COMP 3032 - Machine Learning - Assignment 1 (Due date: 5:00 pm Wednesday, 27 September 2023)

Name: Quang Dong Nguyen
Student_id: 20744696
Class: Machine Learning - Semester 2

Task 1 (12 Marks):

Question 1:

Blood pressure dataset *pressure.csv* contains examples of systolic pressures and other features from different persons.

Answer Below:

Firstly, we import libraries which will be used later on to extract data and to build various models.

- The Pandas library is used for working with datasets, especially here, we want to see how the dataset is structurised in a dataframe.
- Then the Numpy library will be used to construct some arrays for later on modelling.

In [164]:
```
import numpy as np
import pandas as pd
```

Using pandas read.csv function, we can load the csv file, named as'*bloodpressure-23.csv*' and stored in my local directory.

In [165]:
```
B_p = pd.read_csv('D:/dong;s junior (WSU)/Second Year - 2023/Semester 2/Machine Le
B_p.head(5)
```

Out[165]:

	ID-NUMBER	AGE	ED-LEVEL	SMOKING STATUS	EXERCISE	WEIGHT	SERUM-CHOL	SYSTOLIC	IQ	SOI
0	1	27	2	1	1	120	193	126	118	
1	2	18	1	0	1	145	210	120	105	
2	3	32	2	0	0	118	196	128	115	
3	4	24	2	0	1	162	208	129	108	
4	5	19	1	2	0	106	188	119	106	

Question 2:

Create polynomial regression models, to predict systolic pressure using the SERUMCHOL feature, for degrees vary from 1 to 14. Perform 10-fold cross validation. Calculate its square roots for the mean square errors (RMSE), and the mean RMSE. Display the mean RMSEs for the 14 different degrees. Produce a cross validation error plot using the mean RMSE with 1 to 14 different degrees.

Let's view the 'SERUM-CHOL' and 'SYSTOLIC' features in the dataframe first.

In [166]:
```
B_p[['SERUM-CHOL','SYSTOLIC']].head(5)
```

Out[166]:

	SERUM-CHOL	SYSTOLIC
0	193	126
1	210	120
2	196	128
3	208	129
4	188	119

Answer Below:

In [167]:
```
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
```

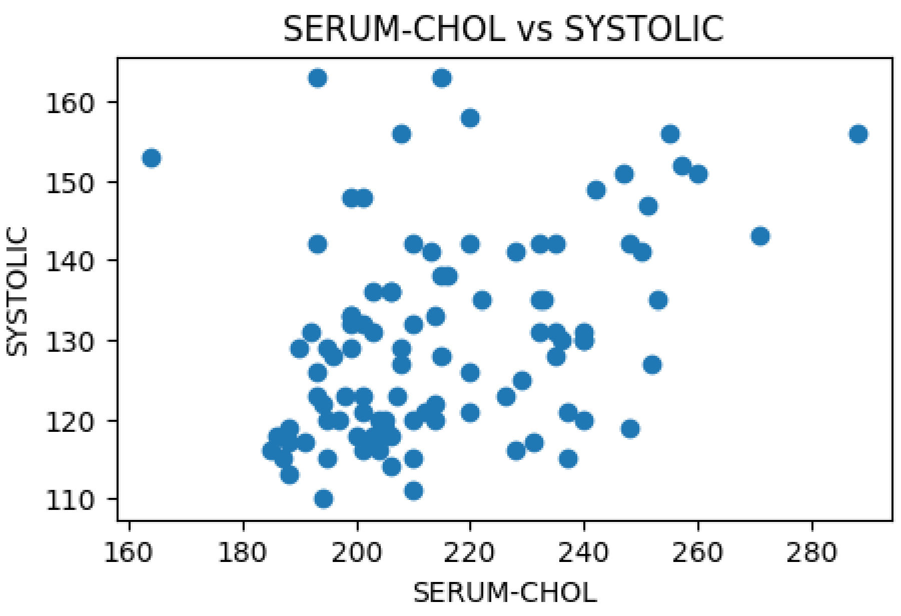
We now extract the columns `SERUM-CHOL` and `SYSTOLIC` from `B_p` dataframe, then reshape it to numpy array

In [168]:
```
X = B_p['SERUM-CHOL'].values.reshape(-1, 1)
y = B_p['SYSTOLIC'].values
```

Visualisation of data

We can then follow on with visualising the extracted features on graph, in order to see how well do the data correlate in general.

In [169]:
```
#visualisation of data
import matplotlib.pyplot as plt
plt.figure(figsize = (5,3))
plt.scatter(X, y)
plt.xlabel('SERUM-CHOL')
plt.ylabel('SYSTOLIC')
plt.title('SERUM-CHOL vs SYSTOLIC')
plt.show()
```



Graph Explanation:

- At first we can see that between SYSTOLIC level and SERUM-CHOL level, there is a positive, medium correlation. As Serum-chol level increases, the systolic level also increases.
- However, there is also an increase in variance between them as one of the feature increases, e.g. the higher increase in Serum-chol level, the larger the variance is among the systolic and serum-chol level.

Now we build a polynomial regression model with degrees ranging from 1 to 14 on the two extracted features above. By using `PolynomialFeatures` we can build the polynomial model with our own desired number of degrees.

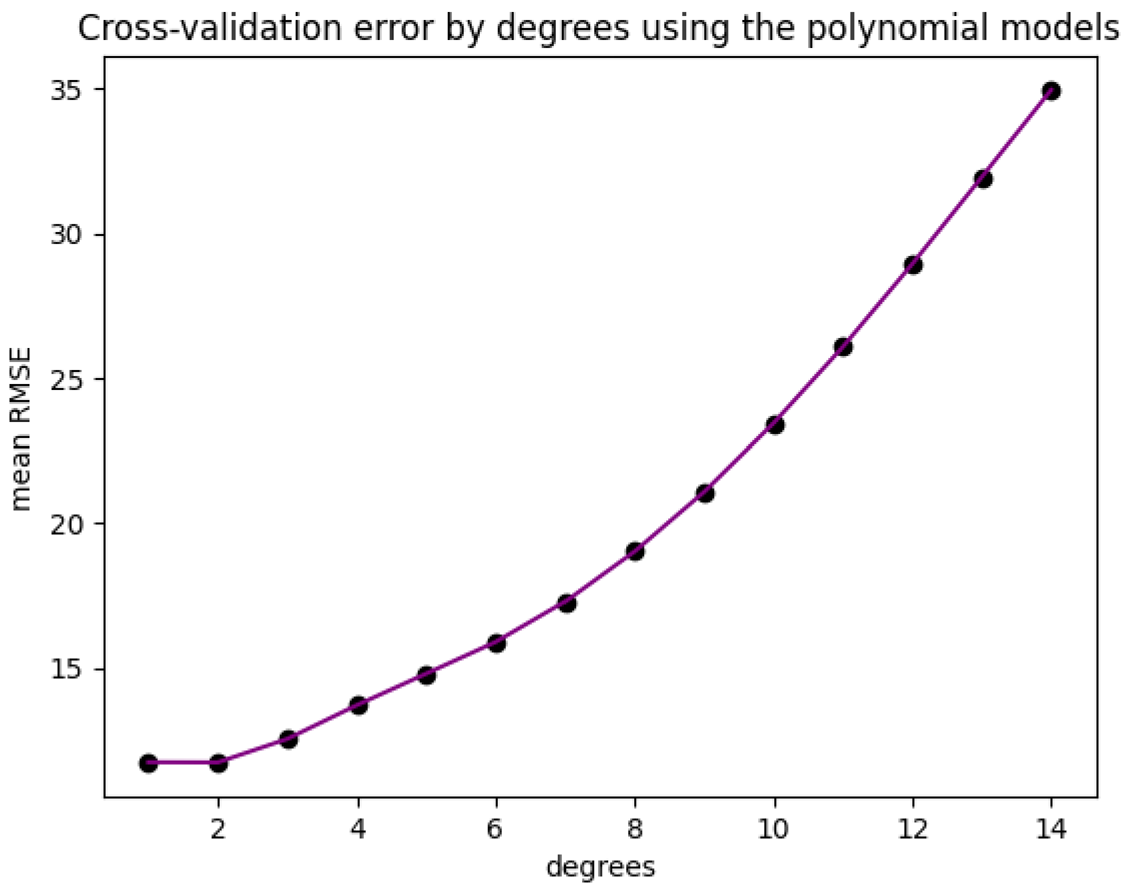
Using for loop to produce polynomial regression models for degrees vary from 1 to 14 and a cross-validation error plot using the mean RMSE with 1 to 14 different degrees

In [170]:
```
m_poly_rmse_scores = []

for i in range(1,15):
    poly = PolynomialFeatures(degree = i, include_bias = False)
    X_poly= poly.fit_transform(X)
    poly_reg_scores = cross_val_score(LinearRegression(), X_poly, y, scoring = 'ne
    poly_rmse_scores = np.sqrt(-poly_reg_scores) #converts to square roots of the
    mean_poly_rmse_scores = poly_rmse_scores.mean() #converts to mean square roots
    m_poly_rmse_scores.append(mean_poly_rmse_scores) #we append it to the assigned
```

In [171]:
```
line = [*range(1,15)]
plt.scatter(line, m_poly_rmse_scores, c= 'black')
plt.plot(line, m_poly_rmse_scores, c = 'purple')
plt.xlabel('degrees')
plt.ylabel('mean RMSE')
plt.title('Cross-validation error by degrees using the polynomial models')
```

Out[171]:
```
Text(0.5, 1.0, 'Cross-validation error by degrees using the polynomial models')
```



In [172]:
```
print('Mean RMSE: {}'.format(m_poly_rmse_scores))
```

```
Mean RMSE: [11.743903387181604, 11.73807811241585, 12.547523654668435, 13.71093092
37205, 14.800290560812925, 15.894371890471254, 17.28980827219123, 19.0265760094120
96, 21.08073172398104, 23.44036925723232, 26.07802523227423, 28.940500707085448, 3
1.940518498970636, 34.951218567229475]
```

Question 3:

Select the best degree, and explains why briefly. Print its intercept and coefficients.

Answer Below:

The best degree for the polynomial model is 2. According to the plot, Cross-validation error by degrees using the polynomial models, the only polynomial model that produced the lowest mean RMSE value is the polynomial model with the degree of 2, where its produced mean RMSE is approximately 11.73807911241585. This value is justifiable and valid, because it demonstrates that the average deviation between the actual values and the predicted values by the model is lower than every other models. And it can also be told that the model is rightly fitted, since it used low number of degrees and still got the lowest mean RMSE value among other degrees.

In [173]:
```
poly = PolynomialFeatures(degree = 2, include_bias = False)
X_poly= poly.fit_transform(X)
Lin_reg_model = LinearRegression().fit(X_poly,y)
print("Polynomial model Coefficient: {}".format(Lin_reg_model.coef_)) #coefficien
print("Polynomial model Intercept: {}".format(Lin_reg_model.intercept_)) #Intercep
```

```
Polynomial model Coefficient: [-1.54847161  0.00394948]
Polynomial model Intercept: 278.5603624163489
```

Question 4:

Create a multiple linear regression model to predict systolic pressure using all the relevant features. Print its coefficients. Perform 10-fold cross validation. Calculate its square roots of the mean square errors (RMSE), and the mean RMSE, and dislp y the mean RMSE.

In [174]:
```
from sklearn.linear_model import LinearRegression
```

In [175]:
```
B_p.head(2)
```

Out[175]:

	ID-NUMBER	AGE	ED-LEVEL	SMOKING STATUS	EXERCISE	WEIGHT	SERUM-CHOL	SYSTOLIC	IQ	SOI
0	1	27	2	1	1	120	193	126	118	
1	2	18	1	0	1	145	210	120	105	

Answer Below:

Select the relevant features for linear regression model:

- `AGE` , `ED-LEVEL` , `SMOKING STATUS` , `EXERCISE` , `WEIGHT` , `SERUM-CHOL` , `IQ` , `SODIUM` , `GENDER` and `MARITAL-STATUS` are all relevant features can be used to build the linear regression model.
- `ID-NUMBER` and `NAME` are irrelevant due to neither containing meaningful information about the data nor having impact on systolic blood level.

In [176]:
```
X = B_p[['AGE','ED-LEVEL','SMOKING STATUS', 'EXERCISE', 'WEIGHT', 'SERUM-CHOL', 'I
X['GENDER'].replace(['M','F'],[1,2], inplace = True)
X['MARITAL-STATUS'].replace(['D','M','S','W'],[1,2,3,4], inplace = True)
X = X.values
y = B_p[['SYSTOLIC']].values
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_13712\2457168825.py:2: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  X['GENDER'].replace(['M','F'],[1,2], inplace = True)
C:\Users\Dell\AppData\Local\Temp\ipykernel_13712\2457168825.py:3: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  X['MARITAL-STATUS'].replace(['D','M','S','W'],[1,2,3,4], inplace = True)
```

Now, we create a multiple linear regression model to predict systolic pressure

In [177]:
```
lin_model = LinearRegression().fit(X, y)
print('Multi-linear model Coefficient: {}'.format(lin_model.coef_))
print('Multi-linear model Intercept: {}'.format(lin_model.intercept_))
```

```
Multi-linear model Coefficient: [[ 0.33056784 -0.87090274 -0.08753526 -0.07509179
0.30396451  0.01745461
  -0.04049222  0.06966721 10.95029683 -1.09102542]]
Multi-linear model Intercept: [50.3590063]
```

Perform 10-fold Cross-Validation on multiple linear regression model

In [178]:
```
#### CROSS-VALIDATION
#10 fold
from sklearn.model_selection import cross_val_score
lin_scores = cross_val_score(LinearRegression(), X, y, scoring = 'neg_mean_square
lin_rmse_scores = np.sqrt(-lin_scores) # sqrt mean squared error
mean_lin_rmse_scores = np.mean(lin_rmse_scores) # mean_RMSE
print("RMSE: {}".format(lin_rmse_scores))
print('Mean RMSE: {}'.format(mean_lin_rmse_scores))
```

```
RMSE: [8.32114789 7.55446484 8.58877345 5.94879844 7.51700528 7.26700244
 6.06383438 8.86175363 7.84214218 7.57693122]
Mean RMSE: 7.5541853765972835
```

Question 5:

Build a ridge regression model of the above (i.e. item 4) using α = 0.1. Print its coefficients. Perform 10-fold cross validation. Calculate its square roots of the mean square errors (RMSE), and the mean RMSE, and display the mean RMSE.

Answer Below:

In [179]:
```
from sklearn.linear_model import Ridge
```

In [180]:
```
X = B_p[['AGE','ED-LEVEL','SMOKING STATUS', 'EXERCISE', 'WEIGHT', 'SERUM-CHOL', '
X['GENDER'].replace(['M','F'],[1,2], inplace = True)
X['MARITAL-STATUS'].replace(['D','M','S','W'],[1,2,3,4], inplace = True)
X = X.values
y = B_p[['SYSTOLIC']].values
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_13712\2457168825.py:2: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  X['GENDER'].replace(['M','F'],[1,2], inplace = True)
C:\Users\Dell\AppData\Local\Temp\ipykernel_13712\2457168825.py:3: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  X['MARITAL-STATUS'].replace(['D','M','S','W'],[1,2,3,4], inplace = True)
```

In [181]:
```
Rig_model = Ridge(alpha = 0.1)
Rig_model = Rig_model.fit(X,y)
print("Ridge Model Coefficient: {}".format(Rig_model.coef_))
print("Ridge Model Intercept: {}".format(Rig_model.intercept_))
```

```
Ridge Model Coefficient: [[ 0.33183712 -0.87329823 -0.08645678 -0.08344129  0.3021
4144  0.01745305
  -0.04027576  0.07087436 10.83667682 -1.09241833]]
Ridge Model Intercept: [50.57634328]
```

Build a 10-fold cross validation with Ridge model

In [182]:
```
Rig_cv = cross_val_score(Rig_model, X, y, scoring = 'neg_mean_squared_error', cv =
Rig_rmse_scores = np.sqrt(-Rig_cv)
mean_Rig_rmse_scores = Rig_rmse_scores.mean()
print("RMSE: {}".format(Rig_rmse_scores))
print("Mean RMSE: {}".format(mean_Rig_rmse_scores))
```

```
RMSE: [8.3382567  7.56784831 8.58048433 5.94191182 7.48934397 7.26350031
 6.05626059 8.86595721 7.85123569 7.57849498]
Mean RMSE: 7.553329392826423
```

Question 6:

Select the best model of the three, and explains why briefly

Answer Below:

Now, we have the mean RMSE for each model as shown below:

- Polynomial Regression model: 11.73807911241585
- Multiple Linear Regression model: 7.5541853765972835
- Ridge Regression model: 7.55332939286423

Among all models, the best model is the Ridge regression Model with alpha 0.1, because it produced the lowest mean Root Mean Squared Error value among all three models. This depicts that its predicted values do not deviate too far away from the actual values, meaning that the model has a higher capability of predicting the target value than other models.

Task 2 (8 marks):

Question 1:

This task involves MNIST Digit Classification using PCA and Logistic Regression. Load the renowned MNIST (‘mnist 784’) dataset, which consists of a large collection of handwritten digit images. Your task is to reduce the number of features first, n d then build a binary classification model to distinguish between the digit “6” and ll other digits (not “6”).

Answer Below:

Firstly, we load the libraries that will be used to:

- Load the MNIST dataset,
- Perform PCA,
- Split datasets into training set and testing set,
- Build Logistic Regression model,
- Calculate accuracy score of the model and
- Create confusion matrix

In [183]:
```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

From them, we load the MNIST dataset from `sklearn.datasets` library using the given function `fetch_openml`

In [184]:
```
mnist = fetch_openml('mnist_784', version = 1, as_frame = False) #we want to set v
#as_frame = False, will set dataset in Numppy arrays format
mnist.target = mnist.target.astype(np.uint8) #uint8 cause we want the range to hav
```

```
E:\Other_tool\envpckg\py3_10_9\lib\site-packages\sklearn\datasets\_openml.py:1002:
FutureWarning: The default value of `parser` will change from `'liac-arff'` to `'a
uto'` in 1.4. You can set `parser='auto'` to silence this warning. Therefore, an `
ImportError` will be raised from 1.4 if the dataset is dense and pandas is not ins
talled. Note that the pandas parser may return different data types. See the Notes
Section in fetch_openml's API doc for details.
  warn(
```

In [185]:
```
X_6 = mnist['data']
y_6 = mnist['target']
```

Question 2:

Perform Principal Component Analysis (PCA) on the feature data to reduce its dimensionality while retaining 88% of the overall explained variance ratio.

Answer Below:

We must fit the MNIST's extracted data to the PCA model to find the final number dimension that will preserve 88% of the variance in the data.

In [186]:
```
pca = PCA() #create PCA
pca = pca.fit(X_6)
```

In [187]:
```
#find the number of dimensions that preserve 88% of the variance in the data
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.88) + 1 #np.argmax return the index, therefore + 1 to ge

#now we fit new pca X values to obtain X_reduced in lower dimension.
pca = PCA(n_components = d)
X_reduced = pca.fit_transform(X_6)
```

Therefore, it can be seen in the next code that 88% of the variance can be preserved through only 74 numbers of dimensions/features. This means that Principal Component Analysis model has worked effortlessly well with the MNIST dataset on preserving the data's meaning.

In [188]:
```
print(X_6.shape)
print(X_reduced.shape) #reduced in number of dimensions/features, from 784 to 74
print('Number of principal components preserved: {}'.format(d))
```

```
(70000, 784)
(70000, 74)
Number of principal components preserved: 74
```

Question 3:

Split the data into training and testing sets. A common split ratio is 80% training and 20% testing.

Answer Below:

In [189]:
```
X_train, X_test, y_train, y_test = train_test_split(X_reduced, y_6, test_size = 0
```

Question 4:

Create a logistic Regression model using the reduced feature dataset.

Answer Below:

In [190]:
```
#here we create a new variable that will predict only number 6
y_train1 = y_train == 6
y_test1 = y_test == 6
```

In [191]:
```
log_reg = LogisticRegression()
log_reg = log_reg.fit(X_train, y_train1) #fit the training sets into logistic Regr
```

```
E:\Other_tool\envpckg\py3_10_9\lib\site-packages\sklearn\linear_model\_logistic.p
y:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Question 5:

Use this model to predict the language for the training dataset and the testing dataset.

Answer Below:

Predict the language for the training set

Use the `log_reg` model created above to predict values for the training set

In [192]:
```
y_pred_train = log_reg.predict(X_train)
```

Predict the language for the testing set

Same as above, use the `log_reg` model created to predict values for the testing set

In [193]:
```
y_pred_test = log_reg.predict(X_test)
```

Question 6:

Print the number of principal components preserved. Print the prediction accuracy (proportion of correct predictions) of your model on the training set. Print the prediction accuracy, the confusion matrix, and the misclassified digits (i.e. wrn g predtiitons) of your model on the testing set.

Answer Below:

Number of Principal Components preserved

In [194]:
```
print('Number of principal components preserved: {}'.format(d))
```

```
Number of principal components preserved: 74
```

Prediction Accuracy

In [195]:
```
#accuracy score on the training set
ac_score = accuracy_score(y_train1, y_pred_train)
print("Accuracy: {}".format(ac_score*100), "%") #the prediction accuracy on the tr
```

```
Accuracy: 98.64642857142857 %
```

In [196]:
```
#accuracy score on the testing set
ac_score = accuracy_score(y_test1, y_pred_test)
print("Accuracy: {}".format(ac_score*100), "%") #the prediciton accuracy on the te
```

```
Accuracy: 98.65714285714286 %
```

Confusion matrix

In [197]:
```
#confusion matrix on the training set using the model above
confusion_matrix(y_train1, y_pred_train)
#The confusion matrix shows that we have a total of:
# 324 values are misclassified as the digit 6 while they are actually not
# 434 values are misclassified as not the digit 6 while they are truly 6
# 50147 values are correctly classified as not 6 and
# 5095 values are correctly classified as 6.
```

Out[197]:
```
array([[50147,   324],
       [  434,  5095]], dtype=int64)
```

```
In [198]: #confusion matrix for the test set using the model above
          confusion_matrix(y_test1, y_pred_test)
          #The confusion matrix shows that we have a total of:
          # 81 values are misclassified as the digit 6 while they are actually not
          # 107 values are misclassified as not the digit 6 while they are truly 6.
          # 12572 values are correctly classified as not 6 and
          # 1240 values are correctly classified as 6
```

```
Out[198]: array([[12572,    81],
                 [  107,  1240]], dtype=int64)
```

Misclassified Digits

```
In [199]: misclas_digit = np.where(y_test1 != y_pred_test)[0]
          y_test[misclas_digit] #these are the misclassified digits, where the predicted y
          #of the testing set.
```

```
Out[199]: array([6, 0, 6, 6, 6, 6, 5, 6, 6, 8, 6, 5, 6, 6, 0, 8, 8, 2, 6, 6, 5, 6,
                 6, 6, 5, 2, 4, 6, 6, 6, 6, 2, 6, 6, 6, 5, 3, 4, 5, 6, 6, 5, 5, 6,
                 1, 0, 6, 6, 6, 6, 6, 6, 0, 6, 2, 6, 2, 6, 5, 5, 3, 6, 4, 6, 0, 5,
                 6, 6, 0, 6, 6, 6, 6, 6, 2, 2, 6, 6, 2, 4, 6, 0, 6, 8, 6, 6, 4, 6,
                 0, 6, 4, 2, 6, 6, 6, 7, 2, 6, 6, 6, 6, 6, 8, 6, 6, 6, 6, 6, 6, 6,
                 6, 6, 6, 6, 2, 6, 6, 8, 6, 6, 6, 6, 0, 4, 6, 6, 6, 4, 0, 6, 2, 3,
                 0, 6, 6, 2, 4, 5, 5, 2, 6, 0, 6, 6, 6, 2, 6, 2, 2, 6, 8, 6, 6, 5,
                 6, 6, 0, 2, 6, 6, 8, 6, 6, 6, 6, 6, 4, 0, 6, 2, 5, 6, 0, 8, 6, 2,
                 4, 6, 6, 8, 6, 4, 8, 4, 6, 5, 6, 8], dtype=uint8)
```

Question 7:

What do you think of the model generated (good, underfit, overfit)? Briefly explains why

Answer Below:

The model is accurately good due to its highly well prediction performance on the training data and as well as on the testing set. With an accuracy of 98.65% for the training set and approximately 98.66% for the testing set, it is a good model to be used for predicting the language. Furthermore, with the uses of PCA to reduce the number of dimensions while preserving 88% of the variance of data, the model's complexity is reduced to a safe zone without overcomplicating the model with a high number of dimensions/features which could potentially lead overfitting.

END OF TASK