

**CLASSIFIERS LABORATORY REPORT**

**Bayes Classifier**

**Authors:**

Ankit RATHI

**Understanding what is a Bayes Classifier?**

Bayes is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithms. A Bayes classifier is best interpreted as a decision rule.

The naive Bayes classifier is an approximation to the Bayes classifier, in which we assume that the features are conditionally independent given the class instead of modeling their full conditional distribution given the class. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.

## Short Description

In machine learning, Naive Bayes Classifier belongs to the category of **Probabilistic Classifiers*.*** A probabilistic classifier can predict given observation by using a probability distribution over a set of classes and based on that distribution it will predict the most likely class that the observation should belong to.

Naive Bayes classification is a probabilistic approach to classify the data set based on the famous and well known Bayes Theorem of probability. Key terms in Naive Bayes classification are **Prior probability**, **Posterior Probability**, **Likelihood probability**, and **Evidence probability**.

## Classification Workflow

Whenever you perform classification, the first step is to understand the problem and identify potential features and label. Features are those characteristics or attributes which affect the results of the label.

These characteristics are known as **features** which help the model classify customers.

The classification has two phases, a **learning phase**, and **the evaluation phase**. In the learning phase, classifier trains its model on a given dataset and in the evaluation phase, it tests the classifier performance. Performance is evaluated on the basis of various parameters such as **accuracy**, **error**, **precision**, and **recall**.

Schema:

Data

Training

Model

Development

Test set

Performance

measure

Model

Evaluation

1

.Accuracy

2

.Precision

**Project Task**

Our aim in this exercise was to understand, solve the task (given by on the platforma) and implement a Bayes classifier and the simple classifier accuracy estimator.

We have generated functions for distribution estimation, test the classifier accuracy and train Bayes classifier with synthetic dataset.

Project Implementation

Task 1

First of all we have to import some libraries which will be helpful us.

|  |
| --- |
| import numpy as np  from sklearn.naive\_bayes import GaussianNB  import pandas as pd  from sklearn.model\_selection import train\_test\_split  import matplotlib.pyplot as plt  import scipy.stats from functools import reduce  import math  from sklearn.metrics import confusion\_matrix  import scipy.io import mat4py |

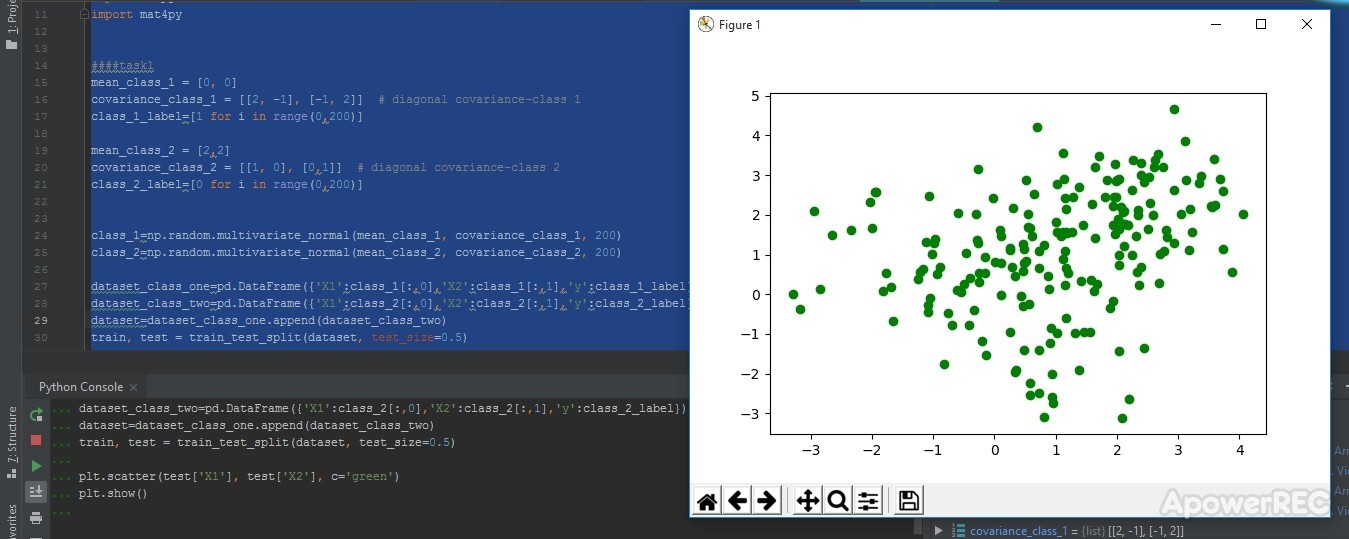
1. Generate synthetic dataset with 2 features and 200 probes. The data for each feature coming from two distinctclasses A1 and A2. From class A1, 100 data points are generated from a Gaussian with parameters:

1=(00) Σ1=(2−1−12)

And from class A2, 100 data points using a Gaussian with parameters: 2=(22) Σ1=(1001)

To assess the performance of the classifier please split the dataset randomly into two groups with 100 data points (training set and test set). The test set will be used only for assessment of the classifier accuracy and not during the training phase.

Plot the two classes training data on the scatter plot.



Task 2

1. Write a function that computes Bayes classifier using 2-class dataset with changeable n features (use own implementation).

To estimate the feature distributions use two different methods: a) Parametric method based on normal distribution

b) Non parametric method based on the Parzen windows probability density function estimation (please test the influence of window size on the classifier accuracy).

Train the Bayes classifier with synthetic dataset (only training set can be used).

|  |
| --- |
| def get\_summary\_by\_class(dataset:pd.DataFrame,target\_var\_name:str): res={}  means=dataset.groupby([target\_var\_name], sort=True).mean()# transposition to\_dictionary sd=dataset.groupby([target\_var\_name], sort=True).std() return (means,sd) ### using 2classes dataset def predict\_parametric(dataset:pd.DataFrame,target\_var\_name:str,training\_class\_summaries:tuple):  dataset=dataset.drop([target\_var\_name], axis=1)  ##shapes look like  means=training\_class\_summaries[0] sd=training\_class\_summaries[1] class\_summaries\_shapes=means.shape dataset\_shapes=dataset.shape res=np.empty(dataset\_shapes[0]) for row in range(dataset\_shapes[0]): |

class\_probs={} for i in range(class\_summaries\_shapes[0]): class\_probs[means.index[i]]=reduce(lambda x, y: x\*y,scipy.stats.norm(means.iloc[i,:],sd.iloc[i,:]).pdf(np.array(dataset.iloc[row,:])))

best=max(zip(class\_probs.values(), class\_probs.keys()))

res[row]=best[1] return res

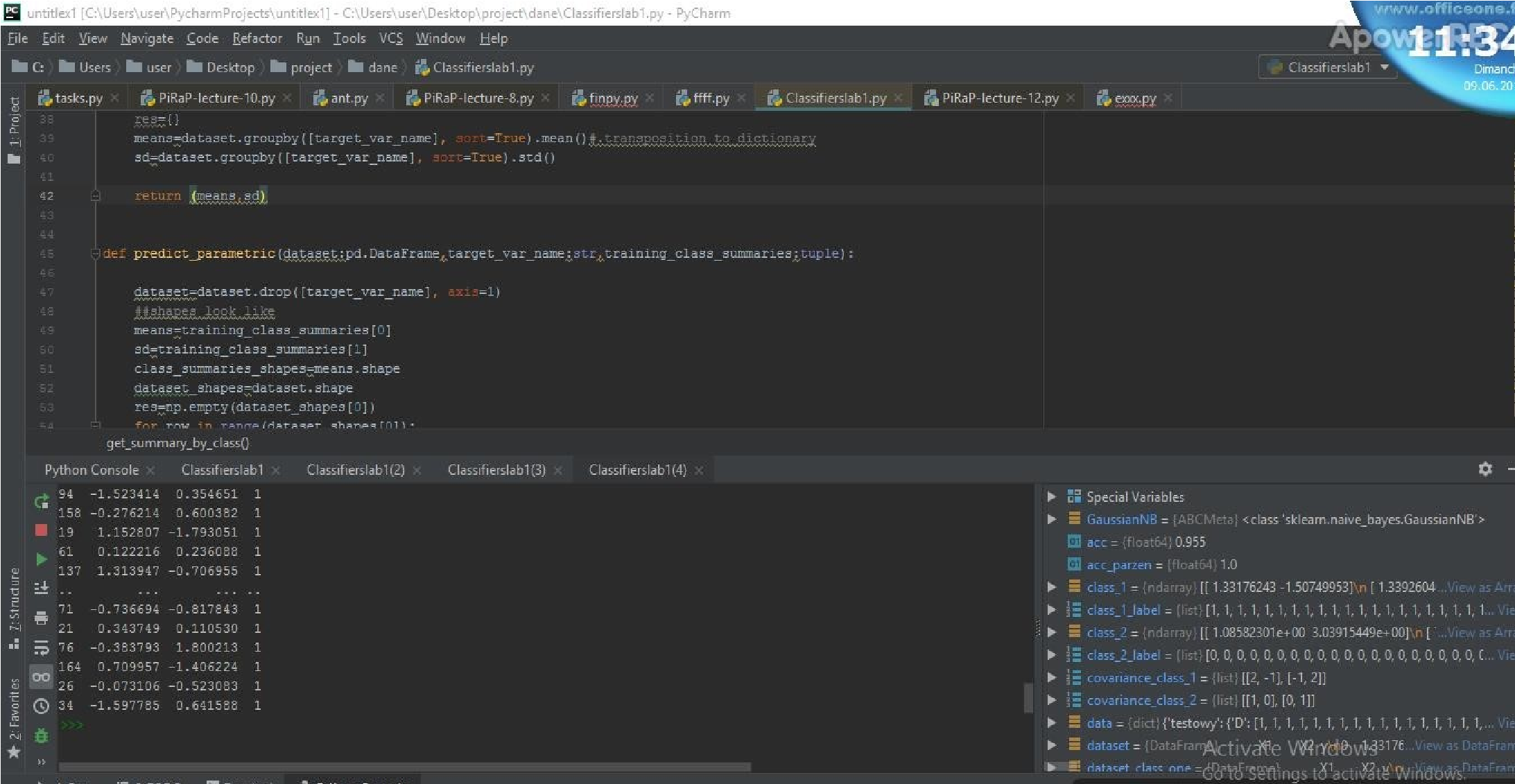
Comment:

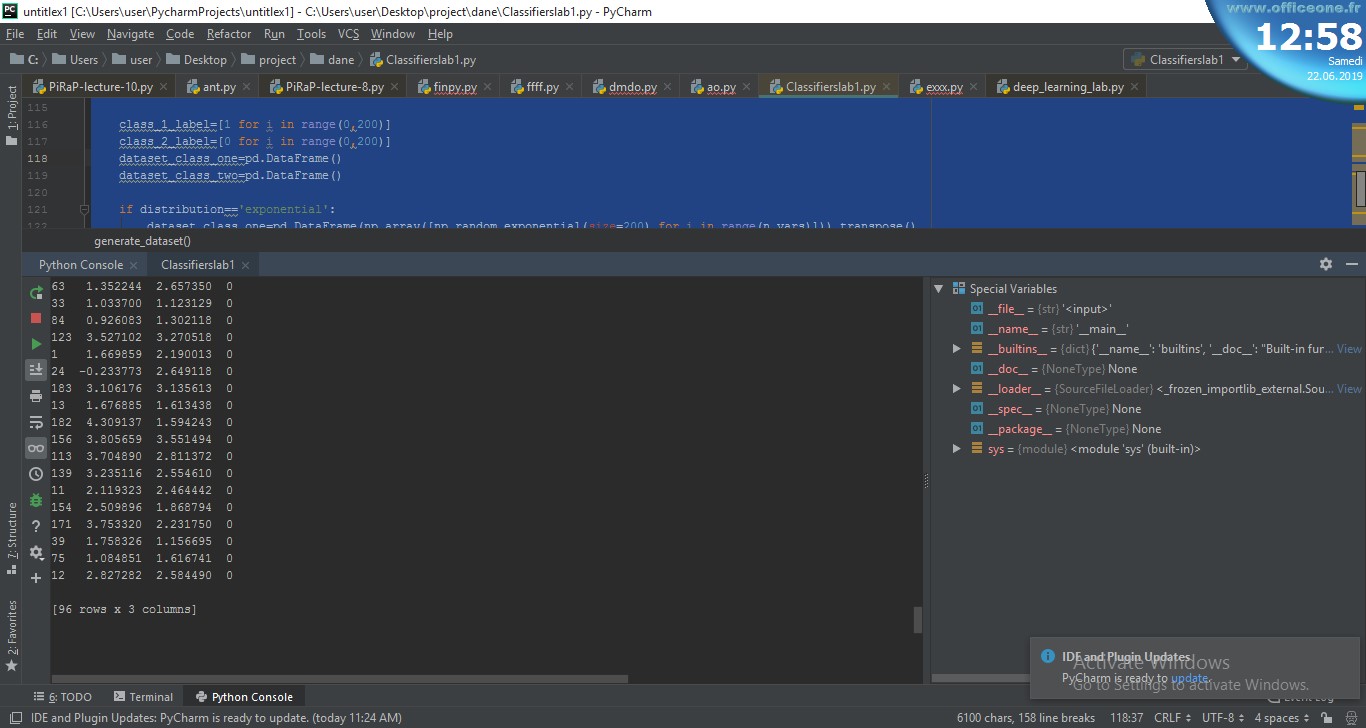
Above we have a function that computes Bayes classifier using 2-class dataset with changeable n features.

Parametric method is based on normal distribution. Parametric density has single local maximum

|  |
| --- |
| def parzen(dataset\_train:pd.DataFrame,target\_var\_name:str,dataset\_test:pd.DataFrame,h:int):  test\_shape=dataset\_test.shape res=np.empty(test\_shape[0])  grouped=dataset\_train.groupby(target\_var\_name)  prob\_groups={} for row\_test\_index in range(test\_shape[0]): for name, group in grouped: print(name) print(group)  train\_shape=group.shape  group.drop([target\_var\_name], axis=1)  parzen\_val=0 for row\_train in range(train\_shape[0]): parzen\_val=parzen\_val+reduce(lambda x, y:  x\*y,scipy.stats.norm(0,1).pdf(np.array(group.iloc[row\_train,:]-  dataset\_test.iloc[row\_test\_index,:])/h))  parzen\_val=parzen\_val/((h\*\*train\_shape[1])\*train\_shape[0]) prob\_groups[name]=parzen\_val res[row\_test\_index]=max(zip(prob\_groups.values(), prob\_groups.keys()))[1] return res  summaries=get\_summary\_by\_class(train,'y')  prediction= predict\_parametric(test,'y',summaries) prediction\_parzen=parzen(train,'y',test,0.1) |

The Parzen windows probability density function estimation.





Comment:

This summary table provides the mean and standard deviation of the data grouped by class so that we can estimate the probabilities P(xi|c) on the assumption that each of these can be represented by a Gaussian distribution.

We can now make predictions. Task3

1. Based on the classifier confusion matrix for test set estimate the performance of the classifier using the classification accuracy rate.

tn, fp, fn, tp = confusion\_matrix(test.iloc[:,2], prediction).ravel() acc=(tn+tp)/(tp+tn+fp+fn)

##acc parzen

tn, fp, fn, tp = confusion\_matrix(test.iloc[:,2], prediction\_parzen).ravel() acc\_parzen=(tn+tp)/(tp+tn+fp+fn)

Comment:

Classification confusion matrix for test set estimation using the classification accuracy rate.

The best accuracy is 1.0, whereas the worst is 0.0. Accuracy is calculated as the total number of two correct predictions (TP + TN) divided by the **total** number of parameters .

Task4

1. To show the differences between used probability density function estimation methods generate different synthetic datasets with more than 5 features

|  |
| --- |
| data=mat4py.loadmat('C:\\Users\\user\\Desktop\\project\\dane\\dane1.mat')  train\_set=data['learning'] test\_set=data['test']  X\_train=pd.DataFrame(data['learning']['X']).transpose()  y\_train=pd.DataFrame(data['learning']['D'])  X\_train['target']=y\_train  X\_train.dropna(inplace=True)  X\_test=pd.DataFrame(data['test']['X']).transpose()  y\_test=pd.DataFrame(data['test']['D'])  X\_test['target']=y\_test  X\_test.dropna(inplace=True) |

Comment:

We used micro array data given to compute Bayes classifier accuracy rate using the probability of testing function density estimation. The real microarray data description the collected dataset includes 2classes.

The dataset is divided into training and test set .

|  |
| --- |
| def generate\_dataset(n\_vars:int,distribution='exponential'):  class\_1\_label=[1 for i in range(0,200)] class\_2\_label=[0 for i in range(0,200)] dataset\_class\_one=pd.DataFrame() dataset\_class\_two=pd.DataFrame()  if distribution=='exponential': dataset\_class\_one=pd.DataFrame(np.array([np.random.exponential(size=200) for i in |
| range(n\_vars)])).transpose()  dataset\_class\_two=pd.DataFrame(np.array([np.random.exponential(size=200) for i in  range(n\_vars)])).transpose() elif distribution=='logistic': dataset\_class\_one=pd.DataFrame(np.array([np.random.logistic(size=200) for i in  range(n\_vars)])).transpose()  dataset\_class\_two=pd.DataFrame(np.array([np.random.logistic(size=200) for i in range(n\_vars)])).transpose()  elif distribution=='lognormal': dataset\_class\_one=pd.DataFrame(np.array([np.random.lognormal(size=200) for i in  range(n\_vars)])).transpose()  dataset\_class\_two=pd.DataFrame(np.array([np.random.lognormal(size=200) for i in range(n\_vars)])).transpose()  dataset\_class\_one['y']=class\_1\_label dataset\_class\_two['y']=class\_2\_label  dataset=dataset\_class\_one.append(dataset\_class\_two) train, test = train\_test\_split(dataset, test\_size=0.5) return (train,test)  def compare\_methods(train:pd.DataFrame,test:pd.DataFrame,target\_var\_name:str,h=0.2): summaries=get\_summary\_by\_class(train,target\_var\_name)  prediction\_parametric= predict\_parametric(test,target\_var\_name,summaries)  prediction\_parzen= parzen(train,target\_var\_name,test,h)  print(prediction\_parametric) print(prediction\_parzen)  tn, fp, fn, tp = confusion\_matrix(test[target\_var\_name],  prediction\_parametric.astype(int)).ravel() acc\_parametric=(tn+tp)/(tp+tn+fp+fn)  tn, fp, fn, tp = confusion\_matrix(test[target\_var\_name], prediction\_parzen.astype(int)).ravel()  acc\_parzen=(tn+tp)/(tp+tn+fp+fn) return(acc\_parametric,acc\_parzen)  res\_microarray=compare\_methods(X\_train,X\_test,'target',0.5)  (train1,test1)=generate\_dataset(7)  (train2,test2)=generate\_dataset(11,'logistic') (train3,test3)=generate\_dataset(7,'lognormal')  res1=compare\_methods(train1,test1,'y') res2=compare\_methods(train2,test2,'y') res3=compare\_methods(train3,test3,'y') |

We represented dataset accuracy methods set by different distribution as follow:

Data set

parametric

Parzen(h=0.1)

Micro array

0.925

0.5

Artificial Normal

0.925

0.955

Artificial logistic

0.5

0.6

Artificial exponential

0.47

0.87

Artificial lognormal

0.46

0.7

Comment:

In case we increment value of h parameter to h=0.5 , we will see a significant modification on microarray dataset accuracy up to 0.9. This will affect the prediction.

## Conclusion

The parameters setting must follow a certain balance to avoid the performance (for example we don’t have interest in setting them at a very high or a very low level).

The accuracy is much better while using parametric method.