CS420 Homework3

* Name: Zhixin Ling Student ID:516021910495

1 SVM vs. Neural Networks

1.1 Classification on A datasets

In this part, I select five datasets to do classification using SVM and neural networks. The information of the datasets are illustrated in table **??**. The first three datasets are actually the same but with training and test set differently split. We scale all data to interval [0.0,1.0] and all class numbers start from 0. For neural networks, both MLP and CNN are tried. We elaborate the results below:

1. First we try SVM on these datasets. We conduct experiments with different hyper-parameters settings: different kernel types and kernel parameters, different C's and different preprocessing tricks. Some representative results are presented in table 2. From left to right, the columns stand for dataset name, training accuracy, test accuracy, f1 score(a balance of recall and precision), number of support vectors, whether to do l2 normalization and PCA, number of reduced dimensions, ratio of remained variance Var_r, the error tolerant factor C, a parameter in kernel γ and the kernel applied. X_v is variance of training set in each dimension. And we have observations below:

   (a) It is found that use PCA or l2 normalization for preprocessing can help improve the test accuracy and the improvement is even more significant when combining them together. From the first two experiments, it can be concluded that the preprocessing can reduce noices since with preprocessing applied, the acc_{tr} is reduced while acc_{te} is improved.

   (b) The results of preprocessing is similar when C is different. With C decreased, number of support vectors increase because more points are required to reduce fitting error (on training set).

   (c) From results of the first six experiments, there comes an interesting phenomena: the closer the number of support vectors to 659, the better the test accuracy will be.

   (d) However, when the kernel change into Gaussian kernel(experiment 5 and 6), the preprocessing does not help any more, but our conclusion about the relation of acc_{te} and sv still holds. Also, it is found Gaussian kernel does not work better than linear kernel.

   (e) When using sigmoid kernel, the preprocessing helps again. But the kernel still works worse than linear kernel.

   (f) Compared to a1a dataset, a7a has training and test set evenly split. The number of support vectors increase a lot, more than 5000. But the test accuracy does not improve that much. Preprocessing does not help a lot any more since an enlarged dataset can always reduce overfitting. Similarly, other kernels doesn't work well with the dataset as linear kernel does. item When classifying a8a, which put more data into training set, still, the preprocessing does not help and Gaussian kernel, not linear kernel, performs the best. Increasing of training set size ease overfitting a lot. The core idea of Gaussian kernel is to map the data into a space of infinite dimensions and this can easily lead to overfitting(check experiment 7 and 8). However, if a large number of training samples are available, diversities of the samples can be ensured and overfitting can be controlled.

Table 1: Information of datasets

Index	name	features(m)	total samples(S)	test samples(S_te)	training set size(S_tr)
1	a1a	123	32561	30956	1605
2	a7a	123	32561	16461	16100
3	a8a	123	32561	9865	22696
4	pendigits	16	10992	3498	7494
5	usps	256	9298	2007	7291
6	mnist	784	65000	10000	55000
7	cifar10	3072	60000	10000	50000

Table 2: SVM results on A datasets

index	ds	acc_{tr}	acc_{te}	$f1$	sv	prep	dims	Var_r	C	γ	kernel
1	a1a	0.8623	0.8382	0.8347	588	×	123	1.0	1.0	-	$< x, x' >$
2	a1a	0.8461	0.8432*	0.8360	659	✓	61	0.99	1.0	-	$< x, x' >$
3	a1a	0.8573	0.8403	0.8362	602	×	123	1.0	0.5	-	$< x, x' >$
4	a1a	0.8436	0.8417	0.8323	697	✓	61	0.99	0.5	-	$< x, x' >$
5	a1a	0.8660	0.8373	0.8340	575	×	123	1.0	3.0	-	$< x, x' >$
6	a1a	0.8505	0.8422	0.8365	618	✓	61	0.99	3.0	-	$< x, x' >$
7	a1a	0.8748	0.8421	0.8343	716	×	123	1.0	1.0	$1.0/(m * X_v)$	$exp(-\gamma\|\|x - x'\|\|^2)$
8	a1a	0.8972	0.8406	0.8331	771	✓	61	0.99	1.0	$1.0/(m * X_v)$	$exp(-\gamma\|\|x - x'\|\|^2)$
9	a1a	0.8180	0.8310	0.8209	652	×	123	1.0	1.0	$1.0/(m * X_v)$	$tanh(\gamma < x, x' >)$
10	a1a	0.8118	0.8366	0.8317	626	✓	61	0.99	1.0	$1.0/(m * X_v)$	$tanh(\gamma < x, x' >)$
11	a1a	0.8168	0.8267	0.8129	704	×	123	1.0	1.0	$1.0/(m * X_v)$	$tanh(\gamma < x, x' > +0.5)$
12	a7a	0.8488	0.8483	0.8435	5762	×	123	1.0	1.0	-	$< x, x' >$
13	a7a	0.8457	0.8482	0.8425	5884	✓	61	0.99	1.0	-	$< x, x' >$
14	a7a	0.8464	0.8490*	0.8440	5817	✓	61	0.99	1.0	-	$< x, x' >$
15	a8a	0.8469	0.8516	0.8468	8138	×	123	1.0	1.0	-	$< x, x' >$
16	a8a	0.8448	0.8509	0.8455	8254	✓	61	0.99	1.0	-	$< x, x' >$
17	a8a	0.8625	0.8530*	0.8480	8402	×	123	1.0	1.0	$1.0/(m * X_v)$	$exp(-\gamma\|\|x - x'\|\|^2)$
18	pendigits	0.9966	0.9817	0.9817	1289	×	16	1.0	1.0	$1.0/(m * X_v)$	$exp(-\gamma\|\|x - x'\|\|^2)$
19	pendigits	0.9967	0.9837*	0.9837	1253	✓	13	0.99	1.0	$1.0/(m * X_v)$	$exp(-\gamma\|\|x - x'\|\|^2)$
20	pendigits	0.9135	0.8605	0.8562	4941	✓	13	0.99	1.0	$1.0/m$	$exp(-\gamma\|\|x - x'\|\|^2)$
21	usps	0.9982	0.9547*	0.9547	1581	×	256	1.0	1.0	$1.0/(m * X_v)$	$exp(-\gamma\|\|x - x'\|\|^2)$

(g) For pendigits dataset, Gaussian kernel performs the best while others result test accuracies at least 0.05 smaller(not presented, though). And it is the first time that γ parameter of Gaussian kernel play an important role for the result. γ controls influence of a single sample to the resulted hyper-plane. If γ is large, the hyper-plane will be distorted by samples on the edge, and if γ is small, say $\gamma = 0.0$, all samples will gather to one point and thus $\gamma = 0.0$ makes no sense. item As for usps dataset, it has the most features. The best result is presented on the experiment 21.

(h) No matter which dataset is used, $C = 1.0$ always results the best test accuracy. It achieves a good balance between variance and bias.

2. The we use MLP for classification. Some representative results are presented in table 3. We set the learning rate a constant $l_r = 0.0004$ and Adam optimizer is used for parameter update. Among the headers of the table, activation means activation function used for output of an intermediate layer, l2 means l2 regularization factor for weights of the MLP and layers means the number of neurons of each layer within the MLP. The batch size is 128. I explore how the hyper-parameters affect the classification result. We have observations below:

   (a) From experiment 1-6, MLP has a great power of expression and overfitting can easily happen, especially when RELU is used for neuron activation. The reason could be that

Table 3: MLP results on A datasets

index	ds	acc_{tr}	acc_{te}	$f1$	prep	dims	Var_r	l2	layers	activation
1	a1a	0.9483	0.8305	0.8285	$\times$	123	1.0	0.0001	(100)	$max(0,x)$
2	a1a	0.8903	0.8417	0.8383	✓	61	0.99	0.0001	(100)	$max(0,x)$
3	a1a	0.9913	0.8110	0.8105	$\times$	123	1.0	0.0001	(128, 96)	$max(0,x)$
4	a1a	0.9501	0.8301	0.8275	$\times$	123	1.0	0.001	(100)	$max(0,x)$
5	a1a	0.9507	0.8320	0.8288	$\times$	123	1.0	0.00001	(100)	$max(0,x)$
6	a1a	0.8604	0.8413	0.8378	$\times$	123	1.0	0.0001	(100)	$1/(1+e^{-x})$
7	a1a	0.8474	0.8424*	0.8358	✓	61	0.99	0.0001	(100)	$1/(1+e^{-x})$
8	a1a	0.8629	0.8400	0.8367	$\times$	123	1.0	0.0001	(128,96)	$1/(1+e^{-x})$
9	a1a	0.8567	0.8423	0.8399	✓	61	0.99	0.0001	(128,96)	$1/(1+e^{-x})$
10	a1a	0.8748	0.8383	0.8346	$\times$	123	1.0	0.0001	(100)	$tanh(x)$
11	a1a	0.8586	0.8414	0.8370	✓	61	0.99	0.0001	(100)	$tanh(x)$
12	a1a	0.8530	0.8419	0.8372	✓	61	0.99	0.001	(100)	x
13	a7a	0.9048	0.8354	0.8323	$\times$	123	1.0	0.0001	(100)	$max(0,x)$
14	a7a	0.8780	0.8447	0.8412	✓	61	0.99	0.0001	(100)	$max(0,x)$
15	a7a	0.8469	0.8512*	0.8464	✓	61	0.99	0.0001	(100)	$1/(1+e^{-x})$
16	a8a	0.9016	0.8371	0.8367	$\times$	123	1.0	0.0001	(100)	$max(0,x)$
17	a8a	0.8711	0.8499	0.8483	✓	61	0.99	0.0001	(100)	$max(0,x)$
18	a8a	0.8465	0.8551*	0.8520	✓	61	0.99	0.0001	(100)	x
19	pendigits	0.9924	0.9668	0.9668	✓	13	0.99	0.00001	(100)	$max(0,x)$
20	pendigits	0.9992	0.9731	0.9730	$\times$	16	1.0	0.00001	(128, 96)	$max(0,x)$
21	pendigits	0.9993	0.9783*	0.9783	✓	13	0.99	0.00001	(128, 96)	$max(0,x)$
22	usps	0.9997	0.9452*	0.9453	$\times$	256	1.0	0.0001	(128, 96)	$max(0,x)$
23	usps	0.9999	0.9452*	0.9452	$\times$	256	1.0	0.0001	(100)	$max(0,x)$

RELU can always keep gradient of parameter. And Preprocessing help ease overfitting and thus improves acc_{te} (in experiment2). Preprocessing can help improve acc_{te} in almost every configuration.

(b) From experiment2, we can see deeper structure do not always result better performance, and overfitting can really be a challenge. So, it is unwise to make a network deeper at liberty.

(c) By observing experiment 4, 1 and 5, it is found L2 regularization does not influence the result significantly. And it seems the regularization constrains the performance.

(d) In terms of acc_{te}, different activations do not differ much. Currently, RELU is widely used for activation since it preserves gradient, but our network is simple, so it cannot show its advantage. And, unexpectedly, the best accuracy is achieved without any activation (in experiment 12).

(e) By observing experiments on a7a and a8a, increasing of training data can always help classifier's training. Indeed, we have $acc_{tr} < acc_{te}$ in the best cases.

(f) Comparing results of experiment 19 and 21, it is the first time that we find depth brings improvement.

(g) When classifying usps, it is found preprocessing does not help any more.

3. It is amazing that MLP, a simple network structure that extract relation from adjacent input, can show such performances on image datasets (pendigits and usps). So I'd like to explore how CNN can work compared with MLP. While using sklearn lib for SVM and MLP, I construct CNN myself. Fix number of epoch to 25. And drop neurons in fully-connected layers with a probability of d_r. The learning rate is annealed according to the formula $l_r(e) = 0.01 \times 0.95^e$ while learning rate using MLP is fixed. We also use a batch size of 128. Also, batch normalization is applied to avoid covariate shift. We do not use L2 regularization since the effect is insignificant in MLP experiments. Image structure should be maintained for CNN, so preprocessing using PCA and l2 normalization is not applied. Padding is also applied.

Table 4: CNN results on A pendigits and usps

index	*ds*	acc_{tr}	acc_{te}	$f1$	d_r	C-layers	FC-layers	activation
1	pendigits	0.9975	0.9737	0.9737	0.5	-	(128)	$max(0, x)$
2	pendigits	0.9997	0.9874*	0.9874	0.9	([16, 5], 2, [32, 5], 2)	(128)	$max(0, x)$
3	usps	0.9952	0.9442	0.9442	0.5	-	(128)	$max(0, x)$
4	usps	0.9996	0.9706*	0.9706	0.5	([16, 3], 2, [32, 3], 2)	(512)	$max(0, x)$

Table 5: Summary of A datasets's classification

index	*ds*	SVM acc_{te}	MLP acc_{te}	CNN acc_{te}
1	a1a	0.8432	0.8424	-
2	a7a	0.8490	0.8512	-
3	a8a	0.8530	0.8551	-
4	pendigits	0.9837	0.9783	0.9874
5	usps	0.9547	0.9452	0.9706

C-layers and FC-layers mean convolution layers and fully connected layers. For a member of C-layer's denotation, [a, b] means a b-sized kernels and a scalar c means the pooling layer's kernel size(max pooling).

Some representative results are presented in table 4. To match the baseline with MLP, we conduct experiment 1 on pendigits and experiment 3 on usps. The results are similar. However, when convolution is applied, acc_{te} improves obviously. The best accuracies, 0.9874 and 0.9706, are presented in experiment 2 and 4, greater than MLP's 0.9783 and 0.9452 respectively.

4. In the end, we present a summary in table 5. We cannot tell which is better among SVM and MLP. The key factors we to consider in SVM are nothing but C, kernel functions. And MLP's regularization, L2 or L1, is just like C of SVM. MLP is more flexible and expressive, which also means more hyper-parameters should be considered, such as activation and network structure. CNN is the best when it comes to classify data with space structure, such as images and videos. And more complicated architectures are introduced.

1.2 Classification on dataset B

Our training and searching process is similar to subsection 1.1. We use mnist and cifar 10 datasets for experiments (information of the datasets can be seen in table 1). So we present the result in table 6.

Polynomial kernel performs well on mnist's classification. And, the fewer dimensions reserved, the better the acc_{te} can be. The best acc_{te} reaches 0.9870, while the state-of-art result is 0.9977, achieved by Ciresan et al. in CVPR 2012. They uses a complicated convolution network and apply data augmentation.

The best acc_{te} on cifar-10 is 0.5752 while the state-of-art accuracy is 0.89 by Alex Krizhevsky et al. [1]. They use an CNN architecture of ImageNet, and many methods like local response normalization are applied.

My first impression for SVM on large datasets is that the training take more time. The dimension has to be reduced; otherwise, the training cannot be done. This is because GPU is not used for training while many parallel architectures are available for neural networks. Kernel computation contributes a lot for time complexity. Also, space consumption is also considerable since a kernel matrix need to be pre-computed to make the training easier, making the space complexity $O(n^2)$, where n is the number of training samples. We can perform dimension reduction to ease the situation. However, with some dimensions dropped, some information, maybe useful, is

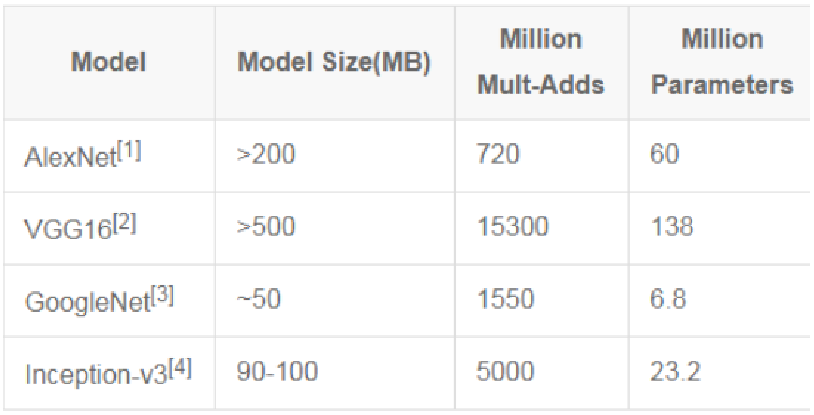
Table 6: Summary of B datasets's classification using SVM

index	ds	acc_{tr}	acc_{te}	$f1$	sv	prep	dims	Var_r	C	γ	kernel
1	mnist	0.9995	0.9858	0.9858	12339	✓	332	0.99	3.0	$1.0/(m * X_v)$	$exp(-\gamma\|\|x - x'\|\|^2)$
2	mnist	0.9997	0.9858	0.9858	14530	✓	332	0.99	3.0	$1.0/(m * X_v)$	$tanh(\gamma < x, x' >)$
3	mnist	0.9996	0.9864	0.9864	13445	✓	161	0.95	3.0	$1.0/(m * X_v)$	$tanh(\gamma < x, x' >)$
4	mnist	0.9994	0.9870*	0.9870	12206	✓	94	0.90	3.0	$1.0/(m * X_v)$	$tanh(\gamma < x, x' >)$
5	cifar10	0.6647	0.5378	0.5357	43635	✓	58	0.80	1.0	$1.0/(m * X_v)$	$exp(-\gamma\|\|x - x'\|\|^2)$
6	cifar10	0.7136	0.5461	0.5444	44026	✓	150	0.90	1.0	$1.0/(m * X_v)$	$exp(-\gamma\|\|x - x'\|\|^2)$
7	cifar10	0.7311	0.5530	0.5512	44324	✓	294	0.95	1.0	$1.0/(m * X_v)$	$exp(-\gamma\|\|x - x'\|\|^2)$
8	cifar10	0.9041	0.5752*	0.5744	44840	✓	781	0.99	3.0	$1.0/(m * X_v)$	$exp(-\gamma\|\|x - x'\|\|^2)$

inevitably lost so there must be a ceiling for SVM, as illustrated in table 6, while neural networks are able to select useful dimensions automatically. Besides, selection of different kernels can have great blindness without special knowledge about the data.

SVM also has advantages. Although I mention time and space complexity above. Training of neural networks can always be more time and space consuming. Some complicated convolution networks can have more than one million parameters, so many methods, in terms of hardware or software, are thought of to accelerate the training. And SVM is more interpretable and also easier to tune parameters because of fewer hyper-parameters. To avoid huge space complexity, neural networks always train data in batches, which possibly brings negative effect for convergence of the training. Besides, as discussed above, SVM's space complexity is determined by number of samples n, while a large number of dimensions m can greatly overwhelm neural networks's space complexity. Therefore, if the big dataset has a great m but relatively smaller n, maybe SVM can be considered if the computation resources are limited.

However, neural networks' performances usually overcome SVM's. It can extract connections of different features efficiently.



Model	Model Size(MB)	Million Mult-Adds	Million Parameters
AlexNet[1]	>200	720	60
VGG16[2]	>500	15300	138
GoogleNet[3]	~50	1550	6.8
Inception-v3[4]	90-100	5000	23.2

Figure 1: Sizes of several famous CNN models

2 Causal discovery algorithms

I use LiNGAM algorithm, proposed by Shohei Shimizu et al. [2], on a diamonds dataset. The dataset comes from Kaggle(https://www.kaggle.com/shivam2503/diamonds), collected by shivam-agrawal. And the descriptions are shown in figure 2.

2.1 Preprocessing

To apply LiNGAM algorithm, the datasets must be continuous values. So all of discrete values are transformed into continuous values. Values of the diamonds dataset we select can easily be continuously evaluated: map a worse discrete value to a small continuous value and a better discrete value to a greater continuous value. After that, we decentralize and whiten the data and get the input $n \times m$ matrix, M. where n is the number of samples and m number of the dimensions.

The diamonds dataset has $n = 53940$ and $m = 10$, satisfying $n \gg m$, which help the algorithm's convergence a lot.

2.2 Algorithm

After preprocessing, I can run the LiNGAM algorithm.

First there are some assumptions of the algorithm. LiNGAM has three important properties: *Linear, Non-Gaussian, Acyclic Model.* The diamonds dataset can be assumed to satisfy these properties. Also, LiNGAM requires disturbances e are independent, implying there are "no unobserved confounders". This assumption limits many actual situations since a real-world problem is always very complicated, and it impractical to collect information in every aspect. The diamonds dataset may have satisfy the assumption to a great extent. For example, disturbance to carat is unlikely correlated to disturbance to color.

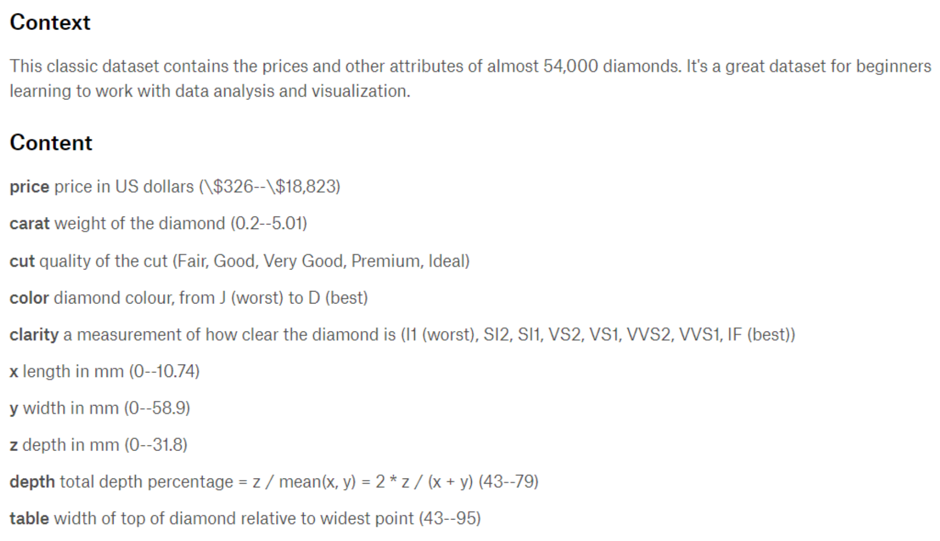
If X is decentralized, then the problem to solve is the function below:

$$X = BX + e \tag{1}$$

where X is a $m \times 1$ matrix, e are disturbances subject to non-Gaussian distributions. Let $A = (I - B)^{-1}$, we have the formula below:

$$X = Ae \tag{2}$$

This is a standard form of ICA model. However, ICA cannot solve orders of e while LiNGAM can. Shimizu et al. briefly summarize the algorithm in two steps: "First, use a standard ICA algorithm to obtain an estimate of the mixing matrix, and subsequently permute it and normalize it appropriately before using it to compute B containing the sought connection strengths b_{ij}".



Context

This classic dataset contains the prices and other attributes of almost 54,000 diamonds. It's a great dataset for beginners learning to work with data analysis and visualization.

Content

price price in US dollars (\$326--\$18,823)

carat weight of the diamond (0.2--5.01)

cut quality of the cut (Fair, Good, Very Good, Premium, Ideal)

color diamond colour, from J (worst) to D (best)

clarity a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))

x length in mm (0--10.74)

y width in mm (0--58.9)

z depth in mm (0--31.8)

depth total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43--79)

table width of top of diamond relative to widest point (43--95)

Figure 2: Descriptions of raw diamond dataset

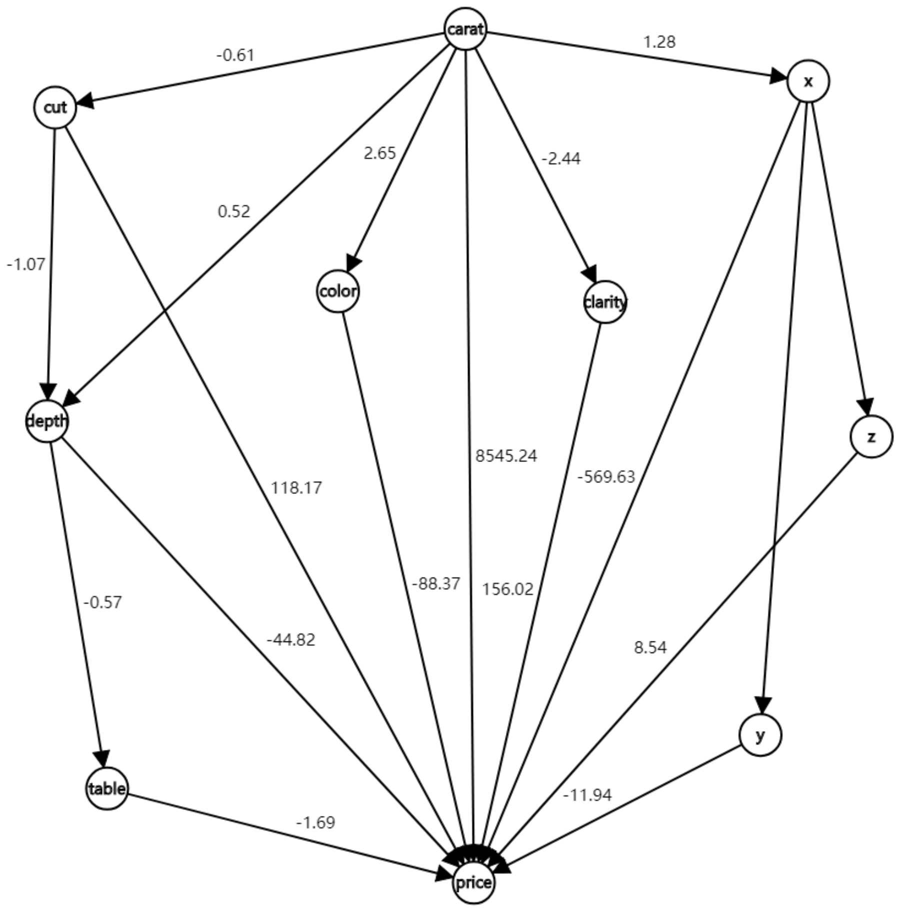


Figure 3: Visualization of LiNGAM algorithm's result

Table 7: the resulted B matrix using LiNGAM algorithm

B	carat	cut	color	clarity	depth	table	price	x	y	z
carat	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
cut	-0.61	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
color	2.65	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
clarity	-2.44	-0.05	0.29	0.00	-0.05	0.00	0.00	0.00	0.00	0.00
depth	0.52	-1.07	-0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
table	-0.38	-0.05	0.06	-0.38	-0.57	0.00	0.00	0.00	0.00	0.00
price	8545.24	118.17	-88.37	156.02	-44.82	-1.69	0.00	-569.63	-11.94	8.54
x	1.28	-0.01	0.02	0.02	-0.05	-0.02	0.00	0.00	0.00	0.00
y	-0.01	-0.01	0.00	0.00	-0.01	-0.00	0.00	0.98	0.00	0.00
z	0.02	-0.00	-0.00	0.00	0.06	-0.00	0.00	0.53	0.07	0.00

2.3 Results

The resulted B matrix is detailed in table 7 and the visualization in figure 3 (some little weighted edges are discarded). The result is OK. At least, the DAG constructed indicates price is a consequences of various factors and does not decide value of any another. Each element b_{ij} in matrix B indicates the coefficient of variable x_j for the linear combination of variable x_i.

However, the carat becomes a casual factor for many others. I think this is a result of violation of the assumption of no unobserved confounder. Factors such as carat, x, y, z can have a common confounder, depending on how the diamond is processed. Besides, maybe the linear assumption is too simple to represent complex casual relations. So far, current casual models are always based on too many preconditions on the data and cannot be well applied to real-world problems.

References

[1] Alex Krizhevsky and Sutskever, Ilya and Hinton, Geoffrey E, *ImageNet Classification with Deep Convolutional Neural Networks.* NIPS2012-4824. 2012.

[2] Shohei Shimizu, Patrik O. Hoyer, Aapo Hyvarinen, Antti Kerminen, *A Linear Non-Gaussian Acyclic Model for Causal Discovery.* Journal of Machine Learning Research. 2016.

[3] Tichavsky, et al. *Performance analysis of the FastICA algorithm and Cramer-Rao bounds for linear independent component analysis.* IEEE Trans. on Signal Processing, 54(4):1189-1203, 2006.