```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 30 18:18:31 2017

@author: Falguni Das Shuvo
"""


import pandas as pd
import numpy as np

df = pd.read_excel("TrainingSet.xlsx", "Sheet1")

_class = "class"

def Create_Initial_Frequency_Table(data_frame, attribute):  #attribute = name of the
attribute which is to be discretized

    data_frame.sort_values(by = attribute, inplace = True)  # here without inplace
sort_values would just return a copy of data_frame with changes
                                                            #but the original one
remains unchanged
    unique_class_values = data_frame[_class].unique().tolist()
    frequency_table_column_names = unique_class_values[:]
    frequency_table_column_names.insert(0,attribute)
    frequency_table_column_names.append("Total")


    initial_frequency_table = pd.DataFrame(columns = frequency_table_column_names)

    #print initial_frequency_table

    distinct_attribute_values = data_frame[attribute].unique().tolist()   # already
sorted . See above.

    for val in distinct_attribute_values[:]:
        _tuple =  [val]
        total = 0
        for c_value in unique_class_values[:]:
            df = data_frame[data_frame[attribute] == val]
            temp = df[_class][df[_class] == c_value].count()
            total = total +  temp
            _tuple.append(temp)

        _tuple.append(total)
       # print _tuple

        initial_frequency_table = initial_frequency_table.append(pd.Series(_tuple,
```

```
index = frequency_table_column_names[:]), ignore_index = True)


    return initial_frequency_table


def Create_Contingency_Table_for_Observed_Values(frequency_table, lower_row_index):
    contingency_table = frequency_table [lower_row_index : (lower_row_index + 2)]

    column_names = frequency_table.columns.tolist()
    column_names.pop()
    column_names.append("row_sum")

    #contingency_table.rename(columns = column_names, inplace = True)

    contingency_table.columns = column_names

    column_names.pop(0)

    col_sum = ["column_sum"]

    for _name in column_names:
        col_sum.append(contingency_table[_name].sum())


    contingency_table = contingency_table.append(pd.Series(col_sum, index =
contingency_table.columns.tolist()), ignore_index = True)
    index_name = contingency_table.columns.tolist()[0]
    contingency_table.index = contingency_table[index_name]
    contingency_table.drop(index_name, axis = 1, inplace = True)

    return contingency_table


def Create_Contingency_Table_for_Expected_Values(observed_table):
    expected_table = observed_table.copy()

    column_names = observed_table.columns.tolist()
    column_names.pop()

    row_names = observed_table.index.tolist()
    row_names.pop()

    _sum = float(observed_table.loc['column_sum', 'row_sum'])

    for row in row_names:
        for col in column_names:
            expected_table.loc[row,col] = (observed_table.loc['column_sum', col] *
```

```
observed_table.loc[row, 'row_sum'] ) / _sum

    return expected_table


def Create_Contingency_Table_for_ChiSquare_Values(observed_table, expected_table):
    chi_square_table = observed_table.copy()

    column_names = observed_table.columns.tolist()
    column_names.pop()

    row_names = observed_table.index.tolist()
    row_names.pop()

    for row in row_names:
        for col in column_names:
            O = observed_table.loc[row,col]
            E = expected_table.loc[row,col]
            temp = (O - E) * (O - E)
            if E < 0.5 :
                denominator = 0.5
            else:
                denominator = E

            chi_square_table.loc[row,col] = temp / float(denominator)

    return chi_square_table

def Get_ChiSquare_Value(chi_square_table):
    column_names = chi_square_table.columns.tolist()
    column_names.pop()

    row_names = chi_square_table.index.tolist()
    row_names.pop()

    value = 0
    for row in row_names:
        for col in column_names:
            value = value + chi_square_table.loc[row,col]

    return value



def ChiMerge(attribute, threshold = 2.71, minIntervals = 2, maxIntervals = np.inf ):


    print "Result from ChiMerge algorithm for attribute = ", attribute , ",
```

```
threshold = ",
    print threshold, ", minIntervals = ", minIntervals, ", maxIntervals = ",
maxIntervals

    master_frequency_table = Create_Initial_Frequency_Table(df,attribute)

    print "Initial Frequency Table:"
    print master_frequency_table

    slave_frequency_table = master_frequency_table.copy()
    master_frequency_table['chi_square'] = None



    column_names = master_frequency_table.columns.tolist()
    column_names.pop()
    column_names.pop(0)


    index_list = slave_frequency_table.index.tolist()
    index_list.pop()

    while True:
        #chi_square_value = []
        print ""


        if master_frequency_table.shape[0] <= minIntervals : break

        for i in index_list[:]:
            observed_table =
Create_Contingency_Table_for_Observed_Values(slave_frequency_table,i)
            print "Contingency table with observed values:"
            print observed_table
            print ""
            expected_table =
Create_Contingency_Table_for_Expected_Values(observed_table)
            print "Contingency table with expected values:"
            print expected_table
            print ""
            chi_square_table =
Create_Contingency_Table_for_ChiSquare_Values(observed_table, expected_table)
            print "Contingency table with Chi square value:"
            print chi_square_table
            print ""
            #chi_square_value.append(Get_ChiSquare_Value(chi_square_table))
            master_frequency_table.loc[i,'chi_square'] =
Get_ChiSquare_Value(chi_square_table)
```

```
        print "Frequency Table with Chi Square (Before merging):"
        print master_frequency_table
        print ""

        smallest_chi_square_value = master_frequency_table['chi_square'].min()

        if smallest_chi_square_value >= threshold and
not(master_frequency_table.shape[0] > maxIntervals):
            break

        for x in master_frequency_table.index.tolist():
            if master_frequency_table['chi_square'][x] == smallest_chi_square_value:
                row_of_smallest_chi_square_value = x
                break


        for col_name in column_names[:]:
            master_frequency_table[col_name][row_of_smallest_chi_square_value] =
master_frequency_table[col_name][row_of_smallest_chi_square_value] +
master_frequency_table[col_name][row_of_smallest_chi_square_value + 1]

        master_frequency_table['chi_square'][row_of_smallest_chi_square_value] =
None

        master_frequency_table.drop((row_of_smallest_chi_square_value+1), axis = 0,
inplace = True)

        master_frequency_table.reset_index(drop = True, inplace = True)

        slave_frequency_table = master_frequency_table.drop('chi_square', axis = 1,
inplace = False)

        if row_of_smallest_chi_square_value == 0:
            index_list = [0]
        elif row_of_smallest_chi_square_value ==
master_frequency_table.last_valid_index():
            index_list = [row_of_smallest_chi_square_value-1]
        else:
            index_list = [row_of_smallest_chi_square_value-1,
row_of_smallest_chi_square_value]


        #print index_list


        #print row_of_smallest_chi_square_value
```

```
print "Frequency Table with Chi Square (After merging):"
print master_frequency_table
print ""
```