Final Project Report - Scientific Computing and Algorithms in Python
By Itay Eliav

Special Queue

In the following code, I built an object called SpecialQueue which is a queue implemented by a linked list.

- **Classes**
  There are two classes used:
  The Node Class, which is the building block of the linked list, it is used to build the queue by linking Node objects to each other, while the first Node in the queue is called 'head'.
  The class has two arguements; one is data (which is set as Null by default whenever a Node is created, given that it hadn't been given any data. This usually happens when the head of the linked list is created which happens with the creation of the list object itself.) The second is the 'next' arguement. It's purpose being to act or point as the next Node in the list. If the node is the last or the only node in the list, 'next' will be set as Null.
  The second class is the SpecialQueue Class which is the implementation of the queue itself. This class and it's methods let the user check if the queue is empty, the size of the queue, add or remove (enqueue or dequeue) objects from the queue, and print the queue's data (data of each node~object). This class is not given any arguements when it is invoked but a Node called 'head' is created once an instance is created.

- **Methods**
  - **Enqueue:** Creates a new node with the data. Then checks if there is some node as the head (meaning, if there is someone in the queue). If not, it puts the new node as the head. If there is one, it passes through the nodes (checking if each node's 'next' is not null. When it finds a null pointer, it converts that pointer into the new node which was created by the method.

  - **Dequeue:** Dequeues the head of the list (the first one to have entered the queue or one that was left after a past dequeue). If there is no node in the queue, a relevant statement is printed. Otherwise, a statement which includes the data of the head node is printed (saying it is it's turn), and the method then defines again the head node being the 'next' pointer~node of that last head.

  - **Is_empty:** Checks if the head has data and returns a boolean variable. If it doesn't have data, it means the queue is empty and 'True' is returned, otherwise, 'False' is returned.

  - **Size:** Returns an integer declaring how many nodes exist in the list. It starts with a counter and adds one if the head has data. Then it proceeds to add one for each 'next' that is not defined as Null. This is done with a while loop which proceeds to the node defined as 'next' of the current node till there is no node defined ('.next' defined as Null), meaning that was the last node. The method then stops the loop and returns the counter.

  - **objectsToArray:** This method is used in the __str__ function in order to print the specialQueue object as an array to show the user the data of each node in the queue. It first checks if the head node has data (meaning that there are objects in the queue). If it doesn't, it returns a statement stating there are no objects in the list, otherwise, it appends the head and the nodes* defined as 'next' of each other with a while loop (stopping when there is no 'next' node). The array is returned containing the data of each node linked to the head of the specialQueue object.
    ***** Meaning the data in those nodes.

Machine Learning

In the following code I built a KNN Classifier using sklearn library, that gets an input of a CSV file with Titanic passengers data, including if they survived or not, and generates a classifier algorithm which could be used to determine if a person with certain aspects would survive the Titanic tragedy.

- **Data-Sets**
  I used a data set that contains 892 records of Titanic passengers and certain aspects including: class of travel, sex, age, number of siblings / spouse abroad, number of parents / child abroad, fare, the port each passenger had embarked, and if they survived.
  The training set consists of 80% of the records, while the test set consists of 20% of the records.

- **Data Preparation**
  I had to normalize the categorical variables and assign them a number. This included the Sex of the passenger and the port which they embarked. Also, some of the Age data records were missing, so I assigned the mean age of the passengers that had the data. The choice was giving up some accuracy about the age but using the rest of the data of those records. Note: This could generate some inaccuracy regarding the results.

  The data was standardized using SKLearn's 'StandardScaler', which transforms the values and assigns them a number between -1 and 1, in order to reduce fluctuations and to make better use of the KNN algorithm.

- **Choosing K**
  The K-Nearest-Neighbors Classifier uses this number to determine which group the record tested should be classified to. This has to be an odd number in order to avoid being in a situation where the point's distance is close to the two groups equally. This would deny the possibility of classifying the record to either of them.

  Likewise, we *can't* choose K as the data length of the set because that would make the algorithm classify the record as the majority of the set, which would be pointless.

  In this model, I first chose K as the square-root of the length of the data set (and rounding to the nearest odd number). This is because choosing a K too low would make the result biased, based just on what's around it, which can give a skewed result. On the other hand, choosing a K too big, in general could cause processing and resource issues, and could, again, be biased by the majority of the group, as we discussed earlier.

  After that I varied and chose higher and lower K's, finding that the best results were given between K's of 5-9, the optimal being 7 with an accuracy of 83.80%.
  I also experimented with different distance metrics but didn't find any significant difference. The 'Euclidean' metric was chosen as a known good heuristic.
  Note: It could be the case that the square root of the data length, K=27, is better from a statistics perspective (for example: a case where the data was obtained via a method that would make it biased. This would make the algorithm good on this specific test set because it was split apart from the same sample dataset. Maybe a dataset obtained by a different method or company would give other results if tested on.

- **Results and conclusion**
  I evaluated my results using a confusion matrix, which helped me see the whole picture, emphasizing true-positives and true-negatives (the main diagonal). This showed me the number of records the algorithm guessed right.

  The algorithm using K=27 (square-root of the data length) did fairly well, classifying 95 out of 110 people who did not survive ‘as survived’, and 48 out of 69 who survived as ‘survived’. The accuracy score of the model was 79.89%.

  But trying different values for K I found out that 7 was the optimal one, giving 101 true positives out of 110, and 49 true negatives out of 69. The accuracy score this time was 83.80%.