School of Electronics and Computer Science
University of Southampton

COMP3223/6245(2024/25): Foundations of Machine Learning **Lab One**

Issue	30 September 2024
Deadline	07 October 2024

Objective

- Review some concepts from linear algebra, probability and statistics
- To study properties of multi-variate Gaussian densities (sampling and projection):

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{m}, C), \; \boldsymbol{y} = A\,\boldsymbol{x} \implies \boldsymbol{y} \sim \mathcal{N}(A\boldsymbol{m}, A\,C\,A^T)$$

1 Getting started

- For exercises in this module, we will use `Python` programming language in a `Jupyter` notebook environment. Snippets of code will be provided, along with tasks you are required to carry out. What is provided is purely guides to help you get started and should not be taken to be complete working software.
- For most of the material covered in this module, we will use Pattern Recognition and Machine Learning, by C.M.Bishop as reference material. This book is available in `pdf` format for free. Find and download trhe book. We will not study the book cover to cover because this is only a 15-credit module, but will point to material drawn from sections of it.

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
print("Hello World!")
```

2 Linear Algebra

Here is a quick refresher on some manipulations on vectors and matrices we will need in this module along with some simple commands in Python:

1. Scalar product of two vectors: $a = \boldsymbol{x}^T\,\boldsymbol{y}$
2. Vector norm: $b = \sqrt{\sum_{i=1}^{d} x^2(i)}$
3. Angle between two vectors
4. Symmetric matrix: $B^T = B$
5. Rank of a matrix as the number of linearly independent rows/columns
6. Matrix vector multiplication: $A\,\boldsymbol{x}$
7. Quadratic form: $\boldsymbol{x}^T\,A\,\boldsymbol{x}$
8. Trace of a matrix B: $\sum_i B_{ii}$

9. Determinant of a matrix, denoted $\texttt{det}B$ or $|B|$

10. Eigenvalues and eigenvectors: $B\,\boldsymbol{u} = \lambda\boldsymbol{u}$

11. Advanced topic: Singular Value Decomposition (SVD)

12. Please try the following to get started. At each step, you should ask yourself if you notice any specific property of matrices, vector etc. you recall from previous phase of your study.

Scalar product and angle between vectors

```
# Define vectors
x = np.array([1, 2])
y = np.array([-2, 1])

# Scalar product and vector norm
a = np.dot(x, y)
b = np.linalg.norm(x)
c = np.sqrt(x[0]**2 + x[1]**2)

print("%5.2f %5.2f %5.2f"%(a, b, c))

# angle between vectors
theta = np.arccos(np.dot(x,y) / (np.linalg.norm(x) * np.linalg.norm(x)))
print("Angle in Radians: %5.2f  in Degrees: %5.2f"%(theta, theta*180/(np.pi)))
```

Transpose & Inverse of Matrices

```
# Define a square matrix
B = np.array([[3,2,1], [2,6,5], [1,5,9]], dtype=float)
print(B)

# Transpose, Inverse
print(B - B.T)
print(B @ np.linalg.inv(B))
```

Quadratic function of a vector

```
# Quadratic form, we will see this often
z = x = np.array([1, 2, 3])
B = np.array([[3,2,1], [2,6,5], [1,5,9]], dtype=float)
print("%6.3f"%(z.T @ B @ z))
print(z.shape, B.shape)
```

Eigenvalues and eigenvectors

```
# Trace, determinant and Eigenvalues
print(np.trace(B))
print(np.linalg.det(B))

D, U = np.linalg.eig(B)
print(D)


# Eigenvectors
print(U)
print("%5.2f"%(np.dot(U[:,0], U[:,1])))
print("%5.2f"%(np.dot(U[:,0], U[:,0])))
```

What do you observe for the last command above (i.e. print(np.dot(U[:,0], U[:,1])))? Can you formally prove that this is the result you would expect for the specific structure in the matrix B?

13. Now, for some advanced material and fun, find the following two items:

    - A document with title `The Matrix Cookbook` written by K.B.Petersen and M.S.Pedersen. This is a neat resource with all the basics we need and much more. A very useful reference material to have around.
    - "*It had to be U - the SVD song*" on `youtube`, which is a nice piece of art that tells you what Singular Value Decomposition is and gives an example of where it is used.

3 Probability and Statistics

Histogram

Generate 1000 uniform random numbers and plot a histogram.

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

x = np.random.rand(1000,1)
x = np.random.rand(1000,1)

fig, ax = plt.subplots(nrows=1, ncols=2,figsize=(10,4))
n1bins, n2bins = 4, 40
ax[0].hist(x, bins=n1bins)
ax[0].set_ylim(0,250)
ax[0].set_xlabel("Bins", fontsize=16)
ax[0].set_ylabel("Count", fontsize=16)
ax[0].tick_params(axis='both', which='major', labelsize=14)
ax[0].set_title("Histogram: bins=%4d"%(n1bins), fontsize=16)

ax[1].hist(x, bins=n2bins)
ax[1].set_ylim(0,250)
ax[1].set_xlabel("Bins", fontsize=16)
ax[1].set_ylabel("Count", fontsize=16)
ax[1].tick_params(axis='both', which='major', labelsize=14)
ax[1].set_title("Histogram: bins=%4d"%(n2bins), fontsize=16)

plt.savefig("histograms_uniform.png")
plt.tight_layout()
```

Think through the following:

- Though the data is from a uniform distribution, the histogram does not appear flat. Why?
- Every time you run it, the histogram looks slightly different? Why?
- How do the above observations change (if so how) if you had started with more data?

Transformation

Let us now add and subtract some uniform random numbers:

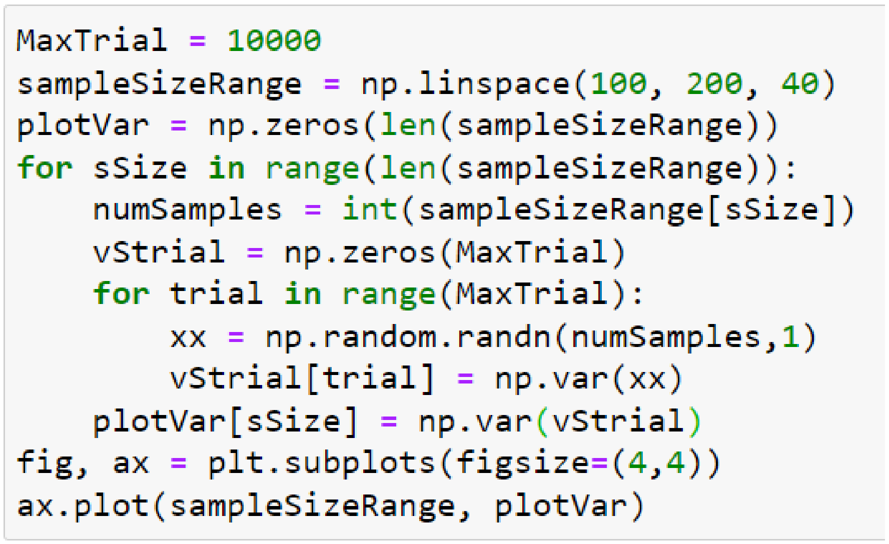
```
N = 1000
x1 = np.zeros(N)
for n in range(N):
    x1[n] = np.sum(np.random.rand(12,1)) - np.sum(np.random.rand(12,1))
fig, ax = plt.subplots(figsize=(5,5))
ax.hist(x1, 20)
```

What do you observe? How does the resulting histogram change when you change the number of uniform random numbers you add and subtract? Is there a theory that explains your observation?

Uncertainty in Estimation

Much of what we study in machine learning has to do with estimating parameters of models from a finite set of data. Consider estimating the variance of a uni-variate Gaussian density using samples drawn from it. When we estimate the variance from different sets of samples ("different realizations of a process"), the answer we get each time will be slightly different. But if we had more data, we would expect this variation to be small.

Let's see if this is true. In the code below, we explore different sample sizes (`sSize`) taken from an array(`sampleSizeRange`), and with each sample size, run several trials (`trial`) running to (`MaxTrial`). On each trial, we generate uniform samples into array `xx` and compute its variance. We store these in `plotVar` and plot.



```
MaxTrial = 10000
sampleSizeRange = np.linspace(100, 200, 40)
plotVar = np.zeros(len(sampleSizeRange))
for sSize in range(len(sampleSizeRange)):
    numSamples = int(sampleSizeRange[sSize])
    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)
fig, ax = plt.subplots(figsize=(4,4))
ax.plot(sampleSizeRange, plotVar)
```

4 Multi-variate Gaussian Distribution

We will come across the multi-variate Gaussian distribution, defined in some $d-$ dimensional space quite a lot in this module. We can study some properties of this with $d = 2$, bi-variate, because in two dimensions we can visualize some of these.

```
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
```

This is a function of two variables. You can plot contours on this function or visualize it as a three dimensional surface plot.

```
# Plot contours
#
nx, ny = 50, 40
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, 5)
```

Draw contours of the following distributions: $\mathcal{N}\left(\begin{bmatrix} 2.4 \\ 3.2 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}\right)$, $\mathcal{N}\left(\begin{bmatrix} 1.2 \\ 0.2 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}\right)$ and $\mathcal{N}\left(\begin{bmatrix} 2.4 \\ 3.2 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\right)$

Sampling from a multi-variate Gaussian

Suppose we are tasked with drawing several samples from a multi-variate Gaussian density with mean $\boldsymbol{m} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and covariance matrix $C = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$. Here is a way to do this using the properties of multi-variate Guassians we have learnt:

- Factorize the covariance matrix into a lower triangular matrix and its transpose, $C = A\,A^T$; multiply to see if the factorization is correct:

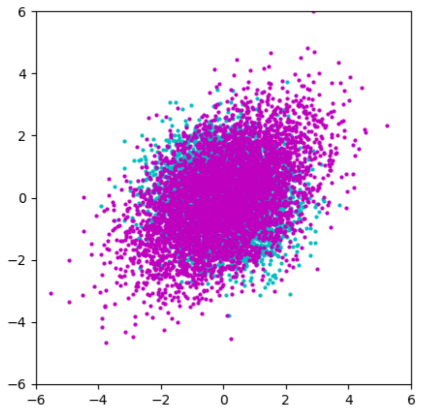
```
C = np.array([[2.0,1.0], [1.0,2]])
A = np.linalg.cholesky(C)
print(A)
print(A @ A.T)
```

- Generate 5000 bivariate Gaussian random data by `X = np.random.randn(5000,2)` and transform each of the data (rows of X) by $Y = X\,A.Y$. Note our purpose is to take each of the items of data we generated and transform it by multiplying by A; i.e. $\boldsymbol{y} = \boldsymbol{Ax}$. We do this by storing all of them in matrix X and multiplying in a single shot to get Y (avoiding a `for` loop.

```
X = np.random.randn(1000,2)
Y = X @ A.T
print(X.shape)
print(Y.shape)
```

- Draw scatter plots of X and Y

```
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(X[:,0], X[:,1], c="c", s=4)
ax.scatter(Y[:,0], Y[:,1], c="m", s=4)
ax.set_xlim(-6, 6)
ax.set_ylim(-6, 6)
```



Distribution of Projections

- Construct a vector $\boldsymbol{u} = [\sin\theta \;\; \cos\theta]^T$, parameterized by the variable θ.

```
theta = np.pi/3
u = [np.sin(theta), np.cos(theta)]
print("The vector: ", u)
print("Magnitude : ", np.sqrt(u[0]**2 + u[1]**2))
print("Angle     : ", theta*180/np.pi)
```

- Compute the variance of projected data along this direction

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

- Now, using the above write a program that plots the variance of the projected data as you change θ over the range 0 to 2π.

```
# Store projected variances in pVars & plot
#
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)

fig, ax = plt.subplots(figsize=(5,3))
ax.plot(pVars)
```

  What are the maxima and minima of the resulting plot?

- Compute the eigenvalues an eigenvectors of the covariance matrix C
- Can you see a relationship between the eignevalues and eigenvectors and the maxima and minima of the way the projected variance changes?
- The shape of the graph might have looked sinusoidal for this two dimensional problem. Can you analytically confirm if this might be true?

Report

Upload a report of **no more than four pages** describing your work. Make sure you write your name and email on the work you upload.