Prediction of future demographic data of countries via ML methods

March 2022

Introduction

The importance of demographic data is hard to overstate. For example, population and its growth give insight into relative power of countries, the need to build new infrastructure and housing, and the ecological burden human consumption has on the planet. A stark instance of the significance of population would be modern Egypt with its levels of population growth coupled with food insecurity and water scarcity considered unsustainable.[1] Alongside that statistic age dependency ratio will be of interest in this project. By the end of 21st century virtually all countries are expected to experience a strong decline in birth rates which is most likely going to result in an obvious economic calamity that comes with an aging society having to be sustained by an increased tax burden on people of working age. Today, this problem is most acute in South Korea with a total fertility rate of a mere 0.84.[2]

This report will start off by formulating the machine learning problem. Then an explanation of what ML methods were used and why. Lastly, the results and their performance will be discussed.

Problem formulation

The objective of the algorithm is to predict the population size and the age dependency ratio of a country sometime in the near future (2050, for instance) based on a multitude of demographic and other statistical data. From this the labels and datapoints can be inferred, with datapoints being countries and the numerical labels being its population and the age dependency ratio of said population.

When it comes to the features, the selection is trickier, however. Of course, there are obvious feature choices for our problem such as the total fertility rate, migration data and life expectancy. Yet, it is difficult to extrapolate accurately from these features, since it is very hard to predict how they would evolve in isolation. That is why this is framed as a problem, where ML applications may help. For example, the total fertility rate may stay unchanging for decades or decline very rapidly (like in India and Iran, for instance). The same applies to migration rates. Therefore, to get a more accurate picture more features have to be included.

A list of possible features and why they matter:

- GDP per capita: it is well-known that economic prosperity tends to result in people having smaller families, so one would expect the population of a rich country to grow less and it to age faster. However, a rich country may attract more immigrants and have higher life expectancy which would result in the previous effect being compensated for. Since the objective is to predict the future, GDP growth rate is also going to be included as a feature.
- The region the country belongs to: countries from the same region tend to be similar so one would expect that if we created a new fictional country in Northeast Asia, it would age faster, and its population growth would slow down quicker as is the case for Japan, South Korea and China. (While the type of data of other features mentioned here would be float values, for regions binary categories of 1 and 0 would be used.)

- Other features which might be related to demographics such as the religion of the country and how strongly its people identify with religion, urbanization, the degree of gender equality, and how democratic the country is.

Thankfully, most countries and supranational organizations are very interested in collecting this kind of data, so there is a lot to go around. One website that might come in handy when it comes to TFR and migration could be worldometers.info. For urbanization, GDP and its growth UN or World Bank data could be used. Other data that could be included are various indexes such as the Human Development Index, Democracy index and The Global Gender Gap Index.

Methods:

Data

The source of data are the various country-specific statistical indicators that can be found on the World Bank website (https://data.worldbank.org/indicator). The dataset was created using the Databank tool also offered by World Bank (https://databank.worldbank.org/home.aspx). It included average values of indicators from 1965 to 1975 as features and a couple of indicator values from 2020 (total population and age dependency ratio) that are there to serve as labels. Then, to represent the regions countries are in, more features of the binary categorical type were added via Excel, creating the final version of the dataset. The datapoints consist of 112 countries, for which the required data was available.

Feature selection

Since the goal is to predict 45 years into the future, many features that might give meaningful information were used (as was explained in the Problem formulation section). Though, for the same reason some of the possible features were not chosen due to the data relating to those indicators not being collected as much in the 60s and 70s, as it is today (indexes relating to female participation in the workforce, for example). Values were averaged from 1965 to 1975 because the data was not available for all countries in some specific year and to find long-term trends.

The features are as follows:

Float:

- GDP per capita growth (annual %)
- GDP per capita (current US$)
- Fertility rate, total (births per woman)
- Net migration
- Life expectancy at birth, total (years)
- Urban population (% of total population)
- Urban population growth (annual %)
- Population, total
- Age dependency ratio (% of working-age population)

1 (belongs to that region) or 0 (does not belong to that region)

- Europe and Oceania
- North America
- Latin America

- MENA
- Africa
- NE Asia
- SE Asia

Model, loss, and validation

Since the problem is numerical, a linear method was chosen. There is no reason to initially assume that one can make such a complicated prediction related to social sciences linearly, so polynomial maps were used (though it is not necessarily non-linear, so degree 1 is also used with the PolynomialFeatures class which is the same as just using LinearRegression class alone). Concerning loss functions, PolynomialFeatures only seems to accept the LinearRegression class, so mean squared error was used. In addition, for training and validation error root mean squared error and absolute error were sometimes used to make large numbers more readable (when it comes to evaluating the total population label the error also divided by one million).

To evaluate how well the polynomial map model performs at different polynomial degrees k–fold cross validation with 3 splits was employed. The reason for this is that the feature to datapoint ratio of the dataset was quite large (16/112), so it was important to avoid overfitting.

The second method that was employed was a multi-layer perceptron was used, again with mean squared error loss because that is what the convenient MLPRegressor SK-learn class uses. This method was chosen because neural networks are known to find complex relationships between a multitude of features well. Layers with 15 neurons were used and the optimal number of layers was found by using k-fold cross validation again, for the same reasons as those discussed above.

Results

Polynomial regression

Unexpectedly, both for population and for age dependency ratio linear maps performed the best (using degree 1 in the polynomial class). With regards to predicting population, after cross validation the average root mean square training error was 27.5 million, and the validation error was 43.3 million. We saw better results when it comes to age dependency ratio, root mean square training error was 6.8 percent, and the validation error was 9.65 percent when it came to using a linear map.

Artificial neural network

Concerning population, using the number of layers between 1 and 4 gave similar performance, but 3 layers made marginally better predictions than the other options with the RMSE for training and validation being 37.47 million and 35.33 million respectively.

For age dependency ratio worse performance was seen with training RMSE being 66.4 percent and the validation set having an error of 126.3 percent.

Choosing the best-performing methods

For age dependency ratio linear regression was chosen, because its superiority to ANN is obvious from the training and validation errors discussed above.

However, when predicting the population label whether linear regression or neural networks fare better is not as apparent, so one last test set, that separates the data into training, validation and test subsets was constructed. Since we only have 112 data points, the validation and test sets were chosen to be small (first split had test size 0.2 and it was 0.175 for the second one). The results were inconclusive because a similar performance was seen when taken overall, though the neural network performed significantly better when it came to the test set:

Training, validation, and test errors for linear map = 34.85106, 24.92887, 11.05578 training, validation, and test errors for ANN = 43.12675, 28.52594, 2.93365.

Conclusion

To summarize, when using a linear map satisfactory predictions of the dependency ratio were made. However, with regards to population both linear regression and artificial neural networks gave subpar performance: an error in tens of millions, especially when it comes to smaller countries is way too high.

What improvements could be made?

In retrospect, I think that with population prediction the problem lies not in the methods that were used but in the initially constructed dataset. Firstly, when age dependency ratio from the 70s is used as a feature it has for the method to tell how it would affect the population growth because it includes both people who are retired and minors who are too young to work, so other similar statistics should be used that split it into two features. That's because right now the methods cannot tell whether the feature should lead to population increase or decline – it is ambiguous and does not reveal whether the society in question is old or young. In addition, our dataset only uses data from the World Bank that was readily available to the western world from 1965 to 1975, so it only includes around half of all countries. Most of the former Eastern Bloc is missing from our dataset for example. Lastly, it is possible that taking a 10-year average is not the right choice. Our model may have fared better if we instead included more features with the data of every year standing alone to capture the existing trends better.

References

[1] climate-diplomacy.org. (n.d.). Security Implications of Growing Water Scarcity in Egypt | Climate-Diplomacy. [online] Available at: https://climate-diplomacy.org/case-studies/security-implications-growing-water-scarcity-egypt.

[2] www.bloomberg.com. (n.d.). Bloomberg [online] Available at: https://www.bloomberg.com/news/articles/2021-12-09/world-s-lowest-fertility-rate-to-get-even-lower-korea-reports.

Code ML Project

March 31, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, HuberRegressor, Ridge
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
import math
from sklearn.neural_network import MLPRegressor
```

[2]:
```
df = pd.read_csv('CountryData.csv')
pd.set_option('display.max_rows', None)
df.head(5)
```

[2]:
```
   Country Name Country Code  1965-75 - GDP per capita growth (annual %)  \
0       Algeria          DZA                                    3.562091
1     Argentina          ARG                                    2.608148
2     Australia          AUS                                    2.486779
3       Austria          AUT                                    3.996664
4  Bahamas, The          BHS                                   -1.835232

   1965-75 - GDP per capita (current US$)  \
0                               432.133271
1                              1559.786628
2                              3812.174539
3                              2645.920392
4                              3010.071201

   1965-75 - Fertility rate, total (births per woman)  \
0                                           7.609636
1                                           3.124000
2                                           2.734545
3                                           2.300000
4                                           3.584091

   1965-75 - Net migration  1965-75 - Life expectancy at birth, total (years)  \
0                -187385.5                                           50.463636
```

```
1              136000.0                                          66.595545
2              532862.0                                          71.334501
3               66201.0                                          70.291885
4                7439.0                                          65.948455

   1965-75 - Urban population (% of total population)  \
0                                          39.406909
1                                          78.780727
2                                          83.982545
3                                          65.219273
4                                          66.730818

   1965-75 - Urban population growth (annual %)  1965-75 - Population, total  \
0                                     3.784758                 1.450397e+07
1                                     2.138896                 2.394057e+07
2                                     2.202653                 1.261155e+07
3                                     0.489130                 7.463840e+06
4                                     4.153330                 1.671479e+05

   1965-75 - Age dependency ratio (% of working-age population)  \
0                                         102.469745
1                                          57.573863
2                                          59.646961
3                                          61.361121
4                                          83.005674

   Europe and Oceania  North America  Latin America  MENA  Africa  NE Asia  \
0                   0              0              0     1       0        0
1                   0              0              1     0       0        0
2                   1              0              0     0       0        0
3                   1              0              0     0       0        0
4                   0              0              1     0       0        0

   SE Asia  2020 - Population, total  \
0        0                  43851043
1        0                  45376763
2        0                  25687041
3        0                   8917205
4        0                    393248

   2020 - Age dependency ratio (% of working-age population)
0                                          60.067725
1                                          55.767661
2                                          55.051941
3                                          50.639108
4                                          41.541651
```

[3]:
```
features = df.drop(['Country Name','Country Code','2020 - Population,␣
↪total','2020 - Age dependency ratio (% of working-age population)'],axis=1)
X = np.array(features.to_numpy()).reshape(112,16)
#print(X[:, 0])
#print(X)
labelpop=df['2020 - Population, total']
y_pop=np.array(labelpop.to_numpy())
labelage=df['2020 - Age dependency ratio (% of working-age population)']
y_age=np.array(labelage.to_numpy())
```

[4]:
```
#population k-fold
from sklearn.model_selection import KFold
k, shuffle, seed = 3, True, 27
kfold = KFold(n_splits=k, shuffle=shuffle, random_state=seed)
degrees = [1, 2, 3, 4, 5, 6, 7, 8]

tr_errors = {}
val_errors = {}

for i, degree in enumerate(degrees):
    tr_errors[degree] = []
    val_errors[degree] = []

    for j, (train_indices, val_indices) in enumerate(kfold.split(X)):


        X_train, y_train, X_val, y_val = X[train_indices],␣
↪y_pop[train_indices], X[val_indices], y_pop[val_indices]

        lin_regr = LinearRegression()
        poly = PolynomialFeatures(degree=degree)
        X_train_poly = poly.fit_transform(X_train)
        lin_regr.fit(X_train_poly, y_train)


        y_pred_train = lin_regr.predict(X_train_poly)
        tr_error = math.sqrt(mean_squared_error(y_train, y_pred_train))/1000000
        X_val_poly = poly.transform(X_val)
        y_pred_val = lin_regr.predict(X_val_poly)
        val_error = math.sqrt(mean_squared_error(y_val, y_pred_val))/1000000

        tr_errors[degree].append(tr_error)
        val_errors[degree].append(val_error)
```

[5]:
```
average_train_error, average_val_error = {}, {}
for degree in degrees:
    average_train_error[degree] = np.mean(tr_errors[degree])
    average_val_error[degree] = np.mean(val_errors[degree])

    print(f"Degree {degree}, avg train error(RMSE normalized to millions) =␣
↪{average_train_error[degree]:.5f}, "
          f"avg val error(RMSE normalized to millions) =␣
↪{average_val_error[degree]:.5f}")
```

```
Degree 1, avg train error(RMSE normalized to millions) = 27.58444, avg val
error(RMSE normalized to millions) = 43.38040
Degree 2, avg train error(RMSE normalized to millions) = 1.73945, avg val
error(RMSE normalized to millions) = 257.09964
Degree 3, avg train error(RMSE normalized to millions) = 5.35801, avg val
error(RMSE normalized to millions) = 28476.78676
Degree 4, avg train error(RMSE normalized to millions) = 12.37076, avg val
error(RMSE normalized to millions) = 5114497.16254
Degree 5, avg train error(RMSE normalized to millions) = 8.72024, avg val
error(RMSE normalized to millions) = 87219787.55370
Degree 6, avg train error(RMSE normalized to millions) = 12.19736, avg val
error(RMSE normalized to millions) = 6943614088.92022
Degree 7, avg train error(RMSE normalized to millions) = 16.90102, avg val
error(RMSE normalized to millions) = 59632093047.58840
Degree 8, avg train error(RMSE normalized to millions) = 27.17230, avg val
error(RMSE normalized to millions) = 2035405990876.12817
```

[6]:
```
#dependency ratio k-fold
from sklearn.model_selection import KFold
k, shuffle, seed = 3, True, 27
kfold = KFold(n_splits=k, shuffle=shuffle, random_state=seed)
degrees = [1, 2, 3, 4, 5, 6, 7, 8]

tr_errors = {}
val_errors = {}

for i, degree in enumerate(degrees):
    tr_errors[degree] = []
    val_errors[degree] = []

    for j, (train_indices, val_indices) in enumerate(kfold.split(X)):


        X_train, y_train, X_val, y_val = X[train_indices],␣
↪y_age[train_indices], X[val_indices], y_age[val_indices]

        lin_regr = LinearRegression()
        poly = PolynomialFeatures(degree=degree)
        X_train_poly = poly.fit_transform(X_train)
        lin_regr.fit(X_train_poly, y_train)
        y_pred_train = lin_regr.predict(X_train_poly)
        tr_error = math.sqrt(mean_squared_error(y_train, y_pred_train))
        X_val_poly = poly.transform(X_val)
        y_pred_val = lin_regr.predict(X_val_poly)
        val_error = math.sqrt(mean_squared_error(y_val, y_pred_val))

        tr_errors[degree].append(tr_error)
        val_errors[degree].append(val_error)
```

[7]:
```
average_train_error, average_val_error = {}, {}
for degree in degrees:
    average_train_error[degree] = np.mean(tr_errors[degree])
    average_val_error[degree] = np.mean(val_errors[degree])

    print(f"Degree {degree}, avg train error(RMSE) =␣
 ↪{average_train_error[degree]:.5f}, "
          f"avg val error(RMSE) = {average_val_error[degree]:.5f}")
```

```
Degree 1, avg train error(RMSE) = 6.79515, avg val error(RMSE) = 9.65088
Degree 2, avg train error(RMSE) = 5.25121, avg val error(RMSE) = 634.81688
Degree 3, avg train error(RMSE) = 9.36685, avg val error(RMSE) = 13323.21294
Degree 4, avg train error(RMSE) = 12.63436, avg val error(RMSE) = 12168961.83228
Degree 5, avg train error(RMSE) = 13.44522, avg val error(RMSE) =
447662745.28637
Degree 6, avg train error(RMSE) = 14.09625, avg val error(RMSE) =
1041370014.36327
Degree 7, avg train error(RMSE) = 14.39855, avg val error(RMSE) =
17505482794.91799
Degree 8, avg train error(RMSE) = 14.63337, avg val error(RMSE) =
701568142807.45520
```

[8]:
```
#Turns out that for both labels the relationship to features seems to be more␣
 ↪linear
#degree=1 outperforms polynomial maps for both.
```

[9]:
```
#Artificial neural network
```

[10]:
```
#population k-fold
from sklearn.model_selection import KFold
k, shuffle, seed = 3, True, 27
kfold = KFold(n_splits=k, shuffle=shuffle, random_state=seed)
num_layers = [1, 2, 3, 4, 5]
num_neurons = 15
tr_errors = {}
val_errors = {}
```

```

for i, num in enumerate(num_layers):
    tr_errors[num] = []
    val_errors[num] = []

    for j, (train_indices, val_indices) in enumerate(kfold.split(X)):


        X_train, y_train, X_val, y_val = X[train_indices],␣
↪y_pop[train_indices], X[val_indices], y_pop[val_indices]
        hidden_layer_sizes = tuple([num_neurons]*num)
        mlp_regr = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,␣
↪max_iter=15000, random_state=40)
        mlp_regr.fit(X_train, y_train)


        y_pred_train = mlp_regr.predict(X_train)
        tr_error = math.sqrt(mean_squared_error(y_train, y_pred_train))/1000000
        y_pred_val = mlp_regr.predict(X_val)
        val_error = math.sqrt(mean_squared_error(y_val, y_pred_val))/1000000

        tr_errors[num].append(tr_error)
        val_errors[num].append(val_error)
```

[11]:
```
average_train_error, average_val_error = {}, {}
for num in num_layers:
    average_train_error[num] = np.mean(tr_errors[num])
    average_val_error[num] = np.mean(val_errors[num])

    print(f"Number of layers {num}, avg train error(RMSE normalized to␣
↪millions) = {average_train_error[num]:.5f}, "
          f"avg val error(RMSE normalized to millions) =␣
↪{average_val_error[num]:.5f}")
```

```
Number of layers 1, avg train error(RMSE normalized to millions) = 35.48744, avg
val error(RMSE normalized to millions) = 39.94629
Number of layers 2, avg train error(RMSE normalized to millions) = 35.58603, avg
val error(RMSE normalized to millions) = 38.97111
Number of layers 3, avg train error(RMSE normalized to millions) = 37.47505, avg
val error(RMSE normalized to millions) = 35.33877
Number of layers 4, avg train error(RMSE normalized to millions) = 38.60379, avg
val error(RMSE normalized to millions) = 35.35158
Number of layers 5, avg train error(RMSE normalized to millions) = 41.46304, avg
val error(RMSE normalized to millions) = 35.97543
```

[12]:
```
#dependency ratio k-fold
from sklearn.model_selection import KFold
```

```
k, shuffle, seed = 3, True, 27
kfold = KFold(n_splits=k, shuffle=shuffle, random_state=seed)
num_layers = [1, 2, 3, 4, 5]
num_neurons = 15
tr_errors = {}
val_errors = {}

for i, num in enumerate(num_layers):
    tr_errors[num] = []
    val_errors[num] = []

    for j, (train_indices, val_indices) in enumerate(kfold.split(X)):


        X_train, y_train, X_val, y_val = X[train_indices],␣
 ↪y_age[train_indices], X[val_indices], y_age[val_indices]
        hidden_layer_sizes = tuple([num_neurons]*num)
        mlp_regr = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,␣
 ↪max_iter=1000, random_state=40)
        mlp_regr.fit(X_train, y_train)


        y_pred_train = mlp_regr.predict(X_train)
        tr_error = math.sqrt(mean_squared_error(y_train, y_pred_train))
        y_pred_val = mlp_regr.predict(X_val)
        val_error = math.sqrt(mean_squared_error(y_val, y_pred_val))

        tr_errors[num].append(tr_error)
        val_errors[num].append(val_error)
```

```
/opt/conda/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[13]:
```
average_train_error, average_val_error = {}, {}
for num in num_layers:
    average_train_error[num] = np.mean(tr_errors[num])
    average_val_error[num] = np.mean(val_errors[num])

    print(f"Number of layers {num}, avg train error(RMSE) =␣
 ↪{average_train_error[num]:.5f}, "
          f"avg val error(RMSE) = {average_val_error[num]:.5f}")
```

```
Number of layers 1, avg train error(RMSE) = 977992.64372, avg val error(RMSE) =
792382.52704
```

```
Number of layers 2, avg train error(RMSE) = 66.39175, avg val error(RMSE) =
126.35364
Number of layers 3, avg train error(RMSE) = 409732.95552, avg val error(RMSE) =
421018.51848
Number of layers 4, avg train error(RMSE) = 412650.45416, avg val error(RMSE) =
347823.71838
Number of layers 5, avg train error(RMSE) = 185123.07029, avg val error(RMSE) =
150759.58372
```

[14]:
```
#There is no point to compare linear regression with neural networks when it␣
↪comes to dependency ratio
#However, when it comes to population, the errors between ANN and linear␣
↪regression are comparable, so we will
#Which method fares better with a test error split
```

[15]:
```
X_train, X_rem, y_train, y_rem = train_test_split(X, y_pop, test_size=0.2,␣
↪random_state=45)
X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.175,␣
↪random_state=45)
```

[16]:
```
lin_regr=LinearRegression()
lin_regr.fit(X_train, y_train)
y_pred_train_linear = lin_regr.predict(X_train)
tr_error_linear = math.sqrt(mean_squared_error(y_train, y_pred_train_linear))/
↪1000000
y_pred_val_linear = lin_regr.predict(X_val)
val_error_linear = math.sqrt(mean_squared_error(y_val, y_pred_val_linear))/
↪1000000
y_pred_test_linear = lin_regr.predict(X_test)
test_error_linear = math.sqrt(mean_squared_error(y_test, y_pred_test_linear))/
↪1000000



mlp_regr=MLPRegressor(hidden_layer_sizes=tuple([15]*3), max_iter=15000,␣
↪random_state=40)
mlp_regr.fit(X_train, y_train)
y_pred_train_neural = mlp_regr.predict(X_train)
tr_error_neural = math.sqrt(mean_squared_error(y_train, y_pred_train_neural))/
↪1000000
y_pred_val_neural = mlp_regr.predict(X_val)
val_error_neural = math.sqrt(mean_squared_error(y_val, y_pred_val_neural))/
↪1000000
y_pred_test_neural = mlp_regr.predict(X_test)
test_error_neural = math.sqrt(mean_squared_error(y_test, y_pred_test_neural))/
↪1000000
```

```
[17]: print(f"training, validation, and test errors for linear map = {tr_error_linear:
       ↪.5f}, {val_error_linear:.5f}, {test_error_linear:.5f}",
            f"training, validation, and test errors for ANN = {tr_error_neural:.5f},␣
       ↪{val_error_neural:.5f}, {test_error_neural:.5f}")
```

```
training, validation, and test errors for linear map = 34.85106, 24.92887,
11.05578 training, validation, and test errors for ANN = 43.12675, 28.52594,
2.93365
```

```
[ ]:
```