Report on Homework 2

CS420, Machine Learning, Shikui Tu, Summer 2018

Zelin Ye 515030910468

1 PCA algorithm

1.1 Original PCA

When it comes to principal component, the most common used approach is principal component analysis (PCA) algorithm. Thus, I can use the original PCA algorithm to extract the first principal component of the dataset. The computational details can refer to Alg. 1.

Algorithm 1: Original PCA

Input : The dataset X, a $n \times N$ matrix
Output: The first principal component w

Conduct normalization for X, and make sure the mean of X is 0;
Find the covariance matrix of X, denoted by C:

$$C \leftarrow XX^T;$$

Calculate the eigenvalues λ and eigenvectors V of X;
Choose the maximal eigenvalue λ_m and corresponding eigenvector v_m;
Calculate the first principal component:

$$w \leftarrow v_m^T X;$$

return w;

The above algorithm (PCA) is a classical and commonly used method to solve principal components, which has the following characteristics.

Pros:

1. PCA has simple logic and is easy to implement;
2. The orthogonality among principal components chosen by PCA can eliminate the interaction between original data components;
3. PCA belongs to unsupervised learning, and it is not restricted by sample labels.

Cons:

1. PCA treats all samples, namely the collection of eigenvectors, as a whole and neglects the category attributes. Nevertheless, the projection direction it neglects might contain some important separability information;
2. PCA would be time-consuming when encountering large amount of data;
3. The actual meanings of principal components extracted by PCA are a little bit ad-hoc and hard to explain.

1.2 SVD

In fact, I can also extract principle components via SVD (Singular Value Decomposition), which is a way to decompose a matrix as the following equation.

$$X_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n},$$

where Σ is a diagonal matrix, and the values on the diagonal are called the singular values. The vectors in U and V are orthogonal.

In most cases, the sum of singular values of the first 10% or even 1% accounts for more than 99% of the sum of all singular values. Therefore, I can obtain first principle components through the biggest singular value. Corresponding details can refer to Alg. 2.

Algorithm 2: SVD

Input : The dataset X, a $n \times N$ matrix
Output: The first principal component w

1 Conduct normalization for X, and make sure the mean of X is 0;
2 Carry out SVD for X:

$$X \leftarrow U\Sigma V^T;$$

3 Multiply the transposition of matrix U on both sides of the above formula:

$$U^T X \leftarrow \Sigma V^T;$$

4 Then the compression of original data is ΣV^T, denoted by $X^{'}$;
5 Assign the first row of $X^{'}$ to w;
6 **return** w;

Actually, PCA is a wrapper of SVD, SVD is usually used as a implementation of PCA (e.g. sklearn). In addition to most of the characteristics of PCA, SVD has the following ones:

Pros:

1. SVD could handle matrices that are not square matrices, while the eigenvalue decomposition in PCA is only applicable to the square matrices;
2. Different from PCA, there is no need for SVD to calculate XX^T, where some small values in X might be lost in squares;
3. SVD can extract the principle components of two directions. For instance, if I multiply V on both sides of $X = U\Sigma V^T$, I will implement the compression on samples, namely merge some similar samples or eliminate those inconsequential ones. Whereas, PCA cannot achieve this.

Cons:

1. The complexity of calculating the singular values is $O(N^3)$, which is not suitable for large scale samples. Parallel computing can solve this problem a little bit;
2. The compression results of SVD are lack of interpretability.

1.3 KPCA

The original PCA and SVD might have good performance when handling linear data, but it would encounter difficulties under non-linear data. In non-linear cases, a variant of PCA called KPCA (kernel principal components analysis) shows its strength.

The innovation of KPCA is that it introduces a non-linear mapping function $\phi(x)$, mapping the data from original space to high-dimensional space. Besides, the deduction of KPCA takes advantage of the theorem that *each vector in the space could be expressed linearly by all the samples in the space*. The details of KPCA can refer to Alg. 3.

Algorithm 3: KPCA

Input : The dataset X, a $n \times N$ matrix; kernel function $\phi(x)$
Output: The first principal component w

Conduct normalization for X, and make sure the mean of X is 0;

Calculate kernel matrix K:

$$K \leftarrow X^T X,$$

where $X = [\phi(x_1), ..., \phi(x_N)]$;

Find the covariance matrix of X, denoted by C:

$$C \leftarrow XX^T;$$

Calculate the eigenvalues λ and eigenvectors U of K;

Choose the maximal eigenvalue λ_m and calculate corresponding eigenvector u_m;

Calculate the corresponding eigenvectors v_m of covariance matrix C by u_m:

$$\begin{aligned} v_m &\leftarrow \frac{1}{\|X^T u_m\|} X^T u_m \\ &= \frac{1}{\sqrt{u_m^T X X^T u_m}} X^T u_m \\ &= \frac{1}{\sqrt{u_m^T (\lambda_m u_m)}} X^T u_m \\ &= \frac{1}{\sqrt{\lambda_m}} X^T u_m; \end{aligned}$$

Calculate the first principal component:

$$w \leftarrow v_m^T X;$$

return w;

KPCA is a ingenious extension of PCA, and is also widely used (e.g. dimension reduction, clustering). The pros and cons of KPCA are as follows.

Pros:

1. Basically, KPCA owns almost all of the advantages of PCA;
2. KPCA has a stronger universality, which could find out the non-linear information contained in dataset;
3. KPCA uses simple kernel functions (e.g. polynomial kernel function) to realize complex non-linear transform;

Cons:

1. The logic and implementation of KPCA is a little bit complicated;
2. The dimension of kernel matrix K is $N \times N$ (N is the number of samples). It might take quantities of time to process K when handling large scale of samples;

3. Different choices of kernel functions and parameters would affect the result to a certain extent, which makes this algorithm more tricky.

4. Same as PCA and SVD, the actual meanings of principal components extracted by KPCA are also inexplicable.

2 Factor Analysis (FA)

$$\begin{aligned} p(y|x) &= \frac{p(x|y)p(y)}{p(x)} \\ &= \frac{G(x|Ay+\mu, \Sigma_e)G(y|0, \Sigma_y)}{p(x)} \\ &= \frac{G(x|Ay+\mu, \Sigma_e)G(y|0, \Sigma_y)}{G(x|\mu+\mu_e, A\Sigma_y A^T+\Sigma_e)} \end{aligned}$$

where μ_e denotes the mean value of e, generally considered as 0.

3 Independent Component Analysis (ICA)

ICA aims to decompose the source signal into independent parts. If the source signals are non-Gaussian, the decomposition is unique, or there would be a variety of such decompositions.

Suppose the source signal s consists of two components, conforming to multi-valued normal distribution, namely $s \sim N(0, I)$. Obviously, the probablity density funciton of s is centered on the mean 0, and the projection plane is an ellipse.

Meanwhile, we have $x = As$, where x denotes the actual signals received while A represents a mixing matrix. Then x is also Gaussian, with a mean of 0 and a covariance of

$$E[xx^T] = E[Ass^TA^T] = AA^T$$

Let C be a orthogonal matrix, and $A^{'} = AR$. If A is replaced by A', then we can get $x^{'} = A^{'}s$. It is easy to find that $x^{'}$ also conforms to normal distribution, with a mean of 0 and a covariance of

$$E[x^{'}(x^{'})^T] = E[A^{'}ss^T(A^{'})^T] = E[ACss^T(AC)^T] = ACC^TA^T = AA^T$$

Apparently, x and $x^{'}$ conform to the same distribution with different mixing matrices. Then we cannot determine the mixing matrix or the source signals from the received signals. Nevertheless, if x is non-Gaussian (e.g. Uniform Distribution), such case would be effectively avoided.

Therefore, maximizing non-Gaussianity should be used as a principle for ICA estimation.

4 Causal Discovery Algorithms

4.1 Problem Description

Generally speaking, the housing price are affected by many factors, such as the number of rooms, crime rate, pupil-teacher ratio and etc. There also exist some causalities among those factors. It is beneficial for

further mastering the housing market if one can figure out these causalities. However, such causalities are often difficult to determine intuitively, it is thus necessary to conduct causal discovery on these factors.

I consider the features in the Boston Housing dataset as factors, then utilize these features and corresponding values for causal discovery. The dataset is available at the UCI Repository [1], whose details can refer to Appendix A.1.

4.2 Approach

Taking the data type of features and the scale of samples into account, I apply *Kernel-based Conditional Independence Test* (KCI-test) [2] to casual discovery, which performs well when the conditioning set is large and the sample size is not very large.

Basically, conditional independence test for continuous variables is rather challenging due to the curse of dimensionality. KCI-test constructs an appropriate test statistic and deriving its asymptotic distribution under the null hypothesis of conditional independence.

Suppose the centralized kernel matrix of sample X and its eigenvalues are K_X and λ_X respectively. The unconditional independence testing method is similar to HSIC [3]. Under the null hypothesis that X **and** Y **are statistically independent**, the statistic is defined as:

$$\begin{aligned} T_{UI} &\triangleq \frac{1}{n} Tr(K_X K_Y) \\ &= \frac{1}{n^2} \sum_{i,j=1}^{n} \lambda_{x,i} \lambda_{y,j} z_{ij}^2 \end{aligned}$$

where z_{ij}^2 denotes the χ_1^2-distributed variables.

Analogously, under the null hypothesis that X **and** Y **are conditionally independent given** Z, the statistic for conditional independence testing is defined as:

$$\begin{aligned} T_{CI} &\triangleq \frac{1}{n} Tr(K_{X|Z} K_{Y|Z}) \\ &= \frac{1}{n} \sum_{k=1}^{n^2} \lambda_k z_k^2 \end{aligned}$$

where $K_{X|Z} = R_Z K_X R_Z$, $K_{Y|Z} = R_Z K_Y R_Z$, $R_Z = \epsilon (K_Z + \epsilon I)^{-1}$ and ϵ is a hyperparameter.

Finally, after the *p-value* is calculated according to the above formulae, KCI-test approximates the null distribution with a Gamma distribution or just generates it by Monte Carlo simulation.

The main purpose of this application is to figure out the causalities among the features in Boston Housing dataset. Therefore, I use KCI-test to test the dependence and conditional independence among features. Then I construct the DAG of casualities based on PC algorithm [4]. The whole process could be summed up as $PC_{KCI-test}$ algorithm.

4.3 Experiments

Initially, I conduct unconditional independence testing on every two features. With the significance level as 0.001 and null hypothesis specified in Sec. 4.2, the results of unconditional independence testing are shown in Tab. 1.

Table 1: The *p-values* of unconditional independence testing

p \ id / id	1	2	3	4	5	6	7	8	9	10	11	12
1	0.000	0.000	0.000	0.000	**0.003**	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
5	**0.003**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.005**	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
10	0.000	0.000	0.000	0.000	**0.002**	0.000	0.000	0.000	0.000	0.000	0.000	0.000
11	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
12	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

According to the above table, only a small parts of the features have unconditional independence (rough parts in the table). That is, most features may have causalities with others. However, it is impossible to specify those causalities based on current results.

Therefore, I carry out conditional independence testing. I also choose the significance level as 0.001, and the corresponding results can refer to Tab. 2.

Table 2: The results of conditional independence testing (Since the number of results is excessive, I only count the numbers here and not specify the results).

Condition (Feature) id z	The Number of Independent Features under z	Radio of Independent Features
1	96	0.67
2	11	0.08
3	55	0.38
4	15	0.10
5	121	0.84
6	112	0.78
7	59	0.41
8	79	0.55
9	104	0.72
10	8	0.06
11	117	0.81
12	112	0.78

It is easy to find out that the cases of conditional independence among these features are quite different. Most of the features are independent under 2th, 4th, 10th conditions (features), whereas the results are opposite under 5th, 11th, 12th conditions.

Ultimately, I exploit the above results to execute PC algorithm. Although each one could argue about the ground truth for the causalities of these features, I approve of the criterion proposed by the author of KCI-test: one algorithm is promising if it finds links between number of rooms (RM) and median value of houses (MED) and between non-retail business (IND) and nitric oxides concentration (NOX). The final result of PC algorithm can refer to Fig. 1, which is quite similar to that in [2].

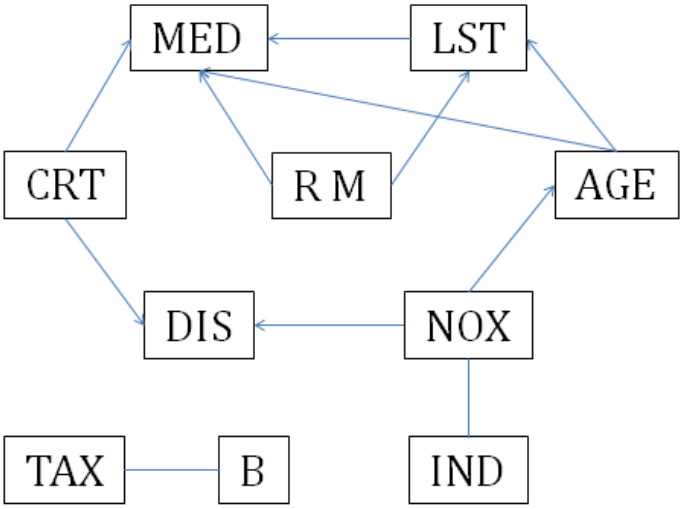

Figure 1: The final result of PC algorithm, namely the causalities of the features in Boston Housing dataset.

It is doubtless that this result meets the criterion perfectly. Besides, the $PC_{KCI-test}$ algorithm also finds out some other causalities that make sense (e.g. nitric oxides concentration (NOX) and distances to five Boston centres (DIS)). Nevertheless, I think this algorithm still omits some causalities that meet our common sense, such as DIS [1] and TAX, IND and TAX, which could be discovered by $PC_{CI_{PERM}}$ algorithm according to the results in [2]. In the meantime, $PC_{KCI-test}$ also ignores the causalities of two features (ZN and PTRATIO).

In conclusion, the KCI-test is a concise and efficient algorithm for independence testing, especially for continuous variables. The experiments on Boston Housing dataset indicates that KCI-test could achieve satisfactory results in practical applications.

5 Causal Tree Reconstruction

Firstly, consider three variables x_1, x_2, x_3, form their star decomposition, then choose a fourth variable x_4 and determine the leg where x_4 should be appended. Such leg could be found by testing which pairs of triplets share centers, deciding on appropriate topology, connecting x_4 accordingly.

If there is a tree T_i, with i leaves, ready to append the next, namely $(i+1)th$, leaf. The solution is similar: choose any triplet of leaves from T_i with central variable w, test which leg of w the x_{i+1} should append. This identifies a subtree $T_i^{'}$ of T_i that should receive x_{i+1} and permits me to remove from further consideration the subtrees emanating from the unselected legs of w.

Repeating the above operations on the selected subtree $T_i^{'}$ would eventually reduce it to a single branch, to which x_{i+1} is appended. The detailed construction procedure could be treated as two parts: 1) reconstructing rooted trees; 2) constructing unrooted tree.

[1] The meaning of abbreviations can refer to A.1

5.1 Reconstructing Root Trees

Suppose T is a rooted tree, with n leaves $x_1, x_2, ..., x_n$, each node of T, which is not a leaf, has 2 to k sons. x_i is defined as the leader of the triple (x_i, x_j, x_k) if the path from the root to it does not contain the deepest common ancestor of x_j and x_k. Note that the deepest common ancestor of all the three leaves is also a common ancestor of any two of them if (x_i, x_j, x_k) does not have a leader.

The algorihm constructs a sequence of trees $T_2, T_3, ..., T_n$, where T_2 is a binary tree with three nodes and $T_n = T$. T_{i+1} is constructed by appending x_{i+1} as a new leaf to T_i. T_i contains only the leaves $x_1, ..., x_i$ and corresponding edges, which would be the subtree of T. The location where x_{i+1} should be appended to is determined by Alg. 4. The lemmas used in the process can refer to Appendix A.2.

Algorithm 4: Appending

Input : T_i, x_{i+1}, i
Output: T_{i+1}

$T_c \leftarrow T_i$;
$n \leftarrow$ the number of leaves in T_c ;
if $s = 2$ **then**
  $\overline{v} \leftarrow$ the root of T_c ; $x_j, x_k \leftarrow$ the two leaves of T_c ;
else if $s > 2$ **then**
  $\overline{v} \leftarrow$ any node of T_c, for which

$$\frac{s}{k+1} < des(\overline{v} \leqslant \frac{sk}{k+1}) \quad (Lemma\ A.1);$$

  $x_j, x_k \leftarrow$ the two leaves whose common ancestor is $\overline{v}$;
Calculate the leader of (x_{i+1}, x_j, x_k) (with respect to T) ;
if $s = 2$ **then**
  if x_j *(or* x_k*) is the leader* **then**
    Append a new node on the edge of x_k or x_j respectively, making it the father of x_{i+1} ;
  else if x_{i+1} *is the leader* **then**
    Append a new root, making x_{i+1} and the old root $\overline{v}$ as its sons ;
  else if *there is no leader* **then**
    Make x_{i+1} as a son of $\overline{v}$;
else if $s > 2$ **then**
  Define a partition of T_c into two subtrees: T_{c1} rooted at $\overline{v}$ with all the descendants of $\overline{v}$ and $T_{c2} = T_c - T_{c1}$ in which $\overline{v}$ is considered as a leaf ; **if** x_j *(or* x_k*) is the leader* **then**
    $T_c \leftarrow$ the subtree of T_{c1} rooted at that son of $\overline{v}$ which is the ancestor of x_k (or x_j respectively) ;
  else if x_{i+1} *is the leader of* x_{i+1}, x_j, x_k **then**
    $T_c \leftarrow T_{c2}$;
  else if *there is no leader* **then**
    $T_c \leftarrow T_{c1}$, from which the two sons of $\overline{v}$ whose descendants are x_j and x_k are removed with all their descendants ;
  Go to 2 ;
return T_{i+1};

5.2 Constructing Unrooted Trees

Suppose T is an unrooted tree in which the degree of each node is at least three, and suppose u, v, w, x is any quadruple of leaves. Note that x pairs with u relative to (v, w) if the path from x to u is edge-disjoint relative to the path from v to w.

Remove a leaf x of T and examine the remaining tree T_1 as rooted at the node x_1 with which x is adjacent in T.

Choosing arbitrarily a fixed leaf x, I can use Lemma A.2 and Alg. 4 to reconstruct T_1 rooted at x_1. Then appending x as a son of x_1 to get the required tree T.

A Appendix

A.1 Details of the Boston Housing Dataset

The Boston Housing Dataset consists of 506 samples, each has 12 features, namely the factors that affect housing price. The values of all features are positive real numbers. Table 3 shows a detailed description of these features.

Table 3: Details of the features in Boston Housing dataset

Identifier	Feature Names	Description
1	CRIM	per capita crime rate by town
2	ZN	proportion of residential land zoned for lots over 25,000.000 sq.ft.
3	INDUS	proportion of non-retail business acres per town
4	NOX	nitric oxides concentration (parts per 10.0 million)
5	RM	average number of rooms per dwelling
6	AGE	proportion of owner-occupied units built prior to 1940.000
7	DIS	weighted distances to five Boston employment centres
8	TAX	full-value property-tax rate per $10,000.0
9	PTRATIO	pupil-teacher ratio by town
10.000	B	1000(Bk - 0.63)2 where Bk is the proportion of blacks by town
11	LSTAT	lower status of the population
12	MEDV	Median value of owner-occupied homes in $1000's

A.2 Lemmas

Lemma A.1. *Suppose k is the maximum number of sons of a node in a rooted tree T with n leaves. There exists a node v of T such that*

$$\frac{n}{k+1} < des(v) \leqslant \frac{nk}{k+1},$$

where des(v) is defined to be the number of leaves which are descendants of v, and des(v) = 1 if v is a leaf.

Lemma A.2. *x pairs with u relative to (v, w) in the tree T if and only if u is the leader of u, v, w in the tree* T_1 *rooted at* x_1.

References

[1] A. Asuncion and D. Newman, "Uci machine learning repository," 2007.

[2] K. Zhang, J. Peters, D. Janzing, and B. Schölkopf, "Kernel-based conditional independence test and application in causal discovery," *arXiv preprint arXiv:1202.3775*, 2012.

[3] A. Gretton, K. Fukumizu, C. H. Teo, L. Song, B. Schölkopf, and A. J. Smola, "A kernel statistical test of independence," in *Advances in neural information processing systems*, pp. 585–592, 2008.

[4] P. Spirtes and C. Glymour, "An algorithm for fast recovery of sparse causal graphs," *Social science computer review*, vol. 9, no. 1, pp. 62–72, 1991.