Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2
1st Semester	AY 2023-2024

Assignment 5.2	Build and Apply Multilayer Perceptron
Name	Abad, Julia Marie Iberet
Section	CPE32S3
Date Performed:	March 20, 2024
Date Submitted:	March 26, 2024
Instructor:	Engr. Roman Richard

In this assignment, you are task to build a multilayer perceptron model. The following are the requirements:

- Choose any dataset
- Explain the problem you are trying to solve
- Create your own model
- Evaluate the accuracy of your model

Dataset

Dataset Link: https://archive.ics.uci.edu/dataset/186/wine+quality

About the Dataset

The two datasets are related to red and white variants of the Portuguese "Vinho Verde" wine. For more details, consult: http://www.vinhoverde.pt/en/ or the reference [Cortez et al., 2009]. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are many more normal wines than excellent or poor ones). Outlier detection algorithms could be used to detect the few excellent or poor wines. Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.

Has Missing Values?

No

The problem to solve

The problem to solve in this dataset is to predict the quality of wine based on its physicochemical properties. The dataset contains information about various attributes such as fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol, along with the quality score ranging from 0 to 10. By utilizing this data, the goal is to develop a multilayer perceptron (MLP) model that can accurately predict the wine quality score given these physicochemical characteristics. This model will be trained on a portion of the dataset and then evaluated on a separate test set to assess its accuracy and predictive capabilities.

Creating own model and Evalating the accuracy of the model

Import necesary libraries and dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Activation

whitewine = pd.read_csv('winequality-white.csv', sep=';')
redwine = pd.read_csv('winequality-red.csv', sep=';')
wine = pd.concat([redwine, whitewine])
```

Understanding the features and structure of the dataset

```
redwine = redwine.iloc[:1000]
print(redwine)
```

```
     fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0              7.4              0.70         0.00            1.90      0.076
1              7.8              0.88         0.00            2.60      0.098
2              7.8              0.76         0.04            2.30      0.092
3             11.2              0.28         0.56            1.90      0.075
4              7.4              0.70         0.00            1.90      0.076
..             ...               ...          ...             ...        ...
995            7.7              0.60         0.06            2.00      0.079
996            5.6              0.66         0.00            2.20      0.087
997            5.6              0.66         0.00            2.20      0.087
998            8.9              0.84         0.34            1.40      0.050
999            6.4              0.69         0.00            1.65      0.055

     free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                   11.0                  34.0  0.99780  3.51       0.56
1                   25.0                  67.0  0.99680  3.20       0.68
2                   15.0                  54.0  0.99700  3.26       0.65
3                   17.0                  60.0  0.99800  3.16       0.58
```

```
4                   11.0                  34.0  0.99780  3.51       0.56
..                   ...                   ...      ...   ...        ...
995                 19.0                  41.0  0.99697  3.39       0.62
996                  3.0                  11.0  0.99378  3.71       0.63
997                  3.0                  11.0  0.99378  3.71       0.63
998                  4.0                  10.0  0.99554  3.12       0.48
999                  7.0                  12.0  0.99162  3.47       0.53

     alcohol  quality
0        9.4        5
1        9.8        5
2        9.8        5
3        9.8        6
4        9.4        5
..       ...      ...
995     10.1        6
996     12.8        7
997     12.8        7
998      9.1        6
999     12.9        6

[1000 rows x 12 columns]
```

```
whitewine = whitewine.iloc[:1000]
print(whitewine)
```

```
     fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0              7.0              0.27         0.36           20.70      0.045
1              6.3              0.30         0.34            1.60      0.049
2              8.1              0.28         0.40            6.90      0.050
3              7.2              0.23         0.32            8.50      0.058
4              7.2              0.23         0.32            8.50      0.058
..             ...               ...          ...             ...        ...
995            7.8              0.27         0.34            1.60      0.046
996            6.0              0.26         0.34            1.30      0.046
997            6.1              0.24         0.27            9.80      0.062
998            8.0              0.24         0.30           17.45      0.056
999            7.6              0.21         0.60            2.10      0.046

     free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                   45.0                 170.0   1.0010  3.00       0.45
1                   14.0                 132.0   0.9940  3.30       0.49
2                   30.0                  97.0   0.9951  3.26       0.44
3                   47.0                 186.0   0.9956  3.19       0.40
4                   47.0                 186.0   0.9956  3.19       0.40
..                   ...                   ...      ...   ...        ...
995                 27.0                 154.0   0.9927  3.05       0.45
996                  6.0                  29.0   0.9924  3.29       0.63
997                 33.0                 152.0   0.9966  3.31       0.47
998                 43.0                 184.0   0.9997  3.05       0.50
999                 47.0                 165.0   0.9936  3.05       0.54

     alcohol  quality
0        8.8        6
1        9.5        6
2       10.1        6
3        9.9        6
4        9.9        6
..       ...      ...
995     10.5        6
996     10.4        5
997      9.5        6
998      9.2        6
999     10.1        7

[1000 rows x 12 columns]
```

```
redwine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1000 non-null   float64
 1   volatile acidity      1000 non-null   float64
 2   citric acid           1000 non-null   float64
 3   residual sugar        1000 non-null   float64
 4   chlorides             1000 non-null   float64
 5   free sulfur dioxide   1000 non-null   float64
 6   total sulfur dioxide  1000 non-null   float64
 7   density               1000 non-null   float64
 8   pH                    1000 non-null   float64
 9   sulphates             1000 non-null   float64
 10  alcohol               1000 non-null   float64
 11  quality               1000 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 93.9 KB
```

```
whitewine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1000 non-null   float64
 1   volatile acidity      1000 non-null   float64
 2   citric acid           1000 non-null   float64
 3   residual sugar        1000 non-null   float64
 4   chlorides             1000 non-null   float64
 5   free sulfur dioxide   1000 non-null   float64
 6   total sulfur dioxide  1000 non-null   float64
 7   density               1000 non-null   float64
 8   pH                    1000 non-null   float64
 9   sulphates             1000 non-null   float64
```

```
 10  alcohol               1000 non-null   float64
 11  quality               1000 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 93.9 KB
```

```
redwine.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

Next steps: View recommended plots

```
whitewine.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

Next steps: View recommended plots

```
redwine.tail()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
995	7.7	0.60	0.06	2.00	0.079	19.0	41.0	0.99697	3.39	0.62	10.1	6
996	5.6	0.66	0.00	2.20	0.087	3.0	11.0	0.99378	3.71	0.63	12.8	7
997	5.6	0.66	0.00	2.20	0.087	3.0	11.0	0.99378	3.71	0.63	12.8	7
998	8.9	0.84	0.34	1.40	0.050	4.0	10.0	0.99554	3.12	0.48	9.1	6

```
whitewine.tail()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
995	7.8	0.27	0.34	1.60	0.046	27.0	154.0	0.9927	3.05	0.45	10.5	6
996	6.0	0.26	0.34	1.30	0.046	6.0	29.0	0.9924	3.29	0.63	10.4	5
997	6.1	0.24	0.27	9.80	0.062	33.0	152.0	0.9966	3.31	0.47	9.5	6
998	8.0	0.24	0.30	17.45	0.056	43.0	184.0	0.9997	3.05	0.50	9.2	6

```
redwine.dtypes
```

```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

```
whitewine.dtypes
```

```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

```
redwine.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alc
count	1000.000000	1000.00000	1000.000000	1000.00000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1000.00
mean	8.728900	0.52829	0.294580	2.57940	0.090375	15.171000	48.328000	0.997349	3.299100	0.66852	10.24
std	1.836602	0.17855	0.200153	1.23896	0.049917	9.972949	33.309788	0.001778	0.157948	0.18321	1.03
min	4.600000	0.12000	0.000000	1.20000	0.012000	1.000000	6.000000	0.990640	2.740000	0.33000	8.40
25%	7.400000	0.40000	0.120000	2.00000	0.072000	7.000000	23.000000	0.996372	3.190000	0.56000	9.50
50%	8.300000	0.52000	0.280000	2.30000	0.081000	13.000000	39.000000	0.997300	3.300000	0.62000	9.90
75%	9.800000	0.63500	0.470000	2.70000	0.093000	20.250000	64.250000	0.998400	3.400000	0.74000	10.80

```
whitewine.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	6.848900	0.283355	0.338150	6.545850	0.047133	36.114000	145.130000	0.994497	3.210210	0.495410
std	0.767951	0.098114	0.130787	5.164771	0.023811	16.914494	44.713912	0.002746	0.149255	0.110882
min	4.800000	0.080000	0.000000	0.800000	0.017000	3.000000	19.000000	0.988600	2.850000	0.270000
25%	6.300000	0.220000	0.270000	1.700000	0.037000	24.000000	113.000000	0.992400	3.100000	0.417500
50%	6.800000	0.270000	0.330000	5.150000	0.044000	35.000000	145.000000	0.994100	3.200000	0.480000
75%	7.300000	0.330000	0.400000	10.700000	0.050000	47.000000	174.250000	0.997000	3.320000	0.550000

Creating own model and Evaluating the accuracy of the model

Define features and target variable

```
X = wine.drop('quality', axis=1)
y = wine['quality']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(X.head())
print(y.head())
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol
0      9.4
1      9.8
2      9.8
3      9.8
4      9.4
0    5
1    5
2    5
3    6
4    5
Name: quality, dtype: int64
```

Train the model

```
model = Sequential([
    Flatten(input_shape=(X_train_scaled.shape[1],)),
    Dense(512, activation='relu'),
    Dense(256, activation='relu'),
    Dense(1, activation='linear')
])

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_3 (Flatten)         (None, 11)                0
```

```
 dense_9 (Dense)             (None, 512)               6144

 dense_10 (Dense)            (None, 256)               131328

 dense_11 (Dense)            (None, 1)                 257

=================================================================
Total params: 137729 (538.00 KB)
Trainable params: 137729 (538.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(optimizer='adam',
              loss='mean_squared_error',
              metrics=['mae', 'mse'])
```

Fit the model

```
history = model.fit(X_train_scaled, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
Epoch 1/10
130/130 [==============================] - 3s 20ms/step - loss: 0.4492 - mae: 0.5208 - mse: 0.4492 - val_loss: 0.4904 - val_mae: 0.5387
Epoch 2/10
130/130 [==============================] - 1s 9ms/step - loss: 0.4271 - mae: 0.5099 - mse: 0.4271 - val_loss: 0.4860 - val_mae: 0.5385 -
Epoch 3/10
130/130 [==============================] - 1s 11ms/step - loss: 0.4335 - mae: 0.5118 - mse: 0.4335 - val_loss: 0.4383 - val_mae: 0.5059
Epoch 4/10
130/130 [==============================] - 1s 11ms/step - loss: 0.4344 - mae: 0.5064 - mse: 0.4344 - val_loss: 0.4972 - val_mae: 0.5319
Epoch 5/10
130/130 [==============================] - 2s 13ms/step - loss: 0.4318 - mae: 0.5119 - mse: 0.4318 - val_loss: 0.4428 - val_mae: 0.5078
Epoch 6/10
130/130 [==============================] - 1s 11ms/step - loss: 0.4195 - mae: 0.5055 - mse: 0.4195 - val_loss: 0.4869 - val_mae: 0.5246
Epoch 7/10
130/130 [==============================] - 2s 12ms/step - loss: 0.4368 - mae: 0.5116 - mse: 0.4368 - val_loss: 0.4894 - val_mae: 0.5358
Epoch 8/10
130/130 [==============================] - 2s 13ms/step - loss: 0.4295 - mae: 0.5079 - mse: 0.4295 - val_loss: 0.4875 - val_mae: 0.5441
Epoch 9/10
130/130 [==============================] - 3s 20ms/step - loss: 0.4128 - mae: 0.4968 - mse: 0.4128 - val_loss: 0.4694 - val_mae: 0.5230
Epoch 10/10
130/130 [==============================] - 1s 11ms/step - loss: 0.4250 - mae: 0.5053 - mse: 0.4250 - val_loss: 0.5165 - val_mae: 0.5520
```

```
results = model.evaluate(X_test_scaled, y_test, verbose=1)
print(f'Test Loss: {test_loss:.4f}, Mean Absolute Error: {mae:.4f}, Mean Squared Error: {mse:.4f}')
print('Test Accuracy:', results[1])
```

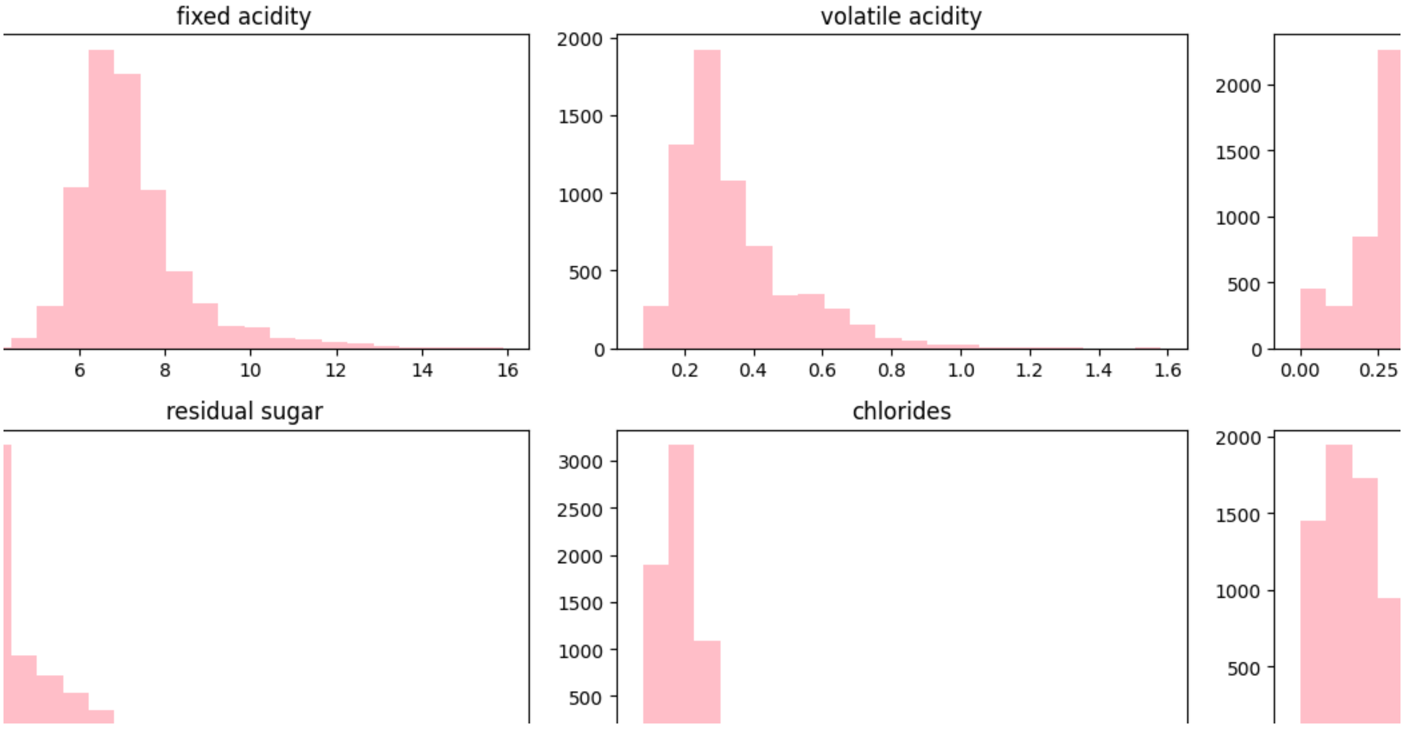
```
41/41 [==============================] - 0s 3ms/step - loss: 0.4978 - mae: 0.5536 - mse: 0.4978
Test Loss: 0.4978, Mean Absolute Error: 0.5536, Mean Squared Error: 0.4978
Test Accuracy: 0.5535941123962402
```

Plotting histograms for each feature in the wine dataset

```
fig, axs = plt.subplots(4, 3, figsize=(15, 12))

for i, feature in enumerate(wine.columns[:-1]):
    row = i // 3
    col = i % 3
    axs[row, col].hist(wine[feature], bins=20, color='pink')
    axs[row, col].set_title(feature)

plt.tight_layout()
plt.show()
```



Plotting the training history

```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```