```
# This is formatted as code
```

DMA Fall 21

Note : This entire lab will be manually evaluated.

Name : 'Yinglu Deng'

Collaborator : ''

Lab 4: Neural Networks

```
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.feature_extraction import DictVectorizer

from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV, ParameterGrid

import numpy as np

import warnings
warnings.filterwarnings("ignore")
```

```
!wget http://askoski.berkeley.edu/~zp/lab_4_training.csv
!wget http://askoski.berkeley.edu/~zp/lab_4_test.csv

df_train = pd.read_csv('./lab_4_training.csv')
df_test = pd.read_csv('./lab_4_test.csv')
df_train.head()
```

```
--2021-09-22 04:54:33--  http://askoski.berkeley.edu/~zp/lab_4_training.csv
Resolving askoski.berkeley.edu (askoski.berkeley.edu)... 169.229.192.179
Connecting to askoski.berkeley.edu (askoski.berkeley.edu)|169.229.192.179|:80...
HTTP request sent, awaiting response... 200 OK
Length: 79177 (77K) [text/csv]
Saving to: ‘lab_4_training.csv’

lab_4_training.csv  100%[===================>]  77.32K  57.7KB/s    in 1.3s

2021-09-22 04:54:37 (57.7 KB/s) - ‘lab_4_training.csv’ saved [79177/79177]

--2021-09-22 04:54:37--  http://askoski.berkeley.edu/~zp/lab_4_test.csv
Resolving askoski.berkeley.edu (askoski.berkeley.edu)... 169.229.192.179
Connecting to askoski.berkeley.edu (askoski.berkeley.edu)|169.229.192.179|:80...
HTTP request sent, awaiting response... 200 OK
Length: 26519 (26K) [text/csv]
Saving to: ‘lab_4_test.csv’

lab_4_test.csv      100%[===================>]  25.90K  34.8KB/s    in 0.7s

2021-09-22 04:54:38 (34.8 KB/s) - ‘lab_4_test.csv’ saved [26519/26519]
```

Unnamed: 0	gender	age	year	eyecolor	height	miles	brothers	sisters	compu

Question 1

Calculate a baseline accuracy measure using the majority class, assuming a target variable of 'gender'. The majority class is the most common value of the target variable in a particular dataset. Accuracy is calculated as (true positives + true negatives) / (all negatives and positives)

Question 1.a

Find the majority class in the training set. If you always predicted this class in the training set, what would your accuracy be?

```
# YOUR CODE HERE
train_group = df_train.groupby('gender').size()
train_group
```

```
gender
female    647
male      545
dtype: int64
```

```
label_list = list(train_group)
majority_train_accuracy = max(label_list) / np.sum(label_list)
majority_train_accuracy
```

```
0.5427852348993288
```

ANSWER: The majority class is female. And the accuracy will be 0.5427852348993288.

Question 1.b

If you always predicted this same class (majority from the training set) in the test set, what would your accuracy be?

```
# YOUR CODE HERE
test_group = df_test.groupby('gender').size()
test_group
```

```
gender
female    208
male      190
dtype: int64
```

```
label_list = list(test_group)
majority_test_accuracy = max(label_list) / np.sum(label_list)
majority_test_accuracy
```

```
0.5226130653266332
```

ANSWER: The majority class is female. And the accuracy will be 0.5226130653266332.

Question 2

Get started with Neural Networks.

Choose a NN implementation (eg: scikit-learn) and specify which you choose. Be sure the implementation allows you to modify the number of hidden layers and hidden nodes per layer.

NOTE: When possible, specify the logsig (sigmoid/logistc) function as the transfer function (another word for activation function) and use Levenberg-Marquardt backpropagation (lbfgs). It is possible to specify logistic in Sklearn MLPclassifier (Neural net).

Question 2.a

Train a neural network with a single 10 node hidden layer. Only use the Height feature of the dataset to predict the Gender. You will have to change Gender to a 0 and 1 class. After training, use your trained model to predict the class using the height feature from the training set. What was the accuracy of this prediction?

```
# YOUR CODE HERE
# use keras as a NN implementation
from keras.models import Sequential
from keras.layers import Dense
from sklearn import preprocessing

# use the height feature and transform the gender to binary
X_train = np.array(df_train[['height']])
df_train['gender_binary'] = df_train['gender'].apply(lambda x: 1 if x == 'female' else
Y_train = np.array(df_train[['gender_binary']])

clf = MLPClassifier(hidden_layer_sizes=(10), activation = 'logistic',
                    solver='lbfgs', random_state=1)  #sgd = Stochastic Gradient desce
                                                      #Default activation is
                                                      #Default n_iter_no_cha

clf.fit(X_train,Y_train)

print('Accuracy on training---')
y_pred_train=clf.predict(X_train)
print(accuracy_score(Y_train,y_pred_train))
```

```
Accuracy on training---
0.802013422818792
```

ANSWER: The accuracy of this prediction for training set is 0.802013422818792.

Question 2.b

Take the trained model from question 2.a and use it to predict the test set. This can be accomplished by taking the trained model and giving it the Height feature values from the test set. What is the accuracy of this model on the test set?

```
# YOUR CODE HERE
X_test = np.array(df_test[['height']])
df_test['gender_binary'] = df_test['gender'].apply(lambda x: 1 if x == 'female' else (
Y_test = np.array(df_test[['gender_binary']])

print('Accuracy on test---')
y_pred_test=clf.predict(X_test)
print(accuracy_score(Y_test,y_pred_test))
```

```
Accuracy on test---
0.7939698492462312
```

ANSWER: The accuracy of this prediction for test set is 0.7939698492462312.

Question 2.c

Neural Networks tend to prefer smaller, normalized feature values. Try taking the log of the height feature in both training and testing sets or use a Standard Scalar operation in SKlearn to centre and normalize the data between 0-1 for continuous values. Repeat question 2.a and 2.b with the log version or the normalized and centered version of this feature

```
# YOUR CODE HERE
from sklearn import preprocessing
# use a Standard Scalar operation in SKlearn to centre and normalize the data between
sc=preprocessing.StandardScaler()
X_train_scaled=sc.fit_transform(X_train)
X_test_scaled=sc.transform(X_test)

clf = MLPClassifier(hidden_layer_sizes=(10), activation = 'logistic',
                     solver='lbfgs', random_state=1)  #sgd = Stochastic Gradient desce
                                                       #Default activation is
                                                       #Default n_iter_no_cha

clf.fit(X_train_scaled,Y_train)

print('Accuracy on training---')
y_pred_scaled_train=clf.predict(X_train_scaled)
print(accuracy_score(Y_train, y_pred_scaled_train))

print('')
print('Accuracy on test---')
y_pred_scaled_test=clf.predict(X_test_scaled)
print(accuracy_score(Y_test, y_pred_scaled_test))
```

```
Accuracy on training---
0.8439597315436241

Accuracy on test---
0.8542713567839196
```

ANSWER: After normalizing the data, the accuracy of this prediction for training set is 0.8439597315436241 and the accuracy of this prediction for test set is 0.8542713567839196.

Question 3

The rest of features in this dataset barring a few are categorical. No ML method accepts categorical features, so transform year, eyecolor, exercise into a set of binary features, one feature per unique original feature value, and mark the binary feature as '1' if the feature value matches the original value and '0' otherwise. Using only these binary variable transformed features, train and predict the class of the test set. What was your accuracy using Neural Network with a single 10 node hidden layer? During training, use a maximum number of iterations of 50.

```
# only get three features
df_train_bin = df_train[['year', 'eyecolor', 'exercise']]
df_test_bin = df_test[['year', 'eyecolor', 'exercise']]

#get dummy variable and convert to array
X_train_dummy = np.array(pd.get_dummies(df_train_bin))
y_train = df_train['gender']
X_test_dummy = np.array(pd.get_dummies(df_test_bin))
y_test = df_test['gender']

#label coding for target
le = LabelEncoder()
le.fit(['female', 'male'])
y_train = le.transform(y_train).reshape(-1, 1)
y_test = le.transform(y_test).reshape(-1, 1)

#keras training model
model = Sequential()
model.add(Dense(units=10, activation='sigmoid'))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
model.fit(X_train_dummy, y_train, epochs=50) #iter = 50

print('Accuracy on training---')
y_pred_scaled_train=model.predict(X_train_dummy)
y_pred_scaled_train=(y_pred_scaled_train>.5)*1
print(accuracy_score(y_train, y_pred_scaled_train))

print('')
print('Accuracy on test---')
y_pred_scaled_test=model.predict(X_test_dummy)
y_pred_scaled_test=(y_pred_scaled_test>.5)*1
print(accuracy_score(y_test, y_pred_scaled_test))
```

```
Epoch 24/50
38/38 [==============================] - 0s 819us/step - loss: 0.6875 - accuracy
Epoch 25/50
38/38 [==============================] - 0s 810us/step - loss: 0.6875 - accuracy
```

```
Epoch 26/50
38/38 [==============================] - 0s 821us/step - loss: 0.6874 - accuracy
Epoch 27/50
38/38 [==============================] - 0s 846us/step - loss: 0.6874 - accuracy
Epoch 28/50
38/38 [==============================] - 0s 834us/step - loss: 0.6873 - accuracy
Epoch 29/50
38/38 [==============================] - 0s 964us/step - loss: 0.6873 - accuracy
Epoch 30/50
38/38 [==============================] - 0s 773us/step - loss: 0.6872 - accuracy

Epoch 31/50
38/38 [==============================] - 0s 805us/step - loss: 0.6872 - accuracy
Epoch 32/50
38/38 [==============================] - 0s 859us/step - loss: 0.6872 - accuracy
Epoch 33/50
38/38 [==============================] - 0s 874us/step - loss: 0.6871 - accuracy
Epoch 34/50
38/38 [==============================] - 0s 794us/step - loss: 0.6872 - accuracy
Epoch 35/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6870 - accuracy: (
Epoch 36/50
38/38 [==============================] - 0s 910us/step - loss: 0.6870 - accuracy
Epoch 37/50
38/38 [==============================] - 0s 860us/step - loss: 0.6869 - accuracy
Epoch 38/50
38/38 [==============================] - 0s 803us/step - loss: 0.6869 - accuracy
Epoch 39/50
38/38 [==============================] - 0s 807us/step - loss: 0.6868 - accuracy
Epoch 40/50
38/38 [==============================] - 0s 870us/step - loss: 0.6869 - accuracy
Epoch 41/50
38/38 [==============================] - 0s 871us/step - loss: 0.6868 - accuracy
Epoch 42/50
38/38 [==============================] - 0s 990us/step - loss: 0.6868 - accuracy
Epoch 43/50
38/38 [==============================] - 0s 809us/step - loss: 0.6867 - accuracy
Epoch 44/50
38/38 [==============================] - 0s 919us/step - loss: 0.6867 - accuracy
Epoch 45/50
38/38 [==============================] - 0s 840us/step - loss: 0.6866 - accuracy
Epoch 46/50
38/38 [==============================] - 0s 815us/step - loss: 0.6866 - accuracy
Epoch 47/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6866 - accuracy: (
Epoch 48/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6865 - accuracy: (
Epoch 49/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6866 - accuracy: (
Epoch 50/50
38/38 [==============================] - 0s 868us/step - loss: 0.6865 - accuracy
Accuracy on training---
0.5511744966442953

Accuracy on test---
0.5100502512562815
```

ANSWER: The accuracy of this prediction for training set is 0.5411073825503355 and the accuracy of this prediction for test set is 0.5402010050251256.

Question 4

Using a NN, report the accuracy on the test set of a model that trained only on the height and the eye color features of instances in the training set.

Question 4.a

What is the accuracy on the test set using the original height values (no pre-processing) and eye color as a one-hot?

```
# get the height feature
df_train_height = df_train[['height']]
df_test_height = df_test[['height']]

#get dummy variable and convert to array
X_train_dummy = pd.get_dummies(df_train[['eyecolor']])
y_train = df_train['gender']
X_test_dummy = pd.get_dummies(df_test[['eyecolor']])
y_test = df_test['gender']

X_train_q4 = X_train_dummy.join(df_train_height)
X_test_q4 = X_test_dummy.join(df_test_height)

#label coding for target
le = LabelEncoder()
le.fit(['female', 'male'])
y_train = le.transform(y_train).reshape(-1, 1)
y_test = le.transform(y_test).reshape(-1, 1)

#keras training model
model = Sequential()
model.add(Dense(units=10, activation='sigmoid'))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
model.fit(X_train_q4, y_train, epochs=50) #iter = 50

print('Accuracy on training---')
y_pred_scaled_train=model.predict(X_train_q4)
y_pred_scaled_train=(y_pred_scaled_train>.5)*1
print(accuracy_score(y_train, y_pred_scaled_train))
```

```
print('')
print('Accuracy on test---')
y_pred_scaled_test=model.predict(X_test_q4)
y_pred_scaled_test=(y_pred_scaled_test>.5)*1
print(accuracy_score(y_test, y_pred_scaled_test))
```

```
Epoch 24/50
38/38 [==============================] - 0s 900us/step - loss: 0.6870 - accuracy
Epoch 25/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6886 - accuracy:
Epoch 26/50
38/38 [==============================] - 0s 829us/step - loss: 0.6862 - accuracy
Epoch 27/50
38/38 [==============================] - 0s 805us/step - loss: 0.6867 - accuracy
Epoch 28/50
38/38 [==============================] - 0s 808us/step - loss: 0.6865 - accuracy
Epoch 29/50
38/38 [==============================] - 0s 811us/step - loss: 0.6879 - accuracy
Epoch 30/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6868 - accuracy:
Epoch 31/50
38/38 [==============================] - 0s 849us/step - loss: 0.6872 - accuracy
Epoch 32/50
38/38 [==============================] - 0s 882us/step - loss: 0.6866 - accuracy
Epoch 33/50
38/38 [==============================] - 0s 895us/step - loss: 0.6868 - accuracy
Epoch 34/50
38/38 [==============================] - 0s 883us/step - loss: 0.6874 - accuracy
Epoch 35/50
38/38 [==============================] - 0s 850us/step - loss: 0.6872 - accuracy
Epoch 36/50
38/38 [==============================] - 0s 957us/step - loss: 0.6866 - accuracy
Epoch 37/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6879 - accuracy:
Epoch 38/50
38/38 [==============================] - 0s 960us/step - loss: 0.6882 - accuracy
Epoch 39/50
38/38 [==============================] - 0s 912us/step - loss: 0.6873 - accuracy
Epoch 40/50
38/38 [==============================] - 0s 894us/step - loss: 0.6872 - accuracy
Epoch 41/50
38/38 [==============================] - 0s 950us/step - loss: 0.6867 - accuracy
Epoch 42/50
38/38 [==============================] - 0s 990us/step - loss: 0.6863 - accuracy
Epoch 43/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6881 - accuracy:
Epoch 44/50
38/38 [==============================] - 0s 843us/step - loss: 0.6868 - accuracy
Epoch 45/50
38/38 [==============================] - 0s 877us/step - loss: 0.6861 - accuracy
Epoch 46/50
38/38 [==============================] - 0s 872us/step - loss: 0.6880 - accuracy
Epoch 47/50
38/38 [==============================] - 0s 779us/step - loss: 0.6872 - accuracy
Epoch 48/50
38/38 [==============================] - 0s 856us/step - loss: 0.6873 - accuracy
Epoch 49/50
38/38 [==============================] - 0s 916us/step - loss: 0.6870 - accuracy
```

```
38/38 [==============================] - 0s 916us/step - loss: 0.6870 - accuracy
Epoch 50/50
38/38 [==============================] - 0s 844us/step - loss: 0.6870 - accuracy
Accuracy on training---
0.5427852348993288

Accuracy on test---
0.5226130653266332
```

ANSWER:

Original height values (no pre-processing) and eye color as a one-hot:

The accuracy of this prediction for training set is 0.5427852348993288 and the accuracy of this prediction for test set is 0.5226130653266332.

Question 4.b

What is the accuracy on the test set using the log of height values (applied to both training and testing sets) and eye color as a one-hot?

```
# apply log to the height
df_train_temp = df_train.copy()
df_train_temp['height'] = df_train_temp['height'].apply('log')
df_test_temp = df_test.copy()
df_test_temp['height'] = df_test_temp['height'].apply('log')

df_train_height = df_train_temp[['height']]
df_test_height = df_test_temp[['height']]

#get dummy variable and convert to array
X_train_dummy = pd.get_dummies(df_train[['eyecolor']])
y_train = df_train['gender']
X_test_dummy = pd.get_dummies(df_test[['eyecolor']])
y_test = df_test['gender']

X_train_q4b = X_train_dummy.join(df_train_height)
X_test_q4b = X_test_dummy.join(df_test_height)

#label coding for target
le = LabelEncoder()
le.fit(['female', 'male'])
y_train = le.transform(y_train).reshape(-1, 1)
y_test = le.transform(y_test).reshape(-1, 1)

#keras training model
model = Sequential()
model.add(Dense(units=10, activation='sigmoid'))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
model.fit(X_train_q4b, y_train, epochs=50) #iter = 50
```

```
model.fit(X_train_q4b, y_train, epochs=50) #iter = 50

print('Accuracy on training---')
y_pred_scaled_train=model.predict(X_train_q4b)
y_pred_scaled_train=(y_pred_scaled_train>.5)*1
print(accuracy_score(y_train, y_pred_scaled_train))

print('')
print('Accuracy on test---')
y_pred_scaled_test=model.predict(X_test_q4b)
y_pred_scaled_test=(y_pred_scaled_test>.5)*1
print(accuracy_score(y_test, y_pred_scaled_test))
```

```
Epoch 24/50
38/38 [==============================] - 0s 901us/step - loss: 0.6930 - accuracy
Epoch 25/50
38/38 [==============================] - 0s 826us/step - loss: 0.6931 - accuracy
Epoch 26/50
38/38 [==============================] - 0s 949us/step - loss: 0.6930 - accuracy
Epoch 27/50
38/38 [==============================] - 0s 971us/step - loss: 0.6930 - accuracy
Epoch 28/50
38/38 [==============================] - 0s 877us/step - loss: 0.6930 - accuracy
Epoch 29/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6930 - accuracy: (
Epoch 30/50
38/38 [==============================] - 0s 952us/step - loss: 0.6931 - accuracy
Epoch 31/50
38/38 [==============================] - 0s 919us/step - loss: 0.6929 - accuracy
Epoch 32/50
38/38 [==============================] - 0s 810us/step - loss: 0.6928 - accuracy
Epoch 33/50
38/38 [==============================] - 0s 783us/step - loss: 0.6928 - accuracy
Epoch 34/50
38/38 [==============================] - 0s 795us/step - loss: 0.6928 - accuracy
Epoch 35/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6927 - accuracy: (
Epoch 36/50
38/38 [==============================] - 0s 983us/step - loss: 0.6927 - accuracy
Epoch 37/50
38/38 [==============================] - 0s 868us/step - loss: 0.6927 - accuracy
Epoch 38/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6927 - accuracy: (
Epoch 39/50
38/38 [==============================] - 0s 928us/step - loss: 0.6926 - accuracy
Epoch 40/50
38/38 [==============================] - 0s 981us/step - loss: 0.6926 - accuracy
Epoch 41/50
38/38 [==============================] - 0s 914us/step - loss: 0.6925 - accuracy
Epoch 42/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6927 - accuracy: (
Epoch 43/50
38/38 [==============================] - 0s 845us/step - loss: 0.6926 - accuracy
Epoch 44/50
38/38 [==============================] - 0s 922us/step - loss: 0.6924 - accuracy
Epoch 45/50
38/38 [==============================] - 0s 972us/step - loss: 0.6925 - accuracy
Epoch 46/50
```

```
Epoch 46/50
38/38 [==============================] - 0s 893us/step - loss: 0.6924 - accuracy
Epoch 47/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6923 - accuracy: 
Epoch 48/50
38/38 [==============================] - 0s 959us/step - loss: 0.6924 - accuracy
Epoch 49/50
38/38 [==============================] - 0s 873us/step - loss: 0.6924 - accuracy
Epoch 50/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6923 - accuracy: 

Accuracy on training---
0.5427852348993288

Accuracy on test---
0.5226130653266332
```

ANSWER:

The log of height values and eye color as a one-hot:

The accuracy of this prediction for training set is 0.5427852348993288 and the accuracy of this prediction for test set is 0.5226130653266332.

Question 4.c

What is the accuracy on the test set using the Z-score of height values and eye color as a one-hot?

Z-score is a normalization function. It is the value of a feature minus the average value for that feature (in the training set), divided by the standard deviation of that feature (in the training set). Remember that, whenever applying a function to a feature in the training set, it also has to be applied to that same feature in the test set.

```
# z score function
h_mean = df_train['height'].mean()
h_std = df_train['height'].std()
calculate_Zscore = lambda x: (x - h_mean) / h_std

# apply z score to the height
df_train_temp = df_train.copy()
df_train_temp['height'] = df_train_temp['height'].apply(calculate_Zscore)
df_test_temp = df_test.copy()
df_test_temp['height'] = df_test_temp['height'].apply(calculate_Zscore)

df_train_height = df_train_temp[['height']]
df_test_height = df_test_temp[['height']]

#get dummy variable and convert to array
X_train_dummy = pd.get_dummies(df_train[['eyecolor']])
y_train = df_train['gender']
X_test_dummy = pd.get_dummies(df_test[['eyecolor']])
```

```
y_test = df_test['gender']

X_train_q4c = X_train_dummy.join(df_train_height)
X_test_q4c = X_test_dummy.join(df_test_height)

#label coding for target
le = LabelEncoder()
le.fit(['female', 'male'])
y_train = le.transform(y_train).reshape(-1, 1)
y_test = le.transform(y_test).reshape(-1, 1)

#keras training model
model = Sequential()
model.add(Dense(units=10, activation='sigmoid'))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
model.fit(X_train_q4c, y_train, epochs=50) #iter = 50

print('Accuracy on training---')
y_pred_scaled_train=model.predict(X_train_q4c)
y_pred_scaled_train=(y_pred_scaled_train>.5)*1
print(accuracy_score(y_train, y_pred_scaled_train))

print('')
print('Accuracy on test---')
y_pred_scaled_test=model.predict(X_test_q4c)
y_pred_scaled_test=(y_pred_scaled_test>.5)*1
print(accuracy_score(y_test, y_pred_scaled_test))
```

```
Epoch 24/50
38/38 [==============================] - 0s 864us/step - loss: 0.5211 - accuracy
Epoch 25/50
38/38 [==============================] - 0s 813us/step - loss: 0.5167 - accuracy
Epoch 26/50
38/38 [==============================] - 0s 864us/step - loss: 0.5124 - accuracy
Epoch 27/50
38/38 [==============================] - 0s 894us/step - loss: 0.5083 - accuracy
Epoch 28/50
38/38 [==============================] - 0s 816us/step - loss: 0.5042 - accuracy
Epoch 29/50
38/38 [==============================] - 0s 873us/step - loss: 0.5004 - accuracy
Epoch 30/50
38/38 [==============================] - 0s 840us/step - loss: 0.4965 - accuracy
Epoch 31/50
38/38 [==============================] - 0s 827us/step - loss: 0.4927 - accuracy
Epoch 32/50
38/38 [==============================] - 0s 977us/step - loss: 0.4889 - accuracy
Epoch 33/50
38/38 [==============================] - 0s 987us/step - loss: 0.4853 - accuracy
Epoch 34/50
38/38 [==============================] - 0s 917us/step - loss: 0.4817 - accuracy
Epoch 35/50
38/38 [==============================] - 0s 1ms/step - loss: 0.4783 - accuracy:
Epoch 36/50
38/38 [==============================] - 0s 870us/step - loss: 0.4750 - accuracy
Epoch 37/50
```

```
Epoch 37/50
38/38 [==============================] - 0s 880us/step - loss: 0.4718 - accuracy
Epoch 38/50
38/38 [==============================] - 0s 891us/step - loss: 0.4686 - accuracy
Epoch 39/50
38/38 [==============================] - 0s 821us/step - loss: 0.4655 - accuracy
Epoch 40/50
38/38 [==============================] - 0s 896us/step - loss: 0.4625 - accuracy
Epoch 41/50
38/38 [==============================] - 0s 840us/step - loss: 0.4596 - accuracy

Epoch 42/50
38/38 [==============================] - 0s 887us/step - loss: 0.4568 - accuracy
Epoch 43/50
38/38 [==============================] - 0s 786us/step - loss: 0.4541 - accuracy
Epoch 44/50
38/38 [==============================] - 0s 820us/step - loss: 0.4514 - accuracy
Epoch 45/50
38/38 [==============================] - 0s 876us/step - loss: 0.4489 - accuracy
Epoch 46/50
38/38 [==============================] - 0s 993us/step - loss: 0.4465 - accuracy
Epoch 47/50
38/38 [==============================] - 0s 865us/step - loss: 0.4441 - accuracy
Epoch 48/50
38/38 [==============================] - 0s 862us/step - loss: 0.4418 - accuracy
Epoch 49/50
38/38 [==============================] - 0s 889us/step - loss: 0.4395 - accuracy
Epoch 50/50
38/38 [==============================] - 0s 938us/step - loss: 0.4374 - accuracy
Accuracy on training---
0.8414429530201343

Accuracy on test---
0.8417085427135679
```

ANSWER:

The Z-score of height values and eye color as a one-hot:

The accuracy of this prediction for training set is 0.8053691275167785 and the accuracy of this prediction for test set is 0.7989949748743719.

Question 5

Repeat question 4 for exercise hours + eye color

```
#same as 4a: no prepocessing for hour + eyecolor one hot
#exercise hour features
df_train_hour = df_train[['exercisehours']]
df_test_hour = df_test[['exercisehours']]

#get dummy variable and convert to array
```

```
X_train_dummy = pd.get_dummies(df_train[['eyecolor']])
y_train = df_train['gender']
X_test_dummy = pd.get_dummies(df_test[['eyecolor']])
y_test = df_test['gender']

X_train_q5 = X_train_dummy.join(df_train_hour)
X_test_q5 = X_test_dummy.join(df_test_hour)

#label coding for target
le = LabelEncoder()
le.fit(['female', 'male'])
y_train = le.transform(y_train).reshape(-1, 1)
y_test = le.transform(y_test).reshape(-1, 1)

#keras training model
model = Sequential()
model.add(Dense(units=10, activation='sigmoid'))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
model.fit(X_train_q5, y_train, epochs=50) #iter = 50

print('Accuracy on training---')
y_pred_scaled_train=model.predict(X_train_q5)
y_pred_scaled_train=(y_pred_scaled_train>.5)*1
print(accuracy_score(y_train, y_pred_scaled_train))

print('')
print('Accuracy on test---')
y_pred_scaled_test=model.predict(X_test_q5)
y_pred_scaled_test=(y_pred_scaled_test>.5)*1
print(accuracy_score(y_test, y_pred_scaled_test))
```

```
Epoch 24/50
38/38 [==============================] - 0s 858us/step - loss: 0.6827 - accuracy
Epoch 25/50
38/38 [==============================] - 0s 814us/step - loss: 0.6826 - accuracy
Epoch 26/50
38/38 [==============================] - 0s 935us/step - loss: 0.6826 - accuracy
Epoch 27/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6825 - accuracy:
Epoch 28/50
38/38 [==============================] - 0s 986us/step - loss: 0.6825 - accuracy
Epoch 29/50
38/38 [==============================] - 0s 859us/step - loss: 0.6825 - accuracy
Epoch 30/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6824 - accuracy:
Epoch 31/50
38/38 [==============================] - 0s 853us/step - loss: 0.6824 - accuracy
Epoch 32/50
38/38 [==============================] - 0s 909us/step - loss: 0.6824 - accuracy
Epoch 33/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6823 - accuracy:
Epoch 34/50
38/38 [==============================] - 0s 899us/step - loss: 0.6823 - accuracy
Epoch 35/50
```

```
38/38 [==============================] - 0s 887us/step - loss: 0.6822 - accuracy
Epoch 36/50
38/38 [==============================] - 0s 914us/step - loss: 0.6822 - accuracy
Epoch 37/50
38/38 [==============================] - 0s 797us/step - loss: 0.6821 - accuracy
Epoch 38/50
38/38 [==============================] - 0s 938us/step - loss: 0.6821 - accuracy
Epoch 39/50
38/38 [==============================] - 0s 869us/step - loss: 0.6822 - accuracy
Epoch 40/50

38/38 [==============================] - 0s 905us/step - loss: 0.6822 - accuracy
Epoch 41/50
38/38 [==============================] - 0s 923us/step - loss: 0.6820 - accuracy
Epoch 42/50
38/38 [==============================] - 0s 829us/step - loss: 0.6819 - accuracy
Epoch 43/50
38/38 [==============================] - 0s 971us/step - loss: 0.6820 - accuracy
Epoch 44/50
38/38 [==============================] - 0s 854us/step - loss: 0.6820 - accuracy
Epoch 45/50
38/38 [==============================] - 0s 851us/step - loss: 0.6819 - accuracy
Epoch 46/50
38/38 [==============================] - 0s 841us/step - loss: 0.6819 - accuracy
Epoch 47/50
38/38 [==============================] - 0s 961us/step - loss: 0.6819 - accuracy
Epoch 48/50
38/38 [==============================] - 0s 904us/step - loss: 0.6819 - accuracy
Epoch 49/50
38/38 [==============================] - 0s 811us/step - loss: 0.6819 - accuracy
Epoch 50/50
38/38 [==============================] - 0s 817us/step - loss: 0.6818 - accuracy
Accuracy on training---
0.5763422818791947

Accuracy on test---
0.5678391959798995
```

Original exercisehours values (no pre-processing) and eye color as a one-hot:

The accuracy of this prediction for training set is 0.5780201342281879 and the accuracy of this prediction for test set is 0.5527638190954773.

```
df_train[df_train['exercisehours']  == 0]
```

	Unnamed: 0	gender	age	year	eyecolor	height	miles	brothers	sisters	co
0	577	male	20	third	hazel	72.0	180.0	0	0	
5	878	female	27	third	hazel	67.0	0.0	2	1	
6	1689	female	20	third	blue	64.0	50.0	1	0	
8	1857	male	21	third	hazel	72.0	200.0	0	1	
12	809	female	20	second	brown	62.0	220.0	1	0	
...	...	...	...	...	...	...	...	...	...	
1179	1834	female	20	second	green	71.0	120.0	1	2	

```
df_train.shape
```

```
(1192, 16)
```

```
#same as 4b: log + one hot

#drop all the 0 value before converting to log base
df_train_temp = df_train.copy()
df_train_temp = df_train_temp[df_train_temp['exercisehours'] != 0].dropna()
df_test_temp = df_test.copy()
df_test_temp = df_test_temp[df_test_temp['exercisehours'] != 0].dropna()

# apply log to the exercisehours
df_train_temp['exercisehours'] = df_train_temp['exercisehours'].apply('log')
df_test_temp['exercisehours'] = df_test_temp['exercisehours'].apply('log')

df_train_exe = df_train_temp[['exercisehours']]
df_test_exe = df_test_temp[['exercisehours']]

#get dummy variable and convert to array
X_train_dummy = pd.get_dummies(df_train[['eyecolor']])
y_train = df_train['gender']
X_test_dummy = pd.get_dummies(df_test[['eyecolor']])
y_test = df_test['gender']

X_train_q5b = X_train_dummy.join(df_train_exe)
X_test_q5b = X_test_dummy.join(df_test_exe)

#label coding for target
le = LabelEncoder()
le.fit(['female', 'male'])
y_train = le.transform(y_train).reshape(-1, 1)
y_test = le.transform(y_test).reshape(-1, 1)

#keras training model
model = Sequential()
model.add(Dense(units=10, activation='sigmoid'))
model.add(Dense(units=1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
model.fit(X_train_q5b, y_train, epochs=50) #iter = 50

print('Accuracy on training---')
y_pred_scaled_train=model.predict(X_train_q5b)
y_pred_scaled_train=(y_pred_scaled_train>.5)*1
print(accuracy_score(y_train, y_pred_scaled_train))

print('')
print('Accuracy on test---')
y_pred_scaled_test=model.predict(X_test_q5b)
y_pred_scaled_test=(y_pred_scaled_test>.5)*1
print(accuracy_score(y_test, y_pred_scaled_test))
```

```
Epoch 24/50
38/38 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.5
Epoch 25/50
38/38 [==============================] - 0s 838us/step - loss: nan - accuracy: 0
Epoch 26/50
38/38 [==============================] - 0s 879us/step - loss: nan - accuracy: 0
Epoch 27/50
38/38 [==============================] - 0s 885us/step - loss: nan - accuracy: 0
Epoch 28/50
38/38 [==============================] - 0s 835us/step - loss: nan - accuracy: 0
Epoch 29/50
38/38 [==============================] - 0s 963us/step - loss: nan - accuracy: 0
Epoch 30/50
38/38 [==============================] - 0s 910us/step - loss: nan - accuracy: 0
Epoch 31/50
38/38 [==============================] - 0s 861us/step - loss: nan - accuracy: 0
Epoch 32/50
38/38 [==============================] - 0s 855us/step - loss: nan - accuracy: 0
Epoch 33/50
38/38 [==============================] - 0s 824us/step - loss: nan - accuracy: 0
Epoch 34/50
38/38 [==============================] - 0s 933us/step - loss: nan - accuracy: 0
Epoch 35/50
38/38 [==============================] - 0s 948us/step - loss: nan - accuracy: 0
Epoch 36/50
38/38 [==============================] - 0s 913us/step - loss: nan - accuracy: 0
Epoch 37/50
38/38 [==============================] - 0s 948us/step - loss: nan - accuracy: 0
Epoch 38/50
38/38 [==============================] - 0s 926us/step - loss: nan - accuracy: 0
Epoch 39/50
38/38 [==============================] - 0s 869us/step - loss: nan - accuracy: 0
Epoch 40/50
38/38 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.5
Epoch 41/50
38/38 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.5
Epoch 42/50
38/38 [==============================] - 0s 919us/step - loss: nan - accuracy: 0
Epoch 43/50
38/38 [==============================] - 0s 862us/step - loss: nan - accuracy: 0
Epoch 44/50
38/38 [==============================] - 0s 914us/step - loss: nan - accuracy: 0
Epoch 45/50
```

```
38/38 [==============================] - 0s 855us/step - loss: nan - accuracy: 0
Epoch 46/50
38/38 [==============================] - 0s 881us/step - loss: nan - accuracy: 0
Epoch 47/50
38/38 [==============================] - 0s 970us/step - loss: nan - accuracy: 0
Epoch 48/50
38/38 [==============================] - 0s 790us/step - loss: nan - accuracy: 0
Epoch 49/50
38/38 [==============================] - 0s 1ms/step - loss: nan - accuracy: 0.5

Epoch 50/50
38/38 [==============================] - 0s 877us/step - loss: nan - accuracy: 0
Accuracy on training---
0.5427852348993288

Accuracy on test---
0.5226130653266332
```

The log of exercisehours values and eye color as a one-hot:

The accuracy of this prediction for training set is 0.5427852348993288 and the accuracy of this prediction for test set is 0.5226130653266332.

```
#same as 4c: z score + one hot
# z score function
e_mean = df_train['exercisehours'].mean()
e_std = df_train['exercisehours'].std()
calculate_Zscore = lambda x: (x - e_mean) / e_std

# apply z score to the exercisehours
df_train_temp = df_train.copy()
df_train_temp['exercisehours'] = df_train_temp['exercisehours'].apply(calculate_Zscore
df_test_temp = df_test.copy()
df_test_temp['exercisehours'] = df_test_temp['exercisehours'].apply(calculate_Zscore)

df_train_exe = df_train_temp[['exercisehours']]
df_test_exe = df_test_temp[['exercisehours']]

#get dummy variable and convert to array
X_train_dummy = pd.get_dummies(df_train[['eyecolor']])
y_train = df_train['gender']
X_test_dummy = pd.get_dummies(df_test[['eyecolor']])
y_test = df_test['gender']

X_train_q5c = X_train_dummy.join(df_train_exe)
X_test_q5c = X_test_dummy.join(df_test_exe)

#label coding for target
le = LabelEncoder()
le.fit(['female', 'male'])
y_train = le.transform(y_train).reshape(-1, 1)
y_test = le.transform(y_test).reshape(-1, 1)
```

```
#keras training model
model = Sequential()
model.add(Dense(units=10, activation='sigmoid'))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
model.fit(X_train_q5c, y_train, epochs=50) #iter = 50

print('Accuracy on training---')
y_pred_scaled_train=model.predict(X_train_q5c)
y_pred_scaled_train=(y_pred_scaled_train>.5)*1
print(accuracy_score(y_train, y_pred_scaled_train))

print('')
print('Accuracy on test---')
y_pred_scaled_test=model.predict(X_test_q5c)
y_pred_scaled_test=(y_pred_scaled_test>.5)*1
print(accuracy_score(y_test, y_pred_scaled_test))
```

```
Epoch 1/50
38/38 [==============================] - 0s 921us/step - loss: 0.7337 - accuracy
Epoch 2/50
38/38 [==============================] - 0s 929us/step - loss: 0.7167 - accuracy
Epoch 3/50
38/38 [==============================] - 0s 918us/step - loss: 0.7073 - accuracy
Epoch 4/50
38/38 [==============================] - 0s 896us/step - loss: 0.7024 - accuracy
Epoch 5/50
38/38 [==============================] - 0s 878us/step - loss: 0.7000 - accuracy
Epoch 6/50
38/38 [==============================] - 0s 868us/step - loss: 0.6986 - accuracy
Epoch 7/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6977 - accuracy: (
Epoch 8/50
38/38 [==============================] - 0s 955us/step - loss: 0.6971 - accuracy
Epoch 9/50
38/38 [==============================] - 0s 845us/step - loss: 0.6966 - accuracy
Epoch 10/50
38/38 [==============================] - 0s 885us/step - loss: 0.6962 - accuracy
Epoch 11/50
38/38 [==============================] - 0s 841us/step - loss: 0.6960 - accuracy
Epoch 12/50
38/38 [==============================] - 0s 948us/step - loss: 0.6956 - accuracy
Epoch 13/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6953 - accuracy: (
Epoch 14/50
38/38 [==============================] - 0s 897us/step - loss: 0.6949 - accuracy
Epoch 15/50
38/38 [==============================] - 0s 916us/step - loss: 0.6946 - accuracy
Epoch 16/50
38/38 [==============================] - 0s 821us/step - loss: 0.6943 - accuracy
Epoch 17/50
38/38 [==============================] - 0s 852us/step - loss: 0.6939 - accuracy
Epoch 18/50
38/38 [==============================] - 0s 827us/step - loss: 0.6937 - accuracy
```

```
Epoch 19/50
38/38 [==============================] - 0s 848us/step - loss: 0.6934 - accuracy
Epoch 20/50
38/38 [==============================] - 0s 875us/step - loss: 0.6931 - accuracy
Epoch 21/50
38/38 [==============================] - 0s 905us/step - loss: 0.6929 - accuracy
Epoch 22/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6926 - accuracy: (
Epoch 23/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6924 - accuracy: (
Epoch 24/50
38/38 [==============================] - 0s 901us/step - loss: 0.6921 - accuracy
Epoch 25/50
38/38 [==============================] - 0s 872us/step - loss: 0.6919 - accuracy
Epoch 26/50
38/38 [==============================] - 0s 883us/step - loss: 0.6916 - accuracy
Epoch 27/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6914 - accuracy: (
Epoch 28/50
38/38 [==============================] - 0s 983us/step - loss: 0.6912 - accuracy
Epoch 29/50
38/38 [==============================] - 0s 892us/step - loss: 0.6910 - accuracy
Epoch 30/50
```

The Z-score of exercisehours values and eye color as a one-hot:

The accuracy of this prediction for training set is 0.5528523489932886 and the accuracy of this prediction for test set is 0.550251256281407.

Question 6

Combine the features from question 3, 4, and 5 (year, eyecolor, exercise, height, exercise hours). For numeric features use the best normalization method from questions 4 and 5.

Question 6.a

What was the NN accuracy on the test set using the single 10 node hidden layer?

```
#best normalization:
# q4 -- The Z-score of height values and eye color as a one-hot
# q5 -- The Z-score of exercise hours values and eye color as a one-hot

# q3:
df_train_bin = df_train[['year', 'eyecolor', 'exercise']]
df_test_bin = df_test[['year', 'eyecolor', 'exercise']]
X_train_dummy = pd.get_dummies(df_train_bin)
X_test_dummy = pd.get_dummies(df_test_bin)

# q4: z score function
h_mean = df_train['height'].mean()
h_std = df_train['height'].std()
```

```
calculate_Zscore = lambda x: (x - h_mean) / h_std

# apply z score to the height
df_train_temp = df_train.copy()
df_train_temp['height'] = df_train_temp['height'].apply(calculate_Zscore)
df_test_temp = df_test.copy()
df_test_temp['height'] = df_test_temp['height'].apply(calculate_Zscore)

df_train_height = df_train_temp[['height']]
df_test_height = df_test_temp[['height']]

# q5: z score function
e_mean = df_train['exercisehours'].mean()
e_std = df_train['exercisehours'].std()
calculate_Zscore = lambda x: (x - e_mean) / e_std

# apply z score to the exercisehours
df_train_temp = df_train.copy()
df_train_temp['exercisehours'] = df_train_temp['exercisehours'].apply(calculate_Zscore
df_test_temp = df_test.copy()
df_test_temp['exercisehours'] = df_test_temp['exercisehours'].apply(calculate_Zscore)

df_train_exe = df_train_temp[['exercisehours']]
df_test_exe = df_test_temp[['exercisehours']]

# combine table
df_train_q6 = X_train_dummy.join(df_train_height)
df_test_q6 = X_test_dummy.join(df_test_height)
df_train_q6 = df_train_q6.join(df_train_exe)
df_test_q6 = df_test_q6.join(df_test_exe)

#target
y_train = df_train['gender']
y_test = df_test['gender']

#label coding for target
le = LabelEncoder()
le.fit(['female', 'male'])
y_train = le.transform(y_train).reshape(-1, 1)
y_test = le.transform(y_test).reshape(-1, 1)

# keras training model
model = Sequential()
model.add(Dense(units=10, activation='sigmoid'))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
model.fit(df_train_q6, y_train, epochs=50) #iter = 50

print('Accuracy on training---')
y_pred_scaled_train=model.predict(df_train_q6)
y_pred_scaled_train=(y_pred_scaled_train>.5)*1
```

```
print(accuracy_score(y_train, y_pred_scaled_train))

print('')
print('Accuracy on test---')
y_pred_scaled_test=model.predict(df_test_q6)
y_pred_scaled_test=(y_pred_scaled_test>.5)*1
print(accuracy_score(y_test, y_pred_scaled_test))
```

```
Epoch 15/50
38/38 [==============================] - 0s 955us/step - loss: 0.6505 - accuracy
Epoch 16/50
38/38 [==============================] - 0s 872us/step - loss: 0.6466 - accuracy
Epoch 17/50
38/38 [==============================] - 0s 893us/step - loss: 0.6426 - accuracy
Epoch 18/50
38/38 [==============================] - 0s 897us/step - loss: 0.6387 - accuracy
Epoch 19/50
38/38 [==============================] - 0s 876us/step - loss: 0.6347 - accuracy
Epoch 20/50
38/38 [==============================] - 0s 881us/step - loss: 0.6307 - accuracy
Epoch 21/50
38/38 [==============================] - 0s 903us/step - loss: 0.6266 - accuracy
Epoch 22/50
38/38 [==============================] - 0s 779us/step - loss: 0.6226 - accuracy
Epoch 23/50
38/38 [==============================] - 0s 937us/step - loss: 0.6184 - accuracy
Epoch 24/50
38/38 [==============================] - 0s 928us/step - loss: 0.6144 - accuracy
Epoch 25/50
38/38 [==============================] - 0s 878us/step - loss: 0.6103 - accuracy
Epoch 26/50
38/38 [==============================] - 0s 1ms/step - loss: 0.6061 - accuracy:
Epoch 27/50
38/38 [==============================] - 0s 821us/step - loss: 0.6019 - accuracy
Epoch 28/50
38/38 [==============================] - 0s 891us/step - loss: 0.5976 - accuracy
Epoch 29/50
38/38 [==============================] - 0s 918us/step - loss: 0.5934 - accuracy
Epoch 30/50
38/38 [==============================] - 0s 852us/step - loss: 0.5891 - accuracy
Epoch 31/50
38/38 [==============================] - 0s 802us/step - loss: 0.5848 - accuracy
Epoch 32/50
38/38 [==============================] - 0s 978us/step - loss: 0.5805 - accuracy
Epoch 33/50
38/38 [==============================] - 0s 817us/step - loss: 0.5763 - accuracy
Epoch 34/50
38/38 [==============================] - 0s 921us/step - loss: 0.5719 - accuracy
Epoch 35/50
38/38 [==============================] - 0s 824us/step - loss: 0.5676 - accuracy
Epoch 36/50
38/38 [==============================] - 0s 835us/step - loss: 0.5634 - accuracy
Epoch 37/50
38/38 [==============================] - 0s 915us/step - loss: 0.5591 - accuracy
Epoch 38/50
38/38 [==============================] - 0s 902us/step - loss: 0.5548 - accuracy
Epoch 39/50
38/38 [==============================] - 0s 834us/step - loss: 0.5506 - accuracy
```

```
Epoch 40/50
38/38 [==============================] - 0s 1ms/step - loss: 0.5464 - accuracy: 
Epoch 41/50
38/38 [==============================] - 0s 1ms/step - loss: 0.5423 - accuracy: 
Epoch 42/50
38/38 [==============================] - 0s 848us/step - loss: 0.5381 - accuracy
Epoch 43/50
38/38 [==============================] - 0s 953us/step - loss: 0.5339 - accuracy
Epoch 44/50
38/38 [==============================] - 0s 891us/step - loss: 0.5298 - accuracy
```

ANSWER:

The accuracy of this prediction for training set is 0.8238255033557047 and the accuracy of this prediction for test set is 0.821608040201005.

Question 7- Bonus (10%)

Can you improve your test set prediction accuracy by 5% or more?

See how close to that milestone of improvement you can get by modifying the tuning parameters of Neural Networks (the number of hidden layers, number of hidden nodes in each layer, the learning rate aka mu). A great guide to tuning parameters is explained in this guide: http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

While the guide is specific to SVM and in particular the C and gamma parameters of the RBF kernel, the method applies to generally to any ML technique with tuning parameters.

Please also write a paragraph in a markdown cell below with an explanation of your approach and evaluation metrics.

```
from keras.layers import Dropout, BatchNormalization
from tensorflow.keras.optimizers import SGD
from keras.constraints import maxnorm

#best normalization:
# q4 -- The Z-score of height values and eye color as a one-hot
# q5 -- The Z-score of exercise hours values and eye color as a one-hot

# q3:
df_train_bin = df_train[['year', 'eyecolor', 'exercise']]
df_test_bin = df_test[['year', 'eyecolor', 'exercise']]
X_train_dummy = pd.get_dummies(df_train_bin)
X_test_dummy = pd.get_dummies(df_test_bin)

# q4: z score function
h_mean = df_train['height'].mean()
h_std = df_train['height'].std()
```

```
calculate_Zscore = lambda x: (x - h_mean) / h_std

# apply z score to the height
df_train_temp = df_train.copy()
df_train_temp['height'] = df_train_temp['height'].apply(calculate_Zscore)
df_test_temp = df_test.copy()
df_test_temp['height'] = df_test_temp['height'].apply(calculate_Zscore)

df_train_height = df_train_temp[['height']]
df_test_height = df_test_temp[['height']]

# q5: z score function
e_mean = df_train['exercisehours'].mean()
e_std = df_train['exercisehours'].std()
calculate_Zscore = lambda x: (x - e_mean) / e_std

# apply z score to the exercisehours
df_train_temp = df_train.copy()
df_train_temp['exercisehours'] = df_train_temp['exercisehours'].apply(calculate_Zscore
df_test_temp = df_test.copy()
df_test_temp['exercisehours'] = df_test_temp['exercisehours'].apply(calculate_Zscore)

df_train_exe = df_train_temp[['exercisehours']]
df_test_exe = df_test_temp[['exercisehours']]

# combine table
df_train_q6 = X_train_dummy.join(df_train_height)
df_test_q6 = X_test_dummy.join(df_test_height)
df_train_q6 = df_train_q6.join(df_train_exe)
df_test_q6 = df_test_q6.join(df_test_exe)

#target
y_train = df_train['gender']
y_test = df_test['gender']

#label coding for target
le = LabelEncoder()
le.fit(['female', 'male'])
y_train = le.transform(y_train).reshape(-1, 1)
y_test = le.transform(y_test).reshape(-1, 1)

#keras training model
model = Sequential()
# model.add(Dropout(0.2, input_shape=(14,)))
model.add(Dense(units=200, activation='sigmoid'))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy']) #sgd,
model.fit(df_train_q6, y_train, epochs=300) #iter = 300 batch_size=512,

print('Accuracy on training---')
y_pred_scaled_train=model.predict(df_train_q6)
y_pred_scaled_train=(y_pred_scaled_train>.5)*1
```

```
print(accuracy_score(y_train, y_pred_scaled_train))

print('')
print('Accuracy on test---')
y_pred_scaled_test=model.predict(df_test_q6)
y_pred_scaled_test=(y_pred_scaled_test>.5)*1
print(accuracy_score(y_test, y_pred_scaled_test))
```

```
Epoch 260/300
38/38 [==============================] - 0s 928us/step - loss: 0.3786 - accuracy
Epoch 261/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3784 - accuracy:
Epoch 262/300
38/38 [==============================] - 0s 905us/step - loss: 0.3781 - accuracy
Epoch 263/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3783 - accuracy:
Epoch 264/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3784 - accuracy:
Epoch 265/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3783 - accuracy:
Epoch 266/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3784 - accuracy:
Epoch 267/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3784 - accuracy:
Epoch 268/300
38/38 [==============================] - 0s 956us/step - loss: 0.3783 - accuracy
Epoch 269/300
38/38 [==============================] - 0s 965us/step - loss: 0.3789 - accuracy
Epoch 270/300
38/38 [==============================] - 0s 936us/step - loss: 0.3782 - accuracy
Epoch 271/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3779 - accuracy:
Epoch 272/300
38/38 [==============================] - 0s 955us/step - loss: 0.3783 - accuracy
Epoch 273/300
38/38 [==============================] - 0s 930us/step - loss: 0.3780 - accuracy
Epoch 274/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3783 - accuracy:
Epoch 275/300
38/38 [==============================] - 0s 970us/step - loss: 0.3778 - accuracy
Epoch 276/300
38/38 [==============================] - 0s 933us/step - loss: 0.3782 - accuracy
Epoch 277/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3783 - accuracy:
Epoch 278/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3785 - accuracy:
Epoch 279/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3783 - accuracy:
Epoch 280/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3780 - accuracy:
Epoch 281/300

38/38 [==============================] - 0s 1ms/step - loss: 0.3781 - accuracy:
Epoch 282/300
38/38 [==============================] - 0s 952us/step - loss: 0.3781 - accuracy
Epoch 283/300
38/38 [==============================] - 0s 923us/step - loss: 0.3781 - accuracy
```

```
Epoch 284/300
38/38 [==============================] - 0s 955us/step - loss: 0.3787 - accuracy
Epoch 285/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3785 - accuracy: (
Epoch 286/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3785 - accuracy: (
Epoch 287/300
38/38 [==============================] - 0s 1ms/step - loss: 0.3784 - accuracy: (
Epoch 288/300
38/38 [==============================] - 0s 927us/step - loss: 0.3783 - accuracy
Epoch 289/300
```

ANSWER:

The test accuracy improved 4% from 0.821608040201005 to 0.8618090452261307.

1. I set the epochs to 300, so the entire dataset will be passed forward and backward through the neural network 300 times.
2. Then I also change the dense units to 200, dimensionality of the output space. It is the unit parameter itself that plays a major role in the size of the weight matrix along with the bias vector.

✓ 13s completed at 11:19 PM