

Lab 8 - Mobile Data Analysis

Homework assignments and some of the previous labs have taught us how to measure certain aspects of wireless connectivity quality directly from a mobile device. These, client-side measurements, have completely revolutionized the way wireless measurements are done. Just ten years ago, if a cellular telephony provider wanted to know more about the quality of its network, they had to go around their coverage area with a measurement tool and painstakingly record data at every point. Now, thanks to smartphone applications and crowdsourcing, end-user devices can perform all of the measurements at any time, location, and for free.

Today, we are going to analyze measurements taken without such measurement effort. A company called OpenSignal (<http://opensignal.com/>) runs a very successful distributed measurement tool in the form of [a mobile app](#). They have kindly given us a small chunk of the collected data for educational purposes. The measurements we have contain cellular network measurements for four large cities (Jakarta, Sao Paolo, San Francisco Bay Area, and London). A full description of the dataset is on Ucilnica, please read it carefully, and then download the data (data_london.csv, data_saopaulo.csv, data_bayarea.csv, and data_jakarta.csv). The data is in the comma separated values format, so you can read it in Excel, if you want to quickly check what's in there.

Wireless connectivity statistics

With a rich dataset, such as the one provided by OpenSignal, there are numerous ways to dissect wireless connectivity quality. One obvious thing to do is to check if using different network providers results in different download/upload speeds. Let's check that.

You are free to do this part of the lab in any programming language, but the instructions will concentrate on Python. You should have the following Python modules installed: **pandas**, **numpy** and **matplotlib**.

Our task is to calculate the average upload/download speed for each of the network providers, and then compare them in a bar chart. The whole task can be done in a single short Python script - or better yet, in a Jupyter Notebook. Examine the CSV files. Notice that each row, representing a single measurement, has "network_name", "download_speed" and "upload_speed" fields. Let's read the rows and keep track of the measured speed for each of the providers.

We already worked with CSV data in the first labs. In this assignment we will be using a similar approach for reading and handling the CSV data using a pandas **DataFrame**. For example:

```
data = pd.read_csv(data_path, header=0, index_col=0)
print("Done. Loaded data frame (rows, columns):", data.shape, "")
```

Open one of the files (corresponding to one of the cities) for reading. Next, print the header of the DataFrame to inspect the content:

```
data.head()
```

Real-world data is often "dirty" and contains missing values (i.e. NaN values), so cleaning it is a necessary thing to do. First, let's perform a quick check on our dataset. We will use an approach based on the following [tutorial](#) to first check the number of NaN (missing values) in each column of the DataFrame and then clean up the three columns that are of interest to our analysis ("network_name",

“download_speed” and “upload_speed”). The following code will return a DataFrame denoting the total number of NaN values and the percentage of NaN values in each column of a given DataFrame:

```
def assess_NA(data):
    """
    Parameters
    -----
    data: dataframe
    """
    # pandas series denoting features and the sum of their null values
    null_sum = data.isnull().sum()# instantiate columns for missing data
    total = null_sum.sort_values(ascending=False)
    percent = ( ((null_sum / len(data.index))*100).round(2) ).sort_values(ascending=False)

    # concatenate along the columns to create the complete dataframe
    df_NA = pd.concat([total, percent], axis=1, keys=['Number of NA', 'Percent NA'])

    # drop rows that don't have any missing data; omit if you want to keep all rows
    df_NA = df_NA[ (df_NA.T != 0).any() ]

    return df_NA
```

Let’s run this method on our data and inspect the NaN statistics for the three fields we are interested in:

```
df_NA = assess_NA(data)
df_NA
```

We see that for all three we have some missing values, so we will next remove these NaN values and create new DataFrames with the cleaned download and upload speeds, together with the network names:

```
# Drop rows with null values on specific columns
upload_data=data.dropna(subset=['network_name', 'upload_speed'])
download_data=data.dropna(subset=['network_name', 'download_speed'])
```

We need to aggregate data for each of the providers, and then find the mean and standard deviation of the measurements for each of the providers. An easy way to do this is by using the **.groupby()** and **.agg()** methods on a pandas DataFrame, for example:

```
upload_data = upload_data.groupby('network_name').agg({'upload_speed': ['mean', 'std', 'size']})
download_data = download_data.groupby('network_name').agg({'download_speed': ['mean', 'std', 'size']})
```

Please note that providers with a very small number of measurements can skew the results. Thus, calculate the statistics for providers for which you have at least 100 valid measurements:

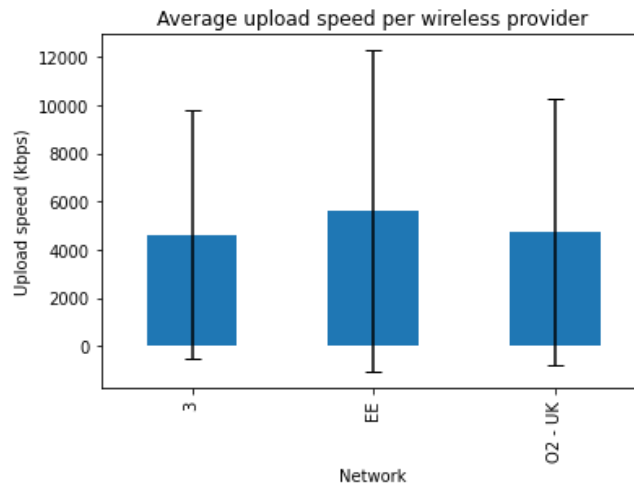
```
upload_data = upload_data.drop(upload_data[upload_data.iloc[:,2] < 100].index)
download_data = download_data.drop(download_data[download_data.iloc[:,2] < 100].index)
```

Finally, plot the data in a bar g code was used to generate the comparison of upload speeds for wireless prochart. **Matplotlib** provides a smooth way to do this. The followinviders operating in the London area:

```
upload_data.iloc[:,0].plot.bar(yerr=upload_data.iloc[:,1], capsize=5)
plt.ylabel("Upload speed (kbps)")
plt.xlabel("Network")
```

```
plt.show()
```

Where `upload_data.iloc[:,0]` and `upload_data.iloc[:,1]` represent the DataFrame columns containing the mean/std deviation values for download speeds. Of course, you should also take care of the labelling the graph. You should get something similar to the following:

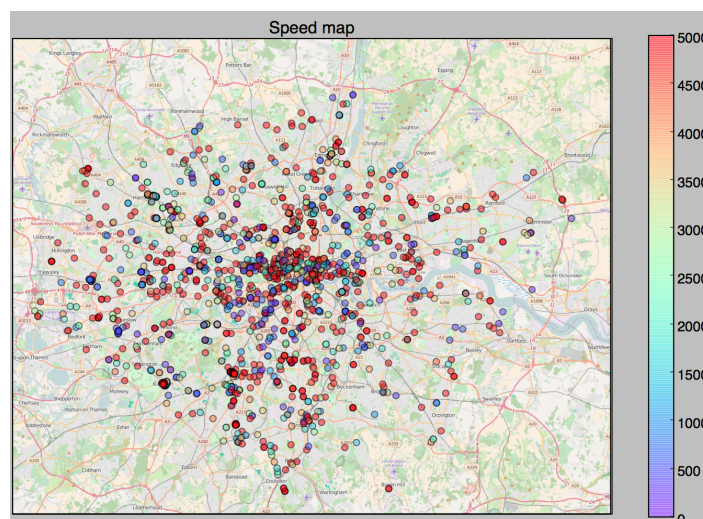


Now, let's compare the connectivity speed across different cities. Process the info for London, Sao Paolo, Jakarta and Bay Area, and compare the speeds by the top three providers in each of the cities. Which city has the fastest upload speed? What about the download speed?

Plotting connectivity data on a map

A quick way to check wireless connectivity in an area is to plot it on a map. In this lab we are going to plot OpenSignal measurements on a map, one marker per each measurement, so that the location of a marker corresponds to the location of a measurement, and the colour of the marker corresponds to the measured connectivity speed. We will use the following Python modules, so ensure they are installed: **matplotlib**, **basemap**, **matplotlib.pyplot**, **matplotlib.cm**, **urllib**, **PIL**, **BytesIO**. To show a realistic map, we get tiles from an open source mapping service -- Open Street Map.

We already know how to access the connectivity data from a CSV file. Extract the longitude, latitude and upload or download speed for each row. You can continue working in the same Jupyter Notebook. Our goal is to indicate the speed of connectivity at each of the measurement locations. See, for example the resulting map for London, where points of different download speed are shown in different colour (the speed is in kbps):



The data is plotted on top of an actual map of London. Such a map we can get from the [OpenStreetMap](https://www.openstreetmap.org/) project. The following code, adapted from <http://stackoverflow.com/a/28530369> will help us fetch the appropriate map tiles for the measurement data we have:

```
from urllib.request import urlopen
from urllib.request import Request
from io import BytesIO
from PIL import Image

def deg2num(lat_deg, lon_deg, zoom):
    lat_rad = math.radians(lat_deg)
    n = 2.0 ** zoom
    xtile = int((lon_deg + 180.0) / 360.0 * n)
    ytile = int((1.0 - math.log(math.tan(lat_rad) + (1 / math.cos(lat_rad)))) / math.pi) / 2.0 *
n)
    return (xtile, ytile)

def num2deg(xtile, ytile, zoom):
    n = 2.0 ** zoom
    lon_deg = xtile / n * 360.0 - 180.0
    lat_rad = math.atan(math.sinh(math.pi * (1 - 2 * ytile / n)))
    lat_deg = math.degrees(lat_rad)
    return (lat_deg, lon_deg)

def getImageCluster(lat_deg, lon_deg, delta_lat, delta_long, zoom):
    smurl = r"http://a.tile.openstreetmap.org/{0}/{1}/{2}.png"
    xmin, ymax = deg2num(lat_deg, lon_deg, zoom)
    xmax, ymin = deg2num(lat_deg + delta_lat, lon_deg + delta_long, zoom)

    bbox_ul = num2deg(xmin, ymin, zoom)
    bbox_ll = num2deg(xmin, ymax + 1, zoom)
    #print bbox_ul, bbox_ll

    bbox_ur = num2deg(xmax + 1, ymin, zoom)
    bbox_lr = num2deg(xmax + 1, ymax + 1, zoom)
    #print bbox_ur, bbox_lr

    Cluster = Image.new('RGB', ((xmax-xmin+1)*256-1, (ymax-ymin+1)*256-1) )
    for xtile in range(xmin, xmax+1):
        for ytile in range(ymin, ymax+1):
            try:
                imgurl=smurl.format(zoom, xtile, ytile)
                print("Opening: " + imgurl)
                imgstr = urlopen(Request(imgurl, headers={'User-Agent':
'Mozilla/5.0'})).read()
                tile = Image.open(BytesIO(imgstr))
                Cluster.paste(tile, box=((xtile-xmin)*255 , (ytile-ymin)*255))
            except Exception as inst:
                print(inst)
                print("Couldn't download image")
                tile = None

    return Cluster, [bbox_ll[1], bbox_ll[0], bbox_ur[1], bbox_ur[0]]
```

We just need to call `getImageCluster(lat_deg, lon_deg, delta_lat, delta_long, zoom)` with the minimum latitude and the minimum longitude found in our dataset, as well as the difference between the maximum latitude and minimum latitude, and the difference between the maximum longitude and minimum longitude. The last argument in the call is the zoom level - the higher this number is, the more detailed map tiles fetched from OpenStreetMaps will be. The function returns an image representing the map of a city (`a`), and a bounding box of the image (`bbox`).

Basemap is a Python library that enables geographical map plotting. It understands geographical coordinates (latitude/longitude) and different cartographic projections. The following code will instantiate a Basemap map object within the bounding box defined by the data from OpenSignal measurements:

```
from mpl_toolkits.basemap import Basemap, cm
from mpl_toolkits.axes_grid1 import make_axes_locatable

fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(111)
m = Basemap(
    llcrnrlon=bbox[0], llcrnrlat=bbox[1],
    urcrnrlon=bbox[2], urcrnrlat=bbox[3],
    projection='merc', ax=ax)
```

Now, fill the map with the map tiles from OpenStreetMap (that should be saved in a variable)

```
m.imshow(a, interpolation='lanczos', origin='upper')
```

Finally, add the scatterplot over the map, so that the colour of each data point corresponds to the achieved transmission speed, and instruct matplotlib to show your map.

If you did not attend this lab in-person, you can commit your solution (ideally a single Jupyter Notebook file `.ipynb`) to a private Bitbucket repository named **FRIMS2021-LAB-8** and a user **pbdfrita** (pbdfrita@gmail.com) should be added as a read-only member. The solutions will be pulled from your repository on **Sunday, May 15th, 23:59**.

Happy coding!