Mouna Mokaddem

Challenge Report

Question 1

I implemented a function called **emptyCountry** that prints all the searches that have the value for the field country an empty field and the total number of such searches which is 2820. Below I show a caption of my function's output.
Looking at the output, I didn't find any signal in it or any dependance or commun points between the entries. Furthermore, there isn't any sign that shows that it is more likely to be Canada than Mexico for example.
I noticed a field that figure in most of the entries of an empty field country which is called _row but it does not seem to be a relevant field, besides it does not give any further information to guess the missing country.

```
def emptyCountry():
    l = []
    tot = 0
    for i in range(len(data)):
        if data[i]["user"][0][0]["country"] == "":
            tot += 1
            l.append(data[i])

    pprint(l)
    print '\n',tot
```

```
[{u'cities': [u'Montreal QC, Chicago IL'],
  u'session_id': [u'YOVUIM79SGS5Y'],
  u'unix_timestamp': [1443171887],
  u'user': [[{u'country': u'',
              u'joining_date': u'2015-03-02',
              u'user_id': 8831}]]},
 {u'cities': [u'Calgary AB, New York NY'],
  u'session_id': [u'SHXEY67QWKP9K'],
  u'unix_timestamp': [1431766104],
  u'user': [[{u'country': u'',
              u'joining_date': u'2015-03-16',
              u'user_id': 587}]]},
 {u'cities': [u'New York NY'],
  u'session_id': [u'UY49RTVRX3GL7'],
  u'unix_timestamp': [1439104667],
  u'user': [[{u'country': u'',
              u'joining_date': u'2015-03-21',
              u'user_id': 5995}]]},
 {u'cities': [u'Toronto ON, New York NY'],
  u'session_id': [u'YQAEPPG214CNU'],
  u'unix_timestamp': [1439713760],
  u'user': [[{u'country': u'',
              u'joining_date': u'2015-03-30',
              u'user_id': 957}]]},
 {u'cities': [u'New York NY'],
  u'session_id': [u'51H4C2DEYXG5D'],
  u'unix_timestamp': [1436444694],
  u'user': [[{u'country': u'',
              u'joining_date': u'2015-03-05',
              u'user_id': 5433}]]},
 {u'cities': [u'New York NY'],
  u'session_id': [u'IZMJ8EDM6YA9U'],
  u'unix_timestamp': [1434954513],
  u'user': [[{u'country': u'',
              u'joining_date': u'2015-03-14',
              u'user_id': 6236}]]},
 {u'cities': [u'Toronto ON, New York NY'],
  u'session_id': [u'GZ121D8QYYVTN'],
  u'unix_timestamp': [1425603472],
  u'user': [[{u'country': u'',
              u'joining_date': u'2015-03-05',
```

Questions 2

Method:

For this question, what I do is given an input city my method will predict the most likely next city to be searched for based on all sessions. That means, for an input city, retrive neighboring cities (i.e., cities that come with the input city in any session search) and calculate their frequencies; how often they come together with the input city based on all sessions of the json file. Then the most frequent one is identified as the most likely next city to be searched for, for the input city.
This could be viewed also in terms of probability by calculating the total number of frenquencies of the neighboring cities and then divide the frenquence of each city by this number so it will give the probability of the next city.
My choice of the method was based on the fact that doing statistics on all sessions comes as a very natural choice for a basic predicator.

Implementation:

I implemented a function called **nextCity** that takes as a parameter a city (a string) and returns a list of the most likely next cities (the list could contain one or more cities).
I implemented also a function called **nextCityforAllCities** as I wanted to see if there is a city that tend to be the most likely next city more than others.
Here I show two captions describing the output of both aforementioned functions. The first one being for **nextCity** whereas the second one for **nextCityforAllCities**.

In [21]:
```
def nextCity(city):
    city_dict = dict()
    total_cities = 0
    for i in range(len(data)):
        cities = data[i]["cities"][0].split(', ')
        if city in cities:
            for c in cities:
                if c != city:
                    total_cities += 1
                    if c in city_dict:
                        city_dict[c][0] += 1
                    else:

#     print total_cities
    cities_prob = []
    for values in city_dict.values():
        city_prob = round(float(values[0])/total_cities,5)
        values.append(city_prob)
        cities_prob.append(city_prob)
# print statics (probabilities of all next cities)
#     for keys,values in city_dict.items():
#         print(keys)
#         print(values)

    m = max(cities_prob)
    Bool = True
    next_city_list = []
    for keys,values in city_dict.items():
        if values[1] == m:

#             if Bool == True:
#                 print '\nThe most likely next city of'+' '+city+' '+'is'+'...'+'\n'+'\n', keys
#                 Bool = False
#             else:
#                 print '\n',keys
            next_city_list.append(keys)
    return next_city_list

print nextCity('San Antonio TX')
print nextCity('Philadelphia PA')
print nextCity('Vancouver BC')
```

```
[u'Austin TX', u'Houston TX']
[u'New York NY']
[u'Victoria BC']
```

```
# return the most likely next city of all cities of json file and also calculate the probability of having next city
def nextCityforAllCities():
    citiesDict = dict()
    NewYorknumber = 0
    for i in range(len(data)):
        cities = data[i]["cities"][0].split(', ')
        for city in cities:
            if not (city in citiesDict):
                nextC = nextCity(city)
                citiesDict[city] = nextC
                if 'New York NY' in nextC:
                    NewYorknumber += 1
    stat = float(NewYorknumber)/len(citiesDict)
    return citiesDict, stat

pprint(nextCityforAllCities())
```

```
({u'Anaheim CA': [u'Los Angeles CA'],
  u'Arlington TX': [u'Dallas TX'],
  u'Atlanta GA': [u'Jacksonville FL'],
  u'Austin TX': [u'Houston TX'],
  u'Bakersfield CA': [u'Los Angeles CA'],
  u'Baltimore MD': [u'New York NY'],
  u'Birmingham AL': [u'Atlanta GA', u'Jacksonville FL'],
  u'Boston MA': [u'New York NY'],
  u'Buffalo NY': [u'Toronto ON'],
  u'Calgary AB': [u'Vancouver BC'],
  u'Chandler AZ': [u'Phoenix AZ'],
  u'Charlotte NC': [u'Atlanta GA', u'Raleigh NC', u'Jacksonville FL'],
  u'Chesapeake VA': [u'New York NY'],
  u'Chicago IL': [u'New York NY'],
  u'Cincinnati OH': [u'Indianapolis IN'],
  u'Cleveland OH': [u'Detroit MI'],
  u'Columbus OH': [u'Cincinnati OH'],
  u'Corpus Christi TX': [u'San Antonio TX'],
  u'Dallas TX': [u'Plano TX'],
  u'Detroit MI': [u'Windsor ON'],
  u'Edmonton AB': [u'Calgary AB'],
  u'Fort Wayne IN': [u'Chicago IL'],
  u'Fort Worth TX': [u'Dallas TX'],
  u'Fresno CA': [u'San Jose CA'],
  u'Glendale AZ': [u'Phoenix AZ'],
  u'Greensboro NC': [u'New York NY',
                     u'Atlanta GA',
                     u'Norfolk VA',
                     u'Montreal QC',
                     u'Baltimore MD',
                     u'Raleigh NC',
                     u'Jersey City NJ',
                     u'Boston MA',
                     u'WASHINGTON DC',
```

Observations:

According to the output given by my function **nextCityforAllCities**, New York NY seems to figure more likely than other cities as the predicted next city for many cities with a probability of approximately 0.18. However, the previous fact is just an observation and can not make us conclude anything as there are as well many other cities that do not have New York NY as their predicted next city.

Question 3

Method:

Before answering this question, I will give a brief explanation of my interpretation of it.

This question is about the effect of the use of two user's features (country/joining_date) on the predicator's performance. In other words, in the previous question (i.e., question 2) the predicator was

based on the history, that means given the history of search (i.e., all searches' sessions), what is the most likely next city of the input city. In this question the main goal is to see whether using the feature country or joining_date will be useful to predict the most likely city to be searched. In other words, if we take the example of the feature country, the question will be given the country from where a user logged, what is the most likely city to be searched in contrast with the question 2 which says given the history search of all users what is the most likely city to be searched.

The method designed for this questions is as follows: for each country, retrieve all cities that were searched and calculate their frequency. The most frequent city, within sessions of users logged from the same country, will be the one that will be predicted as the most likely city to be searched. In terms of probability, one could the same thing as question 2 by summing frequencies of all cities within the same country and dividing the frequency of each city by this number.

Implementation:

I implemented a function called **likelyCity** that takes as parameter a field, whether country or joining_date and prints each country and its corresponding predicted city.
Below I will show two captions one for the execution of the function with country and the second one with joining_date.

In [26]:
```
# This function takes as a parameter a field (it can be country or joining_date) and predict the most likely city to
def likelyCity(field):
#     dictionary containing for each logged country the list of cities searched for and their frequencies
    countryDict = dict()
    for i in range(len(data)):
        f = data[i]["user"][0][0][field]
        if not (f in countryDict):
            countryDict[f] = dict()
        cities = data[i]["cities"][0].split(', ')
        for city in cities:
            if not(city in countryDict[f]):
                countryDict[f][city] = 1
            else:
                countryDict[f][city] += 1
#     pprint(countryDict)
    l = dict()
    for c in countryDict:
        m = max(countryDict[c].values())
        for keys, values in countryDict[c].items():
            if values == m:
                l[c] = keys
    print '\n'
    pprint(l)


likelyCity("country")
# likelyCity("joining_date")
```

```
{u'': u'New York NY',
 u'DE': u'New York NY',
 u'ES': u'New York NY',
 u'FR': u'New York NY',
 u'IT': u'New York NY',
 u'UK': u'New York NY',
 u'US': u'New York NY'}
```

In [28]:
```
# This function takes as a parameter a field (it can be country or joining_date) and predict the most likely city to
def likelyCity(field):
#     dictionary containing for each logged country the list of cities searched for and their frequencies
    countryDict = dict()
    for i in range(len(data)):
        f = data[i]["user"][0][0][field]
        if not (f in countryDict):
            countryDict[f] = dict()
        cities = data[i]["cities"][0].split(', ')
        for city in cities:
            if not(city in countryDict[f]):
                countryDict[f][city] = 1
            else:
                countryDict[f][city] += 1
#     pprint(countryDict)
    l = dict()
    for c in countryDict:
        m = max(countryDict[c].values())
        for keys, values in countryDict[c].items():
            if values == m:
                l[c] = keys
#     print '\n'
    pprint(l)


# likelyCity("country")
likelyCity("joining_date")
```

```
{u'2015-02-28': u'New York NY',
 u'2015-03-01': u'New York NY',
 u'2015-03-02': u'New York NY',
 u'2015-03-03': u'New York NY',
 u'2015-03-04': u'New York NY',
 u'2015-03-05': u'New York NY',
 u'2015-03-06': u'New York NY',
 u'2015-03-07': u'New York NY',
 u'2015-03-08': u'New York NY',
 u'2015-03-09': u'New York NY',
 u'2015-03-10': u'New York NY',
 u'2015-03-11': u'New York NY',
 u'2015-03-12': u'New York NY',
 u'2015-03-13': u'New York NY',
 u'2015-03-14': u'New York NY',
 u'2015-03-15': u'New York NY',
 u'2015-03-16': u'New York NY',
 u'2015-03-17': u'New York NY',
 u'2015-03-18': u'New York NY',
 u'2015-03-19': u'New York NY',
 u'2015-03-20': u'New York NY',
 u'2015-03-21': u'New York NY',
 u'2015-03-22': u'New York NY',
```

Observations:

The output of the function I implemented was New York NY for all countries as well as for all joining dates. So given those features, the result is the same and is always New York NY. Therefore, I can conclude that both of the features (i.e., country and joining date) are not useful to predict the most likely city to be searched. And so, the predicator of question 2 which was based on history searches is more likely to give better information and thus better predictions.

Question 4

The prediction algorithm of question 3 is very bad because given on of the features does not give any information about the user since for all countries and joining_date, the city of New York NY is always predicted.
The prediction algorithm of question 2 although was not evaluated formally, but is very basic relying only on searches' history and does not distinguish the users in any sense. Therefore, one could imagine improving the predicition by using machine learning (an idea that I thought about from the begining) but many machine learning techniques for prediction, to give good results need that the input space

dimension be higher than the output space dimension. That means the space of features like here the user id, the user’s country, the session id, ...etc should be higher than the space of predictions which is the space of cities. Features that one could think about and that could be useful for prediction of the city is the age of the user, it is maritial status, has childran or not, ...etc. Those features can give more information about the vacation destination of a user and thus the most likely city to be searched.