

The synergistic effect of QR decomposition with t-SNE

Mohsin Ali, Jitendra Choudhary

Department of Computer Science, Faculty of Science Medi-Caps University, Indore, India

Article Info

Article history:

Received Jan 24, 2024

Revised Feb 18, 2024

Accepted Feb 26, 2024

Keywords:

Anderson-Darling test MDS

Mann-Whitney U test

QRPCA-t-SNE

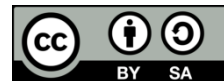
t-SNE

UMAP

ABSTRACT

The study utilized non-parametric tests, specifically, the Mann-Whitney U test, to evaluate the performance of a proposed model called QRPCA-t-SNE, along with two other models, MDS and UMAP. The study compared these three models with two datasets on performance metrics such as model accuracy, training accuracy, testing accuracy, mean square error, AUC scores, precision, recall, and F1 scores. Once the model's performance was conducted, the Anderson-Darling test was to check for data normality before applying the hypothesis for model proof. The analysis revealed that Model 1 (QRPCA-t-SNE) significantly outperformed Model 2 (UMAP) and Model 3 (MDS) in terms of accuracy, with p-values of 0.0027 and 0.0003, respectively. This finding suggests that Model 1 (QRPCA-t-SNE) is suitable for high-accuracy and reliability applications, providing valuable insights into predictive analytics with a 95% confidence interval (confidence level $\alpha=0.05$).

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Mohsin Ali

Department of Computer Science, Faculty of Science Medi-Caps University

A.B Road Rao, 43331, Indore, India

Email: coolbuddy.next.door@gmail.com

1. INTRODUCTION

Dimensionality reduction is used in data analysis and machine learning [1]. It is utilized to reduce a dataset's input variable count to make it easier to handle and understand. This technique offers several benefits, such as making the data more accessible to interpret, increasing computing performance, reducing the chances of overfitting [2]-[5] during model training and making data visualization more accessible.

Data dimensionality reduction is frequently employed for visual data analysis in social sciences and bioinformatics. In the past, principal component analysis (PCA) [6], multidimensional scaling (MDS), and uniform manifold approximation and projection (UMAP) have been among the most popular DR techniques employed. Nevertheless, emerging techniques have expanded the possibilities for reducing data dimensionality, providing researchers and analysts with a broader spectrum of options to optimize their data analysis processes. These techniques are widely used, as evidenced by published references such as [7]-[12].

T-distributed stochastic neighbor embedding (t-SNE)[13]-[18] is a powerful technique to reduce data dimensions, especially for visualizing data with many dimensions. It helps to reveal complex patterns and inherent groupings in the data by transforming similarities between data points into shared probabilities in a space with fewer dimensions using an exponential distribution. Even though t-SNE provides informative visualizations. Furthermore, t-SNE is utilized in many areas, such as biomedical signal processing, genomics [19], [20], and physics [21]. MDS [22] is a technique used to analyze how similar or different items are. It helps us understand patterns in data that have many other dimensions. MDS converts the distances between objects into points plotted in a more straightforward space, like two or three dimensions. The goal is to maintain the distances between the objects as accurately as possible. The technique helps visualize trends in complex data.

Uniform manifold approximation and projection (UMAP) [23] is a method that reduces the complexity of data, primarily when visualizing data with many dimensions. Leland McInnes and John Healy created it, and it is popular because it maintains local and global data structures. UMAP is used for manifold learning, meaning data is dispersed along a curved path. The goal is to make data more accessible to analyze by unfolding this path more simply.

The QRPCA-t-SNE [24] is a deep learning model proposed to help visualize high-dimensional datasets, particularly in genomics. The model is built using a combination of machine learning algorithms and algebraic methods, such as the QR decomposition algorithm [25], principal component analysis, and t-SNE. One of the key benefits of this proposed model is the improvement it brings to essential parameters such as training, testing, F1-Score, model score, mean square error, and AUC score. This makes it a valuable tool for researchers and scientists working with large datasets, as it allows them to analyze and understand complex data sets more efficiently in a way that was not previously possible. Overall, the QRPCA-t-SNE model is an innovative solution that promises to significantly impact the field of genomics and other areas where large, high-dimensional datasets are standard.

This paper compares different dimensionality reduction techniques, such as UMAP and MDS, with the Proposed model (QRPCA-t-SNE) [24] based on various parameters. We aim to demonstrate that the Proposed model is more robust than other dimensionality reduction techniques. To prove its robustness, we use two different datasets related to genomics, namely the cancer dataset (Kaggle website) and the Mouse Whole Cortex and Hippocampus dataset (Allen Atlas) [26].

The paper is organized into multiple sections. Section 2 covers a detailed explanation of how each model parameter is implemented with different datasets. Section 3 covers the model validation process with a statistical test. Finally, section 4 summarizes the study's findings and conclusions.

2. METHOD

The first section covers the dataset description, and the second section provides an overview of various dimension reduction techniques. In the third section, we will describe the comparison tool that can be used to compare different methods. Lastly, the last section will cover the investigation tool.

2.1. Dataset description

We used the molecular cancer gene expression dataset from Kaggle to compare various models. The dataset has 7,129 rows and 38 columns, as illustrated in Figure 1. The second dataset is the mouse whole cortex and hippocampus dataset (Allen Atlas [26]), which contains 10,122 rows and 387 columns. This dataset helps us assess the model's robustness.

2.2. Dimensionality reduction techniques overview

This paper presents a comparative analysis of the dimensionality reduction technique and proposed model QRPCA- t-SNE [24]. The first method we considered is QRPCA-t-SNE [24], which combines QR decomposition and principal component analysis with t-distributed stochastic neighbor edging. The second method is UMAP, which stands for uniform manifold approximation and projection.

2.3. Comparison tools

The parameters which are used to compare the models are as follows. The first parameter is the model score, which is obtained using the train test split function of the sklearn library. The second parameter is the mean squared error. The third parameter is the training and testing results. The fourth parameter is the AUC Score obtained using the roc AUC score function from the sklearn metrics library. The last parameter is the confusion matrix score, which tests precision, recall, accuracy, and F1-Score.

2.4. Investigation tool

This analysis used a system that ran on a Google Collab Pro, having a TPU processor with 35 GB RAM and a 120 GB hard disk. We used Google Colab as our programming tool to achieve optimal results quickly and accurately. Additionally, we compared the dimensionality reduction technique using two popular libraries: TensorFlow and skit-learn.

2.5. Dataset 1

Dataset 1 covers the detailed model implementation of (QRPCA-t-SNE), UMAP, and MDS. Dataset 2 is accompanied with the evaluation parameters like model score, mean squared error, training, and testing. Results: the AUC, receiver operating characteristic curve (ROC score), precision, recall, accuracy, and F1-score.

2.5.1. Model 1: QRPCA-t-SNE model

First, we imported dataset 1, which is displayed in Figure 1. Next, we applied the preprocessing step to visualize the dataset using t-SNE with a perplexity of 40, as illustrated in Figure 2. After that, we split the data into training and testing sets and calculated the model score using LogisticRegressionCV with max_iter=5,000, as demonstrated in Figure 3, to achieve a model score. Once we computed the model score, we estimated the mean squared error (MSE) amount in this model, shown in Figure 4. The next step was to examine the training and testing results displayed in Figure 5. Before the confusion matrix, we checked the AUC score to verify the true and false positive rates, as demonstrated in Figure 6. Finally, we utilized the F1, precision, recall, and accuracy scores to evaluate the model, as shown in Figure 7.

```
import numpy as np

df=np.asarray(train)
print(df.shape)
jason=df

(7129, 38)
```

Figure 1. Dataset_1 size [24]

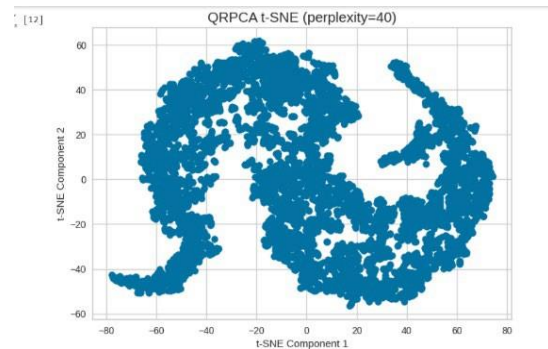


Figure 2. QRPCA-t-SNE visualization [24]

```
X_train, X_test, y_train, y_test = train_test_split(X_transformed_tsrne, labels, test_size=0.33, shuffle=False)

model = LogisticRegressionCV(cv=5, Cs=[0.001, 0.01, 0.1, 1, 10], penalty='elasticnet', solver='saga',
                             l1_ratios=[0.25, 0.5, 0.75], multi_class='ovr', max_iter=5000) ## Takes a looong time

model.fit(X_train, y_train)

LogisticRegressionCV
LogisticRegressionCV(Cs=[0.001, 0.01, 0.1, 1, 10], cv=5,
                     l1_ratios=[0.25, 0.5, 0.75], max_iter=5000,
                     multi_class='ovr', penalty='elasticnet', solver='saga')

score1 = model.score(X_test, y_test)
print("score1 Accuracy: {0:.4f}".format(score1))

score1 Accuracy: 0.9460
```

Figure 3. Model score of QRPCA-t-SNE [24]

```
y_pred = model.predict(X_test)

# Predict the classes on the test data, and return the probabilities
y_proba = model.predict_proba(X_test)

from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)

0.9528261793455164
```

Figure 4. Mean square error of QRPCA-t-SNE [24]

```
from sklearn.metrics import accuracy_score
y_train_pred = model.predict(X_train)

# Calculate training accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)

# Calculate testing accuracy
test_accuracy = accuracy_score(y_test, y_pred)

print("Training Accuracy: {0:.4f}".format(train_accuracy))
print("Testing Accuracy: {0:.4f}".format(test_accuracy))

Training Accuracy: 0.9403
Testing Accuracy: 0.9460
```

Figure 5. Training and testing accuracy of QRPCA-t-SNE [24]

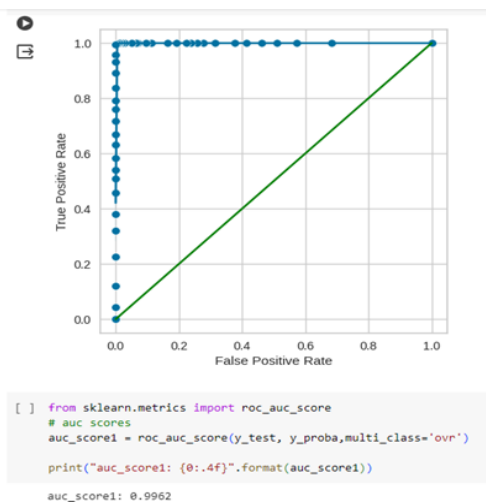


Figure 6. AUC score of QRPCA-t-SNE [24]

```
[ ] from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
print('Precision: %.4f' % precision_score(y_test, y_pred, average='macro'))
print('Recall: %.4f' % recall_score(y_test, y_pred, average='macro'))
print('Accuracy: %.4f' % accuracy_score(y_test, y_pred))
print('F1 Score: %.4f' % f1_score(y_test, y_pred, average='macro'))

Precision: 0.9511
Recall: 0.9459
Accuracy: 0.9460
F1 Score: 0.9472
```

Figure 7. Confusion matrix score QRPCA-t-SNE [24]

2.5.2. Model 2: uniform manifold approximation and projection

The same dataset 1 underwent preprocessing and was visualized using UMAP, as shown in Figure 8. Subsequently, the dataset was partitioned into training and testing sets, following which the model score was computed using the 'LogisticRegressionCV max iter=5,000 as depicted in Figure 9. Once we calculated the model score, we estimated the MSE amount in this model, shown in Figure 10. The training and testing results depicted in Figure 11 were analyzed to identify discrepancies, and the AUC score was utilized to evaluate the True and False Positive rates, illustrated in Figure 12. Finally, the F1 score, precision, recall, and accuracy scores thoroughly understood the model's performance, as demonstrated in Figure 13.

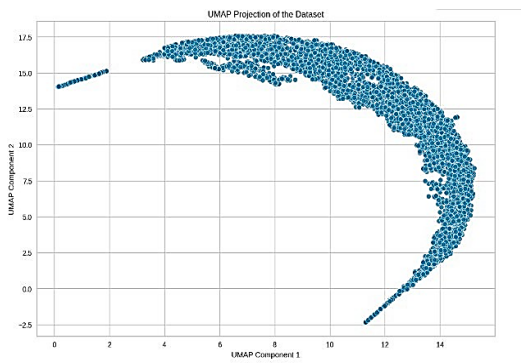


Figure 8. UMAP data visualization (self made)

```
X_train, X_test, y_train, y_test = train_test_split(umap_results, labels, test_size=0.33, shuffle=False)
model = LogisticRegressionCV(cv=5, Cs=[0.001, 0.01, 0.1, 1, 10], penalty='elasticnet', solver='saga',
                             l1_ratios=[0.25, 0.5, 0.75], multi_class='ovr', max_iter=5000) ## Takes a looong time

model.fit(X_train, y_train)
score = model.score(X_test, y_test)
print("score Accuracy: {0:.4f}".format(score))

score Accuracy: 0.9086
```

Figure 9. Model score of UMAP [24]

```
y_pred = model.predict(X_test)
# Predict the classes on the test data, and return the probability
y_proba = model.predict_proba(X_test)

from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)

0.8083297917552061
```

Figure 10. Mean square error of UMAP [24]

```
print("Training Accuracy: {0:.4f}".format(train_accuracy))
print("Testing Accuracy: {0:.4f}".format(test_accuracy))

Training Accuracy: 0.9008
Testing Accuracy: 0.9086
```

Figure 11. Training and testing accuracy of UMAP [24]

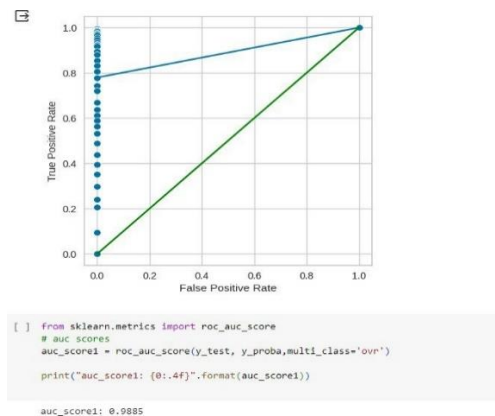


Figure 12. AUC score of UMAP [24]

```
[ ] from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
print('Precision: %.4f' % precision_score(y_test, y_pred, average='macro'))
print('Recall: %.4f' % recall_score(y_test, y_pred, average='macro'))
print('Accuracy: %.4f' % accuracy_score(y_test, y_pred))
print('F1 Score: %.4f' % f1_score(y_test, y_pred, average='macro'))

Precision: 0.9138
Recall: 0.9101
Accuracy: 0.9086
F1 Score: 0.9108
```

Figure 13. Confusion matrix score of UMAP [24]

2.5.3. Model 3: multidimensional scaling

Again, we consider the next model, which is MDS. Firstly, the dataset underwent preprocessing, done by the previous stage at the first model, and then visualized using MDS as depicted in Figure 14. Subsequently, the dataset was partitioned into training and testing sets, following which the model score was computed using the LogisticRegressionCV with max_iter=5,000, as shown in Figure 15. As illustrated in Figure 16, the Mean Squared Error method was employed to calculate the average squared difference between estimated and actual values. The training and testing results were analyzed to identify discrepancies, as shown in Figure 17, and the AUC score was utilized to evaluate the true and false positive rates, as shown in Figure 18. Finally, the F1 score, precision, recall, and Accuracy scores thoroughly understood the model's performance, as demonstrated in Figure 19.

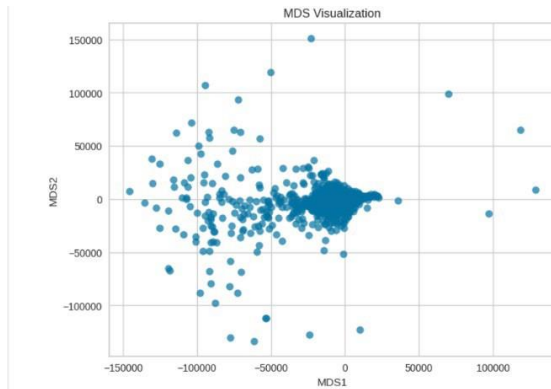


Figure 14. Data visualization using MDS (self-made)

```
X_train, X_test, y_train, y_test = train_test_split(mds_results, labels, test_size=0.33, shuffle=False)
model = LogisticRegressionCV(cv=5, cs=[0.001, 0.01, 0.1, 1, 10], penalty='elasticnet', solver='saga',
                             l1_ratios=[0.25, 0.5, 0.75], multi_class='ovr', max_iter=5000) ## Takes a looong time
model.fit(X_train, y_train)

LogisticRegressionCV
LogisticRegressionCV(cv=5, cs=[0.001, 0.01, 0.1, 1, 10], cv=5,
                    l1_ratios=[0.25, 0.5, 0.75], max_iter=5000,
                    multi_class='ovr', penalty='elasticnet', solver='saga')

score1 = model.score(X_test, y_test)
print("score1 Accuracy: {:.4f}".format(score1))

score1 Accuracy: 0.2835
```

Figure 15. Model score of MDS [24]

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)

8.501062473438164
```

Figure 16. Mean square error of MDS [24]

```
# Calculate training accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)

# Calculate testing accuracy
test_accuracy = accuracy_score(y_test, y_pred)

print("Training Accuracy: {:.4f}".format(train_accuracy))
print("Testing Accuracy: {:.4f}".format(test_accuracy))

Training Accuracy: 0.2693
Testing Accuracy: 0.2835
```

Figure 17. Training and testing accuracy of MDS [24]

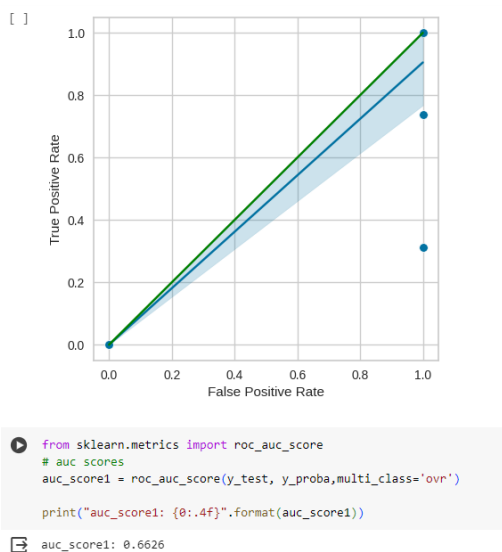


Figure 18. AUC score of MDS [24]

```
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
print('Precision: {:.4f}'.format(precision_score(y_test, y_pred, average='macro')))
print('Recall: {:.4f}'.format(recall_score(y_test, y_pred, average='macro')))
print('Accuracy: {:.4f}'.format(accuracy_score(y_test, y_pred)))
print('F1 Score: {:.4f}'.format(f1_score(y_test, y_pred, average='macro')))

Precision: 0.2495
Recall: 0.3221
Accuracy: 0.2835
F1 Score: 0.1859
```

Figure 19. Confusion matrix score of MDS [24]

2.6. Dataset 2

Dataset 2 covers the detailed model implementation of (QRPCA-t-SNE), UMAP, and MDS. Dataset 2 is accompanied with the evaluation parameters like model score, mean squared error, training, and testing. Results: the AUC score (area under the ROC Curve), ROC score, precision, recall, accuracy, and F1-score.

2.6.1. Model 1: QRPCA - t- SNE model

First, we imported Dataset 2, which is displayed in Figure 20. Next, we applied the preprocessing step to visualize the dataset using t-SNE with a perplexity of 40, as illustrated in Figure 21. After that, we split the data into training and testing sets and calculated the model score using LogisticRegressionCV with max_iter=5,000, as demonstrated in Figure 22, to achieve a model score. Once we computed the model score, we estimated the MSE amount in this model, shown in Figure 23. The next step was to examine the training and testing results displayed in Figure 24. Before the confusion matrix, we checked the AUC score to verify the true and false positive rates, as demonstrated in Figure 25. Finally, we utilized the F1, precision, recall, and accuracy scores to evaluate the model, as shown in Figure 26.

```
df=np.asarray(df_train)
jason=df
print(df.shape)
(10122, 387)
```

Figure 20. Dataset_2 size (self-made)

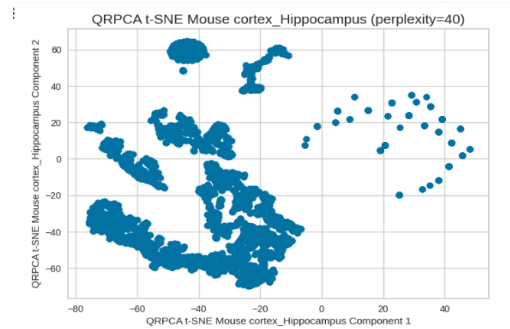


Figure 21. QRPCA-t-SNE data visualization [24]

```
X_train, X_test, y_train, y_test = train_test_split(X_transformed_tsn, labels, test_size=0.33, shuffle=False)
model = LogisticRegressionCV(cv=5, Cs=[0.001, 0.01, 0.1, 1, 10], penalty='elasticnet', solver='saga',
                             l1_ratios=[0.25, 0.5, 0.75], multi_class='ovr', max_iter=5000) ## Takes a looonng time
model.fit(X_train, y_train)
[35] score1 = model.score(X_test, y_test)
print("score1 Accuracy: {0:.4f}".format(score1))
score1 Accuracy: 0.9740
```

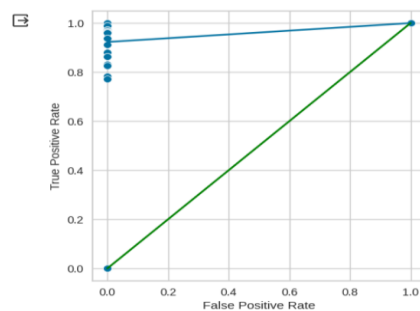
Figure 22. Model score of QRPCA-t-SNE [24]

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
0.08260999700688416
```

Figure 23. Mean square error of QRPCA-t-SNE [24]

```
from sklearn.metrics import accuracy_score
y_train_pred = model.predict(X_train)
# Calculate training accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
# Calculate testing accuracy
test_accuracy = accuracy_score(y_test, y_pred)
print("Training Accuracy: {0:.4f}".format(train_accuracy))
print("Testing Accuracy: {0:.4f}".format(test_accuracy))
Training Accuracy: 0.9749
Testing Accuracy: 0.9740
```

Figure 24. Training and testing accuracy of QRPCA-t-SNE [24]



```
from sklearn.metrics import roc_auc_score
# auc scores
auc_score1 = roc_auc_score(y_test, y_proba, multi_class='ovr')
print("auc_score1: {0:.4f}".format(auc_score1))
auc_score1: 0.9976
```

Figure 25. AUC score of QRPCA-t-SNE [24]

```

from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
print('Precision: %.4f' % precision_score(y_test, y_pred, average='macro'))
print('Recall: %.4f' % recall_score(y_test, y_pred, average='macro'))
print('Accuracy: %.4f' % accuracy_score(y_test, y_pred))
print('F1 Score: %.4f' % f1_score(y_test, y_pred, average='macro'))
    
```

Precision: 0.9522
 Recall: 0.9356
 Accuracy: 0.9740
 F1 Score: 0.9374

Figure 26. Confusion matrix score of QRPCA-t-SNE [24]

2.6.2. Model 2: uniform manifold approximation and projection

The same dataset 2 underwent preprocessing and was visualized using UMAP, as shown in Figure 27. Subsequently, the dataset was partitioned into training and testing sets, following which the model score was computed using the 'LogisticRegressionCV max iter=5,000 as depicted in Figure 28. Once we calculated the model score, we estimated the MSE amount in this model, shown in Figure 29. The training and testing results depicted in Figure 30, were analyzed to identify discrepancies, and the AUC score was utilized to evaluate the True and false positive rates, illustrated in Figure 31. Finally, the F1 score, precision, recall, and Accuracy scores thoroughly understood the model's performance, as demonstrated in Figure 32.

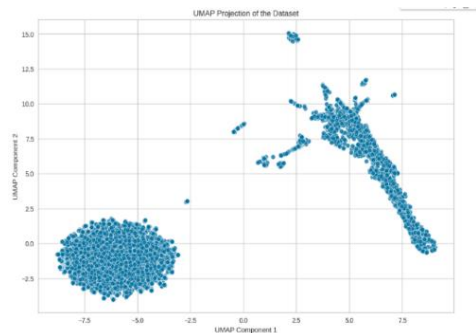


Figure 27. Dataset_2 visualization using UMAP (self-made)

```

X_train, X_test, y_train, y_test = train_test_split(umap_results, labels, test_size=0.33, shuffle=False)
model = LogisticRegressionCV(cv=5, Cs=[0.001, 0.01, 0.1, 1, 10], penalty='elasticnet', solver='saga',
                             l1_ratios=[0.25, 0.5, 0.75], multi_class='ovr', max_iter=5000) ## Takes a looonng ti
model.fit(X_train, y_train)
score = model.score(X_test, y_test)
print("score1 Accuracy: {0:.4f}".format(score))
    
```

score1 Accuracy: 0.9297

Figure 28. Model score of UMAP [24]

```

y_pred = model.predict(X_test)
# Predict the classes on the test data, and return the proba
y_proba = model.predict_proba(X_test)

from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
    
```

0.8608201137384017

Figure 29. Mean square error of UMAP [24]

```

[58] print("Training Accuracy: {0:.4f}".format(train_accuracy))
print("Testing Accuracy: {0:.4f}".format(test_accuracy))
    
```

Training Accuracy: 0.9336
 Testing Accuracy: 0.9297

Figure 30. Training and testing accuracy of UMAP [24]

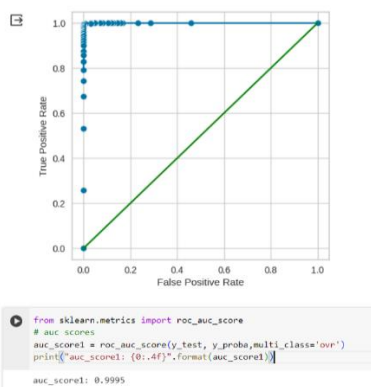


Figure 31. AUC score of UMAP [24]

```

[66] from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
print('Precision: %.4f' % precision_score(y_test, y_pred, average='macro'))
print('Recall: %.4f' % recall_score(y_test, y_pred, average='macro'))
print('Accuracy: %.4f' % accuracy_score(y_test, y_pred))
print('F1 Score: %.4f' % f1_score(y_test, y_pred, average='macro'))
    
```

Precision: 0.7730
 Recall: 0.7404
 Accuracy: 0.9297
 F1 Score: 0.7350

Figure 32. Confusion matrix score of UMAP [24]

2.6.3. Model 3: multidimensional scaling

Again, we consider the next model, which is MDS. Firstly, the dataset underwent preprocessing, done by the previous stage at the first model, and then visualized using MDS as depicted in Figure 33. Subsequently, the dataset was partitioned into training and testing sets, following which the model score was computed using the LogisticRegressionCV with max_iter=5,000, as shown in Figure 34. As illustrated in Figure 35, the mean squared error method was employed to calculate the average squared difference between estimated and actual values. The training and testing results were analyzed to identify discrepancies, as shown in Figure 36, and the AUC score was utilized to evaluate the true and false positive rates, as shown in Figure 37. Finally, the F1 score, precision, recall, and accuracy scores thoroughly understood the model's performance, as demonstrated in Figure 38.

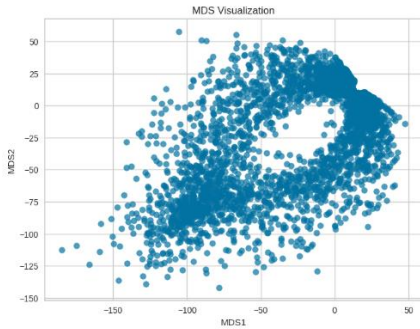


Figure 33. Data visualization using MDS (self-made)

```

X_train, X_test, y_train, y_test = train_test_split(mds_results, labels, test_size=0.33, shuffle=False)
model = LogisticRegressionCV(cv=5, Cs=[0.001, 0.01, 0.1, 1, 10], penalty='elasticnet', solver='saga',
                             l1_ratios=[0.25, 0.5, 0.75], multi_class='ovr', max_iter=5000) # Takes a loooooong
model.fit(X_train, y_train)

LogisticRegressionCV
LogisticRegressionCV(Cs=[0.001, 0.01, 0.1, 1, 10], cv=5,
                     l1_ratios=[0.25, 0.5, 0.75], max_iter=5000,
                     multi_class='ovr', penalty='elasticnet', solver='saga')

[73] score1 = model.score(X_test, y_test)
print("score1 Accuracy: {}".format(score1))

score1 Accuracy: 0.4750
    
```

Figure 34. Model score of MDS [24]

```

[75] from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)

5.98084405866507
    
```

Figure 35. Mean square error of MDS [24]

```

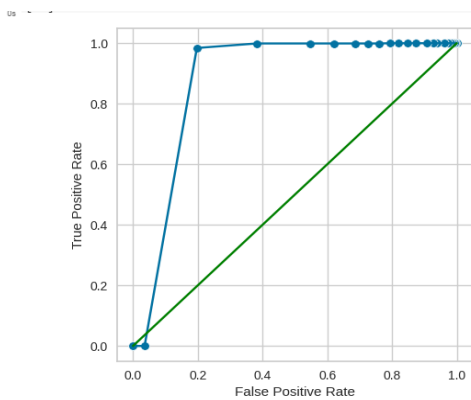
[77] # Calculate training accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)

# Calculate testing accuracy
test_accuracy = accuracy_score(y_test, y_pred)

print("Training Accuracy: {:.4f}".format(train_accuracy))
print("Testing Accuracy: {:.4f}".format(test_accuracy))

Training Accuracy: 0.4818
Testing Accuracy: 0.4750
    
```

Figure 36. Training and testing accuracy of MDS [24]



```

[85] from sklearn.metrics import roc_auc_score
# auc scores
auc_score1 = roc_auc_score(y_test, y_proba, multi_class='ovr')
print("auc_score1: {:.4f}".format(auc_score1))

auc_score1: 0.8363
    
```

Figure 37. AUC score of MDS [24]

```

[85] from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
print("Precision: {:.4f} % precision_score(y_test, y_pred, average='macro')")
print("Recall: {:.4f} % recall_score(y_test, y_pred, average='macro')")
print("Accuracy: {:.4f} % accuracy_score(y_test, y_pred)")
print("F1 Score: {:.4f} % f1_score(y_test, y_pred, average='macro')")

Precision: 0.4371
Recall: 0.4249
Accuracy: 0.4750
F1 Score: 0.3707
    
```

Figure 38. Confusion matrix score of MDS [24]

3. RESULTS AND DISCUSSION

We have used the Mann-Whitney U Test to show that our proposed model is better than other tools, namely UMAP and MDS. This test compares multiple models with different datasets. We first checked the normality of the data by applying the Anderson-Darling test to both datasets, as shown in Figure 39. Similarly, we used the Anderson-Darling test for the second dataset, also depicted in Figure 40. Both datasets did not follow the normal distribution based on the Anderson-Darling test. To establish the model's superiority over MDS and LDAs, hypothesis testing was conducted using the Mann-Whitney U statistical test:

H0: No difference in the median accuracy scores between the two models being compared.

H1: Model 1(QRPCA-t-SNE) exhibits a statistically higher median accuracy compared to either Model 2 (UMAP) or Model 3 (MDS).

The Mann-Whitney U statistical test is commonly used to compare two data groups. By default, it is implemented with a 95% confidence interval, and an α is 0.05 through `scipy.stats`. In this study, we used the Mann-Whitney U Statistical Test to compare Model 1 with Model 2 and Model 1 with Model 3. The results are shown in Figure 41, and the p-values for both comparisons are 0.00265 and 0.000296, respectively.

For the first comparison, the p-value of 0.00265 is less than the ($\alpha=0.05$), which means we reject the null hypothesis at a 95% confidence interval. This indicates a statistically significant difference between Model 1 and Model 2. Similarly, for the second comparison, the p-value of 0.000296 is less than ($\alpha=0.05$), leading us to reject the null hypothesis at a confidence level of α . This suggests a difference in the median accuracy scores between the two models (MDS and UMAP). In summary, the Mann-Whitney U Statistical Test helped us to identify significant differences between the models and support our objective.

```

import numpy as np
from scipy.stats import anderson

# Assuming 'data' is your (31052, 387) dataset
# Let's focus on the first variable/column
variable_data = jason[:, 0] # Replace 'data' with your actual numpy array

# Anderson-Darling test for the selected variable
result = anderson(variable_data)
print('Anderson-Darling Test for Variable 1:')
print(f"Statistic: {result.statistic:.4f}")
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    if result.statistic < cv:
        print(f" {sl}%: Data looks normal (fail to reject H0).")
    else:
        print(f" {sl}%: Data does not look normal (reject H0).")

```

Anderson-Darling Test for Variable 1:
Statistic: 1390.5779
15.0%: Data does not look normal (reject H0).
10.0%: Data does not look normal (reject H0).
5.0%: Data does not look normal (reject H0).
2.5%: Data does not look normal (reject H0).
1.0%: Data does not look normal (reject H0).

Figure 39. Anderson-Darling for dataset 1(self-made)

```

import numpy as np
from scipy.stats import anderson

variable_data = jason[:, 0] # Replace 'data' with your actual numpy array
# Anderson-Darling test for the selected variable
result = anderson(variable_data)
print('Anderson-Darling Test for Variable 1:')
print(f"Statistic: {result.statistic:.4f}")
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    if result.statistic < cv:
        print(f" {sl}%: Data looks normal (fail to reject H0).")
    else:
        print(f" {sl}%: Data does not look normal (reject H0).")

```

Anderson-Darling Test for Variable 1:
Statistic: 3226.2944
15.0%: Data does not look normal (reject H0).
10.0%: Data does not look normal (reject H0).
5.0%: Data does not look normal (reject H0).
2.5%: Data does not look normal (reject H0).
1.0%: Data does not look normal (reject H0).

Figure 40. Anderson-Darling for dataset 2 (self-made)

```

import numpy as np
from scipy.stats import mannwhitneyu

# Combined accuracies for Model 1 from both datasets
accuracy_model_1 = [0.9086, 0.9962, 0.9459, 0.9511, 0.9460, 0.9472, 0.9740, 0.9976, 0.9522, 0.9356, 0.9356, 0.9374, 0.082, 0.9749, 0.740, 0.9528, 0.9403, 0.9460]

# Combined accuracies for Model 2 from both datasets
accuracy_model_2 = [0.8083, 0.9008, 0.9086, 0.8608, 0.9336, 0.9297, 0.9086, 0.9138, 0.9101, 0.9086, 0.9108, 0.9885, 0.9297, 0.9995, 0.7730, 0.7404, 0.9297, 0.7350]

# Combined accuracies for Model 3 from both datasets
accuracy_model_3 = [8.5010, 0.2693, 0.2835, 5.980, 0.4818, 0.4750, 0.2835, 0.6626, 0.2495, 0.3221, 0.2835, 0.1859, 0.4750, 0.8363, 0.4371, 0.4249, 0.4750, 0.3707]

# Mann-Whitney U Test to compare models
# Model 1 vs Model 2
stat_12, p_value_12 = mannwhitneyu(accuracy_model_1, accuracy_model_2, alternative='greater')

# Model 1 vs Model 3
stat_13, p_value_13 = mannwhitneyu(accuracy_model_1, accuracy_model_3, alternative='greater')

p_value_12, p_value_13

```

Figure 41. Mann Whitney U test result (self-made)

4. CONCLUSION

Our comprehensive analysis revealed the performance of the QRPCA-t-SNE model over both UMAP and MDS across various vital accuracy parameters. The Mann-Whitney U test is a non-parametric method for Model proving with the Anderson-Darling test to check the data normality. The p-values of Model 1(QRPCA-t-SNE) vs. Model 2 (UMAP) is 0.0027, which is less than ($\alpha = 0.05$), which gives evidence to reject H_0 (null hypothesis). Similarly, the p-value of Model 1(QRPCA-t-SNE) vs. Model 3(MDS) p-values is 0.0003, which is less than the ($\alpha = 0.05$). Additionally, QRPCA-t-SNE performs consistently across training and testing accuracy, mean square error, model accuracy, AUC scores, precision, recall, and F1 scores better than other models, which are UMAP and MDS. The model is tested via two datasets with different sources in the same genomic field. Not only does QRPCA-t-SNE demonstrate its effectiveness in predictive modeling, but it also highlights the potential for exploring its scalability and adaptability to various data-intensive domains.




REFERENCES

- [1] S. Velliangiri, S. Alagumuthukrishnan, and S. I. Thankumar Joseph, "A review of dimensionality reduction techniques for efficient somputation," *Procedia Computer Science*, vol. 165, pp. 104–111, 2019, doi: 10.1016/j.procs.2020.01.079.
- [2] M. A. Salam, A. T. Azar, M. S. Elgandy, and K. M. Fouad, "The effect of different dimensionality reduction techniques on machine learning overfitting problem," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 4, pp. 641–655, 2021, doi: 10.14569/IJACSA.2021.0120480.
- [3] M. F. Kabir, T. Chen, and S. A. Ludwig, "A performance analysis of dimensionality reduction algorithms in machine learning models for cancer prediction," *Healthcare Analytics*, vol. 3, p. 100125, Nov. 2023, doi: 10.1016/j.health.2022.100125.
- [4] Y. R. Yeh, S. Y. Huang, and Y. J. Lee, "Nonlinear dimension reduction with kernel sliced inverse regression," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 11, pp. 1590–1603, Nov. 2009, doi: 10.1109/TKDE.2008.232.
- [5] L. Xu *et al.*, "Dimensionality reduction for visualizing spatially resolved profiling data using spasne," 2024, doi: 10.2139/ssrn.4689296.
- [6] A. Maćkiewicz and W. Ratajczak, "Principal components analysis (PCA)," *Computers and Geosciences*, vol. 19, no. 3, pp. 303–342, Mar. 1993, doi: 10.1016/0098-3004(93)90090-R.
- [7] T. Kohonen, "The self-organizing map, a possible model of brain maps," in *Medical and Biological Engineering and Computing*, vol. 34, no. SUPPL. 1, Psychology Press, 1996, pp. 5–8.
- [8] P. Horst, "Theory and methods of scaling . warren S. Torgerson. Wiley, New York; Chapman and Hall, London, 1958. xiii + 460 pp. Illus. \$9.50.," *Science*, vol. 129, no. 3348, pp. 560–560, Feb. 1959, doi: 10.1126/science.129.3348.560-a.
- [9] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, Dec. 2000, doi: 10.1126/science.290.5500.2323.
- [10] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, Dec. 2000, doi: 10.1126/science.290.5500.2319.
- [11] K. Q. Weinberger and L. K. Saul, "An introduction to nonlinear dimensionality reduction by maximum variance unfolding," *In AAAI*, vol. 6, pp. 1683–1686, 2006.
- [12] R. Taghizadeh-Mehrjardi *et al.*, "High-performance soil class delineation via UMAP coupled with machine learning in Kurdistan Province, Iran," *Geoderma Regional*, vol. 36, p. e00754, Mar. 2024, doi: 10.1016/j.geodrs.2024.e00754.
- [13] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of machine learning research*, vol. 9, no. 11, 2008.
- [14] H. Liu *et al.*, "Using t-distributed stochastic Neighbor embedding (t-SNE) for cluster analysis and spatial zone delineation of groundwater geochemistry data," *Journal of Hydrology*, vol. 597, p. 126146, Jun. 2021, doi: 10.1016/j.jhydrol.2021.126146.
- [15] G. C. Linderman, M. Rachh, J. G. Hoskins, S. Steinerberger, and Y. Kluger, "Fast interpolation-based t-SNE for improved visualization of single-cell RNA-seq data," *Nature Methods*, vol. 16, no. 3, pp. 243–245, Feb. 2019, doi: 10.1038/s41592-018-0308-4.
- [16] G. Dimitriadis, J. P. Neto, and A. R. Kampff, "t-SNE visualization of large-scale neural recordings," *Neural Computation*, vol. 30, no. 7, pp. 1750–1774, Jul. 2018, doi: 10.1162/neco_a_01097.




- [17] T. T. Cai and R. Ma, "Theoretical foundations of t-SNE for visualizing high-dimensional clustered data," *The Journal of Machine Learning Research*, vol. 23, no. 1, pp. 13581–13634, 2022.
- [18] A. Chatzimpampas, R. M. Martins, and A. Kerren, "T-viSNE: Interactive assessment and interpretation of t-SNE projections," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 8, pp. 2696–2714, Aug. 2020, doi: 10.1109/TVCG.2020.2986996.
- [19] T. Chari and L. Pachter, "The specious art of single-cell genomics," *PLoS Computational Biology*, vol. 19, no. 8, p. e1011288, Aug. 2023, doi: 10.1371/journal.pcbi.1011288.
- [20] E. J. Amézquita, F. Nasrin, K. M. Storey, and M. Yoshizawa, "Genomics data analysis via spectral shape and topology," *PLoS ONE*, vol. 18, no. 4 April, p. e0284820, Apr. 2023, doi: 10.1371/journal.pone.0284820.
- [21] M. P. Das, V. K. Dhar, S. Verma, and K. K. Yadav, "Dimensionality reduction and sensitivity improvement for TACTIC Cherenkov data using t-SNE machine learning algorithm," *Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 1057, p. 168683, Dec. 2023, doi: 10.1016/j.nima.2023.168683.
- [22] M. C. Hout, M. H. Papesch, and S. D. Goldinger, "Multidimensional scaling," *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 4, no. 1, pp. 93–103, Oct. 2013, doi: 10.1002/wcs.1203.
- [23] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.
- [24] M. Ali, J. Choudhary, and T. Kasbe, "A hybrid model for data visualization using linear algebra methods and machine learning algorithm," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 33, no. 1, pp. 463–475, Jan. 2024, doi: 10.11591/ijeecs.v33.i1.pp463-475.
- [25] QR-Decomposition Algorithm, web. Available: <https://rpubs.com/aaronsc32/qr-decomposition-gram-schmidt>. (accessed Dec.23 2023)
- [26] Allen Brain Atlas, web. Available: <https://portal.brain-map.org/>. (accessed Dec.23 2023)

BIOGRAPHIES OF AUTHORS



Mr. Mohsin Ali    is an Assistant Professor in Medi-Caps University in the Department of Computer Science. He has completed BCA (2016), and MCA (2018). He is a Research scholar at Medi-Caps University. His research areas are Machine learning and Data science. He has done Certain certifications from the Massachusetts Institute of Technology from edX like machine learning, introduction to probability, and fundamentals of statistics. He has five years of Teaching experience. He can be contacted at email: coolbuddy.next.door@gmail.com.



Dr. Jitendra Choudhary    was born in 1984 at Dewas, M.P. India. He received his B.Sc. degree in Computer Science from Holkar Science College, Indore in 2003, M.Sc. degree in Computer Science in 2005, and his M.Tech. degree in Computer Science (with Distinction) in 2010 from SCSIT, Devi Ahilya University Indore. He received his Ph.D. degree from Devi Ahilya University Indore in 2014. His areas are software engineering and software testing. His research area includes Extreme Programming and Software Maintenance. He has published more than 20 research paper in reputed international journal and conferences. He has received Gold Medal (AIR-1) in Software Engineering course run by IIT Kharagpur through Swayam-NPTEL. He is an Associate Professor and HOD, CS at Medi-Caps University, Indore, M.P., India. He has 17 years of teaching work experience at the UG and PG levels. He can be contacted at email: jitendra.scsit@gmail.com.