# **Breast Cancer Classifier**

In this project, we will be using several Python libraries to make a K-Nearest Neighbor classifier that is trained to predict whether a patient has breast cancer.

**You will need to employ the following tools to complete this project:**

### **[Sklearn](http://scikit-learn.org/stable/)**

### **[matplotlib](https://matplotlib.org/)**

### **Explore the data**

**1.**

Let's begin by making an empty script.py file and then importing the breast cancer data from sklearn. We want to import the function load\_breast\_cancer from sklearn.datasets.

Once we've imported the dataset, let's load the data into a variable called breast\_cancer\_data. Do this by setting breast\_cancer\_data equal to the function load\_breast\_cancer().

**2.**

Before jumping into creating our classifier, let's take a look at the data. Begin by printing breast\_cancer\_data.data[0]. That's the first datapoint in our set. But what do all of those numbers represent? Let's also print breast\_cancer\_data.feature\_names.

**3.**

We now have a sense of what the data looks like, but what are we trying to classify? Let's print both breast\_cancer\_data.target and breast\_cancer\_data.target\_names.

Was the very first data point tagged as malignant or benign?

### **Splitting the data into Training and Validation Sets**

**4.**

We have our data, but now it needs to be split into training and validation sets. Luckily, sklearn has a function that does that for us. Begin by importing the train\_test\_split function from sklearn.model\_selection.

**5.** Call the train\_test\_split function. It takes several parameters:

* The data you want to split (for us breast\_cancer\_data.data)
* The labels associated with that data (for us, breast\_cancer\_data.target).
* The train\_size. This is what percentage of your data you want to be in your training set. Let's use train\_size = 0.8
* random\_state. This will ensure that every time you run your code, the data is split in the same way. This can be any number. We used random\_state = 100.

**6.** Right now we're not storing the return value of train\_test\_split. train\_test\_split returns four values in the following order:

* The training set
* The validation set
* The training labels
* The validation labels

Store those values in variables named training\_data, validation\_data, training\_labels, and validation labels.

**7.**

Let's confirm that worked correctly. Print out the length of training\_data and training\_labels. They should be the same size - one label for every piece of data!

### **Running the classifier**

**8.**

Now that we've created training and validation sets, we can create a KNeighborsClassifierand test its accuracy. Begin by importing KNeighborsClassifier from sklearn.neighbors.

**9.**

Create a KNeighborsClassifier where n\_neighbors = 3. Name the classifier classifier.

**10.**

Train your classifier using the fit function. This function takes two parameters: the training set and the training labels.

**11.**

Now that the classifier has been trained, let's find how accurate it is on the validation set. Call the classifier's score function. score takes two parameters: the validation set and the validation labels. Print the result!

**12.**

The classifier does pretty well when k = 3. But maybe there's a better k! Put the previous 3 lines of code inside a for loop. The loop should have a variable named k that starts at 1 and increases to 100. Rather than n\_neighbors always being 3, it should be this new variable k.

You should now see 100 different validation accuracies print out. Which k seems the best?

### **Graphing the results**

**13.**

We now have the validation accuracy for 100 different ks. Rather than just printing it out, let's make a graph using matplotlib. Begin by importing matplotlib.pyplot as plt.

**14.**

The x-axis should be the values of k that we tested. This should be a list of numbers between 1 and 100. You can use the range function to make this list. Store it in a variable named k\_list.

**15.**

The y-axis of our graph should be the validation accuracy. Instead of printing the validation accuracies, we want to add them to a list. Outside of the for loop, create an empty list named accuracies. Inside the for loop, instead of printing each accuracy, append it to accuracies.

**16.**

We can now plot our data! Call plt.plot(). The first parameter should be k\_list and the second parameter should be accuracies.

After plotting the graph, show it using plt.show().

**17.**

Let's add some labels and a title. Set the x-axis label to "k" using plt.xlabel(). Set the y-axis label to "Validation Accuracy". Set the title to "Breast Cancer Classifier Accuracy".

**18.**

Great work! If you want to play around with this more, try changing the random\_state parameter when making the training set and validation set. This will change which points are in the training set and which are in the validation set.

Ideally, the graph will look the same no matter how you split up the training set and test set. This data set is fairly small, so there is slightly more variance than usual.

Submit your completed script.py file when you are ready.