Georgia Institute of Technology

ECE 4252/8803: Fundamentals of Machine Learning (FunML)

Spring 2025

Homework Assignment # 1

Due: January 24 @8PM

Please read the following instructions carefully.

- The entire homework assignment is to be completed on this `ipython` notebook. It is designed to be used with `Google Colab`, but you may use other tools (e.g., AI Makerspace, Jupyter Notebook) as well.
- Make sure that you execute all cells in a way so their output is printed beneath the corresponding cell. Thus, after successfully executing all cells properly, the resulting notebook has all the questions and your answers.
- Make sure you delete any scratch cells before you export this document as a PDF. Do not change the order of the questions and do not remove any part of the questions. Edit at the indicated places only.
- Print a PDF copy of the notebook with all its outputs printed. **Submit PDF on `Gradescope`.** When submitting PDF on Gradescope, make sure to match each question to the corresponding pages. **Incorrect page assignment may lead to reduction of points.**
  Then, zip both **PDF** and **IPYNB** in a single **ZIP** file and submit it on `Canvas` under Assignments.
- It is encouraged for you to discuss homework problems amongst each other, but any copying is strictly prohibited and will be subject to Georgia Tech Honor Code.
- Late homework is not accepted unless arranged otherwise and in advance.
- Comment on your codes.
- **IMPORTANT:** Start your solution with a **BOLD RED** text that includes the words *solution* and the part of the problem you are working on. For example, start your solution for Part (c) of Problem 2 by having the first line as:
  Solution to Problem 2 Part (c). Failing to do so may result in a *20% penalty* of the total grade.

Assignment Objectives:

This homework assignment is designed with the following objectives:

- Familiarizing students with Markdown and LaTeX in Ipython Notebooks
- Introduction to the use of Numpy for Matrix and vector operations and for setting up arrays
- Loading and visualizing data
- Perform simple data analysis

Guide for Exporting Ipython Notebook to PDF:

Remeber to convert your homework into PDF format before submitting it.
Here is a video summarizing how to export ipython Notebook into PDF.

- **[Method1: Print to PDF]**
  After you run every cell and get their outputs, you can use **[File] -> [Print]** and then choose **[Save as PDF]** to export this Ipython Notebook to PDF for submission.
  Note: Sometimes figures or texts are splited into different pages. Try to tweak the layout by adding empty lines to avoid this effect as much as you can.
- **[Method2: colab-pdf script]**
  The author of that video provided an alternative method that can generate better layout PDF. However, it only works for Ipythin Notebook without embedded images.
  How to use: Put the script below into cells at the end of your Ipython Notebook. After you run the fisrt cell, it will ask for google drive permission. Executing the second cell will generate the PDF file in your google drive home directory. Make sure you use the correct path and file name.

```
## this will link colab with your google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
%%capture
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('LastName_FirstName_ECE_4252_F24_assignment_#.ipynb') ## change path and file name
```

- **[Method3: GoFullPage Chrome Extension] (most recommended)**
  Install the extension and generate PDF file of the Ipython Notebook in the browser.

Note: Since we have embedded images in HW1, it's recommended to generate PDF using the first method. Also, Georgia Tech provides a student discount for Adobe Acrobat subscription. Further information can be found here.

Problem 1: Markdown Basics (10pts)

Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML). Jupyter notebooks use Markdown cells for inserting any content that is *not* code. For problem1, we are going to have you create a **level 3 heading** and a **blockquote** containing a **list** that has **in turn additional sublists**, like shown in the image below. Your task is to recreate this list with a heading within a single markdown cell and execute it.

Level 3 Heading

> - item 1
>   - subitem 1
>   - subitem 2
>   - subitem 3
> - item 2
>   - subitem 1
>   - subitem 2
>   - subitem 3

Solution to Problem 1

Level 3 Heading

- item 1
  - subitem 1
  - subitem 2
  - subitem 3
- item 2
  - subitem 1
  - subitem 2
  - subitem 3

Problem 2: Creating Codeblocks (10pts)

Markdown enables the creation of code blocks within markdown cells for a better differentiation of regular text from code. Write and execute the markdown syntax required to create the block of `Python` code within a code block as depicted in the image below:

```
import numpy as np
import matplotlib.pyplot as plt
import sklearn
```

Solution to Problem 2

```
import numpy as np
import matplotlib.pyplot as plt
import sklearn
```

Problem 3: LaTeX in Markdown (10pts)

LaTeX is widely used in academia for the communication and publication of scientific documents in many fields, including engineering, mathematics, statistics, computer science, physics, economics, linguistics, quantitative psychology, philosophy, and political science. It is especially useful when it comes to typesetting mathematical expressions, tables, and matrices. The image below shows various mathematical expressions typed out using `LaTeX` in markdown. Your task is to recreate them in the cell underneath.

$$A = \begin{bmatrix} 4 & 5 \\ 3 & 4 \end{bmatrix}, \text{and } x = \begin{bmatrix} 2 \\ 4 \end{bmatrix}. \text{Then } y = Ax = \begin{bmatrix} 28 \\ 22 \end{bmatrix}$$

$$\sigma(x) = \frac{1}{1+e^{-\theta^T x}}$$

$$P(\theta|x_1) = \frac{P(x_1|\theta) \times P(\theta)}{P(x_1)}$$

Solution to Problem 3

$$A = \begin{bmatrix} 4 & 5 \\ 3 & 4 \end{bmatrix}, \text{ and } x = \begin{bmatrix} 2 \\ 4 \end{bmatrix}. \text{ Then } y = Ax = \begin{bmatrix} 28 \\ 22 \end{bmatrix}$$

$$\sigma(x) = \frac{1}{1+e^{-\theta^T x}}$$

$$P(\theta|x_1) = \frac{P(x_1|\theta) \times P(\theta)}{P(x_1)}$$

Problem 4: Introduction to Linear Algebra with `Python` and `Numpy` (20pts)

This exercise introduces a few basic linear algebra concepts that you can carry out with `Python` and the standard library `numpy`. We will go through some basic matrix operations, analyze inverse problem using several matrix properties and decompositions and calculate matrix approximations.

Hint: In order to import numpy, check the following url.

Problem 4 (a) Basic Numpy operations

Given $A = \begin{bmatrix} 1 & 1 & 2 \\ -2 & -1 & 1 \end{bmatrix}, B = \begin{bmatrix} 2 & -3 & 1 \\ 0 & -1 & 2 \end{bmatrix}, v_1 = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$ and $v_2 = \begin{bmatrix} -5 \\ 3 \\ 2 \end{bmatrix}$. Use `numpy` to calculate the following:

i. Create matrix A, B and vector v_1 and v_2 using `numpy` array.
ii. Calculate $v_3 = v_1 + v_2$ and $v_4 = v_1 - v_2$, using v_1 and v_2 from i.
iii. Calculate $C = A \odot B$, which is the **Hadamard product** (also known as element-wise multiplication) and **matrix products** $D = Av_1$ and $E = A^T A$.
iv. Calculate the inverse of E, i.e. $F = E^{-1}$. (You might also try taking the inverse of a singular matrix to see the output.)

Solution to Problem 4a

```
import numpy as np

## Problem 4(a) question i
A = np.array([[1, 1, 2], [-2, -1, 1]])
B = np.array([[2, -3, 1], [0, -1, 2]])

v1 = np.array([[2], [1], [0]])
v2 = np.array([[-5], [3], [2]])
```

```
## Problem 4(a) question ii
v3 = v1 + v2
v4 = v1 - v2

## Problem 4(a) question iii
C = A * B
D = A @ v1
E = A.T @ A

## Problem 4(a) question iv
F = np.linalg.inv(E)

#-----------------Don't change anything below-----------------------#
print('[Question i]')
print('A: \n',A)
print('B: \n',B)
print('v1: \n', v1)
print('v2: \n', v2)

print('[Question ii]')
print('v3=v1+v2: \n', v3)
print('v4=v1-v2: \n', v4)

print('[Question iii]')
print('C = Hadamard Product of A and B: \n',C)
print('D = Matrix Product of A and v1: \n',D)
print('E = Matrix Product of A^T and A: \n',E)

print('[Question iv]')
print('F = E^-1: \n',F)
```

```
[Question i]
A:
 [[ 1  1  2]
 [-2 -1  1]]
B:
 [[ 2 -3  1]
 [ 0 -1  2]]
v1:
 [[2]
 [1]
 [0]]
v2:
 [[-5]
 [ 3]
 [ 2]]
[Question ii]
v3=v1+v2:
 [[-3]
 [ 4]
 [ 2]]
v4=v1-v2:
 [[ 7]
 [-2]
 [-2]]
[Question iii]
C = Hadamard Product of A and B:
 [[ 2 -3  2]
 [ 0  1  2]]
D = Matrix Product of A and v1:
 [[ 3]
 [-5]]
E = Matrix Product of A^T and A:
 [[5 3 0]
 [3 2 1]
 [0 1 5]]
[Question iv]
F = E^-1:
 [[-1.35107989e+15  2.25179981e+15 -4.50359963e+14]
 [ 2.25179981e+15 -3.75299969e+15  7.50599938e+14]
 [-4.50359963e+14  7.50599938e+14 -1.50119988e+14]]
```

Problem 4 (b) Linear inverse problem $y = Ax$

Inverse problem is a very common engineering problem across different fields of study, e.g. medical image applications and parameter estimation. Its simplest form (without noise) is composed of **a measurement** y, **a system matrix** A, and **the system parameter** x. We also know the forward part $y = Ax$. Our task is to find x using y and A. There are several algorithms focusing on inverse problems such as least

square (we will learn this in later lectures), compressed sensing and deep learning approach. Part of this homework is to get you try to seek assistance by searching through `numpy` and other resources.

Suppose that $A = \begin{bmatrix} 1 & -2 & 0 & 2 \\ 0 & -1 & 0 & 1 \\ 0 & 1 & -2 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$ and $y_a = \begin{bmatrix} 3 \\ 1 \\ -1 \\ 2 \end{bmatrix}$.

i. Are the rows in A linearly independent? Calculate the **inverse** of A and the **eigen decomposition** $V_A \Lambda_A V_A^{-1}$ of A using `numpy`.

ii. Using the inverse of A, calculate the solution x for $y_a = Ax$. (Assign it to x_1)

iii. Using the eigen decomposition of A, calculate the solution x for $y_a = Ax$. (Assign it to x_2)

iv. Now suppose we have a different system matrix B and measuerment y_b: $B = \begin{bmatrix} 2 & -3 & 1 & -4 \\ 1 & 0 & 0 & 2 \\ 0 & 1 & -3 & 5 \\ 2 & 2 & -1 & 1 \\ 3 & 3 & 1 & -2 \end{bmatrix}$ and $y_b = \begin{bmatrix} -1 \\ 3 \\ 2 \\ 2 \\ 2 \end{bmatrix}$.

What is the rank of B? Calculate the **SVD decomposition** $U_B D_B V_B^T$ of B using `numpy`.

v. Since B has more rows than columns, we need Moore-Penrose Pseudo-inverse $B^\dagger = (B^T B)^{-1} B^T$ to calculate the solution x. Calculate x using U_B, D_B and V_B. (Assign it to x_3)

Solution to Problem 4b

```
y_a = np.transpose(np.array([[3, 1, -1, 2]]))
A = np.array([[1, -2, 0, 2],
              [0, -1, 0, 1],
              [0, 1, -2, 1],
              [1, 0, 0, 1]])

y_b = np.transpose(np.array([[-1, 3, 2, 2, 2]]))
B = np.array([[2, -3, 1, -4],
              [1, 0, 0, 2],
              [0, 1, -3, 5],
              [2, 2, -1, 1],
              [3, 3, 1, -2]])
#-----------------Don't change anything above-----------------------#

## Problem 4(b) question i
A_inverse = np.linalg.inv(A)
eigenvalues, eigenvectors = np.linalg.eig(A)
Lamda_A = np.diag(eigenvalues)
V_A = eigenvectors


## Problem 4(b) question ii
x1 = np.dot(A_inverse, y_a)

## Problem 4(b) question iii
V_A_inverse = np.linalg.inv(V_A)
Lamda_inverse = np.linalg.inv(Lamda_A)
x2 = np.dot(np.dot(V_A_inverse, Lamda_inverse), np.dot(V_A, y_a))


## Problem 4(b) question iv
B_rank = np.linalg.matrix_rank(B)
U_B, D_B, V_B_transpose = np.linalg.svd(B)
D_B = np.diag(D_B)

## Problem 4(b) question v
D_B_inverse = np.zeros(B.shape).T
np.fill_diagonal(D_B_inverse, 1 / D_B)

B_pseudo_inverse = np.dot(np.dot(V_B_transpose.T, D_B_inverse), U_B.T)
x3 = np.dot(B_pseudo_inverse, y_b)

#-----------------Don't change anything below-----------------------#
print('inverse of A: \n', A_inverse)
print('lambda of A: \n', Lamda_A)
print('V_A(eigenvectors): \n', V_A)

print('x1: \n', x1)
```

```
print('x2: \n', x2)

print('rank of B: ', B_rank)
print('U_B:, \n', U_B)
print('D_B:, \n', D_B)
print('V_B^T:, \n', V_B_transpose)

print('x3: \n', x3)
```

```
inverse of A:
 [[ 1.  -2.   0.   0. ]
 [-1.   1.   0.   1. ]
 [-1.   1.5 -0.5  1. ]
 [-1.   2.   0.   1. ]]
lambda of A:
 [[-2.          0.          0.          0.        ]
 [ 0.         -1.4811943   0.          0.        ]
 [ 0.          0.          0.31110782  0.        ]
 [ 0.          0.          0.          2.17008649]]
V_A(eigenvectors):
 [[ 0.          0.6243338   0.42414154  0.73014725]
 [ 0.          0.52292041 -0.46959249  0.19684364]
 [ 1.          0.52292041 -0.46959249  0.19684364]
 [ 0.         -0.25162632 -0.61568639  0.62401135]]
x1:
 [[1.]
 [0.]
 [1.]
 [1.]]
x2:
 [[15.03451634]
 [-1.91789047]
 [ 0.05470931]
 [ 0.47107754]]
rank of B:  4
U_B:,
 [[-0.61637039  0.08678363  0.75940516  0.1606979   0.10018049]
 [ 0.19248146 -0.08968427  0.36573823 -0.90417375 -0.06010829]
 [ 0.70886224 -0.02864004  0.45517644  0.30855955  0.44079414]
 [ 0.18257136 -0.51213591  0.25558792  0.24366463 -0.7613717 ]
 [-0.21730267 -0.84930671 -0.13049356 -0.0454376   0.46083024]]
D_B:,
 [[8.12182025 0.         0.         0.        ]
 [0.         5.23174328 0.         0.        ]
 [0.         0.         2.85295385 0.        ]
 [0.         0.         0.         1.23513257]]
V_B^T:,
 [[-0.16339005  0.27964275 -0.38696143  0.86334494]
 [-0.66675843 -0.73802989 -0.03143638  0.09877668]
 [ 0.7025153  -0.59704571 -0.34778188  0.17045949]
 [-0.18764098  0.14369496 -0.85341688 -0.46456673]]
x3:
 [[nan]
 [nan]
 [nan]
 [nan]]
<ipython-input-38-7dc00996dbfb>:38: RuntimeWarning: divide by zero encountered in divide
  np.fill_diagonal(D_B_inverse, 1 / D_B)
```

Problem 5: Vector Norms (15pts)

Having seen some examples of matrix vector products, we now move onto the subject of vector norms, which quantify vectors, in magnitude. There are a number of vector norms. L_2 and L_1 are among the most common and frequently used norms. For a vector $v = [v_1, v_2, \cdots, v_n]^T$, we have:

$||v||_2 = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$ (L_2 norm of v)

$||v||_1 = |v_1| + |v_2| + \cdots + |v_n|$ (L_1 norm of v)

You can use the `numpy` function `numpy.linalg.norm` to compute vector and matrix norms of different kinds by passing the appropriate arguments as shown below:

```
from numpy.linalg import norm

l2 = norm(v,2)  # computes l2 norm of v
l1 = norm(v,1)  # computes l1 norm of v
```

Your tasks are as follows:

a. Write your own function to compute the L_1 **norm**. Complete the function definition given below by using the norm definition above. In this part, **do not use** the `numpy` L_1 function nor any for loop.
b. Write your own function to compute the L_2 **norm**. Complete the function definition given below by using the norm definition above. In this part, **do not use** the `numpy` L_2 function nor any for loop.
c. Compare your answers on any vector of your choice to the ones obtained by numpy's norm function

Hint: You may need to utilize one of the following functions:

- np.array
- np.linalg
- np.matmul
- np.sum

Solution to Problem 5

```
from numpy.linalg import norm
import numpy as np

def l2(v):
    '''Function computes l2 norm for given vector v'''

    l2_norm = np.sqrt(np.sum(v ** 2))

    return l2_norm

def l1(v):
    '''Function computes l1 norm for given vector v'''

    l1_norm = np.sum(np.abs(v))

    return l1_norm

v =  np.array([2,3,4])


#---------------------------------Don't change below!-------------------------------------#
print("L2 norm with \'norm\': %.4f" %norm(v,2),' | L2 norm with function: %.4f' %l2(v))
print('L1 norm with \'norm\': %.3f' %norm(v,1),' | L2 norm with function: %.3f'%l1(v))
```

```
L2 norm with 'norm': 5.3852  | L2 norm with function: 5.3852
L1 norm with 'norm': 9.000  | L2 norm with function: 9.000
```

Problem 6: Probability theory review (Optional)

In this problem, we will test some of the basic concepts in probability theory. Please answer these questions in the text cell with Latex in Markdown.
Note: This question is optional and it's not necessary to turn it in, but it may serve as a good excercise for you to review some of fundamentals of probability theory.

Problem 6 (a)
Suppose that X and Y are two independent random variables. Show that they are uncorrelated in the discrete case. Recall that the two random variables X and Y are independent when $P(X \cap Y) = P(X)P(Y)$. They are uncorrelated when $E[(X - E(X))(Y - E[Y])] = 0$.

Problem 6 (b)
Let X and Y be two independent random variabales, and $X \sim N(\mu_X, \sigma_X^2)$ and $Y \sim N(\mu_Y, \sigma_Y^2)$. Assume $Z = X + Y$, show that $Z \sim N(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2)$.

Problem 6 (c)
Let $X_1, X_2, \ldots, X_n$ be independent random variables (not necessarily Gaussian) with means $\mu_{X_1}, \mu_{X_2}, \ldots, \mu_{X_n}$ and variances $\sigma_{X_1}^2, \sigma_{X_2}^2, \ldots, \sigma_{X_n}^2$ respectively. Let $Y = \sum_{k=1}^n X_k$. Show that the mean of Y, μ_Y satisfies $\mu_Y = \sum_{k=1}^n \mu_{X_k}$ and the variance of Y, σ_Y^2 satisfies $\sigma_Y^2 = \sum_{k=1}^n \sigma_{Y_k}^2$.

Problem 6 (d)

X is a uniform random variable that $X \sim Unif(0, 10)$ and given an event $A = \{1 \leq X \leq 5\}$

i. What is the conditional PDF, $f_{X|A}(x)$
ii. What is the conditional expected value, $E[X|A]$
iii. What is the conditional variance, $Var[X|A]$

[Put your answer in this cell]

Problem 7: Loading and Visualizing Datasets (35pts)

Problem 7 (a)

For the purposes of this section, we will have you load and visualize the Iris dataset and its various features. The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by the British statistician, eugenicist, and biologist Ronald Fisher in his 1936 paper *The use of multiple measurements in taxonomic problems* as an example of linear discriminant analysis. You may import the dataset using the popular sklearn library by running the following snippet of code:

```
from sklearn.datasets import load_iris

iris = load_iris()
```

(***Note***: You may have to install sklearn via the `pip install sklearn` command.)

Explore various features of the dataset, and then answer the following questions:

i. How many feature attributes does the dataset have?
ii. What are the feature names for the attributes?
iii. How many different target classes exist in the dataset?
iv. What are the class names?

The names of the various features and targets should printed in an output cell once you execute it. (***Hint***: It may be helpful to refer to the documentation for the `load_iris` function.)

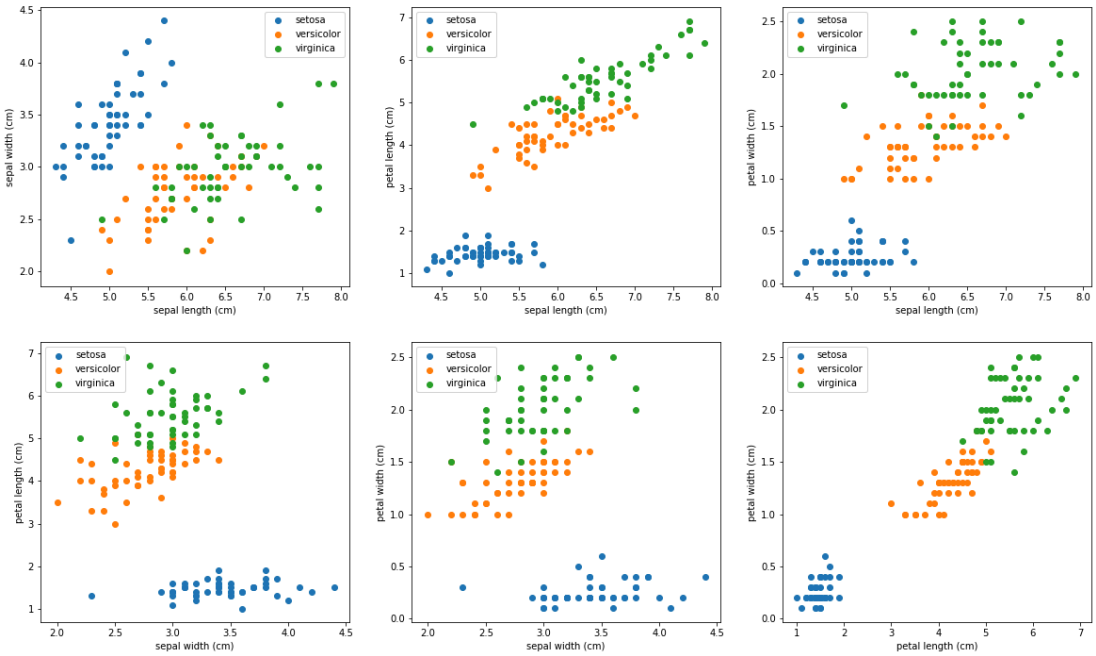
Solution to Problem 7a

```
# pip install sklearn  (keep commented if sklearn already installed)
from sklearn.datasets import load_iris

iris = load_iris()

#-----------Write Code below----------------#

print(iris.keys())

## question i
print('Number of feature is:', len(iris.feature_names))

## question ii
print('Feature names are:', iris.feature_names)


## qustion iii
print('Number of target classes is:', len(iris.target_names))


## question iv
print('Class names are:', iris.target_names)
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
Number of feature is: 4
Feature names are: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Number of target classes is: 3
Class names are: ['setosa' 'versicolor' 'virginica']
```

Problem 7 (b)

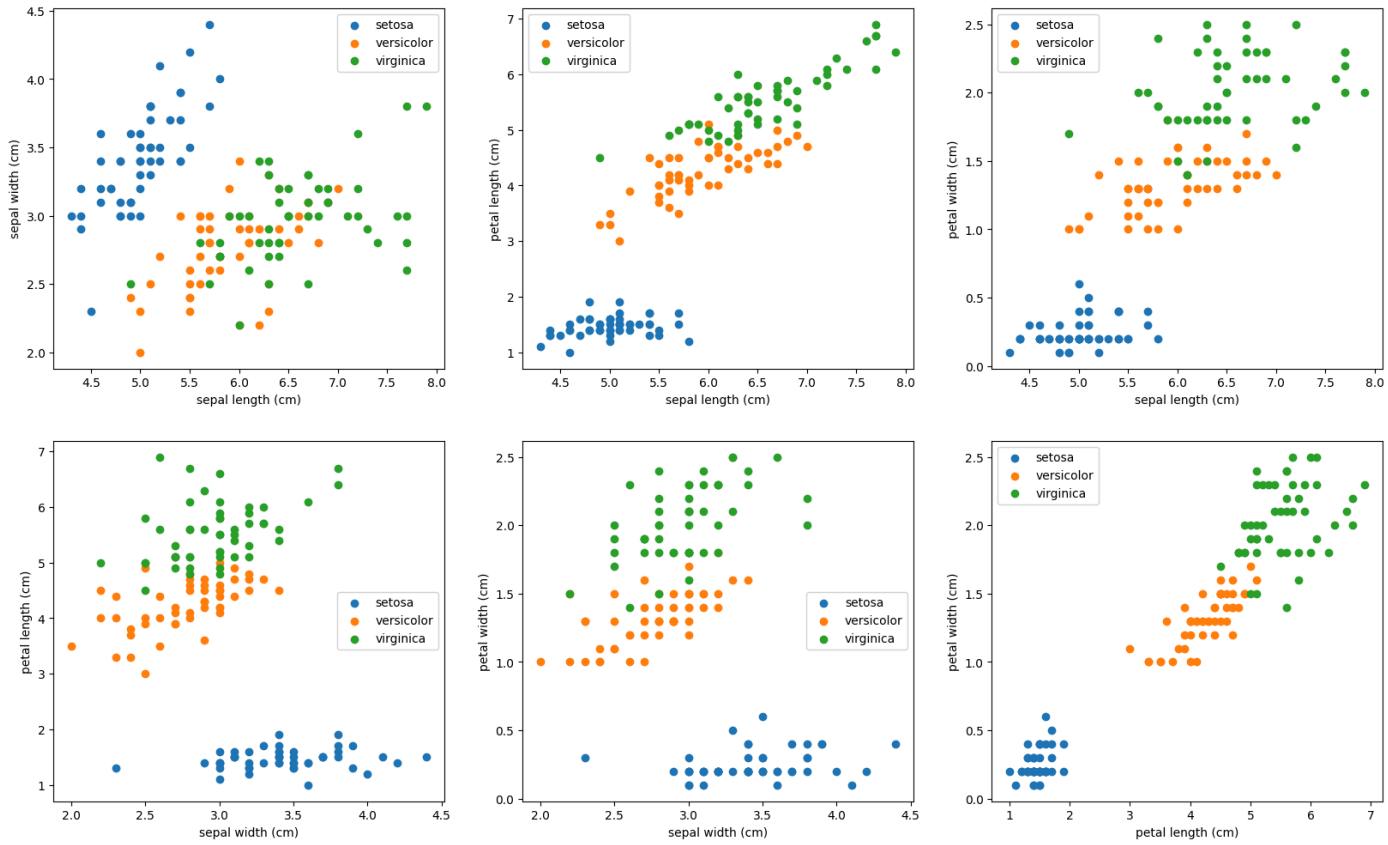
We are now going to visualize the data. Your analysis of the Iris dataset above should have revealed that it has more than two features. For ease of use, however, we are only going to use two of these features and plot them against each other in a scatter plot for each training example in the dataset. We are going to use the standard plotting library, **Matplotlib**, for this purpose. Once again, you might have to install the matplotlib library via a `pip install matplotlib` command. **Fill in the template code provide below in the places indicated to generate 2D scatter plots for various feature combinations as shown in the image below.**



Solution to Problem 7b

```
import matplotlib.pyplot as plt
import numpy as np
import itertools

def combinations(iterable, r=2):
    """Function generates unique pairwise permutations for the given iterable"""
    pool = tuple(iterable)
    n = len(pool)
    if r > n:
        return
    indices = list(range(r))
    yield tuple(pool[i] for i in indices)
    while True:
        for i in reversed(range(r)):
            if indices[i] != i + n - r:
                break
        else:
            return
        indices[i] += 1
        for j in range(i+1, r):
            indices[j] = indices[j-1] + 1
        yield tuple(pool[i] for i in indices)


feature_names = iris.feature_names
class_names =  iris.target_names

features =  iris.data
labels =  iris.target

# Extract total number of crossplots possible with the given feature sizes
```

```
# Extract total number of crossplots possible with the given feature sizes
total_plots = len([*combinations(list(range(len(feature_names))), 2)])

# generate array of plots of the desired shape. play around with this to get the appropriate shape
fig, axes = plt.subplots(2,3, figsize=(20,12))

for ax, feature_combination in zip(axes.flatten(), combinations(list(range(len(feature_names))), 2)):

    feature_1 = feature_combination[0]
    feature_2 = feature_combination[1]

    for class_num in np.unique(labels):
        ax.scatter(features[:,feature_1][labels==class_num], features[:,feature_2][labels==class_num], label = class_names[class_num])

    ax.set_xlabel(feature_names[feature_1])
    ax.set_ylabel(feature_names[feature_2])
    ax.legend()

plt.show()
```



Problem 7 (c)

Do all feature crossplots result in well-distinguished classes? Based off your plots, which feature combinations result in the most poorly-distinuished classes? Which two features give the most well-distinuished classes?

Solution to Problem 7c

No, almost all of the feature crossplots have moderate amounts of class overlap which means it is difficult for classification based on these pairs of features.

The feature combination of sepal width and sepal length resulted in the most poorly distinguished classes.

The feature combination of petal width and petal length resulted in the most well-distinguished classes.