1. 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.

***CANDIDATE-ELIMINATION Learning Algorithm***

The CANDIDATE-ELIMINTION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

Initialize G to the set of maximally general hypotheses in H

Initialize S to the set of maximally specific hypotheses in H

For each training example d, do

* 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 any hypothesis that is more general than another hypothesis in S

* 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

CANDIDATE- ELIMINTION algorithm using version spaces

***Training Examples:***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Example** | **Sky** | **AirTemp** | **Humidity** | **Wind** | **Water** | **Forecast** | **EnjoySport** |
| **1** | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| **2** | Sunny | Warm | High | Strong | Warm | Same | Yes |
| **3** | Rainy | Cold | High | Strong | Warm | Change | No |
| **4** | Sunny | Warm | High | Strong | Cool | Change | Yes |

***Program:***

import csv

def g\_0(n):

return ("?",)\*n

def s\_0(n):

return ('0',)\*n

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

with open('enjoysport.csv') as csvFile:

examples = [tuple(line) for line in csv.reader(csvFile)]

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]

get\_domains(examples)

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]

G = set([g\_0(len(domains))])

S = set([s\_0(len(domains))])

i=0

print("\n G[{0}]:".format(i),G)

print("\n S[{0}]:".format(i),S)

for instance in examples:

i=i+1

x, label = instance[:-1], instance[-1] # Splitting data into attributes and decisions

if label=='yes': # 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("\n G[{0}]:".format(i),G)

print("\n S[{0}]:".format(i),S)

return

candidate\_elimination(examples)

**Data Set:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| sunny | warm | normal | strong | warm | same | yes |
| sunny | warm | high | strong | warm | same | yes |
| rainy | cold | high | strong | warm | change | no |
| sunny | warm | high | strong | cool | change | yes |

**Output:**

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

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

G[1]: {('?', '?', '?', '?', '?', '?')}

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

G[2]: {('?', '?', '?', '?', '?', '?')}

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

G[3]: {('sunny', '?', '?', '?', '?', '?'),

('?', 'warm', '?', '?', '?', '?'),

('?', '?', '?', '?', '?', 'same')}

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

G[4]: {('sunny', '?', '?', '?', '?', '?'),

('?', 'warm', '?', '?', '?', '?')}

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

**Separated output (Not the output)(Don’t write this)(only for reference)**

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

G[1]: {('?', '?', '?', '?', '?', '?')}

G[2]: {('?', '?', '?', '?', '?', '?')}

G[3]: **{('sunny', '?', '?', '?', '?', '?'),**

**('?', 'warm', '?', '?', '?', '?'),**

**('?', '?', '?', '?', '?', 'same')}**

G[4]: {('sunny', '?', '?', '?', '?', '?'),

('?', 'warm', '?', '?', '?', '?')}

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

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

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

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

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