Instructions: (Please read carefully and follow them!)

Try to solve all problems on your own. If you have difficulties, ask the instructor or TAs.

In this session, we will discuss Clustering algorithms for supervised and unsupervised learning.

The implementation of the optimization algorithms in this lab will involve extensive use of the `numpy` Python package. It would be useful for you to get to know some of the functionalities of `numpy` package. For details on `numpy` Python package, please consult `https://numpy.org/doc/stable/index.html`

For plotting purposes, please use `matplotlib.pyplot` package. You can find examples in the site `https://matplotlib.org/examples/`.

Please follow the instructions given below to prepare your solution notebooks:

- Please use different notebooks for solving different Exercise problems.
- The notebook name for Exercise 1 should be `YOURROLLNUMBER_IE684_Lab9_Ex1.ipynb`.
- Similarly, the notebook name for Exercise 2 should be `YOURROLLNUMBER_IE684_Lab9_Ex2.ipynb`, etc.

There are only 3 exercises in this lab. Try to solve all problems on your own. If you have difficulties, ask the Instructors or TAs.

Only the questions marked [**R**] need to be answered in the notebook. You can either print the answers using `print` command in your code or you can write the text in a separate text tab. To add text in your notebook, click `+Text`. Some questions require you to provide proper explanations; for such questions, write proper explanations in a text tab. Some questions require the answers to be written in LaTeX notation. Please see the demo video (posted in Lab 1) to know how to write LaTeX in Google notebooks. Some questions require plotting certain graphs. Please make sure that the plots are present in the submitted notebooks. Please include all answers in your `.pynb` files.

After completing this lab's exercises, click File → Download .ipynb and save your files to your local laptop/desktop. Create a folder with name `YOURROLLNUMBER_IE684_Lab9` and copy your `.ipynb` files to the folder. Then zip the folder to create `YOURROLLNUMBER_IE684_Lab9.zip`. Then upload only the `.zip` file to Moodle. There will be extra marks for students who follow the proper naming conventions in their submissions.

Please check the **submission deadline announced in moodle.**

Exercise 1: Knapsack Problem

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

$$\begin{aligned} \text{maximizes} \sum_{i \in T} v_i, \\ \text{subject to} \quad \sum_{i \in T} w_i \leq W. \end{aligned}$$

$W > 0$, $T \subseteq \{1, 2, \ldots, n\}$

Description of dataset: First line contains two integers N and K, where K is the maximum knapsack size and N is the number of items. From second line onwards, first and second column represent values(v_i) and weight(w_i) of that item respectively.

1. Download knapsack.txt uploaded in moodle.
2. **[R]** Try to find an optimal solution for the above problem.
3. **[R]** Find the optimal subset of items to include in knapsack for this problem.
4. **[R]** Report the solution and note the solution time.
5. **[R]** If it seems that the time to solve the problem is more, try to solve the problem in lesser time complexity and report the solution and time. (Hint- dynamic programming)

Exercise 2: The Subset Sum Problem

You are given a list of N positive integers, A = a[1], a[2], ..., a[N] and another integer S. You have to find whether there exists a non-empty subset of A whose sum is greater than or equal to S. You have to print the size of minimal subset whose sum is greater than or equal to S. If there exists no such subset then print -1 instead.

1. Given target sum(N) = 1489 and set(A) of integers: [381, 76, 55, 11, 329, 178, 110, 420, 17, 225, 139, 315, 455, 292, 189, 184, 294, 45, 196, 125]
2. **[R]** Solve the Subset Sum problem in the following ways:
   - Greedy
   - Dynamic Programming
   - Simulated Anealing
3. **[R]** Test the approximation algorithms(greedy and simulated anealing) to see how good of an approximation they give.
4. **[R]** Test and report the speed of all algorithms.

Exercise 3: Simulated Annealing
We will now use simulated annealing which is a probabilistic technique for approximating the global optimum of a given function. Our aim is to solve a Travelling Salesman Problem (TSP) using this technique.

Algorithm : Simulated annealing

```
1. Generate a random initial solution and set $T_0$
2. Calculate its cost using cost() function
3. For k = 0 through $max_{iter}$,
    a: Generate a random neighbouring state of the current solution using $neighbour()$ function and calculate it's cost.
    b: $T_k$ = updated temperature given by cooling schedule.
    c: If $c_{new} \leq c_{old}$, move to new solution, else move to new solution with some probability given by accept prob() function
    d: In each iteration store the best solution you got till that time
4. Return the best solution
```

---

1. **(R)**Describe travelling salesman problem
2. We will consider a small instance of the problem with 11 cities. The distance between any two cities are given in TSP11.csv. Try numpy.loadtxt(open("filename", "rb"), delimiter=",") to read the csv file.
3. The pseudo-code for simulated annealing given above. We start with a initial random state. A state is given by a particular sequence of cities which the salesman travels. This is known as a "tour" of the salesman. Note that the salesman should travel all the cities only once.
4. **(R)**Given a tour of the salesman compute the cost, which is total distance travelled in that tour. Create cost() function which takes a sequence of cities as input and calculates the total distance travelled. For example in a 4 city scenario, given the sequence of cities as [B,C,D,A], your function should calculate the total distance travelled in that particular order.
5. **(R)** Now we will create the neighbour() function. In TSP, neighbour of a state is defined as the new states obtained by swapping two consecutive cities. For example, in 4 city case, every state has 4 neighbours and neighbour of [A,B,C,D] are [B,A,C,D], [A,C,B,D], [A,B,D,C], [D,B,C,A]. Your function should choose one of the neighbours uniformly.
6. The acceptance probability is given by $e^{-(c_0-c)}/\ T_k$ where c_0 and c are the cost of new state and the old state. T_k is the current temperature of the k^{th} iterate. Explain the behaviour of acceptance probability when (1) new cost gets more worse than the current one and (2) temperature decreases.
7. Usually, the temperature is started at high value and slowly decreased to 0 in each iteration by a cooling schedule. Consider the cooling schedule given by $T_{k+1} = T_k$ for $\alpha \geq 0.80$ and $T_0 = 1$.
8. **(R)** Run the simulated annealing algorithm and report the solution. Plot the cost function for each iteration and explain. Choose different T0, values and observe the effect.
9. **(R)** Come up with another cooling schedule (not of the form $T_{k+1} = \alpha T_k$) and comment on the change in behaviour of the algorithm.
10. Do the same for the slightly larger problem with 48 cities. The distance matrix is given in TSP48.csv