Department of Computer Engineering
University of Peradeniya

CO 544 Machine Learning and Data Mining
Lab 01*

January 13, 2022

Objective

To demonstrate two use cases (sampling and projections) of the property of multivariate Gaussian densities and provide students hands on experience in Python modules, Numpy and Matplotlib.

Preliminaries

For laboratory exercises, we use Python programming language in a **Google Colab Notebook** environment. Snippets of code will be provided along with tasks you are being asked to do. These are provided to help you get started. These should not be assumed to be complete working pieces of software.
Familiarise yourselves with the desktop computing environment in the laboratory. Start a cmd window, navigate to a directory where you can save your files and activate the notebook by typing jupyter notebook. Here is how to get started with some commands that we will often find on top of the notebook in this course:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

1. Linear Algebra

Here is a quick refresher:

1. Scalar (dot) product: $a = x.y$

   ```
   x = np.array([1, 5])
   y = np.array([2, 3])
   a = np.dot(x, y)
   print(a)
   ```

2. Linear combination of two vectors: $d = 2x + 5y$

   ```
   d = 2*x + 2*y
   ```

3. Norm of a vector: $b = \sqrt{x(1)^2 + x(2)^2}$

*Adopted from a similar course at University of Southampton, UK

```
b = np.sqrt(x[0]**2+x[1]**2)
c = np.linalg.norm(x)
print(b, c)
```

4. Angle between two non-zero vectors: $\theta = \arccos\left(\frac{x.y}{\|x\|.\|y\|}\right)$

```
theta_radians = np.arccos(np.dot(x, y)/(np.linalg.norm(x)*np.linalg.norm(y)))
theta_degrees = np.degrees(theta_radians)
print(theta_degrees)
```

5. Matrix vector multiplication: $v = Au$

```
A = np.array([[1, -1, 2], [0, -3, 4]], dtype=float)
u = np.array([2, 1, 3])
v = np.dot(A, u)
print(v)
```

6. Solve a linear matrix equation, or system of linear scalar equations:
   Consider the system of equations $2x_1 + 5x_2 = 3$ and $3x_1 - 5x_2 = 2$

```
d = np.array([[2, 5], [3, -5]])
e = np.array([3, 2])
results = np.linalg.solve(d, e)
print(results)
```

7. Inverse of a matrix: $I = A^{-1}$

```
I = np.linalg.inv(d)
```

8. Trace of a matrix

```
B = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])
trace = np.trace(B)
```

9. Determinant of a matrix: $det(B)$ or $|B|$

```
B_det = np.linalg.det(B)
```

10. Eigenvalues and Eigenvectors: $Au = \lambda u$

```
a, M = np.linalg.eig(B)
print(a)
print(M)
print(np.dot(M[:,0], M[:,1]))
print(M @ M.T)
```

@ -an operator ,performs actual matrix multiplication

Where, `a` represents the vector containing the eigenvalues and `M` represents the array containing the corresponding eigenvectors.

11. Singular Value Decomposition (SVD)

What do you observe for the very last command in the above exercise (i.e. `print(U @ U.T)`? Can you formally prove this is what you would expect for the specific structure in matrix B?
Advanced material: Search for a document with title `The Matrix Cookbook` which is a beautiful collection of matrix identities and results.

2. Random Numbers and Univariate Distributions

Generate 1000 uniform random numbers and plot a histogram.

```
x = np.random.rand(1000, 1)
plt.figure(figsize=(10, 5))
n, bins, patches = plt.hist(x, bins=20, color='b', alpha=0.8, rwidth=0.8)
```

Note how the choice of the number of bins and the number of samples affects the histogram you observe. Here is a bit of code to help you study this.

```
MaxTrials = 10
NumSamples = 200
NumBins = 20
for trial in range(MaxTrials):
    x = np.random.rand(NumSamples, 1)
    counts, bins, patches = plt.hist(x, NumBins)
    plt.clf()
    print("Variation within bin counts: ", np.var(counts/NumSamples))
```

Repeat the above with 1000 random numbers drawn from a Gaussian distribution of mean 0 and standard deviation 1 using `x = random,randn(1000, 1);`.

Now try the following where we add and subtract uniform random numbers:

```
N = 1000
x1 = np.zeros(N)
for n in range(N):
    x1[n] = sum(np.random.rand(12, 1)) - np.sum(np.random.rand(12, 1));
plt.hist(x1, 40, color='b', alpha=0.8, rwidth=0.8)
plt.xlabel('Bin', FontSize=16)
plt.ylabel('Counts', FontSize=16)
plt.grid(True)
plt.title('Histogram', FontSize=16)
```

What do you observe? How does the resulting histogram change when you change the number of numbers you add/subtract? Is there a theorem that explains it?

3. Uncertainty in Estimation

Much of what we do in Machine Learning is estimating parameters of a model from a finite set of data. Here is why one might think having more data is a good thing: Consider estimating the variance of a uni-variate Gaussian distribution from samples drawn from it. If we did this from different relations of the samplings, we would expect slightly different answers (every time we do it). We would, of course, like this variation to be small. Figure 1 shows how the variance of the estimation varies with sample size. Here is a snippet of code to study this.

```
sampleSizeRange = np.linspace(100, 200, 40)
plotVar = np.zeros(len(sampleSizeRange))
for sSize in range(len(sampleSizeRange)):
    numSamples = np.int(sampleSizeRange[sSize])
    MaxTrial=2000
    vStrial=np.zeros(MaxTrial)
```

```
    for trial in range(MaxTrial):
        xx = np.random.randn(numSamples,1)
        vStrial[trial] = np.var(xx)
    plotVar[sSize] = np.var(vStrial)

plt.plot(sampleSizeRange, plotVar)
```

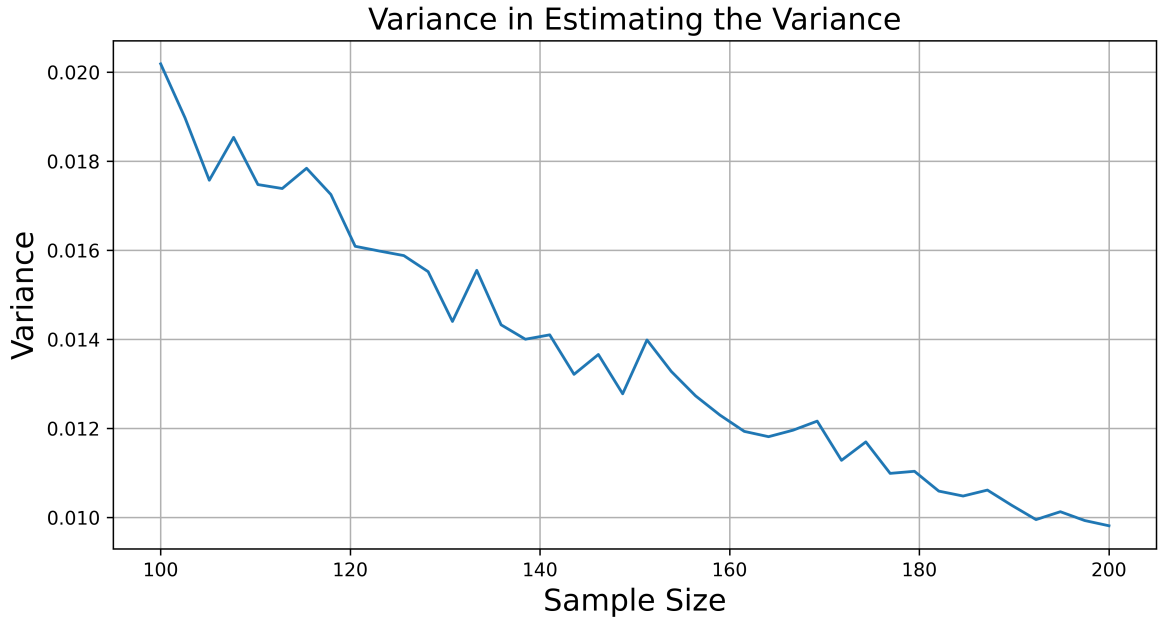


Figure 1: Uncertainty in Estimation

4. Bivariate Gaussian Distribution

Here are two functions to help us calculate and plot contours on a multivariate Gaussian density.

```
def gauss2D(x, m, C):
    Ci = np.linalg.inv(C)
    dC = np.linalg.det(C1)
    num = np.exp(-0.5 * np.dot((x-m).T, np.dot(Ci, (x-m))))
    den = 2 * np.pi * dC

    return num/den

def twoDGaussianPlot (nx, ny, m, C):
    x = np.linspace(-5, 5, nx)
    y = np.linspace(-5, 5, ny)
    X, Y = np.meshgrid(x, y, indexing='ij')

    Z = np.zeros([nx, ny])
    for i in range(nx):
        for j in range(ny):
            xvec = np.array([X[i,j], Y[i,j]])
            Z[i,j] = gauss2D(xvec, m, C)
    return X, Y, Z
```

Let us now plot these for three different values of means and covariance matrices.

```
nx, ny = 50, 40
plt.figure(figsize=(6,6))

m1 = np.array([0, 2])
C1 = np.array([[2, 1], [1,2]], np.float32)
Xp, Yp, Zp = twoDGaussianPlot(nx, ny, m1, C1)
plt.contour(Xp, Yp, Zp, 3)
plt.grid(True)
```

Explain
Maximum point (Mean)
Distribution (shape)

On the same plot draw contours on the two bivariate Gaussian densities:

$$m2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \quad C2 = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \quad and \quad m3 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}, \quad C2 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

5. Sampling from a Multivariate Gaussian Distribution

Consider the covariance matrix $C = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$.

Factorize this into a lower trinagular matrix and its transpose (Cholosky decomposition), $A^t A = C$, and confirm the factorization is correct by multiplying.

```
C = [[2, 1], [1, 2]]
print(C)
A = np.linalg.cholesky(C)
print(A)
print(A @ A.T)
```

Generate 10000 bivariate Gaussian random numbers by `X = np.random.randn(1000, 2);` Transform each of the two dimensional vectors (rows of X) by `Y = X @ A`.

```
X = np.random.randn(10000, 2)
Y = X @ A
print(X.shape)
print(Y.shape)
```

Now draw a scatter plot of `X` and `Y`.

```
plt.scatter(Y[:, 0], Y[:, 1], s=3, c='m')
plt.scatter(X[:, 0], X[:, 1], s=3, c='c')
plt.grid(True)
plt.title("Scatter of Isotropic and Correlated Gaussian Densities")
```

What do you observe?

6. Distribution of Projections

Construct a vector $u = [\sin\theta \cos\theta]$, parameterized by the variable θ.

```
theta = np.pi / 3
u = [np.sin(theta), np.cos(theta)]
print("The vector: ", u)
print("Sum of squares: ", u[0]**2 + u[1]**2)
print("Degrees: ", theta*180 / np.pi)
```

Compute the variance of projections of the data in X along this direction:

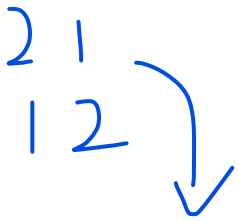
```
yp = Y @ u
print(yp.shape)
print("Projected Variance: ", np.var(yp))
```

Plot how this projected variance changes as a function of θ:

```
nPoints = 50;
pVars = np.zeros(nPoints)
thRange = np.linspace(0, 2*np.pi, nPoints)

for n in range(nPoints):
  theta = thRange[n]
  u = [np.sin(theta), np.cos(theta)]
  pVars[n] = np.var( Y @ u )

plt.plot(pVars)
plt.grid(True)
plt.xlabel("Direction", fontsize=14)
plt.ylabel("Variance of Projections", fontsize=14)
```



Explain what you observe by calculating the eigenvectors of the covariance matrix.
How does what you have done above differ for $C = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$.
Export the figures for inclusion in a report. Try `plt.savefig`.

Report

Describe the work you have done as a short report by answering all the discussion points/questions were covered during the lab session. Submit a PDF file no longer than four pages. Rename it as XXYYYlab01.pdf, where XX indicates your batch and YYY indicates your registration number.