cancer_classification.ipynb

Code | Python 3 (ipykernel)

Basic Imports

[1]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
```

Loading the data

[2]:
```
data=pd.read_csv("Cancer_Data.csv")
pd.set_option('display.max_columns', None)
```

[3]:
```
data
```

[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mea
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.24
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.18
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.20
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.25
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.18
...	...	...	...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.17
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.17
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.15
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.23
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.15

569 rows × 33 columns

Check data for nulls

[4]:
```
data.isnull().sum()
```

[4]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32                569
dtype: int64
```

Drop unnecessary column

[5]:
```
data.drop(['Unnamed: 32', 'id'], axis=1,inplace=True)
```

Check data types

[6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   diagnosis                569 non-null    object 
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), object(1)
memory usage: 137.9+ KB
```

Check data statistics

[7]:
```
data.describe()
```

[7]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimensi
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	56
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	

Split features and classes

For classes turn 'M' (malignant) and 'B' (benign) to integers: 0 for Benign, 1 for Malignant

[8]:
```
diagnosis=pd.get_dummies(data['diagnosis'],dtype=int)['M'].values
```

[9]:
```
features=data.drop('diagnosis',axis=1).values
```

[10]:
```
diagnosis.shape
```

[10]:
```
(569,)
```

[11]:
```
features.shape
```

[11]:
```
(569, 30)
```

[12]:
```
data.shape
```

[12]:
```
(569, 31)
```

Split train and test sets

[13]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(features, diagnosis, test_size=0.3)
```

Scale input data

[14]:
```
from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Create model and compile

[15]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(1, input_shape=(features.shape[1],), activation='sigmoid'))
```

[16]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1)                 31

=================================================================
Total params: 31
Trainable params: 31
Non-trainable params: 0
_________________________________________________________________
```

[17]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Train model (using early stopping based on validation loss)

[18]:
```
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                            patience=50)

r=model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs = 500, callbacks=[callback])
```

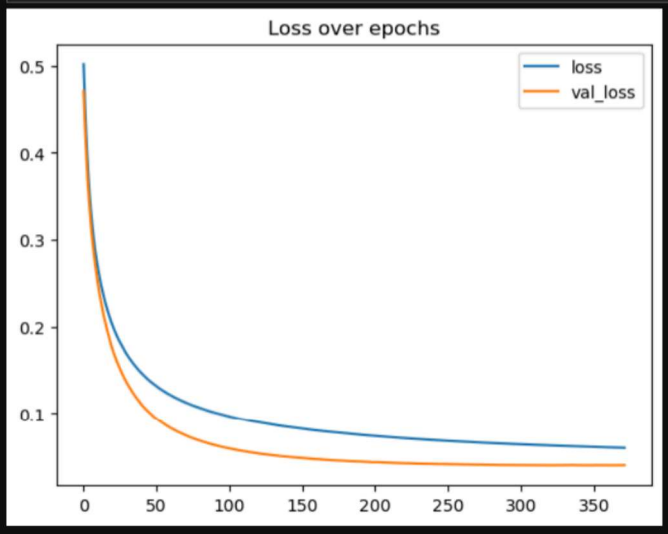
```
Epoch 1/500
13/13 [==============================] - 1s 15ms/step - loss: 0.5018 - accuracy: 0.7487 - val_loss: 0.4710 - val_accuracy: 0.7836
Epoch 2/500
13/13 [==============================] - 0s 2ms/step - loss: 0.4526 - accuracy: 0.8065 - val_loss: 0.4271 - val_accuracy: 0.8304
Epoch 3/500
13/13 [==============================] - 0s 3ms/step - loss: 0.4119 - accuracy: 0.8568 - val_loss: 0.3916 - val_accuracy: 0.8596
Epoch 4/500
13/13 [==============================] - 0s 2ms/step - loss: 0.3806 - accuracy: 0.8769 - val_loss: 0.3620 - val_accuracy: 0.8538
Epoch 5/500
13/13 [==============================] - 0s 3ms/step - loss: 0.3542 - accuracy: 0.8819 - val_loss: 0.3381 - val_accuracy: 0.8596
Epoch 6/500
13/13 [==============================] - 0s 3ms/step - loss: 0.3334 - accuracy: 0.8920 - val_loss: 0.3170 - val_accuracy: 0.8713
Epoch 7/500
13/13 [==============================] - 0s 2ms/step - loss: 0.3154 - accuracy: 0.9020 - val_loss: 0.2995 - val_accuracy: 0.8772
Epoch 8/500
13/13 [==============================] - 0s 3ms/step - loss: 0.3003 - accuracy: 0.9070 - val_loss: 0.2843 - val_accuracy: 0.8772
Epoch 9/500
13/13 [==============================] - 0s 3ms/step - loss: 0.2871 - accuracy: 0.9171 - val_loss: 0.2705 - val_accuracy: 0.8889
Epoch 10/500
13/13 [==============================] - 0s 3ms/step - loss: 0.2756 - accuracy: 0.9171 - val_loss: 0.2583 - val_accuracy: 0.9064
Epoch 11/500
13/13 [==============================] - 0s 3ms/step - loss: 0.2654 - accuracy: 0.9171 - val_loss: 0.2472 - val_accuracy: 0.9064
Epoch 12/500
13/13 [==============================] - 0s 3ms/step - loss: 0.2564 - accuracy: 0.9171 - val_loss: 0.2368 - val_accuracy: 0.9064
Epoch 13/500
13/13 [==============================] - 0s 3ms/step - loss: 0.2479 - accuracy: 0.9196 - val_loss: 0.2280 - val_accuracy: 0.9064
Epoch 14/500
13/13 [==============================] - 0s 3ms/step - loss: 0.2404 - accuracy: 0.9196 - val_loss: 0.2196 - val_accuracy: 0.9064
Epoch 15/500
13/13 [==============================] - 0s 3ms/step - loss: 0.2336 - accuracy: 0.9221 - val_loss: 0.2109 - val_accuracy: 0.9240
Epoch 16/500
13/13 [==============================] - 0s 3ms/step - loss: 0.2269 - accuracy: 0.9246 - val_loss: 0.2040 - val_accuracy: 0.9240
Epoch 17/500
13/13 [==============================] - 0s 3ms/step - loss: 0.2211 - accuracy: 0.9246 - val_loss: 0.1971 - val_accuracy: 0.9240
Epoch 18/500
13/13 [==============================] - 0s 3ms/step - loss: 0.2155 - accuracy: 0.9246 - val_loss: 0.1910 - val_accuracy: 0.9240
Epoch 19/500
13/13 [==============================] - 0s 3ms/step - loss: 0.2104 - accuracy: 0.9246 - val_loss: 0.1844 - val_accuracy: 0.9240
Epoch 20/500
13/13 [==============================] - 0s 3ms/step - loss: 0.2054 - accuracy: 0.9246 - val_loss: 0.1783 - val_accuracy: 0.9357
Epoch 21/500
13/13 [==============================] - 0s 3ms/step - loss: 0.2009 - accuracy: 0.9271 - val_loss: 0.1730 - val_accuracy: 0.9357
Epoch 22/500
13/13 [==============================] - 0s 3ms/step - loss: 0.1966 - accuracy: 0.9296 - val_loss: 0.1680 - val_accuracy: 0.9357
Epoch 23/500
13/13 [==============================] - 0s 3ms/step - loss: 0.1926 - accuracy: 0.9296 - val_loss: 0.1634 - val_accuracy: 0.9357
Epoch 24/500
13/13 [==============================] - 0s 3ms/step - loss: 0.1888 - accuracy: 0.9296 - val_loss: 0.1592 - val_accuracy: 0.9357
Epoch 25/500
13/13 [==============================] - 0s 3ms/step - loss: 0.1852 - accuracy: 0.9296 - val_loss: 0.1552 - val_accuracy: 0.9415
Epoch 26/500
13/13 [==============================] - 0s 3ms/step - loss: 0.1821 - accuracy: 0.9322 - val_loss: 0.1511 - val_accuracy: 0.9474
Epoch 27/500
```

```
Epoch 366/500
13/13 [==============================] - 0s 3ms/step - loss: 0.0600 - accuracy: 0.9899 - val_loss: 0.0396 - val_accuracy: 0.9825
Epoch 367/500
13/13 [==============================] - 0s 3ms/step - loss: 0.0601 - accuracy: 0.9899 - val_loss: 0.0395 - val_accuracy: 0.9825
Epoch 368/500
13/13 [==============================] - 0s 3ms/step - loss: 0.0599 - accuracy: 0.9899 - val_loss: 0.0395 - val_accuracy: 0.9825
Epoch 369/500
13/13 [==============================] - 0s 3ms/step - loss: 0.0599 - accuracy: 0.9899 - val_loss: 0.0396 - val_accuracy: 0.9825
Epoch 370/500
13/13 [==============================] - 0s 3ms/step - loss: 0.0599 - accuracy: 0.9899 - val_loss: 0.0396 - val_accuracy: 0.9825
Epoch 371/500
13/13 [==============================] - 0s 3ms/step - loss: 0.0598 - accuracy: 0.9899 - val_loss: 0.0396 - val_accuracy: 0.9825
Epoch 372/500
13/13 [==============================] - 0s 3ms/step - loss: 0.0597 - accuracy: 0.9899 - val_loss: 0.0396 - val_accuracy: 0.9825
```

Plot training and validation loss

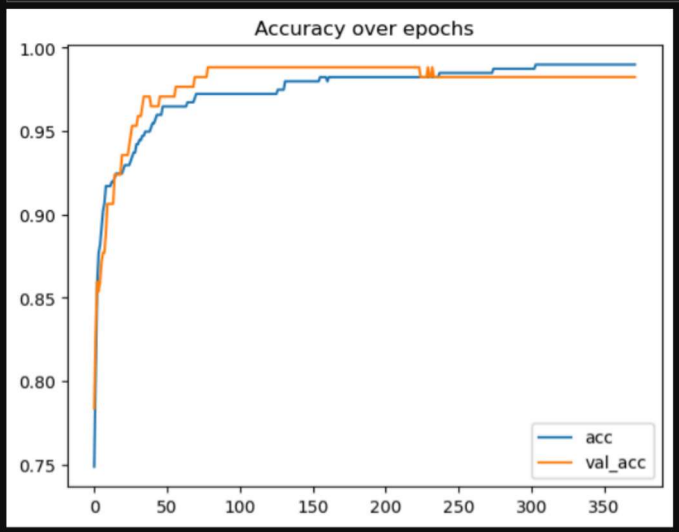
[19]:
```
import matplotlib.pyplot as plt

plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.legend()
plt.title('Loss over epochs')
plt.show()
```



PLot training and validation accuracy

[20]:
```
plt.plot(r.history['accuracy'], label='acc')
plt.plot(r.history['val_accuracy'], label='val_acc')
plt.legend()
plt.title('Accuracy over epochs')
plt.show()
```



Create and plot Confusion Matrix

[21]:
```
predictions = np.rint(model.predict(X_test)).astype(int)

conf_matrix = tf.math.confusion_matrix(y_test, predictions)
```

```
6/6 [==============================] - 0s 807us/step
```

[22]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, predictions, labels=[0, 1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=['Benign', 'Malignant'])
disp.plot()
```

[22]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1798005dd50>
```