University of Canberra

Faculty of Science and Technology

**Programming for Data Science G (11521)**

**Week 12 Tutorial**

**GUI, Cross Validation and Python Class**

**Objectives**

* To implement a simple graphical user interface (GUI) application
* To implement a cross validation program to find best parameters.
* To create a class and add fields and methods to this class.

**Project**

* Create a new project **Week12Tutorial** using Visual Studio or PyCharm

**GUI App**

* You will use examples in Python GUI lecture (Week 6) to implement this app
* Add a new file to project and name it **gui\_app.py**.
* Add the following code to create a window with title, size and offset then run it

import tkinter as tk

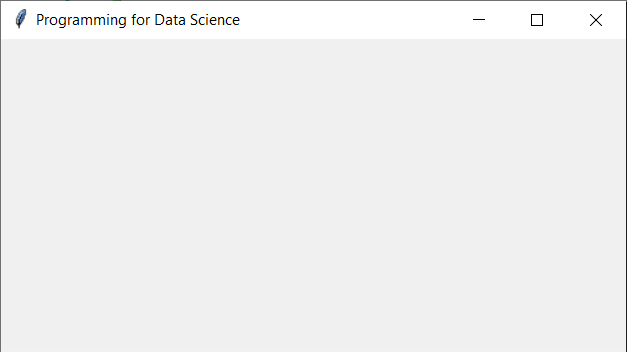
window = tk.Tk()

window.title('Programming for Data Science')

# width x height + x\_offset + y\_offset:

window.geometry("500x250+100+100")

window.mainloop()



* **Note: all Python code in the next steps must be added to the place shown below**

window.geometry("500x250+100+100")

Add Python code in the next steps here

window.mainloop()

* Add the following code to set font and display a label on that window

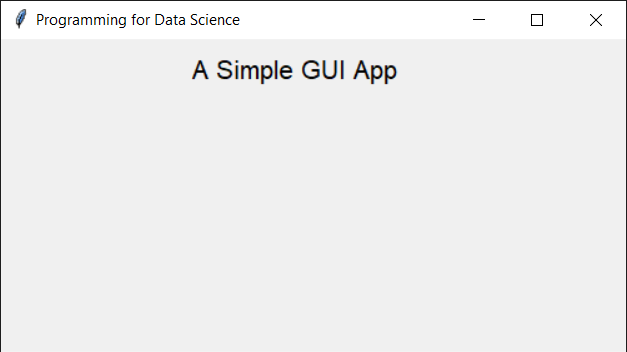
#Set font

myfont = "Arial, 16"

#Add a label

lbl\_header = tk.Label(text="A Simple GUI App", font=myfont, height=1)

lbl\_header.place(x=150, y=10)



* And a label and two radio buttons for user to select a colour

#Add label

lbl = tk.Label(text="Select a colour: ", fg="navy", anchor="w", width=25, height=1, font=myfont)

lbl.place(x=10, y=50)

#Add variable var and 2 radio buttons

var = tk.StringVar()

rb1 = tk.Radiobutton(text="red", variable=var, value='r', font=myfont)

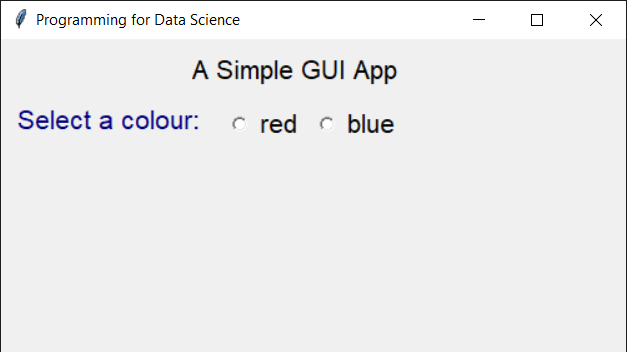
rb1.place(x=180, y=50)

rb1.deselect()

rb2 = tk.Radiobutton(text="blue", variable=var, value='b', font=myfont)

rb2.place(x=250, y=50)

rb2.deselect()



* The two radio buttons have the same variable var and have their own value ('r' or 'b'). When the user selects a radio button, its value will be set to the variable var.
* You will add a button for the user to click after the user selects a colour. You will also add another label to display a message that shows what colour the user has selected. This is done by the function select\_item attached to the button. Add the following code to do that

#Label to display output when button is clicked

lb\_output = tk.Label(text="", fg="navy", anchor="w", width=25, height=1, font=myfont)

lb\_output.place(x=10, y=100)

#################################

def select\_item():

selected = var.get()

if selected == 'r':

output = 'Red selected'

elif selected == 'b':

output = 'Blue selected'

else:

output = "Please select a colour"

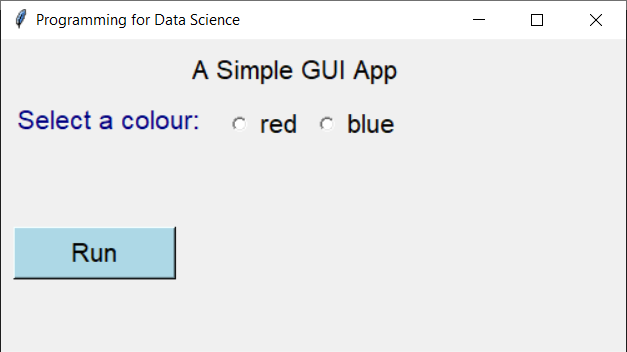
lb\_output.config(text=output)

#################################

#Add button

button = tk.Button(text="Run", fg="black", bg="lightblue", width=10, height=1, font=myfont, command=select\_item)

button.place(x=10, y=150)



* Now you run the program. Click the button to see the message then select a radio button then click the button again.

**Parameter estimation using grid search with cross-validation for Classification**

* You will write a program to implement Step 6 in Assignment 2 here using **GridSearchCV** function in **Scikit-learn** package. This function will input a classifier with different values of its parameter, a training dataset, and a score. The function will then partition the training dataset into K subsets, where K-1 subsets are used to train the classifier with different parameter values and the remaining subset is to validate the classifier. The **GridSearchCV** function will output scores and parameter values and selects the best parameter value that corresponds to the highest score.
* Add a new Python file to your project and name it **gridsearchcv.py**.
* Add the following code to load dataset, parameter (n\_neighbors), and classifier (KNeighborsClassifier) that you already tried them in last week’s tutorial)

from sklearn import datasets, neighbors, metrics, mixture, svm

from sklearn.model\_selection import train\_test\_split, GridSearchCV

import matplotlib.pyplot as plt

import numpy as np

# Loading a dataset

dataset = datasets.load\_iris()

X = dataset.data

y = dataset.target

class\_names = dataset.target\_names

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=0)

# Load classifier and its parameter for cross-validation

parameter = [{'n\_neighbors': [1, 2, 3, 4, 5]}]

classifier = neighbors.KNeighborsClassifier()

* Add the following code to load grid search cross validation function for parameter estimation

# Load grid search cross validation

gscv\_classifier = GridSearchCV(

estimator=classifier,

param\_grid=parameter,

cv=5, #5-fold cross validation

scoring='accuracy'

)

* This function returns classifier gscv\_classifier and we use it to train the training set

gscv\_classifier.fit(X\_train, y\_train)

* Get parameter values, scores (accuracies) and best parameter from this gscv\_classifier classifier

print("Grid scores on validation set:")

print()

means = gscv\_classifier.cv\_results\_['mean\_test\_score']

stds = gscv\_classifier.cv\_results\_['std\_test\_score']

results = gscv\_classifier.cv\_results\_['params']

for mean, std, param in zip(means, stds, results):

print("Parameter: %r, accuracy: %0.3f (+/-%0.03f)" % (param, mean, std\*2))

print()

print("Best parameter:", gscv\_classifier.best\_params\_)

* This gscv\_classifier classifier now applies the best parameter, so we just use it to test the testing dataset

y\_pred = gscv\_classifier.predict(X\_test)

* Plot confusion matrix and accuracy

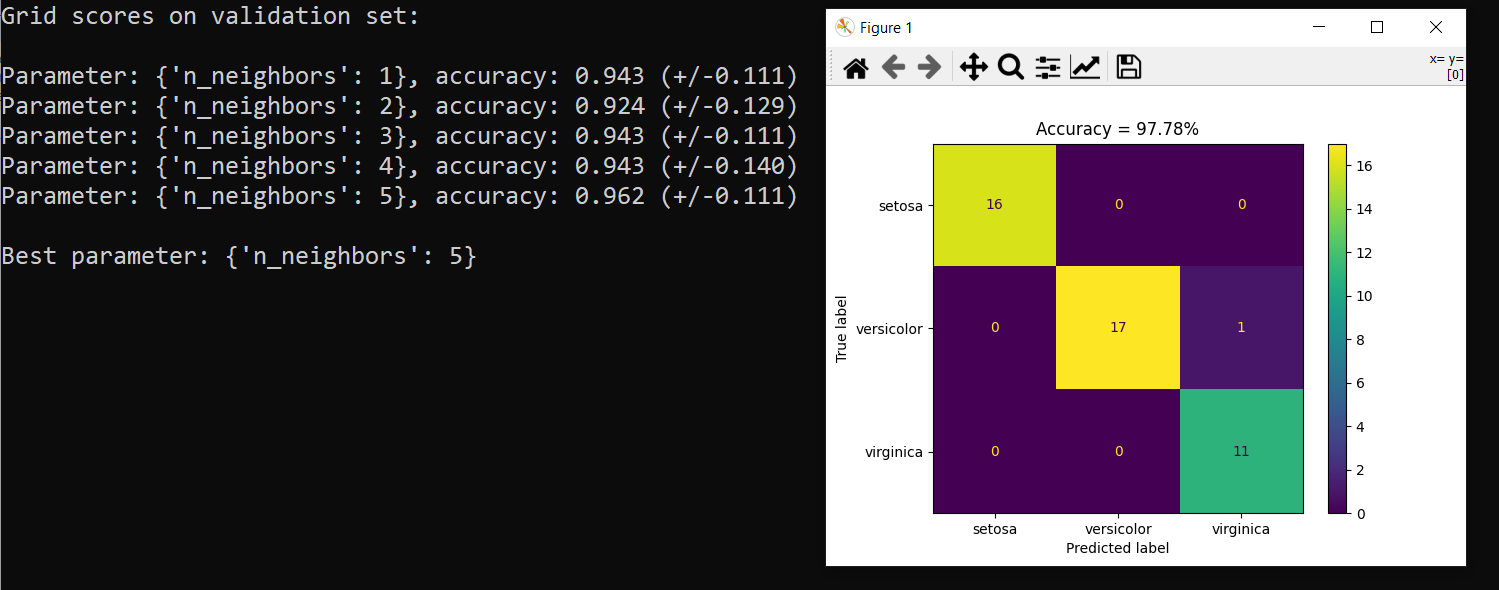
accuracy = metrics.accuracy\_score(y\_test, y\_pred) \* 100

plotcm = metrics.plot\_confusion\_matrix(gscv\_classifier, X\_test, y\_test, display\_labels=class\_names)

plotcm.ax\_.set\_title('Accuracy = {0:.2f}%'.format(accuracy))

plt.show()

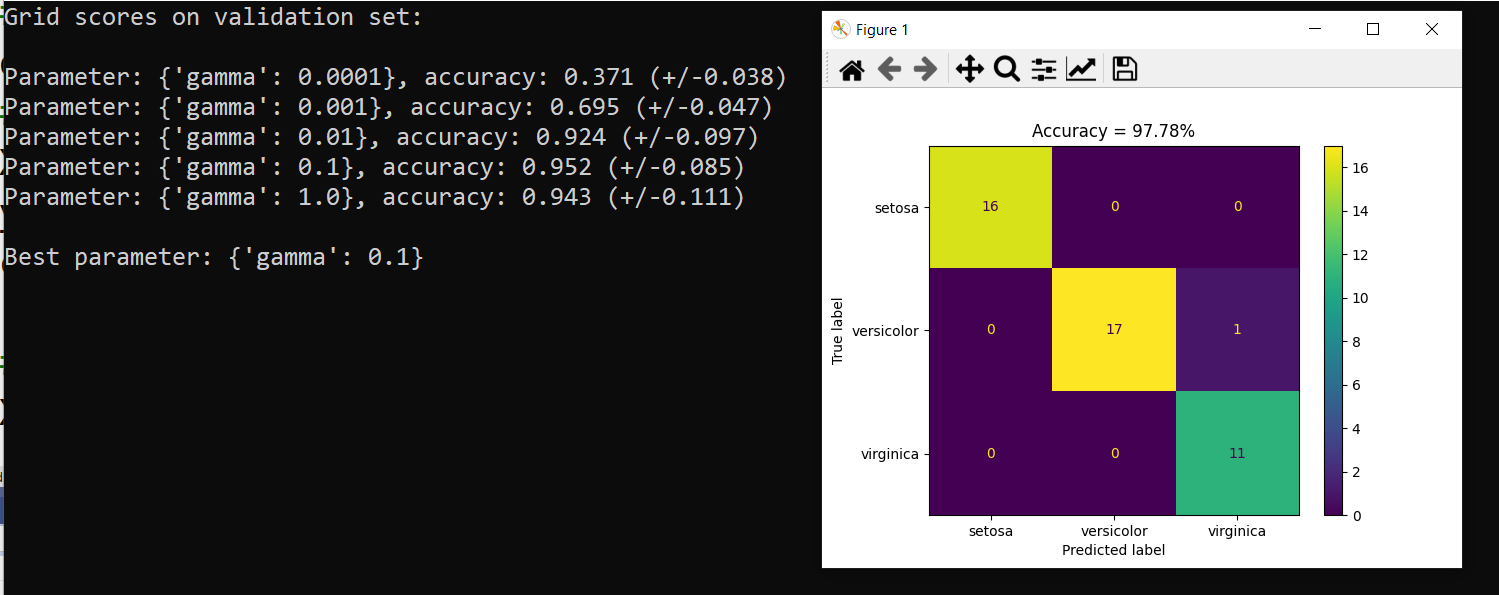
* Run the program.



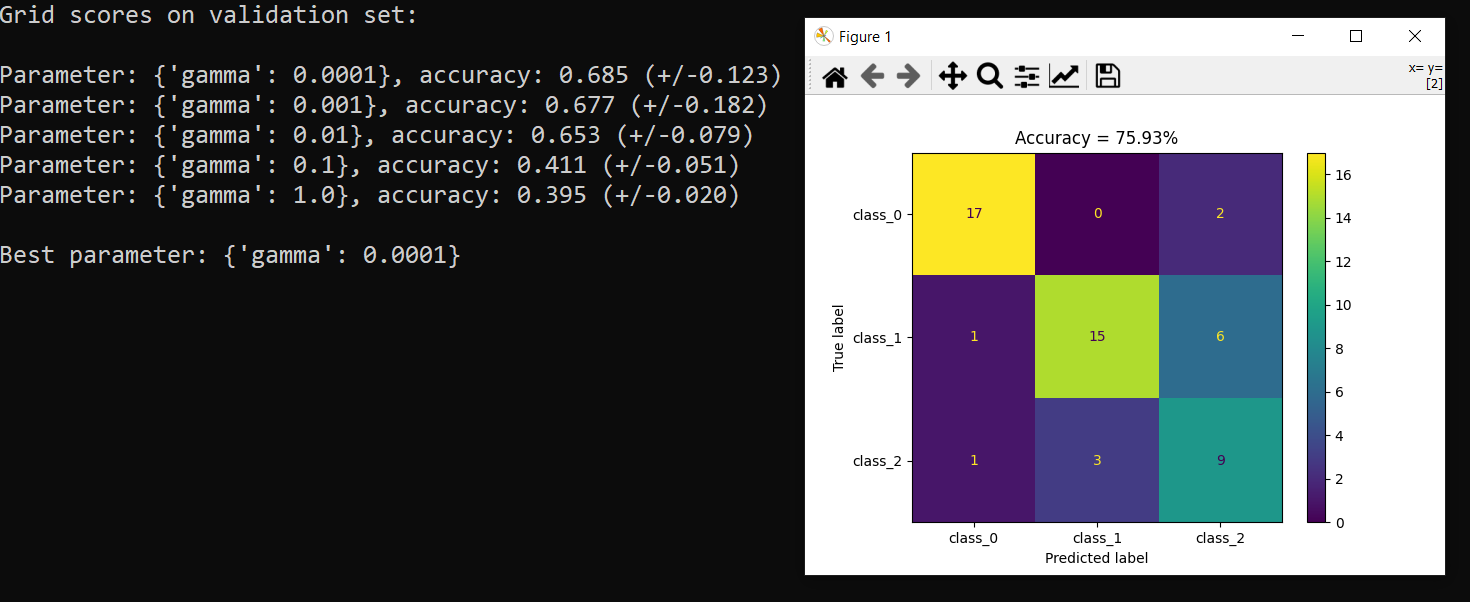
* Change test\_size=0.3 to test\_size=0.25, or test\_size=0.2 and run your program to see the outputs.
* Change the current dataset (datasets.load\_iris()) to **breast cancer** dataset (dataset = datasets.load\_breast\_cancer()) or wine (dataset = datasets.load\_wine()) and run your program.

**Question 1:**

* Modify your program to run the SVM classifier (svm.SVC()) and its parameter gamma with the following values: 0.0001, 0.001, 0.01, 0.1, and 1.0..
* Output for the wine dataset, test\_size=0.3



* Output for the iris dataset, test\_size=0.3



**Question 2 (Python Class):**

* Add a new file to the project to implement a Python class that is similar to the Phone class in lecture with the following details
  + Class name: **Computer**
  + Class variable: **brand**
  + Object variables: **model** and **harddisk**
  + \_\_init\_\_ method to set up initial values for all variables
  + Class method: **change\_brand** to change brand
  + Object methods: **change\_model** and **change\_harddisk** to change model and harddisk, respectively.
* Write a program that creates 2 instances (objects) of Computer class and call all methods to change all the fields and to demonstrate the differences between class method and object method.

--END--