Towards
Data Science

A Simple Breast Cancer Classifier using Neural Network

Spoiler — No bullshit, to the point

Dhrumil Patel Follow
May 7 · 10 min read ★

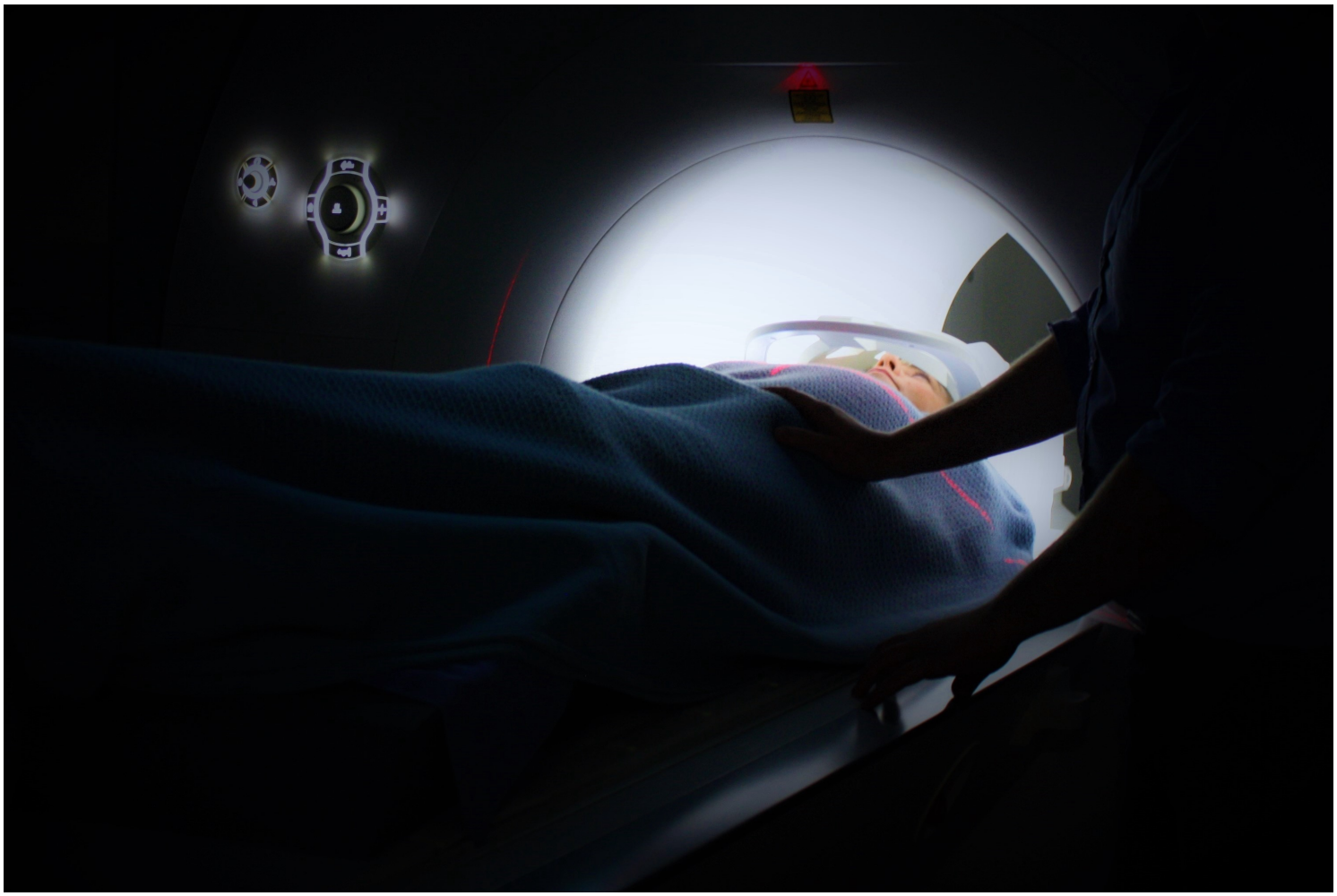

Photo by Ken Treloar on Unsplash

I won't repeat cliche statements like "… deep learning is the next big thing." No. If

you are here reading this article, you most certainly know what Deep Learning or Neural Network is and how it is going to evolve. Let's cut to the chest and build a classifier using a Neural Network that will classify the tumor as ***benign*** *(safe)* or ***malignant*** *(risk)* based on several features provided in the data.

> Time is different now. We are on the edge of a technological revolution that might change the way we work, live, or relate to each other. We yet don't know how it will unfold, but one thing is rational; as a response to it, we must not forget that humans first.
>
> — That's me

Understanding Data — Why and What

The reason I wrote the statement above is we are mostly everyone is following the hype. No thoughts, no purpose just doing things. Machine Learning isn't just building models. You must have a purpose behind building it. Here is ***why*** we are taking the Breast Cancer data. According to the World Cancer Research Fund, Breast Cancer was the most common cancer in women all over the world in 2018. The number of Breast Cancer cases diagnosed in 2018 was **2,088,849** and it is **25.4% of the total number of new cases** diagnosed. I think these numbers are fair enough to convince you to work on a solution to detect Breast Cancer in the early stage. A purpose. Isn't it? All lives matter after all.

Alright, Let's move to the ***what*** part. You'll find the data on the UCI Machine Learning repository, which is in `.data` format, but if you are looking for a `.csv`, here is the link. Below shown are the features in our data set. I am sure, just like me, you won't understand a single one. And that's totally okay. The simple interpretation is: if the value is near 1 that's a benign tumor and if close to 10, you guessed it right, it's malignant. Pretty easy, huh?

```
Sample code number          : Id number (Not going to use)
Clump Thickness             : 1-10
```

```
Uniformity of Cell Size     : 1-10
Uniformity of Cell Shape    : 1-10
Marginal Adhesion           : 1-10
Single Epithelial Cell Size : 1-10
Bare Nuclei                 : 1-10
Bland Chromatin             : 1-10
Normal Nucleoli             : 1-10
Mitoses                     : 1-10
Class                       : 2 for benign, 4 for malignant
                             (I know, we will change it to 0 and 1)
```

There is a total of 699 instances. I mean entries/rows. Out of which, 16 has one missing value. We will handle it in pre-processing. Now that you know what data is and what we are seeking as an output — whether the tumor is benign or malignant — let's proceed to build the classifier. Shall we?

Data Pre-processing

I know, I know, it's not the most exciting part but — ugh — it is mandatory. Luckily the data is fairly clean and all values are in numeric, thanks to the amazing UCI repository, but as we discussed, we will fill those 16 missing values and will convert the class to 0 for benign and 1 for malignant for the sake of simplicity *shrug*

Note — I'll be using my local machine for pre-processing and Google Colab for training the classifier. Why? Free GPU, so why not. If you want to use it too, check out Getting Started With Google Colab by fellow Data Science Communicator Anne. It's easy and Anne made it even easier to understand. Thanks, Anne.

Another Note — I'll provide entire .ipynb and .py files at the end of the article. Let's check your patience too while we are at it.

Step 1 — Missing Values

We can handle missing values in several ways. For example, replacing it with the most occurred value in the column or replacing with the average of the column. Here, if we look into the data set, we'll find that 1 and 10 are the most occurred values, we have two, so a wise decision to fill those missing values is to replace it with the *mean* of the entire column.

```
1  import pandas as pd
2  from google.colab import files
3  file = files.upload()
4  data = pd.read_csv("breastCancer.csv", header = None)
```

UploadingDataToColab.py hosted with ❤ by GitHub view raw

Uploading data to Google Colab

```
df = pd.DataFrame(data)
df[6] = df[6].replace('?',0)
df[6] = df[6].astype(int)
print(df.dtypes)

mean = int(df[6].mean())
df[6] = df[6].replace(0,mean)
print(df)
```

MissingValues1.py hosted with ❤ by **GitHub** **view raw**

Cleaning feat. replace, mean and astype

Data type and Data. Look at the 23rd entry in the 6th column. It's changed.

Here, the 6th column, in which missing values are, contains `?` as a placer for missing values. We can't find mean with that, so first I'll replace it with zeros, then change its type to integer and then apply mean function. After that, we will just replace those zeros with the mean of the column.

Step 2 — Zero and One

You simply use the `.replace` to replace `2` with `0` and `4` with `1` . I am doing this because we have a habit of seeing classification outputs as either 0 or 1. Any value other than that will surely be confusing to many people. And it's just a line of code so why not?

```
df[10] = df[10].replace(2,0).replace(4,1)
print(df)
```

zeroOne.py hosted with ❤ by GitHub view raw

Class equals to 0 and 1

The 10th column

Step 3 — Splitting Data

Now we will split the data into 4 categories that I am sure you are aware of. If not, read on. In Machine Learning, the data is mainly divided into two parts — Training and Testing (the third split is *validation*, but you don't have to care about that right now). Training data is for training our algorithm and Testing data is to check how well our algorithm performs. The split ratio between the train and test data is usually around 80–20. So we are going for 80–20 as well.

```
X : X_train : All the features for training (excluding class)
Y : Y_train : Actual output (Class)
```

```
from sklearn.model_selection import train_test_split

X = df.iloc[:, 1:10]
Y = df.iloc[:, 10]

```

```
6  X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state =
```

spliting.py hosted with ❤ by GitHub view raw

Step 4 — Feature Scaling

For those of you who don't know what feature scaling is, it is a method that will standardize the range of features of data. In our data, the range is 1–10, therefore, at an elementary level, even if we don't use feature scaling, it won't make a huge difference. Reason? Because the range is small. So the reason for using it? Scaling features will speed up gradient descent, which we are using as an optimizer for our network[2]. We'll see that later on.

```
1  from sklearn.preprocessing import StandardScaler
2  scaler = StandardScaler()
3  X_train = scaler.fit_transform(X_train)
4  X_test = scaler.transform(X_test)
```

featureScaling.py hosted with ❤ by GitHub view raw

There are four common methods for feature scaling, out of which we are using **Standardisation.** Here is how our data looks now.

. . .

Building the Classifier

Phew! At last the boring part is over. Now it's time to build the real deal. Assuming you have theoretical knowledge of how neural networks work, we will build one for

ourselves in 5 easy steps. If you don't or need a little refresher, take a look at this 5 minute Neural Networks Fundamentals and come back fast. I am waiting for you. Got your fundamentals clear? Let's begin with importing and understanding the library.

Step 0 — Importing Library

We are using Keras to implement our neural network. It is open source and written in Python. Apart from its user-friendliness and ease of model building, it provides a wide range of production options and integration with at least five back-end engines, with TensorFlow being the most used. Plus, Keras is backed by Google, Microsoft, Amazon, Apple, Nvidia, Uber, and others[3]. So why not us?

```
1  from keras.models import Sequential
2  from keras.layers import Dense
```

library.py hosted with ❤ by **GitHub** **view raw**

Here, we are using `Sequential` model to initiate the neural network. And the `Dense` is used to create input and hidden layers.

Step 1 — Initiate the Neural Network

As we are building a classifier, we will create an object called classifier and use `Sequential()` to initiate the neural network.

```
1  classifier = Sequential()
```

initialize.py hosted with ❤ by **GitHub** **view raw**

Step 2 — Adding Input and Hidden Layers

Now we will use `.add` method to add input layers and hidden layers with the help of `Dense()` which we imported earlier. Here comes an important part. There is no rule of thumb to "how many hidden layers should I add", it is more like an art or a personal touch to your model. You can use according to your theory. Find a pattern in what I did. Let's understand the parameters.

Units: Number of hidden layers

Activation: Activation function decides, whether a neuron should be activated or not by calculating a weighted sum and further adding bias with it. The purpose of

the activation function is to introduce non-linearity into the output of a neuron[4]. For input and hidden layers, most commonly used activation function is *Rectifier Activation Function*. And for the output layer, *Sigmoid* is preferred. Learn more.

Input dimension: Number of independent variables/features

```
classifier.add(Dense(units = 5, activation = 'relu', input_dim=9))
classifier.add(Dense(units = 3, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))
```

layers.py hosted with ❤ by GitHub view raw

Step 3 — Configure The Model for Training

With the help of `.compile` we configure our model. Which has many arguments for the advanced level but at the beginner level, 3 of them are mandatory. Let's see what they are, but first, look at the line of code.

```
classifier.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])
```

configuration.py hosted with ❤ by GitHub view raw

Optimizer: Neural Network learn basically by updating the weight. How do we choose the optimal weight and make our model powerful? That's optimizer. We are using stochastic gradient descent here. And specifically *'adam'*. There are several others. You'll learn as you dive deeper.

Loss: It is basically the cost function, you can call whichever you like. Our model will not generate the right output in the first attempt, it will make errors, consider those errors (loss) and then learn from it. We are using Mean squared errors function, check out others as well.

Metrics: To find the accuracy of the trained model.

Step 4 — Training our Model

Here comes the best part, seeing training our network in real time. After providing training data there are two other parameters. First is **batch size**, I've set it to 10 that means after evaluating a batch of 10 (ten iterations) it will update weights, it's called *batch learning*. However; another approach is to update at every iteration, it's called *reinforcement learning*. And finally there's **epoch** — I am sure you've

heard/seen this word many times — it's just a fancy word for iterations. Like, yeah, 100 iterations. Repeating the same procedure 100 times.

```
classifier.fit(X_train, Y_train, batch_size = 10, epochs = 100)
```

training.py hosted with ❤ by GitHub view raw

That's Colab

As you can see the model we built seem to have 98.03% accuracy in training data. So let's predict our output now and then compare it with the actual output to see how fine our model works on new data. Here comes the challenge Mr. Classifier, get ready.

Step 5 — Predict Output

Predicting is easy after all that we have done. Using the predict method on testing data and we are done. Moreover, as we just need yes or no as an output, we will classify that. A general rule of thumb is it is more than 0.5 than positive and if less then that, negative. We have our output, let's cross-validate.

```
Y_pred = classifier.predict(X_test)
Y_pred = [ 1 if y>=0.5 else 0 for y in Y_pred]
```

```
print(Y_pred)
```

predict.py hosted with ❤ by GitHub — view raw

Step 6 — Performance Measure

Measuring the performance of our model is like walking in the park, thanks to the confusion matrix. A confusion matrix is nothing but a 2x2 matrix indicating all correct and incorrect values. I'll show you how. Look at the image below.

Source

Here, TN is True Negative, that means the actual output was no and our model also predicted no. TP is True positive, I think I don't have to explain that now. These both are our correct answers and the other two are false. To find the accuracy, we will **add** the number to *correct observations* and **divide** it *by the total number of observations*. You'll get a better idea in a minute.

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)
print(cm)
```

validate.py hosted with ❤ by GitHub — view raw

Confusion Matrix

Here, in our test data, we have 140 entries. Out of which our confusion matrix says we have 82+54 = 136 correct observations and 3+1 = 4 incorrect. Therefore the accuracy of our model on new training data would be 136/140 = 0.9714 which is 97.14% accuracy. Not at all bad for the first time.

Endnotes

I was theoretically aware of what neural networks are and how they work but was not able to find the right resources to get started with practically building one, and I thought let's assemble pieces and create a decent blog post so that it'll be easier for people just like me. Moreover, there are several ways we can improve our model. If you want me to work on an article for that too, let me know in the comments. Here is the Github repository for data files and code.

Also, my Twitter and LinkedIn DMs are always open. If you have any feedback, let me know. Even if you don't, you can digitally come by and say hello. Always love to connect with new people. Find further reading below. Happy Learning.

Reference

[1]Global cancer statistics for the most common cancers (link)

[2]Why, How and When to Scale your Features (link)

[3]What is Keras? Thoroughly explained (link)

[4]Activation functions in Neural Networks(link)

Further Reading

Analysis of the Wisconsin Breast Cancer Dataset (link)

ML fundamentals (Cost functions and gradient descent) (link)

Gentle Introduction to the Adam Optimization Algorithm (link)

Technology | Python | Deep Learning | Artificial Intelligence | Machine Learning

252 claps

WRITTEN BY

Dhrumil Patel

Learning and Sharing | Applied Computing @UWindsor

Follow

Towards Data Science

Sharing concepts, ideas, and codes.

Follow