Case Study 6

Hollie Gardner

Assignment Instructions

Build a dense neural network to accurately detect the particle. The goal is to maximize your accuracy. Include a discussion of how you know your model has finished training as well as what design decisions you made while building the network.

In [ ]:
```
# libraries from videos
import pandas as pd
import numpy as np
import os

#model prep
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# modeling
import tensorflow as tf

# example of training a final classification model
from keras.models import Sequential
from keras.layers import Dense

#evaluation metrics
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

#data prep and viz
import seaborn as sns
import matplotlib.pyplot as plt
```

Initial EDA

In [ ]:
```
#read in the data
os.chdir('/Users/holliegardner/Desktop/qtw6/')
df = pd.read_csv('all_train.csv')
```

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

Out[ ]:
```
(7000000, 29)
```

Dataset Overview

- 7 million observations
- 1 dependent variable
- 28 features

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7000000 entries, 0 to 6999999
Data columns (total 29 columns):
 #   Column   Dtype  
---  ------   -----  
 0   # label  float64
 1   f0       float64
 2   f1       float64
 3   f2       float64
 4   f3       float64
 5   f4       float64
 6   f5       float64
 7   f6       float64
 8   f7       float64
 9   f8       float64
 10  f9       float64
 11  f10      float64
 12  f11      float64
 13  f12      float64
 14  f13      float64
 15  f14      float64
 16  f15      float64
 17  f16      float64
 18  f17      float64
 19  f18      float64
 20  f19      float64
 21  f20      float64
 22  f21      float64
 23  f22      float64
 24  f23      float64
 25  f24      float64
 26  f25      float64
 27  f26      float64
 28  mass     float64
dtypes: float64(29)
memory usage: 1.5 GB
```

All features came in as float even though the dependent variable should be integer type.

Changing in the next cell.

In [ ]:
```
#changing target from float to integer
df['# label'] = df['# label'].astype(int)
print(df['# label'])
```

```
0          1
1          1
2          0
3          1
4          0
          ..
6999995    0
6999996    0
6999997    1
6999998    1
6999999    1
Name: # label, Length: 7000000, dtype: int64
```

In [ ]:
```
# counts of each class
df['# label'].value_counts()
```

Out[ ]:
```
1    3500879
0    3499121
Name: # label, dtype: int64
```

In [ ]:
```
#separating target values from dataset
X = df.drop(['# label'], axis=1)
y = df['# label']
```

In [ ]:
```
#test train split 60/40
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=.2, random_state=143)

#standardizing data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:
```
#sequential model - sequential class
model = tf.keras.Sequential()

#use the keras api call for input layer.
model.add(tf.keras.Input(shape=(28,)))

#hidden layer
model.add(tf.keras.layers.Dense(50, activation='relu'))
model.add(tf.keras.layers.Dense(50, activation='relu'))

#add the output layer, match final layer with dimension.
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

Resources for selecting parameters

https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008

In [ ]:
```
# compiling
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.BinaryCrossentropy(),
    metrics=['accuracy']
    )
```

In [ ]:
```
#fit the model with -- data, target, # of epochs
model.fit(X_train_scaled, y_train, epochs=10, batch_size=100000)
```

```
Epoch 1/10
56/56 [==============================] - 10s 133ms/step - loss: 0.5370 - accurac
y: 0.7196
Epoch 2/10
56/56 [==============================] - 7s 131ms/step - loss: 0.3799 - accurac
y: 0.8249
Epoch 3/10
56/56 [==============================] - 7s 132ms/step - loss: 0.3470 - accurac
```

```
y: 0.8396
Epoch 4/10
56/56 [==============================] - 7s 132ms/step - loss: 0.3314 - accurac
y: 0.8463
Epoch 5/10
56/56 [==============================] - 7s 128ms/step - loss: 0.3222 - accurac
y: 0.8505
Epoch 6/10
56/56 [==============================] - 7s 133ms/step - loss: 0.3148 - accurac
y: 0.8539
Epoch 7/10
56/56 [==============================] - 7s 128ms/step - loss: 0.3081 - accurac
y: 0.8568
Epoch 8/10
56/56 [==============================] - 8s 134ms/step - loss: 0.3029 - accurac
y: 0.8590
Epoch 9/10
56/56 [==============================] - 8s 135ms/step - loss: 0.2990 - accurac
y: 0.8605
Epoch 10/10
56/56 [==============================] - 7s 128ms/step - loss: 0.2960 - accurac
y: 0.8617
```

Out[ ]:
```
<keras.callbacks.History at 0x1068dc760>
```

In [ ]:
```
#creating predictions for test holdout.
y_pred = (model.predict(X_test_scaled) >= 0.50).astype(int)
```

```
43750/43750 [==============================] - 68s 2ms/step
```

In [ ]:
```
#quick visual check of predictions
print(y_pred)
```

```
[[1]
 [0]
 [0]
 ...
 [0]
 [1]
 [1]]
```

In [ ]:
```
# evaluation metrics
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.83      0.86    700525
           1       0.84      0.89      0.87    699475

    accuracy                           0.86   1400000
   macro avg       0.86      0.86      0.86   1400000
weighted avg       0.86      0.86      0.86   1400000
```

In [ ]:
```
#Generate a visual confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)
sns.set(font_scale=1.4)
ax = sns.heatmap(cf_matrix, linewidths=.5, fmt='.5g', annot=True, cmap="coolwarm
ax.set_title('Model 1: Confusion Matrix\n\n')
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ')
```

```
ax.xaxis.set_ticklabels(['Noise','Signal'])
ax.yaxis.set_ticklabels(['Noise','Signal'])
plt.show()
```