Department of Computer Engineering
University of Peradeniya

Machine Learning Laboratory

Guest Lecture
Professor Mahesan Niranjan
University of Southampton
January 2022

1 Objectives

1. To study two uses of properties of multi-variate Gaussian densities (sampling and projection):
$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{m}, C)\,,\ \boldsymbol{y} = A\,\boldsymbol{x} \implies \boldsymbol{y} \sim \mathcal{N}(A\boldsymbol{x},\, A\,C\,A^T)$$
2. To implement the perceptron learning algorithm

2 Quick Preliminaries

For exercises in this module, we will use `Python` programming language in a `Jupyter` notebook environment with snippets of code provided as guides to help you get started. These code snippets should be seen as pseudo-code not as complete working software.

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

Here is a quick refresher on some manipulations on vectors and matrices we will need 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.

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

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

theta = np.arccos(np.dot(x,y) / (np.linalg.norm(x) * np.linalg.norm(x)))

B = np.array([[3,2,1], [2,6,5], [1,5,9]], dtype=float)
print(B)
print(B - B.T)

z = np.random.rand(3)
v = B @ z
print(v.shape)

print(z.T @ B @ z)

print(np.trace(B))
print(np.linalg.det(B))

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

print(np.dot(U[:,0], U[:,1]))
print(U @ U.T)
```

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.
    - "*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 Random Numbers and Uni-variate Densities

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?

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?

4 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:

```
MaxTrial = 2000
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((plotVar))
```

5 Bi-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 $\mathcal{N}\left(\begin{bmatrix} 2.4 \\ 3.2 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\right)$

6 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: $C = A\,A^T$:

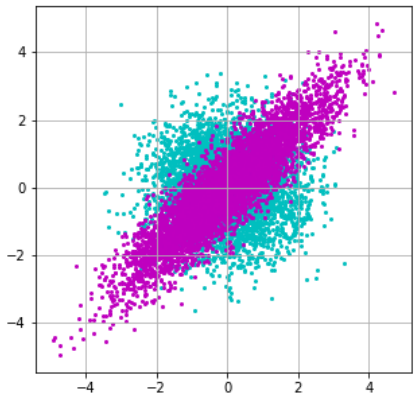
```
C = np.array([[2.0,1.0], [1.0,2]])
A = np.linalg.cholesky(C)
print(A @ A.T)
```

- Generate 5000 bivariate Gaussian random data by `X = np.random.randn(5000,2)` and transform each of the data (rows of X) by $Y = X\,A$

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



7 Distribution of Projections

- Construct a vector $\boldsymbol{u} = [\sin\theta \quad \cos\theta]^T$, 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 $2\,\pi$.

```
# 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?

8 Perceptron

The perceptron algorithm computes a linear classifier using a stochastic error correcting learning algorithm. It is simple and has much historic relevance to this subject and makes a good starting point to learn more sophisticated models and algorithms.

Implementation

Generate 100 samples each from two bi-variate Gaussian densities with distinct means $\boldsymbol{m}_1 = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$ and $\boldsymbol{m}_2 = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$, and identical covariance matrix $\boldsymbol{C} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$. (Hint: Use material developed in Lab One).

```
NumDataPerClass = 200

# Two-class problem, distinct means, equal covariance matrices
#
m1 = [[0, 5]]
m2 = [[5, 0]]
C  = [[2, 1], [1, 2]]

# Set up the data by generating isotropic Guassians and
# rotating them accordingly
#
A = np.linalg.cholesky(C)

U1 = np.random.randn(NumDataPerClass,2)
X1 = U1 @ A.T + m1

U2 = np.random.randn(NumDataPerClass,2)
X2 = U2 @ A.T + m2
```

The distribution of your data should look like what is shown in Fig. 1. Note the data are linearly separable (*i.e.* a linear class boundary will classify the data correctly).

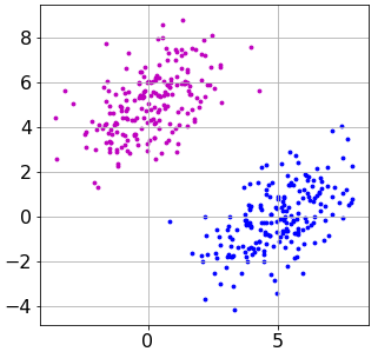


Figure 1: Training Data, sampled from two Bivariate Gaussian Densities

We will now train a perceptron algorithm to classify this data. A perceptron is a linear classifier whose training is done by error correction. If the weights of the perceptron are denoted $\boldsymbol{w}$ and the input features are in vector $\boldsymbol{x}$, a perceptron decision function assigns the data to one class or the other depending on whether $\boldsymbol{w}^T \boldsymbol{x} \lessgtr 0$.
The algorithm is as follows:

Inputs $\{\boldsymbol{x}_n, y_n\}_{n=1}^N$, $\boldsymbol{x}_n \in \mathcal{R}^d$, $y_n \in (-1, +1)$
Initialize weights $\boldsymbol{w}$
Generate index $0 \leq \tau \leq N$ at random
If $(y^{(\tau)} \boldsymbol{w}^T \boldsymbol{x}^{(\tau)} \leq 0)$
. $\quad \boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} + \alpha\, y^{(\tau)}\, \boldsymbol{x}^{(\tau)}$

Note the scalar product $\boldsymbol{w}^T \boldsymbol{x}^{(\tau)}$ times the target $y^{(\tau)}$ being positive over all the data is our goal. Upon seeing a random data, we only update if this data is misclassified. This is why we refer to this algorithm as a *stochastic error correcting* algorithm. Stochastic because we are lookign at random presentations of data and error correcting because we only update when the current example is misclassified.
We will derive this in a formal setting after we have studied regression, by setting up an error function and optimising it (minimising it) by gradient descent.
Here are snippets of code to help you do this.

1. For simplicity, I have assumed the following (some of which you are free to change and study the effect):

- There is an equal number of data `NumDataPerClass` in each class
- We use an equal partition of the data into training and test sets, taken at random.

2. Concatenate data from two classes into one array. (Fig. 1).

```
X = np.concatenate((X1, X2), axis=0)
```

3. Setting up targets (labels): we set $+1$ and -1 as labels to indicate the two classes.

```
labelPos = np.ones(NumDataPerClass)
labelNeg = -1.0 * np.ones(NumDataPerClass)
y = np.concatenate((labelPos, labelNeg))
```

4. Partitioning the data into training and test sets

```
rIndex = np.random.permutation(2*NumDataPerClass)
Xr = X[rIndex,]
yr = y[rIndex]

# Training and test sets (half half)
#
X_train = Xr[0:NumDataPerClass]
y_train = yr[0:NumDataPerClass]
X_test  = Xr[NumDataPerClass:2*NumDataPerClass]
y_test  = yr[NumDataPerClass:2*NumDataPerClass]
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

Ntrain = NumDataPerClass;
Ntest  = NumDataPerClass;
```

5. Calculating the percentage of correctly classified examples

```
def PercentCorrect(Inputs, targets, weights):
    N = len(targets)
    nCorrect = 0
    for n in range(N):
        OneInput = Inputs[n,:]
        if (targets[n] * np.dot(OneInput, weights) > 0):
            nCorrect +=1
    return 100*nCorrect/N
```

6. Iterative error correcting learning

```
# Perceptron learning loop
#

# Random initialization of weights
#
w = np.random.randn(2)
print(w)

# What is the performance with the initial random weights?
#
print('Initial Percentage Correct: %6.2f' %(PercentCorrect(X_train, y_train, w)))

# Fixed number of iterations (think of better stopping criterion)
#
MaxIter=1000

# Learning rate (change this to see convergence changing)
#
alpha = 0.002

# Space to save answers for plotting
#
P_train = np.zeros(MaxIter)
P_test  = np.zeros(MaxIter)

# Main Loop
#
for iter in range(MaxIter):

    # Select a data item at random
    #
    r = np.floor(np.random.rand()*Ntrain).astype(int)
    x = X_train[r,:]

    # If it is misclassified, update weights
    #
    if (y_train[r] * np.dot(x, w) < 0):
        w += alpha * y_train[r] * x

    # Evaluate trainign and test performances for plotting
    #
    P_train[iter] = PercentCorrect(X_train, y_train, w);
    P_test[iter]  = PercentCorrect(X_test, y_test, w);

print('Percentage Correct After Training: %6.2f  %6.2f'
      %(PercentCorrect(X_train, y_train, w), PercentCorrect(X_test, y_test, w)))
```

7. Plot learning curves

```
fig, ax = plt.subplots(figsize=(6,4))
ax.plot(range(MaxIter), P_train, 'b', label = "Training")
ax.plot(range(MaxIter), P_test, 'r', label = "Test")
ax.grid(True)
ax.legend()
ax.set_title('Perceptron Learning')
ax.set_ylabel('Training and Test Accuracies', fontsize=14)
ax.set_xlabel('Iteration', fontsize=14)
plt.savefig('learningCurves.png')
```

The expected results of training a perceptron might look similar to Fig. 2

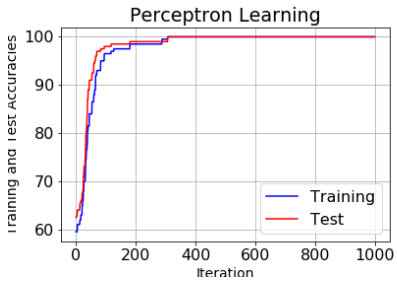


Figure 2: Learning Curves for Classifying two Gaussian Distributed Data

8. The `scikitlearn` package is an excellent source of machine learning algorithms in `Python`. Compare the performance of your perceptron algorithm on the two-class Gaussian dataset with that of the perceptron tool in the `scikitlearn` package. Here is a snippet of code to help you get started:

```
# Scikitlearn can do it for us
#
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
model = Perceptron()
model.fit(X_train, y_train)
yh_train = model.predict(X_train)
print("Accuracy on training set: %6.2f" %(accuracy_score(yh_train, y_train)))

yh_test = model.predict(X_test)
print("Accuracy on test set: %6.2f" %(accuracy_score(yh_test, y_test)))

if (accuracy_score(yh_test, y_test) > 0.99):
    print("Wow, Perfect Classification on Separable dataset!")
```

9. Consider the problem with means at $\boldsymbol{m}_1 = \begin{bmatrix} 2.5 \\ 2.5 \end{bmatrix}$ and $\boldsymbol{m}_2 = \begin{bmatrix} 10.0 \\ 10.0 \end{bmatrix}$ with the covariance matrices equal and the same as before. Does the perceptron as implemented solve this problem? If not what modification is needed to help solve this problem? Hint:

```
O = np.ones((2*NumDataPerClass, 1))
X = np.append(X, O, axis=1)

w = np.random.randn(3)
```

10. Download a two class classification problem from the UCI machine Learning Repository of benchmark datasets `https://archive.ics.uci.edu/ml/index.php` and classify using your own perceptron algorithm. How does the performance compare to any quoted results on this dataset by other researchers? You may need more python tools to read and manipulate downloaded data (*e.g.* `Pandas`).