**PRACTICAL NO – 13**

**TITLE- Naive Bayes**

**THEORY-**- The Parzen-window technique is a widely used non-parametric approach to estimate a probability density function p(x) for a specific point x from a sample xn that doesn't require any knowledge or assumption about the underlying distribution.

**Putting it in context - Where would this method be useful?**

A popular application of the Parzen-window technique is to estimate the class-conditional densities (or also often called 'likelihoods') p(x|ωi) in a supervised pattern classification problem from the training dataset (where x is a multi-dimensional sample that belongs to particular class ωi).

Imagine that we are about to design a Bayes classifier for solving a statistical pattern classification task using Bayes's rule:

P(ωi|x) = [p(x|wi) ⋅P(ωi)]/p(x)]

⇒posterior probability = likelihood \* prior probability evidence

However, it becomes much more challenging, if we don't have prior knowledge about the underlying parameters that define the model of our data.

Imagine we are about to design a classifier for a pattern classification task where the parameters of the underlying sample distribution are not known. Therefore, we wouldn't need the knowledge about the whole range of the distribution; it would be sufficient to know the probability of the particular point, which we want to classify, in order to make the decision. And here we are going to see how we can estimate this probability from the training sample.  
However, the only problem of this approach would be that we would seldom have exact values - if we consider the histogram of the frequencies for an arbitrary training dataset. Therefore, we define a certain *region* (i.e., the **Parzen-window**) around the particular value to make the estimate.

**PROBLEM STATEMENT**- To develop an algorithm for Parzen Window Classifier

**ALGORITHM-**

1. **Import libraries**
2. **Create Gaussian Point**
3. **Define the Parzen estimation function**
4. **Visualize the actual vs predicted**
5. **END**

**CODE-**

# -\*- coding: utf-8 -\*-

"""

Spyder Editor

This is a test of basic NN

Author - @RoKu

"""

import numpy as np

from matplotlib import pyplot as plt

import operator

import prettytable

from matplotlib.mlab import bivariate\_normal

from mpl\_toolkits.mplot3d import Axes3D

# Generate 10,000 random 2D-patterns

mu\_vec = np.array([0,0])

cov\_mat = np.array([[1,0],[0,1]])

x\_2Dgauss = np.random.multivariate\_normal(mu\_vec, cov\_mat, 10000)

f, ax = plt.subplots(figsize=(7, 7))

ax.scatter(x\_2Dgauss[:,0], x\_2Dgauss[:,1],

marker='o', color='green', s=4, alpha=0.3)

plt.title('10000 samples randomly drawn from a 2D Gaussian distribution')

plt.ylabel('x2')

plt.xlabel('x1')

ftext = 'p(x) ~ N(mu=(0,0)^t, cov=I)'

plt.figtext(.15,.85, ftext, fontsize=11, ha='left')

plt.ylim([-4,4])

plt.xlim([-4,4])

plt.show()

fig = plt.figure(figsize=(10, 7))

ax = fig.gca(projection='3d')

x = np.linspace(-5, 5, 200)

y = x

X,Y = np.meshgrid(x, y)

Z = bivariate\_normal(X, Y)

surf = ax.plot\_surface(X, Y, Z, rstride=1,

cstride=1, cmap=plt.cm.coolwarm,

linewidth=0, antialiased=False

)

ax.set\_zlim(0, 0.2)

ax.zaxis.set\_major\_locator(plt.LinearLocator(10))

ax.zaxis.set\_major\_formatter(plt.FormatStrFormatter('%.02f'))

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.set\_zlabel('p(x)')

plt.title('Bivariate Gaussian distribution')

fig.colorbar(surf, shrink=0.5, aspect=7, cmap=plt.cm.coolwarm)

plt.show()

def pdf\_multivariate\_gauss(x, mu, cov):

assert(mu.shape[0] > mu.shape[1]),\

'mu must be a row vector'

assert(x.shape[0] > x.shape[1]),\

'x must be a row vector'

assert(cov.shape[0] == cov.shape[1]),\

'covariance matrix must be square'

assert(mu.shape[0] == cov.shape[0]),\

'cov\_mat and mu\_vec must have the same dimensions'

assert(mu.shape[0] == x.shape[0]),\

'mu and x must have the same dimensions'

part1 = 1 / ( ((2\* np.pi)\*\*(len(mu)/2)) \* (np.linalg.det(cov)\*\*(1/2)) )

part2 = (-1/2) \* ((x-mu).T.dot(np.linalg.inv(cov))).dot((x-mu))

return float(part1 \* np.exp(part2))

def parzen\_window\_est(x\_samples, h=1, center=[0,0,0]):

dimensions = x\_samples.shape[1]

assert (len(center) == dimensions), 'Number of center coordinates have to match sample dimensions'

k = 0

for x in x\_samples:

is\_inside = 1

for axis,center\_point in zip(x, center):

if np.abs(axis-center\_point) > (h/2):

is\_inside = 0

k += is\_inside

return (k / len(x\_samples)) / (h\*\*dimensions)

# generate a range of 400 window widths between 0 < h < 1

h\_range = np.linspace(0.001, 1, 400)

# calculate the actual density at the center [0, 0]

mu = np.array([[0],[0]])

cov = np.eye(2)

actual\_pdf\_val = pdf\_multivariate\_gauss(np.array([[0],[0]]), mu, cov)

# get a list of the differnces (|estimate-actual|) for different window widths

parzen\_estimates = [np.abs(parzen\_window\_est(x\_2Dgauss, h=i, center=[0, 0])

- actual\_pdf\_val) for i in h\_range]

# get the window width for which |estimate-actual| is closest to 0

min\_index, min\_value = min(enumerate(parzen\_estimates), key=operator.itemgetter(1))

print('Optimal window width for this data set: ', h\_range[min\_index])

p1 = parzen\_window\_est(x\_2Dgauss, h=h\_range[min\_index], center=[0, 0])

p2 = parzen\_window\_est(x\_2Dgauss, h=h\_range[min\_index], center=[0.5, 0.5])

p3 = parzen\_window\_est(x\_2Dgauss, h=h\_range[min\_index], center=[0.3, 0.2])

mu = np.array([[0],[0]])

cov = np.eye(2)

a1 = pdf\_multivariate\_gauss(np.array([[0],[0]]), mu, cov)

a2 = pdf\_multivariate\_gauss(np.array([[0.5],[0.5]]), mu, cov)

a3 = pdf\_multivariate\_gauss(np.array([[0.3],[0.2]]), mu, cov)

results = prettytable.PrettyTable(["", "predicted", "actual"])

results.add\_row(["p([0,0]^t",p1, a1])

results.add\_row(["p([0.5,0.5]^t",p2, a2])

results.add\_row(["p([0.3,0.2]^t",p3, a3])

##############################################

### Predicted bivariate Gaussian densities ###

##############################################

fig = plt.figure(figsize=(10, 7))

ax = fig.gca(projection='3d')

X = np.linspace(-5, 5, 100)

Y = np.linspace(-5, 5, 100)

X,Y = np.meshgrid(X,Y)

Z = []

for i,j in zip(X.ravel(),Y.ravel()):

Z.append(parzen\_window\_est(x\_2Dgauss, h=h\_range[min\_index], center=[i, j]))

Z = np.asarray(Z).reshape(100,100)

surf = ax.plot\_surface(X, Y, Z, rstride=1, cstride=1, cmap=plt.cm.coolwarm,

linewidth=0, antialiased=False)

ax.set\_zlim(0, 0.2)

ax.zaxis.set\_major\_locator(plt.LinearLocator(10))

ax.zaxis.set\_major\_formatter(plt.FormatStrFormatter('%.02f'))

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.set\_zlabel('p(x)')

plt.title('Predicted bivariate Gaussian densities')

fig.colorbar(surf, shrink=0.5, aspect=7, cmap=plt.cm.coolwarm)

###########################################

### Actual bivariate Gaussian densities ###

###########################################

fig = plt.figure(figsize=(10, 7))

ax = fig.gca(projection='3d')

x = np.linspace(-5, 5, 100)

y = x

X,Y = np.meshgrid(x, y)

Z = bivariate\_normal(X, Y)

surf = ax.plot\_surface(X, Y, Z, rstride=1, cstride=1, cmap=plt.cm.coolwarm,

linewidth=0, antialiased=False)

ax.set\_zlim(0, 0.2)

ax.zaxis.set\_major\_locator(plt.LinearLocator(10))

ax.zaxis.set\_major\_formatter(plt.FormatStrFormatter('%.02f'))

fig.colorbar(surf, shrink=0.5, aspect=7, cmap=plt.cm.coolwarm)

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.set\_zlabel('p(x)')

plt.title('Actual bivariate Gaussian densities')

plt.show()

print(results)

**RESULTS**-

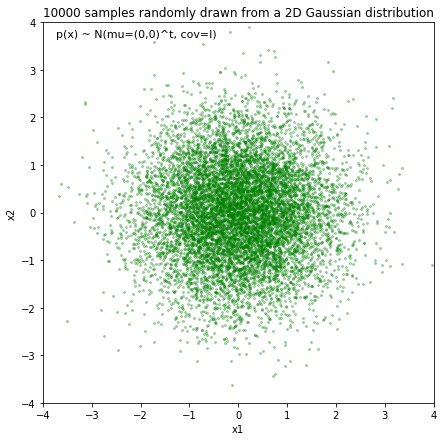
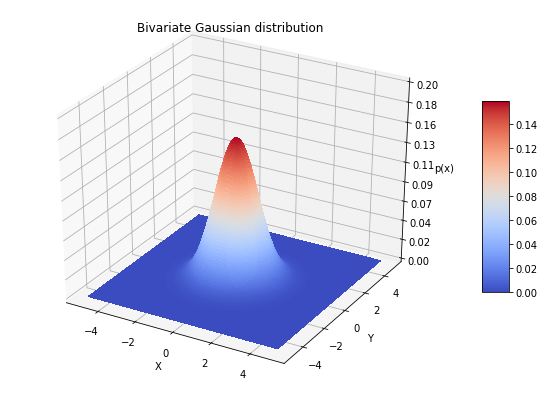
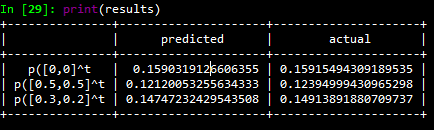
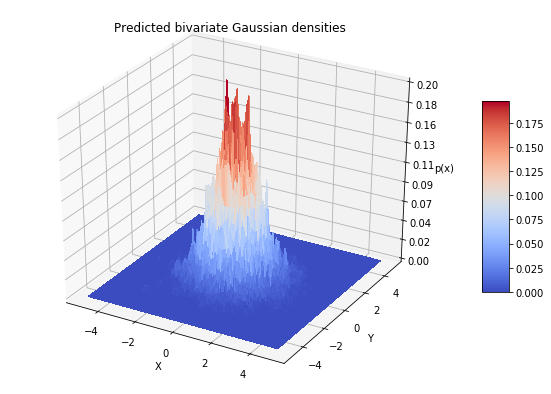


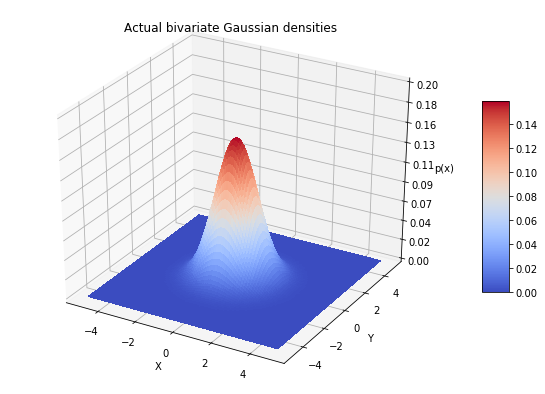
Fig.1- Visualization.



**Optimal window width for this data set: 0.5343007518796993**







**CONCLUSIONS-**

**Computation and performance**

One of the biggest drawbacks of the Parzen-window technique is that we have to **keep our training dataset** around for estimating (computing) the probability densities. For example, if we design a Bayes’ classifier and use the Parzen-window technique to estimate the class-conditional probability densities *p(****x****i | ωj)*, the computational task requires the whole training dataset to make the estimate for every point

* this is a disadvantage of non-parametric approaches in general.  
  In contrast, parametric methods, e.g., the Maximum Likelihood Estimate (MLE) or Bayesian Estimation (BE) only require the training data to calculate the value of the parameters. Once, the parameters were obtained from the training dataset, it can be discarded and re-computation is not required for classifying data in the test dataset. However, in Bayesian Learning (BL), new samples can be incorporated to improve the estimated parameter values.

|  |  |
| --- | --- |
| **SUBMISSION DATE**- | **SIGN OF COURSE INSTRUCTOR**- |
| **ROLL NO OF THE STUDENT-**  **I-62** | **NAME OF THE STUDENT-**  **Rohit Kulkarni** |