CS3033 - CS6405 - Data Mining
First Assignment

Lecturer: Dr Andrea Visentin

Submission

This assignment is due on 13/03/22 at 23:59. You should submit a single .ipnyb file with your python code and analysis electronically via Canvas.

Please note that this assignment will account for **25 Marks** of your module grade.

Declaration

By submitting this assignment. I agree to the following:
"I have read and understand the UCC academic policy on plagiarism and I agree to the requirements set out thereby in relation to plagiarism and referencing. I confirm that I have referenced and acknowledged properly all sources used in preparation of this assignment.
I declare that this assignment is entirely my own work based on my personal study. I further declare that I have not engaged the services of another to either assist me in, or complete this assignment"

Specification

The objective of this project is to build a k-Nearest Neighbour algorithm that takes as input training and test dataset and will predict the target binary variable with a reasonable degree of accuracy.

You will find a template for your python code in Canvas along with the dataset. In this project, you must implement your own machine learning library and therefore **you are only allowed to use the following python packages and libraries: NumPy, pandas.**

This project focuses on the Titanic dataset that we used during the lectures. You can find the dataset at:
https://github.com/andvise/DataAnalyticsDatasets/blob/main/titanic.csv

Tasks

Data Preparation (5 marks)

1. Load the dataset on Colab
2. Display the attributes' name and their data type
3. Delete the columns **PassengerId** and **Name**
4. Replace all missing values with 0
5. Transform the **Sex** column into a numerical one
6. Use **Survived** as the target label and the rest of the data frame as features
7. Divide your dataset in 80% for training and 20% for test
8. Scale the columns using min-max scalers
9. Print the shape of the train and test set

k-NN implementation (10 Marks)

1. Implement your k-NN method. To classify **each** point (query point) of the test set, you need to:
    a. Calculate the **Euclidean** distance between the query point (each point in the testing set) and all the training points of the training set.
    b. Then pick the **k** points with the smallest distance to the query point (**k** must be a hyperparameter)
    c. Select the most common class among the **k** points
2. Compute the accuracy and plot the confusion matrix
    a. Briefly describe what the confusion matrix shows us.

Reminder: You can not use any library with the exception of pandas and NumPy. So scikit-learn is forbidden!

Hyperparameters search (5 Marks)

1. Test [1, 3, 5, 7, 9, 11] as possible k values
    a. Select the best one and explain why

Weighted k-NN (5 Marks)

1. Create a second version of your k-NN approach that implements a distance weighted k-NN algorithm. Use the inverse distance squared as your distance weighted metric
    a. Does it outperform the normal k-NN?

Consider the possibility that two points might have 0 distance

Marking Scheme

Program Correctness: Your program should work correctly on all inputs (including new datasets). Also if there are any specifications about how the program should be written, or how the output should appear, those specifications should be followed.

Readability: Variables functions should have meaningful names. Code should be organized into functions/methods where appropriate.
There should be an appropriate amount of white space so that the code is readable, and the indentation should be consistent.

Documentation: your code and functions/methods should be appropriately commented. However, not every line should be commented because that makes your code overly busy. Think carefully about where comments are added.

Code Elegance: There are many ways to write the same functionality into your code, and some of them are needlessly slow or complicated. For example, if you are repeating the same code, it should be inside creating a new method/function or for loop

Code efficiency: The implementation is logically well designed without inappropriate design choices (e.g., unnecessary loops).