L2_Statistics - introduction

In this exercise we will try to create a mean and variance function where the variance is unbiased. Following that we will look at creating a function for generating an auto covariance matrix.

Mean and Variance

The mean and variance (and hence the standard deviation) for a random variable X can for a population of N samples be estimated as

$$\begin{aligned} E[X] &= \frac{1}{N}\sum_{i=1}^{N} X_i \\ &= \mu_X \\ V[X] &= E[(X_i - E[X])(X_i - E[X])] \\ &= E[X_i X_i] - E[X]^2 \\ &= \left(\frac{1}{N}\sum_{i=1}^{N} X_i X_i\right) - \mu_X^2 \\ &= \sigma_X^2 \\ \sigma &= \sqrt{V} \end{aligned}$$

When using the factor $1/(N-1)$, $\hat{V}$ is said to be the best unbiased estimator, when it is $1/N$ it will be biased, both assuming an underlying normal distribution.

Auto covariance in Matrix Notation

Now let's goto full matrix notation for the covariance. For a data matrix $\mathbf{X}$

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_d^{(2)} \\ \vdots & & & \\ x_1^{(n)} & x_2^{(n)} & \cdots & x_d^{(n)} \end{bmatrix}$$

the previous one-dimensional random variable X can now be seen as a d-dimensional vector $\mathbf{x}^{(i)}$, that is one of the data rows in the full data matrix

$$X_i \rightarrow \mathbf{x}^{(i)} = \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & \cdots & x_d^{(i)} \end{bmatrix}^T$$

The (biased) auto–covariance matrix can be estimated as

$$\begin{aligned} E[\mathbf{X}] &= \mu_{\mathbf{X}} \\ &= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}^{(i)} \\ \\ \Sigma(\mathbf{X}) &= \mathrm{cov}(\mathbf{X}, \mathbf{X}) \\ &= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}^{(i)} \mathbf{x}^{(i)T} - \mu_{\mathbf{X}} \mu_{\mathbf{X}}^T \end{aligned}$$

Implementation

The assignment description will more or less come as given before each task.

Qa Creating a mean and variance function for some input data

Your python function should be named `MeanAndVariance()` , it takes a vector as input parameter, and returns TWO parameters, namely mean and variance.

Python allows for return of zero, one, or more parameters from a function, without having to place, say two output parameters in a tuple, like in C++ `return make_pair<float,float>(mean, variance)` .

Test it via the `y` input and test-vectors below, go for the biased variance estimator at first.

In [97]:

```
# Qa
import numpy as np

def MeanAndVariance(x, bias = 0):
    # implementing the mean and variance function, which returns the Variance.
    # Biased or unbiased variance can be chosen by seeting param 'bias' to 1 for unbias
ed, 0 for biased.
    # Thus the function will per default be biased.
    n = len(x)
    m1 = (1/(n))*sum(x)

    v1 = ( ( 1/(n-bias) )*( sum( pow(x,2) ) ) ) - ( ( n/(n-bias) )*pow(m1,2) )

    return m1, v1

# Test-case - testing the mean and varianve calculation.
y = np.array([1,2,3,4])
m, v = MeanAndVariance(y, 1)

expected_v_unbiased = 1.6666666666666667 # factor 1/(n-1)

print("Expected_m =", np.mean(y), ", m =",m,", difference =", m-np.mean(y))

print("Expected_v_biased=", np.var(y), ", v=", v)

print("\nDifference in biased and unbiased variance =", v-np.var(y))

print("\nDifference in expected and calculated ubiased variance =", v-expected_v_unbias
ed)
```

```
Expected_m = 2.5 , m = 2.5 , difference = 0.0
Expected_v_biased= 1.25 , v= 1.6666666666666679

Difference in biased and unbiased variance = 0.41666666666666785

Difference in expected and calculated ubiased variance = 1.110223024625156
5e-15
```

Qb Create a function that generates the auto-covariance matrix $\Sigma(\mathbf{X})$.

At this task we are creating a function that can genereate an auto-covariance matrix.

`numpy.cov` will only serve as a test stub, to check that we got the right calculations

In [1]:

```
#Qb
import numpy as np

def covar(x, y, bias = 0):
    #bias is set to 0 for biased variance and 1 for unbiased.
    n = len(x)
    if(n != len(y)):
        print("Err, different lengths.")
        return 0

    mx = (1/(n))*sum(x)
    my = (1/(n))*sum(y)

    cv1 = (1/(n-bias))*sum((x-mx)*(y-my))

    return cv1

def covar_mat(X, bias = 0):
    #bias is set to 0 for biased variance matrix and 1 for unbiased.
    mat = np.array([[covar(X[0], X[0], bias),  \
                     covar(X[0], X[1], bias)], \
                    [covar(X[1], X[0], bias),  \
                     covar(X[1], X[1], bias)]])
    return mat

#Get a normal distributed x and y vector with mean = 0 and stanard deviation = 1
N = 50
x = np.random.normal(0, 1, N)
y = np.random.normal(0, 1, N)

X = np.vstack((x, y)).T

# we rotate the matrix
theta = 0.77*np.pi
cos, sin = np.cos(theta), np.sin(theta)
rot = np.array([[cos, -sin], \
                [sin, cos]])

# we scale the matrix
s_x = 0.7
s_y = 3.4
scale = np.array([[s_x, 0], \
                  [0, s_y]])

# Transformation matrix
T_mat = scale.dot(rot)

# Apply transformation matrix to X
Y = X.dot(T_mat)

cv_mat_biased = covar_mat(Y.T, 0)
cv_mat_unbiased = covar_mat(Y.T, 1)
expected_cv_biased = np.cov(Y.T, rowvar=True, bias=True)
expected_cv_unbiased = np.cov(Y.T, rowvar=True, bias=False)

#printing the results.
print("Expected biased covariance matrix")
print(expected_cv_biased)
print("\nCalculated biased covariance matrix")
print(cv_mat_biased)

print("\nExpected unbiased covariance matrix")
print(expected_cv_unbiased)

print("\nCalculated unbiased covariance matrix")
print(cv_mat_unbiased)
```

```
Expected biased covariance matrix
[[ 6.42908842 -6.7827325 ]
 [-6.7827325   8.30601927]]

Calculated biased covariance matrix
[[ 6.42908842 -6.7827325 ]
 [-6.7827325   8.30601927]]

Expected unbiased covariance matrix
[[ 6.5602943  -6.92115562]
 [-6.92115562  8.47552987]]

Calculated unbiased covariance matrix
[[ 6.5602943  -6.92115562]
 [-6.92115562  8.47552987]]
```

Results & Conclusion.

From the MeanAndVariance function a small difference in the expected unbiased variance and the calculated unbiased variance was detected, but such a small difference it would only make sense to be some form of error in rounding the number.

From the covariance function some good and expected results was found which is seen just above this section. It can be seen that the diagonal elements Σii represents the variance for x and y respectivly on spots 0,0 and 1,1. Where the off diagonal elements Σij for $i \neq j$ then represents the co-varaince between x and y on spots 0,1 and 1,0.

```
In [ ]:
```