Instructions

- This assignment is based on a total of 12.5 points.
- Due on September 6, 11.59pm AEST.
- Assignments are to be solved individually. We will run plagiarism detection software.
- Your code **should be in Python (any version greater than or equal to 2.7)**. For clarity, the algorithms presented here will assume zero-based indices for arrays, vectors, matrices, etc.
- We **only need** the Python scripts (.py files). We **do not need** the Python (compiled) bytecodes (.pyc files). You will get 0 points in you fail to include the Python scripts (.py files) even if you mistakingly include the bytecodes (.pyc files). We will deduct points, if you do not use the right name for the Python scripts (.py) as described on each question, or if the input/output matrices/vectors/scalars have a different type/size from what is described on each question. **Test the provided examples before submission.**
- Please, submit a single ZIP file **through Canvas**. Your Python scripts (**svm.py**, **predict.py**, **kerperceptron.py**, **kerpredict.py**) should be directly inside the ZIP file. There should not be any folder inside the ZIP file, just Python scripts. The ZIP file should be named according to your UniMelb account. For instance, if my UniMelb account is jhonorio, the ZIP file should be named **jhonorio.zip**

Problem I: SVMs

1) [5 points] Recall support vector machines (SVM), introduced in Lecture 8. We removed the scalar b from Lecture 8 for easier implementation. (Here $w \cdot x_i$ represents the dot product of vectors w and x_i.)

$$\text{minimize } \frac{1}{2} w \cdot w + C \sum_{i=1}^{n} \max(0, 1 - y_i(w \cdot x_i))$$

Recall that $C > 0$. Now we ask you to implement the following gradient descent approach. Note that we are assuming step size (i.e., learning rate) equal to $1/(\text{iter} + 1)$.

Input: scalar C, number of iterations L, training data $x_i \in \mathbb{R}^d$, $y_i \in \{+1, -1\}$ for $i = 0, \ldots, n-1$
Output: $w \in \mathbb{R}^d$
$w \leftarrow 0$
for $\text{iter} = 0, \ldots, L-1$ **do**
  gradient $\leftarrow w$
  for $i = 0, \ldots, n-1$ **do**
    if $1 - y_i(w \cdot x_i) > 0$ **then**
      gradient $\leftarrow$ gradient $- C y_i x_i$
    end if
  end for
  $w \leftarrow w - \dfrac{\text{gradient}}{\text{iter} + 1}$
end for

The header of your **Python script svm.py** should be:

```
import numpy as np
# Input: scalar C
#        number of iterations L
#        numpy matrix X of features, with n rows (samples), d columns (features)
#            X[i,r] is the r-th feature of the i-th sample
#        numpy vector y of labels, with n entries (samples)
#            y[i] is the label (+1 or -1) of the i-th sample
# Output: numpy vector w, with d entries
def run(C,L,X,y):
```

```
    n, d = X.shape
    w = np.zeros((d,))
    # Your code goes here
    return w
```

2) [1.5 points] Implement the following linear predictor function, introduced in Lecture 8. We removed the scalar b from Lecture 8 for easier implementation. (Again, here $w \cdot z$ represents the dot product of vectors w and z.)

Input: $w \in \mathbb{R}^d$, testing point $z \in \mathbb{R}^d$
Output: label $\in \{+1, -1\}$
if $w \cdot z > 0$ **then**
  label $\leftarrow +1$
else
  label $\leftarrow -1$
end if

The header of your **Python script predict.py** should be:

```
import numpy as np
# Input: numpy vector w, with d entries
#        numpy vector z, with d entries
# Output: label (+1 or -1)
def run(w,z):
    label = -1.
    # Your code goes here
    return label
```

Problem II: Kernels

You will implement algorithms that depend on a kernel function K. You should call the following script **K.py**

```
import numpy as np
import math
# Input: numpy vector x of d entries
#        numpy vector z of d entries
# Output: kernel K(x,z) = exp(-1/2 * norm(x-z)^2)
# Example on how to call the script:
#    import K
#    v = K.run( np.array([1, 4, 3]) , np.array([2, 7, -1]) )
def run(x,z):
    x = np.asarray(x).flatten()
    z = np.asarray(z).flatten()
    if x.size != z.size:
        raise ValueError
    return math.exp(-0.5 * np.sum((x-z) ** 2))
```

Here are the questions:

3) [4.5 points] We introduced the perceptron algorithm with kernels in Lecture 10. Implement the following algorithm, which stops when all samples are classified correctly.

Input: number of iterations L, training data $x_i \in \mathbb{R}^d$, $y_i \in \{+1, -1\}$ for $i = 0, \ldots, n-1$
Output: $\alpha \in \mathbb{R}^n$, number of iterations that were actually executed (iter+1)
$\alpha \leftarrow 0$
for iter $= 0, \ldots, L-1$ **do**

all_points_classified_correctly ← True
for $i = 0, \ldots, n-1$ **do**
  if $y_i \left(\sum_{j=0}^{n-1} \alpha_j y_j K(x_j, x_i) \right) \leq 0$ **then**
    $\alpha_i \leftarrow \alpha_i + 1$
    all_points_classified_correctly ← False
  end if
end for
if all_points_classified_correctly **then**
  break (Note: this is Python's **break** statement, which should exit the **for** loop)
end if
end for

The header of your **Python script kerperceptron.py** should be:

```
import K
import numpy as np
# Input: number of iterations L
#        numpy matrix X of features, with n rows (samples), d columns (features)
#            X[i,r] is the r-th feature of the i-th sample
#        numpy vector y of labels, with n entries (samples)
#            y[i] is the label (+1 or -1) of the i-th sample
# Output: numpy vector alpha, with n entries
#         number of iterations that were actually executed (iter+1)
def run(L,X,y):
    n, d = X.shape
    alpha = np.zeros((n,))
    iter = 0
    # Your code goes here
    return alpha, iter+1
```

4) [1.5 points] Implement the following predictor function with kernels, introduced in Lecture 10.

Input: $\alpha \in \mathbb{R}^n$, training data $x_i \in \mathbb{R}^d$, $y_i \in \{+1, -1\}$ for $i = 0, \ldots, n-1$, testing point $z \in \mathbb{R}^d$
Output: label $\in \{+1, -1\}$
if $\sum_{i=0}^{n-1} \alpha_i y_i K(x_i, z) > 0$ **then**
  label $\leftarrow +1$
else
  label $\leftarrow -1$
end if

The header of your **Python script kerpredict.py** should be:

```
import K
# Input: numpy vector alpha, with n entries
#        numpy matrix X of features, with n rows (samples), d columns (features)
#            X[i,r] is the r-th feature of the i-th sample
#        numpy vector y of labels, with n entries (samples)
#            y[i] is the label (+1 or -1) of the i-th sample
#        numpy vector z, with d entries
# Output: label (+1 or -1)
def run(alpha,X,y,z):
    label = -1.
    # Your code goes here
    return label
```

TEST CASES

We provide few small synthetic datasets to test your Python scripts.

Test Case 1/3: Learning and predicting for 2-dimensional data

```
>>> import numpy as np
>>> X = np.array([[-3, 2],
  [-2, 1.5],
  [-1, 1],
  [0, 0.5],
  [1, 0]])
>>> y = np.array([1, 1, 1, -1, -1])

>>> import svm
>>> svm.run(1.0,20,X,y)
array([-0.95 , -0.025])
>>> svm.run(0.1,10,X,y)
array([-0.36 ,  0.165])

>>> import predict
>>> w = svm.run(0.1,10,X,y)
>>> w
array([-0.36 ,  0.165])
>>> predict.run(w,np.array([1, -2]))
-1.0
>>> predict.run(w,np.array([-2, 2]))
1.0

>>> import kerperceptron
>>> kerperceptron.run(1,X,y)
(array([1., 0., 0., 1., 0.]), 1)
>>> kerperceptron.run(2,X,y)
(array([1., 0., 1., 1., 0.]), 2)
>>> kerperceptron.run(3,X,y)
(array([1., 0., 1., 1., 0.]), 3)
>>> kerperceptron.run(4,X,y)
(array([1., 0., 1., 1., 0.]), 3)

>>> import kerpredict
>>> alpha, tmp = kerperceptron.run(10,X,y)
>>> alpha
array([1., 0., 1., 1., 0.])
>>> kerpredict.run(alpha,X,y,np.array([1, -2]))
-1.0
>>> kerpredict.run(alpha,X,y,np.array([-2, 2]))
1.0
```

Test Case 2/3: Learning and predicting for 3-dimensional data

```
>>> import numpy as np
>>> X = np.array([[-2, 2, 0],
  [-3, -1.5, -2],
  [-1, 1, 4],
  [1, -0.5, 5],
  [2, 0, -2],
  [2, 2, 1],
  [-0.5, -1, 0]])
>>> y = np.array([1, 1, 1, 1, -1, -1, -1])

>>> import svm
>>> svm.run(1.0,20,X,y)
array([-0.85,  0.05,  0.55])
>>> svm.run(0.1,10,X,y)
array([-0.45 , -0.055,  0.23 ])

>>> import predict
>>> w = svm.run(0.1,10,X,y)
>>> w
array([-0.45 , -0.055,  0.23 ])
>>> predict.run(w,np.array([1, 2, -5]))
-1.0
>>> predict.run(w,np.array([-2, 2, 10]))
1.0

>>> import kerperceptron
>>> kerperceptron.run(1,X,y)
(array([1., 0., 0., 0., 1., 0., 1.]), 1)
>>> kerperceptron.run(2,X,y)
(array([1., 1., 0., 1., 1., 0., 1.]), 2)
>>> kerperceptron.run(3,X,y)
(array([1., 1., 0., 1., 1., 0., 1.]), 3)
>>> kerperceptron.run(4,X,y)
(array([1., 1., 0., 1., 1., 0., 1.]), 3)

>>> import kerpredict
>>> alpha, tmp = kerperceptron.run(10,X,y)
>>> alpha
array([1., 1., 0., 1., 1., 0., 1.])
>>> kerpredict.run(alpha,X,y,np.array([1, 2, -5]))
-1.0
>>> kerpredict.run(alpha,X,y,np.array([-2, 2, 10]))
1.0
```

Test Case 3/3: Learning and predicting for 7-dimensional data

```
>>> import numpy as np
>>> X = np.array([[-2, 2, 0, -2, 2, 0, 4],
  [-3, -1.5, -2, 6, 5, 1, 4],
  [-1, 1, 4, 0, 5, -4, 5],
  [1, -0.5, 5, -9, -9, 0, 0],
  [2, 0, -2, -4.5, 3, 3, 1],
  [2, 2, 1, 4.5, 1, 1, 0],
  [-0.5, -1, 0, 3.5, -2, -2, 3],
  [0.5, 0, -2, 7, 1, 3, 4],
  [0.25, -1, 4, -2, 3, 1, 1],
  [-2, -2, -1, 0, 0, -1, -2]])
>>> y = np.array([1, 1, 1, 1, 1, -1, -1, -1, -1, -1])

>>> import svm
>>> svm.run(1.0,20,X,y)
array([-0.75 ,  0.85 , -0.75 , -1.825, -0.15 ,  0.2  ,  0.85 ])
>>> svm.run(0.1,10,X,y)
array([-1.25000000e-01,  1.95000000e-01, -1.50000000e-01, -3.70000000e-01,
1.00000000e-02, -1.38777878e-17,  1.80000000e-01])

>>> import predict
>>> w = svm.run(0.1,10,X,y)
>>> w
array([-1.25000000e-01,  1.95000000e-01, -1.50000000e-01, -3.70000000e-01,
1.00000000e-02, -1.38777878e-17,  1.80000000e-01])
>>> predict.run(w, np.array([1, -2, 0, 1, 2, 0.5, 0]))
-1.0
>>> predict.run(w, np.array([-2, 2, 0, 1, 2, 0.5, 0]))
1.0

>>> import kerperceptron
>>> kerperceptron.run(1,X,y)
(array([1., 0., 0., 0., 0., 1., 0., 0., 1., 0.]), 1)
>>> kerperceptron.run(2,X,y)
(array([1., 1., 0., 1., 0., 1., 0., 1., 1., 0.]), 2)
>>> kerperceptron.run(3,X,y)
(array([1., 1., 0., 1., 0., 1., 0., 1., 1., 0.]), 3)
>>> kerperceptron.run(4,X,y)
(array([1., 1., 0., 1., 0., 1., 0., 1., 1., 0.]), 3)

>>> import kerpredict
>>> alpha, tmp = kerperceptron.run(10,X,y)
>>> alpha
array([1., 1., 0., 1., 0., 1., 0., 1., 1., 0.])
>>> kerpredict.run(alpha,X,y,np.array([1, -2, 0, 1, 2, 0.5, 0]))
-1.0
>>> kerpredict.run(alpha,X,y,np.array([-2, 2, 0, 1, 2, 0.5, 0]))
1.0
```