Problem 2: 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.

Refrences :

https://www.uni-weimar.de/fileadmin/user/fak/medien/professuren/Webis/teaching/ws15/machine-learning/concept-learning.slides.html#/4/5

In [5]:

```
# Author : Dr.Thyagaraju G S , Context Innovations Lab , DEpt of CSE , SDMIT - Ujire
# Date : July 11 2018
import random
import csv
```

In [6]:

```
class Factors:

    factors={}
    attributes = ()

    def __init__(self,attr):
        self.attributes = attr
        for i in attr:              # Set of Attributes
            self.factors[i]=[]

    def add_values(self,factor,values):    #  Values of Each attributes
        self.factors[factor]=values
```

In [7]:

```
class Candidate_elimination:
    Positive={}
    Negative={}

    # Constructor
    def __init__(self,data,fact):
        self.num_factors = len(data[0][0])
        self.factors = fact.factors
        self.attr = fact.attributes
        self.dataset = data

   # Main Algorithm Method
    def run_algo(self):
        G = self.initializeG()
        S = self.initializeS()
        i=1
        for example in self.dataset: # For Each Training Example Data ,d

            if self.is_positive(example):# Positive Training Examples
                #Remove from G any hypothesis which is inconsistent with d
                G = self.remove_inconsistent_G(G,example[0])

                S_new = S[:]

                for s in S:
                    if not self.consistent(s,example[0]):
                        S_new.remove(s)
                        generalization = self.generalize_inconsistent_S(s,example[0])
                        generalization = self.get_general(generalization,G)
                        if generalization:
                            S_new.append(generalization)
                    S = S_new[:]
                    S = self.remove_more_general(S)
```

```python
                   # print("S+:\n",S)
                   #print("G+:\n",G)


            else: # Negative Training Examples
                    S = self.remove_inconsistent_S(S,example[0])
                    G_new = G[:]
                    for g in G:
                            if self.consistent(g,example[0]):
                                G_new.remove(g)
                                specializations = self.specialize_inconsistent_G(g,example[0])
                                specializationss = self.get_specific(specializations,S)
                                if specializations != []:
                                    G_new += specializations
                    G = G_new[:]
                    G = self.remove_more_specific(G)
                    #print("S-:\n",S)
                    #print("G-:\n",G)
            print("S[%d]:" %i,S,"\n")
            print("G[%d]:" %i,G,"\n")
            i=i+1

        #print ("Final S:",S)
        #print ("Final G:",G)


    def initializeS(self):
        ''' Initialize the specific boundary '''
        S = tuple(['0' for factor in range(self.num_factors)])
        return [S]


    def initializeG(self):
        ''' Initialize the general boundary '''
        G = tuple(['?' for factor in range(self.num_factors)])
        return [G]



    def is_positive(self,example):
        ''' Check if a given training example is positive '''
        if example[1] == 'Y':
            return True
        elif example[1] == 'N':
            return False
        else:
            raise TypeError("invalid target value")


    def is_negative(self,example):
        ''' Check if a given training example is negative '''
        if example[1] == 'N':
            return False
        elif example[1] == 'Y':
            return True
        else:
            raise TypeError("invalid target value")


    def match_factor(self,value1,value2):
        ''' Check for the factors values match,
            necessary while checking the consistency of
            training example with the hypothesis '''
        if value1 == '?' or value2 == '?':
            return True
        elif value1 == value2 :
            return True
        return False


    def consistent(self,hypothesis,instance):
        ''' Check whether the instance is part of the hypothesis '''
        for i,factor in enumerate(hypothesis):
            if not self.match_factor(factor,instance[i]):
                return False
        return True
```

```
def remove_inconsistent_G(self,hypotheses,instance):
    ''' For a positive example, the hypotheses in G
        inconsistent with it should be removed '''
    G_new = hypotheses[:]
    for g in hypotheses:
        if not self.consistent(g,instance):
            G_new.remove(g)
    return G_new


def remove_inconsistent_S(self,hypotheses,instance):
    ''' For a negative example, the hypotheses in S
        inconsistent with it should be removed '''
    S_new = hypotheses[:]
    for s in hypotheses:
        if self.consistent(s,instance):
            S_new.remove(s)
    return S_new

def remove_more_general(self,hypotheses):
    '''  After generalizing S for a positive example,
    the hypothesis in S general than others in S should
    be removed '''
    S_new = hypotheses[:]
    for old in hypotheses:
        for new in S_new:
            if old!=new and self.more_general(new,old):
                S_new.remove[new]
    return S_new

def remove_more_specific(self,hypotheses):
    ''' After specializing G for a negative example,
    the hypothesis in G
    specific than others in G should be removed '''
    G_new = hypotheses[:]
    for old in hypotheses:
        for new in G_new:
            if old!=new and self.more_specific(new,old):
                G_new.remove[new]
    return G_new


def generalize_inconsistent_S(self,hypothesis,instance):
    ''' When a inconsistent hypothesis for positive example
    is seen in the specific boundary S, it should be generalized
    to be consistent with the example ... we will get one hypothesis'''
    hypo = list(hypothesis) # convert tuple to list for mutability
    for i,factor in enumerate(hypo):
        if factor == '0':
            hypo[i] = instance[i]
        elif not self.match_factor(factor,instance[i]):
            hypo[i] = '?'
    generalization = tuple(hypo) # convert list back to tuple for immutability
    return generalization


def specialize_inconsistent_G(self,hypothesis,instance):
    ''' When a inconsistent hypothesis for negative example is
    seen in the general boundary G  should be
    specialized to be consistent with the example.. we will get a set of hypotheses '''
    specializations = []
    hypo = list(hypothesis) # convert tuple to list for mutability
    for i,factor in enumerate(hypo):
        if factor == '?':
            values = self.factors[self.attr[i]]
            for j in values:
                if instance[i] != j:
                    hyp=hypo[:]
                    hyp[i]=j
                    hyp=tuple(hyp) # convert list back to tuple for immutability
                    specializations.append(hyp)
    return specializations


def get_general(self,generalization,G):
    ''' Checks if there is more general hypothesis in G
```

```
            for a generalization of inconsistent hypothesis in S
            in case of positive example and returns valid generalization '''

        for g in G:
            if self.more_general(g,generalization):
                return generalization
        return None


    def get_specific(self,specializations,S):
        ''' Checks if there is more specific hypothesis in S
            for each of hypothesis in specializations of an
            inconsistent hypothesis in G in case of negative example
            and return the valid specializations'''
        valid_specializations = []
        for hypo in specializations:
            for s in S:
                if self.more_specific(s,hypo) or s==self.initializeS()[0]:
                    valid_specializations.append(hypo)
        return valid_specializations


    def exists_general(self,hypothesis,G):
        '''Used to check if there exists a more general hypothesis in
            general boundary for version space'''

        for g in G:
            if self.more_general(g,hypothesis):
                return True
        return False


    def exists_specific(self,hypothesis,S):
        '''Used to check if there exists a more specific hypothesis in
            general boundary for version space'''

        for s in S:
            if self.more_specific(s,hypothesis):
                return True
        return False


    def get_version_space(self,specific,general):
        ''' Given the specific and the general boundary of the
            version space, evaluate the version space in between '''
        while get_order(VS):
            for hypothesis in VS[:]:
                hypo = list(hypothesis) # convert tuple to list for mutability
                for i,factor in enumerate(hypo):
                    if factor != '?':
                        hyp=hypo[:]
                        hyp[i]='?'
                        if self.exists_general(hyp,general)and self.exists_specific(hyp,specific):
                            VS.append(tuple(hyp))
        return VS


    def get_order(self,hypothesis):
        pass


    def more_general(self,hyp1,hyp2):
        ''' Check whether hyp1 is more general than hyp2 '''
        hyp = zip(hyp1,hyp2)
        for i,j in hyp:
            if i == '?':
                continue
            elif j == '?':
                if i != '?':
                    return False
            elif i != j:
                return False
            else:                           # i==j
                continue
        return True
    def more_specific(self,hyp1,hyp2):
        ''' hyp1 more specific than hyp2 is
            equivalent to hyp2 being more general than hyp1 '''
        return self.more_general(hyp2,hyp1)

    '''
dataset=[(('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')]
  '''

with open('C:\\Users\\thyagaragu\\Desktop\\Data\\wsce.csv', 'r') as csvFile:
        dataset = [tuple([tuple(line[:-1]),''.join(line[-1:])]) for line in csv.reader(csvFile)]

attributes =('Sky','Temp','Humidity','Wind','Water','Forecast')


f = Factors(attributes)
f.add_values('Sky',('Sunny','Rainy'))
f.add_values('Temp',('Warm','Cold'))
f.add_values('Humidity',('Normal','High'))
f.add_values('Wind',('Strong','Weak'))
f.add_values('Water',('Warm','Cool'))
f.add_values('Forecast',('Same','Change'))

a = Candidate_elimination(dataset,f)
a.run_algo()
```

```
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]: [('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?'), ('?', '?', 'Normal',
'?', '?', '?'), ('?', '?', '?', 'Weak', '?', '?'), ('?', '?', '?', '?', 'Cool', '?'), ('?', '?',
'?', '?', '?', 'Same')]

S[4]: [('Sunny', 'Warm', '?', 'Strong', '?', '?')]

G[4]: [('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?'), ('?', '?', '?', '?',
'Cool', '?')]
```