9_probstat

February 17, 2025

1 Probability and Statistics

variationalform https://variationalform.github.io/

Just Enough: progress at pace https://variationalform.github.io/

https://github.com/variationalform

Simon Shaw https://www.brunel.ac.uk/people/simon-shaw.

This document uses python

and also makes use of LaTeX

in Markdown

1.1 What this is about:

This is a very quick recap of essential (for us) concepts in

- Probability.
- Statistics.

As usual our emphasis will be on *doing* rather than *proving*: *just enough: progress at pace*

1.2 Assigned Reading

For this worksheet you are recommended Chapter 6 of [MML] and Appendix C of [DSML].

- MML: Mathematics for Machine Learning, by Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. Cambridge University Press. https://mml-book.github.io.
- DSML: Data Science and Machine Learning, Mathematical and Statistical Methods by Dirk P. Kroese, Zdravko I. Botev, Thomas Taimre, Radislav Vaisman https://people.smp.uq.edu.au/DirkKroese/DSML/DSML.pdf
- There are also various resources here: https://stats.libretexts.org/Bookshelves

These can be accessed legally and without cost. **NOTE:** we haven't referred to the second of these before.

There are also these useful references for coding:

- PT: `python`: https://docs.python.org/3/tutorial
- NP: `numpy`: https://numpy.org/doc/stable/user/quickstart.html
- MPL: `matplotlib`: https://matplotlib.org

And, DSML (as above): Appendix D has a very useful python primer.

1.3 Discrete Probability

Probability is a subtle topic, and also not without its interpretational controversy.

Suppose you flip a coin S times and it lands heads n times.

- What is the probability that it will land heads next time you flip it?
- What is the probability it will not land heads, but land tails?

We do this by looking at the relatve frequencies and say:

- The probabilty of a head, written $\mathrm{P}(H)$ is given by the limiting value of n/S as $S \to \infty$.
- Since the coin land heads or tails and it can't land both: $\mathrm{P}(H) + \mathrm{P}(T) = 1$. This is *certainty*.
- It follows that $\mathrm{P}(T) = 1 - \mathrm{P}(H)$.

This seems fine - we can flip a coin as many times as we like to approximate $S \to \infty$.

We can also introduce *prior beliefs*. If the coin is fair, a judgement we make by appeal to its physical symmetry and the laws of physics, then we can assert that

$\mathrm{P}(H) = 1/2$, and

$\mathrm{P}(T) = 1/2$.

Sometimes though we can't appeal to this type of simple intuitive interpretation of probability.

There is a 70% chance of rain tomorrow.

Really? What does that mean? It isn't like the coin toss. We can't repeat 'tomorrow' S times and count the number n of times it rains.

What this means is that for 10 meteorologically similar days we can expect to need an umbrella on 7 of them.

There is a lot of history and lively debate around these questions of interpretation. See for example, https://plato.stanford.edu/entries/probability-interpret/

We're fortunate though. We will usually be able to run our codes many times on large enough data sets, and so we can think about relative frequencies.

1.3.1 Key axioms of Discrete Probability

We think of running an experiment. We will have a sample space Ω of all the possible outcomes of the experiment, and an event space $\mathcal{E}$ of all possible results. The event space is the power set of Ω.

For example, if we toss a coin three times there are 2^3 possible outcomes.

$\Omega = \{HHH, HHT, HTH, THH, HTT, THT, THH, TTT\}$.

Example of events are

(i) 'Two heads and one tail': $\{HHT, HTH, THH\}$.

(ii) 'Head on first fall': $\{HHH, HHT, HTH, HTT\}$.

There is a function $\mathrm{P}\colon \mathcal{E} \to [0,1]$ that assigns a probability to each event $E \in \mathcal{E}$. This function gives the probability of $E \in \mathcal{E}$.

On the assumption that this is revision, we wont work through examples here.

1.3.2 Conditional Probability

Suppose you want to know the probability that A occurs **given that** B does occur. For example,

- What is the probability of a negative test, A, given the patient is healthy, B?
- What is probability of a DNA match, A, on a guilty defendant, B?
- What is the probability the penguin is a *Chinstrap* given that your k-NN classifier predicts so?

We write this **conditional probability** as $\mathrm{P}(A \mid B)$.

To understand it, suppose that in S trials A and B have simultaneously occurred m times while B has occurred n times. We must have that $n \geq m$ and so the probability that A and B both occurred given that B occurred is reasoned out like this:

$$\mathrm{P}(A \mid B) \approx \frac{m}{n} = \frac{m}{S}\frac{S}{n} = \frac{m}{S}\left(\frac{n}{S}\right)^{-1} \to \frac{\mathrm{P}(A \text{ and } B)}{\mathrm{P}(B)}.$$

We take the right hand side as the definition of the left hand side, given the intuitive calculation in the middle.

1.3.3 Bayes' Theorem

This is very useful. It allows us to switch the conditioning around.

- What is the probability that the patient is healthy, B, given a negative test, A?
- What is probability that the defendant is guilty, B, given a DNA match, A?
- What is the probability that the classifier predicts *Gentoo* for *Adelie*?

$$\mathrm{P}(B \mid A) = \frac{\mathrm{P}(A \text{ and } B)}{\mathrm{P}(A)} = \frac{\mathrm{P}(A \text{ and } B)}{\mathrm{P}(B)}\frac{\mathrm{P}(B)}{\mathrm{P}(A)} = \frac{\mathrm{P}(B)}{\mathrm{P}(A)}\mathrm{P}(A \mid B).$$

It is useful to recognise that $\mathrm{P}(A \text{ and } B) = \mathrm{P}(A \mid B)\mathrm{P}(B) = \mathrm{P}(B \mid A)\mathrm{P}(A)$.

1.3.4 Key Formulae for Probability.

$\mathrm{P}(A \text{ or } B) = \mathrm{P}(A) + \mathrm{P}(B) - \mathrm{P}(A \text{ and } B)$.

$\mathrm{P}(A \text{ or } B) = \mathrm{P}(A) + \mathrm{P}(B)$ if A and B are **mutually exclusive**.

$\mathrm{P}(A \text{ and } B) = \mathrm{P}(A)\mathrm{P}(B)$ if A and B are **independent**.

$\mathrm{P}(A) + \mathrm{P}(\neg A) = 1$ where $\neg A$ ('not' A) means that A did not occur.

The **Partition Theorem** (or the **Law of Total Probability**)

- $\mathrm{P}(A) = \mathrm{P}(A \text{ and } B) + \mathrm{P}(A \text{ and } \neg B)$

- $\mathrm{P}(A) = \mathrm{P}(A \mid B)\mathrm{P}(B) + \mathrm{P}(A \mid \neg B)\mathrm{P}(\neg B)$

Bayes formula (reprised)

$$\mathrm{P}(B \mid A) = \frac{\mathrm{P}(A \text{ and } B)}{\mathrm{P}(A)} = \frac{\mathrm{P}(A \mid B)\mathrm{P}(B)}{\mathrm{P}(A \mid B)\mathrm{P}(B) + \mathrm{P}(A \mid \neg B)\mathrm{P}(\neg B)}$$

We wont have too much need for these, but we are interested in the connection to **confusion matrices**…

1.3.5 Confusion Matrices

Recall that for a binary classifier our confusion matrices took the very specific form:

$$\begin{array}{rcc} & & \text{output, or predicted} \\ \text{target, or true} & & + \quad - \\ \text{label/class} & Y & \mathrm{TP} \quad \mathrm{FN} \\ & N & \mathrm{FP} \quad \mathrm{TN} \end{array}$$

These numbers represent estimates (that get better as $S \to \infty$) of conditional probabilities…

Example: suppose we have this set of results where Y or N are the known labels and $+$ and $-$ are the predictions:

$$\begin{array}{rcc} & & \text{predicted} \\ \text{label} & & + \quad - \\ & Y & \mathrm{TP} \quad \mathrm{FN} \\ & N & \mathrm{FP} \quad \mathrm{TN} \end{array} \qquad \text{with, specifically,} \qquad \begin{pmatrix} 62 & 5 \\ 9 & 44 \end{pmatrix}.$$

There are $S = 120$ results. Look along the first row - these are the actual numbers in the sample which are labelled as `Y` (healthy, innocent, passed, safe, …) as opposed to `N` (sick, guilty, failed, unsafe, …).

So $62 + 5$ are in the `Y` class out of a total of 120. If this sample represents the population then we can estimate…

$\mathrm{P}(Y) = 67/120$. Similarly, $P(+) = (62 + 9)/120$.

Further, in the second row, we **know** that `N` occurs, for these are all in the `N` class. So, with similar reasoning …

$\mathrm{P}(+ \mid N) = 9/(9 + 44)$. Similarly, $P(Y \mid -) = 5/(5 + 44)$.

Let's see all the calculations…

[1]:
```
import numpy as np
cm = np.array([[62,5],[9,44]])
N = cm.sum()
print('Number of samples: ', N,' with base rates...')
print('P(Y) = (62+5)/120 = ', (62+5)/120, end=' and ')
print('P(N) = (9+44)/120 = ', (9+44)/120, ' = 1-P(Y)')
```

```
print('P(+) = (62+9)/120 = ', (62+9)/120, end=' and ')
print('P(-) = (5+44)/120 = ', (5+44)/120, ' = 1-P(+)')
print('Conditionals...')
print('P(Y|+) = 62/(62+9) = ', 62/(62+9), end=' and ')
print('P(N|+) =  9/(62+9) = ',  9/(62+9))
print('P(Y|-) =  5/(5+44) = ',  5/(5+44), end=' and ')
print('P(N|-) = 44/(5+44) = ', 44/(5+44))
print('P(+|Y) = 62/(62+5) = ', 62/(62+5), end=' and ')
print('P(-|Y) =  5/(62+5) = ',  5/(62+5))
print('P(+|N) =  9/(9+44) = ',  9/(9+44), end=' and ')
print('P(-|N) = 44/(9+44) = ', 44/(9+44))
```

```
Number of samples:  120  with base rates…
P(Y) = (62+5)/120 =  0.5583333333333333 and P(N) = (9+44)/120 =
0.44166666666666665  = 1-P(Y)
P(+) = (62+9)/120 =  0.5916666666666667 and P(-) = (5+44)/120 =
0.4083333333333333  = 1-P(+)
Conditionals…
P(Y|+) = 62/(62+9) =  0.8732394366197183 and P(N|+) =  9/(62+9) =
0.1267605633802817
P(Y|-) =  5/(5+44) =  0.10204081632653061 and P(N|-) = 44/(5+44) =
0.8979591836734694
P(+|Y) = 62/(62+5) =  0.9253731343283582 and P(-|Y) =  5/(62+5) =
0.07462686567164178
P(+|N) =  9/(9+44) =  0.16981132075471697 and P(-|N) = 44/(9+44) =
0.8301886792452831
```

1.3.6 Relation to Earlier Formulae and Measures

Earlier for a binary classifier we defined some useful terms for measuring performance. Some of these can be related to conditional probabilites.

$$\begin{array}{rcc} \text{target, or true} & Y & \begin{pmatrix} \text{TP} & \text{FN} \\ \text{FP} & \text{TN} \end{pmatrix} \\ \text{label/class} & N & \\ & & \begin{array}{cc} + & - \end{array} \\ & & \text{output, or predicted} \\ & & \text{label/class} \end{array}$$

Recall that we used P and N for the number of positives and negatives overall in the test set.

- **Prevalence**: Prevalence $= \frac{\text{P}}{\text{P+N}} = \text{P}(Y)$
- TPR: **True Positive Rate**, *sensitivity*, *recall*: $\text{TPR} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP+FN}} = \text{P}(+ \mid Y)$
- TNR: **True Negative Rate**, *specificity*, *selectivity*: $\text{TNR} = \frac{\text{TN}}{\text{N}} = \frac{\text{TN}}{\text{TN+FP}} = \text{P}(- \mid N)$

And also…

$$\begin{array}{rcc} \text{target, or true} & Y & \begin{pmatrix} \text{TP} & \text{FN} \\ \text{FP} & \text{TN} \end{pmatrix} \\ \text{label/class} & N & \\ & & + \quad - \\ & & \text{output, or predicted} \\ & & \text{label/class} \end{array}$$

- FPR: **False Positive Rate**: $\text{FPR} = \frac{\text{FP}}{\text{N}} = \frac{\text{FP}}{\text{FP+TN}} = \text{P}(+ \mid N)$
- FNR: **False Negative Rate**: $\text{FNR} = \frac{\text{FN}}{\text{P}} = \frac{\text{FN}}{\text{FN+TP}} = \text{P}(- \mid Y)$
- PPV: **Positive Predictive Value**, *precision*: $\text{PPV} = \frac{\text{TP}}{\text{TP+FP}} = \text{P}(Y \mid +)$
- NPV: **Negative Predictive Value**: $\text{NPV} = \frac{\text{TN}}{\text{TN+FN}} = \text{P}(N \mid -)$

1.4 Statistics and Associated Concepts

In addition to needing an understanding of how to infer probabilities from our results, we'll also need to understand some related concepts from *Mathematical Statistics*.

We review these terms:

- expected value and mean, median and mode.
- variance and standard deviation.
- correlation and covariance.

A random variable is a function $Z \colon \Omega \to \mathbb{R}$. We can use them to take probabilities.

For example, a dice is thrown and Z is assigned the value shown on the upward face.

The **Expected Value** of the random variable is the sum of all the probabilities weighted by the value of the variable:

- $\mathbb{E}(Z) = \sum_{k=1}^{6} k \, \text{P}(Z = k)$

This coincides with the notion of *average* or *mean* value. Why?

1.4.1 Mean, Average, Expected Value

The numerical data we will usually be working with will typically be lists of **samples** of the random variable, with each value of the random variable occuring with equal probability.

For example, if the random variable, Z, takes one of N equally probable values $Z_1, Z_2, \ldots, Z_N$, then the probability that a given value is taken is N^{-1} and then the expected value of Z is,

$$\mathbb{E}(Z) = \sum_{k=1}^{N} kP(Z_k) = \frac{1}{N} \sum_{k=1}^{N} k = \bar{Z}.$$

This, the expected value of Z, is called the **mean**, or **average**, value of Z.

We use $\bar{Z}$ to denote the sample mean. It is common to denote the population mean by μ_Z, but this isn't usually accessible to us - we'll almost always be working with samples and so we write $\bar{Z} \approx \mu_Z$.

1.4.2 Median and Mode

The mean is a measure of the **centre** of a distribution. Two other measures are also in common use. Confining ourselves to the discrete case these are...

- **median**: this is the value in the middle of an ordered set. For example $\{1, 3, 4, 78, 90\}$ has median 4. When the set has an even number of elements the median can be taken as the average of the two centre elements.
- **mode**: this is the most frequently occuring value. The set above doesn't have a mode (or all elements are modes). The set $\{1, 3, 3, 78, 90\}$ has mode 3.

1.4.3 Variance

The **variance** of the random variable, X, is defined (for our purposes) as

$\mathrm{Var}(X) = \mathbb{E}(X^2) - \big(\mathbb{E}(X)\big)^2$.

For us, with sample size N, this is,

$$\mathrm{Var}(X) = \frac{1}{N}\sum_{k=1}^{N}\big(X_k - \mathbb{E}(X)\big)^2$$

Also, the **standard deviation** is given by

$\sigma_X = \mathrm{Std}(X) = \sqrt{\mathrm{Var}(X)}$.

These formulae are sometimes altered slightly for smaller samples sizes, with the denominator N replaced by $N-1$ to get an unbiased estimate. When N is large this has negligible effect.

Let's see concrete examples with the data set $X \in \{1, 3, 4, 5, 7\}$. We'll see that `numpy` can make life easy for us...

[2]:
```
X = np.array([1,3,4,5,7])
N = X.shape[0]
Xbar = X.sum()/N
print('E(X) = mean = ', Xbar, ' or with numpy: ', X.mean())
# centre X using mean, then sum of squares using dot product
Xc = X-Xbar
VarX = Xc.T.dot(Xc) / N
print('Var(X) = variance = ', VarX, ' or with numpy: ', X.var())
print('SD(X) = Std Dev = ', np.sqrt(VarX), ' or with numpy: ', X.std())
# or, the unbiased result..
VarX = Xc.T.dot(Xc) / (N-1)
print('Var(X) = variance = ', VarX, ' or with numpy: ', X.var(ddof=1))
print('SD(X) = Std Dev = ', np.sqrt(VarX), ' or with numpy: ', X.std(ddof=1))
```

```
E(X) = mean =  4.0  or with numpy:  4.0
Var(X) = variance =  4.0  or with numpy:  4.0
SD(X) = Std Dev =  2.0  or with numpy:  2.0
Var(X) = variance =  5.0  or with numpy:  5.0
SD(X) = Std Dev =  2.23606797749979  or with numpy:  2.23606797749979
```

1.4.4 Covariance and Correlation

Often we have more than one random variable in play. We saw four numerical columns in the penguins data set for example. We can calculate stats for each column as shown above, but how can we assess how **related** these variables might be?

We define the **covariance** of two random variables as

$$\mathrm{Cov}(X,Y) = \mathbb{E}\Big(\big(X - \mathbb{E}(X)\big)\big(Y - \mathbb{E}(Y)\big)\Big) = \frac{1}{N}\sum_{k=1}^{N}\big(X_k - \bar{X}\big)\big(Y_k - \bar{Y}\big)$$

and the **correlation coefficient** of two random variables as

$$\rho_{XY} = \frac{\mathrm{Cov}(X,Y)}{\sigma_X\,\sigma_Y}$$

It is easy to see that $\mathrm{Cov}(X,X) = \mathrm{Var}(X)$ and that $\rho_{XX} = 1$.

These measurements indicate how strongly related the random variables are to each other: positive correlations indicate that both tend to grow or diminish together, while negatives indicate that one grows as the other shrinks. A zero correlation indicates that the variables are unrelated.

We'll normally work with covariance rather than correlation. Let's see an example - using penguins again…

Grab the data and clean it up just like before.

[3]:
```
import numpy as np
import seaborn as sns
dfp = sns.load_dataset('penguins')
dfp.head()
dfp = dfp.dropna()
dfp = dfp.reset_index(drop=True)
dfp.head()
```

[3]:
```
   species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
0  Adelie  Torgersen            39.1           18.7              181.0
1  Adelie  Torgersen            39.5           17.4              186.0
2  Adelie  Torgersen            40.3           18.0              195.0
3  Adelie  Torgersen            36.7           19.3              193.0
4  Adelie  Torgersen            39.3           20.6              190.0

   body_mass_g     sex
0       3750.0    Male
1       3800.0  Female
2       3250.0  Female
3       3450.0  Female
4       3650.0    Male
```

Now assign the numerical data in columns 3 - 6 in `X`

[4]:
```
X = dfp.iloc[:, 2:6].values
X[:4,:]
```

[4]:
```
array([[  39.1,   18.7,  181. , 3750. ],
       [  39.5,   17.4,  186. , 3800. ],
       [  40.3,   18. ,  195. , 3250. ],
       [  36.7,   19.3,  193. , 3450. ]])
```

Each column represents a random variable: X_0, X_1, X_2, X_3. We can calculate means, variances and covariances. For example...

[5]:
```
print('Mean of column 1 (indexed at 0) :  ', X[:,0].mean())
print('Std Dev of column 3 (population):  ', X[:,2].std())
print('Std Dev of column 3 (unbiased)  :  ', X[:,2].std(ddof=1))
```

```
Mean of column 1 (indexed at 0) :   43.99279279279279
Std Dev of column 3 (population):   13.994704772576716
Std Dev of column 3 (unbiased)  :   14.015765288287879
```

[6]:
```
# remember that we can access some summary stats like this...
dfp.describe()
```

[6]:

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
count	333.000000	333.000000	333.000000	333.000000
mean	43.992793	17.164865	200.966967	4207.057057
std	5.468668	1.969235	14.015765	805.215802
min	32.100000	13.100000	172.000000	2700.000000
25%	39.500000	15.600000	190.000000	3550.000000
50%	44.500000	17.300000	197.000000	4050.000000
75%	48.600000	18.700000	213.000000	4775.000000
max	59.600000	21.500000	231.000000	6300.000000

What about the covariance? Let's calculate $\text{Cov}(X_1, X_2)$...

[7]:
```
# first center the data using the column means...
X1 = X[:,[1]] - X[:,[1]].mean()
X2 = X[:,[2]] - X[:,[2]].mean()
# then multiply, sum and take the unbiased average
N = X.shape[0]
CV12 = np.sum(X1*X2)/(N-1)
print("Cov(X1,X2) = ", CV12)
```

```
Cov(X1,X2) =  -15.94724845327255
```

Rather than `np.sum()`, we can use the dot product, $\boldsymbol{X}_1 \cdot \boldsymbol{X}_2 = \boldsymbol{X}_1^T \boldsymbol{X}_2$, like this...

[8]:
```
CV12 = X1.T @ X2 / (N-1)
print("Cov(X1,X2) = ", CV12, " or as a scalar Cov(X1,X2) = ", float(CV12[0,0]) )
```

```
# The above is a bug fixed version of the code below.
# We needed the [0,0] for float to work.
# CV12 = X1.T @ X2 / (N-1)
# print("Cov(X1,X2) = ", CV12, " or as a scalar Cov(X1,X2) = ", float(CV12) )
```

```
Cov(X1,X2) =  [[-15.94724845]]  or as a scalar Cov(X1,X2) =  -15.94724845327255
```

A useful concept is the **covariance matrix**, it takes this form:

$$\boldsymbol{M} = \begin{pmatrix} \mathrm{Var}(X_0) & \mathrm{Cov}(X_0, X_1) & \mathrm{Cov}(X_0, X_2) & \mathrm{Cov}(X_0, X_3) \\ \mathrm{Cov}(X_1, X_0) & \mathrm{Var}(X_1) & \mathrm{Cov}(X_1, X_2) & \mathrm{Cov}(X_1, X_3) \\ \mathrm{Cov}(X_2, X_0) & \mathrm{Cov}(X_2, X_1) & \mathrm{Var}(X_2) & \mathrm{Cov}(X_2, X_3) \\ \mathrm{Cov}(X_3, X_0) & \mathrm{Cov}(X_3, X_1) & \mathrm{Cov}(X_3, X_2) & \mathrm{Var}(X_3) \end{pmatrix}$$

Recall that $\mathrm{Cov}(X, X) = \mathrm{Var}(X)$ and note that $\mathrm{Cov}(X, Y) = \mathrm{Cov}(Y, X)$. This matrix is therefore **symmetric** and so has real eigenvalues.

The covariance matrix is also positive semidefinite. This means that

$$\boldsymbol{u} \cdot \boldsymbol{M}\boldsymbol{u} \geq 0$$

for all vectors $\boldsymbol{u}$. This in turn means that the eigenvalues of the covariance matrix are non-negative. To see this inequality assume without loss of generality that the X_i's are already centered and collect the observed value into the column vectors $\boldsymbol{X}_i$. Then,

$$(N-1)\boldsymbol{M} = \begin{pmatrix} \boldsymbol{X}_0 \cdot \boldsymbol{X}_0 & \boldsymbol{X}_0 \cdot \boldsymbol{X}_1 & \boldsymbol{X}_0 \cdot \boldsymbol{X}_2 & \boldsymbol{X}_0 \cdot \boldsymbol{X}_3 \\ \boldsymbol{X}_1 \cdot \boldsymbol{X}_0 & \boldsymbol{X}_1 \cdot \boldsymbol{X}_1 & \boldsymbol{X}_1 \cdot \boldsymbol{X}_2 & \boldsymbol{X}_1 \cdot \boldsymbol{X}_3 \\ \boldsymbol{X}_2 \cdot \boldsymbol{X}_0 & \boldsymbol{X}_2 \cdot \boldsymbol{X}_1 & \boldsymbol{X}_2 \cdot \boldsymbol{X}_2 & \boldsymbol{X}_2 \cdot \boldsymbol{X}_3 \\ \boldsymbol{X}_3 \cdot \boldsymbol{X}_0 & \boldsymbol{X}_3 \cdot \boldsymbol{X}_1 & \boldsymbol{X}_3 \cdot \boldsymbol{X}_2 & \boldsymbol{X}_3 \cdot \boldsymbol{X}_3 \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}_0^T \\ \boldsymbol{X}_1^T \\ \boldsymbol{X}_2^T \\ \boldsymbol{X}_3^T \end{pmatrix} \begin{pmatrix} \boldsymbol{X}_0 & \boldsymbol{X}_1 & \boldsymbol{X}_2 & \boldsymbol{X}_3 \end{pmatrix}$$

Write this as $(N-1)\boldsymbol{M} = \boldsymbol{K}^T\boldsymbol{K}$ and then for $\boldsymbol{u}$ arbitrary

$$\boldsymbol{u} \cdot \boldsymbol{M}\boldsymbol{u} = \frac{1}{N-1}\boldsymbol{u}^T\boldsymbol{K}^T\boldsymbol{K}\boldsymbol{u} = \frac{1}{N-1}(\boldsymbol{K}\boldsymbol{u})^T\boldsymbol{K}\boldsymbol{u} \geq 0$$

We have seen how to get a covariance matrix entry using `numpy`, but there are a lot more - and this is for just four columns in the data set.

Lots of work… Fortunately `numpy` can do the heavy lifting for us…

[9]:
```
# note the transpose...
print(np.cov(X.T))
```

```
[[ 2.99063334e+01 -2.46209134e+00  5.00581949e+01  2.59562330e+03]
 [-2.46209134e+00  3.87788831e+00 -1.59472485e+01 -7.48456122e+02]
 [ 5.00581949e+01 -1.59472485e+01  1.96441677e+02  9.85219165e+03]
 [ 2.59562330e+03 -7.48456122e+02  9.85219165e+03  6.48372488e+05]]
```

We can see that in the third column, second row we have $\text{Cov}(X_1, X_2) = -15.94724845\ldots$ as expected.

But, the `pandas` library that gives us the data frames has already thought of …

… both **covariance** and **correlation**, like this:

[10]:
```
dfp[['bill_length_mm', 'bill_depth_mm','flipper_length_mm', 'body_mass_g']].
↪cov()
# dfp.cov() # simpler, but may not work on newer installations
```

[10]:
```
                   bill_length_mm  bill_depth_mm  flipper_length_mm  \
bill_length_mm          29.906333      -2.462091          50.058195
bill_depth_mm           -2.462091       3.877888         -15.947248
flipper_length_mm       50.058195     -15.947248         196.441677
body_mass_g           2595.623304    -748.456122        9852.191649

                     body_mass_g
bill_length_mm       2595.623304
bill_depth_mm        -748.456122
flipper_length_mm    9852.191649
body_mass_g        648372.487699
```

[11]:
```
dfp[['bill_length_mm', 'bill_depth_mm','flipper_length_mm', 'body_mass_g']].
↪corr()
# dfp.corr() # simpler, but may not work on newer installations
```

[11]:
```
                   bill_length_mm  bill_depth_mm  flipper_length_mm  \
bill_length_mm           1.000000      -0.228626           0.653096
bill_depth_mm           -0.228626       1.000000          -0.577792
flipper_length_mm        0.653096      -0.577792           1.000000
body_mass_g              0.589451      -0.472016           0.872979

                   body_mass_g
bill_length_mm        0.589451
bill_depth_mm        -0.472016
flipper_length_mm     0.872979
body_mass_g           1.000000
```

> **THINK ABOUT:** do you need both `flipper_length_mm` and `body_mass_g` in your analysis?

1.4.5 Review

We covered *just enough*, to make *progress at pace*. We looked at

- How conditional probabilities can be estimated from confusion matrices.
- How we can obtain statistical quantities using `python` tools.

Now we can start putting all of this material to work.

1.5 Technical Notes, Production and Archiving

Ignore the material below. What follows is not relevant to the material being taught.

Production Workflow

- Finalise the notebook material above
- Clear and fresh run of entire notebook
- Create html slide show:
  - `jupyter nbconvert --to slides 9_probstat.ipynb`
- Set `OUTPUTTING=1` below
- Comment out the display of web-sourced diagrams
- Clear and fresh run of entire notebook
- Comment back in the display of web-sourced diagrams
- Clear all cell output
- Set `OUTPUTTING=0` below
- Save
- git add, commit and push to FML
- copy PDF, HTML etc to web site
  - git add, commit and push
- rebuild binder

Some of this originated from

https://stackoverflow.com/questions/38540326/save-html-of-a-jupyter-notebook-from-within-the-n

These lines create a back up of the notebook. They can be ignored.

At some point this is better as a bash script outside of the notebook

[12]:
```
%%bash
NBROOTNAME='9_probstat'
OUTPUTTING=1

if [ $OUTPUTTING -eq 1 ]; then
  jupyter nbconvert --to html $NBROOTNAME.ipynb
  cp $NBROOTNAME.html ../backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.html
  mv -f $NBROOTNAME.html ./formats/html/

  jupyter nbconvert --to pdf $NBROOTNAME.ipynb
  cp $NBROOTNAME.pdf ../backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.pdf
  mv -f $NBROOTNAME.pdf ./formats/pdf/

  jupyter nbconvert --to script $NBROOTNAME.ipynb
  cp $NBROOTNAME.py ../backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.py
  mv -f $NBROOTNAME.py ./formats/py/
else
  echo 'Not Generating html, pdf and py output versions'
fi
```

```
Not Generating html, pdf and py output versions
```