Data Preprocessing

Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Import Dataset

In [2]:
```
train_data = pd.read_csv(r'D:\B. Tech\6th Semester\Tools and Techniques Laboratory\Minor project\Training.csv')
test_data = pd.read_csv(r'D:\B. Tech\6th Semester\Tools and Techniques Laboratory\Minor project\Testing.csv')
```

In [3]:
```
train_data.head()
```

Out[3]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	...	scurring	skin_p
0	1	1	1	0	0	0	0	0	0	0	...	0	
1	0	1	1	0	0	0	0	0	0	0	...	0	
2	1	0	1	0	0	0	0	0	0	0	...	0	
3	1	1	0	0	0	0	0	0	0	0	...	0	
4	1	1	1	0	0	0	0	0	0	0	...	0	

5 rows × 134 columns

In [4]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4920 entries, 0 to 4919
Columns: 134 entries, itching to Unnamed: 133
dtypes: float64(1), int64(132), object(1)
memory usage: 5.0+ MB
```

Data has 134 columns. The 132 of the columns are symptoms, encoded integer data, and the "prognosis" column is categorical data for disease labels.

In [5]:
```
train_data.isnull().any()
```

Out[5]:
```
itching                False
skin_rash              False
nodal_skin_eruptions   False
continuous_sneezing    False
shivering              False
                       ...
blister                False
red_sore_around_nose   False
yellow_crust_ooze      False
prognosis              False
Unnamed: 133            True
Length: 134, dtype: bool
```

In [6]:
```
train_data.drop(["Unnamed: 133"], axis = 1, inplace = True)
```

In [7]:
```
train_data
```

Out[7]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	...	blackheads
0	1	1	1	0	0	0	0	0	0	0	...	0
1	0	1	1	0	0	0	0	0	0	0	...	0
2	1	0	1	0	0	0	0	0	0	0	...	0
3	1	1	0	0	0	0	0	0	0	0	...	0
4	1	1	1	0	0	0	0	0	0	0	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
4915	0	0	0	0	0	0	0	0	0	0	...	0
4916	0	1	0	0	0	0	0	0	0	0	...	1
4917	0	0	0	0	0	0	0	0	0	0	...	0
4918	0	1	0	0	0	0	1	0	0	0	...	0
4919	0	1	0	0	0	0	0	0	0	0	...	0

4920 rows × 133 columns

Encoding Categorical Data

The data is already encoded.

Create Test and Train Data

In [8]:
```
X_train = train_data.iloc[:, :-1]
X_test = test_data.iloc[:, :-1]
y_train = train_data.iloc[:, -1:]
y_test = test_data.iloc[:, -1:]
```

In [9]:
```
print("X train shape: ",X_train.shape)
print("y train shape: ",y_train.shape)
```

```
X train shape:  (4920, 132)
y train shape:  (4920, 1)
```

- Our x train data has 132 features and 4920 observation unit that means our input matrix has 4920 rows and 132 columns for training
- Our y train data has one feature (itself) and 4920 observation unit that means our output matrix has 4920 rows and 1 column for training

In [10]:
```
print("X test shape: ",X_test.shape)
print("y test shape: ",y_test.shape)
```

```
X test shape:  (42, 132)
y test shape:  (42, 1)
```

- Our x test data has 132 features and 42 observation unit that means our input matrix has 42 rows and 132 columns for prediction
- Our y test data has one feature (itself) and 42 observation unit that means our output matrix has 42 rows and 1 column for prediction

Feature Scaling for Numerical Data

In [11]:
```
# Example of standardization and normalization

x = np.array([1,23,5,564,56,876,7,-123])

standardized_X = (x - np.mean(x)) / np.std(x)
normalized_X = (x-np.min(x) / np.max(x) - np.min(x))
print("Standardized array: ",standardized_X)
print("Normalized array: ",normalized_X )
```

```
Standardized array:  [-0.53531619 -0.46806733 -0.52308913  1.18564322 -0.36719405  2.13935429
 -0.5169756  -0.91435521]
Normalized array:  [1.24140411e+02 1.46140411e+02 1.28140411e+02 6.87140411e+02
 1.79140411e+02 9.99140411e+02 1.30140411e+02 1.40410959e-01]
```

In [12]:
```
# With sklearn scalers
from sklearn.preprocessing import StandardScaler
stds = StandardScaler()
```

```
x = x.reshape(-1,1)  # for sklearn methods. they use two dimensional vectors
x = stds.fit_transform(x)
```

What if we apply standard scaling to our train symptoms data (X_train)?

- This is a nonsensical situation because our symptoms are not numerical. They are categorical. So do not these bullshit =)

Model Building

Implementing Basic Example ANN Structure with OOP

Reference: https://www.geeksforgeeks.org/implementing-ann-training-process-in-python/

In [13]:
```
class NeuralNet(object):
    def __init__(self):
        # Generate random numbers
        np.random.seed(1)

        # Assign random weights to a 3 * 1 matrix
        self.synaptic_weights = 2 * np.random.random((3, 1)) - 1

    # The sigmoid function method
    def _sigmoid(self,x):
        return 1 / (1 + np.exp(-x))

    # Derivative sigmoid
    def derivative_sigmoid(self, x):
        return x * (1 - x)

    # Train the neural network and adjust the weights each time
    def train(self, inputs, outputs, iteration_number):
        for iteration in range(iteration_number):

            # Pass the training set through network
            output = self.learn(inputs)

            # Calculate the error
            error = outputs - output

            # Adjust the weights by a factor
            factor = np.dot(inputs.T, error * self.derivative_sigmoid(output))
            self.synaptic_weights += factor

    # calculate z
    def learn(self, test_inputs):
        return self._sigmoid(np.dot(test_inputs, self.synaptic_weights))
```

Fitting Model and Prediction

In [14]:
```
# Initialize
neural_net = NeuralNet()

# The training set
inputs = np.array([[0, 1, 1], [1, 0, 0], [1, 0, 1]])
outputs = np.array([[1, 0, 1]]).T

# train the neural network
neural_net.train(inputs, outputs, 50)

test_inputs = np.array([1, 0, 1])
threshold = 0.5
if neural_net.learn(test_inputs) >= threshold:
    print("Our test example output is: 1")
else:
    print("Our test example output is: 0")
```

```
Our test example output is: 1
```

In [15]:
```
# transform into dummies for y_train (prognosis variable)
y_train_dum = pd.get_dummies(y_train)
y_train_dum
```

Out[15]:

	prognosis_(vertigo) Paroymsal Positional Vertigo	prognosis_AIDS	prognosis_Acne	prognosis_Alcoholic hepatitis	prognosis_Allergy	prognosis_Arthritis	prognosis_Bronchial Asthma	prognosis_Cervspondyl
0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...
4915	1	0	0	0	0	0	0	
4916	0	0	1	0	0	0	0	
4917	0	0	0	0	0	0	0	
4918	0	0	0	0	0	0	0	
4919	0	0	0	0	0	0	0	

4920 rows × 41 columns

Building ANN Structure with Keras Library

Import Libraries

In [16]:
```
# import tensorflow and keras
import tensorflow as tf
from tensorflow.keras.models import Sequential   # used for initialize ANN model
from tensorflow.keras import layers   # used for different layer structure
from tensorflow.keras.layers import Dense
```

Initialize the ANN Model

In [17]:
```
classifier = Sequential()
```

Adding the Layers

In [18]:
```
# adding first hidden layer with input layer. there is init parameter that represents how to initialize weights
classifier.add(Dense(64, activation = "relu", input_dim = X_train.shape[1]))
# adding second hidden layer
classifier.add(Dense(32, activation = "relu"))
# adding last layer
classifier.add(Dense(y_train_dum.shape[1], activation = "softmax"))
```

Compiling the ANN Model

In [19]:
```
classifier.compile(optimizer = "adam", loss = "categorical_crossentropy", metrics = ["accuracy"])
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                8512

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 41)                1353

=================================================================
Total params: 11,945
Trainable params: 11,945
Non-trainable params: 0
_________________________________________________________________
```

Fitting the ANN

In [20]:
```
history = classifier.fit(X_train, y_train_dum, epochs = 5, batch_size = 30)
```

```
Epoch 1/5
164/164 [==============================] - 2s 3ms/step - loss: 2.2772 - accuracy: 0.6642
Epoch 2/5
164/164 [==============================] - 1s 3ms/step - loss: 0.1532 - accuracy: 0.9998
Epoch 3/5
164/164 [==============================] - 1s 3ms/step - loss: 0.0241 - accuracy: 1.0000
Epoch 4/5
164/164 [==============================] - 1s 3ms/step - loss: 0.0105 - accuracy: 1.0000
Epoch 5/5
164/164 [==============================] - 1s 3ms/step - loss: 0.0060 - accuracy: 1.0000
```

Making Prediction

In [21]:
```
prediction = classifier.predict(X_test)
prediction
```

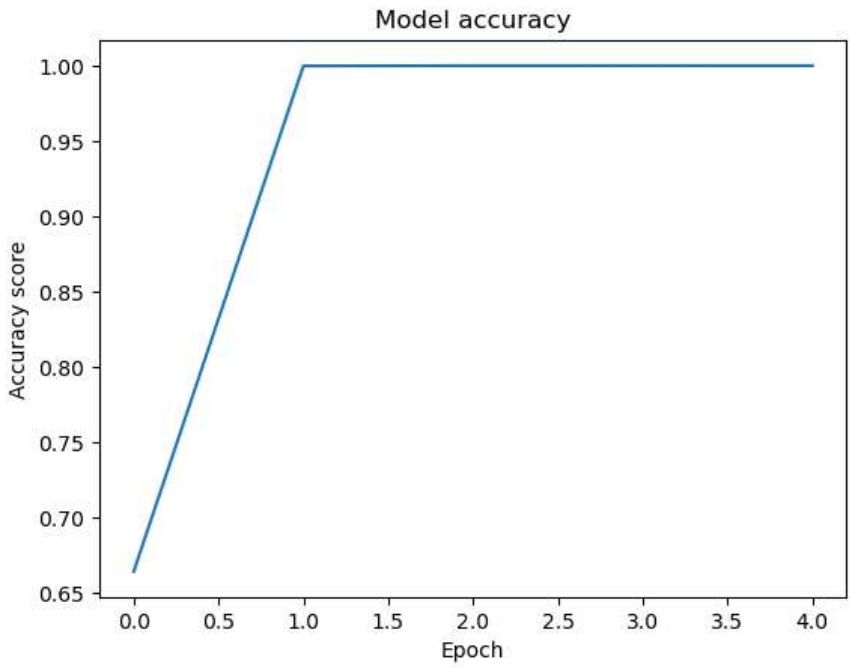
Out[21]:
```
2/2 [==============================] - 0s 0s/step
array([[8.9385016e-05, 2.8338879e-06, 1.9294847e-04, ..., 1.2613091e-07,
        1.5484959e-08, 5.7957312e-08],
       [1.1233162e-06, 2.4943677e-07, 6.8215479e-05, ..., 1.5949510e-05,
        2.5018592e-06, 1.8429466e-07],
       [1.9909254e-07, 8.3021832e-06, 7.4270616e-08, ..., 5.1075517e-07,
        5.7156009e-05, 8.5134225e-06],
       ...,
       [4.0074356e-06, 1.6048818e-05, 2.1959388e-05, ..., 1.0959538e-07,
        1.0436369e-06, 5.4690569e-05],
       [1.0398187e-07, 3.1388190e-05, 1.5587311e-06, ..., 1.5888900e-05,
        7.5858115e-05, 1.1237473e-05],
       [6.1834236e-03, 1.8855247e-03, 1.0966868e-03, ..., 3.2633690e-03,
        3.1428575e-04, 7.1838708e-04]], dtype=float32)
```

Accuracy and Loss Visualization

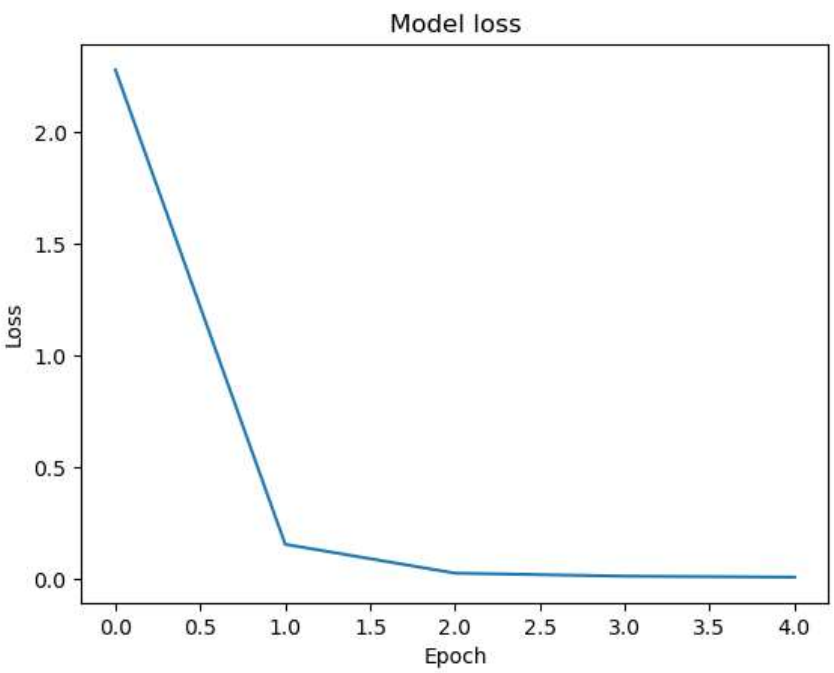
In [22]:
```
history.history["accuracy"]
```

Out[22]:
```
[0.6642276644706726, 0.9997967481613159, 1.0, 1.0, 1.0]
```

In [23]:
```
import matplotlib.pyplot as plt
plt.plot(history.history["accuracy"])
plt.title("Model accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy score")
plt.show()
```



In [24]:
```
plt.plot(history.history["loss"])
plt.title("Model loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
```



In [ ]: