algorithms

MDPI

Article

Computational Test for Conditional Independence

Christian B. H. Thorjussen [1,2,*], Kristian Hovde Liland [2], Ingrid Måge [1] and Lars Erik Solberg [1]

[1] Nofima AS, Osloveien 1, 1431 Ås, Norway; ingrid.mage@nofima.no (I.M.); lars.erik.solberg@nofima.no (L.E.S.)
[2] Faculty of Science and Technology, Norwegian University of Life Science, 1432 Ås, Norway; kristian.liland@nmbu.no
* Correspondence: christian.thorjussen@nofima.no

Abstract: Conditional Independence (CI) testing is fundamental in statistical analysis. For example, CI testing helps validate causal graphs or longitudinal data analysis with repeated measures in causal inference. CI testing is difficult, especially when testing involves categorical variables conditioned on a mixture of continuous and categorical variables. Current parametric and non-parametric testing methods are designed for continuous variables and can quickly fall short in the categorical case. This paper presents a computational approach for CI testing suited for categorical data types, which we call computational conditional independence (CCI) testing. The test procedure is based on permutation and combines machine learning prediction algorithms and Monte Carlo cross-validation. We evaluated the approach through simulation studies and assessed the performance against alternative methods: the generalized covariance measure test, the kernel conditional independence test, and testing with multinomial regression. We find that the computational approach to testing has utility over the alternative methods, achieving better control over type I error rates. We hope this work can expand the toolkit for CI testing for practitioners and researchers.

Keywords: conditional independence; computational hypothesis testing; categorical variables; graphical models; causal inference

check for updates

Citation: Thorjussen, C.B.H.; Liland, K.H.; Måge, I.; Solberg, L.E. Computational Test for Conditional Independence. *Algorithms* **2024**, *17*, 323. https://doi.org/10.3390/a17080323

Academic Editor: Tatsuya Akutsu

Received: 27 June 2024
Revised: 18 July 2024
Accepted: 20 July 2024
Published: 24 July 2024

1. Introduction

Robust conditional independence (CI) testing in statistics is an arduous challenge. The no-free-lunch theorem for CI testing, as stated in [1], postulates that no single statistical test or algorithm is universally superior for detecting CI between variables of all types in all situations. Nevertheless, CI testing is a pivotal task in applied statistical research. For instance, in the domain of causal inference, a causal graph, which can form the basis of a causal estimator, can be validated through testing a set of implied marginal independence and CI statements [2,3]. In bioinformatics and genomics, CI aids in understanding genetic linkage and is vital in distinguishing between genetic and environmental factors, essential for tasks like mapping disease traits [4,5]. Because of its importance, considerable work [1,6–11] has been dedicated to creating tests for CI, especially for continuous cases. However, due to the no-free-lunch theorem, the performance of different testing methods can vary substantially depending on the underlying data-generating structure.

Consequently, it is necessary to develop and explore multiple testing methods for CI testing tailored for various data characteristics and types, adopting both asymptotic and computational approaches. Due to the flexibility of regression and classification algorithms in machine learning (ML), state-of-the-art prediction modeling offers a promising starting point for testing CI [1]. This article proposes and examines a computational method based on ML classification (and regression) for testing CI in this context. While the approach applies to various data types, we will focus on CI testing involving cases with a mixture of categorical and continuous variables in the CI statement. To our knowledge, no suitable testing method currently exists for such cases.

Conditional Independence Testing

A common notation of CI between random variables Y and X given Z is

$$Y \perp\!\!\!\perp X|Z \Leftrightarrow X \perp\!\!\!\perp Y|Z, \tag{1}$$

where Z can be extended to a set of variables $\mathbf{Z}$. One can also express CI using probability distributions; $p(Y \mid X, \mathbf{Z}) = p(Y \mid \mathbf{Z})$ where $p(\mathbf{Z}) > 0$, or $p(X, Y \mid \mathbf{Z}) = p(X \mid \mathbf{Z})p(Y \mid \mathbf{Z})$, which is to say that given that we know $\mathbf{Z} = \mathbf{z}$ neither X nor Y carries information about each other's distribution [12].

When Y and X are continuous and the conditioning set $\mathbf{Z}$ contains a mixture of variable types, numerous testing methods for CI exist. The two state-of-the-art and readily available methods are the kernel-based test (KCI test) from [6] and the generalized covariance measure (GCM test) [1]. The KCI test is based on the foundational idea that the uncorrelatedness between functions in kernel Hilbert spaces can characterize statistical independence and CI. The test uses kernel matrices associated with the variables to construct a test statistic based on these matrices. As it stands, the test is designed for continuous data. However, since the KCI test uses a kernel function, it can potentially be modified to account for other data types by changing the function. The GCM test is based on the fact that the standardized covariance between residuals from the two generic regressions $y = h(z)$ and $x = g(z)$ follows, under the null hypothesis and given relatively mild assumptions, an asymptotic standard normal distribution. Although the GCM test statistic is designed for continuous data types, it also handles binary data well.

When dealing with purely categorical variables within the CI statement, i.e., categorical variables with more than two categories, the literature on testing methods is somewhat scarcer. In cases where Y, X, and Z in Equation (1) are all categorical or of discrete types, the conditional χ^2 test, also known as the Cochran-Mantel-Haenszel test, is a well-known and reliable test. The χ^2 test requires sufficiently many observations in each conditioned category to function optimally [13] (pp. 231–232). Testing based on mutual information is also a common choice for testing CI with only discrete data [14]. However, in CI statements where either X or Y or both are categorical and Z consists of continuous variables, or a mix of continuous variables and other types, there are, to our knowledge, no reliable testing methods available. For example, it is suggested to employ multinomial regression or abstain from testing altogether [2]. Testing CI with a parametric model is problematic as it relies on specifying the correct, or close to correct, functional form.

To create a CI testing framework for categorical variables, we suggest using ML classification algorithms in combination with permutation and Monte Carlo cross-validation (MCCV) to create a null distribution. One utilizes the same ML classification model without permutation with one train-test split to obtain the test statistic and p-value. As an extension to this, one can also create a test statistic distribution with MCCV and assess a distribution of p-values, which is a helpful extension in cases where testing is done with a limited sample size. The organization of our article is as follows: Section 2 details our proposed testing procedure, Section 3 presents the simulation study exploring key aspects such as type I error rates and power under difficult testing conditions. In Section 4, we present an illustrative example of the testing procedure, and test results of multiple simulations over a range of multivariate distributions. A general discussion follows in Section 5, and concluding remarks in Section 6.

2. Computational Testing with Categorical Variables

Our proposed computational procedure for testing CI, which we call the computational conditional independence test (CCI test), is, in principle, quite flexible. Still, we focus on the case where the aim is to test CI between two categorical variables.

Let Y and X represent categorical variables with $j = 1 \dots J$ and $k = 1 \dots K$ categories, respectively. Under the null hypothesis of CI, a classification model for Y, $P(Y = k \mid X, Z)$,

should perform equivalently to a model using a permuted X^* instead of X. Hence, we posit the null and alternative hypotheses as follows

$$\begin{aligned} H_0 :& M(P(Y = k \mid X^*, Z)) = M(P(Y = k \mid X, Z)) \\ & \Leftrightarrow M(P(X = j \mid Y^*, Z)) = M(P(X = j \mid Y, Z)), \end{aligned} \tag{2}$$

$$\begin{aligned} H_1 :& M(P(Y = k \mid X^*, Z)) \neq M(P(Y = k \mid X, Z)) \\ & \Leftrightarrow M(Pr(X = j \mid Y^*, Z)) \neq M(Pr(X = j \mid Y, Z)), \end{aligned} \tag{3}$$

where M denotes a suitable performance metric whose distribution is (usually) unknown. A performance metric, is a quantified evaluation of an ML model's performance; how effective an ML model is on a given task. The type of performance metric in ML depends on the problem being solved (classification, regression, clustering). Within a specific problem, different metrics assess various aspects of an ML model [15]. The CCI test approximates the distributions of $M()$ under H_0 and H_1 by computing over many Monte Carlo samples.

2.1. Establishing a Null Distribution by MCCV

A fundamental challenge in statistical testing lies in controlling the significance level, defined as the false positive rate or type I error rate. We manage this error rate by estimating an empirical null distribution of metric scores using Monte Carlo Cross-Validation (MCCV) in combination with permutation. Generally, statistical testing based on permutation is a robust alternative when fundamental assumptions of parametric and nonparametric tests are violated [16]. MCCV is a resampling technique typically used to assess ML models' performance by randomly partitioning data into training and testing sets. For each iteration, MCCV randomly selects a subset of data for training, and the remaining observations serve as the test (validation) set.

The first step in setting up a test is to choose between a classification model for Y or for X. Tree-based models, which are robust against overfitting when predicting new data, are probably the best choice. However, we will discuss this further in Section 2.3. For now, let us assume that we have a robust classification model for Y and therefore $M(P(Y = k \mid X^*, Z)) = M(P(Y = k \mid X, Z))$ is the null hypothesis.

Now we can estimate the empirical null distribution(s). For each MCCV iteration, we first permute X into X^*, then randomly split the data into a training set of size pN and a testing set of size $(1 - p)N$. We then train a classification model for Y using X^* and the conditioning set Z with the training set data. We then calculate the classification performance metric using the testing set. After many iterations, this approach theoretically derives the empirical distribution of metric scores under the null being true.

2.2. Calculating p-Values

Once the null distribution is established, we generate a test statistic from a single estimation of the classification model for Y using the original, unpermuted X. Again, we split the data into a random training and testing set, estimate the classification model using the training data, and calculate the performance metric using the testing data. Under the null hypothesis, the metric score calculated using the unpermuted X should be a random draw from the empirical null distribution.

Conversely, if the alternative hypothesis is true, including the unpermuted X should enhance classification performance, thereby facilitating the p-value calculation based on the test statistic's relative position within the null distribution. The (one-sided test) p-value (e.g., using log loss as the performance metric) is given by

$$p = \frac{\sum(\text{Null} \leq \text{test statistic}) + 1}{n + 1}, \tag{4}$$

where $\sum(\text{Null} \leq \text{test statistic})$ represents the count of instances in the null distribution that are less than or equal to the test statistic, and n is the total number of observations in the null

distribution. Note that the direction of $\leq$ in Equation (4) depends on which metric score is used. If, for instance, we use Kappa scores, where a higher value means a better-performing model, we would have to change $\leq$ to $\geq$. The "+1" in the numerator and the denominator is a correction factor that prevents a p-value of 0 and makes the p-value somewhat more conservative. A p-value of 0 is good to avoid since it implies absolute certainty.

Testing CI by classification modeling for an accurate p-value calculation demands a large sample. When the sample size is small, the statistical power is often low. With an insufficient sample size, minor yet meaningful improvements in performance by including the original X rather than the permuted X^* may not be detected from a single train-test split of the unpermuted data. Therefore, when testing CI with a relatively small sample, one should not rely on one p-value but instead perform multiple train-test splits and examine the resulting distribution of p-values, providing additional robustness to the analysis. According to the probability integral transform theorem, if a random variable T is drawn from a known cdf $F_T(t)$, as with a test statistic under the null hypothesis, the transformed variable $P = F_T(T)$ follows a uniform(0,1) distribution [17] (pp. 54–55). This means that CCI test p-values should be approximately uniformly distributed under the null hypothesis.

By calculating and plotting several p-values in Quantile-Quantile (QQ) plots, one can visually assess if the empirical p-values approximately follow the distribution against their theoretical counterparts. It is important to emphasize that evaluating the distribution of p-values is not a test per se, and one cannot and should not test statistically if the empirical distribution of p-values follows a theoretical uniform distribution, for instance, by a Kolmogorov-Smirnov test [18]. Firstly, such tests assume an underlying continuous distribution, which is not the case. Secondly, these tests are susceptible to sample size, where a large sample size almost guarantees a low p-value, and the "sample size" can be manipulated by the number of MCCV iterations. Thirdly, the Kolmogorov-Smirnov and other similar tests test against uniformity, which is, in this case, incorrect.

2.3. Choice of Classification Model and Performance Metric

The computational method described above assumes that modeling the classification probabilities $P(Y = k|X, Z)$ or $P(X = j|Y, Z)$ allows us to extract all relevant information from the conditioned variables. Significantly, the method relies on the robustness of the classification model in generalizing from the training data to the testing data. Although an optimal classification model cannot be guaranteed, modern off-the-shelf tree-based ML algorithms such as XGBoost https://xgboost.readthedocs.io/en/stable/ (eXtreme Gradient Boosting) [19] have demonstrated remarkable performance in predicting class categories and robustness against overfitting training data. Other alternatives are random forest [20] or other gradient boosting frameworks such as LightGBM [21] or CatBoost [22].

Since different metrics evaluate various aspects of a classification model's performance, choosing a performance metric carries implications in CCI testing. The metrics we use in this article are log loss and Kappa scores; these are well-known, continuous, and have somewhat different qualities. Kappa scores are better when comparing the same model across different datasets, especially when categories are unbalanced. Log loss offers greater precision than Kappa scores by accounting for the certainty of class predictions, which is relevant. We have yet to determine which metrics best suit the task; we will attempt to shed some light on this question in Section 3. Note that performance metrics are different from loss functions, which are necessarily differentiable. However, loss functions (such as log loss) can also be used as performance metrics.

3. Simulation Setup

The simulation study aims to evaluate the performance of CCI testing and compare it against some "established" methods. We also aim to assess and compare the two performance metrics: log loss and Kappa scores.

The basic simulation setup is that we draw data from seven different multivariate distributions with four variables $\{Z_1, Z_2, X, Y\}$. The dependencies between the variables

are as shown in the directed acyclic graph depicted in Figure 1, which implies the CI statement $Y \perp\!\!\!\perp X|Z_1, Z_2 \Leftrightarrow X \perp\!\!\!\perp Y|Z_1, Z_2$. All multivariate distributions involve non-linear relations between Z_1 and Z_2, made such that testing CI is not trivial. In all simulations, Z_1 and Z_2 are continuous variables, either normally or uniformly distributed, while both X and Y are categorical. A short non-technical description of the data-generating functions follows, and pseudocodes of the functions are found in Appendix B.

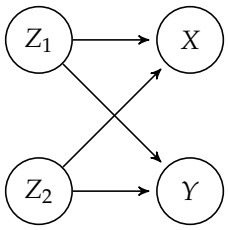


Figure 1. Simulations are based on a common structural framework. Z_1 and Z_2 are continuous variables of distinct types, whereas X and Y are categorical variables.

Interaction: interactions between Z_1 and Z_2. Z_1 and Z_2 are standard normal, and X and Y are set deterministically by a series of if statements which assign values from 0 to 3, depending on the quadrant in which the coordinate (Z_1, Z_2) falls into, creating an interaction between Z_1 and Z_2.

ExpLog: Exponential and logarithmic distributions. Z_1 and Z_2 are standard normal. X is influenced by the exponentiation of the sum of Z_1 and Z_2, where the categories of X change as the sum exceeds certain thresholds. The categorization of Y is based on the logarithm of the absolute value of Z_1 plus one, added to Z_2, passing specific thresholds. The simulation imitates the modeling of exponential and logarithmic growth.

PolyData: Z_1 and Z_2 are standard normal. The categorization of X is based on different polynomial combinations of Z_1 and Z_2. The categorization of Y similarly involves higher-degree polynomials of Z_1 and Z_2.

TrigData: Z_1 is uniform($-\pi$, π), while Z_2 is standard normal. X is created based on adding $\sin(Z_1)$ and $\cos(Z_2)$, and the categorization of Y is based on subtracting $\sin(Z_1)$ and $\cos(Z_2)$. The simulation imitates the modeling of cyclic phenomena.

Non-Linear: A non-linear and trigonometric simulation testing if CI testing methods are robust against complex real-world data patterns involving periods or cycles such as in time-series analysis.

ComplexCategorization: A simulation that emulates data scenarios where multiple factors and their interactions influence outcomes.

Multinomial: This simulation is based on multinomial regression equations, with non-linear effects from Z_1 and Z_2. This simulation aims to generate a data set where the influence of Z_1 is relatively weak and, therefore, harder to test $Y \perp\!\!\!\perp X|Z_2$.

In all testing scenarios, we employ the XGBoost classification model from the R package xgboost [23], using default parameter settings with two exceptions: the learning rate is set to $\eta = 0.1$, and the number of boosting trees is set at $ntrees = 120$. The train-test split ratio in MCCV is set to 0.825, meaning for each iteration, a random portion of 82.5% of the data is utilized as training data. Additionally, we use the createDataPartition function from the caret package [24] to balance the class distributions within the splits. As "feature" variables in the XGBoost classification model we include Z_1 and Z_2 and their transformations: squared and cubed.

We also test the conditional independence statements with the GCM test, the KCI test, and multinomial regression. The GCM test, conducted via the R package Generalised-CovarianceMeasure [25], uses default settings. The KCI test, implemented through the R package CondIndTests [26], has its Gaussian Process hyperparameter routine disabled to reduce computation time. Note that the KCI test demands extensive computational resources with increasing sample sizes. Multinomial regression analyses are performed

using the nnet package [27], incorporating Z_1, Z_2, and their squared and cubed terms as predictors.

We ran our simulations over three sample sizes (500, 800, and 2000) and simulated 100 datasets from the multivariate distributions for each size. For each dataset, we tested $Y \perp\!\!\!\perp X \mid Z_1, Z_2$ to assess the type I error rate. We tested $Y \perp\!\!\!\perp X \mid Z_2$ to assess power, noting that type II error is equal to $1 -$ power. For all tests, we set the significant level at 5%, which means that if we test a true null hypothesis, about 5% of tests should be refuted.

4. Results

4.1. Illustrative Example

Before we present the simulation results, we will illustrate the testing procedure using a specific simulated example. Our dataset comprises 800 simulated observations drawn from the interaction multivariate distribution. We hypothesize that the null hypothesis—$M(P(Y = k \mid X^*, Z_1, Z_2)) = M(P(Y = k \mid X, Z_1, Z_2))$—will not be rejected by the test.

For each MCCV iteration, we start by permuting X into X^* and then train the classification model $\hat{P}(y|x^*, z_1, z_2)$ using 82.5% of the data, and the performance metrics log loss and Kappa score were calculated on the remaining data. We performed 1000 iterations to create the null distributions of log loss and Kappa scores shown in Figure 2. Next, we swap the permuted X^* with the unpermuted X. Perform one instance of a train-test split to estimate $\hat{P}(y|x, z_1, z_2)$, and again, the log loss and Kappa score are extracted by predicting using the testing data. The resulting one-sided p-values are 0.34 (log loss) and 0.36 (Kappa score), and the null hypothesis is not rejected.

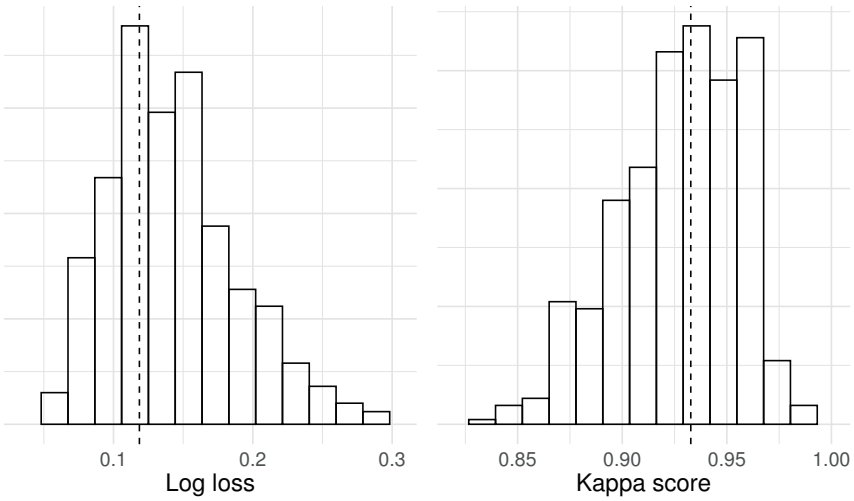


Figure 2. The empirical null distributions of log loss and Kappa scores estimated by MCCV. The dashed lines are the test statistics resulting in p-values of 0.34 and 0.36, respectively.

To show that we have control over type I error in this case, we can estimate the corresponding test statistic distributions and overlay the null distributions, which is shown in Figure 3. The distributions almost perfectly overlay each other, and if we then calculate a p-value for every test statistic, we get that the distribution of p-values follows a *uniform*(0,1) distribution, as shown in Figure 4. Since the p-values are uniformly distributed, we control the type I error rate under a true null, and 5% of the p-values are less than 0.05.

Further, to illustrate how the test works in the opposite case, we remove Z_1 from the test, which gives the null hypothesis $M(P(Y \mid X^*, Z_2)) = M(P(Y \mid X, Z_2))$—which we hypothesize will be rejected by the test. The new null distributions and test statistic distributions are given in Figure 5—the two distributions have almost zero overlap. A random test statistic yields p-values of 0.000099 for both log loss and Kappa score. (Since p-values are calculated from an empirical distribution, 0.000099 is the lowest p-value possible).

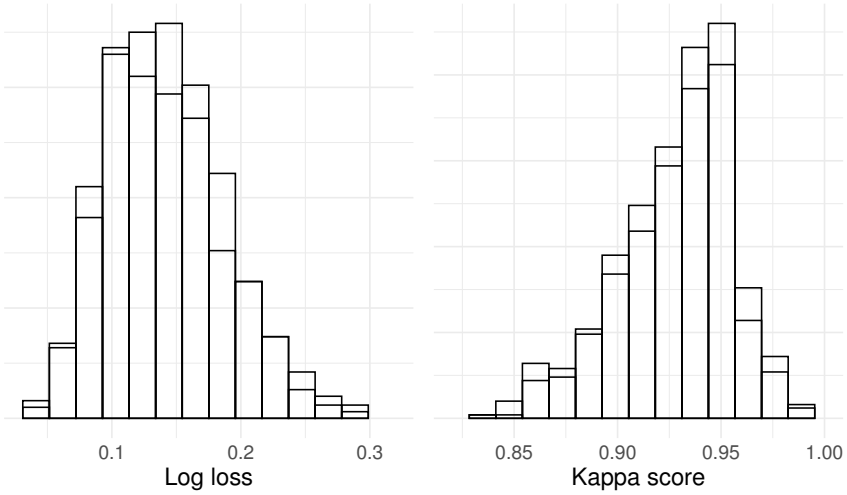


Figure 3. The empirical null distributions overlayed by "test" distributions.

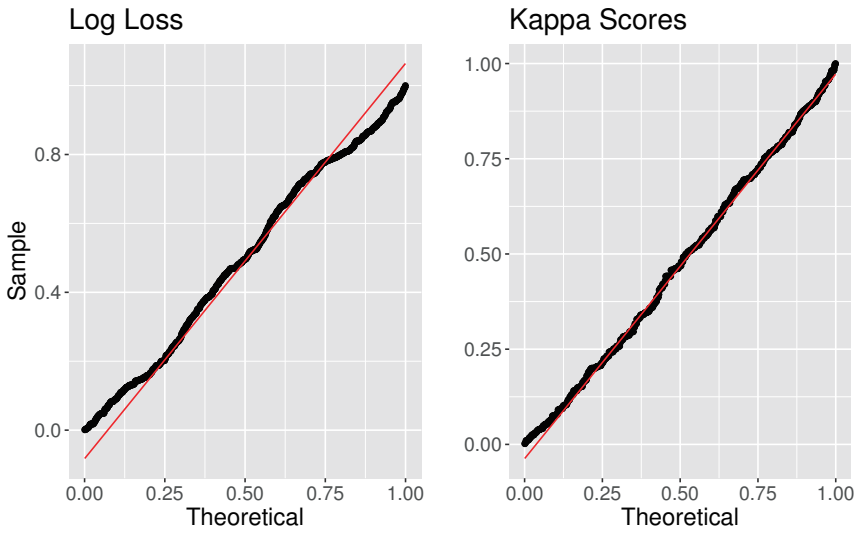


Figure 4. QQ plots of *p*-values from log loss and Kappa scores comparing the empirical distribution (black dots) with the theoretical *uniform*(0,1) (red line).

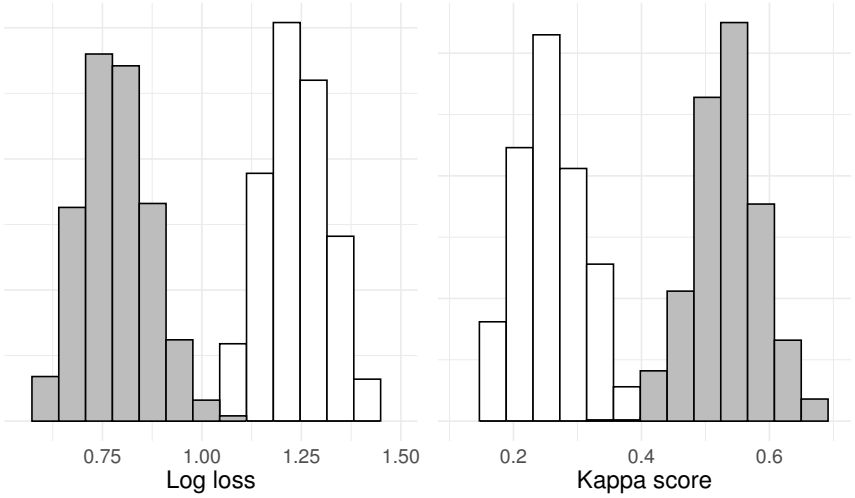
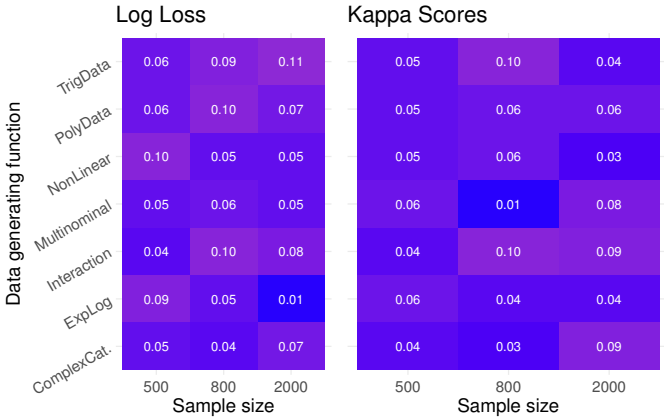


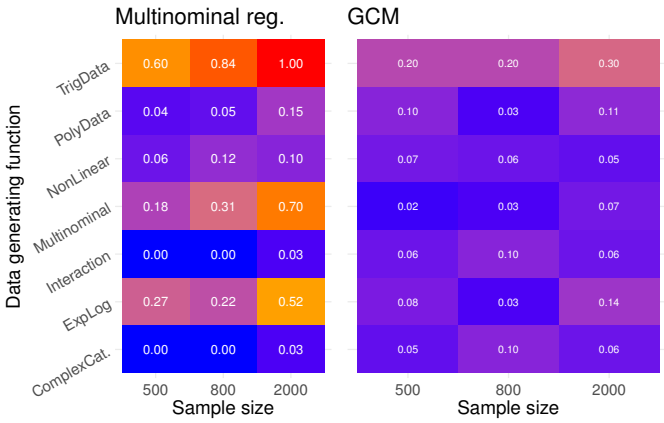
Figure 5. Null (white) and test (gray) empirical Monte Carlo distributions when the null hypothesis is incorrect; the two distributions diverge.

4.2. Testing with One p-Value

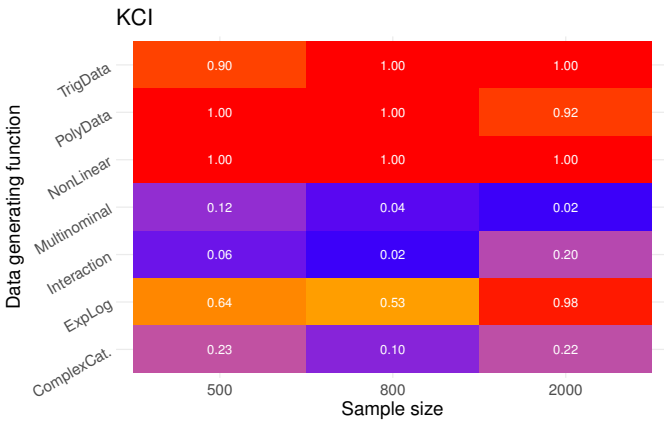
In the heat and overlay plots in Figure 6a, we see the type I error rate for 100 tests for each sample size using the seven proposed simulations. The computational testing procedure exhibits fairly good type I error across different data-generating functions, especially if one were to compare against alternative methods, shown in the heat plots in Figure 6b,c. Data simulated from the TrigData seems like the simulation scenario where it is most difficult to control the error rate. Overall, Kappa scores seem more stable in type I error control.



(**a**)



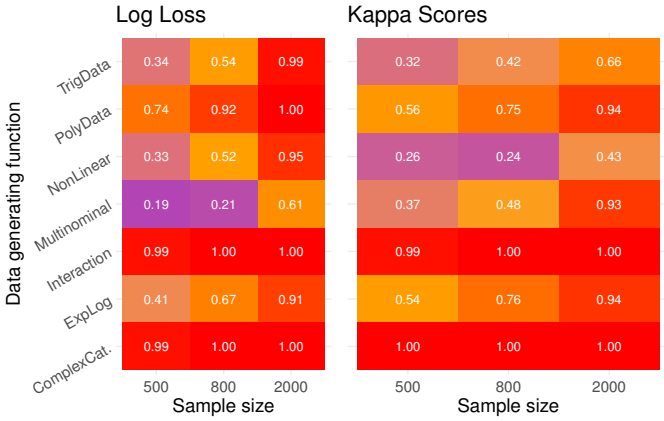
(**b**)



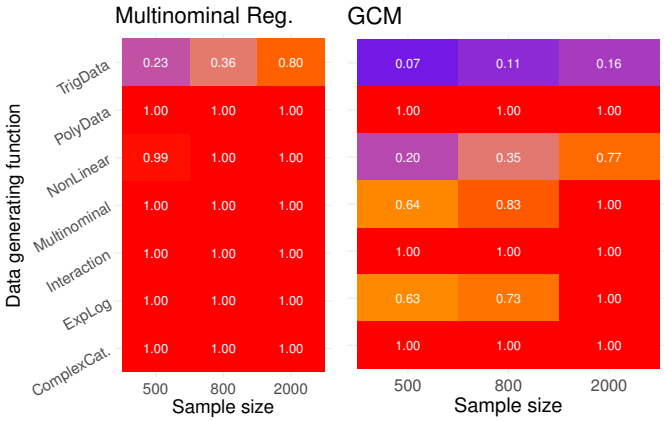
(**c**)

Figure 6. Type I error rates for different testing methods. A bluish color means the error rate is low and should ideally be around 0.05; purple, orange, yellow, and red indicate a high type I error rate. The CCI test exhibits the best type I error control across simulation scenarios. (**a**) CCI testing using log loss (left) and Kappa scores (right). (**b**) Multinominal regression and GCM. (**c**) KCI.

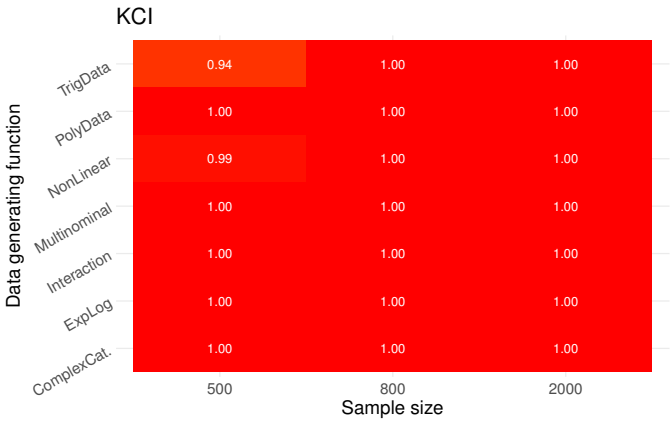
In Figure 7a, we see the estimated power of the computational testing procedure. The power is calculated as the proportion of p-values less than 0.05 when testing $Y \perp\!\!\!\perp X|Z_2$. The power increases with the sample size, and using log loss seems to give more power than Kappa scores. The possibility for low power with small samples emphasizes that computational testing is a "large" sample method due to its reliance on ML classification methodology.



(**a**)



(**b**)

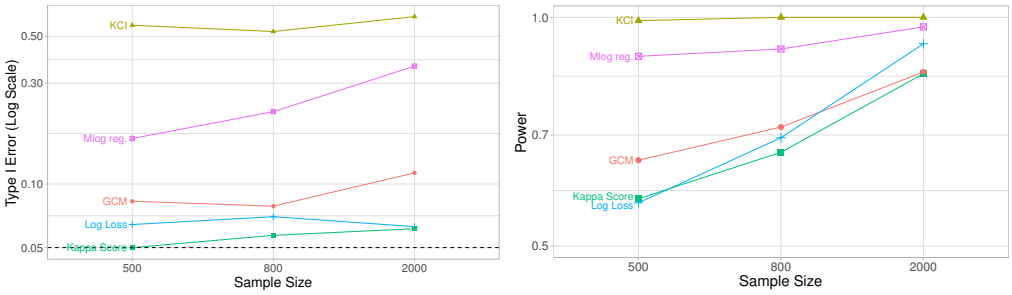


(**c**)

Figure 7. Heatmaps of power (null: $Y \perp\!\!\!\perp X|Z_2$). More red means higher power, and a power of 1.00, means a false null hypothesis is always rejected. The CCI test demands a relatively large sample size to reject a false null consistently, using only one p-value. Ideally, a test needs to achieve at least 80% power. (**a**) Power over simulation scenarios for CCI test using log loss (left) and Kappa scores (right). (**b**) Multinominal regression and GCM. (**c**) KCI.

The other methods, multinominal regression, GCM test (Figure 7b), and KCI test (Figure 7c), have high power, but testing CI with categorical variables with these methods can be very misleading as the type I error rate can be as high as 100%.

In Figure 8a,b, we have averaged the type I error and power over all simulations, which leads us to conclude that the proposed computational testing is a viable alternative to existing methods.



(**a**) (**b**)

Figure 8. Means of type I error and power over all simulation scenarios and testing methods. (**a**) Type I error means for all simulation scenarios the CCI test is represented with the blue and green line (log loss and Kappa scores). Logarithmic Y-scale. (**b**) Testing power over all simulation scenarios, which are all increasing with sample size. Note that the Y-scale goes from 0.5 to 1.

4.3. Assessing with p-Value Distribution

To assess how CCI can analyze CI statements by calculating a distribution of p-values instead of relying on one p-value, we generated data from the three simulation scenarios, Multinomial, TrigData, and NonLinearData, where a false CI is the most challenging to test. For each simulation scenario, we generated 20 datasets, each containing 500 observations for testing both the true null of $M(P(Y = k \mid X^*, Z_1, Z_2)) = M(P(Y = k \mid X, Z_1, Z_2))$, and an erroneous null, $M(P(Y = k \mid X^*, Z_2)) = M(P(Y = k \mid X, Z_2))$. For each dataset (i.e., 60 datasets), we estimated 500 p-values by estimating an empirical null distribution and the corresponding distribution of test statistics.

In Figure 9, the 500 p-value points for each test are color-coded. Each QQ plot has 20 lines of color-code dots, one for each dataset. When testing a true null, we obtain the plots in Figure 9a,c,e. There is a close alignment with the theoretical uniform distribution, which suggests good adherence to the null hypothesis. Some degree of variability, as in minor deviations from the diagonal, is acceptable. Kappa scores seem to have somewhat less idiosyncratic variation.

The QQ plots, depicted in Figure 9b,d,f, show the distribution of p-values for testing $Y \perp X|Z_2$. The points, especially using Kappa scores as the underlying metric, have a notable deviation compared to their theoretical counterparts. This deviation suggests a systematic tendency towards lower p-values than those expected under the null hypothesis, indicating a likely violation of the null, the desired outcome. One can note that when assessing the QQ plots, one should evaluate the difference between the points and the line based on vertical distance [28].

Kappa scores provide better testing performance using QQ plots. With Kappa scores, p-values seem more stable around the diagonal line when testing a true null and more deviate from it when testing a false null.

In addition to providing a testing framework in small samples, using QQ plots has some other benefits. QQ plots can reveal patterns that can indicate fundamental errors, for instance, if the p-values tend to be high, meaning that the points would bend upwards to the left; such a pattern can indicate the non-independence of samples or highly correlated features.

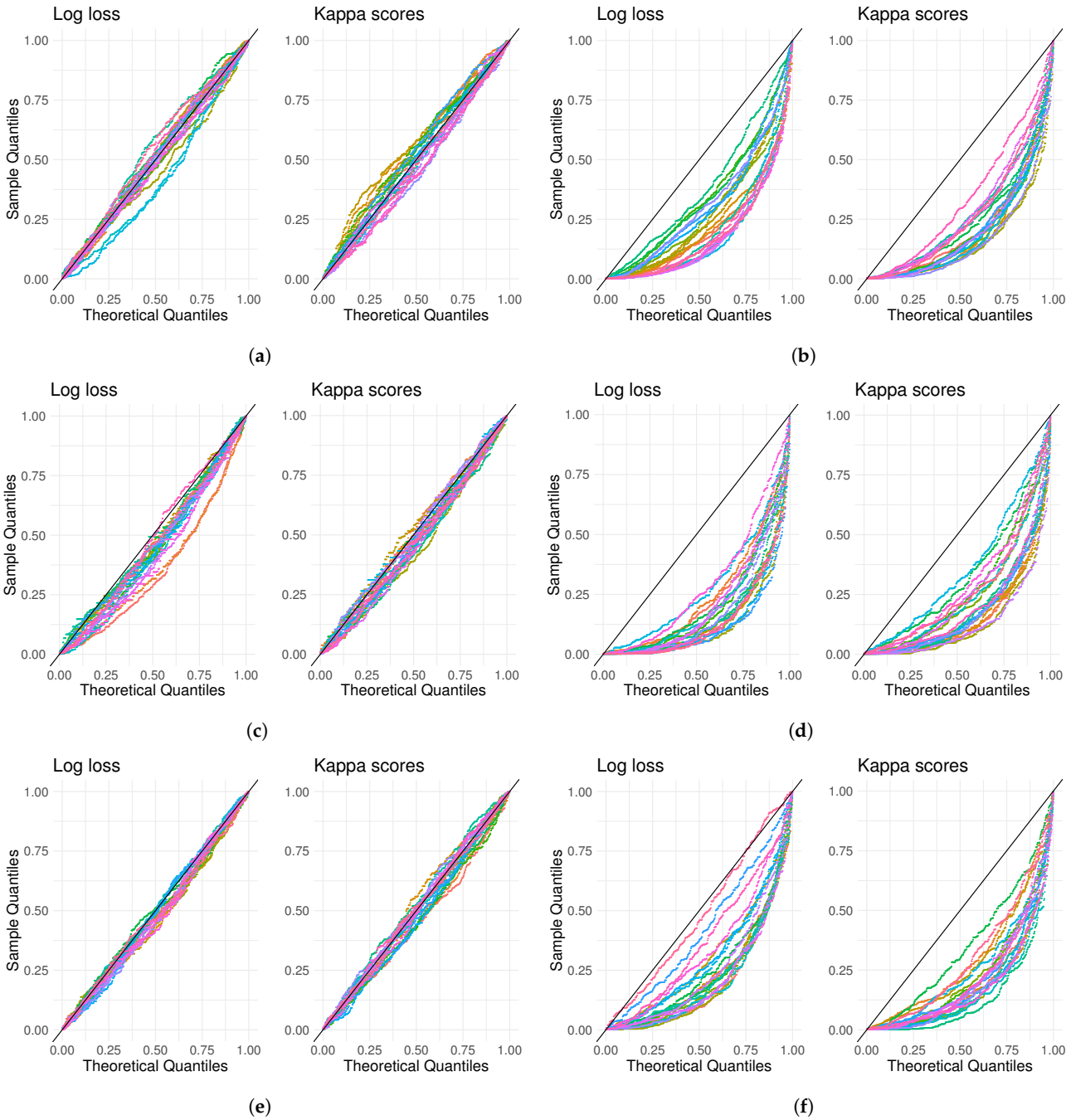


Figure 9. QQ plots: These plots visualize the distribution of p-values testing with the CCI test compared to the theoretical counterpart of a *uniform*(0,1) distribution. The different colors in each plot represent separate datasets within the simulation. (**a**) Multinomial Simulation, True Null Scenario: 500 p-values testing $Y \perp\!\!\!\perp X|Z_1, Z_2$. (**b**) Multinomial Simulation, False Null Scenario: 500 p-values testing $Y \perp\!\!\!\perp X|Z_2$. (**c**) TrigData Simulation, True Null Scenario: 500 p-values testing $Y \perp\!\!\!\perp X|Z_1, Z_2$. (**d**) TrigData Simulation, False Null Scenario: 500 p-values testing $Y \perp\!\!\!\perp X|Z_2$. (**e**) NonLinearData Simulation, True Null Scenario: 500 p-values testing $Y \perp\!\!\!\perp X|Z_1, Z_2$. (**f**) NonLinearData Simulation, False Null Scenario: 500 p-values testing $Y \perp\!\!\!\perp X|Z_2$.

5. Discussion

When testing CI with categorical variables, we can expect that CCI outperforms the continuous testing methods GCM and KCI since, in these cases, the fundamental assumptions of GCM and KCI are violated. Although the basic assumptions of multinomial regression are not violated when used to test CI, any parametric method is restrictive as the

functional form needs to be defined. As the simulation results show, the better performance of CCI is not due to superiority in rejecting false hypotheses but due to both better control of type I error and increasing power as the sample size increases.

Given $Y \perp\!\!\!\perp X|Z$, it should not matter if Y or X is the dependent variable in the classification algorithm in CCI. A criterion for choosing Y or X is if some variable in Z is highly correlated with either Y or X. Then, choose the one with the highest correlation as the dependent variable to reduce the correlation among the features in the classification model. In practical settings, we recommend running a single case to see if there are any estimation problems by choosing Y or X as the dependent variable. Furthermore, to ensure optimal results, we recommend tuning the hyperparameters of the classification model using a separate dataset not involved in the final testing. This practice can potentially improve the performance and reliability of the test results. Given that CCI testing relies on ML techniques, familiarity with these methodologies is recommended for practitioners employing the CCI test.

5.1. Assumptions and Limitations

There are two key implicit assumptions in the CCI testing procedure using XGBoost. Firstly, the XGBoost classification modeling must be able to unravel potentially complex relationships between the variables. Secondly, the XGBoost classification algorithm must generalize well to unseen data. Of course, there is no way to ensure this, but a few steps can increase the likelihood of good performance. Include transformation of the original features; we include squared and cubed terms in our example, but a richer set is possible. As mentioned above; if sufficient data is available, setting aside a portion for tuning the model before testing can enhance its ability to utilize all information effectively when testing for CI.

The computational method is inherently more computationally intensive than most other CI testing methods, except KCI testing with a large sample size. For instance, with 2000 observations from the Interaction simulation, creating a null distribution with 1000 iterations on a standard laptop with 8 GB RAM and a 1.60 GHz processor using a single core required approximately 13 min. For larger datasets, especially those exceeding 10,000 observations, employing methods such as subsampling may help mitigate computational demands. Furthermore, optimizing computation time will also depend on settings in the XGBoost classification model; hence, adopting specific strategies informed by expert knowledge is recommended for efficiency improvements in practical applications.

No matter how many simulation examples are applied, computational methods can only be shown to work for the specific datasets and scenarios provided. We have attempted to provide a range of difficult testing scenarios to show that computational testing is viable in cases where one needs to establish CI between categorical variables, even when the underlying data structure is complex. However, computational methods are usually less efficient and less generalizable compared to asymptotic methods.

5.2. Strengths

In computational statistics, permutation is a well-known tool for breaking any observed association between variables and forms the basis of many tests. Testing CI by permutation is an intuitive and versatile concept that, in principle, can be applied to any CI testing situation involving various data types. CCI is flexible and demands few assumptions about the nature of the data, only that the observations are independent. As shown through simulations, CCI can handle complex relationships between variables and is, in principle, scalable as you can expand the condition set Z "indefinitely", making it applicable to many settings.

Using an off-the-shelf classification method offers transparency in any specific test. Practitioners can further analyze test results by studying the classification model itself; for example, looking at feature contributions can give insights into the nature of any CI statement. In combination, using QQ plots can give practitioners deeper insights into the behavior of the data under the null hypothesis.

5.3. Further Research

There are several possibilities for further exploration in computational testing of CI and testing CI in general. For instance, although our simulation examples are highly non-linear, they are also low dimensional; the number of observations is much larger than the number of variables. Exploring computational CI in high-dimensional settings, where the number of variables may approach or exceed the number of observations, is perhaps a natural next step. Another avenue of research is to replace the performance metrics log loss and Kappa scores and instead use conditional mutual information criteria or Shapley values to create the null distribution. Additionally, exploring the capabilities of the CCI test under circumstances where observations are not independent would be valuable.

Another avenue for further exploration is a more robust assessment of QQ plots, for instance, by a lineup test introduced by [29]. The lineup test concept consists of embedding a plot that contains the actual data effect among several plots of randomized data or noise and seeing if people can identify the plot with actual data.

Further research should also explore the exponential distribution in the context of CCI testing. Specifically, it should investigate how the exponential distribution's maximum entropy property influences testing performance.

6. Conclusions

This paper has demonstrated a practical application of CI testing, employing a computational approach that leverages permutation, the XGBoost classification algorithm, and MCCV. The proposed method fills a gap in the empirical validation of CI with categorical variables, enhancing CI testing for practitioners in real-world scenarios.

While advantageous for testing CI with categorical data, CCI has limitations. First and foremost, it requires a relatively large sample to achieve sufficient power, it demands a somewhat working knowledge of classification modeling, and it is computationally demanding, especially with a large feature set.

Author Contributions: Conceptualization, C.B.H.T., I.M., K.H.L. and L.E.S.; methodology, C.B.H.T., I.M., K.H.L. and L.E.S.; software, C.B.H.T. and K.H.L.; validation, C.B.H.T.; formal analysis, C.B.H.T.; data curation, I.M. and C.B.H.T.; writing—original draft preparation, C.B.H.T.; writing—review and editing, C.B.H.T., I.M., K.H.L. and L.E.S.; visualization, C.B.H.T.; supervision, I.M., K.H.L. and L.E.S.; project administration, I.M. and K.H.L.; funding acquisition, I.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by "Stiftelsen for Landbrukets Næringsmiddelforskning" (SLNF), The Research Council of Norway, through the projects SFI Digital Food Quality (project number 309259), and by the Norwegian Agricultural Food Research Foundation through the project Precision Food Production (project number 314111).

Data Availability Statement: R scripts used in this manuscript are available at the GitHub repository https://github.com/ChristianBHT/CI_article (accessed on 16 July 2024).

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

CI	Conditional Independence
GCM test	Generalized Covariance Measure test
KCI test	Kernel Conditional Independence test
CCI test	Computational Conditional Independence test
MCCV	Monte Carlo cross-validation
QQ	Quantile–Quantile
XGBoost	Extreme Gradient Boost

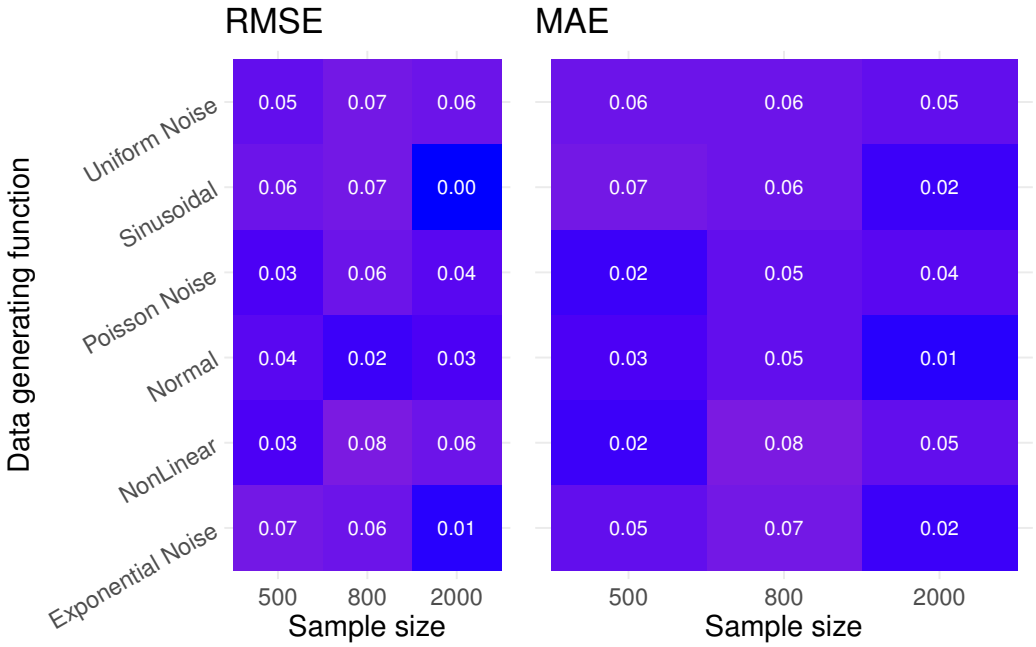
Appendix A. Testing with Continous Data

Although we focus on testing with categorical variables in this article, CCI can also be used with various data types. In Appendix A, we assess the performance of the CCI test by comparing to the GCM test, when Y, X, Z_1, and Z_2 are all continuous variables. Again we use the dependency structure depicted in Figure 1 and generate data from five multivariate distributions. Below are descriptions of various multivariate functions that generate simulated datasets:

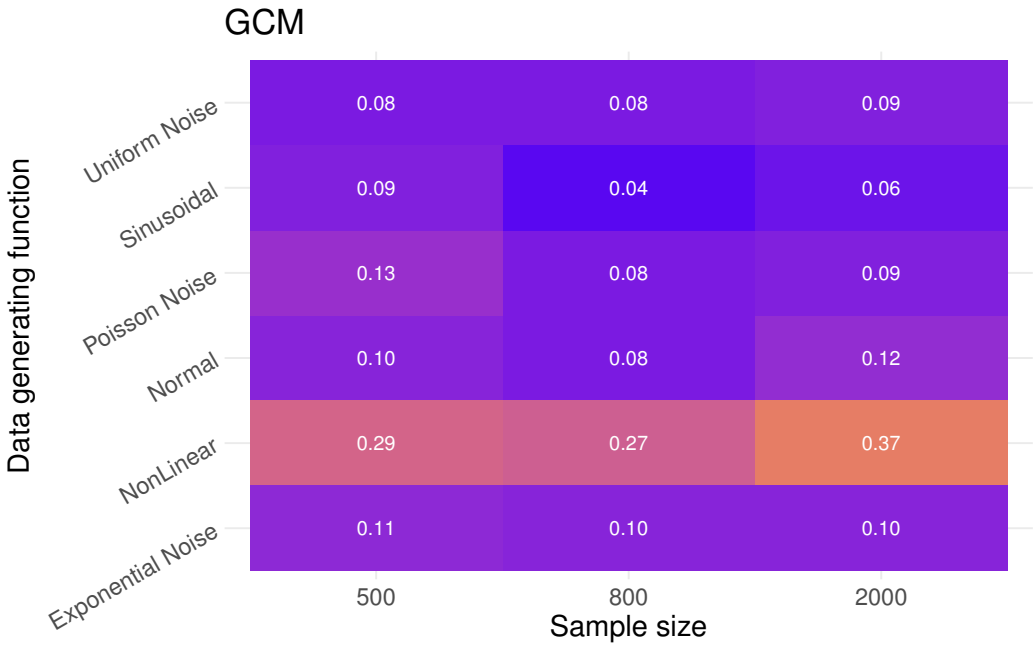
- Normal Data Function:
  Generates a dataset where both Z_1 and Z_2 are drawn from standard normal distributions. The variables X and Y are also normally distributed, influenced by the sum of Z_1 and Z_2, showcasing a simple additive model.
- Nonlinear Normal Data Function: Creates data with Z_1 and Z_2 as normal variables, while X includes a nonlinear transformation involving the exponential of the product $Z_1 \times Z_2$, with normal noise added. Y is formed by the product $Z_1 \times Z_2$ with additional normal noise.
- Uniform Noise Data Function: This function produces a dataset where Z_1 and Z_2 are normally distributed, but X and Y are derived from a linear combination of Z_1 and Z_2 plus a term involving their product, with uniform noise added to introduce non-normal error distribution and test model robustness against such noise.
- Exponential Noise Data: Similar to the uniform noise mode, it replaces the uniform noise with exponential noise with a rate parameter $a = 1$. This function is useful for examining the impact of skewed, heavy-tailed noise on statistical inferences.
- Poisson Noise Function: Generates data where Z_1 and Z_2 are normally distributed, but X and Y include Poisson-distributed noise influenced by the product of Z_1 and Z_2.
- Sinusoidal Data: This function creates data where Z_1 and Z_2 are normal, but X and Y involve sinusoidal transformations modulated by an exponential decay function based on the sum of Z_1 and Z_2. The sinusoidal component depends on linear combinations of Z_1 and Z_2.

For each simulated distribution and test method, we perform 100 tests under both a true null and a false null hypothesis. We calculate the type I error rate by looking at the number of significant results at a significant level $\alpha = 0.05$. The power of the test under each simulated multivariate distribution is estimated by the number of rejected tests under a false null. The performance metrics for computation testing are Root Mean Square Error and Mean Absolute Error (MAE).

In Figure A1, we see type I error rates; for these simulation examples, CCI testing is more conservative than GCM (Figure A1b). The GCM test exhibits a relatively high type I error in scenarios involving the Nonlinear, Normal, and Exponential Noise distributions. This high error rate is likely due to GCM's reliance on the even distribution of residuals around the regression line, making it susceptible to outliers, which these generating functions can produce. Assessing the power of the CCI testing, we see that with a sample of 2000, CCI testing can consistently reject a false null; GCM has a superior rejection rate with lower sample sizes.

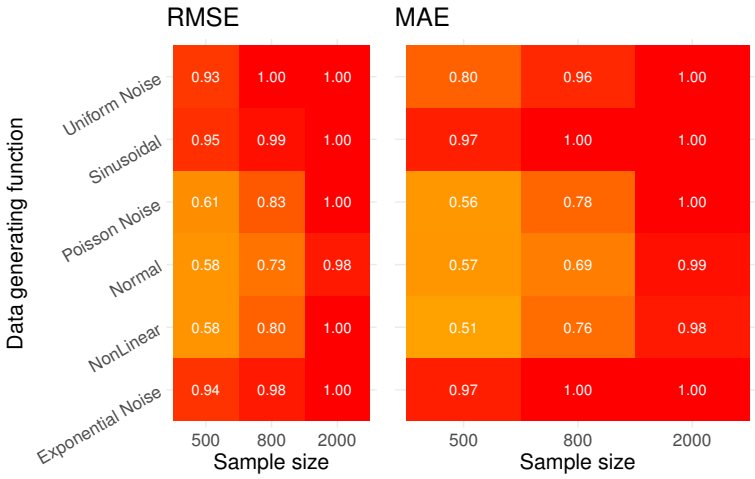


(**a**)

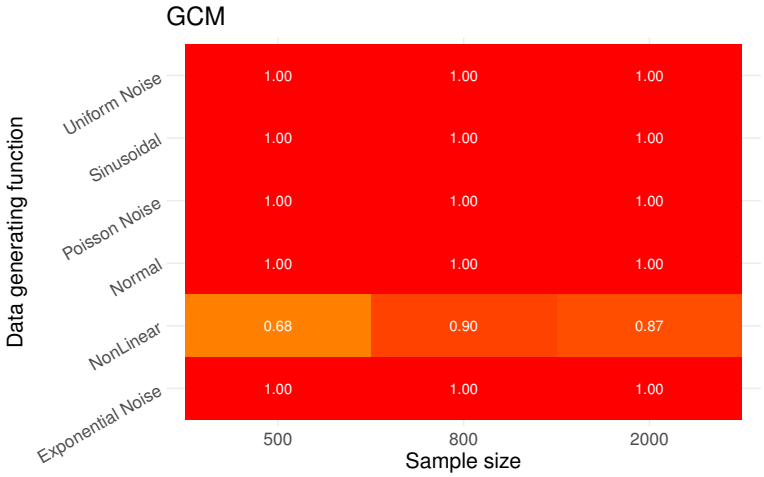


(**b**)

Figure A1. Type I error rates: Rejection rates testing a true null $Y \perp\!\!\!\perp X | Z_1, Z_2$. A bluish color means the error rate is low and should ideally be around 0.05; purple, orange, yellow, and red indicate a high type I error rate. The CCI test performs better in many scenarios, as GCM relies on the assumption of "well behaved" residuals. (**a**) CCI. (**b**) GCM.



(**a**)



(**b**)

Figure A2. Power: Rejection rates testing the false null $Y \perp\!\!\!\perp X | Z_2$. More red means higher power. The results of the CCI test are given in Figure A1a using the performance metrics RMSE and MAE. At sample size 500, CCI testing has low power. The GCM shows high power in almost all scenarios. (**a**) CCI. (**b**) GCM.

In conclusion, we see that CCI testing is a viable testing method in continuous cases and can compete with the GCM test, given a sufficient sample size.

Appendix B. Pseudocode for Multivariate Distributions Generating Functions for Categorical Data

The R version 4.3.2 functions created to generate data are presented in pseudo-code to convey the procedural steps in generating datasets.

Algorithm A1 Interaction Data (Interaction)

$Z_1 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
$Z_2 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
for $i \leftarrow 1$ to N **do**
    if $Z_1[i] < 0$ **and** $Z_2[i] < 0$ **then**
        $X[i] \leftarrow 0$
    else if $Z_1[i] < 0$ **and** $Z_2[i] \geq 0$ **then**
        $X[i] \leftarrow 1$
    else if $Z_1[i] >= 0$ **and** $Z_2[i] < 0$ **then**
        $X[i] \leftarrow 2$
    else
        $X[i] \leftarrow 3$
    end if

    if $Z_1[i] < 0 + Z_2[i] < -1$ **then**
        $Y[i] \leftarrow 0$
    else if $Z_1[i] < 0 + Z_2[i] < 0$ **then**
        $Y[i] \leftarrow 1$
    else if $Z_1[i] < 0 + Z_2[i] < 1$ **then**
        $Y[i] \leftarrow 2$
    else
        $Y[i] \leftarrow 3$
    end if
end for
return $\{Z_1, Z_2, X, Y\}$

Algorithm A2 Exponent and Logarithmic Data (ExpLog)

$Z_1 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
$Z_2 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
for $i \leftarrow 1$ to n **do**
    Compute $tempX \leftarrow \exp(Z1[i]) + Z2[i]$
    if $tempX > 1$ **then**
        $X[i] \leftarrow 3$
    else if $tempX > 0$ **then**
        $X[i] \leftarrow 2$
    else if $tempX > -1$ **then**
        $X[i] \leftarrow 1$
    else
        $X[i] \leftarrow 0$
    end if
    Compute $tempY \leftarrow \log(|Z1[i]| + 1) + Z2[i]$
    if $tempY > 0.5$ **then**
        $Y[i] \leftarrow 3$
    else if $tempY > 0$ **then**
        $Y[i] \leftarrow 2$
    else if $tempY > -0.5$ **then**
        $Y[i] \leftarrow 1$
    else
        $Y[i] \leftarrow 0$
    end if
end for
return $\{Z_1, Z_2, X, Y\}$

Algorithm A3 Polynomial Data (PolyData)

Initialize number of samples N
$Z_1 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
$Z_2 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
for $i \leftarrow 1$ to N **do**
    if $Z_1[i]^2 + Z_2[i]^2 > 2$ **then**
        $X[i] \leftarrow 3$
    else if $Z_1[i]^2 + Z_2[i] > 0.5$ **then**
        $X[i] \leftarrow 2$
    else if $Z_1[i] + Z_2[i]^2 > 0$ **then**
        $X[i] \leftarrow 1$
    else
        $X[i] \leftarrow 0$
    end if
    if $Z_1[i]^3 + Z_2[i] > 1$ **then**
        $Y[i] \leftarrow 3$
    else if $Z_1[i]^2 + Z_2[i]^2 > 0$ **then**
        $Y[i] \leftarrow 2$
    else if $Z_1[i] + Z_2[i]^3 > -1$ **then**
        $Y[i] \leftarrow 1$
    else
        $Y[i] \leftarrow 0$
    end if
end for
return $\{Z_1, Z_2, X, Y\}$

Algorithm A4 Trigonometric Data (TrigData)

Initialize number of samples N
$Z_1 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
$Z_2 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
for $i \leftarrow 1$ to n **do**
    if $sin(Z_1[i]) + cos(Z_2[i]) > 1$ **then**
        $X[i] \leftarrow 3$
    else if $sin(Z_1[i]) + cos(Z_2[i]) > 0$ **then**
        $X[i] \leftarrow 2$
    else if $sin(Z_1[i]) > -1$ **then**
        $X[i] \leftarrow 1$
    else
        $X[i] \leftarrow 0$
    end if
    if $cos(Z_1[i]) - sin(Z_2[i]) > 1$ **then**
        $Y[i] \leftarrow 3$
    else if $\cos(Z_1[i]) - sin(Z_2[i]) > 0$ **then**
        $Y[i] \leftarrow 2$
    else if $cos(Z_1[i]) > -1$ **then**
        $Y[i] \leftarrow 1$
    else
        $Y[i] \leftarrow 0$
    end if
end for
return $\{Z_1, Z_2, X, Y\}$

Algorithm A5 Nonlinear Data (Nonlinear)

Initialize number of samples N
$Z_1 \leftarrow \text{runif}(N, -1, 1)$
$Z_2 \leftarrow \text{runif}(N, -1, 1)$
for $i \leftarrow 1$ to N **do**
    if $\sin(Z_1[i] \times \pi) + Z_2[i] > 1$ **then**
        $X[i] \leftarrow 3$
    else if $\sin(Z_1[i] \times \pi) + Z_2[i] > 0.5$ **then**
        $X[i] \leftarrow 2$
    else if $\sin(Z_1[i] \times \pi) + Z_2[i] > 0$ **then**
        $X[i] \leftarrow 1$
    else
        $X[i] \leftarrow 0$
    end if
    if $\cos(Z_1[i] \times \pi) + Z_2[i] > 1$ **then**
        $Y[i] \leftarrow 3$
    else if $\cos(Z_1[i] \times \pi) + Z_2[i] > 0.5$ **then**
        $Y[i] \leftarrow 2$
    else if $\cos(Z_1[i] \times \pi) + Z_2[i] > 0$ **then**
        $Y[i] \leftarrow 1$
    else
        $Y[i] \leftarrow 0$
    end if
end for
return $\{Z_1, Z_2, X, Y\}$

Algorithm A6 Complex Categorization (ComplexCategorization)

Initialize number of samples N
$Z_1 \leftarrow$ Draw N samples from $\mathcal{N}(0, 1)$
$Z_2 \leftarrow$ Draw N samples from $\mathcal{N}(0, 1)$
for $i \leftarrow 1$ to N **do**
    if $Z_1[i] > 0$ **and** $Z_2[i] > 0$ **then**
        $X[i] \leftarrow 3$
    else if $Z_1[i] > 0$ **and** $Z_2[i] \leq 0$ **then**
        $X[i] \leftarrow 2$
    else if $Z_1[i] \leq 0$ **and** $Z_2[i] > 0$ **then**
        $X[i] \leftarrow 1$
    else
        $X[i] \leftarrow 0$
    end if
    if $Z_1[i] + Z_2[i] > 1$ **then**
        $Y[i] \leftarrow 3$
    else if $Z_1[i] + Z_2[i] > 0$ **then**
        $Y[i] \leftarrow 2$
    else if $Z_1[i] + Z_2[i] > -1$ **then**
        $Y[i] \leftarrow 1$
    else
        $Y[i] \leftarrow 0$
    end if
end for
return $\{Z_1, Z_2, X, Y\}$

Algorithm A7 Multinomial (Multinominal)

Initialize number of samples N
Initialize $\zeta \leftarrow 1.5$
$Z_1 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
$Z_2 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
$random \leftarrow$ Draw N samples from $\mathcal{U}(0,1)$
for $i \leftarrow 1$ to N **do**
    $xb_1[i] \leftarrow Z_2[i] + \zeta \times Z_1[i] \times Z_2[i] + \zeta \times Z_1[i]$
    $xb_2[i] \leftarrow Z_2[i] - \zeta \times Z_1$
    $xp_1[i] \leftarrow \frac{1}{1+\exp(xb_1[i])+\exp(xb_2[i])}$
    $xp_2[i] \leftarrow \frac{\exp(xb_1[i])}{1+\exp(xb_1[i])+\exp(xb_2[i])}$
    if $random[i] < xp_1[i]$ **then**
        $X[i] \leftarrow$ "C"
    else if $random[i] < xp_1[i] + xp_2[i]$ **then**
        $X[i] \leftarrow$ "A"
    else
        $X[i] \leftarrow$ "B"
    end if
end for
$random2 \leftarrow$ Draw N samples from $\mathcal{U}(0,1)$
for $i \leftarrow 1$ to N **do**
    $yb_1[i] \leftarrow \zeta \times Z_1[i] \times Z_2[i]$
    $yb_2[i] \leftarrow \exp(Z_2[i]) + \zeta \times Z_1[i]$
    $yp_1[i] \leftarrow \frac{1}{1+\exp(yb_1[i])+\exp(yb_2[i])}$
    $yp_2[i] \leftarrow \frac{\exp(yb_1[i])}{1+\exp(yb_1[i])+\exp(yb_2[i])}$
    if $random2[i] < yp_1[i]$ **then**
        $Y[i] \leftarrow$ "X"
    else if $random2[i] < yp_1[i] + yp_2[i]$ **then**
        $Y[i] \leftarrow$ "Y"
    else
        $Y[i] \leftarrow$ "Z"
    end if
end for
return $\{Z_1, Z_2, X, Y\}$

Appendix C. Pseudocode for Multivariate Distributions Generating Functions for Continuous Data

Pseudo code for R functions generating data from multivariate continuous distributions.

Algorithm A8 Normal Data

Initialize number of samples N
$Z_1 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
$Z_2 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
$X \leftarrow$ Draw N samples from $\mathcal{N}(Z_1 + Z_2, 1)$
$Y \leftarrow$ Draw N samples from $\mathcal{N}(Z_1 + Z_2, 1)$
return $\{Z_1, Z_2, X, Y\}$

Algorithm A9 Nonlinear Normal Data

1: Initialize number of samples N
2: $Z_1 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
3: $Z_2 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
4: $X \leftarrow$ Compute $\exp(Z_1 \times Z_2)$ and add normal noise
5: $Y \leftarrow$ Compute $Z_1 \times Z_2$ and add normal noise
6: **return** $\{Z_1, Z_2, X, Y\}$

Algorithm A10 Uniform Noise Data

1: Initialize number of samples N
2: $Z_1, Z_2 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
3: $X \leftarrow$ Compute $Z_1 + Z_2 + Z_1 \times Z_2$ and add uniform noise
4: $Y \leftarrow$ Compute $Z_1 + Z_2 + Z_1 \times Z_2$ and add uniform noise
5: **return** $\{Z_1, Z_2, X, Y\}$

Algorithm A11 Exponential Noise Data

1: Initialize number of samples N and rate parameter rate_param
2: $Z_1, Z_2 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
3: $X \leftarrow$ Compute $Z_1 + Z_2 + Z_1 \times Z_2$ and add exponential noise
4: $Y \leftarrow$ Compute $Z_1 + Z_2 + Z_1 \times Z_2$ and add exponential noise
5: **return** $\{Z_1, Z_2, X, Y\}$

Algorithm A12 Poisson Noise Data

1: Initialize number of samples N
2: $Z_1, Z_2 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
3: $X \leftarrow$ Compute $Z_1 \times Z_2$ and add poisson noise with $\lambda = 1$
4: $Y \leftarrow$ Compute $Z_1 \times Z_2$ and add poisson noise with $\lambda = 1$
5: **return** $\{Z_1, Z_2, X, Y\}$

Algorithm A13 Sinusoidal Data

1: Initialize number of samples N and coefficient $a = 1$
2: $Z_1, Z_2 \leftarrow$ Draw N samples from $\mathcal{N}(0,1)$
3: $Z \leftarrow Z_1 + Z_2$
4: $X \leftarrow$ Compute $\exp(-Z^2/2) \times \sin(a \times (2Z_1 + 0.1Z_2))$ and add noise
5: $Y \leftarrow$ Compute $\exp(-Z^2/2) \times \sin(a \times (2Z_2 + 0.1Z_1))$ and add noise
6: **return** $\{Z_1, Z_2, X, Y\}$

References

1. Shah, R.D.; Peters, J. The Hardness of Conditional Independence Testing and the Generalised Covariance Measure. *Ann. Stat.* **2018**, *48*, 1514–1538. [CrossRef]
2. Ankan, A.; Wortel, I.M.N.; Textor, J. Testing Graphical Causal Models Using the R Package "dagitty". *Curr. Protoc.* **2021**, *1*, e45. [CrossRef] [PubMed]
3. Pearl, J.; Glymour, M.; Jewell, N.P. *Causal Inference in Statistics—A Primer*; Wiley: Hoboken, NJ, USA, 2016.
4. Chu, T.; Glymour, C.; Scheines, R.; Spirtes, P. A statistical problem for inference to regulatory structure from associations of gene expression measurements with microarrays. *Bioinformatics* **2003**, *19*, 1147–1152. [CrossRef] [PubMed]
5. VanderWeele, T.J.; Ko, Y.A.; Mukherjee, B. Environmental Confounding in Gene-Environment Interaction Studies. *Am. J. Epidemiol.* **2013**, *178*, 144–152. [CrossRef] [PubMed]
6. Zhang, K.; Peters, J.; Janzing, D.; Schölkopf, B. Kernel-based Conditional Independence Test and Application in Causal Discovery. *arXiv* **2012**, arXiv:1202.3775. [CrossRef]
7. Runge, J. Conditional independence testing based on a nearest-neighbor estimator of conditional mutual information. *arXiv* **2017**, arXiv:1709.01447. [CrossRef]
8. Taoufik, B.; Rombouts, J.V.; Taamouti, A. Nonparametric Copula-Based Test for Conditional Independence with Applications to Granger Causality. *J. Bus. Econ. Stat.* **2012**, *30*, 275–287.

9. Petersen, L.; Hansen, N.R. Testing Conditional Independence via Quantile Regression Based Partial Copulas. *J. Mach. Learn. Res.* **2021**, *22*, 3280–3326.
10. Su, L.; Spindler, M. Nonparametric Testing for Asymmetric Information. *J. Bus. Econ. Stat.* **2013**, *31*, 208–225. [CrossRef]
11. Fan, J.; Feng, Y.; Xia, L. A projection-based conditional dependence measure with applications to high-dimensional undirected graphical models. *J. Econom.* **2020**, *218*, 119–139. [CrossRef]
12. Dawid, A.P. Conditional Independence in Statistical Theory. *J. R. Stat. Soc. B* **1979**, *41*, 1–31. [CrossRef]
13. Agresti, A. *Categorical Data Analysis*; John Wiley and Sons: Hoboken, NJ, USA, 2002.
14. Scutari, M.; Strimmer, K. *Introduction to Graphical Modelling*; Springer: New York, NY, USA, 2010.
15. Baja, A. Performance Metrics in Machine Learning [Complete Guide]. 2023. Available online: https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide (accessed on 16 July 2024).
16. Collingridge, D.S. A Primer on Quantitized Data Analysis and Permutation Testing. *J. Mix. Methods Res.* **2013**, *7*, 81–97. [CrossRef]
17. Casella, G.; Berger, R.L. *Statistical Inference*; Cengage: Boston, MA, USA, 2002.
18. Massey, F.J. The Kolmogorov-Smirnov Test for Goodness of Fit. *J. Am. Stat. Assoc.* **1951**, *46*, 68–78. [CrossRef]
19. Chen, T.; Guestrin, C. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16), San Francisco, CA, USA, 13–17 August 2016; pp. 785–794. [CrossRef]
20. Ho, T.K. Random decision forests. In Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, Canada, 14–16 August 1995; Volume 1, pp. 278–282.
21. Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; Liu, T.Y. Lightgbm: A highly efficient gradient boosting decision tree. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 3146–3154.
22. Dorogush, A.V.; Ershov, V.; Gulin, A. CatBoost: Gradient boosting with categorical features support. *arXiv* **2018**, arXiv:1810.11363. [CrossRef]
23. Chen, T.; He, T.; Benesty, M.; Khotilovich, V.; Tang, Y.; Cho, H.; Chen, K.; Mitchell, R.; Cano, I.; Zhou, T.; et al. *XGBoost: Extreme Gradient Boosting*; R Package Version 1.7.6.1; 2023. Available online: https://CRAN.R-project.org/package=xgboost (accessed on 19 July 2024). [CrossRef]
24. Kuhn, M. Building Predictive Models in R Using the caret Package. *J. Stat. Softw.* **2008**, *28*, 1–26. [CrossRef]
25. Peters, J.; Shah, R.D. *GeneralisedCovarianceMeasure: Test for Conditional Independence Based on the Generalized Covariance Measure (GCM)*; R Package Version 0.2.0; 2022. Available online: https://CRAN.R-project.org/package=GeneralisedCovarianceMeasure (accessed on 19 July 2024). [CrossRef]
26. Heinze-Deml, C.; Peters, J.; Meinshausen, N. Invariant Causal Prediction for Nonlinear Models. *arXiv* **2017**, arXiv:1706.08576. [CrossRef]
27. Venables, W.N.; Ripley, B.D. *Modern Applied Statistics with S*, 4th ed.; Springer: New York, NY, USA, 2002; ISBN 0-387-95457-0.
28. Loy, A.; Follett, L.; Hofmann, H. Variations of Q-Q Plots—The Power of our Eyes! *arXiv* **2015**, arXiv:1503.02098. [CrossRef]
29. Buja, A.; Cook, D.; Hofmann, H.; Lawrence, M.; Lee, E.K.; Swayne, D.F.; Wickham, H. Statistical inference for exploratory data analysis and model diagnostics. *Philos. Trans. R. Soc. Math. Phys. Eng. Sci.* **2009**, *367*, 4361–4383. [CrossRef]