Alex Johnson | Assignment 9--IMT 574 | 03.05.23

For this assignment, we are going to use the Steel Plates Faults Dataset

Dependent Variables

Attribute No.	Attribute
1	Pastry
2	Z_Scratch
3	K_Scratch
4	Stains
5	Dirtiness
6	Bumps
7	Other_Faults

Independent Variables

Attribute No.	Attribute
1	X_Minimum
2	X_Maximum
3	Y_Minimum
4	Y_Maximum
5	Pixels_Areas
6	X_Perimeter
7	Y_Perimeter
8	Sum_of_Luminosity
9	Minimum_of_Luminosity
10	Maximum_of_Luminosity
11	Length_of_Conveyer
12	TypeOfSteel_A300
13	TypeOfSteel_A400 27 SigmoidOfAreas
14	Steel_Plate_Thickness
15	Edges_Index
16	Empty_Index
17	Square_Index
18	Outside_X_Index
19	Edges_X_Index

Attribute No.	Attribute
20	Edges_Y_Index
21	Outside_Global_Index
22	LogOfAreas
23	Log_X_Index
24	Log_Y_Index
25	Orientation_Index
26	Luminosity_Index

Objectives:

1. For this exercise use a neural network and see how well you could predict the type of faults in steel plates from numeric attributes only.
2. Note: To save time and energy use the hidden layer numbers and number of nodes in hidden layers that your computer can handle.

In [1]:
```
# load libraries
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
from pandas import read_csv, set_option
from pandas.plotting import scatter_matrix
import seaborn as sns
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_s
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
2023-02-28 15:05:08.619019: I tensorflow/core/platform/cpu_feature_guard.cc:19
3] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical operati
ons:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate co
mpiler flags.
```

In [2]:
```
# load data
Path = ''

headernames = ['X_Minimum','X_Maximum','Y_Minimum','Y_Maximum','Pixels_Areas',
               'X_Perimeter','Y_Perimeter','Sum_of_Luminosity','Minimum_of_Lumi
               'Maximum_of_Luminosity','Length_of_Conveyer','TypeOfSteel_A300',
               'TypeOfSteel_A400 27', 'SigmoidOfAreas','Steel_Plate_Thickness',
               'Empty_Index','Square_Index','Outside_X_Index','Edges_X_Index','
               'Outside_Global_Index','LogOfAreas','Log_X_Index','Log_Y_Index',
               'Orientation_Index','Luminosity_Index', 'Pastry','Z_Scratch','K_
               'Stains','Dirtiness','Bumps','Other_Faults']
```

```
df1 = pd.read_csv(Path, sep = '\t',
                  header = None, delimiter = None, names = headernames)

# baseline dataframe
df = pd.DataFrame(df1)

df.head()
```

Out[2]:

	X_Minimum	X_Maximum	Y_Minimum	Y_Maximum	Pixels_Areas	X_Perimeter	Y_Perimeter
0	42	50	270900	270944	267	17	44
1	645	651	2538079	2538108	108	10	30
2	829	835	1553913	1553931	71	8	19
3	853	860	369370	369415	176	13	45
4	1289	1306	498078	498335	2409	60	260

5 rows × 34 columns

In [3]:
```
df.shape
```

Out[3]:
```
(1941, 34)
```

Baseline model

In [4]:
```
X = df.drop(['Pastry','Z_Scratch','K_Scratch','Stains',
             'Dirtiness','Bumps','Other_Faults'], axis=1)
y = df[['Pastry','Z_Scratch','K_Scratch','Stains',
        'Dirtiness','Bumps','Other_Faults']]
```

In [5]:
```
X = np.array(X)
y = np.array(y)
```

In [6]:
```
# Citing ChatGPT
model = Sequential()
model.add(Dense(10, input_dim=X.shape[1],
                activation='relu'))

model.add(Dense(7, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])

model.fit(X, y, epochs=100, batch_size=50, verbose=0)
```

```
2023-02-28 15:05:10.376218: I tensorflow/core/platform/cpu_feature_guard.cc:19
3] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical operati
ons:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate co
mpiler flags.
```

Out[6]:
```
<keras.callbacks.History at 0x7fe1f83baf70>
```

In [7]:
```
model = Sequential()
model.add(Dense(10, input_dim=X.shape[1],
                activation='relu'))

model.add(Dense(7, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])

epochs = 100
batch_size = 10
history = model.fit(X, y, epochs=epochs,
                    batch_size=batch_size, verbose=0)

# calculate average accuracy rate over n epochs
total_acc = 0.0
for i in range(epochs):
    loss, acc = model.evaluate(X, y, verbose=0)
    total_acc += acc
average_acc = total_acc / epochs

print("Average accuracy rate over {} epochs: {:.2f}%".format(
    epochs, average_acc*100))
```

```
Average accuracy rate over 100 epochs: 35.29%
```

Taking a look at the baseline sequential model's hidden layers

In [8]:
```
hidden_layer = model.layers[0]

hidden_layer
```

Out[8]:
```
<keras.layers.core.dense.Dense at 0x7fe218d2dd00>
```

In [9]:
```
weights = hidden_layer.get_weights()[0]
```

These weights in the array above shows the weights and biases of my 26 dependent variables and 1 congomerate target variable (we are predicting one binary output to indicate if ANY ONE of the 7 dependent variable features (i.e. faults) will be positive/=1)

In [10]:
```
biases = hidden_layer.get_weights()[1]
biases
```

Out[10]:
```
array([-0.10359178, -0.32543772,  0.        ,  0.81374955, -0.02031646,
       -0.09674598, -0.9818728 , -0.15660724, -1.3970426 ,  0.02831549],
      dtype=float32)
```

Baseline model summary and sample predictions

In [11]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_2 (Dense)             (None, 10)                280       
                                                                 
 dense_3 (Dense)             (None, 7)                 77        
                                                                 
=================================================================
Total params: 357
Trainable params: 357
Non-trainable params: 0
_________________________________________________________________
```

In [12]:
```
loss, accuracy = model.evaluate(X, y)
```

```
61/61 [==============================] - 0s 555us/step - loss: 3.1381 - accura
cy: 0.3529
```

In [13]:
```
predictions = model.predict(X)
```

```
61/61 [==============================] - 0s 485us/step
```

In [14]:
```
predictions.astype(int)
```

Out[14]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

In [15]:
```
print(len(predictions))
```

```
1941
```

Reflections

From the baseline model where I instantiated a Neural Netweork model via Keras in Python, it seems that for 100 epochs the average accuracy rate was approximately 35%.

I might see significant or at least moderate improvement with some hyperparameter tuning--definitely want to make this model at least or more predictive than a coin toss!

Round 2 (hyperparameter tuning the baseline model)

Model 2

For model 2 I test the model with a train test split and only test 50 epochs or iterations to reach peak accuracy

In [16]:
```
X = df.drop(['Pastry','Z_Scratch','K_Scratch','Stains',
             'Dirtiness','Bumps','Other_Faults'], axis=1)
y = df[['Pastry','Z_Scratch','K_Scratch','Stains',
        'Dirtiness','Bumps','Other_Faults']]
```

In [17]:
```
#X = X.values
#y = y.values
```

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

In [19]:
```
### adding an additional dense layer to the neural network for model 2
```

In [20]:
```
# Define the Keras Sequential model
model2 = Sequential()
model2.add(Dense(64, input_dim=27, activation='relu'))
model2.add(Dense(32, activation='relu'))
model2.add(Dense(7, activation='softmax'))
```

In [21]:
```
# Compile the model
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc
```

In [22]:
```
# Train the model
history = model2.fit(X_train, y_train,
                     validation_data=(X_test, y_test), epochs=50, batch_size=64
```

In [23]:
```
loss, accuracy = model2.evaluate(X_train, y_train)
```

```
49/49 [==============================] - 0s 544us/step - loss: 3116.3298 - acc
uracy: 0.4446
```

We see some improvement in accuracy with the addition of a second 'Dense' layer as well as a train/test split--45% accuracy for model 2

Model 3

In model 3 I add validation data to the mix in addition to train/test split

In [24]:
```
# Split the data into independent and dependent variables
X = df.drop(['Pastry','Z_Scratch','K_Scratch','Stains',
             'Dirtiness','Bumps','Other_Faults'], axis=1)
y = df[['Pastry','Z_Scratch','K_Scratch','Stains',
        'Dirtiness','Bumps','Other_Faults']]

# Split the data into training, validation, and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42)

# Define the model
model3 = Sequential()
model3.add(Dense(64, input_dim=27, activation='relu'))
model3.add(Dense(32, activation='relu'))
model3.add(Dense(7, activation='softmax'))

# Compile the model
model3.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy

# Train the model for 100 epochs
```

```
history = model3.fit(X_train, y_train, epochs=100,
                     batch_size=32, validation_data=(X_val, y_val), verbose=0)

# Evaluate the model on the test set
loss, accuracy = model3.evaluate(X_test, y_test)
print('Test accuracy:', accuracy)
```

```
13/13 [==============================] - 0s 585us/step - loss: 554.1855 - accu
racy: 0.4859
Test accuracy: 0.4858611822128296
```

With these hyperparameter tunings I have not positively impacted the accuracy of Model 3 compared to the previous models--I am at ~48% with model 3 compared to 35% from model 1 and 45% from model 2. For models 4/5 below, I will try scaling or standardizing the data to improve scores.

Model 4

For the next model I will introduce a StandardScaler() function to see if standardization helps the accuracy of the model.

In [25]:
```
# Split data into independent and dependent variables
X = df.drop(['Pastry','Z_Scratch','K_Scratch','Stains',
             'Dirtiness','Bumps','Other_Faults'], axis=1)
y = df[['Pastry','Z_Scratch','K_Scratch','Stains',
        'Dirtiness','Bumps','Other_Faults']]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

In [26]:
```
# convert to categorical data type
#y = y.astype('category')

# display the converted DataFrame
#print(y.head())
```

In [27]:
```
# Scale independent variables
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [28]:
```
# Define the model architecture
model4 = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(27,)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(7, activation='softmax')
])
```

In [29]:
```
# Compile the model
model4.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy',
                       tf.keras.metrics.Precision(),
                       tf.keras.metrics.Recall(),
                       tf.keras.metrics.AUC(),
```

```
                                tf.keras.metrics.FalsePositives(),
                                tf.keras.metrics.FalseNegatives()])
```

In [30]:
```
# Define a callback for early stopping
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', min_delta=0.001,
    patience=5, restore_best_weights=True)
```

In [31]:
```
X_val = X
y_val = y
```

In [32]:
```
y_pred2 = model4.predict(X_test_scaled)
```

```
13/13 [==============================] - 0s 654us/step
```

In [33]:
```
# Train the model for 100 epochs
history = model4.fit(X_train, y_train, epochs=100, batch_size=32,
                     validation_data=(X_val, y_val), callbacks=[early_stopping]

#history = model4.fit(X_train, y_train, epochs=100, batch_size=32,
#                      validation_data=(X_val, y_val), verbose=0)
```

In [34]:
```
# assuming y_pred contains the continuous predicted values
y_pred_discrete = (y_pred2 > 0.5).astype(int)
```

In [35]:
```
# Evaluate the model on the test set
test_loss, test_acc, test_prec, test_rec, test_auc, test_fp, test_fn = model4.e

test_f1 = f1_score(y_test, y_pred_discrete, average='weighted')

print('Test Accuracy: {:.2f}%'.format(test_acc*100))
print('Test Precision: {:.2f}%'.format(test_prec*100))
print('Test Recall: {:.2f}%'.format(test_rec*100))
print('Test F1-score: {:.2f}%'.format(test_f1*100))
print('Test AUC: {:.2f}%'.format(test_auc*100))
print('Test False Positives: {:.2f}%'.format(test_fp))
print('Test False Negatives: {:.2f}%'.format(test_fn))
```

```
13/13 - 0s - loss: 791.9764 - accuracy: 0.4242 - precision: 0.4242 - recall:
0.4242 - auc: 0.6653 - false_positives: 224.0000 - false_negatives: 224.0000 -
25ms/epoch - 2ms/step
Test Accuracy: 42.42%
Test Precision: 42.42%
Test Recall: 42.42%
Test F1-score: 2.84%
Test AUC: 66.53%
Test False Positives: 224.00%
Test False Negatives: 224.00%
```

Having implemented an early stopper function to only run a few epochs from this model, we are seeing an accuracy of about 42% for the first iterations of this neural network--for model 5 below, I remove this stopper and run another full 100 iterations.

Model 5

In [36]:
```
# Split data into independent and dependent variables
X = df.drop(['Pastry','Z_Scratch','K_Scratch','Stains',
             'Dirtiness','Bumps','Other_Faults'], axis=1)
```

```
y = df[['Pastry','Z_Scratch','K_Scratch','Stains',
        'Dirtiness','Bumps','Other_Faults']]

X = X.values
y = y.values

# convert to categorical data type
#y = y.astype('category')

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

In [37]:
```
# Scale independent variables
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [38]:
```
# Define the model
model5 = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(27,)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(7, activation='sigmoid')
])
```

In [39]:
```
# Compile the model
model5.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metri
```

In [40]:
```
# Train the model
model5.fit(X_train_scaled, y_train, epochs=100, batch_size=32, verbose=0)
```

Out[40]:
```
<keras.callbacks.History at 0x7fe218d281f0>
```

In [41]:
```
# Evaluate the model
model5.evaluate(X_test_scaled, y_test)
```

```
13/13 [==============================] - 0s 875us/step - loss: 0.1774 - accura
cy: 0.7635 - precision_1: 0.7705 - recall_1: 0.7249
```

Out[41]:
```
[0.1774272918701172,
 0.7634961605072021,
 0.7704917788505554,
 0.7249357104301453]
```

Much better! Our model 5 training shows much better accuracies--showing a accuracy of about 76% with training data!

Let's take a look at test data performance:

In [42]:
```
# Make predictions
y_pred3 = model5.predict(X_test_scaled)
```

```
13/13 [==============================] - 0s 649us/step
```

Out[43]:
```
array([[7.14624775e-06, 1.67913677e-04, 4.16993687e-04, ...,
        5.54998805e-05, 5.19833386e-01, 7.63620257e-01],
       [1.07627448e-05, 3.81157442e-04, 1.17456106e-11, ...,
        9.65836371e-05, 5.55761471e-05, 9.95756447e-01],
       [1.36539899e-02, 4.63639537e-07, 2.44229559e-05, ...,
        1.20628743e-08, 6.53136313e-01, 6.60266817e-01],
       ...,
       [5.17498856e-11, 6.59235093e-06, 9.96776760e-01, ...,
        1.29701291e-10, 3.49347159e-04, 6.84443582e-03],
       [3.07379966e-09, 7.19083437e-06, 9.99464512e-01, ...,
        8.31373165e-11, 2.36022170e-04, 4.39221971e-03],
       [4.84260619e-01, 1.29652107e-02, 5.84533787e-04, ...,
        7.46594509e-03, 6.72112219e-05, 4.43182558e-01]], dtype=float32)
```

In [44]:
```python
# train the model for 100 epochs
n_epochs = 100
acc = np.zeros(n_epochs)
for i in range(n_epochs):
    history = model5.fit(X_train_scaled, y_train, epochs=1,
                         validation_data=(X_test, y_test), verbose=0)
    acc[i] = history.history['accuracy'][0]

# compute the average accuracy across all epochs
avg_acc = np.mean(acc)
print("Average accuracy over 100 epochs:", avg_acc)
```

```
Average accuracy over 100 epochs: 0.9419136613607406
```

In this exercise, we changed hyperparameters for the neural network in several ways as well as implemented standardization procedures to improve accuracy from ~35% in model 1 to 95% with model 5 over 100 epochs (rounds) of iteration

References

OpenAI. (2021). ChatGPT. OpenAI. https://openai.com/api-docs/models/gpt-3/ (Accessed on February 27, 2023).

> Use cases: asked GPT on creating a Neural Network model using Keras in python after an example I used did not work--needed to implement additional hyperparameter tuning to Sequential() model