Linear Regression and Decision Tree Regressor for cost of living prediction

1. Introduction

Cost of living is an essential aspect to take into consideration from multiple standpoints and applications. For officers, this property indicates the current financial situations or development level. For the average citizen, this is among the most important factors in moving or immigrating decisions. However, while the cost of living of countries and large cities are available, this is not the case for smaller or minor cities because this property must be calculated from various elements (spending habits, goods prices, …). Therefore, this project aims to predict the cost of living of a city based on more readily available information.

This section briefly introduces the aim and application of this project, then Section 2 explains the problem formulation in more details, including data points, features and labels, and data sources. Afterwards, Section 3 discusses the methods for this problem, including models, loss functions and splitting of dataset for model validation. Section 4 then presents the results about the performance of such methods and Section 5 concludes the report with brief summary, result interpretation and some future directions.

2. Problem Formulation

The aim of this project is to predict the cost of living indicator of a city. With respect to the data source Numbeo [1], the cost of living in this project represents consumer goods prices, including groceries, restaurants, transportation and utilities, while excluding the rent. Also, the cost of living indicator is an evaluation of the cost of living in comparison with that of New York City, meaning that if the cost of living indicator of a specific city or country is 120, then the cost of living in that city or country would be 20% higher than that of New York City. In this project, the prediction for the cost of living indicator of a city will be made based on the population of the city and the cost of living indicator of its country.

Therefore, the data points in this project are cities, each characterized with properties including city name, country, population, cost of living indicator of the city and cost of living indicator of its country. The features are the population of the city (integer above 0) and the cost of living indicator of its country (numerical features). The label is the cost of living indicator of the city (numerical).

To summarize:

- Data point: a city
- Features: population of the city (people – numerical), country cost of living indicator (numerical)
- Label: city cost of living indicator (numerical)

As mentioned above, Numbeo is one of the data sources. The data for cost of living indicators of cities and countries can be extracted from ‘Cost of Living Index by City 2021 Mid-Year’, Section Cost of

Living Index on the website and ‘Cost of Living Index by Country 2021 Mid-Year’, Section Cost of Living by Country respectively. The statistics for those datasets are from Mid-year 2021.

The other source of data is Simplemaps [2] (‘https://simplemaps.com/data/world-cities’), from which the dataset for city population can be downloaded. This dataset was last updated in July 2021. Hence, the data from both sources is relatively close in time and up-to-date.

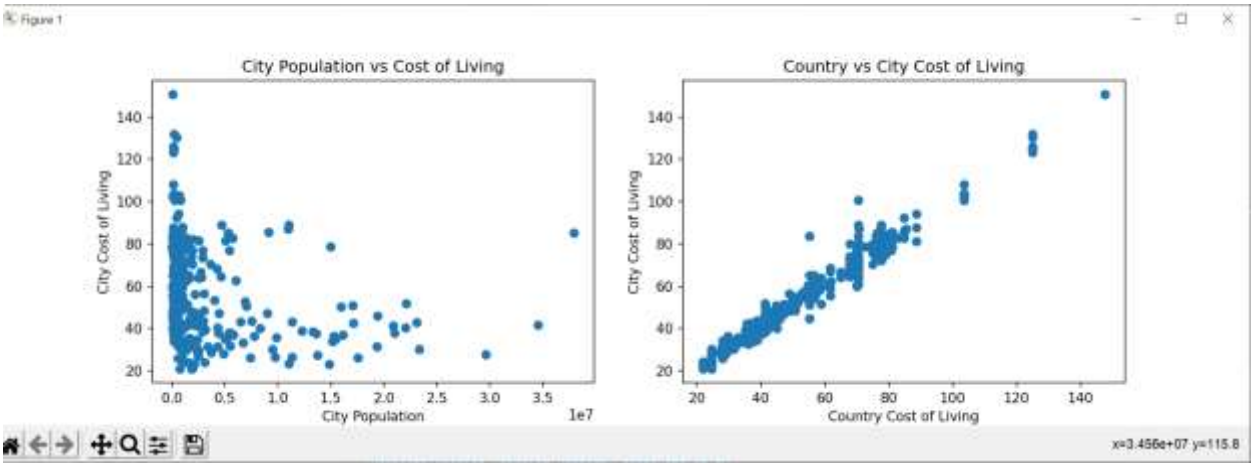
3. Methods

The sources are combined by the name and country of each city, then the unnecessary columns (not features nor labels) are discarded, creating the final data set (‘https://drive.google.com/file/d/18_W3fyrvkucvLhquS0Stsfw_wNPuPF89/view?usp=sharing’).

This dataset contains 352 data points, each characterized with properties including city name, country, city population, city cost of living indicator and country cost of living indicator. These properties are also the columns of this data set, amounting to a total of 5 columns.

Among the properties of the data points, the features chosen are the population of the city and the cost of living indicator of its country. Population of a city correlates with the cost of living in many aspects. One noticeable aspect is that population growth generally correlates with increased cost of living [3]. However, depending on the general economic situations (reflected with cost of living of the country), there are cases that low cost of living indicator correlates with high population due to population shifts [4] or poverty causing cost of living to be regulated [5].

Meanwhile, the cost of living indicator of the country can be a standard for trades, spending habits and hence, cost of living for cities in that country. Additionally, by plotting the data points, the cost of living indicator shows strong linear correlation with the quantity of interest (label).



As the plot demonstrates, the country cost of living shows a strong linear correlation with city cost of living. While the correlation between the city population and the cost of living indicator is not clear, the linear regression model would still be a suitable candidate for this ML problem and thus, a linear model is chosen. The label is calculated with:

$$\hat{y} = w_1 x + w_0$$

With y being the label, x being the features.

Another suitable model is the decision tree regressor. This model takes into account the highly non-linear correlation between the population of a city (feature) and its cost of living indicator (label). Decision tree regressor is also quite intuitive to interpret, for example by dividing ranges of the features into cases such as numerical ranges corresponding to "large" or "small" populations. It also provides a powerful tool to identify relevant variables and their relationships [6], which is useful in this case as the plotting cannot efficiently demonstrate the relationship of the features or the population feature with the label. In this project, decision tree models with max depths from 1 to 20 are used and validated, and the model with lowest validation error is selected.

The models would be fitted to the training dataset by minimizing mean squared error, which is calculated via the loss function:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - h(x_i))^2$$

With n being the number of data points, y_i and $h(x_i)$ being the label and the predicted label based on features of the i^{th} data point. This loss function is differentiable which yields high computational efficiency [7] when fitting the models to the data. It is suitable for both of the chosen models, especially the linear regression model. This loss function is also supported by the function mean-squared-error in the sci-kit learn package, and this same loss function can be used for training, validating and testing.

The dataset will be split (single split) into a test set (10%) and the remaining set (90%) for training and validating. This proportion allows sufficient data for training and model selection, while a test set has been created to evaluate overall performance of the final chosen model. The model selection will be based on k-fold validation. K-Fold validation reduces the possibility of choosing outliers for training and validation set [7], and thus is suitable for this small and diverse dataset. The dataset (remaining set) will be divided and folded with k = 5 (5 subsets, folded 5 times) to yield decently reliable results without requiring paramount time cost.

4. Results

For the linear regression model, the mean squared errors obtained with the CV iterations fluctuate quite considerably (details in appendix), with an average training error of 19.827687364910663 and average validation error of 20.08476216115988. For the decision tree models, the model with lowest training and validation errors before the models start to overfit (low training error but high validation error) has a max-depth of 5. This model has an average training error of 13.978593556539755 and average validation error of 28.840896377311577. While the training error of the linear regression model is higher than decision tree regression, the validation error of linear regression model is lower and thus this model is chosen.

The test set, which contains 10% of the original dataset, has been created while the remaining 90% of original data is used for K-Fold validation (described in last paragraph of Section 3). The final chosen model is obtained by fitting linear regression to the set for K-Fold cross validation and this model is

tested with the test set. The training mean squared error of this model is 18.83074604885515, while the test error is 12.68378981952182.

5. Conclusion

The report discusses the performance of linear regression and decision tree regression models for this Machine Learning problem. The mean squared error is chosen as the loss function, and K-Fold cross validation is used for model selection. Linear regression model has relatively close average training (19.827687364910663) and validation (20.08476216115988) errors; while the decision tree regression model with the lowest average validation error (max depth 5) has an average validation error (13.978593556539755) considerably higher than the training error (28.840896377311577), which hints overfitting.

Linear regression model is chosen with respect to its lower validation errors, and the final model has a training error of 18.83074604885515 and a test error of 12.68378981952182. With regards to the range of the label, these errors are relatively small as most data points in the current dataset have cost of living indicator (label) lower than 140 (demonstrated in the plot). It is also noteworthy that the test error is relatively close and even lower than the training error, which suggests that the model does not overfit and is likely to perform well on unseen data.

There are currently two directions to improve the methods for this problem. The first direction is to use and compare more models. For example, as this dataset contains outliers (demonstrated in the plot), Huber loss function might be useful when fitting the model to the data as this loss function is less sensitive to outliers. Another direction is to increase the number of features, such as population density, information about demographics, … However, both of these directions will require more data as the dataset now is still rather small to analyze correlations between properties of data points for determining suitable models or features.

6. Reference List

[1] Numbeo. https://www.numbeo.com/cost-of-living/

[2] Simplemaps. https://simplemaps.com/

[3] Sean Dennison, 2019. “The Cost of Overpopulation Around the World”. https://finance.yahoo.com/news/cost-overpopulation-around-world-090000594.html?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAJwQjnwU2Oll_rMaijvoiMyXSsn-M9WqNr1AS8CCia2MIVTZx9WNCZ4fbMkDlffc3vdSECD1cyeUyEhhEyGRJHjIBxky5RNMIB7IXGYALhSHDcMkwyeazivlc0-wg3hSiHy8F0gy0gow-oj0dpqA16EZa9vpqLxTdid8pa6psJQb

[4] Tim Henderson, 2016. “High Housing Costs Driving Population Shifts?”. https://www.pewtrusts.org/en/research-and-analysis/blogs/stateline/2016/04/08/high-housing-costs-driving-population-shifts

[5] Laura Bliss, 2016. “To Cut Down Poverty, Cut Down the Cost of Living”. https://www.bloomberg.com/news/articles/2016-08-04/center-for-neighborhood-technology-shows-that-reducing-household-costs-of-living-can-help-cut-poverty-levels

[6] Georgios Drakos, 2019. “Decision Tree Regressor explained in depth”. https://gdcoder.com/decision-tree-regressor-explained-in-depth/

[7] Coursebook, Machine Learning: The Basics. Chapter 6.2, Chapter 2.3

7. Appendix

Dataset

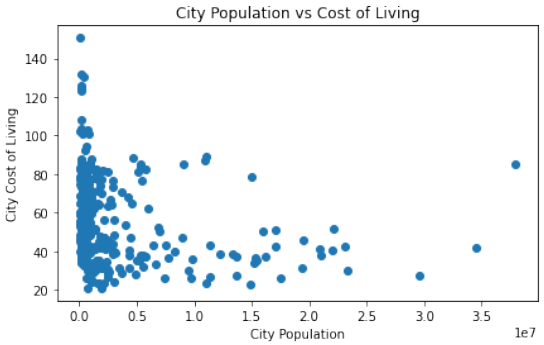
March 25, 2022

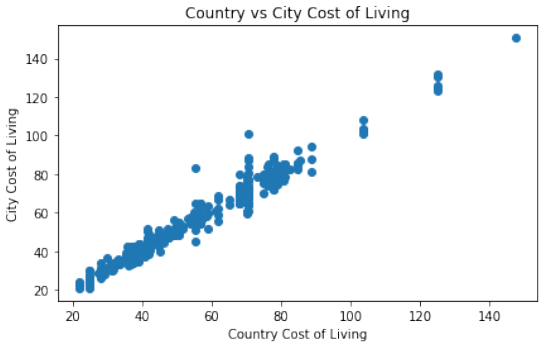
[5]:
```
import pandas as pd
import matplotlib.pyplot as plt

#Read datasets
df_cost_city = pd.read_excel("CityCostOfLivingIndex.xlsx")
df_cost_city = df_cost_city.dropna(axis = 0)
df_cost_city = df_cost_city[["City", "Cost of Living Index"]]
data_cost_city = []

for value in df_cost_city.values.tolist():
    city = value[0].split(", ")[0]
    country = value[0].split(", ")[-1]
    data_cost_city.append([city,country,value[1]])

df_cost_country = pd.read_excel("CountryCostOfLiving.xlsx")
df_cost_country = df_cost_country.dropna(axis=0)
df_cost_country = df_cost_country[["Country", "Cost of Living Index"]]
data_cost_country = df_cost_country.values.tolist()

df_population = pd.read_excel("CityPopulation.xlsx")
df_population = df_population.dropna(axis = 0)
df_population = df_population[["city_ascii", "country","population"]]
data_population = df_population.values.tolist()
```

[6]:
```
#Combine dataframes from different sources
class Datapoint:
    def __init__ (self, info):
        self.city = info[0]
        self.country = info[1]
        self.city_cost = info[2]
        self.country_cost = -1
        self.population = -1

    def findCountry (self, countryList):
        for country in countryList:
            if country[0] == self.country:
                self.country_cost = country[1]
```

```
    def findPopulation (self, populationList):
        for city in populationList:
            if city[0] == self.city and city[1] == self.country:
                self.population = city[2]

    def giveInfo (self):
        return [self.city, self.country, self.population, self.city_cost, self.
 ↪country_cost]

    def checkAppropriate(self):
        return not -1 in self.giveInfo()

all_data = []

for value in data_cost_city:
    newDatapoint = Datapoint(value)
    newDatapoint.findCountry(data_cost_country)
    newDatapoint.findPopulation(data_population)
    if newDatapoint.checkAppropriate():
        all_data.append(newDatapoint.giveInfo())

df = pd.DataFrame(all_data, columns=["City", "Country", "City Population",␣
 ↪"City Cost of Living", "Country Cost of Living"])
df.to_csv('CostOfLivingData.csv')
```

[7]:
```
#Visualize data
fig, axes = plt.subplots(1, 2, figsize=(15,4))
axes[0].scatter(df["City Population"],df["City Cost of Living"])
axes[0].set_xlabel("City Population")
axes[0].set_ylabel("City Cost of Living")
axes[0].set_title("City Population vs Cost of Living")

axes[1].scatter(df["Country Cost of Living"],df["City Cost of Living"])
axes[1].set_xlabel("Country Cost of Living")
axes[1].set_ylabel("City Cost of Living")
axes[1].set_title("Country vs City Cost of Living")

plt.show()
```


City Population vs Cost of Living
City Cost of Living
City Population
1e7


Country vs City Cost of Living
City Cost of Living
Country Cost of Living

[ ]:

Models

March 25, 2022

[86]:
```
#Import the necessary items from libraries

from statistics import mean
import numpy as np
import pandas as pd

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor

from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error
```

[87]:
```
#Load the data and divide the data into sets, initiate K-Fold cross validation

df = pd.read_csv('CostOfLivingData.csv')
df = df[["City", "Country", "City Population", "City Cost of Living", "Country␣
 ↪Cost of Living"]]

X = df[['City Population', 'Country Cost of Living']].values[:, 0:2]
y = df.values[:, 3]

X_rem, X_test, y_rem, y_test = train_test_split(X,y,test_size=0.
 ↪1,random_state=42)
kFold = KFold(n_splits=5, shuffle=True, random_state=42)
```

[88]:
```
#Linear regression model

tr_errors = []
val_errors = []
coefficients = []
intercepts = []
i = 0

print("===LINEAR REGRESSION===")
```

```
for (train_indices, val_indices) in kFold.split(X_rem):
    X_train, y_train, X_val, y_val = X[train_indices], y[train_indices],␣
 ↪X[val_indices], y[val_indices]

    lin_regr = LinearRegression()
    lin_regr.fit(X_train, y_train)

    y_pred_train = lin_regr.predict(X_train)
    tr_error = mean_squared_error(y_train, y_pred_train)
    y_pred_val = lin_regr.predict(X_val)
    val_error = mean_squared_error(y_val, y_pred_val)

    tr_errors.append(tr_error)
    val_errors.append(val_error)
    coefficients.append(lin_regr.coef_)
    intercepts.append(lin_regr.intercept_)
    i += 1

    print("\n\nCV iteration " + str(i))
    print("Training error: " + str(tr_error))
    print("Validation error: " + str(val_error))

print("\n\nAverage training error: " + str(sum(tr_errors)/5))
print("Average validation error: " + str(sum(val_errors)/5))
print("\nLearned weights: ",)
for coef in coefficients:
    print(coef)
print("\nLearned intercepts: " + str(intercepts))
```

```
===LINEAR REGRESSION===


CV iteration 1
Training error: 22.241421351907544
Validation error: 10.440545211933722


CV iteration 2
Training error: 17.43881582219404
Validation error: 29.883323836956745


CV iteration 3
Training error: 21.454969202113478
Validation error: 13.49497389745292
```

```
CV iteration 4
Training error: 17.436935128651196
Validation error: 29.540055921295867


CV iteration 5
Training error: 20.56629531968704
Validation error: 17.06491193816013


Average training error: 19.827687364910663
Average validation error: 20.084762161159876

Learned weights:
[1.84098082e-07 1.01210710e+00]
[1.63547091e-07 1.00679802e+00]
[1.59144116e-07 1.00953156e+00]
[1.80489106e-07 1.00907648e+00]
[1.87166681e-07 1.01172866e+00]

Learned intercepts: [0.4766260694986073, 0.5809287487540118, 0.802067402758226,
0.6757875643785525, 0.6151013780858676]
```

[89]:
```
#Decision tree models

tr_errors = []
val_errors = []
maxDepth = 1
print("===Decision tree regressor===")
while maxDepth < 21:
    tr_errors = []
    val_errors = []
    for (train_indices, val_indices) in kFold.split(X_rem):
        X_train, y_train, X_val, y_val = X[train_indices], y[train_indices],␣
 ↪X[val_indices], y[val_indices]

        tree_reg = DecisionTreeRegressor(max_depth=maxDepth)
        tree_reg.fit(X_train,y_train)

        y_pred_train = tree_reg.predict(X_train)
        tr_errors.append(mean_squared_error(y_train, y_pred_train))
        y_pred_val = tree_reg.predict(X_val)
        val_errors.append(mean_squared_error(y_val, y_pred_val))
    print("\n\nDecision tree max depth " + str(maxDepth) + ":")
    print("Average training error: " + str(sum(tr_errors)/5))
    print("Average validation error: " + str(sum(val_errors)/5))
    maxDepth += 1
```

```
===Decision tree regressor===


Decision tree max depth 1:
Average training error: 137.62533666126706
Average validation error: 143.1095726113014


Decision tree max depth 2:
Average training error: 49.74042874006814
Average validation error: 57.27070094432911


Decision tree max depth 3:
Average training error: 24.97944589061118
Average validation error: 33.3612001992028


Decision tree max depth 4:
Average training error: 17.563827324565757
Average validation error: 31.525029876085267


Decision tree max depth 5:
Average training error: 13.978593556539755
Average validation error: 28.840896377311584


Decision tree max depth 6:
Average training error: 11.659275778211022
Average validation error: 30.252709235612247


Decision tree max depth 7:
Average training error: 8.565431364307516
Average validation error: 32.300985055808816


Decision tree max depth 8:
Average training error: 5.798310270416641
Average validation error: 31.674608671914136


Decision tree max depth 9:
Average training error: 3.581709861092757
Average validation error: 31.411233058883333
```

```
Decision tree max depth 10:
Average training error: 2.174080546771098
Average validation error: 34.02196793905648


Decision tree max depth 11:
Average training error: 1.06965910939624
Average validation error: 34.30274322690223


Decision tree max depth 12:
Average training error: 0.42770548647186146
Average validation error: 34.814670445053146


Decision tree max depth 13:
Average training error: 0.19138248393876658
Average validation error: 34.977840428835975


Decision tree max depth 14:
Average training error: 0.07062659981282807
Average validation error: 35.80213085079365


Decision tree max depth 15:
Average training error: 0.025055060281908088
Average validation error: 35.279066647927685


Decision tree max depth 16:
Average training error: 0.013617410282953787
Average validation error: 33.550994176587295


Decision tree max depth 17:
Average training error: 0.0
Average validation error: 34.40774318948412


Decision tree max depth 18:
Average training error: 0.0
Average validation error: 34.383307951388886


Decision tree max depth 19:
Average training error: 0.0
Average validation error: 34.51865899801587
```

```
Decision tree max depth 20:
Average training error: 0.0
Average validation error: 33.250023020833325
```

[90]:
```
#After comparing the errors, linear regression model is chosen.
#The final model is obtained by fitting to the set for K-Fold validation (90%).

print("===Final model===")

lin_regr = LinearRegression()
lin_regr.fit(X_rem, y_rem)
y_pred_rem = lin_regr.predict(X_rem)
tr_error = mean_squared_error(y_rem, y_pred_rem)
print("\nTraining error: " + str(tr_error))

y_pred_test = lin_regr.predict(X_test)
test_error = mean_squared_error(y_test, y_pred_test)
print("Testing error: " + str(test_error))
```

```
===Final model===

Training error: 18.83074604885515
Testing error: 12.68378981952182
```

[ ]:
```

```