Analysis of Spectral Data using Machine Learning Techniques

Michael Simpson

11th November, 2024

1 Introduction

Over the past few decades, the use of Machine Learning in different disciplines has increased significantly, especially due to the vast quantities of data that experts have to deal with. Machine Learning is very useful in the face of this data as it allows for accurate prediction of the target variable of interest when given new data (supervised learning), and also can reveal patterns in data that one might not have noticed without it (unsupervised learning). These are the two pillars of Machine Learning that have been applied to a dataset provided in the area of Biochemistry in the following project. The dataset contains parameters and measurements collected during an experiment to detect a particular type of virus using Spectroscopy. A more in depth description of the dataset is given in the next section.

There were three main techniques of Machine Learning used to analyse the dataset: Regression, Classification and Clustering. The first two fall under the category of supervised learning, as they include models that are trained on sections of the dataset along with a target variable, to then are tested on a seperate section with the goal of predicting values belonging to the target column. Regression was used to predict viral load as it contains several values that follow a numerical order, whereas Classification was used to predict the virus Type, designated either X or Y. Clustering is then a form of unsupervised learning and was used to assign observations to a particular cluster, thus revealing an underlying pattern in the dataset.

2 Data Description and Preparation

The data shown in Table 1 was taken from an experiment in which a Type of virus (X or Y) was held in a particular medium ("dmem" or "pbs"), of which was diluted a certain amount, given by the Load value. Light of various wavelengths (I001 to I512) were then fired at the virus, with the resulting light intensity recorded. This was repeated over 4 sessions (S1 to S4). The *.describe*

method was used on the database to obtain values such as the *mean* and *standard deviation* for the numerical columns. It was found that the wavelengths around I320 had much lower light intensities than wavelengths at the lower and higher end. This suggested some high correlation within the data. Using the *isna().any()* methods, it was also found that there were no missing values in the dataset.

Attribute	**Description**	**Data Type**	**Data**
SID	Session ID	Object	S1 - S4
Type	Type of Virus	Object	X or Y
Matrix	Medium	Object	dmem or pbs
Load	Serial Dilution Level	Numeric	1 - 8
I001 - I512	Light Intensity at various wavelengths	Numeric	Float value

Table 1: Overview of database columns

The medium the virus was held in was not to be used as either a target variable or a predictor variable for each of the experiments and thus provided a means to compare different models. Therefore, the dataset was split by Medium i.e. into "dmem" and "pbs" datasets. The distribution of values for wavelength I001 is shown in Figure 1. A rough Gaussian distribution can be observed with the exception of some dips around 93 for "dmem" and 98 for "pbs", as well as the outliers, which are dealt with in future sections.

A correlation heatmap was also created for each dataset as shown in Figure 2. The graph shows that there is no correlation between approximately the first half of wavelength columsn and the rest of the wavelengths. There is slight correlation at the borders (given by the red colour) but this seems very minimal. This suggested the possibility of reducing the 512 wavelength columns into 2 columns via Principal Component Analysis. This is shown in the Clustering section.

The wavelength data was then scaled using *StandardScaler* as this is required by many Machine Learning models. The outliers were then removed using *IsolationForest*. This class assigns a value of -1 to an observation if it is deemed an outlier, and a value of 1 if not. It was this list that was used to select only values that were not outliers from the datasets. To illustrate, Figure 3 shows the distribution of I001 before and after outlier removal, proving it's success.

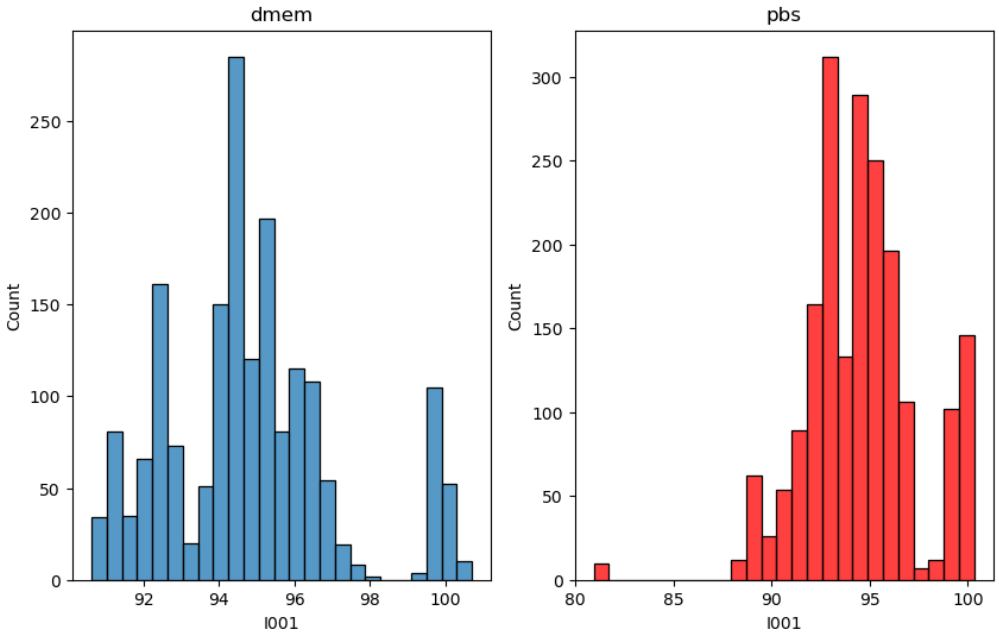


Figure 1: Distribution of light intensities for wavelength I001, split by medium. Both show rough Gaussian shapes

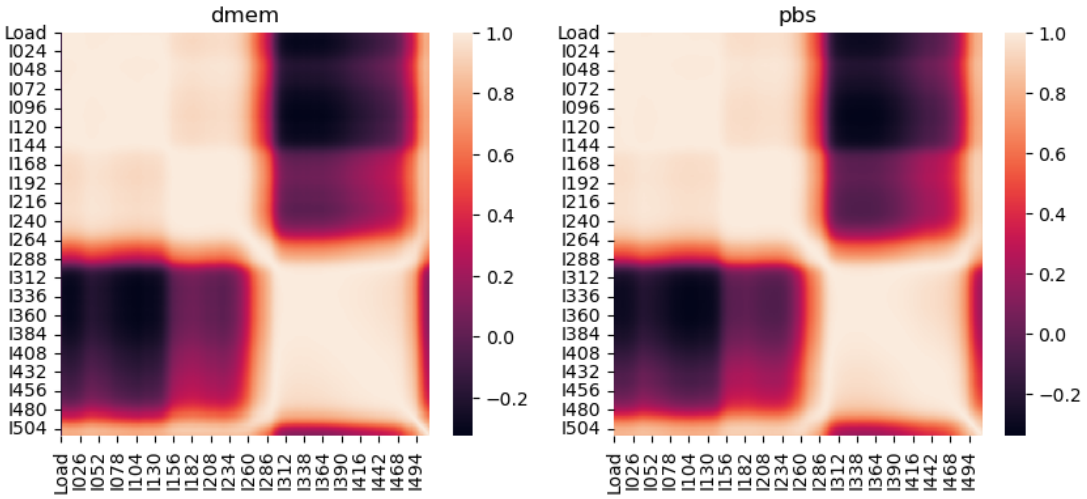
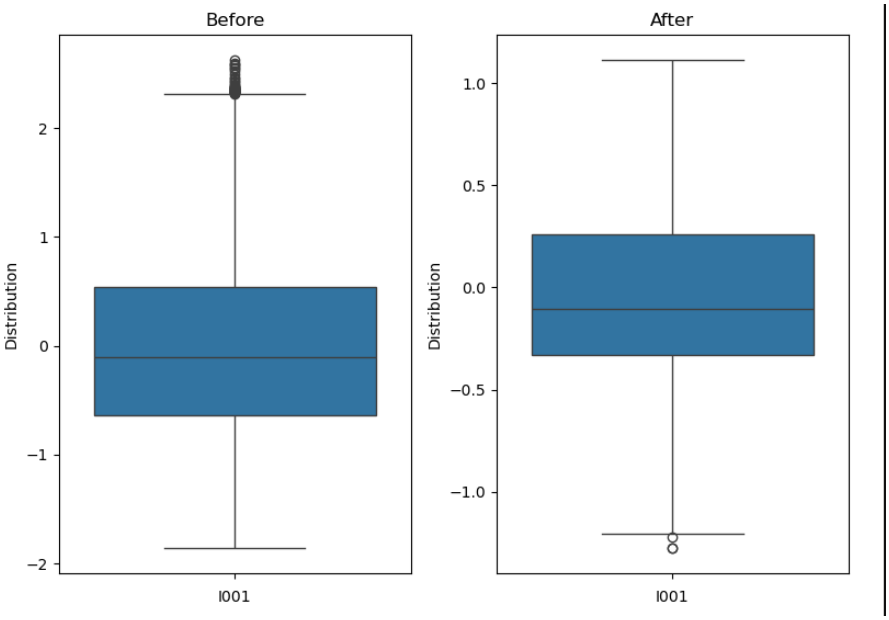


Figure 2: Correlation heatmaps for all wavelength columns between datasets. No correlation found between both halfs of the datasets.



t

Figure 3: Distribution of I001 for dmem dataset before and after outlier removal

3 Regression

For Regression the Load was chosen as the target variable as there are 8 unique numerical values and so there's a possibility of correlation with light intensity. To begin, each dataset representing the mediums were split into the target and predictor variables. The predictor variable (X) included I001 to I512 while the target variable was the Load. These were then split into training and test sets using Sklearn's *train-test-split*. They were also stratified relative to the Load column as to include an equal proportion of each Load value across training and test sets.

The models then chosen were Linear Regression, due to the Gaussian and highly correlated nature of the data, and a Random Forest Regressor. The Linear Regression model was trained on the "dmem" set, white the Random Forest Regressor was trained on the "pbs" set, then the predicted values for Load were found. These were then plotted against the true values for the Load, as shown in Figure 4. Both models were successful in predicting values close to the actual Load value, thus producing a nice correlation, especially the Random Forest Regressor.

Both of the models were then evaluated using K-fold Cross-Validation with a K value of 5. This is better than simply calculating the metrics directly from

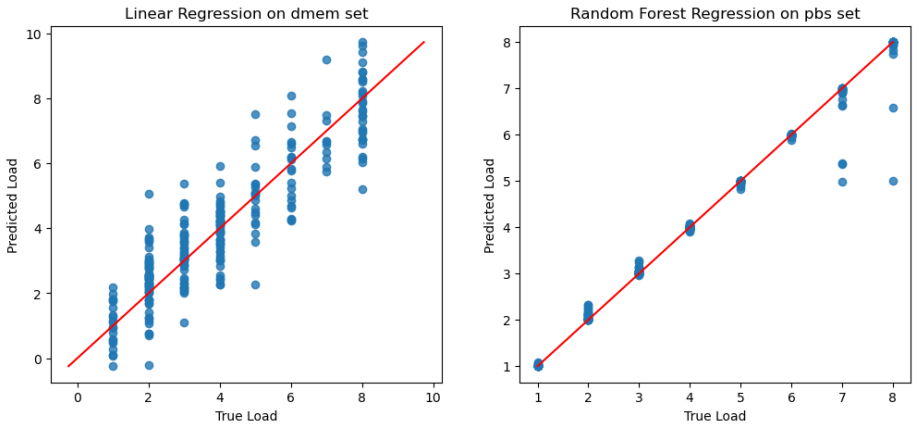


Figure 4: Values for Load as predicted by LR and RFR. True values lie on the red line

the predicted and test values as it calculates K number of the chosen metric from different parts of the dataset. The mean of the K values (in our case, 5 values) was then found to get the final value. In this case, the Mean Squared Error and R^2 score were calculated for each model, as shown in the table below. The RFR clearly predicted values much closer to the actual values than the LR model, due to the Mean Squared Error being much smaller and the R^2 score being closer to 1.

	MSE	R^2
LR (dmem)	1.22	0.721
RFR (pbs)	0.0396	0.9898

Table 2: Mean Squared Error and R^2 score for LR and RFR models

Due to RFR having the higher R^2 score it was chosen for model optimisation. A range of parameters belonging to the RFR model were chosen and Grid Search Cross Validation was then used to perform an exhaustive search through these parameters to find the optimal values. Doing this resulted in an R^2 score of 0.9977 and MSE of 0.00927. A graph of the predicted values is shown in the figure below, where the remaining predicted values far from the actual values in the previous graph are now much closer to the line.

From the results obtained, it is clear that RFR performed much better than LR. This may be due to the fact that the dataset is very large, an aspect that RFR is able to handle very well, unlike LR which is suitable for smaller datasets containing less dimensions. RFR is also an ensemble method, combining methods such as Bagging and Boosting thus greatly increasing it's predictive power.

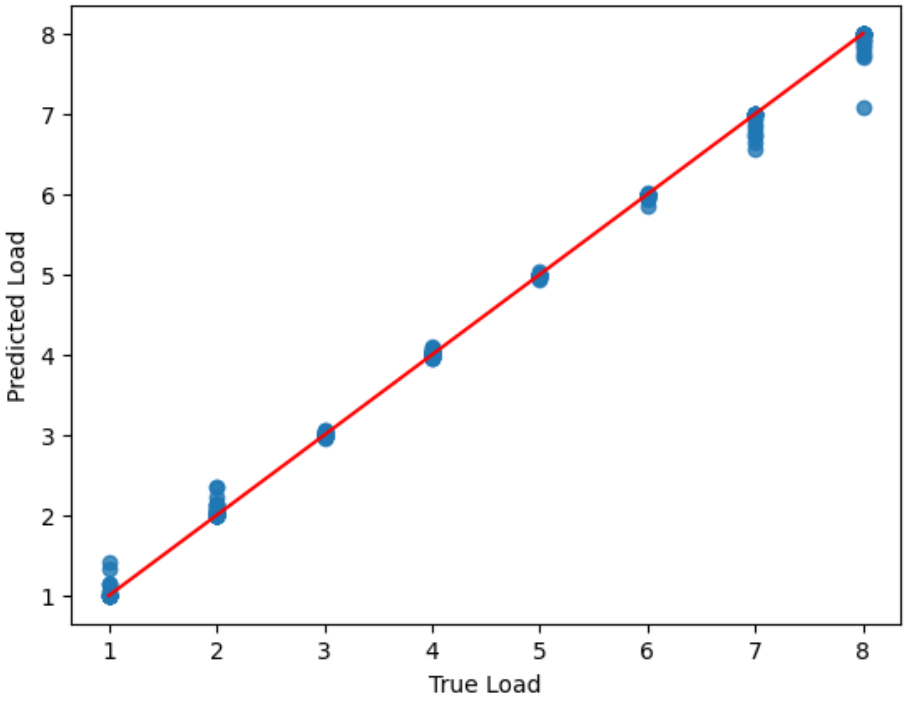


Figure 5: Values for Load as predicted by RFR after optimisation of the model

4 Classification

For Classification, the predictor variables were kept the same as regression (being the wavelengths of light), but the target variable was changed to the Virus Type, turning this into a Binary Classification problem. Firstly, the Type column was converted into Binary code using *Label Binarizer*, which simply transformed each X and Y value into a 0 and 1, respectively. This is required by the Machine Learning models as they can only work when all data is in numercal form.

The number of each type of virus was counted for both target columns and it was found that there was nearly twice the amount of Type Y in each, thus indicating that the datasets were imbalanced. The outcome of the models could therefore be biased towards one Type of virus, which is not ideal. To balance the datasets, *SMOTE* was used which synthesises new observations belonging to the minority class and adds them to both the predictor and target sets. SMOTE was also combined with the under-sampling technique *Random Under Sampler* as models perform better when both methods are used.[1] Tables 3 and 4 show the result of the over and under sampling. The predictor data was then split into training and test sets and stratified relative to virus Type.

Logistic Regression and Linear Support Vector Classifier (SVC) were chosen

	dmem	pbs
0	472	505
1	804	901

Table 3: Number of 0s and 1s in each set before applying SMOTE and Random Under Sampler

	dmem	pbs
0	638	703
1	638	703

Table 4: Number of 0s and 1s in each set after applying SMOTE and Random Under Sampler

as the models for the same reason as Regression i.e. Gaussian dataset with high correlation. Logistic Regression was then trained on the "dmem" dataset whereas Linear SVC was trained on the "pbs" dataset, and the predicted virus Types for each observation was found. Cross-validation was used first to evaluate the models, producing R^2 scores of 0.953 and 0.989 for Logistic Regression and Linear SVC, respectively. Clearly, the models were predicting the correct virus Type with a high degree of accuracy, especially Linear SVC. A classification report was also created for each model, which includes important metrics such as *precision*,*recall* and *f1-score*.

Logistic Regression (dmem)			Linear SVC(pbs)		
	0	1	0	1	
precision	1	0.993	1	0.993	
recall	0.992	1	0.992	1	
f1-score	0.9959	0.9963	0.9964	0.9965	

Table 5: Classification report for each model

Based on the report, the models are clearly performing very well, especially considering the precision score of 1 for both models, meaning all classified values were true. The f1-score, which combines both precision and recall, is very high as well, although slightly higher for Linear SVC. To illustrate this further, Figure 6 shows a confusion matrix for both models. For both, there is only one False Positive, so the reason Linear SVC has scored more highly is simply due to it having more observations to classify i.e. "pbs" contains more samples than "dmem". Finally, Figure 7 shows a ROC Curve for the Logistic Regression model, indicating that it is a near perfect model with the slight dip in the top left corner being due to the 1 False Positive prediction.

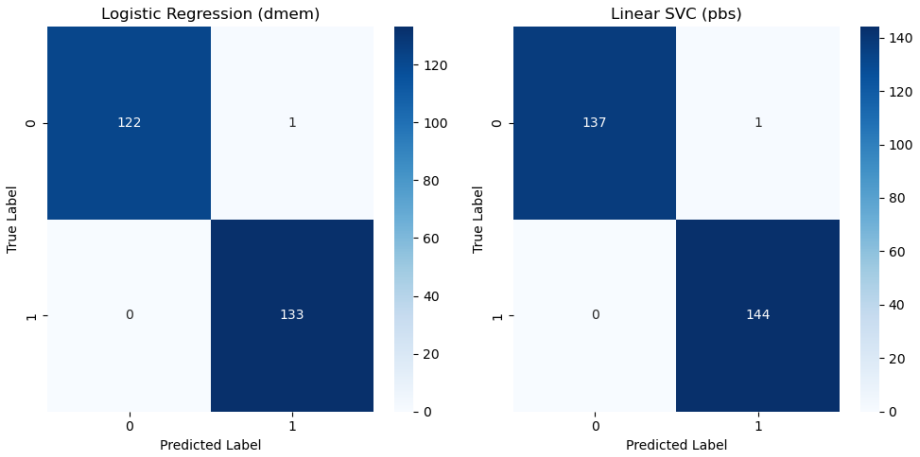


Figure 6: Confusion Matrix for both models. Note that 0 and 1 correspond to Virus X and Y, respectively

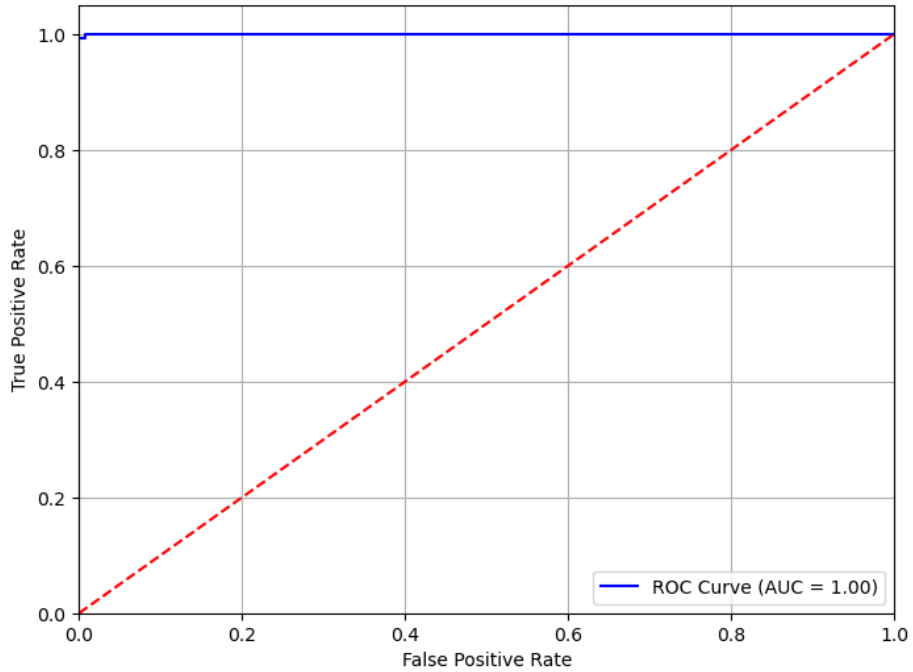


Figure 7: ROC Curve for Logistic Regression. Dashed red line represents a completely random whereas blue line represents the Logistic Regression model.

There are other models that could be considered such as Decision Trees

and Non-Linear Support Vector Classifiers, but due to the dataset being highly correlated, the linear models suffice. Other models may eliminate the 1 False Positive produced by both Linear models, but given they are more computationally expensive it does not seem worth it.

5 Clustering

Being a form of unsupervised learning, Clustering doesn't require a target variable to be specified for prediction, and instead only some input data that it will find patterns in. In this case, the input data is the wavelengths of light (I001 to I512). Clearly though, this data has too many dimensions and so can't be easily visualised graphically. Therefore, Principal Component Analysis was applied to this data due to the fact that two sections of the wavelength data are very highly correlated, and thus could possibly be represented as two columns, reducing the dataset from 512 to 2 dimensions. This was done through the use of *PCA* class from *sklearn.decomposition* on both datasets ("dmem" and "pbs") with the outliers included, as they formed clusters of their own. The Explained Variance Ratio for both Principal Components are shown in the table below for each dataset. More than 90 percent of the variance was explained through these components, therefore is wasn't necessary to include any more.

	dmem	pbs
PC1	0.567	0.577
PC2	0.423	0.416
Total	0.990	0.930

Table 6: The Explained Variance Ratio for the first two Principal Components of each dataset. Most of the variance is explained by these components

K-means clustering was then applied to the "dmem" dataset which a range of K values from 4 to 10 being tested. This range was chosen because there are 8 unique Load values, thus it was theorised that the ideal number of clusters would be approxiamately 8. After applying K-means clustering, the values transformed by the Principal Components were plotted for each value of K as shown in Figure 8. The corresponding Silhouette Score is also displayed for each K-value and peaks after K = 8. It is unclear which K value is optimal in this case, so the Elbow Method was also employed, shown in Figure 9. This plot was obtained by finding the *Within-Cluster Sum of Squares* using *.inertia* on the models. The "Elbow" of the plot corresponds to the optimal K-value and is around 8, which matches the number of unique Load values. Therefore, each cluster corresponds approximately to a particular Load value.

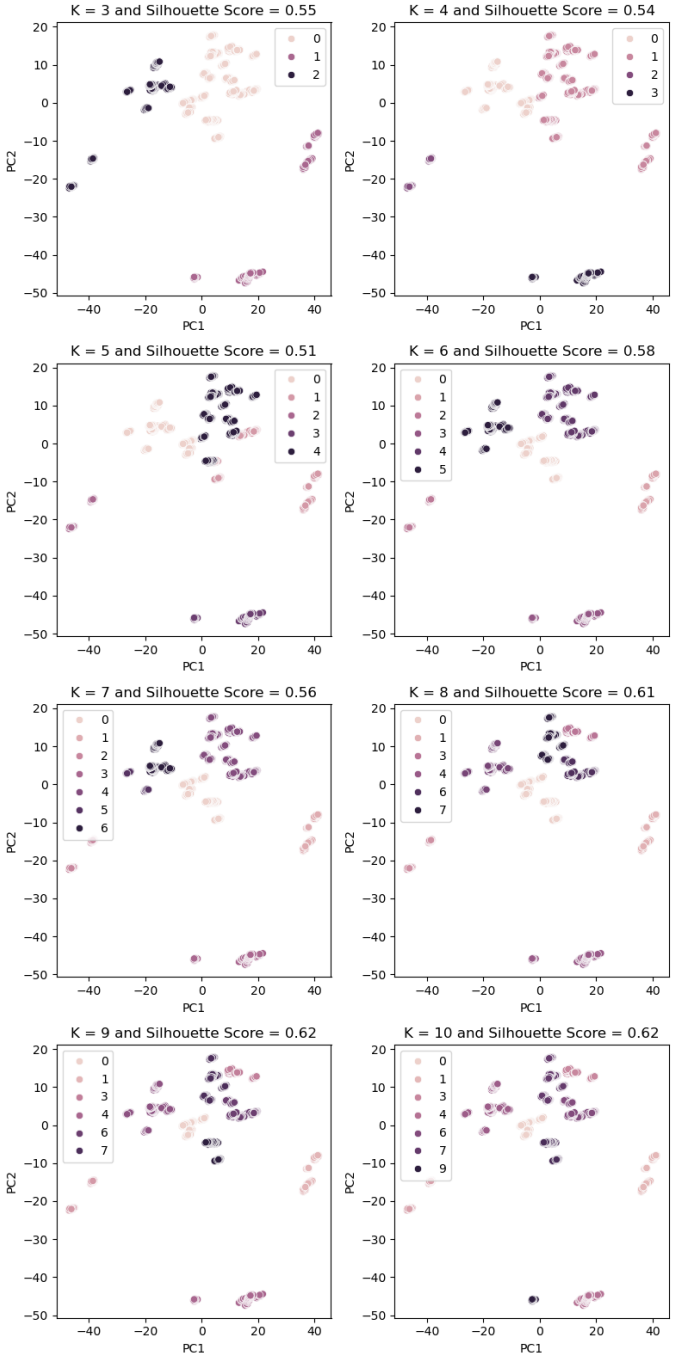


Figure 8: K-means clustering applied to "dmems" dataset in Principal Component format. K values range from 3 to 10

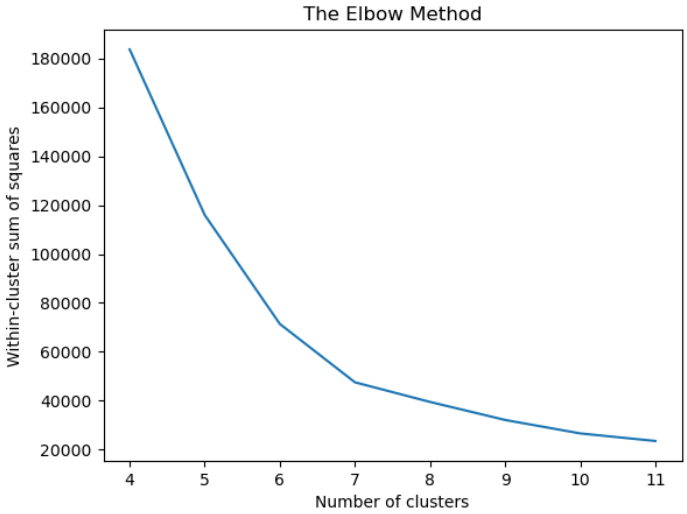


Figure 9: The Within-Cluster Sum of Squares for each K cluster. The "Elbow corresponds to the optimal K"

Hierarchical Clustering was then applied to the "pbs" dataset. This type of clustering is sensitive to outliers so they had to be removed using Isolation Forest, in the same way as for Regression and Classification. Complete linkage was then applied to the dataset and a Dendrogram was formed as shown in Figure 10.

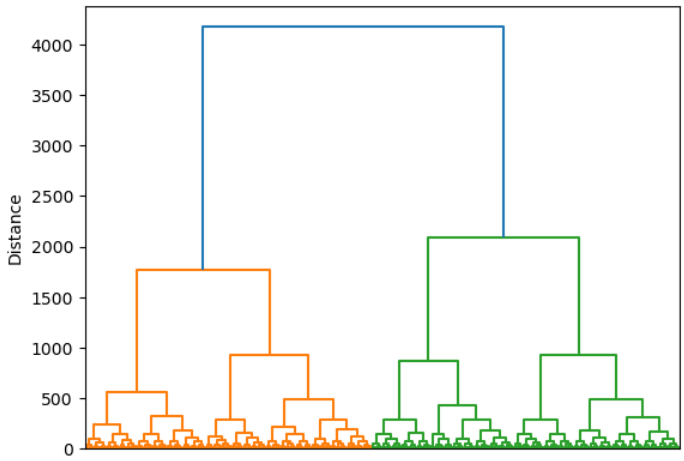


Figure 10: Dendrogram formed from Hierarchical Clustering of "pbs" dataset

6 Conclusion

This project has shown the application of 3 different Machine Learning techniques: Regression, Classification and Clustering to a dataset involving virus detection using Spectroscopy. For Regression it was found that the Random Forest Regressor performed much better than the Linear regression model, due to its capability in dealing with large amounts of data. The Linear Regression model could still be used though when saving on computational power, as the R^2 score was not low. In Classification, both the Logistic Regression and Linear SVC were found to be very effective in classifying the type of virus for each observation. This was likely due to the data being highly correlated with virus Type. Therefore, when this is not the case, other non-linear models would have to be considered. Finally, for Clustering the K-means model was successful in identifying an optimal number of approximately 8 clusters, in line with the unique numbers of Load values.

Overall, each technique has shown how effective they can be at predicting chosen variables and identifying patterns in large quantities of data. Although, it is clear that alot of experimentation has to be done in order to obtain the optimal parameters and hyperparameters, and thus the best model for the chosen dataset. With this method, there is much more to discover about the capabilities of these Machine Learning techniques.

7 References

[1] Chawla, N.V., Bowyer, K.W., Hall, L.O. and Kegelmeyer, W.P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16(16), pg 352. doi:https://doi.org/10.1613/jair.953.

Word Count: Approx. 2200

8 Appendix

```
#Import some neccessary packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#Load in the dataset and view
virus_df = pd.read_csv('2022QUB.csv')
virus_df.head()
#Summary values for each column
virus_df.describe()
```

```
virus_df.info()
#Check for missing values
#pd.set_option('display.max_rows',None)
virus_df.isna().any()
#Split the dataset by Matrix type (dmem and pbs)
is_dmem = virus_df['Matrix'] == ' dmem'
virus_df_dmem = virus_df[is_dmem]

is_pbs = virus_df['Matrix'] == ' pbs'
virus_df_pbs = virus_df[is_pbs]
#Drop Matrix column is each split dataset as it's not
    needed now
virus_df_dmem = virus_df_dmem.drop(columns=['Matrix'],
    axis = 1)

virus_df_pbs = virus_df_pbs.drop(columns=['Matrix'], axis
    = 1)
virus_df_pbs
#Display distribution of light intensity values for each
    load
fig, ax = plt.subplots(1,2, figsize = (10,6))

sns.scatterplot(data = virus_df_dmem, x = 'I001', y = '
    Load', ax = ax[0])
ax[0].set_title("dmem")

sns.scatterplot(data = virus_df_pbs, x = 'I001', y = '
    Load', ax = ax[1])
ax[1].set_title("pbs")
fig, ax = plt.subplots(1,2, figsize = (10,6))

sns.histplot(virus_df_dmem['I001'], bins = 25, ax = ax
    [0])
ax[0].set_title("dmem")

sns.histplot(virus_df_pbs['I001'], bins = 25, ax = ax[1],
    color = 'r')
ax[1].set_title("pbs")
fig, ax = plt.subplots(1,2, figsize = (10,4))

#Correlation between all light wavelengths
corr_mat_dmem = virus_df_dmem.corr(numeric_only=True)
corr_mat_pbs = virus_df_pbs.corr(numeric_only=True)

#Display heatmap for each correlation matrix
sns.heatmap(corr_mat_dmem, ax = ax[0])
```

```
ax[0].set_title("dmem")

sns.heatmap(corr_mat_pbs, ax = ax[1])
ax[1].set_title("pbs")

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

#Scale the data
light_df_dmem = virus_df_dmem.drop(columns = ['SID','Type
    ','Load'], axis = 1)
light_df_scaled_dmem = pd.DataFrame(scaler.fit_transform(
    light_df_dmem), index = light_df_dmem.index, columns =
     light_df_dmem.columns)

light_df_pbs = virus_df_pbs.drop(columns = ['SID','Type
    ','Load'], axis = 1)
light_df_scaled_pbs = pd.DataFrame(scaler.fit_transform(
    light_df_pbs), index = light_df_pbs.index, columns =
    light_df_pbs.columns)

light_cols = virus_df_dmem.columns[3:516]

#Creation of scaled original dataframe

virus_df_scaled_dmem = pd.concat([(virus_df_dmem.drop(
    columns=light_cols, axis=1)), light_df_scaled_dmem],
    axis = 1)
virus_df_scaled_pbs = pd.concat([(virus_df_pbs.drop(
    columns=light_cols, axis=1)), light_df_scaled_pbs],
    axis = 1)

from sklearn.ensemble import IsolationForest

#Detect outliers using Isolation Forest
IForest = IsolationForest(random_state=42)

outlier_pred_dmem = IForest.fit_predict(
    light_df_scaled_dmem)
outlier_pred_pbs = IForest.fit_predict(
    light_df_scaled_pbs)

#Outliers when -1 so single out rows with 1
virus_df_scaled_no_out_dmem = virus_df_scaled_dmem.iloc[
    outlier_pred_dmem == 1]
```

```
virus_df_scaled_no_out_pbs = virus_df_scaled_pbs.iloc[
    outlier_pred_pbs == 1]

print("Number of rows in dmem dataset after outlier
    removal = {}".format(virus_df_scaled_no_out_dmem.shape
    [0]))
print("Number of rows in pbs dataset after outlier
    removal = {}".format(virus_df_scaled_no_out_pbs.shape
    [0]))

#Display the distribution of light intensity values
    before and after outlier removal for dmem dataset
fig, ax = plt.subplots(1,2, figsize = (8,6))
fig.suptitle("Disbrution of Light Intensity values for
    I001 wavelength before and after Outlier Removal (dmem
    )")

sns.boxplot(virus_df_scaled_dmem['I001'], ax = ax[0])
ax[0].set_xlabel("I001")
ax[0].set_ylabel("Distribution")
ax[0].set_title("Before")

sns.boxplot(virus_df_scaled_no_out_dmem['I001'], ax = ax
    [1])
ax[1].set_xlabel("I001")
ax[1].set_ylabel("Distribution")
ax[1].set_title("After")

plt.tight_layout()

#Display the distribution of light intensity values
    before and after outlier removal for pbs dataset
fig, ax = plt.subplots(1,2, figsize = (8,6))
fig.suptitle("Disbrution of Light Intensity values for
    I001 wavelength before and after Outlier Removal (pbs)
    ")

sns.boxplot(virus_df_scaled_pbs['I001'], ax = ax[0])
ax[0].set_xlabel("I001")
ax[0].set_ylabel("Distribution")
ax[0].set_title("Before")

sns.boxplot(virus_df_scaled_no_out_pbs['I001'], ax = ax
    [1])
ax[1].set_xlabel("I001")
ax[1].set_ylabel("Distribution")
```

```
ax[1].set_title("After")

plt.tight_layout()

from sklearn.model_selection import train_test_split

#Spliting datasets into independent and dependent
    variables
X_dmem = virus_df_scaled_no_out_dmem.drop(columns=['SID
    ','Type','Load'])
X_dmem = np.asarray(X_dmem)
y_dmem = virus_df_scaled_no_out_dmem['Load']
y_dmem = np.asarray(y_dmem)

X_pbs = virus_df_scaled_no_out_pbs.drop(columns=['SID','
    Type','Load'])
X_pbs = np.asarray(X_pbs)
y_pbs = virus_df_scaled_no_out_pbs['Load']
y_pbs = np.asarray(y_pbs)

#Split the datasets into training and test data with
    stratification relative to Load value
X_train_dmem, X_test_dmem, y_train_dmem, y_test_dmem =
    train_test_split(X_dmem, y_dmem, test_size = 0.2,
    random_state=42, stratify=virus_df_scaled_no_out_dmem
    ['Load'])
X_train_pbs, X_test_pbs, y_train_pbs, y_test_pbs =
    train_test_split(X_pbs, y_pbs, test_size = 0.2,
    random_state=42, stratify=virus_df_scaled_no_out_pbs['
    Load'])

#Returns list of unique values in y training data and the
     counts of each
unique_train_dmem, count_train_dmem= np.unique(
    y_train_dmem, return_counts=True)
unique_test_dmem, count_test_dmem= np.unique(y_test_dmem,
     return_counts=True)

unique_train_pbs, count_train_pbs= np.unique(y_train_dmem
    , return_counts=True)
unique_test_pbs, count_test_pbs= np.unique(y_train_dmem,
    return_counts=True)

#Returns the ratio of each unique value
train_dmem_ratio = dict(zip(unique_train_dmem,
    count_train_dmem/len(y_train_dmem)))
```

```
test_dmem_ratio = dict(zip(unique_test_dmem,
    count_test_dmem/len(y_test_dmem)))

train_pbs_ratio = dict(zip(unique_train_pbs,
    count_train_pbs/len(y_train_pbs)))
test_pbs_ratio = dict(zip(unique_test_pbs, count_test_pbs
    /len(y_test_pbs)))

#Check value counts
print("dmem:\nTrain set:\n\n{}      \n==========\n\
    nTest set: \n\n{}".format(train_dmem_ratio,
    test_dmem_ratio))
print("pbs:\nTrain set:\n\n{}      \n==========\n\
    nTest set: \n\n{}".format(train_pbs_ratio,
    test_pbs_ratio))

#Import neccesary regression models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor

#Initiate regression models and fit to training data to
    predict y
lin_reg = LinearRegression()
lin_reg.fit(X_train_dmem, y_train_dmem)
y_pred_lin = lin_reg.predict(X_test_dmem)

RFR = RandomForestRegressor()
RFR.fit(X_train_pbs, y_train_pbs)
y_pred_RFR = RFR.predict(X_test_pbs)

fig, ax = plt.subplots(1,2, figsize = (12,5))

ax[0].scatter(y_test_dmem, y_pred_lin, alpha = 0.8)
ax[0].plot([min(y_pred_lin), max(y_pred_lin)],[min(
    y_pred_lin), max(y_pred_lin)], color = 'r')
ax[0].set_xlabel("True Load")
ax[0].set_ylabel("Predicted Load")
ax[0].set_title("Linear Regression on dmem set")

ax[1].scatter(y_test_pbs, y_pred_RFR, alpha = 0.8)
ax[1].plot([min(y_pred_RFR), max(y_pred_RFR)],[min(
    y_pred_RFR), max(y_pred_RFR)], color = 'r')
ax[1].set_xlabel("True Load")
ax[1].set_ylabel("Predicted Load")
ax[1].set_title("Random Forest Regression on pbs set")
```

```
from sklearn.metrics import mean_squared_error,
    mean_absolute_error, r2_score

#Function to calculate metrics including mean squared
    error, mean absolute error and r2 score
def metrics(y, y_hat):
    mse = mean_squared_error(y, y_hat)
    mae = mean_absolute_error(y, y_hat)
    r2 = r2_score(y, y_hat)

    return mse, mae, r2

#Metrics calculated for each model
lin_reg_metrics = metrics(y_test_dmem, y_pred_lin)
print("Linear Regression:\nMean Squared Error = {}\nMean
    Absolute Error = {}\nR^2 Score = {}\n".format(
    lin_reg_metrics[0], lin_reg_metrics[1],
    lin_reg_metrics[2]))

RFR_metrics = metrics(y_test_pbs, y_pred_RFR)
print("Random Forest Regression:\nMean Squared Error =
    {}\nMean Absolute Error = {}\nR^2 Score = {}\n".format
    (RFR_metrics[0], RFR_metrics[1], RFR_metrics[2]))

from sklearn.model_selection import cross_val_score

#Perform Cross validation for each model to obtain Mean
    Squared Error
lin_reg_scores = cross_val_score(lin_reg, X_train_dmem,
    y_train_dmem, scoring = 'neg_mean_squared_error', cv =
     5)
lin_reg_score_mean = np.mean(-lin_reg_scores)

RFR_scores = cross_val_score(RFR, X_train_pbs,
    y_train_pbs, scoring = 'neg_mean_squared_error', cv =
    5)
RFR_score_mean = np.mean(-RFR_scores)

print("Mean Squared Error using Cross Validation Score:\
    nLinear Regression MSE = {}\nRandom Forest Regressor
    MSE = {}".format(np.round(lin_reg_score_mean, 2), np.
    round(RFR_score_mean, 4)))

#Perform Cross validation for each model to obtain r^2
    score
```

```
lin_reg_r_scores = cross_val_score(lin_reg, X_train_dmem,
    y_train_dmem, scoring = 'r2', cv = 5)
lin_reg_r_mean = np.mean(lin_reg_r_scores)

RFR_r_scores = cross_val_score(RFR, X_train_pbs,
   y_train_pbs, scoring = 'r2', cv = 5)
RFR_r_mean = np.mean(RFR_r_scores)

print("r^2 using Cross Validation Score:\nLinear
   Regression r^2 = {}\nRandom Forest Regressor r^2 =
   {}".format(np.round(lin_reg_r_mean, 2), np.round(
   RFR_r_mean, 2)))

#Model Optimization
from sklearn.model_selection import GridSearchCV

RF_Regressor = RandomForestRegressor(random_state=42)

#Select distribution of parameter values to be tested
params = {'n_estimators': [80, 100], 'max_depth': [None,
   10], 'min_samples_split': [2,3], 'min_samples_leaf':
   [1], 'max_features': ['log2',None]}

#Grid search of parameter values i.e. exhaustive search
grid_search = GridSearchCV(RF_Regressor, params, cv = 3,
   scoring = 'neg_mean_squared_error', error_score='raise
   ')
grid_search.fit(X_train_pbs, y_train_pbs)

#Return the best Random Forest Regressor model
best_RFR = grid_search.best_estimator_
#Return the parameters corressponding to the best model
grid_search.best_params_

#Use best model to predict values
predict_RFR_best = best_RFR.predict(X_test_pbs)
mse_best = mean_squared_error(y_test_pbs,
   predict_RFR_best)
print("Best Mean Squared Error for Random Forest
   Regressor = {}".format(mse_best))
print(r2_score(y_test_pbs, predict_RFR_best))

plt.scatter(y_test_pbs, predict_RFR_best, alpha = 0.8)
plt.plot([min(predict_RFR_best), max(predict_RFR_best)],[
   min(predict_RFR_best), max(predict_RFR_best)], color =
    'r')
```

```
plt.xlabel("True Load")
plt.ylabel("Predicted Load")

from sklearn.preprocessing import LabelBinarizer

#X and y data for classification
X_c_dmem = virus_df_scaled_no_out_dmem.drop(columns=['SID
    ','Type','Load'], axis = 1)
X_c_dmem = np.asarray(X_c_dmem)
y_c_dmem = virus_df_scaled_no_out_dmem['Type']

#Turning y data into binary code (X - 0 and Y - 1)
lb_dmem = LabelBinarizer()
y_c_dmem = lb_dmem.fit_transform(y_c_dmem)
y_c_dmem = np.squeeze(y_c_dmem)

X_c_pbs = virus_df_scaled_no_out_pbs.drop(columns=['SID
    ','Type','Load'], axis = 1)
X_c_pbs = np.asarray(X_c_pbs)
y_c_pbs = virus_df_scaled_no_out_pbs['Type']

lb_pbs = LabelBinarizer()
y_c_pbs = lb_pbs.fit_transform(y_c_pbs)
y_c_pbs = np.squeeze(y_c_pbs)

#Obtaining number of X and Y values within each dataset
unique_dmem, count_dmem = np.unique(y_c_dmem,
    return_counts=True)
ratio_samps_dmem = dict(zip(unique_dmem, count_dmem))
print("dmem - Ratio of Types before Sampling: \n{}\n".
    format(ratio_samps_dmem))

unique_pbs, count_pbs = np.unique(y_c_pbs, return_counts=
    True)
ratio_samps_pbs = dict(zip(unique_pbs, count_pbs))
print("pbs - Ratio of Types before Sampling: \n{}".format
    (ratio_samps_pbs))

from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

#SMOTE used to synthesize more observations with Type X
over_dmem = SMOTE(sampling_strategy=0.794, random_state
    =42)
X_c_dmem, y_c_dmem = over_dmem.fit_resample(X_c_dmem,
    y_c_dmem)
```

```
#Random Under Sampler used to eliminate observations with
     Type Y to balance dataset
under_dmem = RandomUnderSampler(sampling_strategy='auto',
     random_state=42)
X_c_dmem, y_c_dmem = under_dmem.fit_resample(X_c_dmem,
    y_c_dmem)

over_pbs = SMOTE(sampling_strategy=0.781, random_state
    =42)
X_c_pbs, y_c_pbs = over_pbs.fit_resample(X_c_pbs, y_c_pbs
    )

under_pbs = RandomUnderSampler(sampling_strategy='auto',
    random_state=42)
X_c_pbs, y_c_pbs = under_pbs.fit_resample(X_c_pbs,
    y_c_pbs)

#Number of Types counted to ensure balanced dataset
unique_dmem, count_dmem = np.unique(y_c_dmem,
    return_counts=True)
ratio_samps_dmem = dict(zip(unique_dmem, count_dmem))
print("dmem - Ratio of Types After Sampling: \n{}\n".
    format(ratio_samps_dmem))

unique_pbs, count_pbs = np.unique(y_c_pbs, return_counts=
    True)
ratio_samps_pbs = dict(zip(unique_pbs, count_pbs))
print("pbs - Ratio of Types After Sampling: \n{}".format(
    ratio_samps_pbs))

#Spliting datasets into training and test data
X_c_train_dmem, X_c_test_dmem, y_c_train_dmem,
    y_c_test_dmem = train_test_split(X_c_dmem, y_c_dmem,
    test_size=0.2, random_state=42, stratify=
    virus_df_scaled_no_out_dmem['Type'])
X_c_train_pbs, X_c_test_pbs, y_c_train_pbs, y_c_test_pbs
    = train_test_split(X_c_pbs, y_c_pbs, test_size=0.2,
    random_state=42, stratify=virus_df_scaled_no_out_pbs['
    Type'])

unique_train_dmem_c, count_train_dmem_c= np.unique(
    y_c_train_dmem, return_counts=True)
unique_test_dmem_c, count_test_dmem_c= np.unique(
    y_c_test_dmem, return_counts=True)
```

```
unique_train_pbs_c, count_train_pbs_c= np.unique(
    y_c_train_pbs, return_counts=True)
unique_test_pbs_c, count_test_pbs_c= np.unique(
    y_c_test_pbs, return_counts=True)

train_dmem_ratio_c = dict(zip(unique_train_dmem_c,
    count_train_dmem_c/len(y_c_train_dmem)))
test_dmem_ratio_c = dict(zip(unique_test_dmem_c,
    count_test_dmem_c/len(y_c_test_dmem)))

train_pbs_ratio_c = dict(zip(unique_train_pbs_c,
    count_train_pbs_c/len(y_c_train_pbs)))
test_pbs_ratio_c = dict(zip(unique_test_pbs_c,
    count_test_pbs_c/len(y_c_test_pbs)))

#Check value counts
print("CLASSIFICATION:\n\n dmem:\nTrain set:\n\n{}      \n
    ==========\n\nTest set: \n\n{}".format(
    train_dmem_ratio_c, test_dmem_ratio_c))
print("pbs:\nTrain set:\n\n{}      \n==========\n\
    nTest set: \n\n{}".format(train_pbs_ratio_c,
    test_pbs_ratio_c))

#import classification models
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC

#Logistic Regressiong model declared and applied to the
    test data
model_LR = LogisticRegression(max_iter=300)
model_LR.fit(X_c_train_dmem, y_c_train_dmem)
predict_LR = model_LR.predict(X_c_test_dmem)
score_LR = accuracy_score(y_c_test_dmem, predict_LR)

#Linear SVC model declared and applied to the test data
model_SVC = LinearSVC()
model_SVC.fit(X_c_train_pbs, y_c_train_pbs)
predict_SVC = model_SVC.predict(X_c_test_pbs)
score_SVC = accuracy_score(y_c_test_pbs, predict_SVC)

print("Logistic Regression Accuracy = {}".format(score_LR
    ))
print("Linear SVC Accuracy = {}".format(score_SVC))
```

```
#Bias and varience calculated for Logistic Regression
    model
variance_LR = np.var(predict_LR)
sse_LR = np.mean((np.mean(predict_LR) - y_c_test_dmem) **
    2)
bias_LR = sse_LR - variance_LR

print(variance_LR)
print(bias_LR)

#Bias and varience calculated for Linear SVC model
variance_SVC = np.var(predict_SVC)
sse_SVC = np.mean((np.mean(predict_SVC) - y_c_test_pbs)
    ** 2)
bias_SVC = sse_SVC - variance_SVC

print(variance_SVC)
print(bias_SVC)

#Cross Validation performed for both models
log_reg_score = cross_val_score(model_LR, X_c_train_dmem,
    y_c_train_dmem, scoring='r2', cv = 10)
lin_SVC_score = cross_val_score(model_SVC, X_c_train_pbs,
    y_c_train_pbs, scoring='r2', cv = 10)
print("Logistic Regression r^2 score = {}\nLinear SVC r^2
    score = {}".format(np.mean(log_reg_score), np.mean(
    lin_SVC_score)))

from sklearn.metrics import classification_report

#Create classification reports for both models to observe
    precision, recall and f1 scores
log_reg_report = pd.DataFrame(classification_report(
    y_c_test_dmem, predict_LR, output_dict = True))
print("Logistic Regression")
print(log_reg_report)
print("==========================================================\n")
lin_SVC_report = pd.DataFrame(classification_report(
    y_c_test_pbs, predict_SVC, output_dict = True))
print("Linear SVC")
print(lin_SVC_report)

from sklearn.metrics import confusion_matrix

fig, ax = plt.subplots(1,2, figsize = (10,5))
ax = ax.flatten()
```

```
#Confusion matrix and heatmap created for each model
conf_matrix_LR = confusion_matrix(y_c_test_dmem,
    predict_LR)
sns.heatmap(conf_matrix_LR, annot=True, fmt = 'd', cmap =
     'Blues', ax = ax[0])
ax[0].set_xlabel("Predicted Label")
ax[0].set_ylabel("True Label")
ax[0].set_title("Logistic Regression (dmem)")

conf_matrix_SVC = confusion_matrix(y_c_test_pbs,
    predict_SVC)
sns.heatmap(conf_matrix_SVC, annot=True, fmt = 'd', cmap
    = 'Blues', ax = ax[1])
ax[1].set_xlabel("Predicted Label")
ax[1].set_ylabel("True Label")
ax[1].set_title("Linear SVC (pbs)")

plt.tight_layout()

LR_probs = model_LR.predict_proba(X_c_test_dmem)[:,1]

from sklearn.metrics import roc_curve, auc

#Obtain False Positive Rate, True Positive Rate and
    threshold for LR model
fpr, tpr, thresholds = roc_curve(y_c_test_dmem, LR_probs)

#Area under curve value
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label='ROC Curve (AUC =
    {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='red', linestyle='--') #
    Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

from sklearn.decomposition import PCA
#Initiate PCA model with number of components
```

```
pca_light_dmem = PCA(n_components=2)
pca_light_pbs = PCA(n_components=2)

#Fit PCA model to only wavelength data
pca_light_dmem.fit(light_df_scaled_dmem)
pca_light_pbs.fit(light_df_scaled_pbs)

#Transform original data
scores_dmem = pca_light_dmem.transform(
    light_df_scaled_dmem)
scores_pbs = pca_light_pbs.transform(light_df_scaled_pbs)

print("Explained Variance Ratio for dmem dataset {}".
    format(pca_light_dmem.explained_variance_ratio_))
print("Explained Variance Ratio for pbs dataset {}".
    format(pca_light_pbs.explained_variance_ratio_))

virus_df_light_dmem  = virus_df_dmem.drop(columns=
    light_cols, axis = 1)
virus_df_light_pbs  = virus_df_pbs.drop(columns=
    light_cols, axis = 1)

#Create dataframe with reduced data
scores_df_dmem = pd.DataFrame(scores_dmem, columns=['PC1
    ','PC2'])
scores_df_pbs = pd.DataFrame(scores_pbs, columns=['PC1','
    PC2'])

virus_df_light_dmem = virus_df_light_dmem.reset_index()
scores_df_dmem = scores_df_dmem.reset_index()

virus_df_light_pbs = virus_df_light_pbs.reset_index()
scores_df_pbs = scores_df_pbs.reset_index()

virus_df_reduced_dmem = pd.concat([virus_df_light_dmem,
    scores_df_dmem], axis = 1)
virus_df_reduced_dmem.drop(columns=['index'], axis = 1)

virus_df_reduced_pbs = pd.concat([virus_df_light_pbs,
    scores_df_pbs], axis = 1)
virus_df_reduced_pbs.drop(columns=['index'], axis = 1)

virus_df_reduced_dmem = virus_df_reduced_dmem.drop(
    columns=['index'], axis = 1)

#Distribution for both Principal Components in dmem set
```

```
fig, ax = plt.subplots(1,2, figsize = (8,6))
fig.suptitle("Distribution of values for both Principal
    Components")

sns.boxplot(virus_df_reduced_dmem['PC1'], ax = ax[0])
ax[0].set_xlabel("PC1")
ax[0].set_ylabel("Distribution")

sns.boxplot(virus_df_reduced_dmem['PC2'], ax = ax[1])
ax[1].set_xlabel("PC2")
ax[1].set_ylabel("Distribution")

scores_df_pbs = scores_df_pbs.drop(columns=['index'])

#Detect outliers using Isolation Forest
IForest = IsolationForest(random_state=42)

outlier_pred_pbs_clus = IForest.fit_predict(scores_df_pbs
    )

#Outliers when -1 so single out rows with 1
virus_df_reduced_pbs = virus_df_reduced_pbs.iloc[
    outlier_pred_pbs_clus == 1]

virus_df_reduced_pbs = virus_df_reduced_pbs.drop(columns
    =['index'], axis = 1)

X_clus_pbs = virus_df_reduced_pbs.drop(columns=['SID','
    Type','Load'])
y_clus_pbs = virus_df_reduced_pbs['Load']

#X_train_clus_pbs, X_clus_test_pbs, y_train_clus_pbs,
    y_test_clus_pbs = train_test_split(X_clus_pbs,
    y_clus_pbs, test_size=0.2, random_state=1)

from scipy.cluster.hierarchy import linkage

complete_clustering = linkage(X_clus_pbs, method="
    complete", metric="euclidean")

from scipy.cluster.hierarchy import dendrogram

dendrogram(complete_clustering)
plt.xlabel("Data Point")
plt.ylabel("Distance")
plt.show()
```