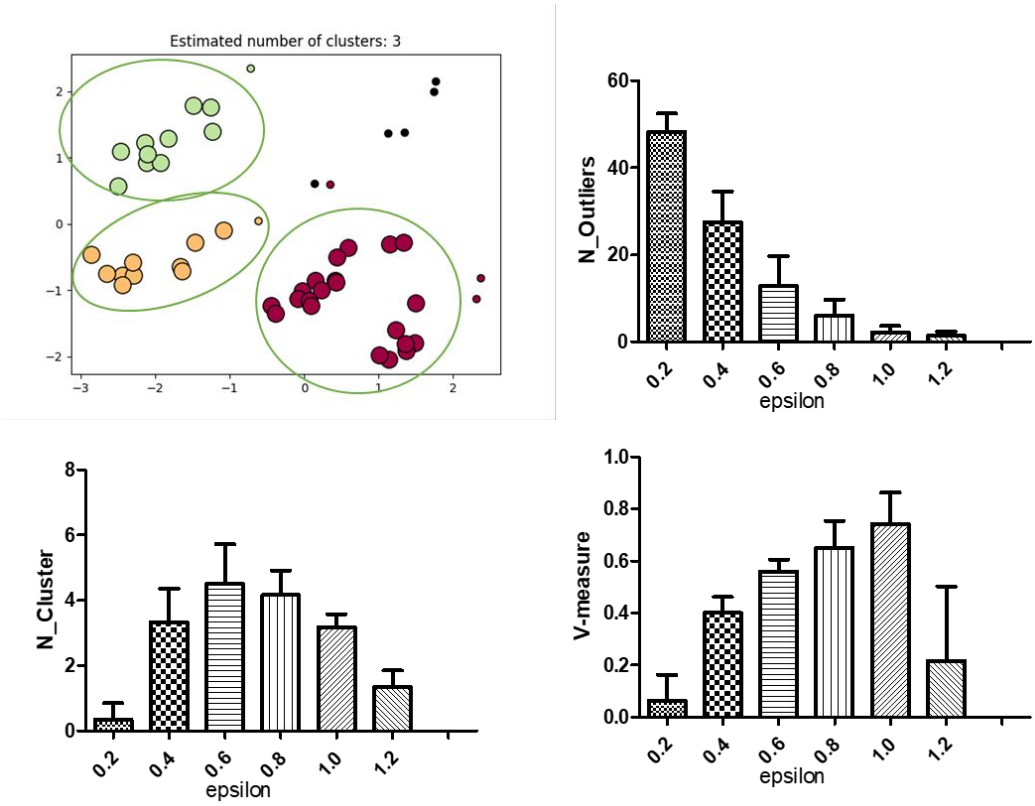


Figure 3. The DBSCAN clustering of the three common mass spectrometry groups (whole tissue lysates, immune precipitated samples, and SCX-fractionated samples) with epsilon value equal to 1. The number of outliers, optimal clusters, and v-measure score are shown with varying number of epsilon value. The maximum v-measure score occurs at epsilon equal to 1 which corresponds to the presence of three clusters in the compressed dataset.

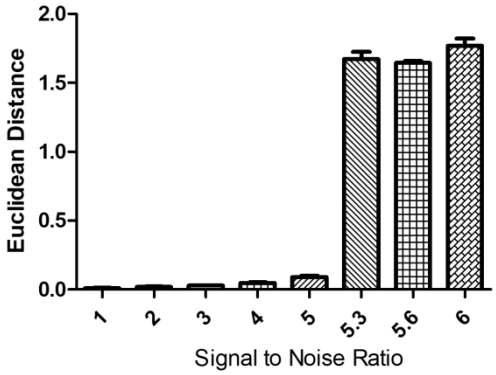


Figure S1. The impact of increasing level of signal to noise ratio for the fixed number of training samples (i.e., 2000) and fixed number of informative features (i.e., 6) on the Euclidean distance between cluster centers, as an indicative parameter for the quality of the informative features.

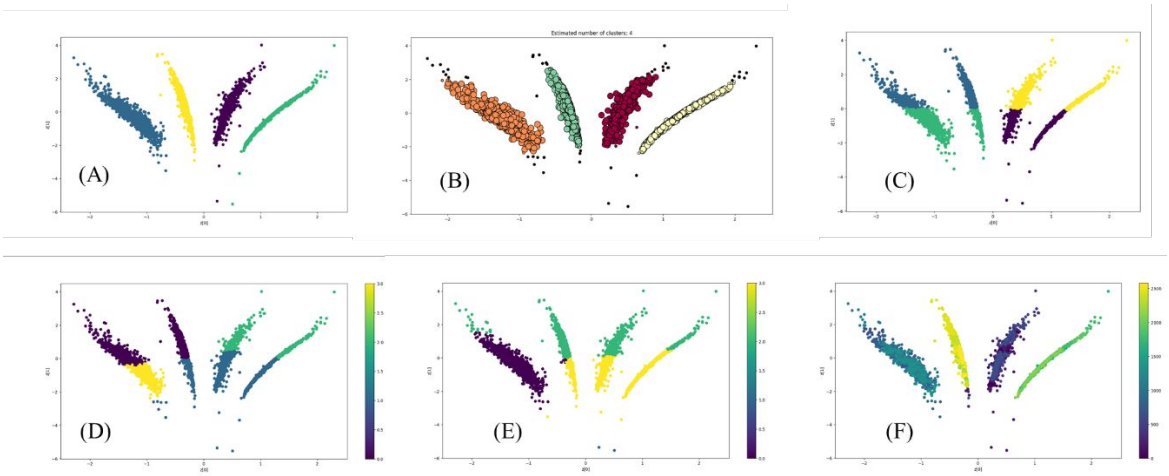


Figure S2. Clustering of synthetic noise dataset with 4 different set of informative features compressed in the latent space layer based on the original labels (a), DBSCAN (b), k-means (c), agglomerative clustering (d), spectral clustering (e), and affinity propagation (f).

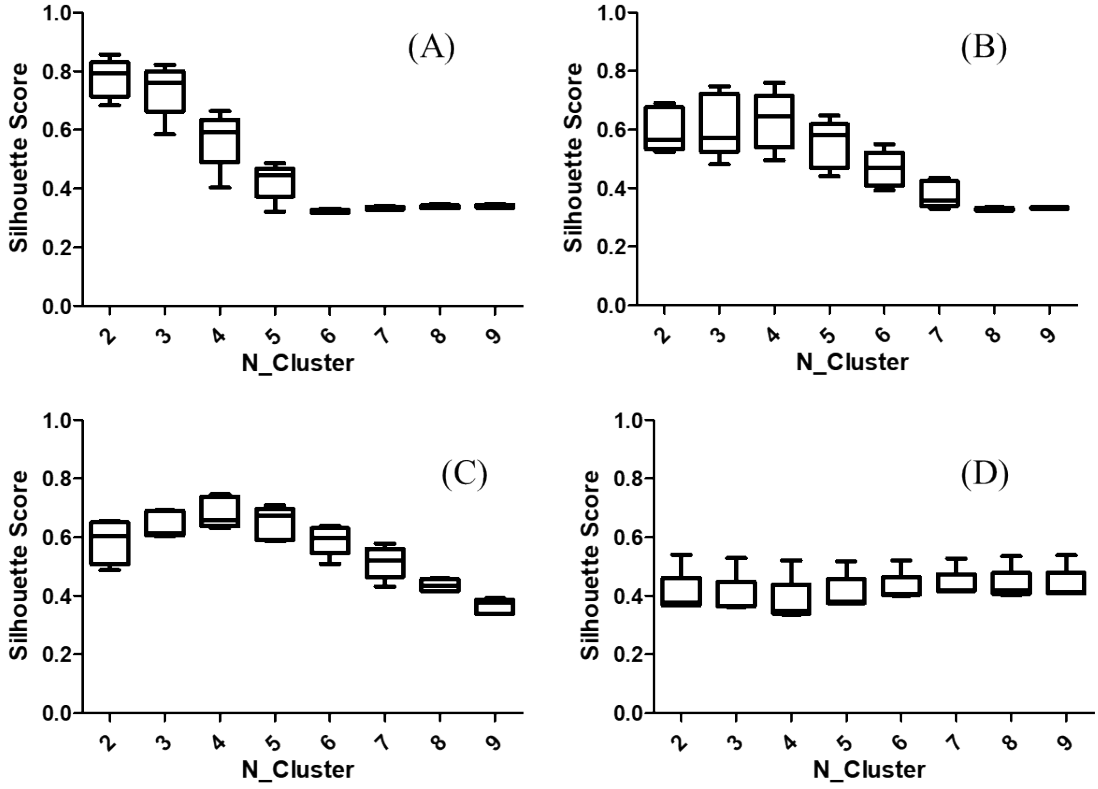


Figure S3. Maximum k-means silhouette score estimated from the increasing number of blub-shape clusters: 3 (a), 4 (b), and 5 (c), in compare with k-means silhouette score estimated from synthetic noise datasets with 4 stripe-shape clusters generated from 4 set of informative features (d)

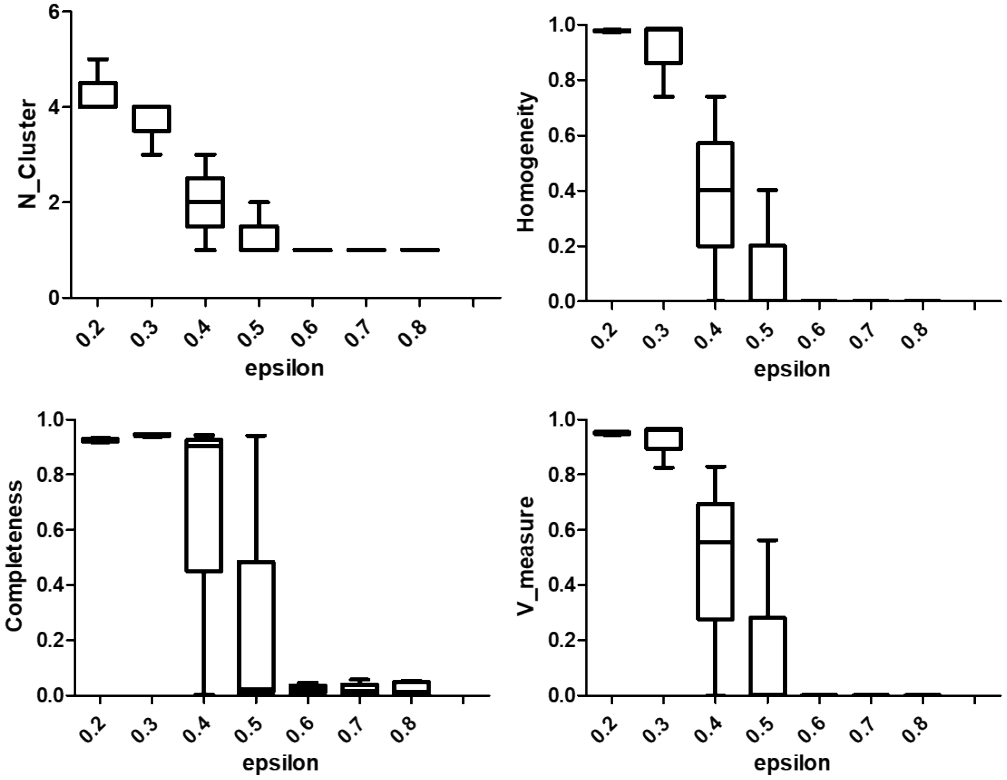


Figure S4. The impact of epsilon value on the assignment of correct number of clusters and thereof homogeneity of the clusters and completeness of data from each labeled class and V-measure which is the harmonic mean value of homogeneity and completeness

Code s1: Preparation of the proteomic dataset

```
import os
import numpy as np
import pandas as pd
from numpy import genfromtxt

path_search = "///input"
os.chdir(path_search)
n= 53
x = range(1, n)
for i in x:
    NAME = i
    os.system("msconvert.exe {}.mZML".format(NAME))
    os.system("comet.exe {}.mZML".format(NAME))


#The result submitted to TPP and signal extracted from Skyline


df = pd.DataFrame.from_csv('input.csv')
group = df.groupby(['Replicate Name', 'Protein Name'])
df_sum = group.sum()
df1 = df_sum.xs('output',level=0,axis=0)
df1.to_csv('output.csv')

ds = genfromtxt('output.csv', delimiter=',')
ds_abs = np.abs(ds)
ds_abs_log2 = np.log2(ds_abs)
ds_abs_log2_abs = np.abs(ds_abs_log2)

maxInRows = np.amax(ds_abs_log2_abs, axis= 1)

list_i = list(range(0, ds_abs_log2_abs.shape[0]))
list_j = list(range(0, ds_abs_log2_abs.shape[1]))

ds_final =[]

for i in list_i:
    ds_final.append(ds_abs_log2_abs[i]/maxInRows[i])

for i in list_i:
    for j in list_j:
        if ds_final[i][j] < 0.05:
            ds_final[i][j] = 0

np.savetxt("ds_final.csv", ds_final, delimiter=",")
```

Code s2: Data compression by variational autoencoders and clustering benchmark

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.callbacks import TensorBoard
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn import metrics
import sklearn.datasets
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


NAME = "Self_Supervised"
tensorboard = TensorBoard(
    log_dir='///{}'.format(NAME))

file_path = '///ds_input.csv'
df = pd.read_csv(file_path)
X = df.copy()
Y = X.pop("target")
X_np = np.array(X)
x_train = np.reshape(X_np,(-1,28,28, 1))
y_train = np.array(Y)



class Sampling(layers.Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon

latent_dim = 2

encoder_inputs = keras.Input(shape=(28, 28, 1))
x = layers.Flatten()(encoder_inputs)
x = layers.Dense(120 , activation="relu")(x)
x = layers.Dense(120 , activation="relu")(x)
x = layers.Dense(16, activation="relu")(x)
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)
z = Sampling()([z_mean, z_log_var])
encoder = keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")
encoder.summary()

latent_inputs = keras.Input(shape=(latent_dim,))
x = layers.Dense(28 * 28 * 1, activation="relu")(latent_inputs)
x = layers.Dense(28 * 28 * 1, activation="sigmoid")(x)
decoder_outputs = layers.Reshape((28, 28, 1))(x)
decoder = keras.Model(latent_inputs, decoder_outputs, name="decoder")
decoder.summary()
```

```
class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.total_loss_tracker = keras.metrics.Mean(name="total_loss")
        self.reconstruction_loss_tracker = keras.metrics.Mean(
            name="reconstruction_loss"
        )
        self.kl_loss_tracker = keras.metrics.Mean(name="kl_loss")

    @property
    def metrics(self):
        return [
            self.total_loss_tracker,
            self.reconstruction_loss_tracker,
            self.kl_loss_tracker,
        ]

    def train_step(self, data):
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)
            reconstruction_loss = tf.reduce_mean(
                tf.reduce_sum(
                    keras.losses.binary_crossentropy(data, reconstruction),
axis=(1, 2)
                )
            )
            kl_loss = -0.5 * (1 + z_log_var - tf.square(z_mean) -
tf.exp(z_log_var))
            kl_loss = tf.reduce_mean(tf.reduce_sum(kl_loss, axis=1))
            total_loss = reconstruction_loss + (kl_loss)
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
        self.total_loss_tracker.update_state(total_loss)
        self.reconstruction_loss_tracker.update_state(reconstruction_loss)
        self.kl_loss_tracker.update_state(kl_loss)
        return {
            "loss": self.total_loss_tracker.result(),
            "reconstruction_loss": self.reconstruction_loss_tracker.result(),
            "kl_loss": self.kl_loss_tracker.result(),
        }


vae = VAE(encoder, decoder)
vae.compile(optimizer=keras.optimizers.Adam())
vae.fit(x_train, epochs=500, batch_size=128, callbacks=[tensorboard])

def plot_label_clusters(vae, data, labels):
    # display a 2D plot of the digit classes in the latent space
    z_mean, _, _ = vae.encoder.predict(data)
    plt.figure(figsize=(12, 10))
    plt.scatter(z_mean[:, 0], z_mean[:, 1], c=labels)
    plt.colorbar()
```

```
    plt.xlabel("z[0]")
    plt.ylabel("z[1]")
    plt.show()
plot_label_clusters(vae, x_train, y_train)

#DBSCAN Clustering
z_mean, _, _ = vae.encoder.predict(x_train)
z_mean = np.array(z_mean)
X = np.reshape(z_mean,(-1,2))
labels_true = y_train
print(X.shape)
print(y_train.shape)

db = DBSCAN(eps=0.2, min_samples=10).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_


n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print("Estimated number of clusters: %d" % n_clusters_)
print("Estimated number of noise points: %d" % n_noise_)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true,
labels))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(labels_true,
labels))
print(
    "Adjusted Mutual Information: %0.3f"
    % metrics.adjusted_mutual_info_score(labels_true, labels)
)
print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X, labels))


unique_labels = set(labels)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1,
len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = labels == k

    xy = X[class_member_mask & core_samples_mask]
    plt.plot(
        xy[:, 0],
        xy[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=14,
    )
```

```
        xy = X[class_member_mask & ~core_samples_mask]
        plt.plot(
            xy[:, 0],
            xy[:, 1],
            "o",
            markerfacecolor=tuple(col),
            markeredgecolor="k",
            markersize=6,
        )

    plt.title("Estimated number of clusters: %d" % n_clusters_)
    plt.show()

    # k-means clustering
    z_mean, _, _ = vae.encoder.predict(x_train)
    z_mean = np.array(z_mean)
    clusterer = KMeans(n_clusters=4, random_state=1)
    y_train = clusterer.fit_predict(z_mean)
    plot_label_clusters(vae, x_train, y_train)

    # agglomerative clustering
    z_mean, _, _ = vae.encoder.predict(x_train)
    z_mean = np.array(z_mean)
    clusterer = sklearn.cluster.AgglomerativeClustering(4)
    y_train = clusterer.fit_predict(z_mean)
    plot_label_clusters(vae, x_train, y_train)

    # spectral clustering
    z_mean, _, _ = vae.encoder.predict(x_train)
    z_mean = np.array(z_mean)
    clusterer = sklearn.cluster.SpectralClustering(4)
    y_train = clusterer.fit_predict(z_mean)
    plot_label_clusters(vae, x_train, y_train)

    #affinty propagation
    z_mean, _, _ = vae.encoder.predict(x_train)
    z_mean = np.array(z_mean)
    clusterer = sklearn.cluster.AffinityPropagation()
    y_train = clusterer.fit_predict(z_mean)
    plot_label_clusters(vae, x_train, y_train)
```