cancer

May 9, 2018

Steps

```
Import Libraries
Load dataset
Build Model
Train Model
Find Accuracy on Validation Set
Predict Values
```

```
In [1]: import numpy as np
        import pandas as pd

        from keras.models import Sequential
        from keras.layers import Dense,Activation,Layer,Lambda

        from sklearn.cross_validation import train_test_split
```

```
/home/nikhil/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversio
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
/home/nikhil/anaconda3/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWa
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
In [25]: dataset=pd.read_csv("input/data.csv")
         dataset.head()
```

```
Out[25]:          id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
         0    842302         M        17.99         10.38          122.80     1001.0
         1    842517         M        20.57         17.77          132.90     1326.0
         2  84300903         M        19.69         21.25          130.00     1203.0
         3  84348301         M        11.42         20.38           77.58      386.1
         4  84358402         M        20.29         14.34          135.10     1297.0

            smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
         0          0.11840           0.27760          0.3001              0.14710
         1          0.08474           0.07864          0.0869              0.07017
         2          0.10960           0.15990          0.1974              0.12790
```

```
3          0.14250          0.28390         0.2414               0.10520
4          0.10030          0.13280         0.1980               0.10430

      ...         texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0     ...                 17.33           184.60      2019.0            0.1622
1     ...                 23.41           158.80      1956.0            0.1238
2     ...                 25.53           152.50      1709.0            0.1444
3     ...                 26.50            98.87       567.7            0.2098
4     ...                 16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

In [3]:
```
dataset=dataset.drop(["id","Unnamed: 32"],axis=1)
dataset.head()
```

Out[3]:
```
  diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0         M        17.99         10.38          122.80     1001.0
1         M        20.57         17.77          132.90     1326.0
2         M        19.69         21.25          130.00     1203.0
3         M        11.42         20.38           77.58      386.1
4         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   symmetry_mean          ...             radius_worst  texture_worst  \
0         0.2419          ...                    25.38          17.33
1         0.1812          ...                    24.99          23.41
2         0.2069          ...                    23.57          25.53
3         0.2597          ...                    14.91          26.50
```

```
4         0.1809            ...                   22.54          16.67

   perimeter_worst  area_worst  smoothness_worst  compactness_worst  \
0           184.60      2019.0            0.1622             0.6656
1           158.80      1956.0            0.1238             0.1866
2           152.50      1709.0            0.1444             0.4245
3            98.87       567.7            0.2098             0.8663
4           152.20      1575.0            0.1374             0.2050

   concavity_worst  concave points_worst  symmetry_worst  \
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

   fractal_dimension_worst
0                  0.11890
1                  0.08902
2                  0.08758
3                  0.17300
4                  0.07678

[5 rows x 31 columns]
```

In [4]:
```
pd.isnull(dataset).sum()
```

Out[4]:
```
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
dtype: int64
```

first map diagnosis to integer value.

```
In [5]: def mapping(data,feature):
            featureMap=dict()
            count=0
            for i in sorted(data[feature].unique(),reverse=True):
                featureMap[i]=count
                count=count+1
            data[feature]=data[feature].map(featureMap)
            return data
```

```
In [6]: dataset=mapping(dataset,feature="diagnosis")
```

Malignant is mapped to 0, Benign is mapped to 1

```
In [7]: #divide dataset into x(input) and y(output)
        X=dataset.drop(["diagnosis"],axis=1)
        y=dataset["diagnosis"]
```

```
In [8]: #divide dataset into training set, cross validation set, and test set
        trainX, testX, trainY, testY = train_test_split(X, y, test_size=0.2, random_state=42)
        trainX, valX, trainY, valY = train_test_split(trainX, trainY, test_size=0.2, random_stat
```

```
In [9]: def getModel(arr):
            model=Sequential()
            for i in range(len(arr)):
                if i!=0 and i!=len(arr)-1:
                    if i==1:
                        model.add(Dense(arr[i],input_dim=arr[0],kernel_initializer='normal', act
                    else:
                        model.add(Dense(arr[i],activation='relu'))
            model.add(Dense(arr[-1],kernel_initializer='normal',activation="sigmoid"))
            model.compile(loss="binary_crossentropy",optimizer='rmsprop',metrics=['accuracy'])
            return model
```

Above I've defined a function that will return us a neural network model. We will pass an array of integer to define the no. of hidden units in each layer. First layer will have same no.

of units as input dimension. Each subsequent layer will have the units set in the array passed. model=Sequential() will give us a model. model.add() is used to add a layer to the model. We will set activation function for each layer. Since we need binary classification, we'll use sigmoid activation in the output layer. At the end we will compile the build model, with loss function, optimizer and the metrics we want when we will evaluate the model.

Now we will define different models so that we can test each of them on validation set and check the accuracy.

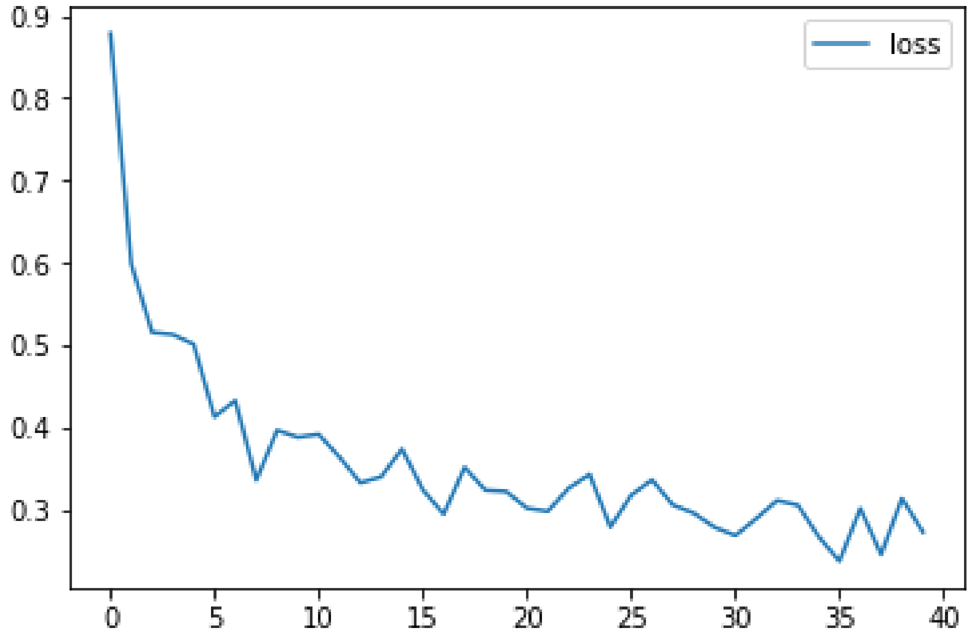
```
Firstly, we'll use a small model which contains 3 layers with hidden units 30, 50 and 1.
Then we'll use a wider network which will also have 3 layers but more hidden units in the hidden
Then we'll use a deeper network which will have 5 layers.
```

```
In [10]: firstModel=getModel([30,50,1])
```

Now we will create a callback function which will plot loss on each epoch end. we will override on_epch_end() method to plot the graph

```
In [11]: import keras
         import matplotlib.pyplot as plt
         from IPython.display import clear_output
         class PlotLosses(keras.callbacks.Callback):
             def on_train_begin(self, logs={}):
                 self.i = 0
                 self.x = []
                 self.losses = []
                 self.val_losses = []

                 self.fig = plt.figure()

                 self.logs = []

             def on_epoch_end(self, epoch, logs={}):

                 self.logs.append(logs)
                 self.x.append(self.i)
                 self.losses.append(logs.get('loss'))
                 self.val_losses.append(logs.get('val_loss'))
                 self.i += 1

                 clear_output(wait=True)
                 plt.plot(self.x, self.losses, label="loss")
                 plt.legend()
                 plt.show();

         plot_losses = PlotLosses()
```

```
In [12]: firstModel.fit(np.array(trainX),np.array(trainY),epochs=40,callbacks=[plot_losses])
```




```
Out[12]: <keras.callbacks.History at 0x7f679294ab00>
```

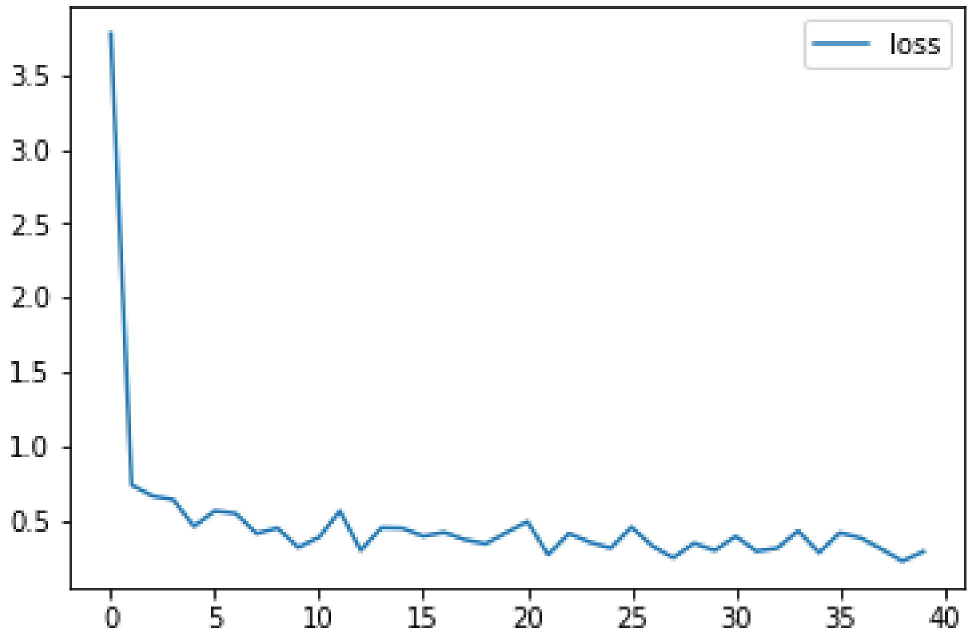
```
In [13]: scores=firstModel.evaluate(np.array(valX),np.array(valY))
```

```
91/91 [==============================] - 0s 225us/step
```

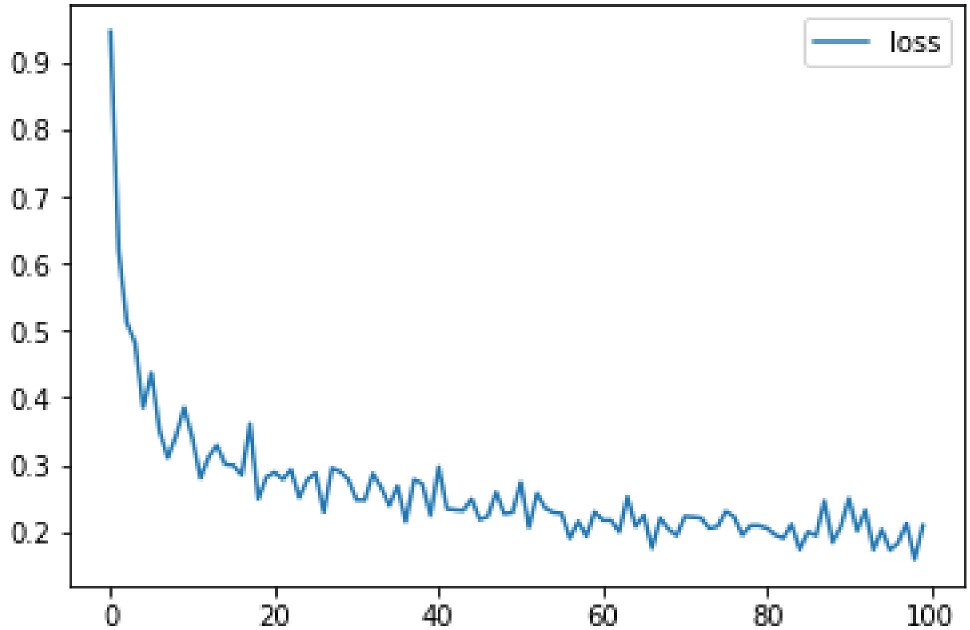
```
In [14]: print("Loss:",scores[0])
         print("Accuracy",scores[1]*100)
```

```
Loss: 0.2656468202124585
Accuracy 91.20879140529003
```

```
In [15]: secondModel=getModel([30,100,1])
         secondModel.fit(np.array(trainX),np.array(trainY),epochs=40,callbacks=[plot_losses])
```




```
Out[15]: <keras.callbacks.History at 0x7f6777dec5c0>

In [16]: scores2=secondModel.evaluate(np.array(valX),np.array(valY))

91/91 [==============================] - 0s 286us/step

In [17]: print(scores2)

[0.5112767815589905, 0.8461538435338618]

In [18]: thirdModel=getModel([30,50,70,40,1])

In [19]: thirdModel.fit(np.array(trainX),np.array(trainY),epochs=100,callbacks=[plot_losses])
```




```
Out[19]: <keras.callbacks.History at 0x7f677ceddeb8>

In [20]: scores3=thirdModel.evaluate(np.array(valX),np.array(valY))

91/91 [==============================] - 0s 428us/step

In [21]: print(scores3)

[0.264685516501521, 0.9340659294809613]

In [22]: predY=firstModel.predict(np.array(testX))
         predY=np.round(predY).astype(int).reshape(1,-1)[0]

In [23]: from sklearn.metrics import confusion_matrix
         m=confusion_matrix(predY,testY)
         tn, fn, fp, tp=confusion_matrix(predY,testY).ravel()
         m=pd.crosstab(predY,testY)
         print("Confusion matrix")
         print(m)

Confusion matrix
diagnosis   0   1
row_0
0          33   0
1          10  71
```

In [24]:
```
sens=tp/(tp+fn)
spec=tn/(tn+fp)
print("Senstivity:",sens)
print("Specificity:",spec)
```

```
Senstivity: 1.0
Specificity: 0.7674418604651163
```