PHYS243: Foundation of Applied Machine Learning

Final Project - Neural Networks
Prof. Bahram Mobasher, Inst. Abtin Shahidi
Submitted By:
Ray Felipe
Student ID: 862120029
Aug. 19, 2019

Executive Summary

In this final project, we'll build a neural network to classify a test data set. A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes. Thus a neural network is either a biological neural network, made up of real biological neurons, or an artificial neural network, for solving artificial intelligence (AI) problems.[1]

1.0 Build a Neural Network Model

As you saw in the lectures and notebook, neural nets are quite effective in classification if you build a reasonable network, and here you are going to build a neural network model for a very simple classification task on the data provided here. (Use any library you want but explain any method you use from them)

1.1 Load and pre-process data

Before any data analysis project can begin, we must perform data loading and preprocessing.

In [48]:
```
import numpy as np

import pandas as pd
df = pd.read_csv('dataset/train_set.txt', header=None)
print(df.shape)
df.columns=['col1', 'col2', 'col3_class']
```

```
(685, 3)
```

Build X input for analysis. The function takes in the dataframe, our train data, along with input_num. As outlined in the exercise definition, the input_num is defined as follows:

```
1. {X3, X4}
2. {X3, X5}
3. {X3, X4, X5}
4. {X1, X2, X3, X4, X5}
0. default - original attribute in the dataset, X1 and X2

Formula:
Xsub3 = Xsub1^2
Xsub4 = Xsub2^2
Xsub5 = Xsub1 * Xsub2
```

This addresses the question in Section 3.0 below - Building New Attributes.

In [49]:
```
def build_X(dataframe_for_X, input_num=0):

    df = dataframe_for_X
    X = []

    if input_num == 0:
        for i in range(len(df)):
            X_pair = []
            X_pair.append(df.col1[i])
            X_pair.append(df.col2[i])
            #X_pair.append(df.col2[i]*2) # new attributes to add
            X.append(X_pair)
        X_numpyarray = np.array([np.array(xi) for xi in X])
        X = X_numpyarray

    if input_num == 1:
        for i in range(len(df)):
            X_pair = []
            X_pair.append(df.col1[i]**2) #X3 = X1**2
            X_pair.append(df.col2[i]**2) #X4 = X2**2
            X.append(X_pair)
        X_numpyarray = np.array([np.array(xi) for xi in X])
        X = X_numpyarray

    if input_num == 2:
        for i in range(len(df)):
            X_pair = []
            X_pair.append(df.col1[i]**2) #X3 is X1**2
            X_pair.append(df.col1[i] * df.col2[i]) #X5 is X1 * X2
            X.append(X_pair)
        X_numpyarray = np.array([np.array(xi) for xi in X])
        X = X_numpyarray

    if input_num == 3:
        for i in range(len(df)):
            X_pair = []
            X_pair.append(df.col1[i]**2) #X3 is X1**2
            X_pair.append(df.col2[i]**2) #X4 is X1**2
            X_pair.append(df.col1[i] * df.col2[i])  # X5 is X1 * X2
            X.append(X_pair)
        X_numpyarray = np.array([np.array(xi) for xi in X])
        X = X_numpyarray

    if input_num == 4:
        for i in range(len(df)):
            X_pair = []
            X_pair.append(df.col1[i])
            X_pair.append(df.col2[i])
            X_pair.append(df.col1[i]**2) #X3 is X1**2
            X_pair.append(df.col2[i]**2) #X4 is X1**2
            X_pair.append(df.col1[i] * df.col2[i])  # X5 is X1 * X2
            X.append(X_pair)
        X_numpyarray = np.array([np.array(xi) for xi in X])
        X = X_numpyarray

    return X
```

The function above will build the X feature attributes based on input_num. If input_num = 1 it will build an X3 and X4 pair, if input_num is 2 it will build X3 and X5 pair, as outlined below.

1. {X3, X4}
2. {X3, X5}
3. {X3, X4, X5}
4. {X1, X2, X3, X4, X5}
5. default - original attribute in the dataset, X1 and X2

The computations for these X values are as follows:

Formula:

Xsub3 = Xsub1^2

Xsub4 = Xsub2^2

Xsub5 = Xsub1 * Xsub2

Next, we build the y attribute. This is the target class, the 3rd column in our train dataset.

In [50]:
```
def build_y(df):
    y = []
    for i in range(len(df)):
        y.append(df.col3_class[i])
    y_numpyarray = np.array([np.array(yi) for yi in y])
    y = y_numpyarray
    return y
```

1.2 Build a Neural Network Classifier

For our neural network classifier I used SKLearn's multi-layer perceptron classifier.

MLPClassifier is a multilayer perceptron (MLP). A class of feedforward artificial neural network. A MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function [3]

In [51]:
```
from sklearn.neural_network import MLPClassifier

def mlp_classifier(hidden_layer_sizes, activation='relu', solver='lbfgs'):
    clf = MLPClassifier(activation=activation, solver='lbfgs', alpha=1e-5, hidden_l
ayer_sizes=(hidden_layer_sizes), random_state=1)
    # abov: hidden_layer_sizes, first digit is hidden units, second digit is hidden
layer
    clf = clf.fit(X_for_mlp, y_for_mlp)
    return clf
```

In addition to MLP Classifier, let's use Keras NN library.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research[4].

In [52]:
```
from keras.models import Sequential
from keras.layers import Dense

# define the keras model
def keras_classifier():
    keras_model = Sequential()
    keras_model.add(Dense(12, input_dim=input_dim, activation='relu'))
    keras_model.add(Dense(8, activation='relu'))
    keras_model.add(Dense(3, activation='sigmoid'))
    #keras_model.add(Dense(1, activation='sigmoid'))
    # compile the keras model
    keras_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc
uracy'])

    # fit the keras model on the dataset
    #keras_model.fit(X_for_keras, y_for_keras, epochs=150, batch_size=10, verbose=
0)
    keras_model.fit(X_for_mlp, y_for_keras, epochs=150, batch_size=10, verbose=0)

    # make class predictions with the model
    keras_predictions = keras_model.predict_classes(X_for_mlp)
    return keras_predictions
```

Before we can use the Keras function above for prediction, the target class value in our dataset must be encoded to a numeric value.

In [53]:
```
def encode_target_class():
    pd_encoded_output = pd.get_dummies(df, columns=["col3_class"])
    encoded_output = pd_encoded_output.values[0:, 2:5] # the last encoded row, whic
h is "r". 2 is "b", 3 is "g"
    #y = pd_encoded_output.values[0:, 3] # index 4 is "r", index 3 is "g", index 2
is "b"
    y_for_keras = encoded_output
    return y_for_keras
```

In the encode_target_class function above I used the Pandas get_dummies() function[5].

The get_dummies function is a Pandas libray for converting categorical values to numeric values.

Let's try it out...

In [54]:
```
y_for_keras = encode_target_class()
y_for_keras
```

Out[54]:
```
array([[0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       ...,
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.]])
```

As we can see the col3_class values have been converted from "r", "g", "b" to a numeric binary value, with each values in having it's own column in the encoded dataset.

Let's now use these NN classifiers to predict some values using our train dataset. (next section)

2.0 Find the Simplest Neural Network

Use different activations, number of layers, number of neurons at each layer, compare their performance and find the simplest neural net. There could be couple of networks that are fairly close in terms of the performance choose anyone you think has the least complexity and explain your reasoning.

2.1 Use different activations, number or layers, number of neurons

Let's run our classifiers given different activation, layers, and neuron. To do this, we'll have to set variables that we'll use as inputs to our classifiers.

In [55]:
```
hidden_layer_sizes = [5, 2] #original hidden_layer_sizes
#hidden_layer_sizes = [10, 10, 10]

# Activations: ‘identity’, ‘logistic’, ‘tanh’, ‘relu’
mlp_activation = 'relu'
```

The hidden layers sizes are set to two (5, 2), with activation values as either identity, logistic, tanh, or relu.

Let's also build our input features, X, as well as our target feature, y. We'll build X based on input_num, which determinses our set of transformed values for X. i.e. (calculated given the formula shown in the first section of this notebook)

1. {X3, X4}
2. {X3, X5}
3. {X3, X4, X5}
4. {X1, X2, X3, X4, X5}
5. default - original attribute in the dataset, X1 and X2

In [56]:
```
input_num = 0
X_for_mlp = build_X(df, input_num)
y_for_mlp = build_y(df)
```

Now that we haver X and y, let's now run our classifier to make predictions.

2.1.1 MLP Classifier

In [57]:
```
clf = mlp_classifier(hidden_layer_sizes, activation=mlp_activation)
mlp_classifier_predictions = clf.predict(X_for_mlp)

mlp_classifier_predictions
```

Out[57]:
```
array(['r', 'r', 'r', 'r', 'r', 'g', 'r', 'r', 'r', 'g', 'r', 'r', 'r',
       'g', 'r', 'r', 'r', 'b', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'g', 'r', 'r', 'g', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'b', 'r',
       'r', 'r', 'r', 'r', 'r', 'g', 'r', 'r', 'r', 'b', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'b', 'r', 'r', 'b', 'r', 'r', 'b',
       'g', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'b', 'r', 'r', 'r', 'r', 'r', 'r', 'g', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'g',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'b', 'r', 'r', 'r', 'r', 'r',
       'g', 'r', 'b', 'r', 'r', 'r', 'b', 'g', 'b', 'r', 'r', 'r', 'r',
       'b', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'b', 'r', 'r', 'r',
       'b', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'b', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'g', 'b', 'b', 'r', 'r',
       'r', 'r', 'r', 'r', 'b', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'b',
       'g', 'r', 'r', 'r', 'r', 'b', 'r', 'r', 'r', 'r', 'r', 'r', 'g',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'g', 'r', 'b', 'r', 'g', 'r', 'b', 'b', 'r', 'r',
       'r', 'r', 'r', 'b', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'g',
       'r', 'g', 'r', 'g', 'b', 'r', 'r', 'r', 'r', 'r', 'r', 'b', 'r',
       'r', 'r', 'r', 'r', 'r', 'b', 'r', 'g', 'r', 'r', 'g', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'b', 'g', 'r', 'r', 'r', 'r', 'b', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'b', 'b', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'g', 'r', 'r', 'r',
       'b', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'g', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'b', 'r', 'r', 'r', 'r', 'r', 'b', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'b', 'r', 'r', 'g', 'g', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'b', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'g',
       'r', 'r', 'b', 'r', 'b', 'g', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'b', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'b', 'g', 'r', 'r', 'g', 'r', 'r', 'r', 'g', 'b', 'r', 'r', 'r',
       'g', 'b', 'r', 'b', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'g', 'r', 'r', 'b', 'r', 'g', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'g', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'g', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'b', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'g', 'r', 'r', 'r', 'r', 'r', 'r', 'g', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'b', 'r', 'r', 'r',
       'r', 'r', 'r', 'b', 'g', 'g', 'r', 'g', 'r', 'r', 'r', 'r', 'r',
       'g', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'b', 'g', 'r', 'r', 'r',
       'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'b', 'r',
       'r', 'g', 'g', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r',
       'g', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'g', 'r', 'r', 'r',
       'g', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'g', 'r', 'g', 'r', 'r',
       'r', 'b', 'r', 'r', 'r', 'r', 'r', 'r', 'r'], dtype='<U1')
```

Let's also try probability predictions.

In [58]:
```
mlp_classifier_prob_predictions = clf.predict_proba(X_for_mlp)
mlp_classifier_prob_predictions
```

Out[58]:
```
array([[4.12493477e-22, 9.73897867e-21, 1.00000000e+00],
       [7.49967236e-24, 3.43153223e-19, 1.00000000e+00],
       [5.37903021e-36, 2.13934071e-08, 9.99999979e-01],
       ...,
       [3.80650711e-23, 8.09843448e-20, 1.00000000e+00],
       [4.26843967e-38, 1.57507601e-06, 9.99998425e-01],
       [1.13222809e-24, 1.84225743e-18, 1.00000000e+00]])
```

2.1.2 Keras Classifier

Now let's predictions using the same X and y values, as well as input_num value, with Keras.

But first, let's setup some values required by our Keras function.

In [12]:
```
if input_num == 0 or input_num == 1 or input_num == 2:
    input_dim = 2
if input_num == 3:
    input_dim = 3
if input_num == 4:
    input_dim = 5
```

In [13]:
```
y_for_keras = encode_target_class()
```

Depending on the input_num value, we set the number of input dimensions for creating our Keras model. input_dim is the number of values in the array given X1 and X2 values, the scaled or transformed values.

In [14]:
```
keras_predictions = keras_classifier()
keras_predictions
```

```
WARNING: Logging before flag parsing goes to stderr.
W0819 15:46:55.633548 10272 deprecation_wrapper.py:119] From C:\Users\ramon\Anac
onda3\lib\site-packages\keras\backend\tensorflow_backend.py:74: The name tf.get_
default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

W0819 15:46:55.689508 10272 deprecation_wrapper.py:119] From C:\Users\ramon\Anac
onda3\lib\site-packages\keras\backend\tensorflow_backend.py:517: The name tf.pla
ceholder is deprecated. Please use tf.compat.v1.placeholder instead.

W0819 15:46:55.713859 10272 deprecation_wrapper.py:119] From C:\Users\ramon\Anac
onda3\lib\site-packages\keras\backend\tensorflow_backend.py:4138: The name tf.ra
ndom_uniform is deprecated. Please use tf.random.uniform instead.

W0819 15:46:55.787861 10272 deprecation_wrapper.py:119] From C:\Users\ramon\Anac
onda3\lib\site-packages\keras\optimizers.py:790: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.

W0819 15:46:55.803481 10272 deprecation_wrapper.py:119] From C:\Users\ramon\Anac
onda3\lib\site-packages\keras\backend\tensorflow_backend.py:3376: The name tf.lo
g is deprecated. Please use tf.math.log instead.

W0819 15:46:55.819102 10272 deprecation.py:323] From C:\Users\ramon\Anaconda3\li
b\site-packages\tensorflow\python\ops\nn_impl.py:180: add_dispatch_support.<loca
ls>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be rem
oved in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
W0819 15:46:55.988335 10272 deprecation_wrapper.py:119] From C:\Users\ramon\Anac
onda3\lib\site-packages\keras\backend\tensorflow_backend.py:986: The name tf.ass
ign_add is deprecated. Please use tf.compat.v1.assign_add instead.
```

Out[14]:
```
array([2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 0, 2, 2, 0, 2, 2, 0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 1, 2,
       0, 2, 2, 2, 0, 1, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2,
       2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 1, 0, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 2, 2, 2, 2, 0, 2, 2,
       2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1,
       2, 0, 2, 2, 2, 0, 0, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 1,
       2, 1, 2, 1, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 1, 2, 2, 2, 2, 2, 2, 2, 0, 1, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2,
       2, 2, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 0,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 0, 2, 0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2,
       2, 2, 2, 2, 2, 2, 0, 1, 2, 2, 1, 2, 2, 2, 1, 0, 2, 2, 2, 1, 0, 2,
       0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 0, 2, 1, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 0, 1, 1, 2, 1, 2,
       2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2,
       2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1,
       2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 0, 2, 2, 2, 2,
       2, 2, 2], dtype=int64)
```

2.2 Compare performance of classifiers using different activations, layers, neurons

To compare performance, let's create a function for measuring accuracy.

In [15]:
```
def accuracy_measure(mlp_or_keras, turn_off_printouts="no"):
    correct_ctr = 0
    accuracy_pct = 0
    if mlp_or_keras == "keras":
        for i in range(len(X_for_mlp)):
            #print(str(X_for_keras[i]) + "=>" + " predicted: " + str(keras_predicti
ons[i]) + ", expected: " + str(y_for_keras[i]))
            if int(keras_predictions[i]) == 0 and int(y_for_keras[i][0]) == 1:
                correct_ctr += 1
                #print(str(X_for_keras[i]) + "=>" + " predicted: " + str(keras_pred
ictions[i]) + ", expected: " + str(
                #    y_for_keras[i]))
            if int(keras_predictions[i]) == 1 and int(y_for_keras[i][1]) == 1:
                correct_ctr += 1
                #print(str(X_for_keras[i]) + "=>" + " predicted: " + str(keras_pred
ictions[i]) + ", expected: " + str(
                #    y_for_keras[i]))
            if int(keras_predictions[i]) == 2 and int(y_for_keras[i][2]) == 1:
                correct_ctr += 1
                #print(str(X_for_keras[i]) + "=>" + " predicted: " + str(keras_pred
ictions[i]) + ", expected: " + str(
                #    y_for_keras[i]))

        if turn_off_printouts == "no":
            print("Keras - accuracy percentage:")
            print("correct predictions: " + str(correct_ctr))
            print("Number of rows/instances:" + str(len(X_for_mlp)))
            print(correct_ctr/len(X_for_mlp))

    if mlp_or_keras == "mlp":
        for i in range(len(X_for_mlp)):
            #print(str(X_for_mlp[i]) + "=>" + " predicted: " + str(mlp_classifier_p
redictions[i]) +
                # ", expected: " + str(y_for_mlp[i]))
            if mlp_classifier_predictions[i] == y_for_mlp[i]:
                correct_ctr += 1

        if turn_off_printouts == "no":
            print("MLP - accuracy percentage:")
            print("correct_ctr: " + str(correct_ctr))
            print("len(X_for_mlp):" + str(len(X_for_mlp)))
            print(correct_ctr/len(X_for_mlp))

    accuracy_pct = correct_ctr/len(X_for_mlp)
    return accuracy_pct
```

Before we can call the accuracy measure function above, let's create different activations, layers, and neuron values for our classifiers. This means we'll have to rebuild our X and y attributes as well.

In the following performance test section, I created random values and inputs to test the outcome of each classification's accuracy.

2.2.1 Performance Test 1

In [16]:
```
hidden_layer_sizes = [5, 2]

# Activations: 'identity', 'logistic', 'tanh', 'relu'
mlp_activation = 'relu'

input_num = 0
X_for_mlp = build_X(df, input_num)
y_for_mlp = build_y(df)
```

Let's now rebuild our MLP Classifier

In [17]:
```
clf = mlp_classifier(hidden_layer_sizes, activation=mlp_activation)
mlp_classifier_predictions = clf.predict(X_for_mlp)
```

In [18]:
```
accuracy_measure("mlp")
```

```
MLP - accuracy percentage:
correct_ctr: 681
len(X_for_mlp):685
0.9941605839416059
```

Out[18]:
```
0.9941605839416059
```

In [19]:
```
accuracy_measure("keras")
```

```
Keras - accuracy percentage:
correct predictions: 680
Number of rows/instances:685
0.9927007299270073
```

Out[19]:
```
0.9927007299270073
```

But classifiers are at 99% accuracy given the randomly chosen inputs.

Let's try a few more tests.

2.2.2 Performance Test 2

In [25]:
```
#hidden_layer_sizes = [5, 2] #original hidden_layer_sizes
hidden_layer_sizes = [10, 10, 10]

# Activations: 'identity', 'logistic', 'tanh', 'relu'
mlp_activation = 'logistic'

input_num = 1
X_for_mlp = build_X(df, input_num)
y_for_mlp = build_y(df)
```

In [26]:
```
clf = mlp_classifier(hidden_layer_sizes, activation=mlp_activation)
mlp_classifier_predictions = clf.predict(X_for_mlp)
```

In [27]:
```
accuracy_measure("mlp")
```

```
MLP - accuracy percentage:
correct_ctr: 635
len(X_for_mlp):685
0.927007299270073
```

Out[27]:
```
0.927007299270073
```

In [28]:
```
if input_num == 0 or input_num == 1 or input_num == 2:
    input_dim = 2
if input_num == 3:
    input_dim = 3
if input_num == 4:
    input_dim = 5

y_for_keras = encode_target_class()
keras_predictions = keras_classifier()
```

In [29]:
```
accuracy_measure("keras")
```

```
Keras - accuracy percentage:
correct predictions: 654
Number of rows/instances:685
0.9547445255474453
```

Out[29]:
```
0.9547445255474453
```

2.2.3 Performance Test 3

In [30]:
```
hidden_layer_sizes = [6, 3, 2] #original hidden_layer_sizes
#hidden_layer_sizes = [10, 10, 10]

# Activations: 'identity', 'logistic', 'tanh', 'relu'
mlp_activation = 'logistic'

input_num = 3
X_for_mlp = build_X(df, input_num)
y_for_mlp = build_y(df)
```

In [31]:
```
clf = mlp_classifier(hidden_layer_sizes, activation=mlp_activation)
mlp_classifier_predictions = clf.predict(X_for_mlp)
```

In [32]:
```
accuracy_measure("mlp")
```

```
MLP - accuracy percentage:
correct_ctr: 635
len(X_for_mlp):685
0.927007299270073
```

Out[32]:
```
0.927007299270073
```

In [33]:
```
if input_num == 0 or input_num == 1 or input_num == 2:
    input_dim = 2
if input_num == 3:
    input_dim = 3
if input_num == 4:
    input_dim = 5

y_for_keras = encode_target_class()
keras_predictions = keras_classifier()
```

In [34]:
```
accuracy_measure("keras")
```

```
Keras - accuracy percentage:
correct predictions: 635
Number of rows/instances:685
0.927007299270073
```

Out[34]:
```
0.927007299270073
```

2.2.4 Performance Test 4

In [35]:
```
hidden_layer_sizes = [2, 1, 1] #original hidden_layer_sizes
#hidden_layer_sizes = [10, 10, 10]

# Activations: 'identity', 'logistic', 'tanh', 'relu'
mlp_activation = 'relu'

input_num = 0
X_for_mlp = build_X(df, input_num)
y_for_mlp = build_y(df)
```

In [36]:
```
clf = mlp_classifier(hidden_layer_sizes, activation=mlp_activation)
mlp_classifier_predictions = clf.predict(X_for_mlp)
```

In [37]:
```
accuracy_measure("mlp")
```

```
MLP - accuracy percentage:
correct_ctr: 585
len(X_for_mlp):685
0.8540145985401459
```

Out[37]:
```
0.8540145985401459
```

In [38]:
```
if input_num == 0 or input_num == 1 or input_num == 2:
    input_dim = 2
if input_num == 3:
    input_dim = 3
if input_num == 4:
    input_dim = 5

y_for_keras = encode_target_class()
print(y_for_keras)
keras_predictions = keras_classifier()
```

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

In [39]:
```
accuracy_measure("keras")
```

```
Keras - accuracy percentage:
correct predictions: 681
Number of rows/instances:685
0.9941605839416059
```

Out[39]:
```
0.9941605839416059
```

The performance tests above shows around 98% accuracy.

3.0 Build New Attributes

In this section you will create new attributes and you are going to use them instead to train the neural network. If we call the first attributes X1 and the second attributes X2, we can build new attributes.

X3 = X2/1, X4 = X2/2, X5 = X1X2

Find the simplest Neural network for the following set of inputs: (The data that you feed into the neural network.)

1. {X3;X4}
2. {X3;X5}
3. {X3;X4;X5}
4. {X1;X2;X3;X4;X5}

Our *build_X()* and *build_y()* functions above already supports re-buildling of our X and y attributes. They have been demonstrated above in the performance accuracy section.

In this section, we'll dynamically build these attributes by looping through different values, while also performing accuracy measure at runtime. This is so that we can identify the parameters that yields the highest accuracy.

3.1 Rebuild attributes using function

Now let's rebuild new attributes for our function. The attributes are based on the values X1 and X2 computations provided in the homework material, and is shown in the first section of this paper.

In [40]:
```
def rebuild_attributes_based_on_params(hidden_layer_sizes, activation, input_num):
    hidden_layer_sizes = hidden_layer_sizes
    mlp_activation = activation
    X_for_mlp = build_X(df, input_num)
    y_for_mlp = build_y(df)
```

Let's randomly pick a few input values.

In [ ]:
```
hidden_layer_sizes = [16, 4] #original hidden_layer_sizes
#hidden_layer_sizes = [10, 10, 10]

# Activations: 'identity', 'logistic', 'tanh', 'relu'
mlp_activation = 'relu'
input_num = 4
```

Let's run our attribute build function.

In [59]:
```
rebuild_attributes_based_on_params(hidden_layer_sizes, mlp_activation, input_num)
```

Once the attribute have been rebuilt, we have to rerun them in our classifer. See below.

In [41]:
```
clf = mlp_classifier(hidden_layer_sizes, activation=mlp_activation)
mlp_classifier_predictions = clf.predict(X_for_mlp)
```

We now have an updated MLP Classifier prediction that is based on the rebuilt or scaled attributes. Let's use them below to find the best model for different input values.

3.1 Find best model for different input values

In [46]:
```
#hidden_layer_sizes = [10, 10, 10]

# Activations: 'identity', 'logistic', 'tanh', 'relu'
mlp_activation = ['relu', 'logistic', 'tanh', 'identity']
input_num = 0

for i in range(len(mlp_activation)): # Activation loop
    print("MLP Activation: " + mlp_activation[i])
    index_size = 10
    acc_pct_sum = []
    for j in range(index_size):
        hidden_layer_sizes = [j+3, j+2, j+1]
        rebuild_attributes_based_on_params(hidden_layer_sizes, mlp_activation[i], i
nput_num)
        clf = mlp_classifier(hidden_layer_sizes, activation=mlp_activation[i])
        mlp_classifier_predictions = clf.predict(X_for_mlp)
        acc_pct = accuracy_measure("mlp", turn_off_printouts="yes")
        print(acc_pct)
        acc_pct_sum.append(acc_pct)
    print("Avg: " + str(sum(acc_pct_sum)/index_size))
    print()
```

```
MLP Activation: relu
0.8540145985401459
0.9927007299270073
1.0
0.9985401459854014
1.0
0.997080291970803
1.0
0.9985401459854014
0.9985401459854014
1.0
Avg: 0.983941605839416

MLP Activation: logistic
0.8540145985401459
0.927007299270073
0.9386861313868613
0.9255474452554745
0.9635036496350365
0.9343065693430657
0.9401459854014599
0.9737226277372263
0.9912408759124087
0.9708029197080292
Avg: 0.9418978102189781

MLP Activation: tanh
0.927007299270073
0.927007299270073
0.9313868613138686
0.9532846715328467
0.964963503649635
0.981021897810219
0.962043795620438
0.9401459854014599
0.981021897810219
0.9635036496350365
Avg: 0.953138686131387

MLP Activation: identity
0.8540145985401459
0.8540145985401459
0.8540145985401459
0.8540145985401459
0.8540145985401459
0.8540145985401459
0.8540145985401459
0.8540145985401459
0.8540145985401459
0.8540145985401459
Avg: 0.8540145985401459
```

Given the outcome of the sample test above, we can see that the best classifier, given a set of attributes, is with activation 'relu', and with three neuron layers, *j+3, j+2, j+1*

REFERENCES

[1] https://en.wikipedia.org/wiki/Neural_network (https://en.wikipedia.org/wiki/Neural_network)

[2] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)

[3] https://en.wikipedia.org/wiki/Multilayer_perceptron (https://en.wikipedia.org/wiki/Multilayer_perceptron)

[4] https://keras.io/ (https://keras.io/)

[5] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html)

[6] #https://www.python-course.eu/neural_networks_with_scikit.php (https://www.python-course.eu/neural_networks_with_scikit.php)

[7] https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/ (https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/)

```
In [ ]:
```