Breast Cancer Detection with a simple Neural Network

Dependencies:

In [42]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.datasets
from sklearn.model_selection import train_test_split
```

Data Collection and Processing

In [43]:
```
breast_cancer_dataset = sklearn.datasets.load_breast_cancer()
```

- we are using the Breast Cancer Dataset from SKLearn

In [44]:
```
print(breast_cancer_dataset)
```

```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]), 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]), 'frame': None, 'target_names': a
rray(['malignant', 'benign'], dtype='<U9'), 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer
wisconsin (diagnostic) dataset\n--------------------------------------------\n\n**Data Set Charact
eristics:**\n\n    :Number of Instances: 569\n\n    :Number of Attributes: 30 numeric, predictive
attributes and the class\n\n    :Attribute Information:\n        - radius (mean of distances from
center to points on the perimeter)\n        - texture (standard deviation of gray-scale values)\n
- perimeter\n        - area\n        - smoothness (local variation in radius lengths)\n        - c
ompactness (perimeter^2 / area - 1.0)\n        - concavity (severity of concave portions of the co
ntour)\n        - concave points (number of concave portions of the contour)\n        - symmetry\n
- fractal dimension ("coastline approximation" - 1)\n\n        The mean, standard error, and "wors
t" or largest (mean of the three\n        worst/largest values) of these features were computed fo
r each image,\n        resulting in 30 features.  For instance, field 0 is Mean Radius, field\n
10 is Radius SE, field 20 is Worst Radius.\n\n        - class:\n                - WDBC-Malignant\n
- WDBC-Benign\n\n    :Summary Statistics:\n\n    ===================================== ====== ====
==\n                                           Min    Max\n    ===================================
== ====== ======\n    radius (mean):                        6.981  28.11\n    texture (mean):
9.71   39.28\n    perimeter (mean):                     43.79  188.5\n    area (mean):
143.5  2501.0\n    smoothness (mean):                    0.053  0.163\n    compactness (mean):
0.019  0.345\n    concavity (mean):                     0.0    0.427\n    concave points (mean):
0.0    0.201\n    symmetry (mean):                      0.106  0.304\n    fractal dimension (mea
n):            0.05   0.097\n    radius (standard error):             0.112  2.873\n    texture
(standard error):            0.36   4.885\n    perimeter (standard error):          0.757  21.98
\n    area (standard error):               6.802  542.2\n    smoothness (standard error):
0.002  0.031\n    compactness (standard error):         0.002  0.135\n    concavity (standard erro
r):           0.0    0.396\n    concave points (standard error):      0.0    0.053\n    symmetry
(standard error):            0.008  0.079\n    fractal dimension (standard error):   0.001  0.03\n
radius (worst):                       7.93   36.04\n    texture (worst):                      12.0
2  49.54\n    perimeter (worst):                    50.41  251.2\n    area (worst):
185.2  4254.0\n    smoothness (worst):                   0.071  0.223\n    compactness (worst):
0.027  1.058\n    concavity (worst):                    0.0    1.252\n    concave points (worst):
```

```
0.0    0.291\n    symmetry (worst):                     0.156  0.664\n    fractal dimension (wors
t):           0.055  0.208\n    ===================================== ====== ======\n\n    :Missi
ng Attribute Values: None\n\n    :Class Distribution: 212 - Malignant, 357 - Benign\n\n    :Creato
r:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian\n\n    :Donor: Nick Street\n\n
:Date: November, 1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.\n
https://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image of a fine needle\naspirate
(FNA) of a breast mass.  They describe\ncharacteristics of the cell nuclei present in the image.\n
\nSeparating plane described above was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Ben
nett, "Decision Tree\nConstruction Via Linear Programming." Proceedings of the 4th\nMidwest Artifi
cial Intelligence and Cognitive Science Society,\npp. 97-101, 1992], a classification method which
uses linear\nprogramming to construct a decision tree.  Relevant features\nwere selected using an
exhaustive search in the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual linear pr
ogram used to obtain the separating plane\nin the 3-dimensional space is that described in:\n[K.
P. Bennett and O. L. Mangasarian: "Robust Linear\nProgramming Discrimination of Two Linearly Insep
arable Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis database is also availab
le through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-learn/WD
BC/\n\n.. topic:: References\n\n   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear featu
re extraction \n     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on \n
Electronic Imaging: Science and Technology, volume 1905, pages 861-870,\n     San Jose, CA, 199
3.\n   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and \n     progno
sis via linear programming. Operations Research, 43(4), pages 570-577, \n     July-August 1995.\n
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques\n     to diagnose b
reast cancer from fine-needle aspirates. Cancer Letters 77 (1994) \n     163-171.', 'feature_name
s': array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23'), 'filename': 'breast_cancer.cs
v', 'data_module': 'sklearn.datasets.data'}
```

In [45]:
```
data_frame = pd.DataFrame(breast_cancer_dataset.data, columns = breast_cancer_dataset.feature_names
```

In [46]:
```
data_frame.head()
```

Out[46]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	w tex
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	1
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	2
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	2
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	2
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	1

5 rows × 30 columns

In [47]:
```
data_frame['label'] = breast_cancer_dataset.target
```

- we need to add the 'target' column to the data frame

In [48]:
```
data_frame.tail()
```

Out[48]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37

5 rows × 31 columns

In [50]:
```
data_frame.shape
```

Out[50]: (569, 31)

In [51]:
```
data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  label                    569 non-null    int32  
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

In [52]:
```
data_frame.isnull().sum()
```

Out[52]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
label                      0
dtype: int64
```

- we have to check for any missing values in the data

In [53]:
```
data_frame.describe()
```

Out[53]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	c
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	5(
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	

8 rows × 31 columns

In [54]:
```
data_frame['label'].value_counts()
```

Out[54]:
```
1    357
0    212
Name: label, dtype: int64
```

In [55]:
```
data_frame.groupby('label').mean()
```

Out[55]:

label	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension
0	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188	0.160775	0.087990	0.192909	0.062680
1	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085	0.046058	0.025717	0.174186	0.062867

2 rows × 30 columns

1 = Beign (not harmful)

0 = Malignant (harmful)

In [56]:
```
data_frame.groupby('label').mean()
```

Out[56]:

label	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension
0	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188	0.160775	0.087990	0.192909	0.062680
1	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085	0.046058	0.025717	0.174186	0.062867

2 rows × 30 columns

Seperating the Features and Target

In [57]:
```
X = data_frame.drop(columns='label', axis=1)
Y = data_frame['label']
```

In [58]:
```
print(X)
```

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0          17.99         10.38          122.80     1001.0          0.11840   
1          20.57         17.77          132.90     1326.0          0.08474   
2          19.69         21.25          130.00     1203.0          0.10960   
3          11.42         20.38           77.58      386.1          0.14250   
4          20.29         14.34          135.10     1297.0          0.10030   
..           ...           ...             ...        ...              ...   
564        21.56         22.39          142.00     1479.0          0.11100   
565        20.13         28.25          131.20     1261.0          0.09780   
566        16.60         28.08          108.30      858.1          0.08455   
567        20.60         29.33          140.10     1265.0          0.11780   
568         7.76         24.54           47.92      181.0          0.05263   

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0             0.27760         0.30010              0.14710         0.2419   
1             0.07864         0.08690              0.07017         0.1812   
2             0.15990         0.19740              0.12790         0.2069   
3             0.28390         0.24140              0.10520         0.2597   
4             0.13280         0.19800              0.10430         0.1809   
..                ...             ...                  ...            ...   
564           0.11590         0.24390              0.13890         0.1726   
565           0.10340         0.14400              0.09791         0.1752   
566           0.10230         0.09251              0.05302         0.1590   
567           0.27700         0.35140              0.15200         0.2397   
568           0.04362         0.00000              0.00000         0.1587   

     mean fractal dimension  ...  worst radius  worst texture  \
0                   0.07871  ...        25.380          17.33   
1                   0.05667  ...        24.990          23.41   
2                   0.05999  ...        23.570          25.53   
3                   0.09744  ...        14.910          26.50   
4                   0.05883  ...        22.540          16.67   
..                      ...  ...           ...            ...   
564                 0.05623  ...        25.450          26.40   
565                 0.05533  ...        23.690          38.25   
566                 0.05648  ...        18.980          34.12   
567                 0.07016  ...        25.740          39.42   
568                 0.05884  ...         9.456          30.37   

     worst perimeter  worst area  worst smoothness  worst compactness  \
0             184.60      2019.0           0.16220            0.66560   
1             158.80      1956.0           0.12380            0.18660   
2             152.50      1709.0           0.14440            0.42450   
3              98.87       567.7           0.20980            0.86630   
4             152.20      1575.0           0.13740            0.20500   
..               ...         ...               ...                ...   
564           166.10      2027.0           0.14100            0.21130   
565           155.00      1731.0           0.11660            0.19220   
566           126.70      1124.0           0.11390            0.30940   
567           184.60      1821.0           0.16500            0.86810   
568            59.16       268.6           0.08996            0.06444   

     worst concavity  worst concave points  worst symmetry  \
0             0.7119                0.2654          0.4601   
1             0.2416                0.1860          0.2750   
2             0.4504                0.2430          0.3613   
3             0.6869                0.2575          0.6638   
4             0.4000                0.1625          0.2364   
..               ...                   ...             ...   
564           0.4107                0.2216          0.2060   
565           0.3215                0.1628          0.2572   
566           0.3403                0.1418          0.2218   
567           0.9387                0.2650          0.4087   
568           0.0000                0.0000          0.2871   

     worst fractal dimension  
0                    0.11890  
1                    0.08902  
```

```
2                   0.08758
3                   0.17300
4                   0.07678
..                      ...
564                 0.07115
565                 0.06637
566                 0.07820
567                 0.12400
568                 0.07039

[569 rows x 30 columns]
```

In [59]:
```
print(Y)
```

```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: label, Length: 569, dtype: int32
```

---

we need to split the data into:

- Traing Data
- Testing Data

In [60]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

In [61]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(569, 30) (455, 30) (114, 30)
```

In [62]:
```
from sklearn.preprocessing import StandardScaler
```

In [63]:
```
scaler = StandardScaler()

X_train_std = scaler.fit_transform(X_train)

X_test_std = scaler.transform(X_test)
```

- now we are standardizing the data

Building the Neural Network

Dependencies

In [64]:
```
import tensorflow as tf
tf.random.set_seed(3)
from tensorflow import keras
```

In [65]:
```
model = keras.Sequential([
                          keras.layers.Flatten(input_shape=(30,)),
                          keras.layers.Dense(20, activation='relu'),
                          keras.layers.Dense(2, activation='sigmoid')
])
```

In [66]:
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

- now we need to train the neural network

In [67]:
```
history = model.fit(X_train_std, Y_train, validation_split=0.1, epochs=10)
```

```
Epoch 1/10
13/13 [==============================] - 0s 12ms/step - loss: 0.4279 - accuracy: 0.8289 - val_los
s: 0.2574 - val_accuracy: 0.9565
Epoch 2/10
13/13 [==============================] - 0s 4ms/step - loss: 0.2983 - accuracy: 0.8900 - val_loss:
0.1759 - val_accuracy: 0.9783
Epoch 3/10
13/13 [==============================] - 0s 4ms/step - loss: 0.2351 - accuracy: 0.9218 - val_loss:
0.1393 - val_accuracy: 0.9783
Epoch 4/10
13/13 [==============================] - 0s 4ms/step - loss: 0.2018 - accuracy: 0.9291 - val_loss:
0.1186 - val_accuracy: 0.9783
Epoch 5/10
13/13 [==============================] - 0s 3ms/step - loss: 0.1787 - accuracy: 0.9364 - val_loss:
0.1044 - val_accuracy: 0.9783
Epoch 6/10
13/13 [==============================] - 0s 4ms/step - loss: 0.1612 - accuracy: 0.9413 - val_loss:
0.0950 - val_accuracy: 0.9783
Epoch 7/10
13/13 [==============================] - 0s 3ms/step - loss: 0.1468 - accuracy: 0.9438 - val_loss:
0.0874 - val_accuracy: 0.9783
Epoch 8/10
13/13 [==============================] - 0s 3ms/step - loss: 0.1350 - accuracy: 0.9462 - val_loss:
0.0815 - val_accuracy: 0.9783
Epoch 9/10
13/13 [==============================] - 0s 3ms/step - loss: 0.1249 - accuracy: 0.9535 - val_loss:
0.0764 - val_accuracy: 0.9783
Epoch 10/10
13/13 [==============================] - 0s 4ms/step - loss: 0.1160 - accuracy: 0.9633 - val_loss:
0.0732 - val_accuracy: 0.9783
```

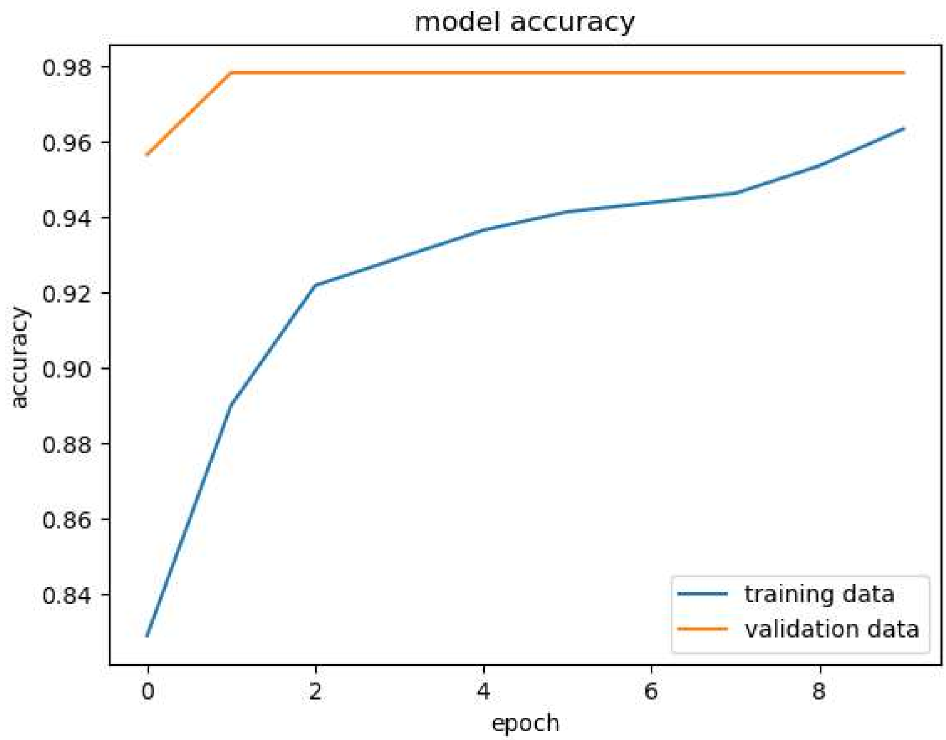
Visualizing the Accuracy and Loss of the Model

In [68]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')

plt.legend(['training data', 'validation data'], loc = 'lower right')
```

Out[68]: <matplotlib.legend.Legend at 0x287685808e0>

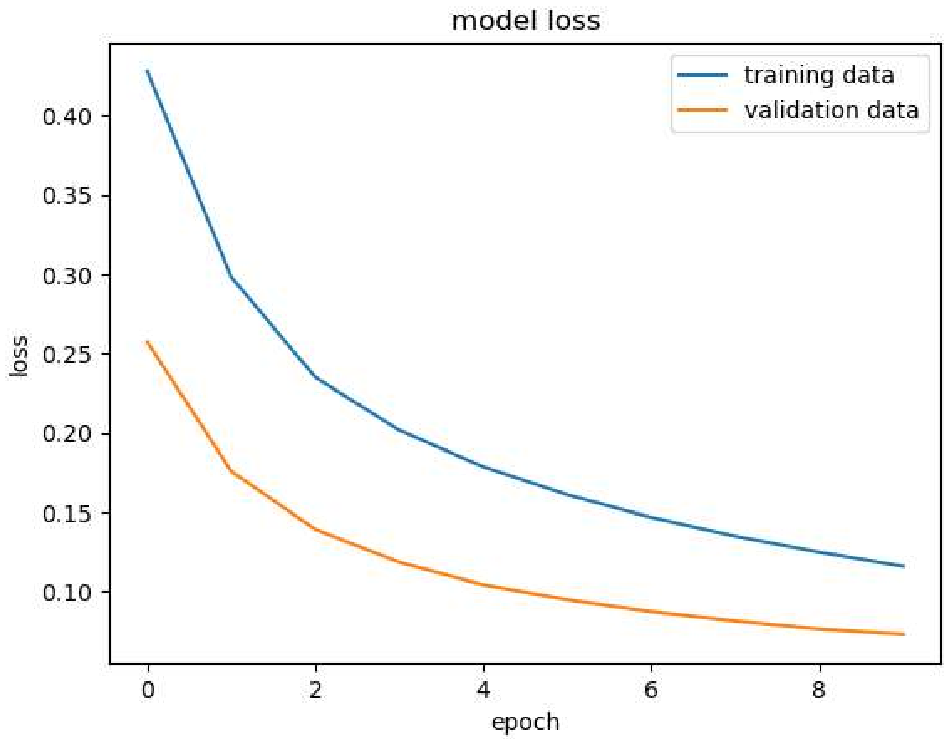


In [69]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')

plt.legend(['training data', 'validation data'], loc = 'upper right')
```

Out[69]:
```
<matplotlib.legend.Legend at 0x287685bf3d0>
```



Accuracy of the Model on Test Data

In [70]:
```
loss, accuracy = model.evaluate(X_test_std, Y_test)
print(accuracy)
```

```
4/4 [==============================] - 0s 3ms/step - loss: 0.1007 - accuracy: 0.9649
0.9649122953414917
```

In [71]:
```
print(X_test_std.shape)
print(X_test_std[0])
```

```
(114, 30)
[-0.04462793 -1.41612656 -0.05903514 -0.16234067  2.0202457  -0.11323672
  0.18500609  0.47102419  0.63336386  0.26335737  0.53209124  2.62763999
  0.62351167  0.11405261  1.01246781  0.41126289  0.63848593  2.88971815
 -0.41675911  0.74270853 -0.32983699 -1.67435595 -0.36854552 -0.38767294
  0.32655007 -0.74858917 -0.54689089 -0.18278004 -1.23064515 -0.6268286 ]
```

In [72]:
```
Y_pred = model.predict(X_test_std)
```

```
4/4 [==============================] - 0s 1ms/step
```

In [73]:
```
print(Y_pred.shape)
print(Y_pred[0])
```

```
(114, 2)
[0.17592672 0.5889425 ]
```

In [74]:
```
print(X_test_std)
```

```
[[-0.04462793 -1.41612656 -0.05903514 ... -0.18278004 -1.23064515
  -0.6268286 ]
 [ 0.24583601 -0.06219797  0.21802678 ...  0.54129749  0.11047691
   0.0483572 ]
 [-1.26115925 -0.29051645 -1.26499659 ... -1.35138617  0.269338
  -0.28231213]
 ...
 [ 0.72709489  0.45836817  0.75277276 ...  1.46701686  1.19909344
   0.65319961]
 [ 0.25437907  1.33054477  0.15659489 ... -1.29043534 -2.22561725
  -1.59557344]
 [ 0.84100232 -0.06676434  0.8929529  ...  2.15137705  0.35629355
   0.37459546]]
```

In [75]:
```
print(Y_pred)
```

```
 [0.26602924 0.7364886 ]
 [0.02009718 0.9580364 ]
 [0.03137774 0.8677221 ]
 [0.9997541  0.11960152]
 [0.8577166  0.37892827]
 [0.08701973 0.9970844 ]
 [0.08464034 0.84557164]
 [0.83521426 0.21721838]
 [0.08596357 0.8846782 ]
 [0.01210442 0.92110646]
 [0.02697744 0.95870054]
 [0.999794   0.17439036]
 [0.9784517  0.28472006]
 [0.09358877 0.8261178 ]
 [0.89373046 0.4091013 ]
 [0.66597706 0.77793026]
 [0.06702574 0.9221596 ]
 [0.01634618 0.9161694 ]
 [0.75439984 0.47034293]
 [0.10702883 0.94944364]
```

model.predict() gives the prediction probability of each class for that data point

---

In [76]:
```
# ARGMAX Function

my_list = [0.25, 0.56]

index_of_max_value = np.argmax(my_list)
print(my_list)
print(index_of_max_value)
```

```
[0.25, 0.56]
1
```

In [77]:
```
Y_pred_labels = [np.argmax(i) for i in Y_pred]
print(Y_pred_labels)
```

```
[1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0,
0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0,
1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
```

Building the Prediction System

In [78]:
```
input_data = (11.76,21.6,74.72,427.9,0.08637,0.04966,0.01657,0.01115,0.1495,0.05888,0.4062,1.21,2.6
```

Replace the values in 'input_data' to values you want a prediction for

In [79]:
```
# change the input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array as we are predicting for one data point
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardizing the input data
input_data_std = scaler.transform(input_data_reshaped)

prediction = model.predict(input_data_std)
print(prediction)

prediction_label = [np.argmax(prediction)]
print(prediction_label)

if(prediction_label[0] == 0):
  print('The tumor is Malignant (harmful)')

else:
  print('The tumor is Benign (harmless)')
```

```
1/1 [==============================] - 0s 30ms/step
[[0.07666554 0.9476789 ]]
[1]
The tumor is Benign (harmless)

C:\Users\Arindal Char\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does not hav
e valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

In [ ]: