Assignment 3 by vipin_2011MT22

Using Task1, we have to again create a NN model.The additional elements you have to put your NN is mentioned below.

❏Apply kernel_regularizer, bias_regularizer and activity_regularizer in both Hidden layers and set them to L2.

❏Also report the results for L1.

Note: You have to print the accuracy, f1 score, precision and recall

For l2

```
#importing the libraries
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from tensorflow.keras import regularizers
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score
```

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

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	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

```
#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=42)
```

```
x_train.shape
```

```
(614, 8)
```

```
; our Neural Network
Sequential()
d(Dense(500, input_dim=8, activation='sigmoid',kernel_regularizer=regularizers.l2(0.01), bias_regularizer=regularizers.l2(0.0
       activity_regularizer=regularizers.l2(0.01)))
d(Dense(100, activation='sigmoid',kernel_regularizer=regularizers.l2(0.01), bias_regularizer=regularizers.l2(0.01),
       activity_regularizer=regularizers.l2(0.01)))
d(Dense(2, activation='softmax'))
mpile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
:(x_train,y_train, epochs=1000, batch_size=70, validation_data=(x_test, y_test))
```

```
9/9 [==============================] - 0s 9ms/step - loss: 0.4190 - accuracy: 0.7562 - val_loss: 0.4282 - val_accurac
Epoch 973/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4108 - accuracy: 0.7670 - val_loss: 0.4281 - val_accura
Epoch 974/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4161 - accuracy: 0.7505 - val_loss: 0.4293 - val_accura
Epoch 975/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4107 - accuracy: 0.7891 - val_loss: 0.4353 - val_accura
Epoch 976/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4093 - accuracy: 0.7597 - val_loss: 0.4289 - val_accura
Epoch 977/1000
9/9 [==============================] - 0s 15ms/step - loss: 0.4109 - accuracy: 0.7735 - val_loss: 0.4306 - val_accura
Epoch 978/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4144 - accuracy: 0.7562 - val_loss: 0.4272 - val_accura
Epoch 979/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.4232 - accuracy: 0.7424 - val_loss: 0.4371 - val_accura
Epoch 980/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4180 - accuracy: 0.7376 - val_loss: 0.4293 - val_accura
Epoch 981/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4120 - accuracy: 0.7690 - val_loss: 0.4304 - val_accura
Epoch 982/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4072 - accuracy: 0.7888 - val_loss: 0.4283 - val_accura
Epoch 983/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4118 - accuracy: 0.7716 - val_loss: 0.4292 - val_accura
Epoch 984/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.4117 - accuracy: 0.7600 - val_loss: 0.4272 - val_accura
Epoch 985/1000
9/9 [==============================] - 0s 15ms/step - loss: 0.4112 - accuracy: 0.7621 - val_loss: 0.4320 - val_accura
Epoch 986/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4255 - accuracy: 0.7371 - val_loss: 0.4369 - val_accura
Epoch 987/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.4313 - accuracy: 0.7323 - val_loss: 0.4294 - val_accura
Epoch 988/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4171 - accuracy: 0.7584 - val_loss: 0.4319 - val_accura
Epoch 989/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.4091 - accuracy: 0.7769 - val_loss: 0.4287 - val_accura
Epoch 990/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4115 - accuracy: 0.7635 - val_loss: 0.4283 - val_accura
Epoch 991/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4132 - accuracy: 0.7756 - val_loss: 0.4389 - val_accura
Epoch 992/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.4208 - accuracy: 0.7478 - val_loss: 0.4290 - val_accura
Epoch 993/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4161 - accuracy: 0.7638 - val_loss: 0.4275 - val_accura
Epoch 994/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4149 - accuracy: 0.7556 - val_loss: 0.4310 - val_accura
Epoch 995/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.4200 - accuracy: 0.7310 - val_loss: 0.4295 - val_accura
Epoch 996/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.4033 - accuracy: 0.7821 - val_loss: 0.4261 - val_accura
Epoch 997/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4063 - accuracy: 0.7876 - val_loss: 0.4313 - val_accura
Epoch 998/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.4090 - accuracy: 0.7650 - val_loss: 0.4263 - val_accura
Epoch 999/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.4104 - accuracy: 0.7759 - val_loss: 0.4280 - val_accura
Epoch 1000/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.4166 - accuracy: 0.7421 - val_loss: 0.4276 - val_accura
<tensorflow.python.keras.callbacks.History at 0x7f73b20ec7d0>
```

```
#predictating whether diabetic or not
Y_pred = model.predict(x_test)
Y_pred =(Y_pred>0.5)
```

```
#Calculating accuracy, f1 score, precision, recall

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:  0.7662337662337663
F1-score:  0.7616056131190341
Precision:  0.7614407862966225
recall:  0.7662337662337663
```

For l1

```
#building our Neural Network
model = Sequential()
model.add(Dense(500, input_dim=8, activation='sigmoid',kernel_regularizer=regularizers.l1(0.0001), bias_regularizer=regulari
                activity_regularizer=regularizers.l1(0.0001)))
model.add(Dense(100, activation='sigmoid',kernel_regularizer=regularizers.l1(0.0001), bias_regularizer=regularizers.l1(0.000
                activity_regularizer=regularizers.l1(0.0001)))
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: 0.0991 - accuracy: 0.9416 - val_loss: 0.2850 - val_accura
Epoch 973/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.1162 - accuracy: 0.9211 - val_loss: 0.2835 - val_accura
Epoch 974/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.1118 - accuracy: 0.9221 - val_loss: 0.2872 - val_accura
Epoch 975/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.1038 - accuracy: 0.9207 - val_loss: 0.2823 - val_accura
Epoch 976/1000
9/9 [==============================] - 0s 14ms/step - loss: 0.1218 - accuracy: 0.9073 - val_loss: 0.2878 - val_accura
Epoch 977/1000
9/9 [==============================] - 0s 14ms/step - loss: 0.1072 - accuracy: 0.9230 - val_loss: 0.2793 - val_accura
Epoch 978/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.1033 - accuracy: 0.9404 - val_loss: 0.2884 - val_accura
Epoch 979/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.1094 - accuracy: 0.9298 - val_loss: 0.2757 - val_accura
Epoch 980/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.1036 - accuracy: 0.9508 - val_loss: 0.2873 - val_accura
Epoch 981/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.1059 - accuracy: 0.9332 - val_loss: 0.2808 - val_accura
Epoch 982/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.1023 - accuracy: 0.9385 - val_loss: 0.2893 - val_accura
Epoch 983/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.1088 - accuracy: 0.9326 - val_loss: 0.2781 - val_accura
Epoch 984/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.1025 - accuracy: 0.9425 - val_loss: 0.2891 - val_accura
Epoch 985/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.1035 - accuracy: 0.9463 - val_loss: 0.2819 - val_accura
Epoch 986/1000
9/9 [==============================] - 0s 14ms/step - loss: 0.1033 - accuracy: 0.9283 - val_loss: 0.2874 - val_accura
Epoch 987/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.1097 - accuracy: 0.9206 - val_loss: 0.2867 - val_accura
Epoch 988/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.0997 - accuracy: 0.9435 - val_loss: 0.2806 - val_accura
Epoch 989/1000
9/9 [==============================] - 0s 13ms/step - loss: 0.0986 - accuracy: 0.9336 - val_loss: 0.2853 - val_accura
Epoch 990/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.1054 - accuracy: 0.9356 - val_loss: 0.2816 - val_accura
Epoch 991/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.1054 - accuracy: 0.9341 - val_loss: 0.2786 - val_accura
Epoch 992/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.1077 - accuracy: 0.9372 - val_loss: 0.2917 - val_accura
Epoch 993/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.1013 - accuracy: 0.9494 - val_loss: 0.2789 - val_accura
Epoch 994/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.1097 - accuracy: 0.9175 - val_loss: 0.2873 - val_accura
Epoch 995/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.1051 - accuracy: 0.9291 - val_loss: 0.2831 - val_accura
Epoch 996/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.1092 - accuracy: 0.9258 - val_loss: 0.2853 - val_accura
Epoch 997/1000
9/9 [==============================] - 0s 12ms/step - loss: 0.1004 - accuracy: 0.9446 - val_loss: 0.2796 - val_accura
Epoch 998/1000
9/9 [==============================] - 0s 11ms/step - loss: 0.1161 - accuracy: 0.9150 - val_loss: 0.2823 - val_accura
Epoch 999/1000
9/9 [==============================] - 0s 14ms/step - loss: 0.1064 - accuracy: 0.9321 - val_loss: 0.2872 - val_accura
Epoch 1000/1000
9/9 [==============================] - 0s 10ms/step - loss: 0.1102 - accuracy: 0.9360 - val_loss: 0.2774 - val_accura
<tensorflow.python.keras.callbacks.History at 0x7f73abb4a490>
```

```
#predictating whether diabetic or not
```

```
Y_pred = model.predict(x_test)
Y_pred =(Y_pred>0.5)
```

```
#Calculating accuracy, f1 score, precision, recall

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:  0.6818181818181818
F1-score:  0.6846136517233611
Precision:  0.6888297872340425
recall:  0.6818181818181818
```

✓ 0s completed at 3:49 PM