Assignments
APSSS 2020

Emma Blomgren
DTU

March 2020

1. Part I

1.1. Task 1: Decision Trees (DT) and Neural Networks (NN) for Power System Security Assessment

3. My comments on the code can be read in the file: `DT_and_NN_Security_Assessment_14bus-emvb.py`

4. I train a decision tree based on the given code in the assignment file. I get the following:

```
train a decision tree and compute accuracy
Decision tree training accuracy 100.0
Decision tree test accuracy 99.99496120124962
```

I train a neural network based on the given code in the assignment file and get the following:

```
train a neural network and compute accuracy
Neural network training accuracy 99.75308409127588
Neural network test accuracy 99.94333470898326
```

The decision tree get 100 % accuracy for the training data set, and also a very high accuracy for the test data set (99.99%). Since the rules for the decision tree is based on the training data set, it might not be a surprise that the accuracy for the training data set is 100%. Given that the test data set comes from the same data base it is also not a surprise that also this accuracy is very high. However, applying the decision tree on other data bases will probably not result in such high values for the accuracy. The neural network get a higher accuracy for the test data set (99.5%) than for the training data set (98.3%). Although these results might seem good at a first glance, it is not good to have a misleadingly high accuracy for the test data, which could be the case here and more metrics are needed to evaluate this.

5. The data base is not balanced. The number of safe classifications are 885 out of 49615 data points. Thus, it is not enough to only look at the accuracy. For instance the accuracy could be good for the predicted negative (unsafe) values, but not the positive (safe) or vice versa. Hence, I evaluate recall (true positive rate) an Matthews correlation coefficient (MCC) for both the decision tree and the neural network.

```
Decision tree train recall 1.0
Decision tree train MCC 1.0
Decision tree test recall 0.9894736842105263
Decision tree test MCC 0.9946207329975496
----------------------
Neural network train recall 0.9424460431654677
Neural network train MCC 0.9687932098677032
Neural network test recall 0.9631578947368421
Neural network test MCC 0.9810533557377396
```

From this it is seen that the prediction is perfect for the decision tree training data set, which

is a natural outcome since the decision rules are made for that exact data set. The value for the MCC on the test data set is very close to one, which indicates a close to perfect prediction. The test recall is close to the test accuracy, which indicates a fairly balanced result, i.e. the accuracy is a little bit skewed towards less true positive predictions, but not much. The values for the neural network for both test and training data set as well as both recall and MCC are slightly lower, which leaves some room for improvement but might also reflect realistic measures of that it is not possible to make a perfect prediction.

6. In he previous results the neural network had the activation function ReLU. Thus I try with sigmoid activation function instead. I get the following results:

```
train a neural network and compute accuracy
Neural network training accuracy 99.63722694779624
Neural network test accuracy 99.9119686392605
Neural network train recall 0.994579945799458
Neural network train MCC 0.9836648766890023
Neural network test recall 0.9863945578231292
Neural network test MCC 0.9537640767928418
```

From these metrics there is not a significant difference. Given the MCC the test predictions are a bit more random, but also the outcome of this metric could be a result of natural randomness.

Since the model is already quite accurate I try to 'decrease' it by first removing one layer and then decreasing the number of neurons by 80% and 50% (I also removed another layer to only have one layer but values of 0.0 were already shown for 2 layers). The results are given for accuracy and MCC for each case in the table below, i.e. on the form (accuracy value, MCC value).

	3 layers	2 layers
100% neurons (20)	Train (99.64, 0.98)	Train (96.93, 0.0)
	Test (99.91, 0.95)	Test (99.3, 0.0)
80% neurons (20)	Train (99.43, 0.98)	Train (96.47, 0.0)
	Test (99.87, 0.96)	Test (99.21, 0.0)
50% neurons (20)	Train (97.49, 0.0)	Train (97.73, 0.0)
	Test (99.45, 0.0)	Test (99.49, 0.0)

From the table it can be seen that the accuracy is not affected a lot by removing layers or neurons. However, removing only one layer gives a MCC value of zero which indicates random predictions in both the training and test data set. Decreasing the neurons by 80% still gives valid results given the MCC, whereas decreasing to 50% (only having 10 neurons) gives random results (MCC is 0). Thus, the network seems sensitive towards removing layers and neurons in the applicability of the model, since it gives deceivingly high values of accuracy given the random predictions indicated by the MCC. A more balanced data set could mitigate the results for the MCC when decreasing the network.

The above neural networks are trained with sigmoid. I try ReLU for 50% of neurons and 2 layers and get (98.14, 0.0) for the training set and (99.63, 0.0) for the test set, which is not very different from the results in the table. Thus, the choice between ReLU and sigmoid as activation function,

does not seem to impact the result of these metrics severely.

7. The decision tree get very good performance metrics, which are almost perfect. However, these metrics are for the specific database that was given, which is also unbalanced in the safe vs unsafe data points. Thus the decision tree might not be very accurate for another database, which might not have the same balance between safe and unsafe points and is not as flexible as a neural network when it comes to being updated to new sets of data. The performance metrics of the neural network might reflect more realistic numbers when it comes to applying it to another database, especially if the neural network is updated or retrained for the new data. It should however be mentioned that tractability is clearer and better in a decision tree compared to a neural network (that is not physics informed).

1.2. Taks 2: Physics-Informed Neural Networks (PINN)

2. See my comments in file: `PINN_inference_swing_equation-emvb.py`

3. Setting Nf to 100 gives:

```
Objective function value:  0.000000
Number of iterations:  4500
Number of functions evaluations:  4793
Training time:  57.5064
Error u:  9.739179e-02, error u_t:  1.143034e+00
Error u:  6.692930e-01, error u_t:  6.997319e-01
Error u:  6.927840e-01, error u_t:  8.507968e-01
Error u:  8.638784e-01, error u_t:  7.386798e-01
Error u:  4.375559e-01, error u_t:  1.942450e+00
Error u:  1.316649e+00, error u_t:  7.405907e-01
Error u:  1.945489e+00, error u_t:  9.815011e-01
Error u:  4.778797e+00, error u_t:  3.485711e+00
Error u:  1.215896e+00, error u_t:  6.731130e-01
Error u:  6.581933e-01, error u_t:  1.248247e+00
```

4. Setting Nf to 1000 gives:

```
Objective function value:  0.000003
Number of iterations:  19578
Number of functions evaluations:  21100
Training time:  507.5964
Error u:  5.255047e-01, error u_t:  4.181949e-01
Error u:  9.872104e-02, error u_t:  8.903215e-02
Error u:  8.455374e-01, error u_t:  1.031300e+00
Error u:  3.584366e-01, error u_t:  3.245188e-01
Error u:  1.036915e-01, error u_t:  1.197704e-01
Error u:  9.486763e-02, error u_t:  1.746584e-01
Error u:  5.163786e-01, error u_t:  5.630936e-01
Error u:  7.237756e-02, error u_t:  1.381382e-01
Error u:  1.547295e-01, error u_t:  1.191469e-01
Error u:  5.037165e-01, error u_t:  4.030034e-01
```

Given the increase in collocation points the PINN takes much more time to train (507.6 compared to 57.5 seconds). However, the errors are generally slightly lower compared to using Nf=100, since there is more 'physics data' (collocation points) to verify with in the PINN with Nf=1000. Given these results deciding on the amount of collocation points is a comprise between precision of the PINN and the computational time.

1.3. Task 3: General Questions

1. In section 1.3.1 the DT and NN are trained on data without any realization of the physical system/model. In section 1.3.2 the physical system is taken into account when training the PINN, by including the swing equation in the loss function (minimizing the error between f(x,t) and zero). Additionally, the DT and the NN in section 1.3.1 are trained to classify set points as either safe or unsafe and is thus a classification problem. The PINN in in section 1.3.2 is trained to determine the different variables in the power system and the solutions to corresponding differential equations. That means that it is trained to set variables in the power system that are safe and is thus a more of a regression problem.

2. The training data in 1.3.1 is 39692 data points whereas in 1.3.1 it is 110 (Nu=10, and Nf=100). I would expect the regression to require more data, since the classification problem have two different outcomes (in this case) and the continuous outcome space is much larger (infinite). However, it depends on the problem is stated and for regression the amount of data required could depend on how far ahead that you would like to predict for example. Since, the regression neural network in this case is validated against physical information I however see it as a natural outcome that this NN requires less input data.

3. A balanced database is of high importance, to get a valid DT and NN that is applicable to other databases, however the DT is probably a lot more sensitive towards unbalanced data. The accuracy metric cannot be valid, since if the database is skewed the accuracy could be skewed giving misleading information about the applicability of the trained NN or DT. However, as we discussed in class, in some cases it can be good to have unbalanced databases for an NN if one would like to investigate a specific outcome especially. For example if the correct classification of unsafe is more important or if the boundary between unsafe and safe is of higher importance, one could have more data points from these cases.

4. Benefits are that the solution of the NN is validated against physical data, and thus the neural network is not a black-box model anymore. Compared to other models it has the potential to be faster in computational time and require less work to develop compared to white-box models. It could also utilize and handle a lot of data compared to white-box models. However, there are some areas that requires future research to remove the barriers and move the technology in to the real world. Firstly, the precision around the boundaries between safe and unsafe solution needs to be better. More research about adversarial examples in PINN are required to investigate the probability of these occurring and what impact it has in the physical system. Additionally, real world testing of the PINN is required as well as investigations of which degree of tractability that will be required from the PINN. For example, if there is a contingency, is the tractability already enough or will a higher degree of tractability be required to account for and solve these situations.

2. Part II

2.1. Tasks

1.a) Reward and performance metrics are provided for the best run of the algorithm, using SAC and 15 episodes, for each climate zone respectively below. (for any further details see file: _central_agent-emvb.ipynb)

Best run for climate zone 1 takes 687 seconds and gives:

```
Cumulated reward:  -45996.91317344912
'ramping':  1.8300288108504037,
'1-load_factor':  1.202397031089419,
'average_daily_peak':  1.3219636870683984,
'peak_demand':  1.2404710178570737,
'net_electricity_consumption':  0.9753661615529869,
'quadratic':  1.163771630580182,
'total':  1.2889997231664108
```

Best run for climate zone 2 takes 798 seconds and gives:

```
Cumulated reward:  -26939.56077285986
'ramping':  1.77044382014489,
'1-load_factor':  1.0458981415481223,
'average_daily_peak':  1.0996847799956055,
'peak_demand':  1.0640327705405013,
'net_electricity_consumption':  0.9701009601905082,
'quadratic':  1.022703666637404,
'total':  1.1621440231761717
```

Best run for climate zone 3 takes 895 seconds and gives:

```
Cumulated reward:  -22812.015215049392
'ramping':  1.6182309723270267,
'1-load_factor':  1.1155746878116084,
'average_daily_peak':  1.1071333094046378,
'peak_demand':  1.0520294290800842,
'net_electricity_consumption':  0.9846132461519269,
'quadratic':  1.0553418539209354,
'total':  1.1554872497827031
```

Best run for climate zone 4 takes 864 seconds and gives:

```
Cumulated reward:  -14773.854150837034
'ramping':  1.6409662809141874,
'1-load_factor':  1.143924528770195,
'average_daily_peak':  1.2006750054416162,
'peak_demand':  1.1877147082247694,
```

```
'net_electricity_consumption':  0.9761626848001709,
'quadratic':  1.1499390219472876,
'total':  1.2165637050163711
```

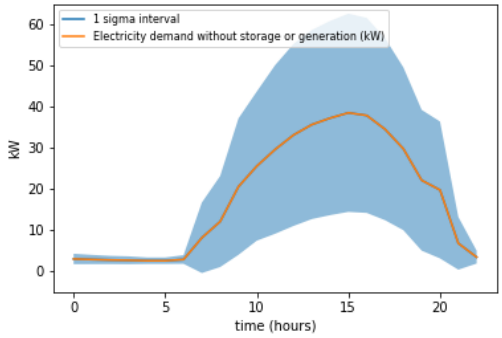
b) **Batch size**: A larger batch size makes the performance better, i.e. the cumulated reward and the metrics lower. **Learning rate** adjusts the weight on the learning, i.e. if learning rate is higher, it learns with more weight for each step that is wrong or right. A too low learning rate affects the perfomance in a negative way. However, a higher learning rate, can make the algorithm quicker, but also at the risk of making the process 'jumpy' which could affect the performance in a negative way too. **Tau** is an update coefficient to slowly and more softly update the network. If 'softening' too much this could affect the performance negatively. **Gamma** is a discount factor, meaning that it is a weight for the importance of rewards in the future. For this case it seems that a too low gamma decreases the performance in the cumulated reward. This seems reasonable, since the algorithm needs to 'plan' for future energy consumption and production in the dynamical system.

c) A larger batch size increases the computing as more data is passed through the algorithm during one epoch. In my case, a too low learning rate made the running time a bit longer as well, which I assume is because the algorithm finds a better solution quicker with a higher or more well adjusted learning rate.

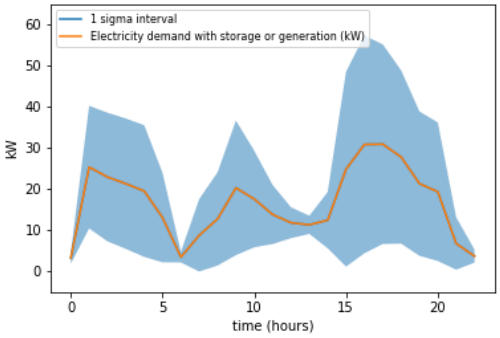
d) I tried to add states, such as direct solar radiation, as well as predictions 6 hours and 12 hours ahead. However, this increased the computing time severely without making a significant improvement in the performance. This happens since adding states means to increase the neural network, making it slower. Additionally, adding too many states could possibly mean adding redundant information and/or correlated inforomation. Thus I choose to work with the states: hour, t_out, t_out_pred_12h, non_shiftable_load and cooling_storage_soc. I chose to have the non shiftable electrical load as a state because I think this information is important to the algorithm to account for how much flexibility that can be achieved with the cooling storage and heating/cooling demand and leaving the load that is not shiftable 'as it is' to reflect reality. Cooling storage is utilized, since the building is assumed to have cooling storage. T_out as well as a 12 hour prediction of t_out is used to account for heating/cooling demand at the moment and plan for demand in the future.

2. The plots before and after energy storage for each climate zone can be seen below in the figures.

Climate Zone 1:



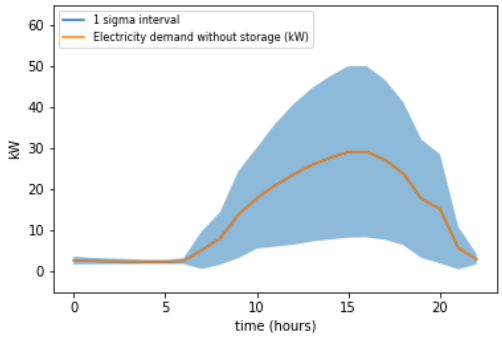
(a) *Climate zone 1 without/before storage.*



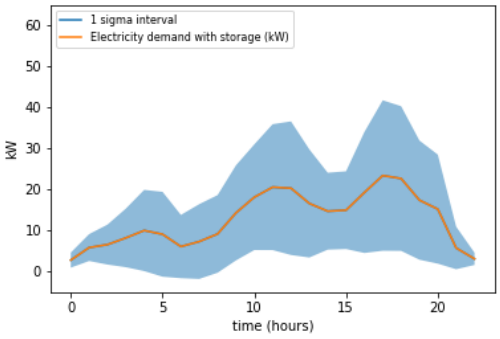
(b) *Climate zone 1 with/after storage.*

Figure 1: *The orange lines represent the average value for each hour given every day of the year and the blue spaces represent ± one standard deviation.*

Climate zone 2:



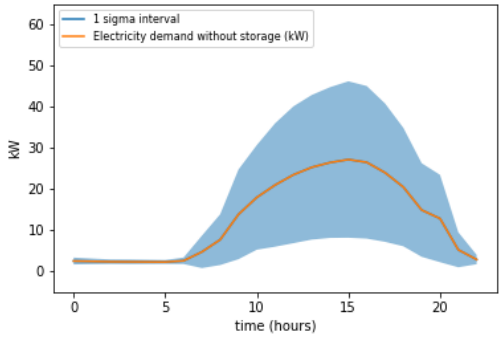
(a) *Climate zone 2 without/before storage.*



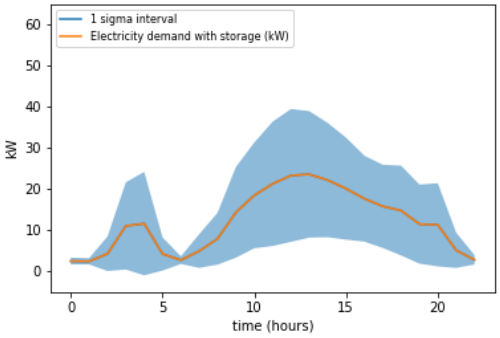
(b) *Climate zone 2 with/after storage.*

Figure 2: *The orange lines represent the average value for each hour given every day of the year and the blue spaces represent ± one standard deviation.*

Climate zone 3:



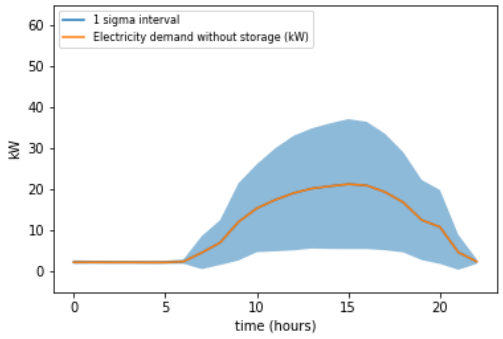
(a) *Climate zone 3 without/before storage.*



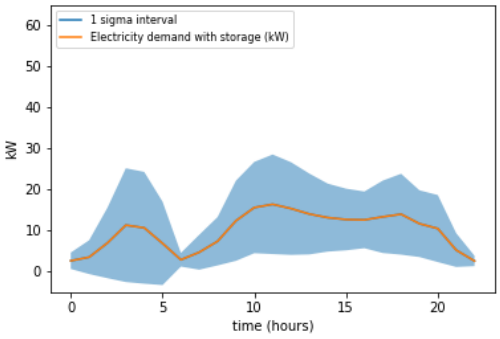
(b) *Climate zone 3 with/after storage.*

Figure 3: *The orange lines represent the average value for each hour given every day of the year and the blue spaces represent ± one standard deviation.*

Climate zone 4:



(a) *Climate zone 4 without/before storage.*



(b) *Climate zone 4 with/after storage.*

Figure 4: *The orange lines represent the average value for each hour given every day of the year and the blue spaces represent ± one standard deviation.*

3. For climate zone 1 and 2, three smaller peaks are achieved instead of one large peak during the day. For all climate zones some consumption is shifted to the night (0 to 5 am) adding a peak, where only a base load could be seen before. This is due to the fact that storage is added, which allows for energy to be stored at these hours and used during the day when it is required to lower the peak consumption. For climate zone 1 the standard deviation varies quite a lot throughout the day, while it does not vary as much for climate zone 2, 3 and 4. However, for climate zones 3 and 4 there is one time span around 6 am where the standard deviation is almost zero. Apparently, at this time the algorithm finds very similar solutions each day of the year. In general similar hyperparameters work well for the climate zones. However, as seen in the table below, for climate zone 1 I get a better result with a learning rate of 0.015 and a batch size of 200 instead of 0.01 and 300 respectively. This could be due to the fact that the data and the solutions for climate zone 1 appears to have a larger variance and also larger differences in variance compared to the other

climate zones. I assume that this could affect the training process of the network.

All climate zones receive similar results in the decrease in net electricity consumption, except for climate zone 3 where the metric is 0.985 as opposed to being below 0.98 as for the other zones. One reason for this could be due to the cooling/heating demand being lower in relation to the demand from non shiftable loads and thus, leading to a lower degree of flexibility. This can also be seen slightly in the figure 3. b) where some of the peak is still there during day time hours.

Climate Zone	learning rate	gamma	tau	batch size
1	0.015	0.99	0.0003	200
2	0.01	0.99	0.0003	300
3	0.01	0.99	0.0003	300
4	0.01	0.99	0.0003	300

4. No, they do not reach similar values. Particularly, the net electricity consumption reaches values below 1 while the rest of the metrics reach values above 1. According to the city learn github website (`https://github.com/intelligent-environments-lab/CityLearn`), the reward function is a function of the net electricity consumption. Thus it seems logical that the this metric, specifically, will be better compared to the rule based controller as the reward increases.

3. Part III

3.1. Tasks

For reference the results from the example given in class are the following figures and outputs:

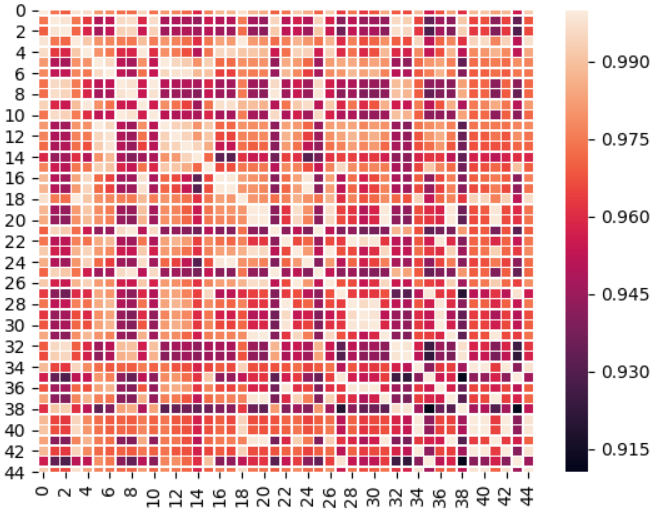


Figure 5: *Correlation matrix for example in class.*

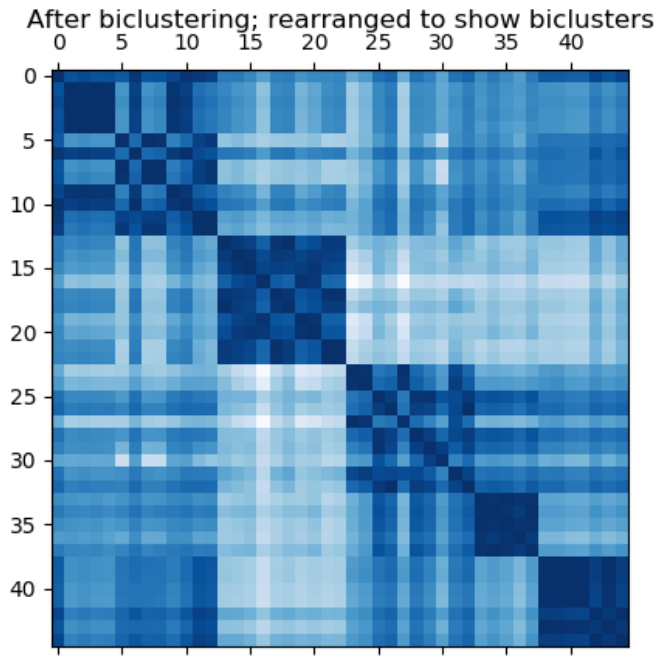


Figure 6: *Biclustered correlation matrix for example in class.*

A multinomial linear regression with 30 data points in the training data set and 14 in the the test data set gives:

```
Predicted labels:  [1 3 2 2 3 1 1 3 2 3 3 3 3 1 3]
True labels:  [1 3 2 2 3 1 1 3 2 3 3 3 3 1 3]
Probability for each label:
   [9.99967802e-01 3.79076567e-06 2.84068349e-05
1.22564943e-06 3.38884176e-11 9.99998774e-01
1.33690730e-05 9.99560681e-01 4.25950095e-04
1.59619559e-06 9.99812134e-01 1.86269757e-04
8.03132494e-09 6.21275650e-08 9.99999930e-01
9.99991305e-01 1.79731269e-06 6.89753158e-06
9.99969693e-01 3.21377885e-06 2.70928134e-05
1.04138017e-07 6.44345651e-12 9.99999896e-01
1.53962254e-08 9.99942050e-01 5.79346605e-05
1.07499974e-08 6.21289458e-08 9.99999927e-01
9.14031796e-09 8.43036905e-08 9.99999907e-01
6.34818338e-08 4.08588178e-12 9.99999937e-01
8.32330214e-09 5.28970641e-08 9.99999939e-01
9.99994030e-01 8.36271980e-07 5.13386212e-06
3.06995061e-07 5.75573612e-07 9.99999117e-01

Target accuracy:  1.0
```

```
Logistic loss:  5.0999655017012436e-05
```

For noisy data:
1. a) I generate noisy data with a scale of 5. And get the results in Fig. 7 and 8 below. For comments, see 1. b) and for code see attached file `noisy_data_solution.py` (I used the code given from the examples in class).

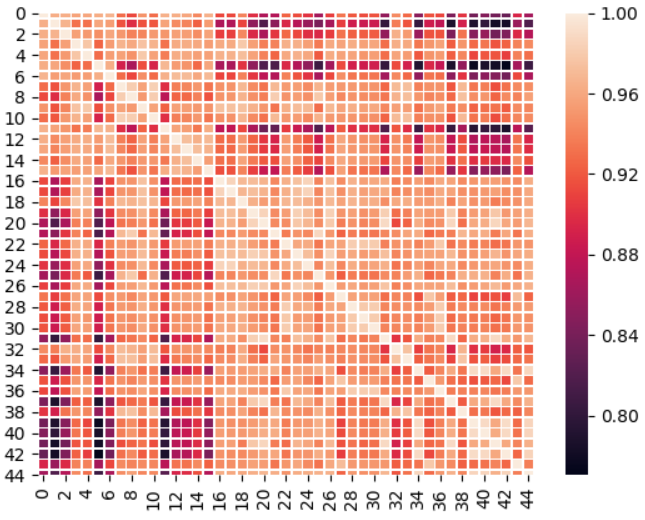


Figure 7: *Correlation matrix for noisy data.*

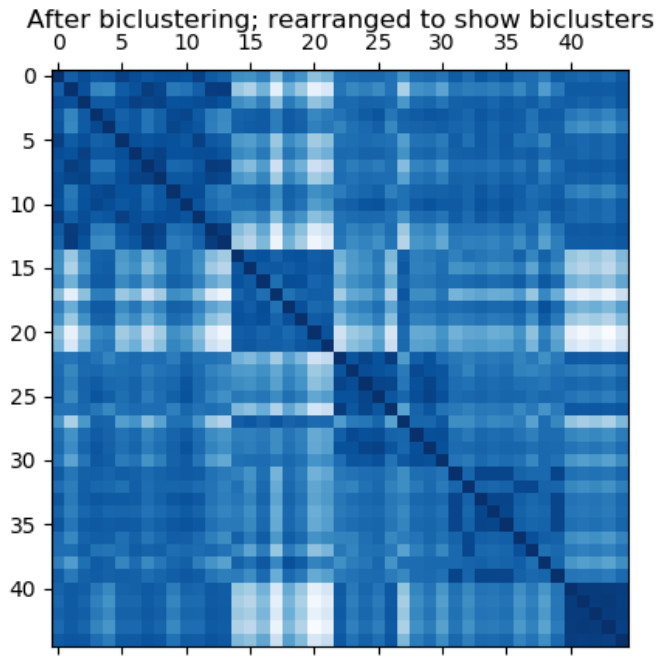


Figure 8: *Biclustered correlation matrix for noisy data.*

b.) The results for the noisy data including the predicted labels with the probability, accuracy and logistic loss are presented below. Comparing the matrices for the noisy data with the example given in class Fig. 8 and 6, respectively, the biclustering is less clear from a visual perspective. However, the multinomial logistic regression manage to find the correct parameters with a target accuracy of 100% and a logistic loss of 0.0006. While, the accuracy remains unchanged from the example in the class the logistic loss has increased by a magnitude of 10 (approximately). This means that although we managed to predict it perfectly, we are less sure about the solution, which is a logical consequence of adding noise to the data.

```
Predicted labels:  [1 3 2 2 3 1 1 3 2 3 3 3 3 1 3]
True labels:  [1 3 2 2 3 1 1 3 2 3 3 3 3 1 3]
Probability for each label:
   [9.99713498e-01 2.81703463e-05 2.58331462e-04
4.10904471e-06 3.89061980e-08 9.99995852e-01
3.17551646e-03 9.95340037e-01 1.48444654e-03
6.30013267e-04 9.98071361e-01 1.29862561e-03
2.40878746e-06 6.71164931e-05 9.99930475e-01
9.99777133e-01 6.22210612e-05 1.60645921e-04
9.99765404e-01 2.07634398e-05 2.13832484e-04
1.61802303e-06 1.60391451e-08 9.99998366e-01
1.93541864e-05 9.99450257e-01 5.30388867e-04
2.80905570e-06 6.56300350e-05 9.99931561e-01
3.11915142e-06 6.74212028e-05 9.99929460e-01
```

```
1.17194403e-06 1.28552268e-08 9.99998815e-01
2.20533851e-06 4.69625767e-05 9.99950832e-01
9.99734649e-01 4.56885951e-05 2.19662896e-04
7.45578476e-05 1.78086429e-04 9.99747356e-01

Target accuracy:  1.0
Logistic loss:  0.0005785345823224771
```

c) I increase the scale parameter to 20.0 and get the results below. By increasing the scale, the noise becomes larger in relation to the original data. Given the central limit theorem stating that when independent identically distributed random variables are added to a set of data, the sum will tend towards a normal distribution even though, the original data is not normally distributed. Thus, when adding larger and larger numbers of noise, i.e. with a larger scale, the data will tend towards a normal distribution. This can somewhat be seen when comparing Fig. 6, 8 and 9, where the coloring becomes less distinct as the noise that is added becomes proportionally larger and is also an indication of the data being more evenly distributed. Given that we now add a normally distributed part to the data that does not reflect the original distribution the classifier will perform worse as it is not a true reflection of the real world system. This is seen in the target accuracy being 0.98 instead of 1.0 and the loss value of 0.12, and indicates that the predictions are less accurate and less certain.

```
scale=20.0 gives:
Target accuracy:  0.9777777777777777
Logistic loss:  0.14341716573799043
```

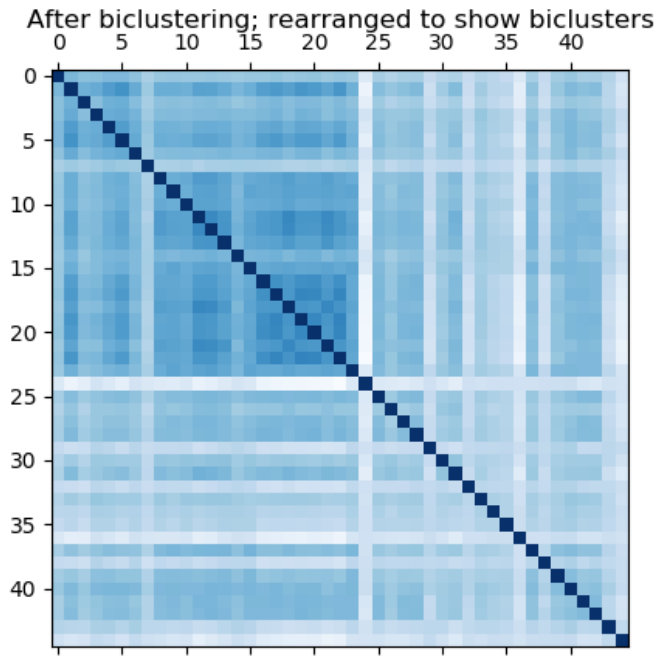


Figure 9: *Biclustered correlation matrix for noisy data and a scale factor of 20.*

d) With a scale of 20.0 as in the previous task I get an accuracy of 0.98 and a logistic loss of 0.008, which I would consider as acceptable results. Below this paragraph are results for a scale of 30 and 50 respectively. What is considered as poor performance depends a lot on the application of the problem. I would consider the the case with a scale of 30 as poor performance, giving an accuracy and loss of 0.93 and 0.39 respectively, meaning that 8% of the predictions would be wrong. However, with a scale of 50 the performance is definitely poor, giving an accuracy of 0.87 and a loss of 0.67. The maximum value of the noise vector in the file `generate_noisy_data.py` I get on one run is 0.47. scaling this with 5 gives 2.36. This scale of error in the measurement/noise seems that it could occur. However, scaling with 30 gives 14.14 and scaling with 50 gives 23.57. Given that the voltage is allowed to deviate ± 10% in the Danish grid, i.e. ± 23 V, it seems unreasonable to have a measuring noise at this scale that would trigger warnings at the operator. Also ± 14 V seems unreasonable. In Sweden the voltage is allowed to deviate ± 3-5%, which would also trigger warnings and I would assume that voltage meters are probably designed to have less noise than this. However, it should be said that unpredicted events in the measurement could occur and possibly leading to outliers at these values.

```
Results:

scale=30.0
0.9333333333333333
0.3932669181755182

scale=50.0
```

```
0.8666666666666667
0.6717816139165517
```

For correlated noisy data (2. a-b): see comparing comments for 2.b) at each question 1.a) to d) and for code see file `corr_noisy_data_solution.py` (I used the code given from the examples in class).

1. a) The correlation matrix and bisclustered correlation matrix can be seen in Fig. 10 and 11 respectively. Comparing Fig. 10 to the non-correlated case in Fig. 7 a distinct difference can be seen that the data points have a stronger correlation with their neighbours in the matrix giving the 'highlighted' diagonal. This is different from the non-correlated data where one data point is not necessarily correlated with its neighbour, but rather with a data point on the same phase. There is a hint of clusters in the biclustered correlation matrix (Fig. 11) but it is not as clear as before in Fig. 8. It is also seen that there are more correlation between neighbouring data points, since correlation between them is added in the noise data.

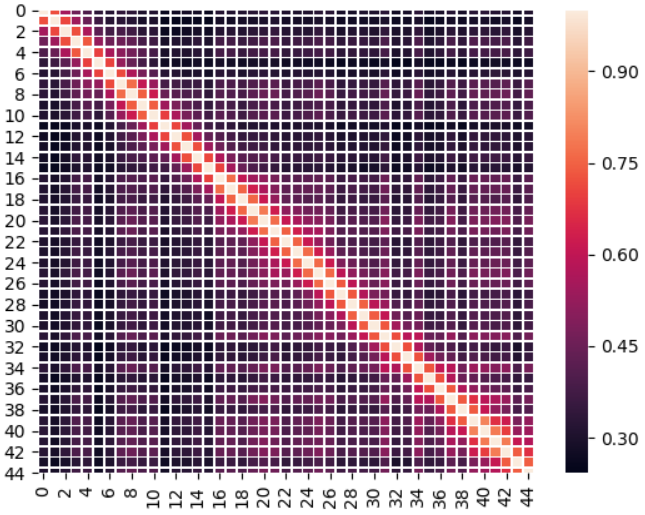


Figure 10: *Correlation matrix for correlated noisy data.*

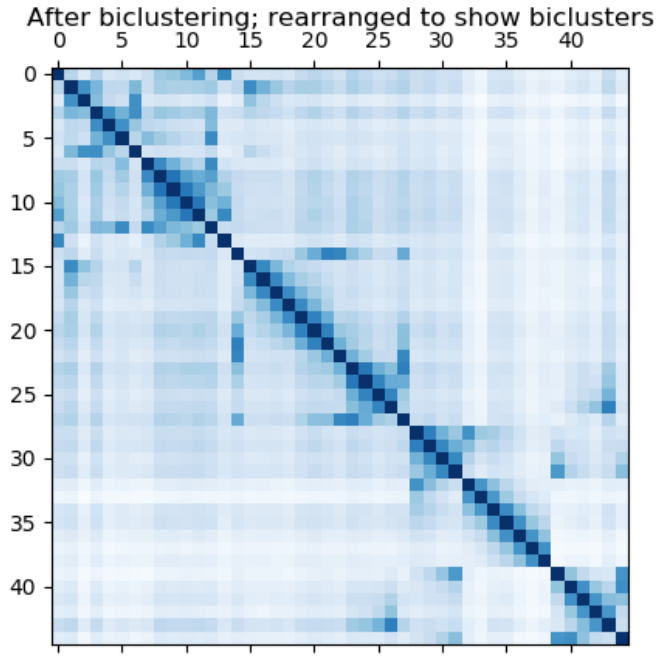


Figure 11: *Biclustered correlation matrix for correlated noisy data.*

b) The results of the accuracy, probabilities of each label and logistic loss can be seen below, when using a scale factor of 5.0 for the correlated noise. The logistic multinomial regression now performs worse with a target accuracy of 0.89 and a logistic loss of 0.67. Since correlation is added to the noise it will be harder for the algorithm to identify the correlation between the data points due to the different phases. Proportionally the real correlation becomes smaller when adding correlated noise which makes the certainty in the predicted labels lower, having a negative impact on the loss function. Additionally, the classifier might be doing wrong prediction on the labels and classify according to the correlated noisy data, instead of the correlations due to phases, and thus the accuracy decreases, which also have a negative impact on the loss function.

```
Predicted labels:  [1 3 1 1 3 1 3 3 3 3 3 3 3 3 3]
True labels:  [1 3 2 2 3 1 1 3 2 3 3 3 3 1 3]
   [9.99111000e-01 2.79847151e-04 6.09152712e-04
8.99174270e-02 5.43623299e-02 8.55720243e-01
9.52717162e-01 3.32127532e-02 1.40700847e-02
8.01075701e-01 1.21502354e-01 7.74219447e-02
1.05651180e-02 3.00175366e-01 6.89259516e-01
8.25166034e-01 1.16302809e-02 1.63203686e-01
3.61060264e-01 1.77722670e-02 6.21167469e-01
3.63711582e-04 2.90281965e-03 9.96733469e-01
2.40989646e-01 3.25441779e-01 4.33568574e-01
6.44164813e-04 6.42106989e-02 9.35145136e-01
1.83144973e-03 1.15803285e-01 8.82365266e-01
```

```
5.18017334e-04 3.50690220e-03 9.95975080e-01
6.95327083e-04 1.38493068e-01 8.60811605e-01
3.38597995e-01 5.46799254e-02 6.06722079e-01
7.03492234e-03 1.56379304e-01 8.36585774e-01

Target accuracy:  0.8888888888888888
Logistic loss:  0.6657126353688566
```

c) To be able to compare with the non-correlated noisy data I set the scale parameter to 20 and get the following results. Due to the reasons mentioned in b) in addition to the reasons in 1.c) for the non-correlated noise, this adds up and affects the performance negatively, both in the loss function and in the accuracy, which is now 1.39 and 0.84 respectively.

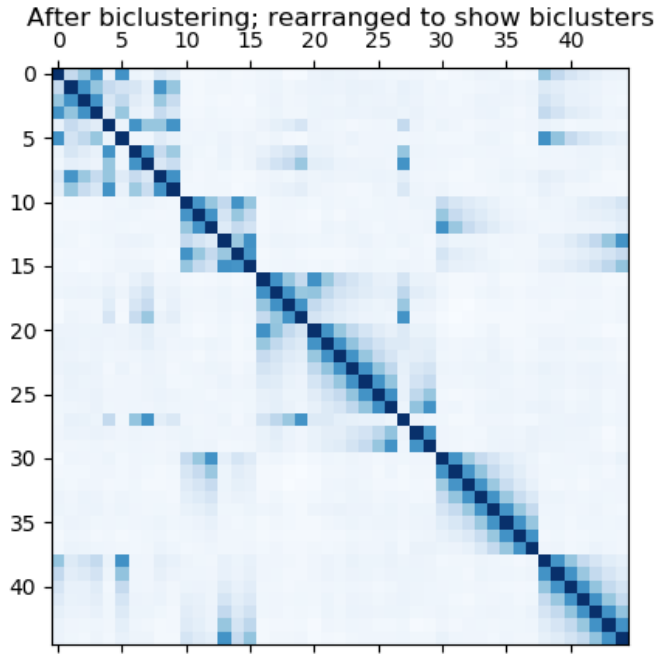


Figure 12: *Biclustered correlation matrix for correlated noisy data with a scale parameter of 20.*

```
   Predicted labels:  [1 1 1 1 1 3 1 3 1 3 3 3 3 3 3]
True labels:  [1 3 2 2 3 1 1 3 2 3 3 3 3 1 3]
Probability for each label:
   [9.99415238e-01 1.27181492e-04 4.57580539e-04
9.66495273e-01 2.54157755e-03 3.09631495e-02
9.31910798e-01 6.59103664e-03 6.14981656e-02
8.00171827e-01 2.06339737e-02 1.79194199e-01
5.28791123e-01 3.27559292e-02 4.38452948e-01
4.23838299e-01 1.59260236e-02 5.60235678e-01
4.88809640e-01 3.73051454e-02 4.73885215e-01
```

```
3.54451055e-01 3.32867092e-02 6.12262236e-01
5.44493897e-01 6.17545481e-02 3.93751555e-01
1.67252495e-01 3.81916746e-02 7.94555830e-01
2.05492305e-01 1.49833945e-02 7.79524301e-01
2.28106687e-01 1.59606092e-02 7.55932704e-01
7.03598610e-02 7.36122885e-03 9.22278910e-01
1.95765331e-01 4.35049786e-02 7.60729691e-01
2.40504047e-01 5.16652990e-02 7.07830654e-01

Target accuracy:  0.8444444444444444
Logistic loss:  1.391227790914845
```

d) The algorithm is already in my opinion performing badly at a scale of 5, but it is definitely performing bad at a scale parameter of 10. Results for accuracy and loss can be seen below. Because of the correlation in the noise the classifier is more sensitive to the data being scaled. When making the noise exponentially correlated, a correlation not reflecting the real system is of course added to the data, but the noise also becomes larger. On one run I for example get my largest value in the noise vector to be 4.74, which is already large noise without any scale parameter. The same value at a run without correlation is 0.47. Given this, the algorithm becomes more sensitive to the scale parameter as the noise is not only correlated but also larger. Scaled with 5 the largest noise is 23.7 and scaled with 10 it is 47.4. Following the discussion in 1.d) for the non-correlated data, these values of noise are very high considering a real world system.

```
Results:

scale = 10.0
Target accuracy:  0.8666666666666667
Logistic loss:  1.0299972415620213
```

2. c) Correlations related to geographic spread could occur due to for example local weather. For instance wind from a certain direction could create a higher or lower demand and thus driving a geographical correlation. The wind could even have more impact if the houses are placed similarly in a neighbourhood, i.e. if the houses are facing the wind and thus creating an even higher (cold country) or even lower demand (warm country). There could also be correlation depending on if customers in one area are industrial customers or businesses compared to another area with more residential customers, since they would have a very different consumer profiles. Additionally, another driver of correlations could be if the customers in different areas receive different electricity prices and adjusts their electricity consumption accordingly.

2. d) I would run the classification in two iterations. If I expect the geographical correlation to have a larger impact I would first cluster according to that. At the second iteration I would cluster according to phase within each geographical cluster. As another approach I would consider to filter based on the geographical correlations and then do the clustering on the filtered data.