1. import csv

The so-called CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases.

The [csv](https://docs.python.org/3/library/csv.html#module-csv) module implements classes to read and write tabular data in CSV format.

The [csv](https://docs.python.org/3/library/csv.html#module-csv) module’s [reader](https://docs.python.org/3/library/csv.html#csv.reader) and [writer](https://docs.python.org/3/library/csv.html#csv.writer) objects read and write sequences.

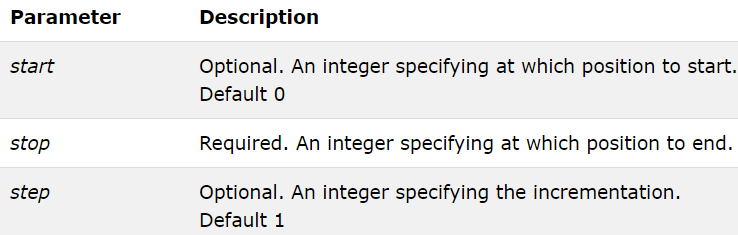
1. import random

Sometimes we want the computer to pick a random number in a given range, pick a random element from a list

The random module provides access to functions that support these types of operations. The **random** module is another library of functions that can extend the basic features of python.

index = random.randrange(len(copy));

The randrange() method returns a randomly selected element from the specified range.



1. import math

The math [module](https://www.programiz.com/python-programming/modules) is a standard module in Python and is always available. To use mathematical functions under this module, you have to import the module using import math.

It gives access to the underlying C library functions. For example,

# Square root calculation

import math

math.sqrt(4)

There are many functions in math module. We are using sqrt, exponent, power, pi.

|  |  |
| --- | --- |
| exp(x) | Returns e\*\*x |
| sqrt(x) | Returns the square root of x |
| pow(x, y) | Returns x raised to the power y |
| pi | Mathematical constant, the ratio of circumference of a circle to it's diameter (3.14159...) |

1. def loadcsv(filename):

lines = csv.reader(open(filename, "r"));

dataset = list(lines)

for i in range(len(dataset)):

#converting strings into numbers for processing

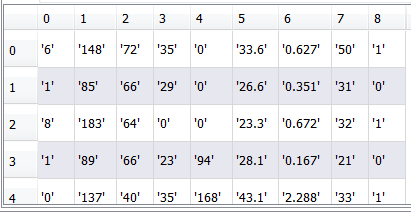
dataset[i] = [float(x) for x in dataset[i]]

return dataset

The first thing we need to do is load our data file. The data is in CSV format without a header line or any quotes. We can open the file with the open function and read the data lines using the reader function in the CSV module.

The lines will have csv.reader object

The dataset will look like the one shown below



It will create a list of list. Each row will represent a list. The first row corresponds to first row in the data set.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Examples** | **Pregnancies** | **Glucose** | **Blood Pressure** | **Skin Thickness** | **Insulin** | **BMI** | **Diabetic Pedigree Function** | **Age** | **Outcome** |
| 1 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 2 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 3 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 4 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 5 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| 6 | 5 | 116 | 74 | 0 | 0 | 25.6 | 0.201 | 30 | 0 |
| 7 | 3 | 78 | 50 | 32 | 88 | 31 | 0.248 | 26 | 1 |
| 8 | 10 | 115 | 0 | 0 | 0 | 35.3 | 0.134 | 29 | 0 |
| 9 | 2 | 197 | 70 | 45 | 543 | 30.5 | 0.158 | 53 | 1 |
| 10 | 8 | 125 | 96 | 0 | 0 | 0 | 0.232 | 54 | 1 |

1. def splitdataset(dataset, splitratio):

#67% training size

trainsize = int(len(dataset) \* splitratio);

trainset = []

copy = list(dataset);

while len(trainset) < trainsize:

#generate indices for the dataset list randomly to pick ele for training data

index = random.randrange(len(copy));

trainset.append(copy.pop(index))

return [trainset, copy]

This code will split the data into training and testing dataset.

We have defined split ratio as (out of 629)

splitratio = 0.67

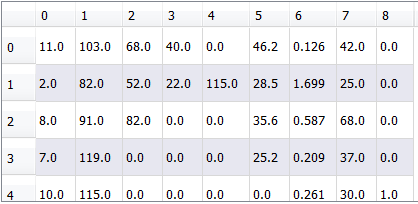
**train size will take the value 514**

index will take random number for example 115

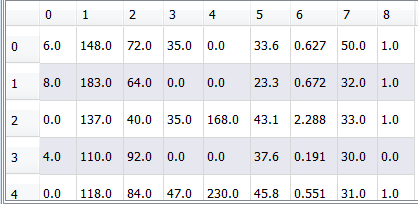
This function will return the training data set and testing data set. Both are randomly picked from the entire dataset.

67% will be training data set and 33% will be testing dataset.

Training dataset will look like below



And Testing dataset will look like below



1. def separatebyclass(dataset):

separated = {}

#creates a dictionary of classes 1 and 0 where the values are the instances belonging to each class

for i in range(len(dataset)):

vector = dataset[i]

if (vector[-1] not in separated):

separated[vector[-1]] = []

separated[vector[-1]].append(vector)

return separated

The **range**() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

Separated={}

for 0 in [0,1,2,…,514]

vector = zeroth row of dataset



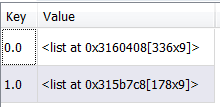
At the end of 1 iteration we will have



At the end of 2nd iteration we will have



At the end of all the iterations we have two dictionaries. One for class 0 and one for class 1 as shown below



1. def mean(numbers):

return sum(numbers)/float(len(numbers))

* **Calculate Mean**

1. def stdev(numbers):

avg = mean(numbers)

variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)

return math.sqrt(variance)

* **Calculate Standard Deviation**

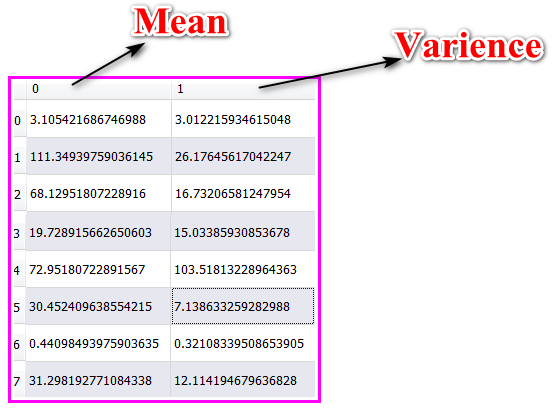
1. def summarize(dataset): #creates a dictionary of classes

summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(\*dataset)];

del summaries[-1] #excluding labels +ve or -ve

return summaries

* **Summarize Dataset**



The **zip**() function returns a **zip** object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.

Example:

a = ("John", "Charles", "Mike")

b = ("Jenny", "Christy", "Monica", "Vicky")

x = zip(a, b)

print(tuple(x))

**output:**

(('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica'))))

1. def summarizebyclass(dataset):

separated = separatebyclass(dataset);

#print(separated)

summaries = {}

for classvalue, instances in separated.items():

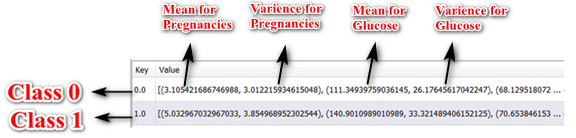
#for key,value in dic.items()

#summaries is a dic of tuples(mean,std) for each class value

summaries[classvalue] = summarize(instances) #summarize is used to cal to mean and std

return summaries

summarizebyclass function **Summarize Attributes By Class**



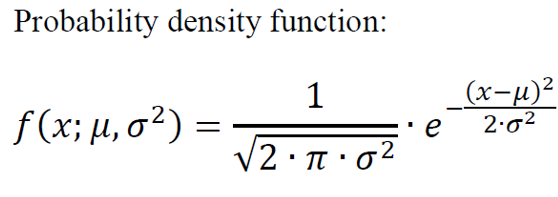
1. def calculateprobability(x, mean, stdev):

exponent = math.exp(-(math.pow(x-mean,2)/(2\*math.pow(stdev,2))))

return (1 / (math.sqrt(2\*math.pi) \* stdev)) \* exponent

We are now ready to make predictions using the summaries prepared from our training data. Making predictions involves calculating the probability that a given data instance belongs to each class, then selecting the class with the largest probability as the prediction.

* **Calculate Gaussian Probability Density Function**



def calculateclassprobabilities(summaries, inputvector):

probabilities = {} # probabilities contains the all prob of all class of test data

for classvalue, classsummaries in summaries.items():#class and attribute information as mean and sd

probabilities[classvalue] = 1

for i in range(len(classsummaries)):

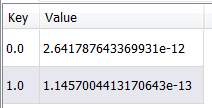
mean, stdev = classsummaries[i] #take mean and sd of every attribute for class 0 and 1 seperaely

x = inputvector[i] #testvector's first attribute

probabilities[classvalue] \*= calculateprobability(x, mean, stdev);#use normal dist

return probabilities

* **Calculate Class Probabilities (ex: probability of class 0 and probability of class 1)(as shown in the table below)**



1. def predict(summaries, inputvector): #training and test data is passed

probabilities = calculateclassprobabilities(summaries, inputvector)

bestLabel, bestProb = None, -1

for classvalue, probability in probabilities.items():#assigns that class which has the highest prob

if bestLabel is None or probability > bestProb:

bestProb = probability

bestLabel = classvalue

return bestLabel

* **Make a Prediction and return the label of the class (0 or 1) for training data**

1. def getpredictions(summaries, testset):

predictions = []

for i in range(len(testset)):

result = predict(summaries, testset[i])

predictions.append(result)

return predictions

* **Make a Prediction and return the label of the class (0 or 1) for testing data**

1. def getaccuracy(testset, predictions):

correct = 0

for i in range(len(testset)):

if testset[i][-1] == predictions[i]:

correct += 1

return (correct/float(len(testset))) \* 100.0

* **Get Accuracy**

Correct predictions divided by length of test dataset x 100

1. def main():

filename = 'naivedata.csv'

splitratio = 0.67

dataset = loadcsv(filename);

trainingset, testset = splitdataset(dataset, splitratio)

print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingset), len(testset)))

# prepare model

summaries = summarizebyclass(trainingset);

# test model

predictions = getpredictions(summaries, testset) #find the predictions of test data with the training data

accuracy = getaccuracy(testset, predictions)

print('\n Accuracy of the classifier is : {0}%'.format(accuracy))

Finally, we define our main function where we call all these methods we have defined, one by one to get the accuracy of the model we have created.

1. main()