

Question 2

1 Team

Team Members	<i>Shuyang Ye</i>	<i>Jinlin Zhu</i>	Kaggle Team Name	<i>TESNO.1</i>
Student ID	<i>96481163</i>	<i>96057468</i>		
CS ID	<i>l8n8s</i>	<i>q6j8i</i>		

2 Solution Summary

Three different methods are adopted individually. The most intuitive and simplest model is to use only Canada's past deaths to predict future deaths. The predicted future deaths curve is a straight line which means the daily deaths is a constant. The second method calculates the daily deaths and adopts Auto Regression model to predict the future daily deaths and then the future total deaths. In the end, I stack this two method together to predict the final result.

Third method is to consider the impact from other countries. Given that case density is more informative than the total number of cases for national outbreak correlation analysis, we used the total national death toll density as an indicator to calculate the **Cross Correlation** between different countries, thus selecting countries with similar outbreaks. Thus we can establish a linear map between deaths density of other countries and Canada. Using AR model own each other countries to predict the future deaths, then using the linear map to predict the deaths of Canada.

Finally we adopted the third method that is to combine factors of impact from other countries and the spec country "Canada" together.

3 Experiments

The formula we used to predict the future deaths of Canada is $\text{Deaths}(\text{CA}, \text{Future}) = \text{Num_people_100k} * \text{Deaths_100k}(\text{CA}, \text{Future})$ and

3.1 Auto Regression Model

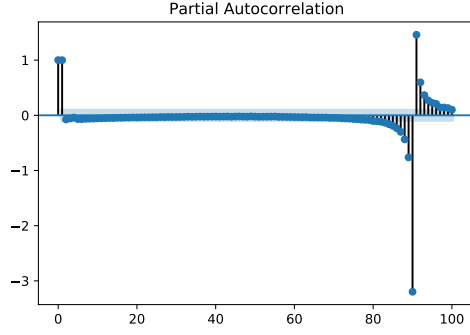
As for the final method, we used this model to predict the short term future of feature "deaths_100k" of all countries. We coded two classes called My_AutoReg and AutoReg1 in "AutoRegression.py" file based on the same algorithm provided by instructions of the Question 2.

3.2 Select lags

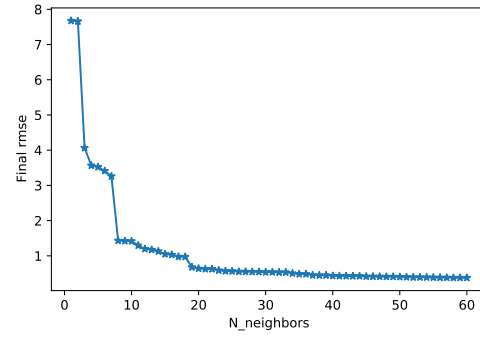
The main hyper-parameter of AR model is the lags we used to generate the Matrix X, by which we predict the value of next day based on the past "num=lags" days. We choose lags by plotting the Partial Autocorrelation Plot (Figure (a)), we can see that only first two past days have significant impact on the value of the present day. Thus we choose **Lags = 2**. We adopted the same lags for all countries future predictions.

3.3 Multivariable Linear Regression Model

We adopted Linear Regression for feature "deaths_100k" of all NearestNeighbors Countries as X and the feature "deaths_100k" of Canada as y. Thus we could get a Linear relationship and could make "Parallel" Predictions to the death density of Canada.



(a) PACF plot of feature "deaths_100k" of Canada



(b) Training Error vs N_Nearest_Neighbors plot

3.4 Select N Nearest Neighbors

We finally used the **Cross Correlation** to measure "distances" between other countries. This distances is actually the similarities. We noticed that the deaths density between two selected countries have significantly higher correlation. However, the starting date of the pandemic outbreak of each country is different by days. So we created a loop to shift the deaths_100k data from other countries by "num=lag_country" days until they has their best correlation with the spec country "CA".

We used these nearest countries to implement a multivariable linear regression from their feature "deaths_100k" as Input Matrix X to the feature "deaths_100k" of the spec country "CA" as y vector. We selected the period of 269 days as training set and newest 11 days as validation set. We plotted training error plots vs N_Nearest_Neighbors as Figure (b). For training phase, we adopted "elbow" method and find approximate best num is 7 or 8 or 9. We chose **N_Nearest_Neighbors = 7** in the end.

4 Results

Team Name	Kaggle Phase 1 Score	Kaggle Phase 2 Score
<i>TESNO.1</i>	<i>10.865</i>	<i>459.16159</i>

5 Conclusion

The final result is good enough according to our very simple model. We have learned how to code the Auto Regression model, how to select features and hyper-parameters of our models in practice. We could try more features and created more sophisticated modesl if given more time.

6 Code

```
1
2 import argparse
3 import os
4 import pandas as pd
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import time
8 from pandas.plotting import autocorrelation_plot
9 from statsmodels.graphics.tsaplots import plot_acf
10 from statsmodels.graphics.tsaplots import plot_pacf
11 #from statsmodels.tsa.ar_model import AutoReg
12
13 from statsmodels.tsa import stattools
14 from sklearn.metrics import mean_squared_error
15 from scipy.stats import pearsonr
16 from math import sqrt
17 import seaborn as sns
18 #from sklearn.neighbors import NearestNeighbors
19 import linear_model
20 import utils
21 import country
22 import warnings
23 from AutoRegression import My_AutoReg
24 from AutoRegression import AutoReg1
25
26 if __name__ == '__main__':
27     parser = argparse.ArgumentParser()
28     parser.add_argument('-q', '--question', required=True)
29     parser.add_argument('-d', '--test_day', required=False)
30     parser.add_argument('-l', '--test_lag', required=False)
31     parser.add_argument('-n', '--test_num', required=False)
32     parser.add_argument('-m', '--test_method', required=False)
33     parser.add_argument('-p', '--test_par', required=False)
34
35     io_args = parser.parse_args()
36     question = io_args.question
37
38
39     def read_dataset(filename):
40         with open(os.path.join("../", "data", filename), "rb") as f:
41             dataset = pd.read_csv(f, keep_default_na=False)
42         return dataset
43
44     if question == 'find_nan_ct':
45         dataset = read_dataset("phase1_training_data.csv")
46         n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind = utils.make_mapper(dataset)
47         dataset = utils.get_all(dataset)
48         dataset_CA = dataset[dataset[ct_key] == "CA"]
49
50         print("Input Country Number:", n_ct)
51         n_ct1 = 0
52         i = 0
53         ct_nan_ind = n_ct * [None]
54         for nct in range(n_ct):
55             if type(ct_inv_mapper[nct]) == str:
56                 n_ct1 += 1
57             else:
58                 ct_nan_ind[i] = nct
59                 i += 1
60         print("Output Country Number:", n_ct1)
61         if n_ct1 == n_ct:
62             print("No 'nan' named country found!")
63
```

```

64 if question == 'lagplot':
65     dataset = read_dataset("phase1_training_data.csv")
66     dataset=utils.get_all(dataset)
67     n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
68
69     dataset_CA=dataset[dataset[ct_key]=='CA']
70     #cases_CA = dataset_CA["cases"]
71     #deaths_CA = dataset_CA["deaths"]
72
73     testname = "deaths_100k"
74     test_CA = dataset_CA[testname]
75     #autocorrelation_plot(cases_CA)
76
77     #pacf analysis
78     nlags = 300
79     #conf_CA = np.zeros(2)
80     pacf_CA, conf_CA = stattools.pacf(test_CA, nlags=nlags,alpha=0.05)
81     #pacf1 = pacf_CA[pacf_CA > conf_CA[1] or pacf_CA < conf_CA[0]]
82     conf0 = conf_CA[:,0]
83     conf1 = conf_CA[:,1]
84     ind_lags = np.argsort(pacf_CA)
85     ind_lags = ind_lags[::-1]
86
87     #pacf_lag_plot for test dataset with testname
88
89     nlags_max =100
90     #pacf output
91     plot_pacf(test_CA,lags=nlags_max)
92     #plotname = testname+"_pacf_lag_plot_CA.pdf"
93     plotname = testname+"_pacf_lag_plot_CA.pdf"
94     fname = os.path.join("..", "figs", plotname)
95     #plt.xlim(left = 80)
96     plt.savefig(fname)
97     plt.clf()
98
99     plt.plot(conf0)
100    plt.plot(conf1)
101    plotname = "conf.pdf"
102    fname = os.path.join("..", "figs", plotname)
103    plt.xlim(left = 80)
104    plt.savefig(fname)
105    plt.clf()
106
107    testname=testname
108    refname="date"
109    sns.scatterplot(dataset_CA[refname],dataset_CA[testname])
110    plotname = testname+"_CA.pdf"
111    fname = os.path.join("..", "figs", plotname)
112    plt.xlim(left = 50)
113    plt.savefig(fname)
114    plt.clf()
115
116 if question == 'ar_ca':
117     dataset = read_dataset("phase1_training_data.csv")
118     #Input arg parameters
119     test_lag = io_args.test_lag
120     test_lag = int(test_lag)
121     test_day= io_args.test_day
122     test_day = int(test_day)
123
124     n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
125
126     dataset_CA=dataset[dataset[ct_key]=='CA']
127     cases_CA = dataset_CA["cases"]
128     deaths_CA = dataset_CA["deaths"]

```

```

129
130     #Choose experiment set
131     exp_set = deaths_CA
132     exp_set = exp_set.reset_index(drop = True)
133
134     X = exp_set
135     train_set, test_set = X[:len(X)-test_day], X[len(X)-test_day:]
136
137     # train autoregression
138     lags=test_lag
139     model1 = AutoReg1(lags=lags)
140     model1.fit(train_set)
141
142     model3 = My_AutoReg(lag=lags)
143     model3.fit(train_set)
144     #print('Coefficients: %s' % model_fit.params)
145
146     # make predictions
147     predictions_train= model1.predict(start=lags, end=len(train_set)-1)
148     # predictions_train= model3.predict(train_set,start=lags, end=len(train_set)-1)
149     # predictions_test= model.predict(start=len(train_set), end=len(train_set)+len(
test_set))
150     # predictions_all= model.predict(start=lags, end=len(train_set)+len(test_set))
151     #Train Error
152     rmse_train = sqrt(mean_squared_error(train_set[lags:], predictions_train))
153     print('Train RMSE: %.3f' % rmse_train)
154     # rmse_test = sqrt(mean_squared_error(test_set, predictions_test))
155     # print('Test RMSE: %.3f' % rmse_test)
156     # plot results
157     plt.plot(train_set[lags:])
158     plt.plot(predictions_train, color='red')
159     fname = os.path.join(".", "figs", "prediction_train_plot_CA.pdf")
160     plt.savefig(fname)
161     plt.clf()
162
163     # plt.plot(test_set)
164     # plt.plot(predictions_test, color='red')
165     # fname = os.path.join(".", "figs", "prediction_test_plot_CA.pdf")
166     # plt.savefig(fname)
167     # plt.clf()
168
169
170     # plt.plot(exp_set[lags:])
171     # plt.plot(predictions_all, color='red')
172     # fname = os.path.join(".", "figs", "prediction_all_plot_CA.pdf")
173     # plt.xlim(left = len(exp_set[lags:]) -2*len(test_set))
174     # plt.savefig(fname)
175     # plt.clf()
176     # plt.close('all')
177
178     # if question == 'knn':
179     #     dataset = read_dataset("phase1_training_data.csv")
180     #     test_num= io_args.test_num
181     #     test_num= int(test_num)
182     #     test_method= io_args.test_method
183     #     #test_method=int(test_method)
184
185
186     #     n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
187
188     #     dataset_CA=dataset[dataset[ct_key]=='CA']
189     #     cases_CA = dataset_CA["cases"]
190     #     cases14_CA =dataset_CA["cases_14_100k"]
191     #     day_n = len(cases_CA)
192

```

```

193 #     exp_name = "cases_14_100k"
194 #     exp_dataset = np.zeros((n_ct, day_n))
195 #     exp_ct_name = 'CA'
196
197 #     #Choose the dataset regarding the name interested
198 #     cases14_exp = dataset[dataset[ct_key]==exp_ct_name][exp_name]
199
200 #     for nct in range(1,n_ct):
201 #         #fillter out nan set
202 #         dataset_ct = dataset[dataset[ct_key]==ct_inv_mapper[nct]]
203 #         dataset1 = dataset_ct[exp_name]
204 #         exp_dataset[nct] = dataset1
205
206 #     exp_dataset = exp_dataset[1:]
207 #     target= np.zeros((1, day_n))
208 #     target[0]= cases14_exp
209 #     #!!!!!!remember the true country index is ind_ct = indices+1
210 #     #Using KNN model to find K nearest countries
211 #     n_neigh = test_num
212
213 #     metric_method =test_method
214 #     model = NearestNeighbors(metric=metric_method)
215 #     model.fit(exp_dataset)
216 #     distances, indices=model.kneighbors(target, n_neighbors=n_neigh)
217 #     #!!!!!!remember the true country index is ind_ct = indices+1
218 #     indices = indices +1
219 #     indices = indices[0]
220 #     distances = distances[0]
221 #     ct_neigh =n_neigh*[None]
222 #     for i in range(n_neigh):
223 #         ct_neigh[i] = ct_inv_mapper[indices[i]]
224 #     #ct_neigh = [ct_inv_mapper[i] for i in range(len(indices))]
225 #     ct_neigh_dict=dict(zip(ct_neigh, list(distances)))
226 #     print(ct_neigh_dict)
227
228 #     #Distance Plot of N_nearest countreis
229 #     plt.plot(distances[1:])
230 #     plt.xlabel("Neighbor_Country")
231 #     plt.ylabel("Distance")
232 #     plotname="Distance_plot_n"+str(n_neigh)+"_"+exp_ct_name+"_"+metric_method+".pdf"
233 #     fname = os.path.join(".", "figs", plotname)
234 #     plt.savefig(fname)
235 #     plt.clf()
236 #     plt.close('all')
237
238 if question == "linear":
239     test_par = io_args.test_par
240
241     dataset = read_dataset("phase1_training_data.csv")
242     dataset = utils.get_all(dataset)
243     n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
244
245     dataset_CA=dataset[dataset[ct_key]=='CA']
246     cases_CA = dataset_CA["cases"]
247     cases14_CA =dataset_CA["cases_14_100k"]
248     day_n = len(cases_CA)
249
250     ref_name = "deaths_100k"
251     exp_name = "deaths_100k"
252
253     #exp_dataset = np.zeros((n_ct, day_n))
254     exp_ct_name = 'CA'
255     ref_ct_name = 'UK'
256
257     exp_dataset= dataset[dataset[ct_key]==exp_ct_name][exp_name]

```

```

258     ref_dataset= dataset[dataset[ct_key]==ref_ct_name][ref_name]
259     exp_dataset = exp_dataset.reset_index(drop = True)
260     ref_dataset = ref_dataset.reset_index(drop = True)
261
262     y=exp_dataset
263
264     test_ratio=11/day_n
265
266     lag = 0
267
268     test_day = int(day_n*test_ratio)
269     train_day = day_n-test_day
270     train_ref_dataset=ref_dataset[:train_day]
271     train_exp_dataset=exp_dataset[:train_day]
272
273
274     if exp_name == ref_name:
275         maxcorr,lag=country.cross_corr(train_ref_dataset,train_exp_dataset,60)
276         print("Best Lag =",lag)
277         print("Cross correlation =",maxcorr)
278     else:
279         maxcorr,_=pearsonr(train_ref_dataset,train_exp_dataset)
280
281     print("Correlation =",maxcorr)
282
283
284     train_ref_dataset=utils.shift_fill0(train_ref_dataset,lag)
285
286
287     X=utils.s2v(train_ref_dataset)
288     y=utils.s2v(train_exp_dataset)
289     Xtest=utils.s2v(ref_dataset[train_day:])
290     ytest=utils.s2v(exp_dataset[train_day:])
291
292     poly_par =1
293     if test_par !=None:
294         poly_par = int(test_par)
295     model = linear_model.LeastSquaresPoly(p=poly_par)
296     model.fit(X,y)
297     titlename = exp_name+"_"+exp_ct_name+"_vs_"+ref_name+"_"+ref_ct_name+"_lag"+str(lag)
298     + "_p"+str(poly_par)
299     filename = "LRplot_" + titlename + ".pdf"
300     utils.test_and_plot(model,X,y,Xtest,ytest,title=titlename,
301                        filename=filename)
302
303     if question == "corr":
304
305         dataset = read_dataset("phase1_training_data.csv")
306         dataset = utils.get_all(dataset)
307         n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
308
309         day_n = len(dataset[dataset[ct_key]=='CA']["cases"])
310
311         exp_name = "cases_14_100k"
312         exp_ct_name = 'CA'
313
314         n_neigh = 100
315         cutoff = 0.5
316         n_neigh, n_neigh_indices, n_neigh_distances, n_neigh_lags, n_neigh_dict,
317         n_neigh_dict_lags = country.select_country(
318             dataset,exp_name,exp_ct_name,n_neigh,method="cross",cutoff=cutoff)
319         print("N_neighbor = %d with cutoff = %.3f"%(n_neigh,cutoff))
320         print(n_neigh_dict)
321         print(n_neigh_dict_lags)

```

```

321     plt.plot(n_neigh_distances)
322     plt.xlabel("Neighbor_Country")
323     plt.ylabel("Correlation")
324     plotname="Ncorr_plot_n"+str(n_neigh)+"_"+exp_ct_name+".pdf"
325     fname = os.path.join(".", "figs", plotname)
326     plt.savefig(fname)
327     plt.clf()
328     plt.close('all')
329
330
331 #-----AMLR: Main prediciton model based on AutoReg and normal Multi-variable Linear
    Regressions-----
332
333 if question == "AMLR":
334     warnings.filterwarnings("ignore")
335     test_par = io_args.test_par
336     test_day = io_args.test_day
337     test_num = io_args.test_num
338
339
340     dataset = read_dataset("phase1_training_data.csv")
341     dataset = utils.get_all(dataset)
342     n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
343
344     dataset_CA=dataset[dataset[ct_key]=='CA']
345     cases_CA = dataset_CA["cases"]
346     cases14_CA =dataset_CA["cases_14_100k"]
347     day_n = len(cases_CA)
348
349     exp_fea_name = "deaths_100k"
350     ref_fea_name = "deaths_100k"
351     #exp_dataset = np.zeros((n_ct,day_n))
352     exp_ct_name ='CA'
353     #ref_ct_name ='UK'
354
355     #Find nearest neighbors of CA,return indices, get a combined all neighbor country
    dataset
356     n_neigh = 100
357     cutoff = 0.9
358     if test_num != None:
359         test_num = int(test_num)
360         n_neigh = test_num
361     if test_par != None:
362         test_par = float(test_par)
363         cutoff = test_par
364     n_neigh, n_neigh_indices, n_neigh_distances,n_neigh_lags, n_neigh_dict,
    n_neigh_dict_lags = country.select_country(
365         dataset,exp_fea_name,exp_ct_name,n_neigh=n_neigh,
366         method="cross",cutoff=cutoff,R=30,ref_fea_name=ref_fea_name)
367     print("y is '%s' from country '%s'"%(exp_fea_name,exp_ct_name))
368     print("N_neighbor = %d with cutoff = %.3f"%(n_neigh,cutoff))
369     #print(n_neigh_dict)
370
371     #
    -----
372
373     # Combine the lags list and modify each country, return a dataset of reference
374     # the reference is used to be a X input matrix
375     # all subset of dataset are shifted based on the best lag according to the cross
    correlation
376     # we are fitting all the correlated countries with the same time-shift
377     # so that the total linear correlation coefficient is significantly improved
378     # we just considered the case that all countries don't have the same date of
    outbreak of pandemic
379     # this is an important factor to fit our data

```



```

379     #
-----
380     ref_dataset=country.crosscombine_ct(dataset,ref_fea_name,n_neigh_indices,
n_neigh_lags)
381     exp_dataset= dataset[dataset[ct_key]==exp_ct_name][exp_fea_name]
382     exp_dataset=exp_dataset.reset_index(drop = True)
383
384     test_ratio=11/day_n
385     if test_day != None:
386         test_day = float(test_day)
387         test_ratio = test_day
388
389     test_day = int(day_n*test_ratio)
390     train_day = day_n-test_day
391
392     X=ref_dataset[:train_day]
393     y=utils.s2v(exp_dataset[:train_day])
394
395     Xtest=ref_dataset[train_day:]
396     ytest=utils.s2v(exp_dataset[train_day:])
397
398     self_AR_X=exp_dataset[:train_day]
399     self_AR_y=exp_dataset[train_day:]
400
401     #k is the lag chosen as parameter in AutoReg model
402     #optimal lags is 2 as simplist optimal lag
403     #optimal lags is 12 as second best lag
404     k=2
405     #Implement auto regression on the self feature of the selected country
406     self_auto_model = AutoReg1(lags=k)
407     self_auto_model.fit(self_AR_X)
408     self_train_predictions = self_auto_model.predict(start=k,end=train_day-1)
409     self_train_rmse = sqrt(mean_squared_error(self_AR_X[k:], self_train_predictions))
410     self_test_predictions = self_auto_model.predict(start=train_day, end=train_day+
test_day-1)
411     self_test_rmse = sqrt(mean_squared_error(self_AR_y, self_test_predictions))
412     #
-----
413     #Implement the parallel Linear Regression on the neighbors of selected country:
414     #
415     #   Feature(selected country,past)= LR (Feature(other countries,past) )
416     #
417     #So we can train a model to fit the past feature of selected country
418     #Later, we will use this model to parallely predict the Future Feature of the spec
country:
419     #
420     #   Feature(selected country,future)= a*LR (Feature(other countries,future) ) + b*AR
(Feature(spec country,past)) + c
421     #
422     #Note that we have already shifted the training set so that all outbreaks happend at
the same time
423     #Based on this we are able to say, the correlation decides whether we can predict
the spec country
424     #
-----
425     parallel_model = linear_model.LeastSquaresPoly(p=1)
426     parallel_model.fit(X,y)
427     parallel_predictions=parallel_model.predict(X)
428     train_parallel_rmse = sqrt(mean_squared_error(parallel_predictions,y))
429     print("MultiLinear train Process rmse is %.3f"%train_parallel_rmse)
430

```

```

431     #Implement auto_regression for all neighbors for feature "cases_100k" except
selected country
432     test_rmse_ct = np.zeros(n_neigh)
433     train_rmse_ct = np.zeros(n_neigh)
434     test_predictions = np.zeros((n_neigh, test_day))
435     for nct in range(n_neigh):
436         dataset_ct = dataset[dataset[ct_key]==ct_inv_mapper[n_neigh_indices[nct]]]
437         X = dataset_ct[ref_fea_name]
438         X = X.reset_index(drop = True)
439         train, test = X[:len(X)-test_day], X[(len(X)-test_day):]
440
441         auto_model=AutoReg1(lags=k)
442         auto_model.fit(train)
443         train_predictions = auto_model.predict(start=k,end=len(train)-1)
444         train_rmse_ct[nct] = sqrt(mean_squared_error(train[k:], train_predictions))
445         test_predictions[nct] = auto_model.predict(start=len(train), end=len(train)+len(
test)-1)
446         test_rmse_ct[nct] = sqrt(mean_squared_error(test, test_predictions[nct]))
447
448     #Compute all errors happend in the AutoReg process
449     #Add up and get the mean value of rmse of each phase of the regression process of
each country:
450     test_auto_rmse = np.mean(test_rmse_ct)
451     train_auto_rmse = np.mean(train_rmse_ct)
452
453     #This is the matrix of the auto predicted value of all other countries
454     #This can be considered as an X of a LR model, then we can predict the future
feature of spec country
455     auto_predictions = test_predictions.T
456
457     #Prediction phase and calculate the final result of the selected feature of the spec
country
458     MLR_predictions = parallel_model.predict(auto_predictions)
459     test_exp_dataset = exp_dataset[train_day:]
460     final_rmse = sqrt(mean_squared_error(test_exp_dataset, MLR_predictions))
461     baseline = sqrt(mean_squared_error(test_exp_dataset, np.zeros(test_day)))
462
463 #-----Output Phase
-----
464     print("AutoReg train Process mean_rmse is %.3f"%train_auto_rmse)
465     print("AutoReg test Process(validation process) mean_rmse is %.3f"%test_auto_rmse)
466     print("Final test rmse is %.3f"%final_rmse)
467     print("Final test percentage error is %.3f%%"%(final_rmse/baseline*100))
468     #Predict the final deaths feature value based on the population information of the
spec country
469     #Theres almost no errors added here regarding the defination of deaths_100k which is
sctrictly proportional to deaths
470
471 #-----deaths feature
-----
472     if exp_fea_name == ref_fea_name and exp_fea_name == "deaths_100k":
473         deaths_test_dataset=dataset[dataset[ct_key]==exp_ct_name]["deaths"][train_day:]
474         num_people_100k = utils.get_num_people_100k(dataset, exp_ct_name)
475
476         MLR_deaths_predictions = num_people_100k * MLR_predictions
477
478         self_deaths_predictions = num_people_100k * self_test_predictions
479
480         MLR_deaths_rmse = sqrt(mean_squared_error(MLR_deaths_predictions.T[0],
deaths_test_dataset))
481
482         print("Without Self_AR, MultiLR deaths test rmse is %.3f"%MLR_deaths_rmse)
483

```

```

484
485         # predict the deaths by combining two models together : stacked accroding to the
         correlation coefficients
486
487         AMLR_deaths_predictions = utils.estimator(MLR_deaths_predictions.T[0],
         self_deaths_predictions,n_neigh_distances)
488
489         AMLR_deaths_rmse = sqrt(mean_squared_error(AMLR_deaths_predictions,
         deaths_test_dataset))
490
491         print("With Self_AR, Final AutoMultiLR deaths test rmse is %.3f"%
         AMLR_deaths_rmse)
492
493         utils.predictions_plot(AMLR_deaths_predictions,deaths_test_dataset,
         exp_ct_name,exp_fea_name,"test",n_neigh=n_neigh
494                                )
495
496 #-----AMLR model training and prediction-----Completed
         -----
497
498
499
500 if question == "find_n": #I found that N_neighbors = 7 is best
501     warnings.filterwarnings("ignore")
502     test_par = io_args.test_par
503     test_day = io_args.test_day
504     test_num = io_args.test_num
505
506
507     dataset = read_dataset("phase1_training_data.csv")
508     n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
509
510     dataset_CA=dataset[dataset[ct_key]=='CA']
511     cases_CA = dataset_CA["cases"]
512     cases14_CA =dataset_CA["cases_14_100k"]
513     day_n = len(cases_CA)
514
515     exp_fea_name = "cases_100k"
516     ref_fea_name = "cases_100k"
517     #exp_dataset = np.zeros((n_ct,day_n))
518     exp_ct_name = 'CA'
519     #ref_ct_name = 'UK'
520
521     #Find nearest neighbors of CA,return indices, get a combined all neighbor country
     dataset
522     # n_neigh = 100
523     #N_Neighbor Loop to find N optimal
524
525     n_neigh_max = 60
526     n_neigh_min = 1
527     cutoff = 0.9
528     if test_num != None:
529         test_num = int(test_num)
530         n_neigh_max=test_num
531     if test_par != None:
532         test_par = float(test_par)
533         cutoff = test_par
534
535     test_ratio=11/day_n
536     if test_day != None:
537         test_day = float(test_day)
538         test_ratio = test_day
539
540     test_day = int(day_n*test_ratio)
541     train_day = day_n-test_day
542     neigh_final_rmse=np.zeros(n_neigh_max+1)

```

```

543     train_parallel_rmse = np.zeros(n_neigh_max+1)
544     report_rmse = np.zeros(n_neigh_max+1)
545     for n in range(n_neigh_min, n_neigh_max+1):
546         n_neigh = n
547
548         n_neigh, n_neigh_indices, n_neigh_distances, n_neigh_lags, n_neigh_dict,
n_neigh_dict_lags = country.select_country(
549             dataset, exp_fea_name, exp_ct_name, n_neigh=n,
550             method="cross", cutoff=cutoff, R=30, ref_fea_name=ref_fea_name)
551
552         #print(n_neigh_dict)
553
554         #ref_dataset=country.combine_ct(dataset, ref_fea_name, n_neigh_indices)
555         ref_dataset=country.crosscombine_ct(dataset, ref_fea_name, n_neigh_indices,
n_neigh_lags)
556         exp_dataset= dataset[dataset[ct_key]==exp_ct_name][exp_fea_name]
557
558         X=ref_dataset[:train_day]
559         y=utils.s2v(exp_dataset[:train_day])
560         Xtest=ref_dataset[train_day:]
561         ytest=utils.s2v(exp_dataset[train_day:])
562
563         # print("y is '%s' from country '%s'"%(exp_fea_name, exp_ct_name))
564         # print("N_neighbor = %d with cutoff = %.3f"%(n_neigh, cutoff))
565         parallel_model = linear_model.LeastSquaresPoly(p=1)
566         parallel_model.fit(X, y)
567         parallel_predictions=parallel_model.predict(X)
568         train_parallel_rmse[n] = sqrt(mean_squared_error(parallel_predictions, y))
569         print("MultiLinear train Process rmse is %.3f"%train_parallel_rmse[n])
570         #optimal lags is 2 as simplist optimal lag
571         #optimal lags is 12 as second best lag
572         k=2
573         #Implement auto_regression for all neighbors for feature "cases_100k"
574         test_rmse_ct = np.zeros(n_neigh)
575         train_rmse_ct = np.zeros(n_neigh)
576         test_predictions = np.zeros((n_neigh, test_day))
577         for nct in range(n_neigh):
578             dataset_ct = dataset[dataset[ct_key]==ct_inv_mapper[n_neigh_indices[nct]]]
579             X = dataset_ct[ref_fea_name]
580             X = X.reset_index(drop = True)
581             train, test = X[:((len(X)-test_day))], X[(len(X)-test_day):]
582
583             auto_model=AutoReg1(lags=k)
584             auto_model.fit(train)
585             train_predictions = auto_model.predict(start=k, end=len(train)-1)
586             train_rmse_ct[nct] = sqrt(mean_squared_error(train[k:], train_predictions))
587             test_predictions[nct] = auto_model.predict(start=len(train), end=len(train)+
len(test)-1)
588             test_rmse_ct[nct] = sqrt(mean_squared_error(test, test_predictions[nct]))
589
590             test_auto_rmse = np.mean(test_rmse_ct)
591             train_auto_rmse = np.mean(train_rmse_ct)
592             auto_predictions = test_predictions.T
593
594             final_predictions = parallel_model.predict(auto_predictions)
595             test_exp_dataset = exp_dataset[train_day:]
596             final_rmse = sqrt(mean_squared_error(test_exp_dataset, final_predictions))
597             baseline = sqrt(mean_squared_error(test_exp_dataset, np.zeros(test_day)))
598
599
600             # print("AutoReg train Process mean_rmse is %.3f"%train_auto_rmse)
601             # print("AutoReg test Process(validation process) mean_rmse is %.3f"%
test_auto_rmse)
602             print("N_neighbor= %d, Final test rmse is %.3f"%(n_neigh, final_rmse))
603             # print("Final test percentage error is %.3f%"%(final_rmse/baseline*100))

```

```

604         neigh_final_rmse[n]=final_rmse
605         # edit to report training phase error or test phase error
606         #report_rmse[n]=train_parallel_rmse[n]
607         report_rmse[n]=neigh_final_rmse[n]
608     plt.plot(np.arange(1,n_neigh_max+1),report_rmse[1:n_neigh_max+1],marker="*")
609     #plt.plot(np.arange(1,n_neigh_max+1),neigh_final_rmse[1:n_neigh_max+1],marker="*")
610     #plt.plot(n_neigh_distances)
611     plt.xlabel("N_neighbors")
612     plt.ylabel("Final rmse")
613
614     plotname="Find_N_plot_d"+str(test_day)+"_"+exp_ct_name+".pdf"
615     fname = os.path.join(".", "figs", plotname)
616     plt.savefig(fname)
617     plt.clf()
618     plt.close('all')
619
620
621 if question == "AMLR_future":
622     warnings.filterwarnings("ignore")
623     test_par = io_args.test_par
624     test_day = io_args.test_day
625     test_num = io_args.test_num
626
627
628     dataset = read_dataset("phase1_training_data.csv")
629     dataset = utils.get_all(dataset)
630     n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
631
632     dataset_CA=dataset[dataset[ct_key]=='CA']
633     cases_CA = dataset_CA["cases"]
634     cases14_CA =dataset_CA["cases_14_100k"]
635     day_n = len(cases_CA)
636
637     exp_fea_name = "deaths_100k"
638     ref_fea_name = "deaths_100k"
639
640     exp_ct_name ='CA'
641     #ref_ct_name ='UK'
642
643     #Find nearest neighbors of CA,return indices, get a combined all neighbor country
dataset
644     n_neigh = 100
645     cutoff = 0.9
646     if test_num != None:
647         test_num = int(test_num)
648         n_neigh = test_num
649     if test_par != None:
650         test_par = float(test_par)
651         cutoff = test_par
652     if test_day != None:
653         test_day = int(test_day)
654
655     train_day = day_n
656
657     n_neigh, n_neigh_indices, n_neigh_distances,n_neigh_lags, n_neigh_dict,
n_neigh_dict_lags = country.select_country(
658         dataset,exp_fea_name,exp_ct_name,n_neigh=n_neigh,
659         method="cross",cutoff=cutoff,R=30,ref_fea_name=ref_fea_name)
660     print("y is '%s' from country '%s'"%(exp_fea_name,exp_ct_name))
661     print("N_neighbor = %d with cutoff = %.3f"%(n_neigh,cutoff))
662     #print(n_neigh_dict)
663
664     #

```

```

665     # Combine the lags list and modify each country, return a dataset of reference
666     # the reference is used to be a X input matrix
667     # all subset of dataset are shifted based on the best lag according to the cross
correlation
668     # we are fitting all the correlated countries with the same time-shift
669     # so that the total linear correlation coefficient is significantly improved
670     # we just considered the case that all countries don't have the same date of
outbreak of pandemic
671     # this is an important factor to fit our data
672     #
-----

673     ref_dataset=country.crosscombine_ct(dataset,ref_fea_name,n_neigh_indices,
n_neigh_lags)
674     exp_dataset= dataset[dataset[ct_key]==exp_ct_name][exp_fea_name]
675     exp_dataset=exp_dataset.reset_index(drop = True)
676
677
678
679
680     X=ref_dataset[:train_day]
681     y=utils.s2v(exp_dataset[:train_day])
682
683     AR_X=exp_dataset[:train_day]
684     AR_y=exp_dataset[train_day:]
685
686     #k is the lag chosen as parameter in AutoReg model
687     #optimal lags is 2 as simplist optimal lag
688     k=2
689     #Implement auto regression on the self feature of the selected country
690
691     AR_model = AutoReg1(lags=k)
692     AR_model.fit(AR_X)
693     AR_train_predictions = AR_model.predict(start=k,end=train_day-1)
694     AR_test_predictions = AR_model.predict(start=train_day, end=train_day+test_day-1)
695     AR_train_rmse = sqrt(mean_squared_error(AR_X[k:], AR_train_predictions))
696
697     #
-----

698     #Implement the parallel Linear Regression on the neighbors of selected country:
699     #
700     # Feature(selected country,past)= LR (Feature(other countries,past) )
701     #
702     #So we can train a model to fit the past feature of selected country
703     #Later, we will use this model to parallely predict the Future Feature of the spec
country:
704     #
705     # Feature(selected country,future)= a*LR (Feature(other countries,future) ) + b*AR
(Feature(spec country,past)) + c
706     #
707     #Note that we have already shifted the training set so that all outbreaks happend at
the same time
708     #Based on this we are able to say, the correlation decides whether we can predict
the spec country
709     #
-----

710     parallel_model = linear_model.LeastSquaresPoly(p=1)
711     parallel_model.fit(X,y)
712     parallel_predictions=parallel_model.predict(X)
713     train_parallel_rmse = sqrt(mean_squared_error(parallel_predictions,y))
714     print("MultiLinear train Process rmse is %.3f"%train_parallel_rmse)
715

```

```

716     #Implement auto_regression for all neighbors for feature "cases_100k" except
selected country
717
718     train_rmse_ct = np.zeros(n_neigh)
719     test_predictions = np.zeros((n_neigh, test_day))
720     for nct in range(n_neigh):
721         dataset_ct = dataset[dataset[ct_key]==ct_inv_mapper[n_neigh_indices[nct]]]
722         X = dataset_ct[ref_fea_name]
723         X = X.reset_index(drop = True)
724
725         auto_model=AutoReg1(lags=k)
726         auto_model.fit(X)
727         train_predictions = auto_model.predict(start=k, end=train_day-1)
728
729         train_rmse_ct[nct] = sqrt(mean_squared_error(X[k:], train_predictions))
730         test_predictions[nct] = auto_model.predict(start=train_day, end=train_day+
test_day-1)
731
732
733     #Compute all errors happend in the AutoReg process
734     #Add up and get the mean value of rmse of each phase of the regression process of
each country:
735
736     train_auto_rmse = np.mean(train_rmse_ct)
737
738     #This is the matrix of the auto predicted value of all other countries
739     #This can be considered as an X of a LR model, then we can predict the future
feature of spec country
740     auto_predictions = test_predictions.T
741
742     #Prediction phase and calculate the final result of the selected feature of the spec
country
743     MLR_predictions = parallel_model.predict(auto_predictions)
744
745
746 #-----Output Phase
-----
747
748     print("AutoReg train Process mean_rmse is %.3f"%train_auto_rmse)
749 #-----deaths feature
-----
750
751     if exp_fea_name == ref_fea_name and exp_fea_name == "deaths_100k":
752         num_people_100k = utils.get_num_people_100k(dataset, exp_ct_name)
753
754         MLR_deaths_predictions = num_people_100k * MLR_predictions
755
756         AR_deaths_predictions = num_people_100k * AR_test_predictions
757
758         # predict the deaths by combining two models together : stacked accroding to the
correlation coefficients
759
760         AMLR_deaths_predictions = utils.estimator(MLR_deaths_predictions.T[0],
AR_deaths_predictions, n_neigh_distances)
761         with open(os.path.join("..", "data", "sample_phase1_submission_origin.csv"), "rb
") as f:
762             sample_submission_dataset=pd.read_csv(f)
763             sample_submission_dataset["deaths"]=AMLR_deaths_predictions
764             sample_submission_dataset.to_csv(r'..\data\sample_phase1_submission.csv', index=
False, header=True)
765             #TrainingSet.to_csv(r'..\data\Modified_data.csv', index=False, header=True)
766
767

```

```

768         # y_test = np.asarray(y_test)
769         # y_pred = AMLR_deaths_predictions
770         # utils.predictions_plot(y_pred,y_test,"CA","deaths","Test")
771
772
773
774 #-----AMLR model training and prediction-----Completed
       -----

```

```

1 import numpy as np
2 from linear_model import LeastSquares
3
4
5 class My_AutoReg:
6     def __init__(self,lag):
7         self.lag = lag
8
9     def fit(self,X):
10         self.X = self.BuildX(X,self.lag)
11         self.y = self.Buildy(X,self.lag)
12
13         model = LeastSquares()
14         #print(self.X.shape)
15         #print(self.y.shape)
16         model.fit(self.X,self.y)
17         self.my_model = model
18
19
20
21     def BuildX(self,X,lag):
22         if type(self.lag) == type(1):
23             lag = np.arange(1,self.lag+1)
24
25         length = len(X)
26         A = np.empty((0,len(lag)+1),dtype=np.int)
27         for i in range(length-lag[-1]):
28             xi = np.ones(len(lag)+1,dtype=np.int)
29
30             for j,term in enumerate(lag):
31                 xi[j+1] = X[i+term-1]
32             A = np.vstack([A, xi])
33         #print(A)
34
35         return A
36
37     def Buildy(self,X,lag):
38         if type(self.lag) == type(1):
39             lag = np.arange(1,self.lag+1)
40
41         maxlag = lag[-1]
42         y=np.empty(int(0),dtype=np.int)
43         length = len(X)
44         for i in range(length-lag[-1]):
45             y = np.append(y, X[i+lag[-1]])
46         #print(y)
47         return y
48
49     def predict(self, Z, start = None, end = None):
50         n_time_topre = end - start + 1
51         start = start - self.lag
52         model = self.my_model
53         w = model.w
54         y_pred = [0]*n_time_topre
55         latest_X = self.X[start-1]
56         y_pred[0] = np.dot(latest_X,w)

```



```

57         i=1
58         while(i<n_time_topre):
59             if(start + i >= len(Z)):
60                 latest_X = np.roll(latest_X,-1)
61                 latest_X[0] = 1
62                 latest_X[-1] = y_pred[i-1]
63                 y_pred[i] = model.predict(latest_X)
64                 i += 1
65             else:
66                 latest_X = self.X[start+i-1]
67                 y_pred[i] = model.predict(latest_X)
68                 i += 1
69         return y_pred
70
71 class AutoReg1():
72     def __init__(self,lags):
73         self.lags = lags
74
75     def fit(self,y):
76         self.y = np.zeros(len(y)) #train set
77         self.y = y
78         self.X = self.BuildX() #matrix constructed from train set by shifting
79         model = LeastSquares()
80         #print(self.X.shape)
81         #print(self.y.shape)
82         model.fit(self.X,self.y[self.lags:].T)
83         self.w = model.w
84         #print(self.w)
85         # self.my_model = model
86
87     def BuildX(self):
88         k = self.lags
89         y = self.y
90         T = len(y)
91         X = np.ones((T-k,k+1))
92         for t in range(k,T):
93             X[t-k][1:] = y[t-k:t]
94         #print(X)
95         #print(y)
96         return X
97
98     def predict(self,start,end):
99         y=self.y
100         k=self.lags
101         T= len(y)
102         y_pred = np.zeros(end-start+1)
103
104         if start <= T:
105             ylag = y[start-k:start]
106         else:
107             ylag = y[T-k:]
108             #predict from y[T] for each y[t] up to y[start-1]
109             for t in range(T,start):
110                 y_pred1 = ylag@self.w[1:]+self.w[0]
111                 ylag = np.roll(ylag,-1)
112                 ylag[-1] =y_pred1
113             #scroll data into y_pred
114             for t in range(start,end+1):
115                 y_pred[t-start] = ylag@self.w[1:]+self.w[0]
116                 ylag = np.roll(ylag,-1)
117                 ylag[-1] =y_pred[t-start]
118         return y_pred

```

```

1 import numpy as np
2 import utils

```

```

3 from scipy.stats import pearsonr
4 import pandas as pd
5
6 def select_country(dataset, exp_fea_name, exp_ct_name, ref_fea_name=None, n_neigh=10, method="
   corr", cutoff=0.5, itself=False, R=30):
7     n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind = utils.make_mapper(dataset)
8     nan_num = 0
9     if method == "corr" or "cross":
10         day_n = len(dataset[dataset[ct_key]=='CA']["cases"])
11         if ref_fea_name == None:
12             ref_fea_name = exp_fea_name
13
14         exp_dataset = np.zeros((n_ct, day_n))
15
16         #Choose the dataset regarding the name interested
17         target= np.zeros((1, day_n))
18         target[0] = dataset[dataset[ct_key]==exp_ct_name][exp_fea_name]
19         corr=np.zeros(n_ct)
20         lags=np.zeros(n_ct)
21         indices=np.zeros(n_ct)
22         if method == "corr":
23             corr_pair = np.zeros((2, n_ct))
24         elif method == "cross":
25             corr_pair = np.zeros((3, n_ct))
26
27
28
29         #Construct Experiment Dataset (Matrix) =feature(country, day)
30         for nct in range(nan_num, n_ct): #fillter out nan set
31             exp_dataset[nct] = dataset[dataset[ct_key]==ct_inv_mapper[nct]][ref_fea_name]
32
33         #Compute Corr between N_CT countries
34         if method == "corr":
35             for nct in range(nan_num, n_ct): #fillter out nan set
36                 corr[nct]=pearsonr(target[0], exp_dataset[nct])[0]
37                 indices[nct]=nct
38         elif method == "cross":
39             for nct in range(nan_num, n_ct): #fillter out nan set
40                 series1=pd.Series(exp_dataset[nct])
41                 series2=pd.Series(target[0])
42                 corr[nct], lags[nct]=cross_corr(series1, series2, R)
43                 indices[nct]=nct
44
45         corr_pair[0] = indices
46         corr_pair[1] = corr
47         if method == "cross":
48             corr_pair[2] = lags
49
50         #remove the nan set
51         #sort the corr_pair
52         corr_pair1 = corr_pair.T[1:]
53         corr_pair= corr_pair1[np.argsort(corr_pair1[:,1], axis =0)]
54         corr_pair= corr_pair[:, :-1]
55
56         #remove itself from the correlation list
57         if itself == False:
58             corr_pair = corr_pair[1:]
59         #choose the neigh distance vector
60         corr_n_neigh =corr_pair[:,1][:n_neigh]
61         n_neigh_distances = corr_n_neigh
62
63         if cutoff != None:
64             n_neigh_distances =n_neigh_distances[n_neigh_distances >= cutoff]
65
66         #update n_neigh after cutoff

```

```

67     n_neigh = len(n_neigh_distances)
68     n_neigh_indices = corr_pair[:,0][:n_neigh]
69     if method == "cross":
70         n_neigh_lags = corr_pair[:,2][:n_neigh]
71
72     #create dict for n neigh countries with descent order of distances
73     ct_neigh = [ct_inv_mapper[n_neigh_indices[i]] for i in range(n_neigh)]
74     n_neigh_dict=dict(zip(ct_neigh,list(corr_n_neigh)))
75     n_neigh_dict_lags=dict(zip(ct_neigh,list(n_neigh_lags)))
76     if method == "corr":
77         return n_neigh, n_neigh_indices, n_neigh_distances, n_neigh_dict
78     elif method == "cross":
79         return n_neigh, n_neigh_indices, n_neigh_distances, n_neigh_lags, n_neigh_dict,
80         n_neigh_dict_lags
81
82 def combine_ct(dataset,fea_name,ind_ct):
83     n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
84     day_n = len(dataset[dataset[ct_key]=='CA']["cases"])
85     nmax = len(ind_ct)
86     exp_dataset = np.zeros((nmax,day_n))
87     exp_name = fea_name
88     for n in range(nmax):
89         exp_dataset[n] = dataset[dataset[ct_key]==ct_inv_mapper[ind_ct[n]]][exp_name]
90     return exp_dataset.T #dim=(cases14,country) fea=country
91
92 def cross_corr(ref_s,exp_s,R):
93
94     ref_s1 = ref_s.reset_index(drop = True)
95     exp_s1 = exp_s.reset_index(drop = True)
96     maxcorr =0.0
97     lag_best = 0
98     for lag in range(R):
99         ref_s1 = ref_s1.shift(lag)
100         ref_s1[:lag]=0.0
101         corr= pearsonr(ref_s1, exp_s1)[0]
102         if corr>maxcorr:
103             maxcorr = corr
104             lag_best = lag
105     return maxcorr,lag_best
106
107
108 def crosscombine_ct(dataset,fea_name,ind_ct,lags):
109     n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
110     day_n = len(dataset[dataset[ct_key]=='CA']["cases"])
111     nmax = len(ind_ct)
112     exp_dataset = np.zeros((nmax,day_n))
113     exp_name = fea_name
114     for n in range(nmax):
115         exp_dataset[n] = dataset[dataset[ct_key]==ct_inv_mapper[ind_ct[n]]][exp_name]
116         exp_dataset[n] = utils.shift_fill0_array(exp_dataset[n],lags[n])
117     return exp_dataset.T #dim=(cases14,country) fea=country

```



```

1 import pickle
2 import os
3 import sys
4 import numpy as np
5 from scipy.optimize import approx_fprime
6 import matplotlib.pyplot as plt
7 import pandas as pd
8 from scipy.sparse import csr_matrix as sparse_matrix
9 from sklearn.metrics import mean_squared_error
10 import datetime
11 import math
12

```

```

13 def create_user_item_matrix(ratings,user_key="user",item_key="item"):
14
15     n = len(set(ratings[user_key]))
16     d = len(set(ratings[item_key]))
17
18     user_mapper = dict(zip(np.unique(ratings[user_key]), list(range(n))))
19     item_mapper = dict(zip(np.unique(ratings[item_key]), list(range(d))))
20
21     user_inverse_mapper = dict(zip(list(range(n)), np.unique(ratings[user_key])))
22     item_inverse_mapper = dict(zip(list(range(d)), np.unique(ratings[item_key])))
23
24     user_ind = [user_mapper[i] for i in ratings[user_key]]
25     item_ind = [item_mapper[i] for i in ratings[item_key]]
26
27     X = sparse_matrix((ratings["rating"], (user_ind, item_ind)), shape=(n,d))
28
29     return X, user_mapper, item_mapper, user_inverse_mapper, item_inverse_mapper, user_ind,
        item_ind
30
31 def standardize_cols(X, mu=None, sigma=None):
32     # Standardize each column with mean 0 and variance 1
33     n_rows, n_cols = X.shape
34
35     if mu is None:
36         mu = np.mean(X, axis=0)
37
38     if sigma is None:
39         sigma = np.std(X, axis=0)
40         sigma[sigma < 1e-8] = 1.
41
42     return (X - mu) / sigma, mu, sigma
43
44
45 def check_gradient(model, X, y):
46     # This checks that the gradient implementation is correct
47     w = np.random.rand(model.w.size)
48     f, g = model.funObj(w, X, y)
49
50     # Check the gradient
51     estimated_gradient = approx_fprime(w,
52                                         lambda w: model.funObj(w,X,y)[0],
53                                         epsilon=1e-6)
54
55     implemented_gradient = model.funObj(w, X, y)[1]
56
57     if np.max(np.abs(estimated_gradient - implemented_gradient)) > 1e-4):
58         raise Exception('User and numerical derivatives differ:\n%s\n%s' %
59                         (estimated_gradient[:5], implemented_gradient[:5]))
60     else:
61         print('User and numerical derivatives agree.')
62
63 def classification_error(y, yhat):
64     return np.mean(y!=yhat)
65
66
67 def test_and_plot(model,X,y,Xtest=None,ytest=None,title=None,filename=None):
68
69     # Compute training error
70     yhat = model.predict(X)
71     #trainError = np.mean((yhat - y)**2)
72     trainError = mean_squared_error(y,yhat)
73     print("Training mse = %.10f" % trainError)
74
75     # Compute test error
76     if Xtest is not None and ytest is not None:

```

```

77     yhat = model.predict(Xtest)
78     # testError = np.mean((yhat - ytest)**2)
79     testError = mean_squared_error(ytest, yhat)
80     print("Test mse      = %.10f" % testError)
81
82     # Plot model
83     plt.figure()
84     plt.plot(X, y, 'b.')
85
86     # Choose points to evaluate the function
87     if (len(X[0]) == 1):
88         Xgrid = np.linspace(np.min(X), np.max(X), 1000)[: , None]
89         ygrid = model.predict(Xgrid)
90         plt.plot(Xgrid, ygrid, 'g')
91
92         if title is not None:
93             plt.title(title)
94
95         if filename is not None:
96             filename = os.path.join("..", "figs", filename)
97             print("Saving", filename)
98             plt.savefig(filename)
99
100 def linear_model_test(model, X, y, Xtest=None, ytest=None):
101
102     # Compute training error
103     yhat = model.predict(X)
104     # trainError = np.mean((yhat - y)**2)
105     trainError = mean_squared_error(y, yhat)
106     print("Training mse = %.10f" % trainError)
107
108     # Compute test error
109     if Xtest is not None and ytest is not None:
110         yhat = model.predict(Xtest)
111         # testError = np.mean((yhat - ytest)**2)
112         testError = mean_squared_error(ytest, yhat)
113         print("Test mse      = %.10f" % testError)
114
115 def linear_model_test_with_plot(model, X, y, Xtest=None, ytest=None):
116
117     # Compute training error
118     yhat = model.predict(X)
119     # trainError = np.mean((yhat - y)**2)
120     trainError = mean_squared_error(y, yhat)
121     print("Training mse = %.10f" % trainError)
122
123     # Compute test error
124     if Xtest is not None and ytest is not None:
125         yhat = model.predict(Xtest)
126         # testError = np.mean((yhat - ytest)**2)
127         testError = mean_squared_error(ytest, yhat)
128         print("Test mse      = %.10f" % testError)
129
130 def predictions_plot(y_pred, y_real, exp_ct_name, exp_fea_name, phase, n_neigh=None):
131     rmse = np.sqrt(mean_squared_error(y_pred, y_real))
132     l = len(y_pred)
133     plt.plot(np.arange(1), y_pred, marker="*", label="pred")
134     plt.plot(np.arange(1), y_real, marker=".", label="real")
135     plt.xlabel("Date")
136     plt.ylabel("Deaths")
137     title = "Pred_" + exp_fea_name + "_rmse is " + str(rmse)
138     plt.title(title)
139     plt.legend()
140     plotname = "Pred_" + phase + "_plot_" + exp_fea_name + "_d" + str(l) + "_" + exp_ct_name + ".pdf"
141     if n_neigh != None:

```

```

142     plotname="Pred_"+phase+"_plot_"+exp_fea_name+"_d"+str(l)+"_n"+str(n_neigh)+"_"+
exp_ct_name+".pdf"
143     fname = os.path.join("../", "figs", plotname)
144     if fname is not None:
145         fname = os.path.join("../", "figs", fname)
146         print("Saving", fname)
147     plt.savefig(fname)
148     plt.clf()
149     plt.close('all')
150
151
152
153 def dif_series(x,day,refine=None):
154     y=x-x.shift(day)
155     if refine == 1:
156         y=y[day:]
157     return y
158
159 def s2m(series):
160     X = np.zeros((1,len(series)))
161     X[0] = series
162     return X
163
164 def s2v(series):
165     return s2m(series).T
166
167 def make_mapper(dataset):
168
169     ct_key="country_id"
170     dataset_ct = dataset[ct_key]
171     n_ct=len(set(dataset_ct))
172     ct_mapper = dict(zip(list(set(dataset_ct)),list(range(n_ct))))
173     ct_inv_mapper = dict(zip(list(range(n_ct)),list(set(dataset_ct))))
174     ct_ind = [ct_mapper[i] for i in dataset_ct]
175     return n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind
176
177
178 def shift_fill0(series,lag):
179     series=series.shift(lag)
180     series[:lag]=0.0
181     return series
182
183 def shift_fill0_array(array,lag):
184     int_lag =int(lag)
185     array=np.roll(array,int_lag)
186     array[:int_lag]=0.0
187     return array
188 # from original dataset generate the death_100k column and return the new dataset
189 def get_death_100k(dataset):
190     temp = dataset["cases"]
191     temp[temp == 0] = 1
192     deaths_100k_dataset=dataset["deaths"]*dataset["cases_100k"]/temp
193     #print(deaths_100k_dataset)
194     dataset["deaths_100k"]=pd.Series(deaths_100k_dataset, index=dataset.index)
195     return dataset
196
197
198 def test_death_100k(dataset,n):
199     deaths=dataset["deaths"]
200     cases=dataset["cases"]
201     cases_100k=dataset["cases_100k"]
202     cases[cases==0]=1
203     return deaths[n]*cases_100k[n]/cases[n]
204
205 def get_num_people_100k(dataset,ct_name):

```

```

206 #choose the newest data to compute population
207 #Here's some assumption that population wont change during the whole process
208 n_row=dataset[dataset["country_id"]==ct_name]["deaths"].index[-1]
209 cases_100k = dataset["cases_100k"]
210 cases_100k[cases_100k ==0] =1
211 num_people_100k = dataset["cases"][n_row]/cases_100k[n_row]
212 return num_people_100k
213
214 def get_all(dataset):
215     n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind = make_mapper(dataset)
216     dataset=get_death_100k(dataset)
217     dataset["daily_cases"] = 0.0
218     dataset["daily_deaths"] = 0.0
219     dataset["daily_deaths_100k"] = 0.0
220     for nct in range(1,n_ct):
221         dataset1=dataset[dataset[ct_key]==ct_inv_mapper[nct]]
222         l=len(dataset1)
223         daily_cases=np.zeros(l)
224         daily_deaths=np.zeros(l)
225         daily_deaths_100k=np.zeros(l)
226         for i in range(1,l):
227             daily_deaths[i]= dataset1.iloc[i,3] - dataset1.iloc[i-1,3]
228             daily_cases[i] = dataset1.iloc[i,2] - dataset1.iloc[i-1,2]
229             daily_deaths_100k[i] =dataset1.iloc[i,6] - dataset1.iloc[i-1,6]
230         dataset1["daily_cases"]=daily_cases
231         dataset1["daily_deaths"]=daily_deaths
232         dataset1["daily_deaths_100k"]=daily_deaths_100k
233         dataset[dataset[ct_key]==ct_inv_mapper[nct]] = dataset1
234     return dataset
235 #Estimator to average the value between ensamble predictions from other coutries and auto
    regression
236 def estimator(MLR_pred,self_pred,distances,method="cross"):
237     if type(self_pred) == pd.core.series.Series:
238         self_pred=self_pred.reset_index(drop = True)
239     if type(MLR_pred) == pd.core.series.Series:
240         MLR_pred=MLR_pred.reset_index(drop = True)
241
242     if method == "cross":
243         #The maximum of weight depends on the mean correlation of n_neighbor countreis
244         #The n_neighbors are selected strictly to provide long term information to the
prediction
245         #Especially those countries with earlier outbreaks
246         weight_corr = np.mean(distances)
247         weight_self = 1.0
248         w_max = weight_corr
249         w_min = 0.0
250         #The weight of the MLR part is made from calculating the mean correlations
251         #Between spec country
252         #They are selected N_neighbor countries
253         #so they have higher weight if they are more similar to spec coutnrey
254         predictions=np.zeros(len(MLR_pred))
255         for day in range(len(MLR_pred)):
256             #The error of auto regression increase as time increases
257             #The curve will tend to go to the averaged predictions made by other countries
258             #This is an linear scaling over the weights of other countries according to time
259             weight_MLR=w_function(day,len(MLR_pred),w_min,w_max)
260             w_MLR = weight_MLR / (weight_corr + weight_self)
261             predictions[day] = w_MLR * MLR_pred[day] + (1-w_MLR) * self_pred[day]
262         return predictions
263     else:
264         print("Error: Method not defined! ")
265
266 def w_function(x,length,w_min,w_max):
267     return w_min+(w_max-w_min)/length*x
268

```

```

269
270
271 # def read_data(filename):
272 #     with open(os.path.join("data", filename), "rb") as f:
273 #         OriginDataSet = pd.read_csv(f, keep_default_na=False, na_values='_')
274 #         TrainingSet = OriginDataSet.copy()
275 #         data = TrainingSet["date"]
276 #         print(data)
277 #         refer_date = datetime.datetime.timestamp(datetime.datetime.strptime(data[0], "%m/%d/%Y
278 #     for i in range(len(data)):
279 #         TrainingSet["date"][i] = math.ceil((datetime.datetime.timestamp(datetime.datetime.
280 #             strptime(data[i], "%m/%d/%Y")) - refer_date) / 86400)
281 #         TrainingSet.to_csv(r'.\data\Modified_data.csv', index=False, header=True)

1 import numpy as np
2 from numpy.linalg import solve
3 from findMin import findMin
4 from scipy.optimize import approx_fprime
5 import utils
6
7 # Ordinary Least Squares
8 class LeastSquares:
9     def fit(self, X, y):
10         self.w = solve(X.T@X, X.T@y)
11
12     def predict(self, X):
13         return X@self.w
14
15 # Least squares where each sample point X has a weight associated with it.
16 class WeightedLeastSquares(LeastSquares): # inherits the predict() function from
17     LeastSquares
18     def fit(self, X, y, z):
19         ''' YOUR CODE HERE '''
20         n, d = X.shape
21         V = np.diag(z)
22         self.w = solve(X.T@V@X, X.T@V@y)
23
24 class LinearModelGradient(LeastSquares):
25
26     def fit(self, X, y):
27         n, d = X.shape
28
29         # Initial guess
30         self.w = np.zeros((d, 1))
31
32         # check the gradient
33         estimated_gradient = approx_fprime(self.w.T[0], lambda w: self.funObj(w, X, y)[0],
34             epsilon=1e-6)
35         implemented_gradient = self.funObj(self.w, X, y)[1]
36         if np.max(np.abs(estimated_gradient - implemented_gradient)) > 1e-4:
37             print('User and numerical derivatives differ: %s vs. %s' % (estimated_gradient,
38                 implemented_gradient));
39         else:
40             print('User and numerical derivatives agree.')
41
42         self.w, f = findMin(self.funObj, self.w, 100, X, y)
43
44     def funObj(self, w, X, y):
45         n, d = X.shape
46         ''' MODIFY THIS CODE '''
47         # Calculate the function value
48         # f = 0.5*np.sum((X@w - y)**2)
49         f=0
50         g=np.zeros((d,1))

```



```

48         for i in range(n):
49             ri = w.T@X[i]-y[i]
50             f += np.log(2*np.cosh(ri))
51             g += X[i]*np.tanh(ri)
52         # Calculate the gradient value
53         #g = X.T@(X@w-y)
54
55
56         return (f,g)
57
58
59 # Least Squares with a bias added
60 class LeastSquaresBias:
61
62     def fit(self,X,y):
63         n, d = X.shape
64         Z=np.hstack((np.ones((n,1)),X))
65
66         self.v = solve(Z.T@Z, Z.T@y)
67         self.w = self.v[1:]
68
69     def predict(self, X):
70         n, d = X.shape
71
72         return X@self.w + np.ones((n,1))*self.v[0]
73
74 # Least Squares with polynomial basis
75 class LeastSquaresPoly:
76     def __init__(self, p):
77         self.leastSquares = LeastSquares()
78         self.p = p
79
80     def fit(self,X,y):
81
82         Z=self.__polyBasis(X)
83         self.v = solve(Z.T@Z, Z.T@y)
84         #self.w = self.v[1:]
85
86     def predict(self, X):
87
88         Z=self.__polyBasis(X)
89         return Z@self.v
90
91     # A private helper function to transform any matrix X into
92     # the polynomial basis defined by this class at initialization
93     # Returns the matrix Z that is the polynomial basis of X.
94     def __polyBasis(self, X):
95         ''' YOUR CODE HERE '''
96         n, d = X.shape
97         Z=np.ones((n,1))
98         if self.p !=0:
99             for k in range(1,self.p+1):
100                 Z=np.hstack((Z,X**k))
101         return Z

```

```

1 import numpy as np
2 from numpy.linalg import norm
3
4 def findMin(funObj, w, maxEvals, *args, verbose=0):
5     """
6     Uses gradient descent to optimize the objective function
7
8     This uses quadratic interpolation in its line search to
9     determine the step size alpha
10    """

```

```

11 # Parameters of the Optimization
12 optTol = 1e-2
13 gamma = 1e-4
14
15 # Evaluate the initial function value and gradient
16 f, g = funObj(w,*args)
17 funEvals = 1
18
19 alpha = 1.
20 while True:
21     # Line-search using quadratic interpolation to find an acceptable value of alpha
22     gg = g.T@g
23
24     while True:
25         w_new = w - alpha * g
26         f_new, g_new = funObj(w_new, *args)
27
28         funEvals += 1
29
30         if f_new <= f - gamma * alpha*gg:
31             break
32
33         if verbose > 1:
34             print("f_new: %.3f - f: %.3f - Backtracking..." % (f_new, f))
35
36         # Update step size alpha
37         alpha = (alpha**2) * gg/(2.*(f_new - f + alpha*gg))
38
39     # Print progress
40     if verbose > 0:
41         print("%d - loss: %.3f" % (funEvals, f_new))
42
43     # Update step-size for next iteration
44     y = g_new - g
45     alpha = -alpha*(y.T@g) / (y.T@y)
46
47     # Safety guards
48     if np.isnan(alpha) or alpha < 1e-10 or alpha > 1e10:
49         alpha = 1.
50
51     if verbose > 1:
52         print("alpha: %.3f" % (alpha))
53
54     # Update parameters/function/gradient
55     w = w_new
56     f = f_new
57     g = g_new
58
59     # Test termination conditions
60     optCond = norm(g, float('inf'))
61
62     if optCond < optTol:
63         if verbose:
64             print("Problem solved up to optimality tolerance %.3f" % optTol)
65         break
66
67     if funEvals >= maxEvals:
68         if verbose:
69             print("Reached maximum number of function evaluations %d" % maxEvals)
70         break
71
72     return w, f

```