School of Computing and Information Systems
The University of Melbourne
COMP30027
MACHINE LEARNING (Semester 1, 2019)

Practical exercises: Week 2

1. Make sure that you have a Python environment where you can run Python. In particular, ensure that the `scipy`, `numpy`, `matplotlib`, and `sklearn` packages are installed (although we won't be using the latter two today).

```
>>> import numpy as np
>>> import matplotlib as mpl
>>> import sklearn
```

(You might wish to examine the installation instructions at `http://scipy.org/install.html` if you are considering using your local machine.)

2. The main `numpy` object is a so-called "homogeneous multidimensional array" — note that this is a little less flexible than using a list or tuple, but it allows mathematical operations to be performed **much** faster. (And we'll be doing a fair bit of number-crunching this semester, so this is an important property.)
Arrays can be created using the `arange()` method (analogous to the `range()` method for lists), for example:

```
>>> a = np.arange(5)
>>> a
array([0, 1, 2, 3, 4])
```

A second option is to use the `array()` method as a wrapper over a list:

```
>>> b = np.array([1, 3, -2, 0, 0])
>>> b
array([1, 3, -2, 0, 0])
```

(a) `numpy` supports vector (and matrix) operations,[1] like addition, subtraction, and scalar multiplication.
Evaluate (and check by hand) the following (based on the above values for $\vec{a}$ and $\vec{b}$): $\vec{a} + \vec{b}$, $\vec{a} - \vec{b}$, $3\vec{b} - 2\vec{a}$

(b) `numpy` arrays can be indexed, sliced, and iterated over, similarly to lists. Write a function to calculate the Euclidean distance between $\vec{a}$ and $\vec{b}$, starting with the following:

```
>>> def my_euclidean_dist(a, b):
...     sum = 0
...     for i in range(len(a)): #assume equal-sized vectors
```

Check your work by comparing it with `np.linalg.norm(a-b)`. (Why does this work?)
(Extension) Modify the function to throw an exception when `a` and `b` are of different dimensionalities.

(c) Write a function which instead calculates the **Hamming distance**. Find the Hamming distance between `[0, "cat", 800, "??"]` and `[1, "dog", -266, "??"]`.

[1] **Be very, very careful about manipulating arrays of different sizes.** `numpy` typically won't throw exceptions. Instead, it will do *something*: that something might be very intelligent, like automatically increasing the dimensionality of the smaller array to match the larger array — but if you aren't expecting it, the errors can be very difficult to find.

3. Recall that the **cosine** between two vectors can be calculated as:

$$cos(A, B) : \frac{A \cdot B}{||A|| \times ||B||}$$

   (a) Write a function that calculates the dot product between two vectors: $A \cdot B = \sum_i a_i b_i$; Use it to find the dot product between the following pairs of vectors:

      i. $\langle 1, 2 \rangle$ and $\langle 1, 1 \rangle$
      ii. $\langle 1, 2 \rangle$ and $\langle 2, 2 \rangle$
      iii. $\langle 1, 2 \rangle$ and $\langle 3, 3 \rangle$
      iv. $\langle 0, 0, 1 \rangle$ and $\langle 1, 0, 0 \rangle$

   (b) Use your dot product function and `my_euclidean_dist(a, np.array([0,0]))` to write a function which calculates the cosine of the angle between two vectors.
   Find the cosine of the angle between the pairs of vectors above. What do you notice?

   (c) (Extension) Compare your version to a more compact `numpy` solution:

```
>>> def my_cosine_numpy(a, b):
    return np.dot(a, b)/(norm(a) * norm(b))
```

   Generate a million random pairs of vectors; how much faster is the version which uses the in-built `numpy` functions?

4. Matrices can be made in `numpy` by wrapping a list of lists. For the following two matrices:

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 2 & 1 \\ 6 & 2 & 0 \end{pmatrix} N = \begin{pmatrix} 0 & 3 & 1 \\ 1 & 1 & 4 \\ 2 & 0 & 3 \end{pmatrix}$$

   (a) Find $M + N$ and $M - N$.

   (b) Somewhat unintuitively, `np.dot()` is used to multiply matrices. Compare:

      i. `M * N` and `np.dot(M, N)`
      ii. `N * M` and `np.dot(N, M)`
      iii. `M * M` and `M**2` and `np.dot(M, M)`