Pima Diabetes Dataset

I decided to use Pima Diabetes dataset that was used in Keras lab for building and training Nueral Network (NN). The dataset is avaiable in the course, however, it is no longer avaiable at UC Irvine Machine Learning repository. The same steps are repeated here as they were in the lab, first by training a Random Forest model to get a performance baseline and then use Keras library to build and train the NN and compare performance between the two models.

In [1]:
```
#Setup imports
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_auc_score, roc
_curve, accuracy_score
from sklearn.ensemble import RandomForestClassifier
```

In [2]:
```
## Import Keras objects for Deep Learning
## This requires both Keras and Tensorflow packages installed in your machine
## if you don't have these two packages, then use below commands to get the packages
## pip install keras
## pip install tensorflow
from keras.models  import Sequential
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from keras.optimizers import Adam, SGD, RMSprop
```

In [3]:
```
## Load in the data set
names = ["times_pregnant", "glucose_tolerance_test", "blood_pressure", "skin_thickness",
"insulin",
         "bmi", "pedigree_function", "age", "has_diabetes"]
diabetes_df = pd.read_csv('data/diabetes.csv', names=names, header=0)
```

Dataset Summary

The dataset has 768 rows and 9 columns. The dataset has 8 numerical features/columns and a label/target values are binary values on 0/1 (yes/no).

In [4]:
```
# Number of records and columns in the dataset
diabetes_df.shape
```

Out[4]:
```
(768, 9)
```

In [5]:
```
# Show the columns in the dataset
diabetes_df.columns
```

Out[5]:
```
Index(['times_pregnant', 'glucose_tolerance_test', 'blood_pressure',
       'skin_thickness', 'insulin', 'bmi', 'pedigree_function', 'age',
       'has_diabetes'],
      dtype='object')
```

In [6]:
```
# Show the data type of each column
diabetes_df.dtypes
```

Out[6]:
```
times_pregnant              int64
glucose_tolerance_test      int64
blood_pressure              int64
skin_thickness              int64
insulin                     int64
bmi                       float64
pedigree_function         float64
age                         int64
has_diabetes                int64
dtype: object
```

In [7]:
```
# Show the values in the label column
diabetes_df.has_diabetes.value_counts()
```

Out[7]:
```
0    500
1    268
Name: has_diabetes, dtype: int64
```

In [8]:
```
# Take a peek at the data -- random 5 records/observations
diabetes_df.sample(5)
```

Out[8]:

	times_pregnant	glucose_tolerance_test	blood_pressure	skin_thickness	insulin	bmi	pedigree_function
682	0	95	64	39	105	44.6	0.366
323	13	152	90	33	29	26.8	0.731
311	0	106	70	37	148	39.4	0.605
418	1	83	68	0	0	18.2	0.624
120	0	162	76	56	100	53.2	0.759

In [9]:
```
X = diabetes_df.iloc[:, :-1].values
y = diabetes_df["has_diabetes"].values
```

In [10]:
```
# Split the data to Train, and Test (75%, 25%)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1
1111)
```

In [11]:
```
np.mean(y), np.mean(1-y)
```

Out[11]:
```
(0.3489583333333333, 0.6510416666666666)
```

There are about 35% of the patients in this dataset have diabetes, while 65% do not.
Next step is to calculate ROC-AUC score and accuracy to evaluate performance of the model.

Summary of Training

Baseline performance using Random Forest

Train Random Forest model using 200 trees.

In [12]:
```
### BEGIN SOLUTION
## Train the RF Model using 200 trees
rf_model = RandomForestClassifier(n_estimators=200)
rf_model.fit(X_train, y_train)
```

Out[12]:
```
RandomForestClassifier(n_estimators=200)
```

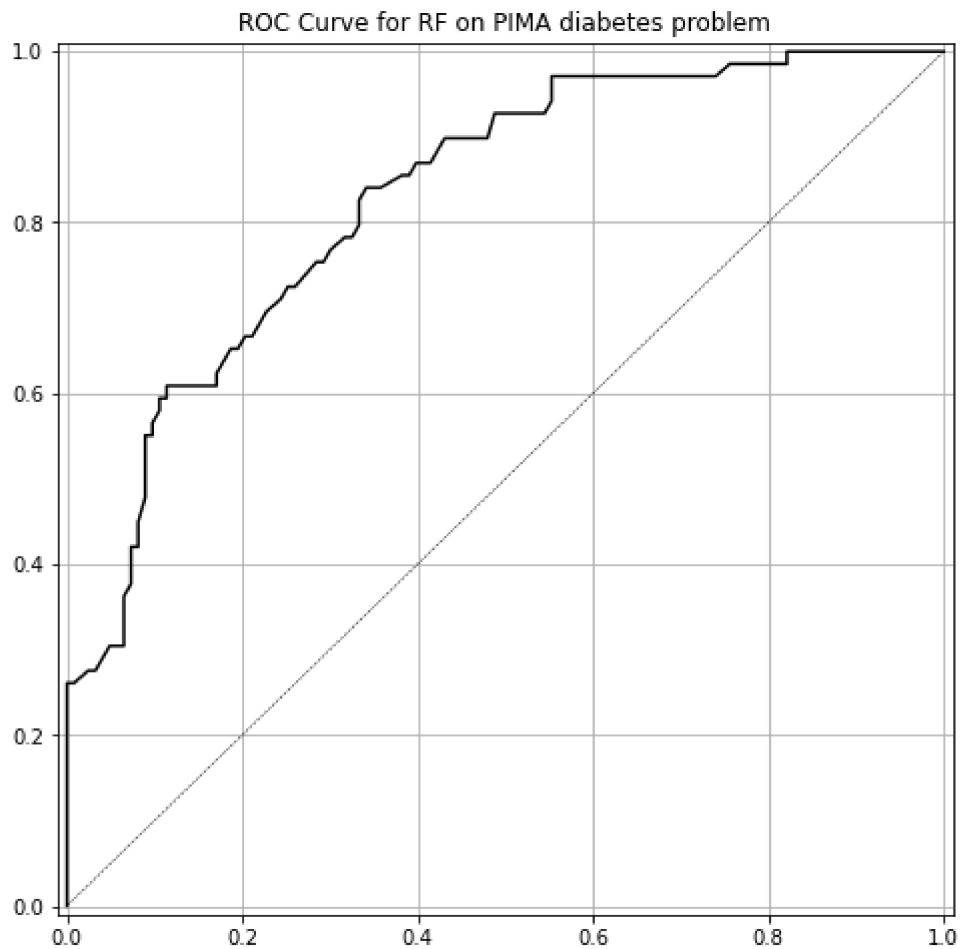
Make predictions on the test set.

In [13]:
```
# Make predictions on the test set - both "hard" predictions, and the scores (percent of trees voting yes)
y_pred_class_rf = rf_model.predict(X_test)
y_pred_prob_rf = rf_model.predict_proba(X_test)


print('accuracy is {:.3f}'.format(accuracy_score(y_test,y_pred_class_rf)))
print('roc-auc is {:.3f}'.format(roc_auc_score(y_test,y_pred_prob_rf[:,1])))
```

```
accuracy is 0.776
roc-auc is 0.829
```

In [14]:
```
def plot_roc(y_test, y_pred, model_name):
    fpr, tpr, thr = roc_curve(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.plot(fpr, tpr, 'k-')
    ax.plot([0, 1], [0, 1], 'k--', linewidth=.5)  # roc curve for random model
    ax.grid(True)
    ax.set(title='ROC Curve for {} on PIMA diabetes problem'.format(model_name),
           xlim=[-0.01, 1.01], ylim=[-0.01, 1.01])
plot_roc(y_test, y_pred_prob_rf[:, 1], 'RF')
### END SOLUTION
```



Performance using Single Hidden Layer Neural Network

Setting the input shape to 8 and single hidden layer with 12 nodes.

In [15]:
```
## First let's normalize the data
## This aids the training of neural nets by providing numerical stability
## Random Forest does not need this as it finds a split only, as opposed to performing m
atrix multiplications

normalizer = StandardScaler()
X_train_norm = normalizer.fit_transform(X_train)
X_test_norm = normalizer.transform(X_test)
```

In [16]:
```
# Define the Model
# Input size is 8-dimensional
# 1 hidden layer, 12 hidden nodes, sigmoid activation
# Final layer has just one node with a sigmoid activation (standard for binary classific
ation)

model_1 = Sequential()
model_1.add(Dense(12,input_shape = (8,),activation = 'sigmoid'))
model_1.add(Dense(1,activation='sigmoid'))
```

In [17]:
```
#  This is a nice tool to view the model you have created and count the parameters

model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 12)                108
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 13
=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
```

Why do we have 121 parameters? Does that make sense?

Fitting the model for 200 epochs.

In [32]:
```
# Fit(Train) the Model

# Compile the model with Optimizer, Loss Function and Metrics
# Roc-Auc is not available in Keras as an off the shelf metric yet, so we will skip it h
ere.

#model_1.compile(SGD(lr = .003), "binary_crossentropy", metrics=["accuracy"])
#run_hist_1 = model_1.fit(X_train_norm, y_train, validation_data=(X_test_norm, y_test),
 epochs=200)
# the fit function returns the run history.
# It is very convenient, as it contains information about the model fit, iterations etc.
```

In [19]:
```
## Like we did for the Random Forest, we generate two kinds of predictions
#  One is a hard decision, the other is a probabilitistic score.

y_pred_class_nn_1 = model_1.predict_classes(X_test_norm)
y_pred_prob_nn_1 = model_1.predict(X_test_norm)
```

In [20]:
```
# Let's check out the outputs to get a feel for how keras apis work.
y_pred_class_nn_1[:10]
```

Out[20]:
```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0]])
```

In [21]:
```
y_pred_prob_nn_1[:10]
```

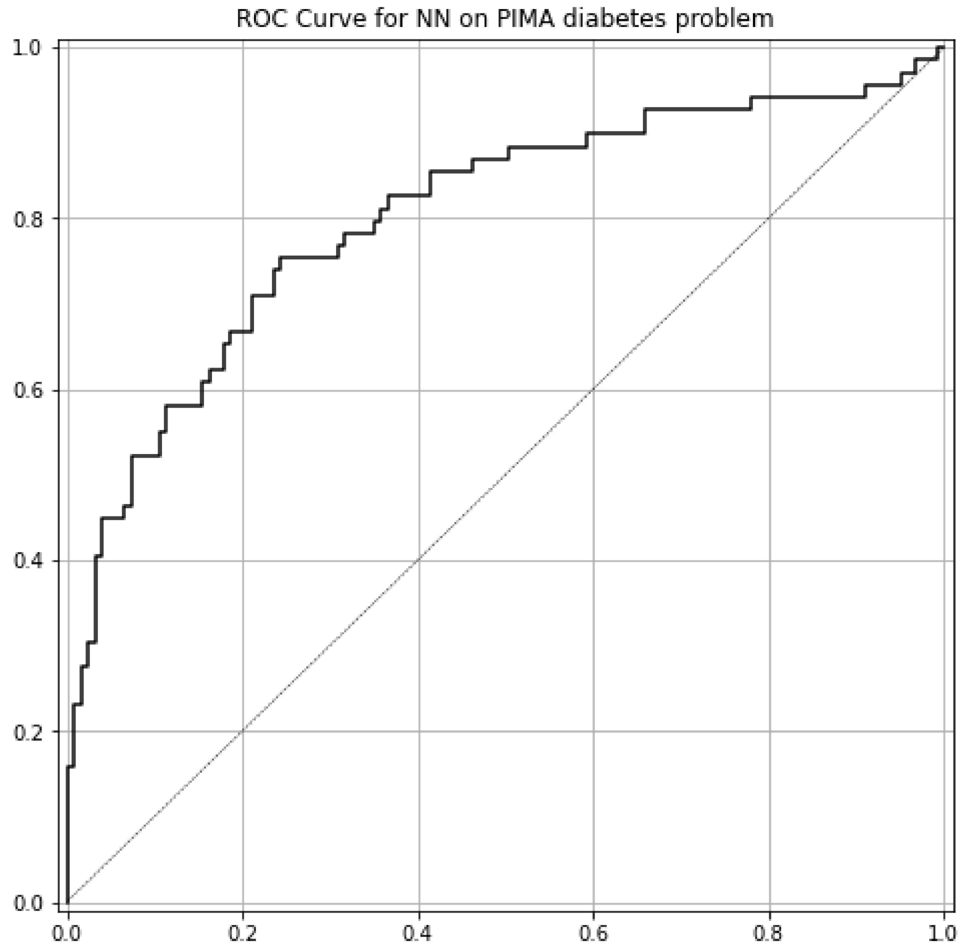
Out[21]:
```
array([[0.30299008],
       [0.39518115],
       [0.31488007],
       [0.31527472],
       [0.28619194],
       [0.37406597],
       [0.27703348],
       [0.2966178 ],
       [0.4701667 ],
       [0.32326555]], dtype=float32)
```

Plotting ROC Curve

In [22]:
```
# Print model performance and plot the roc curve
print('accuracy is {:.3f}'.format(accuracy_score(y_test,y_pred_class_nn_1)))
print('roc-auc is {:.3f}'.format(roc_auc_score(y_test,y_pred_prob_nn_1)))

plot_roc(y_test, y_pred_prob_nn_1, 'NN')
```

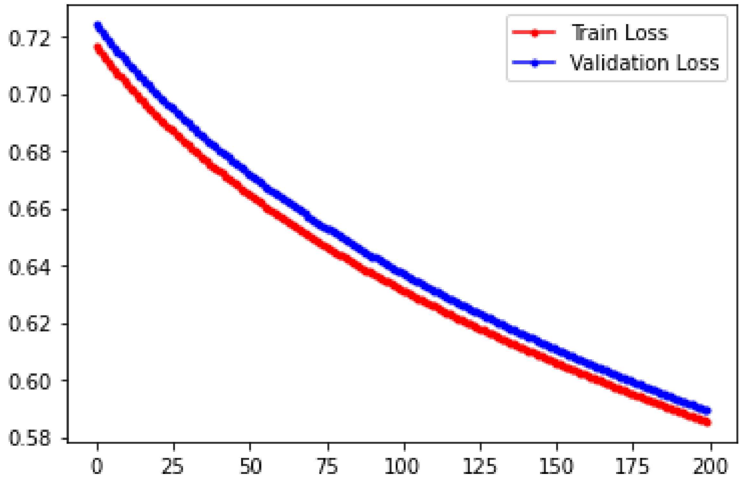
```
accuracy is 0.651
roc-auc is 0.802
```



Plotting the training loss and the validation loss over the different epochs

In [23]:
```
fig, ax = plt.subplots()
ax.plot(run_hist_1.history["loss"],'r', marker='.', label="Train Loss")
ax.plot(run_hist_1.history["val_loss"],'b', marker='.', label="Validation Loss")
ax.legend()
```

Out[23]: <matplotlib.legend.Legend at 0x27dd05cd940>



From the above graph, itlooks like the losses are still going down on both the training set and the validation set. This suggests that the model might benefit from further training. Train for 1000 more epochs.

In [33]:
```
## Note that when we call "fit" again, it picks up where it left off
#run_hist_1b = model_1.fit(X_train_norm, y_train, validation_data=(X_test_norm, y_test),
epochs=1000)
```

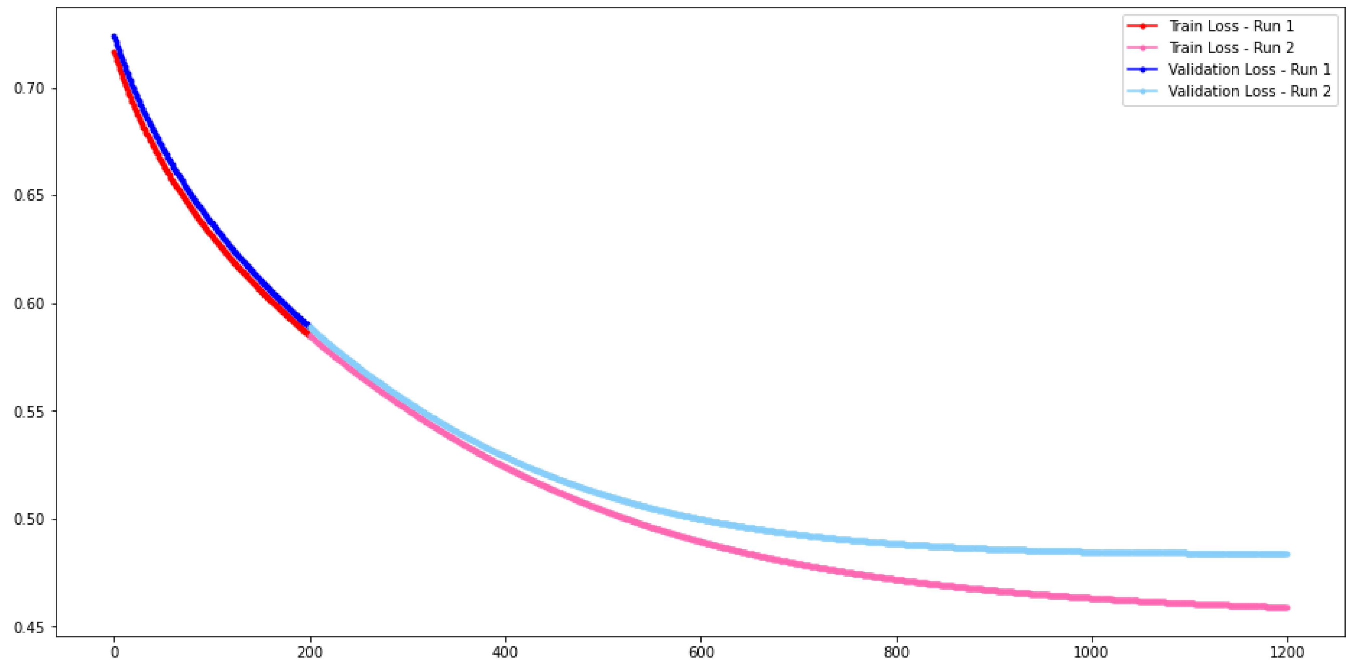
In [25]:
```
n = len(run_hist_1.history["loss"])
m = len(run_hist_1b.history['loss'])
fig, ax = plt.subplots(figsize=(16, 8))

ax.plot(range(n), run_hist_1.history["loss"],'r', marker='.', label="Train Loss - Run 1"
)
ax.plot(range(n, n+m), run_hist_1b.history["loss"], 'hotpink', marker='.', label="Train
 Loss - Run 2")

ax.plot(range(n), run_hist_1.history["val_loss"],'b', marker='.', label="Validation Loss
- Run 1")
ax.plot(range(n, n+m), run_hist_1b.history["val_loss"], 'LightSkyBlue', marker='.',  lab
el="Validation Loss - Run 2")

ax.legend()
```

Out[25]: <matplotlib.legend.Legend at 0x27dd161e860>



In the above graph, it looks like the training loss continues to go down, the validation loss has stopped improving. Further training will not benefit this model.

Performance using NN with 2 hidden layers

For this exercise, do the following in the cells below: Train for 1500 epochs using relu activation function for hidden layers and sigmoid function for final output layer.

In [34]:
```
### BEGIN SOLUTION
#model_2 = Sequential()
#model_2.add(Dense(6, input_shape=(8,), activation="relu"))
#model_2.add(Dense(6,  activation="relu"))
#model_2.add(Dense(1, activation="sigmoid"))

#model_2.compile(SGD(lr = .003), "binary_crossentropy", metrics=["accuracy"])
#run_hist_2 = model_2.fit(X_train_norm, y_train, validation_data=(X_test_norm, y_test),
 epochs=1500)
```

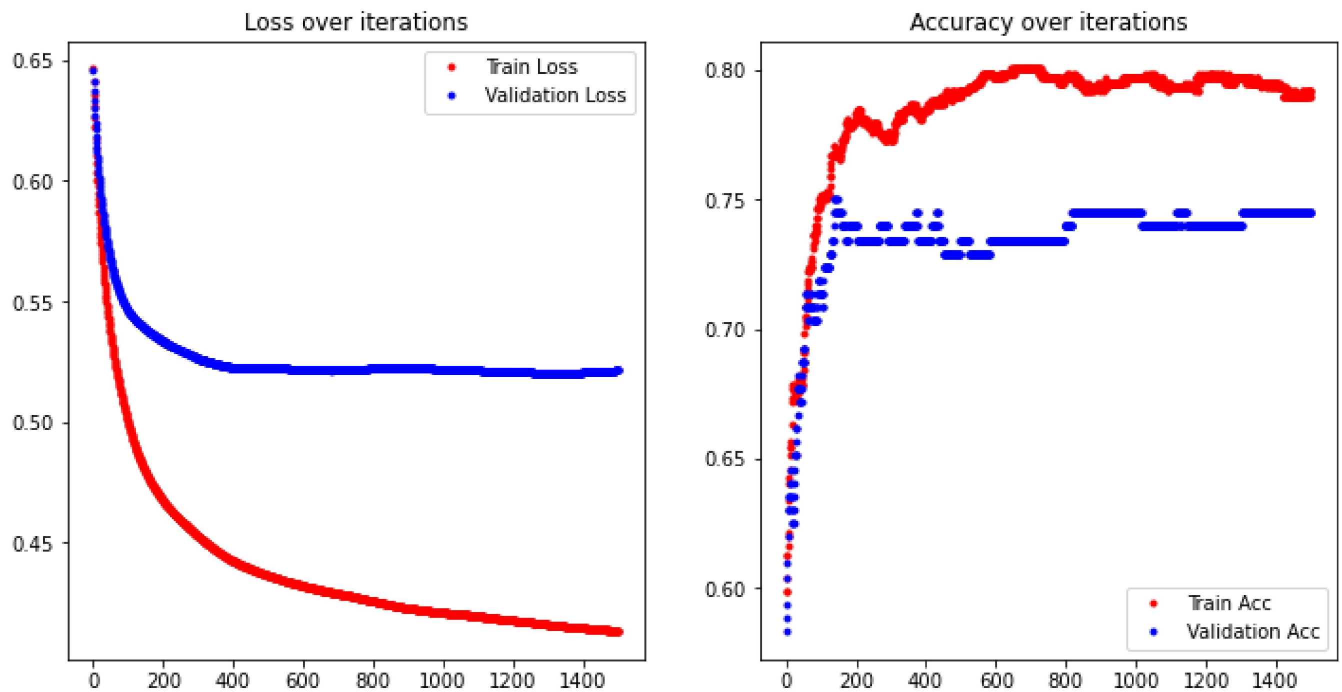
Plotting loss function and accuracy

In [28]:
```
n = len(run_hist_2.history["loss"])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.plot(range(n), (run_hist_2.history["loss"]),'r.', label="Train Loss")
ax.plot(range(n), (run_hist_2.history["val_loss"]),'b.', label="Validation Loss")
ax.legend()
ax.set_title('Loss over iterations')

ax = fig.add_subplot(1, 2, 2)
ax.plot(range(n), (run_hist_2.history["accuracy"]),'r.', label="Train Acc")
ax.plot(range(n), (run_hist_2.history["val_accuracy"]),'b.', label="Validation Acc")
ax.legend(loc='lower right')
ax.set_title('Accuracy over iterations')
```

Out[28]:
```
Text(0.5, 1.0, 'Accuracy over iterations')
```

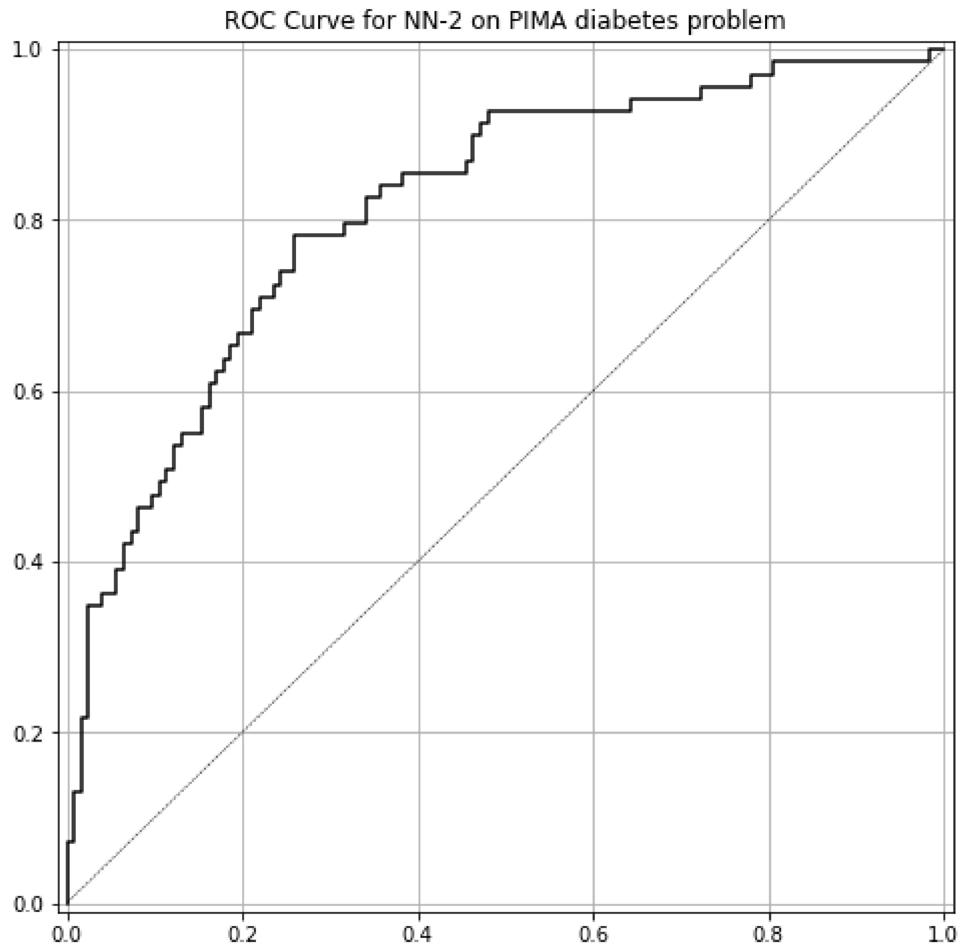


Plotting ROC Curve

In [29]:
```
y_pred_class_nn_2 = model_2.predict_classes(X_test_norm)
y_pred_prob_nn_2 = model_2.predict(X_test_norm)
print('')
print('accuracy is {:.3f}'.format(accuracy_score(y_test,y_pred_class_nn_2)))
print('roc-auc is {:.3f}'.format(roc_auc_score(y_test,y_pred_prob_nn_2)))

plot_roc(y_test, y_pred_prob_nn_2, 'NN-2')
### END SOLUTION
```

```
accuracy is 0.745
roc-auc is 0.814
```



Key Findings and Insights

Training and validation losses continued to decrease when the single hidden layer model was run for 200 epochs, however, after running for 1000 epochs and running 2-hidden layer model for 1500 epochs, the validation loss did not improve after 800 epcohs. Therefore, it can be concluded that running for more than 800 epochs would not improve the performance either we have single hidden layer or 2 hidden layers and tends to overfit the model.

Next Steps

The next step would be to cover all the topics discussed in the course and create a one python notebook to provide end to end solution for predicting diabetes using other Neural Networks. The dataset used for this exercise was also small. It would be much better to find a larger dataset and use these NN to perform predictions such as text and speech analysis using Recurrent-NN and image analysis using Convulotional-NN.