OpenStreetMap から人口を推測する

stranger_86952

0.1 はじめに

「地図いじってなんかやりたい」って思ってたので、今回は地図上の情報からある程度可能そうな、人口予測に挑戦してみます。インド都市部などのググるだけでは古い統計しか得られない地域や、そもそも調べるだけでは人口がわからないエリア (灘校から半径 1km の範囲、など) もあるので、精度さえ出せればある程度需要もある技術だと思っています。

0.2 OpenStreetMap について

OpenStreetMap Wiki によると

> OpenStreetMap は、フリー、かつ編集可能な世界地図であり、数多くのボランティアによってゼロから作られ、オープンコンテンツのライセンスのもとで提供されています。

らしいです。地図版の Wikipedia みたいなかんじですね。今回は OverpassAPI を使って OpenStreetMap の情報を使います。
OverpassAPI では OpenStreetMap のほとんど全ての情報を入手できて、Manual にも載っていますが、空港や高速道路、地名などはもちろん、ATM の位置や氷河の割れ目など、本当に多岐にわたる情報が得られます。API は Overpass turbo で簡単に試すことができ、例えば以下のクエリで「灘校から半径 1km 以内にある駅の数」を得ることができます。

```
[out:json];
node(around:1000,34.71955, 135.26818)["railway"="station"];
out count;
```

このクエリだと住吉 (JR, 阪神, 六甲ライナー)、魚崎 (阪神, 六甲ライナー) が該当するので 5 と返ってきます。

0.3　どうやって人口を予測するのか

先に述べた通り、膨大な種類のデータを得ることができるため、建築物の分布やインフラの整備具合を API で取得し、線形回帰やランダムフォレストといった機械学習の手法を使って人口を予測するモデルを作ります。もちろんこれらの条件が同じでも人口は同じになるわけではないので、国や地域ごとにパラメーターを調整しないといけません。
例えば人口あたりの病院や駅の数は国ごとに異なるため、同じ病院、駅の数でも先進国と発展途上国なら人口がかけ離れている可能性があります。
そもそも OSM から得られる情報を元に直接導けるのはせいぜい世帯数だと考えられるため、一世帯あたりの人数が必要ですが、これは国・地域ごとに明らかに異なります。それらのパラメーターを調整するために、既存の人口データも用います。
次の章から、実際にこれらの要素を用いて人口予測を行い、どれくらいの精度が出るのか調べてみたいと思います。

0.4　日本国内で人口予測を試みる

まずは人口統計が充実していて、地方でも比較的インフラが整備されている日本国内で試します。OSM から学校やコンビニなどの数を取得した後 (いわゆる説明変数)、以下のようなデータセットを大量に用意し、人口を求めるモデルを Python で作ります。ちなみに、学習時に人口データが必要なので総務省が公開している統計をダウンロードして良い感じに使える形に処理しないといけないのですが、"～市～区" や" ～郡～町" などの例外処理がちょっとだけ面倒でした。なお OverpassAPI の使用上、同一名称の市町村を区別することが難しいため、今回は同名の市町村および北方地域の自治体を除外しています。
※実際のデータは CSV ですが、収まりきらないためここでは JSON に変換しています。

Listing 1　Example JSON

```
{
 & "city": 札幌市"",
```

```
 & "level": 7,
 & "population": 1959512,
 & "households": 1096729,
 & "school": 410,
 & "college": 59,
 & "hospital": 184,
 & "doctors": 472,
 & "clinic": 63,
 & "dentist": 485,
 & "restaurant": 1085,
 & "fast_food": 274,
 & "supermarket": 292,
 & "convenience": 960,
 & "park": 2458,
 & "platform": 3709,
 & "station": 75
}
```

このデータのうち city は市区町村名、population と households が予測する値、school より下が予測に用いる値です。

Python の sklearn ライブラリで簡単に試せる手法として、線形回帰、勾配ブースティング木、パーセプトロン、などさまざまなものがありますが、どれが最適なのかわからないのでとりあえず回帰と木を試してみます。少なくともニューラルネットワークは適切じゃなさそうだったので排除しました。コードはほとんど変わらないので、線形回帰の場合だけ以下に載せます。

Listing 2 Linear.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

drop_columns = ['city', 'level', 'population', 'households']

train_data = pd.read_csv('../data/learn.csv')
X_train = train_data.drop(columns=drop_columns)
Y_train = train_data[drop_columns[2:4]]

test_data = pd.read_csv('../data/test-input.csv')
X_test = test_data.drop(columns=drop_columns)

Model = LinearRegression()

Model.fit(X_train, Y_train)
Predictions = Model.predict(X_test)

test_data['population'] = Predictions[:, 0].astype(int)
test_data['households'] = Predictions[:, 1].astype(int)
test_data.to_csv('../data/test-output.csv', index=False)
for i, row in test_data.iterrows():
    print(row.iloc[0],row.iloc[2],row.iloc[3])
```

ランダムに 10 市区町村をピックアップし、残りの市区町村からランダムに選ばれた 10,100,1000 個のデータを学習させ、ピックアップしたものを予測させた結果がそれぞれ以下の通りになります。

※ここでは世帯数の予測については結果を省略しています。

線形回帰	実際の人口	10	100	1000
玉東町	5241	1113	-1640	-3147
門真市	117937	-20453	95077	97586
山陽小野田市	60209	65287	56634	66631
宮代町	33514	26152	25023	24197
留萌市	19234	35599	21203	18606
大津市	344552	142351	216066	237161
砥部町	20510	14172	17633	9370
栄村	1642	7272	916	942
美波町	6071	17680	17290	17541
みなべ町	11988	10799	17628	17655

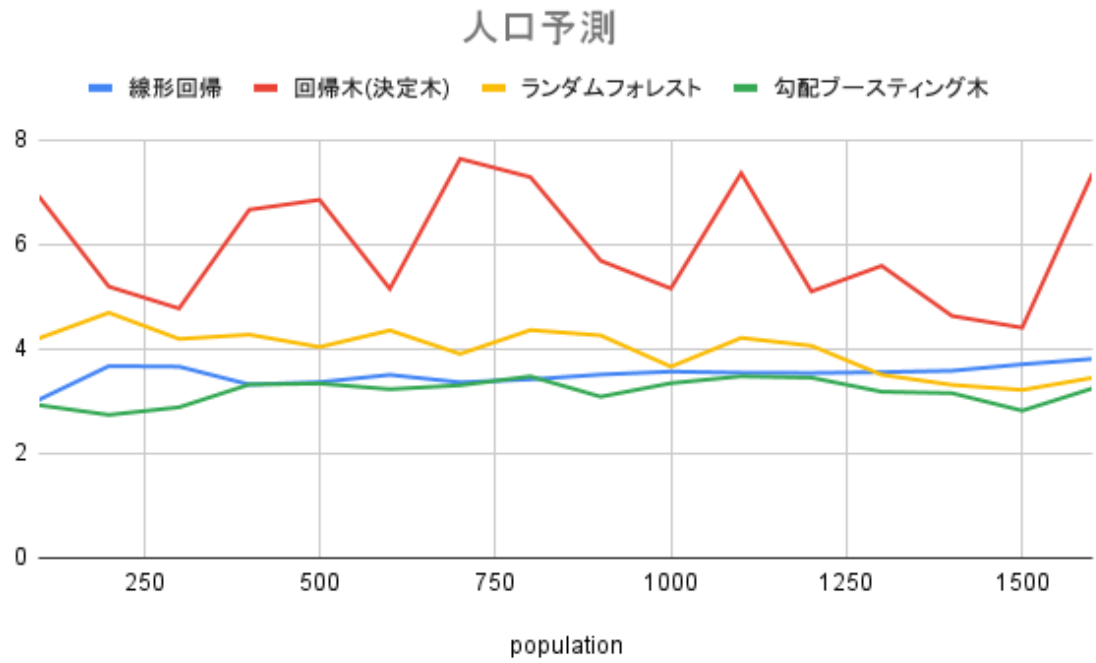
回帰木 (決定木)	実際の人口	10	100	1000
玉東町	5241	13813	1106	1159
門真市	117937	71296	121770	114259
山陽小野田市	60209	31644	27941	46328
宮代町	33514	18611	18611	18511
留萌市	19234	18611	30658	34811
大津市	344552	63299	270085	256005
砥部町	20510	49530	6017	6420
栄村	1642	13813	1075	2953
美波町	6071	31644	21513	5282
みなべ町	11988	12796	7906	5883

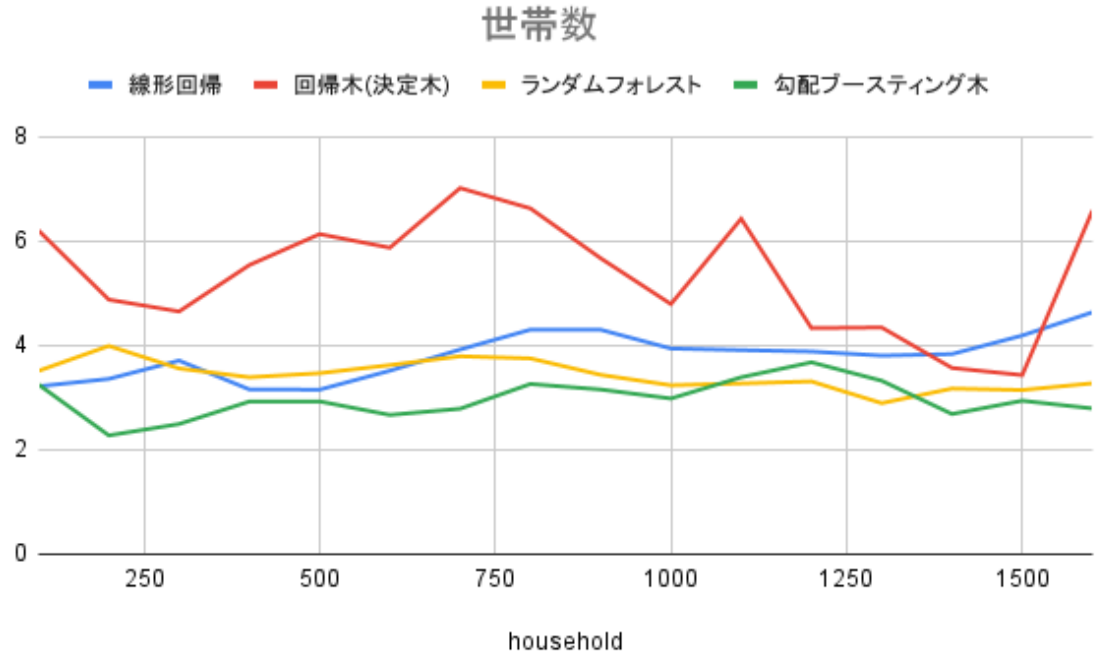
まず線形回帰について、人口の予測で負の値がでているのはかなりひどいです。学習データが 10 個の場合はほとんど当てにならない精度をしていますが、100,1000 個の場合についてはかなり実際のデータと近いものも存在します。ただ、この 2 つの精度について大きな差があるようには感じません。
回帰木 (決定木) についても学習量が 10 この場合はかなり的外れな予測が多いように見えます。線形回帰とは違ってちゃんとすべての予測が正の数になっています。
次に、予測の精度を測ってみたいと思います。単純に人口の差をとってしまうと、大都市であれば無条件に値が大きくなってしまうので、以下の数式で定めたいと思います。

※ただし、N はサンプル数、A を統計上の人口、B を予測した人口とします。

$$\frac{1}{N}\sum_{i=0}^{N}\frac{|A-B|}{A}$$

0 に近ければ精度が良く、元の人口の 2 倍以上の値を予測すれば 1 を超えることもあり得ます。
適切な学習量とモデルを探るため、学習データを 100 1600 市区町村まで 100 ずつ増やしながらそれぞれ精度をグラフにすると、以下の通りになります。





人口、世帯数両方に共通することとして、回帰木はかなり精度が低く、勾配ブースティング木の精度が比較的高いことがわかります。また、世帯数の予測の方が精度が高く見えます。学習量についてはあまり違いが見られませんでした。

0.5　世界の他地域でも予測できるのか

国連が公開している人口 10 万人以上の都市の統計を利用して、今のモデルが世界の他地域に適用できるか試してみます。

ランダムフォレスト	実際の人口	10	100	1000
Saudi Arabia - Jeddah	3430697	125943	487349	616187
Botswana - Francistown	103422	27105	33455	15225
New Zealand - Christchurch	392100	170602	561612	479461
Peru - Talara	102355	58940	59620	84279
Italy - Messina	221788	172539	545927	959142
Russia - Bratsk	242604	81483	138756	180935
Morocco - Ouarzazate	131103	29025	23680	29184
India - Cuttack	610189	75156	144407	92001
Bolivia - Oruro	216724	87998	67434	93057
United States - Billings	119960	109210	201819	154315

勾配ブースティング木	実際の人口	10	100	1000
Saudi Arabia - Jeddah	3430697	124591	423063	650126
Botswana - Francistown	103422	20557	28452	22567
New Zealand - Christchurch	392100	205298	554581	405102
Peru - Talara	102355	69435	29574	90102
Italy - Messina	221788	200102	412702	896973
Russia - Bratsk	242604	74619	171280	190775
Morocco - Ouarzazate	131103	24569	18894	18345
India - Cuttack	610189	62036	222939	86854
Bolivia - Oruro	216724	59413	52505	100622
United States - Billings	119960	140838	151548	141974

ぱっと見で国内より明らかに精度が落ちていることがわかりますが、実際に精度を測ってみると以下のようになります。

精度	10	100	1000
ランダムフォレスト	0.5915473522	0.7227790174	0.8134995744
勾配ブースティング木	0.5963988875	0.6393394947	0.7440039412

ある程度の精度が出ていた国内と違い、データ数が少ないほど精度が上がってることを考

えると、国内のデータで作ったモデルは他地域でほとんど通用していないことがわかります。

0.6　まとめ

いくつか反省点があります。
まず、説明変数の選び方が最適ではなかったと思います。一般的に互いに相関が強い説明変数を選ぶことは避けるべきですが、あまり考えずにやっていました。"school"と"college"、"hospital"と"doctors"と"clinic"と"dentist"など、まとめることができそうなものも多かったです。
他にも、今回扱った変数だけでは、人口が違う自治体でも完全に一致してしまうことがあり、他の変数を追加するべきでした。学習データについても、世界の他都市に応用させるなら国内だけのデータで学習させるのはよくなかったです。
ただ、国内については一定の精度で人口が予測できたので、概算が目的であれば使えるかもしれません。
最後に学習データについて、総務省の Excel をベースに OverpassAPI を Python で叩きながら集めたのですが、クエリが多すぎてほぼ丸一日かかってたみたいなので、使いたい方がいれば下に配布しておきます。2024 年 4 月時点でのデータなのでご了承ください。
https://github.com/stranger86952/Japanese-Data-By-Municipalities