1. EXPERIMENT NO: 2

2. TITLE: **CANDIDATE-ELIMINATION ALGORITHM**

3. LEARNING OBJECTIVES:

- Make use of Data sets in implementing the machine learning algorithms.
- Implement ML concepts and algorithms in Python

4. AIM:

- For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

5. THEORY:

- The key idea in the Candidate-Elimination algorithm is to output a description of the set of all hypotheses consistent with the training examples.
- It computes the description of this set without explicitly enumerating all of its members.
- This is accomplished by using the more-general-than partial ordering and maintaining a compact representation of the set of consistent hypotheses.
- The algorithm represents the set of all hypotheses consistent with the observed training examples. This subset of all hypotheses is called the version space with respect to the hypothesis space H and the training examples D, because it contains all plausible versions of the target concept.
- A version space can be represented with its general and specific boundary sets.
- The Candidate-Elimination algorithm represents the version space by storing only its most general members G and its most specific members S.
- Given only these two sets S and G, it is possible to enumerate all members of a version space by generating hypotheses that lie between these two sets in general-to-specific partial ordering over hypotheses. Every member of the version space lies between these boundaries

<u>Algorithm</u>

1. Initialize G to the set of maximally general hypotheses in H
2. Initialize S to the set of maximally specific hypotheses in H
3. For each training example d, do
    3.1. If d is a positive example
        Remove from G any hypothesis inconsistent with d ,
        For each hypothesis s in S that is not consistent with d ,
            Remove s from S
            Add to S all minimal generalizations h of s such that h is consistent with d, and some member of G is more general than h
            Remove from S, hypothesis that is more general than another hypothesis in S
    3.2. If d is a negative example
        Remove from S any hypothesis inconsistent with d
        For each hypothesis g in G that is not consistent with d
            Remove g from G
            Add to G all minimal specializations h of g such that h is consistent with d, and some member of S is more specific than h
        Remove from G any hypothesis that is less general than another hypothesis in G

6. PROCEDURE / PROGRAMME :

```
import csv

def get_domains(examples):
    d = [set() for i in examples[0]]
    for x in examples:
        for i, xi in enumerate(x):
            d[i].add(xi)
    return [list(sorted(x)) for x in d]

def more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        mg = x == "?" or (x != "0" and (x == y or y == "0"))
        more_general_parts.append(mg)
    return all(more_general_parts)

def fulfills(example, hypothesis):
    # the implementation is the same as for hypotheses:
    return more_general(hypothesis, example)

def min_generalizations(h, x):
    h_new = list(h)
    for i in range(len(h)):
        if not fulfills(x[i:i+1], h[i:i+1]):
            h_new[i] = '?' if h[i] != '0' else x[i]
    return [tuple(h_new)]

def min_specializations(h, domains, x):
    results = []
    for i in range(len(h)):
        if h[i] == "?":
            for val in domains[i]:
                if x[i] != val:
                    h_new = h[:i] + (val,) + h[i+1:]
                    results.append(h_new)
        elif h[i] != "0":
            h_new = h[:i] + ('0',) + h[i+1:]
            results.append(h_new)
    return results

def generalize_S(x, G, S):
    S_prev = list(S)
    for s in S_prev:
        if s not in S:
            continue
        if not fulfills(x, s):
            S.remove(s)
            Splus = min_generalizations(s, x)
            ## keep only generalizations that have a counterpart in G
            S.update([h for h in Splus if any([more_general(g,h)
                                                for g in G])])
            ## remove hypotheses less specific than any other in S
            S.difference_update([h for h in S if
                                 any([more_general(h, h1)
                                     for h1 in S if h != h1])])
    return S
```

```
def specialize_G(x, domains, G, S):
    G_prev = list(G)
    for g in G_prev:
        if g not in G:
            continue
        if fulfills(x, g):
            G.remove(g)
            Gminus = min_specializations(g, domains, x)
            ## keep only specializations that have a conuterpart in S
            G.update([h for h in Gminus if any([more_general(h, s)
                                                for s in S])])
            ## remove hypotheses less general than any other in G
            G.difference_update([h for h in G if
                                 any([more_general(g1, h)
                                      for g1 in G if h != g1])])
    return G

def candidate_elimination(examples):
    domains = get_domains(examples)[:-1]
    n = len(domains)
    G = set([("?",)*n])
    S = set([("0",)*n])

    print("Maximally specific hypotheses - S ")
    print("Maximally general hypotheses  - G ")

    i=0
    print("\nS[0]:",str(S),"\nG[0]:",str(G))
    for xcx in examples:
        i=i+1
        x, cx = xcx[:-1], xcx[-1]  # Splitting data into attributes and decisions
        if cx=='Y': # x is positive example
            G = {g for g in G if fulfills(x, g)}
            S = generalize_S(x, G, S)
        else: # x is negative example
            S = {s for s in S if not fulfills(x, s)}
            G = specialize_G(x, domains, G, S)
        print("\nS[{0}]:".format(i),S)
        print("G[{0}]:".format(i),G)
    return

with open('data22_sports.csv')  as csvFile:
        examples = [tuple(line) for line in csv.reader(csvFile)]

candidate_elimination(examples)
```

7. RESULTS & CONCLUSIONS:

Result-1

Data: data21_sports.csv (Sky,AirTemp,Humidity,Wind,Water,Forecast,EnjoySport)

```
sunny,warm,normal,strong,warm,same,Y
sunny,warm,high,strong,warm,same,Y
rainy,cold,high,strong,warm,change,N
sunny,warm,high,strong,cool,change,Y
```

Output

```
Maximally specific hypotheses - S
Maximally general hypotheses  - G
```

S[0]: {('0', '0', '0', '0', '0', '0')}
G[0]: {('?', '?', '?', '?', '?', '?')}

S[1]: {('sunny', 'warm', 'normal', 'strong', 'warm', 'same')}
G[1]: {('?', '?', '?', '?', '?', '?')}

S[2]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}
G[2]: {('?', '?', '?', '?', '?', '?')}

S[3]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}
G[3]: {('?', 'warm', '?', '?', '?', '?'), ('sunny', '?', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'same')}

S[4]: {('sunny', 'warm', '?', 'strong', '?', '?')}
G[4]: {('?', 'warm', '?', '?', '?', '?'), ('sunny', '?', '?', '?', '?', '?')}

Result-2

Data: data22_shape.csv (Size,Color,Shape,Label)
big,red,circle,N
small,red,triangle,N
small,red,circle,Y
big,blue,circle,N
small,blue,circle,Y

Output
Maximally specific hypotheses - S
Maximally general hypotheses - G

S[0]: {('0', '0', '0')}
G[0]: {('?', '?', '?')}

S[1]: {('0', '0', '0')}
G[1]: {('?', '?', 'triangle'), ('?', 'blue', '?'), ('small', '?', '?')}

S[2]: {('0', '0', '0')}
G[2]: {('big', '?', 'triangle'), ('small', '?', 'circle'), ('?', 'blue', '?')}

S[3]: {('small', 'red', 'circle')}
G[3]: {('small', '?', 'circle')}

S[4]: {('small', 'red', 'circle')}
G[4]: {('small', '?', 'circle')}

S[5]: {('small', '?', 'circle')}
G[5]: {('small', '?', 'circle')}

8. LEARNING OUTCOMES :

- The students will be able to apply candidate elimination algorithm and output a description of the set of all hypotheses consistent with the training examples

9. APPLICATION AREAS:

- Classification based problems.

10. REMARKS:

-