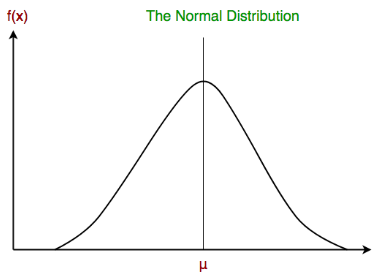


**Gaussian Naive Bayes classifier**

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a **Gaussian distribution**. A Gaussian distribution is also called [Normal distribution](https://en.wikipedia.org/wiki/Normal_distribution). When plotted, it gives a bell shaped curve which is symmetric about the mean of the feature values as shown below:

The likelihood of the features is assumed to be Gaussian, hence, conditional probability is given by:



Other popular Naive Bayes classifiers are:

**Multinomial Naive Bayes**: Feature vectors represent the frequencies with which certain events have been generated by a **multinomial distribution**. This is the event model typically used for document classification.

**Bernoulli Naive Bayes**: In the multivariate Bernoulli event model, features are independent booleans (binary variables) describing inputs. Like the multinomial model, this model is popular for document classification tasks, where binary term occurrence(i.e. a word occurs in a document or not) features are used rather than term frequencies(i.e. frequency of a word in the document).

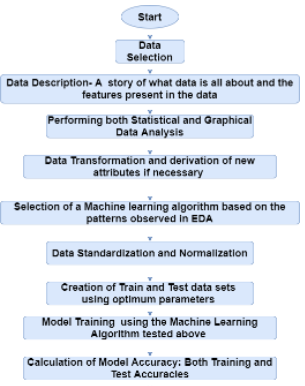
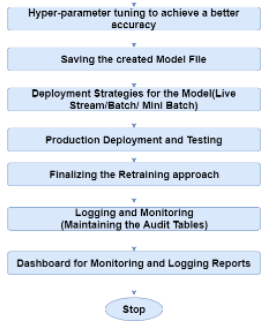
As we reach to the end of this article, here are some important points to ponder upon:

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters.

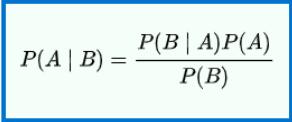
Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

**Application Flow**

Before proceeding with the algorithm, let’s first discuss the lifecycle of any machine learning model. This diagram explains the creation of a Machine Learning model from scratch and then taking the same model further with hyperparameter tuning to increase its accuracy, deciding the deployment strategies for that model and once deployed setting up the logging and monitoring frameworks to generate reports and dashboards based on the client requirements. A typical lifecycle diagram for a machine learning model looks like:



**Bayes’s Theorem**

According to the Wikipedia, In probability theory and statistics,**Bayes’s theorem** (alternatively Bayes’s law or Bayes’s rule) describes the probability of an event, based on prior knowledge of conditions that might be related to the event. Mathematically, it can be written as:

Where A and B are events and P(B)≠0

* P(A|B) is a conditional probability: the likelihood of event A occurring given that B is true.
* P(B|A) is also a conditional probability: the likelihood of event B occurring given that A is true.
* P(A) and P(B) are the probabilities of observing A and B respectively; they are known as the marginal probability.

Let’s understand it with the help of an example:

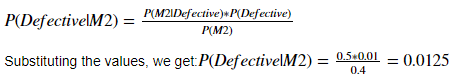
**The problem statement:**

There are two machines which manufacture bulbs. Machine 1 produces 30 bulbs per hour and machine 2 produce 20 bulbs per hour. Out of all bulbs produced, 1 % turn out to be defective. Out of all the defective bulbs, the share of each machine is 50%. What is the probability that a bulb produced by machine 2 is defective?

We can write the information given above in mathematical terms as:

* The probability that a bulb was made by Machine 1, P(M1)=30/50=0.6
* The probability that a bulb was made by Machine 2, P(M2)=20/50=0.4
* The probability that a bulb is defective, P(Defective)=1%=0.01
* The probability that a defective bulb came out of Machine 1, P(M1 | Defective)=50%=0.5
* The probability that a defective bulb came out of Machine 2, P(M2 | Defective)=50%=0.5

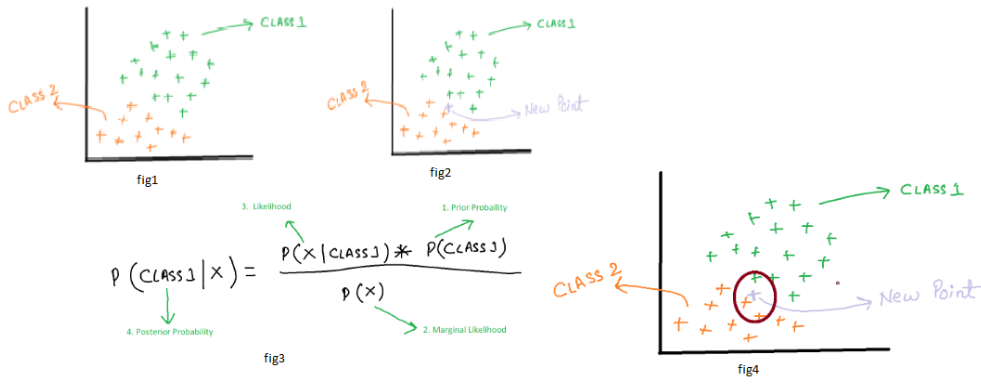
Now, we need to calculate the probability of a bulb produced by machine 2 is defective i.e., P(Defective | M2). Using the Bayes Theorem above, it can be written as:



Task for you is to calculate the probability that a bulb produced by machine 1 is defective.

We’ll extend this same understanding to understand the Naïve Baye’s Algorithm

**Algorithm steps:**



* Let’s consider that we have a binary classification problem i.e., we have two classes in our data as shown in fig1.
* In fig2 Now suppose if we are given with a new data point, to which class does that point belong to?
* In fig3 The formula for a point ‘X’ to belong in class1 can be written as: Where the numbers represent the order in which we are going to calculate different probabilities.
* A similar formula can be utilised for class 2 as well.
* Probability of class 1 can be written as:



* In fig4 For calculating the probability of X, we draw a circle around the new point and see how many points(excluding the new point) lie inside that circle.
* The points inside the circle are considered to be similar points.



* Now, we need to calculate the probability of a point to be in the circle that we have made given that it’s of class



* We can substitute all the values into the formula in step 3. We get:



* And if we calculate the probability that X belongs to Class2, we’ll get 0.69. It means that our point belongs to class 2.

**The Generalization for Multiclass:**

The approach discussed above can be generalised for multiclass problems as well. Suppose, P1, P2, P3…Pn are the probabilities for the classes C1,C2,C3…Cn, then the point X will belong to the class for which the probability is maximum. Or mathematically the point belongs to the result of : argmax(P1,P2,P3….Pn)argmax(P1,P2,P3….Pn)

**The Difference**

You can notice a major difference in the way in which the Naïve Bayes algorithm works form other classification algorithms. It does not first try to learn how to classify the points. It directly uses the label to identify the two separate classes and then it predicts the class to which the new point shall belong.

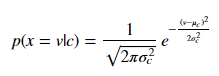
**Why it is called Naïve Bayes?**

The entire algorithm is based on Bayes’s theorem to calculate probability. So, it also carries forward the assumptions for the Bayes’s theorem. But those assumptions(that the features are independent) might not always be true when implemented over a real-world dataset. So, those assumptions are considered Naïve and hence the name.

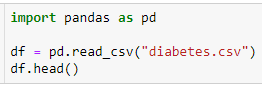
**Gaussian Naive Bayes**

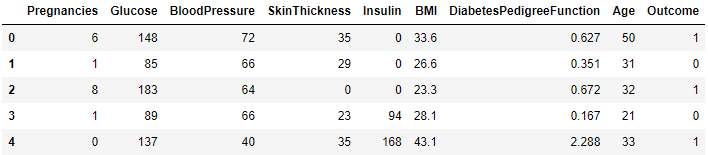
When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution. Go back to the normal distribution lecture to review the formulas for the Gaussian/Normal Distribution.

For example of using the Gaussian Distribution, suppose the training data contain a continuous attribute, x. We first segment the data by the class, and then compute the mean and variance of x in each class. Let μc be the mean of the values in x associated with class c, and let σ2c be the variance of the values in x associated with class c. Then, the probability distribution of some value given a class, p(x=v|c), can be computed by plugging v into the equation for a Normal distribution parameterized by μc and σ2c. That is:

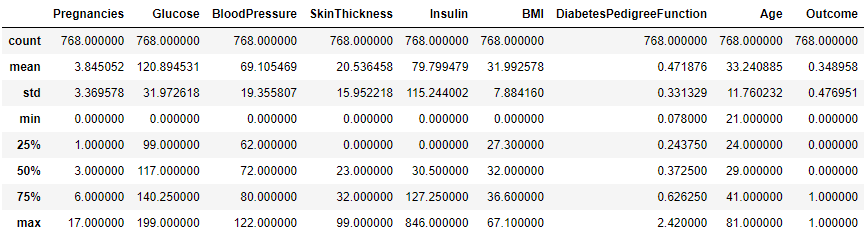


**1. import the data**



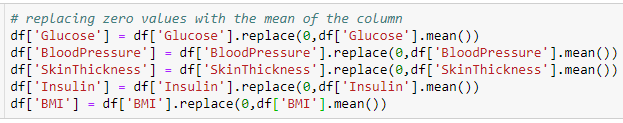


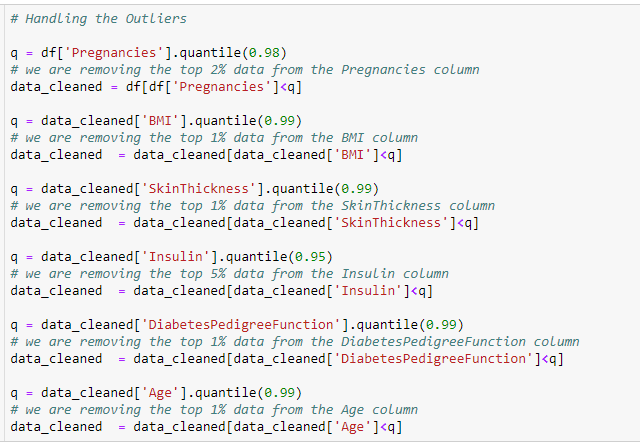


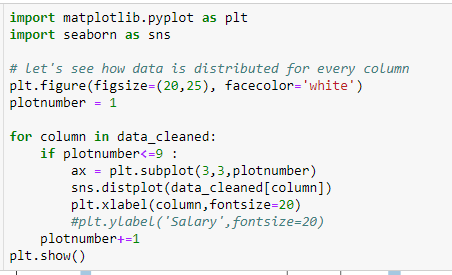


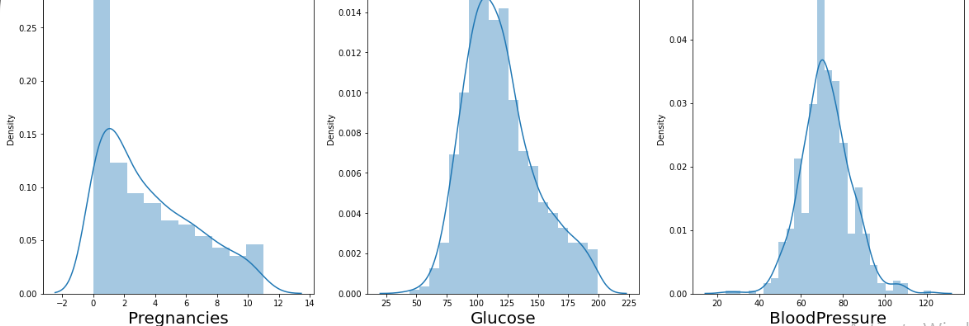
**2. preprocessing of the data**

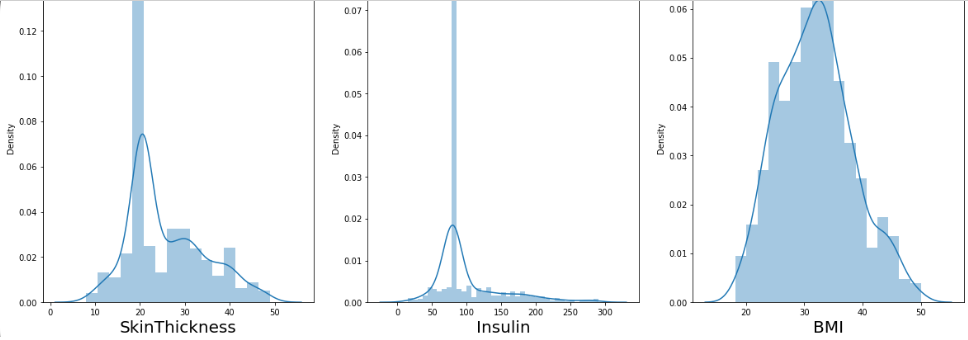
we can see there few data for columns Glucose, Insulin, skin thickness, BMI and Blood Pressure which have value as 0. That's not possible. You can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace it with their respective mean values. Let's do the later.

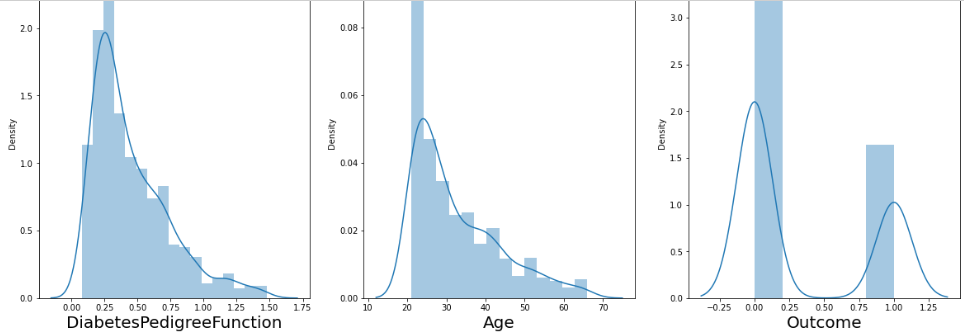




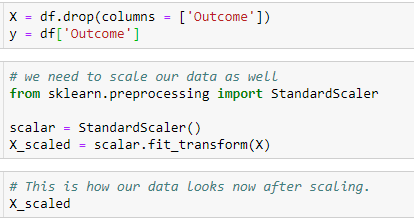


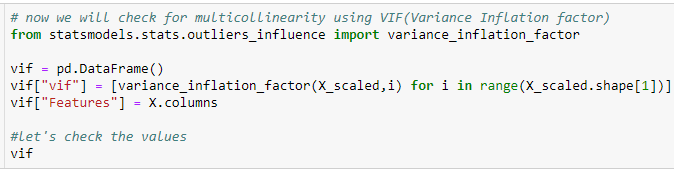




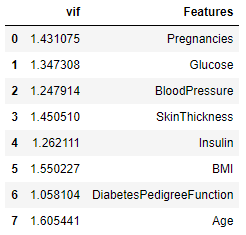


**3. identify the independent and dependent variables**

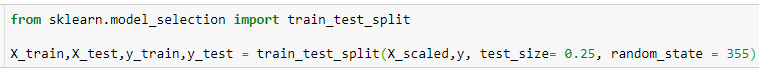




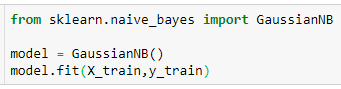
All the VIF values are less than 5 and are very low. That means no multicollinearity. Now, we can go ahead with fitting our data to the model. Before that, let's split our data in test and training set.



**4. split the data into train and test**



**5. fit/ train the model**

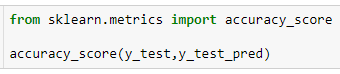


GaussianNB()

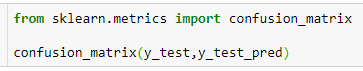
**6. predict on test data**



**7. Accuracy**

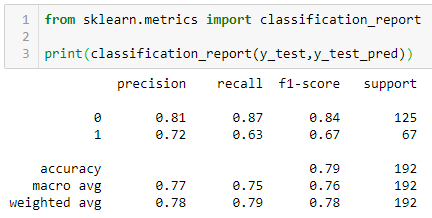


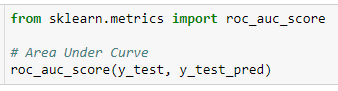
0.7864583333333334



array([[109, 16],

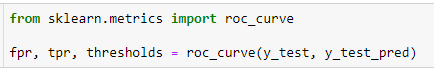
[ 25, 42]], dtype=int64)

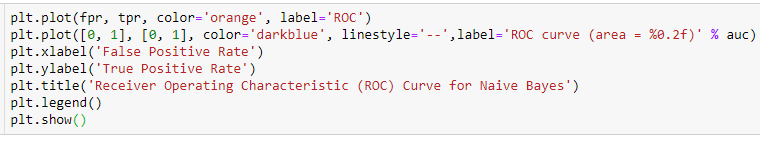


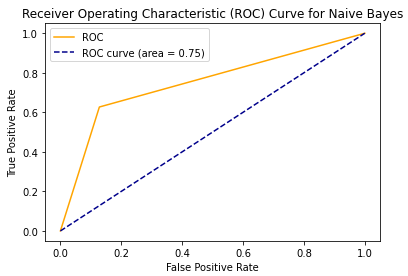


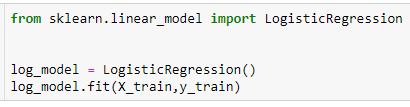
0.7494328358208956

So far we have been doing grid search to maximise the accuracy of our model. Here, we’ll follow a different approach. We’ll create two models, one with Logistic regression and other with Naïve Bayes and we’ll compare the AUC. The algorithm having a better AUC shall be considered for production deployment.







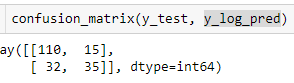


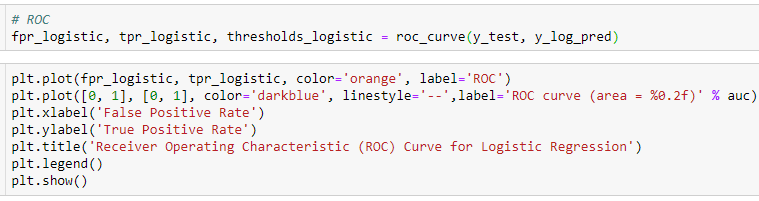
LogisticRegression()

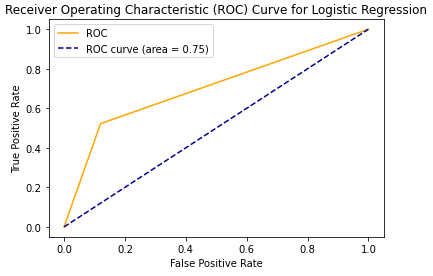


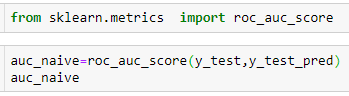


0.7552083333333334

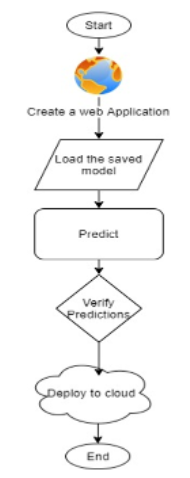








0.7494328358208956



0.7011940298507463

Here, you can see that the AUC for Naïve Bayes is more. So, we’ll take that as our production-ready model.

**Advantages:**

* Naive Bayes is extremely fast for both training and prediction as they not have to learn to create separate classes.
* Naive Bayes provides a direct probabilistic prediction.
* Naive Bayes is often easy to interpret.
* Naive Bayes has fewer (if any) parameters to tune

**Disadvantages:**

* The algorithm assumes that the features are independent which is not always the scenario
* Zero Frequency i.e. if the category of any categorical variable is not seen in training data set even once then model assigns a zero probability to that category and then a prediction cannot be made.

### **Cloud Deployment**

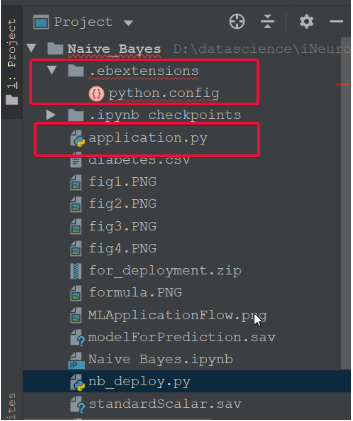
Once the training is completed, we need to expose the trained model as an API for the user to consume it. For prediction, the saved model is loaded first and then the predictions are done using it. If the web app works fine, the same app is deployed to the cloud platform. The flow for that can be shown as:

**Pre-requisites for Cloud Deployment:**

* Basic knowledge of flask framework.
* Any Python IDE installed(we are using PyCharm).
* An AWS account.
* Basic understanding of HTML.

**The Flask App:** As we’ll expose the created model as a web API to be consumed by the client/client APIs, we’d do it using the flask framework.

Create the project structure, as shown below:



The content for **application.py** is:

from flask import Flask, request, app

from flask import Response

from flask\_cors import CORS

from nb\_deploy import predObj

application = Flask(\_\_name\_\_) # initializing a flask app

app=application

CORS(app)

app.config['DEBUG'] = True

class ClientApi:

def \_\_init\_\_(self):

self.predObj = predObj()

@app.route("/predict", methods=['POST'])

def predictRoute():

try:

if request.json['data'] is not None:

data = request.json['data']

print('data is: ', data)

pred=predObj()

res = pred.predict\_log(data)

#result = clntApp.predObj.predict\_log(data)

print('result is ',res)

return Response(res)

except ValueError:

return Response("Value not found")

except Exception as e:

print('exception is ',e)

return Response(e)

if \_\_name\_\_ == "\_\_main\_\_":

clntApp = ClientApi()

host = '0.0.0.0'

port = 5000

app.run(debug=True)

#httpd = simple\_server.make\_server(host, port, app)

# print("Serving on %s %d" % (host, port))

#httpd.serve\_forever()

The content for **nb\_deploy.py** is:

#Let's start with importing necessary libraries

import pickle

import pandas as pd

class predObj:

def predict\_log(self, dict\_pred):

with open("standardScalar.sav", 'rb') as f:

scalar = pickle.load(f)

with open("modelForPrediction.sav", 'rb') as f:

model = pickle.load(f)

data\_df = pd.DataFrame(dict\_pred,index=[1,])

scaled\_data = scalar.transform(data\_df)

predict = model.predict(scaled\_data)

#predict = model.predict(data\_df)

if predict[0] ==1 :

result = 'Diabetic'

else:

result ='Non-Diabetic'

return result

The content for **python.config** is:

option\_settings:

"aws:elasticbeanstalk:container:python":

WSGIPath: application.py

files:

"/etc/httpd/conf.d/wsgi\_custom.conf":

mode: "000644"

owner: root

group: root

content: |

WSGIApplicationGroup %{GLOBAL}

**Points to consider before deployment**

The python application file should be named application.py

Create a requirements.txt using pip freeze > requirements.txt from the project folder

Create a folder .ebextensions and create a file python.config inside it. Make sure to populate the content of python.config, as shown above.

Create the zip file from the project folder itself.

**Deployment Process**

* Go to <https://aws.amazon.com/> and create an account if already don’t have one.
* Go to the console and go to the ‘Build a web app’ section and click it.
* Give the name of the application, give platform as python, and select the option to upload your code.
* Click on Create application to upload your code and create the app