Wanning Zhou

MATH 7241 Probability 1

12/07/2021

Project Report

Selection and Cleaning of the Dataset

The dataset used in this project is *the Beijing PM2.5 Data Set*, which is collected from the UCI Machine Learning Repository https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data. The dataset contains the hourly Beijing PM2.5 data (all integers) between Jan 1st, 2010, and Dec 31st, 2014, and the missing data is donated as "NA". The raw dataset also contains other kinds of data: the dew point (DEWP), temperature (TEMP), pressure (PRES), combined wind direction (cbwd), cumulated wind speed (lws), cumulated hours of snow (ls), and cumulated hours of rain (lr). Table 1 shows an example of the raw dataset.

No	year	month	day	hour	pm2.5	DEWP	TEMP	PRES	cbwd	lws	ls	lr
1	2010	1	1	0	NA	-21	-11	1021	NW	1.79	0	0
2	2010	1	1	1	NA	-21	-12	1020	NW	4.92	0	0
3	2010	1	1	2	NA	-21	-11	1019	NW	6.71	0	0
4	2010	1	1	3	NA	-21	-14	1019	NW	9.84	0	0
5	2010	1	1	4	NA	-20	-12	1018	NW	12.97	0	0
6	2010	1	1	5	NA	-19	-10	1017	NW	16.1	0	0
7	2010	1	1	6	NA	-19	-9	1017	NW	19.23	0	0
8	2010	1	1	7	NA	-19	-9	1017	NW	21.02	0	0
9	2010	1	1	8	NA	-19	-9	1017	NW	24.15	0	0
10	2010	1	1	9	NA	-20	-8	1017	NW	27.28	0	0
11	2010	1	1	10	NA	-19	-7	1017	NW	31.3	0	0
12	2010	1	1	11	NA	-18	-5	1017	NW	34.43	0	0
13	2010	1	1	12	NA	-19	-5	1015	NW	37.56	0	0
14	2010	1	1	13	NA	-18	-3	1015	NW	40.69	0	0
15	2010	1	1	14	NA	-18	-2	1014	NW	43.82	0	0
16	2010	1	1	15	NA	-18	-1	1014	cv	0.89	0	0
17	2010	1	1	16	NA	-19	-2	1015	NW	1.79	0	0
18	2010	1	1	17	NA	-18	-3	1015	NW	2.68	0	0
19	2010	1	1	18	NA	-18	-5	1016	NE	1.79	0	0
20	2010	1	1	19	NA	-17	-4	1017	NW	1.79	0	0
21	2010	1	1	20	NA	-17	-5	1017	cv	0.89	0	0
22	2010	1	1	21	NA	-17	-5	1018	NW	1.79	0	0
23	2010	1	1	22	NA	-17	-5	1018	NW	2.68	0	0
24	2010	1	1	23	NA	-17	-5	1020	cv	0.89	0	0
25	2010	1	2	0	129	-16	-4	1020	SE	1.79	0	0
26	2010	1	2	1	148	-15	-4	1020	SE	2.68	0	0
27	2010	1	2	2	159	-11	-5	1021	SE	3.57	0	0
28	2010	1	2	3	181	-7	-5	1022	SE	5.36	1	0
29	2010	1	2	4	138	-7	-5	1022	SE	6.25	2	0

Table 1. Raw Data Sample

The column “pm2.5” is chosen to be the data for modeling. And since there is no “error” entries in this column, only the missing data - the ones denoted as “NA” are removed in Excel. The number of rows after cleaning is 41757.

	A	B	C	D	E	F	G	H
1	pm2.5							
2	129							
3	148							
4	159							
5	181							
6	138							
7	109							
8	105							
9	124							
10	120							
11	132							
12	140							
13	152							
14	148							
15	164							
16	158							
17	154							
18	159							
19	164							
20	170							
21	149							
22	154							
23	164							
24	156							
25	126							
26	90							
27	63							
28	65							
29	55							
30	65							

Table 2. Data After Cleaning Sample

Choice of States for Markov Chain

The choice of states for the dataset is based on the air quality category from the United States Environmental Protection Agency. To be specific, when the PM2.5 is between 0 and 12, it is in “good” category; when it is between 13 and 35, it is in “moderate”; when it is between 36 and 55, it is in “unhealthy for sensitive groups”; when it is between 56 and 150, it is in “unhealthy”; when it is between 151 and 250, it is in “very unhealthy”; when it is greater than 251, it is in “hazardous”. However, since the range for “hazardous” is bigger than the others, it is divided into three smaller pieces: from 251 to 350; from 351 to 500, and greater than 501. In this case, there are 8 states for the model, with the states {1, 2, 3, 4, …, 8} corresponding to the categories. Table 3 shows an example of choice of the states.

		pm2.5	State
1	0	129	4
2	1	148	4
3	2	159	5
4	3	181	5
5	4	138	4
6	5	109	4
7	6	105	4
8	7	124	4
9	8	120	4
10	9	132	4
11	10	140	4
12	11	152	5
13	12	148	4
14	13	164	5
15	14	158	5
16	15	154	5
17	16	159	5
18	17	164	5
19	18	170	5
20	19	149	4
21	20	154	5
22	21	164	5
23	22	156	5
24	23	126	4
25	24	90	4

Table 3. Data With States Sample

Empirical Distribution

The data is imported into Python for analysis. To calculate the empirical distribution, the occupation frequencies at each state (number of rows for each state) are counted by using "groupby" function in Python. And each frequency is divided by the total number of rows, to achieve the fraction of time spent in each state. And it is the empirical distribution of the chain.

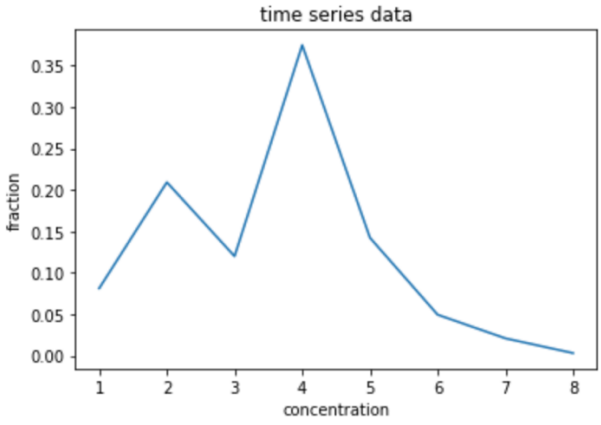

Figure 1. Empirical Distribution from Time Series

Transition Matrix

To calculate the transition matrix, the frequencies of jumps between each pair of states need be computed, and these frequencies should be divided by the occupation frequencies at

each state, so that the total jump probability for each state is 1. To be specific, first, each state can be paired up with the one in the next row by using “zip” function in Python. And by using “count” function in Python to compute the numbers of these pairs of states. Second, these frequencies are divided by the occupation frequencies at each state that are computed previously, so that the total probability is 1. However, in this case, the last state for the data is “1”, which means it cannot jump to the next state, so that the occupation frequency for state 1 should minus 1. And these probabilities set up the transition matrix.

```
[[0.7212 0.2732 0.0035 0.0015 0.0006 0.     0.     0.    ]
 [0.1039 0.7669 0.1135 0.0149 0.0007 0.0002 0.     0.    ]
 [0.0042 0.1749 0.5932 0.2265 0.0012 0.     0.     0.    ]
 [0.0009 0.0136 0.0644 0.8643 0.0556 0.0011 0.0001 0.0001]
 [0.0005 0.0035 0.0032 0.1379 0.7925 0.0608 0.0012 0.0003]
 [0.     0.0015 0.0024 0.016  0.1593 0.7499 0.0699 0.001 ]
 [0.     0.     0.     0.0023 0.0173 0.1554 0.786  0.0391]
 [0.     0.     0.008  0.     0.04   0.     0.264  0.688 ]]
```

Figure 2. Transition Matrix from Time Series

Stationary Distribution

From the equation $wP = w$, where w is the stationary distribution, it can be seen that w is a left eigenvector of P with eigenvalue equals 1. In other words, the stationary distribution can be obtained by finding the eigenvectors in which the corresponding eigenvalues are 1. To be specific, the Python function “scipy.linalg.eig” is used to calculate the left eigenvectors and eigenvalues of the transition matrix. And select the eigenvector where its corresponding eigenvalue is 1, and then normalize it. As a result, the stationary distribution is computed.

```
[0.0812 0.2093 0.1198 0.3744 0.1421 0.0493 0.0208 0.003 ]
```

Figure 3. Stationary Distribution

By comparing the empirical distribution from time series and the stationary distribution of the chain, it can be seen from Figure 4 that the similarity between these two graphs is very high.

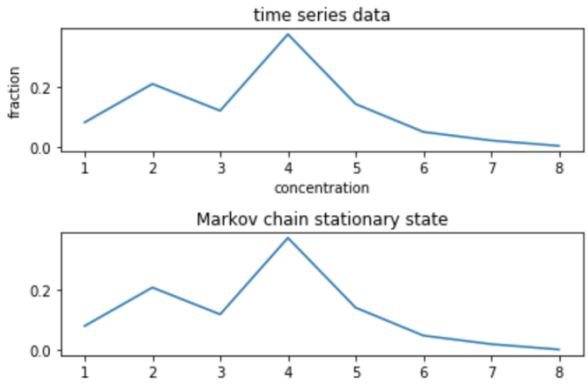

Figure 4. Empirical Distribution Compared to Stationary Distribution

Simulation of Markov Chain

To simulate the Markov Chain, the states {1, 2, …, 8} will be randomly chosen based on the transition matrix computed previously. In addition, the number of steps and the starting state needs to be chosen. In this case, the number of steps is set to be 1001, and the starting state is chosen to be state 4 (according to the actual dataset). By using the "np.random.choice" function with a while-loop in Python, the Markov Chain can be obtained. It can be seen from Figure 5 that there are a lot of differences between the two time series.

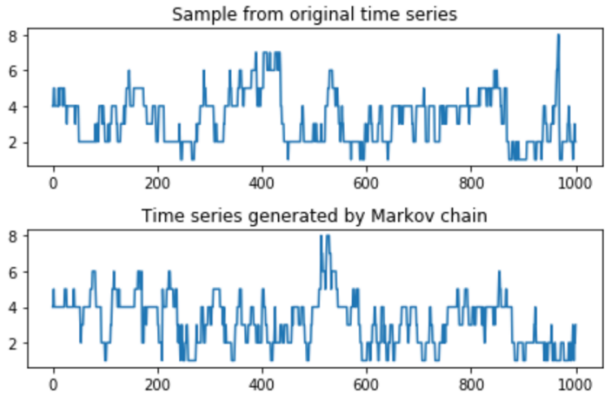

Figure 5. Simulation of Markov Chain Compared with Original Time Series

Autocorrelation

The autocorrelation function is used to calculate the autocorrelation for the simulation of Markov Chain and the original time series. To be specific, the Python function "statsmodels.tsa.stattools.acf" is used to compute the autocorrelation, with time lag equals 10. Figure 6 shows the autocorrelation results, in which each number represents $R(0), \dots, R(10)$ respectively. It can be seen from Figure 6 and 7 that there are only slightly differences between the autocorrelations for the two time series, and it looks like they are describing the same series.

```
origin data:  [1.      0.9286 0.8636 0.8005 0.7442 0.6913 0.6476 0.6141 0.5831 0.5521
 0.5256]
markov chain:  [1.      0.9384 0.8846 0.8331 0.7866 0.7364 0.6936 0.6453 0.5954 0.5477
 0.5069]
```

Figure 6. Autocorrelation Results

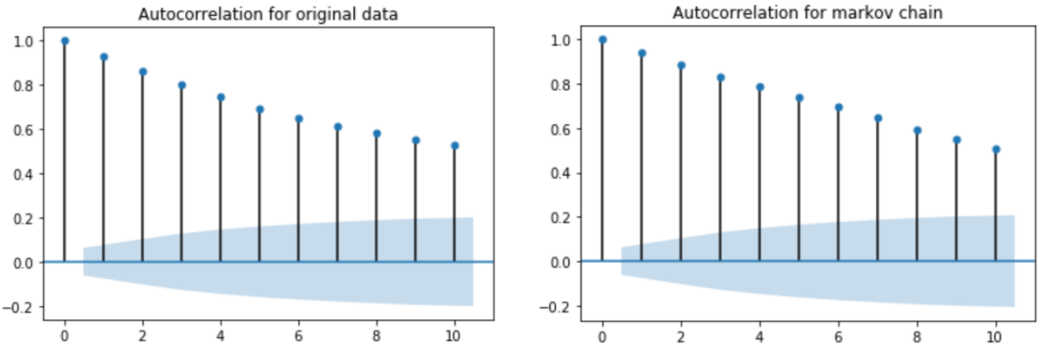

Figure 7. Autocorrelation Comparison

Goodness of Fit Test

Since the goodness of fit test will use the 2-step transition probabilities to compare the simulation of Markov Chain with the original time series, some values need to be computed first. To be specific, first, the matrix N_{ij} contains the frequencies from each state i to each state j in two steps in the original time series. And it can be computed using the same techniques for the transition matrix: using "zip" function in Python to pair up the states, and use "count" function to count the number of different pairs, then divide these frequencies by the time spent in each state

(in this case, since the second last and the last states are both 1, time spent in state 1 should minus 2). Figure 8 shows the matrix N_{ij}. And it can be seen that $N_i = \sum_j N_{ij} \approx 1.0$.

Second, the matrix q_{ij} is computed by the transition matrix $\widehat{p_{ij}}$, by using "np.linalg.matrix_power" function in Python to computer the square of $\widehat{p_{ij}}$. And since $N_i \approx 1.0$, $M_{ij} = N_i q_{ij} \approx q_{ij}$. Figure 9 shows the matrix M_{ij}. Third, the goodness of fit test is used to compare the observed frequencies $\{N_{ij}\}$ and the expected frequencies $\{M_{ij}\}$ for each state i. By using "scipy.stats.chisquare" function in Python for each state (The entries 0 in M_{ij} are removed for running the test), the results of the test are obtained. Figure 10 shows the result for the test.

```
[[0.6263 0.3554 0.0115 0.005  0.0018 0.     0.     0.    ]
 [0.131  0.6709 0.1538 0.0426 0.0013 0.0005 0.     0.    ]
 [0.0108 0.2113 0.4459 0.3282 0.0032 0.0006 0.     0.    ]
 [0.0033 0.0327 0.0834 0.7889 0.0885 0.0027 0.0003 0.0001]
 [0.0022 0.0138 0.0104 0.1979 0.6783 0.0913 0.0054 0.0007]
 [0.0015 0.0083 0.0092 0.0355 0.222  0.6144 0.1054 0.0039]
 [0.     0.0046 0.0023 0.0207 0.0357 0.2278 0.6548 0.0541]
 [0.008  0.     0.016  0.024  0.04   0.032  0.368  0.512 ]]
```

Figure 8. Matrix N_{ii}

```
[[0.5485 0.4072 0.0358 0.0073 0.0012 0.0001 0.     0.    ]
 [0.155  0.6365 0.1557 0.0502 0.0021 0.0004 0.     0.    ]
 [0.0239 0.2421 0.3864 0.3329 0.0144 0.0004 0.     0.    ]
 [0.0031 0.0338 0.0955 0.7695 0.0924 0.0052 0.0004 0.0001]
 [0.0013 0.0082 0.0139 0.2303 0.6455 0.0941 0.0062 0.0006]
 [0.0003 0.0034 0.005  0.0486 0.2478 0.5829 0.1079 0.0042]
 [0.     0.0003 0.0009 0.0087 0.0537 0.2396 0.6389 0.0578]
 [0.0001 0.0015 0.0104 0.0079 0.0638 0.0434 0.3892 0.4837]]
```

Figure 9. Matrix M_{ij}

```
chi square for state 1:  Power_divergenceResult(statistic=0.03524354060641094, pvalue=0.999987751462477)
chi square for state 2:  Power_divergenceResult(statistic=0.007078841481229051, pvalue=0.9999997763049218)
chi square for state 3:  Power_divergenceResult(statistic=0.029138320617812054, pvalue=0.9999923705596081)
chi square for state 4:  Power_divergenceResult(statistic=0.0034624213302579315, pvalue=0.9999999999814648)
chi square for state 5:  Power_divergenceResult(statistic=0.011756875618125414, pvalue=0.9999999986671322)
chi square for state 6:  Power_divergenceResult(statistic=0.023388650326453396, pvalue=0.9999999852660363)
chi square for state 7:  Power_divergenceResult(statistic=0.08761003696559391, pvalue=0.9999864428685025)
chi square for state 8:  Power_divergenceResult(statistic=0.676110153707457, pvalue=0.9985126368198176)
```

Figure 10. Goodness of Fit Test Result

Conclusion

It can be seen from Figure 10 that the p-value for each state is greater than 0.05, which means that we cannot reject the null hypothesis H_0 for each state. In other words, the observed frequencies $\{N_{ij}\}$ and the expected frequencies $\{M_{ij}\}$ fit well. As a result, the Markov Chain method produces a good model for this time series.