COURSE PROJECT

OBSERVATIONS FROM PLACES & THEIR NAMES

CHAPTER 3 – STATS & MACHINE LEARNING

GROUP 1

LÉONARD DE VINCI

GAPMINDER COUNTRIES REGRESSION

GROUP 1

1895

LÉONARD DE VINCI

country-related datasets

- agriculture_GDP.csv
- agriculture_land.csv
- children_per_women.csv
- energy_production.csv
- forest_coverage.csv
- gdp_growth.csv
- hiv_adults.csv
- imports.csv
- income_pp.csv
- male_blood_pressure.csv
- tax.csv

Firstly we combine other country-related data sets with previous geographic data sets and get a new data sets which have many information.

Then we select ‘GDP’ as the variable that we want to predict, and eight related variables as our input.

When we get the train sets and test sets, we consider using six models to make a regression:

- “LinearRegression”
- “DecesionTree”
- “Kneibor”
- “AdaBoostRegressor”
- “GBRTRegression(Gradient Boosting)”
- “Extra Tree”

For each model, we give the Contrast Curve and their Score and input their MSE and RMSE:

DecesionTree:
MSE: 0.0855068678174249 5
RMSE: 0.29241557382845557

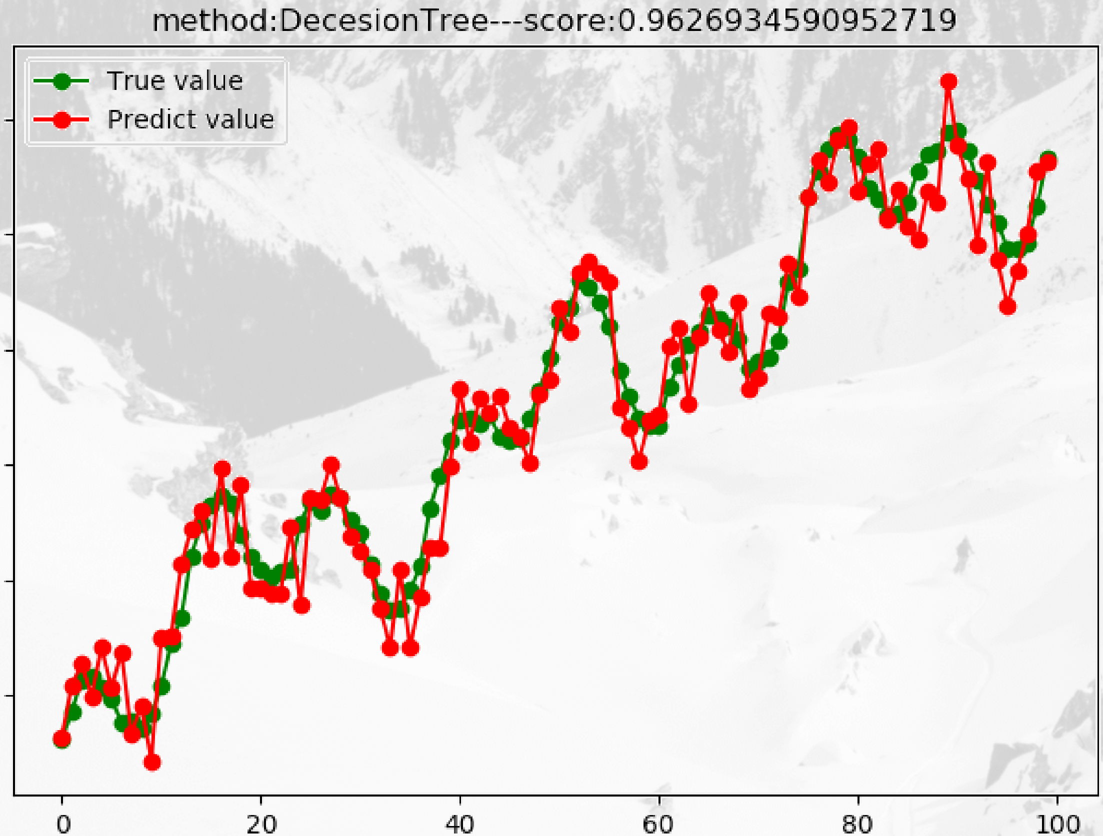
AdaBoostRegressor :
MSE: 0.0636497279190832
RMSE: 0.25228897700669206

Kneibor :
MSE: 0.017216139008 81275
RMSE: 0.1312102854535906 7

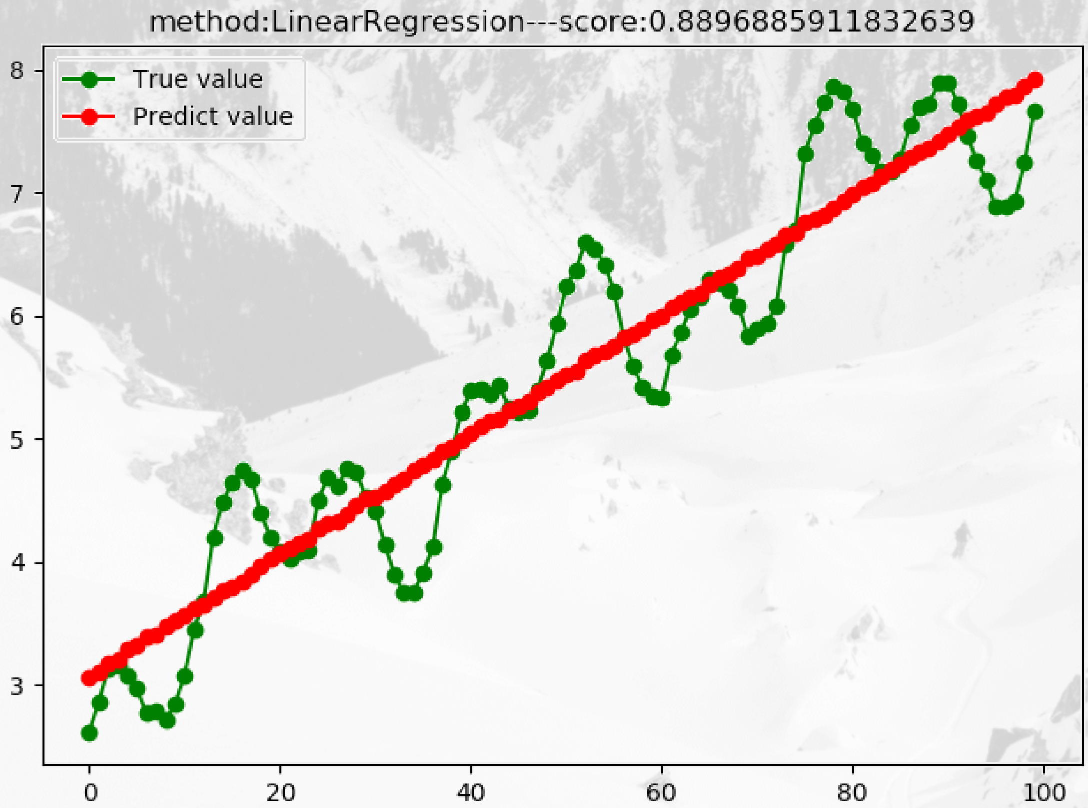
LinearRegression :
MSE: 0.25283456529873943
RMSE: 0.5028265757681663

GBRTRegression :
MSE: 0.013758100596726035
RMSE: 0.11729492997025079

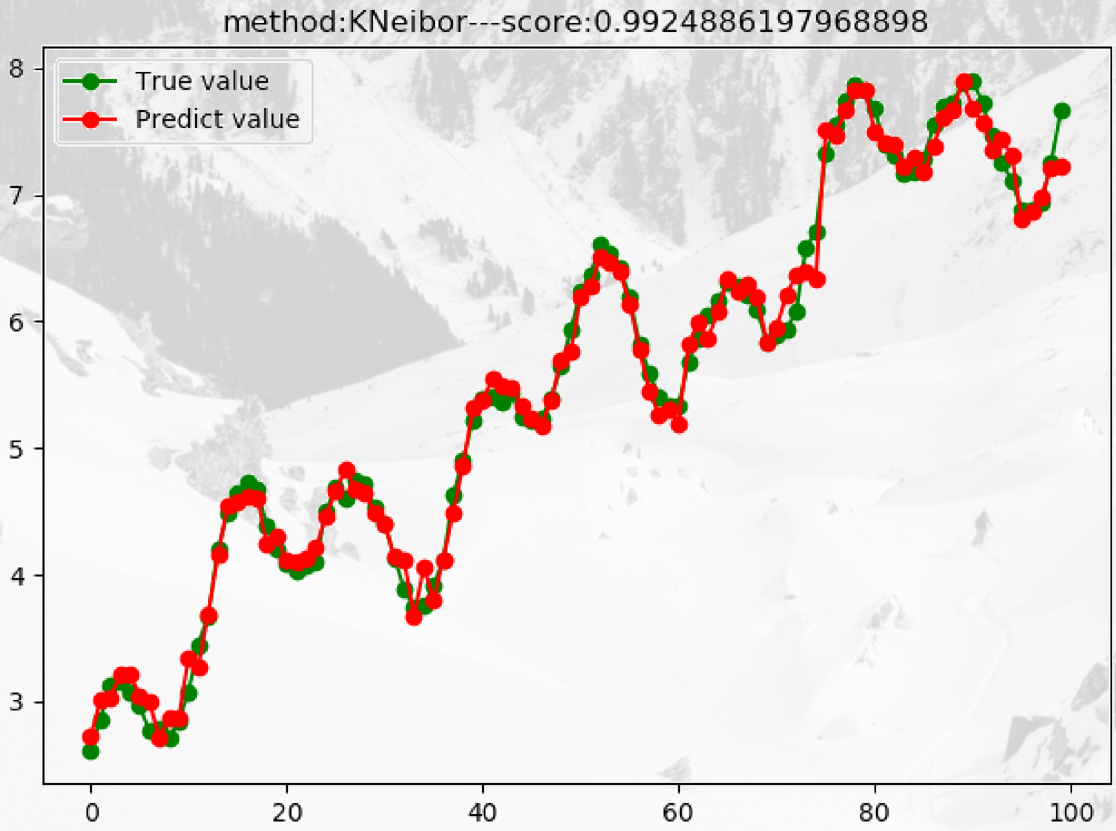
ExtraTree :
MSE: 0.0876803445908834
RMSE: 0.29610867023929477



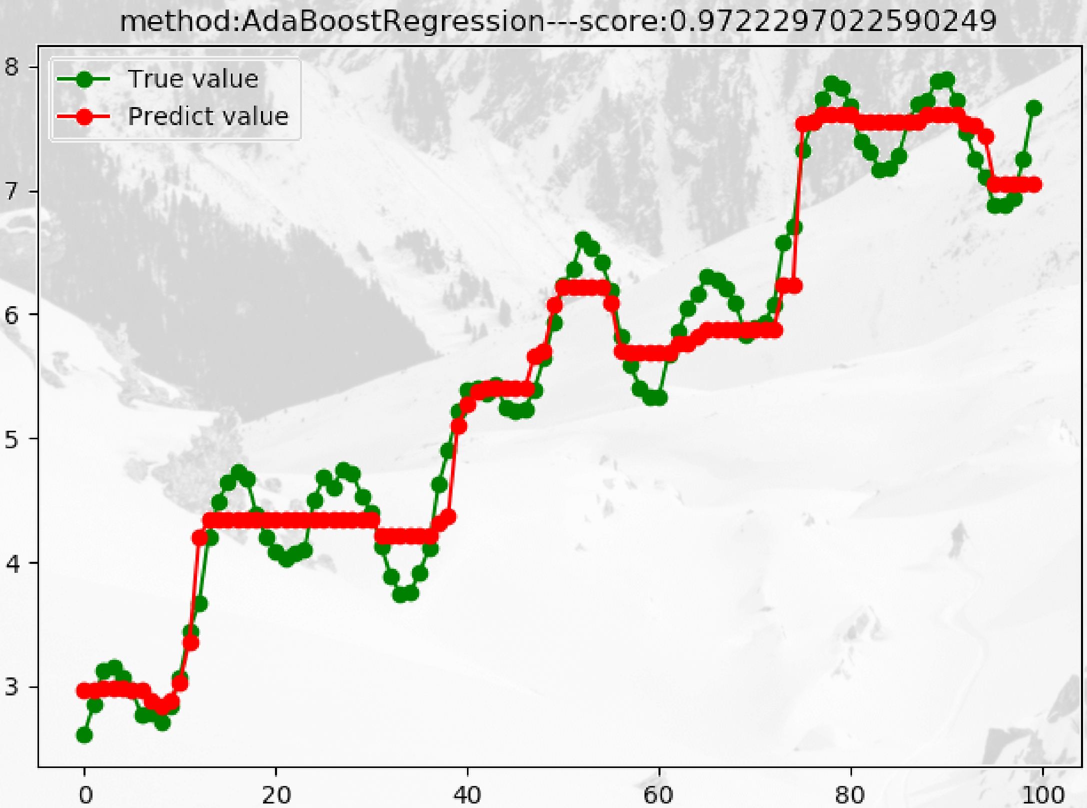
From these images, we can see that the most suitable method for our data is GBRTRegression.



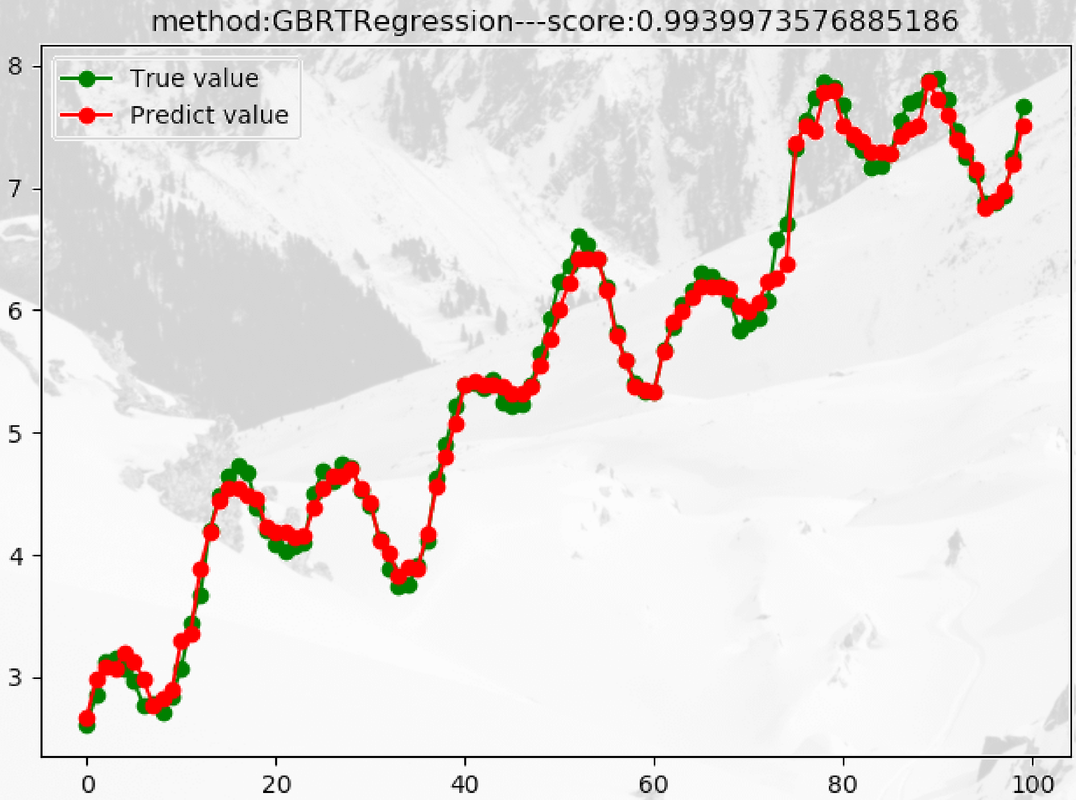
From these images, we can see that the most suitable method for our data is GBRTRegression.



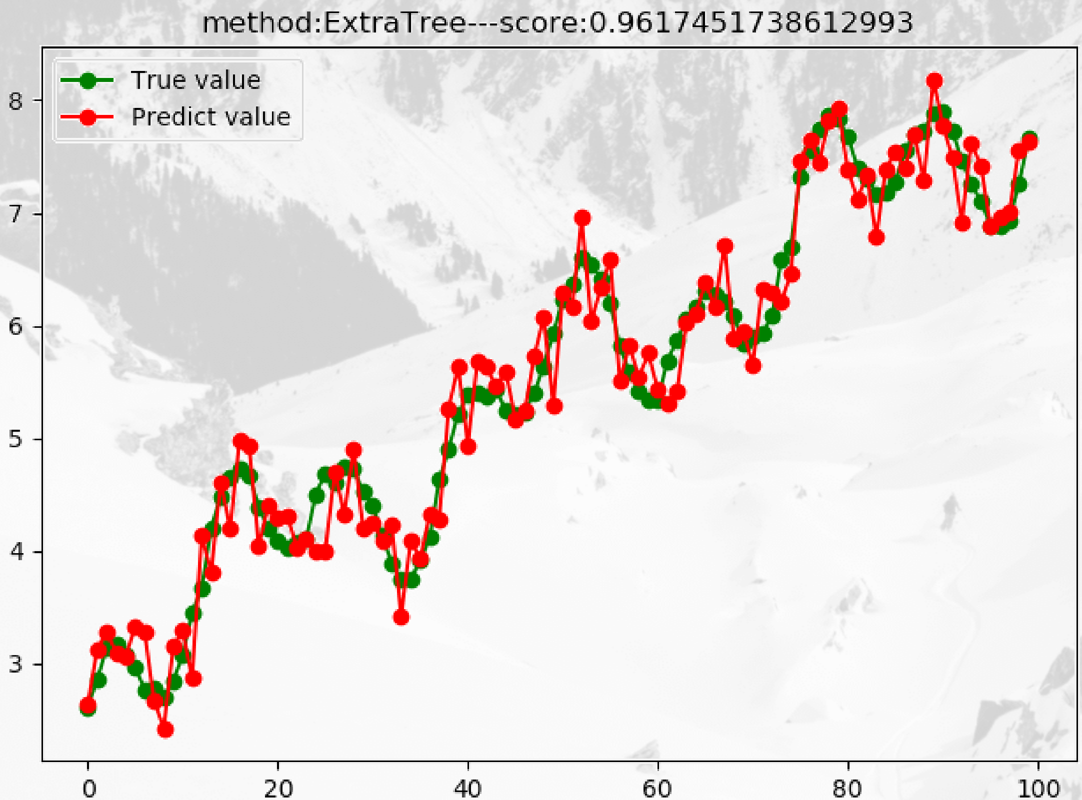
From these images, we can see that the most suitable method for our data is GBRTRegression.



From these images, we can see that the most suitable method for our data is GBRTRegression.



From these images, we can see that the most suitable method for our data is GBRTRegression.



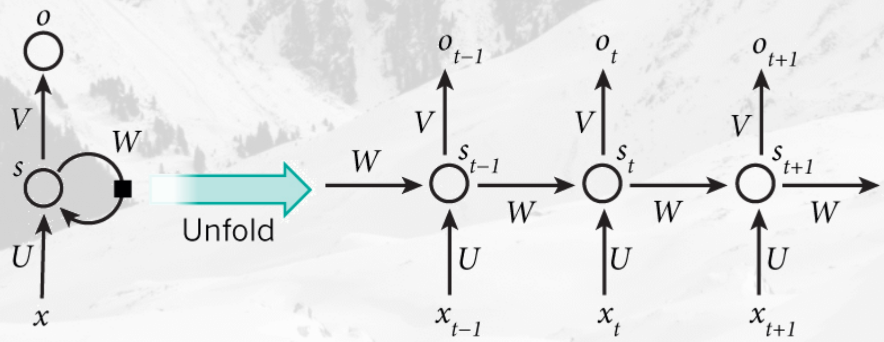
From these images, we can see that the most suitable method for our data is GBRTRegression.

GEONAMES CLASSIFICATION

GROUP 1

EastAsia CN HK JP KP KR LA MO TW VN #
S&SEAsia BD BT BN CC ID IN KH LK MM MV MY NP PH SG TH TL #
EnUsAuNz AU CA CX FK IM IO NZ US VG VI #
Latinos AG AI AR AW BB BL BO BR BZ CL CO CR CU CW … VE #
Arabics AE AF BH DZ EG EH IL IQ IR JO KG KW KZ LB LY OM … #
WEurope AD AL AT BE CH DE DK FI FO FR GL GR HR IE IS IT LI LU MC … VA #
EEurope AM AZ BA BG BA BY CY CZ EE GE HU LT LV MD ME MK …#
Oceania AS BM CK FJ FM KI NR PG PW TK TO TV WS #
SSAfrica AO BF BI BJ BW CD CF CG CI CM CV DJ ER ET GA … #

Two significant drawbacks for a simple feedforward NN model:
- Each time the output of the network depends only on the current input, without considering the effects of the previous several inputs;
- The dimensions of the input and output are fixed, which is not efficient for variable length sequential data – like text.



Recurrent Neural Network (RNN) solves the above problem. It is a variety of neural network that is used specifically for processing time series data samples. The model remembers previous recorded data and utilize the whole serial relations.

```
def lineToTensor(line):
    tensor = torch.zeros(len(line), 1, n_letters)
    for li, letter in enumerate(line):
        tensor[li][0][letterToIndex(letter)] = 1
    return tensor
```

A

1 0 0 0 0 0 ... 0

BAD

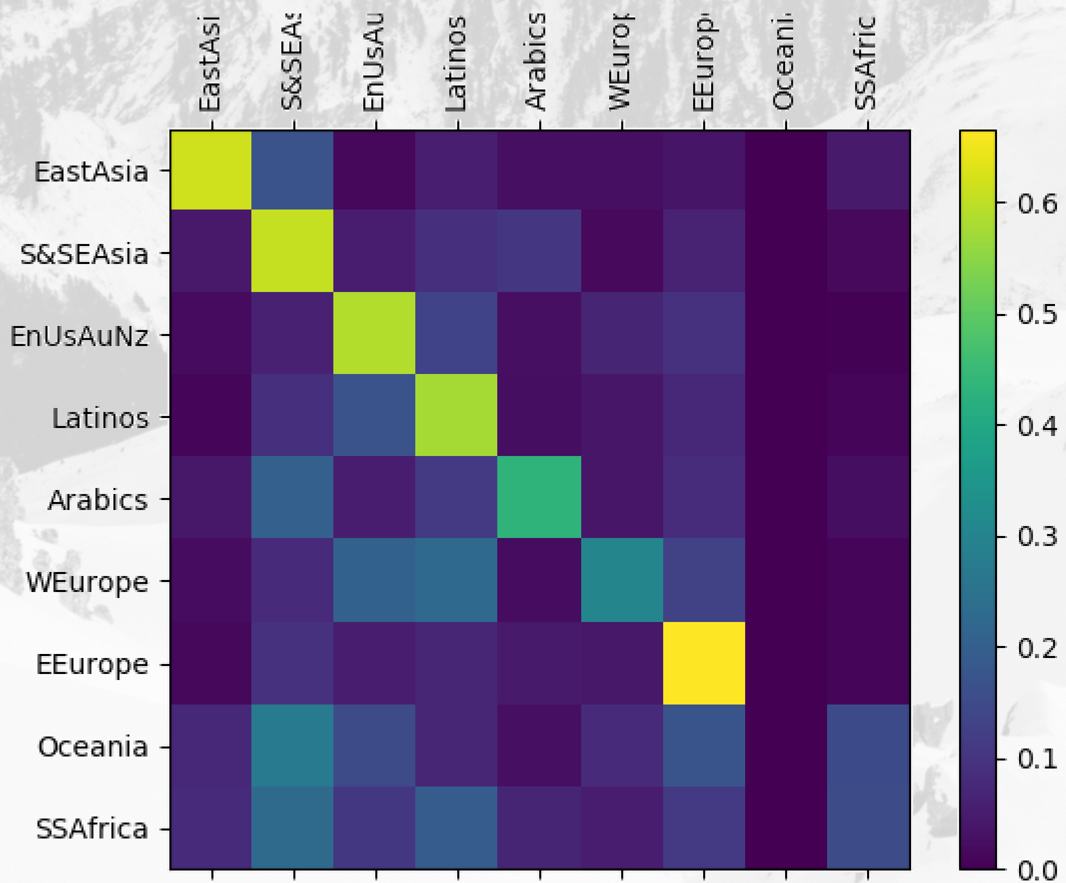
0 1 0 0 0 0 ... 0
1 0 0 0 0 0 ... 0
0 0 0 1 0 0 ... 0

For each iteration of training, the following process is executed:

- Create input and target tensors
- Create a zeroed initial hidden state
- Read each letter in and Keep hidden state for next letter
- Compare final output to target
- Back-propagate
- Return the output and loss

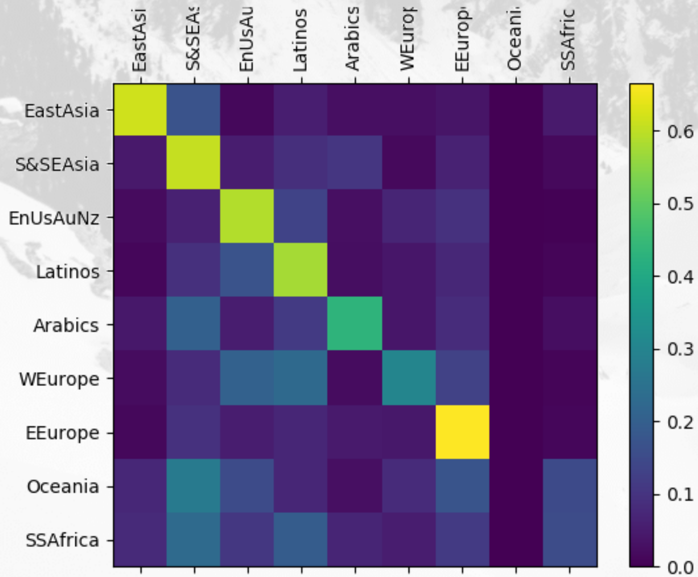
```
D:\Develop\Python\GeonamesN\code
λ py train.py
Average loss: 2.187102
Average loss: 2.122357
Average loss: 2.080929
Average loss: 2.049070
500 1% (0m 1s) 2.1564 fatukanutu / WEurope ✗ (S&SEAsia)
Average loss: 2.025370
Average loss: 1.934250
Average loss: 1.817812
Average loss: 1.787719
Average loss: 1.860767
1000 2% (0m 2s) 1.1032 mailly-le-chateau / WEurope ✓
Average loss: 1.839910
Average loss: 1.760699
Average loss: 1.923570
Average loss: 1.704505
Average loss: 1.790910
1500 3% (0m 4s) 0.7731 colonia la joya / Latinos ✓
Average loss: 1.653982
Average loss: 1.756644
Average loss: 1.746588
Average loss: 1.753437
Average loss: 1.815430
2000 5% (0m 5s) 1.4857 manau / WEurope ✓
Average loss: 1.696322
Average loss: 1.754698
Average loss: 1.754979
Average loss: 1.828953
Average loss: 1.792830
2500 6% (0m 7s) 2.0786 xiayang / WEurope ✗ (EastAsia)
Average loss: 1.762827
Average loss: 1.716784
Average loss: 1.844258
Average loss: 1.782177
Average loss: 1.777431
3000 7% (0m 8s) 1.2567 archigny / WEurope ✓
```

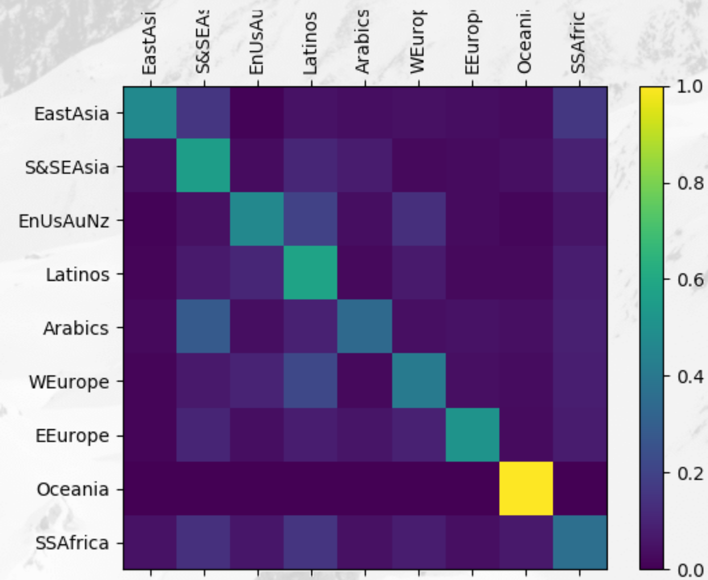
```
D:\Develop\Python\GeonamesN\code
λ python predict.py Fancheng

> fancheng
(-0.63) EastAsia
(-1.82) WEurope
(-2.26) S&SEAsia

D:\Develop\Python\GeonamesN\code
λ python predict.py Nah Truong

> nah truong
(-1.26) S&SEAsia
(-1.43) EnUsAuNz
(-1.72) WEurope

D:\Develop\Python\GeonamesN\code
λ python predict.py Vladimirkoiszavk

> vladimirkoiszavk
(-0.04) EEurope
(-4.60) WEurope
(-4.63) Arabics

D:\Develop\Python\GeonamesN\code
λ python predict.py Rio Je Satino

> rio je satino
(-0.49) Latinos
(-2.21) WEurope
(-2.40) EnUsAuNz

D:\Develop\Python\GeonamesN\code
λ python predict.py Parisburg

> parisburg
(-1.04) WEurope
(-1.51) S&SEAsia
(-2.05) EnUsAuNz
```


EastAsi
S&SEA:
EnUsAu
Latinos
Arabics
WEuroı
EEurop
Oceani.
SSAfric
EastAsia
S&SEAsia
EnUsAuNz
Latinos
Arabics
WEurope
EEurope
Oceania
SSAfrica
0.6
0.5
0.4
0.3
0.2
0.1
0.0

```
D:\Develop\placename-insights\back-end\classification\code (master -> origin)
λ py plot.py
Confusion Matrix:
tensor([261.,  64.,  13.,  28.,   5.,  16.,   6.,   0.,  35.])
tensor([ 26., 250.,  22.,  67.,  19.,  23.,   5.,   0.,  16.])
tensor([  3.,  21., 251.,  93.,   3.,  60.,   2.,   0.,   9.])
tensor([  9.,  42.,  69., 261.,   4.,  24.,   4.,   0.,  11.])
tensor([ 20., 152.,  36.,  60., 105.,  39.,  19.,   0.,  25.])
tensor([ 12.,  32.,  84., 123.,   3., 165.,  21.,   0.,  14.])
tensor([ 15.,  52.,  38.,  75.,  10.,  48., 204.,   0.,  13.])
tensor([ 16., 115.,  28., 157.,   0.,  30.,  18.,   0., 106.])
tensor([ 42.,  72.,  48., 112.,  18.,  44.,   9.,   0.,  98.])
```

```
# Before Modification
category = randomChoice(all_categories)
line = randomChoice(category_lines[category])

# After Modification
tuple = randomChoice(all_lines)
line = tuple[0]
category = tuple[1]
```


EastAsi
S&SEA
EnUsAu
Latinos
Arabics
WEurop
EEurop
Ocean
SSAfric
EastAsia
S&SEAsia
EnUsAuNz
Latinos
Arabics
WEurope
EEurope
Oceania
SSAfrica
0.0
0.1
0.2
0.3
0.4
0.5
0.6


EastAsi
S&SEA
EnUsAu
Latinos
Arabics
WEurop
EEurop
Ocean
SSAfric
EastAsia
S&SEAsia
EnUsAuNz
Latinos
Arabics
WEurope
EEurope
Oceania
SSAfrica
0.0
0.2
0.4
0.6
0.8
1.0

THANKS FOR YOUR TIME

GROUP 1

1895 × LÉONARD DE VINCI