Course Project

CS773: Data Mining and Security

Summer 2016

John Berlin, Yun Han

Contents

List of Tables

List of Figures

Listings

Data Changes

For questions two through four the data was reduced to *cuisine,atmosphere,occasion,price,style*. Any restaurant that did not have one of those features the fake feature none was added and for those that had multiple features per feature category i.e one atmosphere, cuisine and price but two occasion features with non style the following was done. This is shown in table 1.

Table 1: Data change example

cuisine	atmosphere	occasion	price	style
Italian	Excellent Decor	Open on Sundays	$15-$30	none
Italian	Excellent Decor	Open on Mondays	$15-$30	none

Task i

```
Study the provided features and classify them into one of the standard (cuisine, style,
price, atmosphere, and occasion) or into your own created additional categories. Limit
the new categories to at most 5. If you think that a feature fits into more than one of
your categories, do put them in all the categories that they fit in. This could be more
an exception than a role. Typically, there should be a 1-1 mapping of features to
categories.
```

Results

In order to get this classification in a timely manner python was used to perform data manipulation to aid in this by hand classification. The libraries nltk and pycountry were utilized to do the bulk of this work. This file can be seen in the appendices section listing 3. The first step was to group like features through usage of n-grams. Features such as *Fine for Dining Alone* and *Dining After the Theater* were grouped together as both had grams produced that started with Dining. This method did not work for all of the features and left roughly five to eight percent ungrouped. Due to the vast majority of the features being grouped already by this technique it was simple enough to move these features to the correct classification group. The results can be seen below.

Table 2: Atmosphere Classification

An Historic Spot	An Out Of The Way Find	Authentic
Buffet Dining	Business Scene	Cafe/Garden Dining
Classic Hotel Dining	Creative	Credit cards are not accepted
Excellent Decor	Excellent Food	Excellent Service
Extraordinary Decor	Extraordinary Food	Extraordinary Service
Fabulous Views	Fabulous Wine Lists	Fair Decor
Fair Food	Fair Service	Focus on Dessert
For the Young and Young at Heart	Good Decor	Good Food
Good Out of Town Business	Good Service	Good for Younger Kids
Great for People Watching	Health Conscious Menus	Hip Place To Be
Little Known But Well Liked	Near-perfect Decor	Near-perfect Food
Near-perfect Service	Need To Dress	No Liquor Served
No Reservations	No Smoking Allowed	Old World Cafe Charm
On the Beach	Parking/Valet	People Keep Coming Back
Place for Singles	Poor Decor	Pub Feel
Quiet for Conversation	Quirky	Relaxed Senior Scene
Romantic	Singles Scene	Tourist Appeal
Up and Coming	Very Busy - Reservations a Must	Warm spots by the fire
Wheelchair Access		

Table 3: Cuisine Classification

Afghanistan	African	American	Argentinean	Armenian
Asian	Austrian	Bar-B-Q	Belgian	Brazilian
Burmese	Burritos	Cajun	Cambodian	Canadian
Caribbean	Chinese	Coffee and Dessert	Creole	Cuban
Czech	Dim Sum	Egyptian	Ethiopian	Filipino
Fountain and Ice Cream	French	German	Greek	Guatemalan
Hamburgers	Beer & Hot Dogs	Hungarian	Indian	Indonesian
Irish	Italian	Jamaican	Japanese	Jewish
Korean	Latin	Lebanese	Malaysian	Mediterranean
Mexican	Moroccan	Nicaraguan	Pacific New Wave	Pacific Rim
Persian	Peruvian	Polish	Polynesian	Portuguese
Puerto Rican	Romanian	Roumanian	Russian	Salvadoran
Scandinavian	Seafood	Spanish	Sushi	Swiss
Tapas	Tex Mex	Tex-Mex	Thai	Tibetan
Tunisian	Turkish	Ukrainian	Ukranian	Vegetarian
Venezuelan	Vietnamese	Wine and Beer	Yugoslavian	

Table 4: Occasion Classification

After Hours Dining	Catering for Special Events	Dancing	Delivery Available
Dining After the Theater	Dining Outdoors	Early Dining	Entertainment
Fine for Dining Alone	Game	Great Place to Meet for a Drink	Happy Hour
Late Night Menu	Long Drive	Margaritas	Menus in Braille
Open for Breakfast	Open on Mondays	Open on Sundays	Other Quick Food
Parties and Occasions	Picnics	Pre-theater Dining	Private Parties
Private Rooms Available	Prix Fixe Menus	See the Game	Short Drive
Special Brunch Menu	Takeout Available	Walk	Weekend Brunch
Weekend Dining	Weekend Jazz Brunch	Weekend Lunch	

Table 5: Style Classification

A	American (Contemporary)	American (New)	American (Regional)
American (Traditional)	Bakeries	Brasserie	Cab
Cafe/Espresso Bars	Cafeterias	Californian	Carry in Wine and Beer
Caviar	Central	Coffee Houses	Continental
Haute Creole	Haute New Orleans	Coffeehouses	Coffee Shops
Deli	Diners	Down-Home	Eastern European
Eclectic	Down-Home Creole	Southern	Southwestern
South American	Southeast Asian	English	Fast Food
Fondue	Franco-Russian	Frankfurters	French Bistro
French Classic	French Contemporary	French-Japanese	French (New)
French Nouvelle	Grills	Hamburgers	Health Food
High Tea	International	Italian (North	Italian (Northern)
Italian (North & South)	Italian Nuova Cucina	Italian (Southern)	Kosher
Lithuanian	Middle Eastern	N	Noodle Houses
Noodle Shops	Omelettes	Oyster Bars	Pancakes
Pastries	Pastry Shops	Pizza	Pizzerias
Po' Boys	Power Brokers	Scottish	Soul Food
Soulfood	Southern Comfort	Steakhouses	Swiss-French
Tacos	Traditional	Yogurt Bar	

Table 6: Price Classification

$15-$30	$30-$50	below $15	over $50

Task ii

Analyze and form rules to characterize the following five types of cuisine: (i) Indian (ii) Mexican (iii) Italian (iv) French (v) American. For each category, derive a set of rules based on data available from all cities.

Results

The results for task ii were generated from an R script seen in listing 4 which uses the C50 decision tree algorithm to produce rules. Data used in this script was generated using a python file `rules.py` which accompanies this report. The classification was aided through using boosting which is indicated by setting the *trialNum* to ten. The *rules* R script can be seen in this listing 6 in the appendices for this report. We also set the flag of rules to true in order to decompose the tree into the bare rules.

Overall the classification of the data the features *price*, *style*, *occasion* were heavily used when generating the rules as seen in table 9. Boosting resulted in a `38.7%` error margin with trials 0, 7 and 8 resulting in least error. The entire rule set for each trial can be found in the files *c50_10Trial(2).txt* which accompanies this report. Trial 0 generated the most number of rules which was 118 with the average number of rules being between 40-60. We omit listing the rules here as the file is 5977 lines long but include notable rules as seen below. This is a sampling seen in figure 1 is taken from all trials.

```
Rule 0/24: (582, lift 6.4)
style in {Bakeries, Brasserie, Caviar}
->  class French  [0.998]

Rule 0/38: (3, lift 5.1)
atmosphere = Romantic
occasion = After Hours Dining
price = $30-$50
style = none
->  class French  [0.800]

Rule 0/58: (4, lift 56.3)
atmosphere in {Good Food, Good Service}
price = $15-$30
style = Carry in Wine and Beer
->  class Indian  [0.833]

Rule 0/106: (8, lift 11.5)
occasion in {Dining Outdoors, Late Night Menu}
price = below $15
style = Carry in Wine and Beer
->  class Mexican  [0.900]

Rule 3/4: (1984.1/437.8, lift 1.8)
occasion in {Dancing, Delivery Available, Dining After the Theater,
                     Early Dining, Fine for Dining Alone,
                     Great Place to Meet for a Drink, Long Drive,
                     Menus in Braille, Open on Mondays, Open on Sundays,
                     Picnics, See the Game, Short Drive, Special Brunch Menu,
                     Takeout Available, Weekend Dining}
style in {Coffee Shops, Eclectic, High Tea, Omelettes, Pastries,
                   Steakhouses}
->  class American  [0.779]
```

Figure 1: Sampling of rules from C50

Table 7: Task ii Classification Errors

```
Evaluation on training data (61063 cases):
Trial         Rules
-----  ----------------
    No      Errors

   0    119 22584(37.0%)
   1     60 25297(41.4%)
   2     61 24058(39.4%)
   3     45 25904(42.4%)
   4     61 24293(39.8%)
   5     53 25591(41.9%)
   6     43 25873(42.4%)
   7     78 23947(39.2%)
   8     65 23669(38.8%)
   9     52 24269(39.7%)
boost       23614(38.7%)
```

Table 8: Task ii Classification Confusion Matrix

```
(a)   (b)   (c)   (d)   (e)    <-classified as
 ----  ----  ----  ----  ----
25111  1206        2420   360    (a): class American
 4754  3533        1220    10    (b): class French
  640    15         229    20    (c): class Indian
 8781   572        7247   150    (d): class Italian
 2769               468  1558    (e): class Mexican
```

Table 9: Task ii Attribute Usage

```
100.00% price
100.00% style
 99.99% occasion
 89.66% atmosphere
```

Task iii

```
Derive association rules among the given features. In other words, does Creative
atmosphere imply a specific category? In particular, experiment with the following
associations:
a. Cuisine and atmosphere
b. Price and atmosphere
c. Price and style
d. Cuisine and occasion.
e. Dcor and Price
```

Results

Assumptions

1. The fake feature none would not have a great impact in the generated rules
2. The number of item tuples(cuisine,atmosphere,occasion,price,style) used for rule mining, which was 99881, would produce rules with a minimal support of at least 0.2
3. The minimal confidence that the generated rules would have to have to be considered acceptable would be 0.49

We chose to use R for this task and library arules. The *arules* library allows for control over the left(lhs) and right(rhs) hand sides of the *apriori* algorithm which was a specific requirement for this task. Being able to control the lhs or the rhs of the apriori algorithm in this library does not mean the we artificially created the generated rules. Rather it means that the library allows us to state which item labels can appear in regards to what kind of rules we wish to find. For example for part *a* of this task, we were able, through using this library, to specify only the usage of features that are classified as cuisine and atmosphere for the lhs and all others features are the implication of the rule. By implication we mean the rhs whose lhs is either one of or both cuisine and atmosphere features found in the dataset. The R script used for this task can be seen in the appendices listing 5.

After generating the rules it was found that the first two assumptions turned out to be untrue. The maximum support found for the majority of the rules generated was 0.15 but the confidence assumption for all generated rules was confirmed. We believe that is was due to the need for specifying the fake feature none and the number of item tuples generated to meet the requirements of the *apriori* algorithm. The feature classification of *style* had the most *none* items which resulted in rules whose rhs was *style=none*. Since the *apriori* algorithm mines rules from frequent itemsets and the data given to it is considered transactions(the item tuples defined in assumptions). We feel our belief as to why the first two assumptions were proven wrong is correct and that our results are still valid due to the nature of the *apriori* algorithm.

To accommodate for our initial assumptions being proven wrong we re-ran the rule mining for each part of this task but with the following additions:

1. Rules were generated from data that had all item tuples with *style=none* removed
2. Rules were generated from the data whose lhs had only the unique feature pairs for each respective part of this task both with *style=none* item tuples and without *style=none* item tuples. It must be noted for the rules generated through unique pair, it was gotten by filtering the original data with unique pairs used in computing the lhs. Example: (atmosphere=Good for Younger Kids,cuisine=Portuguese) this was present in the data so we looked for this in the generated rule

The data generated for this task can be seen in the folder **q3** which accompanies this report. It must be noted we omit the support, confidence and lift metrics in the rules shown as doing so would not allow the results to be presentable for this report. For the the rules shown in this report, we took at most the first 50 rules from a subset of the rules which meet the constraint of assumption three. To accommodate for this we will first present an overview of the generated rules per section table 10 and then show the rules in tabular format and these tables are included in the appendices of this report due to their size.

As seen in table 10 using the feature Cuisine in combination with another generated the largest amount of rules, with atmosphere pairing generating the second largest. We believe this is the case because of the subjective nature of such ratings(atmosphere), so this pairing is not as substantial to others per the reasons stated before. We apply this same logic to Cuisine and occasion. Price in combination with another did not generate as many rules as was the case when paired with atmosphere, we believe this is indicative of Price being an important determiner to rules in comparison to the others. Our own intuition and experience tells us price is an important factor in where to eat and its quality. Price in combination with Décor also matched our intuition about how we ourselves judge a restaurant.

Table 10: Task iii Rule Overview

Cuisine and atmosphere

Rule Generation	Number of Rules	Support	Confidence	Lift	Conviction
Normal	550	[0.000501, 0.153192]	[0.490066, 1.000000]	[0.862753, 387.135659]	[0.847118, 20.972037]
No *style=none*	417	[0.000501, 0.074147]	[0.491667, 1.000000]	[1.042620, 193.310078]	[1.039538, 29.900309]
Unique pairs	399	[0.000501, 0.014497]	[0.490066, 1.000000]	[0.862753, 25.736673]	[0.847118, 14.471131]
Unique pairs and No *style=none*	417	[0.000501, 0.074147]	[0.491667, 1.000000]	[1.042620, 193.310078]	[1.039538, 29.900309]

Price and atmosphere

Normal	112	[0.000501, 0.332556]	[0.491228, 0.915493]	[0.981150, 10.255605]	[0.981450, 5.908788]
No *style=none*	69	[0.000501, 0.017544]	[0.490196, 1.000000]	[1.354738, 144.982558]	[1.256716, 16.154309]
Unique pairs	56	[0.000501, 0.007760]	[0.490196, 1.000000]	[1.354738, 144.982558]	[1.256716, 16.154309]
Unique pairs and No *style=none*	79	[0.000511, 0.036453]	[0.491228, 0.886792]	[0.981150, 10.255605]	[0.981450, 4.410786]

Price and style

Normal	64	[0.000551, 0.332556]	[0.490196, 1.000000]	[1.169360, 245.407862]	[1.204546, 22.391736]
No *style=none*	80	[0.000541, 0.042607]	[0.490196, 1.000000]	[1.341456, 184.718519]	[1.244751, 22.212175]
Unique pairs	54	[0.000541, 0.032642]	[0.490196, 1.000000]	[1.341456, 184.718519]	[1.244751, 22.212175]
Unique pairs and No *style=none*	41	[0.000551, 0.016299]	[0.490196, 1.000000]	[1.682691, 245.407862]	[1.390110, 22.391736]

Cuisine and occasion

Normal	477	[0.000501, 0.153192]	[0.490385, 1.000000]	[0.883225, 387.135659]	[0.866887, 20.972037]
No *style=none*	367	[0.000501, 0.074147]	[0.490385, 1.000000]	[1.042910, 193.310078]	[1.039817, 20.243142]
Unique pairs	352	[0.000501, 0.022096]	[0.490385, 1.000000]	[0.884756, 243.703641]	[0.868401, 18.229305]
Unique pairs and No *style=none*	271	[0.000501, 0.025785]	[0.490385, 1.000000]	[1.042910, 121.689565]	[1.039817, 11.969015]

Décor and Price

Normal	14	[0.000501, 0.332556]	[0.492777, 0.683673]	[0.984243, 5.889286]	[0.984447, 2.061567]
No *style=none*	7	[0.000622, 0.017544]	[0.525424, 0.648148]	[1.746907, 6.037227]	[1.754707, 2.367642]
Unique pairs	5	[0.000622, 0.004672]	[0.525424, 0.644068]	[1.746907, 6.037227]	[1.754707, 2.355837]
Unique pairs and No *style=none*	8	[0.000781, 0.026251]	[0.498507, 0.683673]	[1.200877, 5.889286]	[1.252212, 2.061567]

Task iv

Let us now concentrate specifically on the quality of the food. This is specified through features 73-78. Assuming that the outcome you are interested is in one of the following categories, determine if there is any relationship between type of cuisine and the quality indicator. Categories to be considered are: Fair, Good, Excellent. You can combine the given 6 categories into these 3 categories.

Results

Task v

```
Find an association between a restaurant offering vegetarian (243) to its price and
cuisine.
```

Results

1. Assumptions
   (a) The cuisine types are divided manually and are most of the origination of the food type.
   (b) The restaurants that do not have vegetarian (243) feature are considered not offering vegetarian
   (c) The first cuisine type encountered in the restaurant data is extracted for association rule mining.
   (d) Weka is used to mine the association rules between the three attributes: cuisine, price and vegietarian

The Tertius method works well to mine the association rules for vegetarian restaurant. Below in figure 2 please find the results from Weka. The default scheme was used (weka.associations.Tertius -K 10 -F 0.0 -N 1.0 -L 4 -G 0 -c 0 -I 0 -P 0). The perl code used for the task is seen in listing 1.

```
 1. /* 0.098122 0.017548 */ price = 162 ==> vegie = Yes or cuisine = 221
 2. /* 0.094522 0.016346 */ price = 162 ==> vegie = Yes or cuisine = 229
 3. /* 0.093922 0.018029 */ price = 162 ==> vegie = Yes or cuisine = 058
 4. /* 0.093168 0.018510 */ price = 162 ==> cuisine = 221
 5. /* 0.092312 0.017548 */ price = 162 ==> vegie = Yes or cuisine = 142
 6. /* 0.091351 0.018269 */ price = 162 ==> vegie = Yes or cuisine = 009
 7. /* 0.089326 0.017308 */ price = 162 ==> cuisine = 229
 8. /* 0.088972 0.018990 */ price = 162 ==> cuisine = 058
 9. /* 0.088802 0.018269 */ price = 162 ==> cuisine = 142
10. /* 0.088764 0.018990 */ price = 162 ==> vegie = Yes or cuisine = 186
Number of hypotheses considered: 10543
Number of hypotheses explored: 6667
```

Figure 2: Tertius results

Task vi

```
Determine the error that would be incurred by categorizing the restaurants based on
the continents they represent: Asia, Europe, Africa, North America, and South
America. For each continent, form rules to determine the outcome (which continent
they come from) based on other attributes such as price, atmosphere, quality of food,
etc.
```

Results

1. Assumptions
   (a) The restaurants that contain geographical information features can be accurately categorized by continent.
   (b) The list of countries and continents is a complete list.
   (c) The restaurants that do not match the list of countries and continents cannot be categorized by continents and generate errors.

The total number of restaurants: 4160. The number of restaurants that contain geographical features: 3758.

$$Error = \frac{4160 - 3758}{4160} * 100\% = 9.66\%$$

Perl code to get the number of total restaurants and the number of restaurants that contain geographical features can bee seen in listing 2.

Task vii

```
Selecting the Europe continent, state rules to determine the city based on the given
features. (Choose your own features or categories that are most relevant)
```

Results

1. Assumptions
   (a) Each restaurant has one feature in each category. If multiple features are found, the last one was chosen.
   (b) Any of the features that contain European geographical information are considered Europe continent and the restaurants are included.
   (c) Weka is used to determine the city based on the features. NaiveBayes method is employed.

Table 11: Task vii Classification Errors

```
Evaluation on training data (29414 cases):

Trial         Rules
-----  ----------------
    No      Errors

   0     57 7711(26.2%)
   1     22 8551(29.1%)
   2     27 8940(30.4%)
   3     38 8752(29.8%)
   4     33 9137(31.1%)
   5     33 9848(33.5%)
   6     36 8883(30.2%)
   7     45 9217(31.3%)
   8     37 8217(27.9%)
   9     30 8220(27.9%)
boost         7778(26.4%)   <<
```

Table 12: Task vii Attribute Usage

```
Attribute usage:
100.00% price
100.00% style
99.81% atmosphere
99.04% occasion
```

Table 13: Task vii Classification Confusion Matrix

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	classified as
									7							(a): class Armenian
				14					40							(b): class Austrian
		21							3							(c): class Belgian
				12					40							(d): class Czech
				6122					3395							(e): class French
				26	56				407							(f): class German
				41		106			603							(g): class Greek
				12					124							(h): class Hungarian
				12					115							(i): class Irish
				1443		48			15211				48			(j): class Italian
				12					159							(k): class Polish
									115		45					(l): class Portuguese
									30							(m): class Romanian
				68					871				75			(n): class Spanish
				6					120							(o): class Swiss
									7							(p): class Ukrainian

Appendices

Perl Scripts

```
use strict;
use warnings;

my @cuisine = split /,/, "
    002,003,005,006,007,008,009,012,013,014,015,018,020,022,031,032,033,034,038,039,045,048,049,058,067,069,070,07
    ";
my %cuisine_elements;
@cuisine_elements{@cuisine} = ();
my @price = split /,/, "161,162,163,164,165,166,167,168,169,170";
my %price_elements;
@price_elements{@price} = ();

my $dir = 'C:/Users/yhan/Documents/data';
my $out = 'C:/Users/yhan/Documents/CS773/q5/vegie_cuisinePrice.arff';
open(my $file, '>', $out) or die "Could not open file '$out' $!";
foreach my $fp (glob("$dir/*.txt")) {
  open my $fh, "<", $fp or die "can't read open '$fp'";
  while (<$fh>) {
    chomp;
    my $line = $_;
    my @features = split /[\s,\t]/, $_;
    my @rescui = grep exists $cuisine_elements{$_}, @features;
    if (length($rescui[0]) == 0) {$rescui[0] = '?'}
    my @respri = grep exists $price_elements{$_}, @features;
    if (length($respri[0]) == 0) {$respri[0] = '?'}
    if ($line =~ /243/) {
      printf $file "$rescui[0], $respri[0], Yes \n"
    } else {
      printf $file "$rescui[0], $respri[0], No \n"
    };

  }

  close $fh or die "can't read close";
}
```

Listing 1: Task v Perl script

```
use strict;
use warnings;

my $features = 'C:/Users/yhan/Documents/CS773/q6/features.txt';
open(my $featurefile, '<', $features) or die "Could not open file!";
my %feature;
while (<$featurefile>) {
  my $line = $_;
  my ($i, $j) = split(/\t/, $line);
  $feature{$i} = $j;
  #print "$i is $feature{$i}\n";
  }


my @countries;
my $country = 'C:/Users/yhan/Documents/CS773/q6/countries.txt';
open(my $file, '<', $country) or die "Could not open file!";
while(<$file>) {
  chomp;
  my $line = $_;
  push(@countries, $line);
  #print $_;
  }
#my %country_elements;
#@country_elements{@countries} = ();

my $countrys = "African Afro-Eurasian ...";
my $total = 0;
my $geo = 0;

my $dir = 'C:/Users/yhan/Documents/data';
```

```perl
foreach my $fp (glob("$dir/*.txt")) {
  open my $fh, "<", $fp or die "can't read open '$fp'";
  while (<$fh>) {
    chomp;
    my $line = $_;
    $total = $total + 1;
    my $flag = 0;
    my @resfeatures = split /[\s,\t]/, $_;
    foreach my $ele (@resfeatures)
    {
    if(exists $feature{$ele}){
      my $cty = $feature{$ele};
      $cty =~ tr/A-Za-z//cd;
      if ($countrys =~ /$cty/)
      {
      $flag = 1
      #print $feature{$ele};
      }

    #print $feature{$ele};
    }
          }
    $geo = $geo + $flag;
    }
close $fh or die "can't read close";
}

print "$total, $geo \n";
```

Listing 2: Task vi Perl script

Python Scripts

```python
import csv
import glob
import json
import os
import re
from collections import Counter
from os import path

from functional import seq
from nltk.collocations import ngrams
from nltk.corpus import gazetteers
from nltk.corpus import stopwords
from nltk.stem.snowball import EnglishStemmer
from nltk.tag import StanfordNERTagger
from nltk.tokenize import RegexpTokenizer
from nltk.util import everygrams

from restaurantOriginalData import Restaurant
from nationalityToCountry import convert

foodExtractor = re.compile('^(\d+)\t(.+)\t(.+)$')
featureExtractor = re.compile('^(\d+)\t(.+)$')

engStop = stopwords.words('english')
eng = EnglishStemmer()


def cleanFeatures(line):
    return featureExtractor.match(line.rstrip('\n')).groups()


def cleanFood(line):
    return foodExtractor.match(line.rstrip('\n')).groups()


class bcolors:
    HEADER = '\033[95m'
    OKBLUE = '\033[94m'
    OKGREEN = '\033[92m'
    WARNING = '\033[93m'
```

```
    FAIL = '\033[91m'
    ENDC = '\033[0m'

    def disable(self):
        self.HEADER = ''
        self.OKBLUE = ''
        self.OKGREEN = ''
        self.WARNING = ''
        self.FAIL = ''
        self.ENDC = ''


def explore1():
    restaurants = []
    features = {}
    print(glob.glob('dataset/*.txt'))
    with open('dataset/features.txt', 'r') as featureIn:
        for line in map(cleanFeatures, featureIn):
            features[line[0]] = line[1]

    for file in glob.glob('dataset/*.txt'):
        if not file == 'dataset/features.txt':
            with open(file, 'r') as fin:
                for food in map(cleanFood, fin):
                    restaurants.append(Restaurant(file, food, features))

    fr = seq(restaurants)  # type: seq

    grouped = fr.flat_map(lambda f: list(map(lambda v: (v, f), f.featureVector))) \
        .group_by(lambda fv: fv[0]) \
        .map(lambda item: (item[0], list(map(lambda it: it[1], item[1])))) \
        .to_dict()
    for k, v in grouped.items():
        print(k, v)
        print('==========\n\n')


def word_grams(words, min=1, max=4):
    s = []
    for n in range(min, max):
        for ngram in ngrams(words, n):
            s.append(' '.join(str(i) for i in ngram))
    return s


def append_elements(n_gram):
    for element in range(len(n_gram)):
        phrase = ''
        for sub_element in n_gram[element]:
            phrase += sub_element + ' '
        n_gram[element] = phrase.strip().lower()
    return n_gram


def compare(n_gram1, n_gram2):
    n_gram1 = append_elements(n_gram1)
    n_gram2 = append_elements(n_gram2)
    common = []
    for phrase in n_gram1:
        if phrase in n_gram2:
            common.append(phrase)
    if not common:
        print("Nothing in common between")
        # or you could print a message saying no commonality was found
    else:
        for i in common:
            print(i)


def firstPassGrouping():
    words = []

    stemmed = []
    features = {}
    tokenizer = RegexpTokenizer('\s+', gaps=True)
    clean = re.compile("[()\/']")
```

```
    split = re.compile("[/]")
    grams = []
    with open('dataset/features.txt', 'r') as featureIn:
        for line in map(cleanFeatures, featureIn):
            ws = []
            for w in tokenizer.tokenize(clean.sub(' ', line[1])):
                if w not in engStop:
                    stemmed.append((eng.stem(w).lower(), line[1]))
                    words.append((w.lower(), line[1]))
                    ws.append(w.lower())

            grams.append((list(everygrams(ws, min_len=2, max_len=2)), line[1]))
            features[line[0]] = line[1]


    # cuisine, style, price, atmosphere, and occasion


    noGrams = set(map(lambda x: x[1], filter(lambda x: len(x[0]) == 0, grams)))

    grams = list(filter(lambda x: len(x[0]) > 0, grams))
    groupedw = seq(grams) \
        .flat_map(lambda x: set([(w, x[1]) for w in seq(x[0]).flat_map(lambda y: list(y)).to_list()])) \
        .group_by(lambda w: w[0]) \
        .map(lambda x: (x[0], list(map(lambda y: y[1], x[1])))) \
        .to_dict()

    noGramsId = {}
    for g in noGrams:
        noGramsId[g] = g
    simGrouped = {}
    simular = set()
    for k, v in sorted(groupedw.items(), key=lambda x: x[0]):
        # print(k, v)
        nl = v.copy()
        match = noGramsId.get(k, None)
        for nk in noGramsId.keys():
            if len(nk) > 1:
                if nk in v:
                    nl.append(nk)
                    simular.add(nk)
                for vv in v:
                    if nk in vv:
                        nl.append(nk)
                        simular.add(nk)

        if match is not None:
            nl.append(match)
            simGrouped[k] = list(set(nl))
            simular.add(match)
        else:
            if len(k) > 1:
                simGrouped[k] = v

    noSim = noGrams - simular
    #
    nationalities = gazetteers.words()

    featureNationality = []
    for nosim in noSim:
        didConvert = convert(nosim)
        if didConvert is not None:
            if didConvert in nationalities:
                featureNationality.append(nosim)
        else:
            if nosim in nationalities:
                featureNationality.append(nosim)
            else:
                split = nosim.split('-')
                for sp in split:
                    if sp in nationalities:
                        featureNationality.append(nosim)

    # print("------------------")


```

```
    noSim = noSim - set(featureNationality)
    # occasions = ['monday']
    # # cuisine, style, price, atmosphere, and occasion
    for k, v in sorted(simGrouped.items(), key=lambda x: x[0]):
        # print(k,v)
        if k in nationalities:
            featureNationality.append(k)
            featureNationality.extend(v)
            simGrouped.pop(k)
        didConvert = convert(k)
        if didConvert is not None:
            if didConvert in nationalities:
                simGrouped.pop(k)
                featureNationality.append(k)
                featureNationality.extend(v)

    with open('q1/noSim.json', 'w+') as nsOut:
        nsOut.write(json.dumps(list(noSim), indent=2, sort_keys=True))

    with open('q1/featureNationality.json', 'w+') as nsOut:
        nsOut.write(json.dumps(featureNationality, indent=2, sort_keys=True))

    with open('q1/grouped.json', 'w+') as nsOut:
        nsOut.write(json.dumps(simGrouped, indent=2, sort_keys=True))


def useOtherDataSet():
    knownCuisine = set()
    cuisineCounter = Counter()
    with open('chefmozcuisine.csv', 'r') as cin:
        for row in csv.DictReader(cin):
            cuisineCounter[row['Rcuisine']] += 1
            knownCuisine.add(row['Rcuisine'])

    with open('usercuisine.csv', 'r') as cin:
        for row in csv.DictReader(cin):
            cuisineCounter[row['Rcuisine']] += 1
            knownCuisine.add(row['Rcuisine'])

    q1Cuisine = set()
    with open('dataset/features.txt', 'r') as featureIn:
        for line in map(cleanFeatures, featureIn):
            if line[1] in knownCuisine:
                q1Cuisine.add(line[1])
                print(line[1])
    #
    with open('notq1/q1Labels.json', 'r') as cin:
        labels = json.load(cin)
        labels["cuisine"] = sorted(list(q1Cuisine))
        print(labels)
        with open('q1/q1Labels.json', 'w+') as cout:
            json.dump(labels, cout, indent=2, sort_keys=1)


def cleanUp():
    split = re.compile("[\-\_]")
    knownCuisine = []
    cuisineCounter = Counter()
    with open('chefmozcuisine.csv', 'r') as cin:
        for row in csv.DictReader(cin):
            cuisineCounter[row['Rcuisine']] += 1
            knownCuisine.append(row['Rcuisine'])

            knownCuisine.extend(split.sub(' ', row['Rcuisine']).split(' '))
            knownCuisine.extend(split.sub(' ', row['Rcuisine']))

    with open('usercuisine.csv', 'r') as cin:
        for row in csv.DictReader(cin):
            cuisineCounter[row['Rcuisine']] += 1
            knownCuisine.append(row['Rcuisine'])
            knownCuisine.extend(split.sub(' ', row['Rcuisine']).split(' '))
            knownCuisine.extend(split.sub(' ', row['Rcuisine']))
    labels = []
    with open('q1/q1Labels.json', 'r') as cin:
        ls = json.load(cin)

```

```
    labelsSorted = {}

    for k, v in ls.items():
        print(k)
        labelsSorted[k] = sorted(list(set(v)))

    with open('q1/q1Labels.json', 'w+') as cout:
        json.dump(labelsSorted, cout, indent=2, sort_keys=1)

    for v in labelsSorted.values():
        labels.extend(v)
    labels = set(labels)
    grouped = {}
    test = {}
    with open('q1/grouped.json', 'r') as nsOut:
        g = json.load(nsOut)
        for k, v in g.items():
            if len(v) != 0:
                test[k] = v

    for k, v in test.items():
        nv = []
        for vv in v:
            if vv not in labels:
                nv.append(vv)
        if len(nv) != 0:
            grouped[k] = nv
    with open('q1/grouped.json', 'w+') as nsOut:
        nsOut.write(json.dumps(grouped, indent=2, sort_keys=True))
    print(grouped)


if __name__ == '__main__':
    print("hi")

    features = {}
    restaurants = []

    with open('dataset/features.txt', 'r') as featureIn:
        for line in map(cleanFeatures, featureIn):
            features[line[0]] = line[1]
            print(line[1])

    for file in glob.glob('dataset/*.txt'):
        if not file == 'dataset/features.txt':
            with open(file, 'r') as fin:
                for food in map(cleanFood, fin):
                    restaurants.append(Restaurant(file, food, features))

    with open('allCities.csv', 'w+') as cout:
        cout.write('restaurant,city,features\n')
        for r in restaurants:
            print(r)
            cout.write(r.dump_csv())
```

Listing 3: Feature Classification Task i

R Scripts

```
#!/usr/bin/Rscript
cwd <- getwd()
setwd(cwd)

#sink to redirect stdout to file

source(file=file.path(cwd,'/r/ruleFunctions.R'))

s1 <- read.csv('q2/cuisineCharacters2.csv')

ruleModel <- ruleGen.c50(d=s1,form =  cuisine ~ .,trialNum = 10,winnow = TRUE)


c50Summary <- capture.output(summary(ruleModel))
```

```
cat(c50Summary, file = file.path(cwd, 'q2', 'c50_10Trials2.txt'), sep="\n")
```

Listing 4: Task ii R script

```
#!/usr/bin/Rscript
cwd <- getwd()
setwd(cwd)

q3Dir <- file.path(cwd, 'q3')


source(file = file.path(cwd, 'r', 'helpers.R'))
source(file = file.path(cwd, 'r', 'ruleFunctions.R'))

q3DataList <- helpers.q3()

styleFiltered <-
  helpers.filter_df(q3DataList$q3Data, style != 'none')

styleFiltered$style <- factor(styleFiltered$style)

#### Q3.A
a_rules <- ruleGen.rules_apriori_lh(q3DataList$q3Data, q3DataList$ca)
a_rules <- subset(a_rules, confidence >= 0.49)

a_rules_no_noneStyle <-
  ruleGen.rules_apriori_lh(styleFiltered, q3DataList$ca)
a_rules_no_noneStyle <-
  subset(a_rules_no_noneStyle , confidence >= 0.49)

a_pairs <- helpers.s1_s2_pairs(q3DataList$ca)

a_rules_filter <-
  a_rules[a_rules$lhs %in% a_pairs$lhs1 |
            a_rules$lhs %in% a_pairs$lhs2, ]
a_rules_filter <- subset(a_rules_filter, confidence >= 0.49)

a_rules_pair_noneStyle <-
  a_rules_no_noneStyle[a_rules_no_noneStyle$lhs %in% a_pairs$lhs1 |
                         a_rules_no_noneStyle$lhs %in% a_pairs$lhs2 , ]
a_rules_pair_noneStyle  <-
  subset(a_rules_no_noneStyle, confidence >= 0.49)

a_rules <- helpers.orderConfidence(a_rules)
write.csv(
  helpers.select_rules_quality(a_rules),
  file = file.path(q3Dir, 'q3a_rule.csv'),
  fileEncoding = 'utf8',
  row.names = F
)

a_rules_no_noneStyle <- helpers.orderConfidence(a_rules_no_noneStyle)
write.csv(
 helpers.select_rules_quality(a_rules_no_noneStyle),
  file = file.path(q3Dir, 'q3a_rule_NNS.csv'),
  fileEncoding = 'utf8',
  row.names = F
)

a_rules_filter <- helpers.orderConfidence(a_rules_filter)
write.csv(
 helpers.select_rules_quality(a_rules_filter),
  file = file.path(q3Dir, 'q3a_rule_pairFilter.csv'),
  fileEncoding = 'utf8',
  row.names = F
)

a_rules_pair_noneStyle <- helpers.orderConfidence(a_rules_pair_noneStyle)
write.csv(
 helpers.select_rules_quality(a_rules_pair_noneStyle),
  file = file.path(q3Dir, 'q3a_rule_NNS_pair.csv'),
  fileEncoding = 'utf8',
  row.names = F
)

```

```
#### Q3.B
b_rules <- ruleGen.rules_apriori_lh(q3DataList$q3Data, q3DataList$ap)
b_rules <- subset(b_rules, confidence >= 0.49)
b_rules_no_noneStyle <-
  ruleGen.rules_apriori_lh(styleFiltered, q3DataList$ap)
b_rules_no_noneStyle <-
  subset(b_rules_no_noneStyle, confidence >= 0.49)
# b_cumaliative_rules = helpers.group_by_lhs(b_rules_no_noneStyle)

b_pairs <- helpers.s1_s2_pairs(q3DataList$ap)

b_rules_filter <-
  b_rules[b_rules$lhs %in% b_pairs$lhs1 |
            b_rules$lhs %in% b_pairs$lhs2, ]
b_rules_filter <- subset(b_rules_filter, confidence >= 0.49)
b_rulesNSN_filter <-
  b_rules_no_noneStyle[b_rules_no_noneStyle$lhs %in% b_pairs$lhs1 |
                         b_rules_no_noneStyle$lhs %in% b_pairs$lhs2, ]
b_rulesNSN_filter <- subset(b_rulesNSN_filter , confidence >= 0.49)


b_rules <- helpers.orderConfidence(b_rules)
write.csv(
 helpers.select_rules_quality(b_rules),
  file = file.path(q3Dir, 'q3b_rule.csv'),
  fileEncoding = 'utf8',
  row.names = F
)

b_rules_no_noneStyle <- helpers.orderConfidence(b_rules_no_noneStyle)
write.csv(
 helpers.select_rules_quality(b_rules_no_noneStyle),
  file = file.path(q3Dir, 'q3b_rule_NNS.csv'),
  fileEncoding = 'utf8',
  row.names = F
)

b_rules_filter <- helpers.orderConfidence(b_rules_filter)
write.csv(
 helpers.select_rules_quality(b_rules_filter),
  file = file.path(q3Dir, 'q3b_rule_pairFilter.csv'),
  fileEncoding = 'utf8',
  row.names = F
)

b_rulesNSN_filter <- helpers.orderConfidence(b_rulesNSN_filter)
write.csv(
 helpers.select_rules_quality(b_rulesNSN_filter),
  file = file.path(q3Dir, 'q3b_rule_NNS_filter.csv'),
  fileEncoding = 'utf8',
  row.names = F
)

#### Q3.C
c_rules <- ruleGen.rules_apriori_lh(q3DataList$q3Data, q3DataList$sp)
c_rules <- subset(c_rules, confidence >= 0.49)
c_rules_no_noneStyle <-
  ruleGen.rules_apriori_lh(styleFiltered, q3DataList$sp)
c_rules_no_noneStyle  <-
  subset(c_rules_no_noneStyle , confidence >= 0.49)
# c_cumaliative_rules = helpers.group_by_lhs(c_rules_no_noneStyle)

c_pairs <- helpers.s1_s2_pairs(q3DataList$sp)

c_rules_filter <-
  c_rules[c_rules$lhs %in% c_pairs$lhs1 |
            c_rules$lhs %in% c_pairs$lhs2, ]
c_rulesNSN_filter <-
  c_rules_no_noneStyle[c_rules_no_noneStyle$lhs %in% c_pairs$lhs1 |
                         c_rules_no_noneStyle$lhs %in% c_pairs$lhs2, ]

c_rules <- helpers.orderConfidence(c_rules)
c_rules_no_noneStyle <- helpers.orderConfidence(c_rules_no_noneStyle)
c_rules_filter <- helpers.orderConfidence(c_rules_filter)
c_rulesNSN_filter <- helpers.orderConfidence(c_rulesNSN_filter)

```

```
write.csv(
 helpers.select_rules_quality(c_rules),
  file = file.path(q3Dir, 'q3c_rule.csv'),
  fileEncoding = 'utf8',
  row.names = F
)

write.csv(
 helpers.select_rules_quality(c_rules_no_noneStyle),
  file = file.path(q3Dir, 'q3c_rule_NNS.csv'),
  fileEncoding = 'utf8',
  row.names = F
)
write.csv(
 helpers.select_rules_quality(c_rules_filter),
  file = file.path(q3Dir, 'q3c_rule_pairFilter.csv'),
  fileEncoding = 'utf8',
  row.names = F
)
write.csv(
 helpers.select_rules_quality(c_rulesNSN_filter),
  file = file.path(q3Dir, 'q3c_rule_NNS_filter.csv'),
  fileEncoding = 'utf8',
  row.names = F
)

#### Q3.D
d_rules <- ruleGen.rules_apriori_lh(q3DataList$q3Data, q3DataList$co)
d_rules <- subset(d_rules, confidence >= 0.49)
d_rules_no_noneStyle <-
  ruleGen.rules_apriori_lh(styleFiltered, q3DataList$co)
d_rules_no_noneStyle  <-
  subset(d_rules_no_noneStyle , confidence >= 0.49)
# d_cumaliative_rules = helpers.group_by_lhs(d_rules_no_noneStyle)

d_pairs <- helpers.s1_s2_pairs(q3DataList$co)

d_rules_filter <-
  d_rules[d_rules$lhs %in% d_pairs$lhs1 |
            d_rules$lhs %in% d_pairs$lhs2, ]
d_rules_filter <- subset(d_rules_filter, confidence >= 0.49)
d_rulesNSN_filter <-
  d_rules_no_noneStyle[d_rules_no_noneStyle$lhs %in% d_pairs$lhs1 |
                         d_rules_no_noneStyle$lhs %in% d_pairs$lh2, ]
d_rulesNSN_filter <- subset(d_rulesNSN_filter, confidence >= 0.49)


d_rules <- helpers.orderConfidence(d_rules)
d_rules_no_noneStyle <- helpers.orderConfidence(d_rules_no_noneStyle)
d_rules_filter <- helpers.orderConfidence(d_rules_filter)
d_rulesNSN_filter <- helpers.orderConfidence(d_rulesNSN_filter)

write.csv(
 helpers.select_rules_quality(d_rules),
  file = file.path(q3Dir, 'q3d_rule.csv'),
  fileEncoding = 'utf8',
  row.names = F
)
write.csv(
 helpers.select_rules_quality(d_rules_no_noneStyle),
  file = file.path(q3Dir, 'q3d_rule_NNS.csv'),
  fileEncoding = 'utf8',
  row.names = F
)
write.csv(
 helpers.select_rules_quality(d_rules_filter),
  file = file.path(q3Dir, 'q3d_rule_pairFilter.csv'),
  fileEncoding = 'utf8',
  row.names = F
)
write.csv(
 helpers.select_rules_quality(d_rulesNSN_filter),
  file = file.path(q3Dir, 'q3d_rule_pairFilter_NNS.csv'),
  fileEncoding = 'utf8',
  row.names = F
)
```

```
#### Q3.E
e_rules <- ruleGen.rules_apriori_lh(q3DataList$q3Data, q3DataList$dp)
e_rules <- subset(e_rules, confidence >= 0.49)
e_rules_no_noneStyle <-
  ruleGen.rules_apriori_lh(styleFiltered, q3DataList$dp)
e_rules_no_noneStyle  <-
  subset(e_rules_no_noneStyle , confidence >= 0.49)
# e_cumaliative_rules = helpers.group_by_lhs(e_rules_no_noneStyle)

e_pairs <- helpers.s1_s2_pairs(q3DataList$dp)

e_rules_filter <-
  e_rules[e_rules$lhs %in% e_pairs$lhs1 |
            e_rules$lhs %in% e_pairs$lhs2, ]
e_rules_filter <- subset(e_rules_filter, confidence >= 0.49)
e_rulesNSN_filter <-
  e_rules_no_noneStyle[e_rules_no_noneStyle$lhs %in% e_pairs$lhs1 |
                         e_rules_no_noneStyle$lhs %in% e_pairs$lhs2, ]
e_rulesNSN_filter <- subset(e_rulesNSN_filter, confidence >= 0.49)


e_rules <- helpers.orderConfidence(e_rules)
e_rules_no_noneStyle <- helpers.orderConfidence(e_rules_no_noneStyle)
e_rules_filter <- helpers.orderConfidence(e_rules_filter)
e_rulesNSN_filter <- helpers.orderConfidence(e_rulesNSN_filter)

write.csv(
 helpers.select_rules_quality(e_rules),
  file = file.path(q3Dir, 'q3e_rule.csv'),
  fileEncoding = 'utf8',
  row.names = F
)
write.csv(
 helpers.select_rules_quality(e_rules_no_noneStyle),
  file = file.path(q3Dir, 'q3e_rule_NNS.csv'),
  fileEncoding = 'utf8',
  row.names = F
)
write.csv(
 helpers.select_rules_quality(e_rules_filter),
  file = file.path(q3Dir, 'q3e_rule_pairFilter.csv'),
  fileEncoding = 'utf8',
  row.names = F
)
write.csv(
 helpers.select_rules_quality(e_rulesNSN_filter),
  file = file.path(q3Dir, 'q3e_rule_NNS_filtered.csv'),
  fileEncoding = 'utf8',
  row.names = F
)

### everything

all_data_rules <- ruleGen.apriori(q3DataList$q3Data)
all_data_rules <- subset(all_data_rules, confidence >= 0.49)

all_data_rule_noStyleNone <- ruleGen.apriori(styleFiltered)
all_data_rule_noStyleNone <-
  subset(all_data_rule_noStyleNone, confidence >= 0.49)

write.csv(
 helpers.select_rules_quality(all_data_rules),
  file = file.path(q3Dir, 'q3All_rule.csv'),
  fileEncoding = 'utf8',
  row.names = F
)
write.csv(
 helpers.select_rules_quality(all_data_rule_noStyleNone),
  file = file.path(q3Dir, 'q3All_rule_NNS.csv'),
  fileEncoding = 'utf8',
  row.names = F
)

unique_features <- helpers.unqiue_all(q3DataList$q3Data)
```

Listing 5: Task iii R script

```
library(caret)
library(arules)
library(C50)
library(Rsenal)
library(rpart)


ruleGen.c50 <- function(d,
                        form,
                        trialNum = 4,
                        winnow = FALSE) {
  C5.0(
    formula = form,
    data = d,
    trials = trialNum,
    control = C5.0Control(subset = TRUE, winnow = winnow),
    rules = TRUE
  )
}

ruleGen.apriori <-
  function(data, minLen = 2, maxLen = 15,
           sup = 0.005, conf = 0.005, targ = 'rules',
           rH = NULL, lH = NULL, deflt = 'lhs',list=F) {
    appear <- NULL
    lhNull <- is.null(lH)
    rhNull <- is.null(rH)
    if (!lhNull && !rhNull) {
      appear <- list(rhs = rH,
                     lhs = lH,
                     default = deflt)
    } else if (!lhNull && rhNull) {
      appear <- list(lhs = lH,
                     default = deflt)
    } else if (lhNull && !rhNull) {
      appear <- list(rhs = rH,
                     default = deflt)
    } else {
      appear <- list(default = 'both')
    }

    rules <- apriori(
      data,
      parameter = list(
        minlen = minLen,
        maxlen = maxLen,
        supp = sup,
        confidence = conf,
        target = targ
      ),
      appearance = appear,
      control = list(verbose = F, filter = 0)
    )

    if(length(rules) > 0) {
      rules2df(addRuleQuality(
        trans = data,
        rules = rules,
        exclude =  c(
          "hyperConfidence",
          "cosine",
          "chiSquare",
          "coverage",
          "doc",
          "gini",
          "hyperLift",
          "fishersExactTest",
          "improvement",
          "leverage",
          "oddsRatio",
          "phi",
          "RLD"
```

```
        )
      ), list = list)
    } else {
      'no rules'
    }

}

ruleGen.part <- function(data, form) {
  train(form, data = data, method = "PART")
}

ruleGen.appearance_pair_list <- function(df,notFromHelper=F) {
  # browser()
  if(notFromHelper){
    unlist(c(levels(unique(df$s1)), levels(unique(df$s2))),use.names=F)
  } else {
    unlist(c(unique(df$s1), unique(df$s2)),use.names=F)
    # isS1_list = is.list(df$s1)
    # isS2_list = is.list(df$s2)
    #
    # if(isS1_list && isS2_list) {
    #   unlist(c(unique(df$s1), unique(df$s2)),use.names=F)
    # } else if(!isS1_list && isS2_list){
    #   unlist(c(df$s1, unique(df$s2)),use.names=F)
    # } else {
    #   unlist(c(unique(df$s1), df$s2),use.names=F)
    # }
  }
}

ruleGen.rules_apriori_lh <- function(data,feature_pair,notFromHelper=F,sup=0.0005, conf = 0.0005,minLen =
    2, maxLen = 15, list=F)  {
  rlh <- ruleGen.appearance_pair_list(feature_pair, notFromHelper = notFromHelper)
  ruleGen.apriori(
    data,
    lH = rlh,
    sup = sup,
    conf = conf,
    minLen = minLen,
    maxLen = maxLen,
    deflt = 'rhs',
    list=list
  )
}

ruleGen.rules_apriori_lhGrouped <- function(data,lhs_grouped,notFromHelper=F,sup=0.0005, conf = 0.0005,
    minLen = 2, maxLen = 15, list=F)  {

  ruleGen.apriori(
    data,
    lH = rlh,
    sup = sup,
    conf = conf,
    minLen = minLen,
    maxLen = maxLen,
    deflt = 'rhs',
    list=list
  )
}

ruleGen.rules_to_df <- function(rules,list=F){
  rules2df(rules,list=list)
}
```

Listing 6: Rule functions

```
library(dplyr)
library(tidyr)
library(purrr)
library(stringr)
library(reshape2)
library(foreach)
library(iterators)

```

```
setwd(getwd())

helpers.filter_df <- function(data,...) {
  data %>% filter(...)
}


helpers.select_rules_quality <- function(data) {
  data %>% select(rules,support,confidence,lift,conviction)
}

helpers.orderConfidence <- function(data) {
  data[with(data,order(confidence,decreasing = T)),]
}

helpers.unqiue_all <- function(data) {

  ucuisine <- data %>% select(cuisine) %>% distinct
  uatmosphere <- data %>% select(atmosphere) %>% distinct
  uoccasion <- data %>% select(occasion) %>% distinct
  uprice <- data %>% select(price) %>% distinct
  ustyle <- data %>% select(style) %>% distinct

  list(
   c = ucuisine,
   a = uatmosphere,
   o = uoccasion,
   p = uprice,
   s = ustyle
  )
}

helpers.test <- function(d,sides,...) {
  # it <- sides %>% by_row(function(r)
  #   c(str_trim(r$s1, side = c(
  #     "both", "left", "right"
  #   )), str_trim(r$s2, side = c(
  #     "both", "left", "right"
  #   ))), .to = 'rdf')
  #
   foreach(r = iter(sides,by='row'),.combine = c) %do% {
    print(d %>% select(cuisine == r$s1 & atmosphere == r$s2))
    pair <- c(r$s1,r$s2)
    1
  }
  # View(it)
}




helpers.s1_to_groupedS2 <- function(d,sides,rulesFun) {
  sides %>% group_by(s1) %>% summarise(s2 = list(s2)) %>%
    by_row(function(r) unlist(list(r$s1,r$s2)), .to='lhs') %>%
    by_row(function(r)  do.call(rulesFun,list(data=d,lH = r$lhs, deflt = 'rhs' )), .to='rdf')
}


helpers.s1_s2_pairs <- function(sides) {
  sides %>% mutate(lhs1 = paste(s1,s2,sep=','),lhs2 = paste(s2,s1,sep=','))
}


helpers.pair_count <- function(df) {
  df %>% summarise()
}

helpers.q3 <- function(write = F, filter = NULL) {
  data <-
    read.csv(file.path('q3','restaurantsq3.csv'))

  filter <- !is.null(filter)

  cuisine_atmosphere <-
    data %>% select(cuisine, atmosphere) %>% distinct %>%
    filter(cuisine != 'none' & atmosphere != 'none') %>%
```

```
    transmute(
      s1 = paste('cuisine', cuisine, sep = '='),
      s2 = paste('atmosphere', atmosphere, sep = '=')
    )

  cuisine_occasion <-
    data %>% select(cuisine, occasion) %>% distinct %>%
    filter(cuisine != 'none' & occasion != 'none') %>%
    transmute(
      s1 = paste('cuisine', cuisine, sep = '='),
      s2 = paste('occasion', occasion, sep = '=')
    )

  cuisine_price <-
    data %>% select(cuisine, price) %>% distinct  %>%
    filter(cuisine != 'none' & price != 'none') %>%
    transmute(s1 = paste('cuisine', cuisine, sep = '='),
              s2 = paste('price', price, sep = '='))

  cuisine_style <-
    data %>% select(cuisine, style) %>% distinct  %>%
    filter(cuisine != 'none' & style != 'none') %>%
    transmute(s1 = paste('cuisine', cuisine, sep = '='),
              s2 = paste('style', style, sep = '='))

  atmosphere_occasion <-
    data %>% select(atmosphere, occasion) %>% distinct  %>%
    filter(atmosphere != 'none' & occasion != 'none') %>%
    transmute(
      s1 = paste('atmosphere', atmosphere, sep = '='),
      s2 = paste('occasion', occasion, sep = '=')
    )

  atmosphere_price <-
    data %>% select(atmosphere, price) %>% distinct  %>%
    filter(atmosphere != 'none' & price != 'none') %>%
    transmute(
      s1 = paste('atmosphere', atmosphere, sep = '='),
      s2 = paste('price', price, sep = '=')
    )

  atmosphere_style <-
    data %>% select(atmosphere, style) %>% distinct  %>%
    filter(atmosphere != 'none' & style != 'none') %>%
    transmute(
      s1 = paste('atmosphere', atmosphere, sep = '='),
      s2 = paste('style', style, sep = '=')
    )

  occasion_price <-
    data %>% select(occasion, price) %>% distinct  %>%
    filter(occasion != 'none' & price != 'none') %>%
    transmute(
      s1 = paste('occasion', occasion, sep = '='),
      s2 = paste('price', price, sep = '=')
    )

  occasion_style <-
    data %>% select(occasion, style) %>% distinct  %>%
    filter(occasion != 'none' & style != 'none') %>%
    transmute(
      s1 = paste('occasion', occasion, sep = '='),
      s2 = paste('style', style, sep = '=')
    )

  style_price <-
    data %>% select(style, price) %>% distinct  %>%
    filter(style != 'none' & price != 'none') %>%
    transmute(s1 = paste('style', style, sep = '='),
              s2 = paste('price', price, sep = '='))

  decor_price <-
    data %>% select(atmosphere, price) %>% distinct  %>%
    filter(atmosphere != 'none' &
             price != 'none' & grepl('Decor', atmosphere)) %>%
    transmute(
```

```
      s1 = paste('atmosphere', atmosphere, sep = '='),
      s2 = paste('price', price, sep = '=')
    )

  if (write) {
    dataForR <- file.path('..', 'r_data')
    dir.create(dataForR, showWarnings = F)

    write.csv(
      cuisine_atmosphere,
      file = file.path(dataForR, 'q3_cuisine_atmosphere.csv'),
      fileEncoding = 'utf8',
      row.names = F
    )

    write.csv(
      cuisine_occasion,
      file = file.path(dataForR, 'q3_cuisine_occasion.csv'),
      fileEncoding = 'utf8',
      row.names = F
    )

    write.csv(
      cuisine_price,
      file = file.path(dataForR, 'q3_cuisine_price.csv'),
      fileEncoding = 'utf8',
      row.names = F
    )

    write.csv(
      cuisine_style,
      file = file.path(dataForR, 'q3_cuisine_style.csv'),
      fileEncoding = 'utf8',
      row.names = F
    )

    write.csv(
      atmosphere_occasion,
      file = file.path(dataForR, 'q3_atmosphere_occasion.csv'),
      fileEncoding = 'utf8',
      row.names = F
    )

    write.csv(
      atmosphere_price,
      file = file.path(dataForR, 'q3_atmosphere_price.csv'),
      fileEncoding = 'utf8',
      row.names = F
    )

    write.csv(
      atmosphere_style,
      file = file.path(dataForR, 'q3_atmosphere_style.csv'),
      fileEncoding = 'utf8',
      row.names = F
    )

    write.csv(
      occasion_price,
      file = file.path(dataForR, 'q3_occasion_price.csv'),
      fileEncoding = 'utf8',
      row.names = F
    )

    write.csv(
      occasion_style,
      file = file.path(dataForR, 'q3_occasion_style.csv'),
      fileEncoding = 'utf8',
      row.names = F
    )

    write.csv(
      style_price,
      file = file.path(dataForR, 'q3_style_price.csv'),
      fileEncoding = 'utf8',
      row.names = F
```

```
    )

    write.csv(
      decor_price,
      file = file.path(dataForR, 'q3_decor_price.csv'),
      fileEncoding = 'utf8',
      row.names = F
    )
  }

  list(
    q3Data = data,
    ca = cuisine_atmosphere,
    co = cuisine_occasion,
    cp = cuisine_price,
    cs = cuisine_style,
    ao = atmosphere_occasion,
    ap = atmosphere_price,
    as = atmosphere_style,
    op = occasion_price,
    os = occasion_style,
    sp = style_price,
    dp = decor_price
  )
}



helpers.q4 <- function(write = F) {
  data <- read.csv(file.path('q3', 'restaurantsq3.csv'))

  cuisine_foodQuality <-
    data %>% select(cuisine, atmosphere) %>%
    filter(grepl('Food', atmosphere))

  cuisine_foodQualitySide <-
    cuisine_foodQuality %>% distinct %>%
    transmute(
      s1 = paste('cuisine', cuisine, sep = '='),
      s2 = paste('atmosphere', atmosphere, sep = '=')
    )

  cuisine_foodQuality <- cuisine_foodQuality %>% transmute(cuisine=cuisine ,quality = atmosphere)

  cuisine_foodQuality <- droplevels(cuisine_foodQuality)
  cq_spread <- cuisine_foodQuality %>% group_by(cuisine,quality) %>% tally %>% spread(quality,n,fill = 0)

  cuisine_foodQualityD = cuisine_foodQuality
  cuisine_foodQualityD$quality <-
    recode_factor(
      cuisine_foodQualityD$quality,
      `Excellent Food` = "Good",
      `Near-perfect Food` = "Excellent",
      `Extraordinary Food` = "Excellent",
      `Fair Food` = 'Fair',
      `Good Food` = 'Good'
    )

  cqD_spread <- cuisine_foodQualityD %>% group_by(cuisine,quality) %>% tally %>% spread(quality,n,fill =
      0)


  # cuisine_foodQuality$cuisine <- levels(cuisine_foodQuality$cuisine)
  # cuisine_foodQuality$quality <- levels(cuisine_foodQuality$quality)

  list(q4Data = data,
       cfqSide = cuisine_foodQualitySide,
       cfq = cuisine_foodQuality,
       cfqd = cuisine_foodQualityD,
       cqD_spread = cqD_spread,
       cq_spread=cq_spread)
}


helpers.pair_count <- function(df) {
  ldf <- df %>% transmute(
```

```
    cuisine = levels(unique(cuisine)),
    atmosphere = levels(unique(atmosphere)),
    occasion = levels(unique(occasion)),
    price = levels(unique(price)),
    style = levels(unique(style))
  )

  list(
      cuisine_atmosphere = df %>% select(cuisine, atmosphere) %>% group_by(cuisine, atmosphere) %>% tally(
          sort = TRUE),
      cuisine_occasion = df %>% select(cuisine, occasion) %>% group_by(cuisine, occasion) %>% tally(sort =
          TRUE),
      cuisine_price = df %>% select(cuisine, price) %>% group_by(cuisine, price) %>% tally(sort = TRUE),
      cuisine_style = df %>% select(cuisine, atmosphere) %>%  group_by(cuisine, atmosphere) %>% tally(sort
          = TRUE),
      atmosphere_occasion = df %>% select(cuisine, style) %>% group_by(cuisine, style) %>% tally(sort =
          TRUE),
      atmosphere_price = df %>% select(atmosphere, price) %>% group_by(atmosphere, price) %>% tally(sort =
          TRUE),
      atmosphere_style = df %>% select(atmosphere, style) %>% group_by(atmosphere, style) %>% tally(sort =
          TRUE),
      occasion_price = df %>% select(occasion, price) %>% group_by(occasion, price) %>% tally(sort = TRUE)
          ,
      occasion_style = df %>% select(occasion, style) %>% group_by(occasion, style) %>% tally(sort = TRUE)
          ,
      style_price = df %>% select(style, price) %>% group_by(style, price) %>% tally(sort = TRUE)
  )
}

helpers.q2_appearance <- function() {
  cuisine_appearance <-
    data %>% distinct(cuisine) %>% transmute(side = paste('cuisine', cuisine, sep = '='))

  atmosphere_appearance <-
    data %>% distinct(atmosphere) %>% transmute(side = paste('atmosphere', atmosphere, sep = '='))

  occasion_appearance <-
    data %>% distinct(occasion) %>% transmute(side = paste('occasion', occasion, sep = '='))

  price_appearance <-
    data %>% distinct(price) %>% transmute(side = paste('price', price, sep = '='))

  style_appearance <-
    data %>% distinct(style) %>% transmute(side = paste('style', style, sep = '='))

  write.csv(
    cuisine_appearance,
    file = file.path(dataForR, 'q2_cuisine_ap.csv'),
    fileEncoding = 'utf8'
  )
  write.csv(
    atmosphere_appearance,
    file = file.path(dataForR, 'q2_atmosphere_ap.csv'),
    fileEncoding = 'utf8'
  )
  write.csv(
    occasion_appearance,
    file = file.path(dataForR, 'q2_occasion_ap.csv'),
    fileEncoding = 'utf8'
  )
  write.csv(
    price_appearance,
    file = file.path(dataForR, 'q2_price_ap.csv'),
    fileEncoding = 'utf8'
  )
  write.csv(
    style_appearance,
    file = file.path(dataForR, 'q2_style_ap.csv'),
    fileEncoding = 'utf8'
  )
}

helpers.group_by_lhs <- function(ruleDF) {
  ruleDF %>% select(lhs, rhs, support, confidence, lift, conviction) %>% group_by(lhs) %>% summarise(
    associated = list(rhs),
    cumSup = sum(support),
```

```
    cumConf = sum(confidence),
    cumLift = sum(lift),
    sumConv = sum(conviction)
  )
}

# from http://www.r-bloggers.com/measuring-associations-between-non-numeric-variables/
helpers.GKtau <- function(x,y){
  #
  #  First, compute the IxJ contingency table between x and y
  #
  Nij <- table(x,y,useNA='ifany')
  #
  #  Next, convert this table into a joint probability estimate
  #
  PIij <- Nij/sum(Nij)
  #
  #  Compute the marginal probability estimates
  #
  PIiPlus = apply(PIij,MARGIN=1,sum)
  PIPlusj = apply(PIij,MARGIN=2,sum)
  #
  #  Compute the marginal variation of y
  #
  Vy <- 1 - sum(PIPlusj^2)
  #
  #  Compute the expected conditional variation of y given x
  #
  InnerSum <- apply(PIij^2,MARGIN=1,sum)
  VyBarx <-  1 - sum(InnerSum/PIiPlus)
  #
  #  Compute and return Goodman and Kruskal s tau measure
  #
  tau <- (Vy - VyBarx)/Vy
  tau
}
```

Listing 7: Helper functions

Task iii Rules

Table 14: Cuisine and atmosphere

Normal

{cuisine=Pacific Rim} =>{style=none}
{cuisine=Austrian} =>{price=$15-$30}
{cuisine=Burritos} =>{price=below $15}
{cuisine=Ethiopian} =>{style=none}
{cuisine=Dim Sum,atmosphere=Extraordinary Food} =>{style=none}
{cuisine=Greek,atmosphere=Parking/Valet} =>{price=$15-$30}
{cuisine=Mexican,atmosphere=Great for People Watching} =>{price=$15-$30}
{cuisine=Coffee and Dessert,atmosphere=For the Young and Young at Heart} =>{price=$15-$30}
{cuisine=Creole,atmosphere=Excellent Decor} =>{price=$15-$30}
{cuisine=Mexican,atmosphere=Hip Place To Be} =>{price=$15-$30}
{cuisine=Seafood,atmosphere=Singles Scene} =>{price=$15-$30}
{cuisine=Asian} =>{style=none}
{cuisine=Thai,atmosphere=Extraordinary Food} =>{price=$15-$30}
{atmosphere=Poor Decor} =>{price=below $15}
{cuisine=Greek,atmosphere=Good Decor} =>{price=$15-$30}
{cuisine=Wine and Beer,atmosphere=Fair Decor} =>{price=below $15}
{cuisine=Japanese,atmosphere=Good Decor} =>{style=none}
{cuisine=Cajun,atmosphere=Excellent Decor} =>{price=$15-$30}
{atmosphere=Fair Food} =>{style=none}
{cuisine=Thai,atmosphere=Excellent Decor} =>{price=$15-$30}
{cuisine=Seafood,atmosphere=Cafe/Garden Dining} =>{price=$15-$30}
{cuisine=Italian,atmosphere=Little Known But Well Liked} =>{style=none}
{cuisine=Chinese,atmosphere=For the Young and Young at Heart} =>{style=none}
{cuisine=Korean} =>{style=none}
{cuisine=Cuban} =>{style=none}
{cuisine=Spanish,atmosphere=Wheelchair Access} =>{price=$15-$30}
{cuisine=Wine and Beer,atmosphere=Fabulous Wine Lists} =>{price=$30-$50}
{cuisine=Indian,atmosphere=Excellent Decor} =>{style=none}
{cuisine=Fountain and Ice Cream,atmosphere=Fabulous Wine Lists} =>{price=$30-$50}
{cuisine=Spanish,atmosphere=Parking/Valet} =>{price=$15-$30}
{cuisine=Greek} =>{price=$15-$30}
{cuisine=Indian,atmosphere=Parking/Valet} =>{style=none}
{cuisine=Japanese,atmosphere=Excellent Food} =>{style=none}
{cuisine=Indian,atmosphere=Good Decor} =>{style=none}
{cuisine=Japanese,atmosphere=Wheelchair Access} =>{price=$15-$30}
{cuisine=German,atmosphere=Excellent Food} =>{price=$15-$30}
{cuisine=Caribbean,atmosphere=Quirky} =>{style=none}
{cuisine=Seafood,atmosphere=Place for Singles} =>{price=$15-$30}
{cuisine=Japanese,atmosphere=Excellent Service} =>{style=none}
{cuisine=Italian,atmosphere=Up and Coming} =>{price=$15-$30}
{cuisine=Thai,atmosphere=Excellent Decor} =>{style=none}
{cuisine=Hamburgers & Beer,atmosphere=Good Service} =>{price=$15-$30}
{cuisine=Indian,atmosphere=Excellent Food} =>{style=none}
{cuisine=Mexican,atmosphere=Quirky} =>{style=none}
{cuisine=Seafood,atmosphere=Quirky} =>{style=none}
{cuisine=Mexican,atmosphere=Extraordinary Food} =>{style=none}
{cuisine=American,atmosphere=Cafe/Garden Dining} =>{style=none}
{cuisine=Japanese} =>{style=none}
{cuisine=Seafood,atmosphere=No Reservations} =>{price=$15-$30}
{cuisine=Thai,atmosphere=Wheelchair Access} =>{style=none}
{cuisine=Indian} =>{style=none}
{cuisine=Pacific Rim} =>{price=$15-$30}
{cuisine=Burritos} =>{style=Tacos}
{cuisine=Afghanistan} =>{price=$15-$30}
{cuisine=Fountain and Ice Cream,atmosphere=Credit cards are not accepted} =>{price=below $15}
{cuisine=Hamburgers & Beer,atmosphere=Good Decor} =>{price=$15-$30}
{cuisine=Greek,atmosphere=Excellent Decor} =>{price=$15-$30}
{cuisine=Swiss} =>{style=none}
{cuisine=French,atmosphere=Good Food} =>{price=$15-$30}
{cuisine=Thai,atmosphere=Extraordinary Food} =>{style=none}
{cuisine=German,atmosphere=Excellent Service} =>{price=$15-$30}
{cuisine=Dim Sum,atmosphere=Excellent Service} =>{style=none}
{cuisine=Vietnamese,atmosphere=Excellent Service} =>{style=none}
{cuisine=German,atmosphere=Excellent Decor} =>{price=$15-$30}
{cuisine=Indian,atmosphere=Excellent Service} =>{style=none}
{cuisine=Fountain and Ice Cream,atmosphere=Quiet for Conversation} =>{price=$30-$50}
{cuisine=Bar-B-Q,atmosphere=Excellent Decor} =>{price=$15-$30}
{cuisine=Fountain and Ice Cream,atmosphere=Good Out of Town Business} =>{price=$30-$50}
{cuisine=German} =>{price=$15-$30}
{cuisine=Seafood,atmosphere=Good Food} =>{price=$15-$30}
{cuisine=Chinese,atmosphere=Excellent Decor} =>{style=none}
{cuisine=Greek,atmosphere=Excellent Service} =>{price=$15-$30}
{cuisine=Italian,atmosphere=Place for Singles} =>{price=$15-$30}
{cuisine=Chinese,atmosphere=Excellent Service} =>{style=none}
{cuisine=Fountain and Ice Cream,atmosphere=Extraordinary Decor} =>{price=$30-$50}
{cuisine=Indian,atmosphere=Wheelchair Access} =>{style=none}
{cuisine=Spanish,atmosphere=Good Decor} =>{price=$15-$30}
{atmosphere=Fair Service} =>{price=below $15}
{cuisine=Ethiopian} =>{price=$15-$30}
{cuisine=Indian,atmosphere=Good Decor} =>{price=$15-$30}
{cuisine=Chinese,atmosphere=Wheelchair Access} =>{style=none}
{cuisine=Seafood,atmosphere=Good Decor} =>{price=$15-$30}
{cuisine=Vietnamese} =>{style=none}
{cuisine=Indian,atmosphere=Extraordinary Food} =>{style=none}
{cuisine=Japanese,atmosphere=Good Decor} =>{price=$15-$30}
{cuisine=Seafood,atmosphere=For the Young and Young at Heart} =>{price=$15-$30}
{cuisine=American,atmosphere=Fair Decor} =>{price=below $15}
{cuisine=Japanese,atmosphere=Excellent Decor} =>{style=none}
{cuisine=Japanese,atmosphere=Extraordinary Food} =>{style=none}
{cuisine=Mexican,atmosphere=Fair Decor} =>{price=below $15}
{cuisine=Caribbean,atmosphere=Good Service} =>{price=$15-$30}
{cuisine=Mexican,atmosphere=Warm spots by the fire} =>{price=$15-$30}
{cuisine=Caribbean,atmosphere=Good Decor} =>{style=none}
{cuisine=Lebanese} =>{style=Middle Eastern}
{cuisine=Dim Sum} =>{style=none}
{cuisine=Japanese,atmosphere=Excellent Food} =>{price=$15-$30}
{cuisine=Caribbean,atmosphere=Excellent Food} =>{style=none}
{cuisine=Chinese,atmosphere=Good Decor} =>{style=none}
{cuisine=Italian,atmosphere=Cafe/Garden Dining} =>{price=$15-$30}
{cuisine=American,atmosphere=Near-perfect Decor} =>{price=$30-$50}
{cuisine=Lebanese} =>{price=$15-$30}
{cuisine=Italian,atmosphere=Good Food} =>{price=$15-$30}

Normal