Yating Liao

ISE 599 Deep Learning Student ID: 7636428840

Diabetes dataset

In [69]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [70]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [71]:
```
from tensorflow import keras
```

In [72]:
```
from tensorflow.keras import layers
```

In [73]:
```
# see keras version
keras.__version__
```

Out[73]: '2.12.0'

In [74]:
```
#pd.set_option('display.max_columns', None)
```

Get data

In [75]:
```
df = pd.read_csv('diabetes.csv')
df
```

Out[75]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	PedigreeFunc	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

The diabetes.csv comes from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements. All patients here are females at least 21 years old of Pima India heritage. The variables are:

Pregnancies: the number of times pregnant

Glucose Plasma: glucose concentration at 2 hours in an oral glucose tolerlance test.

BloodPressure: Diastolic blood pressure (mm Hg)

SkinThickness: Triceps skin fold thickness (mm)

Insulin: 2-Hour serum insulin (mu U/ml)

BMI: Body mass index(weight in kg/(height in m)^2)

DiabetesPedigreeFunction

Age

Outcome(0 or 1)

In [76]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Pregnancies    768 non-null    int64  
 1   Glucose        768 non-null    int64  
 2   BloodPressure  768 non-null    int64  
 3   SkinThickness  768 non-null    int64  
 4   Insulin        768 non-null    int64  
 5   BMI            768 non-null    float64
 6   PedigreeFunc   768 non-null    float64
 7   Age            768 non-null    int64  
 8   Outcome        768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [77]:
```
df.isnull().sum() # It has no missing values, because all missing values are replaced by 0.
```

Out[77]:
```
Pregnancies      0
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
PedigreeFunc     0
Age              0
Outcome          0
dtype: int64
```

In [78]:
```
# How many diabetes patients?
```

In [79]:
```
df.Outcome.value_counts()
```

Out[79]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [80]:
```
df.Outcome.value_counts()/768
```

Out[80]:
```
0    0.651042
1    0.348958
Name: Outcome, dtype: float64
```

In [81]:
```
# There are 35% diabetes patients
```

Data Preparation

In [82]:
```
# looking for missing values and outliers)
# Some columns have entries equal to zero
# For some columns that is not possible (i.e.,BMI, Insulin)
```

In [83]:
```
for col in df.columns:
    zeros = df.loc[df[col]==0].shape[0]
    print(col+": "+str(zeros))
```

```
Pregnancies: 111
Glucose: 5
BloodPressure: 35
SkinThickness: 227
Insulin: 374
BMI: 11
PedigreeFunc: 0
Age: 0
Outcome: 500
```

In [84]:
```
# Imputation
```

In [85]:
```
# replace the zeros with nan
```

In [86]:
```
df['Glucose'] = df['Glucose'].replace(0, np.nan)
df['BloodPressure'] = df['BloodPressure'].replace(0, np.nan)
df['SkinThickness'] = df['SkinThickness'].replace(0, np.nan)
df['Insulin'] = df['Insulin'].replace(0, np.nan)
df['BMI'] = df['BMI'].replace(0, np.nan)
```

In [87]:
```
df.isnull().sum()
```

Out[87]:
```
Pregnancies        0
Glucose            5
BloodPressure     35
SkinThickness    227
Insulin          374
BMI               11
PedigreeFunc       0
Age                0
Outcome            0
dtype: int64
```

In [88]:
```
# replace the nan with the average of that column
```

In [89]:
```
df['Glucose'] = df['Glucose'].fillna(df['Glucose'].mean())
df['BloodPressure'] = df['BloodPressure'].fillna(df['BloodPressure'].mean())
df['SkinThickness'] = df['SkinThickness'].fillna(df['SkinThickness'].mean())
df['Insulin'] = df['Insulin'].fillna(df['Insulin'].mean())
df['BMI'] = df['BMI'].fillna(df['BMI'].mean())
```

Split data

In [90]:
```
X = df.drop(['Outcome'],axis = 1)
Y = df.Outcome
```

In [91]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, stratify = Y,
                                                    test_size=0.2,
                                                    random_state=1)
```

In [92]:
```
y_train.shape
```

Out[92]:
```
(614,)
```

In [93]:
```
y_test.shape
```

Out[93]:
```
(154,)
```

Scale the data

In [94]:
```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:
```

```

Build Model

Build the Dense Neural Network (DNN) with 2 hidden layers

In [95]:
```
network1 = keras.Sequential([
    layers.Dense(32,activation='relu'), # first hidden layer
    layers.Dense(16,activation='relu'), # second hidden layer
    layers.Dense(1,activation='sigmoid') # output layer
])
```

Compile Model

In [96]:
```
# use loss function binary_crossentropy
network1.compile(optimizer='rmsprop',
                loss='binary_crossentropy',
                metrics=['accuracy'])
```

train(fit) model

In [97]:
```
n_epochs = 55
```

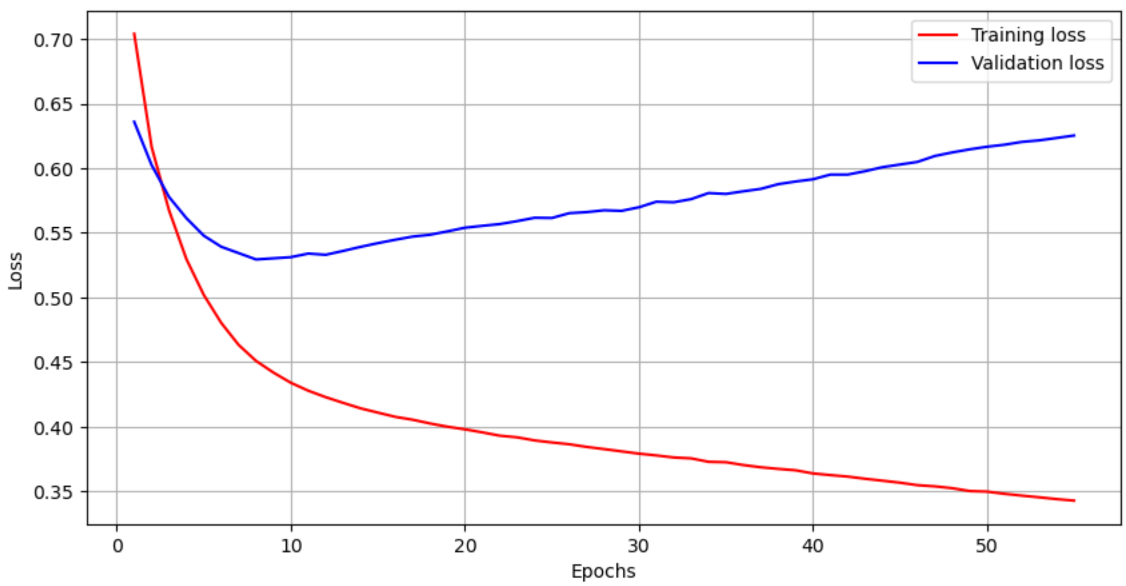
In [98]:
```
history = network1.fit(X_train_scaled, y_train,
                      epochs=n_epochs, batch_size=20,
                      validation_split = 0.20,
                      verbose = 0,
                     )
```

In [99]:
```
# values for train loss and train accuracy are shown at each step
```

In [100]:
```
loss = history.history["loss"]
val_loss = history.history["val_loss"]
```

In [101]:
```
epochs = range(1,n_epochs+1)
```

In [102]:
```
plt.figure(figsize=(10,5))
plt.plot(epochs, loss, "r",
         label="Training loss")
plt.plot(epochs, val_loss, "b",
         label="Validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
```



In [103]:
```
# The model starts overfitting after x epochs
```

In [104]:
```
## Retrain model with all train data (with x epochs)
```

In [105]:
```
model = keras.Sequential([
    layers.Dense(32,activation='relu'),
    layers.Dense(16,activation='relu'),
    layers.Dense(1,activation='sigmoid')
])

model.compile(optimizer='rmsprop',
                loss='binary_crossentropy',
                metrics=['accuracy'])

y_pred_probabilities = model.fit(X_train_scaled, y_train, epochs=10, batch_size=20)
print(y_pred_probabilities)
```

```
Epoch 1/10
31/31 [==============================] - 1s 3ms/step - loss: 0.6411 - accuracy: 0.6336
Epoch 2/10
31/31 [==============================] - 0s 4ms/step - loss: 0.5414 - accuracy: 0.7443
Epoch 3/10
31/31 [==============================] - 0s 3ms/step - loss: 0.4992 - accuracy: 0.7638
Epoch 4/10
31/31 [==============================] - 0s 4ms/step - loss: 0.4763 - accuracy: 0.7687
Epoch 5/10
31/31 [==============================] - 0s 2ms/step - loss: 0.4647 - accuracy: 0.7785
Epoch 6/10
31/31 [==============================] - 0s 2ms/step - loss: 0.4561 - accuracy: 0.7801
Epoch 7/10
31/31 [==============================] - 0s 4ms/step - loss: 0.4504 - accuracy: 0.7818
Epoch 8/10
31/31 [==============================] - 0s 3ms/step - loss: 0.4455 - accuracy: 0.7915
Epoch 9/10
31/31 [==============================] - 0s 4ms/step - loss: 0.4415 - accuracy: 0.7932
Epoch 10/10
31/31 [==============================] - 0s 4ms/step - loss: 0.4380 - accuracy: 0.7948
<keras.callbacks.History object at 0x000002438F083A00>
```

In [106]:
```
import pandas as pd
yhat = model.predict(X_test_scaled)

y_test = np.array(y_test)
yhat = np.array(y_test)

df2 = pd.DataFrame({'y_test':y_test,'yhat':yhat})
df2['y_test'] = pd.Series(y_test)
df2['yhat'] = pd.Series(yhat)
print(df2)
```

```
5/5 [==============================] - 0s 3ms/step
     y_test  yhat
0         0     0
1         0     0
2         0     0
3         0     0
4         0     0
..      ...   ...
149       0     0
150       0     0
151       0     0
152       1     1
153       0     0

[154 rows x 2 columns]
```

In [107]:
```
confusion_matrix = pd.crosstab(y_test, yhat, rownames=['Actual'], colnames=['Predicted'])
print(confusion_matrix)
```

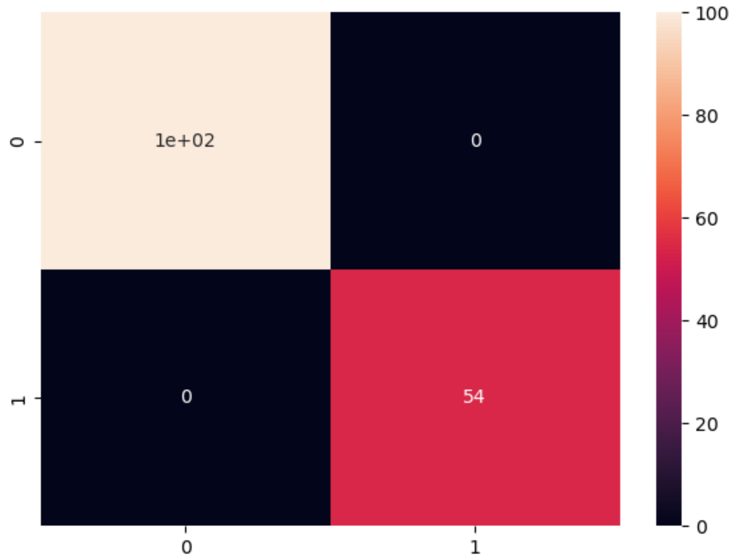
```
Predicted    0   1
Actual
0          100   0
1            0  54
```

In [108]:
```
# 2. Add a column with the error rates (accuracy) for each type of patient:
confusion_matrix["Total"] = confusion_matrix.sum(axis=1)
confusion_matrix["Error Rate"] = 1 - confusion_matrix[1] / confusion_matrix["Total"]
print(confusion_matrix)
```

```
Predicted    0   1  Total  Error Rate
Actual
0          100   0    100         1.0
1            0  54     54         0.0
```

In [109]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(y_test, yhat)
plt.figure(figsize=(7, 5))
sns.heatmap(mat, annot=True)
```

Out[109]: <Axes: >



what is the accuracy rate for predicting if a patient has diabetes, if in fact he has diabetes?

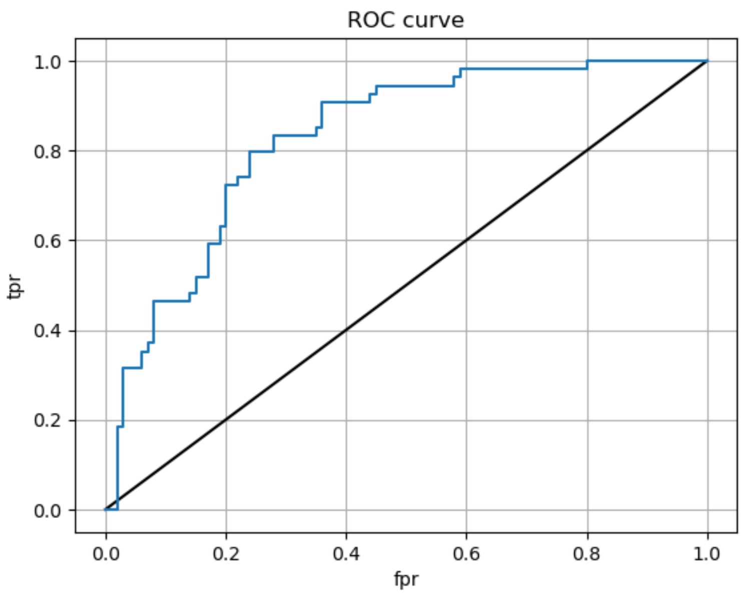
The accuracy rate for incorrection is 0, so there are no cases of incorrectly predict the diabetes status.

In [110]:
```
from sklearn.metrics import classification_report
target_names = ['Diabetes', 'Normal']
print(classification_report(y_test, yhat, target_names=target_names))
```

```
              precision    recall  f1-score   support

    Diabetes       1.00      1.00      1.00       100
      Normal       1.00      1.00      1.00        54

    accuracy                           1.00       154
   macro avg       1.00      1.00      1.00       154
weighted avg       1.00      1.00      1.00       154
```

In [111]:
```
from sklearn.metrics import roc_curve

y_pred_keras = model.predict(X_test_scaled).ravel()
fpr, tpr, thresholds = roc_curve(y_test, y_pred_keras)

plt.plot([0,1],[0,1],'k-')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('ROC curve')
plt.grid()
plt.show()
```

```
5/5 [==============================] - 0s 2ms/step
```



In [112]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred_keras)
```

Out[112]: 0.827962962962963

In [113]:
```
# define a function that accepts a threshold and prints sensitivity and specificity
def evaluate_threshold(threshold):
    print('Sensitivity:', tpr[thresholds > threshold][-1])
    print('Specificity:', 1 - fpr[thresholds > threshold][-1])

evaluate_threshold(0.3)
```

```
Sensitivity: 0.7962962962962963
Specificity: 0.72
```

In [114]:
```
evaluate_threshold(0.35)
```

```
Sensitivity: 0.7222222222222222
Specificity: 0.8
```

In [115]:
```
evaluate_threshold(0.4)
```

```
Sensitivity: 0.5925925925925926
Specificity: 0.83
```

In [116]:
```
evaluate_threshold(0.5)
```

```
Sensitivity: 0.48148148148148145
Specificity: 0.86
```

test model

In [117]:
```
test_loss,test_acc = model.evaluate(X_test_scaled, y_test)
```

```
5/5 [==============================] - 0s 4ms/step - loss: 0.5033 - accuracy: 0.7273
```

In [118]:
```
test_acc  # 0.7597402334213257
```

Out[118]:
```
0.7272727489471436
```

In [119]:
```
test_loss # 0.5030236840248108
```

Out[119]:
```
0.5032797455787659
```