Hong Kong Baptist University
Department of Computer Science
COMP7630 Web Intelligence and its Applications
Semester 2, 2024/25

Assignment 1 (Due March 17, 23:59)

The assignment is divided into three parts: (A) Text Vectorization, Classification & Clustering, (B) Evolutionary Algorithms, and (C) Association Rules Mining.

For part (A), you are required to answer the programming questions provided by submitting a Jupyter Notebook file, with comments that clearly indicate the question number that each piece of code addresses and explain the rationale for your approach to each question.

For parts (B) and (C), the questions provided must be answered by submitting a PDF document. Handwritten answers are accepted, but please make sure they are legible and then scan them into a PDF.

In summary, you will need to submit two files: a Jupyter Notebook for part (A) + a PDF for parts (B) and (C). Submit both files via the "Assignment" item on the Moodle course page by March 17.

SOLUTION FOR PARTS (B) AND (C)

(B) EVOLUTIONARY ALGORITHMS (30 MARKS)

8. Is it possible to improve the effectiveness of the K-Means algorithm by using an evolutionary algorithm? If so, explain how.

Solution for question 8

Yes, it is possible. Evolutionary Algorithms (EAs) are commonly used to tune hyperparameters of machine learning methods. In K-Means, important hyperparameters are the initial centroids of a clustering, therefore an EA can be applied to improve the clustering produced by running multiple K-Means with different initial centroids. To achieve this, the following components are required.

- Representation: The EA should encode solutions as real-valued vectors of size kn, where k is the number of clusters and n is the number of features (columns) in the input data matrix. Therefore, a solution is concatenation of the k initial centroids.
- Variation Operators: Appropriate crossover and mutation to work with continuous representation, or also specialized algorithms for continuous optimization such as DE and PSO.
- Fitness Function: The fitness function evaluates the quality of a solution by running K-Means starting from the centroids encoded by the solution and returning the inertia value outputted by K-Means. Since a lower inertia indicates a better clustering, the fitness function has to be minimized.

9. Considering both the uniform and single bit-flip mutation operators, calculate the probabilities that a binary solution formed by *n* bits will not be changed after applying mutation. Also describe how the two probabilities are calculated.

Solution for question 9

Single bit-flip mutation randomly selects one bit and flips it. Therefore, one bit is always flipped and the probability of not changing a solution is 0.

In uniform mutation, each bit in a bit-string is flipped with probability 1/*n*, where *n* is the length of the bit-string. Consequently, the probability that a given bit remains unchanged is 1 – 1/*n*. Since each bit undergoes mutation independently, the probability that the entire bit-string remains unchanged after uniform mutation is $\left(1-\frac{1}{n}\right)^n$.

10. Let *NoMutAritGA* be a genetic algorithm for continuous fitness functions. This algorithm uses the arithmetic crossover operator and does not incorporate a mutation operator. The selection and replacement operators can be chosen from any of those discussed in the lessons. Assume that, at iteration *t* of an execution, the population of *NoMutAritGA* consists of the following solutions:

    `x1=[1.5,2.8,3.3]` `x2=[1.5,2.8,3.3]`
    `x3=[0.2,1.1,2.2]` `x4=[0.3,0.1,2.2]`

    Without considering the specific fitness function under investigation, determine whether the solutions `A=[0.5,2.5,1.1]` and `B=[1.1,0.5,2.5]` can be visited by *NoMutAritGA* in the iterations *t*+*k*, for *k*≥1. Also motivate your answers.

Solution for question 10

Selection and replacement operators do not generate new solutions, but merely select a subset of them. The arithmetic crossover takes in input two parent vectors *x* and *y* and produces an offspring vector *z* as a convex combination of the parents using a randomly chosen coefficient $\alpha \in [0,1]$. Formally, $z = \alpha x + (1-\alpha)y$. A clear consequence is that *z* necessarily lies in the segment connecting *x* and *y* in the search space. In other words, each entry of the *z* vector is constrained to fall within the minimum and maximum values of the corresponding entries in *x* and *y*. Therefore, without any mutation operator, the region spanned by the population vectors is going to restrict iteration after iteration. This implies that solution `A` cannot be visited by *NoMutAritGA* because its third entry (1.1) falls outside the range [2.2,3.3], defined by the third entries of the population individuals `x1`, `x2`, `x3`, `x4`. This does not happen for solution B, that therefore has chances to be visited in the following iterations.

11. Let *f* and *g* be two simple objective functions, taking in input a bit-string of length 3 and returning a real number to be maximized, defined as follows:

```
def f(x): return 1 + 10*x[0] + 100*x[1] + 1000*x[2]
def g(x): return f(x)*f(x) + log(f(x)) + exp(f(x))
```

Also consider a genetic algorithm, called *MyGA*, which uses the following operators: tournament selection, uniform crossover, single bit-flip mutation, and plus replacement. Suppose that *MyGA* is run once for each objective function by initializing the random number generator with the same seed. Let indicate the two executions with *MyGA*(*f*,*seed*) and *MyGA*(*g*,*seed*). Is there any relationship between the solutions evaluated in the two executions? If so, explain what it is and try to discuss a more general property of *MyGA*. Moreover, what if we change the operators of *MyGA*?

Solution for question 11

It is easy to observe that the function *g* is a simple monotonic transformation of the function *f*. As a consequence, given two pairs of solutions *x* and *y* we have that:

- $f(x)=f(y)$ if and only if $g(x)=g(y)$;
- $f(x)<f(y)$ if and only if $g(x)<g(y)$;
- $f(x)>f(y)$ if and only if $g(x)>g(y)$.

Let also note that *MyGA* uses the objective/fitness values only to make comparisons between solutions (in tournament selection and plus replacement). Moreover, since the two executions *MyGA*(*f*,*seed*) and *MyGA*(*g*,*seed*) use the same seed for the random number generators, they have exactly the same starting population and the two executions generate exactly the same solutions in any iteration. As a simple corollary, the best solution found by both executions is exactly the same (though with a different objective/fitness value). More in general, *MyGA* has the same performance in optimizing any objective function which is a monotonic transformation of *f*. In other words, we can say that the *MyGA* is invariant for monotonic transformations of the objective function. This invariance property remains true also if we consider other selection or variation operators that does not use the magnitude of the values of the fitnesses. Practically, almost all the operators seen in class have this property. The only exception is the roulette wheel selection when no ranking transformation is adopted.

12. Let *h* be the objective function that takes as input a real vector from $[0,100]^4$ and returns a real number to be minimized, defined as follows:

```
def h(x):
    if x[0]==x[1]==x[2]==x[3]==0: return 0
    else: return 1000 - (x[0]+x[1]+x[2]+x[3])
```

Also consider the well-known *Differential Evolution* (DE) algorithm and a very trivial algorithm, called *Random Search* (RS), which iteratively generates random solutions, evaluates them, and returns the best one. Do you expect to observe a difference between RS and DE in the time (e.g., number of fitness evaluations) required to reach the global optimum of *h*? Also explain why and discuss whether one algorithm has some advantage over the other.

Solution for question 12

The function *h* is a deceptive function. Its optimum is at point (0,0,0,0) with value 0, but in the rest of the points the objective value improves moving away from (0,0,0,0) and approaching the other extreme point (100,100,100,100). This function cannot be optimized by a black-box algorithm so, by also considering that the search space is continuous (so there are infinite points), both RS and DE have no chance to reach the optimum. However, if reaching a good enough value is tolerable, DE can exploit the "gradient" of the function by climbing up towards the solution (100,100,100,100) whose objective value is 600 (the second best value among all solutions in $[0,100]^4$). RS cannot do it.

(C) Association Rules Mining (20 marks)

13. Table 1 shows the database of transactions of a music store.

Transaction ID	Items
T1	Guitar, GuitarPick
T2	DrumSet, Microphone, Amplifier
T3	DrumSet, GuitarPick, Guitar
T4	DrumSet, Microphone
T5	Guitar, Amplifier
T6	DrumSet, GuitarPick, Guitar

Table 1

a. Trace the results of using the *Apriori* algorithm, considering minimum support *minsup*=33% and minimum confidence *minconf*=60%, by showing both the candidate and frequent itemsets for each database scan.
b. Enumerate all the final frequent itemsets.
c. How many scans of the database of transactions are required by *Apriori* to calculate all the frequent itemsets (enumerated for the previous question)?
d. Indicate the association rules that are generated, along with their support and confidence, then highlight the valid ones.
e. List the valid association rules, sorted by lift and reporting their lift values.

Solution for question 13a

The support threshold is 33%, that in absolute terms means "at least 2 transactions", so the candidate and frequent itemsets generated by Apriori are as follows.

Scan (k)	Candidate itemsets and their (abs.) support	Frequent itemsets
k=1	DrumSet (4), Microphone (2), Amplifier (2), GuitarPick (3), Guitar (4)	DrumSet, Microphone, Amplifier, GuitarPick, Guitar
k=2	{DrumSet, Microphone} (2), ~~{DrumSet, Amplifier}~~ (1), {DrumSet, GuitarPick} (2), {DrumSet, Guitar} (2), ~~{Microphone, Amplifier}~~ (1), ~~{Microphone, GuitarPick}~~ (0), ~~{Microphone, Guitar}~~ (0), ~~{Amplifier, GuitarPick}~~ (0), ~~{Amplifier, Guitar}~~ (1), {GuitarPick, Guitar} (3)	{DrumSet, Microphone}, {DrumSet, GuitarPick}, {DrumSet, Guitar}, {GuitarPick, Guitar}
k=3	{Amplifier, GuitarPick, Guitar} (2)	{Amplifier, GuitarPick, Guitar}
k=4	Ø	

Solution for question 13b

{DrumSet}; {Microphone}; {Amplifier}; {GuitarPick}; {Guitar}; {DrumSet, Microphone}; {DrumSet, GuitarPick}; {DrumSet, Guitar}; {GuitarPick, Guitar}; {DrumSet, GuitarPick, Guitar}.

Solution for question 13c

Apriori, in step 1, requires one scan per iteration. Also note that, at each iteration *k*, the algorithm finds all the frequent itemsets of size *k* and it stops when the candidate set is empty. Therefore, Apriori requires 3 scans of the database of transactions in order to calculate all the frequent itemsets listed in the previous answer.

Solution for question 13d

The frequent itemsets of size 1 cannot generate association rules, so let list all the possible association rules for the frequent itemsets with size ≥ 2. We also calculate their confidence and highlight in bold those that are valid, i.e., those with confidence greater or equal to 0.6.

- {DrumSet,Micr.} generate: DrumSet -> Micr. (sup=2/6=0.33, conf=2/4=0.5)
  Micr. -> DrumSet (sup=2/6=0.33, conf=2/2=1)
- {DrumSet,GuitarPick} generate: DrumSet -> GuitarPick (sup=0.33, conf=0.5)
  GuitarPick -> DrumSet (sup=0.33, conf=0.66)
- {DrumSet,Guitar} generate: DrumSet -> Guitar (sup=0.33, conf=0.5)
  Guitar -> DrumSet (sup=0.33, conf=0.5)
- {GuitarPick, Guitar} generate: **GuitarPick -> Guitar (sup=0.5, conf=1)**
  Guitar -> GuitarPick (sup=0.5, conf=0.75)
- {DrumSet, GuitarPick, Guitar} generate:
  DrumSet->GuitarPick,Guitar (sup=0.33, conf=0.5)
  GuitarPick->DrumSet,Guitar (sup=0.33, conf=0.66)
  Guitar->DrumSet,GuitarPick (sup=0.33, conf=0.5)
  DrumSet,GuitarPick->Guitar (sup=0.33, conf=1)
  DrumSet,Guitar->GuitarPick (sup=0.33, conf=1)
  GuitarPick,Guitar->DrumSet (sup=0.33, conf=0.66)

Solution for question 13e

1. DrumSet, Guitar -> GuitarPick Lift=2
2. GuitarPick -> DrumSet, Guitar Lift=2
3. GuitarPick -> Guitar Lift=1.5
4. Guitar -> GuitarPick Lift=1.5
5. Microphone -> DrumSet Lift=1.5
6. GuitarPick, DrumSet -> Guitar Lift=1.5
7. GuitarPick -> DrumSet Lift=1
8. GuitarPick, Guitar -> DrumSet Lift=1

14. Suppose that each transaction has a numeric weight, represented by the amount of money paid. Design and discuss a strategy that allows the *Apriori* algorithm to take the weights into account, while keeping the internal mechanisms of the *Apriori* algorithm unchanged.

Solution for question 14

Since it is not possible to modify the internal mechanisms of Apriori, the only possibility is to modify its input. A good strategy to weigh a transaction is to replicate that transaction multiple times based on the weight value. For example, if the amount of money paid for a transaction is 3.4 dollars, we can round 3.4 to the closest integer, i.e., 3, then we consider 3 copies of that transaction in the database fed to Apriori.