Estimate Missing Values with Various Auto Encoders

Xueqin Cui

Research School of Computer Science, Australian National University
u6025011@anu.edu.au

Abstract. In this paper, I mainly discuss different methods we could take to deal with the missing values in the dataset. The simplest way is to remove all the missing values; however, this sometimes could cause the loss of a significant amount of data. Another way is to use some method to estimate the missing values and use the new dataset to train the model. Considering that the missing value is a form of the noise and the auto-associative neural network (auto-encoder) could be used to remove the noise, I trained some auto-encoder to preprocess the data. To study the effects of various auto-encoders, I did experiments on three types of auto-encoders: the standard feedforward network as an auto-encoder, the auto-encoder with shared weights and the variational auto-encoder. After the auto-encoder, a neural network classifier is trained, and the accuracy is used to evaluate the performance. I compared the results generated by the different auto-encoders, and the accuracy was over 80%. Though the results generated by the auto-encoders were not as good as the ones without it, from the comparison with a paper that used the same dataset, we already achieved a decent result by the classifier.

Keywords: Missing Values · Neural Network · Auto Encoder

1 Introduction

How to deal with the missing values in the data set is always a serious topic in the area of machine learning. The effectiveness of the machine learning algorithm not only depends on the model you chose but also relies a lot on the data. However, the quality of the data sets is not always as we expect. For example, according to the widely used machine learning data set source: UCI Machine Learning Repository [1], there are quite a few data sets with missing values, involving both numerical and categorical data. Therefore, motivated by what I experienced when browsing the repository to find a fully complete data set, I decided to experiment different ways to deal with the missing values.

To process the data with the missing values, the most straightforward way is to remove the data instances with the missing values. This method is easy to implement but may lose quite a lot data if the missing values spread over the whole data set. Instead of discarding the data with missing values, we could consider to fill in the missing values with some way of estimation, for example, by the auto-associative neural network. The auto-associative neural network (a.k.a auto-encoder) is well known for its application in image compression since it could help to remove the noise in the image. Since the missing value could be viewed as a particular kind of noise, theoretically, by applying the auto-encoder before the training, I can fill in the missing values from the compressed information I extracted from the data.

The data set I picked up from the repository is the Congressional Voting Records Data Set [2]. The dataset includes the 1984 United State Congressional Voting Records with 16 key votes. I am trying to solve the classification problem: whether a voter is Democrat or Republican based on the sixteen yes-or-no voting choices. This dataset is a perfect match for my experiments: the number of attributes and number of instances meets the minimum requirement; missing values exist for almost all attributes; the dataset is easy to interpret and process (all boolean values including the class label); quite a few papers cited this data set which makes comparison easier.

2 Method and Implementation

The following flow-chart shows the pipeline of my implementation of the system to solve the classification problem. The system mainly consists of four steps: data pre-processing, auto-encoder, classifier, evaluation, which is also shown in Figure 1. I will discuss each step in the following sections.

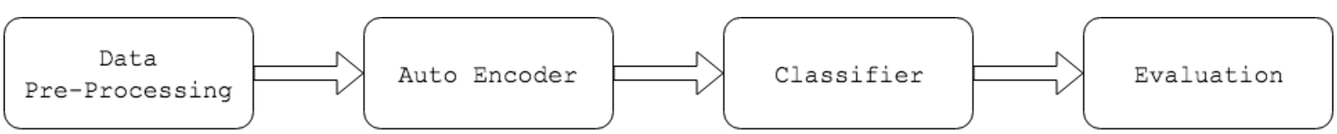


Fig. 1. Flowchart of the whole process of the classification problem.

2.1 Data pre-processing

The Congressional Voting Records Data Set [2] has one class label and sixteen attributes, all binary string variables. The data pre-processing is pretty straightforward; I convert all the binary string variables to boolean values, integers 0 or 1. For example, for the class label, it will either be 'democrat' or 'republican', I define 'democrat' as 0 and 'republican' as 1 and thus convert the string value to a binary variable. Similarly, for the sixteen attributes, they all represent some yes or no votes where obviously 'y' representing yes and 'n' representing no. Following the usual convention of True/False boolean values, I set y' as 1 and 'n' as 0.

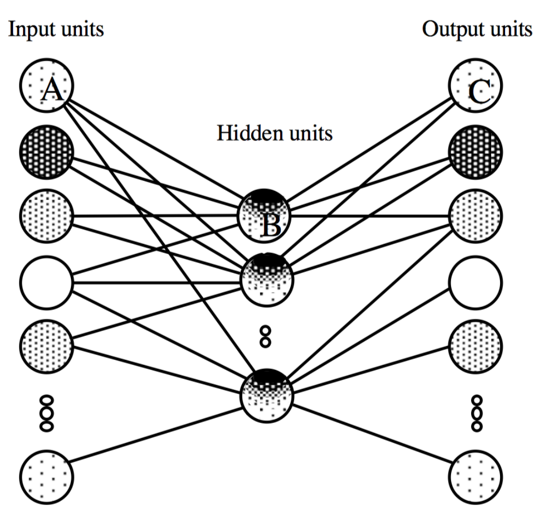
Missing values

However, during the data processing, I noticed that there existed some '?' as the yes/no binary attributes and almost all attributes would have some question marks. As mentioned in the Introduction section, how to deal with the missing values is the primary purpose of this paper. I tried different ways. The first method is to remove all data instances with any missing value. After the elimination of all data instances with the question mark, only 232 voting recording out of the total 435 records remained. About half data will be removed which is not the desired way to do the following training task. Thus I abandoned this method very soon and tried to fill up the missing values by some self-definition. For the continuous values, we will usually use the mean value to make up the blanks, but for categorical data, it is often not a right way to process the data because the value of the category number does not have mathematical meaning. However, since I only have two categories for the attributes, using 0.5 to present the missing value may be a suitable way since it has the same distance to yes and no and won't import any bias to either side.

2.2 Auto-Encoder

As described in the Introduction section, I trained some auto-associative neural network to remove the noise brought from the missing value. According to Gedeon, Catalan and Jin [3], an auto-encoder is a neural network with the same number of input units and output units and there is only one hidden layer in the auto-encoder with the number of neurones smaller than that of the input/output units in order to compress the information (see Figure 2). For the first part of the network (from input layer A to the hidden layer B), it is called the encoder and the second part of the neural network is called the decoder (from hidden layer B to the output layer C). For the auto-encoder that I am using, the number of the input and output units is 16, and the number of neurones in the hidden layer is 4. I choose the back-propagation as the , and the mean squared error as the loss function since multi-target not supported in the cross-entropy loss function and the learning rate is set to be 0.01. Moreover, since the attributes can only be 0 or 1, I used sigmoid function as the activation function to make the output fall in the range (0,1).

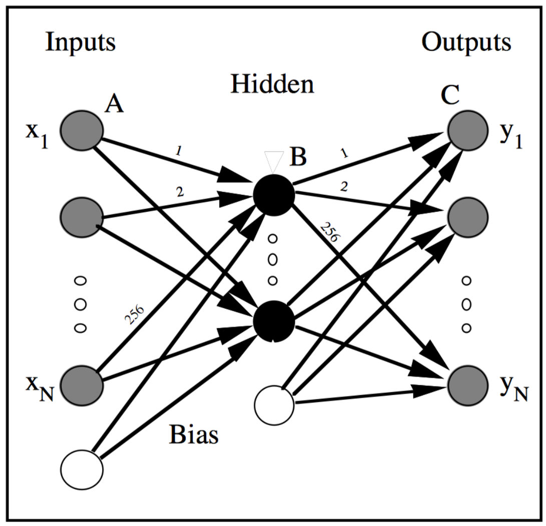
Fig. 2. Auto-associative network topology [3].



Shared weights

Gedeon, Catalan and Jin also proposed some way to improve the auto-associative neural network which is to share the weights between the encoder and the decoder [3]. For example, Figure 3 shows an auto-encoder with the input/output size as 256. For all the weights in the decoder (B to C), we map them with some weight in the encoder (A to B). All matched pairs of weights are labelled with the same number from 1 to 256 in the figure. Gedeon, Catalan and Jin also mentioned that the most straightforward implementation of this sharing weights network is to update the weights after the optimizer with the average of the weights from A to B and from B to C, but this way could help to prevent the two sets of weights diverge from each other, though it may be computationally expensive [3].

Fig. 3. Auto-associative network with shared weights topology [4].



Variational Auto-Encoder (VAE)

To better imitate the pattern of the data, some generative model used in Deep Learning is employed. Variational Auto-Encoder (VAE) is what Kingma and Welling introduced in [5] where a stochastic variational inference and learning algorithm used to decode the compressed information. In a standard auto-encoder, we used the latent variables generated from the encoder to generate the output; however, there is some limitation that we need rely on the input to decode the information but not to generate the output freely. VAE could help to solve this problem.

Figure 4 shows the difference between a standard auto-encoder and a variational auto-encoder. Different from auto-encoder only encoding one latent vector, VAE will generate two vectors: mean vector and standard deviation vector and these two vectors are used together to generate the latent vector to be used by the decoder later [6]. With VAE, some randomness has been imported, and thus we could generate the output not only on the encoded information. The similarity between the original input the new output could be used to measure the accuracy of the variational auto-encoder and the loss could be measured by the KL divergence [5]:

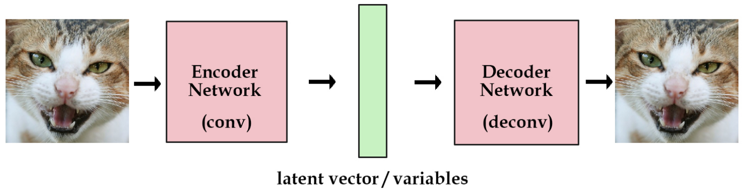
$$DKL(P||Q) = \int_{\infty}^{\infty} p(x) log \frac{p(x)}{q(x)} dx \tag{1}$$

My implementation of VAE is based on the GitHub codes provided in [7]. The input vector was first compressed from 16 to 10, and then two 4-dimension vectors (mean and deviation) were reparameterized to generate the sampled latent vector, which will be used by the decoder.

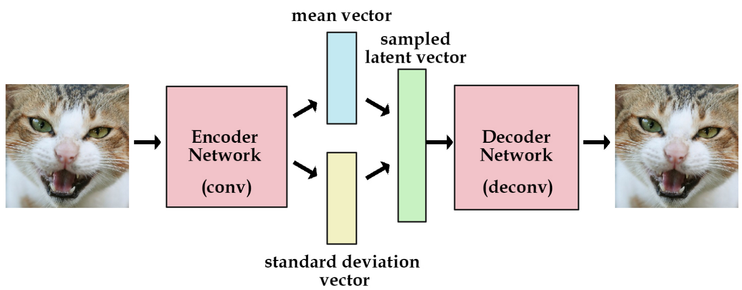
After the auto-encoder applied on the processed data to remove the noise from the missing value, I stored the outputs from the auto-encoder as the new input attributes, together with the target labels from the original preprocessed data set, they formed the new training data which will be used in the next step, the classifier, the key step of this classification problem.

2.3 Classifier

As the primary step to solving the classification problem, I trained a classifier in the form of the neural network. Considering the size of the problem is not that large, the network I am using is quite simple. The input layer has 16 neurones which correspond to the 16 input attributes; there is only one hidden layer with 50 neurones to interpret the information from the input layer and distribute to the output layer; the output layer has two neurones since it



(a) Auto Encoder



(b) Variational Auto Encoder

Fig. 4. Comparison between an Auto Encoder and a Variational Auto Encoder [6].

is a binary classification problem. For the choice of the loss function and the optimizer, I am using the cross-entropy loss and the back-propagation algorithm, and the learning rate is set to be 0.01.

Meanwhile, the model is trained in batches. The two classes in the data set do not share the same data size. Among all 435 data instances, 267 instances are labelled as 'democrat's while only 168 instances are labelled as 'republican's. Considering the difference between the number of two classes, if I train all the data in one batch, the result will be skewed to the main class which is 'democrat'. To avoid this, I need to train the model in batches. Thus, I make use of the data loader when I was retrieving the data, and the data size is set to be 5.

2.4 Evaluation

After training the classifier, I run the model with the input attributes and get the estimation of the target labels. Next step is to compare the estimate with the real labels. To better analyse the results, I implemented several evaluation methods.

Confusion Matrix

Considering the output of the data set is only binary categorical value, confusion matrix [8] may be a good choice since it is easy to implement and also clear to interpret. Figure 5 shows the contents of a confusion matrix. The confusion matrix is a 2*2 matrix where the column will represent the actual class and row will represent the estimated class. For example, for the dataset I am using, the number in the first row and the first column means the number of data instances belonging to 'democrat' and classified as 'democrat' as well. For each class of the target, I could easily see the number of the data instances correctly classified as this category, and the data instances wrongly categorised to the other group.

Accuracy

Since the confusion matrix is still not good enough to make the comparison between different neural network results, I also use the accuracy as a measurement of the classifier. I just merely calculated the percentage of the correctly categorised data instances among the whole data set. However, there is some shortage with this evaluation method; I can only see the accuracy of the entire data set with two classes together but not the accuracy of the specific category. If I want to look at the results within some group, I may need to refer to the confusion matrix for details.

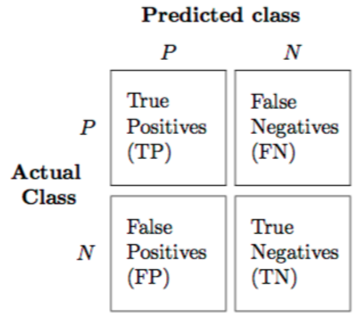


Fig. 5. Representation of a confusion matrix [9].

Losses as the learning curve

To better study the effects of different auto-encoders on the classifier training, I used the losses curve as the indication of the learning process. For each auto-encoder, I stored the loss in each iteration in the learning process of the classifier (not the auto-encoder) in a text file. After the experiments done for all auto-encoders, I read each text file with the loss statistics and plot them together in a figure. A detailed example will be shown in the discussion presented later.

To better test the trained model, I split the whole dataset into the training set and testing set. I first randomly shuffle the data and then keep the first 80% data as the training set and the remaining 20% as the testing set. For the first two evaluation methods (confusion matrix and accuracy), the results of both the training data set and the testing data set are computed.

3 Results and Discussion

As discussed in the Method section, we experimented four ways to remove the noise brought by the missing values: directly train the data without any auto-encoder, estimate the missing values by the auto-encoder, estimate the missing values by the auto-encoder with shared weights, estimate the missing values by a variational auto-encoder. For each method, I repeated the classification task for ten times and recorded the accuracy each time for both training set and testing set. For all experiments, the random seed is set as 12345 at the beginning to repeat the experiments to the most considerable extent.

Following are the results of the ten experiments using the four methods.

Without Auto-Encoder

Table 1 shows the accuracy of the ten experiments conducted with the neural network without any auto-encoder previously applied. From the table, we can see that all the accuracy rates are above 90%. No matter the training set or the testing set, the average accuracy of the ten trails is around 95%. This indicates that the model trained by our neural network is pretty useful and the way that we fill in the missing values (replacing them by 0.5) makes positive effects. Though the network is simple with only one hidden layer, the results are entirely satisfactory.

Table 1. Accuracy of ten experiments conducted without any auto-encoder applied.

Trail	1	2	3	4	5	6	7	8	9	10	Average
Training	94.87	94.63	95.52	95.14	94.88	95.34	95.35	94.57	94.69	94.27	94.93
Testing	94.05	97.00	94.87	96.47	99.03	94.57	93.41	95.29	94.81	95.35	95.49

Feedforward Auto-Encoder

Table 2 includes the results of the ten experiments with the auto-encoder to remove the noise from the missing values. All accuracy rates are above 80%, which is still good, and both the average accuracy are within 85-88%. However, if we compare the results with the ones without auto-encoder applied, the accuracy is lower, which it is expected to be opposite. One possible reason for this unusual phenomenon may be that there is not much noise brought by the missing values, so after the auto-encoder compressing the data, some information is lost which finally affect the accuracy. Another potential reason could be that currently, the new input data after auto-encoder applied are not binary boolean values but real numbers between 0 and 1. To strictly align with the meaning of the data set, we should make them either 0 or 1. This could be extended as one of our future work to see if the data type will change the results.

Also, we can observe that the average accuracy of the training set is greater (1.38%) than the one obtained with the testing set. This may be because I only applied the auto-encoder on the training set and trained the classifier with the training data, but the testing set is the same as the data we got from the original dataset. We can see that auto-encoder does make some effect on the data and lose some information initially inside it and thus the classifier trained on the modified data has a worse performance with the unchanged data.

Table 2. Accuracy of ten experiments conducted with the feedforward auto-encoder.

Trail	1	2	3	4	5	6	7	8	9	10	Average
Training	86.26	85.96	89.80	85.92	89.91	86.74	87.71	86.69	84.86	87.61	87.15
Testing	80.65	87.10	84.78	94.25	82.95	80.82	87.06	85.37	88.24	86.46	85.77

Auto-Encoder with shared weights

We also implemented the auto-encoder with shared weights between encoder and decoder. Similar to the standard feedforward auto-encoder, the results are good but not that great as the ones without auto-encoders applied. All accuracy rates are above 80%, and the average accuracy is within 84-98%, and the average accuracy of the training set is better than the testing set by about 2.5%. We already discussed the same result in the previous section.

Table 3. Accuracy of ten experiments conducted under the auto-encoder with shared weights.

Trail	1	2	3	4	5	6	7	8	9	10	Average
Training	89.11	85.71	88.67	85.14	81.25	90.14	87.43	90.86	91.45	84.42	87.42
Testing	87.21	81.52	82.93	85.88	83.13	85.56	84.95	85.88	84.38	87.80	84.92

Variational Auto-Encoder

For the results generated by the variational auto-encoder, the accuracies are all above 80% which is good but not as good as the one without any auto-encoder applied. Both average accuracies are around 88%, and not much difference between them (only 0.74%) which means the classifier trained performs almost equally on both datasets. There is onemore thing to note that for the previous two auto-encoders, the average accuracy of the training set is higher than the testing one by one or two percent, however, for VAE, there is no such difference anymore, and the testing accuracy is a little better than the training one. Considering that the testing set is from the original data, VAE captures more pattern in the data set than the other two auto-encoders, so the performance of the classifier on the testing data is better.

Table 4. Accuracy of ten experiments conducted with the variational auto-encoder.

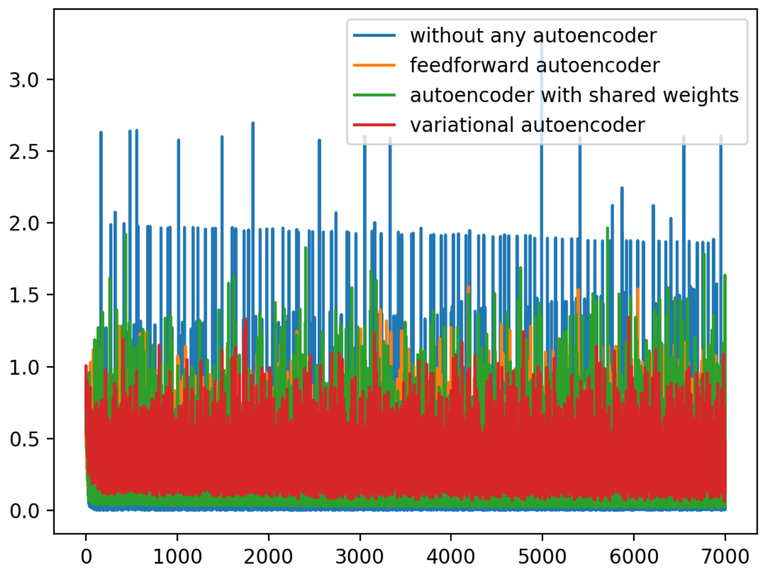
Trail	1	2	3	4	5	6	7	8	9	10	Average
Training	86.47	87.11	86.25	86.48	90.00	85.38	88.13	89.50	89.39	89.24	87.80
Testing	82.11	94.87	91.86	88.75	91.58	87.10	89.80	86.96	84.42	87.91	88.54

Comparison between various Auto-Encoders

As mentioned in the previous section of the paper, I recorded all losses during the training process of the classifier to study the effects on the learning process by different auto-encoders. Figure 6 is the learning curves generated by the four methods plotted together. From the plots, we can see there is no significant difference between the three auto-encoders. Most curves are overlapping with each other which means the losses are quite similar during the training. However, the auto-encoder with shared weights (green) has fewer losses compared with the variational auto-encoder (red). The green curves almost touch the bottom (0.0), but there is still some gap between the bottom and red curves. This may be because that the VAE imports some random noise before the training and makes it harder to reduce the loss to zero.

Meanwhile, compared with the auto-encoders, the learning curves of the classifier without any auto-encoder are much greater sometimes. The blue curves, which representing no auto-encoder used, are much larger than the other three for some time which means that the losses are more significant. Though according to the accuracy displayed above, the auto-encoders may not help with improving the classification (actually even worse), from the plots, we can see that the auto-encoders still help to reduce the losses. Losses with auto-encoder applied are effectively decreased maybe because that the pre-training already learn some pattern of the data.

Fig. 6. Learning curves (losses) of different auto-encoders.



Comparison with another paper

Bonet and Geffner [10] also used the same data set to test their classification models, and they also used the accuracy to evaluate their models. The accuracy of the voting data set classification is mostly around 95% [10]. This result is pretty close to the rates that we got if we trained the data without any auto-encoder applied and is better than the auto-encoder experiments we conducted. This shows that our model has excellent classification performance without using the auto-encoder. However, the auto-encoder applied before the training does some adverse effect on the training data. We proposed some reasons in the previous sections, but the real cause is unknown. More detailed research on the pattern of the dataset and how to make the auto-encoder more applies to this specific context should be conducted, and this could be the focus of my future work.

4 Conclusion and Future Work

To discover the right way to deal with the missing values in the dataset, I tried several methods with the selected voting data set. The simplest way is to remove all the data instances with any missing value; however, this may remove a significant portion of data in the whole data set, so it is not encouraged. Thus we need to develop some way to fill in the blanks with the missing values. One standard way is to use the mean value of the variable, and this is more suitable to the continuous variable, and for the numbered categorical data, this does not make much sense. However, if there are only two classes, we could use 0.5 to replace the missing values which may not bring any bias to either side of the boolean values. My experiment results showed that this way of data pre-processing would result in the accuracy over 90% with average around 95%. Another kind of methods I chose is to use the auto-encoder to estimate the missing values. I did experiments on the standard feedforward neural network, the network with shared weights and the variational auto-encoder. Though we were expecting better results with auto-encoders, the accuracy is not as good as we expected. The accuracy is over 80% with the average in the range 84-89%. Though the results themselves are quite good but compared with the results without auto-encoder applied, it is worse. However, the variational auto-encoder performs better on the testing set which prove that it could better capture the information in the original dataset.

To better determine the missing values in the dataset, my future work may include two aspects. The first one is to research and improve the auto-encoder on the current data set. The dataset should be investigated to see if there is some specific pattern of the data which we make it lost when applying the auto-encoder. Second, as discussed, the problem may be due to that we keep the real numbers after the training the auto-encoder, but the actual target should be a binary integer. We may try to modify the auto-associative network to have the integer output and see if the results will be better. Third, we could also try to remove the effect that the self-defined missing values brought to the auto-encoder. For example, we could try to use the data instances without any missing value to training the auto-encoder then estimate the missing values to see if the performance will be better. The other aspect of the future work will be extending the scope of the data type. Currently, I only apply the auto-encoder on the binary categorical data. In the future, I could extend it to the multiple categorical data or even the continuous numerical values.

References

1. University of California, Irvine, School of Information and Computer Sciences, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

2. J. Schlimmer, "UCI Machine Learning Repository: Congressional Voting Records Data Set," 2018. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records

3. T. Gedeon, J. Catalan, and J. Jin, "Image compression using shared weights and bidirectional networks," in *Proceedings 2nd International ICSC Symposium on Soft Computing (SOCO'97)*, pp. 374-381.

4. T. Gedeon, "Stochastic bidirectional training," in *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, vol. 2. IEEE, 1998, pp. 1968-1971.

5. D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

6. K. Frans. "Variational Autoencoders Explained", 2018. [Online]. Available: http://kvfrans.com/variational-autoencoders-explained/. [Accessed: 30-May-2018].

7. Basic VAE Example, GitHub, 2018. [Online]. Available: https://github.com/pytorch/examples/tree/master/vae. [Accessed: 30-May-2018].

8. R. Kohavi and F. Provost, "Confusion matrix," *Machine learning*, vol. 30, no. 2-3, pp. 271-274, 1998.

9. "Confusion matrix - mlxtend", rasbt.github.io, 2018. [Online]. Available: https://rasbt.github.io/mlxtend/user_guide/evaluate/confusion_matrix/. [Accessed: 30-May-2018].

10. B. Bonet and H. Geffner, "Learning sorting and decision trees with pomdps." in *ICML*. Citeseer, 1998, pp. 73-81.