This notebook is an example of applying getmove on real trajectories.

Installation

The best way is to use Python3 and a virtual environment.
Here are the main commands:

```
virtualenv -p python3 env
source env/bin/activate
pip freeze > requirements.txt
pip install jupyter
#useful extensions for jupyter
pip install jupyter_contrib_nbextensions
#Installation of extensions
pip install jupyter_nbextensions_configurator
jupyter-contrib-nbextension install --sys-prefix
#Activation of extensions
jupyter-nbextensions_configurator enable --sys-prefix
```

Download and install the following files:

GetMove is available here: https://github.com/jGetMove/jGetMove/releases (https://github.com/jGetMove/jGetMove/releases)
BaseMap is available here: https://github.com/matplotlib/basemap/releases/ (https://github.com/matplotlib/basemap/releases/)
A file example "migration_original.csv" is available here:
http://www.lirmm.fr/~poncelet/migration_original.csv (http://www.lirmm.fr/~poncelet/migration_original.csv)
They are Goetland trajectories between Africa and Northern Europe, which were collected between 2009 and 2015.

To install BaseMap

```
#Download the file from: https://github.com/matplotlib/basemap/releases/
#unzip the file or tar -xvf *.tar
#cd file.tar
cd geos-3.3.3
export GEOS_DIR=$(pwd)
./configure --prefix=$GEOS_DIR
make; make install
cd ..
python setup.py install
```

To test if the installation has been well done, open a python session and run the following line:

```
from mpl_toolkits.basemap import Basemap
#maybe you should have to install pyproj with
#pip install pyproj
```

To install GetMove

```
#Download the file from: https://github.com/jGetMove/jGetMove/releases
cd jGetMove-2.0.1
mkdir -p out/
javac -extdirs lib/ -sourcepath src/ src/fr/jgetmove/jgetmove/Main.java -d out/
jar cvfm jGetMove.jar Manifest.mf -C out/ . lib/
```

GetMove considers two files as input.
The first one is for each object the number of clusters.
It is organized as follows:
Example: data.dat

```
1 2 3
2 3
1 3
```

It means that the first object 0 belongs to the clusters 1, 2, 3. The object 1 belongs to the clusters 2 and 3. Finally the object 3 belongs to the clusters 1 and 3.
Note that the separator is a tabulation.

The second file specifies the time the clusters occur.
It is organized as follows:
Example: datatimeindex.dat

```
1   1
1   0
2   2
3   3
```

It means that the clusters 1 and 0 occur at time 1, the cluser 2 at time 2 and the cluster 3 at time 3.
Note that the separator is a tabulation.

To run Getmove:

```
java -jar jGetMove.jar assets/example.dat assets/exampletimeindex.dat -p 2 -s 1
-t 1
# help
java -jar jGetMove.jar --help
```

Installation of libraries:

```
pip install matplotlib
pip install numpy
pip install sklearn
pip install pandas
pip install pprint
pip install pyproj
pip install scipy
pip install image
```

At the end of the installation, the content of the directory must look like:

```
TrajectoriesNoteBook.ipynb  bin               jGetMove-2.0.1        pip-
selfcheck.json
basemap-1.1.0           etc               lib               share
basemap-1.1.0.tar       include            migration_original.csv
```

Running trajectories

In [ ]:

```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from mpl_toolkits.basemap import Basemap
import numpy as np
import datetime
from sklearn.cluster import DBSCAN
import pandas as pd
from matplotlib import interactive
import json
import os
from pprint import pprint
```

In [ ]:

```
#init some constants
#for DBSCAN
epsilon=0.3
mint=10
#for plotting figures
width=20
height=12
#Interval of time (e.g. 7D = 7 days)
Itime='2D'
```

In [ ]:

```
#reading the file
df = pd.read_csv(r"migration_original.csv")
```

Some descriptions on the data

In [ ]:

```
#Description of the data
print ("Number of lines and columns")
print(df.shape)
print ("\n The different attributes\n")
print (df.columns)


print("\n Number of places per birds\n")
print ('The first ten: \n')
print('\t',df.groupby('tag-local-identifier').size().head(10))
print ('\n The last ten')
print('\t',df.groupby('tag-local-identifier').size().tail(10))
print ('\n')
```

In [ ]:

```
print ('\n Keep only useful columns\n')
df=df[['event-id','timestamp','location-long',
       'location-lat','tag-local-identifier']]

#convert the date format
time_format = "%Y-%m-%d %H:%M:%S.%f"
#apply(lambda x : datetime.datetime.strptime(x,
#                                            time_format).strftime("%B"))
df["month"]=df["timestamp"].apply(lambda x :
                datetime.datetime.strptime(x,
                time_format).strftime("%B"))
df["year"] = df["timestamp"].apply(lambda x :
                datetime.datetime.strptime(x,
                time_format).year)

df["timestamp"] = pd.to_datetime(df['timestamp'])
df = df.sort_values(by='timestamp')

print("First", df.loc[[0]].timestamp.values[0])
print("Last", df.tail(1).timestamp.values[0])
print('\n Number of event per year')
print('\n\t',df.groupby('year').size().tail(10))
```

Let's plot the trajectories

In [ ]:

```
#plot the trajectories

gp = df.groupby("tag-local-identifier")

#Plot the map
plt.figure(figsize=(width, height))
map = Basemap(llcrnrlat=-5, urcrnrlat=68,
              llcrnrlon=-10, urcrnrlon=65,
              lat_ts=20, resolution='c')
map.shadedrelief()

#Select the colors
colors = cm.rainbow(np.linspace(0, 1, len(gp.indices)))

#Plot the trajectories
j=0
for i in gp.indices:
    grp = gp.get_group(i)
    map.plot(grp["location-long"], grp["location-lat"],
             zorder=2, color=colors[j])
    map.scatter(grp["location-long"], grp["location-lat"],
                zorder=2, color=colors[j])
    j += 1
plt.show()
```

In order to extract trajectories, it is mandatory to generate clusters. Here DBSCAN (density-based algorithm) is used.
First of all we select the period.

In [ ]:

```
#Create clusters
def selectdate(start_date,end_date):
    pd.options.mode.chained_assignment = None
    start = df.loc[[0]].timestamp.values[0]##
    end = df.tail(1).timestamp.values[0]
    #### not mandatory. Give an indication on the most recent and the older da
    date_range = pd.date_range(start=start_date,
                               end=end_date, freq=Itime)
                           # Interval of time (e.g. 7D = 7 days)
    return date_range

#summer time May - August
start_date="2009-05-27"
end_date="2009-08-27"

date_range=selectdate(start_date,end_date)
```

Plot the trajectories for the selected dates

In [ ]:

```
def plottrajectories (start_date, end_date, df):
    df1 = df[(df['timestamp'] > start_date) & (df['timestamp'] <= end_date)]
    gp = df1.groupby("tag-local-identifier")

    #Plot the map
    plt.figure(figsize=(width, height))
    map = Basemap(llcrnrlat=-5, urcrnrlat=68,llcrnrlon=-10,
                  urcrnrlon=65, lat_ts=20, resolution='c')
    map.shadedrelief()
    title="Trajectories from "+start_date+" to "+end_date
    plt.title(title)

    #Select the colors
    colors = cm.rainbow(np.linspace(0, 1, len(gp.indices)))

    #Plot the trajectories
    j=0
    for i in gp.indices:
        grp = gp.get_group(i)
        map.plot(grp["location-long"], grp["location-lat"],
                 zorder=2, color=colors[j])
        map.scatter(grp["location-long"], grp["location-lat"],
                    zorder=2, color=colors[j])
        j += 1
    plt.show()

plottrajectories (start_date, end_date, df)
```

Then we create the clusters

In [ ]:

```
def createclusters(date_range):
    t, c_id = 0, 0 # t: temporal index,  c_id: cluster id
    clusters = {}
    cluster_positions={} # Have the centroid of all the clusters

    while t + 1 < len(date_range):
        # Select data according to the time interval
        start, end = date_range[t], date_range[t + 1]
        mask = (df['timestamp'] > start) & (df['timestamp'] <= end)
        curr_df = df.loc[mask]

        # Get the coordinate of each entry
        data_geo = curr_df.loc[:, 'location-long':'location-lat']
        data_geo_2 = np.array(data_geo.values.tolist())
        # Run DBSCAN
        db = DBSCAN(eps=epsilon, min_samples=mint).fit(data_geo_2)
        # Each entry is associated to its cluster
        serie=pd.Series(db.labels_, index=curr_df.index)
        curr_df.loc[:, 'cluster'] = serie
        # groupby by cluster
        gp = curr_df.groupby("cluster")
        for i in gp.indices:

            # For each cluster
            grp = gp.get_group(i)

            # Get the centroid
            coord = grp.loc[:, 'location-long':'location-lat']
            coord = np.array(coord.values.tolist())
            cluster_positions[c_id]=[np.mean(coord[:,0]),
                                     np.mean(coord[:,1])]

            # Save the data
            # Interpretation: The cluster 'c_id' belong to time 't'
            #where there are birds 'birds_ids'
            clusters[c_id] = {"time": t,
                              "birds_ids": grp["tag-local-identifier"].unique()}
            c_id += 1
        t += 1

    print("Number of Clusters:",len(clusters))
    # We associate an id to each bird identifier
    # Each id corresponds to the line number in the input file of
    #jGetMove c_id, tag_2_id = 0, {}
    c_id, tag_2_id = 0, {}
    for tag in df["tag-local-identifier"].unique():
        tag_2_id[tag] = c_id
        c_id += 1
    return clusters, cluster_positions, tag_2_id


clusters, cluster_positions, tag_2_id = createclusters(date_range)
```

In [ ]:

```
#map and save data according to the input formats of jGetMove
def save_data(clusters):
    data_table = [[] for i in range(len(np.unique(df["tag-local-identifier"]))
    time_index_table = []
    nb=0
    for clu in clusters:
        for b in clusters[clu]["birds_ids"]:
            data_table[tag_2_id[b]].append(clu)

    for clu in clusters:
        time_index_table.append([clusters[clu]["time"], clu])
    time_index_table = sorted(time_index_table,
                              key=lambda a_entry: a_entry[0])

    np.savetxt("birdstimeindex.dat", np.array( time_index_table,
                                               dtype=int),
               fmt='%d')

    input_data = ""
    for l in data_table:
        if l != []:
            input_data += "\t".join([str(i) for i in l]) + "\n"
    input_data = input_data.strip()
    open("birds.dat", 'w').write(input_data)
    print ("Files generated\n")

save_data(clusters)
```

In [ ]:

```
#run jGetMove
#assume that the path is correctly set
#The full command in two lines for printing
#java -jar jGetMove-2.0.1/jGetMove.jar jGetMove-2.0.1/assets/birds.dat
#jGetMove-2.0.1/assets/birdstimeindex.dat -o results.json -p 2 -s 2 -t 1")


def runjgetmove():
    os.system("cp birds.dat jGetMove-2.0.1/assets/birds.dat")
    os.system("cp birdstimeindex.dat jGetMove-2.0.1/assets/birdstimeindex.dat"
    os.system("java -jar jGetMove-2.0.1/jGetMove.jar jGetMove-2.0.1/assets/bir


runjgetmove()
```

In [ ]:

```
def getpatterns():
    result=json.load(open("results.json"))
    pattern = result['patterns']
    nbpats=0
    for element in result['patterns']:
        nbpats=nbpats+1
    print ("Number of patterns:", nbpats, "\n")
    return result,nbpats


result,nbpatterns=getpatterns()
```

In [ ]:

```
def plotpatterns (nbpat):
    for nb in range (nbpat):
        index_pattern_value=nb+1
        links=pd.DataFrame(result["patterns"][-index_pattern_value]["links"])


        plt.clf()
        plt.figure(figsize=(width, height))

        nodes=result["nodes"]

        map = Basemap(llcrnrlat=-5, urcrnrlat=68,llcrnrlon=-10, urcrnrlon=65, 
        map.shadedrelief()
        map.drawrivers()
        map.fillcontinents(lake_color='#89C4F4', zorder=1)

        colors = cm.rainbow(np.linspace(0, 1, len(nodes)))
        title=result['patterns'][index_pattern_value-1]['name']+ " from "+star
        plt.title(title)

        for node in nodes:
            pos=cluster_positions[node["id"]]
            x,y=map(pos[0],pos[1])
            map.scatter(x,y,color=colors[node["id"]],zorder=2)

        # Plotting trajectories
        gp = links.groupby("value")
        colors_links = cm.rainbow(np.linspace(0, 1, len(gp.indices)))
        id_=0
        for i in gp.indices:
            grp = gp.get_group(i)
            coords=[[cluster_positions[row["source"]],
                     cluster_positions[row["target"]]] for ind, row in grp.ite
            final_cords=[]
            for c in coords:
                for sub_c in c:
                    final_cords.append(sub_c)
            final_cords=np.array(final_cords)
            map.plot(final_cords[:,0],final_cords[:,1],
                     color=colors_links[id_],zorder=3)
            id_+=1
        plt.show()
```

Let's plot the first pattern

In [ ]:

```
plotpatterns(1)
```

And all the patterns

In [ ]:

```
plotpatterns(nbpatterns)
```

Starting the migration (August - September)

In [ ]:

```
#August - September
start_date="2009-08-01"
end_date="2009-09-27"
Itime='5D'
date_range=selectdate(start_date,end_date)
clusters={}
cluster_positions={}
tag_2_id={}
clusters, cluster_positions, tag_2_id = createclusters(date_range)
save_data(clusters)
runjgetmove()
result,nbpatterns=getpatterns()
plotpatterns(nbpatterns)
```

Continuing the migration (September - November)

In [ ]:

```

#September - November
start_date="2009-09-30"
end_date="2009-11-15"
Itime='2D'
date_range=selectdate(start_date,end_date)
clusters={}
cluster_positions={}
tag_2_id={}
clusters, cluster_positions, tag_2_id = createclusters(date_range)
save_data(clusters)
runjgetmove()
result,nbpatterns=getpatterns()
plotpatterns(nbpatterns)
```

Visualising with Sankeys

Initialisation

Mamp must be running. One sankey visualization can be downloaded here: https://github.com/jGetMove/GetD3ed (https://github.com/jGetMove/GetD3ed)

```
#installation
git clone  https://github.com/jGetMove/GetD3ed
#install to the htdocs directory
cp -R GetD3ed /Applications/MAMP/htdocs/.
```

An example of visualization of trajectories with Sankeys

In [ ]:

```

import webbrowser as wb
#url='http://localhost/web/trajectories.html'

#wb.open_new(url)


url='http://localhost/GetD3ed_test/recast.html'
wb.open_new(url)
```