Assignment 2 (Classification Problem) by vipin_2011MT22

```
#importing the libraries
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
from sklearn.model_selection import train_test_split
```

```
#importing the dataset
dataframe = pd.read_csv("diabetes.csv")
dataframe.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

```
#spilting of dataset into feature and label
df_label = dataframe['Outcome']
df_features = dataframe.drop('Outcome', 1)
df_features.replace('?', -99999, inplace=True)
print(df_label.head())
print(df_features.head())
```

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
   Pregnancies  Glucose  BloodPressure  ...   BMI  DiabetesPedigreeFunction  Age
0            6      148             72  ...  33.6                     0.627   50
1            1       85             66  ...  26.6                     0.351   31
2            8      183             64  ...  23.3                     0.672   32
3            1       89             66  ...  28.1                     0.167   21
4            0      137             40  ...  43.1                     2.288   33

[5 rows x 8 columns]
```

```
#hot encoding the label dataset
label = []
```

```
for lab in df_label:
    if lab == 1:
        label.append([1, 0])  # class 1
    elif lab == 0:
        label.append([0, 1])  # class 0
```

```
data = np.array(df_features)
label = np.array(label)
print(data.shape,label.shape)
```

```
(768, 8) (768, 2)
```

```
#spilting dataset into testing and training
x_train, x_test, y_train, y_test = train_test_split(data, label, test_size=0.2, random_state=
x_train.shape
```

```
(614, 8)
```

```
#building our Neural Network
model = Sequential()
model.add(Dense(500, input_dim=8, activation='sigmoid'))
model.add(Dense(100, activation='sigmoid'))
model.add(Dense(2, activation='softmax'))
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
model.fit(x_train,y_train, epochs=1000, batch_size=70, validation_data=(x_test, y_test))
```

```
9/9 [==============================] - 0s 11ms/step - loss: 0.0192 - accuracy: 0.9822
Epoch 973/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.0173 - accuracy: 0.9851
Epoch 974/1000
9/9 [==============================] - 0s 14ms/step - loss: 0.0265 - accuracy: 0.9681
Epoch 975/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.0405 - accuracy: 0.9546
Epoch 976/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.0343 - accuracy: 0.9628
Epoch 977/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.0220 - accuracy: 0.9766
Epoch 978/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.0266 - accuracy: 0.9730
Epoch 979/1000
9/9 [==============================] - 0s 13ms/step - loss: 0.0283 - accuracy: 0.9706
Epoch 980/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.0425 - accuracy: 0.9459
Epoch 981/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.0302 - accuracy: 0.9689
Epoch 982/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.0230 - accuracy: 0.9730
Epoch 983/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.0384 - accuracy: 0.9575
Epoch 984/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.0209 - accuracy: 0.9797
Epoch 985/1000
```

```
Epoch 985/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.0287 - accuracy: 0.9707
Epoch 986/1000
9/9 [==============================] - 0s 9ms/step - loss: 0.0224 - accuracy: 0.9696
Epoch 987/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.0357 - accuracy: 0.9574
Epoch 988/1000
9/9 [==============================] - 0s 14ms/step - loss: 0.0265 - accuracy: 0.9718
Epoch 989/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.0245 - accuracy: 0.9706
Epoch 990/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.0187 - accuracy: 0.9828
Epoch 991/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.0219 - accuracy: 0.9809
Epoch 992/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.0200 - accuracy: 0.9811
Epoch 993/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.0226 - accuracy: 0.9782
Epoch 994/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.0157 - accuracy: 0.9850
Epoch 995/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.0134 - accuracy: 0.9879
Epoch 996/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.0272 - accuracy: 0.9737
Epoch 997/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.0152 - accuracy: 0.9869
Epoch 998/1000
9/9 [==============================] - 0s 13ms/step - loss: 0.0140 - accuracy: 0.9883
Epoch 999/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.0107 - accuracy: 0.9907
Epoch 1000/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.0186 - accuracy: 0.9836
<tensorflow.python.keras.callbacks.History at 0x7f7c0c4e4690>
```

```
#predictating whether diabetic or not
Y_pred = model.predict(x_test)
Y_pred =(Y_pred>0.5)
result = np.delete(Y_pred, 1, 1)
result
```

```
            [ True],
            [False],
            [False],
            [ True],
            [False],
            [ True],
            [False],
            [ True],
            [ True],
            [False],
            [False],
            [ True],
            [ True],
            [False],
            [ True],
            [False],
            [False]
```

```
[False],
[False],
[ True],
[False],
[False],
[ True],
[False],
[False],
[ True],
[False],
[False],
[False],
[False],
[ True],
[False],
[False],
[ True],
[ True],

[False],
[ True],
[ True],
[False],
[ True],
[ True],
[False],
[ True],
[ True],
[ True],
[False],
[False],
[False],
[False],
[False],
[False],
[False],
[ True],
[False],
[False],
[False],
[ True],
[False],
[False],
[ True],
[False]])
```

```
#Calculating accuracy, f1 score, precision, recall
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score


accuracy = accuracy_score(y_test, Y_pred)
print('accuracy: ',accuracy)

score = f1_score(y_test, Y_pred, average='weighted')
```

```
print('F1-score: ',score)

precision = precision_score(y_test, Y_pred, labels=[0,1], average='weighted')
print('Precision: ', precision)

recall = recall_score(y_test, Y_pred, average='weighted')
print("recall: ", recall)
```

```
accuracy:  0.6883116883116883
F1-score:  0.6915024630541873
Precision:  0.6967590843846793
recall:  0.6883116883116883
```