James Gray
graymatter@u.northwestern.edu

Introduction

The purpose of this assignment is to inform a marketing plan that will target future visitors to one of the Wisconsin Dells seven attractions referenced in the Harrington (2007) case study. This study will outline management recommendations to drive more visitors to "The Ducks" tour using an unsupervised learning model based on associative rule mining. The input data is a 1,698 in-person survey conducted at various locations and attractions across the Wisconsin Dells area.

Results

The model uses the "Apriori" algorithm developed by Peter Harrington as outlined in the book Machine Learning in Action (2012). This algorithm reduces the computation cost of candidate itemsets and calculates inference probabilities between an itemset X that correlates with an itemset Y. The first part of the study analyzed correlations between the 33 activities to identify potential marketing strategies to target visitors at activity X with a likelihood to also visit activity Y. Two critical algorithm variables were manipulated to increase the likelihood that the management recommendations would deliver the expected business results. The support algorithm variable (alpha) was set high (0.50) to identify frequent data sets and the confidence was also set higher than chance (0.60) to only identify rules with a strong correlation that a visitor attending activity X is also highly likely to attend activity Y. The model produced a strong market basket correlation between riding the Duck Tour and shopping. A shopping visitor was 62.1% likely to ride the Duck Tour. Conversely, a visitor to the Duck Tour was 87% likely to shop. The model did not identify association rules between other activities beyond shopping. Using this insight, recommendations to the Ducks marketing plan include:

- Print ads in shopping center guide books
- Ad pamphlets placed next shopping center advertisements in locations such as hotel lobbies
- Discount coupons left on cars outside shopping center establishments

- Co-marketing between shopping centers or specific retailers to offer slight Duck Tour discounts on point-of-sale receipts. Given the strong correlation (86%) that Duck Tour visitors also shop, there could also be advertisements on the boat, pamphlets or brochures that also offer discounts at those specific retailers. This strategy would be mutually beneficial.
- Online ads placed on shopping center websites and paid search ads for users searching for shopping information.
- Geo-based mobile ads that target users who are in shopping locations.

The second part of the analysis evaluated visitor demographics to determine if there were any strong correlations of these attributes to visitors that attended the Duck Tour. The categorical demographic data were binarized and the model calculated associations between the following variables:

- There is a 59.94% probability taking the Duck Tour when there are 2 adult visitors
- There is 61.5% probability taking the Duck Tour when visitors plan more than one in advance.

Using these inferences the marketing plan could target parents or couples that plan well in advance. This could include promotions with local hotels that want bookings well in advance by offering small discounts to a Duck Tour. The Duck Tour could also offer online discounts for customers who are willing to purchase tickets more than one month in advance.

Code

```
# CIS435 Assignment 2: James Gray - graymatter@u.northwestern.edu - July 19, 2014
# analyzing the Wisconsin Dells data using the
# apriori program from Peter Harrington (2012)

# Harrington, P. (2012). Machine learning in action. Shelter Island, N.Y.: Manning.
# open-source code from: www.manning.com/MachineLearninginAction (see Chapter 11)

# let's make our programs compatible with Python 3.0/1/2/3
from __future__ import division, print_function
from future_builtins import ascii, filter, hex, map, oct, zip

import numpy as np # array operations
import pandas as pd # pandas for data frame operations

# set maximum number of rows to twenty
pd.options.display.max_rows = 20

# set parameters for apriori algorithm
alpha = 0.5 # required level of support for apriori item sets
beta = 0.6 # required level of confidence for apriori rules
max_other_items = 2 # maximum other items in reported rule set
# frozen set is an immutable set
item_of_interest = frozenset(['rideducks']) # character string of item of interest

from mlia import apriori # this brings in a subset of methods from apriori

# ------------------------------------------------
# Generate list of lists from Wisconsin Dells data
# ------------------------------------------------
# read in the Wisconsin Dells data
wi_dells = pd.read_csv('wisconsin_dells.csv')
print(wi_dells.shape) # check the structure of the data frame
print (wi_dells)

# list of activities for study
activity = ['shopping', 'antiquing', 'scenery', 'eatfine',
'eatcasual', 'eatfamstyle', 'eatfastfood', 'museums',
'indoorpool', 'outdoorpool', 'hiking', 'gambling',
'boatswim', 'fishing', 'golfing', 'boattours', 'rideducks',
'amusepark', 'minigolf', 'gocarting', 'waterpark',
'circusworld', 'tbskishow', 'helicopter', 'horseride',
'standrock', 'outattract', 'nearbyattract', 'movietheater',
'concerttheater', 'barpubdance', 'shopbroadway', 'bungeejumping']
# work with a subset of the data columns corresponding to the activities
df = pd.DataFrame(wi_dells, columns = activity) # reduces DF to the 33 activities
print(df.shape) # check the structure of the data frame
```

```
print(df.head) # examine the first few lines of the data frame

# YES activity cells of the data frame replaced by name of activity
# NO coded as missing data using numpy nan method
for index in range(len(activity)):
activity_to_name = {'NO' : np.nan, 'YES' : activity[index]}
df[activity[index]] = df[activity[index]].map(activity_to_name)
# DF now has activity in columns for each survey row with NaN where no activity

print(df.head) # examine the first few lines of the recoded data frame

# my_data is the list of lists structure that is needed
# for input to the apriori algorithm
my_data = list() # initialize list structure for list of lists

# loop through the rows of the data frame by index ix
# creating the list of lists structure one activity basket at a time
# we use dropna() to omit missing data as coded nan from numpy

# generate the activity basket for this visitor
# and convert it to a list for each of the df.shape[0] visitors
for index in range(df.shape[0]):
basket = list(df.ix[index].dropna()) # this list for one visitor
my_data.append(basket) # add this list to list of lists

# ------------------------------------------------
# Apply the apriori algorithm to the list of lists
# ------------------------------------------------
L, suppData = apriori.apriori(my_data, alpha) # returns list of frozen sets and Support

print('Identified rules with support = ', alpha, 'and confidence = ', beta)
rules = apriori.generateRules(L, suppData, minConf = beta)

# search across the rule set starting with smaller sets
# and moving to larger and larger sets until no rules
# exist that satisfy the requirement of including item_of_interest

n_other_items = 1 # initialize reporting at one other item
while n_other_items <= max_other_items:
print('\nRules with ', n_other_items, ' other item(s)')
for item in L[n_other_items]:
if item.intersection(item_of_interest): print(item)
n_other_items = n_other_items + 1
# ------------------------------------------------------------------------
# The code above identified associations between the activities
# Binarize the categorical data to determine association rules for variables
# that may show correlation to a specific activity ("rideducks")
# the are 9 input nominal variables that need to be binarized and created as
```

```
# new columns. This is similar to the technique outlined in Introduction to
# Data Mining page 416.
# nnights: nights_0,nights_1,nights_2,nights_3,nights_4
# nadults: adults_1, adults_2, adults_3, adults_4, adults_5
# nchildren: child_0, child_1, child_2, child_3, child_4, child_5
# planning: planning_thismonth, planning_morethanmonth, planning_thisweek
# sex: male, female
# age: ageLT25, age25to34, age35to44, age45to54, age55to64, age65plus
# education: ed_HSGrad, ed_PostGrad, ed_CollegeGrad, ed_SomeCollege
# income: income_low, income_mid, income_upper
# region: reg_Chicago, reg_Madison, reg_Milw, reg_Minn, reg_OtherWis, reg_Other
# this will create 39 new columns
# ------------------------------------------------------------------------

# create a new dataframe to hold binarized inputs and 'rideducks' activity
dfbin = pd.DataFrame(wi_dells, columns = ["rideducks"])

# map rideducks column to 0 or 1 for consistency with binarization below
activity_mapper = {'NO': 0, 'YES': 1}
dfbin['rideducks'] = dfbin['rideducks'].map(activity_mapper)

#binarize nnights
nights0_to_binary = {'0':1,'1':0,'2':0,'3':0,'4+':0}
nights1_to_binary = {'0':0,'1':1,'2':0,'3':0,'4+':0}
nights2_to_binary = {'0':0,'1':0,'2':1,'3':0,'4+':0}
nights3_to_binary = {'0':0,'1':0,'2':0,'3':1,'4+':0}
nights4_to_binary = {'0':0,'1':0,'2':0,'3':0,'4+':1}
dfbin['nights_0'] = wi_dells['nnights'].map(nights0_to_binary) #create new column
dfbin['nights_1'] = wi_dells['nnights'].map(nights1_to_binary)
dfbin['nights_2'] = wi_dells['nnights'].map(nights2_to_binary)
dfbin['nights_3'] = wi_dells['nnights'].map(nights3_to_binary)
dfbin['nights_4'] = wi_dells['nnights'].map(nights4_to_binary)

# binarize adults
adults1_to_binary = {'1':1,'2':0,'3':0,'4':0,'5+':0}
adults2_to_binary = {'1':0,'2':1,'3':0,'4':0,'5+':0}
adults3_to_binary = {'1':0,'2':0,'3':1,'4':0,'5+':0}
adults4_to_binary = {'1':0,'2':0,'3':0,'4':1,'5+':0}
adults5_to_binary = {'1':0,'2':0,'3':0,'4':0,'5+':1}
dfbin['adults_1'] = wi_dells['nadults'].map(adults1_to_binary) #create new column
dfbin['adults_2'] = wi_dells['nadults'].map(adults2_to_binary)
dfbin['adults_3'] = wi_dells['nadults'].map(adults3_to_binary)
dfbin['adults_4'] = wi_dells['nadults'].map(adults4_to_binary)
dfbin['adults_5'] = wi_dells['nadults'].map(adults5_to_binary)

# binarize nchildren
child0_to_binary = {'No kids':1, '1':'0','2':'0','3':'0','4':'0','5+':'0'}
child1_to_binary = {'No kids':0, '1':'1','2':'0','3':'0','4':'0','5+':'0'}
```

```
child2_to_binary = {'No kids':0, '1':'0','2':'1','3':'0','4':'0','5+':'0'}
child3_to_binary = {'No kids':0, '1':'0','2':'0','3':'1','4':'0','5+':'0'}
child4_to_binary = {'No kids':0, '1':'0','2':'0','3':'0','4':'1','5+':'0'}
child5_to_binary = {'No kids':0, '1':'0','2':'0','3':'0','4':'0','5+':'1'}
dfbin['child_0'] = wi_dells['nchildren'].map(child0_to_binary) #create new column
dfbin['child_1'] = wi_dells['nchildren'].map(child1_to_binary)
dfbin['child_2'] = wi_dells['nchildren'].map(child2_to_binary)
dfbin['child_3'] = wi_dells['nchildren'].map(child3_to_binary)
dfbin['child_4'] = wi_dells['nchildren'].map(child4_to_binary)
dfbin['child_5'] = wi_dells['nchildren'].map(child5_to_binary)

# binarize planning
planning_morethanmonth_to_binary = {'One Month or More Ago':1, 'This Month': 0,
'This Week': 0}
planning_thismonth_to_binary = {'One Month or More Ago':0, 'This Month': 1,
'This Week': 0}
planning_thisweek_to_binary = {'One Month or More Ago':0, 'This Month': 0,
'This Week': 1}
dfbin['planning_morethanmonth'] = wi_dells['planning'].map(planning_morethanmonth_to_binary)
#create new column
dfbin['planning_thismonth'] = wi_dells['planning'].map(planning_thismonth_to_binary)
dfbin['planning_thisweek'] = wi_dells['planning'].map(planning_thisweek_to_binary)

# binarize sex
sex_to_male = {'Male':1 ,'Female':0}
sex_to_female = {'Male':0, 'Female':1}
df.male = wi_dells['sex'].map(sex_to_male)
df.female = wi_dells['sex'].map(sex_to_female)

# binarize age

age_LT25_to_binary = {'LT 25':1,'25-34':0,'35-44':0,'45-54':0,'55-64':0,'65+':0}
age25to34_to_binary = {'LT 25':0,'25-34':1,'35-44':0,'45-54':0,'55-64':0,'65+':0}
age35to44_to_binary = {'LT 25':0,'25-34':0,'35-44':1,'45-54':0,'55-64':0,'65+':0}
age45to54_to_binary = {'LT 25':0,'25-34':0,'35-44':0,'45-54':1,'55-64':0,'65+':0}
age55to64_to_binary = {'LT 25':0,'25-34':0,'35-44':0,'45-54':0,'55-64':1,'65+':0}
age65plus_to_binary = {'LT 25':0,'25-34':0,'35-44':0,'45-54':0,'55-64':0,'65+':1}
dfbin['age_LT25'] = wi_dells['age'].map(age_LT25_to_binary)
dfbin['age25to34'] = wi_dells['age'].map(age25to34_to_binary)
dfbin['age35to44'] = wi_dells['age'].map(age35to44_to_binary)
dfbin['age45to54'] = wi_dells['age'].map(age45to54_to_binary)
dfbin['age55to64'] = wi_dells['age'].map(age55to64_to_binary)
dfbin['age65plus'] = wi_dells['age'].map(age65plus_to_binary)

# binarize education
ed_HSGrad_to_binary = {'HS Grad or Less':1,'Some College':0,'College Grad': 0,
'Post Grad':0}
ed_PostGrad_to_binary = {'HS Grad or Less':0,'Some College':0,'College Grad': 0,
```

```
'Post Grad':1}
ed_CollegeGrad_to_binary = {'HS Grad or Less':0,'Some College':0,'College Grad':1,
'Post Grad':0}
ed_SomeCollege_to_binary = {'HS Grad or Less':0,'Some College':1,'College Grad':0,
'Post Grad':0}
dfbin['ed_HSGrad'] = wi_dells['education'].map(ed_HSGrad_to_binary)
dfbin['ed_PostGrad'] = wi_dells['education'].map(ed_PostGrad_to_binary)
dfbin['ed_CollegeGrad'] = wi_dells['education'].map(ed_CollegeGrad_to_binary)
dfbin['ed_SomeCollege'] = wi_dells['education'].map(ed_SomeCollege_to_binary)

# binarize income

income_low_to_binary = {'Lower Income':1,'Middle Income':0,'Upper Income':0}
income_mid_to_binary = {'Lower Income':0,'Middle Income':1,'Upper Income':0}
income_upper_to_binary = {'Lower Income':0,'Middle Income':0,'Upper Income':1}
dfbin['income_low'] = wi_dells['income'].map(income_low_to_binary)
dfbin['income_mid'] = wi_dells['income'].map(income_mid_to_binary)
dfbin['income_upper'] = wi_dells['income'].map(income_upper_to_binary)

# binarize region

reg_Chicago_to_binary = {'Chicago':1,'Madison':0,'Milwaukee':0,
'Minneapolis/StPaul':0,'Other Wisconsin':0,'Other':0}
reg_Madison_to_binary = {'Chicago':0,'Madison':1,'Milwaukee':0,
'Minneapolis/StPaul':0,'Other Wisconsin':0,'Other':0}
reg_Milw_to_binary = {'Chicago':0,'Madison':0,'Milwaukee':1,
'Minneapolis/StPaul':0,'Other Wisconsin':0,'Other':0}
reg_Minn_to_binary = {'Chicago':0,'Madison':0,'Milwaukee':0,
'Minneapolis/StPaul':1,'Other Wisconsin':0,'Other':0}
reg_OtherWis_to_binary = {'Chicago':0,'Madison':0,'Milwaukee':0,
'Minneapolis/StPaul':0,'Other Wisconsin':1,'Other':0}
reg_Other_to_binary = {'Chicago':0,'Madison':0,'Milwaukee':0,
'Minneapolis/StPaul':0,'Other Wisconsin':0,'Other':1}
dfbin['reg_Chicago'] = wi_dells['region'].map(reg_Chicago_to_binary)
dfbin['reg_Madison'] = wi_dells['region'].map(reg_Madison_to_binary)
dfbin['reg_Milw'] = wi_dells['region'].map(reg_Milw_to_binary)
dfbin['reg_Minn'] = wi_dells['region'].map(reg_Minn_to_binary )
dfbin['reg_OtherWis'] = wi_dells['region'].map(reg_OtherWis_to_binary)
dfbin['reg_Other'] = wi_dells['region'].map(reg_Other_to_binary)

# ------------------------------------------------------------------------
# transform 0 and 1 to column names to create itemsets
# ------------------------------------------------------------------------

# "1" activity cells of the data frame replaced by name of activity
# "O" coded as missing data using numpy nan method
for index in range(dfbin.columns.size):
bin_to_name = {0 : np.nan, 1 : dfbin.columns[index]}
```

```
dfbin[dfbin.columns[index]] = dfbin[dfbin.columns[index]].map(bin_to_name)
#activity_to_name = {'NO' : np.nan, 'YES' : activity[index]}
#df[activity[index]] = df[activity[index]].map(activity_to_name)

print(dfbin.head) # examine the first few lines of the recoded data frame

# ------------------------------------------------------------------------
# Execute Apriori against categorical inputs to determine association to
# activity of interest "rideducks"
# ------------------------------------------------------------------------

# my_bindata is the list of lists structure that is needed
# for input to the apriori algorithm
my_bindata = list() # initialize list structure for list of lists

# loop through the rows of the data frame by index ix
# creating the list of lists structure one activity basket at a time
# we use dropna() to omit missing data as coded nan from numpy

# generate the activity basket for this visitor
# and convert it to a list for each of the df.shape[0] visitors
for index in range(dfbin.shape[0]):
basket = list(dfbin.ix[index].dropna()) # this list for one visitor
my_bindata.append(basket) # add this list to list of lists

# ------------------------------------------------
# Apply the apriori algorithm to the list of lists
# ------------------------------------------------

alpha2 = 0.2
beta2 = 0.2

L2, suppData2 = apriori.apriori(my_bindata, alpha2) # returns list of frozen sets and Support

print('Identified rules with support = ', alpha2, 'and confidence = ', beta2)
rules = apriori.generateRules(L2, suppData2, minConf = beta2)

# search across the rule set starting with smaller sets
# and moving to larger and larger sets until no rules
# exist that satisfy the requirement of including item_of_interest

n_other_items = 1 # initialize reporting at one other item
while n_other_items <= max_other_items:
print('\nRules with ', n_other_items, ' other item(s)')
for item in L[n_other_items]:
if item.intersection(item_of_interest): print(item)
n_other_items = n_other_items + 1
```

References

Harrington, J. C. (2007). *Wisconsin Dells.* Madison, Wisconsin: Research Publishers.

Harrington, P. (2012). Machine learning in action. Shelter Island, N.Y.: Manning.