Data Models Problem Set I

Jacob Miller, Jacob Shiohira, Reid Gahan

Spring 2018, RAIK370H

All code for this problem set is in Python. The approach for this problem set was to try and divide up the code by creating a function per question. To reduce any confusion throughout the rest of this report, here are a list of imports used for all of the included code:

```
import pandas as panda
import matplotlib.pyplot as plot
import numpy as numpy
import math as math

from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_squared_error, confusion_matrix
```

The code corresponding to each question (if applicable) will be included before the question itself. Additionally, output of the functions that serve as answers to the questions will be included where applicable.

```
def describe_data_frame(dataframe):
    rows, columns = dataframe.shape
    print("num rows (observations): " + str(rows))
    print("num cols (features): " + str(columns))

    print(dataframe.info())
    print(dataframe.describe())
```

1. There are 200 observations in the data set with 4 features. The data types, as classified by Python, for the features are as follows, *fico*, *income*, and *loan* are non-null int64 where as *ratio* is non-null float64. The total memory usage from the data set is 6.3 KB.

Below is a table outlining the satistics for each feature.

	fico	income	ratio	loan
mean	577.96	150828.265	0.52945	0.5
median	575.5	147846.0	0.56	0.5
min	300.0	28700.0	0.0	0.0
max	841.0	271067.0	1.0	1.0

Prepare and Clean the Data

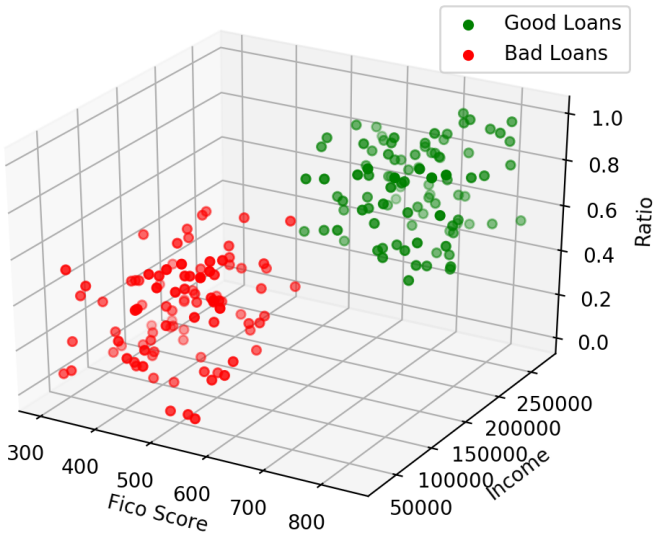
2. There are no NA's in the data. In this case, there are only a few 0 entries in the data set. However, those are totally valid entries because any of the features have a lower bound of 0, but that is in the range of possibilities for each of the features. Any NA's would be classified as missing entries in one of the rows. Thus, we did not have to remove any rows from the data set.

Explore the Data

```
def visualize_data_3d(dataframe, weights=None, plot_flag=False):
    if not plot_flag:
        return

    print(">> Constructing 3D plot of data set values\n")
    fig = plot.figure()
    ax = fig.add_subplot(111, projection='3d')

    good_loans = dataframe.loc[dataframe['loan'] == 1]
    good_x = good_loans['fico']
    good_y = good_loans['income']
    good_z = good_loans['ratio']

    ax.scatter(good_x,good_y,good_z, color='green', label='Good Loans')

    bad_loans = dataframe.loc[dataframe['loan'] == 0]
    bad_x = bad_loans['fico']
    bad_y = bad_loans['income']
    bad_z = bad_loans['ratio']
    ax.scatter(bad_x,bad_y,bad_z, color='red', label='Bad Loans')

    ax.set_xlabel('Fico Score')
    ax.set_ylabel('Income')
    ax.set_zlabel('Ratio')

    if weights is not None:
        xx, yy = numpy.meshgrid(range(2), range(2))

        if verbose:
            print("\nx: " + str(xx))
            print("\ny: " + str(yy))

        my_col = cm.jet(numpy.random.rand(rows,columns))
        x_comp = ((weights[1]/weights[2]) * xx)
        y_comp = ((weights[2]/weights[2]) * yy)
```

```
        bias = (weights[0]/weights[2])
        z = - x_comp - y_comp - bias

        surface = fig.gca(projection='3d')
        surface.plot_surface(xx, yy, z, facecolors=my_col)

    ax.legend()
    plot.show()
```

3. Included below is the resulting plot from the *matplotlib* plot output from the function above. One of the parameters is a dataframe. The function aggregates good and bad loans and then plots them on the 3−D plot.



Below is a table outlining the correclation between features for the unscaled data set.

	fico	income	ratio	loan
fico	1.000000	0.629897	0.755763	0.884133
income	0.629897	1.000000	0.617360	0.757450
ratio	0.755763	0.617360	1.000000	0.815476
loan	0.884133	0.757450	0.815476	1.000000

Because this data set was auto-generated, it does not have good data in it. That is why the correlations are so high. In a real data set, correlation would be much lower.

Prepare the Data for Modeling

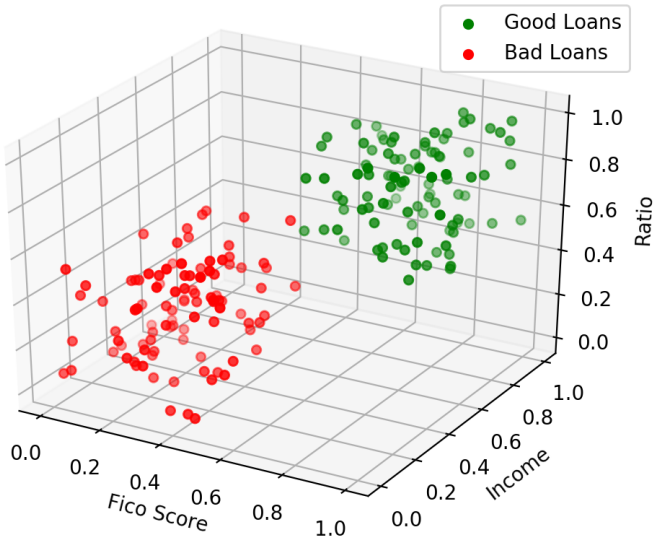
```
def normalize_and_visualize_data(dataframe, plot_flag, file):
    scaler = MinMaxScaler()
```

```
    scaled_data_array = scaler.fit_transform(dataframe)
    df_norm = panda.DataFrame(scaled_data_array, index=dataframe.index, columns=datafram
    describe_data_frame(df_norm, file)
    visualize_data_3d(dataframe, plot_flag)

    return df_norm
```

4. Below is a table outlining the statistics for each feature in the scaled data set.

	fico	income	ratio	loan
mean	0.513789	0.503898	0.52945	0.5
median	0.509242	0.491593	0.56	0.5
min	0.0	0.0	0.0	0.0
max	1.0	1.0	1.0	1.0

Included below is the resulting plot from the scaled data.



Build the Model

```
def build_model(X, Y, eta=0.01, epochs=100, error_threshold=.001):
    rows, columns = X.shape
    weights = numpy.reshape(numpy.zeros(columns), (columns, 1))
    y_hat = numpy.reshape(numpy.zeros(rows), (rows, 1))

    for iteration in range(epochs):
        delta_weights = numpy.reshape(numpy.zeros(columns), (4, 1))
```

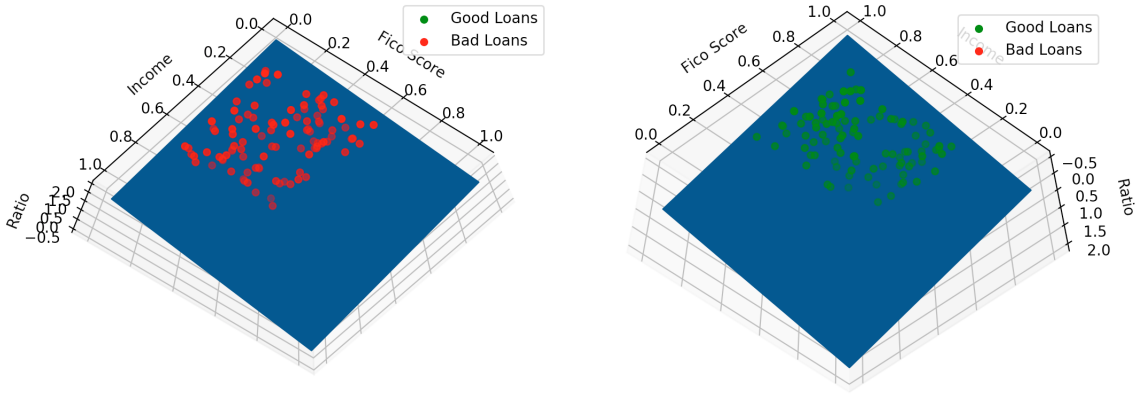
```
        for i, x in enumerate(X):
            y_hat[i] = simple_activation_function(numpy.dot(X[i],weights))
            error = Y[i] - y_hat[i]
            iteration_update = numpy.reshape(eta*error*X[i], (4, 1))
            delta_weights = numpy.add(delta_weights, iteration_update)

        weights = numpy.add(weights, delta_weights)
        mse = mean_squared_error(Y, y_hat)

        if mse < error_threshold:
            break
    return weights
```

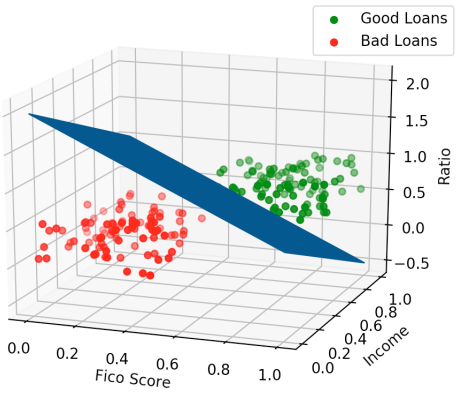
5. The perceptron model converged to a mean squared error less than 0.001 after 8 epochs with an Eta value of 0.01 and the max number of epochs set to 100. Additionally, one example of the final weights after training the model are $w_0 = -1.03$, $w_1 = 0.77179298$, $w_2 = 0.51263592$, and $w_3 = 0.6303$. The final weights were used to construct the value of z according to the equation of a plane,

$$0 = ax + by + cz + d.$$

By rearranging the equation and solving for z, we can see that if we generate values for x and y within some range, we can get a value of z for out plane,

$$z = -\frac{a}{c}x - \frac{b}{c}y - \frac{d}{c}.$$





Test and Validate the Model

```
def test_and_validate_model(X, Y, weights):
    print(">> Validating the custom model on the data set\n")
    rows, columns = X.shape
    Y_hat = numpy.reshape(numpy.zeros(rows), (rows, 1))
    num_correct = 0
    was_right = False

    for i, x in enumerate(X):
        Y_hat[i] = simple_activation_function(numpy.dot(X[i], weights))

        if Y_hat[i] == Y[i]:
            num_correct += 1
            was_right = True
        else:
            was_right = False

        if not was_right:
            print("\tGuess: " + str(Y_hat[i]))
            print("\tActual: " + str(Y[i]))

    accuracy = (num_correct/rows) * 100
    print("The model was " + str(accuracy) + "% correct.")

    print("Here is the confusion matrix generated by this model:")
    print(confusion_matrix(Y, Y_hat))
```

6. Based on the function above, the trained model's predictions for all observations matched all of the actual responses. Thus, the model was 100% correct. Below is the confusion matrix generated by the predictions from the trained model.

100	0
0	100

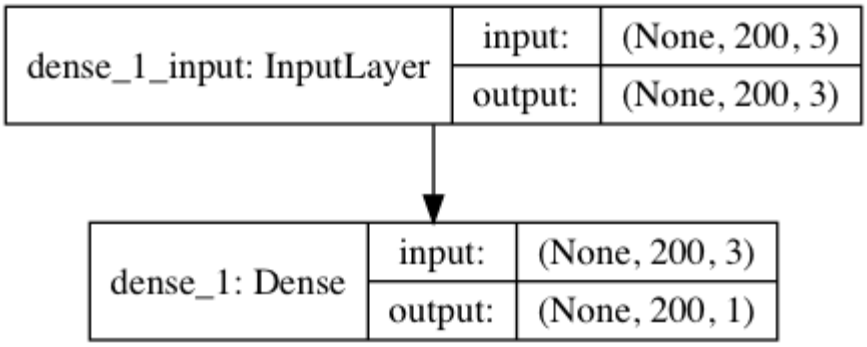
Communicate the Results

7. This simple classification model was acceptable for this data set because the good and bad loans can be separated by a line, or a plane, since the plot is in 3−D space.

8. To create an alternative model, we used Keras, a high-level neural network library written in Python. Below is the code used.

```
def built_in_model(X, Y):
    print(">> Training a Keras Sequential model\n")\

    X = X[:,[1,2,3]]
    rows, columns = X.shape

    X = numpy.reshape(X, (1, rows, columns))
    Y = numpy.reshape(Y, (1, rows, 1))
    model = Sequential()

    # Change/comment the following lines to adjust per problem needs
    model.add(Dense(2, input_shape=(rows, columns)))
    model.add(Dense(1, activation='sigmoid'))
    # model.add(Dense(1))

    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    model.fit(X, Y, epochs=1000, batch_size=45, verbose=0)
    score = model.evaluate(X, Y, batch_size=45)

    plot_model(model, to_file='model_2.png', show_shapes=True)
    weights_0, biases = model.layers[0].get_weights()
    print(">> First layer weights\n")
    print(weights_0)
    print(biases)

    print(">> Second layer weights\n")
    weights_1, biases_1 = model.layers[1].get_weights()
    print(weights_1)
    print(biases_1)

    print(model.metrics_names)
    print(score)
```

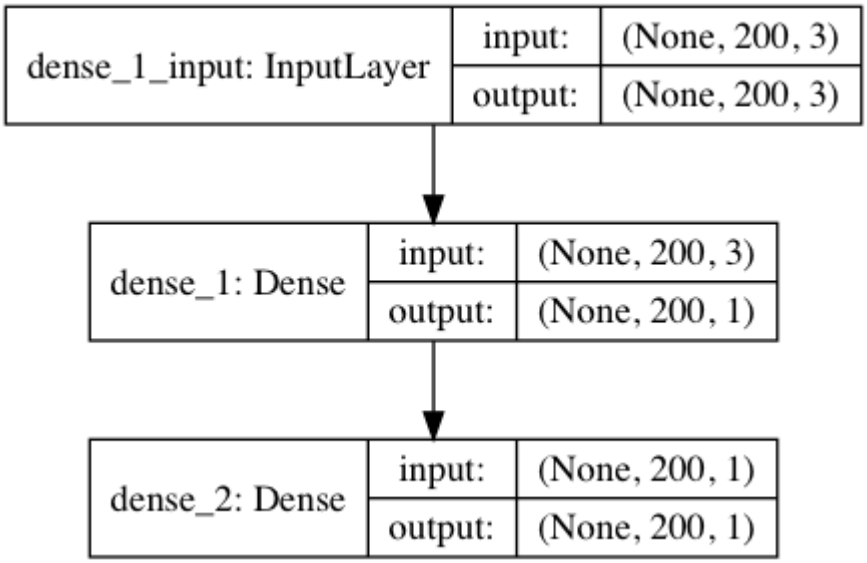
(a) Model with no hidden layers:



With no activation function, the weights were 0.5711, -1.0269, and -1.0201 with a bias of -0.2443. Using binary cross entropy as the loss function, the loss value was 8.059.

With a sigmoid activation function, the weights were 0.6856, 1.6332, and 0.2777 with a bias of -1.2105. The loss value was 0.4558.

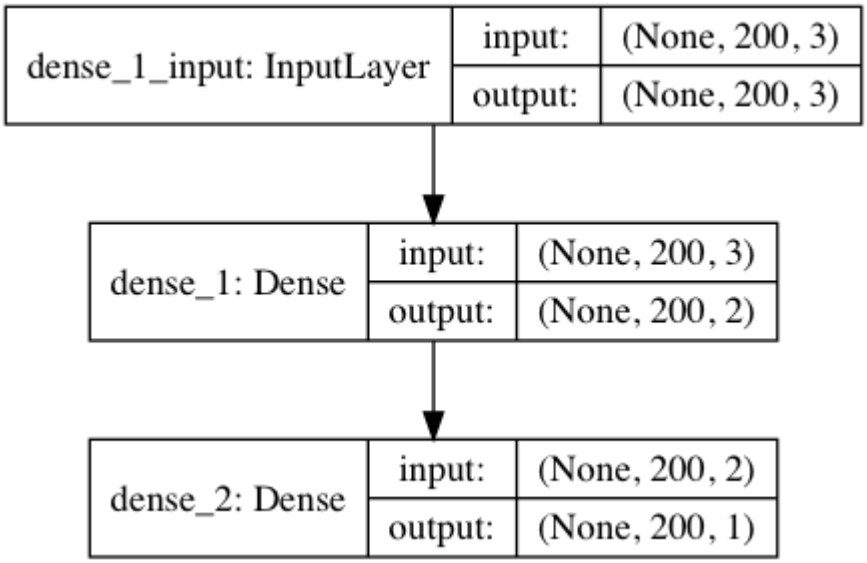
(b) Model with one hidden layer with one node:



With no activation function, the first layer had weights of 0.9444, -0.5064, and 0.3075 with a bias of 0.1428. The second layer had a weight of 0.7673 and a bias of 0.1445. The loss value was 0.7080

With a sigmoid activation function, the first layer had weights of 0.3229, -0.0861, and 0.2513 with a bias of -0.2266. The second layer had a weight of -1.0571 and a bias of 0.2210. The loss value was 0.7667.

(c) Model with one hidden layer with two nodes:



With no activation function, the first node in the first layer had weights of 0.4465, -0.5371, and 0.7391 with a bias of -0.0411. The second node in the first layer had weights of 0.2081, -1.6335, and -0.6190 with a bias of 0.3719. The second layer had weights of 1.2591 and -0.3076 with a bias of -0.0225. The loss value was 0.2250.

With a sigmoid activation function, the first node in the first layer had weights of 0.3240, 1.4410, and 0.7906 with a bias of -1.0627. The second node in the first layer had weights of -0.4445, -1.5673, and -4020 with a bias of 1.0035. The second layer had weights of 1.8803 and -2.3067 with a bias of -0.8589. The loss value was 0.1628.