Different Machine Learning Approaches to Quantify the Relationship of Spectral Data on Virus Load Estimation and Virus Classification

Module Report

submitted towards differentiation for the degree of

Msc Artificial Intelligence

by

Joseph McInerney

EEECS

Queen's University Belfast

Lecturer
Professor Hui Wang

November 2024

Contents

1 | Introduction

The link between viral load and virus severity was established with Covid-19 [1]. This motivates further the importance of potentially being able to predict viral load using machine learning techniques on spectral data. This would permit, not only virus classification [4] but virus severity predictions enabling quick and proportional treatment. In attempting to cluster the viral load levels, one could understand further the underlying patterns in spectral data that correlate to virus severity.

2 | Methodology

2.1 Regression

2.1.1 Data Preprocessing

Data was read from an Excel spreadsheet with access provided by Queen's University, Belfast into a Pandas DataFrame in order to manipulate the data. As the data had previously been used for a study, the data was relatively clean and minimal preprocessing steps had to be taken. It was established that there were no missing values in the data set. The data set was of size $N = 3801$ and $p = 512$. The data was split based on method being either *'dmem'* or *'pbs'*. *Dmem had 1831* entries compared to *pbs* which had 1970 entries. A correlation matrix for both sub data sets was generated in order to assess co-linearity.

The design matrix X was standardized using Sklearn's StandardScalar so that each of the variables has mean 0 in order to prepare the data for Principal Components Analysis (PCA) [2, p.512].

Outliers were removed using Sklearn's supervised algorithm IsolationForest(). Isolation forest leverages properties of outliers being uncommon and unusual to isolate them [3]. It has linear time complexity, so is ideal for the task at hand in removing outliers with a model reducing the bias seen in other techniques such as z-scores, and interquartile range where thresholds must be defined *prior* to outlier detection.

2.1.2 Feature Engineering

Spectrum data often contains wave lengths that are strongly correlated with each other. As such, particular attention to collinearity was taken. A correlation matrix of the features was generated for both data sets to demonstrate this common phenomenon when working with spectral data.

PCA was the first method to reduce dimensionality in order to eliminate collinearity. PCA projects the features to a lower dimensional space, while aiming to maintain the important information contained between the features. This can be useful for data visualisation, time complexity and coefficient estimation.

A closely related method for dimensionality reduction is Partial Least Squares. It is a supervised approach, as it identifies principal components relative to their covariance with the response variable. In the context of regression, PLS can be a more appropriate selection compared to PCA due to this guarantee that the features chosen will be the most effective at predicting the response variable[2, p.260,261], which is the viral load in this case, the target.

Another way to determine which features are of most importance to the model is using shrinkage methods. These methods shrink coefficient estimates which can significantly reduce their variance [2]. Lasso was included along with PCA and PLS as to serve the goal of inference. The Lasso algorithm shrinks the features to a subset of the original features, it can be directly interpreted as to which wave lengths are the most important when it comes to viral load estimation.

2.1.3 Model Selection

The ideal model is in regression is usually determined by the model that minimizes Mean Squared Error (MSE). MSE is the average distance of error terms from the signal. It can be represented as a combination of 3 components [2, p.32], as shown below in fig 2.1.

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\epsilon).$$

Figure 2.1: Expected Test MSE

Here we can see MSE represented as a function of the bias and variance of the model and the variance of error terms. Bias is the price one pays for modelling real world data using constrained models that have a priori assumptions. Variance is the extent to which the model would change if trained on different data. To minimize MSE, variance and bias should be low to prevent under-fitting (high bias) or over-fitting (high variance).

The approach taken for this regression task was to compare the MSE between models that differ in bias and variance in order to select a model that balances the two and minimizes MSE. The following are the list of models to be evaluated in order of increasing variance, and decreasing bias.

[Linear Regression, Polynomial Regression, Neural Network]

Linear regression will be used as a baseline to evaluate the various dimension reduction techniques. The design matrix of the most effective technique will then be used for both polynomial regression and a basic neural network (NN).

2.2 Classification

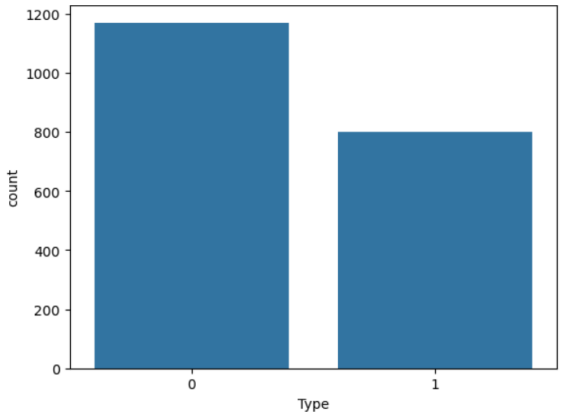
The aim of the classification method of this project was to reproduce the accuracy displayed previously in using spectral data to correctly identify and discriminate between virus types [4]. As such a PLS-DA was implemented. This model was then compared against other classic approaches to classification.

2.2.1 Preprocessing

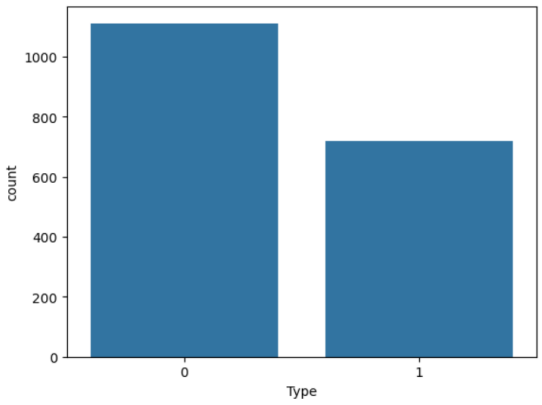
The virus is either of type *'X'* or of type *'Y'*. This was binary encoded, where *'X'* : 1 and *'Y'*: 0. This step was crucial as the data needed to be numeric for PLS. Then outliers were removed using Isolation Forest in for more robust classification modelling. The design matrix X remained the same from regression, being the standardized spectral data.

2.2.2 Feature Engineering

Cross validation was used in order to select the optimal number of components for PLS-DA to prevent multicollinearity present in spectral data. As shown below in figure 2.2, the two virus classes are relatively unbalanced.



(a) pbs



(b) dmem

Figure 2.2: A Consistent Moderate Virus Class Imbalance Across Datasets

PCA was used to generate the design matrix for the other classification approaches. To be cautious with parameter estimation, the *f1* metric was considered for model evaluation to combine both precision and recall, taking into consideration the minority class accuracy. However, *accuracy* was selected instead, to facilitate comparison to the original paper [4]. Classification accuracy is measured as follows in equation 2.1:

$$\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{TN} + \text{TP} + \text{FN} + \text{FP}} \tag{2.1}$$

2.2.3 Model Selection

The PLS-DA model was selected on the basis of its previous predictive power with similar data [4], then other classical models were selected in order to serve as comparison. These models were Gaussian Naive Bayes (NB), Logistic Regression, K Nearest Neighbours (KNN), Linear Discriminate Analysis (LDA) and Quadratic Discriminate Analysis (QDA). None of these models. The performance of these models can also be revealing of properties underpinning the data. The model selection also compares the inclusion of the target variable in dimension reduction, as PLS-DA includes the target variable, whereas the other approaches used PCA.

2.3 Clustering

2.3.1 Preprocessing

Generally, as clustering is an unsupervised learning algorithm, there is no target variable. However, in this case, we do have a target variable. The motivation for clustering even when the target variable is present is that often this is not the case. So, it is useful to implement unsupervised learning algorithms and evaluate them, even when the target variable is present. In this paper, the target variable is the viral load level. So, the viral load y, is split from the spectral data X.

The absence of missing values is verified, then outliers were removed. As the target variable is not to be known until evaluation, an unsupervised approach to outlier removal is selected. Local Outlier Factor (LOF) was selected. This is a density based algorithm, that identifies outliers as being in a sparser region compared to their neighbours, where, unlike z-scores or interquartile range, normality is not assumed, the relationships between wavelengths are considered and finally no a priori thresholds

are defined. All this makes LOF an appropriate outlier detection algorithm for spectral data prior to clustering.

2.3.2 Feature Engineering

In clustering, there are some important decisions to be made. These decisions may appear small, but they can have big impacts on clustering [2, p.532,533]. The correct decision is not always obvious, as such, it can often be helpful to compare variations of clustering method specifications. The choice whether to standardize the data before clustering is one such decision. For model specification, previously with classification and regression tasks, prediction can be deemed as being more important than inference. However, with clustering, while it can be used for predictions, often it can also make up part of the exploratory stage of research. So while standardization, can improve precision and robustness of statistical learning algorithms, it can also sometimes hide something that would not otherwise be seen. Therefore it was decided to perform clustering on two forms of the spectral data. One that was standardized and dimensionally reduced using PCA, and the other being the original spectral data with just outliers removed.

2.3.3 Model Selection

3 different models were selected. The K-means algorithm was selected as a classic approach. Scatter plot visualisation is only applicable to the PCA reduced data. However, K-means does require the number of clusters to be predefined. Different values of K can be compared, but ultimately must be previously specified.

To avoid this, agglomerative (bottom-up) hierarchical clustering (AGC) is implemented [2, p.525], which also provides an interpretable dendrogram that remains functional in high dimensions. The measures of dissimilarity that link leaves and branches can vary. The choice of measure can significantly impact the AGC clustering. As such all measures available using the SKlearn library are used. This includes: complete, average, single, and Ward. Complete linkage computes the largest between cluster dissimilarity. Average linkage competes the average between cluster dissimilarity. Single computes the minimum between cluster similarity. So these dissimilarity measures all measure dissimilarity taking different perspectives on what best quantifies the distance between two clusters. Ward, on the other hand takes a different, variance based approach, which merges clusters that minimize between cluster variance. All of these dissimilarity measures are to be compared.

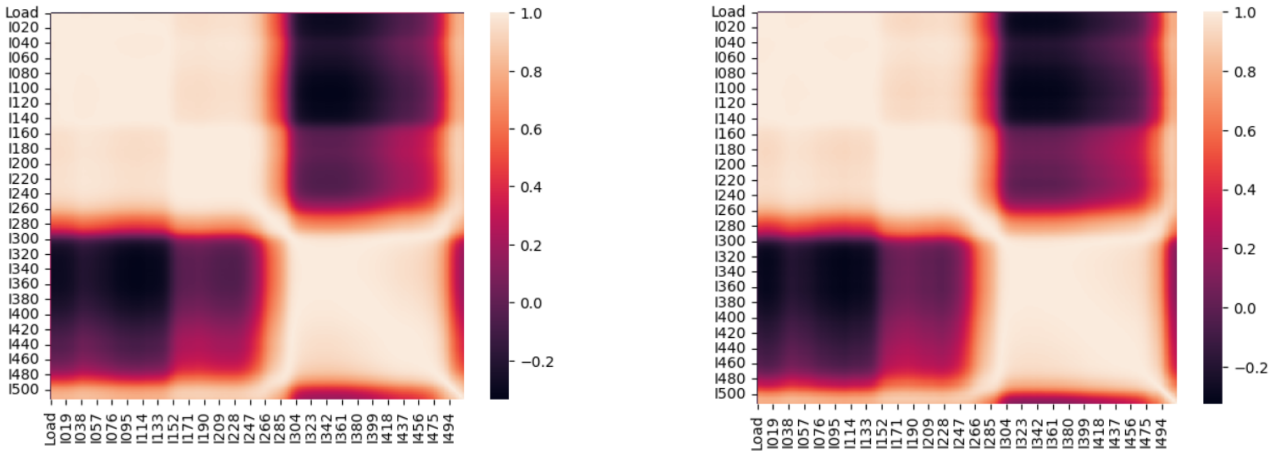
Finally the last model used in the clustering approach is DBSCAN. Where both K-means and AGC assign all data points a cluster, DBSAN can detect noise based on density and assign outliers. This advantage may be mitigated by the fact that the data was preprocessed in identifying outliers using LOF, another density based algorithm.

3 | Results

3.1 Regression

3.1.1 Multicollinearity

As displayed in figure 3.1 below, we can see lots of multicollinearity of the wave lengths. This motivated the preprocessing decision to reduce dimensionality before regression.



(a) Correlation Matrix of PBS dataset

(b) Correlation Matrix of DMEM dataset

Figure 3.1: Correlation Matrices of Features and Target Variable Load

3.1.2 Dimensionality Reduction Comparison

The following table 3.1 is a comparison of linear model performance comparing between PCA, PLS and Lasso.

	PCA	PLS	Lasso
pbs	3.25	3.48	5.27
dmem	15.26	5.58	5.14

Table 3.1: A comparison of **MSE** scores between linear regression models using features obtained from different dimensionality reduction techniques.

Here we can see variance of MSE between data sets and between dimension reduction methods. To gain an understanding of the underlying structure of our feature space, the number of features remaining after applying the various methods was also recorded as shown below in table 3.2

	1	2	3	4	NN
pbs	2.01	4.67	16.019	91.95	3.89
dmem	5.44	8.62	8.80	10.60	6.50

Table 3.3: MSE Comparison in Relation to Model Complexity

	PCA	PLS	Lasso
pbs	3	5	0
dmem	3	1	0

Table 3.2: Number of Features After Applying Dimension Reduction Methods

3.1.3 Bias Variance Model Comparison on MSE

The PLS reduced data was selected as the design matrix based on both the MSE values and the number of features across the various methods. PCA was ruled out due to the significantly poor MSE performance with the *dmem* data set. Lasso was rejected as its optimal number of components was 0. This means that systematic shrinking of features actually improved the model every single time until no features remained. 5-fold cross validation was used for all models to ensure robustness of results and appropriateness of comparison between models.

3.2 Classification

3.2.1 Model Comparison

The following table 3.4 compares the classification accuracy across classification approaches and data sets. It is important to note that all the models except for the PLS-DA model had their design matrix generated using PCA.

	PLS-DA	NB	LR	KNN	LDA	QDA
pbs	0.98	0.60	0.55	0.60	0.55	0.57
dmem	0.99	0.63	0.63	0.74	0.63	0.62

Table 3.4: Virus Type Classification **Accuracy** Comparison by Model

3.3 Clustering

3.3.1 K-means

As the goal is to see whether the clustering algorithm will group the data by viral load levels. The visual representation of K-means where K is set to 8 is the most pertinent as there are 8 different load levels. The following figure 3.2.

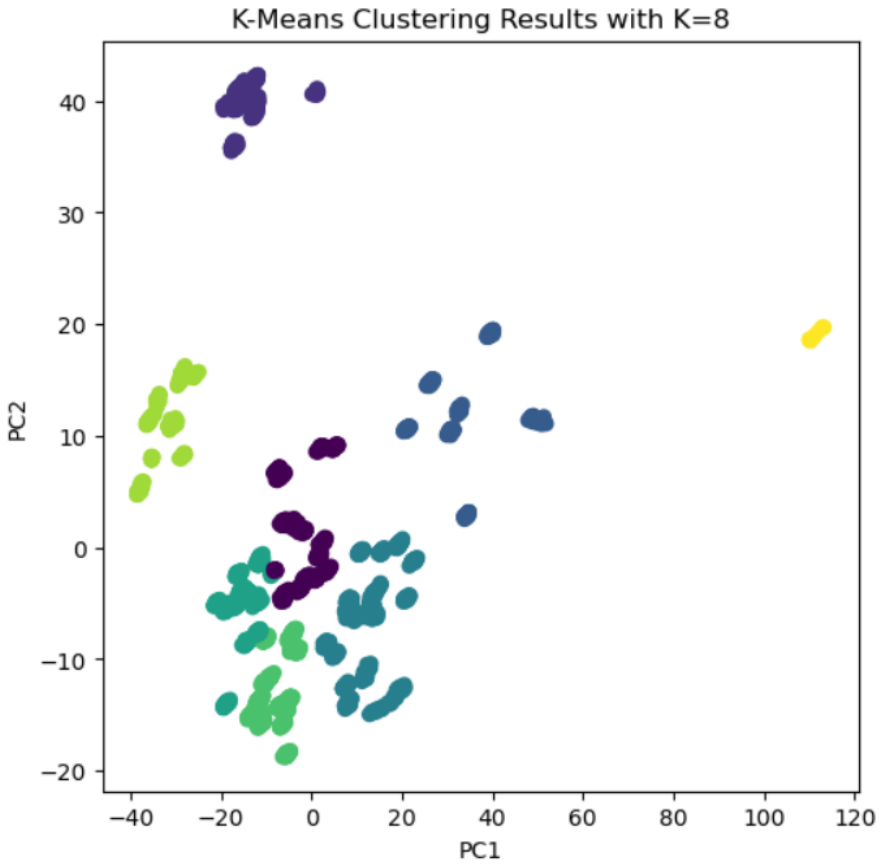
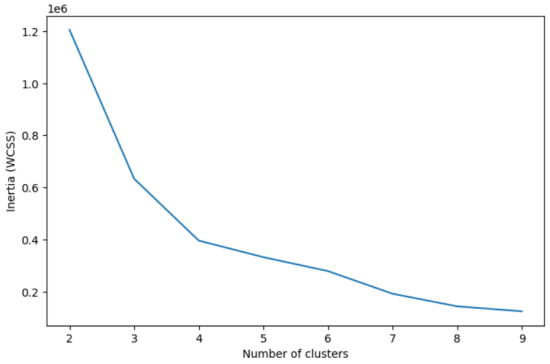
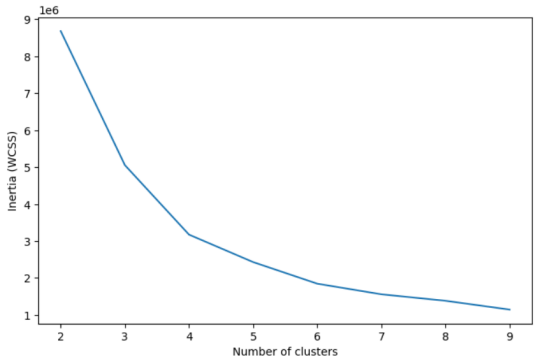


Figure 3.2: K-mans Clustering of PCA Reduced Data Using the First Two Prinicple Components where K is 8

Figure 3.3 represents the inertia scores relative to the number of clusters between 1 through to 10. This inertia is essentially the average distance between points in a cluster and that clusters centroid. This error term appears to be decreasing as K increases.



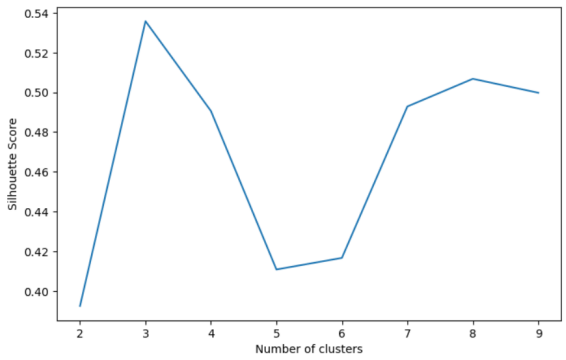
(a) Standardized and PCA Reduced Design Matrix



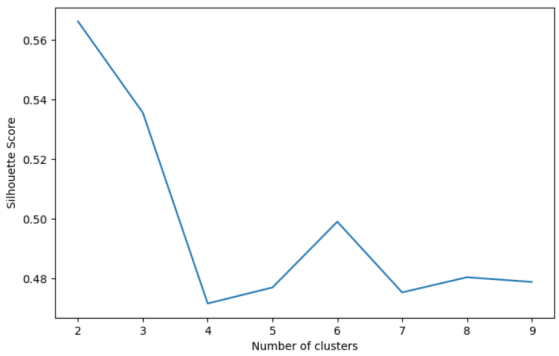
(b) Original Design Matrix

Figure 3.3: A Comparison of Inertia Between PCA Reduced X and Original X Relative to K

Figure 3.4 compares the silhouette scores between the two data sets. The silhouette score ranges from -1 to 1, and measures whether a data point is in the correct cluster. It does so by calculating the pairwise difference of a data point to its nearest centroid versus its assigned centroid. This is then normalized, where positive scores correspond to accurate cluster assignment.



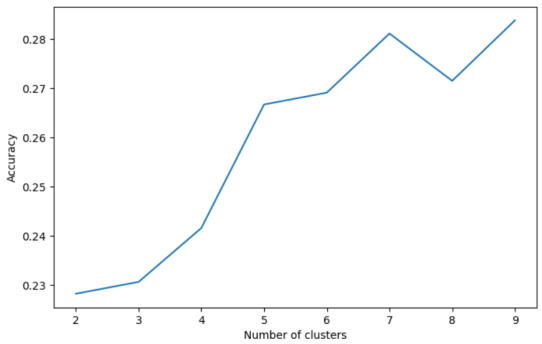
(a) Standardized and PCA Reduced Design Matrix



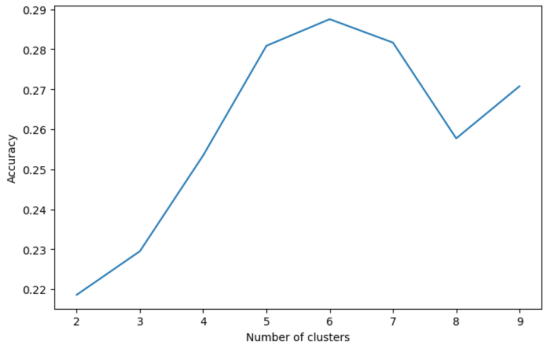
(b) Original Design Matrix

Figure 3.4: A Comparison of Silhouette Score Between PCA Reduced X and Original X Relative to K

The accuracy measure corresponds to the proportion of data points that are assigned to the correct cluster. This evaluation metric differs to the previous two, as it requires true values. Another comparison between the PCA dimensionally reduced, and original data is shown below in figure 3.5. Here we can see an upwards trend of accuracy as K increases with accuracy slightly higher in the PCA reduced design matrix.



(a) Standardized and PCA Reduced Design Matrix



(b) Original Design Matrix

Figure 3.5: A Comparison of Accuracy Scores Between PCA Reduced X and Original X Relative to K

3.3.2 AGC

No cutting of the dendrogram was predefined. The following figure 3.6 compares the clustering relative to the different linkage methods for the PCA dimensionally reduced data.

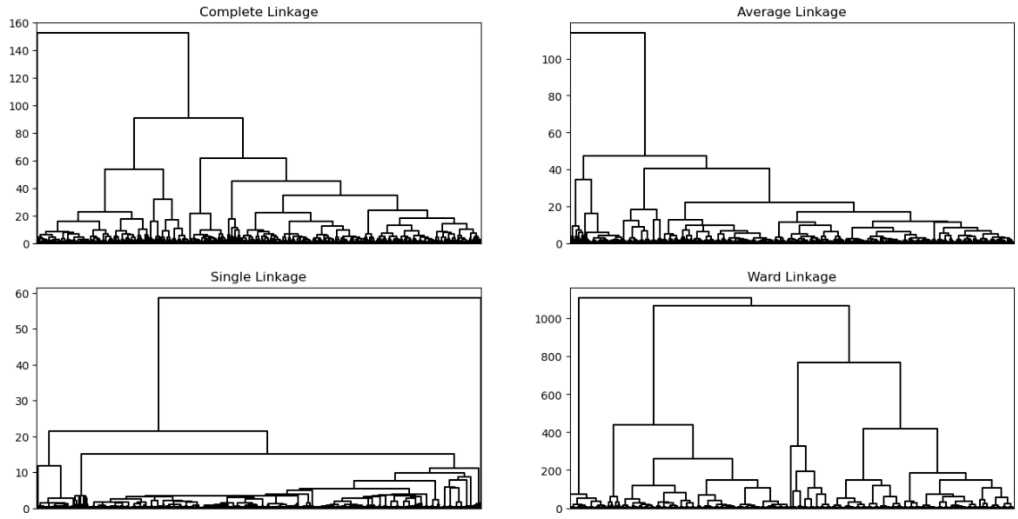


Figure 3.6: A Comparison of Dendrograms by Dissimilarity Metric Using AGC on PCA Dimensionally Reduced Data

As done with K-means, we will also compare this to AGC with the original design matrix. This can be seen below in figure 3.7

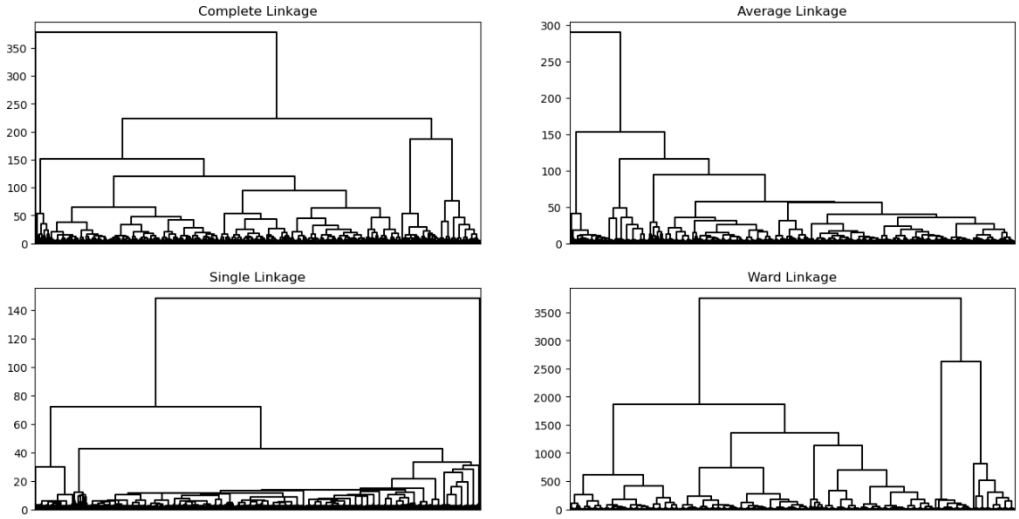


Figure 3.7: A Comparison of Dendrograms by Dissimilarity Metric Using AGC Original Design Matrix

3.3.3 DBSCAN

The following table 3.5 displays the results of the DBSCAN clustering.

	PCA	UNSTD
No. Clusters	30	124
No. Noise Points	0	176

Table 3.5: DBSCAN Clustering Results Displaying the Number of Cluster and Number of Noise Points Identified Relative to Whether X was PCA Dimensionally Reduced (PCA) or Not ($unstd$)

4 | Discussion

4.1 Regression

4.1.1 Mulicollinearity

The overview of the correlation matrix resembles much that of a 2 by 2 identity matrix. This leads to the hypothesis that there are only 2 uncorrelated features. Upon dimension reduction, there was no consensus among methods as to how many independent features there were in the data. PCA was the only method with consensus between data sets, it was also the only unsupervised method. This could indicate a significant difference between the two data sets in how the spectral data relates to the viral load.

4.1.2 Dimensionality Reduction Comparison

This importance of the target variable in reducing the data was echoed in dimension reduction, where PLS outperformed PCA.

4.1.3 Bias Variance Model Comparison on MSE

Ultimately, the linear model and the NN did not differ significantly. With the flexibility of the NN and its similitude to the linear model, it is possible that the NN just learned a linear relationship between the features and the viral load. This is further motivated by the increase in MSE as complexity increases, as seen in the polynomial models with degrees of freedom between 2 and 4.

4.2 Classification

The significantly higher accuracy of the PLS-DA model compared to the other models that used an unsupervised approach to generate the design matrix demonstrates the power of adapting PLS Regression, which classically is used on continuous data, for a classification approach. It shows the importance of the relationship between the target variable and the features in dimension reduction. The KNN model outperformed the other models.

KNN tends to outperform these other algorithms when there are proportionally few features and many observations [2, p.164]. The PCA reduced data had 3 features and over 1000 observations. The NB model performs relatively well, this could be due to the effective meeting of its assumption that features are independent. This is achieved as the design matrix after PCA has orthogonal features. The seemingly small sample size might also favour NB.

4.3 Clustering

4.3.1 K-means

The decrease in inertia as k increases is expected as smaller clusters naturally lead to less distance between points and centroids. This is consistent regardless of dimension reduction. No obvious elbow

is present, most noticeably, not at k = 8. So we can see that the data has not between sub-grouped by load level by the K-means algorithm. The magnitude of the distance between the two data sets is significantly different. The higher dimensional original data set has approximately 10 times the inertia. This is understandable as clusters become further apart in high dimensional space.

Accuracy and Silhouette scores on the other hand remain consistent regardless of dimension reduction. Silhouette scores seem to vary with no obvious pattern, this weak assigning of data points to their correct labels is demonstrated with accuracy scores of 25%. While 25% might not seem very precise, considering that there are 8 load levels, random guessing and balanced class would yield $\tilde{1}2.5\%$ accuracy. So, the clusters can actually be seen to double the probability of correctly assigning load level to a data point compared to random guessing.

4.3.2 AGC

Regardless of dissimilarity measure, all dendrograms except for Ward's linkage with the unstandardized data identify outliers in the data. This could indicate poor outlier detection with LOF. The number of neighbours used in LOF was not cross validated, this could have lead to more robust outlier identification and removal. The Ward linkage appears to generate the dendrogram with the most well-defined clusters. Potentially, Ward's focus on variance and not outliers has made it less susceptible to the outliers in the data. Overall, there is no evidence of the measures generating a tree with 8 clear and distinct clusters representing the different load levels.

4.3.3 DBSCAN

DBSCAN identified 176 noise points of the unstandardized and reduced data and 0 with the PCA reduced data. This was even after identifying outlier using LOF, and subsequently removing them. This highlights potential over identification of points as noise. This could potentially be from the trouble of using distance metric in high dimensional space, or maybe the violation of the assumption of independence that DBSCAN carries.

5 | Conclusion

5.1 Regression

For the estimation of viral loads, *pbs* method seem to provide data that is more effective at predicting viral load than *dmem* method. The spectral data does not predict the viral load to a great accuracy using the models tested. This could motivate exploring other potential predictors or modelling strategies.

5.2 Classification

The application of machine learning techniques such as PLS-DA can be highly accurate classifying viruses using spectral data. This could increase diagnostic capacity of disease, which is especially important in times of outbreak.

5.3 Clustering

The clustering approach demonstrated how important the small decisions, such as whether to standardize data and the number of clusters can have a large impact on evaluation metrics and visualization. Its exploration of the patterns underpinning the spectral data and how they might relate to load levels revealed similarly to the regression approach, that there was no obvious inference to be made using spectral data and load levels.

Bibliography

[1] Fajnzylber, J., Regan, J., Coxen, K., et al. 2020, Nature Communications, 11, 5493, doi: 10.1038/s41467-020-19057-5

[2] James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. 2023, An Introduction to Statistical Learning: with Applications in Python (Springer)

[3] Liu, F. T., Ting, K. M., & Zhou, Z.-H. 2008, 2008 Eighth IEEE International Conference on Data Mining, 413. https://api.semanticscholar.org/CorpusID:6505449

[4] Song, W., Wang, H., Power, U. F., et al. 2023, IEEE Sensors Journal, 23, 9981, doi: 10.1109/JSEN.2022.3207222

A | Appendix

A.1 Code

!/usr/bin/env python coding: utf-8

In[1]:

Standard Imports import numpy as np import pandas as pd import seaborn as sns import matplotlib.pyplot as plt

Data Preprocessing from sklearn.preprocessing import StandardScaler from sklearn.decomposition import PCA from mpl$_{t}oolkits.mplot3dimportAxes3Dfromsklearn.cross_{d}ecompositionimportPLSRegressionfr$

Regression from sklearn.model$_{s}electionimportcross_{v}alidate, cross_{v}al_{s}core, GridSearchCV, train_{t}est_{s}plit, KF$

Deep Learning import tensorflow as tf from tensorflow import keras from tensorflow.keras import layers from keras.models import Sequential from scikeras.wrappers import KerasRegressor from keras.layers import Dense, Input

Classification from sklearn.linear$_{m}odelimportLogisticRegressionfromsklearn.model_{s}electionimportcross_{v}al$

CLustering from sklearn.cluster import KMeans, DBSCAN from sklearn.metrics import silhouette$_{s}core, accura$

First Look at Dataset

In[3]:

virus$_{d}f = pd.read_{e}xcel('2022QUB_{v}irus_{d}ata.xlsx', sheet_{n}ame =' Data')$

In[4]:

virus$_{d}f.info()$

In[5]:

virus$_{d}f.head()$

In[6]:

Function for graphing null values in data frame def graph$_{n}ulls(data) : weather_{d}f = dataplt.figure(figsize = (10, 6))sns.heatmap(weather_{d}f.isnull(), cmap =' viridis', cbar = False)plt.title('NullValuesinEachColumn')pl$

graph$_{n}ulls(virus_{d}f)$

1. Regression

In[8]:

Subset data set by matrix, being careful of proceding whitespace ! dmem$_{v}irus_{d}f = virus_{d}f.loc[virus_{d}f['Matri$ $dmem']pbs_{v}irus_{d}f = virus_{d}f[virus_{d}f['Matrix'] ==' pbs']$

print(f'N — dmem: dmem$_{v}irus_{d}f.shape[0], pbs : pbs_{v}irus_{d}f.shape[0]')$

In[9]:

sns.heatmap(pbs$_{v}irus_{d}f.corr(numeric_{o}nly = True))$

In[10]:

sns.heatmap(dmem$_{v}irus_{d}f.corr(numeric_{o}nly = True))$

Relationship Between Viral Load and Spectrum Values

Split Target Variable and Design Matrix

In[13]:

Split predictor and response variables y$_{d}mem_{r}eg = dmem_{v}irus_{d}f['Load']X_{d}mem = dmem_{v}irus_{d}f.drop(['Load$ 1)

$y_{p}bs_{r}eg = pbs_{v}irus_{d}f['Load']X_{p}bs = pbs_{v}irus_{d}f.drop(['Load',' SID',' Type',' Matrix'], axis = 1)$

Target Variable Distribution

In[15]:

sns.countplot(data=dmem$_virus_df, x = y_dmem_reg$)

In[16]:

sns.countplot(data=pbs$_virus_df, x = y_pbs_reg$)

Standardize

In[18]:

Use standard scaler to standardize features, to prepare for feature engineering. scaler = StandardScaler() X$_dmem_std = scaler.fit_transform(X_dmem)X_pbs_std = scaler.fit_transform(X_pbs)$

Remove Outliers

In[20]:

Use isolation forest to remove outliers. def remove$_outliers_isoforest(X, y) : clf = IsolationForest(random_state42)outlier_pred = clf.fit_predict(X)RemoveOutliersidentifiedX_clean_standardized = X[outlier_pred ==1]y_clean = y[outlier_pred == 1]$

return X$_clean_standardized, y_clean, outlier_pred$

X$_dmem_std_clean, y_dmem_clean, outlier_pred_dmem = remove_outliers_isoforest(X_dmem_std, y_dmem_reg)X_pbs_std_cremove_outliers_isoforest(X_pbs_std, y_pbs_reg)$

In[21]:

dmem lr = LinearRegression() lr$_pca_dmem_results = cross_validate(lr, X_dmem_std_clean, y_dmem_clean, cv =5, scoring =' neg_mean_squared_error')print(lr_pca_dmem_results['test_score'])print(f'MSE : −np.mean(lr_pca_dmem_r$

In[22]:

pbs lr = LinearRegression() lr$_pca_dmem_results = cross_validate(lr, X_pbs_std_clean, y_pbs_clean, cv =5, scoring =' neg_mean_squared_error')print(lr_pca_dmem_results['test_score'])print(f'MSE : −np.mean(lr_pca_dmem_r$

In[ ]:

Dimension Reduction

PCA

In[25]:

def pca$_scree(X, range) : pca = PCA(n_components = range)scores = pca.fit_transform(X)$

fig, axes = plt.subplots(1, 2, figsize=(15, 6)) ticks = np.arange(pca.n$_components_)+1ax = axes[0]ax.plot(ticks, o')ax.set_xlabel('PrincipalComponent'); ax.set_ylabel('ProportionofVarianceExplained')ax.set_ylim([0, 1])ax.set_$

ax = axes[1] ax.plot(ticks, pca.explained$_variance_ratio_cumsum(), marker =' o')ax.set_xlabel('PrincipalCompo$

In[26]:

Scree plots for PCA in order to determine number of componentes, no titles as report in in Latex and easier to add titles then.

In[27]:

pca$_scree(X_dmem_std_clean, 11)$

In[28]:

pca$_scree(X_pbs_std_clean, 11)$

In[29]:

Set PCA to 3 components pca = PCA(n$_components = 3)X_pca_dmem = pca.fit_transform(X_dmem_std_clean)X_ppca.fit_transform(X_pbs_std_clean)$

Conovert back to df X$_pca_dmem_df = pd.DataFrame(X_pca_dmem, columns = ['PC1',' PC2',' PC3'])X_pca_pbs_df pd.DataFrame(X_pca_pbs, columns = ['PC1',' PC2',' PC3'])$

In[30]:

X$_pca_dmem_and_y = X_pca_dmem_df.copy()X_pca_dmem_and_y['Load'] = y_dmem_cleansns.heatmap(X_pca_dmem_and_y.cTrue), cmap = "viridis")$

In[31]:

X$_pca_pbs_and_y = X_pca_pbs_df.copy()X_pca_pbs_and_y['Load'] = y_pbs_cleansns.heatmap(X_pca_pbs_and_y.corr(numeric_oTrue), cmap = "viridis")$

In[32]:

Function to visualise the response variable in relation to 3 principle components def pca$_visual_three_dim(X_pca_df,$

target$_and_PC = X_pca_df.copy()target_and_PC['Load'] = y$

3D scatter plot fig = plt.figure(figsize=(10, 8)) ax = fig.add$_subplot(111, projection =' 3d')$

scatter = ax.scatter(target$_and_PC['PC1'], target_and_PC['PC2'], target_and_PC['PC3'], c = target_and_PC['Load'],coolwarm', edgecolor =' k')$

Add color bar for 'Load' values cbar = plt.colorbar(scatter, ax=ax, pad=0.1) cbar.set$_label('Load')$

Label axes and set title ax.set$_x label('PC1')ax.set_y label('PC2')ax.set_z label('PC3')plt.title("3DScatterPlotof$

plt.show()

In[33]:

pca$_v isual_t hree_d im(X_p ca_d mem_d f, y_d mem_c lean)$

In[34]:

pca$_v isual_t hree_d im(X_p ca_p bs_d f, y_p bs_c lean)$

In[35]:

lr = LinearRegression() lr$_p ca_d mem_r esults = cross_v alidate(lr, X_p ca_d mem_d f, y_d mem_c lean, cv =$
$5, scoring =' r2')print(f'R^2 LinearRegressinowithPCAdmemmatrix : np.mean(lr_p ca_d mem_r esults['test_s core'])'$

lr = LinearRegression() lr$_p ca_p bs_r esults = cross_v alidate(lr, X_p ca_p bs_d f, y_p bs_c lean, cv = 5, scoring ='$
$r2')print(f'R^2 LinearRegressinowithPCApbsmatrix : np.mean(lr_p ca_p bs_r esults['test_s core'])')$

PLS

In[37]:

def fit$_P LS(X, y, max_c omponents) : Specifynumberofcomponentstotestcomponent_r ange = range(1, max_c om$
$1)mean_c v_s cores = []$

for n$_c omponentsincomponent_r ange : pls = PLSRegression(n_c omponents = n_c omponents)cv_s cores =$
$cross_v al_s core(pls, X, y, cv = 5, scoring =' neg_m ean_s quared_e rror')mean_c v_s core = -cv_s cores.mean()mean_c v_s core$
$n_c omponents, MSE : mean_c v_s core")$

optimal$_c omponents = component_r ange[np.argmin(mean_c v_s cores)]print(f"Optimalnumberofcomponents :$
$optimal_c omponents")print(f"Mincross - validatedMSE : min(mean_c v_s cores)")$

optimal$_p ls = PLSRegression(n_c omponents = optimal_c omponents)X_o ptimal = optimal_p ls.fit_t ransform(X,$

return X$_o ptimal, optimal_p ls$

y$_d mem_s td_c lean = scaler.fit_t ransform(y_d mem_c lean.to_f rame())y_p bs_s td_c lean = scaler.fit_t ransform(y_p bs_c lea$

In[38]:

X$_d mem_P LS, dmem_P LS_m odel = fit_P LS(X_d mem_s td_c lean, y_d mem_c lean, 10)X_p bs_P LS, pbs_P LS_m odel =$
$fit_P LS(X_p bs_s td_c lean, y_p bs_c lean, 10)$

Shrinkage Methods

Lasso

In[41]:

Use Lasso instead def lasso(X, y): lasso = Lasso() parameters='alpha': [0.001, 0.01, 0.1, 0.5,
1.0, 100.0, 1000.0] lasso$_r egressor = GridSearchCV(lasso, parameters, scoring =' r2')lasso_m odel =$
$lasso_r egressor.fit(X, y)best_l asso_m odel = lasso_m odel.best_e stimator_p rint(f'OptimalLasso : best_l asso_m odel')non_z$
$np.sum(best_l asso_m odel.coef_! = 0)print(f'Numberofpredictorswithnon-zerocoefficients : non_z ero_p redictors')$
$cross_v alidate(lasso_m odel, X, y, cv = 5, scoring =' neg_m ean_s quared_e rror')print(-cv_r esults['test_s core'])cv_e rr =$
$np.mean(-cv_r esults['test_s core'])print(f'MSE : cv_e rr')$

In[42]:

lasso(X$_d mem_s td_c lean, y_d mem_c lean)$

In[43]:

lasso(X$_p bs_s td_c lean, y_p bs_c lean)$

Move Beyond Linearity

Polynomials

In[46]:

def poly(X,y, degree): Set up a range of polynomial degrees to test degrees = range(1, degree+1)
for degree in degrees: print(degree) Generate polynomial features poly = PolynomialFeatures(degree)
X$_p oly = poly.fit_t ransform(X)$

model = LinearRegression()

MSE scores cv$_s cores = cross_v al_s core(model, X_p oly, y, cv = 5, scoring =' neg_m ean_s quared_e rror')$

print(f'Degree degree - Average MSE score: -np.mean(cv$_s cores) : .4f')print(f'MSEscoresforeachfold :$
$cv_s cores')$

In[47]:

poly(X$_d mem_P LS, y_d mem_c lean, 4)$

In[48]:

poly(X$_p bs_P LS, y_p bs_c lean, 4)$

Neural Network

In[50]:

Must set default paramaters for Grid Search to work def create$_m odel(optimizer =' adam', input_d im =$ $8, activation =' relu', num_h idden_l ayers = 1) : InitialiseNeuralNetorknn = Sequential()tf.keras.Sequential, w$ $...- > output.Inputnn.add(Input(shape = (input_d im,)))Hiddenlayersforiiinrange(num_h idden_l ayers) :$ $nn.add(Dense(64, activation = activation))Outputlayer, loadestimate.nn.add(Dense(1))$

.compile just sets configuration nn.compile(optimizer=optimizer, loss='mean$_s quared_e rror', metrics =$ $[keras.metrics.RootMeanSquaredError()])$

return nn

Function to perform regression given design matrix and target variable def NN$_r egression(X, y) :$ $UseKerasRegressorwrapperinordertousesklearntoolswithKeras.model = KerasRegressor(model =$ $create_m odel, input_d im = X.shape[1], num_h idden_l ayers = 1, epochs = 1, verbose = 0)$

param$_g rid = 'epochs' : [1, 2, 5, 10, 20],' num_h idden_l ayers' : [1, 2]$

grid = GridSearchCV(estimator=model, param$_g rid = param_g rid, scoring =' neg_m ean_s quared_e rror', cv =$ $5)grid_r esult = grid.fit(X, y)$

return grid$_r esult.best_p arams, - grid_r esult.best_s core$

In[51]:

dmem$_n n_p arams_p ls, dmem_n n_M SE_p ls = NN_r egression(X_d mem_P LS, y_d mem_c lean)pbs_n n_p arams_p ls, pbs_n n_M S$ $NN_r egression(X_p bs_P LS, y_p bs_c lean)$

In[52]:

print('PLS') print(f'dmem — optimal parameters: dmem$_n n_p arams_p lsMSE : dmem_n n_M SE_p ls')print(f'pbs|op$ $pbs_n n_p arams_p lsMSE : pbs_n n_M SE_p ls')$

2. Classification

Preprocessing

In[55]:

new target variable target variable y$_c lf_d mem = dmem_v irus_d f['Type']y_c lf_p bs = pbs_v irus_d f['Type']makeybinar$ $(1, 0), handlespacesy_b in_d mem = y_c lf_d mem.str.strip().map('X' : 1,' Y' : 0)y_b in_p bs = y_c lf_p bs.str.strip().map('X' :$

Outlier Removal

In[57]:

def remove$_o utliers_i so_f orest(X, y) : clf = IsolationForest(random_s tate = 42)outlier_p red =$ $clf.fit_p redict(X)RemoveOutliersidentifieX_c lean_s tandardized = X[outlier_p red == 1]y_c lean = y[outlier_p red =$ $1]$

return X$_c lean_s tandardized, y_c lean, outlier_p red$

Already standardized with standard scaler X$_d mem_s td_c lean_c lf, y_d mem_c lean_c lf, outlier_p red_d mem =$ $remove_o utliers_i so_f orest(X_d mem_s td, y_b in_d mem)X_p bs_s td_c lean_c lf, y_p bs_c lean_c lf, outlier_p red_p bs = remove_o utliers_i$

In[58]:

sns.countplot(x=y$_b in_d mem)$

In[59]:

sns.countplot(x=y$_b in_p bs)$

PCA

Scree Plot

In[62]:

pca$_s cree(X_d mem_s td_c lean_c lf, 11)$

In[63]:

pca$_s cree(X_p bs_s td_c lean_c lf, 11)$

Correlation with Target

In[65]:

Set PCA to 3 components pca = PCA(n$_c omponents = 3)X_p ca_d mem_c lf = pca.fit_t ransform(X_d mem_s td_c lean_c$ $pca.fit_t ransform(X_p bs_s td_c lean_c lf)$

In[66]:

Conovert back to df X$_p ca_d mem_c lf_d f = pd.DataFrame(X_p ca_d mem_c lf, columns = ['PC1',' PC2',' PC3'])X_p ca_$ $pd.DataFrame(X_p ca_p bs_c lf, columns = ['PC1',' PC2',' PC3'])$

In[67]:

$X_pca_dmem_and_{yc}lf = X_pca_dmem_clf_df.copy()X_pca_dmem_and_{yc}lf['Load'] = y_dmem_clean_clfsns.heatmap(X_pca_d$
$True), cmap = "viridis")$

In[68]:

$X_pca_pbs_and_{yc}lf = X_pca_pbs_clf_df.copy()X_pca_pbs_and_{yc}lf['Load'] = y_pbs_clean_clfsns.heatmap(X_pca_pbs_and_{yc}lf.co$
$True), cmap = "viridis")$

In[69]:

$\text{pca}_visual_three_dim(X_pca_pbs_clf_df, y_dmem_clean_clf)$

In[70]:

$\text{pca}_visual_three_dim(X_pca_pbs_clf_df, y_pbs_clean_clf)$

PLS

In[72]:

from sklearn.$\text{cross}_decomposition import PLSRegression from sklearn.linear_model import LogisticRegression f$

$\text{def fit}_PLS_for_classification(X, y, max_components) : component_range = range(1, max_components+$
$1)mean cv score relative to number of components.mean_cv_scores = []$

for $\text{n}_components in component_range : FitPLSpls = PLSRegression(n_components = n_components)X_transfo$
$pls.fit_transform(X, y)[0]$

- DA clf = LogisticRegression($\max_iter = 1000)cv_scores = cross_val_score(clf, X_transformed, y, cv =$
$5, scoring =' f1')mean_cv_score = np.mean(cv_scores)mean_cv_scores.append(mean_cv_score)$

optimal number of components based on cross-validated score $optimal_components = component_range[np.argm$
$np.max(mean_cv_scores)transformation using the optimal number of components optimal_pls = PLSRegression(n_c$
$optimal_components)X_optimal = optimal_pls.fit_transform(X, y)[0]$

return $\text{X}_optimal, optimal_pls, optimal_components, best_acc$

In[73]:

$X_pbs_PLS_clf, pbs_PLS_model_clf, num_pbs, score_pbs = fit_PLS_for_classification(X_pbs_std_clean_clf, y_pbs_clean_clf, 20)$
$num_pbs, accuracy : score_pbs')$

$X_dmem_PLS_clf, dmem_PLS_model_clf, num_dmem, score_dmem = fit_PLS_for_classification(X_dmem_std_clean_clf, y$
$num_dmem, accuracy : score_dmem')$

Normality of Features

In[75]:

$\text{def graph}_data_hist(data) : Plothistograms for numerical features plt.rc('font', size = 14)plt.rc('axes', labelsiz$
$14, titlesize = 14)plt.rc('legend', fontsize = 14)plt.rc('xtick', labelsize = 10)plt.rc('ytick', labelsize =$
$10)$

data.hist(bins=50, figsize=(12, 8)) plt.show()

In[76]:

$\text{graph}_data_hist(pd.DataFrame(X_pca_dmem))$

In[77]:

$\text{graph}_data_hist(pd.DataFrame(X_pca_pbs))$

Ratio of Features to Observations

In[79]:

Function for visualising confusion matrix for evaluating classification models. $\text{def CM}_visualise(X, y, model) :$
$Confusion Matrix Generate cross-validated predictions y_pred_log = cross_val_predict(model, X, y, cv =$
$10)Create and plot the confusion matrix conf_matrix = confusion_matrix(y, y_pred_log)disp = ConfusionMatrixDis$
$conf_matrix)disp.plot(cmap =' Blues')plt.title(f"Confusion Matrix for model")plt.show()$

Naive Bayes

In[81]:

gnb = GaussianNB()

In[82]:

$\text{cv} = \text{np.mean}(\text{cross}_val_score(gnb, X_pca_pbs, y_pbs_clean_clf, cv = 5, scoring =' accuracy'))print(cv)$

In[83]:

$\text{CM}_visualise(X_pca_pbs, y_pbs_clean_clf, gnb)$

In[84]:

$\text{cv} = \text{cross}_val_score(gnb, X_pca_dmem, y_dmem_clean_clf, cv = 5, scoring =' accuracy')print(np.mean(cv))$

In[85]:
$\mathrm{CM}_v isualise(X_p ca_d mem, y_d mem_c lean_c lf, gnb)$
Logistic Regression
In[87]:
$\log_m odel = LogisticRegression(max_i ter = 1000)$
In[88]:
$\mathrm{cv} = \mathrm{cross}_v al_s core(log_m odel, X_p ca_p bs, y_p bs_c lean_c lf, cv = 5, scoring =' accuracy')print(np.mean(cv))$
In[89]:
$\mathrm{CM}_v isualise(X_p ca_p bs, y_p bs_c lean_c lf, log_m odel)$
In[90]:
$\mathrm{cv} = \mathrm{cross}_v al_s core(log_m odel, X_p ca_d mem, y_d mem_c lean_c lf, cv = 5, scoring =' accuracy')print(np.mean(cv))$
In[91]:
$\mathrm{CM}_v isualise(X_p ca_d mem, y_d mem_c lean_c lf, log_m odel)$
KNN Classifier
In[93]:
$\mathrm{knn}_c lf = KNeighborsClassifier(n_n eighbors = 5)$
In[94]:
$\mathrm{cv} = \mathrm{cross}_v al_s core(knn_c lf, X_p ca_p bs, y_p bs_c lean_c lf, cv = 5, scoring =' accuracy')print(np.mean(cv))$
In[95]:
$\mathrm{CM}_v isualise(X_p ca_p bs, y_p bs_c lean_c lf, knn_c lf)$
In[96]:
$\mathrm{cv} = \mathrm{cross}_v al_s core(knn_c lf, X_p ca_d mem, y_d mem_c lean_c lf, cv = 5, scoring =' accuracy')print(np.mean(cv))$
In[97]:
$\mathrm{CM}_v isualise(X_p ca_d mem, y_d mem_c lean_c lf, knn_c lf)$
LDA
In[99]:
lda = LDA()
In[100]:
$\mathrm{cv} = \mathrm{cross}_v al_s core(lda, X_p ca_p bs, y_p bs_c lean_c lf, cv = 5, scoring =' accuracy')print(np.mean(cv))$
In[101]:
$\mathrm{CM}_v isualise(X_p ca_p bs, y_p bs_c lean_c lf, lda)$
In[102]:
$\mathrm{cv} = \mathrm{cross}_v al_s core(lda, X_p ca_d mem, y_d mem_c lean_c lf, cv = 5, scoring =' accuracy')print(np.mean(cv))$
In[103]:
$\mathrm{CM}_v isualise(X_p ca_d mem, y_d mem_c lean_c lf, lda)$
QDA
In[105]:
qda = QDA()
In[106]:
$\mathrm{cv} = \mathrm{cross}_v al_s core(qda, X_p ca_p bs, y_p bs_c lean_c lf, cv = 5, scoring =' accuracy')print(np.mean(cv))$
In[107]:
$\mathrm{CM}_v isualise(X_p ca_p bs, y_p bs_c lean_c lf, qda)$
In[108]:
$\mathrm{cv} = \mathrm{cross}_v al_s core(qda, X_p ca_d mem, y_d mem_c lean_c lf, cv = 5, scoring =' accuracy')print(np.mean(cv))$
In[109]:
$\mathrm{CM}_v isualise(X_p ca_d mem, y_d mem_c lean_c lf, qda)$
3. Clustering
Preprocessing
In[112]:
Use full data set $\mathrm{cluster}_d ata = virus_d f$
Split X and y
In[114]:

Sprectrum data as design matrix $X_cluster = cluster_data[cluster_data.columns[4:517]] Forevaluatingclusterin$
$cluster_data['Load']$

Standardization

In[116]:

Standardize, so as to compare performance with and without. scaler = StandardScaler() $X_cluster_std =$
$scaler.fit_transform(X_cluster)$

Outlier Removal (Unsupervised)

In[118]:

Unsupervised outlier removal. def $outlier_removal_lof(X, y) : LOF = LocalOutlierFactor(n_neighbors =$
$2) outlier_pred = LOF.fit_predict(X)$

$X_clean = X[outlier_pred == 1] y_clean = y[outlier_pred == 1]$

return X_clean, y_clean

In[119]:

$X_cluster_std_clean, y_cluster_std_clean = outlier_removal_lof(X_cluster_std, y_cluster_load) X_cluster_clean, y_cluster_clea$
$outlier_removal_lof(X_cluster, y_cluster_load)$

PCA

In[121]:

Use PCA as it is an unsupervised dimension reduction (colinearity removal) method. $pca_scree(X_cluster_std_clean$

In[122]:

pca = PCA($n_components = 3) X_cluster_pca = pca.fit_transform(X_cluster_std_clean)$

Kmeans

In[124]:

Define range for k values (number of centroids) def $plot_kmeans(X, y, max_k = 10) : K = range(2, max_k)$

fits = [] inertia = [] $silhouette_scores = [] accuracies = [] Iterateoverdifferentkvaluesforkin K :$
$kmeans = KMeans(n_clusters = k, init =' k-means++', random_state = 42) kmeans.fit(X) fits.append(kmean$

inertia (wcss), distance of $x_itocentroid => howtightlypacked(defined) clustersare.inertia.append(kmeans.in$
$ispointintherightcluster-1- > 1score = silhouette_score(X, kmeans.labels_) silhouette_scores.append(score)$

labels = kmeans.$labels_labelmapping = np.zeros_like(labels)$

for i in range(k): mask = (labels == i) use mode() to find most frequent label. $label_mapping[mask] =$
$mode(y[mask])[0]$

accuracy = $accuracy_score(y, label_mapping) accuracies.append(accuracy)$

if X.shape[1] ¡ 4: Scatter plot for the clusters based on PC1 and PC2 plt.figure(figsize=(6, 6))
plt.scatter(X[:, 0], X[:, 1], c=kmeans.$labels_) plt.xlabel('PC1') plt.ylabel('PC2') plt.title(f"K-MeansClusteringRe$
$k") plt.show()$

return inertia, $silhouette_scores, accuracies$

In[125]:

$iniertia_load, silhouette_load, accuracies_load = plot_kmeans(X_cluster_pca, y_cluster_std_clean, 10)$

In[126]:

Plot Elbow Method graph plt.figure(figsize=(8, 5)) plt.plot(range(2, len($iniertia_load$)+2), $iniertia_load) plt.title($

In[127]:

Plot Silhouette Analysis graph plt.figure(figsize=(8, 5)) plt.plot(range(2, len($iniertia_load$)+2), $silhouette_load) pl$

In[128]:

Accuracy by K graph plt.figure(figsize=(8, 5)) plt.plot(range(2, len($accuracies_load$)+2), $accuracies_load) plt.title$

In[129]:

$iniertia_load_unstd, silhouette_load_unstd, accuracies_load_unstd = plot_kmeans(X_cluster_clean, y_cluster_clean, 10)$

In[130]:

Plot Elbow Method graph plt.figure(figsize=(8, 5)) plt.plot(range(2, len($inertia_load_unstd$)+2), $inertia_load_unst$

In[131]:

Plot Silhouette Analysis graph plt.figure(figsize=(8, 5)) plt.plot(range(2, len($inertia_load_unstd$) +
$2), silhouette_load_unstd) plt.title('SilhouetteAnalysis') plt.xlabel('Numberofclusters') plt.ylabel('SilhouetteScore$

In[132]:

Accuracy by K graph plt.figure(figsize=(8, 5)) plt.plot(range(2, len($accuracies_load_unstd$)+2), $accuracies_load_un$

Hierarchichal

In[134]:

Complete Linkage hc$_c omplete = HClust(distance_t hreshold = 0, n_c lusters = None, linkage =' complete')AverageLinkagehc_a ve = HClust(distance_t hreshold = 0, n_c lusters = None, linkage =' average')Singlelinkagehc_s ing = HClust(distance_t hreshold = 0, n_c lusters = None, linkage =' single')$

Centroid linkage hc$_w ard = HClust(distance_t hreshold = 0, n_c lusters = None, linkage =' ward')$

PCA

In[136]:

No preconceived chopping cargs= 'color$_t hreshold' : -np.inf,' above_t hreshold_c olor' :' black'$

In[137]:

def plot$_b y_l inkage(X) : BylinkageTypecomplete, average, single, centroidfig, ax = plt.subplots(2, 2, figsize = (16, 8))$

complete hc$_c omplete.fit(X)linkage_c omplete = compute_l inkage(hc_c omplete)dendrogram(linkage_c omplete, ax ax[0, 0], **cargs)ax[0, 0].set_t itle("CompleteLinkage")ax[0, 0].set_x ticks([])averagehc_a ve.fit(X)linkage_a ve = compute_l inkage(hc_a ve)dendrogram(linkage_a ve, ax = ax[0, 1], **cargs)ax[0, 1].set_t itle("AverageLinkage")ax[0,$

single hc$_s ing.fit(X)linkage_s ing = compute_l inkage(hc_s ing)dendrogram(linkage_s ing, ax = ax[1, 0], ** cargs)ax[1, 0].set_t itle('SingleLinkage')ax[1, 0].set_x ticks([])$

centroid hc$_w ard.fit(X)linkage_w ard = compute_l inkage(hc_w ard)dendrogram(linkage_w ard, ax = ax[1, 1], * * cargs)ax[1, 1].set_t itle('WardLinkage')ax[1, 1].set_x ticks([])$

plt.show()

In[138]:

plot$_b y_l inkage(X_c luster_p ca)$

In[139]:

plot$_b y_l inkage(X_c luster_c lean)$

DBSCAN

In[141]:

dbscan = DBSCAN(eps=3, min$_s amples = 2).fit(X_c luster_p ca)n_c lusters = len(set(dbscan.labels_)) - (1if - 1indbscan.labels_e lse0)n_n oise = list(dbscan.labels_).count(-1)$

print('pca') print(f"Number of clusters: n$_c lusters")print(f"Numberofnoisepoints : n_n oise")$

In[247]:

dbscan = DBSCAN(eps=3, min$_s amples = 2).fit(X_c luster_c lean)n_c lusters = len(set(dbscan.labels_)) - (1if - 1indbscan.labels_e lse0)n_n oise = list(dbscan.labels_).count(-1)$

print('unstd') print(f"Number of clusters: n$_c lusters")print(f"Numberofnoisepoints : n_n oise")$