Chi-Merge Binning Algorithm

In [1]:

```
# Import usual libraries
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
import numpy as np

# Import required libraries for chi2 calculation
from scipy.stats import chi2_contingency,chisquare
from scipy.stats.contingency import expected_freq
import more_itertools as mit
```

In [ ]:

```

```

Part (A)

In [2]:

```
# Part(A) practical answers
# Generate the X, Y Features and the Class Labels
X = [i for i in np.arange(15) if i%2!=0]
Y = [i for i in np.arange(15) if (i!=0 | i%2==0)]
Class = ['A' if (i%2==0) else 'B' for i in np.arange(7)]

# Process to generate a DataFrame
data = np.array([X,Y,Class]).T
```

In [3]:

```
# Reproduce the table
table = pd.DataFrame.from_records(data, columns=['X','Y','Class']).astype({'X': 'int64'
, 'Y': 'int64', 'Class' : 'string'})
display(table.style.format({}).hide_index())
```

X	Y	Class
1	2	A
3	4	B
5	6	A
7	8	B
9	10	A
11	12	B
13	14	A

In [4]:

```
# Create the contingency table
# All calculations below show X, it is the same for Y except column labels
tab = pd.crosstab(table.Class, table.X, margins=True, margins_name='Total')
display(tab.style.format({}))
```

X	1	3	5	7	9	11	13	Total
Class								
A	1	0	1	0	1	0	1	4
B	0	1	0	1	0	1	0	3
Total	1	1	1	1	1	1	1	7

In [5]:

```
# Check if chi-score matches the manual calculations answered in the word document
# Observed frequency
O = pd.crosstab(table.Class.iloc[0:2], table.X.iloc[0:2], colnames='X')
O
```

Out[5]:

X	1	3
Class		
A	1	0
B	0	1

In [6]:

```
# Expected frequency
E = pd.DataFrame(expected_freq(O), columns=['1','3'], index=['A', 'B',]).rename_axis([
'Class']).rename_axis(columns='X')
E
```

Out[6]:

X	1	3
Class		
A	0.5	0.5
B	0.5	0.5

In [7]:

```
# Calculate chi-score for this, it should match the answer provided in the word documen
t (2)
chi2score, pvalue = chisquare(f_obs=O.values.ravel(), f_exp=expected_freq(O).ravel(), d
dof=2)
print("""Chi-Score: {:.3f}
P-Value  : {:.3f}""".format(chi2score, pvalue))
```

```
Chi-Score: 2.000
P-Value  : 0.157
```

In [8]:

```
# Chi-score for the whole class, this should match (7) given in the word document
# Remove totals from contingency table
O = tab.iloc[:2,0:-1].values
E = expected_freq(O)
chi2score, pvalue = chisquare(O.ravel(), E.ravel(), 7)
print("""Chi-Score: {:.3f}
P-Value  : {:.3f}""".format(chi2score, pvalue))

# Another way when degrees of freedom and expected frequency is not known
# chi2, p, df, E = chi2_contingency(O)
```

```
Chi-Score: 7.000
P-Value  : 0.321
```

In [ ]:

```

```

Part (B)

In [9]:

```
# Answers to Part(B) of the Assignment
# Iterative bottom-up approach to merge the intervals using Chi-Merge
```

In [10]:

```
# Chi2 for a given contingency matrix
def chiSquared(contingencyMatrix):
    return chisquare(contingencyMatrix, f_exp=expected_freq(contingencyMatrix))

# List and history of chi-squared values per pair for each iteration cycle
def chi2Calculator(obs):
    chi_list = []
    for i in range(obs.shape[0]-1):
        chi2, p_val = 0,0
        chi, p = chiSquared(np.vstack((obs[i],obs[i+1])))
        chi2 = np.nansum(chi)
        p_val = float(sum(np.nan_to_num(p)))
        chi_list.append(chi2)
    return chi_list

# Chimerge algorithm that calculates merge-points according to chi-score and discretize
s/bins the continuous input feature
def chiMerge(obs, f_arr, numOfIntervals=0, chiThreshold=0, mergeType=1):
    # Store the history of merged intervals
    merge_hist = []
    minChi2 = -1
    # Stop criteria is based on intervals, chi2 threshold is also provided for convenie
nce and can be modified
    # chiThreshold > minChi2
    while (obs.shape[0] > numOfIntervals):
        # Get the chi2 scores per adjacent interval
        chi_list=chi2Calculator(obs)
        minChi2 = min(chi_list)
        # Get the indices to be merged with minimum chi-values, there are 2 approaches
        ###### 1
        # This approach merges all intervals with equal minimum chi-score
        # This is faster and helpful in bulk-merge with lesser iterations
        # This can be used when stop criterion is based only on chi2 threshold
        ######
        if(mergeType==1):
            mergeIndex = [list(i) for i in mit.consecutive_groups([i for i, j in enumer
ate(chi_list) if j == minChi2])]
        ###### 2
        # This approach merges only the first interval with minimim chi-score
        # This approach will help when stop criterion has a condition only on number of
intervals
        # This will only merge interval by interval (1 interval per iteration)
        else:
            mergeIndex = [list(i) for i in mit.consecutive_groups([chi_list.index(minCh
i2)])]
        ######

        # Merge and squash intervals (interval binning)
        squashedArr = np.zeros(len(mergeIndex))
        noofelem = 0
        for j,k in enumerate(mergeIndex):
            k.append(k[-1] + 1)
            k = [i - noofelem for i in k]
            noofelem += max(k) - min(k)
            merged = list([f_arr[min(k)][0], f_arr[max(k)][0]])
            merge_hist.append(merged)
            squashedArr = np.sum(obs[k,:], axis=0)
            obs = np.delete(obs,k,0)
            obs = np.insert(obs, min(k), squashedArr, 0)
```

```
            f_arr = np.delete(f_arr,k,0)
            f_arr = np.insert(f_arr, min(k), merged[0], 0)
    return np.concatenate((f_arr,obs,),axis=1),chi2Calculator(obs)
```

In [11]:

```
# Read iris dataset and check that the indexes range from 0 - 149 to represent 150 rows
in iris
iris = pd.read_csv('iris.data', names=['s_len','s_wid','p_len','p_wid','Class'], header
=None)
display(iris.shape, iris.head(3), iris.tail(3))
# s_len = Sepal Length, s_wid = Sepal Width, p_len = Petal Length, p_wid = Petal Width,
Class = Class of Iris
```

(150, 5)

	s_len	s_wid	p_len	p_wid	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa

	s_len	s_wid	p_len	p_wid	Class
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

In [12]:

```
# Function to get the contingency table sorted ascending by the feature
def getContingencyTbl(df, feature):
    return pd.crosstab(df[feature], df.Class).reset_index().sort_values(by=feature).val
ues

# Get the binned intervals
def getBins(f_arr,endVal):
    binList = []
    for i in range(f_arr.shape[0] - 1):
        binList.append(str('[' + str(f_arr[i][0]) + ',' + str(f_arr[i+1][0]) + ')'))
    binList.append(str([f_arr[-1][0], endVal]))
    return binList

# Create a neat dataframe for display instead of raw numpy arrays and numbers
def binnedOutput(f_arr,chi2_arr,feature,cols,typeDict,formatDict,endVal,chi2_0):
    binList = getBins(f_arr,endVal)
    df = pd.DataFrame(f_arr, columns=cols)
    df.insert(0,'Bins',binList,True)
    df = df.astype(typeDict)
    df = df.set_index(pd.DataFrame(np.array(chi2_arr).reshape(1,-1)).astype('float').re
set_index().iloc[0].values.T)
    df = df.rename(index= {df.index[0] : chi2_0})
    df = df.rename_axis('Chi2').rename(index = lambda x : "{:.2f}".format(x))
    display(df.reset_index().style.format(formatDict).hide_index())
```

In [13]:

```
# Ignore runtime warnings
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)

# Calulate and output the discretized values for all features of the iris data-set
for feature in iris.columns:
    # Ignore discretization for the target label
    if feature=='Class':
        continue
    f = getContingencyTbl(iris, feature)

    # Perform discretization, set numOfIntervals (enforced) which is the number of bin
s/discretized intervals
    # Also define a chiThreshold where known, it is set (not enforced) for 95% independ
ence on 2 degrees of freedom
    f_arr, chi2_arr = chiMerge(f[:,1:], f[:,0:1], numOfIntervals=4, chiThreshold=5.99)
    cols = [feature, 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
    typeDict = {'Iris-setosa':'int64', 'Iris-versicolor':'int64', 'Iris-virginica':'int
64', 'Bins':'string'}
    formatDict = {feature:"{:.1f}"}

    # Display all the discretized features one by one
    binnedOutput(f_arr, chi2_arr, feature, cols, typeDict, formatDict, f[-1][0], chi2_c
ontingency(f[:,1:])[0])

# The first Chi2 will be 0 as we are not comparing the first row with anything
# For every subsequent row Chi2 is the value between the row in question and the previo
us row
# Note that it is replaced with the chi2 of the contingency matrix as a whole instead o
f 0
# Split points are neatly captured in the Bins column which also summarized the final i
ntervals

# Check the paper at the below link to see that the results exactly match
# https://sci2s.ugr.es/keel/pdf/algorithm/congreso/1992-Kerber-ChimErge-AAAI92.pdf
```

Chi2	Bins	s_len	Iris-setosa	Iris-versicolor	Iris-virginica
156.27	[4.3,5.5)	4.3	45	6	1
30.91	[5.5,5.8)	5.5	4	15	2
17.85	[5.8,7.1)	5.8	1	29	35
9.07	[7.1, 7.9]	7.1	0	0	12

Chi2	Bins	s_wid	Iris-setosa	Iris-versicolor	Iris-virginica
88.36	[2.0,2.5)	2.0	1	9	1
5.18	[2.5,3.0)	2.5	1	25	20
17.09	[3.0,3.4)	3.0	18	15	24
24.19	[3.4, 4.4]	3.4	30	1	5

Chi2	Bins	p_len	Iris-setosa	Iris-versicolor	Iris-virginica
271.80	[1.0,3.0)	1.0	50	0	0
95.00	[3.0,4.8)	3.0	0	44	1
37.34	[4.8,5.2)	4.8	0	6	15
10.90	[5.2, 6.9]	5.2	0	0	34

Chi2	Bins	p_wid	Iris-setosa	Iris-versicolor	Iris-virginica
271.75	[0.1,1.0)	0.1	50	0	0
78.00	[1.0,1.4)	1.0	0	28	0
5.93	[1.4,1.8)	1.4	0	21	5
48.36	[1.8, 2.5]	1.8	0	1	45

In [14]:

```
# Calculate with another data-set, one given in Part(A) as an example
for feature in table.columns:
    if feature=='Class':
        continue
    f = getContingencyTbl(table, feature)

    # Print for all combinations of intervals from 1-7
    for j in [i for i in reversed(range(8)) if i!=0]:
        # Perform discretization, set numOfIntervals which is the number of bins/discre
tized intervals
        # Also define mergeType as 2 since all intervals will be squished in one go oth
erwise
        f_arr, chi2_arr = chiMerge(f[:,1:], f[:,0:1], numOfIntervals=j, chiThreshold=1.
2, mergeType=2)
        cols = [feature,'A','B']
        typeDict = {'A':'int64', 'B':'int64', 'Bins':'string'}
        formatDict = {}
        binnedOutput(f_arr, chi2_arr, feature, cols, typeDict, formatDict, f[-1][0], ch
i2_contingency(f[:,1:])[0])
```

Chi2	Bins	X	A	B
7.00	[1,3)	1	1	0
2.00	[3,5)	3	0	1
2.00	[5,7)	5	1	0
2.00	[7,9)	7	0	1
2.00	[9,11)	9	1	0
2.00	[11,13)	11	0	1
2.00	[13, 13]	13	1	0

Chi2	Bins	X	A	B
7.00	[1,5)	1	1	1
0.75	[5,7)	5	1	0
2.00	[7,9)	7	0	1
2.00	[9,11)	9	1	0
2.00	[11,13)	11	0	1
2.00	[13, 13]	13	1	0

Chi2	Bins	X	A	B
7.00	[1,7)	1	2	1
1.33	[7,9)	7	0	1
2.00	[9,11)	9	1	0
2.00	[11,13)	11	0	1
2.00	[13, 13]	13	1	0

Chi2	Bins	X	A	B
7.00	[1,9)	1	2	2
0.83	[9,11)	9	1	0
2.00	[11,13)	11	0	1
2.00	[13, 13]	13	1	0

Chi2	Bins	X	A	B
7.00	[1,11)	1	3	2
1.20	[11,13)	11	0	1
2.00	[13, 13]	13	1	0

Chi2	Bins	X	A	B
7.00	[1,13)	1	3	3
0.88	[13, 13]	13	1	0

Chi2	Bins	X	A	B
7.00	[1, 13]	1	4	3

Chi2	Bins	Y	A	B
7.00	[2,4)	2	1	0
2.00	[4,6)	4	0	1
2.00	[6,8)	6	1	0
2.00	[8,10)	8	0	1
2.00	[10,12)	10	1	0
2.00	[12,14)	12	0	1
2.00	[14, 14]	14	1	0

Chi2	Bins	Y	A	B
7.00	[2,6)	2	1	1
0.75	[6,8)	6	1	0
2.00	[8,10)	8	0	1
2.00	[10,12)	10	1	0
2.00	[12,14)	12	0	1
2.00	[14, 14]	14	1	0

Chi2	Bins	Y	A	B
7.00	[2,8)	2	2	1
1.33	[8,10)	8	0	1
2.00	[10,12)	10	1	0
2.00	[12,14)	12	0	1
2.00	[14, 14]	14	1	0

Chi2	Bins	Y	A	B
7.00	[2,10)	2	2	2
0.83	[10,12)	10	1	0
2.00	[12,14)	12	0	1
2.00	[14, 14]	14	1	0

Chi2	Bins	Y	A	B
7.00	[2,12)	2	3	2
1.20	[12,14)	12	0	1
2.00	[14, 14]	14	1	0

Chi2	Bins	Y	A	B
7.00	[2,14)	2	3	3
0.88	[14, 14]	14	1	0

Chi2	Bins	Y	A	B
7.00	[2, 14]	2	4	3